WhereScape® RED

WHERESCAPE RED USER GUIDE

6.8.7.0

www.wherescape.com

WhereScape®

WhereScape RED User Guide

The software described in this book is furnished under a license agreement and may be used only in accordance with the terms of the agreement.

Copyright Notice

Due to continued product development, this information may change without notice. WhereScape Software Limited does not warrant that this document is error-free.

Trademarks

WhereScape USA, Inc

1915 NW AmberGlen Parkway
Suite 400, Beaverton
Oregon 97006
United States
T: 503-466-3979
F: 503-466-3978

WhereScape Limited

P.O.Box 56569, Auckland 1446
12-16 Tapora Street
Quay Park
Auckland 1010, New Zealand
T: +64-9-358-5678
F: +64-9-358-5679

WhereScape Europe

450 Brook Drive
Green Park
Reading RG2 6UU
United Kingdom
T: +44-118-914-4509
F: +44-118-914-4508

WhereScape Asia Pte. Ltd

300 Tampines Avenue 5
#09-02 Singapore 529653
T: +65-6679-5728

WhereScape®

TABLE OF CONTENTS

WhereScape®

OVERVIEW

WhereScape RED User Guide

The WhereScape RED **User Guide** is available either as online help, as a PDF, or in a printed manual. The User Guide provides information on how to use WhereScape RED to build a data warehouse.

This online help version of WhereScape RED's documentation is for RED 6.8.7.0, last updated Monday, January 30, 2017.

IN THIS CHAPTER

OVERVIEW OF WHERESCAPE RED

Traditionally data warehouses take too long to build and are too hard to change. WhereScape RED is an Integrated Development Environment to support the building and managing of data warehouses.

It has the flexibility to enable you to build a variety of architectures including:

- enterprise data warehouses
- dimensional data warehouses
- data marts
- user facing views, aggregates and summaries

In all cases the core values of WhereScape RED are twofold: its rapid building capabilities that enable better data warehouses to be built, faster, and its integrated environment that simplifies management.
As a data warehouse specific tool, WhereScape RED embodies a simple, pragmatic approach to building data warehouses. With WhereScape RED you specify what you want to achieve by dragging and dropping objects to create a meta view, and then let WhereScape RED do the heavy lifting of creating the necessary tables and procedures etc. Data warehouse wizards prompt for additional information at critical points to provide the maximum value from the generated objects.

Different data warehousing approaches including agile, prototyping and waterfall are supported by WhereScape RED. Agile developers will find specific functionality has been included to support common agile practices.
For developers who are new to data warehousing, or are looking for a fast, pragmatic approach, WhereScape's Pragmatic Data Warehousing Methodology can be used.

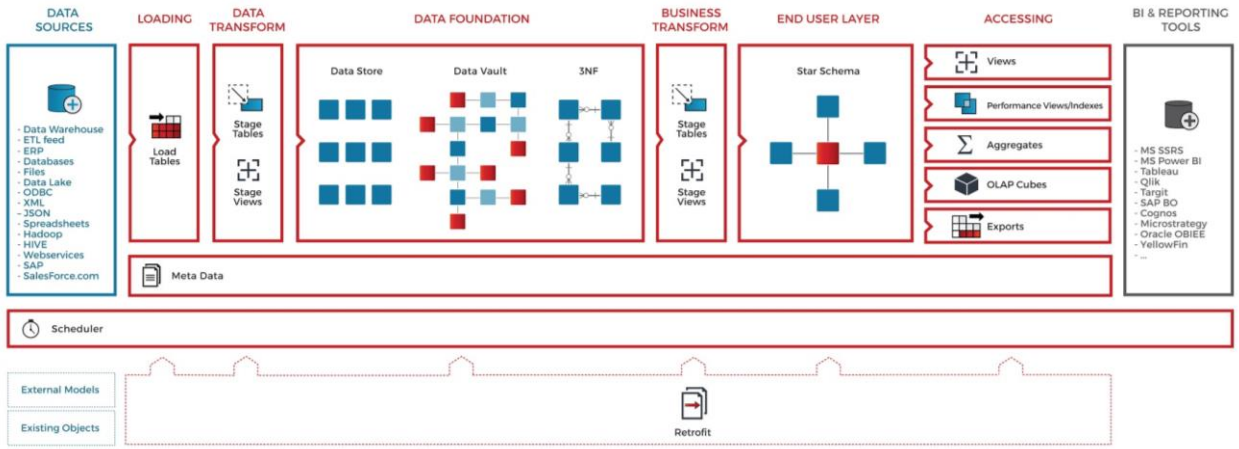
The basic concepts behind WhereScape's Pragmatic Data Warehousing Methodology are:

- minimize the impact on the source systems
- centralize management within the data warehouse
- store transactional data at the lowest practical grain within the data warehouse
- snapshot, combine and rollup transactional tables to provide additional value
- utilize star schemas, views or cubes for end user access
- allow for incremental loads from day one
- use of an iterative approach
- simplifying the reconciliation of load tables to source system tables - data is cleaned and transformed in subsequent stage tables instead
- design the data warehouse independently from the end user tool layer

WhereScape RED supports these concepts to facilitate very rapid delivery of data warehouses. WhereScape RED controls the flow of data from the source systems through transforming and modeling layers to analysis areas.
Different styles of data warehousing (EDW 3NF, dimensional etc.) are supported and utilize different objects, but all follow the same basic flow.

Data Flow - Enterprise models

1. source (OLTP) system
2. load tables
3. stage tables
4. data store tables
5. model tables, dimension tables, or detailed (transactional) fact tables
6. roll up fact table(s)
7. aggregate and/or KPI fact table(s)
8. views
9. export objects
10. Microsoft Analysis Services cubes



Data Flow

Data is moved from source tables to load tables via scripts, database links and ODBC links. These load tables are created by dragging and dropping from a connection object. Load tables are generally based on source system tables. Their main purpose is to be a destination for moving data as simply and quickly as possible from the source system. Load tables will generally hold a single unit of data (e.g. last night or last month), and will be truncated at the start of each extract. Transformations can be performed on the columns during the load process if required.
Load tables feed stage tables, which in turn feed data store, model or dimension tables. Data from multiple load tables can be combined at this level.
First tier transactional tables (fact or model) are created and updated from stage tables. Second tier tables (model, summary rollup, aggregate, KPI etc.) are created and updated from lower level tables. Cubes can be created from transactional tables or views.

Procedural code

WhereScape RED generates procedural code in the target database's native language (e.g. PL/SQL for Oracle) at each stage in the data warehouse build process. The generated code is, in nearly all cases, sufficient to create a rapid prototype of the data warehouse.
While the generation of code is often seen as a key benefit of WhereScape RED, the ability to control and manage custom code is also critical to the long-term management of the data warehouse environment.
In most cases 85-100% of the generated code will be taken through to production with no customization required.

Scheduler

The flow of data from the source systems to data warehouse tables is controlled and managed by the WhereScape RED scheduler. All generated code includes audit and error logging logic that is used by the scheduler.
The scheduler provides a single point of control for the warehouse. From the scheduler the state of all jobs can be ascertained. Any warning or error messages can be investigated, and should a problem occur the scheduler controls the restart of the job from the point of failure.

Documentation

Documenting the warehouse is often a task left until last, and in many cases done once (if at all!) and not kept up to date. WhereScape RED generates user and technical documentation, including diagrams, in HTML format.
Technical documentation includes copies of all current procedures.
User documentation includes a glossary of business terms available independently of any end user tool.
Where additional specific information needs to be included in the documentation, WhereScape RED supports the inclusion of custom HTML pages in the generated output. This means in many cases the entire documentation requirements can be managed from one location, and regenerated as changes occur.

WhereScapeRED and Traditional ETL Tools

WhereScape RED's core strength is in the rapid building of data warehouse structures. Organizations that have already purchased traditional ETL tools can use WhereScape RED as a pureplay data warehouse toolset. WhereScape RED can be used to iteratively build data marts or presentation layer objects that need to be constantly updated to keep relevant for end users. In mosts cases customers will find that WhereScape RED has enough ETL capabilities to build the entire data warehouse, using the database rather than a proprietary engine to perform ETL processing.
The cross over in functionality between ETL tools and WhereScape RED is not large. WhereScape RED is tightly integrated into the data warehouse database and has an embedded data warehouse building approach. For WhereScape data movement is the start of the process - from source system to load tables. The key benefits of the product: development productivity and an integrated environment to manage and maintain your warehouse, come after the data movement stage. Where a traditional ETL tool is already in use, the output of the ETL process is a WhereScape RED's Load, Stage, Dimension, Fact or Model table from which WhereScape RED builds more advanced data warehouse structures.

DOCUMENTATION ROADMAP

The WhereScape RED **Installation and Administration Guide**, the **online help**, and the WhereScape RED **User Guide** assume that you are proficient in the use of the Windows operating system.
WhereScape RED often provides multiple ways to accomplish a task. In some cases, you can use the main menu, the right-click menu, or a toolbar, or a key combination (e.g. Alt/M and Ctrl/M to raise menus).
Instructions in this documentation generally include only the most convenient method of accomplishing a task.

The following sources of information are available with WhereScape RED:

WhereScape RED Installation and Administration Guide

The **Installation and Administration Guide** is available either as online help, as a PDF, or in printed format. The Installation and Administration Guide provides the information needed to:

- Install the WhereScape RED software
- Validate the various software components required by WhereScape RED
- Install the WhereScape RED metadata
- Install a scheduler
- Optionally install third party data warehouse applications
- Upgrade the WhereScape RED software
- Create and load language files.

WhereScape RED User Guide

The WhereScape RED **User Guide** is available either as online help, as a PDF, or in a printed manual. The User Guide provides information on how to use WhereScape RED to build a data warehouse.

WhereScape RED Tutorials

The WhereScape RED **Tutorials** are available either as online help or as a PDF. The Tutorials introduce you to the terms and methodologies embodied in WhereScape RED by guiding you through:

- Building a simple dimensional data warehouse
- Creating roll-up fact tables and aggregates
- Loading an ascii file into a new load table
- Scheduling the data warehouse objects
- Fine tuning the data warehouse by creating complex dimensional hierarchies
- Fine tuning the data warehouse by adding Analysis Services cubes
- Fine tuning the data warehouse by creating KPI fact tables

Sql Admin User Guide

The **Sql Admin User Guide** is available either as online help, or as a PDF. It provides documentation in the use of the stand-alone SQL query tool shipped as part of the WhereScape RED product.

WhereScape Forum

A **web forum** is available at http://www.wherescape.com. This forum contains information on the latest version, and bug reports that may be relevant for installation. In addition, the WhereScape Blog is available at http://www.wherescape.com which may provide additional information.

CHAPTER 1
DESIGN INTRODUCTION

Design Introduction

WhereScape RED can be used to build data warehouses based on any number of design philosophies from EDW 3NF enterprise data warehouses with consumer data marts through to federated or conformed star schema based warehouses. In the absence of another approach, the following methodology can be used for the design of data warehouses.

Note: This section can be skipped if you already have data warehouse design experience or a methodology you wish to utilize. It is meant to provide the novice designer with some tips for designing a data warehouse.

Design Approach

The concepts behind the WhereScape Pragmatic Data Warehouse Methodology are as follows:

1. Building an enterprise-wide data warehouse is a process - an evolution rather than a big bang. Start small and grow the warehouse in manageable chunks until all the pieces are in place. Once you reach that stage, changes and new source systems will continue the process.
2. You need to understand the big picture, but not get lost in it. Talk to all the various departments, business units and companies within the organization. Do so at a relatively high level and try to understand how the information from each area impacts or affects the others. Identify commonalities and areas where the same information is handled in different ways. This process should take days or weeks not months.
3. Identify the high value, high return and possibly easiest areas of the business. Drill down in these areas and break down the workload into small manageable chunks of work, for example, one to two analysis areas. Agree on the first component of the data warehouse and do that.
4. Get an understanding of the source system for this first component or analysis area. If possible, get an entity relationship diagram and talk to the people who built or support the application. Identify the tables that contain the key information you will need. The goal is a quick and initial view, a detailed specification is not required.
5. Design the first component. This design should be a first draft, and can be written rather than using a design tool. Remember at this stage what the end users want is not really known, so don't set the design in concrete, or spend a large amount of time in this area.

   Note: Experienced users of WhereScape RED will often dispense with a design and go straight to building a prototype.

6. Build a prototype. In most cases this should not take more than one or two weeks - experienced WhereScape RED developers can expect to build prototypes in hours or days. Concentrate on the detailed and descriptive data, unless you have a clear picture of the summarized requirements. Do as much as possible in terms of validating the data back to the source system. If dealing with a large or complex source system then only deliver a segment in this prototype, e.g., one branch, one store, one product group, etc. Keep It Simple.
7. Demonstrate the prototype to a group of the key users. Then drill down to a subset of key users (we recommend no more than three) who will help you go forward with the design. If possible, give these users access to the prototype and get them using the data. Stress that data accuracy is not the issue at this stage, rather the look and feel.
8. Enhance the prototype with the feedback provided by the users. Again, a quick process. If complicated requirements evolve then create a plan to implement, doing the highest value parts first. The goal is to get quick buy in and support from the two or three key users.
9. Provide key users access to the reworked prototype and get them using the data. Have them define the business names for all the measures and attributes, and to define any pre-calculated measures that they frequently use. Get them to define the hierarchies in the data. Ascertain the commonly utilized queries and reports, and see if there would be a better way of presenting these.
10. From the user feedback look at the need or possibility of using higher level fact tables, such as summaries, aggregates, snapshot or composite rollup tables.

The concepts and methodologies for designing and building a data warehouse are beyond the scope of this manual.
It is assumed that the reader understands the basic concepts of a data warehouse, and is familiar with modeling, EDW 3NF, star and snowflake schemas, dimensions, fact tables, etc.

Refer to the WhereScape web site for a basic overview of data warehouse design if required.

CHAPTER 2
OBJECTS AND WINDOWS

WhereScape RED makes use of an object concept when dealing with the different components that make up a data warehouse solution.

The main object types are: Connections, Load Tables, Dimensions, Stage Tables, Fact Tables, OLAP Cubes, Aggregates, Procedures, Host Scripts, Indexes, Retros and Exports.

This chapter explains and provides an overview of each of these object types and how they can be managed and organized. The full functionality of each object is covered in the following chapters.

The various Windows, Panes and Views that form the WhereScape RED tool are also explained.

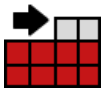
IN THIS CHAPTER

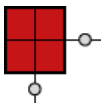
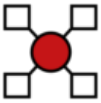
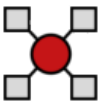
OBJECT TYPES

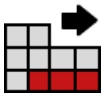
WhereScape RED has a concept of objects which are combined to create real world data warehouses and data marts, fast.
Each WhereScape RED object has properties that allow the data warehouse developer to change how the object is used.

Note: Some object types may not be available for certain types of WhereScape RED licenses.

WhereScape RED objects include:

Object Type	Purpose
Connection 	**Connections** define the path to external objects such as source data. Examples of connection object types are databases, analysis services cubes, operating systems or ODBC sources. Connections isolate environments simplifying, for example, the promotion of code between development and production.
Load Table 	**Load tables** are the first entry point of data into the data repository, and typically hold the latest set of change data. These objects contain their definition. Load tables can be defined as database link, ODBC, external, file, script or XML. Based on their definition they will, for example, run a predefined script or create a load script at run time. Pre-load actions (e.g. truncate) or post load procedures can be defined as part of a load object. In addition, transformations (either during or after the load) can be defined against columns in a load table.
Dimension 	**Dimension tables** are the constraining elements in the star schema design, and are defined by this object type. WhereScape RED will automatically generate procedural code for the three standard types of slowly changing dimensions, as well as date ranged dimensions (where the current version is defined by an external system). WhereScape RED also ships with a standard time dimension which can of course be extended. Dimensions can also be defined as mapping or work tables which do not appear in the generated user documentation.
Dimension View 	A **dimension view** is a database view of a dimension table. It may be a full or partial view. A common usage is to create dimension views where multiple date dimensions exist for one fact table. Other types of views supported by WhereScape RED include fact views, other table views, work views and user defined views.

Object Type	Purpose
Stage Table 	**Stage tables** are used in the transformation of raw data into model or star schema format. They typically hold only the latest set of change data. As well as custom procedures, WhereScape RED can generate different types of procedural code based on the complexity and size of the dataset and performance requirements. Examples of procedural types that can be generated are cursor, sorted cursor, set, set + cursor or set merge procedural code. A stage table can also be defined as a work table, which has the same properties as a stage table but does not appear in the generated user documentation.
EDW 3NF Table 	An **EDW 3NF table** is a data warehouse object used to build third normal form enterprise data warehouses. In WhereScape RED, EDW 3NF objects have many of the code generating attributes of stage, dimension and fact tables. Third normal form enterprise data warehouses can be thought of as a source system for star schema data marts. Alternatively, they may be reported off directly by users and reporting tools.
Data Store Table 	A **Data Store Table** is a data warehouse object used to store any type of data for later processing. In WhereScape RED, Data Store objects have many of the code generating attributes of stage, dimension and fact tables. Data objects can be thought of as a source system for the data warehouse. Alternatively, they may be reported off directly by users and reporting tools. Data Store Objects can be considered either reference or transactional in nature.
Fact Table 	**Fact tables** are the central table in a star schema design. This object type allows the definition of fact tables. They support transactional, rollup, snapshot or partitioned (detail, rollup or exchange) fact tables. Changing a fact table's properties to partitioned will start a partitioning wizard that prompts for the required information.
Kpi Fact Table 	This object type supports a special type of fact table. A mandatory KPI (**Key Performance Indicator**) dimension provides a set of KPIs which are stored and maintained by this object type.
Aggregate 	The **aggregate** object type provides a means to speed up access by summarizing data to a higher grain. For dimensional models a rollup of the fact data will allow removal of dimensions that are no longer valid.
View 	**View** objects are usually created as end user objects from any table in the data warehouse. The data or columns may be restricted or extra descriptions may be added for use by the end user or reporting tools.
OLAP Cube 	The **OLAP Cube** object type uses Analysis Services cubes to deliver OLAP functionality in WhereScape RED. A cube is a set of related measures and dimensions that is used to analyze data from a variety of different front end tools. OLAP Cubes are built from fact objects and aggregate objects in WhereScape RED.

Object Type	Purpose
OLAP Dimension 	An **OLAP Dimension** is built by WhereScape RED for every dimension table associated with the fact (or aggregate) table the OLAP Cube is derived from. OLAP Dimensions are shared across one or more OLAP Cubes. In analysis services, a dimension is a group of attributes that represent an area of interest related to the measures in the cube and which are used to analyze the measures in the cube.
Procedure 	The **procedure** object type is used to define and hold database stored procedures. As such it may contain functions, procedures and packages that are generated, modified or custom developed.
Host script 	**Host script** objects are either Windows or UNIX scripts. These scripts are maintained within the WhereScape RED environment and can be scheduled to run in their host environments.
Index 	This object type defines database **indexes** used to improve the access times on any of the table object types. (i.e. Load, Stage, Dimension, Fact, KPI Fact and Aggregate).
Export 	**Exports** are used to manage exports from the data repository. Exports are the reverse of load tables, taking data from a table to a flat file.
Retro 	**Retros** are used to load predefined data models from modeling tools and to retrofit existing tables into the WhereScape RED metadata.
Retro Copy 	**Retros** can be used to copy data from an existing data warehouse into WhereScape RED metadata. Retros can be set as **Retro Copy** objects to enable data transfer from the existing data warehouse to the new data warehouse.
Template 	**Template** objects are used to generate DDL, update procedures and host scripts. Once a template has been created it can be associated with a table and an operation on that table. The template is then used to generate the script used for the associated operation. Each template is assigned a type and a target database, these properties are used to assist with filtering when associating table operations to templates. Note that not all operations support template script generation on all target databases. Utility type templates can contain common code for use by other templates.
Hub 	A **Hub** is a table of unique business keys, they usually contain a hash key, business key(s), load date and record source. Hubs normally have at least one Satellite.

Object Type	Purpose
Link 	**Links** are many-to-many tables representing current and past relationships between two or more Hub entities and are used to describe associations, transactions, hierarchies and redefinitions of Hub entities in a Data Vault. Links have their own hash key and the hash keys for the Hubs that are linked as well as a Load Date and Record Source. The attributes describing the context of a link are stored in Satellite Tables (see below).
Satellite 	**Satellites** are Data Vault objects which contain metadata that provides context for Hub and Link entities at a given time or over a period of time. Each Satellite entity can contain information on one Hub or Link. Satellite tables contain a hash key for the parent Hub or Link, a timestamp for the date of change and relevant descriptive fields. Satellites are usually created once per source system. However, descriptive attributes can change at different rates, so Satellites can also be created based on rate of change.
Custom1/Custom2 	**Custom1** and **Custom2** objects are user defined objects. These Object Types can be renamed in the **Tools/Options/Object Types/Object Names** menu.

Connections are normally the first objects created. These connections are then used in the creation of load tables through the drag and drop functionality. Subsequent objects can also be created using drag and drop.

Note that although the object types have names that correspond with their primary usage, they can be used for other purposes. For example, the fact object type could be used to create persistent stage tables if required.

Some objects are not supported by all databases, and some advanced properties are specific to the different databases.

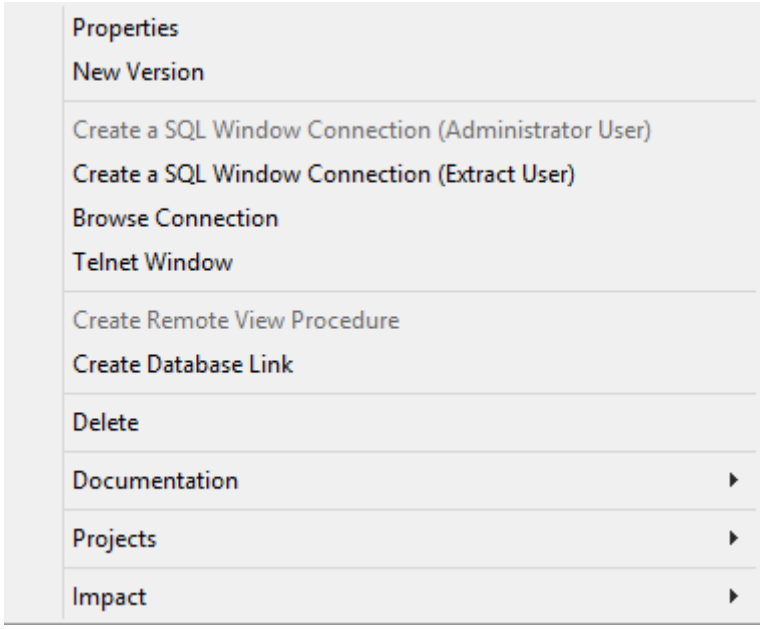
WORKING WITH OBJECTS

Most object types perform some form of action in the data warehouse. For example; dimension, stage, fact and aggregate table based objects are 'Updated' in the data warehouse via the defined update procedure.
Procedures can be executed in the database. When positioned on an Object in the left pane of the WhereScape RED builder window, the right-click pop-up menu provides a number of options for manipulating the object.
Further options may be available through the menus provided in the various windows.

The operations of each of the objects is discussed in the following chapters. A brief overview of some of the more common operations follows:

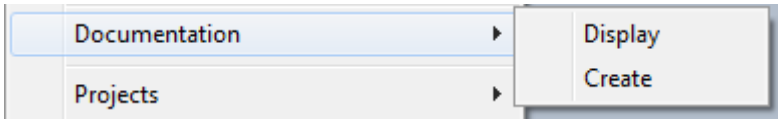
Connections

Connections, once defined are typically browsed and used as a source for drag and drop operations. For database connections, a database link is normally required.
This link can be created via the right-click menu associated with a connection. See the sample menu below:



Other operations available through the menu are editing the Properties of the connection, creating a version of the connection, creating a telnet window for UNIX connections and creating a remote view creation procedure where required for database connections and loads.

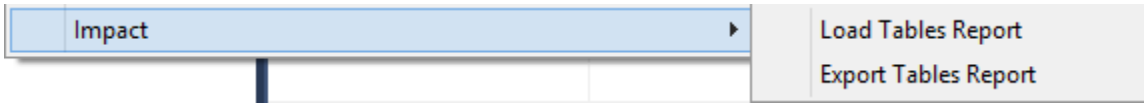
The **Documentation** menu option can be used to generate (or read if already generated) the WhereScape RED HTML documentation for the selected connection.
Two options are available: **Display** and **Create**:



The **Projects** menu option enables you to remove the connection from the current project or to add the connection to the current project. The **List Projects** option displays a list of projects which contain the current object, results are shown in the bottom pane. Multiple objects can be selected by double-clicking the **Connection** icon in the left pane and Ctrl + clicking multiple connections in the Drop Target Pane.

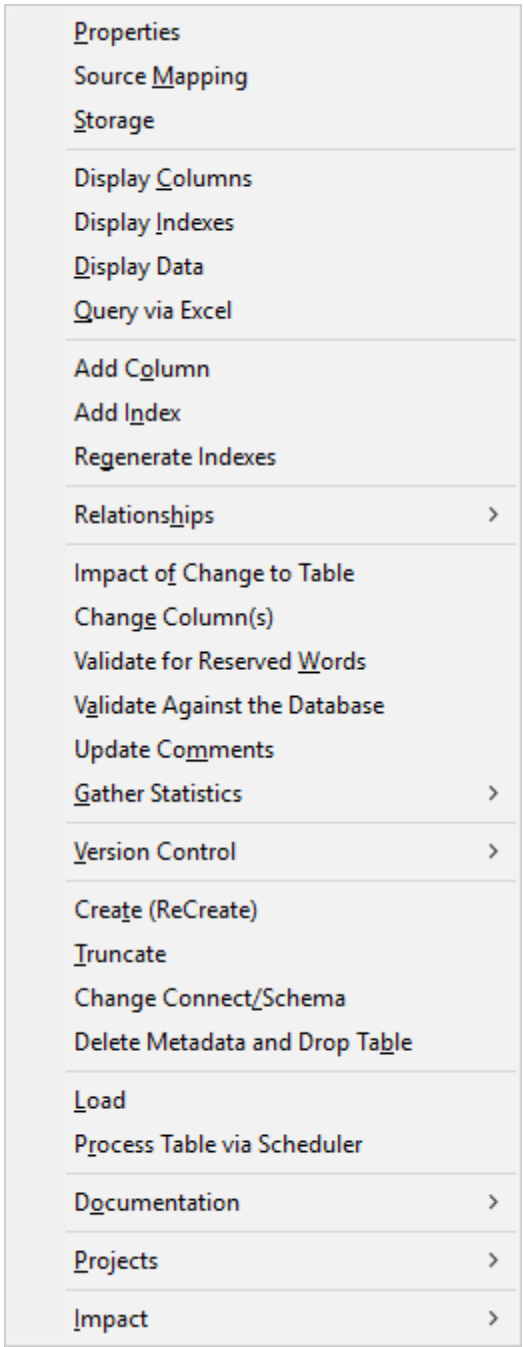
If there aren't any projects in the repository, these options are unavailable.

The **Impact** menu option enables you to run reports on Load and Export objects associated with the connection.



Load Tables

Load tables once defined would normally be created and loaded, unless these actions were performed as part of the drag and drop operation.
The menu below shows the operations that can be performed on load tables.



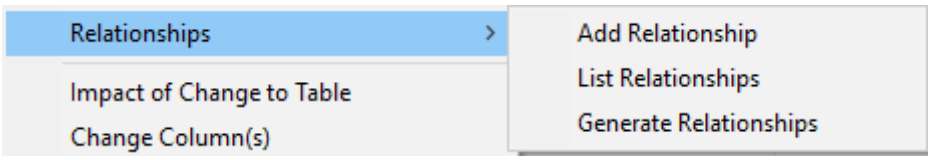
Properties, **Source Mapping** and **Storage** all launch the Properties window for the load table, albeit focused on different tabs within this window.
The columns and indexes of the load table can be displayed using **Display Columns** and **Display Indexes**. Any data in the load table can be displayed using **Display Data**.
If the data is displayed, only the first 100 rows are returned from the table. Either the Sql Admin tool (accessible via the WhereScape start menu option), or the Excel query will have to be used if more detailed data analysis is required.
Query the columns in Excel using **Query via Excel**.

TIP: When a column list has been displayed in the central pane, it is sorted based on the **order** field associated with each column. A click on the column label **Col name** will sort the columns into alphabetical order.
A subsequent click will re-sort based on the **order** field.

New columns and indexes can be manually added through this menu using **Add Column** and **Add Index**. Normally columns are added via drag and drop and most common indexes are created during the procedure generation phase.

The **Regenerate Indexes** menu option is used to add missing standard indexes. Selecting this menu item displays a dialog box with options to regenerate missing indexes in the metadata and recreate them or to just regenerate the missing indexes in the metadata.

The **Relationships** menu options allow the management of enhanced relationships. The **Add Relationship** option opens the Add Relationships dialog, the **List Relationships** option displays a list of enhanced relationships in the Drop Target Pane for the selected object and **Generate Relationships** generates relationships which have not yet been defined in metadata.



The **Impact of Change to Table** menu option produces a list of objects that will be potentially impacted by a change to the load table structure.

The **Change Column(s)** menu option to apply changes to a selected number of columns.

The **Validate for Reserved Words** menu option produces a list of table or column names where reserved words have been used; enabled for supported ODBC Drivers.
The metadata for the load table can be compared with the physical table resident in the database using **Validate Against the database**, and where required the table altered to match the metadata.

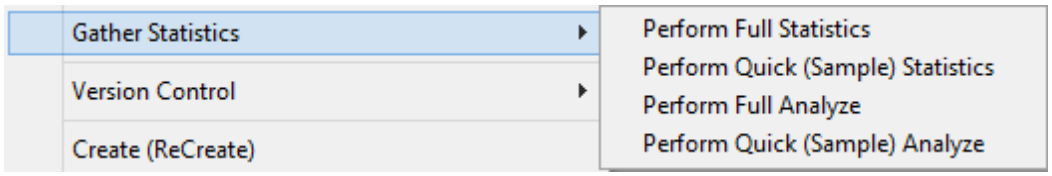
The **Update Comments** menu option refreshes table and column comments on the table from the metadata using the table's description and columns' business definition.

Use **Gather Statistics** to gather statistics on a table. This action will enable the underlying database to optimize each query based on the statistics collected about the data that is being accessed.
Users can either chose to **Perform Full Statistics** or to **Perform Quick (Sample) Statistics**.| Gathering statistics can be performed on any table by selecting this option from a table's right click menu, or to automate this process, by adding a statistics task to a job being processed by the scheduler (Stats, Quick Stats, Analyze or Quick Analyze).
For more information about adding statistics tasks to jobs for SQL Server, Oracle and DB2 see ***Editing Tasks in a Job*** (on page 767).
Oracle has the statistics option in the Options dialog to save a generic statistics statement for analyse, quick analyse, stats and quick stats. If there is no statistics statement saved, then a default statistics statement is used.

The statistics process from the object context menu for tables is described below:

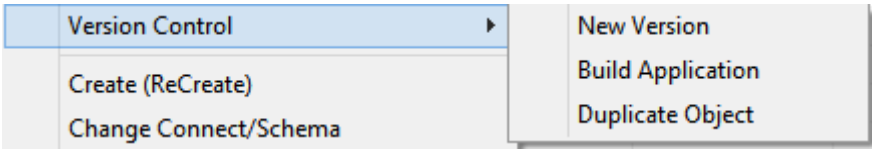
Oracle Tables:	
Perform Full Statistics	BEGIN dbms_stats.gather_table_stats(ownname=>'schema', tabname=>'table' , cascade=>TRUE); END;
Perform Quick (Sample) Statistics	BEGIN dbms_stats.gather_table_stats(ownname=>'schema', tabname=>'table' , estimate_percent=>'3' , cascade=>TRUE); END;
Perform Full Analyze	ANALYZE TABLE schema.table COMPUTE STATISTICS
Perform Quick (Sample) Analyze	ANALYZE TABLE schema.table ESTIMATE STATISTICS SAMPLE 3 PERCENT
SQLServer Tables	
Perform Full Statistics	UPDATE STATISTICS schema.table WITH FULLSCAN
Perform Quick (Sample) Statistics	UPDATE STATISTICS schema.table WITH SAMPLE 3 PERCENT
DB2 Tables:	
Perform Full Statistics	RUNSTATS ON TABLE schema.table WITH DISTRIBUTION AND DETAILED INDEXES ALL
Perform Quick (Sample) Statistics	RUNSTATS ON TABLE schema.table
Perform Full Analyze	RUNSTATS ON TABLE schema.table WITH DISTRIBUTION AND DETAILED INDEXES ALL
Perform Quick (Sample) Analyze	RUNSTATS ON TABLE schema.table

Oracle **Gather Statistics** Context Menu example:



To add specific commands for Oracle statistics performed by the scheduler, see ***Statistics***.

A version of a load table is a copy of the metadata definition of the table at the time of the versioning. This version information can be used to create a new load table, or can simply be left as a backup and reference point.
Use **Version Control, New Version** to version a load table. The **Build Application** option allows you to build an application file for the load table and the **Duplicate Object** option allows you to create a new load table as a duplicate of this table.



The **Create (ReCreate)** menu option creates the table in the database based on the definition stored in the metadata. To alter a table select the Validate against database option (see the section on table validation).

The **Truncate** menu option truncates the table.

The **Change Connect/Schema** menu option allows for the rapid changing of the connection information associated with the load table. This information can be changed in-bulk for several load tables. See ***Changing Load Connection and Schema*** (on page 318).

The **Delete Metadata and Drop table** menu option deletes the metadata definition for the table. It also gives you the option to drop the table in the database (dropping the table in the database is the default option). This is a permanent delete and no recovery is provided, so please use with caution. A version of the objects metadata will normally be auto created (depends on settings in **Tools/Options**).

The **Load** menu option performs an interactive load of the data. The method of loading depends on the type of connection. This menu option is intended for use with small data volumes as in a prototype environment. Large data volumes would normally be scheduled. The Load locks the WhereScape RED screen until completed.

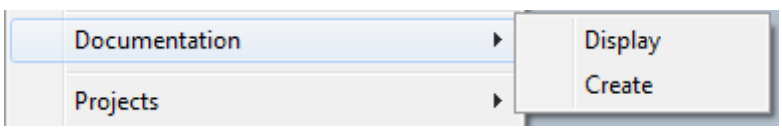
NOTE1: For ODBC based loads in an Oracle data warehouse this interactive load does not use Direct Path loading, so will be slower than a scheduled load.

NOTE2: The load option does not drop or create any indexes. Use the Process option if indexes need to be maintained.

The **Process Table via Scheduler** menu option sends a request to the scheduler to immediately process the load table. This process will drop any indexes marked as pre_drop, load the data and rebuild any required indexes. Control is immediately returned to the user and the loading will occur via the scheduler.

The **Documentation** menu option can be used to generate (or read if already generated) the WhereScape RED HTML documentation for the selected load table.

Two options are available: **Display** and **Create**:

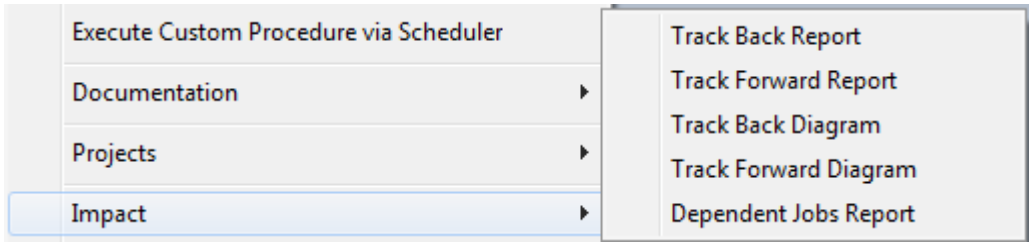


The **Projects** menu option enables you to remove the table from the current project or to add the load table to the current project. The **List Projects** option displays a list of projects which contain the current object, results are shown in the bottom pane. Multiple objects can be selected by double-clicking the **Load Table** icon in the left pane and Ctrl + clicking multiple connections in the Drop Target Pane.

If there aren't any projects in the repository, these options are unavailable.

The **Impact** menu option enables you to produce a number of reports and diagrams:

- **Track Back, Track Forward** or **Dependent Jobs** report
- **Track Back** and **Track Forward** diagrams



The **Code** menu option can be used to view a procedure attached to a table.

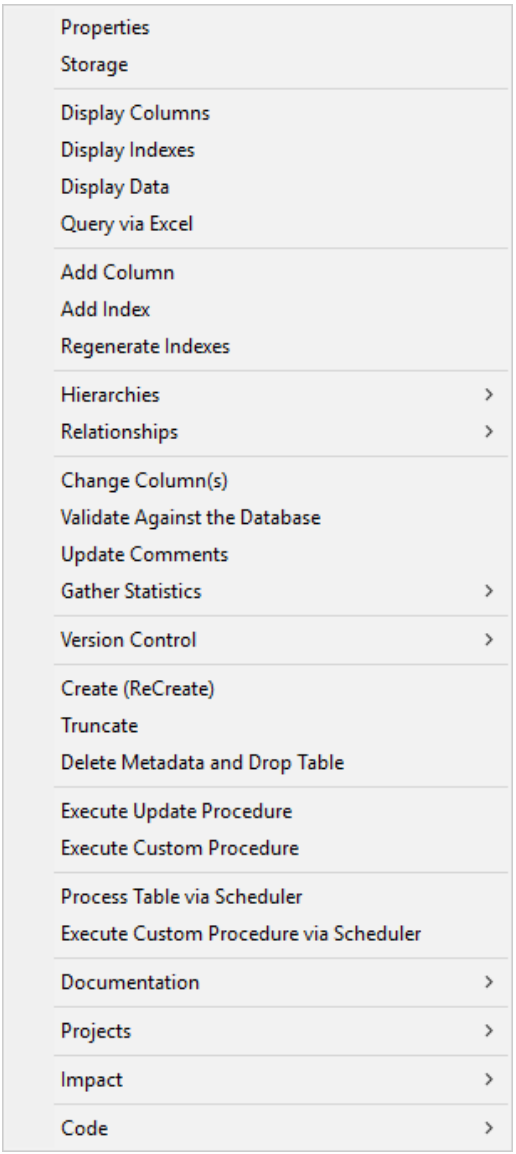
Hover over this option to display an additional menu containing a list of procedures associated with the table:

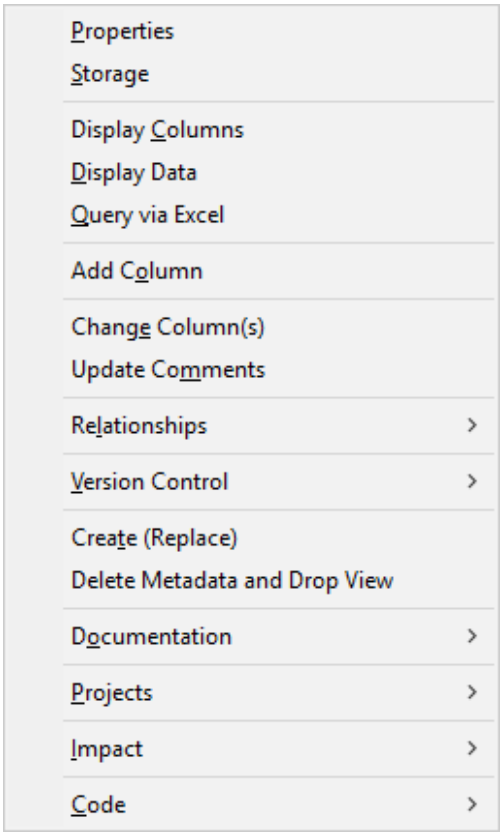
Choose a procedure from the list to open in the procedure editor in view mode.

Note: Only load tables with one or more defined procedures have the **Code view** option.

Dimension Tables

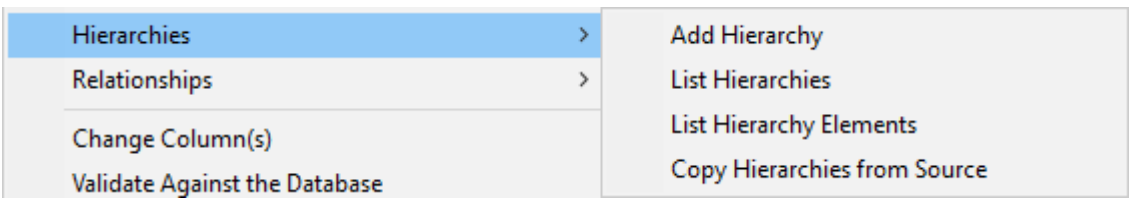
The standard pop-up menus for **dimensions** follow (dimension tables on the left, dimension views on the right):





The bulk of these menu options are the same as for load tables and are described under the load table section above. The differences are:

The **Hierarchies** sub-menu contains the following options:



Hierarchies can be added using **Hierarchies/Add Hierarchy** and listed using **Hierarchies/List Hierarchy**. Hierarchy elements can be listed using **Hierarchies/List Hierarchy Elements**. The **Hierarchies/Copy Hierarchies from Source** features copies all hierarchies from the source table to the destination table. Source hierarchies are copied to the destination table automatically during table creation, but this feature is useful if the source table has been updated since destination table was created.

The **Execute Update Procedure** menu option executes the procedure defined as the 'update procedure' for the table. The procedure is executed interactively and locks the screen until completed. This menu option is only intended for use when working with small/prototype data volumes, and no index handling is performed.

The **Execute Custom Procedure** menu option executes the procedure defined as the 'custom procedure' for the table. As with Update, the procedure is executed interactively.

The **Execute Custom Procedure via Scheduler** menu option executes the procedure defined as the 'custom procedure' for the table, via the Scheduler.

The **Code** menu option can be used to view a procedure attached to a table or to rebuild the table's update procedure; and to view or rebuild the get key function on the Dimension/Dimension View.

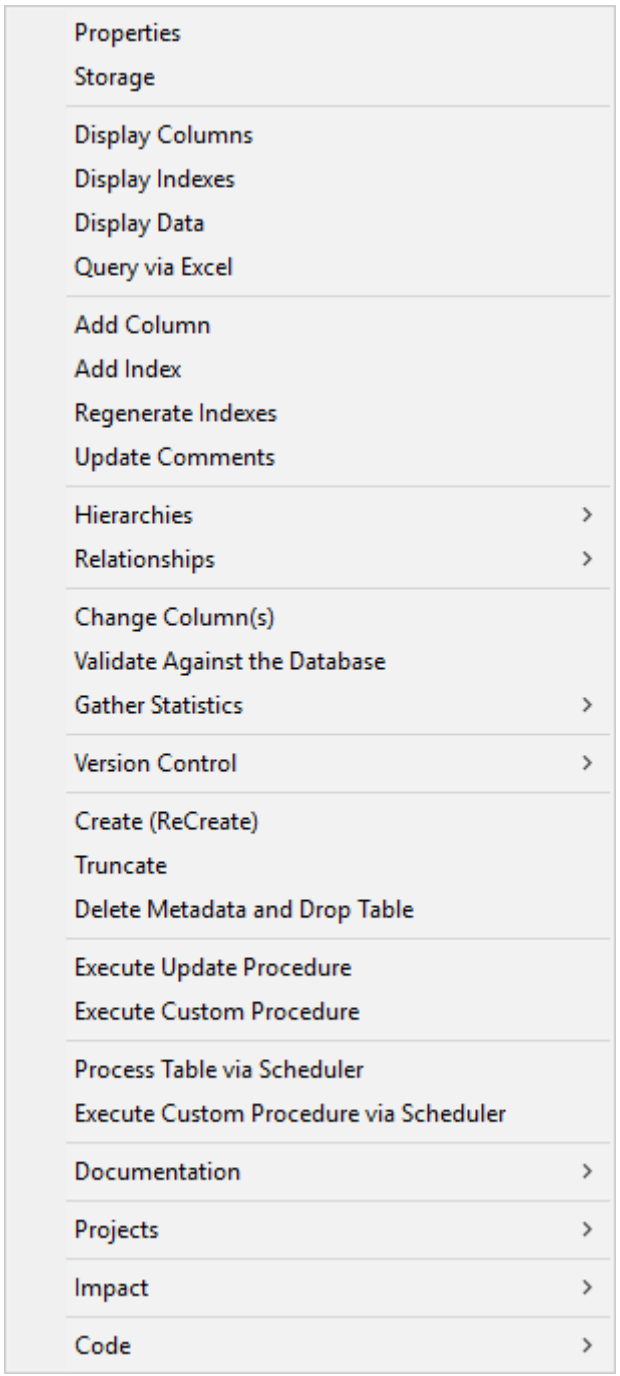
Hover over this option to display an additional menu containing available options:

Choose a procedure from the list to open in the procedure editor in view mode or choose to rebuild the update procedure.

Note: Only tables with one or more defined procedures have the **Code** option.

Data Store Tables

The standard pop-up menus for **data store tables** is:



The menu options available for Data Store Tables is a subset of the options for Dimension Tables, except for an additional option under the code menu.

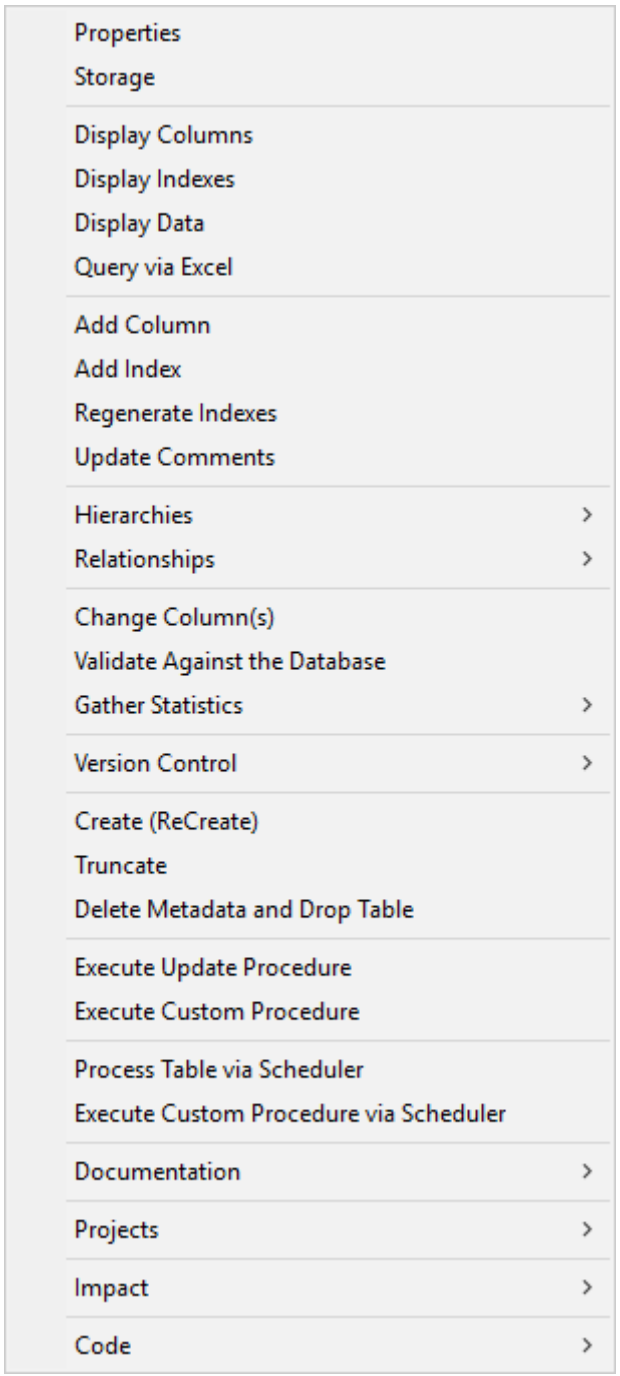
The **Code** menu option can be used to view a procedure attached to a table or to rebuild or regenerate the table's update procedure.

Hover over this option to display an additional menu containing available options:

Choose a procedure from the list to open in the procedure editor in view mode or choose to rebuild or regenerate the update procedure.

EDW 3NF Tables

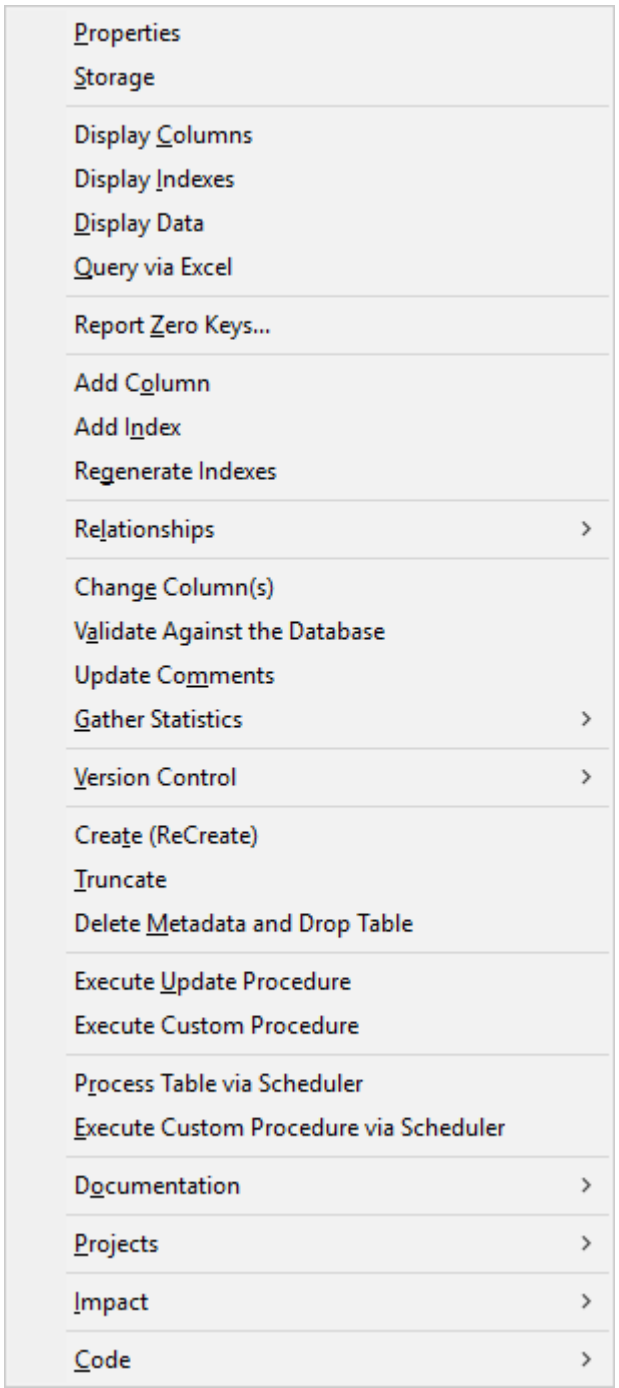
The standard pop-up menu for **EDW 3NF tables** is:



The menu options available for EDW 3NF Tables is a subset of the options for Data Store Tables.

Stage Tables

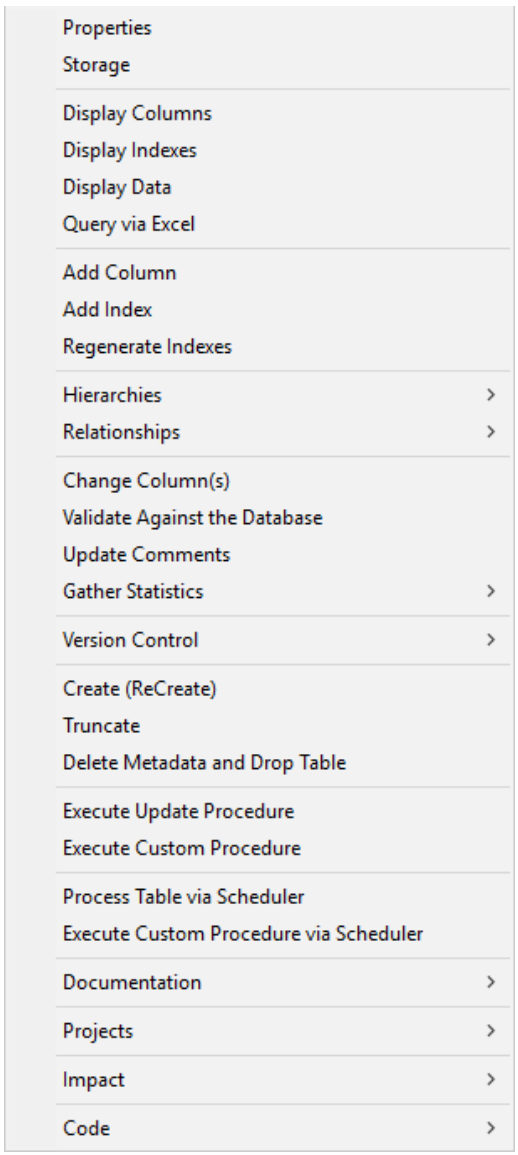
The standard **stage table** menu is as follows:

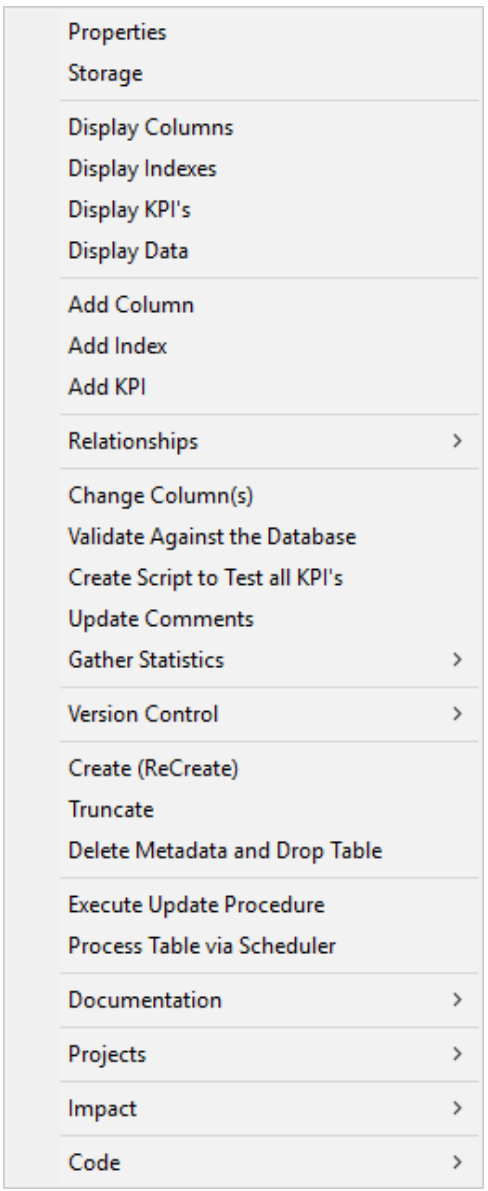


The menu options available for Stage Tables is a subset of the options for Data Store Tables, except for the **Report Zero Keys...** element. This option initiates the report ***Records that Failed a Dimension Join*** (on page 879).

Fact Tables

The standard pop-up menus for **fact tables** follow (fact tables on the left, KPI fact tables on the right):





The menu options available for Fact Tables are a subset of the options for Data Store Tables. KPI Fact Tables have three additional menu options:

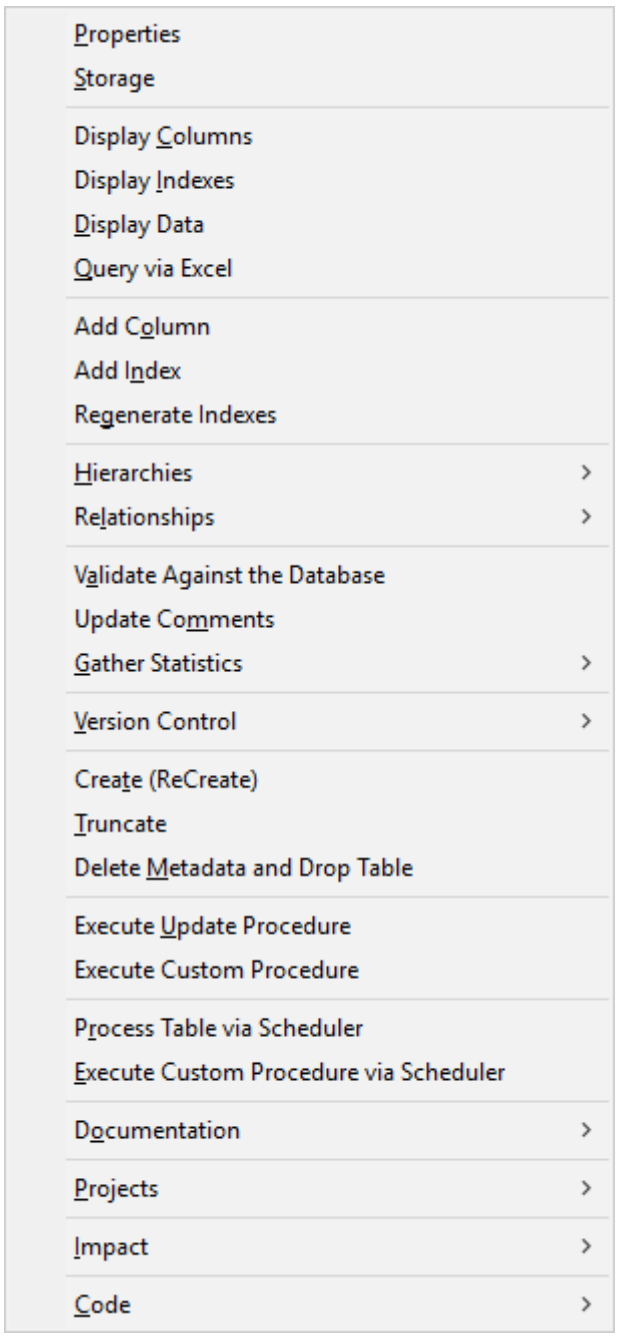
The **Display KPI's** menu option displays all KPIs that have been set up for the KPI fact table in the drag and drop pane Drop Target Pane.

To add a new KPI, select the **Add KPI** menu option.

The **Create Script To Test All KPI's** menu option generates a sql script to test the KPI definitions for a specified time window.

Aggregate Tables

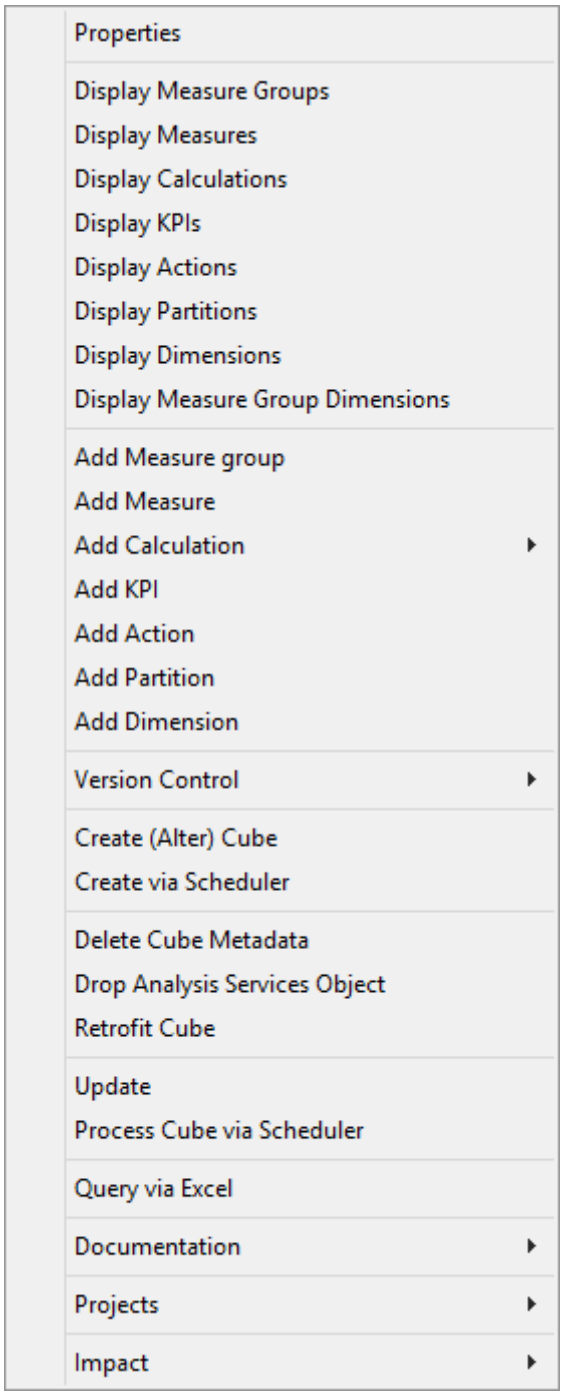
The standard **aggregate table** menu is as follows:



The menu options available for Aggregate Tables is a subset of the options for Data Store Tables.

OLAP Cubes

The standard pop-up menu for **OLAP Cubes** is:



The **Properties** menu option opens the Properties dialog that defines cube creation options and access to documentation tabs.

The **Display Measure Groups** menu option shows the details of the measure groups associated with the cube in the Drop Target Pane.

The **Display Measures** menu option lists the measures associated with the measure groups in the cube. This is the default view in the Drop Target Pane when a cube is selected in the left pane with a single click.

The **Display Calculations** menu option lists all the calculated members defined in the cube.

The **Display KPIs** menu option lists all the Key Performance Indicators defined in the cube.

The **Display Actions** menu option lists all the actions defined in the cube.

The **Display Partitions** menu option lists all the partitions defined against the related measure groups within the cube.

The **Display Dimensions** menu option lists all of the dimensions defined in the cube.

The **Display Measure Group Dimensions** menu option displays a cross tab report in the Drop Target Pane showing cube dimensions relating cube measure groups.

The **Add Measure group** menu option allows a new measure group to be added to the cube.

The **Add Measure** menu option adds another measure to the cube.

The **Add Calculation** menu option adds a new calculated member to the cube.

The **Add KPI** menu option adds a new KPI to the cube.

The **Add Action** menu option adds a new action to the cube.

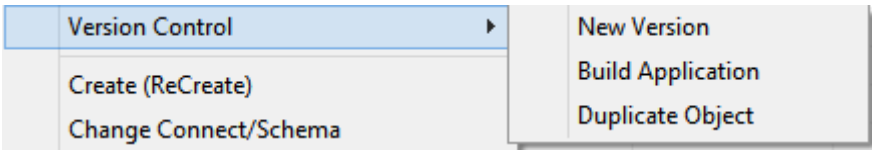
The **Add Partition** menu option adds a new partition to a measure group in the cube.

The **Add Dimension** menu option adds an existing OLAP Dimension to the cube.

A version of an OLAP cube is a copy of the metadata definition of the cube at the time of the versioning. This version information can be used to create a new OLAP cube, or can simply be left as a backup and reference point.

Use **Version Control, New Version** to version an OLAP cube.

The **Build Application** option allows you to build an application file for the OLAP cube and the **Duplicate Object** option allows you to create a new OLAP cube as a duplicate of this OLAP cube.



The **Create (Alter) Cube** menu option creates the cube and supporting objects in Analysis Services (including cube database, data source view (DSV) and dimensions) based on the definition in WhereScape RED.

The **Create via Scheduler** menu option submits a create of the OLAP Cube via the Scheduler.

The **Delete Cube metadata** menu option deletes the cube definition from WhereScape RED.

The **Drop Analysis Services object** menu option drops the selected object in Analysis Services.

The **Retrofit Cube** menu option retrofits the OLAP cube from Analysis Services.

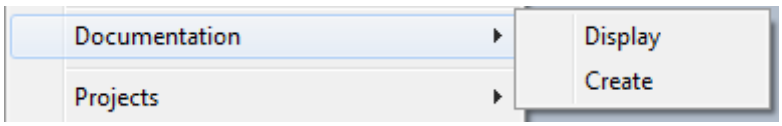
The **Update** menu option processes the cube in Analysis Services interactively from the WhereScape RED interface.

The **Process Cube via Scheduler** menu option generates a WhereScape RED scheduler job to process the cube in Analysis Services.

The **Query via Excel** menu option opens up an .oqy file in Microsoft Excel.

Note: Due to a shortcoming in the Microsoft Office installation it may be necessary to associate the .oqy file extension with Microsoft Excel before this option will succeed.

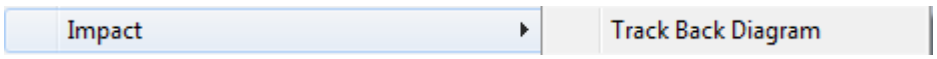
The **Documentation** menu option can be used to generate (or read if already generated) the WhereScape RED HTML documentation for the selected olap cube.
Two options are available: **Display** and **Create**:



The **Projects** menu option enables you to remove the olap cube from the current project or to add the olap cube to the current project. The **List Projects** option displays a list of projects which contain the current object, results are shown in the bottom pane. Multiple objects can be selected by double-clicking the **Olap Cube** icon in the left pane and Ctrl + clicking multiple connections in the Drop Target Pane.

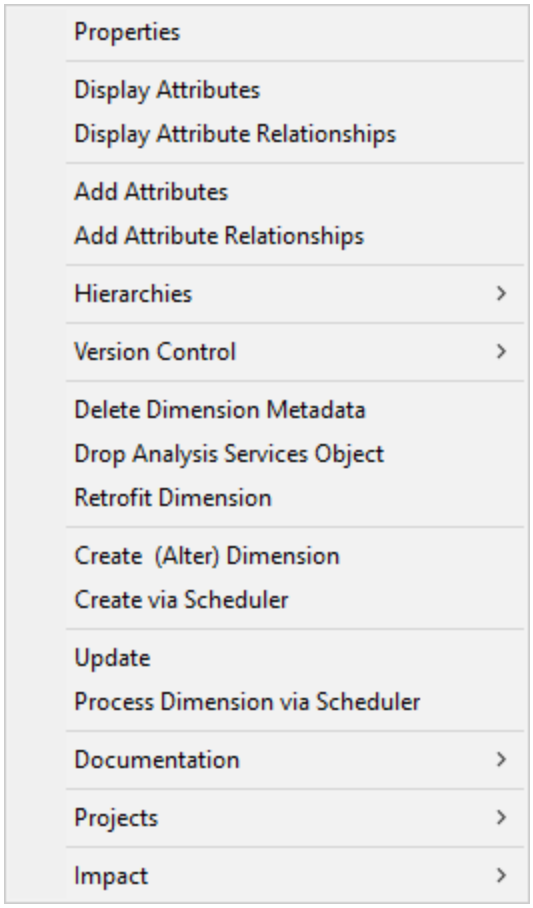
If there aren't any projects in the repository, these options are unavailable.

The **Impact** menu option enables you to produce a **Track Back** diagram on the olap cube.



OLAP Dimensions

The standard pop-up menu for an **OLAP Dimension** is:



The **Properties** menu option displays the OLAP dimension Properties dialog which includes documentation tabs.

The **Display Attributes** menu option lists the attributes for the selected dimension. This is the default view when an OLAP Dimension is selected in the left pane.

The **Display Attribute Relationships** menu option shows the relationships between the dimensional attributes.

The **Add Attributes** menu option adds a new attribute to the dimension.

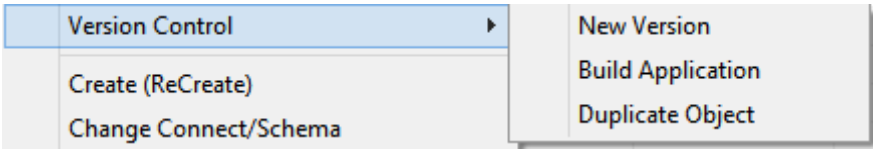
The **Add Attribute Relationships** menu option adds an attribute relationship for the selected dimension.

The **Hierarchies** sub-menu is the same as the Hierarchies sub-menu for Dimensions, with the following addition: The **Add Hierarchy Element** menu option adds a new hierarchy element.

A version of an OLAP Dimension is a copy of the metadata definition of the OLAP Dimension at the time of the versioning. This version information can be used to create a new OLAP Dimension, or can simply be left as a backup and reference point.

Use **Version Control, New Version** to version an OLAP Dimension.

The **Build Application** option allows you to build an application file for the OLAP Dimension and the **Duplicate Object** option allows you to create a new OLAP Dimension as a duplicate of this OLAP Dimension.



The **Delete Dimension metadata** menu option will delete the cube definition from WhereScape RED metadata.

The **Drop Analysis Services object** menu option provides the ability to drop the selected object from Analysis Services.

The **Retrofit dimension** menu option retrofits the OLAP Dimension from Analysis Services.

The **Create (Alter) dimension** menu option creates the OLAP Dimension and supporting objects in Analysis Services (including cube database and dsv) based on the definition in WhereScape RED. This option requires connection and cube database information populated in the OLAP Dimension Properties.

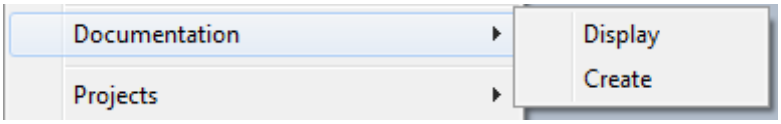
The **Create via Scheduler** menu option submits a create of the OLAP Dimension via the scheduler.

The **Update** menu option processes the OLAP Dimension in Analysis Services interactively from the WhereScape RED interface. This option requires connection and cube database information populated in the OLAP Dimension Properties.

The **Process Dimension via Scheduler** menu option generates a WhereScape RED scheduler job to process the OLAP Dimension in Analysis Services. This option requires connection and cube database information populated in the OLAP Dimension Properties.

The **Documentation** menu option can be used to generate (or read if already generated) the WhereScape RED HTML documentation for the selected OLAP dimension.

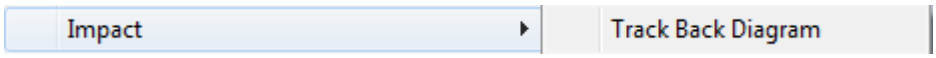
Two options are available: **Display** and **Create**:



The **Projects** menu option enables you to remove the olap dimension from the current project or to add the OLAP dimension to the current project. The **List Projects** option displays a list of projects which contain the current object, results are shown in the bottom pane. Multiple objects can be selected by double-clicking the **Olap Dimension** icon in the left pane and Ctrl + clicking multiple connections in the Drop Target Pane.

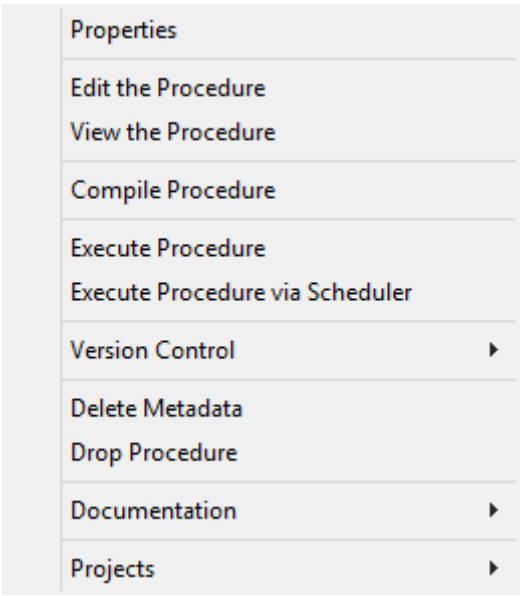
If there aren't any projects in the repository, these options are unavailable.

The **Impact** menu option enables you to produce a **Track Back** diagram on the olap cube dimension.



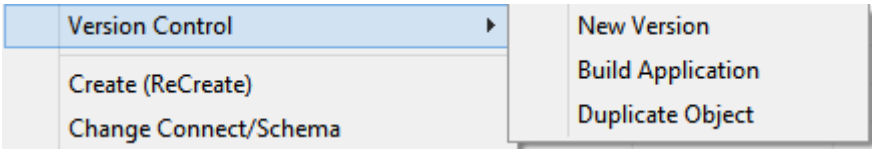
Procedures

Procedures are commonly auto built through the Properties screen of one of the table types. They can also be created manually.
Once created they can be edited, compiled etc. The pop-up menu displayed when you right-click on a procedure name is as follows:



If a procedure has been locked because of the WhereScape RED utility being killed or failing or a database failure, then it can be unlocked via the Properties screen associated with the procedure.
The **Edit the Procedure** menu option invokes the procedure editor and loads the procedure. A procedure can be compiled and executed within the procedure editor.
The **View the Procedure** menu option displays a read only copy of the procedure. If the procedure is locked by another user, then viewing the procedure is the only option available.
The **Compile Procedure** menu option will compile the procedure from the metadata.
The **Execute Procedure** menu option executes the procedure and displays the results in the results window.
The **Execute Procedure via Scheduler** menu option sets up a job to execute the procedure via the scheduler.

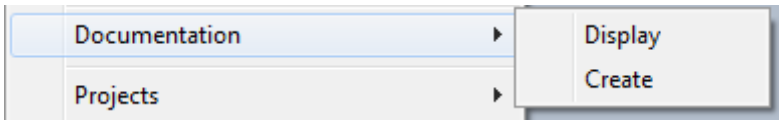
A version of a Procedure can be created at any time via the **Version Control**, **New Version** menu option. The various versions of the procedure can be viewed from within procedure editor, or a new procedure can be created from the version.
The **Build Application** option allows you to build an application file for the Procedure and the **Duplicate Object** option allows you to create a new Procedure as a duplicate of this Procedure.



Selecting the **Delete metadata** menu option will delete the procedure from the metadata. It then asks if the procedure should also be dropped from the database.
Selecting the **Drop Procedure** menu option will drop the procedure from the meta data and from the database as well.

NOTE: Procedures cannot be dropped or deleted if they are protected by, for example, an **Edit Lock**.

The **Documentation** menu option can be used to generate (or read if already generated) the WhereScape RED HTML documentation for the selected procedure.
Two options are available: **Display** and **Create**:

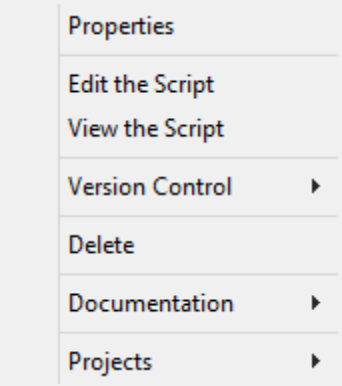


The **Projects** menu option enables you to remove the procedure from the current project or to add the procedure to the current project. The **List Projects** option displays a list of projects which contain the current object, results are shown in the bottom pane. Multiple objects can be selected by double-clicking the **Procedure** icon in the left pane and Ctrl + clicking multiple connections in the Drop Target Pane.

If there aren't any projects in the repository, these options are unavailable.

Scripts

Scripts are very similar in their operations to procedures.
The same menu options are available as for procedures and perform the same functionality.

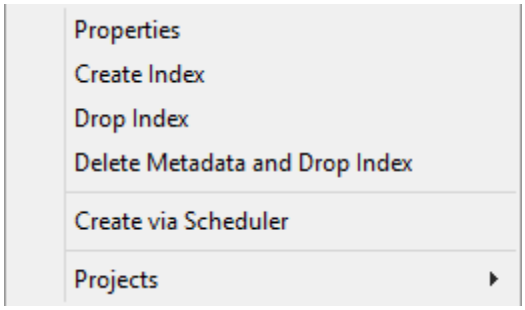


If a script is deleted it is only removed from the metadata. However, as WhereScape RED never stores the script on the host system, this will essentially remove the script permanently.

Indexes

Indexes are always associated with a table. To define a new index the menu option associated with the table that is to have the index must be used.

Once defined the following operations can be performed:



The **Properties** menu options displays the Properties screen, which contains the entire definition of the index, including the columns in use and the index type etc. The way that the scheduler handles the index is also defined in the Properties. An index can be set so that it will be dropped by the scheduler prior to a table update and then rebuilt by the scheduler once the update has been completed. It can also be defined for rebuild on certain days.

The **Create Index** menu option creates the index in the database. This may take some time for large indexes, and in such cases it would be better to schedule a create of the index. See the chapter on the Scheduler if such an activity is required. This menu option is intended for use when working with prototype data volumes.

The **Drop Index** menu option drops the index in the database.

The **Delete Metadata and Drop Index** menu option removes the metadata definition of the index and drops it from the database. No recovery is possible once this option is actioned.

The **Create via Scheduler** menu option submits a create of the index to the scheduler.

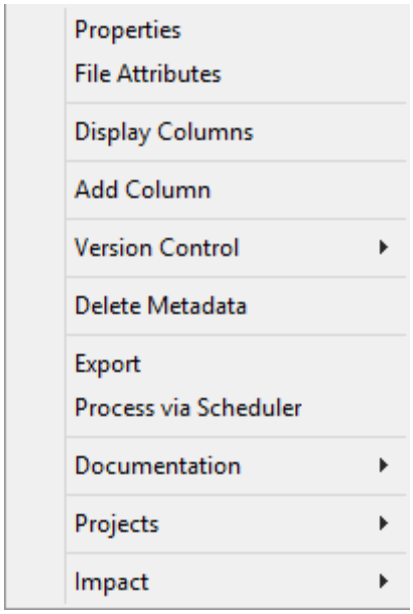
The **Projects** menu option enables you to remove the index from the current project or to add the index to the current project. The **List Projects** option displays a list of projects which contain the current object, results are shown in the bottom pane. Multiple objects can be selected by double-clicking the **Index** icon in the left pane and Ctrl + clicking multiple connections in the Drop Target Pane.

If there aren't any projects in the repository, these options are unavailable.

Exports

Export objects are used in WhereScape RED to produce ascii files from a single database table or view for a downstream feed. Some or all of the columns in a table or view can be exported.

The pop-up menu displayed when you right-click on an export is as follows:



File Attributes launches the properties window for the export table, focusing on the File Attributes tab within this window.

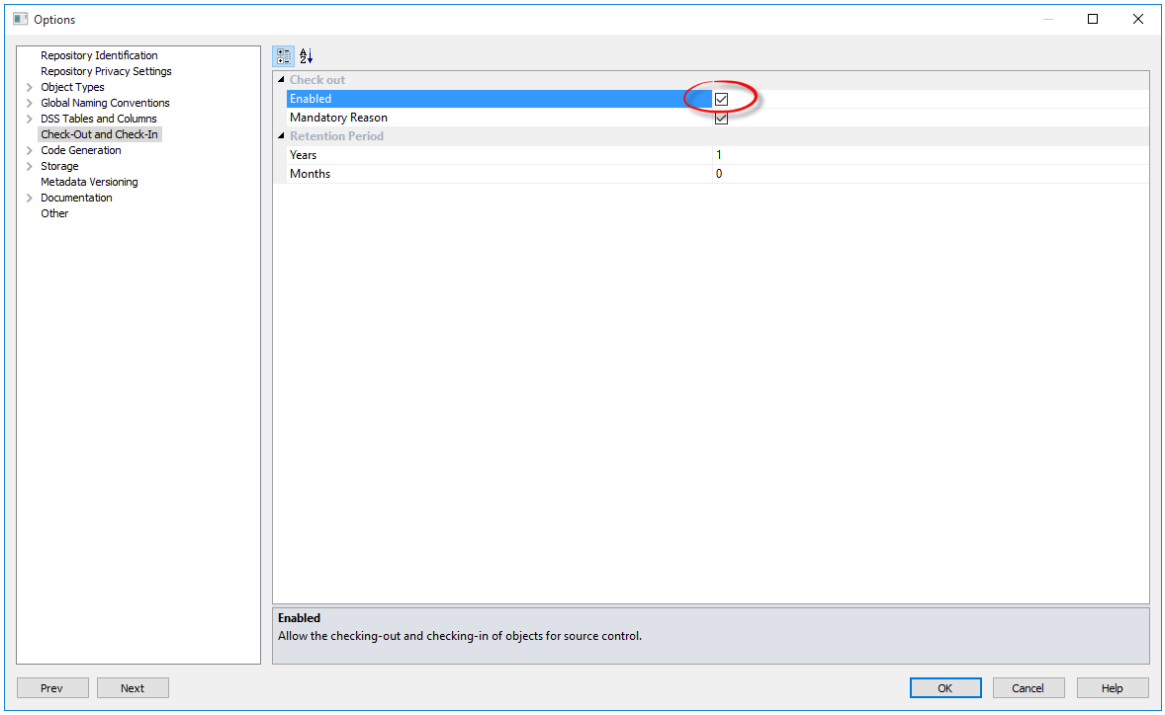
All the other options for export objects are described above under other object types.

OBJECT CHECK-OUTS AND CHECK-INS

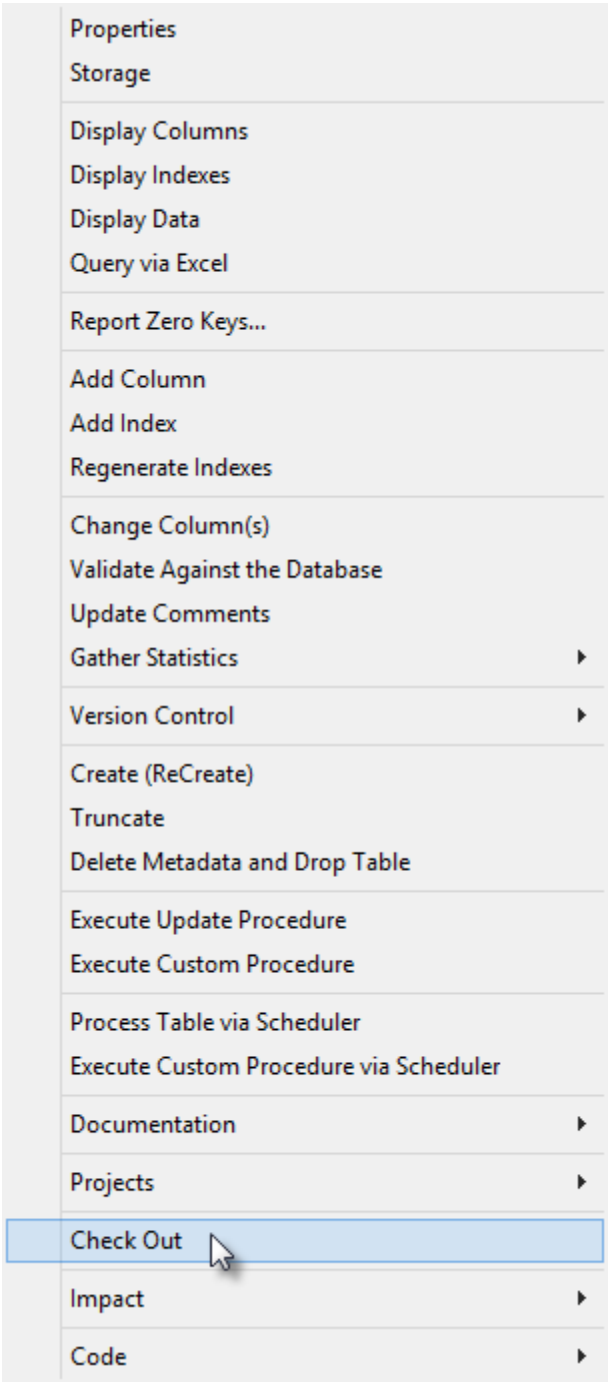
Objects in RED can be checked out for editing to prevent any other users from being able to modify, update or delete any of their associated objects while you are making changes to them.

To use this functionality, it must first be enabled through the **Tools->Options**.

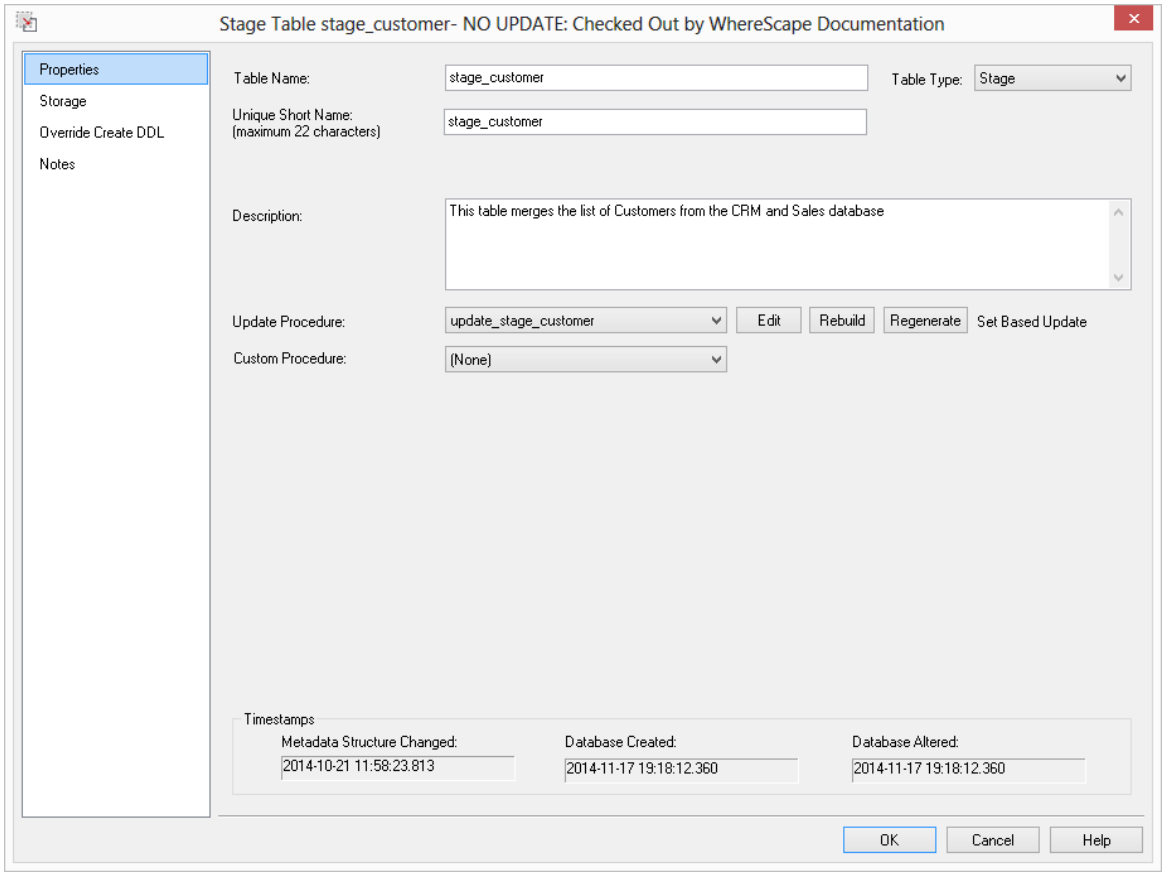
- Click **Check-Out and Check-In** and enable the Check-Out option.



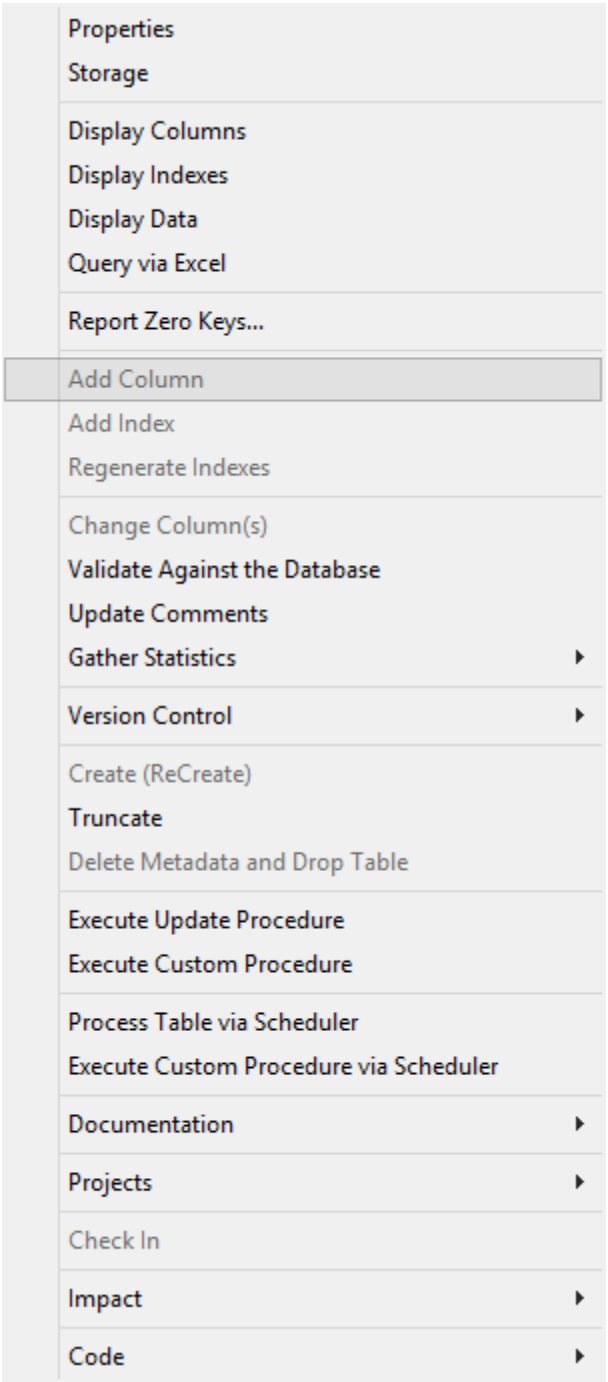
Once this is enabled in Tools/Options, users can check-out/check-in objects in RED through that object's right-click context menus.



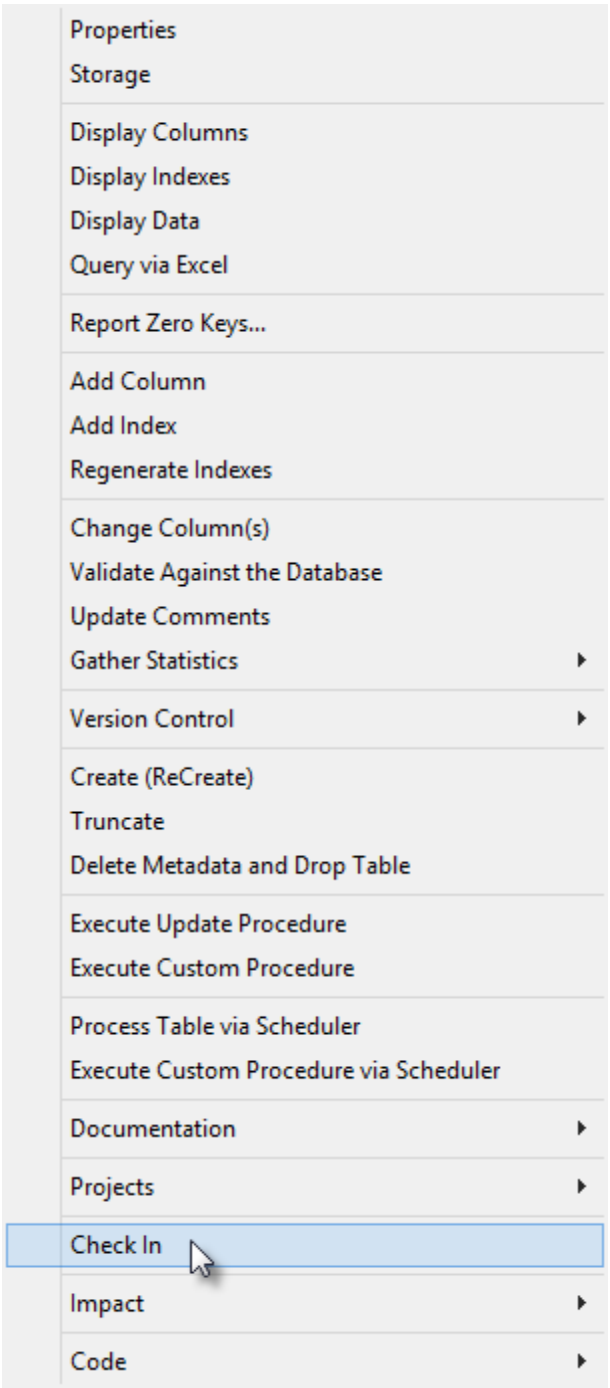
Checked out objects and their associated procedures and scripts cannot be modified, updated or deleted by other users. If a different user tries to access the properties screen or procedure windows for those objects, their fields will be disabled and headed NO UPDATE: Checked Out by (user name).



Other users logging in to the same metadata repository will have some of the checked-out object's context menu options disabled as shown below.

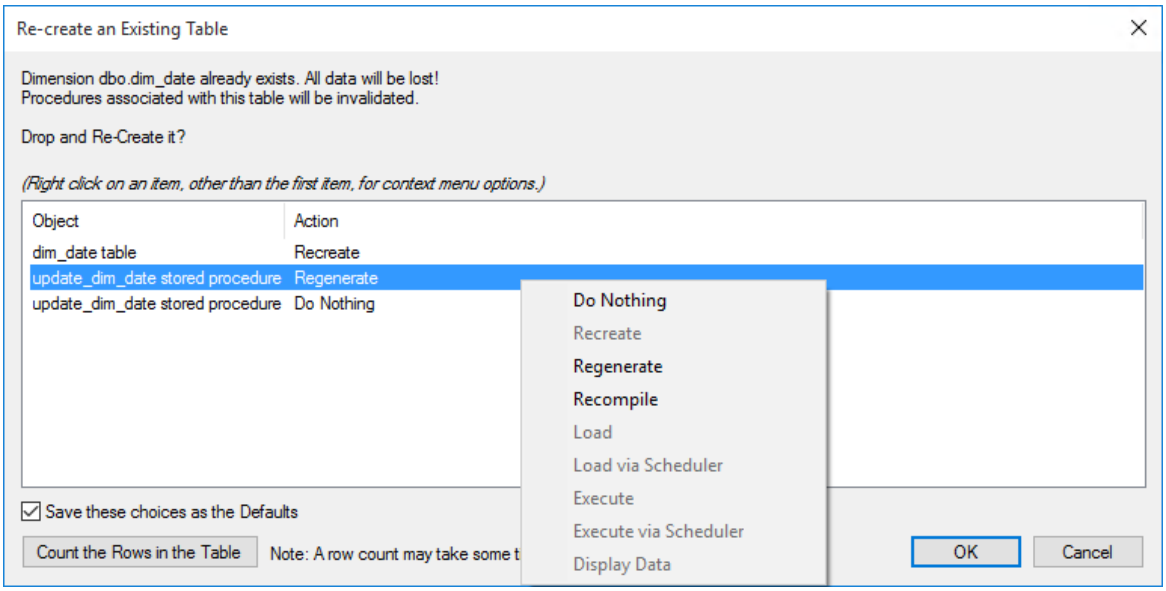


To check-in a checked-out object, simply right-click on it and select Check In from the context menu.



RE-CREATE DIALOG

The Re-create dialog provides available options for dropping and recreating an existing table and associated procedures. Objects and actions available depend on the table being recreated.



Actions in the table are completed from top to bottom; the action performed on each object can be set as desired:

Object(s)	Action	Description
Table	Recreate	Recreate the table structure only. Do not make changes to associated procedures. This action is mandatory.
Artificial key sequence (Oracle only)	Do Nothing	Do not recreate the key sequence. Skip to the next action.
	Recreate (Default)	Recreate the artificial key sequence. Do not make changes to associated procedures.
Update procedure	Do Nothing	Do not regenerate the update procedure, skip to the next action.
	Regenerate (Default)	Recreate the table structure, regenerate the procedure definition in metadata based on the current table and column properties, and recreate the procedure in the database. This is the option to use if there have been changes to column or table properties, e.g. column transformations.
	Recompile	Recreate the table structure and recreate the update procedure based on the existing procedure definition in the metadata.
Load tables	Do Nothing (Default)	Do nothing for this object, skip to the next action.

Object(s)	Action	Description
	Load	Perform an interactive load of the data into the table. WhereScape recommends performing this via the Scheduler for large tables.
	Load via Scheduler	Add a data load task to the scheduler. Useful for large tables where processing may take some time.
Update procedure	Do Nothing (Default)	Do nothing for this object, skip to the next action.
	Execute	Execute the update procedure. WhereScape recommends performing this via the Scheduler for large tables.
	Execute via Scheduler	Execute the update procedure via the scheduler. Useful for large tables where processing may take some time.
Tables	Do Nothing	Do not display the data.
	Display Data	Display the table data after recreating the table and update procedure (if applicable). This is the default setting for Views and Dimension Views. This is the default setting for tables when an update procedure is present and has been executed in the previous action. Unavailable when an update procedure is present but has not been executed in the recreate dialog.

Save these choices as the Defaults

This option can be used to store the current actions as the default choices for all object types. The Recreate dialog is then automatically populated with the saved actions each time it appears.

Count the Rows in the Table

Displays the total number of rows in the table at the bottom of the Recreate dialog. Knowing the number of rows in a table can help estimate how long Load and Execute actions may take.

ORGANIZING OBJECTS

As mentioned in the previous section, there are many object types in WhereScape RED.
The objects in the metadata repository are displayed in the left pane of the Builder window. They are displayed in a tree structure which can be expanded and closed as required.
The tree can be refreshed by using **F5** or the **Ctrl/R** key.

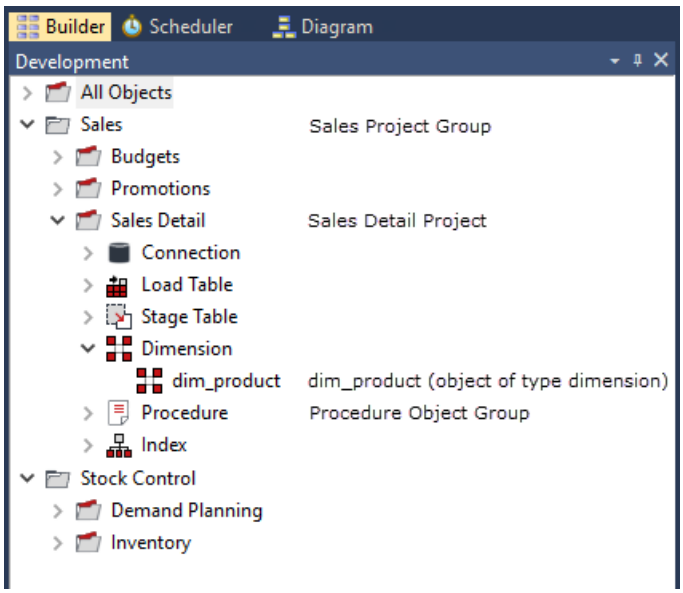
Object Groups

The objects created in WhereScape RED are grouped together into object groups based on object type.
For example, we store all the dimension objects in the Dimension **object group**. Optionally we can choose to display dimension tables as the **Dimension** object group and dimension views as the **Dimension View** object group.

Projects

These object groups are in turn stored within **projects**. When WhereScape RED is first started, the special project called **All Objects** is the only project. This project will always contain all the objects that exist in the metadata repository.
Additional projects can be created if desired. These additional projects can hold some or all of the objects as seen in **All Objects**. An object as such only exists once in the metadata. Therefore, if we have a dimension object called dim_product, there is and can only be one copy of the object dim_product within the metadata.
Projects are used to hold a group of objects that relate back to a similar module or analysis area of the data warehouse.
In the example below we have additional projects called Budgets, Promotions, Sales Detail, Demand Planning and Inventory. In this way projects allow us to restrict the amount of information (objects) we need to deal with to just those relevant to the area being worked on.
Projects can also be used to group objects for an upcoming code promote.

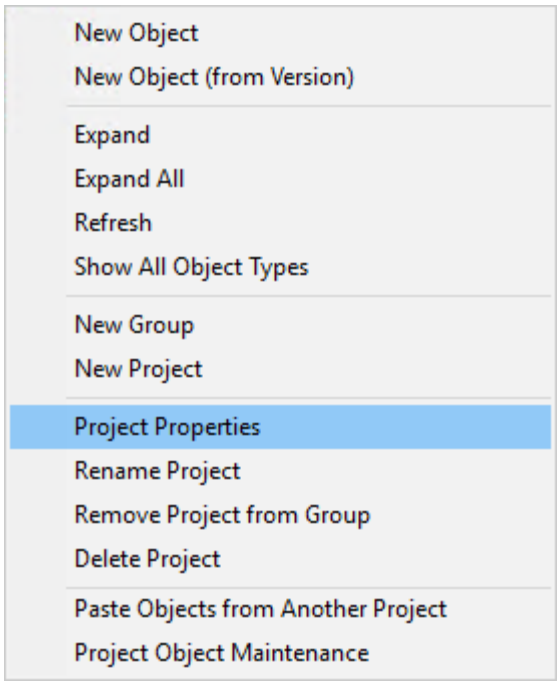
The example below shows a meta repository using two project groups (Sales and Stock Control) and five additional projects (Budgets, Promotions, Sales Detail, Demand Planning and Inventory).



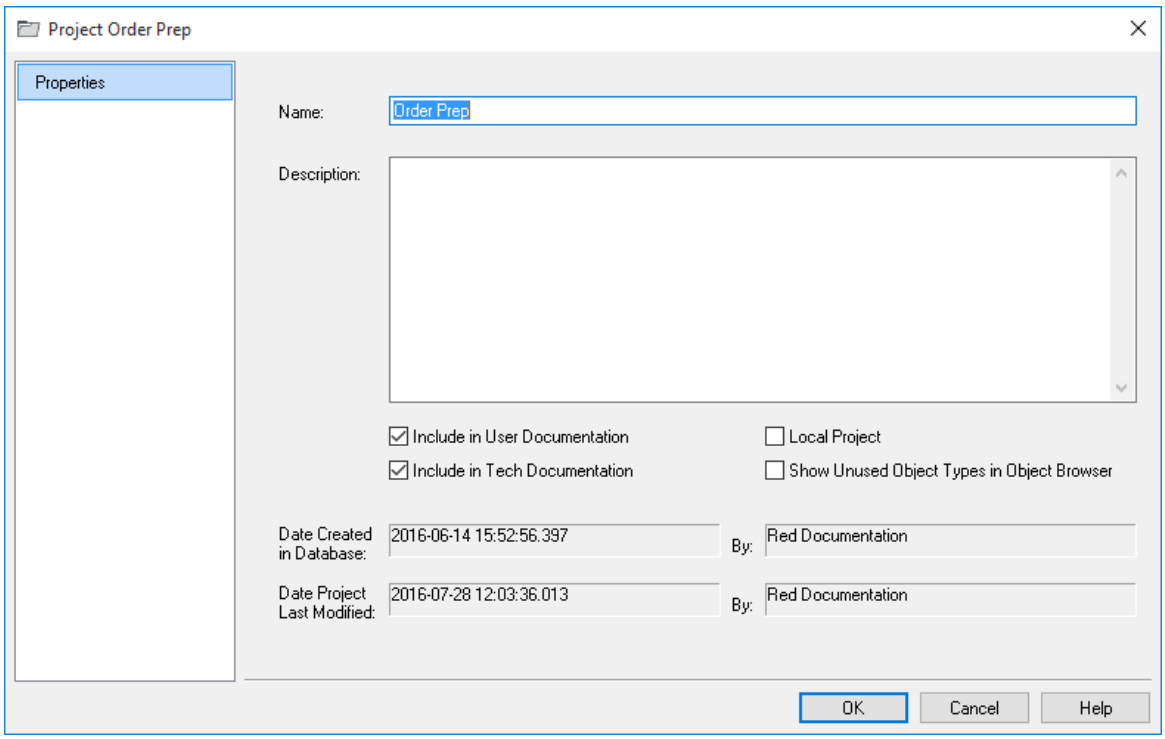
Note: If you delete an object using the right-click menu, then the object will be deleted from the metadata and will be removed from all Projects. Remove the object from the project instead of deleting it.

It is important to understand that these projects are only a means of visualizing the objects. Even though an object may appear in many projects, it only exists *once* in the metadata.
To create a project, right-click in the left pane below the last project and select **New Project**.
The **File/New Project** menu option may also be used. Projects can be renamed, removed from the project group or deleted by using the right-click menu when positioned on the project name.
Deleting a project does not delete the objects from the metadata; it simply removes their reference in the project being deleted.

To view or make changes to the Project's Properties; right-click on the project name and select **Project Properties**.



The following screen shows four check-boxes.



Include in User Documentation - When checked, the project will be included in the User Documentation. The default state is checked.

Include in Tech Documentation - When checked, the project will be included in the Technical Documentation. The default state is checked.

Local Project - When checked as local, the project will not be included in the Application files. The default state is unchecked.

Show Unused Object Types in Object Browser - When checked, all object types are displayed in the Object Browser Pane. When unchecked, only object types currently in use are displayed. The default state is unchecked. Refreshing the Object Browser Pane is required after changing this setting.

Project Groups

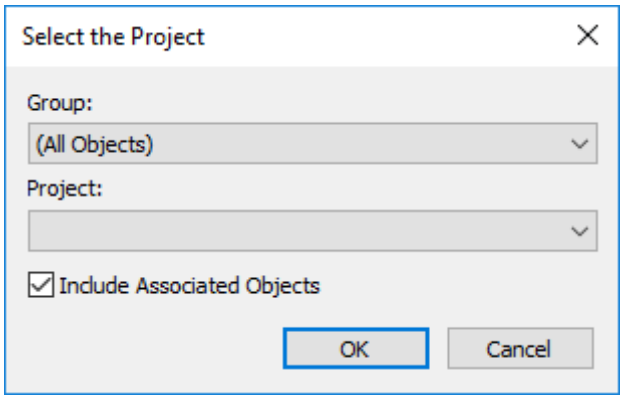
Projects can in turn be grouped together into Groups. A project must only appear in one Project Group.
To create a Project Group, use the **File/New Group** menu option. Project Groups can be removed by using the right-click menu when positioned on a group name.

ADDING OBJECTS TO PROJECTS

There are several different ways to add objects to a project:

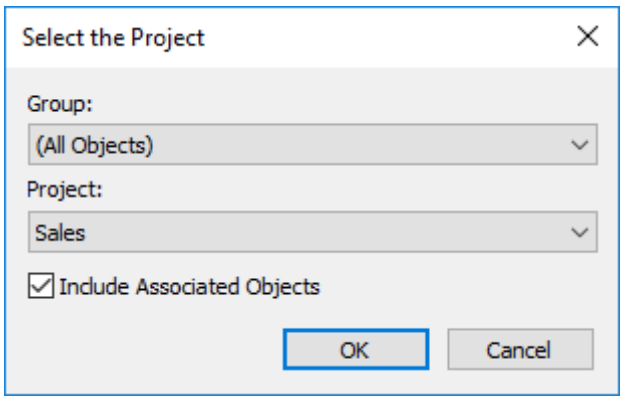
- Click an object in the left pane, type **Ctrl/C**, click the target project (either the project folder or any object group or object within it) and type **Ctrl/V**
- **Drag** the object to the required project
- Right-click on an object in the left pane and select **Projects/Add to Project**
- Highlight a number of objects in the middle pane, right-click and select **Projects/Add to Project**
- ***Using the Project/Object Maintenance Facility*** (see "***Using Project Object Maintenance***" on page 60)

Both options above that use the right-click **Projects/Add to Project** menu result in the following dialog box being displayed:

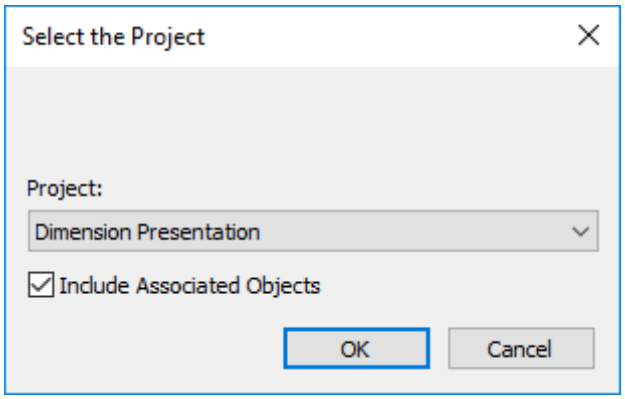


To add an object to an independent project (a project that is not in a group)

Choose the required project from the project drop-down list and click **OK**.

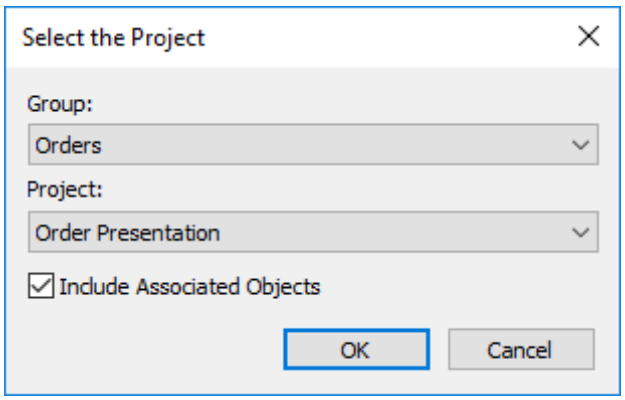


If Groups have been created, both drop-down lists will be visible. If Groups have not been created, only the project drop-down will be visible like this:



To add an object to a dependent project

1. Choose the required **Group** from the group drop-down list.
2. Choose the required **Project** from the project drop-down list.
3. Click **OK**.



Note: The **Include Associated Objects** check-box on the above dialog boxes, will also add any indexes, procedures and scripts to the selected project.

REMOVING OBJECTS FROM PROJECTS

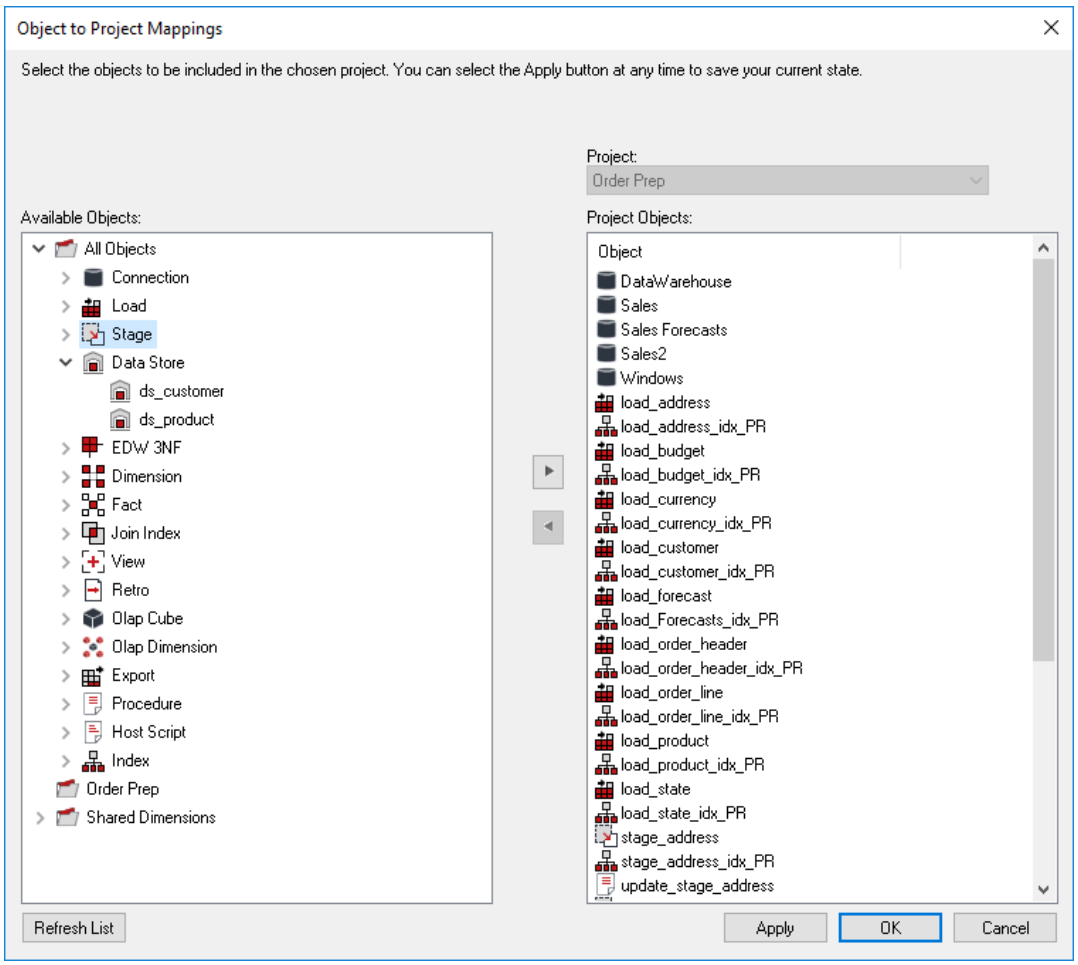
There are several different ways to remove objects from a project:

- **Drag** the object to the blank area at the bottom of the pane
- **Drag** the object into the middle pane
- **Drag** the object to the **All Objects** project
- Right-click on an object in the left pane and select **Projects/Remove from Project**
- Highlight a number of objects in the middle pane, right-click and select **Projects/Remove from Project**
- ***Using the Project/Object Maintenance Facility*** (see "***Using Project Object Maintenance***" on page 60)

USING PROJECT OBJECT MAINTENANCE

The Project Object Maintenance Facility is invoked by right-clicking on a project and choosing **Project Object Maintenance**.

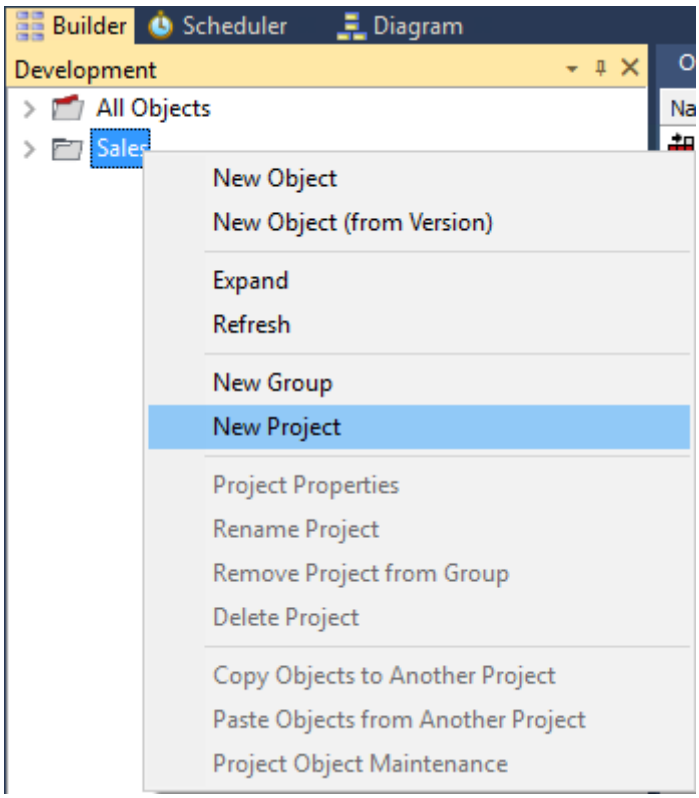
The following dialog is displayed:



- To **add** objects to a project, move objects from the left to the right using the **>** button. By default, associated object (procedures, scripts and indexes) are also moved to the right. These can be manually removed if not required.
- To **remove** objects from a project, move objects from the right to the left using the **<** button.
- When done, click **OK**.
- Clicking **Apply** will update object/project relationships without having to exit the Project Object Maintenance Facility.
- Clicking **Refresh** will refresh the object tree in the left pane.

ADDING PROJECTS TO GROUPS

The best way to move a project into a group is to create the project in the group in the first place. This is done by right-clicking on the group and selecting **New Projec**t:



The other option is to move an existing project from another group into this group. See ***Moving Projects within Groups*** (on page 62)

REMOVING PROJECTS FROM GROUPS

A project can be removed from a group by:

- **Dragging** the project to the blank area at the bottom of the pane
- **Dragging** the project into the middle pane
- **Dragging** the project to the **All Objects** project
- **Removing** the project using right-click **Remove Project from Group**
- **Deleting** the project using right-click **Delete Project**

Note: The first four methods move the project out from the group to become an independent project. The last option removes the project from the metadata repository.

MOVING PROJECTS WITHIN GROUPS

Note: An object can be in any number of projects, but a project can only be in one Group.

Performing a drag from one group to another group will simply create an additional project->group mapping.
To move a project from one group to another group:

1. '**Copy**' the project by dragging the project from the one group to the other group.
2. **Remove** the project from the original group by right-clicking and selecting **Remove Project from Group**.

LIST PROJECT MEMBERSHIPS FOR AN OBJECT(S)

To list projects that contain a specific object, right click on the object in the left pane and select **Project>List Projects**. Results are shown in the bottom pane.

Multiple objects of one type can be selected by double-clicking the **Object Type** icon in the left pane (i.e. **Connection**, **Load Table**, etc.) and Ctrl + clicking multiple connections in the middle pane.

WINDOWS AND PANES

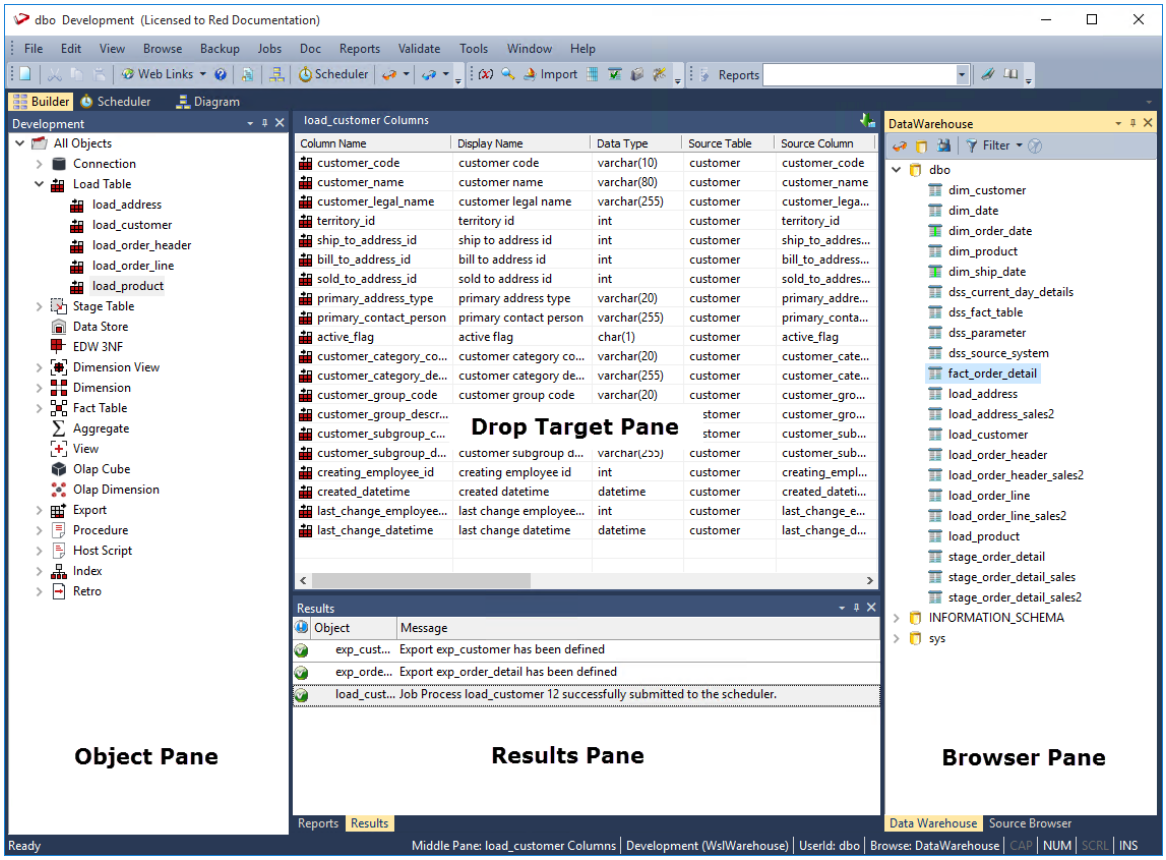
WhereScape RED has a number of different windows that are utilized in the building and maintenance of a data warehouse. Each window may in some cases be broken into panes.

There are four main windows that are used extensively in the building of a data warehouse:

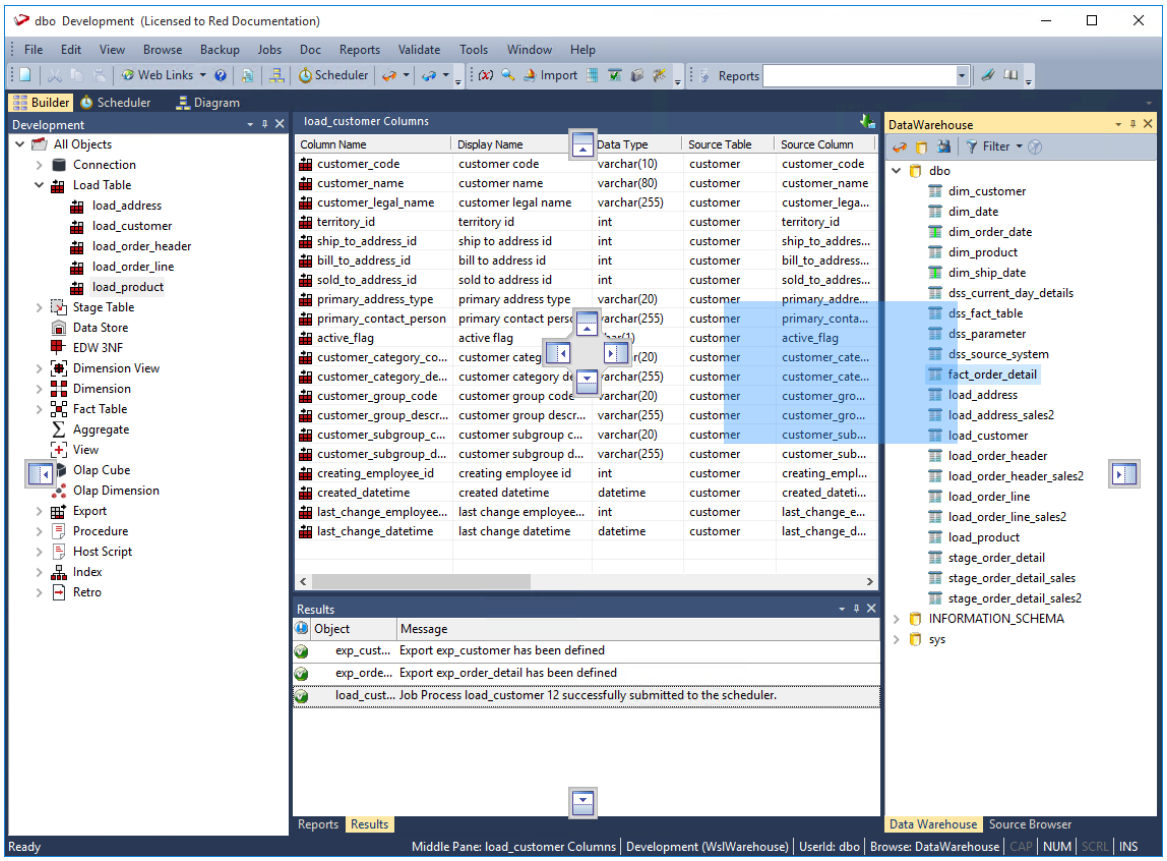
- The **Builder** window
- The **Scheduler** window
- The **Diagram** window
- The **Procedure Editor** window

BUILDER WINDOW

The **Builder Window** has four panes.



The left, bottom and right panes can be dragged out of their docking places and docked elsewhere.
Docking handles appear when a pane is dragged.

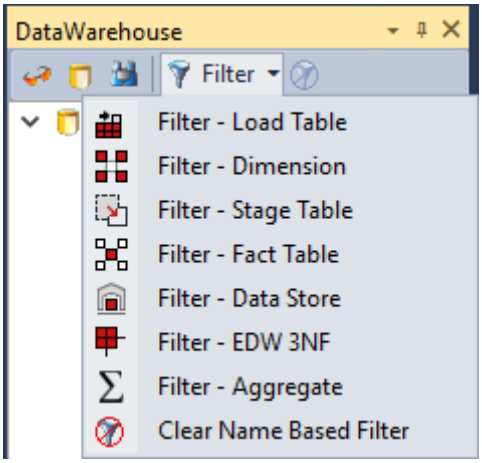


The **left pane** contains all the **objects** within the metadata repository. These objects are stored in object groups (e.g. Dimension).
The object groups are in turn optionally stored in Projects, and the Projects are optionally stored within Project Groups.

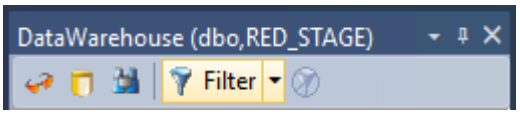
The **middle pane** is used to show the **results** of various **queries** on both the metadata and the underlying source and database tables. The middle pane is also used as the drop target in drag and drop operations.
The status line at the bottom of the screen displays the current contents of the middle pane. To send the middle pane output to a file or to clipboard, see ***Export Middle Pane Output*** (on page 74).

The **right pane** is the **Browser Pane**, it shows both Source and Data Warehouse systems. There are two browser panes available at any one time. The source may be the data warehouse itself. Typically, this pane is used as the source of information in the drag and drop operations. The Browser Pane may be filtered:

- To filter by type, click the down arrow next to the **Filter** button.



- Or to filter by name click the **Filter** button and enter the filter criteria. Name based filters can be cleared by clicking the down arrow next to the **Filter** button and selecting **Clear Name Based Filter**, seen in the image above.



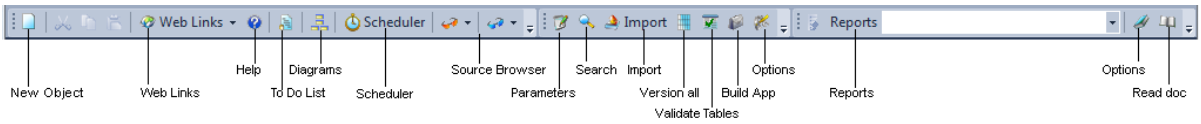
The **bottom** pane is the **results** pane and it shows the results of any command executed on an object. Multiple messages can be displayed. Expand the '+' sign next to an object name to see a complete list of messages relating to the object. When a report is run from the main menu, the results are displayed in a separate tab in the bottom pane.

Pop-up menus are available in all four panes.

TIP: F5 or **Ctrl+R** can be used to refresh the left and right panes, when the cursor is positioned within these panes.

Toolbar

The builder toolbar is shown below:

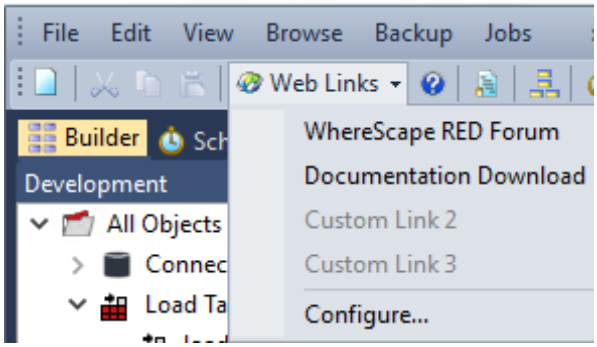


The **diagrammatic view** button (diagram view) switches to the diagrammatic window. This window allows the representation and printing of star schema and track back diagrams.

The **scheduler** button switches to the scheduler control window.

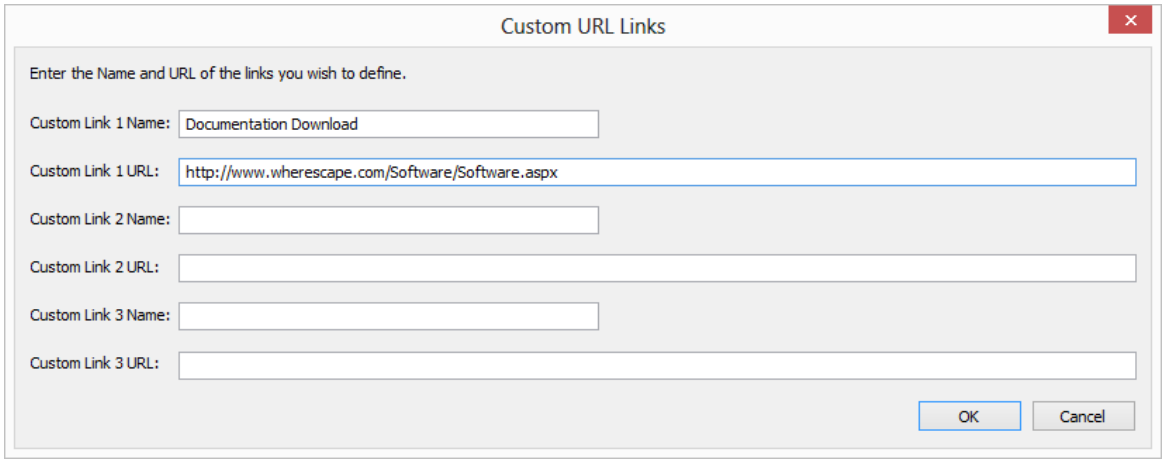
The **two source browse** buttons (one orange and one blue) allow a quick method of invoking the source browser which populates the **Browser pane**.
Each of the two browse buttons, when chosen, browses to the connection last used for that button. To change the connection being browsed click the down arrow beside the glasses icon.
For more information, see ***Browsing a Connection*** (on page 190).

The **Web Links** button brings up the online WhereScape forum in a new tab. To select or enter other web links, click the down arrow beside the **Web Links** button.

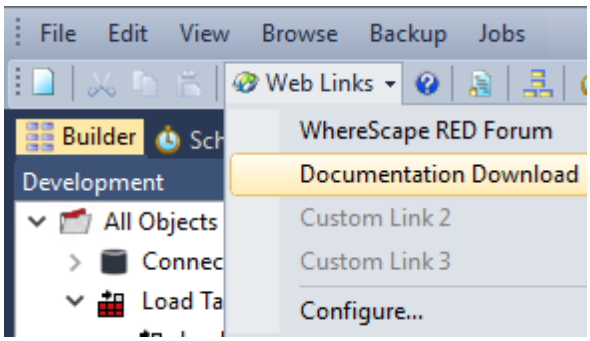


As an example, let us add a Web Link to the Documentation Download on WhereScape's web page.

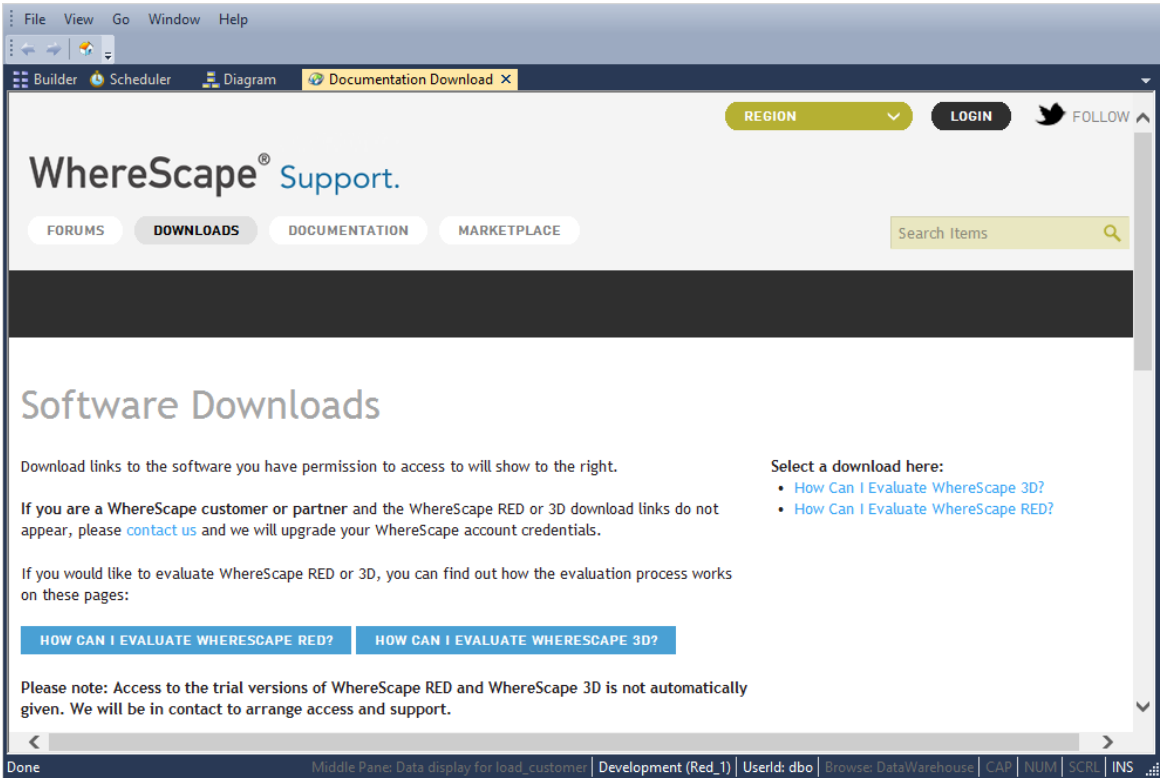
Select **Configure**. The following dialog box will appear, allowing you to enter three custom URLs. Enter the name and the URL for the first custom link. Click **OK**.



To select the newly entered web link, click the down arrow beside the **Web Links** button. Select the newly entered Documentation Download option.



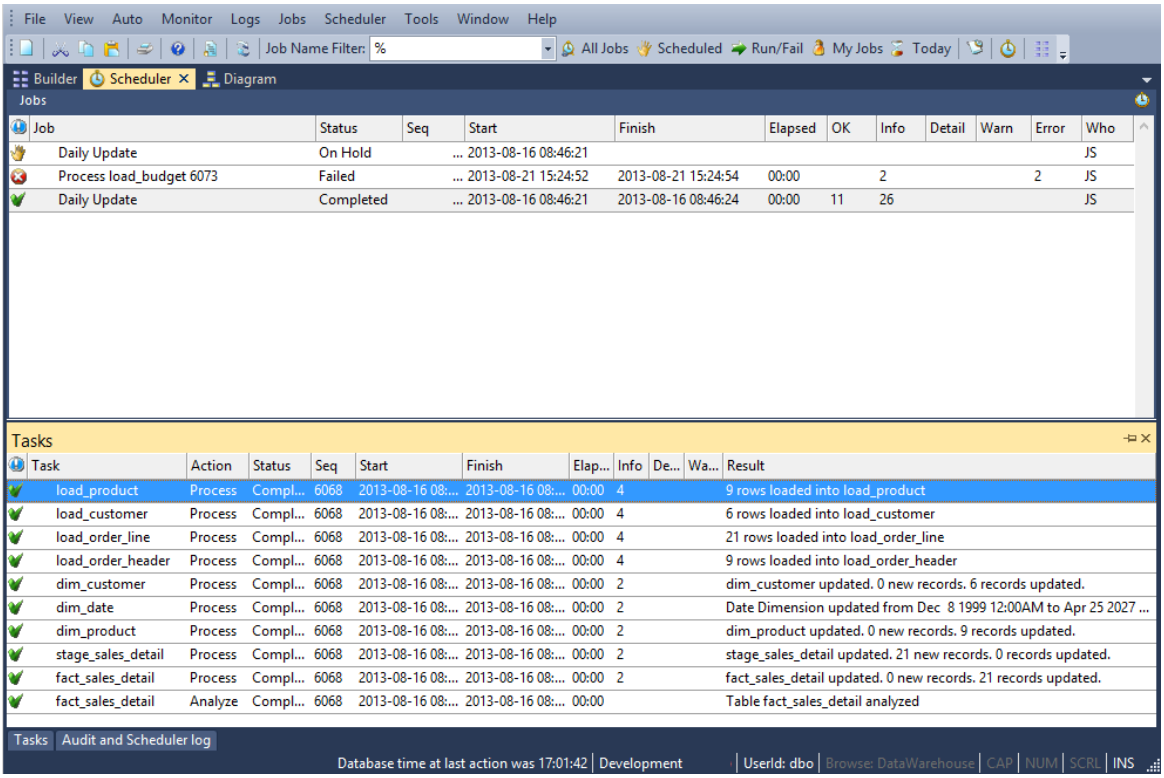
The relevant web page will be displayed in a new tab.



Quick access buttons on the Builder Toolbar also include **versioning**, **building application**, **reports** and **document creation**.

SCHEDULER WINDOW

The **Scheduler Window** is used as the main interface to the scheduler.

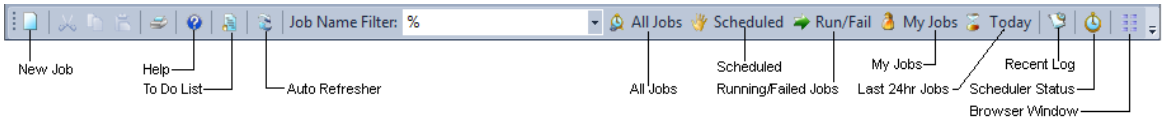


Jobs can be scheduled, monitored, edited and deleted through this window.
The window consists of two panes. The toolbar provides a quick way to display various job selections in the **top pane**.
Double-click on the job in the top pane to see the tasks of the selected job in the **bottom pane**.
Double-click on the task in the bottom pane to see the audit trail displayed in a separate tab in the bottom pane.

See the ***Scheduler*** (on page 736) chapter for more details.

Toolbar

The scheduler toolbar is shown below:



The **New Job** button invokes the dialog to create a new scheduled job.

The **Builder Window** button switches back to the main builder window.

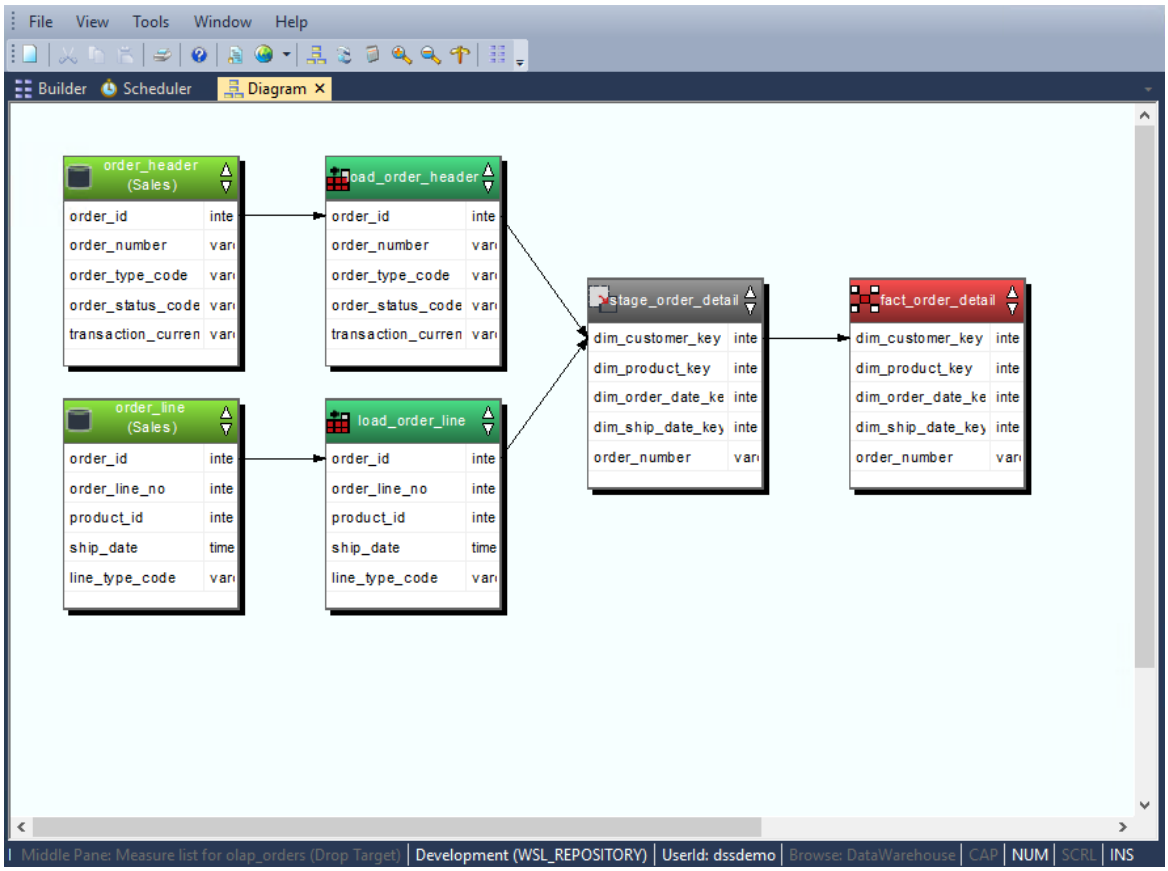
The **Auto Refresh** button, when depressed, will result in a refresh of the current (right pane) display every 10 seconds. Click the button again to stop the auto refresh.

The refresh interval can be adjusted through the menu option **Auto/Refresh** interval.

Quick access to different categories of jobs are also available via the toolbar.

DIAGRAM WINDOW

The **Diagram Window** is used to display the tables of the data warehouse in diagrammatic form, showing the various sources or targets of the selected object.



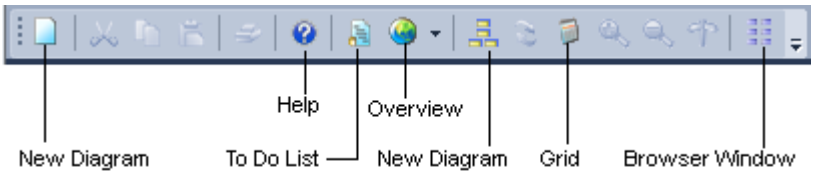
The **Diagram Selection** is as follows:

- Schema Diagram
- Source Diagram
- Joins Diagram
- Links Diagram
- Impact Diagram
- Dependency Diagram

See the ***Diagrams*** (on page 840) chapter for more details.

Toolbar

The diagram toolbar is shown below:



The **New Diagram** button provides a dialog to allow the selection of the diagram type and table.

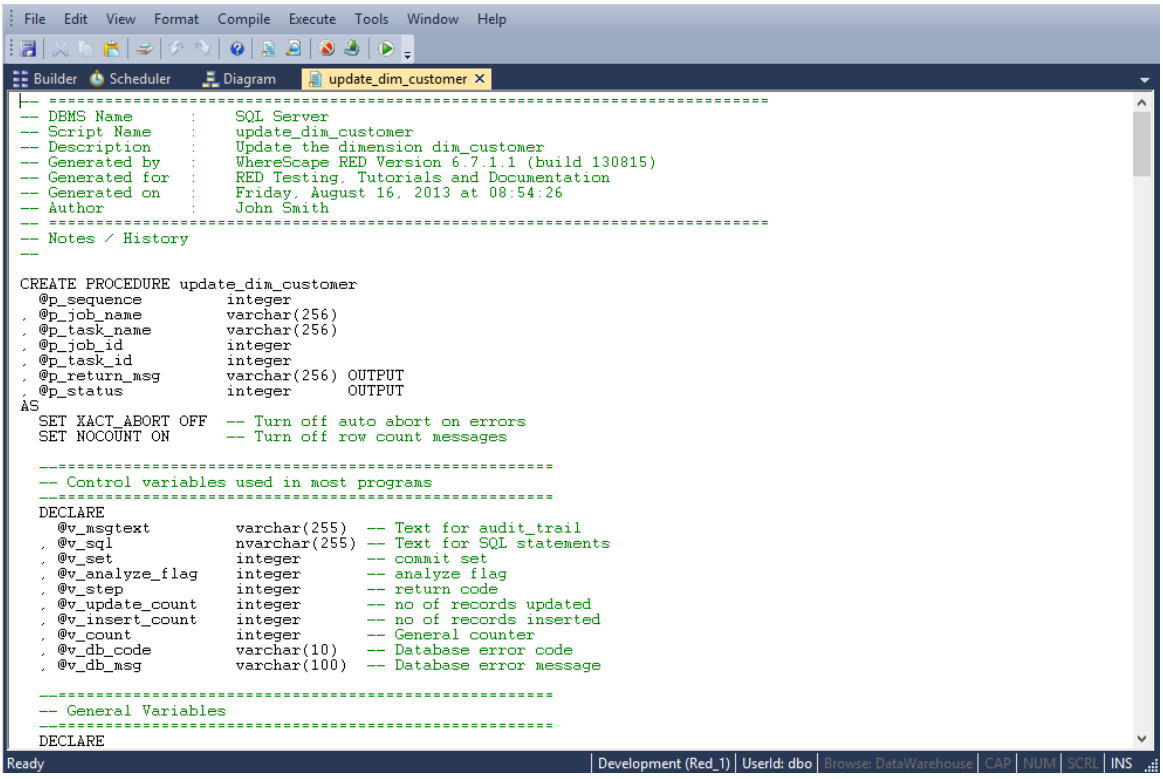
The **Overview** button provides a diagram showing the various objects in the WhereScape metadata and the standard flow of data through these objects. Repeated clicking of the **Overview** button will step through each stage of the data flow.

The **Toggle** button switches between display only diagrams and a printable variant. When the printable variation of a diagram is displayed the **Grid** button will toggle the display of grid lines.

The **Builder** button switches back to the main builder window.

PROCEDURE EDITOR WINDOW

The **Procedure Editor Window** provides a means of viewing, editing, compiling, comparing and running procedures.



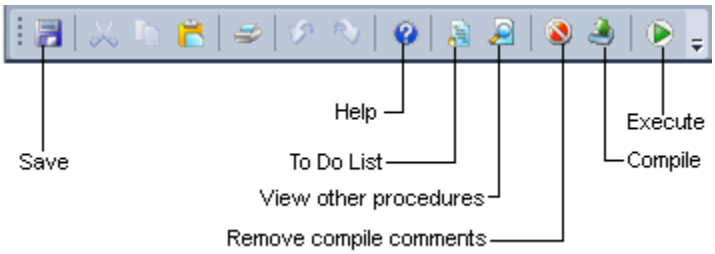
Multiple such windows can be open at any one time, each processing a different procedure.

Comments (identified by a leading double dash --) are displayed in green in this window and the procedural code in black.
The font is a fixed pitch font (by default) to make the indentation and alignment of code easier to view. The font, colors and indent size can all be changed if desired.

See the ***Procedures and Scripts*** (on page 667) chapter for more details.

Toolbar

The procedure editor toolbar is shown below:



The **Save** button will write the procedure to the WhereScape metadata repository in the database.

The **View Other Procedures** button allows the concurrent viewing of older versions of the current procedure, other procedures in the metadata, compiled procedures in the database and templates.

The **Compile** button will attempt to compile the procedure. Once compiled the procedure is stored within the database as well as in the metadata.
For **Oracle** data warehouses, if a procedure fails to compile it is flagged as invalid within the database.

The **Execute** button will run a procedure that conforms to the WhereScape parameter syntax.

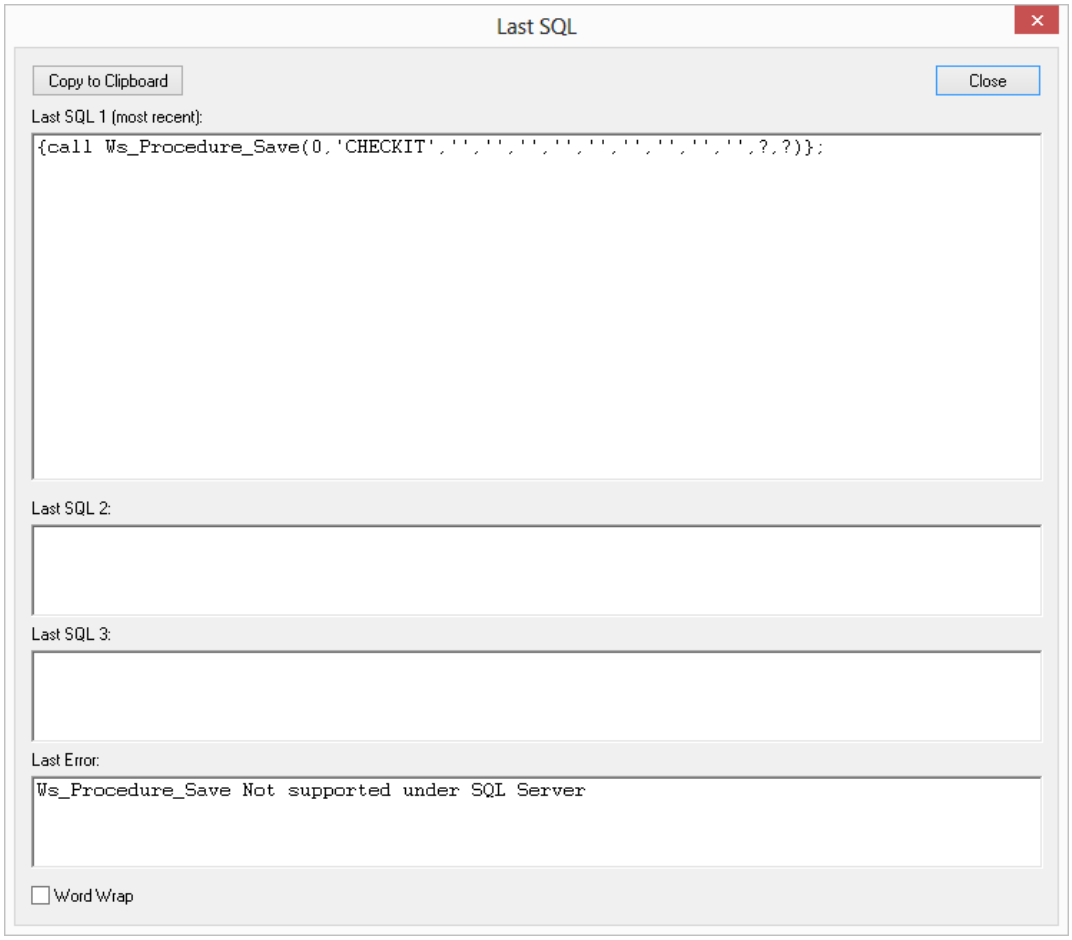
See the chapter on **procedures** for more details.

DATABASE ERROR MESSAGES

From time to time, a SQL error may be raised by WhereScape RED. These are usually caused by the data warehouse database returning an error.
There are many causes of database error messages being returned to RED. The most common causes include permission issues and invalid SQL statements being run.

To find the exact SQL statement run by WhereScape RED, go to the **View** menu and select **Last SQL**.

The following dialog box will be displayed:



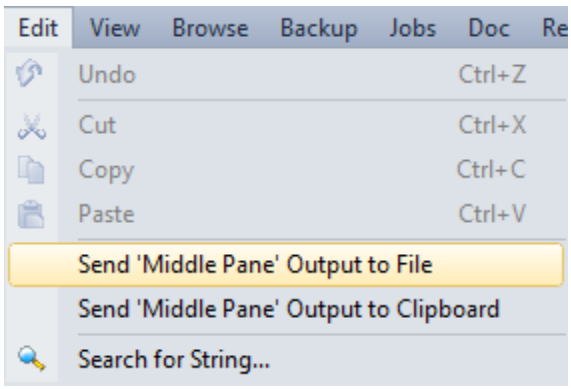
The four fields shown are:

- The last SQL statement run
- The second to last SQL statement run
- The third to last SQL statement run
- The error message (if any) from the last SQL statement

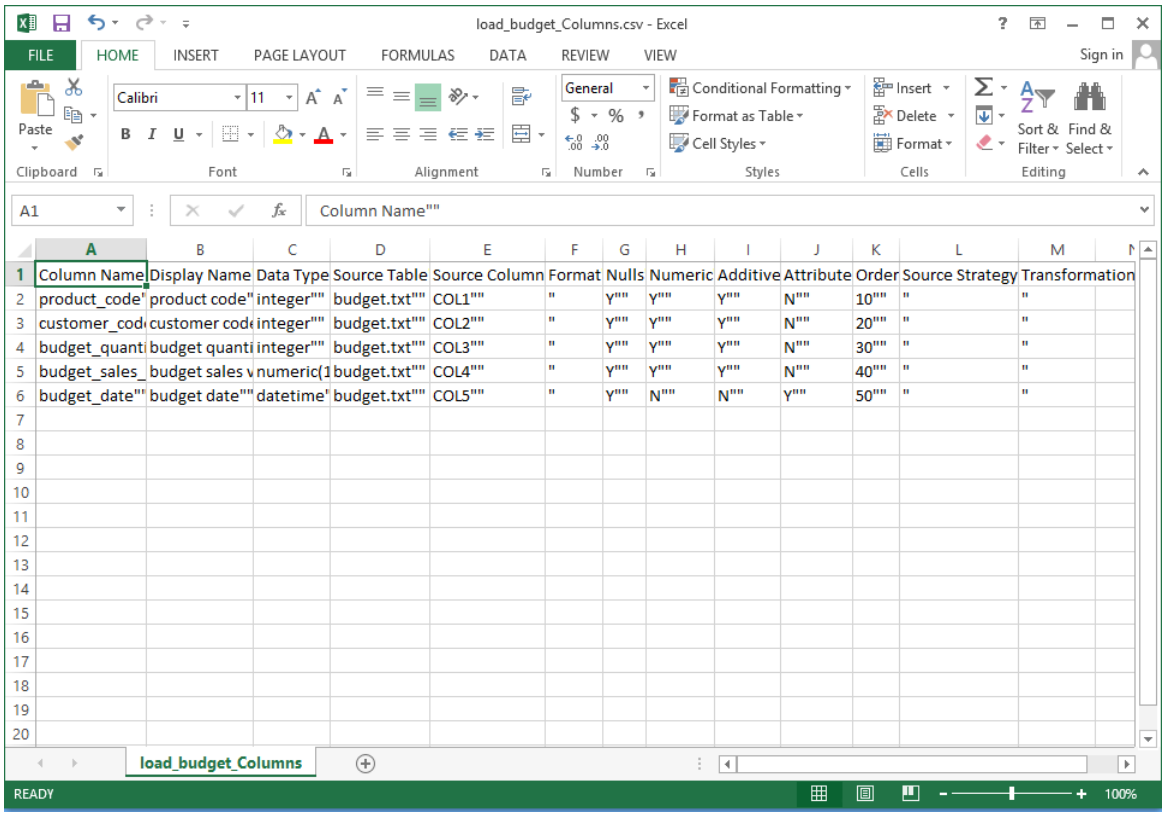
In the example above, we have tried to drop a procedure that does not exist. IBM DB2 returned an error telling us RED.UPDATE_DIM_KPI does not exist in the database.

EXPORT MIDDLE PANE OUTPUT

To send the middle pane output to a file, go to the **Edit** menu and select **Send 'Middle Pane' Output to File**.



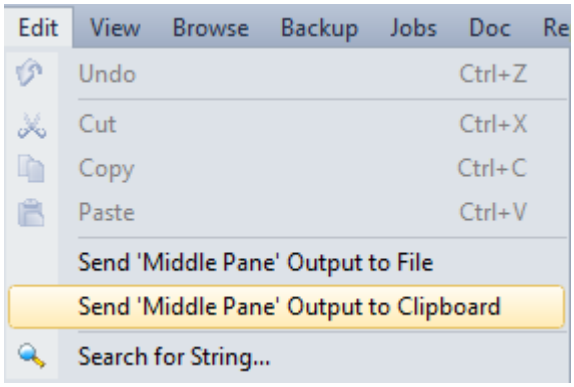
A file will be created in the directory chosen in **Tools/User Preferences/Outputs/Output File Directory**.
If the Output File Extension is set to .csv in **Tools/User Preferences/Outputs/Output File Extension**, then an Excel file will be created.
If the file is set to auto-open in **Tools/User Preferences/Outputs/Output File Auto Open**, then the file will open automatically.



	A	B	C	D	E	F	G	H	I	J	K	L	M
1	Column Name	Display Name	Data Type	Source Table	Source Column	Format	Nulls	Numeric	Additive	Attribute	Order	Source Strategy	Transformation
2	product_code"	product code"	integer""	budget.txt""	COL1""	"	Y""	Y""	Y""	N""	10""	"	"
3	customer_cod	customer cod	integer""	budget.txt""	COL2""	"	Y""	Y""	Y""	N""	20""	"	"
4	budget_quanti	budget quanti	integer""	budget.txt""	COL3""	"	Y""	Y""	Y""	N""	30""	"	"
5	budget_sales_	budget sales v	numeric(1	budget.txt""	COL4""	"	Y""	Y""	Y""	N""	40""	"	"
6	budget_date""	budget date""	datetime"	budget.txt""	COL5""	"	Y""	N""	N""	Y""	50""	"	"

To view the other settings for **Middle Pane File Output**, see ***Export Middle Pane Output Settings*** (see "***Outputs***" on page 132).

To send the middle pane output to clipboard, go to the **Edit** menu and select **Send 'Middle Pane' Output to Clipboard**.

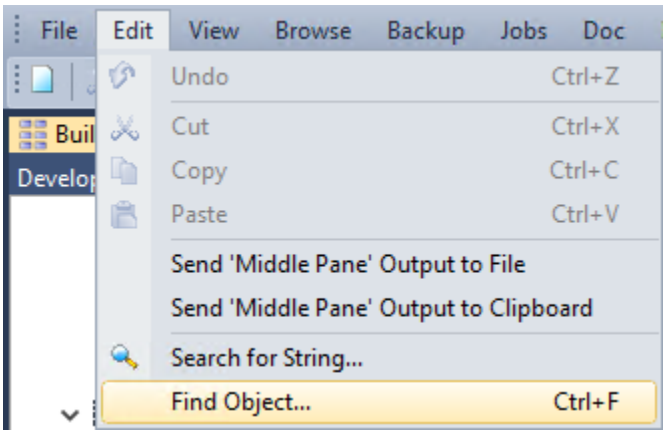


To view the settings for **Middle Pane Clipboard Output**, see ***Export Middle Pane Output Settings*** (see "***Outputs***" on page 132).

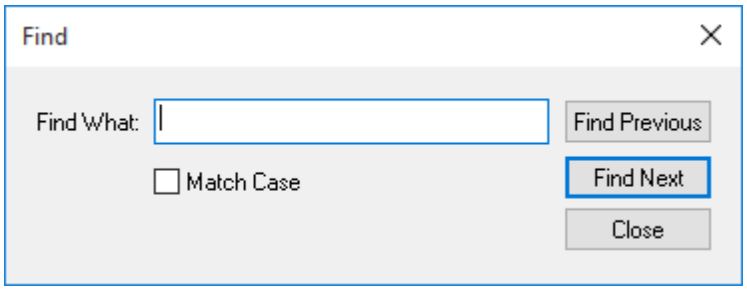
FIND FUNCTION

The Find function can help users quickly find a table when the list of tables has grown to a large size. The Find function can be accessed two ways within WhereScape RED:

- Click anywhere in the Object Pane or Browser Pane and press **Ctrl + F**, or
- select **Edit > Find Object** (this option searches the Object Pane only).



Both methods open the following dialog:



WhereScape®

CHAPTER 3
TUTORIALS

The WhereScape RED tutorials are located in the WhereScape RED **Tutorial** Help.

WhereScape®

CHAPTER 4
DEFAULT SETTINGS

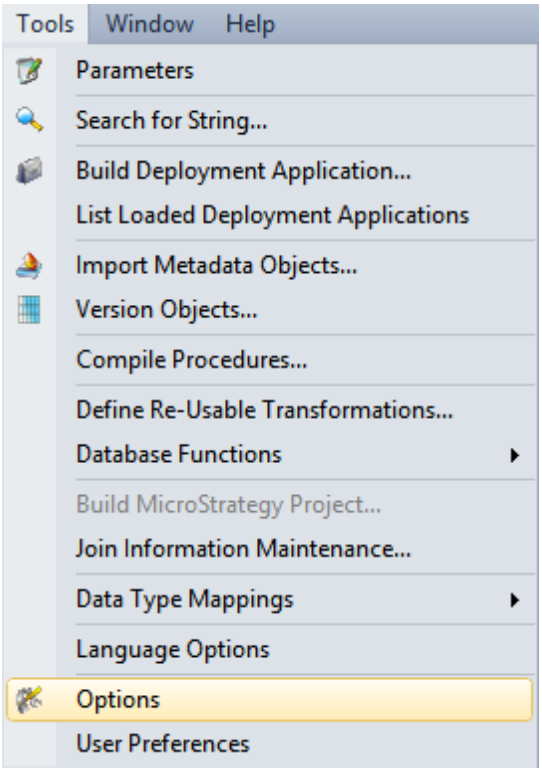
This chapter describes the settings and default values that can be set for a metadata repository.

To access these settings, select the **Tools** menu; and then either **Options** or **User Preferences**.

IN THIS CHAPTER

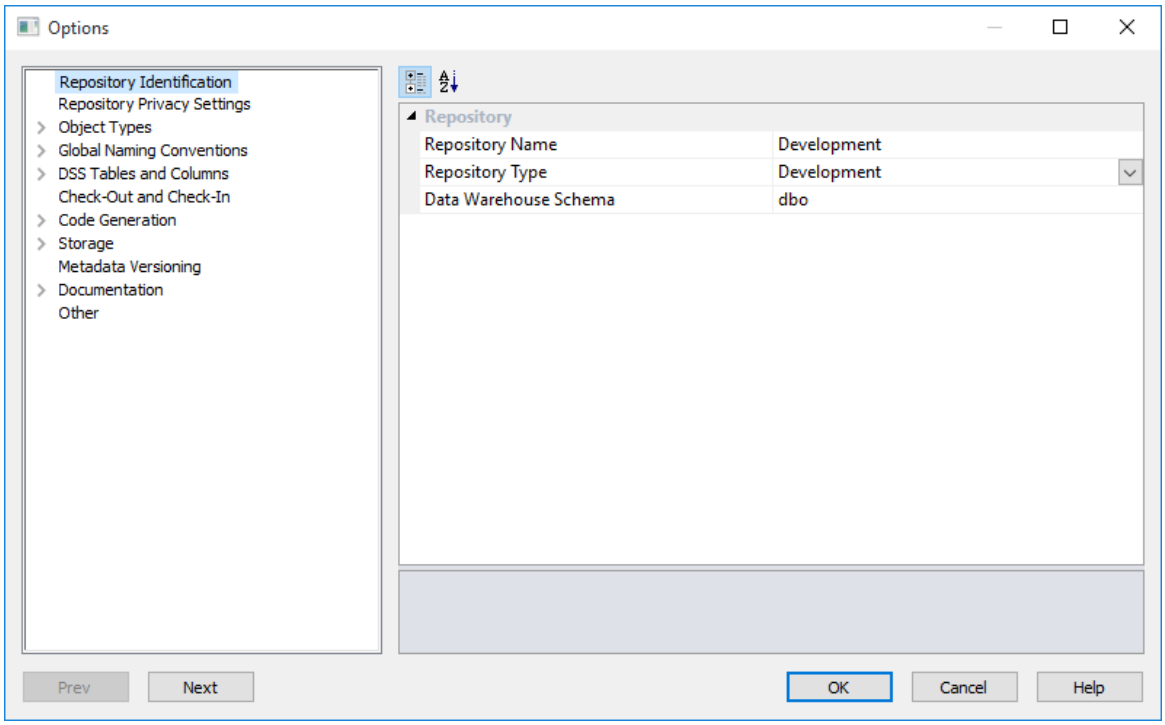
SETTINGS - OPTIONS

Select **Options** from the **Tools** menu.



REPOSITORY IDENTIFICATION

This option allows users to set the **Repository Identification** settings.



Repository Name

Set the **Name** for the repository. This name appears in the title bar in WhereScape RED. Restart WhereScape RED for repository name changes to take effect.

Repository Type

Set the **Type** for the repository. The repository type should reflect the environment. For example, a 'Production' type should be chosen for the production environment.

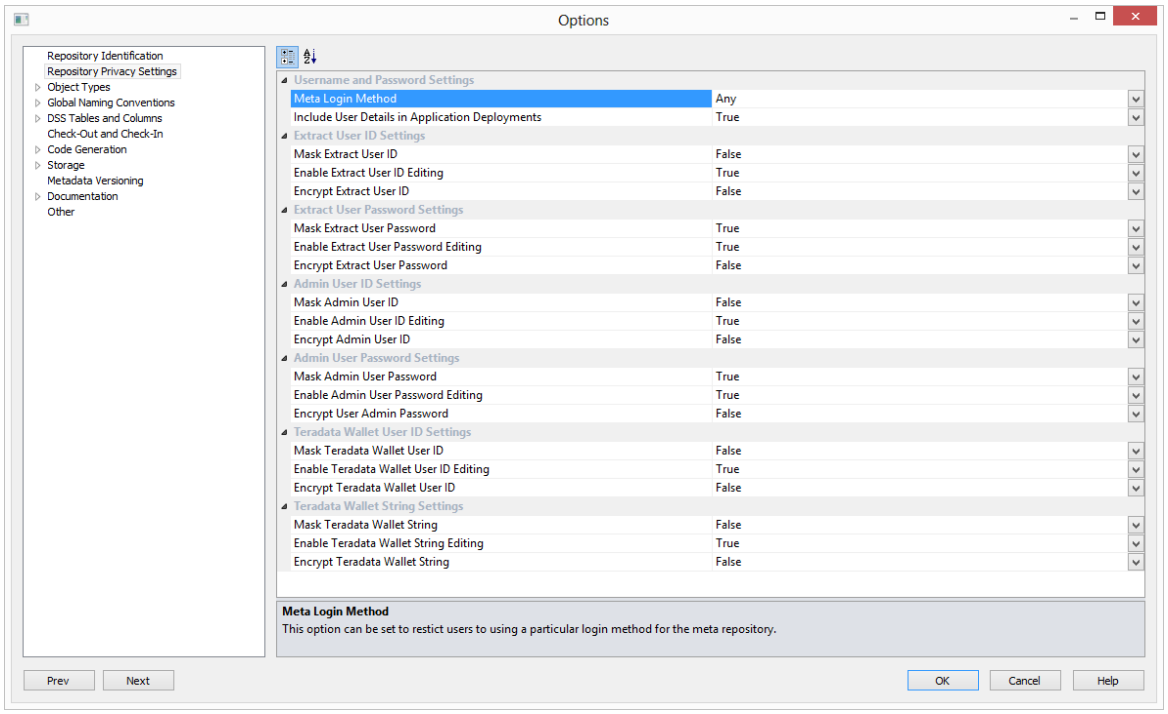
Data Warehouse Schema

Set the **Schema** for the repository. The schema of the data warehouse is used by the WhereScape RED scheduler. You should not normally need to change this value.

REPOSITORY PRIVACY SETTINGS

This option allows users to set the **Repository Privacy Settings**.

WARNING: For UNIX/Linux scheduler processing, the **Encrypt** User and Password options cannot be used. Encrypt options are only supported when using a Windows scheduler.



Changing Repository Settings

Since the repository privacy settings can be configured from the Tools (Options) menu, for an environment to be secured, **a database administrator will need to change the permissions on table ws_meta_admin table to read-only after the appropriate repository privacy change settings** in WhereScape RED have been made.

NOTE: Changing this set of permissions to read-only is something which occurs outside of WhereScape RED and will be dependent on the specific meta data database.

Username and Password Settings

Meta Login Method - This option can be set to restrict users to using a particular login method for the meta repository

Include User Details in Application Deployments - Includes or excludes User Details in Application Deployment packages

Extract User ID Settings

Mask Extract User ID - Masks the input of the "Extract/Unix/Windows User ID" on the connection properties

Enable Extract User ID Editing - Allows editing the "Extract/Unix/Windows User ID" via the connection properties

Encrypt Extract User ID - Encrypts "Extract/Unix/Windows User ID" in the meta repository using WhereScape encryption

Extract User ID Settings

Mask Extract User Password - Masks the input of the "Extract/Unix/Windows User Password" on the connection properties

Enable Extract User Password Editing - Allows editing "Extract/Unix/Windows User Password" via the connection properties

Encrypt Extract User Password - Encrypts "Extract/Unix/Windows User Password" in the meta repository using WhereScape encryption

Admin User ID Settings

Mask Admin User ID - Masks the input of the "Admin/DSS User ID" on the connection properties

Enable Admin User ID Editing - Allows editing the "Admin/DSS User ID" via the connection properties

Encrypt Admin User ID - Encrypts "Admin/DSS User ID" in the meta repository using WhereScape encryption

Admin User Password Settings

Mask Admin User ID - Masks the input of the "Admin/DSS User ID" on the connection properties

Enable Admin User ID Editing - Allows editing the "Admin/DSS User ID" via the connection properties

Encrypt Admin User ID - Encrypts "Admin/DSS User ID" in the meta repository using RED encryption

Teradata Wallet User ID Settings

Mask Teradata Wallet User ID - Masks the input of the "Teradata Wallet User ID" on the connection properties

Enable Teradata Wallet User ID Editing - Allows editing the "Teradata Wallet User ID" via the connection properties

Encrypt Teradata Wallet User ID - Encrypts the "Teradata Wallet User ID" in the meta repository using WhereScape encryption

Teradata Wallet String Settings

Mask Teradata Wallet String - Masks the input of the "Teradata Wallet String" on the connection properties

Enable Teradata Wallet String Editing - Allows editing the "Teradata Wallet String" via the connection properties

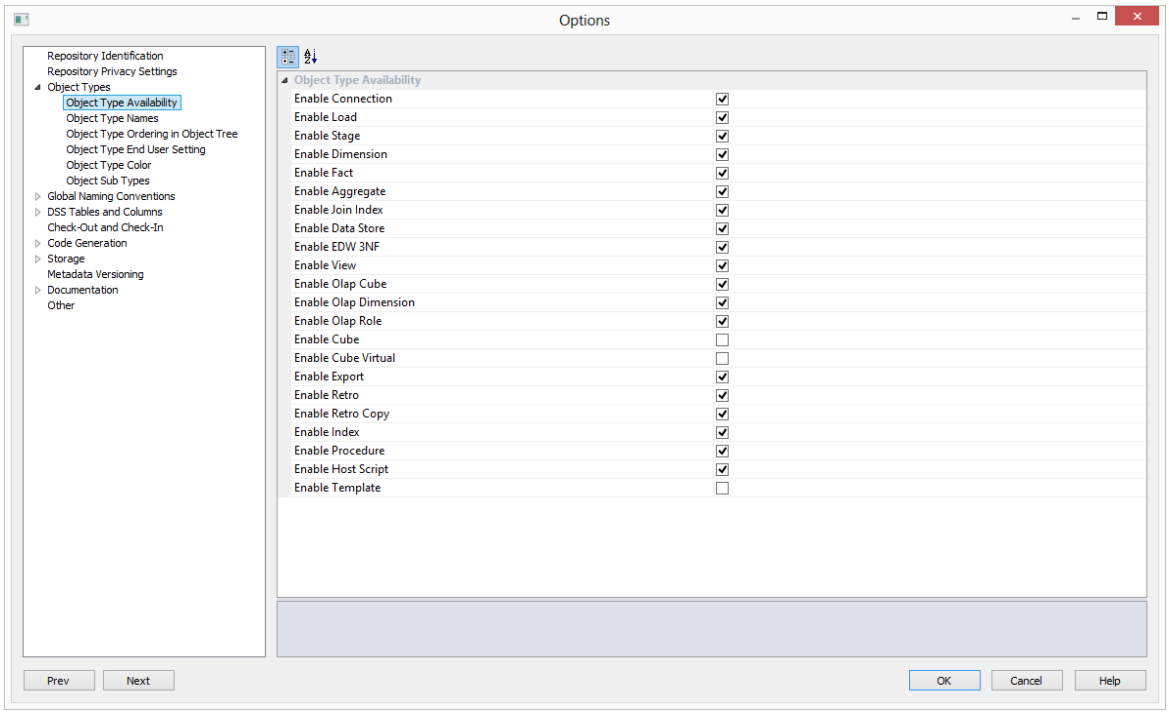
Encrypt Teradata Wallet String - Encrypts the "Teradata Wallet String" in the meta repository using WhereScape encryption

OBJECT TYPES

The various options are described below.

OBJECT TYPE AVAILABILITY

This option allows users to **enable** or **disable** the various object types within the repository.

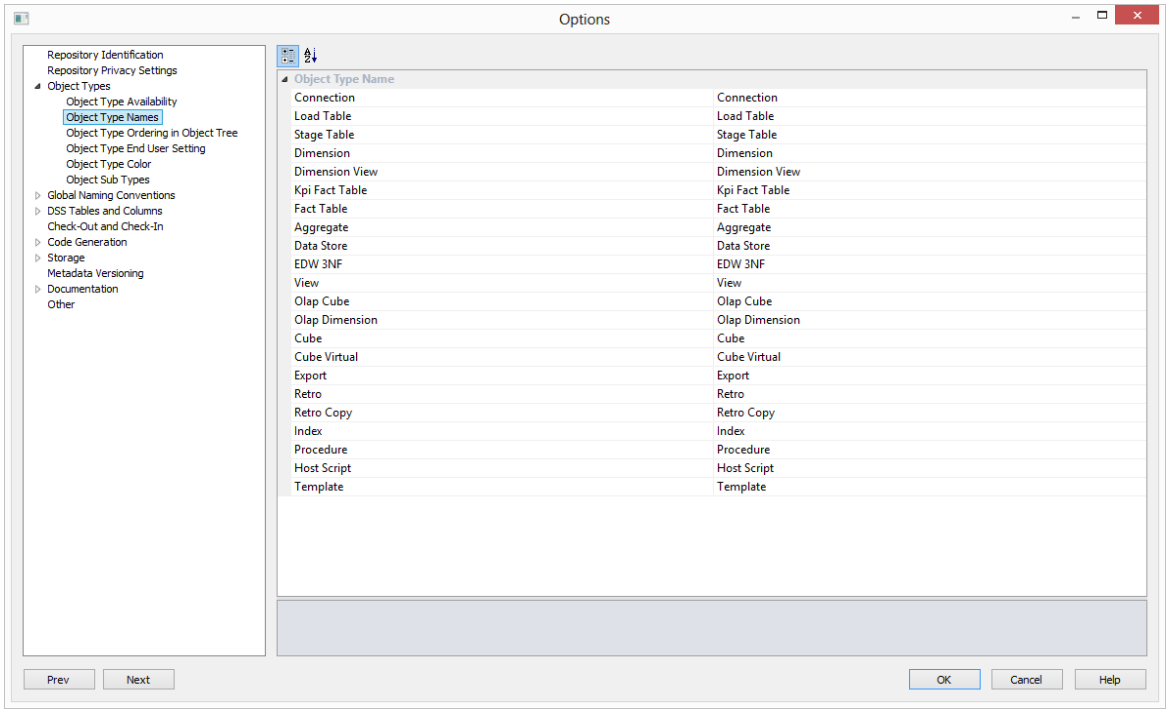


Object Type Availability.

Enable/Disable object types in the data warehouse by setting to each object type's availability option to true/false or by ticking/un-ticking the check-boxes.

OBJECT TYPE NAMES

This option enables users to set the **names** for the various object types.



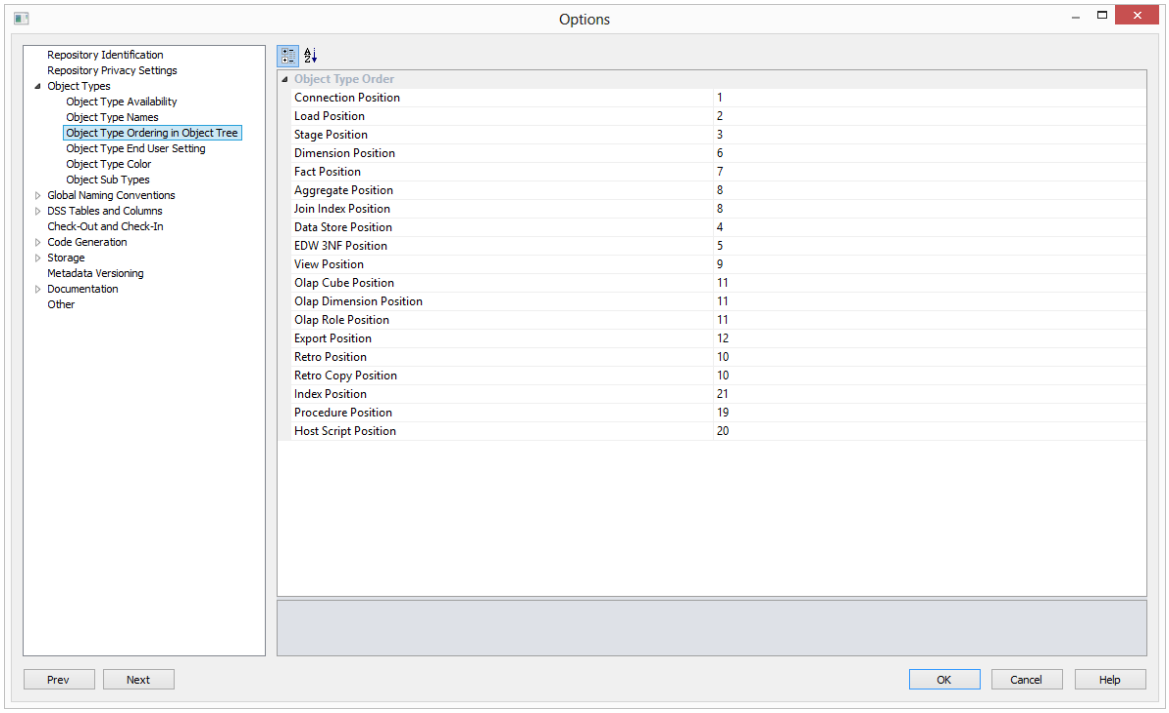
Object Type Name

Set the **Name** for each object type.

NOTE - Data Vault Repository Types: Users with Data Vault model type licenses that chose a **Data Vault repository type** while creating the RED metadata repository will have appropriate Data Vault repository default settings such as Object Type Names, Global Naming of Tables, Indexes, Key Columns and Procedures/Scripts, as well as other repository settings/user preferences.
These repository types will have their default Object Type Names of Normalized and Data Store objects set to Hub/Link and Satellite.

OBJECT TYPE ORDERING

This option allows users to set the **ordering** in which the object types appear in the object tree.

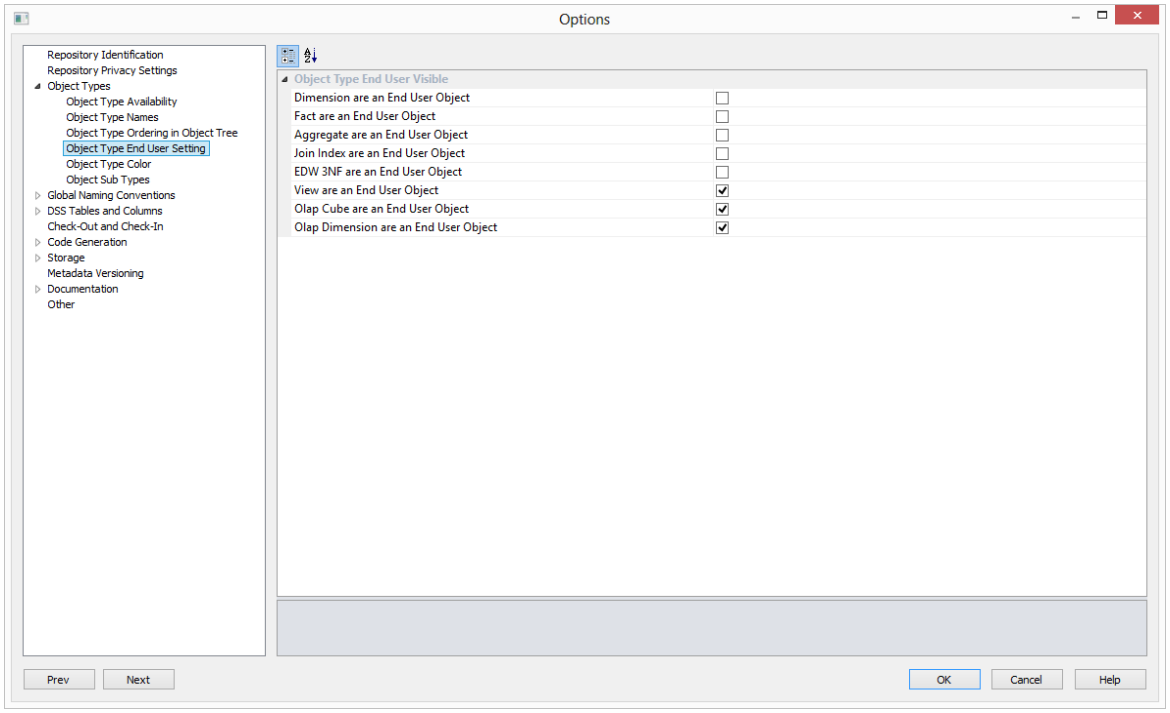


Object Type Order

Set the **Ordering** of the object types as displayed in the object tree.

OBJECT END USER SETTING

This option allows users to set the Object types as **end user objects**.



Object Type End User Visible

Set to **True** to make object type an end user object, else False.

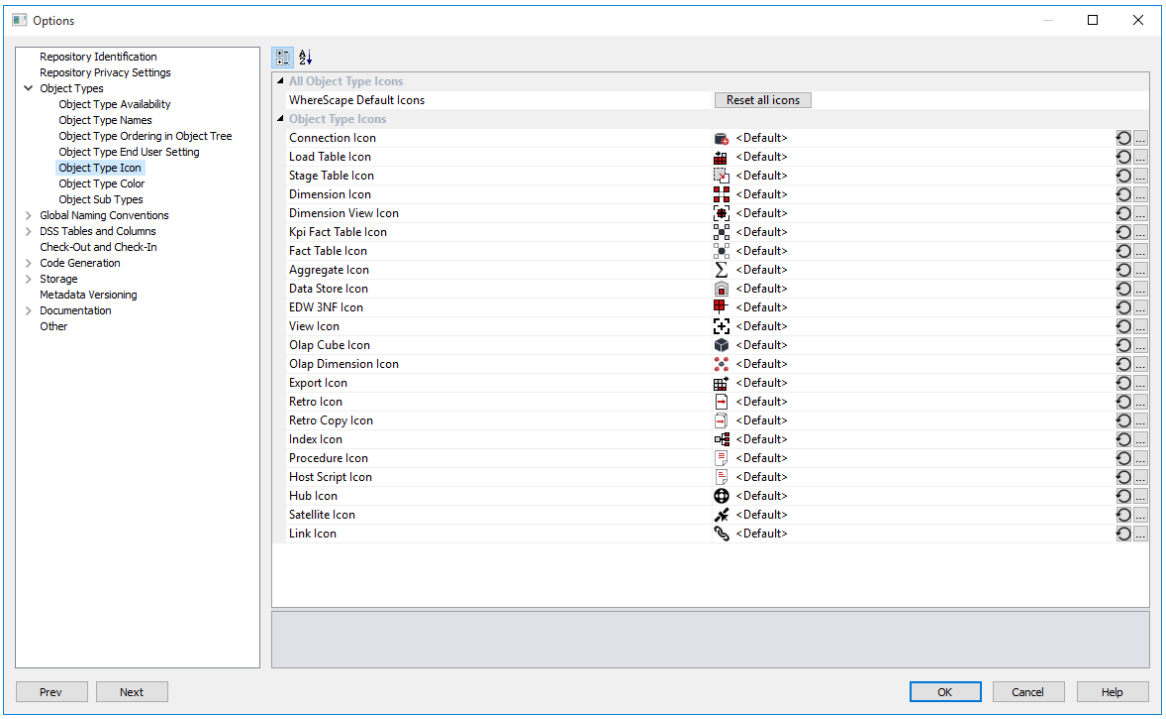
OBJECT TYPE ICON

This option allows users to configure the **Icons** for all Object Types. To configure custom Object Type icons:

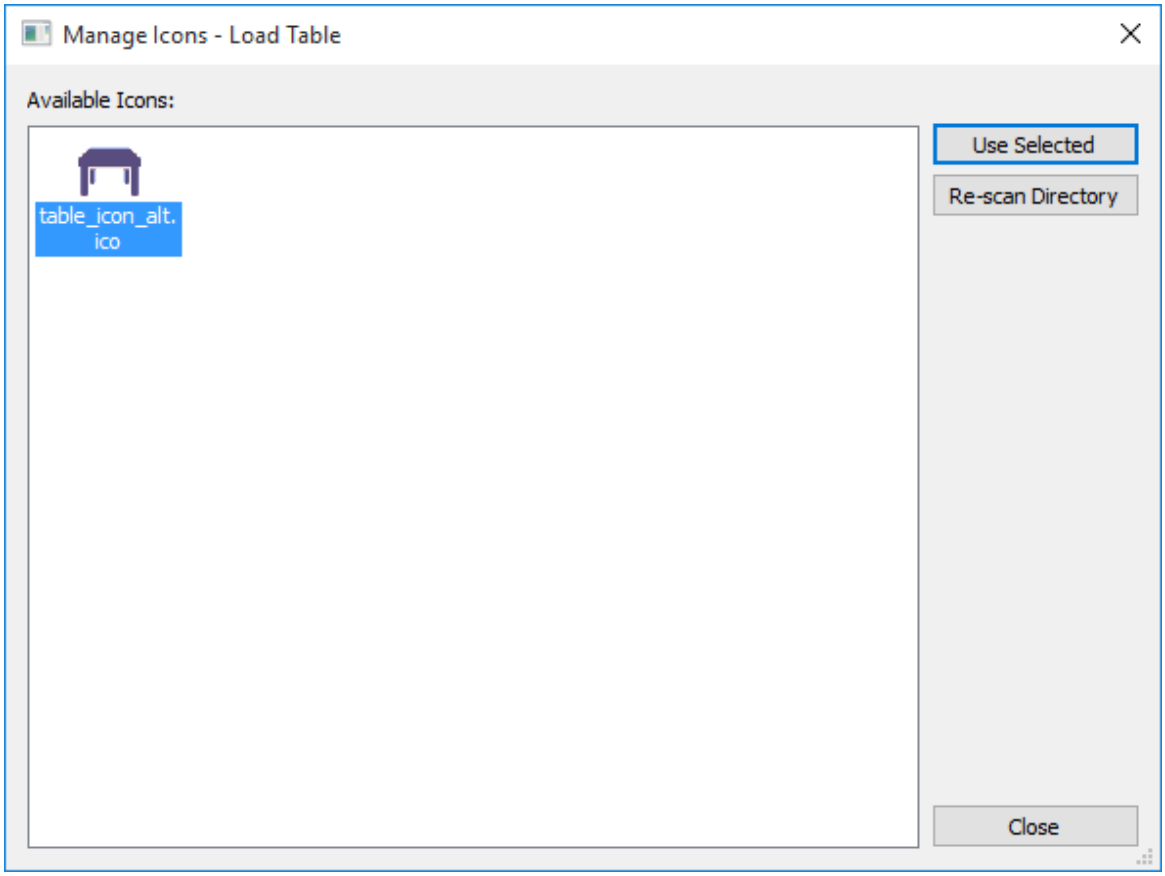
1. Create an 'Icons' folder in the WhereScape RED install directory, if it doesn't already exist. For example:

   C:\Program Files (x86)\WhereScape\Icons

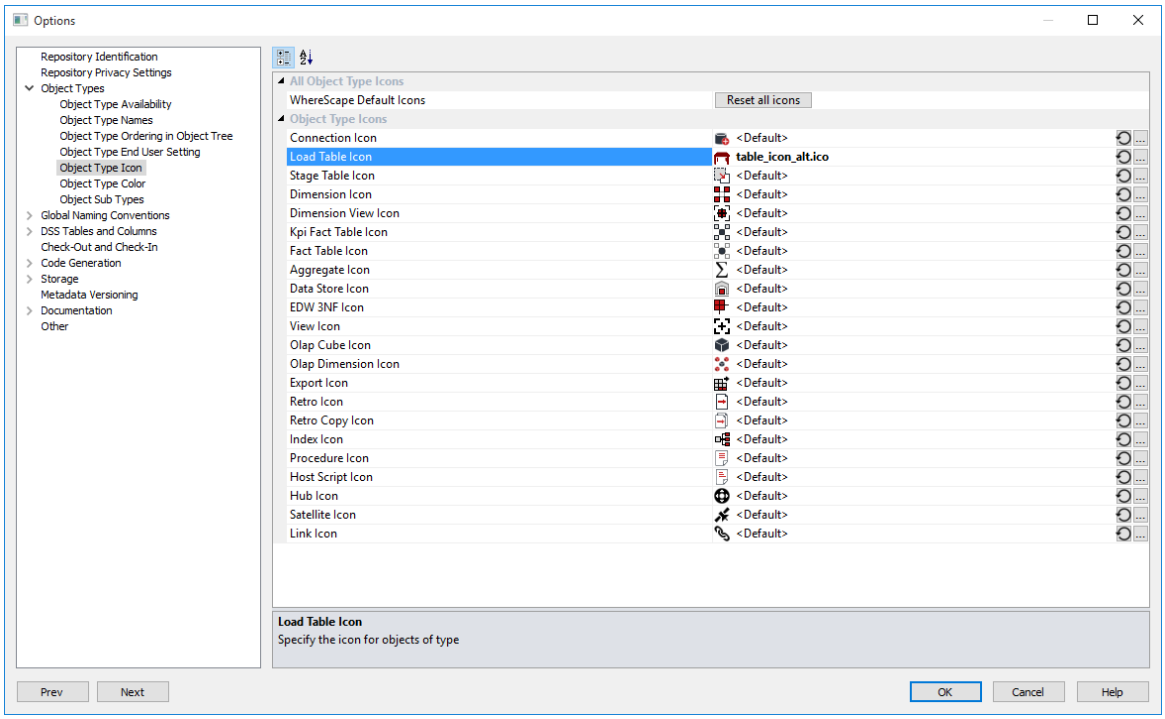
2. Place custom '**.ico**' files in the Icons folder.
3. In RED, select **Options>Object Types>Object Type Icon**.



4 Click the ellipses button '...' next to each Object Type and select the desired icon.



5 All configured icons are displayed as file names in the Options dialog. To save changes click OK.



To Reset An Icon

To reset an icon to default, click the reset button () next to the icon.

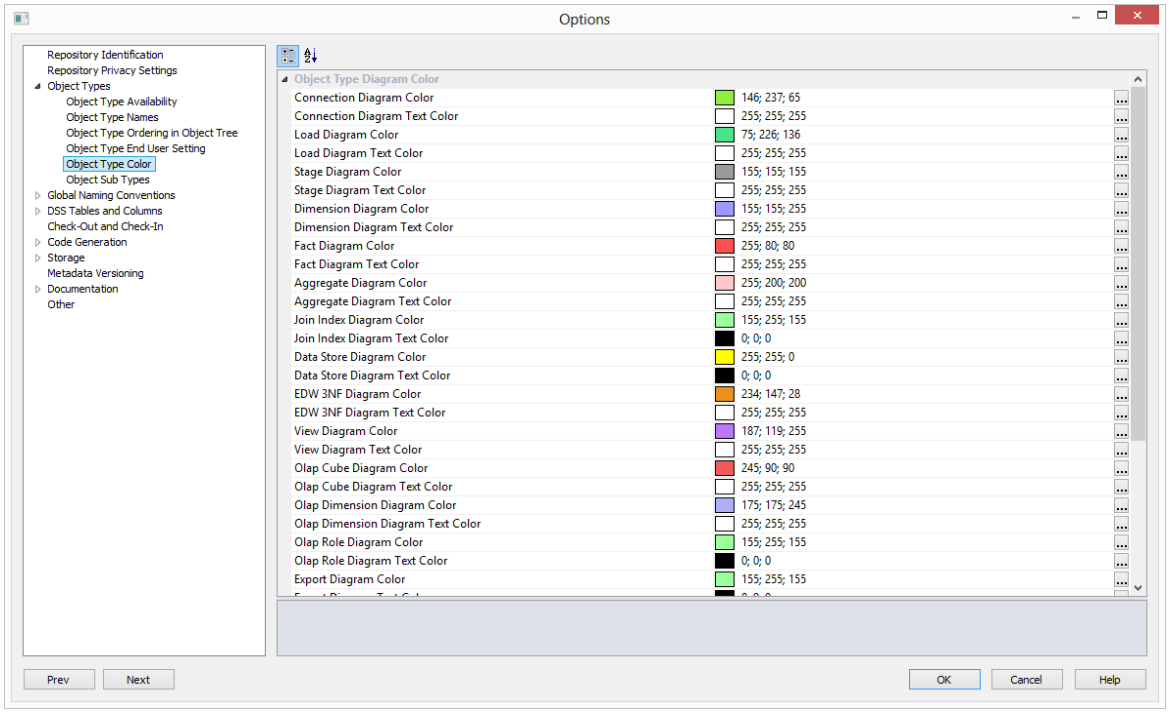
Reset All Icons

To reset all icons to default, click the **Reset All Icons** button at the top of the dialog.

Note: All installations must have a copy of the icon directory.

OBJECT TYPE COLOR

This option allows users to set the **diagram colors** for each object type.

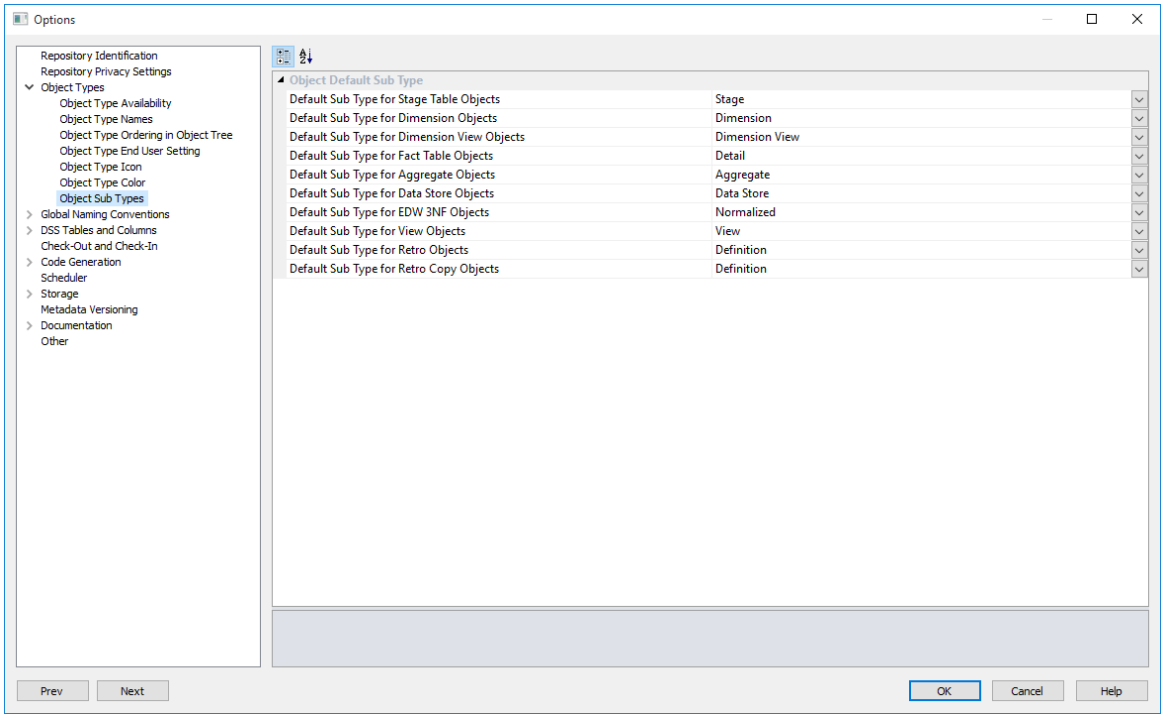


Object Type Diagram Color

Set the **Diagram Color** for each object type.

OBJECT SUB TYPES

This option allows users to set the **default sub type** for enabled objects in RED.

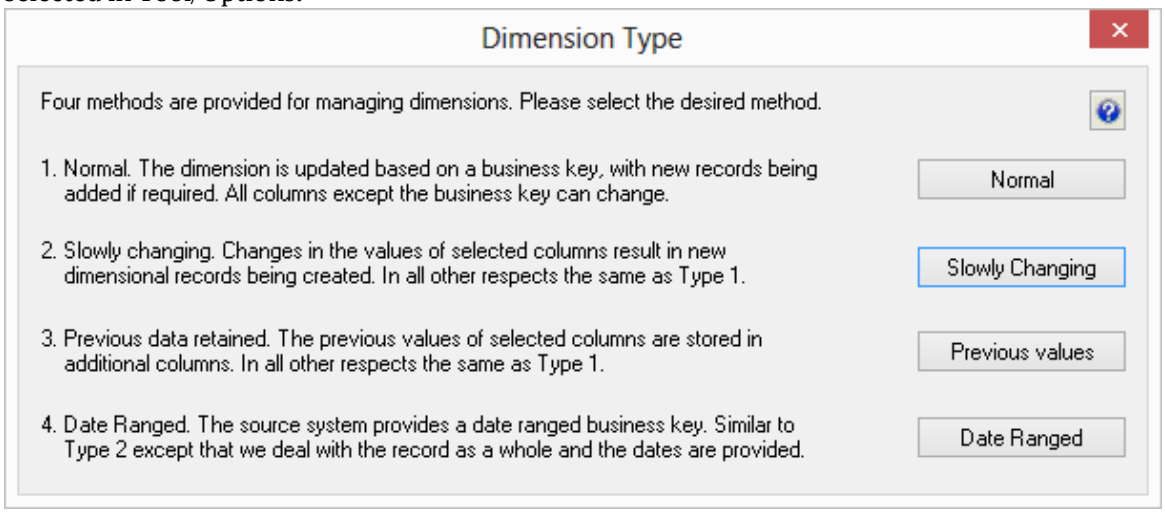


Object Default Sub Type

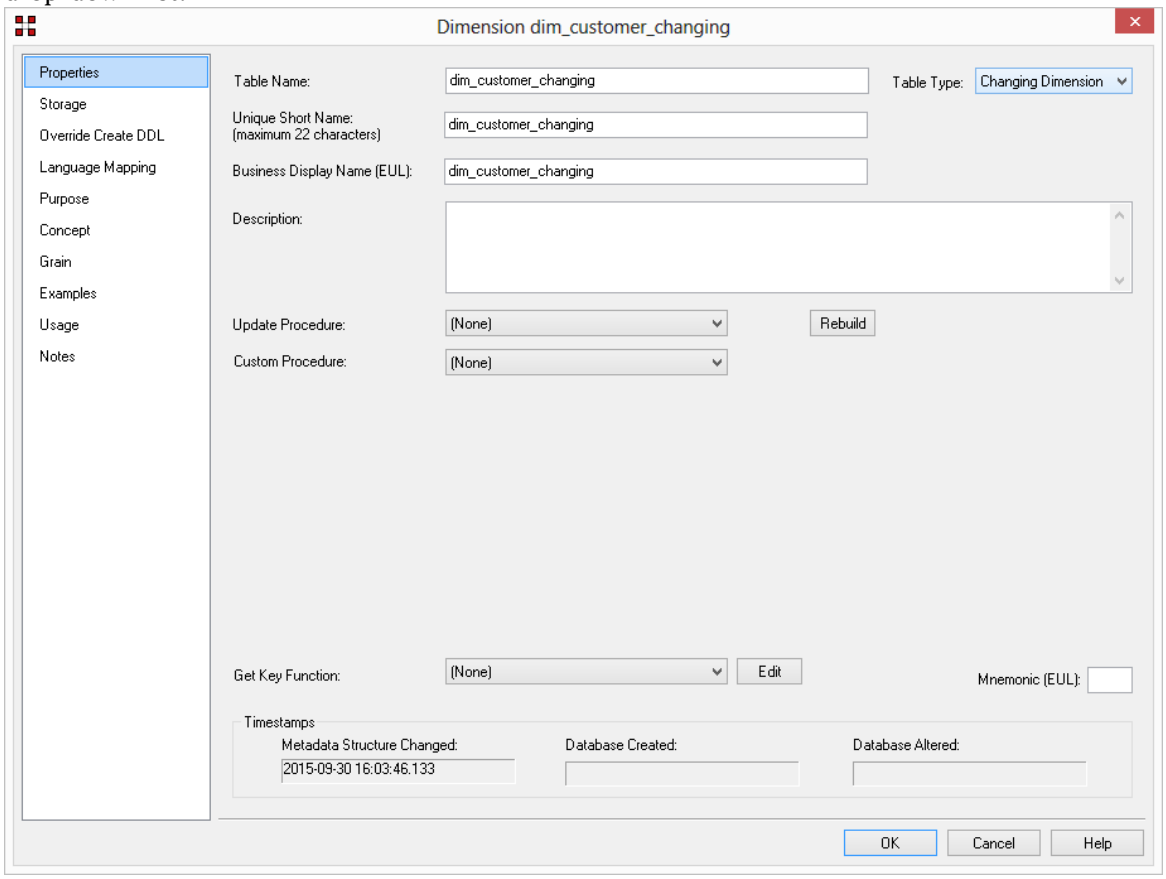
This option enables specifying the **Default Sub Type** for enabled object types. Select the desired default sub-types from the Object's drop-down lists.

As an example, to have **Dimension** objects created in RED as **Changing Dimensions** at the time of drag and drop, select the **Changing Dimension** option in the **Default Sub Type for Dimension Objects**.

After the table is dragged and dropped, users can simply hit enter to proceed on the Dimension Type where the **Slowly Changing** type is already defaulting to the sub type option previously selected in Tool/Options.



The Dimension Properties' screen will reflect the selected table sub type on the **Table Type** drop-down list.

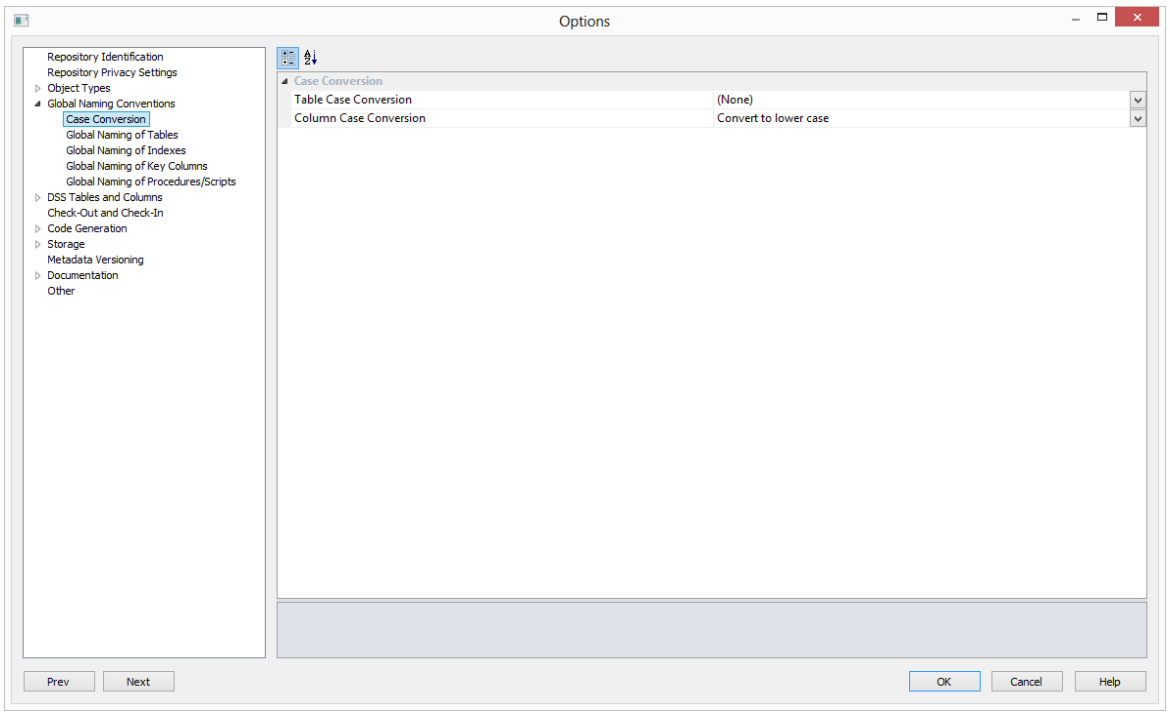


GLOBAL NAMING CONVENTIONS

The various options are described below.

CASE CONVERSION

This option allows users to set the **case conversion** methods for Tables and Columns.

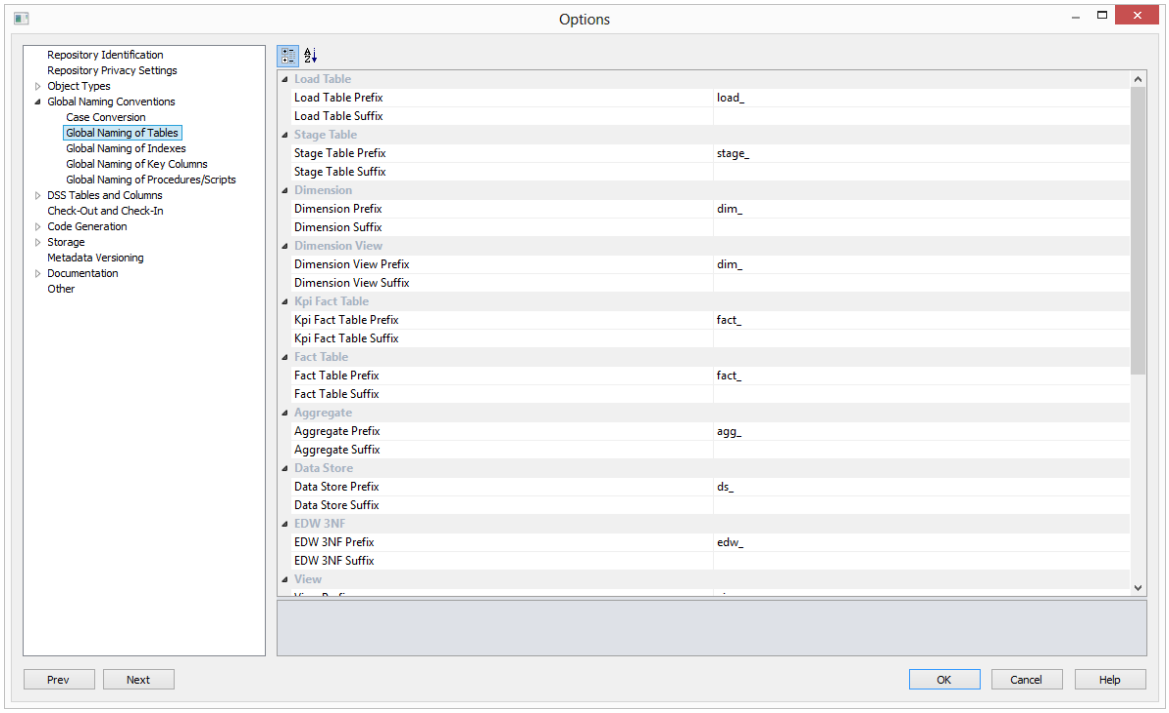


Case Conversion

Set the **Table Case Conversion** method and the **Column Case Conversion** method from the drop-down lists.

GLOBAL NAMING OF TABLES

This option allows users to set the **Global Naming of Tables** options.



A prefix and/or a suffix string can be applied to an object name. Within Oracle and IBM DB2, a table name may be a maximum of 30 characters long, so these pre and post fix strings should not be more than eight characters long (WhereScape RED short names are a maximum of 22 long in Oracle and SQL Server and 12 long in DB2).

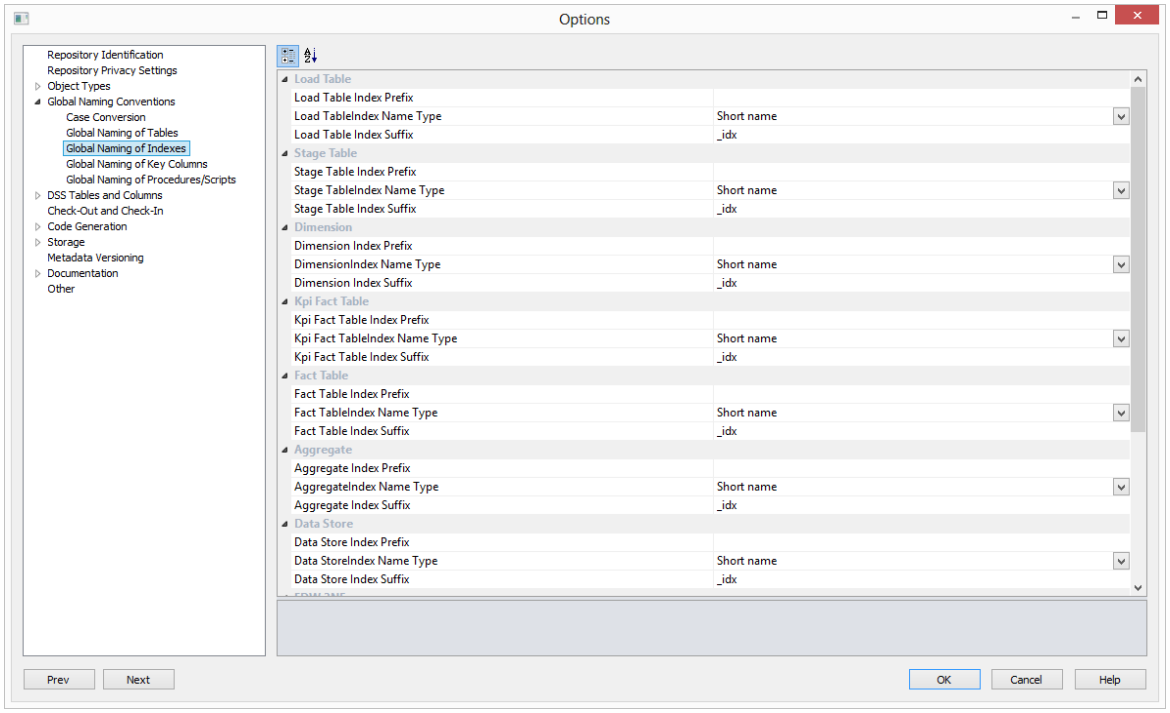
From the example screen above, if a source table called **customer** (with a short name of 'customer') was dragged into a load table drop target then the default name would be **load_customer**.

The object name defaults shown above are the values that are installed with the base metadata.

They can be changed at any stage; however, the change does not affect any existing objects. So, if a new naming regime is chosen any existing objects will need to be renamed through the Properties screen of the object.

GLOBAL NAMING OF INDEXES

This option allows users to set the **Global Naming of Indexes** options.



Whenever a new procedure is defined, WhereScape RED builds or rebuilds a standard set of indexes for the table. These indexes will be created using the standard defined. As with the key naming we can set either a pre-fix or a suffix value, or in fact both, as well as choosing the use of either the table name or the short name associated with the table.

In addition to the naming specifications above WhereScape RED will add up to a further 3 characters to the end of the index name. These additional values will be "_0" through "_99", or "_A" through "_Z", or "_SC".

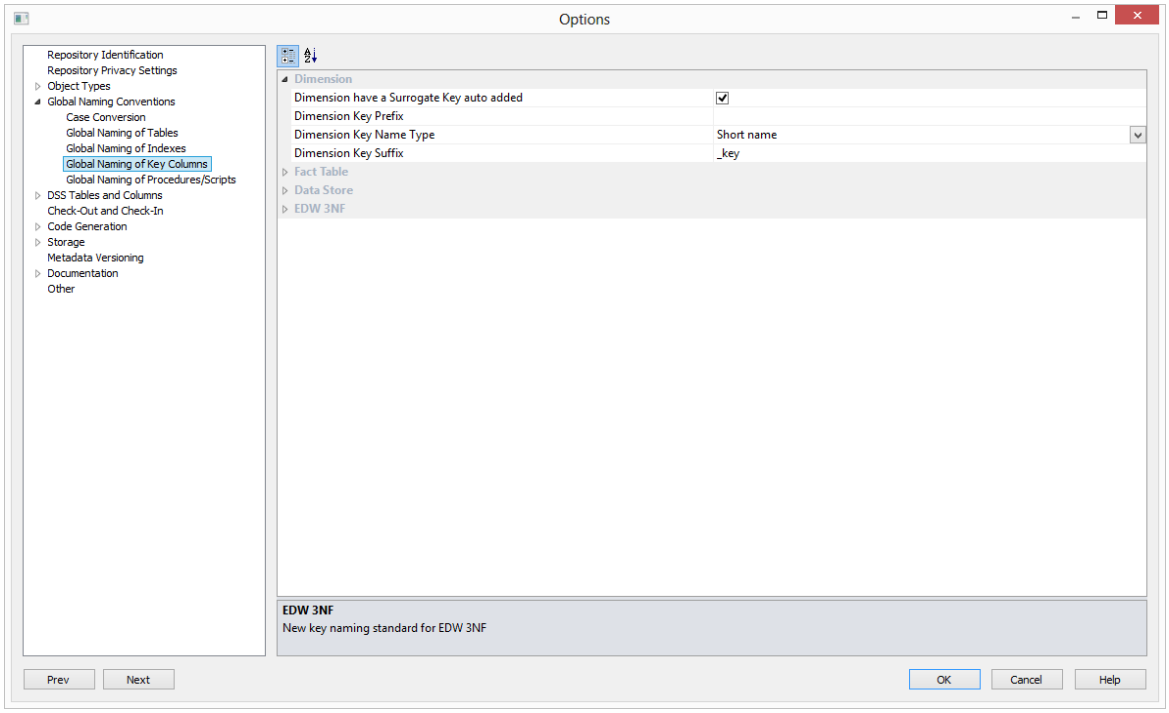
When a new index is manually added, it will have the additional value of "_x" by default. This should be changed. The WhereScape RED naming standard for indexes is described below, but any valid name may be used.

From the example screen above, a fact table would have indexes generated using the short name and with a suffix of "_idx". Therefore, a fact_sales fact table would have indexes such as fact_sales_idx_x.

Ultimate suffix	**meaning**
_0	artificial key
_1 thru _99	bitmap key index on dimensional join
_A	primary business key
_B thru _Z	secondary business keys
_SC	key to support slowly changing dimensions

GLOBAL NAMING OF KEY COLUMNS

This option allows users to set the **Global Naming of Key Columns.**



During the drag and drop generation of new dimension and fact tables, WhereScape RED will build an artificial (surrogate) key for the table.

The naming convention for that key can be set through the same menu option as above. As well as potential pre-fix and suffix values, we also need to choose between the inclusion of the full table name, or the short name assigned to each table.

In the example screen above, which is the default, a dimension table key would use the table short name and have a suffix of "_key". So our load_customer table example would generate a key called dim_customer_key if it were dragged into a dimension drop target.

GLOBAL NAMING OF PROCEDURES/SCRIPTS

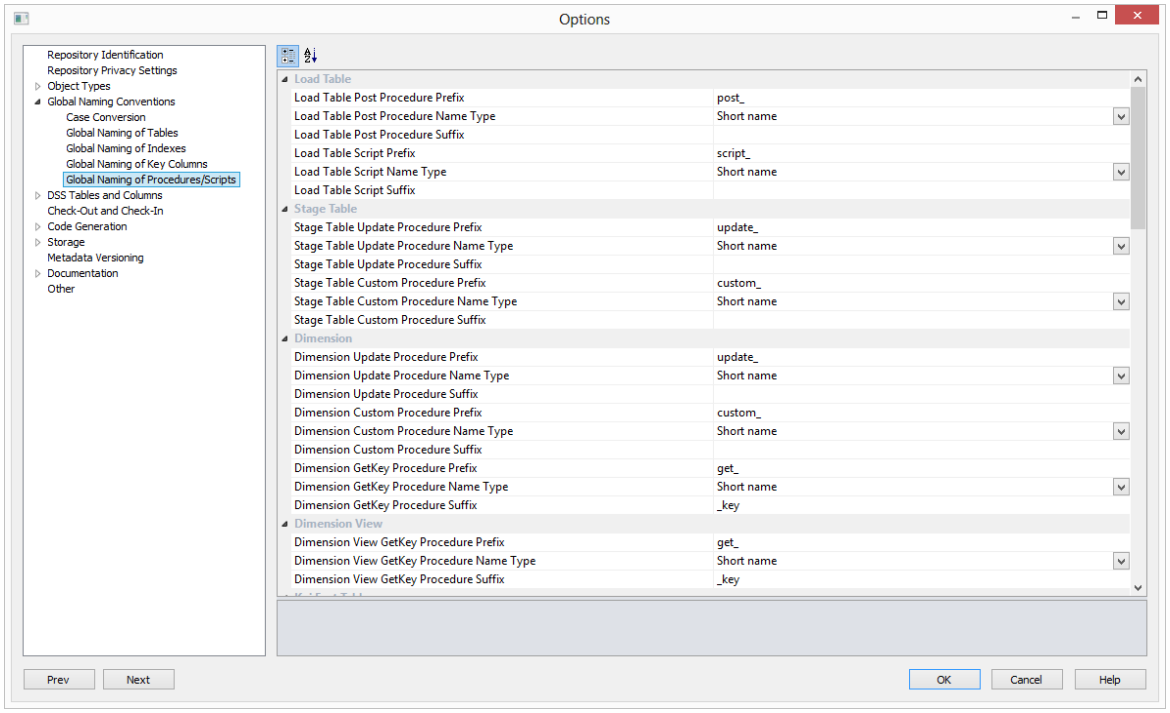
The default naming conventions for generated procedures can be set through the menu option **Tools/Options.**

To generate a procedure, select the **(Build Procedure...)** option from the **Update Procedure** drop list found in the table Properties screen.

Procedure name defaults

The dialog shown in the screen shot below will appear in response to the procedure name defaults menu option. It provides a means of setting the naming defaults for all types of generated procedure.

The values as shown below are the default settings, but may be changed to meet the site requirements. The only restriction is on the size of the resultant name, which is database dependent.



The contents of the **prefix** and **suffix** fields must contain characters that are valid in a database stored procedure name and should preferably not contain spaces.

The **Name Type** may be either the full table name or the unique short name assigned to each table. In the case of smaller table names the short name is usually the same as the table name.

For example, if we have a dimension called dim_product, then from the example screen above the three possible generated procedures would be called update_dim_product, custom_dim_product and get_dim_product_key.

DSS TABLES AND COLUMNS

When building the data warehouse WhereScape RED makes use of a number of special tables and columns. Two tables are used.
These are called by default dss_source_system and dss_fact_table and are discussed in detail in the sections below.
The special columns used are defined in the table below.

Column name	Description
dss_source_system_key	Added to support dimensions that cannot be fully conformed, and the inclusion of subsequent source systems. See the section below for more details.
dss_fact_table_key	Used in composite rollup fact tables to identify the source fact table that contributed the particular row.
dss_create_time	Indicates when a record was created.
dss_update_time	Indicates when the record was last updated in the data warehouse. Used in the updating of rollup fact tables and aggregate tables.
dss_count	Applied to fact tables. Provides a simple row count variable that can be used by end user tools.
dss_current_flag	Used for slowly changing dimensions. This flag identifies the current record where multiple versions exist.
dss_version	Used for slowly changing dimensions. This column contains the version number of a dimension record. Numbered from 1 upwards with the highest number being the latest or current version. It forms part of the unique constraint for the business key of a slowly changing dimension.
dss_start_date	Used for slowly changing dimensions. This column provides a date time stamp when the dimension record began life. If null, then it was the first record. It is used to ascertain which dimension record should be used when multiple is available.
dss_end_date	Used for slowly changing dimensions. This column provides a date time stamp when the dimension record ceased to be the current record. It is used to ascertain which dimension record should be used when multiple is available.

All of these special columns and tables may be renamed through the **Tools/Options/DSS Tables and Columns** menu option. The columns can simply be renamed, but the tables however require valid table names that meet certain criteria. See the appropriate sections below.

Note: When using table names other than the defaults for dss_source_system and dss_fact_table it is worth considering that by default the metadata backups will include any table that begins with "dss_". Therefore, if a table is used it is recommended that it have a name starting with "dss_". The advantages are that a working meta repository will be established through a backup and restore if these tables are included in the backup set.

Dss_source_system

This pseudo dimension is designed to identify a data source for a dimension row. Its purpose is to handle non-conformed dimensions, or changes in source systems. If its use is not desired (default) then leave this field blank.

For example:

An organization has a number of factories. These factories are referenced by all the operational systems. The production system has its own code for each factory and this is the unique means of identifying the factory. The distribution system has a factory short name which it uses for the unique identifier. The raw materials system simply uses the factory name. It is probably not practical or even desirable to force these source systems to utilize a standard factory identification method, so instead we allow the dimension to be non-conformed. We do however, insist on a standard factory name convention, so that our reports and queries will join information when the factory name is used.

In such an example the dss_source_system_key is used to identify the source of the data for the dimension row. It also adds to the unique business key, so that two source systems can utilize the same code to refer to different entities. This key also provides a degree of future proofing in the data warehouse, to assist in the possible changing of an underlying source system.

The generated procedure code will always set the key value of this table to 1. So, manual code changes will be required to make use of the functionality that this table offers.

When dimensions are built the dss_source_system_key is added by default. It can be deleted if the need for such a feature is not seen.

If this table is to be given a different name, then it and all its columns can be renamed or the following steps can be taken:

1. Create a new table by dragging the column dss_source_system_name from dss_source_system into a dimension target.
2. Change the object type from dimension view to dimension and specify the new table name. (see note above on the use of "dss_").
3. Rename the dss_source_system_name column to match the new table name.
4. Delete the last two columns.
5. Under the table Properties, change the table type to **Mapping table**. This will prevent the table from being seen as a dimension in the documentation.
6. Change the dss_source_system table name in the screen above, via the **Tools/Options/DSS Tables and Columns** defaults menu option.

Dss_fact_table

This pseudo dimension is used internally by the WhereScape RED generated procedures to assist in the updating of rollup fact tables. It provides a means of identifying which fact table contributed a particular row to a composite rollup fact table. See the section on rollup fact tables for an explanation of how the generated procedures operate.

This table can be renamed, or a new table created. The table identified as the 'dss_fact_table' must, however, conform to a number of standards if the generated code is to compile and work correctly.

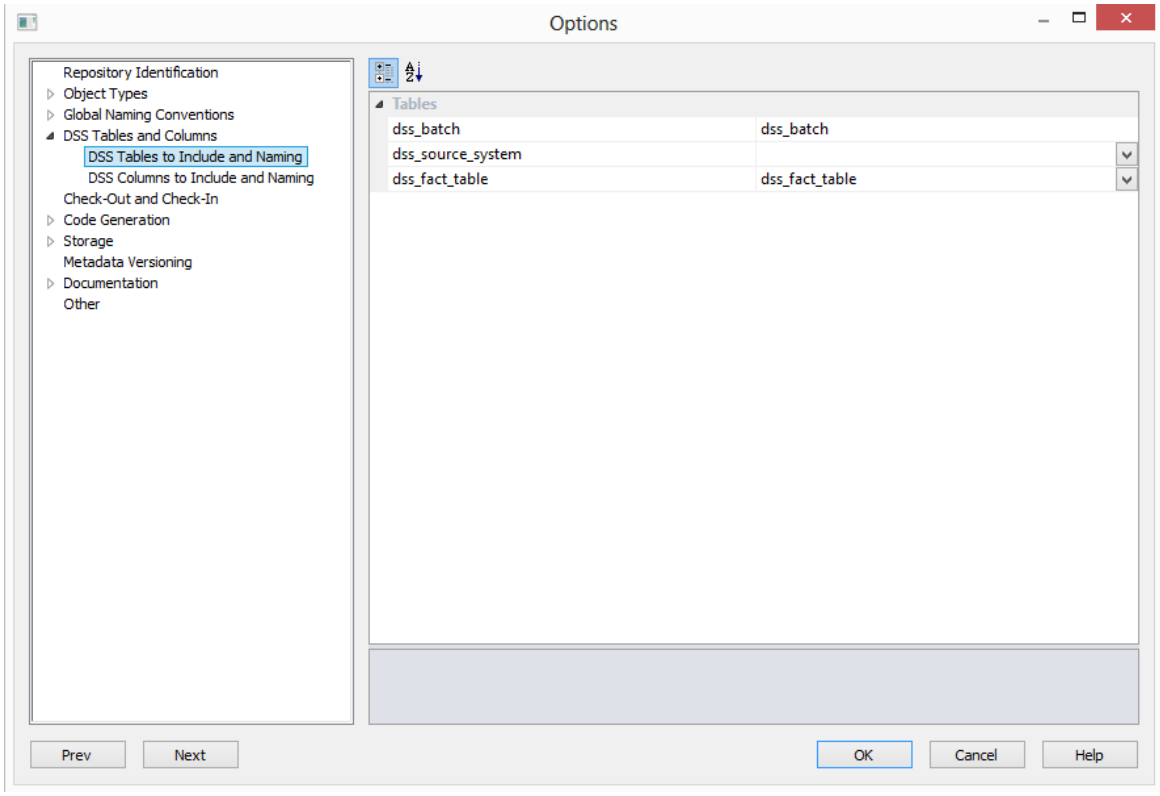
The following columns must exist, where *table_name* is the new name of the table:

Column name	**purpose**
*table_name*_key	The artificial key for the table. Must be a numeric data type.
*table_name*_name	The name of the fact table. Must be at least a varchar2(64) data type.
*table_name*_type	The type of fact table. Must be at least a varchar2(24).

The generated procedures will automatically add rows to this table as required. As with the dss_source_system table, the table type for this table should be set to **Mapping table** to prevent it from being seen as a dimension in the documentation.

DSS TABLES

This option allows you to set the **DSS Tables**.

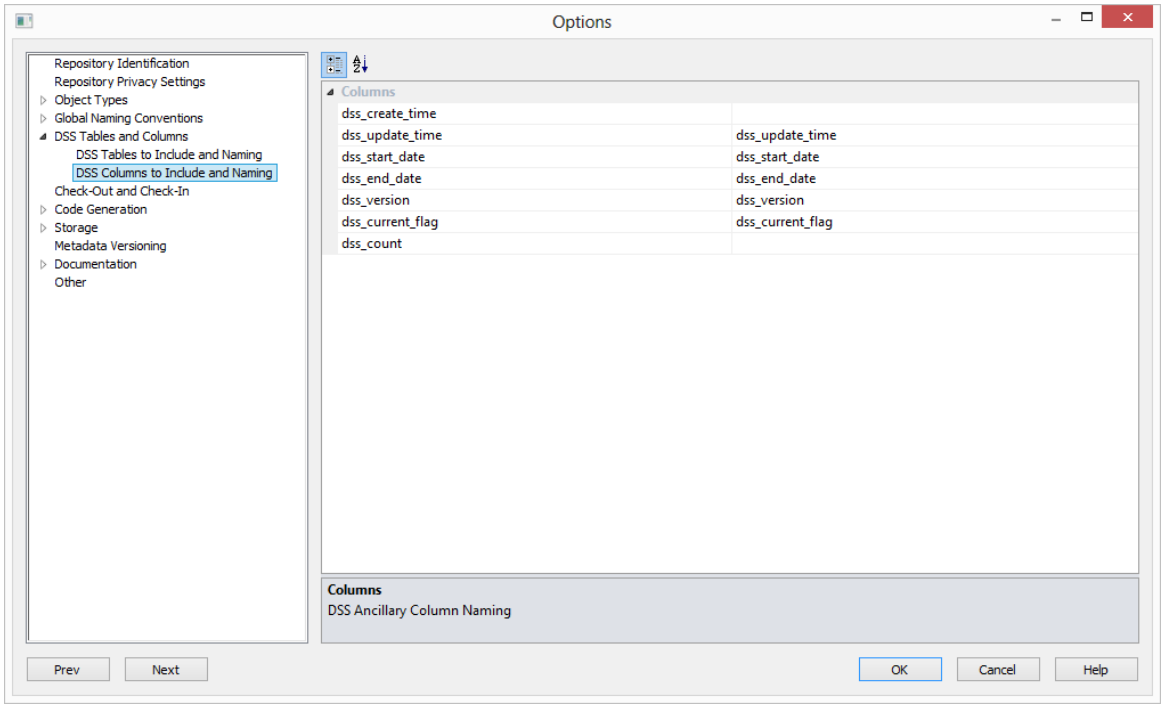


Tables

Set the **DSS Tables**.

DSS COLUMNS

This option allows you to set the **DSS Columns**.



Columns

Set the **DSS Columns**.

dss_create_time

Column added to all stage, ODS, EDW 3NF, dimension, fact and aggregate tables for information only. Leave the field blank to disable or add a name for the to dss_create_time column, i.e. dss_create_time.

dss_update_time

Column added to all dimension, stage and fact tables. It is required if the generated code for fact and aggregate tables is to be used.

dss_start_date

Column used for slowly changing dimensions. It is used to identify when a dimension row was replaced. This is a required field.

dss_end_date

Column used for slowly changing dimensions. It is used to identify when a dimension row was replaced. This is a required field.

dss_version

Column used for slowly changing dimensions. Is it used to store the version of a dimension row. This is required for unique constraints.

dss_current_flag

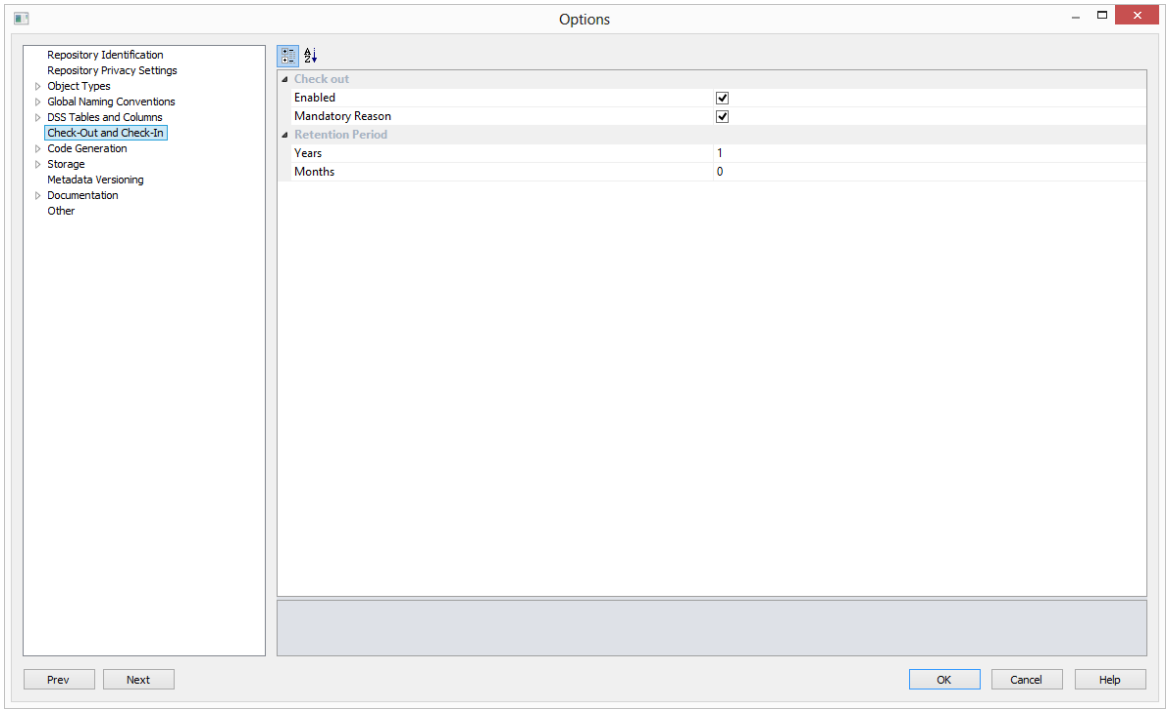
Column used for slowly changing dimensions. It is used to identify the current dimension row. This is a required field.

dss_count

Column added to fact tables to provide a simple row counter. This is required but can be deleted after generation. Leave the field blank to disable.

CHECK-OUT AND CHECK-IN

This option allows users to set up for the **Checking-out** or **Checking-In** of Procedures.



Check out

Enabled: Set to this option to enable procedures to be checked-in or checked-out, else False.

Mandatory Reason: Set this option if a reason is mandatory for checking out or checking in procedures, else False.

Retention Period

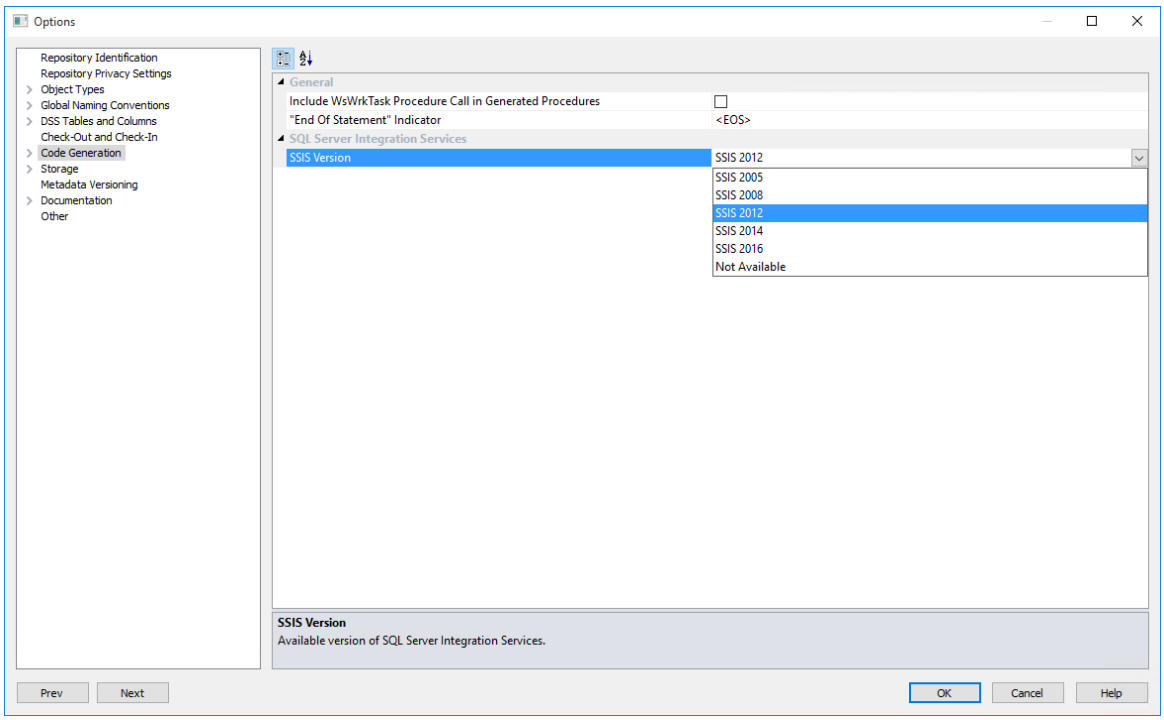
Set the length of time; **Years** and **Months**, for which procedures may be checked-out.

CODE GENERATION

The various options are described below.

GENERAL

This option allows users to set some general **Code Generation** settings.



General

Include WsWrkTask Procedure: When set to **True**, this will result in a call to the WsWrkTask function being placed at the end of most of the generated update procedures. These calls to WsWrkTask result in counters being set in the meta table ws_wrk_task_log. These counters can be viewed via a query on the view **ws_admin_v_task**.

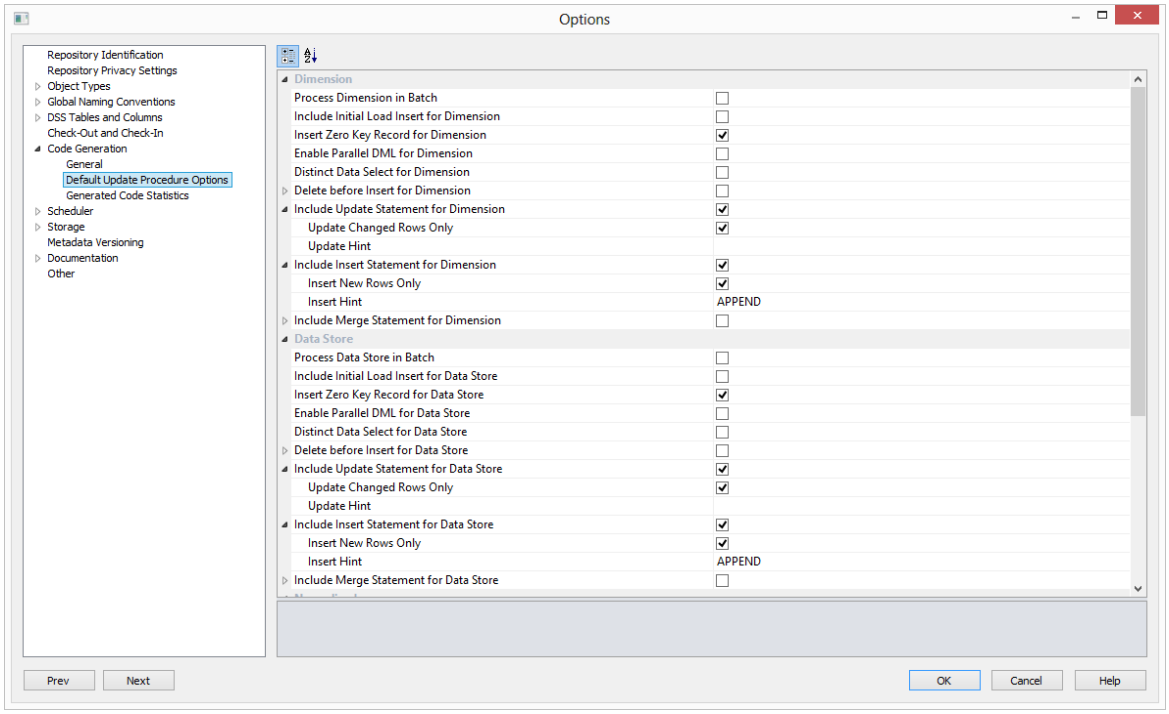
"End of Statement" Indicator: Set the indicator to separate multiple SQL statements in a SQL block. If left blank the default value of <EOS> is used.

SQL Server Integration Services

This field sets the **version** of SSIS being used, otherwise the **Not Available** option is set. To use SSIS to load data, the relevant version of SSIS needs to be selected on this drop-down list.

DEFAULT UPDATE PROCEDURE OPTIONS

This option allows users to set some **default update procedure** settings for specific object types. Not all options are relevant for all combinations of database, object type and build type.
The screenshot below shows an example of an **Oracle** repository's **Default Update Procedure Options** screen.

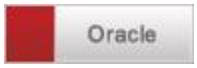


Default Update Procedure Options

Process in Batch - when selected, this option allows users to select a column to drive data processing in a loop based on the distinct ordered values of the selected column.

Include Initial Load Insert - when selected, this options adds an additional insert statement to the update procedure. If the target table is empty, the new insert statement is run in place of the standard generated code.

Insert Zero Key Record - when selected, this option adds an insert statement for an unknown record with an artificial key of zero. Only applicable to tables with an artificial key.



Enable Parallel DML - When selected, this option adds all code required to the update procedure for enabling Oracle parallel inserts. This option is available only for **Oracle**.

Distinct Data Select - when selected, this option ensures duplicate rows are not added to the table.

Select Hint - enter a database-compliant hint to be used in the SELECT statement. Parameters $TABLE$ and $INDEXS will be automatically replaced at procedure generation time.

Delete before Insert - when selected, this option enables a delete statement to be added to the update procedure before any update or insert statement.

Truncate - performs a DDL operation to delete all records from a table, rather than a predicate based DML operation to delete individual rows. This option is automatically enabled when **Delete before Insert** is enabled, but can be disabled separately.

Include Update Statement - when selected, this option includes an update statement in the procedure to update changing rows in the table.

Update Changed Rows Only - when selected, this option uses change detection to work out what rows require updating.

Update Hint - this option allows entering a default database hint to be used in the UPDATE statement. This option is available for **SQL Server** and **Oracle** only.

Oracle default hint is APPEND. SQL Server default hint is TABLOCK.

Include Insert Statement - when selected, this option includes an insert statement in the procedure to insert new rows in the table.

Insert New Rows Only - when selected, this option uses change detection to work out what rows require inserting.

Insert Hint - this option allows entering a default database hint to be used in the INSERT statement. This option is available for **SQL Server** and **Oracle** only.

Oracle default hint is APPEND. SQL Server default hint is TABLOCK.

Include Merge Statement - when selected, this option includes a merge statement in the procedure to merge new/changed rows in the table.

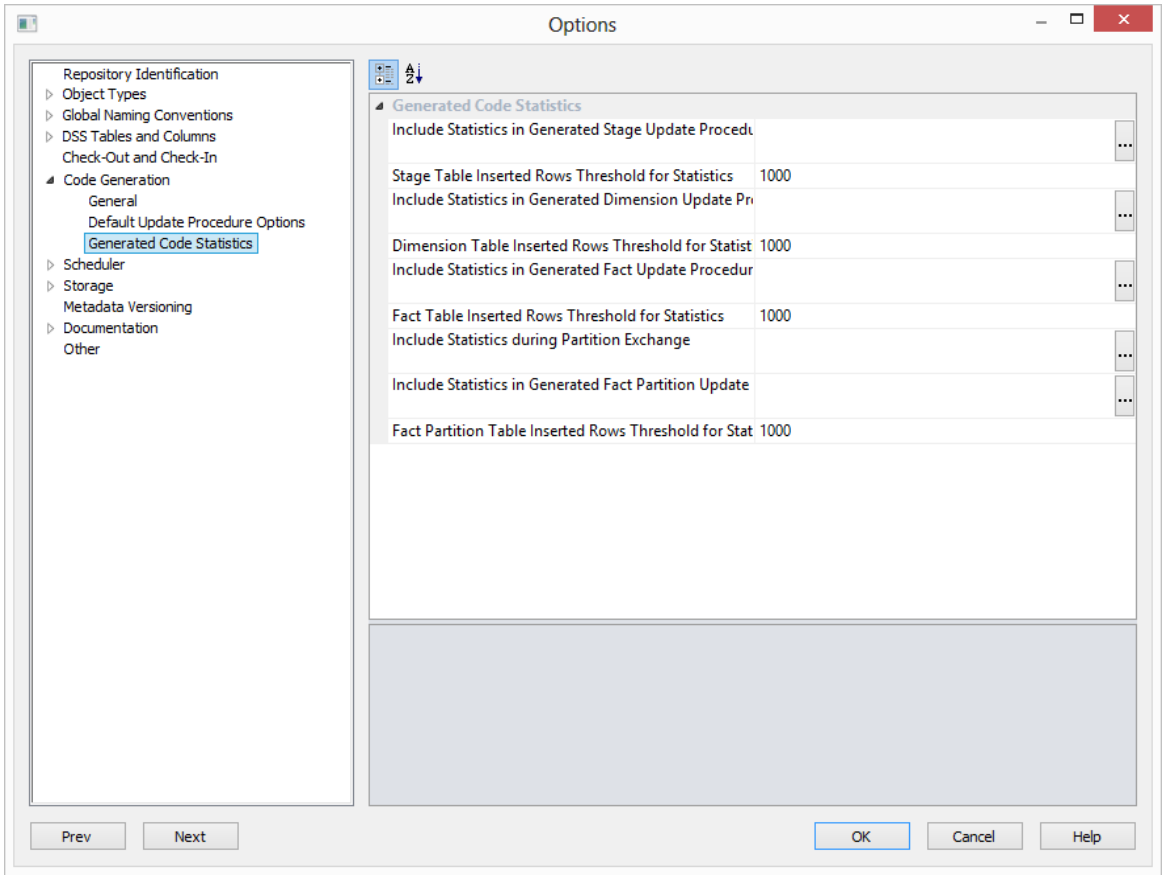
Merge New/Changed Rows Only - when selected, this option uses change detection to interpret which rows require merging.

Merge Hint - this option allows entering a default database hint to be used in the MERGE statement. This option is available for **SQL Server** and **Oracle** only.

Oracle default hint is APPEND. SQL Server default hint is TABLOCK.

GENERATED CODE STATISTICS

This option is only available for **Oracle** databases.



Generated Code Statistics

The **Include Statistics in Generated Stage Update Procedure** option sets the statistics command to be included in the automatically generated procedure for Stage tables. Parameter $TABLE$ will be automatically replaced by the table name. The key word $UNUSED$ will remove the statistics portion from the generated code in the procedure.

The **Stage Table Inserted Rows Threshold for Statistics** option sets the threshold parameter to be used by the four statistic commands that are inserted into the automatically generated procedures. It indicates how many rows are to be inserted before the statistic commands are executed.

The **Include Statistics in Generated Dimension Update Procedure** option sets the statistics command to be included in the automatically generated procedure for Dimension tables. Parameter $TABLE$ will be automatically replaced by the table name. The key word $UNUSED$ will remove the statistics portion from the generated code in the procedure.

The **Dimension Table Inserted Rows Threshold for Statistics** option sets the threshold parameter used by the four statistic commands that are inserted into the automatically generated procedures. It indicates how many rows are to be inserted before the statistic commands are executed.

The **Include Statistics in Generated Fact Update Procedure** option sets the statistics command to be included in the automatically generated procedure for Fact tables. Parameter $TABLE$ will be automatically replaced by the table name. The key word $UNUSED$ will remove the statistics portion from the generated code in the procedure.

The **Fact Table Inserted Rows Threshold for Statistics** option sets the threshold parameter to be used by the four statistic commands that are inserted into the automatically generated procedures. It indicates how many rows are to be inserted before the statistic commands are executed.

The **Include Statistics during Partition Exchange** option sets the statistics command to be included in the automatically generated procedure during partition exchanges. Parameter $TABLE$ will be automatically replaced by the table name. The key word $UNUSED$ will remove the statistics portion from the generated code in the procedure.

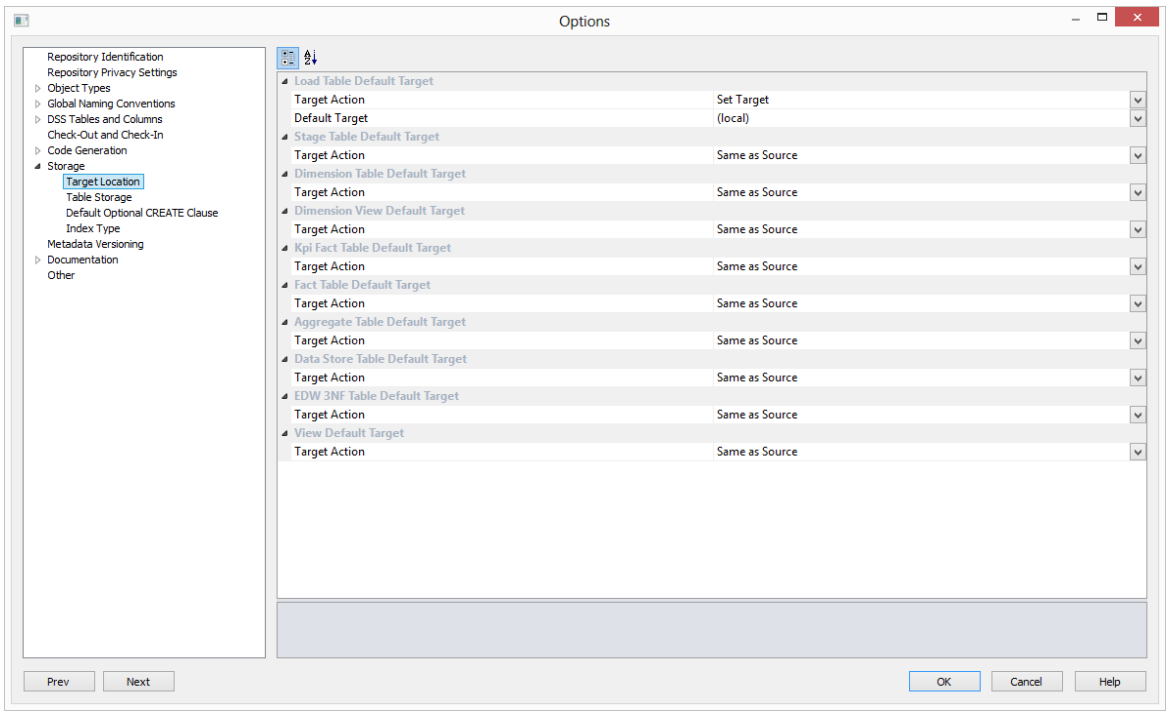
STORAGE

The various options are described below.

TARGET LOCATION

Table Location Options enables users that are placing objects across multiple schemas to set default target locations for new tables.
Default table target locations can be set for the following objects:

- **Load**
- **Stage**
- **Dimension**
- **Dimension View**
- **Kpi Fact**
- **Fact**
- **Aggregate**
- **Data Store**
- **EDW 3NF**
- **View Default**



Target Action

Set Target

This option enables users to set a default target for new tables to be created. It enables the **Default Target** drop-down list where a specific target for new tables can be defined.

Same as Source

This option should be selected if the table's default storage should be same as the original source where the table is coming from. This option cannot be selected for Load Tables.

Default Target

A default target can only be entered if the **Set Target** action has been selected in the **Target Action** drop-down list.
With this option, users can choose between setting a table's default location to **(local)** or to any other **target locations** that have been defined in the relevant connections.

To set a default target location on a table by table basis:

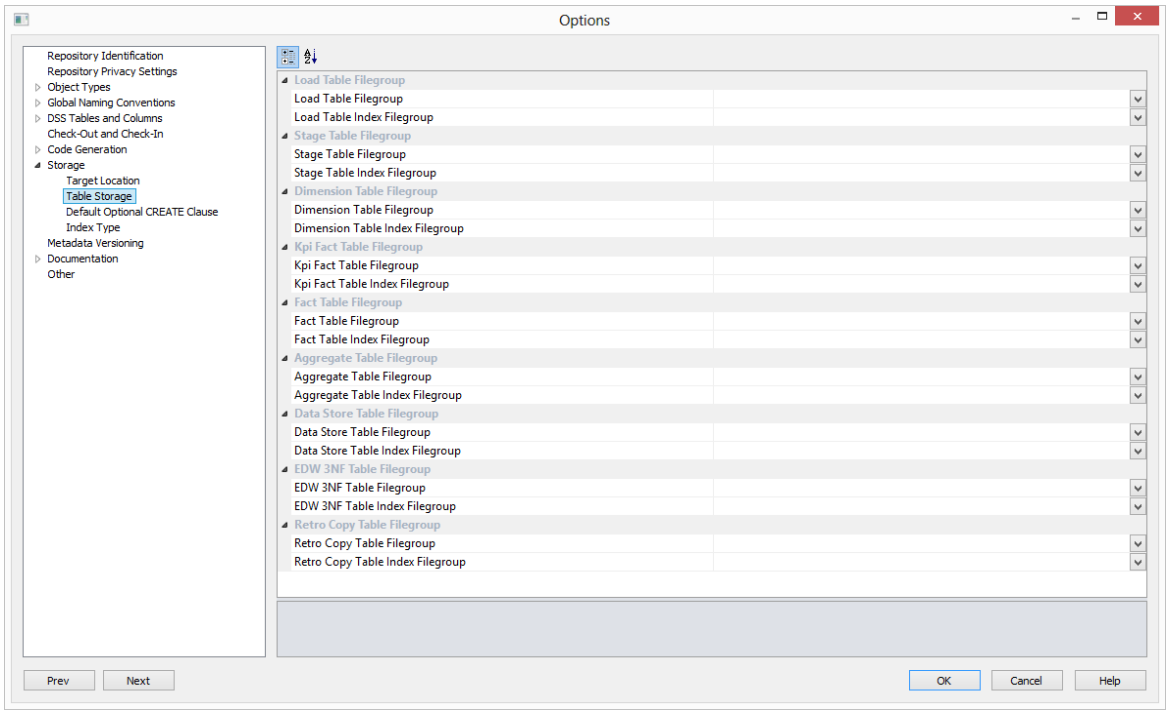
1. Select the **Set Target/Same as Source** option from the **Target Action** drop-down menu.
2. To have tables located in a specific target, select a **default target** where the new object should be placed as the object is dragged and dropped to the middle work pane.

To see more on creating Target table locations see ***Connections to the Data Warehouse/Metadata Repository*** (see "***Database - Data Warehouse/Metadata Repository***" on page 143) for SQL and Oracle Databases and ***Connections - Database*** or ***Connections - ODBC*** for target based databases such as Netezza, Greenplum and PDW.

Even though the default target can be set in the **Target Location** Options, this setting can also be changed after the table has been created in the **Storage** tab of each table's Properties screen. To see more information about changing the schema after a table has been created, see ***Table Storage Properties***.

TABLE STORAGE

This option allows users to set the **Storage** locations and to set the type of indexes built for foreign key columns.

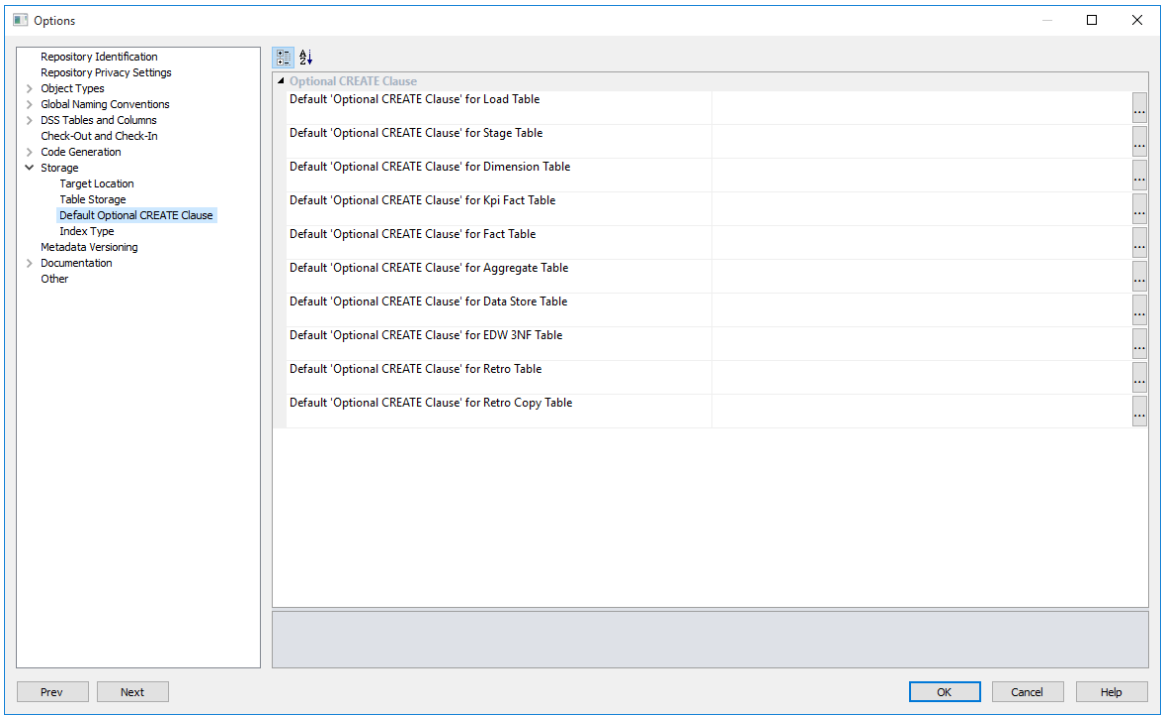


Set the **Storage** locations for each table type.

These defaults are applied when a table is created. They can be changed by selecting the **Storage** tab on the **Properties** screen of a table.

DEFAULT OPTIONAL CREATE CLAUSE

This option allows defining a default value for the "Optional CREATE Clause" property of each object type, which is populated when the object is first created.
The Optional CREATE Clause text is appended to the DDL CREATE statement when the table is generated. An example use of this option is to define default settings for parallelism and logging in an Oracle database - see below for further details.

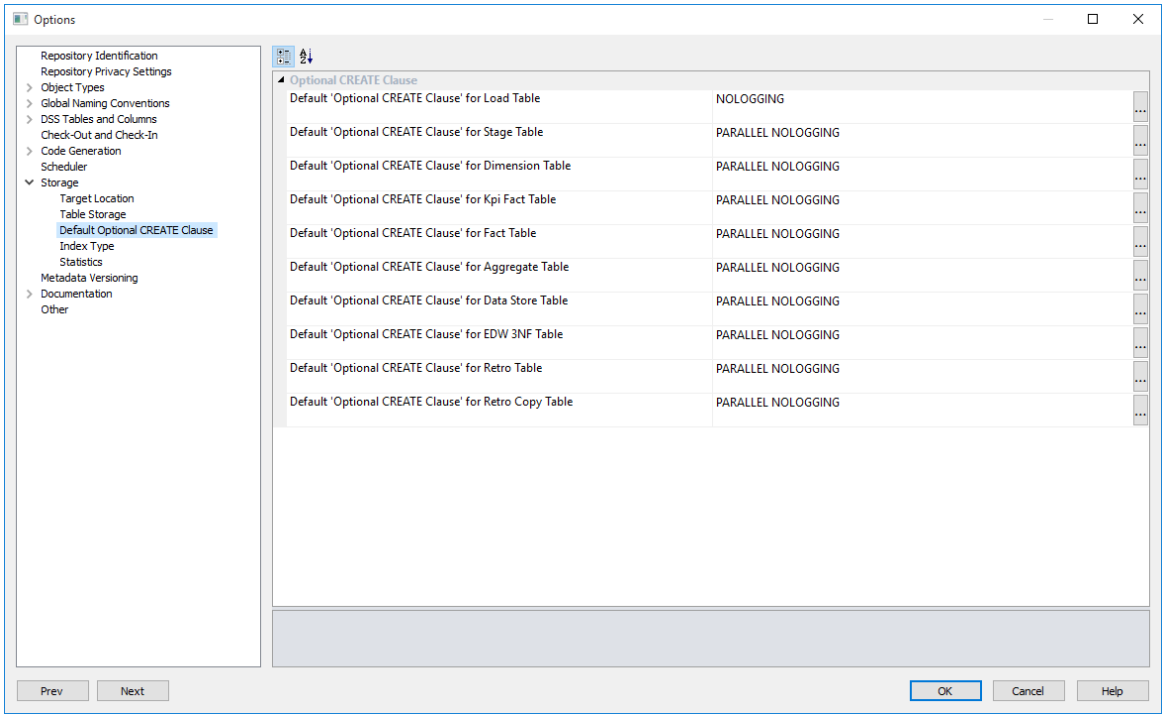


TIP: This option is only to set the default optional create clause for new objects.
To edit the Optional CREATE Clause of an **existing** object or edit the clause on a table by table basis, go to the object's **Properties** screen, click on the **Storage** tab and edit the **Optional CLAUSE Clause** field.
See ***Table Storage Properties*** for more details.

ORACLE

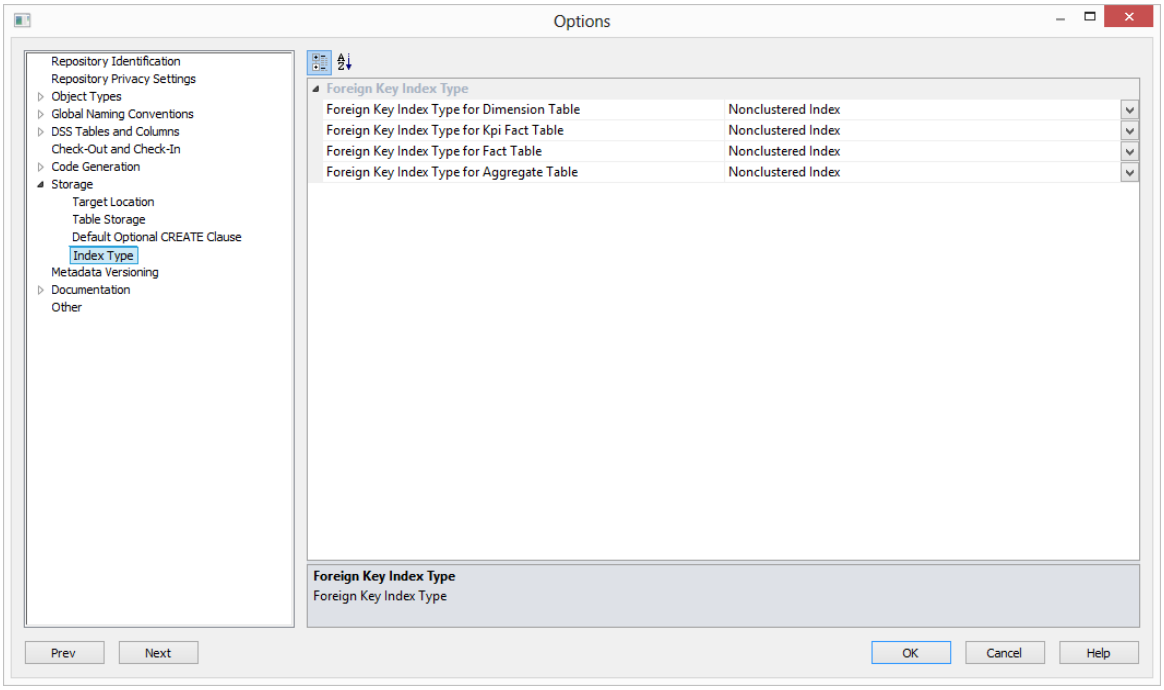
Oracle's default values are set to NOLLOGGING for Load Tables and PARALLEL NOLOGGING for remaining objects.

- To edit the default optional create clause for each object, replace the statement on the free text field or click on the ellipsis button.



INDEX TYPE

This option allows users to set the default type of **Foreign Key Index Type** for each table type.



Foreign Key Index Type

Set the default index types built for foreign key columns of Dimension tables, KPI Fact tables, Fact tables and Aggregate tables.

SQL Server and **DB2** options:

- (None) - we will not define
- BTREE - this is a standard index per key column
- COVERING - this is a single index with all the key columns in it

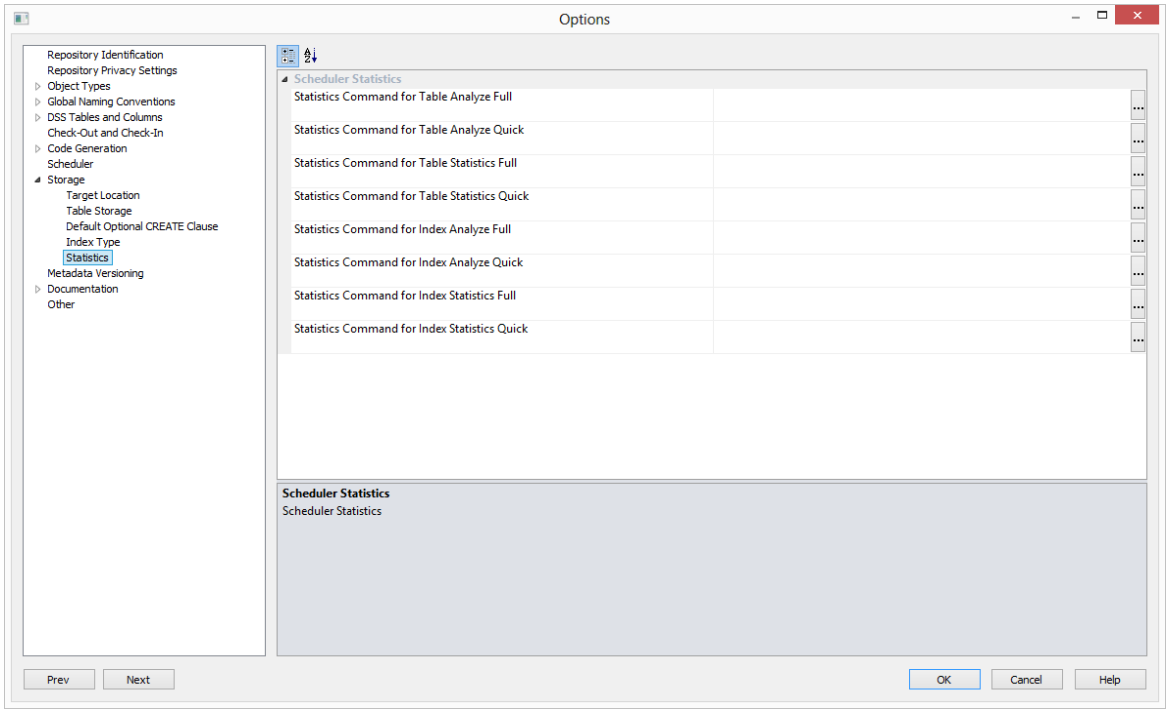
Oracle options:

- (None) - will not be defined
- BITMAP - This is the standard oracle bitmap index
- BTREE - this is a standard index per key column

These defaults are applied when an index definition is created.
They can be changed by selecting the **Storage** tab on the Properties screen of an index.

STATISTICS

Scheduler Statistics - this option is only available for **Oracle** databases.
Statistics command entered here will be executed when using the scheduler with Stats, Quick Stats, Analyze or Quick Analyze actions to gather statistics on tables.
Parameter **$TABLE$** will be automatically replaced by the table name, parameter **$INDEX$** will be replaced by the index name; and parameter **$SCHEMA$** will be replaced by the owner name. Please see examples of this below.
For more information about adding statistics tasks to jobs see ***Editing Tasks in a Job*** (on page 767).

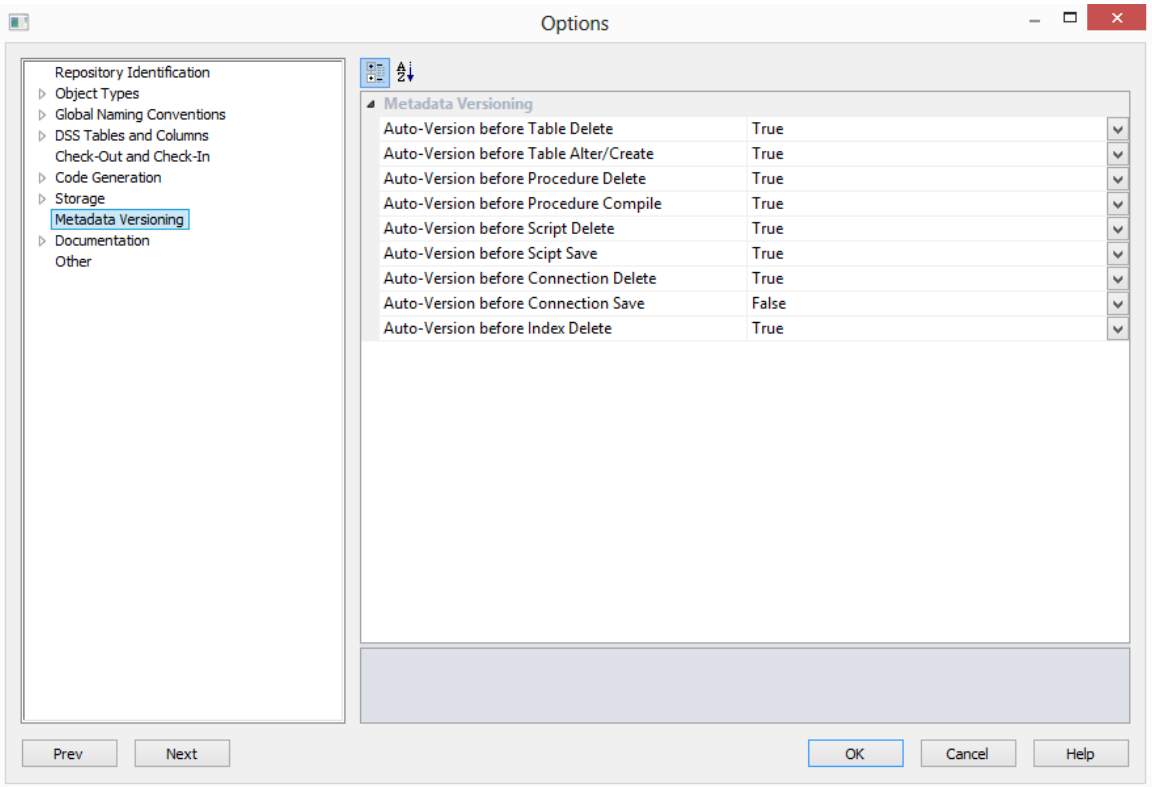


Examples:

Scheduler Statistics	
Statistics Command for Table Analyze Full	**analyze table $TABLE$ estimate statistics sample 35 percent**
Statistics Command for Table Analyze Quick	**analyze table $TABLE$ estimate statistics sample 1000 rows**
Statistics Command for Table Statistics Full	**BEGIN dbms_stats.gather_table_stats(ownname=>'$SCHEMA',tabname=>'$TABLE$',estimate_percent=>'50',degree=>16); END;**
Statistics Command for Table Statistics Quick	**BEGIN dbms_stats.gather_table_stats(ownname=>'$SCHEMA',tabname=>'$TABLE$',estimate_percent=>'10',degree=>16); END;**
Statistics Command for Index Analyze Full	**analyze index $INDEX$ estimate statistics sample 10 percent**
Statistics Command for Index Analyze Quick	**analyze index $INDEX$ estimate statistics sample 1 percent**
Statistics Command for Index Statistics Full	**BEGIN dbms_stats.gather_index_stats(ownname=>'$SCHEMA',indname=>'$INDEX$',estimate_percent=>'20',degree=>16); END;**
Statistics Command for Index Statistics Quick	**BEGIN dbms_stats.gather_index_stats(ownname=>'$SCHEMA',indname=>'$INDEX$',estimate_percent=>'5',degree=>16); END;**

METADATA VERSIONING

This option allows users to alter the **Metadata Versioning** settings.

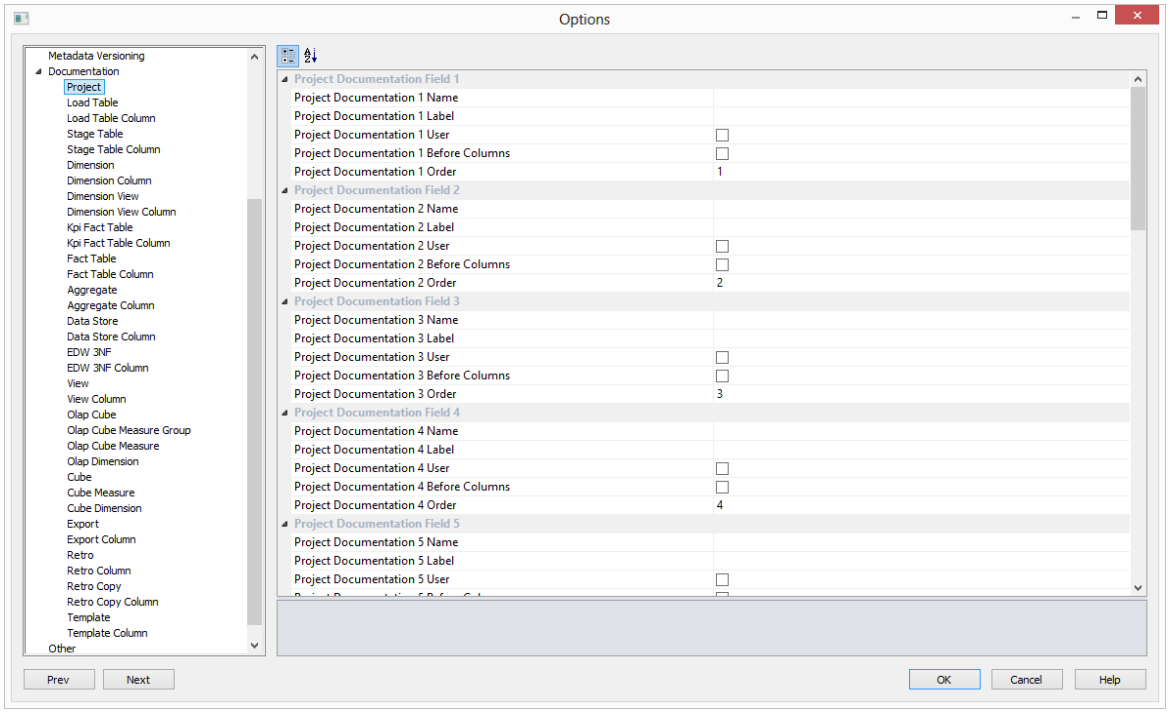


Metadata Versioning

Select **True** or **False** to set when to auto-version the metadata.

DOCUMENTATION

This options allows users to alter the **Documentation** settings.



The **Documentation Name** sets the name of the appropriate tab in the Properties dialog.

The **Documentation Label** sets the label or description of the appropriate documentation tab.

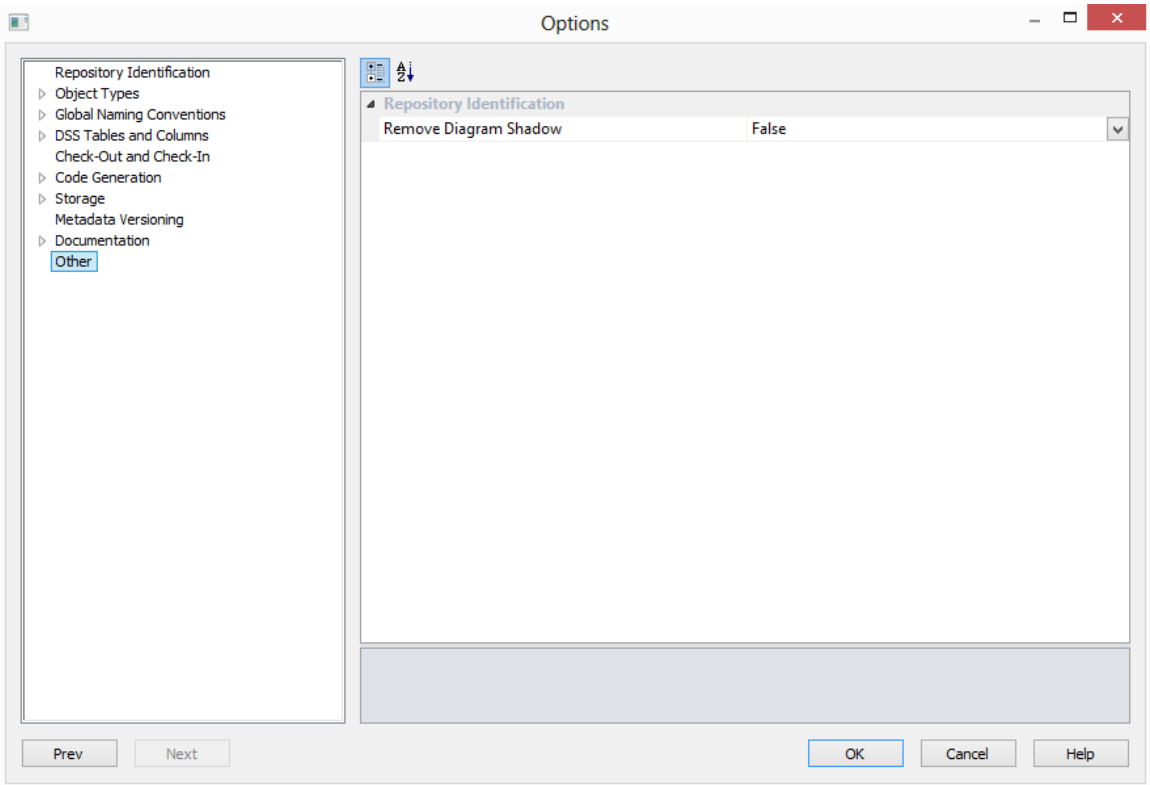
The **Documentation User** defines if the documentation information is visible to end users and included in end user documentation.

The **Documentation Before Columns** defines if the documentation tab information is shown in the documentation before or after the column information.

The **Documentation Order** defines the field order on the Properties dialog tabs.

OTHER

This option allows users to add or remove shadows in the diagrams.

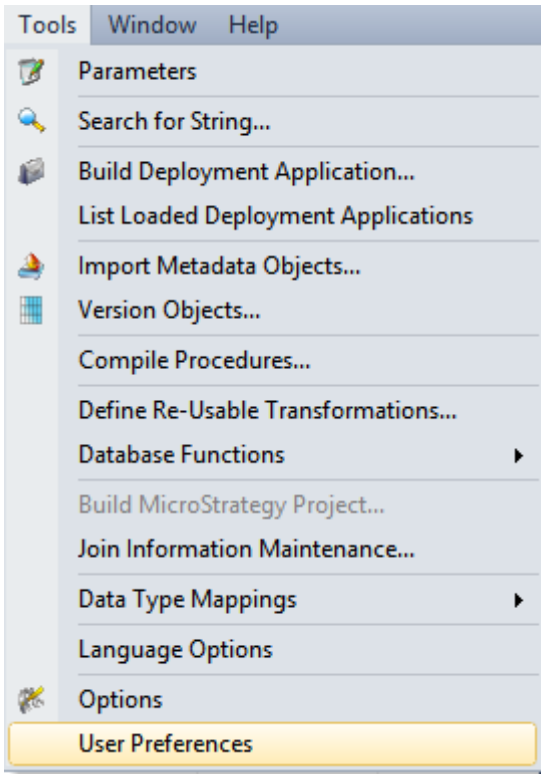


Repository Identification

Set to **True** to prevent a shadow appearing on all printable diagrams produced in the diagrammatic window; else False.

USER PREFERENCES

Select **User Preferences** from the **Tools** menu.



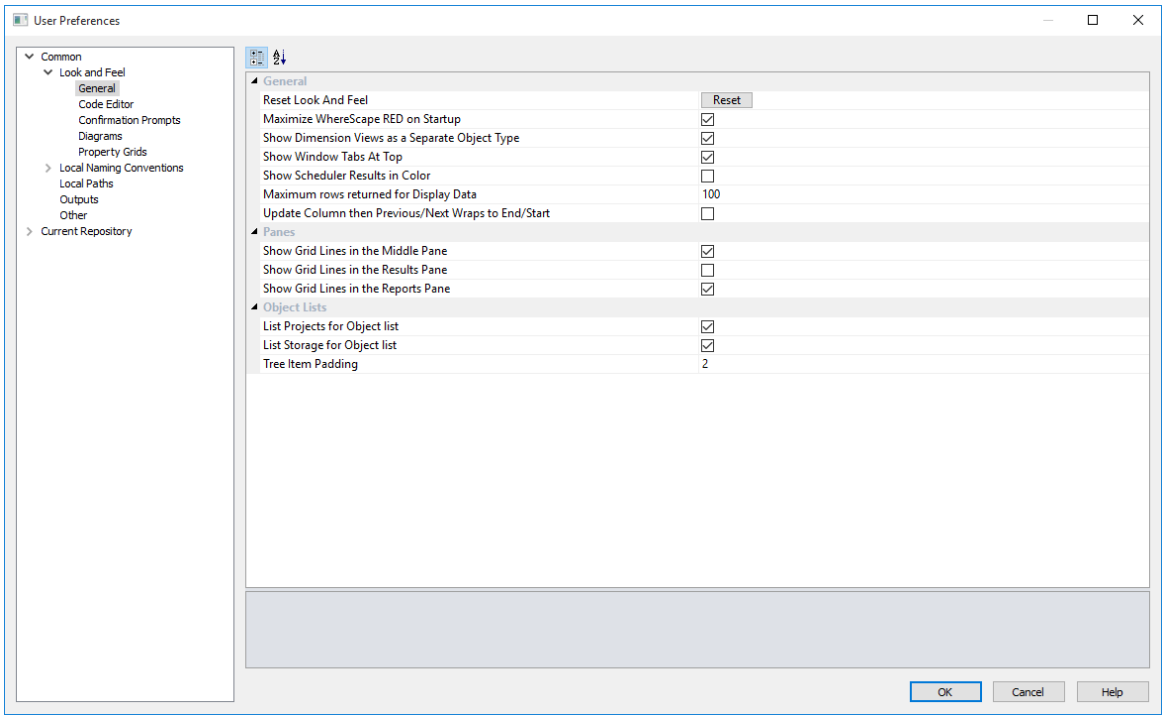
COMMON

LOOK AND FEEL

The various options are described below.

General

This option allows you to set the **Look and Feel** in **General.**



General

The **Reset Look And Feel** option allows you to reset all window tab positions for Builder and Scheduler panes. Reset scheduler and report headings.

The **Maximize WhereScape RED on Startup** option, when selected, starts WhereScape RED in full screen mode.

The **Show Dimension Views as a Separate Object Type** option, when selected, displays a new object group called Dimension Views where all view objects are stored.

The **Show Window Tabs At Top** option, when selected, displays window tabs at the top of the screen.

The **Scheduler Results in Color**, when selected, turns on job status color coding in the scheduler.

The **Maximum rows returned for Display Data** option, sets the maximum number of rows that will be returned when displaying data.

The **Update Column then Previous/Next Wraps to End/Start** option, when selected, controls the behavior of the directional **Update** buttons on the Column Properties dialogs. When enabled the '<-Update' button will wrap to the last column when it moves beyond the first column; and the 'Update ->' button will wrap to the first column when it moves beyond the last column. When disabled (default), the dialog closes after an attempt to navigate before the first column or after the last column.

Panes

The **Show Grid Lines in the Middle Pane** option, when selected, shows grid lines in the main work area.

The **Show Grid Lines in the Results Pane** option, when selected, shows grid lines in the results area.

The **Show Grid Lines in the Reports Pane** option, when selected, shows grid lines in the reports area.

Object Lists

The **List Projects for Object list** option, when selected, shows the projects for each object in the middle pane object list.

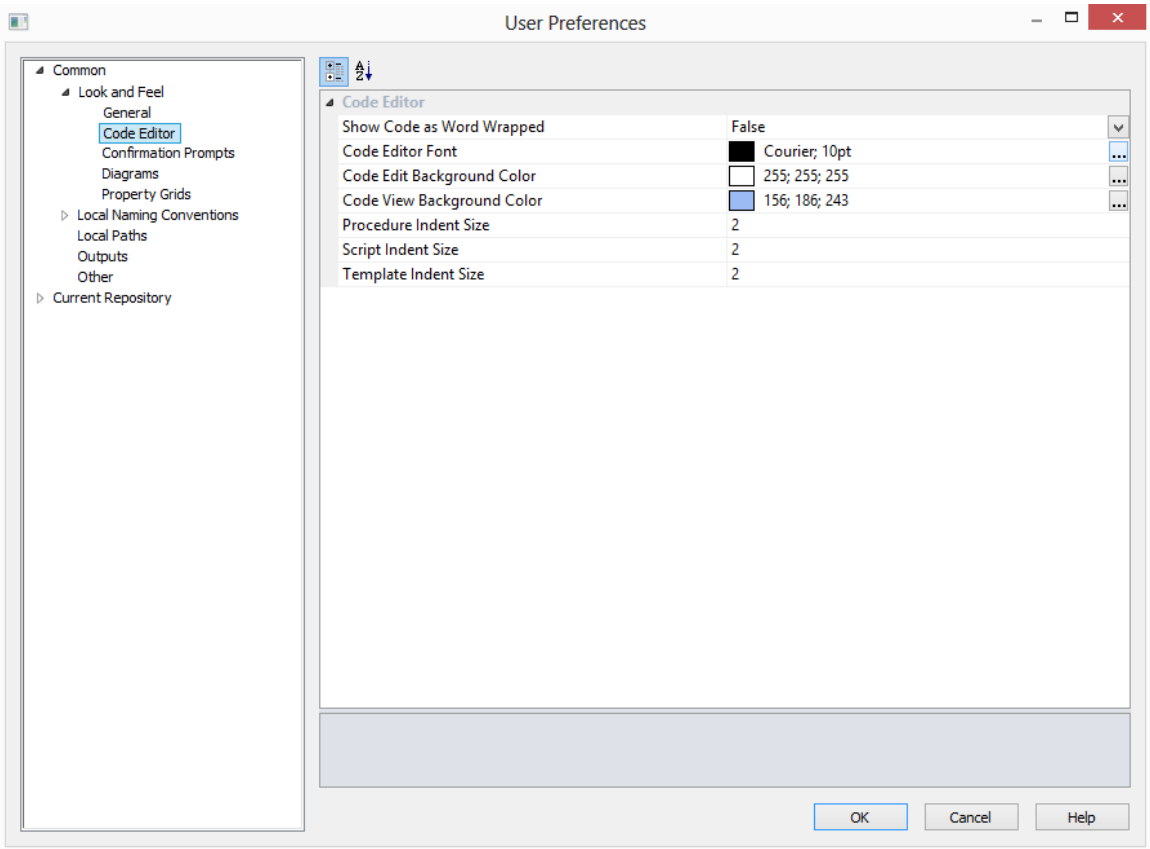
The **List Storage for Object list** option, when selected, shows the storage for each object in the middle pane object list.

The **Tree Item Padding** option allows you to select the number of pixels used to pad tree items when the tree items represent an Object; for example, in the Object Pane and the Browser Pane. Padding is added to the top and bottom of each tree item. Padding can be set from 0-10 pixels, default value is 2.

Note: When selected, both these options impact on the speed lists are generated. Since they are enabled by default, both options should be disabled to speed up the process or if considered irrelevant according to user's preferences.

Code Editor

This option allows you to set the **Look and Feel** in **Code Editor.**



Code Editor

The **Show Code as Word Wrapped** option, when selected, defaults to have word wrapping applied to code.

The **Code Editor Font** option allows you to select the font used in the code editors.

The **Code Editor Background Color** option allows you to select the background color when editing code.

The **Code View Background Color** option allows you to select the background color when viewing code.

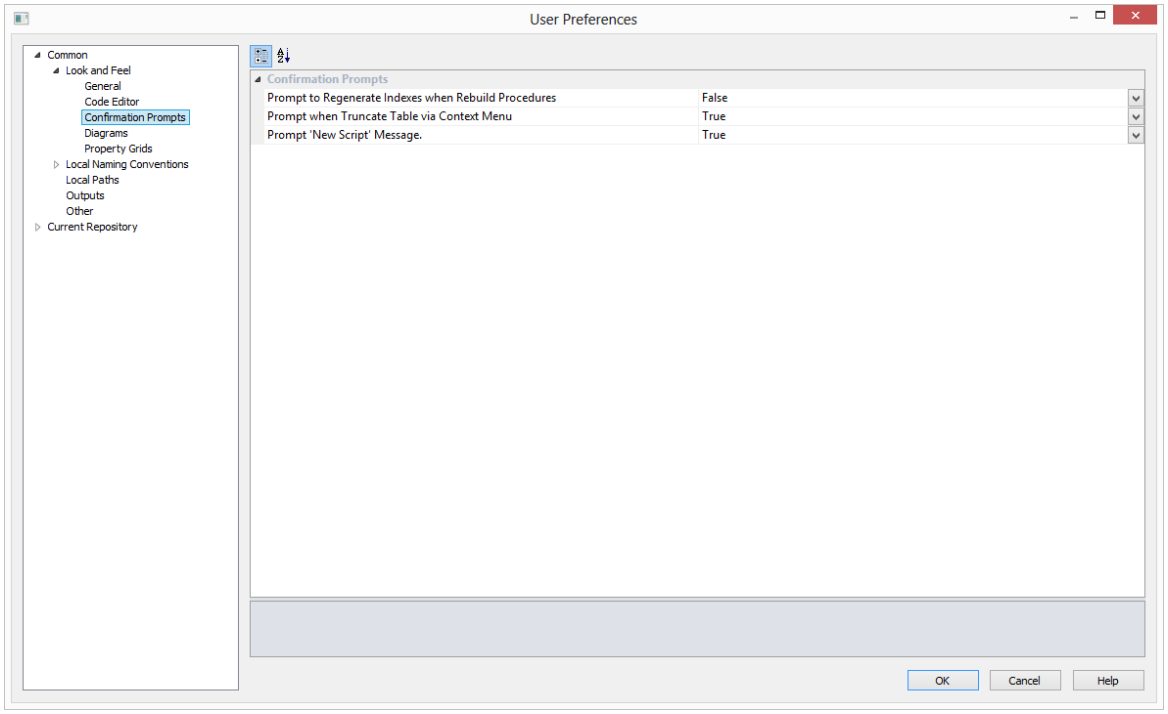
The **Procedure Indent Size** option allows you to specify the number of spaces that are generated when a TAB character is used within the Procedure editor. Permitted range is 2 through 10.

The **Script Indent Size** option specifies the number of spaces that are generated when a TAB character is used within the Script editor. Permitted range is 2 through 10.

The **Template Indent Size** option specifies the number of spaces that are generated when a TAB character is used within the Template editor. Permitted range is 2 through 10.

Confirmation Prompts

This option allows you to set the **Look and Feel** in **Confirmation Prompts**.



Confirmation Prompts

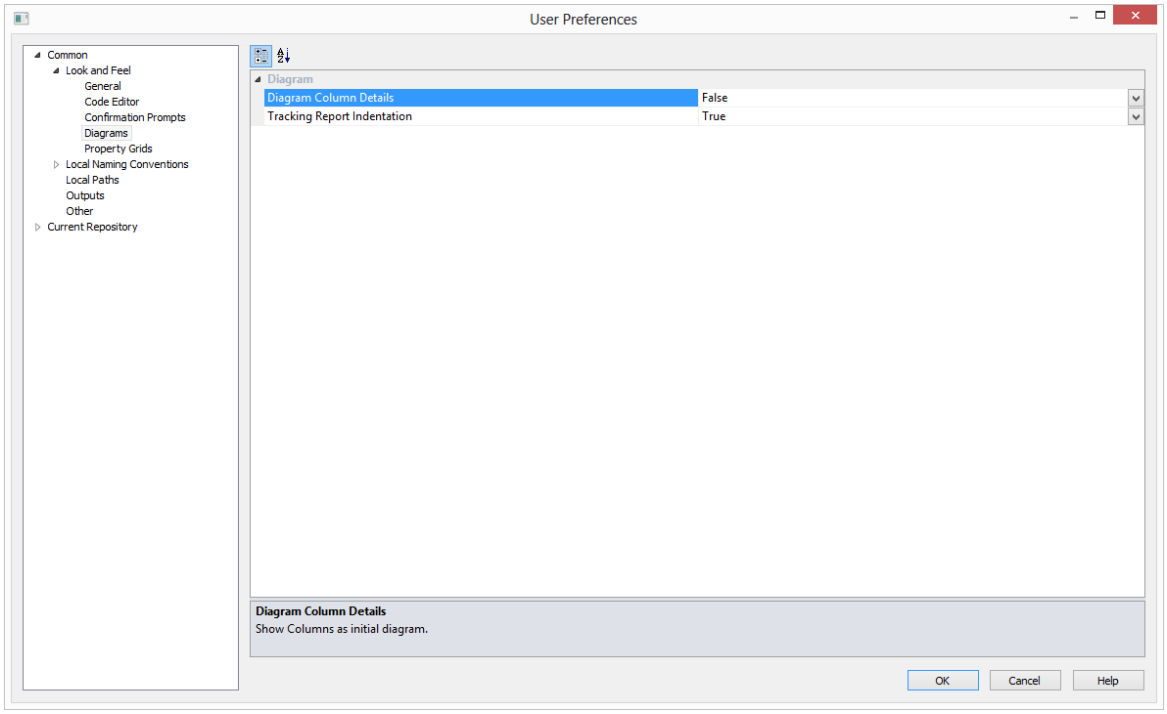
Prompt to Regenerate Indexes when Rebuild Procedures - If set, always prompts for index regeneration whenever an update procedure is rebuilt

Prompt when Truncate Table via Context Menu - If set, always pops up a confirmation message before the truncate command is executed

Prompt "New Script" Message - If set, always pops up an assistance message with expected return codes for scripts

Diagrams

This option allows you to set the **Look and Feel** in the **Diagrams**.



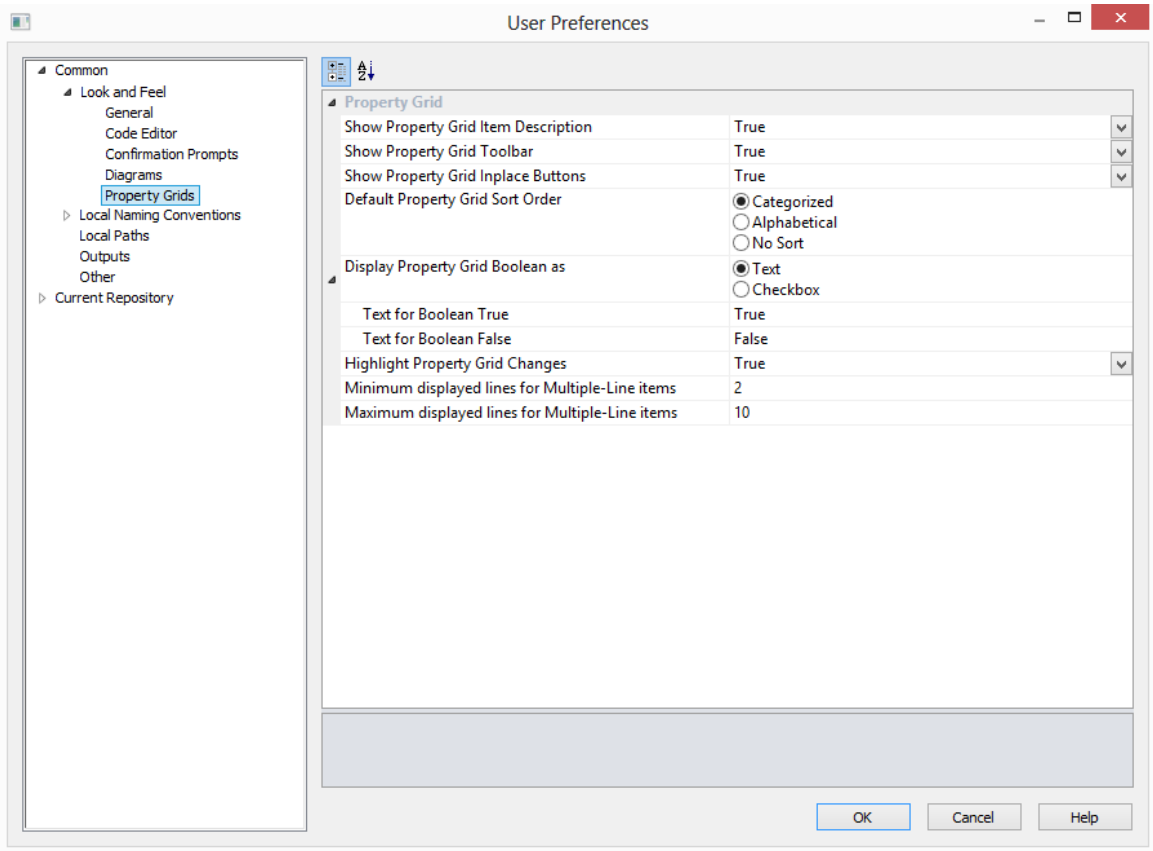
Diagram

The **Diagram Column Details** option will show the columns as the initial diagram.

When set, the **Tracking Report Indentation** output will include tabs to show dependency level.

Property Grids

This option allows you to set the **Look and Feel** in **Property Grids**.



Property Grid

The **Show Property Grid Item Description** option, when selected, shows the property grid item description. The default is selected.

The **Show Property Grid Toolbar** option, when selected, shows the property grid toolbar. The default is selected.

The **Show Property Grid Inplace Buttons** option, when selected, shows property grid buttons for all items. The default is selected.

The **Default Property Grid Sort Order** option, allows you to select the default property grid sort order for items. The options are Categorized, Alphabetical and No Sort and the default is Categorized.

The **Display Property Grid Boolean as** option, allows you to select how boolean items are to be displayed. The options are Text and Checkbox and the default is Text.

The **Text for Boolean True** option, allows you to enter the text for the boolean value True.

The **Text for Boolean False** option, allows you to enter the text for the boolean value False.

The **Highlight Property Grid Changes** option, allows you to highlight changed items in the property grid. The default is True.

The **Minimum displayed lines for MultiLine items** option, gives the minimum display lines for multiline inputs.

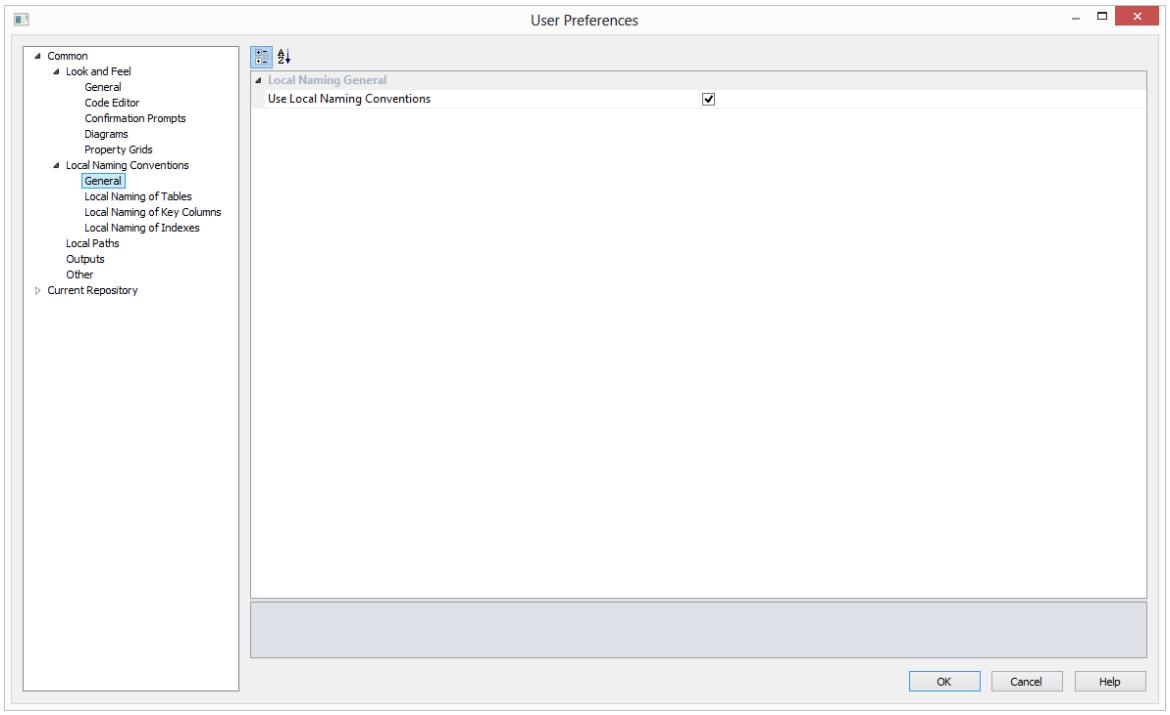
The **Maximum displayed lines for MultiLine items** option, gives the maximum display lines for multiline inputs.

LOCAL NAMING CONVENTIONS

The various options are described below.

General

This option allows you to set the **Local Naming Conventions**.



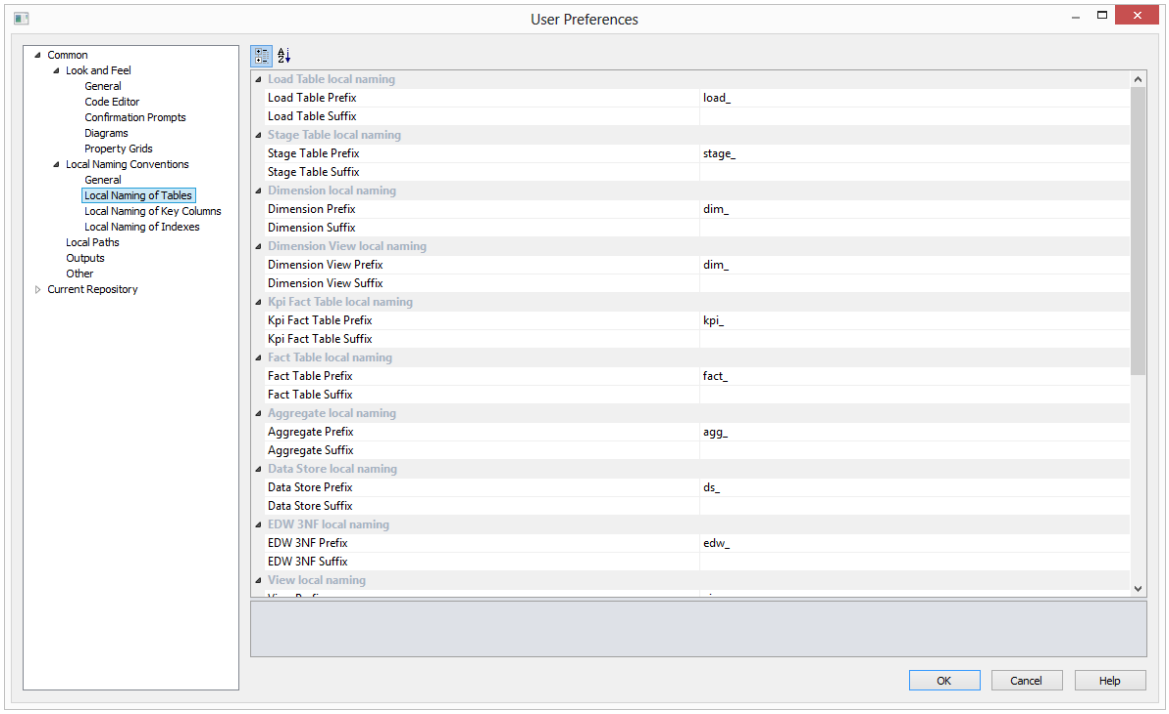
Local Naming General

Set this option if you want to **Use Local Naming Conventions.** If this this option is set the **Local Naming of Tables**, **Key Columns** and **Indexes** options will be enabled in the tree.

Note: If this option is set, it can overwrite short names and object prefixes.

Local Naming of Tables

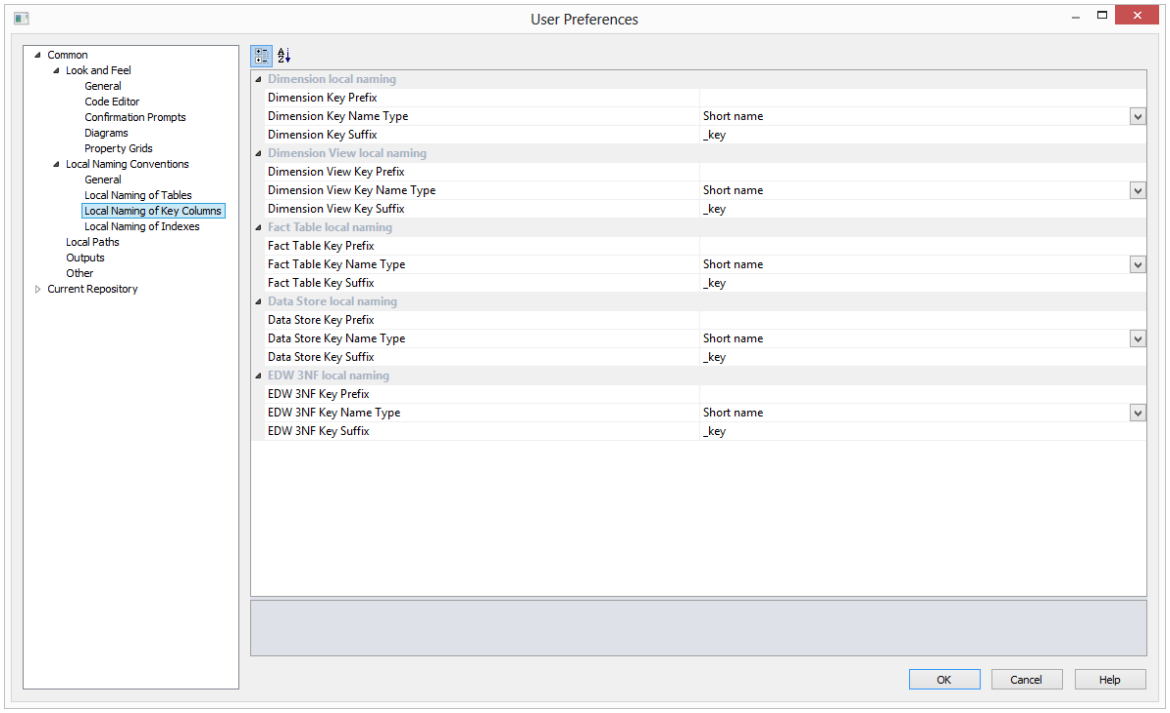
This option allows you to set the **Local Naming of Tables**.



Define the **prefix** and **suffix** that will be used in the default naming convention for each table type.

Local Naming of Key Columns

This option allows you to set the **Local Naming of Key Columns**.



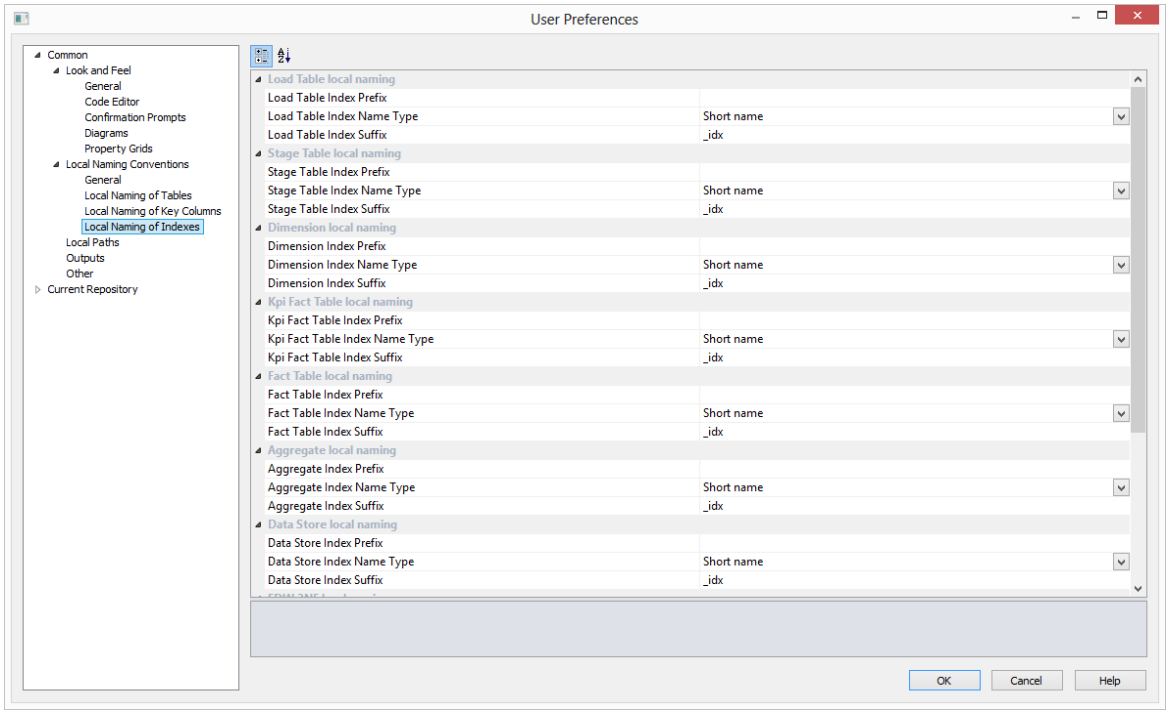
The **Key Prefix** option sets the prefix that will be used in the default key naming convention.

The **Key Name Type** option sets the basis for the key naming.

The **Key Suffix** option sets the suffix that will be used in the default naming convention.

Local Naming of Indexes

This option allows you to set the **Local Naming of Indexes**.



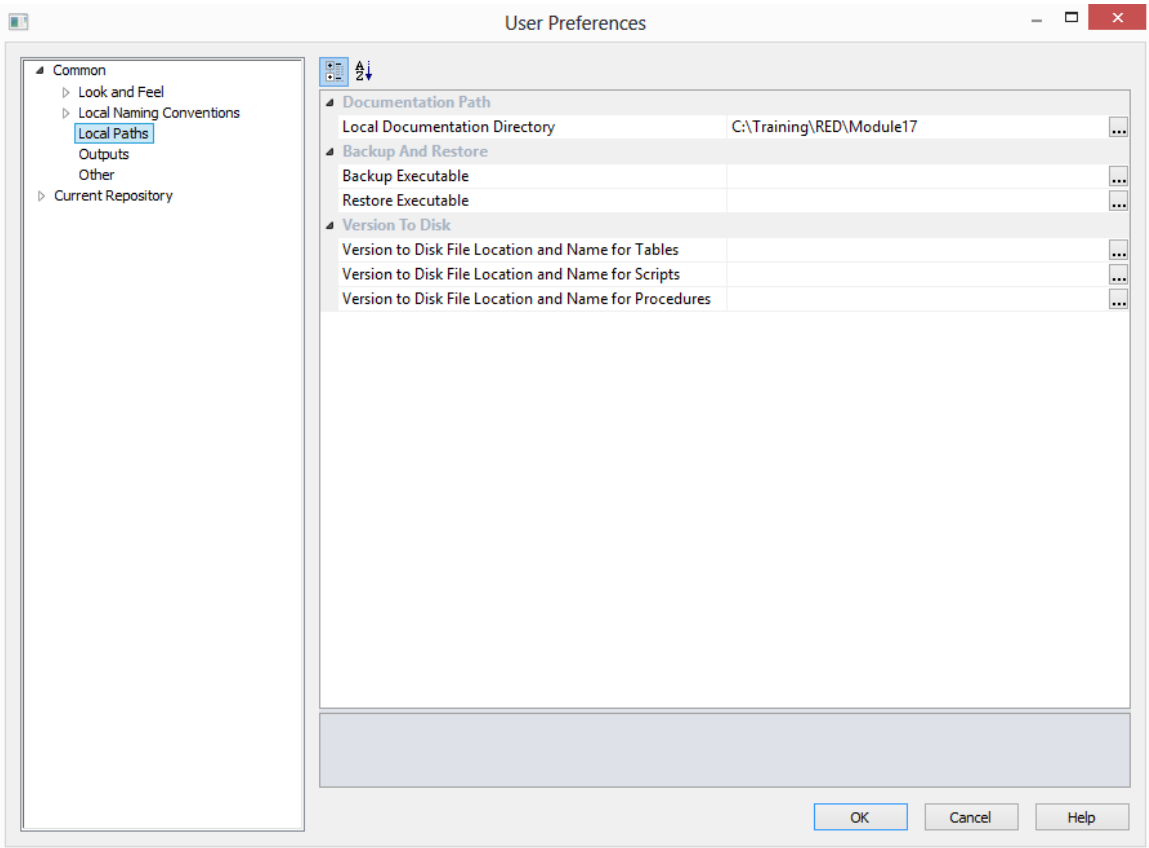
The **Index Prefix** option allows you to set the prefix that will be used in the default index naming convention.

The **Index Name Type** allows you to set the basis for the index naming.

The **Index Suffix** allows you to set the suffix that will be used in the default index naming convention.

LOCAL PATHS

This option allows you to set the **Local Paths** for documentation, backup and restore and for versioning to disk.



Documentation Path

Set the **Local Documentation Directory**.

Backup And Restore

The **Backup Executable** option sets the override for backup executable. By default, WhereScape RED tries to find the path of the backup executable. This is bcp.exe for SQL Server, exp.exe for Oracle and db2cmd.exe for IBM DB2. This edit box provides the ability to specify the exact location and name of the executable. This is useful when WhereScape RED cannot find the program or if there are multiple versions of the program on the PC.

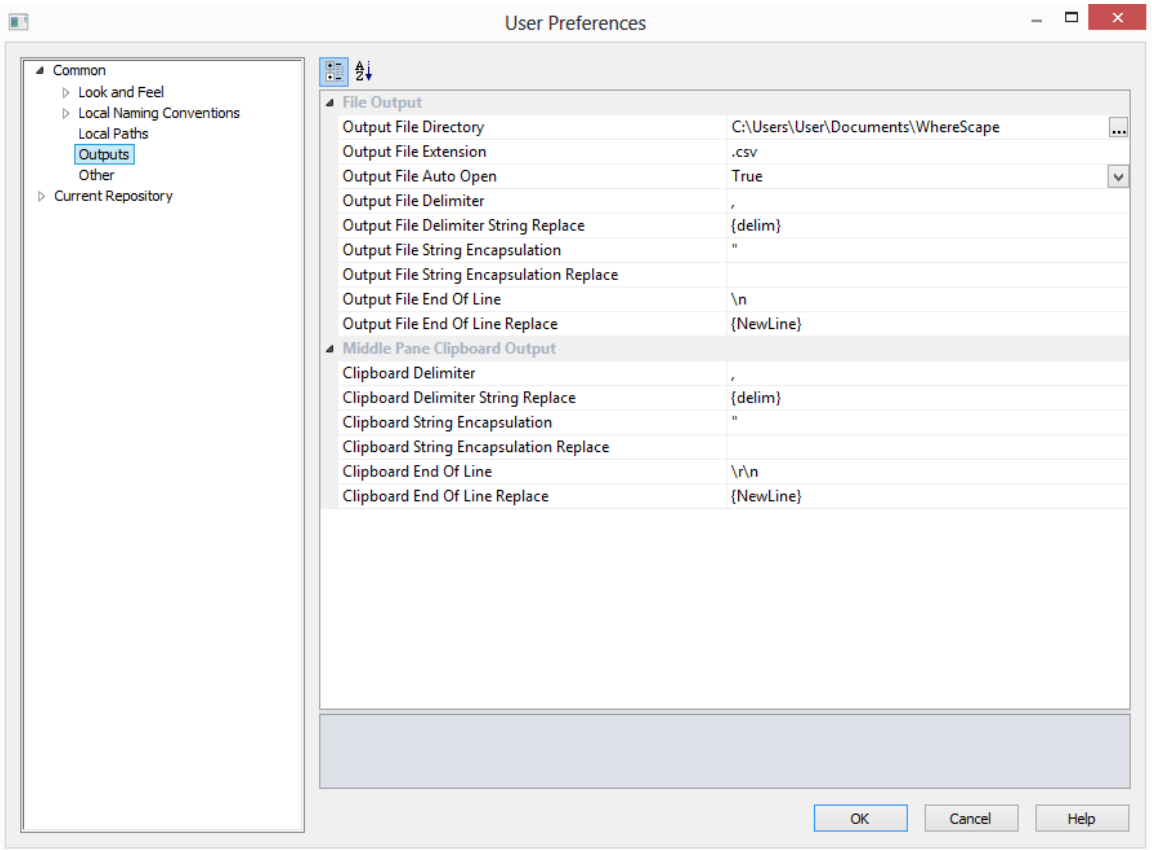
The **Restore Executable** option sets the override for restore executable. By default, WhereScape RED tries to find the path of the restore executable. This is bcp.exe for SQL Server, imp.exe for Oracle and db2cmd.exe for IBM DB2. This edit box provides the ability to specify the exact location and name of the executable. This is useful when WhereScape RED cannot find the program or if there are multiple versions of the program on the PC.

Version to Disk

Set the locations and names for **Versions to Disk**. If any of the three versions to disk paths are set, WhereScape RED will automatically create ascii files containing the applicable ddl or code each time an automated version occurs in the entered directory.

OUTPUTS

This option allows you to set the **Output** user preferences.



File Output

The **Output File Directory** option allows you to set the path for output files created from the middle pane.

The **Output File Extension** option allows you to set the file extension for output files created from the middle pane. This value determines the program that will auto open files.

The **Output File Auto Open** option, when set to True, results in files created from the middle pane being opened automatically.

The **Output File Delimiter** option allows you to set the characters that separate each field within each record of output files created from the middle pane. Common values are , and |.

The **Output File Delimiter String Replace** option allows you to set the characters that will replace the delimiter character if it occurs inside a field.

The **Output File String Encapsulation** option allows you to set the characters that are used to enclose string values of files created from the middle pane. Common values are " and '.

The **Output File String Encapsulation Replace** option allows you to set the characters that will replace the encapsulation string if it occurs inside a field.

The **Output File End Of Line** option allows you to set the characters saved at the end of each record of files created from the middle pane. Common values are \n, \r and \t.

The **Output File End Of Line Replace** option allows you to set the characters that will replace the end of line string if it occurs inside a field.

Middle Pane Clipboard Output

The **Clipboard Delimiter** option allows you to set the characters that separate each field within each record of clipboard output created from the middle pane. Common values are , and |.

The **Clipboard Delimiter String Replace** option allows you to set the characters that will replace the delimiter character if it occurs inside a field.

The **Clipboard String Encapsulation** option allows you to set the characters that are used to enclose string values of clipboard output created from the middle pane. Common values are " and '.

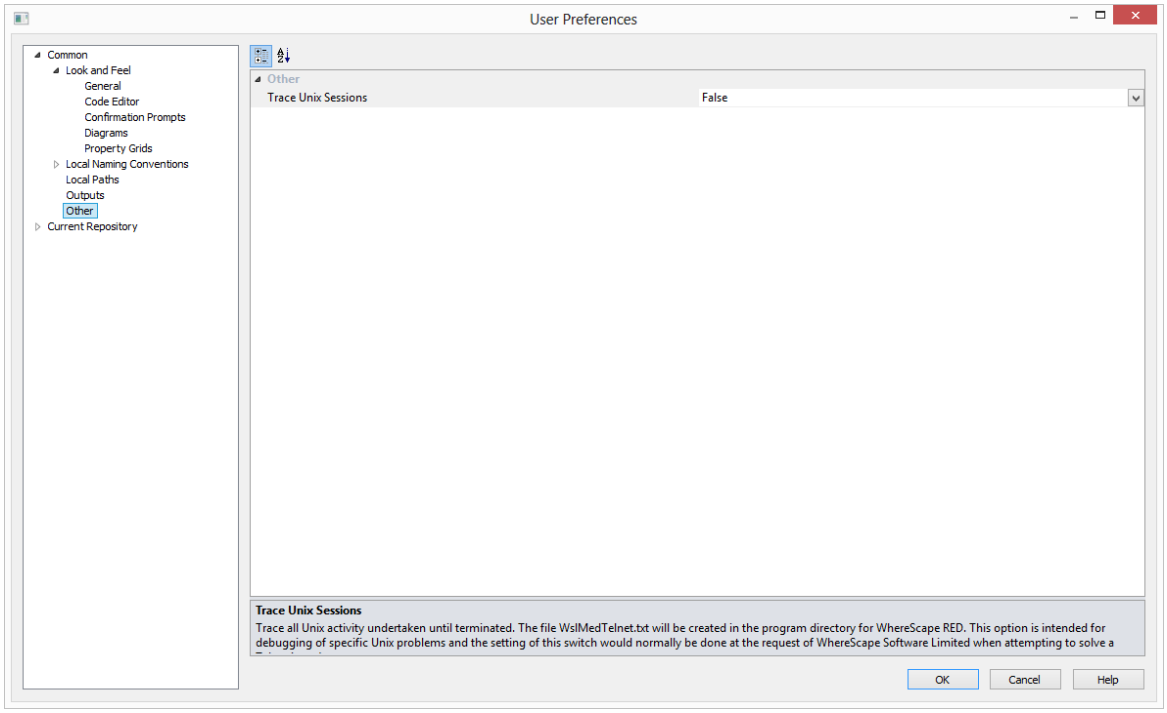
The **Clipboard String Encapsulation Replace** option allows you to set the characters that will replace the encapsulation string if it occurs inside a field.

The **Clipboard End of Line** option allows you to set the characters saved at the end of each record of clipboard output created from the middle pane. Common values are \n, \r and \t.

The **Clipboard End of Line Replace** option allows you to set the characters that will replace the end of line string if it occurs inside a field.

OTHER

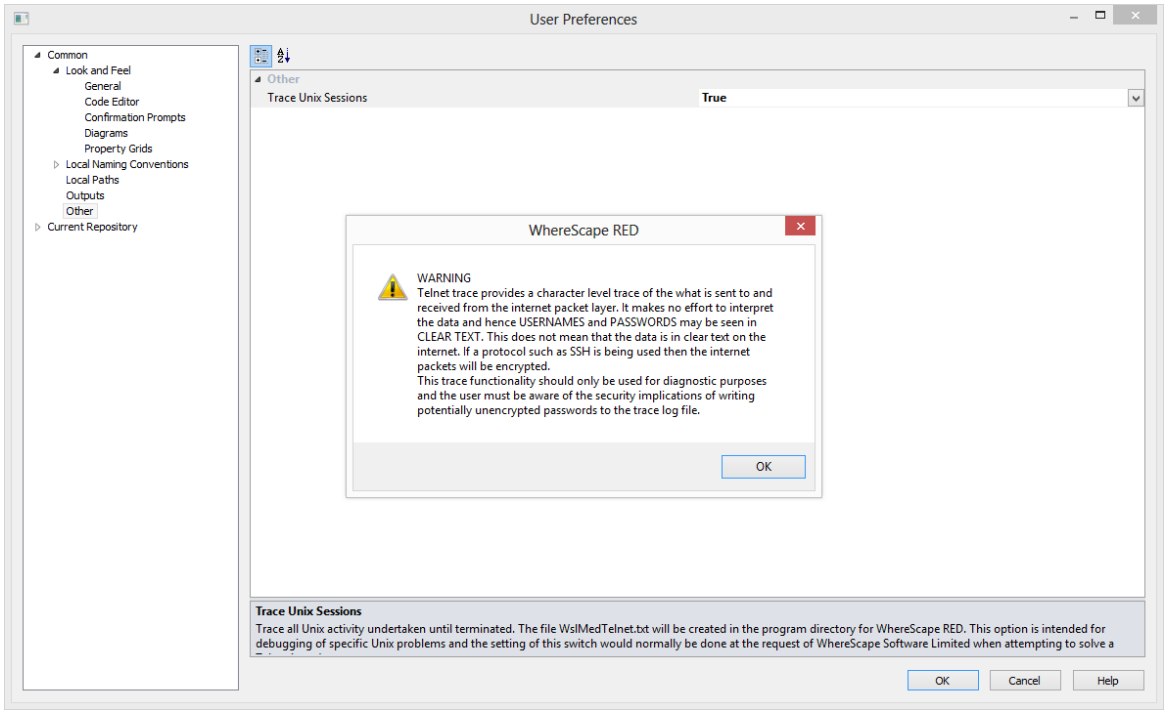
This option allows you to set the **Other** user preferences.



Other

When the **Trace Unix Sessions** option is set, this will trace all Unix activity undertaken by the WhereScape RED program until it is terminated. The file WslMedTelnet.txt will be created in the program directory for WhereScape RED. This option is intended for debugging of specific Unix problems and the setting of this switch would normally be done at the request of WhereScape when attempting to solve a Telnet issue. This setting is only relevant for the PC on which the setting is made. (i.e. it is not a global setting for the repository).

Warning:
When set to true, the following warning will appear: "Telnet trace provides a character level trace of the what is sent to and received from the internet packet layer. It makes no effort to interpret the data and hence USERNAMES and PASSWORDS may be seen in CLEAR TEXT. This does not mean that the data is in clear text on the internet. If a protocol such as SSH is being used, then the internet packets will be encrypted. This trace functionality should only be used for diagnostic purposes and the user must be aware of the security implications of writing potentially unencrypted passwords to the trace log file."

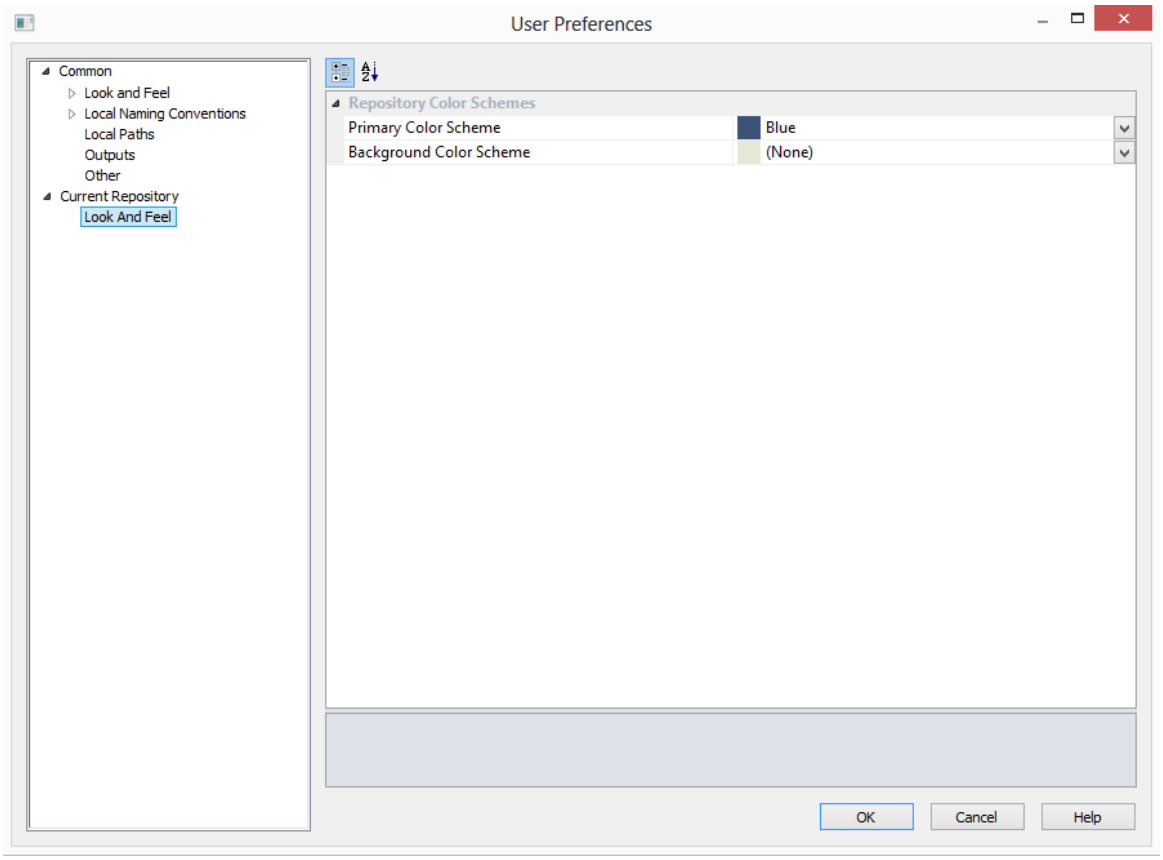

User Preferences
Common
Look and Feel
General
Code Editor
Confirmation Prompts
Diagrams
Property Grids
Local Naming Conventions
Local Paths
Outputs
Other
Current Repository
Other
Trace Unix Sessions
True
WhereScape RED
WARNING
Telnet trace provides a character level trace of the what is sent to and received from the internet packet layer. It makes no effort to interpret the data and hence USERNAMES and PASSWORDS may be seen in CLEAR TEXT. This does not mean that the data is in clear text on the internet. If a protocol such as SSH is being used then the internet packets will be encrypted.
This trace functionality should only be used for diagnostic purposes and the user must be aware of the security implications of writing potentially unencrypted passwords to the trace log file.
OK
Trace Unix Sessions
Trace all Unix activity undertaken until terminated. The file WslMedTelnet.txt will be created in the program directory for WhereScape RED. This option is intended for debugging of specific Unix problems and the setting of this switch would normally be done at the request of WhereScape Software Limited when attempting to solve a
OK
Cancel
Help

CURRENT REPOSITORY

The various options are described below.

LOOK AND FEEL

This allows you to set the **Look and Feel** for the **Current Repository**.

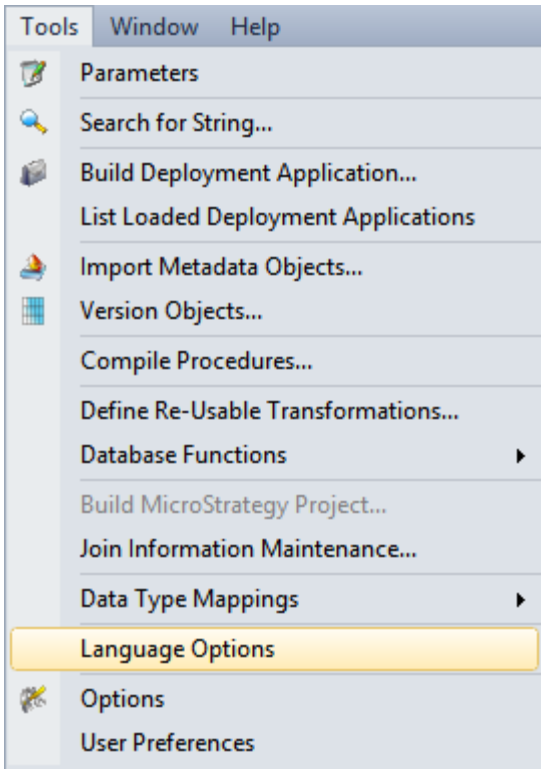


Repository Color Schemes

Set the primary and background **Color Schemes** for the current repository.

LANGUAGE OPTIONS

Select **Language Options** from the **Tools** menu.



Languages can be defined via the **Tools/Language Options** menu. This option is only available for SQL Server databases and applies only to dimension, fact and OLAP objects.

A blank entry means that no languages have been defined and thus no translations can be saved.

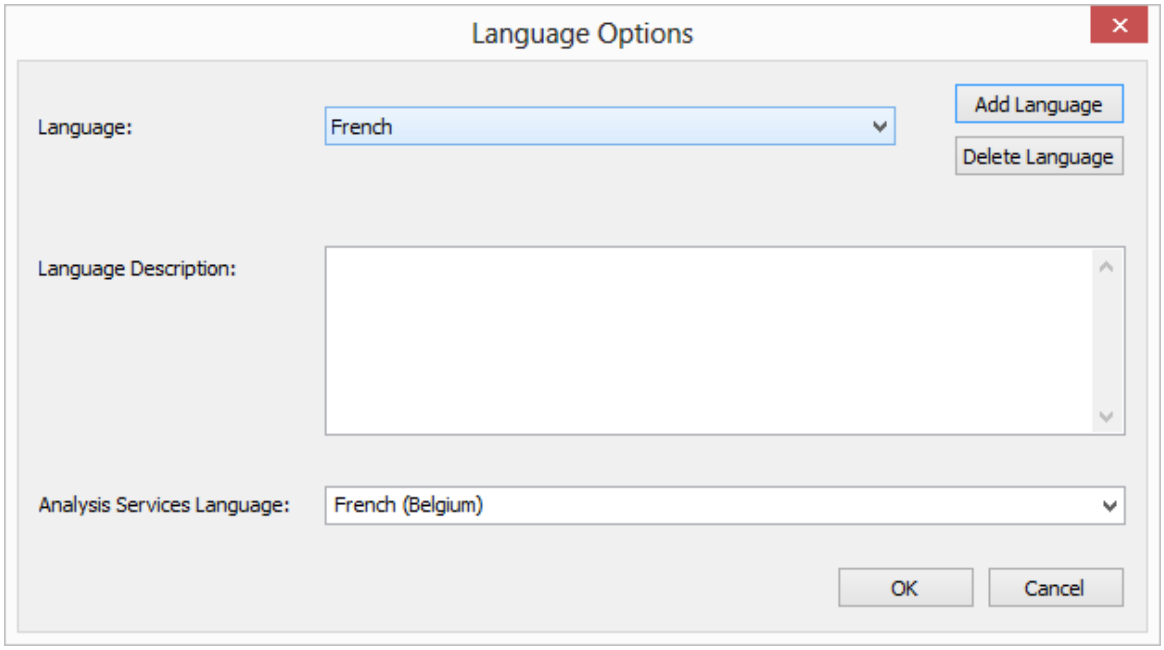
To add a language

Click the **Add Language** button; enter the new language ID and click **OK**.

To delete a language

First select the language from the drop-down list and then click the **Delete Language** button.

Note: All translations for the selected language will also be deleted.



Language

The language Reference/ID.

Language Description

The language Description.

Analysis Services Language

Used to identify the language ID as used in Analysis Services.

WhereScape®

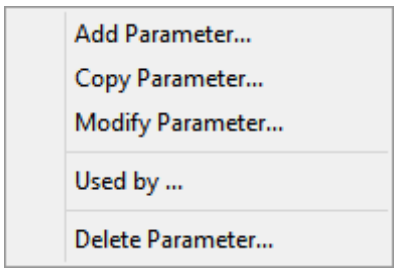
CHAPTER 5
PARAMETERS

Parameters are a means of passing information between two or more procedures and between the WhereScape RED environment and procedures.
They can be edited within the WhereScape RED environment by selecting the **Tools/Parameters** menu option. A list of parameters is displayed as per the example below:

Parameter listing

Parameter	Value	Comments
(x) BUSINESS_DAYS_IN_MONTH	22	updated via daily_time_roll
(x) BUSINESS_DAYS_MTD	9	updated via daily_time_roll
(x) CURRENT_CAL_MONTH	201608	updated via daily_time_roll
(x) CURRENT_CAL_MONTH_NO	8	updated via daily_time_roll
(x) CURRENT_CAL_YEAR	2016	updated via daily_time_roll
(x) CURRENT_DATE	09-AUG-16	updated via daily_time_roll
(x) CURRENT_FIN_MONTH	201608	updated via daily_time_roll
(x) CURRENT_FIN_MONTH_NO	8	updated via daily_time_roll
(x) CURRENT_FIN_YEAR	2016	updated via daily_time_roll
(x) PREVIOUS_CAL_MONTH	201607	updated via daily_time_roll
(x) PREVIOUS_CAL_MONTH_NO	7	updated via daily_time_roll
(x) PREVIOUS_CAL_YEAR	2015	updated via daily_time_roll
(x) PREVIOUS_FIN_MONTH	201607	updated via daily_time_roll
(x) PREVIOUS_FIN_MONTH_NO	7	updated via daily_time_roll
(x) PREVIOUS_FIN_YEAR	2015	updated via daily_time_roll
(x) load_budget_0	18	Rowcount from native load at 08/08/16 21:49:24
(x) load_customer_0	6	Rowcount from native load at 09/08/16 14:19:12
(x) load_forecast_0	18	Rowcount from native load at 08/08/16 22:02:50
(x) load_order_header_0	9	Rowcount from native load at 09/08/16 14:15:51
(x) load_order_line_0	21	Rowcount from native load at 08/08/16 22:03:13

A parameter can be added, edited, copied or deleted using the right-click menu in the Parameter column:



Typical parameter usage may be the global definition of how many days should be looked back for changed data, a month or processing period etc.
Parameters can be used in **load tables** to place limits in a 'Where' clause, etc. See ***Database Link Load - Source Mapping*** (on page 240) for more information.
They are also used by **stage table procedures** as variables. See ***Generating the Staging Update Procedure*** (on page 372) for more information.

IBM DB2: Two procedures are provided to allow procedures to read and write parameters. These procedures are ***WsParameterRead*** (see "***WsParameterReadF***" on page 1060) and ***WsParameterWrite*** (on page 1068). Using these procedures, a procedure can load and use the contents of a parameter, or modify an existing parameter, or add a new parameter.

Oracle: Two functions are provided to allow procedures to read and write parameters. These functions are ***WsParameterRead*** (see "***WsParameterReadF***" on page 1060) and ***WsParameterWrite*** (on page 1068). Using these functions, a procedure can load and use the contents of a parameter, or modify an existing parameter, or add a new parameter.

SQL Server: Two procedures are provided to allow procedures to read and write parameters. These procedures are ***WsParameterRead*** (see "***WsParameterReadF***" on page 1060) and ***WsParameterWrite*** (on page 1068). Using these procedures, a procedure can load and use the contents of a parameter, or modify an existing parameter, or add a new parameter. A function ***WsParameterReadF*** (on page 1060) is also provided. This function will return the value of a parameter.

Global Parameters

A number of global parameters are provided when loading tables either via WhereScape RED or via the scheduler. These global parameters are accessed by calling the function ***WsParameterReadG*** (on page 1063).
The load table name and the source table name can be acquired by using this function.

See ***Callable Routines*** (on page 955) for more information on ***WsParameterRead*** (see "***WsParameterReadF***" on page 1060), ***WsParameterWrite*** (on page 1068), ***WsParameterReadF*** (on page 1060) and ***WsParameterReadG*** (on page 1063).

WhereScape®

CHAPTER 6
CONNECTIONS

Connection objects serve several purposes in WhereScape RED:

1. They are used to browse potential source data in source systems and to acquire metadata. Potential source data includes database tables and flat files.
For **database tables**, WhereScape RED:

- Uses the **ODBC Source** set on each connection to browse the source system.
- Acquires the metadata for new **load tables** built from the source system using drag and drop.

For **files**, WhereScape RED:

- Connects directly to Windows, UNIX/Linux and Hadoop to analyze the source file for the new load table and to acquire its metadata.
- Prompts for user input for any metadata not available in the source file.

NOTE1: ODBC connections must be either **User DSN** or **System DSN**. **File DSN** connections are **not** supported.

NOTE2: *Windows* and *UNIX* connections do not have an **ODBC Source** property. *UNIX* connections are used for UNIX and Linux systems.

---

2. Load tables with a connection of **Connection type** *ODBC* extract data from source systems using ODBC. The **ODBC Source** of the connection is the ODBC DSN used for the extract.

3. Each data warehouse metadata repository must have a Data Warehouse connection to use drag and drop to create new objects (other than load tables) in the data warehouse. WhereScape RED:

- Uses the **ODBC Source** set on the Data Warehouse connection to browse the Data Warehouse database.
- Acquires the metadata for any **tables** built from existing data warehouse tables.
- This connection always has a **Connection type** of *Database*.

4. Cube objects require a connection to define the Analysis Services server used to create and load cubes. This is a connection with a **Connection type** of *Microsoft Analysis Server 2005+*.

5. Export objects require a connection to define the target environment where exported data is written. This is a connection with a **Connection type** of *UNIX* or *Windows*.

IN THIS CHAPTER

CONNECTION TYPES

Connection properties are described in this topic as they apply to the supported database types in WhereScape RED.
Connection types can be set up via the following methods:

- Connection to the Data Warehouse/Metadata repository - Database type connections
- Connections to another Database - Database type connections on the same server or through a linked server
- ODBC Based Connections - ODBC type connections with Native ODBC, ODBC, Externally loaded or Integration Services load types
- Connections to Windows
- Connections to UNIX/Linux
- Connections to Hadoop
- Connections to Microsoft Analysis Services

One or more of the following icons are used to indicate where properties are unique to a particular database type:

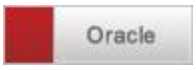
NOTE: Custom type database connections are assigned a unique name during the licensing process. In this document the term 'Custom' refers to all custom database connections.

DATABASE - DATA WAREHOUSE/METADATA REPOSITORY

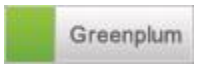
This topic describes the details of the connection Properties as they apply to **Database** type connections and specifically of the Data Warehouse or Metadata repository connection.
The Data Warehouse connection is the connection that stores the metadata repository and it is the connection that is used in the drag and drop functionality to create the dimension, stage, fact and aggregate tables.
This connection is also used to create cubes.

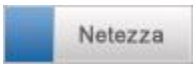
Connection types also impact the available load methods.
For detailed instructions on **how to create a connection** see WhereScape RED's Tutorial 1.

NOTE: The **DataWarehouse connection** must exist if you wish to use drag and drop to create dimensions, stage tables, fact tables, aggregates and cubes.

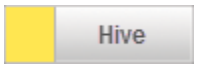


SQL Server, Oracle and **DB2 Data Warehouse** - the connection for the Metadata repository.

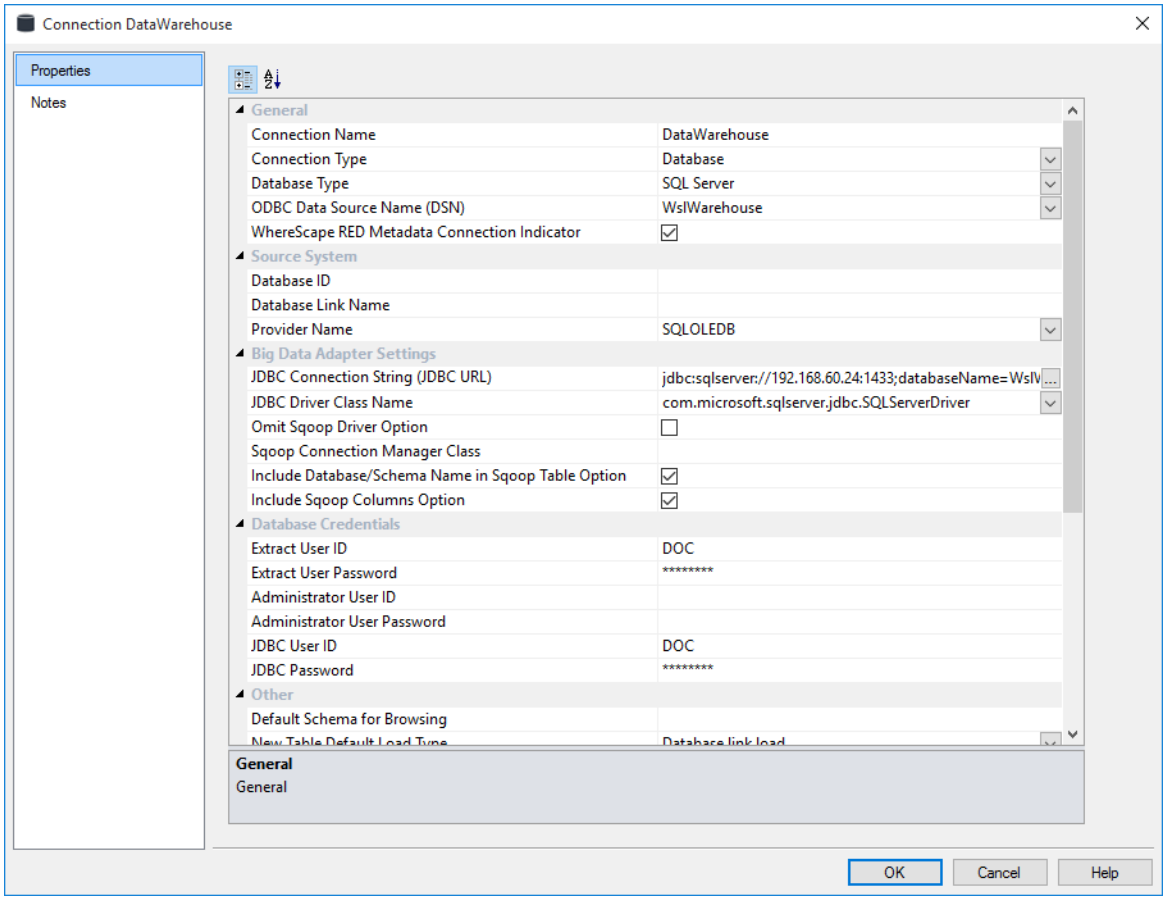




Greenplum, Netezza and PDW Data Warehouse - the connection for the Metadata repository which is always stored in **SQL Server**.
Apart from the Data Warehouse or Repository connection to SQL Server, there will need to be at least one other connection to the Target Data warehouse database such as Greenplum, Netezza or PDW.



Hive can have its metadata repositories in Oracle, SQL Server and Teradata datawarehouses.



General

Connection Name

Name used to label the connection within WhereScape RED. Typically for SQL Server, Oracle and DB2 this is **DataWarehouse.**
For target databases like Greenplum, Netezza or PDW the DataWarehouse connection can be renamed to **Repository.**

Connection Type

Indicates the connection source type or the connection method such as Database, ODBC, Windows, Unix. Select the **Database** connection type.

Database Type

Type of database such as DB2, Greenplum, Hive, Netezza, Oracle, SQL Server, Teradata. Default is **(local)**.
If the database or metadata repository is located on the same server as the RED instance, leave the **(local)** default option selected.
If the database or metadata repository is located on a different server select the relevant database type of SQL Server, Oracle or DB2.

ODBC Data Source Name (DSN)

ODBC Data Source Name (DSN) as defined in the Windows 32-bit **ODBC Data Source Administrator**.
Select the relevant ODBC connection to the datawarehouse database or metadata repository.

Note: The ODBC Source Name defined in RED must be the same on all machines that use the corresponding connection.

Data Warehouse Connection Indicator

Distinguishes the special connection that identifies the WhereScape RED metadata repository. This option must be enabled for **DataWarehouse** type connections.

Note: There should only be one metadata connection in a WhereScape RED repository.

Source System

Database ID

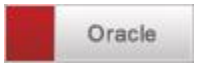
Database Identifier (e.g. Oracle SID or TNS Name, Teradata TDPID) or Database Name (e.g. as in DB2 or SQL Server).

Database Link Name

Optional name of a Database Link that is used to access the database. Only required field for Database type connections where the database is not on the same server as the data warehouse. If the server is on the same database, the link doesn't need to be defined and the field can be left blank.

For connections to databases located in different servers where a database link load is required, if the database link already exists to this source database then enter that link name in this field. If the link doesn't exist, then enter a link name.

Enter the defined SQL LINK name that points to the correct server, the link name does not need to be the same name as the server name.



Under some circumstances Oracle insists that this link name is the same as the tnsnames entry for the database, so it may be good practice to use the source SID as the database link name. If this is a new link, then see the notes later in this section on ***database link creation*** (see "***Creating a Database Link***" on page 196).

Provider Name

Name of the connection provider/driver used to connect to the database.

Full Database Path Name

This field is used when you enter **Microsoft.Jet.OLEDB.4.0** as the Provider Name, ie when the source database is from MS Access or Excel. Enter the full database path name.

Big Data Adapter Settings

JDBC Connection String (JDBC URL)

Connection string used by the WhereScape Big Data Adapter to access this database.

JDBC Driver Class Name

JDBC driver class to be used by the WhereScape Big Data Adapter. This field must be set if the JDBC URL is set.
Select the appropriate JDBC Driver class name from the drop-down list. If this is left empty this will not be specified in generated commands.

Omit Sqoop Driver Option

If set, the --driver option to Sqoop will be omitted. This is required for certain connection types such as Oracle connections.
If you select the **Omit Sqoop Driver Option** check-box, the driver parameter will not be used in sqoop command line. This is a requirement for Oracle at the moment, as suggested by Sqoop documentation for 1.4.5.

Sqoop Connection Manager Class

Custom Sqoop connection manager class. Corresponds to the --connection-manager command line argument. Leave blank of this is not required.

Include Database/Schema Name in Sqoop Table Option

If set, the --table option to Sqoop will include the database/schema name of the destination table when performing Apache Sqoop loads into this connection. This is incompatible with some connection managers, for example the Cloudera Connector Powered by Teradata. If this is not set, users must ensure that the database/schema is otherwise communicated to Sqoop, for example by using the $OBJECT_DATABASE$ token in the in the JDBC Connection String.

Include Sqoop Columns Option

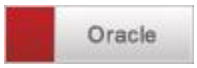
If set, the --columns option to Sqoop will be included when performing Apache Sqoop loads into this connection. This is incompatible with some connection managers, for example the Cloudera Connector Powered by Teradata. If this is not set, users must ensure that the order of columns in the loads match the order in the metadata.

Database Credentials

Extract User ID

Database User that has access to SELECT from the source system tables to extract data.

For SQL Server, this field can be left blank if using a trusted login, or the server login password.



For more information see ***remote view extract*** (see "***Remote View Extract - Oracle Databases Only***" on page 245).

Extract User Password

The password of the data warehouse user.

For SQL Server, this field can be left blank if using a trusted login, or the server login password.

Administrator User ID

Left blank.

Administrator User Password

Left blank.

JDBC User ID

User ID to login as using JDBC via WhereScape Big Data Adapter (optional).

JDBC Password

Password to login with when using JDBC via WhereScapeBig Data Adapter (optional).

Other

Default Schema for Browsing

Optional comma-delimited list of schemas for the browser pane filter. Enter the schema(s) you want the connection to browse by default on the right browser pane.

New Table Default Load Type

The default **Load Type** for new Load tables created using this connection. Selected the desired default load type from **Database Link Load, Integration Services Load** or **Externally Loaded.**

SSIS Connection String (OLEDB)

Connection string to be used by Microsoft SQL Server Integration Services (SSIS) to connect to the data source or destination. For details on how to use the wizard to build an SSIS Connection String see ***Loading Data into RED Load Tables using SSIS***.

Note: A connection string is typically composed of multiple property name/value pairs that are semi-colon delimited.

Data Type Mapping Set

Mapping Set to use when converting from a source database data type to a destination database data type. Setting this field to **(Default)** makes RED automatically select the relevant mapping set, otherwise you can choose one of the standard mapping sets from the drop-down list or create a new set.

Default Transform Function Set

Function Set that is selected by default in the Transformation dialogs.

When Connection is an OLAP Data Source

This section of fields is only relevant and will only be visible from a **Datawarehouse** connection (where the Datawarehouse field is enabled). These fields are required so that the data warehouse can be used as a source for the Analysis Services cubes.

MSAS Connection String

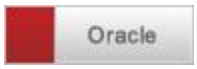
Connection string to be used by Microsoft Analysis Services (MSAS) to connect to the data warehouse. For details on how to use the wizard to build the OLAP connection string, see ***Define Data Source for OLAP Cube*** (see "***OLAP Defining the Data Source for the OLAP Cube***" on page 558).

Note: A connection string is typically composed of multiple property name/value pairs that are semi-colon delimited.

Connection Provider/Driver

Name of the Connection Provider/Driver to use to connect to the data warehouse database when it is used as the data source for OLAP cubes.

set to **SQLOLEDB**.



set to **MSDAORA**.

set to **IBMDADB2**.

Data Warehouse Server

Data Warehouse Server Name, which is used when the data warehouse is used as the data source for OLAP cubes.

Data Warehouse Database ID

Data Warehouse Database Identifier (e.g. Oracle SID or TNS Name, Teradata TDPID) or Database Name (e.g. as in DB2 or SQL Server), which is used when the data warehouse is used as the data source for OLAP cubes.

Target Table Location [For target enabled licenses]

Add new Target Location

This option allows adding new database/schema locations for objects in this connection.
For Greenplum, Netezza and PDW schema target locations setup, see ***Database source system - Local/Linked Servers*** or ***ODBC*** connection topics, depending on the connection type chosen for the database source.

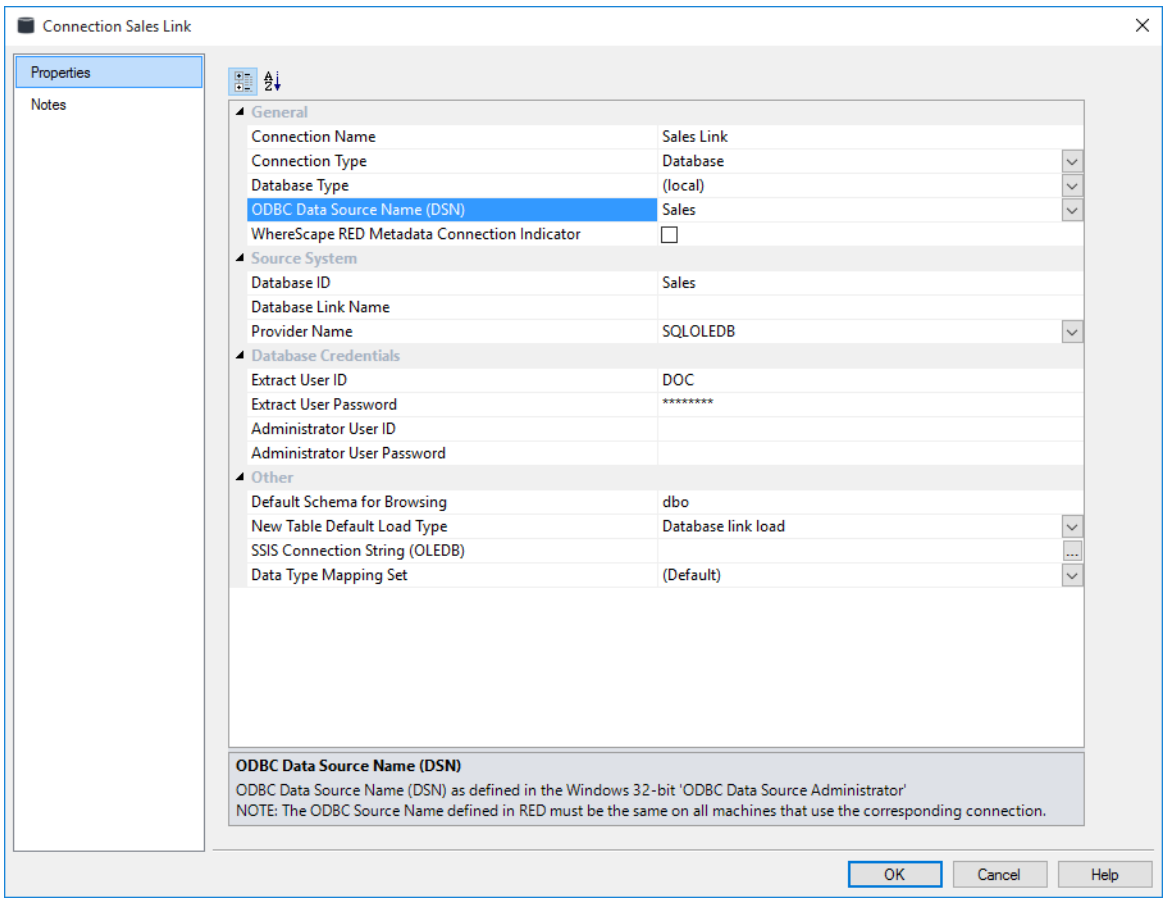
1. Click the **Add** button to add the required target schemas for this connection.
2. Give the new target a name and then enter the target's schema.
3. Default schema location(s) for **New Tables** can also be set from the **Tools/Options** menu – See ***Settings - Storage - Target Location***.
4. For more details on setting specific SQL Server and Oracle on a table by table level see ***Table Storage Properties*** for SQL Server and Oracle specific examples.

Once the connection has been set up, you can right-click on the connection in the middle pane or double click on the connection name from the left pane to view or edit the connection's Properties.

DATABASE

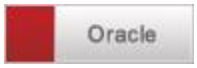
This topic describes the details of the connection Properties as they apply to the Database type connections. For detailed instructions on **how to create a connection** see WhereScape RED's Tutorial 1.
When coming to the Data Warehouse from an OLTP source system, the connection defines the path to get to those source tables. A connection object must be defined for each source system.

Fields presented in the Connection properties dialog (pictured below) depend on the connection and database types selected. See below for a description of all possible fields.



SQL Server: when running a SQL Server data warehouse, it is possible to create database links (linked server) to other SQL Server instances and most of the other major databases such as Oracle, DB2.

- Database link load - from a table in the current SQL Server database.
- Database link load - from a table in another database on the same SQL Server instance - the OBDC Data Source Name must be defined in the Windows 32-bit **ODBC Data Source Administrator** and selected on the ODBC DSN field as well as the Database ID that also needs to be populated in the Database ID field in the connection properties.
- Database link load - through a linked server.
- Integration Services load.



Oracle: using the Oracle database it is possible to create a database link to another Oracle database as standard, or to any other database by using Oracles Heterogeneous Services. See Oracle Examples.

- Database link load - from a table in the current Oracle database server (includes Oracle gateways and Oracle heterogeneous services).
- Database link load - through a database link to another Oracle server.

DB2: using the DB2 database it is possible to create a database link load to another table in the same DB2 system or to any another system via CLI. See DB2 Examples.

- Database link load - from another table in the same DB2 system (includes federation).
- Database link load - via CLI from a DB2 table in another system.

Find examples for each of RED's database type connection screens for SQL Server, Oracle, DB2 Greenplum, Netezza and PDW at the bottom of the connection properties description.

A Database Link Load through a **linked server** can have a number of source databases:

- SQL Server
- Oracle
- DB2 for Linux, UNIX and Windows (the DB2 RED can build a warehouse in)
- Teradata
- Greenplum
- Netezza
- PDW
- Hive
- Custom
- Sybase
- DB2/400 (the DB2 that runs on AS/400 machines - quite different to above)
- MS Access
- Excel

The connection object Properties window has the following fields:

General

Connection Name

Name used to label the connection within WhereScape RED.

Connection Type

Indicates the connection source type or the connection method such as Database, ODBC, Windows, Unix. Set to **Database**.

Database Type

Type of database such as DB2, Greenplum, Hive, Netezza, Oracle, SQL Server, Teradata. Default is **(local)**.

ODBC Data Source Name (DSN)

ODBC Data Source Name (DSN) as defined in the Windows 32-bit **ODBC Data Source Administrator**.

Note: The ODBC Source Name defined in RED must be the same on all machines that use the corresponding connection.

Data Warehouse Connection Indicator

Only required for the DataWarehouse connection. Leave this field disabled.

Source System

Provider Name

Name of the connection provider/driver used to connect to the database.

Database ID

Database Identifier (e.g. Oracle SID or TNS Name, Teradata TDPID) or Database Name (e.g. as in DB2 or SQL Server).

Database Host/Server

PDW Database type connections only: Used for sqlcmd specification.

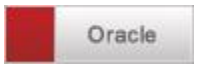
Database Port

PDW Database type connections only: Used for sqlcmd specification.

Database Link Name

Optional name of a Database Link that is used to access the database.

If SQL Server and the database are on the same server as the data warehouse, then no link needs to be defined and this field can be left blank.



If a database link already exists to this source database, then enter that link name in this field. If no link exists then enter a link name. Under some circumstances, Oracle insists that this link name is the same as the tnsnames entry for the database, so it may be good practice to use the source SID as the database link name. If this is a new link then see the notes later in this section on ***database link creation*** (see "***Creating a Database Link***" on page 196).
Enter the defined SQL LINK name that points to the correct server, the link name does not need to be the same name as the server name.

Database Credentials

Extract User ID

Database User that has access to SELECT from the source system tables to extract data. For more information see ***remote view extract*** (see "***Remote View Extract - Oracle Databases Only***" on page 245).

Extract User Password

The password of the data warehouse user. For SQL Server, blank if a trusted login, or the server login password.

Teradata Wallet User ID/Teradata Wallet String

If connecting to a **Teradata** server, the Teradata Wallet log on method can be selected from the ODBC User Default drop-down menu and the Teradata Waller User ID and String can be entered instead of User/Password.

Administrator User ID

Left blank.

Administrator User Password

Left blank.

Other

Default Schema for Browsing

Optional comma-delimited list of schemas for the browser pane filter. Enter the schema(s) you want the connection to browse by default on the right browser pane.

New Table Default Load Type

The default **Load Type** for new Load tables created using this connection. Select between **Database link load**, **Integration Services Load** or **Externally Loaded**.

SSIS Connection String (OLEDB)

Connection string to be used by Microsoft SQL Server Integration Services (SSIS) to connect to the data source or destination. For details on how to use the wizard to build an SSIS Connection String see ***Loading Data into RED Load Tables using SSIS***.

Note: A connection string is typically composed of multiple property name/value pairs that are semi-colon delimited.

SSIS Connection String (SQLPDW)

PDW Database type connections only: Connection string to be used by Microsoft SQL Integration Services (SSIS) to connect to the data source or destination using the SQL Server Parallel Data Warehouse Connection Manager.
Note: A connection string is typically composed of multiple property name/value pairs that are semi-colon delimited.
The **SSIS Connection String (SQLPDW)** is used when the target is PDW and it must be set to process PDW loads.

Staging database

PDW Database type connections only: The staging database to be used by Microsoft SQL Server Integration Services (SSIS) for loading to PDW, if this option is enabled in the ***Load Table Source screen***'s Properties.

Data Type Mapping Set

Mapping Set to use when converting from a source database data type to a destination database data type. Setting this field to **(Default)** makes RED automatically select the relevant mapping set, otherwise you can choose one of the standard mapping sets from the drop-down list or create a new set.

Target Table Location [For target enabled licenses]

Add new Target Location

This option allows adding new database/schema locations for objects in this connection.

1.Click the **Add** button to add the required target schemas for this connection.
2. Give the new target a name and then enter the target's database name and/or schema.
3. Default schema location(s) for new **Load Tables** can also be set from the **Tools/Options** menu – See ***Settings - Storage - Target Location***.

For more details on setting specific Greenplum, Netezza and PDW schema locations on a table by table level see ***Table Storage Properties***.

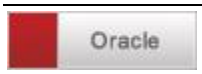
NOTE: The database and schema names for **Custom** database connections are used as follows:
`<database>.<schema>.objectname`

Leave database name blank if not required. Leave schema name blank to use the default schema.

Once the connection has been set up, you can right-click on the connection in the middle pane or double click on the connection name from the left pane to view or edit the connection's Properties.

ODBC

This connection type is via an ODBC link. Most data movement is performed using the ODBC connection.



Oracle can perform **ODBC loads** using **Direct Path OCI** and **standard (single row inserts)** via the scheduler.

ODBC loads can have a number of source databases:

- SQL Server
- Oracle
- DB2 for Linux, UNIX and Windows (the DB2 RED can build a warehouse in)
- Teradata
- Greenplum
- Netezza
- PDW
- Hive
- Sybase
- MySQL
- DB2/400 (the DB2 that runs on AS/400 machines - quite different to above)
- MS Access
- Excel

The connection object Properties window has the following fields:

General

Connection Name

Name used to label the connection within WhereScape RED.

Connection Type

Indicates the connection source type or the connection method. Set to **ODBC**.

Database Type

Type of database such as DB2, Greenplum, Hive, Netezza, Oracle, Sql Server, Teradata.

ODBC Data Source Name (DSN)

ODBC Data Source Name (DSN) as defined in the Windows 32-bit **ODBC Data Source Administrator**.

Note: The ODBC Source Name defined in RED must be the same on all machines that use the corresponding connection.

Data Warehouse Connection Indicator

Only required for the DataWarehouse connection. Leave this field disabled.

ODBC

Work Directory

Windows directory used by WhereScape RED to create temporary files for minimal logged extracts. The directory must exist and allow write access. There must be a different work directory for each WhereScape RED Scheduler running on the same machine to avoid file conflicts. Typically, **C:\Temp** or a **sub-directory of C:\Temp** is used.

Credentials

NOTE: If the source database does not support windows authentication, a username and password must be specified in the User ID and User Password fields below.

Extract User ID

Database User that has access to SELECT from the source system tables to extract data.

Extract User Password

Database Password to use with the Extract User ID to login to the source system to extract data.

Administrator User ID

Leave blank.

Administrator User Password

Leave blank.

Other

Default Schema for Browsing

Optional comma-delimited list of schemas for the browser pane filter. Enter the schema(s) you want the connection to browse by default on the right browser pane.

New Table Default Load Type

The default **Load Type** for new Load tables created using this connection. Select the desired default load type from **Native ODBC, Integration Services load** or **ODBC load**.

SSIS Connection String (OLEDB)

Connection string to be used by Microsoft SQL Server Integration Services (SSIS) to connect to the data source or destination. For details on how to use the wizard to build an SSIS Connection String see ***Loading Data into RED Load Tables using SSIS***.

Note: A connection string is typically composed of multiple property name/value pairs that are semi-colon delimited.

SSIS Connection String (SQLPDW)

PDW Database type connections only: Connection string to be used by Microsoft SQL Integration Services (SSIS) to connect to the data source or destination using the SQL Server Parallel Data Warehouse Connection Manager.
The **SSIS Connection String (SQLPDW)** is used when the target is PDW and it must be set to process PDW loads.

Note: A connection string is typically composed of multiple property name/value pairs that are semi-colon delimited.

SSIS Staging database

PDW Database type connections only: The staging database to be used by Microsoft SQL Server Integration Services (SSIS) for loading to PDW.

Data Type Mapping Set

Mapping Set to use when converting from a source database data type to a destination database data type. Setting this field to **(Default)** makes RED automatically select the relevant mapping set, otherwise you can choose one of the standard mapping sets from the drop-down list or create a new set.

Target Table Location [For target enabled licenses]

Greenplum Hive Netezza PDW

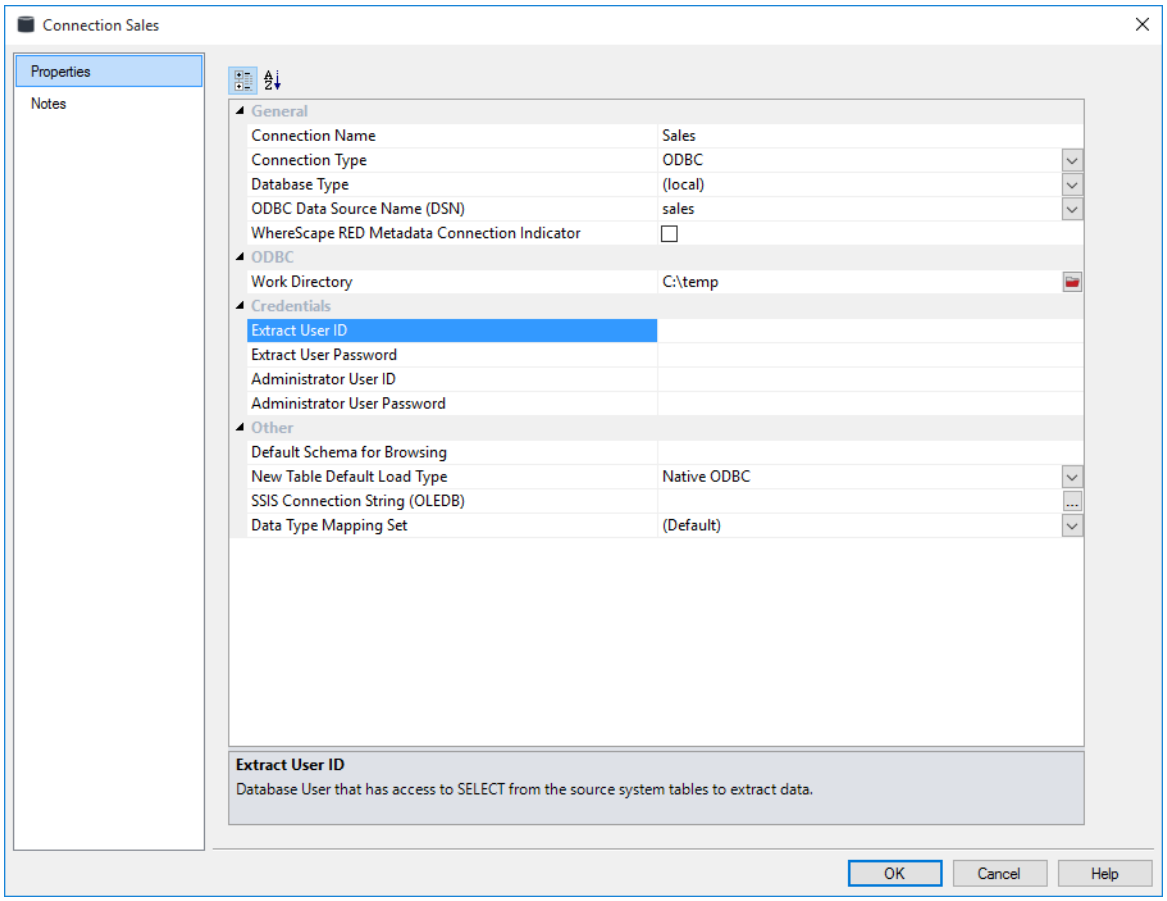
Add new Target Location
This option allows adding new database/schema locations for objects in this connection.

1.Click the **Add** button to add the required target schemas for this connection.
2. Give the new target a name and then enter the target's schema.
3. Default schema location(s) for new **Load Tables** can also be set from the **Tools/Options** menu – See ***Settings - Storage - Target Location***.

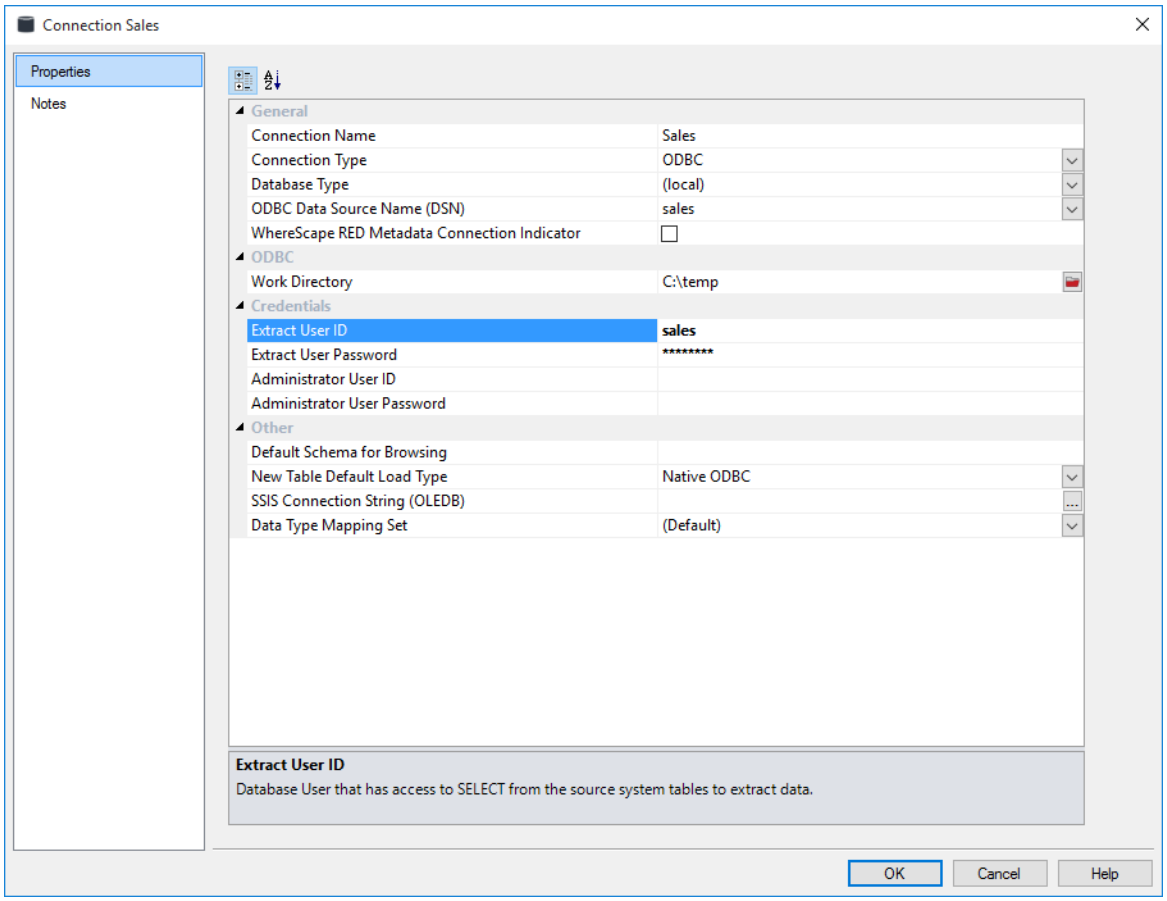
For more details on setting specific Greenplum, Netezza and PDW schema locations on a table by table level see ***Table Storage Properties***.

Once the connection has been set up, you can right-click on the connection in the middle pane or double click on the connection name from the left pane to view or edit the connection's Properties.

Below are two examples of a **Native ODBC** Load.
If the source database supports windows authentication, it is not necessary to specify a username and password:



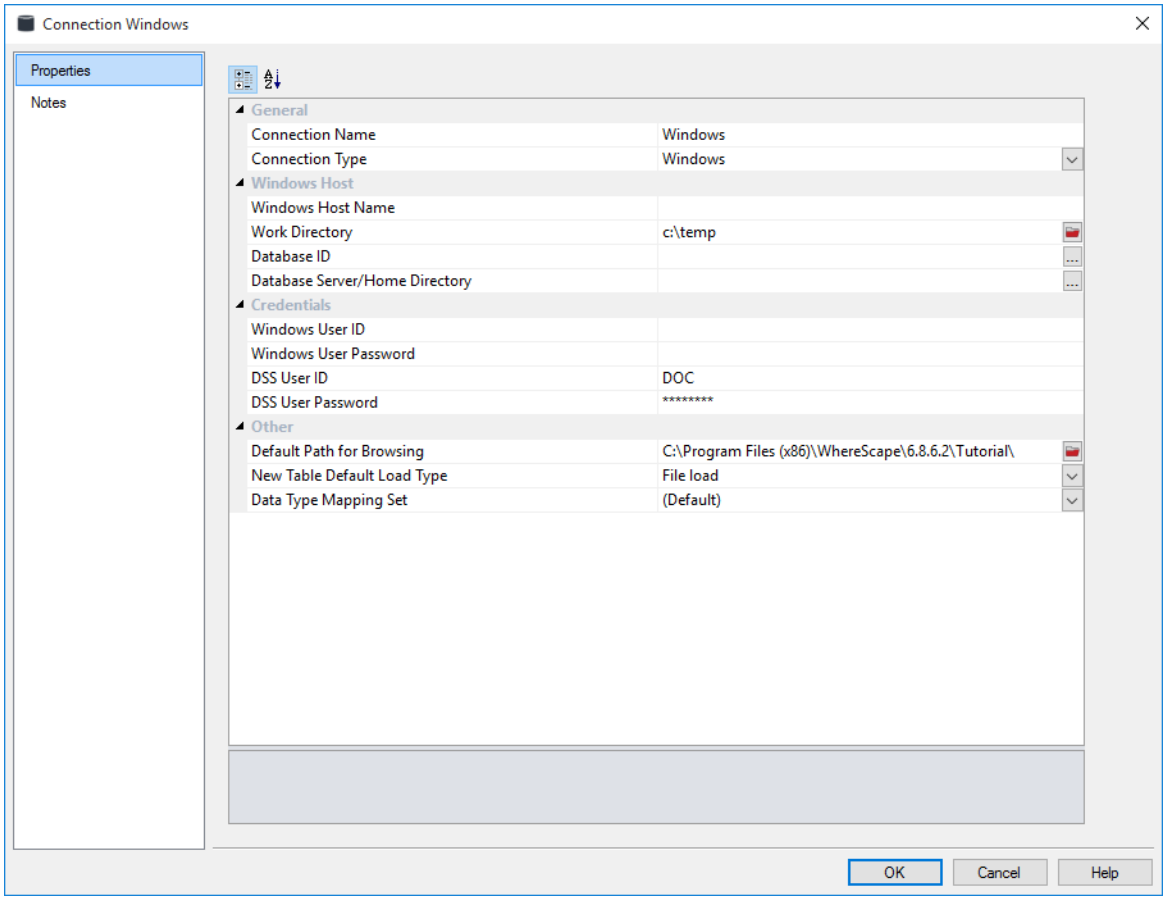
If the source database does not support windows authentication, a username and password must be specified:



WINDOWS

This connection is back to the PC that you are working on, or to a host Windows PC. From Windows you can only do flat file loads or file loads using SSIS.
See ***Loading Data into RED Load Tables using SSIS*** for more details about SSIS File Loads.

Sample **SQL Server Windows Connection** screen:



General

Connection Name

Name used to label the connection within WhereScape RED.

Connection Type

Indicates the connection source type or the connection method such as Database, ODBC, Windows, Unix. Set to **Windows**.

Windows Host

Windows Host Name

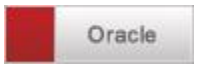
IP address or host name that identifies the Windows machine. Leave blank to connect to the local machine.

Work Directory

Windows directory used by WhereScape RED to create temporary files for minimal logged extracts. The directory must exist and allow write access. There must be a different work directory for each WhereScape RED Scheduler running on the same machine to avoid file conflicts. Typically, C:\Temp or a sub-directory of C:\Temp is used.

Database ID

The Source Database Identifier.

 	The **database name** of the Data Warehouse.
	The Oracle **SID** or **TNS Name**.

Database Server/Home Directory

Optional to specify the Database Home Directory if it is different from the standard home directory.

	The server name (or server/instance name) for the Data Warehouse database.

Credentials

Windows User ID and Password

Leave this blank if you are connecting to your own PC. Enter details if you are connecting remotely to another Windows system.

Dss User ID and Password

Enter the relevant details for connecting to the **Data Warehouse**.

Other

Default Path for Browsing

Optional default Path for browser pane filter. When a path has been selected in this field, it becomes the initial point for browsing and it is also expanded on open in the right-hand browser pane.

New Table Default Load Type

The default Load type for new tables created using this connection. Select from the **File Load**, **Script based load**, **XML file load**, **Integration Services load** or **Externally loaded** options.

Data Type Mapping Set

Mapping Set to use when converting from a source database data type to a destination database data type. Setting this field to **(Default)** makes RED automatically select the relevant mapping set, otherwise you can choose one of the standard mapping sets from the drop-down list or create a new set.

Once the connection has been set up, you can right-click on the connection in the middle pane or double click on the connection name from the left pane to view or edit the connection's Properties.

To test the connection

- Select **Browse** | **Source Tables** from the menu strip
- In the right pane, you should be able to drill down to the area required.

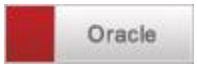
UNIX

UNIX/Linux connections are only supported from **Oracle, DB2** and **Greenplum** data warehouses.

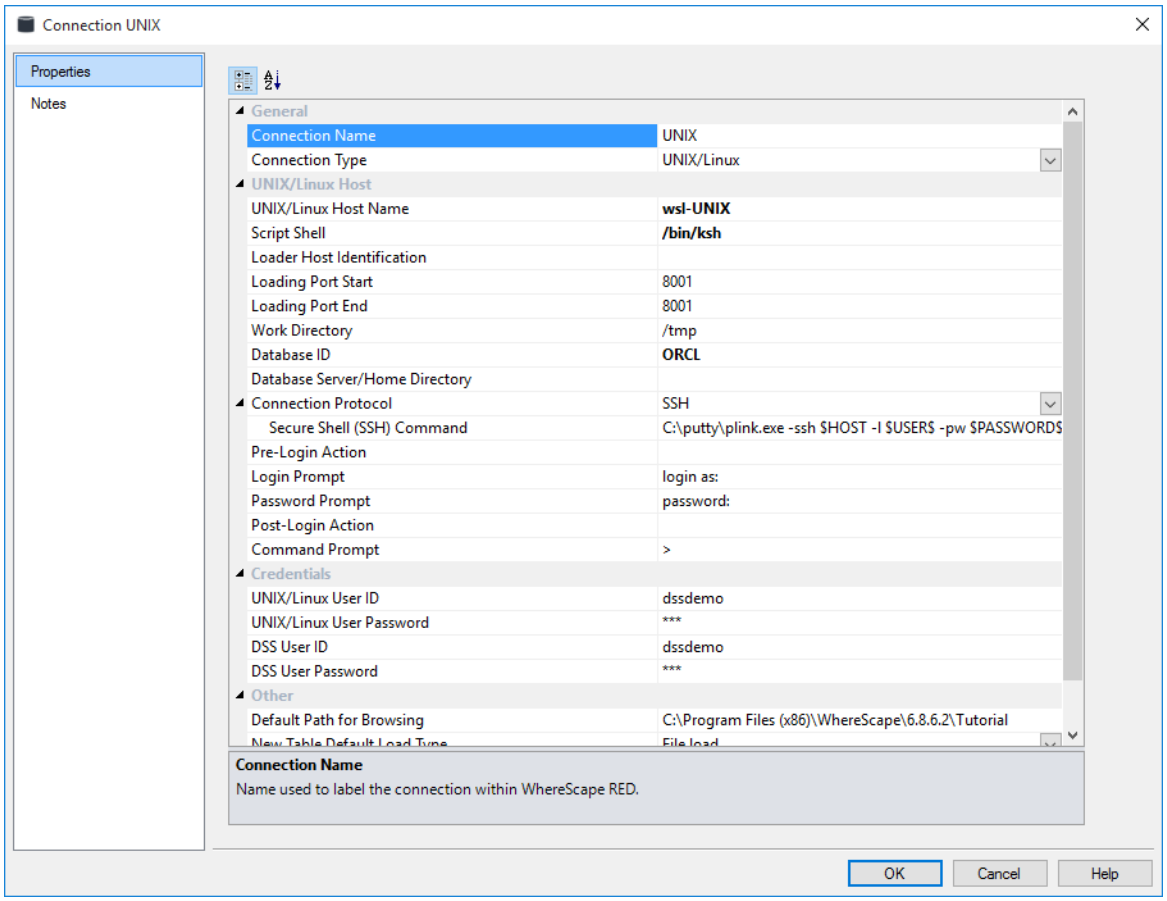
If the **UNIX/Linux** connection returns a blank screen or an error message in the **Results** pane after the connection is browsed, take necessary action through the **Server (SSH)** tab next to the main Builder and Scheduler tabs. 

This tab is displayed after browsing the UNIX connection.

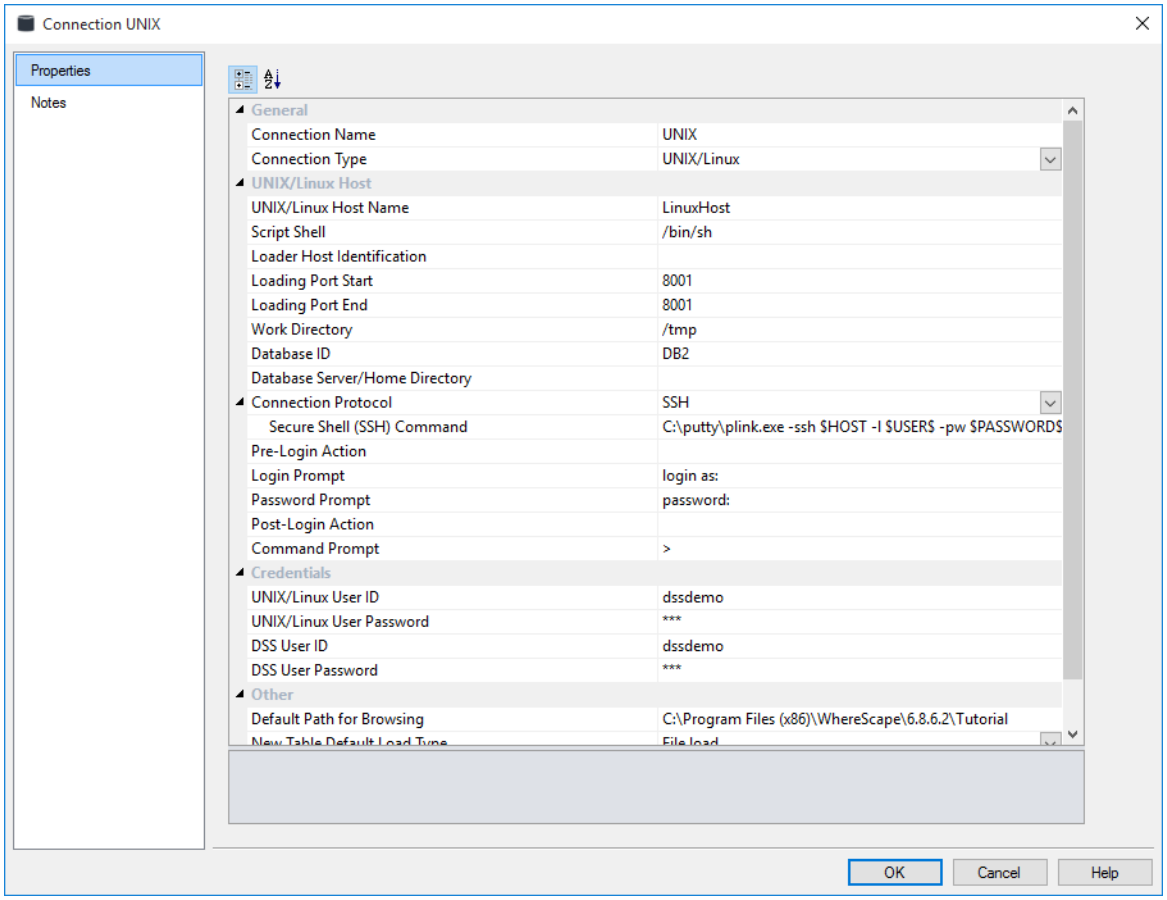
This topic describes in greater detail the connection Properties as they apply to **UNIX** connections. From a UNIX connection, you can only do flat file loads.
Tutorial 1 gives basic instructions for creating a connection.



Sample Oracle UNIX connection screen



Sample DB2 UNIX connection screen



General

Connection Name

Name used to label the connection within WhereScape RED.

Connection Type

Indicates the connection source type or the connection method such as Database, ODBC, Windows, Unix. Select **UNIX** from the drop-down list.

Unix/Linux Host

UNIX/Linux Host Name

IP address or host name that identifies the UNIX machine.

Script Shell

Path to the POSIX-compliant UNIX/Linux shell to use for generated scripts. For **UNIX** hosts, set to **/bin/ksh**. For **Linux** hosts set to **/bin/sh**.
If this field is left blank, a default will be chosen based on the name of the connection and the type of database used for the WhereScape RED metadata repository.

Loader Host Identification

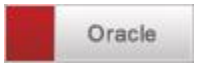
IP Address or host name(s) that identifies the Loader/ Multiple hosts can be entered with using a comma (,) to delimit.

Work Directory

Windows directory used by WhereScape RED to create temporary files for minimal logged extracts. The directory must exist and allow write access. There must be a different work directory for each WhereScape RED Scheduler running on the same machine to avoid file conflicts. Typically, **C:\Temp** or a **sub-directory of C:\Temp** is used.

Database ID

Source Database Identifier (e.g. Oracle SID or TNS Name, Teradata TDPID) or Database Name (e.g. as in DB2 or SQL Server).



For UNIX/Linux exports and loads, if the ORACLE_SID environment variable is set in your UNIX/Linux environment, this will be the variable used. If this environment variable is not set, then the value from the UNIX/Linux connection object will be used.

Database Server/Home Directory

Optional to specify the Database Home Directory if it is different from the standard home directory.

Connection Protocol

Telnet or Secure Shell (SSH) protocol to use to connect to the UNIX/Linux machine. For SSH, the **Secure Shell (SSH) Command** property is enabled to specify how to connect.

Secure Shell (SSH) Command

Command to execute to connect to a UNIX/Linux machine using the Secure Shell (SSH) protocol such as **C:\Program Files(x86)\PuTTY\plink.exe -ssh $HOST$ -l $USER$ -pw $PASSWORD$**

Pre-Login Action, Login Prompt, Password Prompt, Post-Login Action, and Command Prompt.

These fields are only used to create a Telnet connection to the host machine. WhereScape RED uses the Telnet connection in the drag and drop functionality.
It is not used in the actual production running of the Data Warehouse, and is only necessary if you wish to use the drag and drop functionality.

Pre-Login Action

Response or command to send BEFORE logging in to the UNIX/Linux machine. Typically, this is NOT necessary but it can be used to indicate that the UNIX/Linux Login Prompt is preceded by a line-feed (\n). However, it is preferable that the UNIX/Linux login displays the Login Prompt without anything preceding it. [Optional]

Login Prompt

The UNIX login prompt, or the tail end of the login prompt, e.g, **ogin as:**

Password Prompt

The UNIX password prompt, or the tail end of the password prompt, e.g, **ssword:**

Post-Login Action

Not often used but may be necessary to respond to a login question. It is preferable that the UNIX login goes straight to the command prompt.

Command Prompt

Enter the UNIX/Linux command prompt, or the tail end of that prompt, typically **>**

Note: To ascertain some of the above fields, you must log in to the UNIX system.

Credentials

UNIX/Linux User ID

User Account to login to the UNIX/Linux Host.

UNIX/Linux User Password

Password to login to the UNIX/Linux Host.

DSS User ID

Database User to connect to the WhereScape RED metadata repository.

DSS User Password

Database Password to connect to the WhereScape RED metadata repository.

Other

Default Path for Browsing

Optional default Path for browser pane filter. When a path has been selected in this field, it becomes the initial point for browsing and it is also expanded on open in the right-hand browser pane.

New Table Default Load Type

The default Load type for new tables created using this connection. Select from the **File Load**, **Script based load** or **Externally loaded** options.

Data Type Mapping Set

XML files have been created to store mappings from one set of data types to another. Setting this field to **(Default)** will cause RED to automatically select the relevant mapping set; otherwise you can choose one of the standard mapping sets from the drop-down list or create a new one.

To validate the fields

- Right-click on the connection name
- Select **Telnet window**

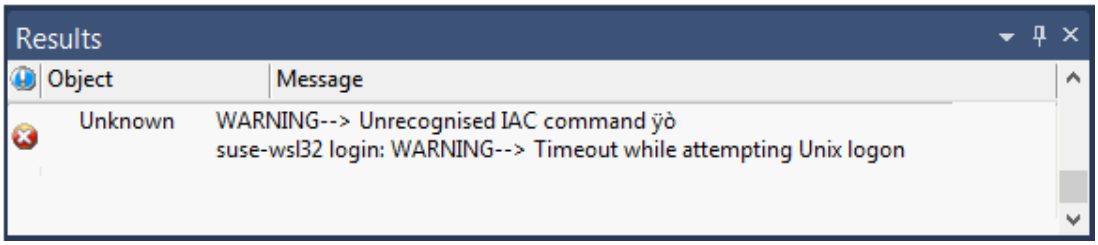
This will provide a telnet window that can be used to log on to the UNIX server.

To test the drag and drop functionality

- From the menu strip select **Browse** | **Source Tables**
- Drill down to the area required
- Drag an item to the middle pane, (having first selected the object in the left pane).

Connection Failures

If a telnet connection cannot be established to the UNIX host the following result box will normally appear after approximately 30 seconds.

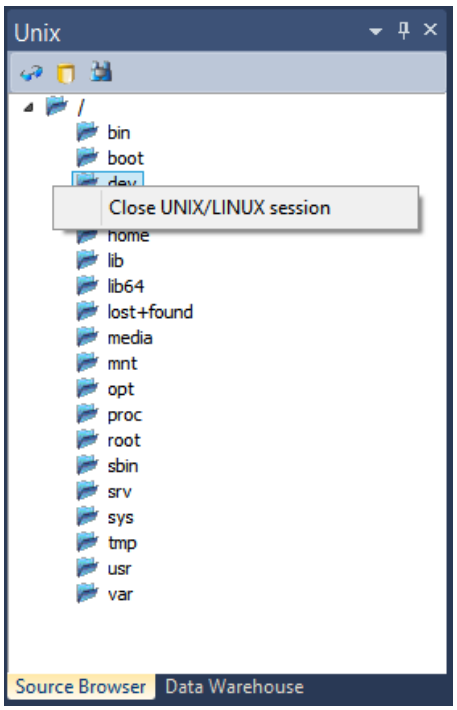


Attempt the connection again, and using the **Window** menu option select the **Telnet** window. This will display the login session, and should provide an insight as to why the connection is not being completed.

If the situation cannot be resolved, a telnet trace can be acquired. Select the menu option **Tools/Options** and click on the **Trace all Unix sessions** checkbox. Then try to do the connection or browse again. A log file called WslMedTelnet.log will be created in the WhereScape program directory. Edit the log file and ensure there are no passwords visible and then contact WhereScape support using the WhereScape forum at http://www.wherescape.com.

Closing the Connection

To close the collection, right-click in the browser pane and select **Close UNIX/LINUX session**:



HADOOP

This topic describes in greater detail the connection properties as they apply to **Hadoop** connections. Hadoop as a source works only with connections to Hadoop from which users can only do flat file loads. The connection must be set via a Secure Shell (SSH) protocol.
For Hadoop script loads from Oracle using **Oracle's Big Data Connectors**, see the next section: ***Hadoop using Oracle's Big Data Connectors***.

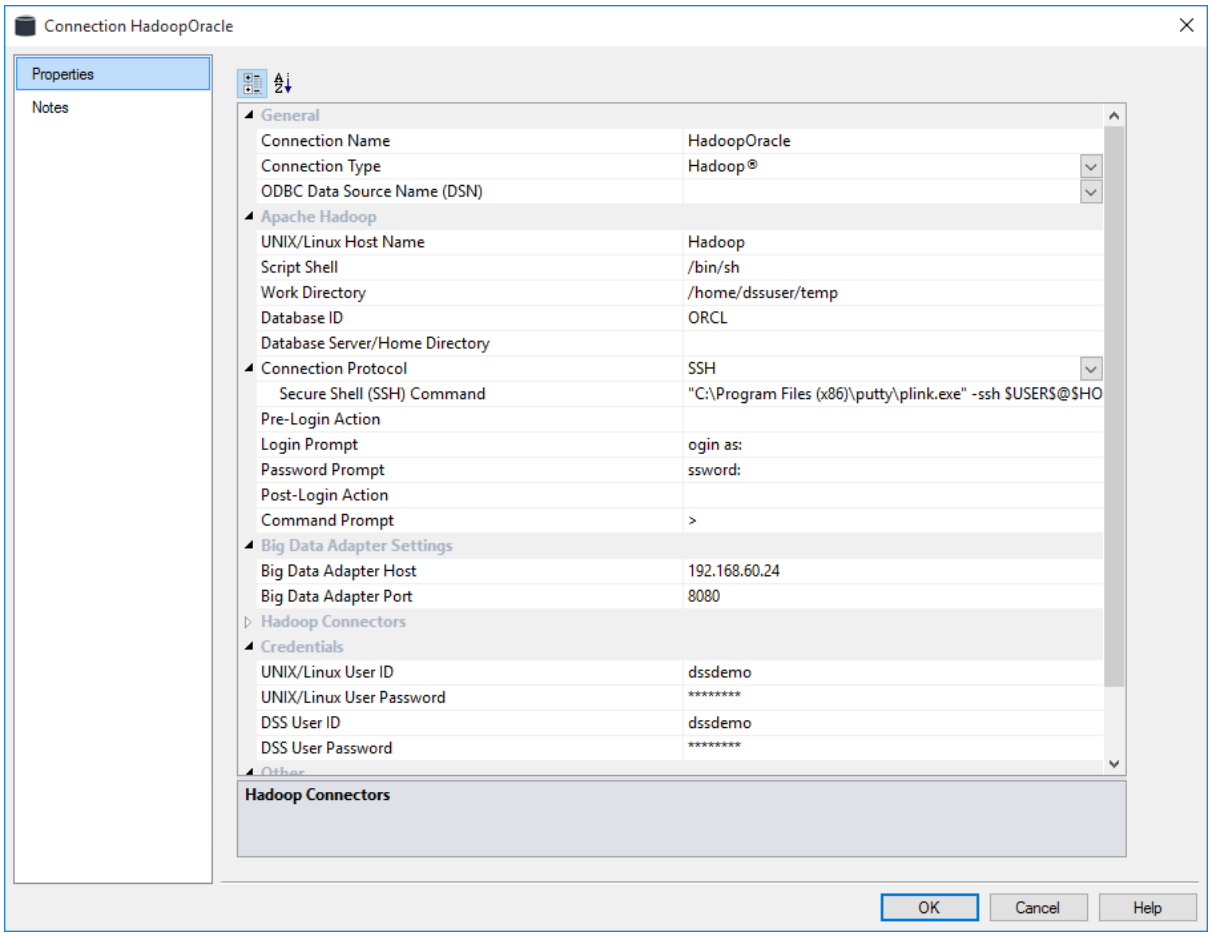
Please note that WhereScape RED only fully supports HDFS as the underlying file system.

WhereScape RED Tip: When the **Big Data Adapter** Settings are populated in Hadoop connections, RED can load data from Hadoop into Hive and/or Datawarehouse tables and also perform loads from Hadoop directly into the Datawarehouse using Sqoop through WhereScape RED's Big Data Adapter (BDA).
For more information about these settings, see the **Big Data Adapter Settings** fields description below and see also ***Connections to the Data Warehouse/Metadata Repository*** (see "***Database - Data Warehouse/Metadata Repository***" on page 143), ***Configuring your database for use by BDA*** and ***Apache Sqoop Load***.

If the **UNIX/Linux** connection returns a blank screen or an error message in the **Results** pane after the connection is browsed, take necessary action through the **Server (SSH)** tab next to the main Builder and Scheduler tabs. 

This tab is displayed after browsing the UNIX connection.

Sample Hadoop connection screen:



General

Connection Name

Name used to label the connection within WhereScape RED.

Connection Type

Indicates the connection source type or the connection method such as Database, ODBC, Windows, Unix. Set to **Hadoop**.

Apache Hadoop

UNIX/Linux Host Name

IP address or host name that identifies the Hadoop server.

Script Shell

Path to the POSIX-compliant UNIX/Linux shell to use for generated scripts. For **UNIX** hosts, set to **/bin/ksh**. For **Linux** hosts set to **/bin/sh**.
If this field is left blank, a default will be chosen based on the name of the connection and the type of database used for the WhereScape RED metadata repository.

Work Directory

Directory used by WhereScape RED to create temporary files for minimal logged extracts. The directory must exist and allow write access. There must be a different work directory for each WhereScape RED Scheduler running on the same machine to avoid file conflicts. Typically, **/tmp** or a **sub-directory of** the UNIX user is used.

Database ID

Source Database Identifier (e.g. Oracle SID or TNS Name, Teradata TDPID) or Database Name (e.g. as in DB2 or SQL Server).

Database Server/Home Directory

Optional to specify the Database Home Directory if it is different from the standard home directory.

Connection Protocol

Telnet or Secure Shell (SSH) protocol to use to connect to the Hadoop machine. For SSH, the **Secure Shell (SSH) Command** property is enabled to specify how to connect. Select **SSH**.

Secure Shell (SSH) Command

Command to execute to connect to a Hadoop machine using the Secure Shell (SSH) protocol such as **C:\putty\plink.exe -ssh some_host_name**

Pre-Login Action, Login Prompt, Password Prompt, Post-Login Action, and Command Prompt:

These fields are only used to create a Telnet connection to the host machine. WhereScape RED uses the Telnet connection in the drag and drop functionality.
They are **not used in the actual production running of the Data Warehouse,** and are only necessary if you wish to use the drag and drop functionality.

Pre-Login Action

Response or command to send BEFORE logging in to the UNIX/Linux machine. Typically, this is NOT necessary but it can be used to indicate that the UNIX/Linux Login Prompt is preceded by a line-feed (\n). However, it is preferable that the UNIX/Linux login displays the Login Prompt without anything preceding it. [Optional]

Login Prompt

The UNIX login prompt, or the tail end of the login prompt, e.g, **ogin as:**

Password Prompt

The UNIX password prompt, or the tail end of the password prompt, e.g, **ssword:**

Post-Login Action

Not often used but may be necessary to respond to a login question. It is preferable that the UNIX login goes straight to the command prompt.

Command Prompt

Enter the UNIX/Linux command prompt, or the tail end of that prompt, typically >

Note: To ascertain some of the above fields, you must log in to the UNIX/Linux system.

Big Data Adapter Settings

Set the two fields below to enable RED to communicate with BDA and enable loading data from Hadoop into Hive and/or into Datawarehouse tables using Sqoop.
For further information about setting these fields, see ***Connections to the Data Warehouse/Metadata Repository*** (see "***Database - Data Warehouse/Metadata Repository***" on page 143) and ***Configuring the BDA Server/Configuring your database for use by BDA***.

Big Data Adapter Host

Host machine on which the Big Data Adapter is running its web-server.

Big Data Adapter Port

Port that Tomcat is running. Default is 8080.

Hadoop Connectors

Only available for Oracle databases using Oracle's Big Data Connectors. See ***Hadoop using Oracle's Big Data Connectors*** for more information.

Credentials

UNIX/Linux User ID

User Account to login to the UNIX/Linux Host.

UNIX/Linux User Password

Password to login to the UNIX/Linux Host.

DSS User ID

Database User to connect to the WhereScape RED metadata repository.

DSS User Password

Database Password to connect to the WhereScape RED metadata repository.

Other

Default Path for Browsing

Optional default Path for browser pane filter. When a path has been selected in this field, it becomes the initial point for browsing and it is also expanded on open in the right-hand browser pane.

Default Path for Browsing

Optional default Path for browser pane filter. When a path has been selected in this field, it becomes the initial point for browsing and it is also expanded on open in the right-hand browser pane.

New Table Default Load Type

The default Load type for new tables created using this connection. Select from the **Script based load**, **Native SSH** or **Externally loaded** options.

SQL Server Hadoop Native SSH Loads

Please note that for SQL Server Hadoop Native SSH loads to be processed successfully, you must be running RED and the RED Scheduler on same machine as SQL Server to ensure that the files are accessible from the same path.

Data Type Mapping Set

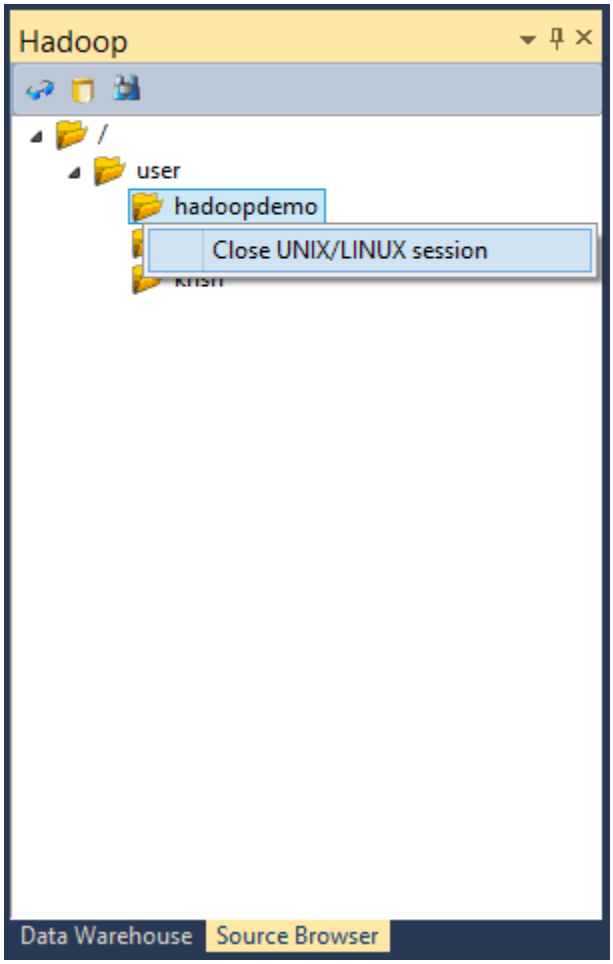
XML files have been created to store mappings from one set of data types to another. Setting this field to **(Default)** will cause RED to automatically select the relevant mapping set; otherwise you can choose one of the standard mapping sets from the drop-down list or create a new one.

To test the drag and drop functionality

- From the menu strip select **Browse** | **Source Tables**
- Drill down to the area required
- Drag an item to the middle pane, (having first selected the object in the left pane).

Closing the Connection

To close the collection, right-click in the browser pane and select **Close UNIX/LINUX session**:


Hadoop
/
user
hadoopdemo
Close UNIX/LINUX session
Data Warehouse
Source Browser

HADOOP USING ORACLE'S BIG DATA CONNECTORS

WhereScape RED can load data from Hadoop into an Oracle repository using Oracle's Big Data Connectors **Oracle SQL Connector for Hadoop (OSCH)** and **Oracle Loader for Hadoop (OLH).**
RED loads data from Hadoop using Oracle's Big Data connectors via a Hadoop connection on UNIX/Linux from which users can do script based loads using the drag and drop functionality.
For more details about loading data from Hadoop after the connection is set up in RED, see section – ***Flat File Load*** and Flat File Load - Source Screen.
The following two sections describe the connection properties from within RED, using either the **Oracle SQL Connector for Hadoop (OSCH)** or the **Oracle Loader for Hadoop (OLH)** connectors.

To do loads from Oracle using the **OSCH** or **OLH** connectors, users will need to have the following system prerequisites before setting up a connection within RED:

- Oracle on Linux/Unix Hadoop tools installed on the Oracle Server and configured to connect to the Hadoop client
- **OLH** installed
- **OSCH** and/or **OLH** configured to connect to the Hadoop cluster and the Oracle database
- Hadoop client installed
- Oracle Wallet created and configured
- For **OSCH** - Oracle Directory Objects configured for the database user
- User permissions

Note: For scheduler configuration settings for the database user see section **9.8.1 Unix Scheduler for Native Loads using Oracle's Big Data Connectors** of the RED Installation Guide.

Hadoop for Oracle Connectors

This topic describes in greater detail the connection properties as they apply to **Hadoop** connections using either **Oracle's SQL Connector for Hadoop (OSCH)** or **Oracle's Loader for Hadoop (OLH)**.
If the connection to Hadoop will not be using Oracle's Big Data connectors, refer to the previous section - ***Hadoop***.
In RED, Hadoop as a source works with connections to Hadoop on UNIX/Linux from which users can do script based loads. The connection must be set via a Secure Shell (SSH) protocol.
To know more about system prerequisites required for setting up a connection within RED using either of these connectors see the previous topic ***Hadoop using Oracle's Big Data Connectors***.

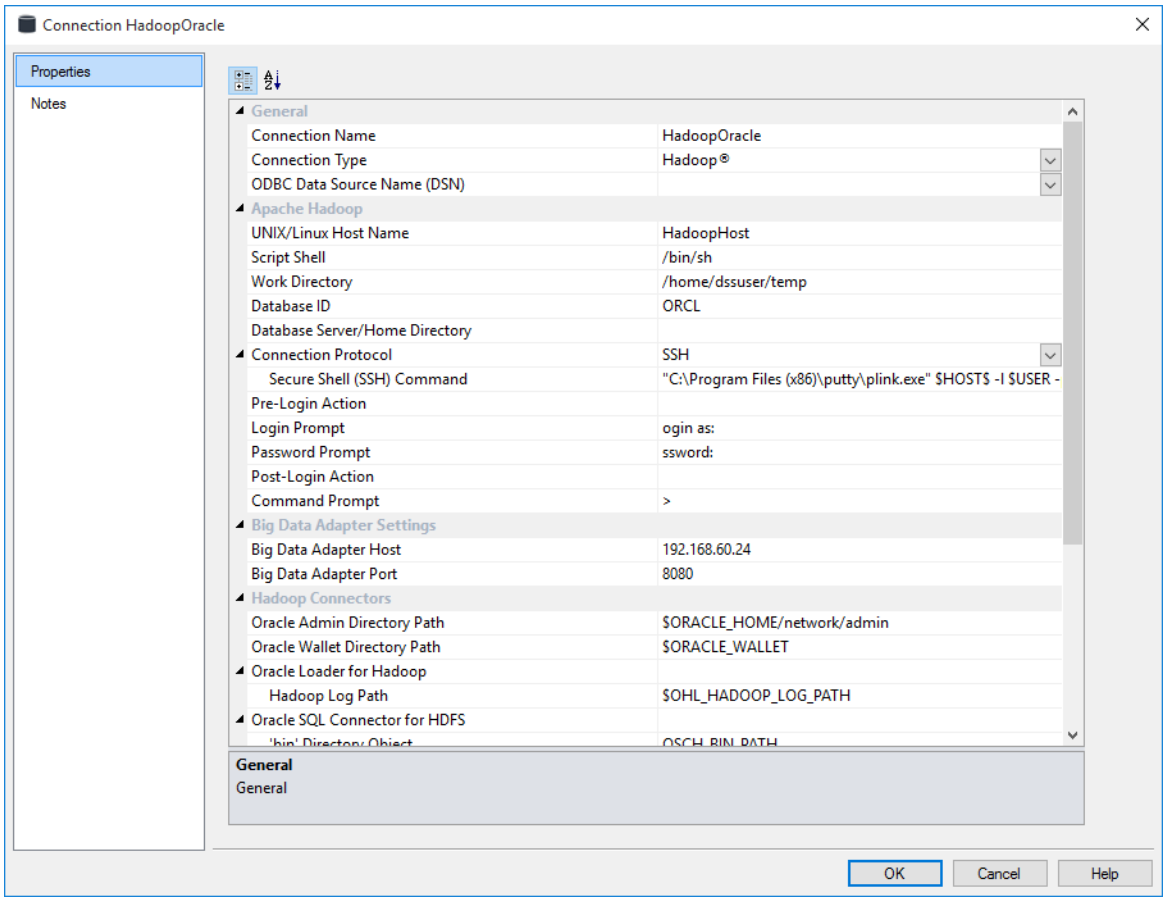
Oracle Wallet for OSCH:
An Oracle Wallet must be configured and used for the OSCH connector to work.
The Oracle user + SID used in the RED Connection object must be added to the Oracle Wallet.
If the Oracle user is not added to the wallet (or if it's added against a different SID, even if equivalent to the one used in the RED Connection object) then OSCH will fail with this message: "KUP04076: file name cannot contain a path specification."

Oracle Wallet for OLH:
RED only supports OLH with the Oracle Wallet, therefore an Oracle Wallet must be used with OLH. Although OLH does not require the use of the Oracle Wallet (unlike OSCH), the use of OLH without it is discouraged by Oracle. The Oracle user + SID used in the RED Connection object must be added to the Oracle Wallet.

If the **UNIX/Linux** connection returns a blank screen or an error message in the **Results** pane after the connection is browsed, take necessary action through the **Server (SSH)** tab next to the main Builder and Scheduler tabs. 

This tab is displayed after browsing the UNIX connection.

Sample **Hadoop OSCH/OLH connection** screen:



General

Connection Name

Name used to label the connection within WhereScape RED.

Connection Type

Indicates the connection source type or the connection method such as Database, ODBC, Windows, Unix. Set to **Hadoop**.

Apache Hadoop

UNIX/Linux Host Name

IP address or host name that identifies the Hadoop server.

Script Shell

Path to the POSIX-compliant UNIX/Linux shell to use for generated scripts. For **UNIX** hosts, set to **/bin/ksh**. For **Linux** hosts set to **/bin/sh**.
If this field is left blank, a default will be chosen based on the name of the connection and the type of database used for the WhereScape RED metadata repository.

Work Directory

Windows directory used by WhereScape RED to create temporary files for minimal logged extracts. The directory must exist and allow write access. There must be a different work directory for each WhereScape RED Scheduler running on the same machine to avoid file conflicts. Typically, **/home/dssdemo/temp** or a sub-directory of **/home/dssdemo/temp** is used.

Database ID

Source Database Identifier (e.g. Oracle SID or TNS Name, Teradata TDPID) or Database Name (e.g. as in DB2 or SQL Server).

Database Server/Home Directory

Optional to specify the Database Home Directory if it is different from the standard home directory.

Connection Protocol

Telnet or Secure Shell (SSH) protocol to use to connect to the Hadoop machine. For SSH, the **Secure Shell (SSH) Command** property is enabled to specify how to connect. Select **SSH**.

Secure Shell (SSH) Command

Command to execute to connect to a Hadoop machine using the Secure Shell (SSH) protocol such as **C:\Program Files (x86)\putty\plink.exe -ssh $HOST$ -l $USER$ -pw $PASSWORD$.**

Pre-Login Action, Login Prompt, Password Prompt, Post-Login Action, and Command Prompt:

These fields are only used to create a Telnet connection to the host machine. WhereScape RED uses the Telnet connection in the drag and drop functionality.
They are **not used in the actual production running of the Data Warehouse,** and are only necessary if you wish to use the drag and drop functionality.

Pre-Login Action

Response or command to send BEFORE logging in to the UNIX/Linux machine. Typically, this is NOT necessary but it can be used to indicate that the UNIX/Linux Login Prompt is preceded by a line-feed (\n). However, it is preferable that the UNIX/Linux login displays the Login Prompt without anything preceding it. [Optional]

Login Prompt

The UNIX login prompt, or the tail end of the login prompt, e.g, **ogin as:**

Password Prompt

The UNIX password prompt, or the tail end of the password prompt, e.g, **ssword:**

Post-Login Action

Not often used but may be necessary to respond to a login question. It is preferable that the UNIX login goes straight to the command prompt.

Command Prompt

Enter the UNIX/Linux command prompt, or the tail end of that prompt, typically **>**

NOTE: To ascertain some of the above fields, you must log in to the UNIX/Linux system.

Big Data Adapter Settings

When using Oracle's Big Data Connectors to load data from Hadoop into Hive, it is not required to set the BDA fields.

Hadoop Connectors

Connector Type

Big Data Connectors that connect Hadoop with the Datawarehouse Database. **Oracle SQL Connector for HDFS** and **Oracle Loader for Hadoop** fields are available. Enter the relevant settings for each connector type in the fields displayed.

NOTE: To load data using one of these two specific connectors, the relevant connector must be selected from the **Hadoop Loader** field in the load table's **Source** screen. See Flat File Load - Source Screen for more details.

Oracle Loader for Hadoop

Hadoop Log Path

Hadoop path to the log directory where a subdirectory will be created with logs for each load run. This is the Hadoop path to main log directory. A subdirectory is created for each job run and is then passed to OLH – if the directory passed to OLH does not exist or is not empty this will result in a job run failure.

Example

If "/user/oracle/hadoop/output/" is supplied in the connection object and the user then runs the job to load "hundredR.csv" file, the subdirectory "/user/oracle/hadoop/output/WSL_hundredR_SEQ_RESTARTCOUNT/" will be created to store the logs, where "SEQ" is RED's sequence number and "RESTARTCOUNT" is the job's restart count.

Oracle SQL Connector for HDFS

'bin' Directory Object

Oracle DIRECTORY object that points to the file location where the hdfs_stream exists.
Oracle Directory object for the 'bin' directory of OSCH that must be installed on the Oracle server. This is best to be set up within the home directory of the user that runs Oracle but outside of ORACLE_HOME. e.g. If the Oracle user is 'oracle' then a good choice would be '/home/oracle/osch/<install_dir>/bin').

NOTE: The Hadoop client should be installed on the Oracle server. Oracle user should have read and execute permissions.

External Tables Directory Object

Oracle DIRECTORY object that points to the directory where OSCH location files live.
Oracle Directory object for the directory where Oracle is to keep 'location files' (xml files that 'link' external table in Oracle and target file in Hadoop). This directory must be on the Oracle server, and it is best to set up within the home directory of the user that runs Oracle, but outside of ORACLE_HOME. e.g. If the Oracle user is 'oracle' then good choice would be '/home/oracle/osch/exttab'). The Oracle user should have read and write permissions.

Log Directory Object

Oracle DIRECTORY object that points to the OSCH log directory.
It can point to the directory where the 'location files' are (it will point to the loader default directory if it is not specified). Oracle user should have read and write permissions.

Oracle Admin Directory Path

OS path to the directory that contains the 'tnsnames.ora' file.
This is the full path to 'tnsnames.ora' – "$ORACLE_HOME/network/admin".

Oracle Wallet Directory Path

OS path to an Oracle wallet directory where the connection credential is stored.
The full path to the Oracle Wallet location. It can be set to anything – e.g. "$ORACLE_HOME/network/admin" or "$ORACLE_HOME/wallets", but if the wallet is created on the Oracle server then it will likely be in Oracle ACFS when available (e.g. "/u02/app/oracle/wallet/").

Example

To create an Oracle directory object named "osch_bin_path", pointing to "/home/oracle/osch/orahdfs-3.1.0/bin" directory and then grant read and execute permissions on that object to Oracle user "oschuser":
CREATE DIRECTORY osch_bin_path AS '/home/oracle/osch/orahdfs-3.1.0/bin';
GRANT EXECUTE ON DIRECTORY osch_bin_path TO oschuser;
GRANT READ ON DIRECTORY osch_bin_path TO oschuser;

NOTE: It is possible to either provide values or to provide names of UNIX environment variables in the connection object parameter values.
If UNIX environment variables are used instead of actual values, note that UNIX/Linux scheduler must be restarted after any changes to these variables or those changes will not be available to any processes scheduled to run on that scheduler. Since these values should be very stable (they probably never change) this is only an issue during the environment setup – the administrator must remember to add those variables to the environment first before starting the scheduler. If the variables are ever changed then the scheduler needs to be restarted.
All environment variables must start with $.

Credentials

UNIX/Linux User ID

User Account to login to the UNIX/Linux Host.

UNIX/Linux User Password

Password to login to the UNIX/Linux Host.

DSS User ID

Database User to connect to the WhereScape RED metadata repository. Not required. All authentication is handled through Oracle Wallet since Oracle SQL Connector for Hadoop will not work without it.

DSS User Password

Database Password to connect to the WhereScape RED metadata repository. Not required. All authentication is handled through Oracle Wallet since Oracle SQL Connector for Hadoop will not work without it.

NOTE: To perform loads from **Hadoop** using multiple schemas, the RED must be granted an extra set of privileges described in section **9.3 Creating an Oracle Dss User** of the RED Installation Guide.

Other

Default Path for Browsing

Optional default Path for browser pane filter. When a path has been selected in this field, it becomes the initial point for browsing and it is also expanded on open in the right hand browser pane.

New Table Default Load Type

The default Load tye for new tables created using this connection. Select from the **Script based load**, **Native SSH** or **Externally loaded** options.

Data Type Mapping Set

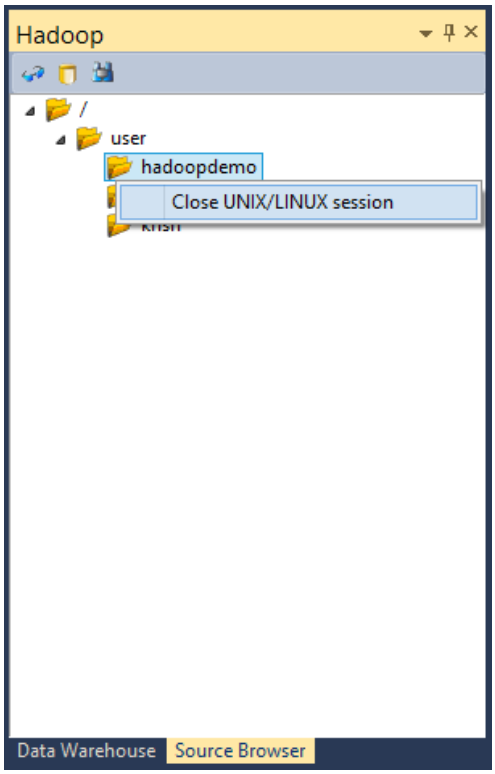
XML files have been created to store mappings from one set of data types to another. Setting this field to **(Default)** will cause RED to automatically select the relevant mapping set; otherwise you can choose one of the standard mapping sets from the drop-down list or create a new one.

To test the drag and drop functionality

- From the menu strip select **Browse** | **Source Tables**
- Drill down to the area required
- Drag an item to the middle pane, (having first selected the object in the left pane).

Closing the Connection

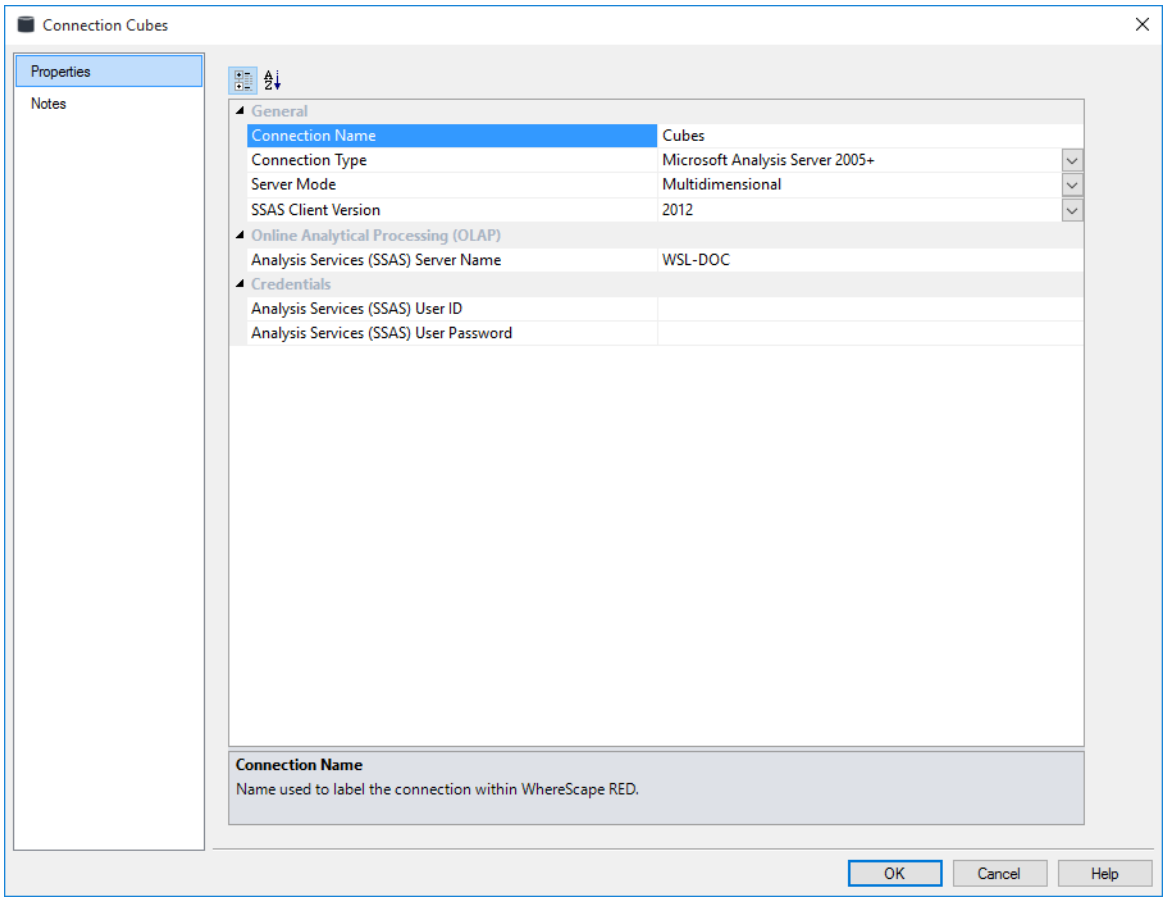
To close the collection, right-click in the browser pane and select **Close UNIX/LINUX session**:



MICROSOFT ANALYSIS SERVER 2005+

MICROSOFT ANALYSIS SERVER 2005+ - OLAP CUBES

A Connection to an **Analysis Services server** provides the location for cubes defined in the metadata repository. This connection is used in the creation and processing of cubes.
A sample screen shot follows:



General

Connection Name

Enter a name to identify the connection to the Analysis Services server

Connection Type

Indicates the connection source type or method. Select Microsoft Analysis Server 2005+.

Server Mode

The operational mode that Microsoft Analysis Services will use. Select **Multidimensional** from the drop-down list.

SSAS Client Version

Microsoft Analysis Services Client version available for connecting to the SSAS database. It is recommended that the client version matches your database version.

If a 'Fail - missing AMO data provider' message is displayed in the Results pane when attempting to execute the OLAP action, check that the correct SSAS client version is specified and that the respective version of the data provider is installed on the client workstation.

Note: If you have SSAS client version 2008 installed on your computer, WhereScape recommends selecting **2012** for the SSAS client version.

If the required SQL Server Analysis Management Objects (AMO) are missing, see the following article for more information: **https://msdn.microsoft.com/en-us/library/dn141152.aspx**.

Online Analytical Processing (OLAP)

Analysis Services Server Name

Enter the name of the Analysis Services server you wish to connect to.

Credentials

Analysis Services (SSAS) User ID and Password

User Name to connect to Analysis Services with when using SQL Server Authentication. Can be left blank for a trusted connection using Windows Authentication.

Analysis Services (SSAS) User ID and Password

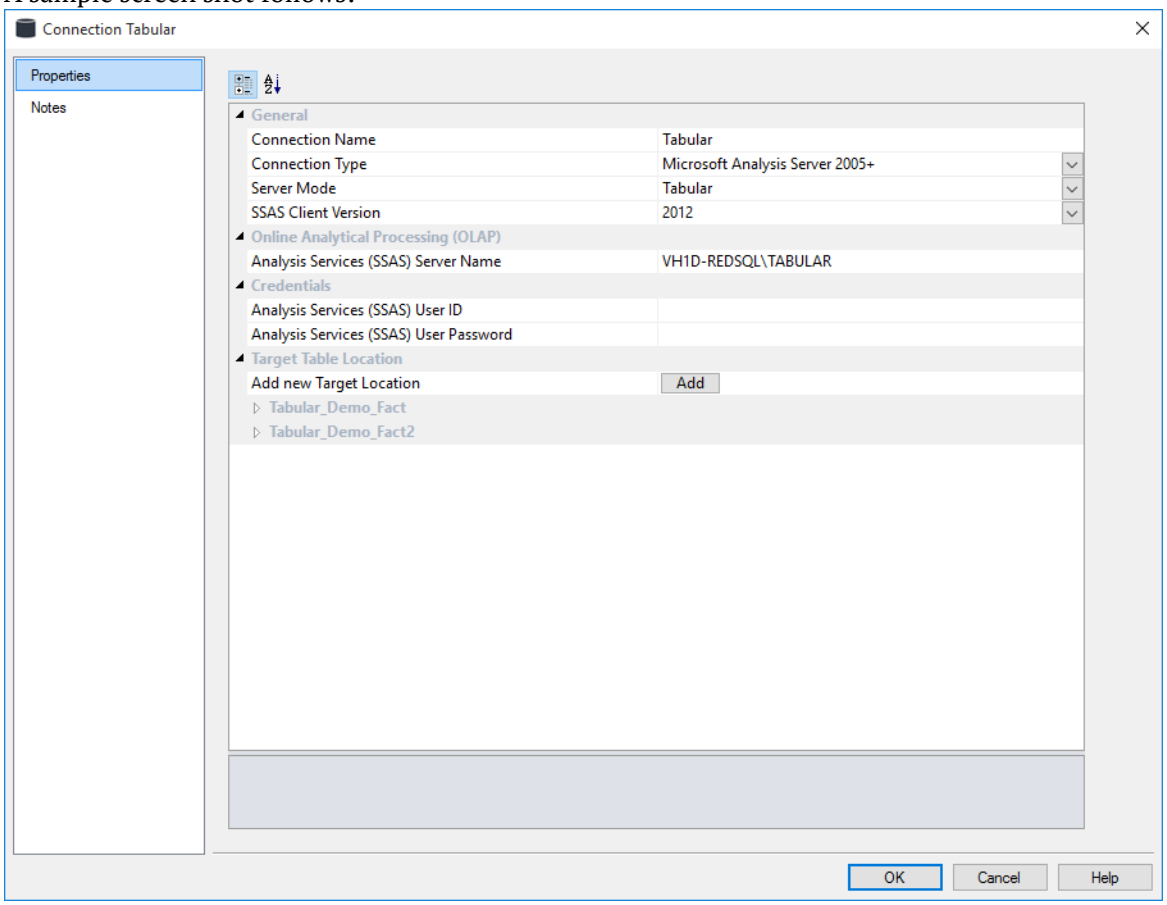
Password to connect to Analysis Services with when using SQL Server Authentication. Can be left blank for a trusted connection using Windows Authentication.

MICROSOFT ANALYSIS SERVER 2005+ - TABULAR MODE

A Connection to an **Analysis Services server** in **Tabular Mode** provides the location for Tabular cubes defined in the metadata repository. This connection is used in the creation and processing of Tabular cubes.

NOTE: Relationships must be created manually for tables stored on a Tabular target, for more information see ***Relationship Maintenance*** (on page 1150).

A sample screen shot follows:



General

Connection Name

Enter a name to identify the connection to the Analysis Services server

Connection Type

Indicates the connection source type or method. Select Microsoft Analysis Server 2005+.

Server Mode

The operational mode that Microsoft Analysis Services will use. Select **Tabular** from the drop-down list.

SSAS Client Version

Microsoft Analysis Services Client version available for connecting to the SSAS database. It is recommended that the client version matches your database version. If the required SQL Server Analysis Management Objects (AMO) are missing, see the following article for more information: **https://msdn.microsoft.com/en-us/library/dn141152.aspx**.

Online Analytical Processing (OLAP)

Analysis Services Server Name

Enter the name of the Analysis Services server you wish to connect to. You may need to specify the port number of the Analysis Services instance. To find your port number, follow the procedure documented in this Microsoft article: **https://support.microsoft.com/en-us/kb/2466860**.

An example of your Analysis Server (SSAS) Server Name using the port number in RED would be:
`VH1D-REDSQL:49449\TABULAR`

Credentials

Analysis Services (SSAS) User ID and Password

User Name to connect to Analysis Services with when using SQL Server Authentication. Can be left blank for a trusted connection using Windows Authentication.

Analysis Services (SSAS) User ID and Password

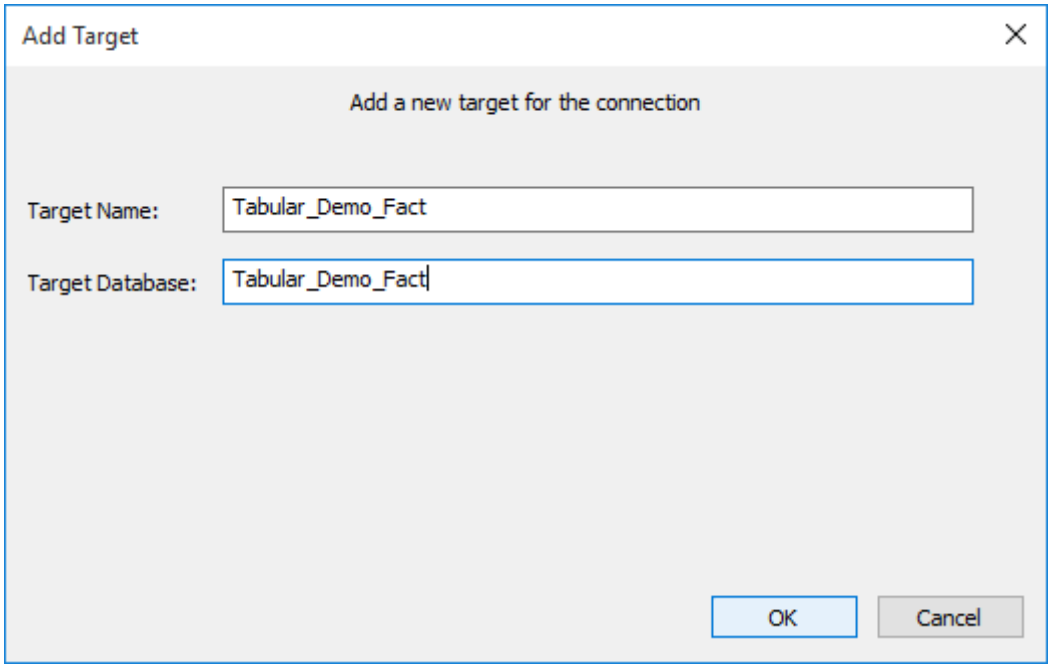
Password to connect to Analysis Services with when using SQL Server Authentication. Can be left blank for a trusted connection using Windows Authentication.

Target Table Location

Add new Target Location

Click the Add new Target Location button to specify the name of the database associated with the targets to be used with this connection. To use the Tabular Mode functionality, it is required to create targets in the Tabular connection.

- The **Target Name** will be the relevant Tabular Database's name displayed in RED.
- The **Target Database** will be the relevant Tabular Database's name displayed in Analysis Services.



BROWSING A CONNECTION

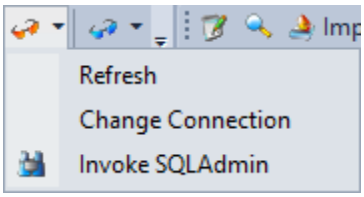
The tables or files associated with a connection can be displayed in the Browser Pane by:

1. selecting the **Browse/Data Warehouse** menu option to browse for the data warehouse connection,
2. selecting the **Browse/Source Tables** menu option to browse a source system connection,
3. right-clicking on a Connection in the Object Pane and selecting **Browse Connection**, or
4. clicking on one of the two browser buttons on the toolbar:

The orange button is used to browse the data warehouse connection and the blue button is used to browse a source system connection.

Each button remembers the last connection it browsed, so in this way one button can be used for the Data Warehouse and one for a source system.

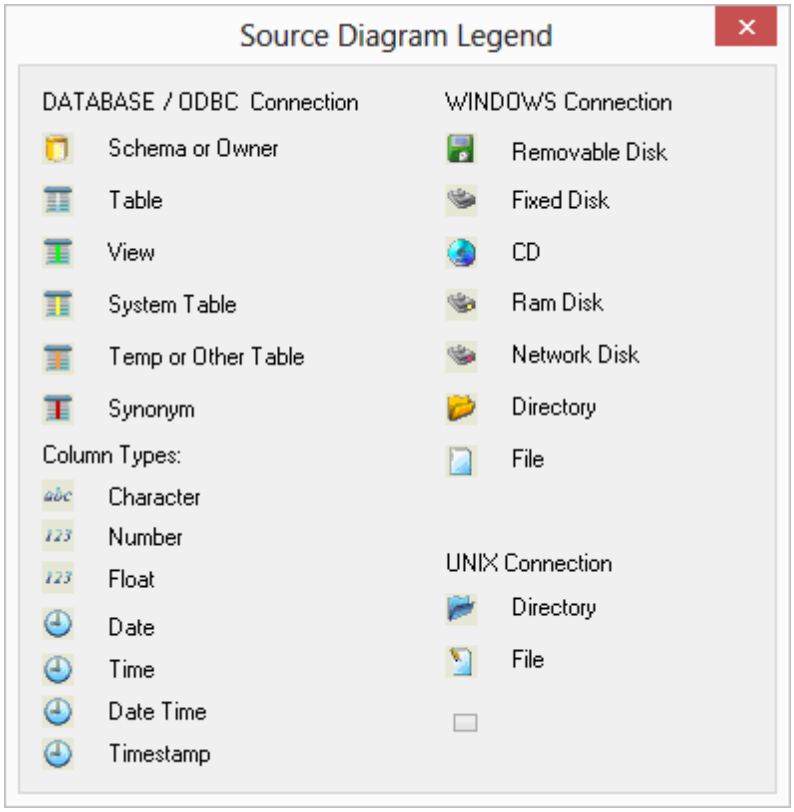
Clicking on one of the buttons will display the source tables without first displaying the source browser dialog box. To change the connection, click the small black down arrow next to one of the browser buttons on the toolbar and select Change Connection.



The current connection being browsed is shown in the status bar at the bottom right of the screen.

Browser Icons

When browsing a connection, the following legend applies for the source tables and objects. The legend is displayed via the **Help/Source Legend** menu:

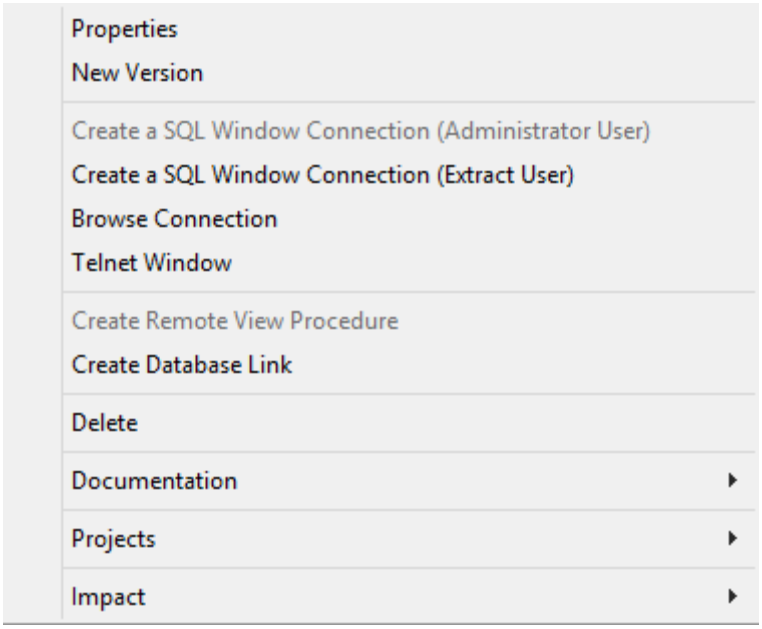


CONNECTION BROWSE PROPERTIES

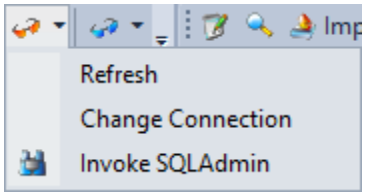
TIP: When browsing to a connection leave the schema field blank to see all schemas. To have RED browsing a specific schema or schemas by default, go to a connection's properties screen and enter the schema(s) to browse on the **Default Schema for Browsing** field.

To change the properties of the connection in the Browser Pane:

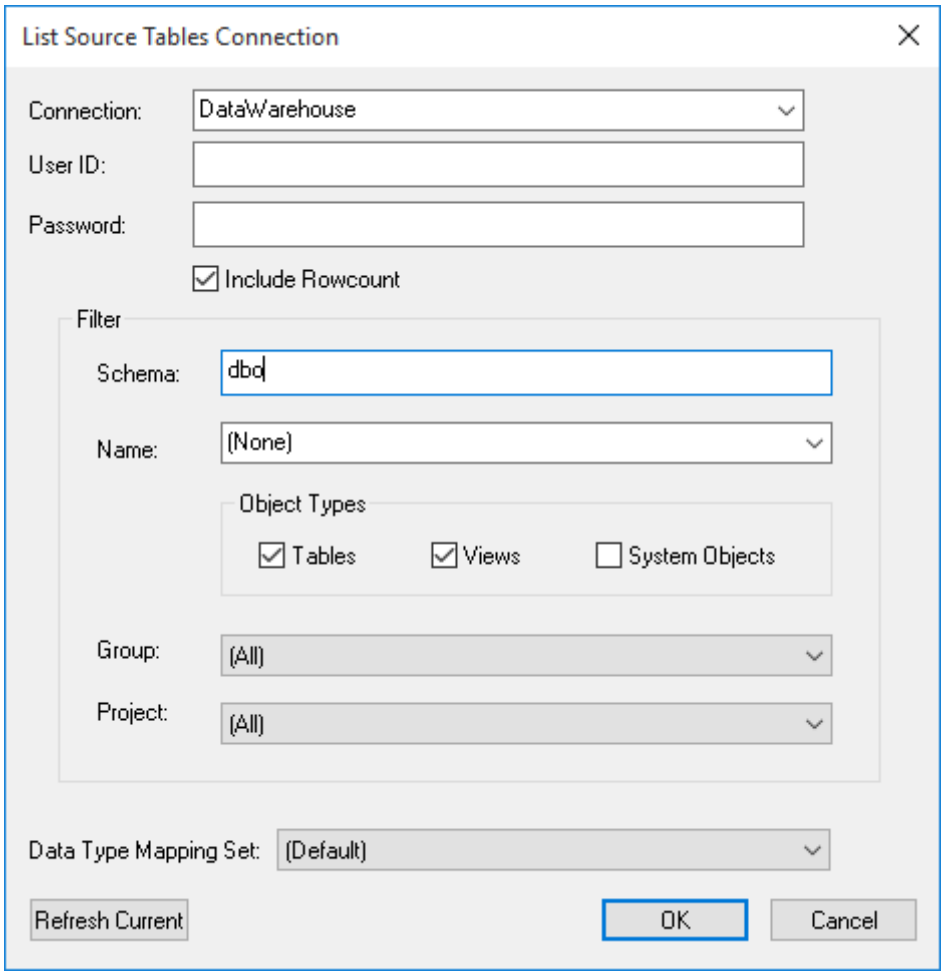
- Right-click on a connection in the Object Pane and select **Browse Connection**, or



- click the small black down arrow next to one of the Browser buttons on the toolbar and select **Change Connection**.



The **List Source Tables Connection** dialog is displayed:



The dialog allows you to change the properties of the connection you are browsing.

The **User ID** and **Password** fields can be changed to browse the connection as a different user.

Ticking the **Include Rowcount** check-box displays a row count in brackets next to each source table in the Browser Pane. This is only available for databases which update table statistics.

A **filter** can also be applied when browsing a connection. Filters can be applied to any combination of:

- One or more Schema names (separated by commas),
- a standard SQL table name,
- specific Object Types (Tables, Views or System Objects),
- a Group, or
- a Project

The **Data Type Mapping Set** drop-down can be used to change the data type conversion used during drag and drop operations. If set to (Default) the Data Type Mapping to use for each drag and drop operation is set based on the source and the Target Location selected, but can be changed in the Add New Metadata Object dialog if needed.

CHANGING A CONNECTION'S PROPERTIES

Whenever a connection's Properties are changed, the impact on the objects that use that connection must be considered.
Load tables have information from the connection stored within their Properties. This information is stored in the load objects to minimize the complexity of the scheduled tasks.
The database link and database name are stored locally in each load table. When either the database link or database name are changed on a connection, WhereScape RED displays the **Update Associated Load Tables** dialog box.
Click **Yes** to automatically update the database link and/or database name on all associated load tables.

This can also be done manually:

1. Double-click on the **Load Tables object group** in the left pane. This will display all load tables in the middle pane.
2. Select those load tables that use the connection that has changed. Use standard Windows selection.
3. Right-click to bring up a menu and select **Change Connect/Schema**.
4. Select a different connection (e.g. Data warehouse) to change all the selected load tables.
5. Repeat step (3) and now change the tables back to the altered connection. This will update all the load tables with the new connection information.

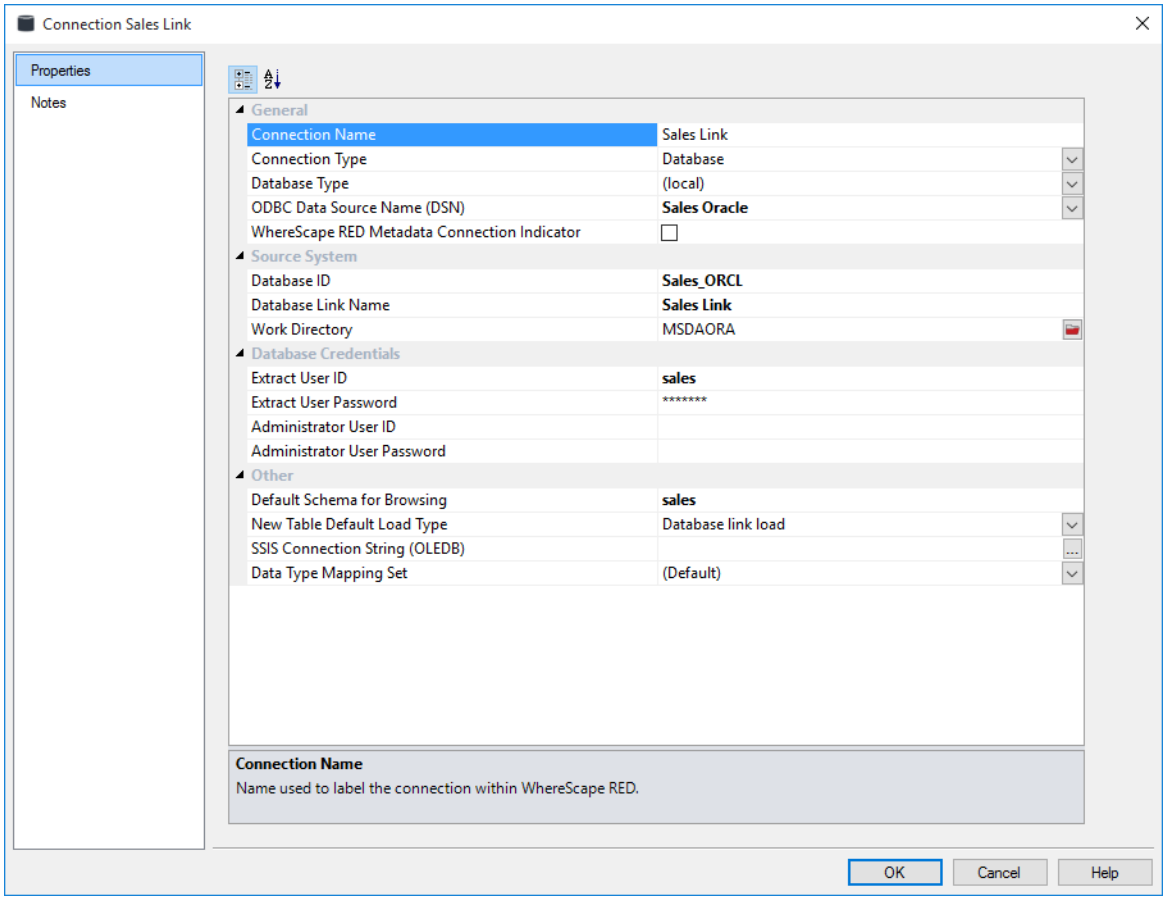
CREATING A DATABASE LINK

To create a database link connection, the database link will need to be established by clicking on the **Create Database Link** option after the relevant connection has been created in RED.
Any existing connections can be used as a database link connections if:

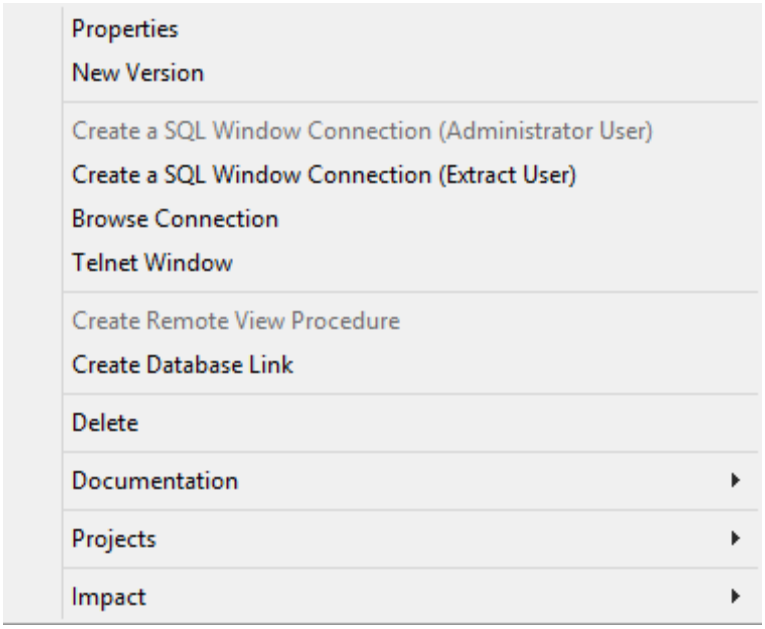
- the connection type is **Database**;
- the **Database Link Name** field is populated with **a name** for the database link;
- the **Create Database Link** option has been selected from right-clicking on the connection name menu.

To create a **new** Database Link type connection, you need to do the following:

1. Enter a **name** for the new Connection.
2. Set the Connection Type to **Database.**
3. Select an **ODBC Source (DSN)** to extract the source data.
4. Populate the **Database ID** field with a valid Oracle SID or TNS Name for **Oracle** Databases.
5. Enter a **Database Link Name** for the database link. This is a required field for establishing the **Database Link** connection.
6. Enter the **Extract User ID** and **Extract User Password** for the user that has access to the source database when not using Windows Authentication.



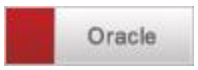
7 Once a connection has been set up, you can right-click on the connection name to see the following menu:



8 Once the **Create Database Link** option is selected, RED will attempt to create a user database link to the source database. If the link already exists, a prompt will appear asking if it is ok to overwrite the existing link.

A number of problems can occur during this action, and your database administrator should be consulted:

For **SQL Server** a trusted link will be created if the extract user in the connection is set to **dbo**. In all other cases the extract user/password will be used as the logon for the remote server.



For **Oracle** data warehouses the more common problems are:
Insufficient privileges to create the link. You need the **Create Database Link** privilege in order to create a new link.
TNS could not resolve the connection to the remote database. The database link uses the value entered in the **SID** field of the connection to identify the remote database. This value must be a valid entry in the Tnsnames table for the source database.

RESET META DATABASE CONNECTIONS

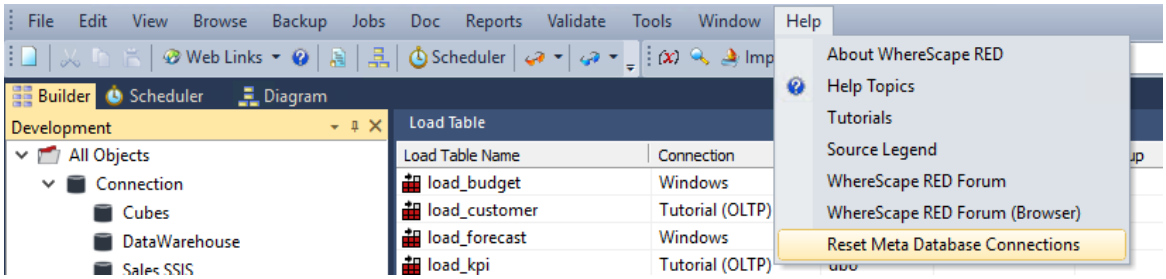
From the **Help** menu, on RED's main top bar, users can select the the **Reset Meta Database** Connections option.

This option disconnects and frees most connections that RED has to any existing ODBC connections. This option can be useful for users that are already connected to existing ODBC sources but want to alter the credentials used.
When this option is used, RED attempts to release most existing ODBC connections so the next time an ODBC Connection is used, RED will attempt to reestablish a link.

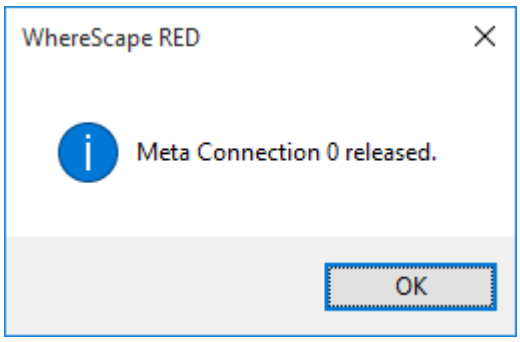
NOTE: At this stage, not all RED connections are handled via this mechanism and therefore not all connections will be reset when this option is used.

---

1 To reset meta database connections, click the **Help** menu in the main top bar and then click **Reset Meta Database Connections**.



2 Click **OK** on the following reset connection dialogs.



CONNECTION SETTINGS FOR BDA

This topic describes the required settings for connections using the WhereScape Big Data Adpater (BDA).

The WhereScape Big Data Adpater (BDA) is designed as an adapter to RED, focused on executing ELT related processing within the Hadoop/Hive eco-system.
For more information about the initial BDA setup, overview of BDA, the prerequisites and step-by-step instructions to set up BDA, please refer to section **18. BDA** of the RED Setup Administrator Guide.

The BDA connection settings are always visible on both Hive and Hadoop connection types, which include the BDA server settings.

BDA settings are also displayed for any other database connection types if there is a Hive or Hadoop connection in the Datawarehouse, but these will only have the JDBC settings displayed.

BDA enables RED to use Sqoop as load method to load data from Hive and HDFS into the Datawarehouse and also to load data into Hive as a target.

Only one BDA server connection is supported per Metadata repository.

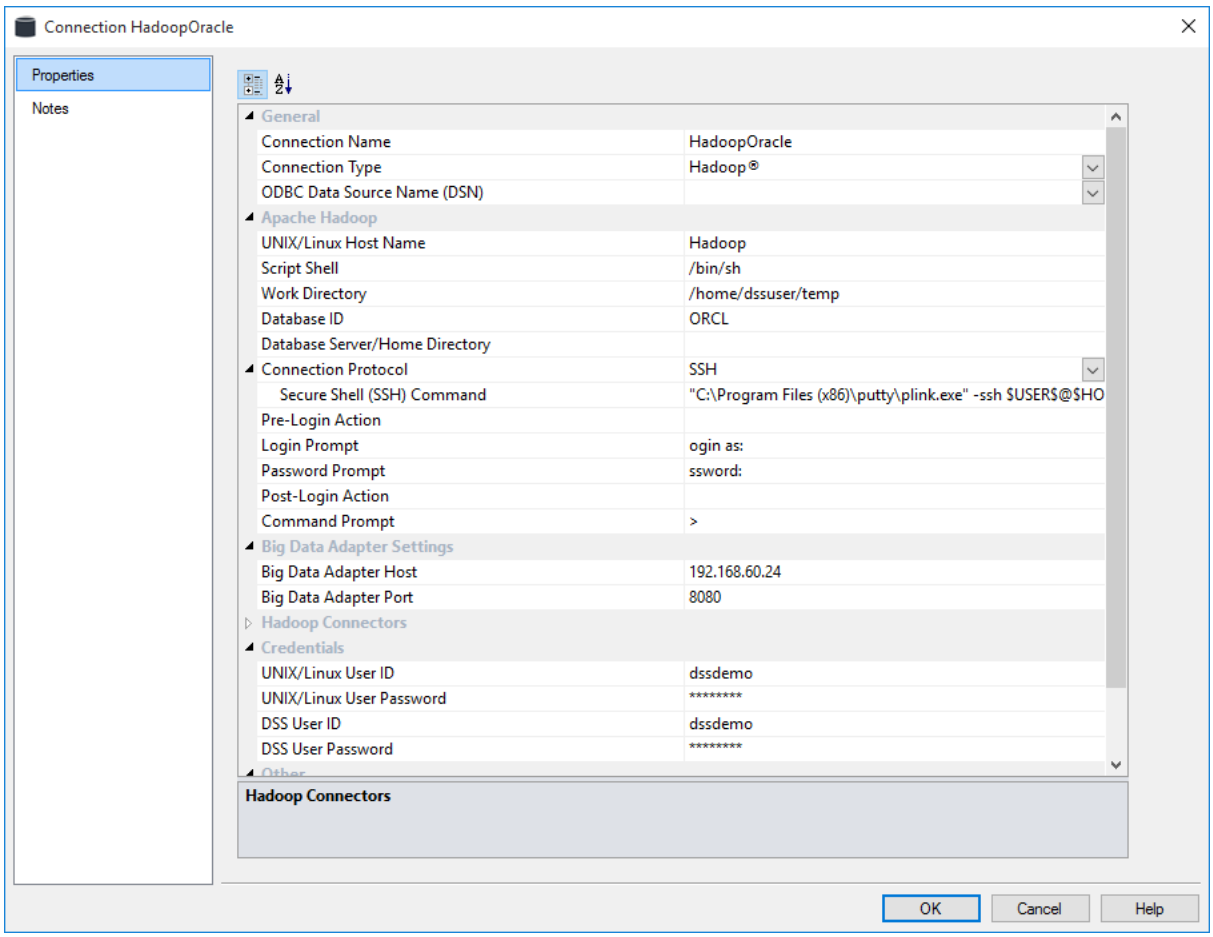
CONFIGURING THE BDA SERVER

This topic explores the configuration of the BDA server in Hadoop connections to enable loading data from Hadoop into Hive and/or Datawarehouse tables as a source database into RED Datawarehouse tables using Sqoop.

For RED to be able to load data from Hadoop into Hive and/or Datawarehouse tables and also perform loads from Hadoop directly into the Datawarehouse using Sqoop, the **Big Data Adapter** settings need to be populated in Hadoop and Hive connections.

For more information about what is required for loading data into RED using Sqoop, see the **Big Data Adapter Settings** fields description below and see also ***Connections to the Data Warehouse/Metadata Repository*** (see "***Database - Data Warehouse/Metadata Repository***" on page 143), ***Hadoop*** and ***Apache Sqoop Load***.

Hadoop connection example



Big Data Adapter Settings

Big Data Adapter Host

Host machine on which the Big Data Adapter is running its web-server.

Big Data Adapter Port

Port that Tomcat is running. Default is 8080.

CONFIGURING YOUR DATABASE FOR USE BY BDA

Connections using BDA enable loading data into Hive as a target database and they also enable loading data from Hive into Datawarehouse tables as a source, using the Apache Sqoop load method.

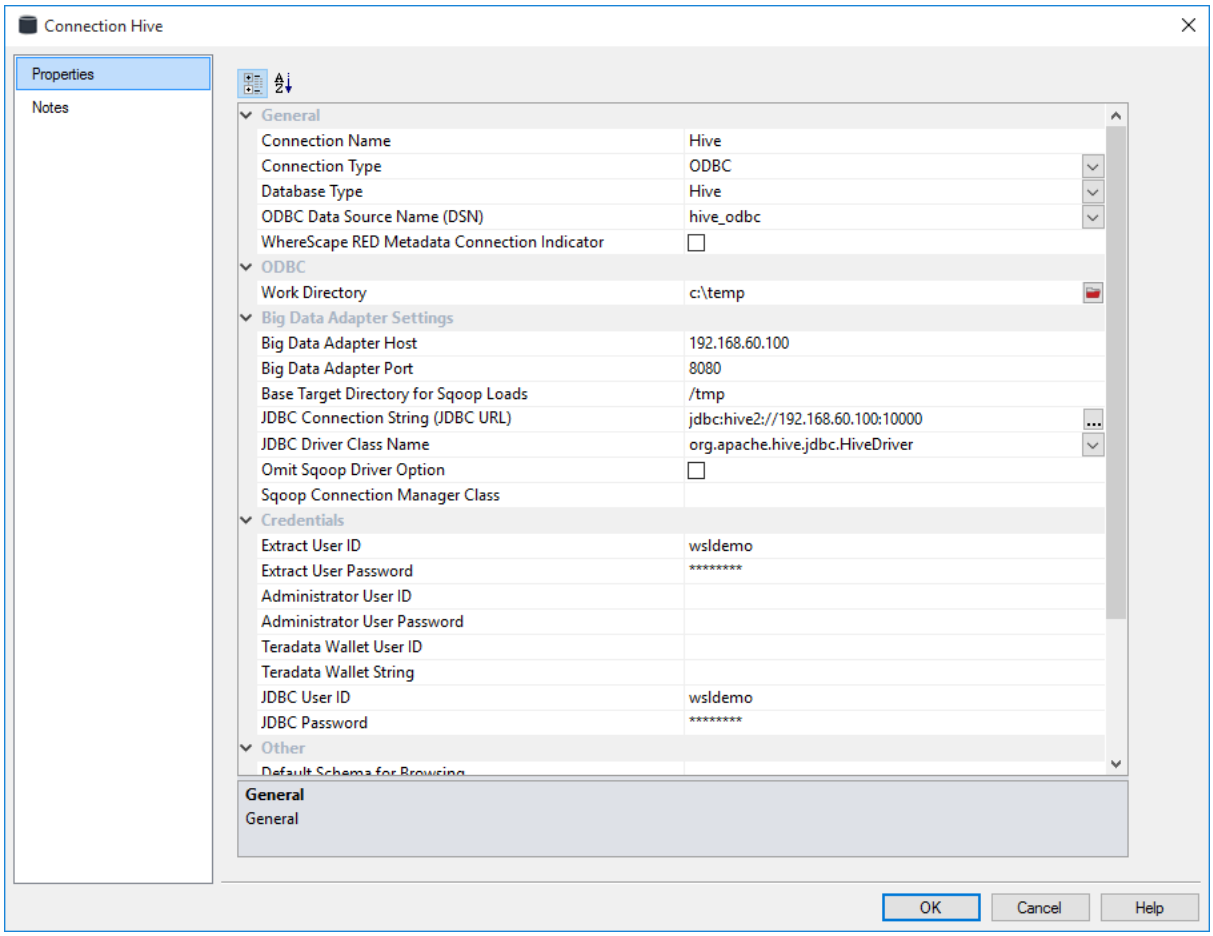
This connection type needs to include the JDBC connection string (JDBC URL) and related attributes (username, password) for the Hive database. The JDBC User and Password is usually the same as the Extract User ID but users can specify different credentials if necessary.

The Big Data Adapter settings will also need to be populated in the Datawarehouse connection. For more details, see ***Connections to the Data Warehouse/Metadata Repository*** (see "***Database - Data Warehouse/Metadata Repository***" on page 143).

RED can also load data directly into Hive from any database source. This load can also be processed via an Apache Sqoop load and the JDBC settings on the Hive connection will need to be populated. Please see the connection example and field description below for more details about this.
When loading data into Hive as a target, users can also add specific target locations in their Hive ODBC connections, if they have a Hive target license enabled.

Hive connection properties will be the same for any database sources.

Example of a Hive ODBC connection



Big Data Adapter Settings

Big Data Adapter Host

Host machine on which the Big Data Adapter is running its web-server.

Big Data Adapter Port

Port that Tomcat is running. Default is 8080.

Base Target Directory for Sqoop Loads

HDFS directory in which to create target directories for Sqoop loads using the Big Data Adapter.

JDBC Connection String (JDBC URL)

Connection string used by the WhereScape Big Data Adapter to access this database.

JDBC Driver Class Name

JDBC driver class to be used by the WhereScape Big Data Adapter. This field must be set if the JDBC URL is set.
Select the appropriate JDBC Driver class name from the drop-down list. If this is left empty this will not be specified in generated commands.

Omit Sqoop Driver Option

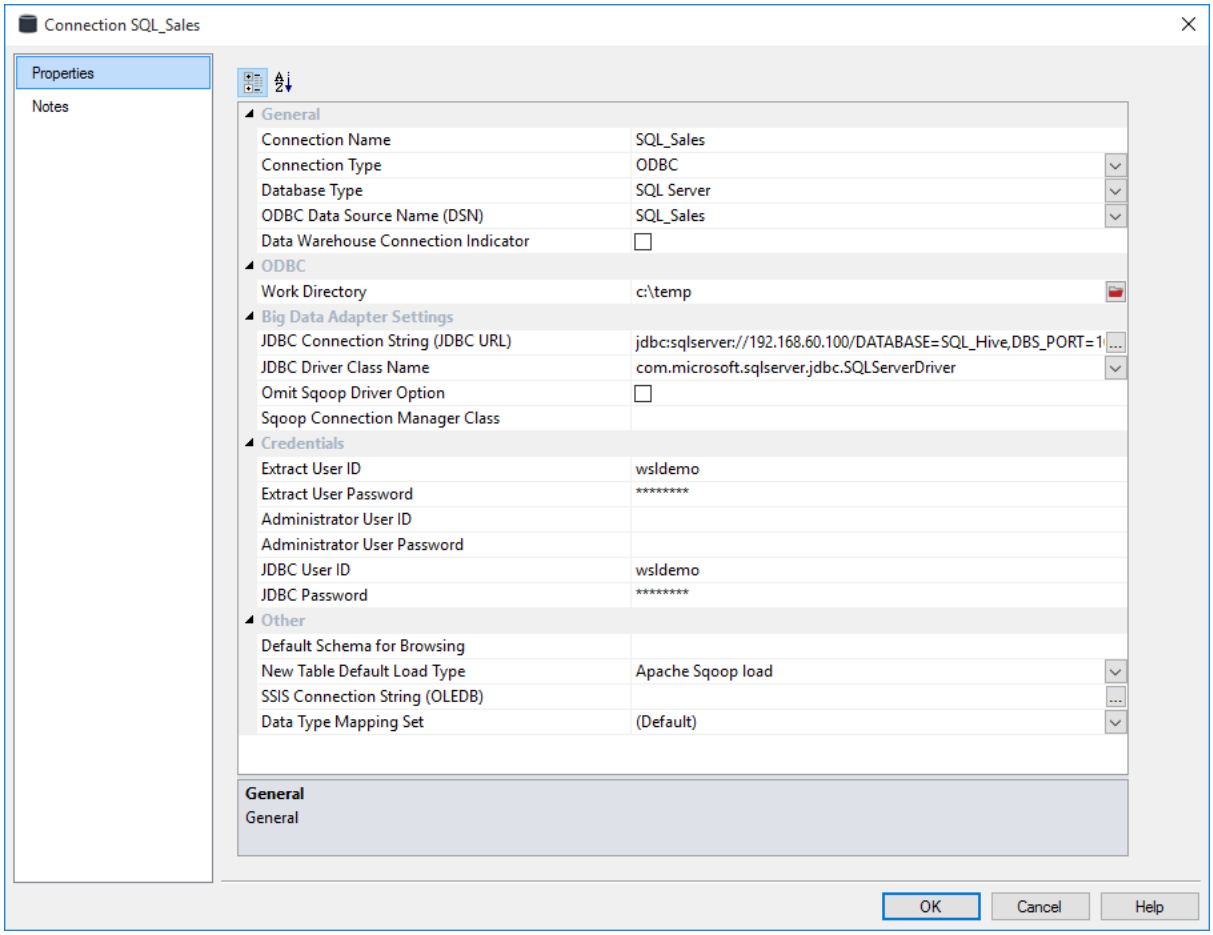
If set, the --driver option to Sqoop will be omitted. This is required for certain connection types such as Oracle connections.
If you select the **Omit Sqoop Driver Option** check-box, the driver parameter will not be used in sqoop command line. This is a requirement for Oracle at the moment, as suggested by Sqoop documentation for 1.4.5.

Sqoop Connection Manager Class

Custom Sqoop connection manager class. Corresponds to the --connection-manager command line argument. Leave blank of this is not required.

Example connection from a source database to Hive

The full JDBC connection string is:
jdbc:sqlserver://192.168.60.100/DATABASE=SQL_Hive,DBS_PORT=1025



WhereScape®

CHAPTER 7
TABLE PROPERTIES

Various properties can be set on all table objects in WhereScape RED. The screens available in the Table Properties Dialog depend on the object type selected and can be a subset of:

- ***Properties*** (on page 205)
- ***Storage***
- ***Override Create DDL*** (on page 225)
- ***Source*** (on page 227)
- ***Documentation Fields*** (on page 227)
- ***Notes*** (on page 229)
- ***Statistics*** (see "***Statistics in DB2***" on page 230) (DB2 only)

IN THIS CHAPTER

PROPERTIES

The fields available on the Properties screen depend on the Object Type selected. Specific information is available in the chapters describing each Object Type (***Load Tables*** (see "***Loading Data***" on page 232), ***Dimensions*** (on page 321), ***Stage Tables*** (see "***Staging***" on page 367), etc.).

STORAGE

The **Storage** screen of the Table Properties Dialog displays the options applicable for storing data in the associated RDBMS.

The fields available on the Storage screen depend on the RDBMS on which you are storing the data:

For a **SQL Server** example, see ***Table Storage Screen - SQL Server*** (on page 206).

For an **Oracle** example, see ***Table Storage Screen - Oracle*** (on page 208).

For a **DB2** example, see ***Table Storage Screen - DB2*** (on page 212).

For a **Greenplum** example, see ***Table Storage Screen - Greenplum***.

For a **Netezza** example, see ***Table Storage Screen - Netezza***.

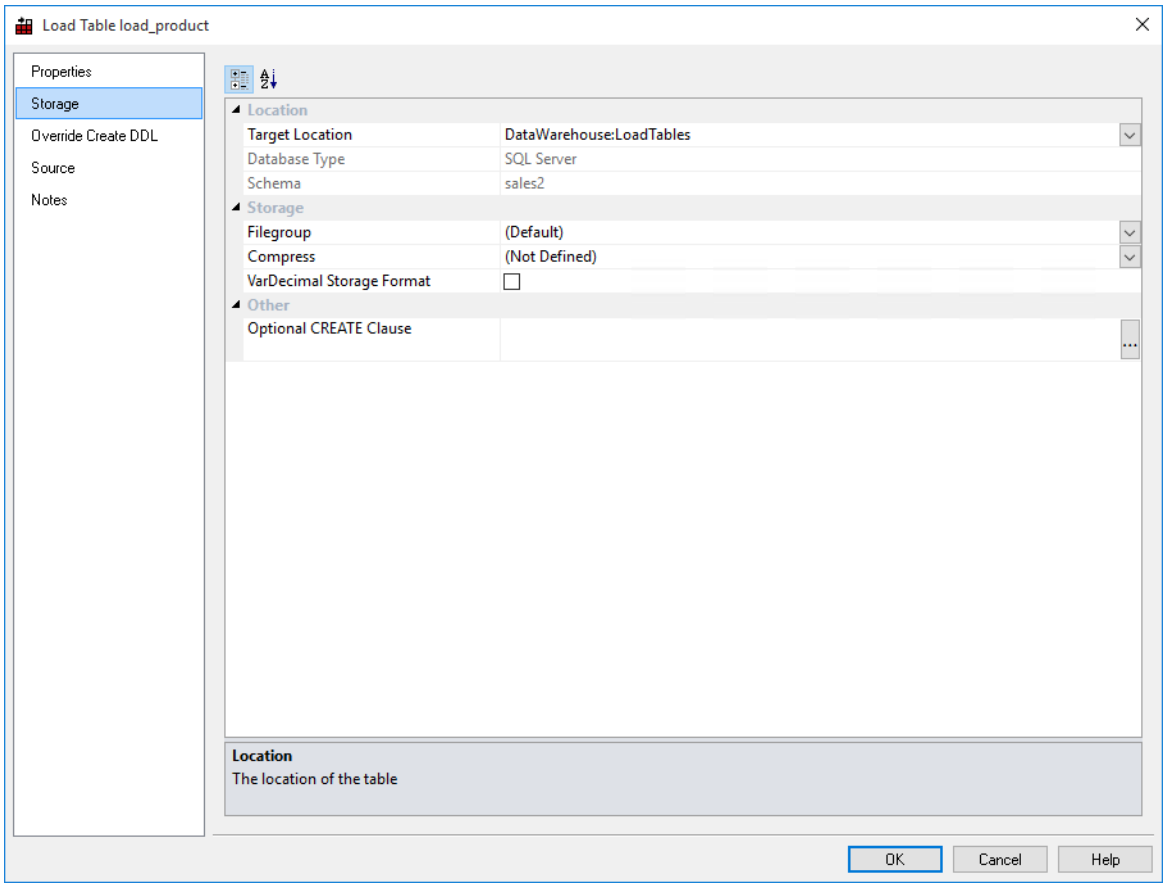
For a **PDW** example see, ***Table Storage Screen - PDW***.

For a **Tabular** example see, ***Table Storage Screen - Tabular*** (on page 222).

For information on changing storage locations for multiple tables at once see, ***Bulk Table Storage Change*** (on page 223).

TABLE STORAGE SCREEN - SQL SERVER

Typical Storage screen for a **SQL Server** Table



Location

Target Location

The target location that defines the path of the table. Select (local) for a local table or select the target schema if you are locating tables in different schemas.

To add another database/schema to the list see more details on ***Connections to the Data Warehouse/Metadata Repository*** (see "***Database - Data Warehouse/Metadata Repository***" on page 143). To set default target locations for tables see ***Settings - Storage: Target Location***.

NOTE: When upgrading from a RED version before 6.8.2.0 and moving existing objects to a target location, all procedures that reference those objects will need to be rebuilt.
Any **FROM** clauses will also need to be manually regenerated for the table references to be updated to the new [TABLEOWNER] form.

Database Type

The database type for a connection that is used for target DataWarehouse tables.

Schema

The schema where the table is located.

Create DDL Template

Optional. Specify the template to use when creating a new DDL procedure script. This option is only visible if a DDL template is available for this database type. Default value is *None*.

Storage

Filegroup

Select a name from the drop-down list box. If no filegroup name is selected for SQL Server, the default for the database is used.

Compress

This field enables you to compress a particular table/index.

Compression options are:

- NONE
- PAGE
- ROW

VarDecimal Storage Format

Enables variable numeric storage in SQL Server for the table. The default is off - no variable numeric storage. To enable variable number storage, the following command must also be run for the SQL Server database: *sp_db_vardecimal_storage_format 'DatabaseName', 'ON'*

Other

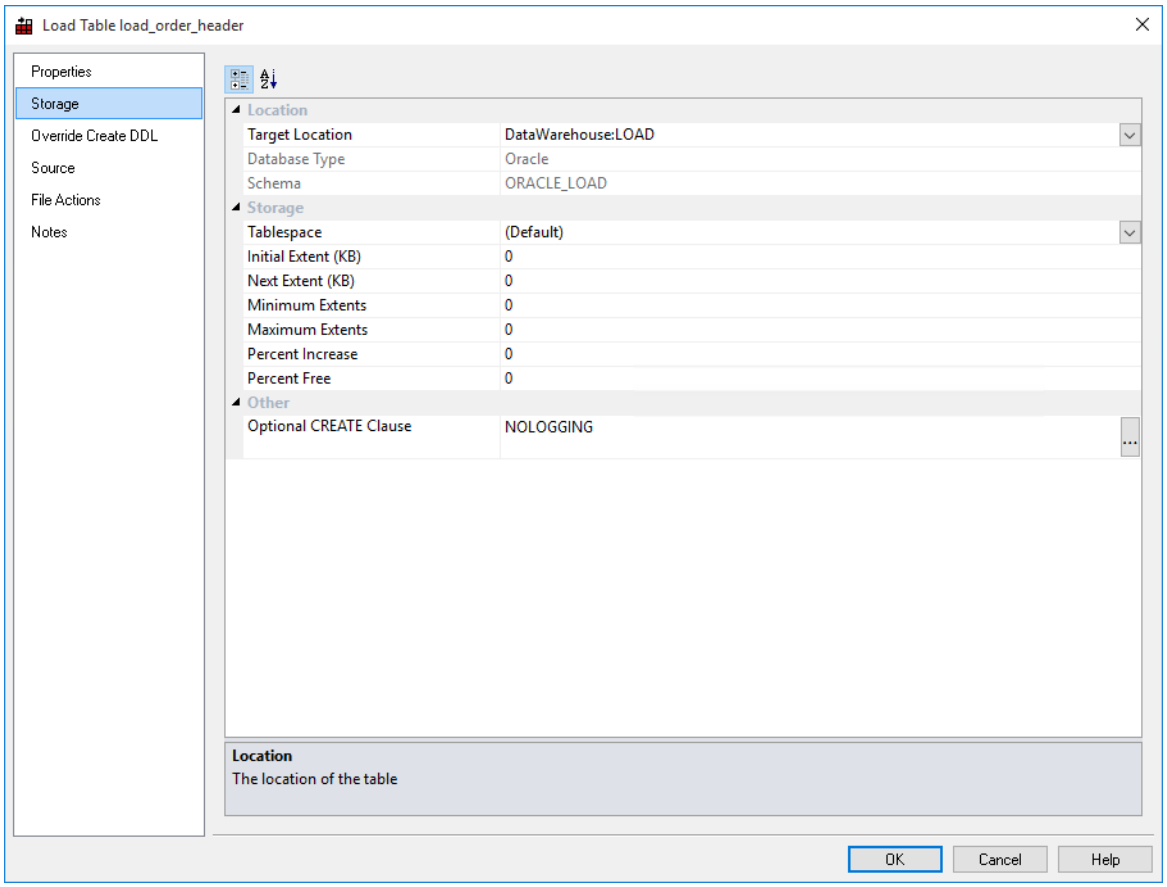
Optional CREATE Clause

Database-specific-and-compliant DDL to append to the generated CREATE TABLE statement.

TIP: To set default create clause values for all new objects, go to the **Tools/Options** menu - ***Default Optional CREATE Clause***.

TABLE STORAGE SCREEN - ORACLE

Typical Storage screen for an **Oracle** Table



Location

Target Location

The target location object that defines the path of the table. Select (local) for a local table or select the target schema if you are locating tables in different schemas.
To add another database/schema to the list see more details on ***Connections to the Data Warehouse/Metadata Repository*** (see "***Database - Data Warehouse/Metadata Repository***" on page 143). To set default target locations for tables see ***Settings - Storage: Target Location***.

Database Type

The database type for a connection that is used for target DataWarehouse tables.

Schema

The schema where the table is located, if applicable.

Create DDL Template

Optional. Specify the template to use when creating a new DDL procedure script. This option is only visible if a DDL template is available for this database type. Default value is *None*.

Storage

NOTE: When upgrading from a RED version before 6.8.2.0 and moving existing objects to a target location, all procedures that reference those objects will need to be rebuilt.
Any **FROM** clauses will also need to be manually regenerated for the table references to be updated to the new [TABLEOWNER] form.

WARNING: If the database type does not support moving tables such as Oracle, Greenplum, Netezza, Teradata and PDW, all affected tables will also need to be **manually recreated** after any storage changes.
Please note that changing the Storage for Dimension and Fact tables will need to be handled very carefully as artificial key relationships between Dimension and Fact could become out of sync. Recreating Fact Tables and large Dimension tables might take a considerable amount of time.

Tablespace

Select a name from the drop-down list box. If no tablespace is selected in Oracle, then the default tablespace for the schema is used.

Initial extent (KB)

The size of the first extent of a created database table. The default is **0** kilobytes - use the tablespace's default value. For locally managed tablespaces, this field is ignored.

Next extent (KB)

The size of the next (second) extent of a created database table, unless percent increase is specified. The default is **0** kilobytes - use the tablespace's default value. For locally managed tablespaces, this field is ignored.

Minimum extents

The number of extents the table is created with. The default is **0** - use the tablespace's default value. For locally managed tablespaces, this field is ignored.

Maximum extents

The maximum number of extents the table can have. The default is **0** - use the tablespace's default value. For locally managed tablespaces, this field is ignored.

Percent increase

The size the next extent is as a percent of the previous extent added. For example, a value of 5 will create the next extent 5% bigger than the previous extent added. The default is **0** - use the tablespace's default value. For locally managed tablespaces, this field is ignored.

Percent free

The amount of free space to leave in each extent in Oracle. The default is **0** - use the database default.

Other

Optional CREATE Clause

Database-specific-and-compliant DDL to append to the generated CREATE TABLE statement such as a logging clause and/or a partition clause.

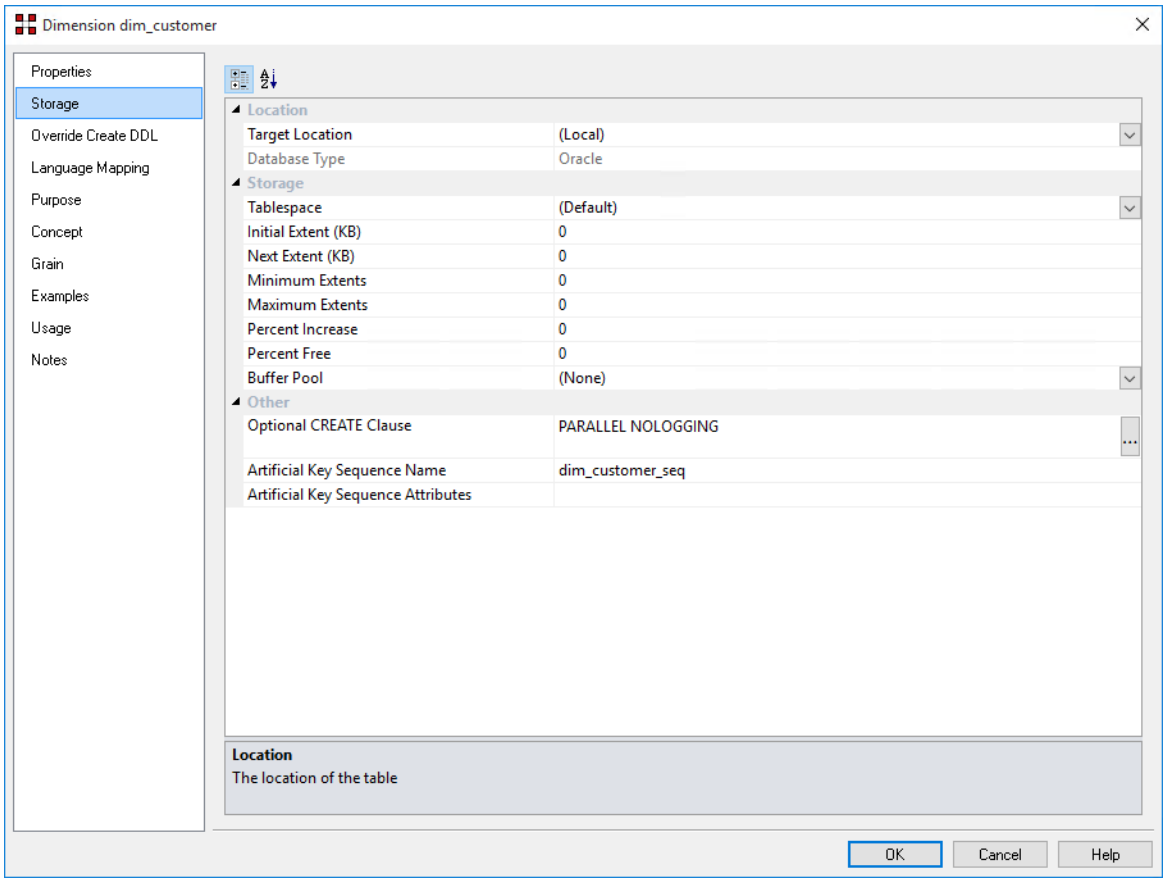
- The default values for Oracle are NOLOGGING for Load Tables and PARALLEL NOLOGGING for remaining objects.
- These values can be edited on the free text field of the **Storage** tab from every table's **Properties** screen.

TIP: To edit Oracle's default values for new objects, go to the **Tools/Options** menu - ***Default Optional CREATE Clause***.

Buffer Pool

Oracle Dimension tables only: Mark the table to be retained in a special buffer pool. Possible values are **Default**, **Keep**, **Recycle** and not set (blank). The default is not set.

DIMENSION TABLES:



Artificial Key Sequence Name

Name of the database sequence associated with the specific object. Enter the required new sequence name when creating a new table. The default key value for Dimensions is **dim_tablename_seq**.
Specific Artificial Key Sequence Names can also be added to Data Store and EDW 3NF tables when an artificial key column is added to the table.

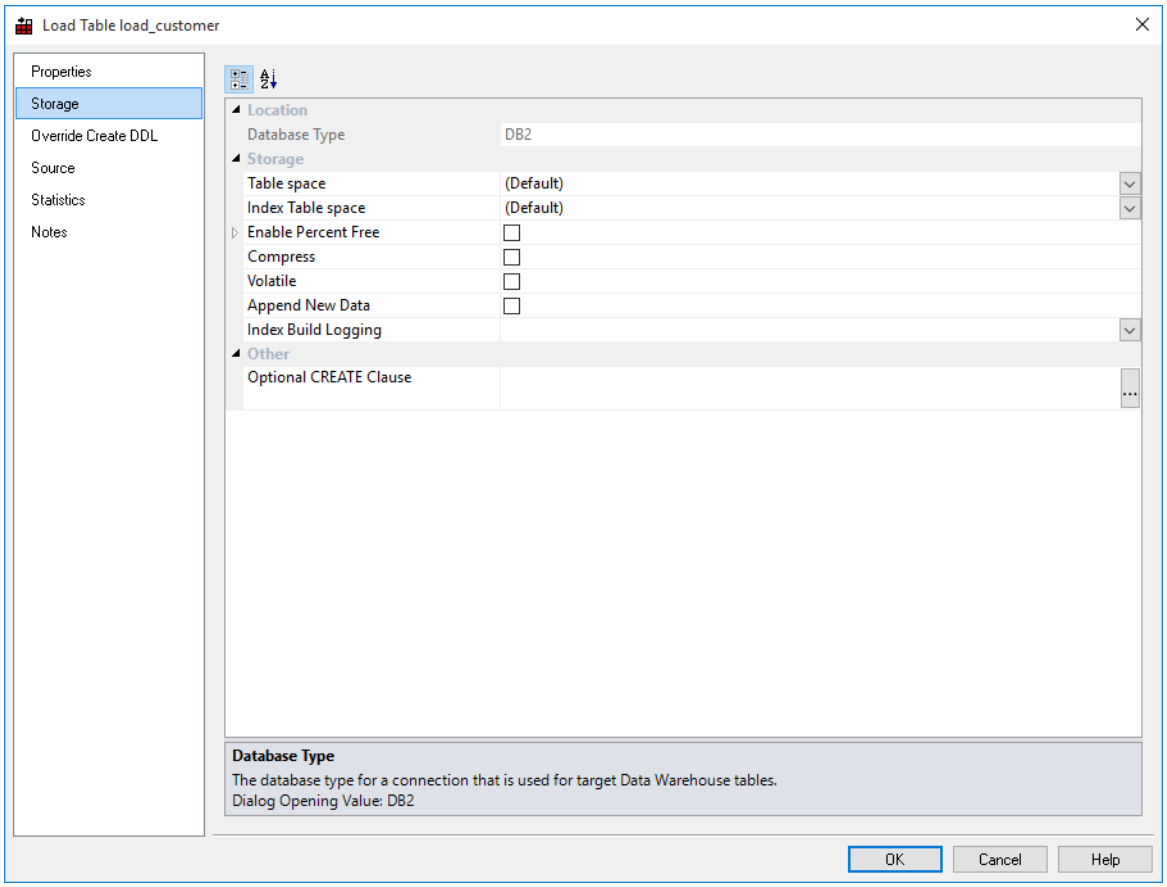
WARNING - When altering the Artificial Key Sequence Name for existing tables ensure the START WITH option is configured properly to avoid unique constraint violations.

Artificial Key Sequence Attributes

This option allows adding database specific and compliant options to the generated create sequence statement such as START WITH, INCREMENT BY or CACHE.

TABLE STORAGE SCREEN - DB2

Typical Storage screen for a **DB2** Table



Database Type

For information purposes only, this displays DB2.

Create DDL Template

Optional. Specify the template to use when creating a new DDL procedure script. This option is only visible if a DDL template is available for this database type. Default value is *None*.

Storage

Table space

Select a table space name from the drop-down list box. If no table space is selected for DB2, then the default table space for the schema is used.

Index Table space

Select an index table space from the drop-down list box. If no index table space is selected for DB2, then the default table space for the schema is used.

Enable Percent Free

When enabled, this sets the amount of free space to leave in each page in DB2. The default is disabled, which uses the database default.

Compress

Enable table compression. Default is off - no compression.

Volatile

Enable this option to alter the table after it is created to indicate size volatility to the DB2 optimizer.

Append New Data

Enable this option to alter the table after it is created to append new data to the end of the table, rather than using available free space in the table.

Index Build Logging

Specifies the degree of logging performed by DB2 when creating, recreating and re-organizing indexes. Value is one of not selected, **NULL**, **OFF** or **ON**. Not selected and NULL are the same. The table is created without this option and then altered to add this property. The default is not set (no alter takes place). The following describes the values:

- NULL - Database parameter **logindexbuild** determines whether logging is minimal or full.
- OFF - only minimal logging occurs.
- ON - all operations are logged.

Other

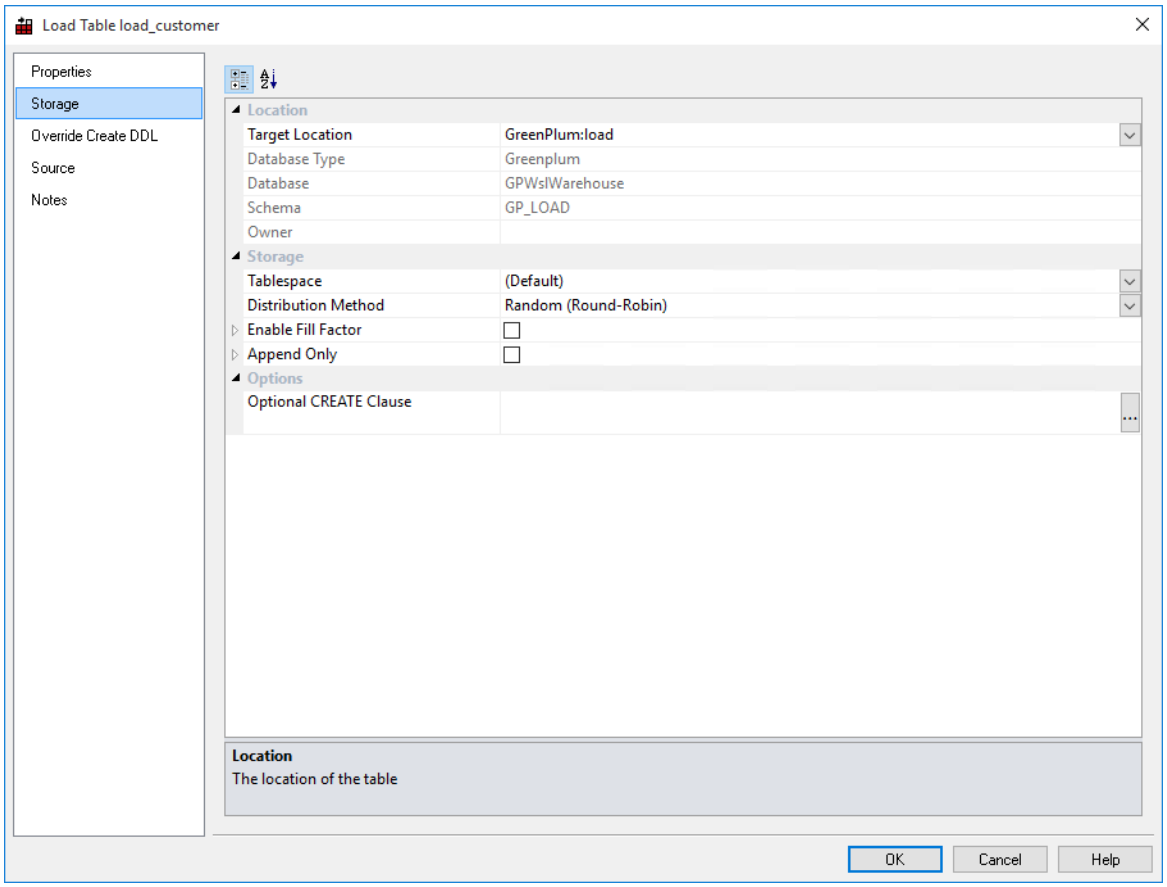
Optional CREATE Clause

Database-specific-and-compliant DDL to append to the generated CREATE TABLE statement. Database-specific-and-compliant DDL to append to the generated CREATE TABLE statement.

TIP: To set default create clause values for new objects, go to the **Tools/Options** menu - ***Default Optional CREATE Clause***.

TABLE STORAGE SCREEN - GREENPLUM

Typical **Storage** screen for a **Greenplum** table



Location

Target Location

The target location that defines the path of the table. Select (local) for a local table or select the target schema if you are locating tables in different schemas.

To add another database/schema to the list see more details on ***Connections - Database*** or ***Connections - ODBC***. To set default target locations for tables see ***Settings - Storage: Target Location***.

NOTE: When upgrading from a RED version before 6.8.2.0 and moving existing objects to a target location, all procedures that reference those objects will need to be rebuilt.
Any **FROM** clauses will also need to be manually regenerated for the table references to be updated to the new [TABLEOWNER] form.

WARNING: If the database type does not support moving tables such as Oracle, Greenplum, Netezza, Teradata and PDW, all affected tables will also need to be **manually recreated** after any storage changes.
Please note that changing the Storage for Dimension and Fact tables will need to be handled very carefully as artificial key relationships between Dimension and Fact could become out of sync. Recreating Fact Tables and large Dimension tables might take a considerable amount of time.

Database Type

The database type for a connection that is used for target DataWarehouse tables.

Database

The database where the table is located. Leave blank to use the default for the connection or local environment.

Schema

The schema where the table is located.

Owner

Owner to which the object is altered.

Create DDL Template

Optional. Specify the template to use when creating a new DDL procedure script. This option is only visible if a DDL template is available for this database type. Default value is *None*.

Storage

Tablespace

Default tablespace of the table that determines the storage location it is created in. Select (Default) to use the default.

Distribution Method

Select the Data Distribution Method. Options are Hash and Random (Round-Robin)

Enable Fill Factor

Setting this will enable setting of the Fill Factor.

Fill Factor

Percentage [10-100] between 10 and 100. 100 (complete packing) is the default. When a small fill factor is specified, INSERT operations pack table pages only to the indicated percentage. The remaining space on each page is reserved for updating rows on that page.

Append only

When set, will create the table as append-only. If this field is not set, the table will be created as a heap-storage table.

Orientation

Columns selected to make up the Distribution Key.

Compression Type

Table-level compression is applied to an entire table.

Compression level

Table-level compression is applied to an entire table.

Other

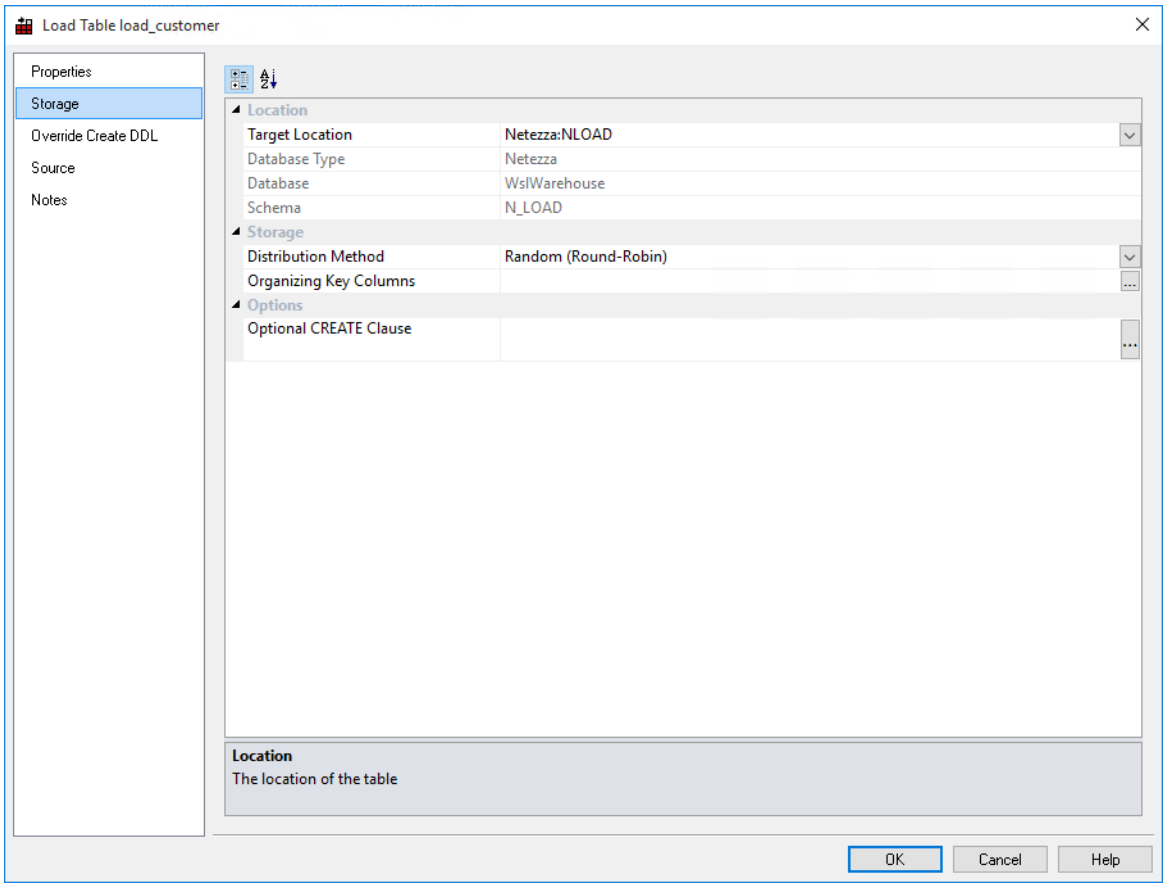
Optional CREATE Clause

Database-specific-and-compliant DDL to append to the generated CREATE TABLE statement.

 **TIP:** To set default create clause values for all new objects, go to the **Tools/Options** menu - ***Default Optional CREATE Clause***.

TABLE STORAGE SCREEN - NETEZZA

Typical **Storage** screen for a **Netezza** table:



Location

Target Location

The target location that defines the path of the table. Select (local) for a local table or select the target schema if you are locating tables in different schemas.

To add another database/schema to the list see more details on ***Connections - Database*** or ***Connections - ODBC***. To set default target locations for tables see ***Settings - Storage: Target Location***.

NOTE: When upgrading from a RED version before 6.8.2.0 and moving existing objects to a target location, all procedures that reference those objects will need to be rebuilt.
Any **FROM** clauses will also need to be manually regenerated for the table references to be updated to the new [TABLEOWNER] form.

WARNING: If the database type does not support moving tables such as Oracle, Greenplum, Netezza, Teradata and PDW, all affected tables will also need to be **manually recreated** after any storage changes.
Please note that changing the Storage for Dimension and Fact tables will need to be handled very carefully as artificial key relationships between Dimension and Fact could become out of sync. Recreating Fact Tables and large Dimension tables might take a considerable amount of time.

Database Type

The database type for a connection that is used for target DataWarehouse tables.

Database

The database where the table is located. Leave blank to use the default for the connection or local environment.

Schema

The schema where the table is located.

Owner

Owner to which the object is altered.

Create DDL Template

Optional. Specify the template to use when creating a new DDL procedure script. This option is only visible if a DDL template is available for this database type. Default value is *None*.

Storage

Distribution Method

Select the Data Distribution Method. Options are Hash, Random (Round-Robin) and Netezza Designated.

Other

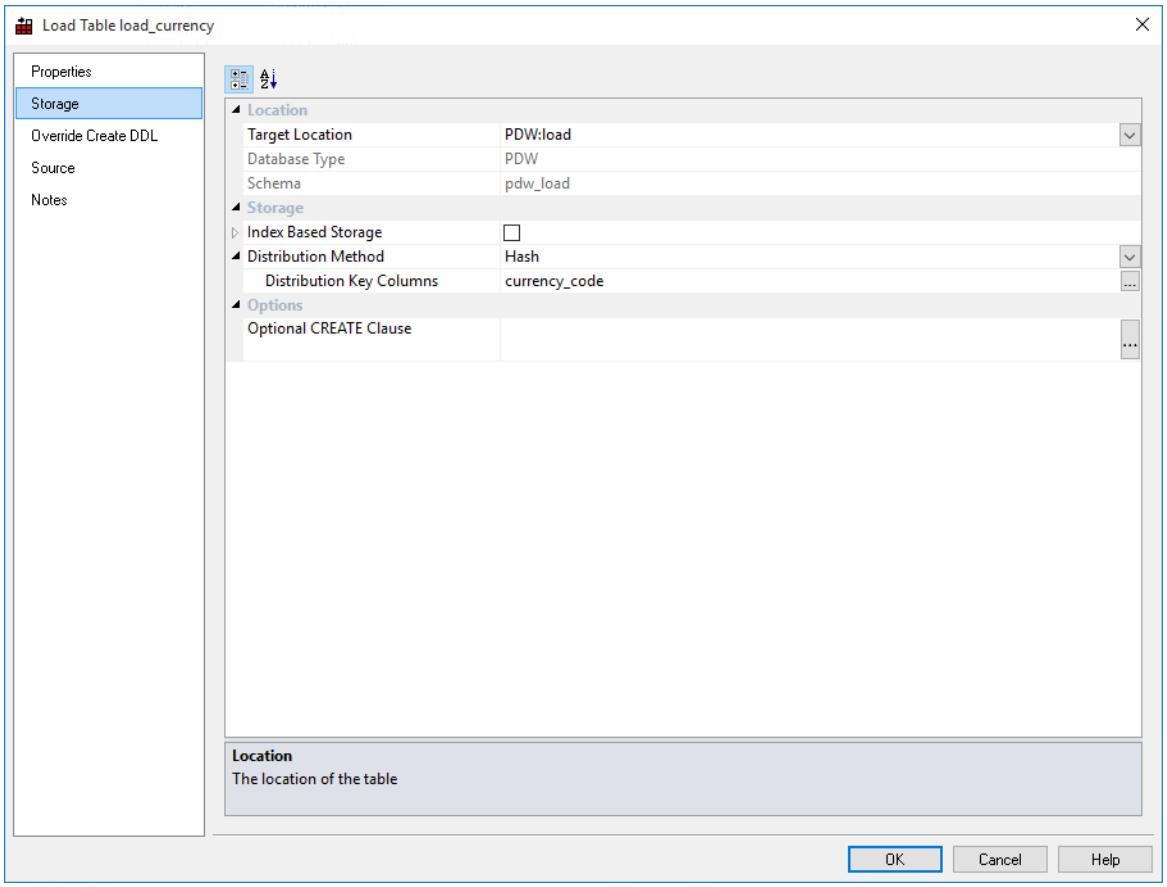
Optional CREATE Clause

Database-specific-and-compliant DDL to append to the generated CREATE TABLE statement.

 **TIP:** To set default create clause values for all new objects, go to the **Tools/Options** menu - ***Default Optional CREATE Clause***.

TABLE STORAGE SCREEN - PDW

Typical **Storage** screen for a **PDW** table:



Location

Target Location

The target location that defines the path of the table. Select (local) for a local table or select the target schema if you are locating tables in different schemas.

To add another database/schema to the list see more details on ***Connections - Database*** or ***Connections - ODBC***. To set default target locations for tables see ***Settings - Storage: Target Location***.

NOTE: When upgrading from a RED version before 6.8.2.0 and moving existing objects to a target location, all procedures that reference those objects will need to be rebuilt.
Any **FROM** clauses will also need to be manually regenerated for the table references to be updated to the new [TABLEOWNER] form.

WARNING: If the database type does not support moving tables such as Oracle, Greenplum, Netezza, Teradata and PDW, all affected tables will also need to be **manually recreated** after any storage changes.
Please note that changing the Storage for Dimension and Fact tables will need to be handled very carefully as artificial key relationships between Dimension and Fact could become out of sync. Recreating Fact Tables and large Dimension tables might take a considerable amount of time.

Database Type

The database type for a connection that is used for target DataWarehouse tables.

Schema

The schema where the table is located.

Create DDL Template

Optional. Specify the template to use when creating a new DDL procedure script. This option is only visible if a DDL template is available for this database type. Default value is *None*.

Storage

Index Based Storage

Allows setting of table data to be stored as either Clustered or Clustered Columnstore Index.

Index Type

Select the Index Type. Options are Clustered Columnstore Index or Clustered Index.

Distribution Method

Select the Data Distribution Method. Options are Hash, Replicate or Round_Robin.

Other

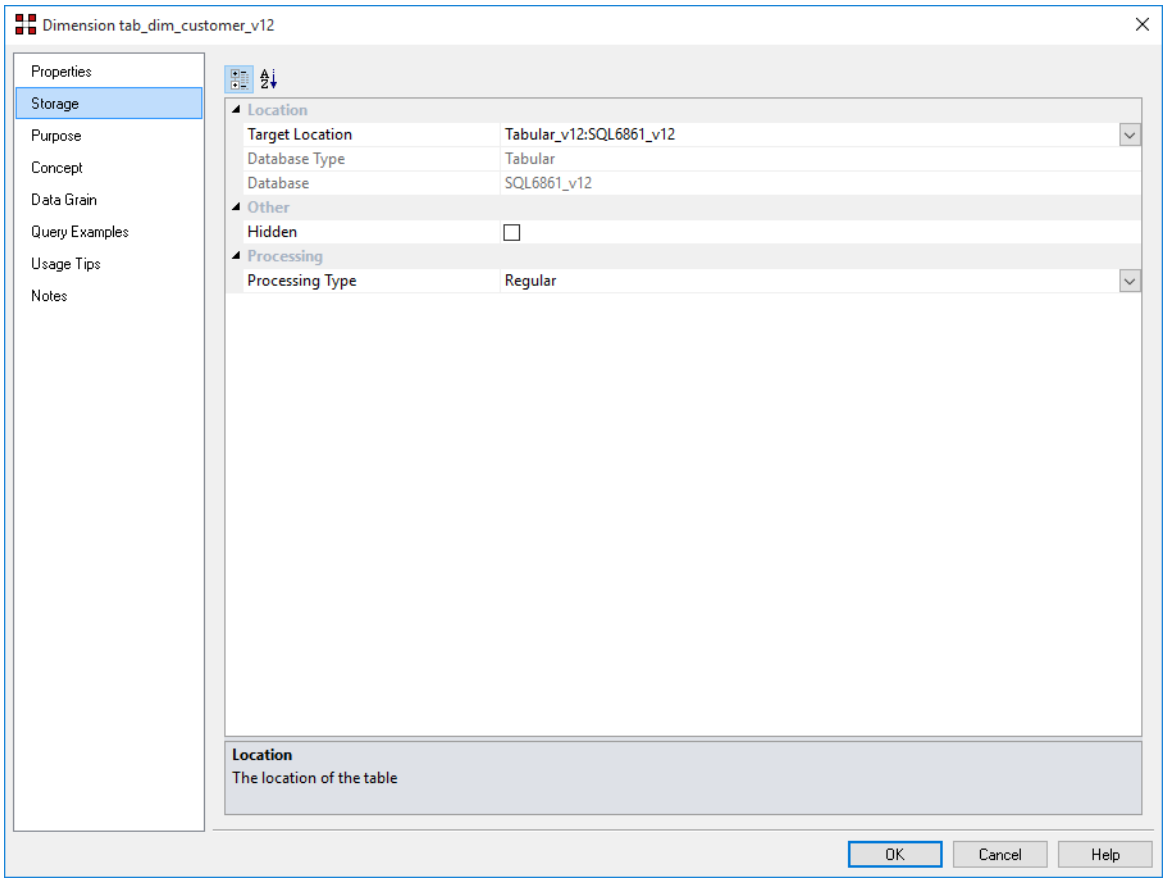
Optional CREATE Clause

Database-specific-and-compliant DDL to append to the generated CREATE TABLE statement.

TIP: To set default create clause values for all new objects, go to the **Tools/Options** menu - ***Default Optional CREATE Clause***.

TABLE STORAGE SCREEN - TABULAR

Typical Storage screen for a **Tabular** Table:



Location

Target Location

The path for the MSAS target. To add another MSAS target see ***Microsoft Analysis Server 2005+ - Tabular Mode*** (on page 187). To set default target locations for tables see ***Settings - Storage: Target Location***.

Database Type

For information purposes only, this displays Tabular, for MSAS Tabular targets.

Database

For information purposes only, this displays the database name.

Other

Hidden

Specifies whether the table is hidden from reporting client field lists.

Processing

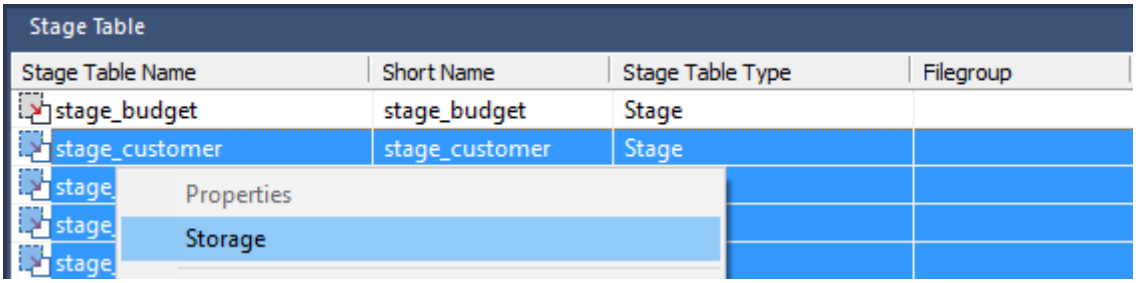
Processing Type

Set the database reporting tools processing to Regular or LazyAggregations in the target Analysis Services database.

BULK TABLE STORAGE CHANGE

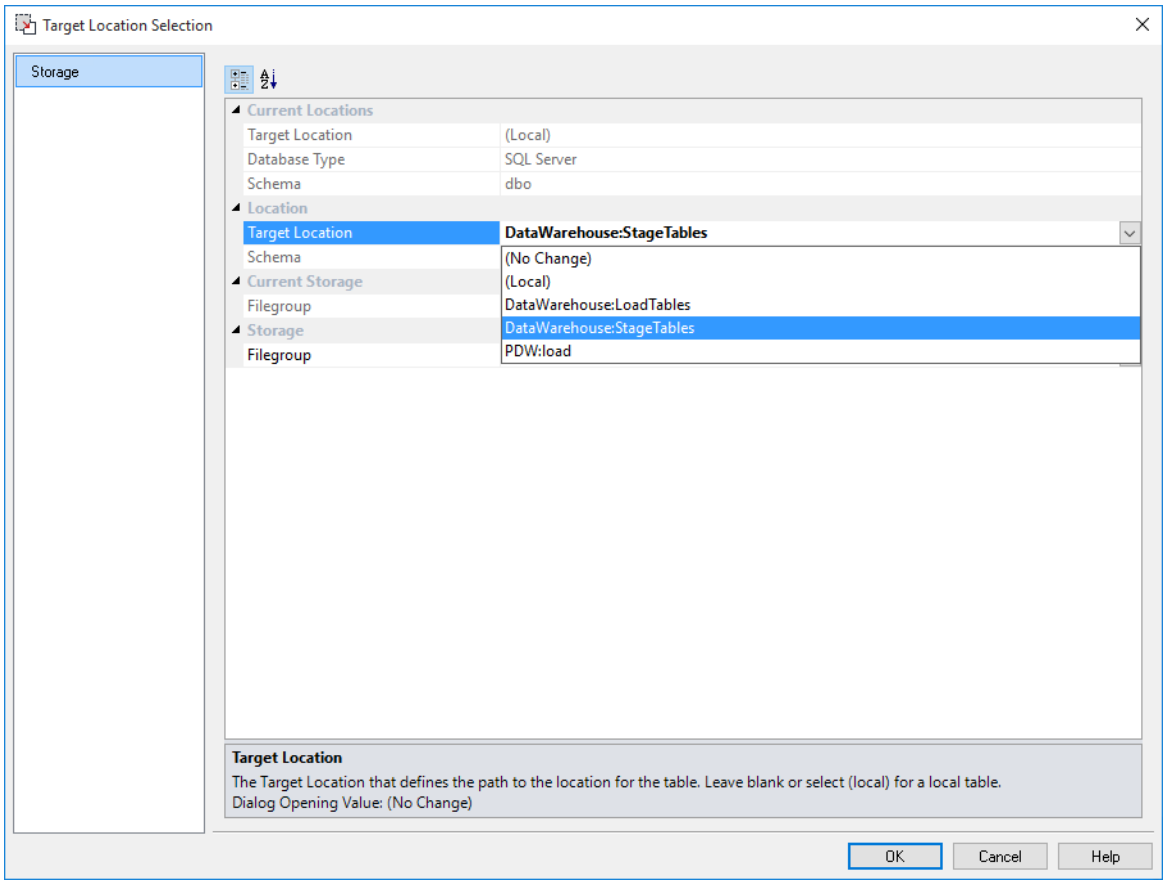
Table Storage locations can be changed through the Storage tab on a table's Properties dialog but they can also be changed in bulk by using the following process:

1. Double-click on the desired object group in the left pane. This will display all the tables in that group in the middle pane.
2. Select the tables that you wish to change the storage for using standard Windows selection.
3. Right-click to bring up a menu and select **Storage**.



4 On the Target Schema Location Selection dialog, select the desired **Target Location** for the schema change.

- Select the new Target location to change all the selected load tables in bulk on the **Target** drop-down list.
- Select any new required Filegroups in the **Filegroup** drop-down list in Oracle and SQL Server databases.



5 Follow the next dialogs to complete the bulk storage change.
Please note that **all procedures** from the affected tables will need to be **manually changed or regenerated** after a bulk storage change.

6 If the database type does not support moving tables such as **Oracle, Greenplum, Netezza, Teradata** and **PDW**, all affected tables will also need to be **manually recreated** after any storage changes.

WARNING: Please note that changing the Storage for Dimension and Fact tables will need to be handled very carefully as artificial key relationships between Dimension and Fact could become out of sync.
Recreating Fact Tables and large Dimension tables might take a considerable amount of time.

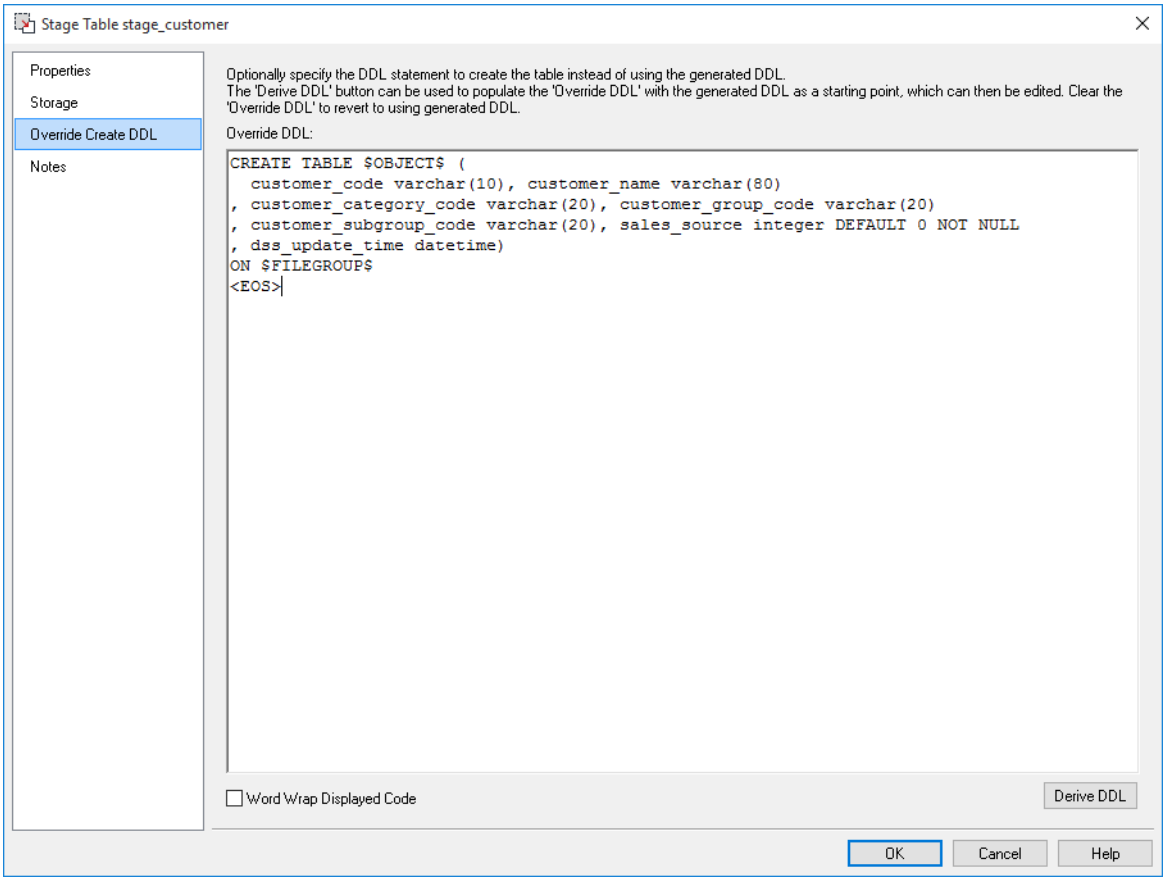
OVERRIDE CREATE DDL

WhereScape RED allows to create tables using a specific DDL statement instead of the RED generated DDL.
You can override and edit the created DDL on load, stage, data store, EDW 3NF, dimension, fact, aggregate, user defined view and retro table's **Override Create DDL** tab.

Creating a table using a specific DDL statement:

1 If you are creating a **new** table, drag and drop the table from the right pane to the middle pane.
- On the Properties dialog, click on the **Override Create DDL tab**.
- Click on the **Derive DDL** button at the bottom of the dialog to get the generated DDL as your starting point.
- Edit or clear the generated DDL and enter the desired DDL.

2 If you are **editing** an existing table, double click on the table object on the left pane to open the Properties dialog.
- Click on the **Override Create DDL** tab.
- Click on the **Derive DDL** button at the bottom of the dialog to get the generated DDL as your starting point.
- Edit or clear the generated DDL and enter the desired DDL.



3 The $OBJECT$ reference is auto translated to the fully qualified (schema/db.table name) at runtime.

For **SQL Server** and **Oracle** respectively, $FILEGROUP$ and $TABLESPACE$ get replaced at runtime with the storage defined values.

For **DB2**, $TABLESPACE$ and $INDEXSPACE$ get replaced at runtime with the storage defined values.

For Greenplum, OWNER gets replaced at runtime with the storage defined values.

4 The "end of statement" indicator is <EOS> by default but can be configured in **Tools/Options/Code Generation/General**.

 **TIP:** To revert to RED deriving the original generated DDL at runtime, leave the Override DDL box blank or clear out the contents.

Clicking the Derive DDL *button* pops-up a warning message asking if you want to replace the current DDL definition with new DDL.

SOURCE

The Source tab is only available for Load Tables. More information is available in the ***Loading Data*** (on page 232) chapter, or more specifically:

- ***Database Link Load - Source Screen*** (see "***Database Link Load - Source Mapping***" on page 240)
- ***SSIS Loads - Source Screen***
- ***Native ODBC Based Load - Source Screen*** (see "***Native ODBC Based Load - Source Mapping***" on page 266)
- ***Flat File Load - Source Screen***

DOCUMENTATION FIELDS

Each table has a set of associated documentation fields. These are used to store descriptive metadata about the tables and how they are used.
The contents of each field appear in the generated documentation. Their order and placement can be customized in the generated documentation using the **Tools/Options** menu.

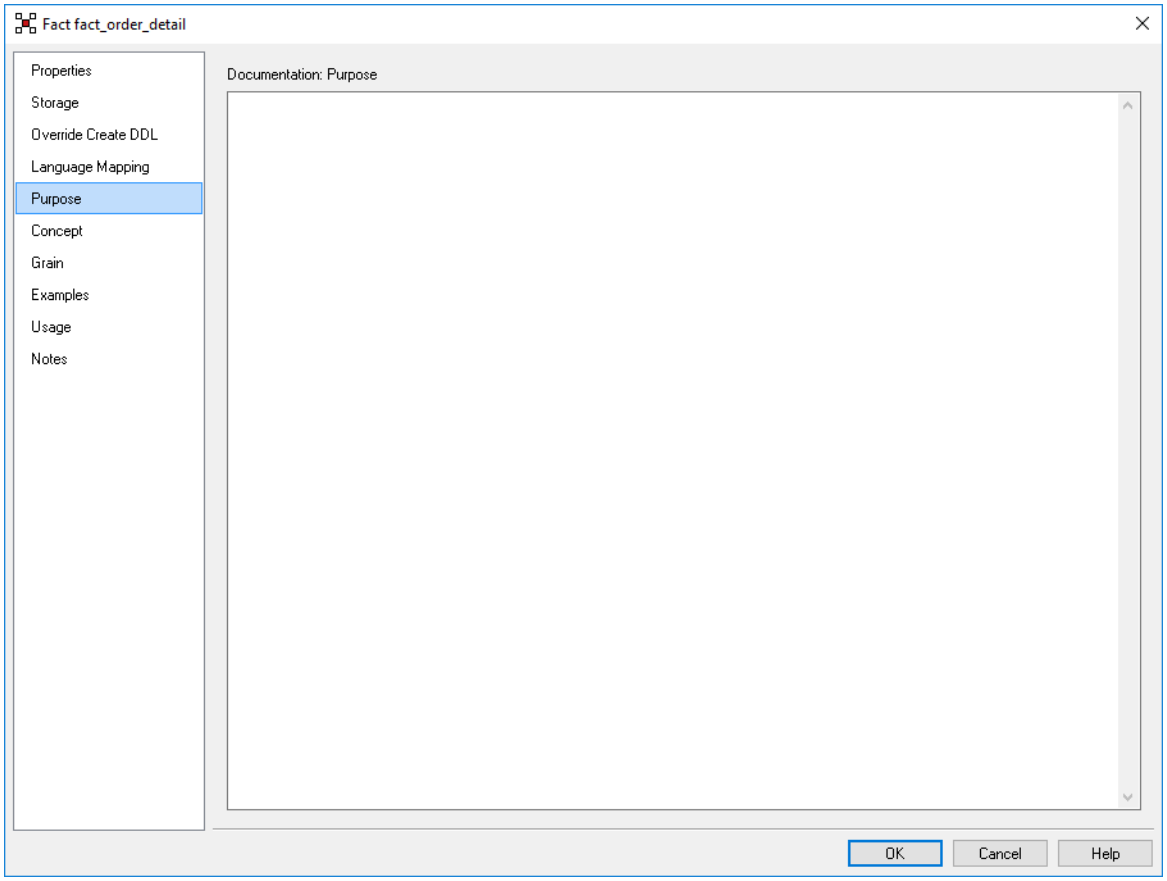
The following fields are available to store additional informational metadata that will appear in the generated WhereScape RED documentation:

- Purpose
- Concept
- Grain
- Examples
- Usage

NOTE: The names of these fields are completely arbitrary and can be changed in the generated documentation in the **Tools/Options/Documentation** menu.
By default, these fields are not enabled for load and stage tables, but can be enabled using the **Tools/Options/Documentation** menu.

Up to 4000 characters of information can be stored in each field.
Some or all of these fields can be removed from the documentation via the **Tools/Options/Documentation** menu.

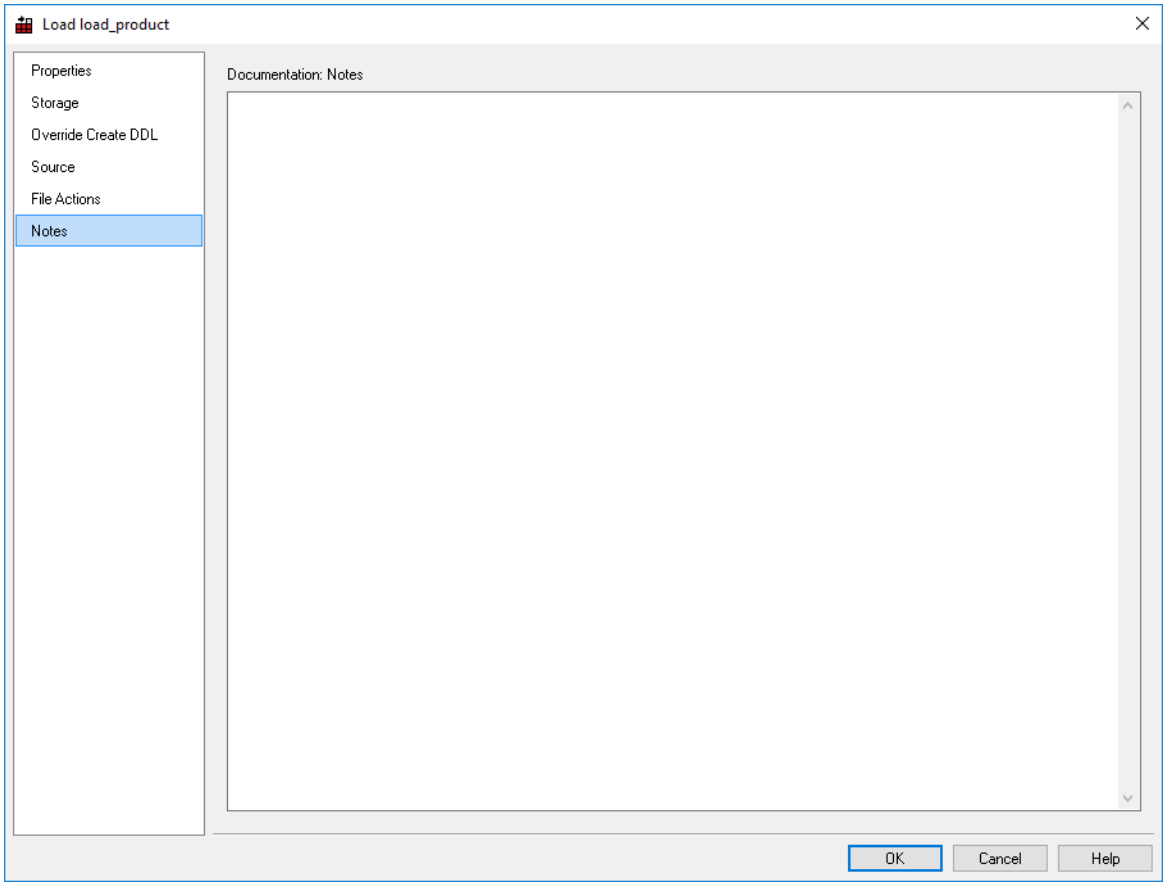
DOCUMENTATION FIELDS SCREEN



NOTES

The Notes screen is used to enter notes against WhereScape RED objects.
The notes on an object are included in the WhereScape RED documentation. Up to 4000 characters of information can be stored in the Notes field.
By default, the Notes field is enabled for all object types.
The Notes field can be removed from the documentation via the **Tools/Options/Documentation** menu.

NOTES SCREEN



STATISTICS IN DB2

WhereScape RED for DB2 allows for table level statistics definition. Four different statistics commands may be defined for a table:

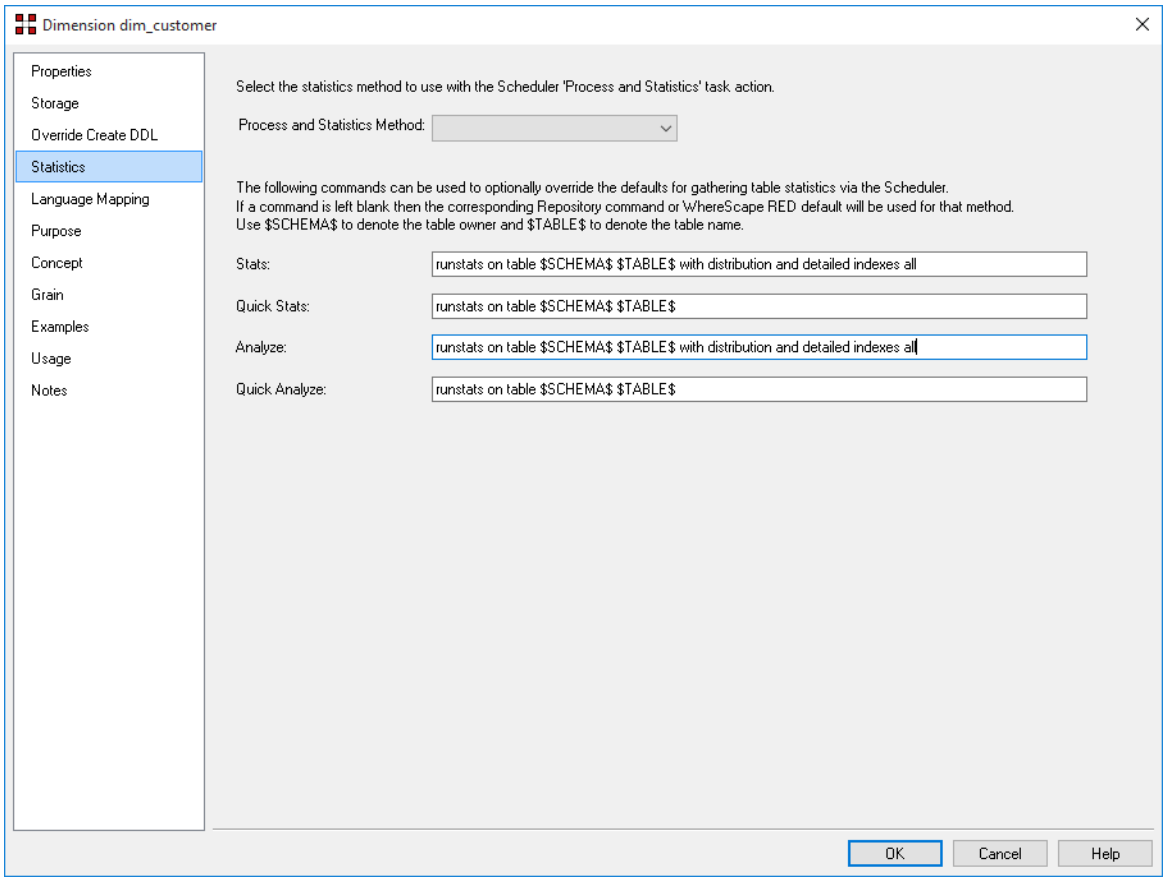
- Stats Full
- Stats Quick
- Analyze Full
- Analyze Quick

These are arbitrary names, used as **actions** in the scheduler.
To set the command that is run for one of the actions above, enter the command to be run using the $SCHEMA$ and $TABLE$ variables.

Note: The variable $SCHEMA$ allows for metadata portability between environments. Variable $TABLE$ avoids any requirement to change the command if the table is renamed.

For example: enter *runstats on table $SCHEMA$.$TABLE$* to do basic statistics for the table. If a particular statistics field is left blank, then the WhereScape RED default statistics options are used.
The default statistics method is the statistics method used in the scheduler if the chosen action is **Process with default stats**.

STATISTICS SCREEN



This screen shot shows the default values; but if these fields are left blank, then the default values will be used.

WhereScape®

CHAPTER 8
LOADING DATA

Load tables are the first entry point for all information coming into the data warehouse. There are multiple ways of getting data into load tables.

- Database link load - loaded from another database
- Externally loaded - the load table is populated by some external process, e.g. an ETL (Extract, Transform and Load) tool or EAI (Enterprise Application Integration) tool, putting the data directly into the load tables.
- ODBC based load - the data is loaded via an ODBC connection, either directly by reading and inserting each row (a Load Type of **ODBC load**), or via a file by reading from the source system using ODBC and writing to a file and then loading the file (a Load Type of **Native ODBC**). ODBC connections require a Windows scheduler.
- File load - a flat file load where most of the processing is managed and controlled by the scheduler.
- Script-based load - a flat file load where a host system, i.e. UNIX or Windows script file is executed to perform the load.
- XML file load - loading an XML file from a Windows directory.

IN THIS CHAPTER

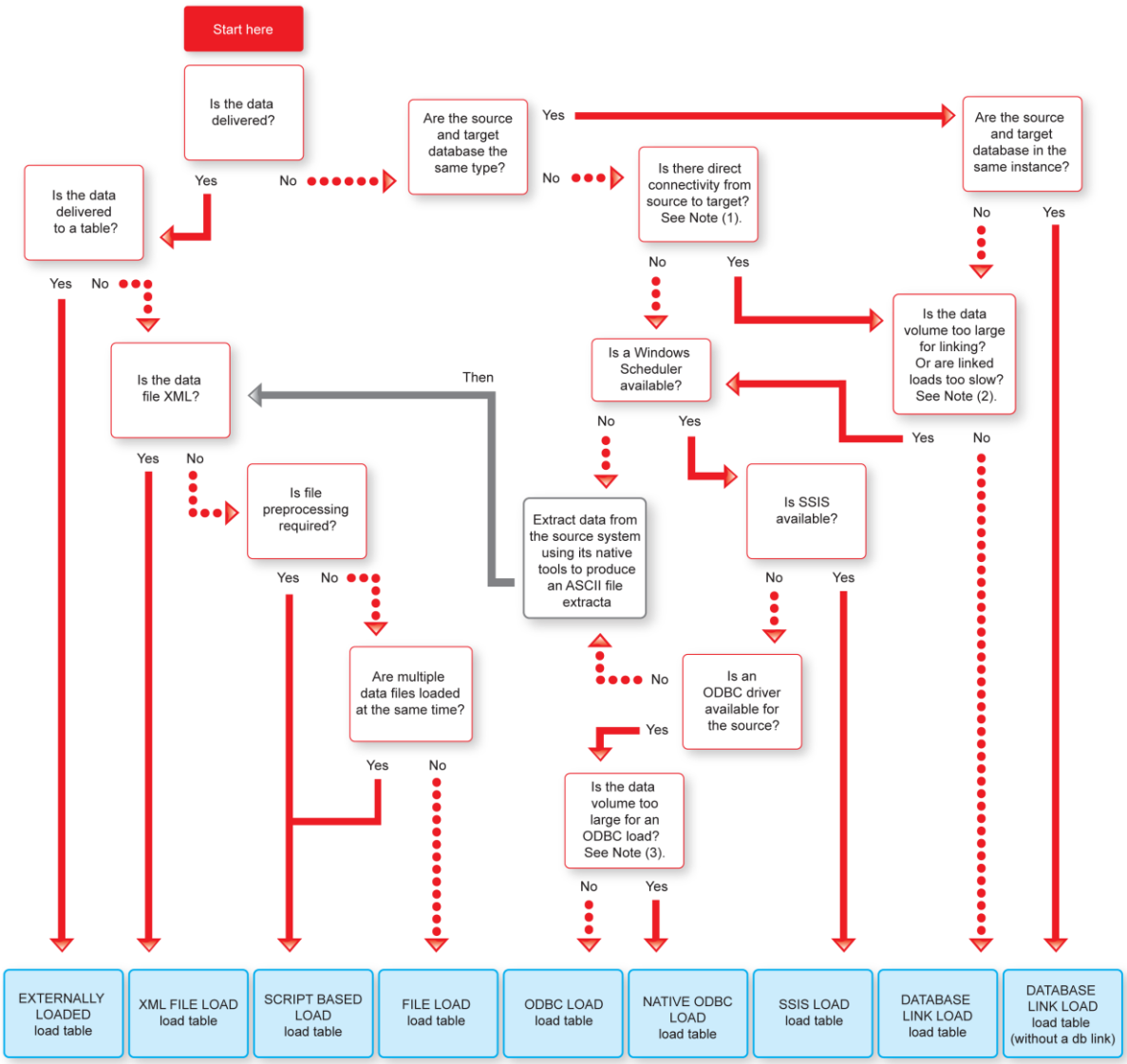
CHOOSING THE BEST LOAD METHOD

Several different factors need to be considered when choosing the best type of load table to use:

- Source and target database types
- Locations of source and target databases
- Available connectivity options and performance
- Amount of data being loaded
- Is the data delivered or fetched?
- For delivered data, what format is it in and does it require processing to make it loadable?

Load Table Decision Tree

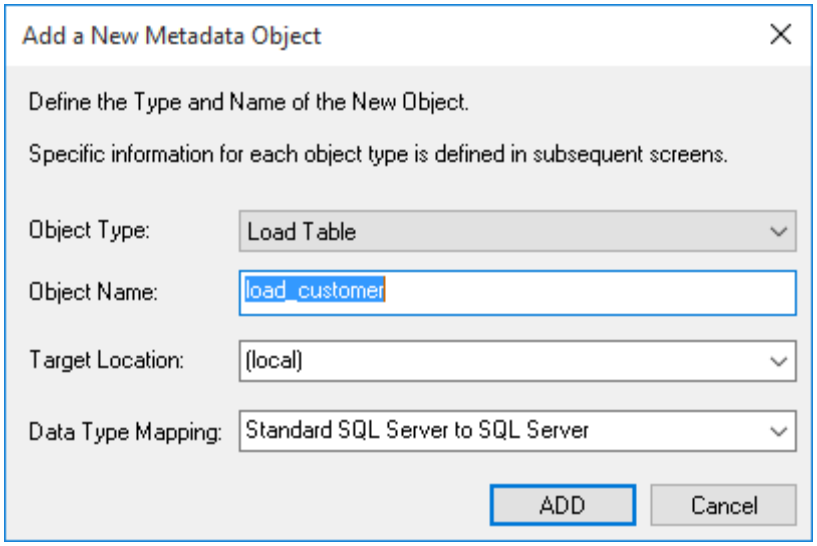
The following decision tree can be used when selecting the best type of load table:



LOAD DRAG AND DROP

The simplest way to create a load table is to use the drag and drop functionality of WhereScape RED.
Drag and drop can be used for all connection types and the process is the same in all cases.

1. Browse to the source system connection (Browse/Source Tables).
2. Create a drop target by double-clicking on the Load Table object group in the left pane. The middle pane should have a column heading of **Load Table Name** for the leftmost column.
3. Select a table or file in the right pane and drag it into the middle pane. Drop the table or file anywhere in the middle pane.
4. Answer the resulting prompts to create the load table. See the tutorials for examples on how to create load tables using drag and drop.
5. When using targets, you can also change the predefined **Target Location** settings in **Tools->Options**. At the time of the table's drag and drop, the **Add a New Metadata Object** dialog enables editing the **Target Location** and **Data Type Mapping**, so the table's location can be changed on a table by table basis.
   The **Data Type Mapping** field is automatically set based on the source and the **Target Location** selected, but can be changed if needed.



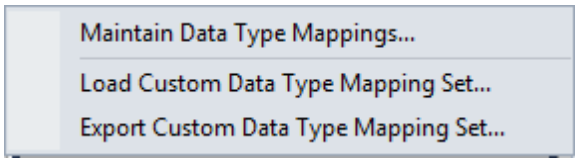
WhereScape RED supports loading tables of up to 2048 columns, however this maximum number of column loading restriction can in fact be lower than the target database or tools permit.
The target database will provide users the appropriate warning if the maximum number of columns loaded is breached at runtime.

NOTE: When creating a load table in WhereScape RED by dragging over a source table, RED will read the structure of the table on the source system and attempt to construct an equivalent load table in the data warehouse. There are occasions when the load table creation will fail due to incompatible data types in the target data warehouse. The remedy is to change the data types of the particular attributes which are causing the load failure. Once corrected the load table should create.
It is important that the table load is tested to ensure that data can be INSERTED into the load table from the source table.
If the load fails, then the data may need to be explicitly converted to the destination data type using a column transformation that is executed during the load (see ***Load Table Transformations*** (on page 318)).
Incompatible data types that cause load table creation errors are typically caused by:

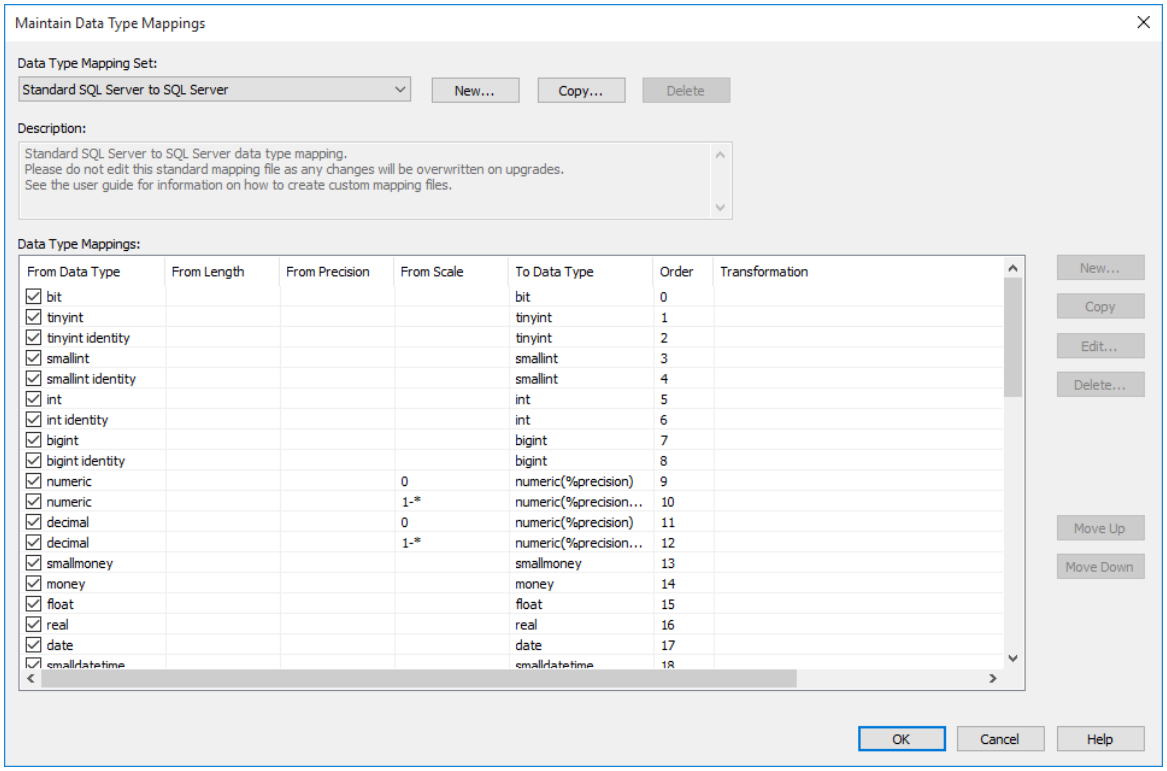
1. User defined data types in the source database.
2. Incorrect data type mapping during load table definition in WhereScape RED.
3. Data types that cannot be inserted into (e.g. identity columns on SQL Server).

DATA TYPE MAPPINGS

In **Tools/Data Type Mappings**, the first menu option will allow you to maintain the data type mappings.



The user interface for maintaining data type mappings is as follows:



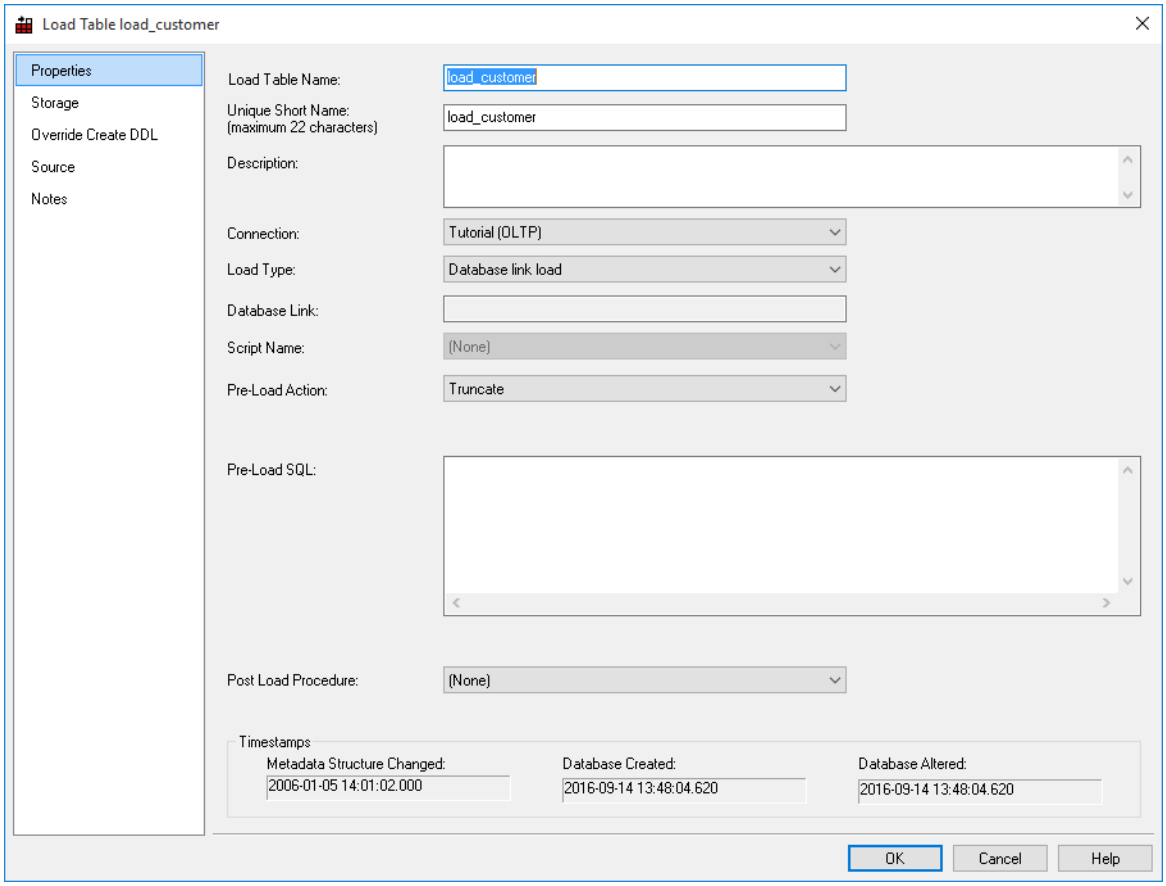
See ***Data Type Mappings*** (on page 1164) for a detailed explanation of Data Type Mappings.

DATABASE LINK LOAD

The various options are described below.

DATABASE LINK LOAD - PROPERTIES

The fields on the Database Link Load Properties Screen are described below:



Load Table Name

The table name is limited by most relational databases to a maximum of 30 characters and must be unique. Table name defaults can be set up in **Tools/Options** to define a prefix or a post fix that can be added to identify clearly that this is a load table. Example: **load** customer. By default, WhereScape RED uses the prefix **load_** for load tables.

Unique Short Name

The table short name is limited in size to 22 characters in Oracle and SQL Server and 12 characters in DB2. It must be unique. The short name is used in the naming of indexes, keys, procedures and scripts.

Description

Enter here a description of the table. This description appears in the documentation that can be generated once the data warehouse is built.

Connection

Enter the connection being used to get the data. The connections for load tables can be changed in bulk see ***Changing load Connection and Schema*** (on page 318).

Load Type

The load type is typically defined by the connection, and should not normally be changed. This drop-down shows all valid load types for the connection.

Database Link

The database link will display in this field.

Script Name

This field is only active if this is a script based load.

TIP: When doing a Script based load, use the **Rebuild** button after selecting the relevant scrip to be rebuilt on the the **Scrip Name** drop-down menu.

Pre-load Action

Select an action to be performed on the table before the load occurs. Options are:

- Truncate
- Execute pre-load Sql
- Both Truncate and Execute pre-load Sql
- No action

Pre-load Sql

If a Pre-load Action of **Execute pre-load Sql** was selected, then the Sql statement to execute before the load takes place should be entered in this box.

Truncate Options

Oracle Only: RED allows controlling the Oracle TRUNCATE options such as **REUSE/DROP STORAGE** for load processing.

- Add a Truncate option for load table processing by typing it in the Truncate options box on the Load Table Properties screen.
- Truncate options can also be added to the procedure generation on the **Define Procedure Type** dialog.

The contents of pre-load sql can be a sql statement or a procedural block.
If using a single statement, then the trailing semi-colon is not required for Oracle.
If using a procedural block, then the final semi-colon is required.

The following examples illustrate the possible values in this field.
Note the trailing semi-colon on the procedural block example.

Example of pre-load statements:
delete from load_customer where code < 23
truncate table load_customer

DECLARE
v_status number;
v_result varchar2(256);
BEGIN
update_dim_customer(1,'a','b',0,0,v_result, v_status);
END;

NOTE: Not all pre-load SQL commands will work in a SQL Server database. These commands are executed as a batch. For example: create trigger as the second command will fail as it is not supported in batch mode unless it is the first command.

Table Properties clauses (e.g, Partition)

These are clauses that are added to the end of the table create statement. Typically used for putting partition information on the table. In Oracle, this can also be used to add a parallel clause or compression clause to the table.

Post Load Procedure

A procedure that is executed immediately following the load. If you execute an externally loaded table, no load occurs, but a post load procedure can still be executed.

NOTE: At the bottom of the Load Table Properties screen are three fields that display information on that load table:
1. Date table structure last updated
2. Date created in database
3. Date last updated in database

DATABASE LINK LOAD - SOURCE MAPPING

The fields on the ***Database Link Load Source Screen*** (see "***Load Table Source Mapping Screen***" on page 242) are described below:

Load Type

Method of loading data into the table. The available options are dependent on the **Source Connection**. Defaults to the 'Default Load Type' of the **Source Connection**. Can be specified via the Properties page.

Source Connection

The connection that identifies the source database. Can be specified via the Properties page.

General

Select Distinct Values

Include the DISTINCT keyword in the SQL SELECT statement.

Source Schema

Schema within the source database where the source table resides.

Derive Source Tables(s) and Source Columns

Derive the **Source Table(s)** and **Source Column(s)** properties (of this dialog) from the source details of this table's columns.

Note: The existing property values will be overwritten.

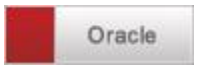
Source Table(s)

Name of the table or tables that the data is sourced from.

Override Source Column/Transformations

Ignore the source and transformation details of this table's columns and instead use the override details specified below. See the section on ***Load Table Transformations*** (on page 318) below for a fuller explanation.

Allow Missing Source Columns



Oracle Only – Allows the load to happen when one or more of the source columns do not exist. (see the section on ***handling missing source columns*** (on page 316)).

Exit Status when Missing Source Columns

Load exit status when source columns are missing. Only enabled when **Allow Missing Source Columns** is set to True.

Where and Group By Clauses

Optional SQL SELECT WHERE-clause and/or GROUP BY-clause. Parameter names can be specified using $Pparameter_name$ (with leading $P and trailing $), which are replaced at run-time by the parameter's value.


TIP: This is where you can build a statement to handle change data.

Override Load SQL

Optional SQL statement to load data into the table, which overrides all other properties. The specified SQL will be executed instead of generated SQL. For a linked database specify a complete INSERT statement. For an ODBC source specify only the SELECT statement.

SQL Insert Hint

Optional hint to include in the SQL insert statement.

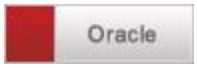
DB2 Load Logging

Controls the level of logging during the load (Minimal Logging or Recoverable Load). The default is to reduce logging, which does NOT provide for full recovery.

DB2 Load Modifier

Optional Modifiers for the Load utility.

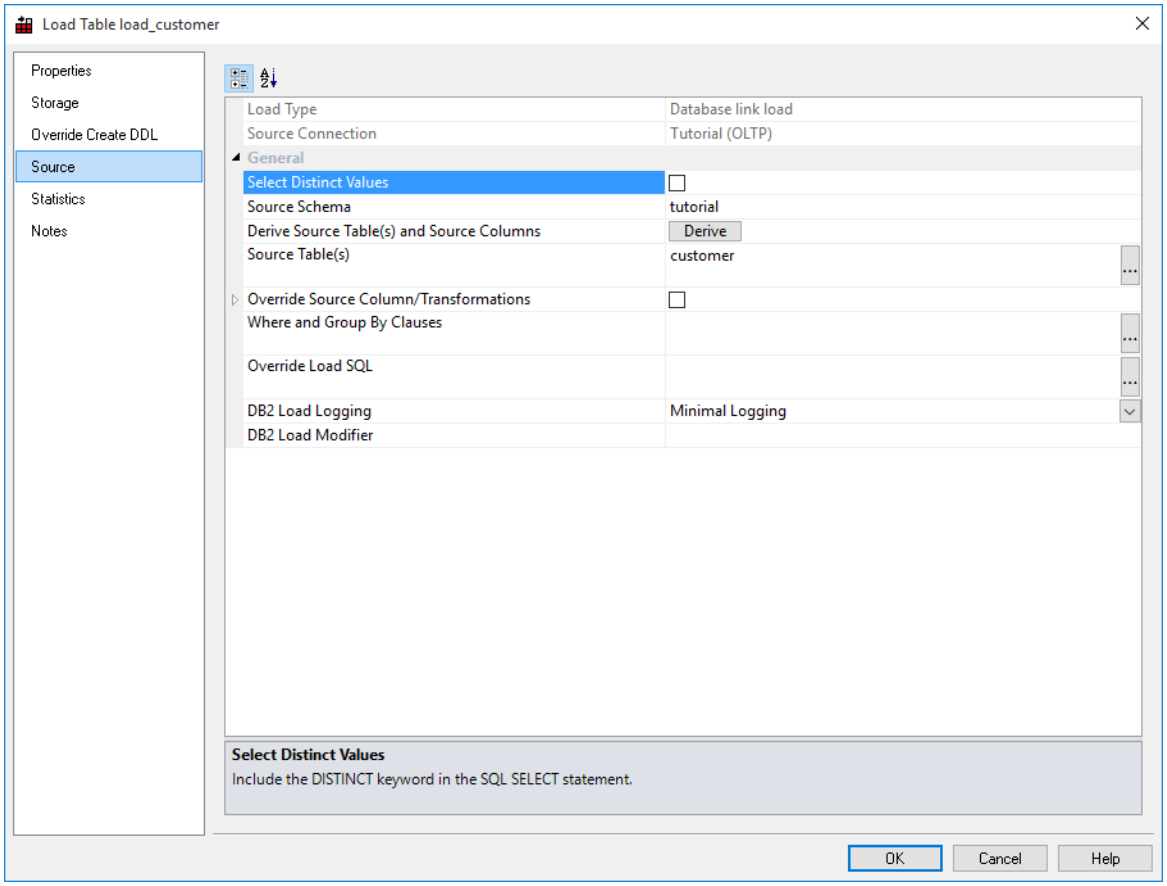
Source View Name



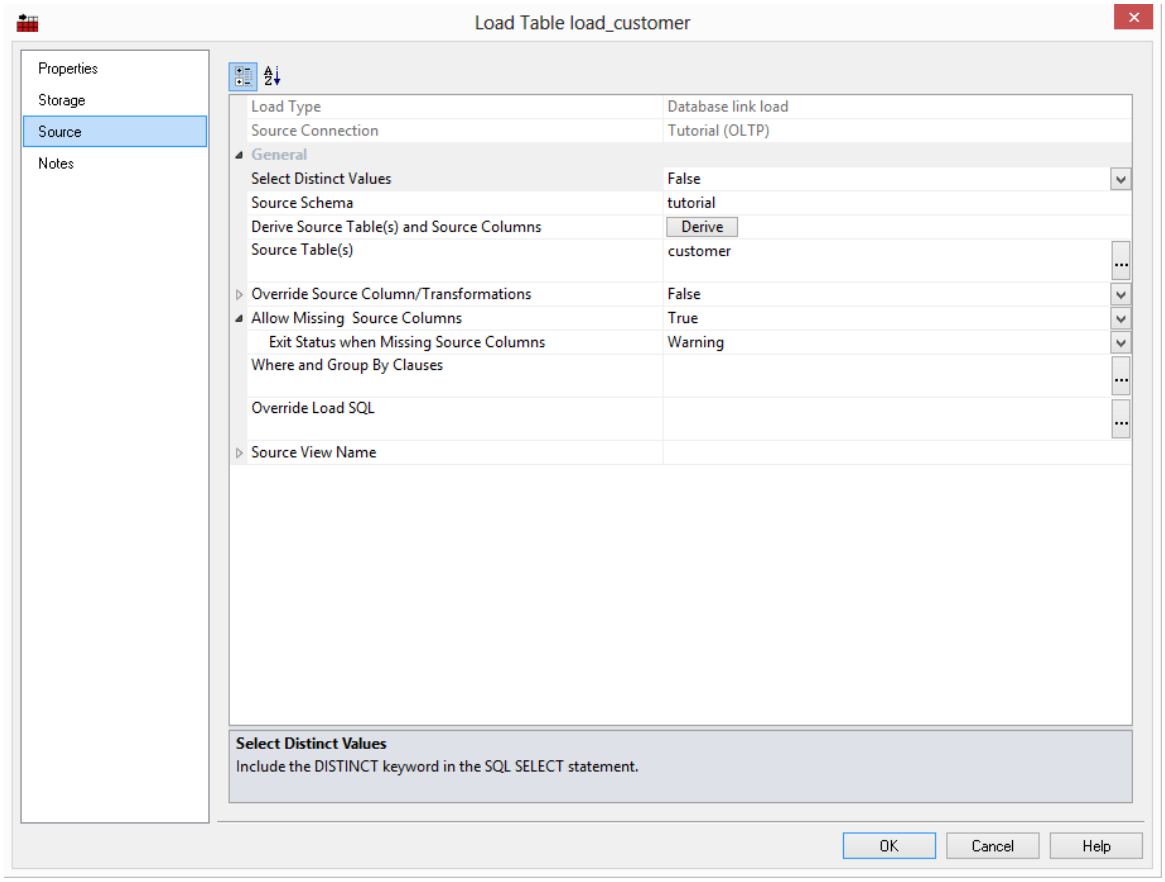
Oracle only option: Name of the database view to be created by WhereScape RED (if necessary) in the source system that will be used to select the source data. See ***Remote View Extract*** (see "***Remote View Extract - Oracle Databases Only***" on page 245) topic.

LOAD TABLE SOURCE MAPPING SCREEN

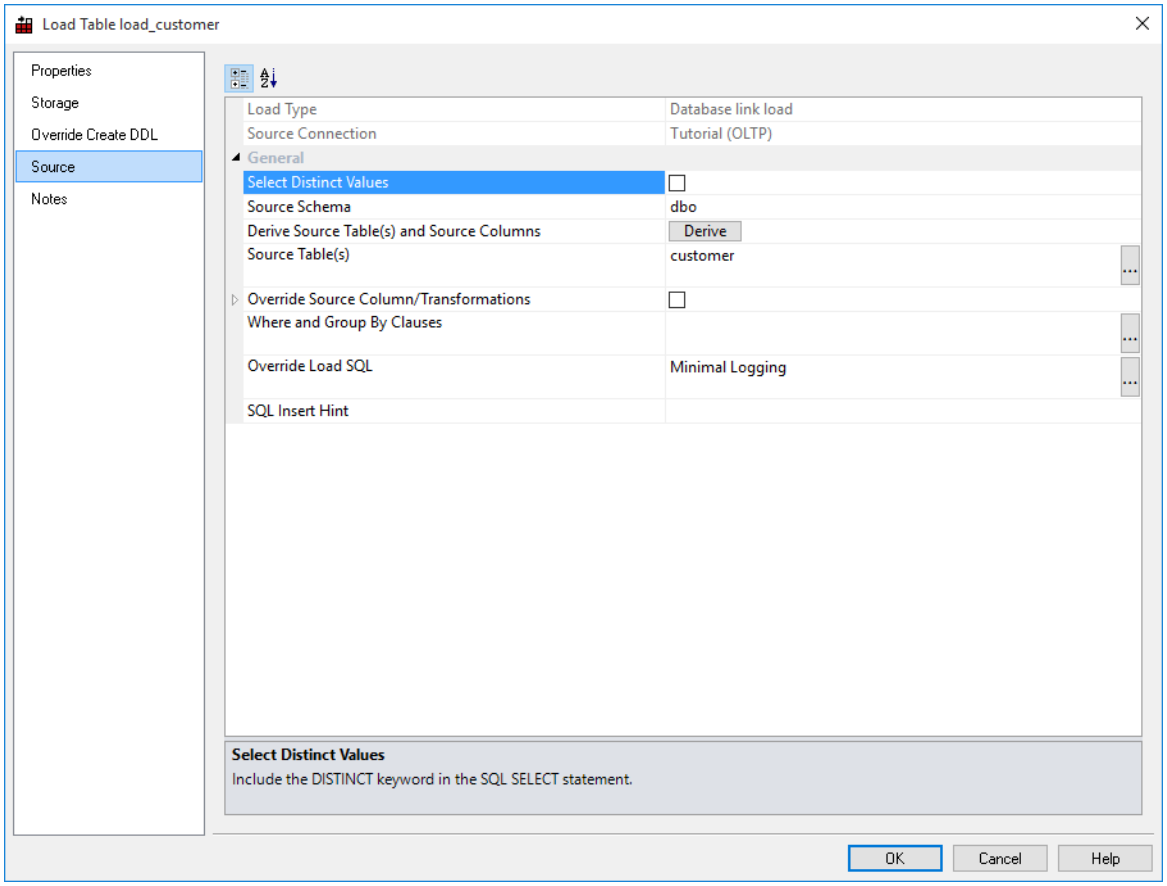
DB2 **Source Screen:**



Oracle **Source Screen:**



SQL Server **Source Screen:**


Load Table load_customer
Properties
Storage
Override Create DDL
Source
Notes
Load Type
Database link load
Source Connection
Tutorial (OLTP)
General
Select Distinct Values
Source Schema
dbo
Derive Source Table(s) and Source Columns
Derive
Source Table(s)
customer
Override Source Column/Transformations
Where and Group By Clauses
Override Load SQL
Minimal Logging
SQL Insert Hint
Select Distinct Values
Include the DISTINCT keyword in the SQL SELECT statement.
OK
Cancel
Help

REMOTE VIEW EXTRACT - ORACLE DATABASES ONLY

Note: To use **Remote View Extracts**, both the **source** database and **target** data warehouse database must be **Oracle**.

Remote view extracts are used to force a remote Oracle source system database to perform complex extraction logic in the remote source system database and return the results to the data warehouse.

When specifying a join in an extract (load) over a database link the Oracle cost based optimizer decides whether it is better to join the tables on the remote machine, or if the tables should be copied onto the local machine and the join performed locally. If the decision to copy is made for large tables this can result in huge extract times. The simplest method of forcing the join to occur on the remote machine is to create a view on the remote machine from which the extract is done.

The view is re-created every time the load table is run. Then the load table selects all columns from the remote view and inserts the results into the load table over the defined database link. To create a remote view load table, users will need to create a Database Link connection to load the remote view table from that connection.
To see more about creating a Database Link connection see ***Creating a Database Link***.

The steps for creating Database Link connection to then creating a remote view load table are:

1 Create a connection object for the remote system or use an existing connection, populating the following fields:
   - Connection Name -> enter a name for the connection
   - Connection Type -> set to **Database**
   - ODBC Source -> enter the ODBC source used to extract the source data
   - Database ID -> enter the Oracle SID or TNS Name
   - Database Link Name -> enter a name for the database link. This is a required field for establishing the **Database Link** connection
   - Extract User ID -> enter the user name that has access to the source database
   - Extract User Password -> enter the password for user that has access to the source database

2 Ensure the user name and password being used to connect to the remote system have the following privileges:
   - create view (granted as a specific privilege and not part of a role)
   - compile procedure (granted as a specific privilege and not part of a role)
   - execute on sys.dbms_sql

3 Right-click on the connection to the remote system and select **Create Database Link**. When the dialog with the database link has been created message is displayed, click **OK**.

4 Right-click on the connection to the remote system and select **Create Remote View Procedure**. This will create the remote view procedure in the linked database.

5 When the dialog with the Procedure dss_view_create compiled is displayed, click **OK**.

6 Browse the connection in the right pane.

7 Drag and drop a source table or columns of the source tables to create the required load table.

8 Create the load table in the data warehouse database by selecting **Create (Recreate)** from the right-click menu.

9 Right-click on the load table and select **Source Mapping**.

10 Enter a name for the remote view to be created into the **Source View Name** box. For example, use the name of the load table.

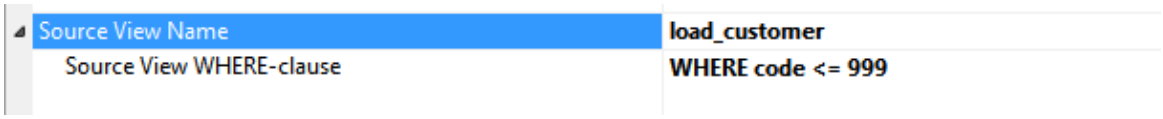
11 Cut or copy any existing 'Where' clause from the **Where, Group etc. clauses** box and paste it into the **Source View WHERE-clause** box. Alternatively, enter the 'Where' clause (including any table joins, if multiple source tables are being extracted from) into the **Source View WHERE-clause** box.

Note: Once the load table has a remote view name, the 'Where' clause in the **Where, Group etc. clauses** box is ignored and the 'Where' clause in the **Source View WHERE-clause** box is used instead.

12 Click **OK**.

13 Load or schedule the load table.

Example remote view load table source mapping properties:



Note: During Load, column transformations are ignored with Remote View Extract load tables. After Load, column transformations can be used. To create a derived column on a Remote View Extract load table, add the column leaving its source table blank and setting its source column to NULL. Then create an After Load transformation to populate the derived column.

ODBC BASED LOAD

An ODBC based load provides an extensive option for acquiring data from many sources. It will be slower than most other forms of loads, but may still be a viable option depending on the volume of data being loaded.

An ODBC based 'interactive load' when using the WhereScape RED tool will use the established ODBC connection to pull the data back to the local PC and then push the data to the data warehouse database via a sql ODBC statement.

A scheduler load will perform in the same way except that the data is loaded into the server that is running the scheduler and then pushed to the data warehouse database.
For Oracle, this push to the data warehouse can be performed via Oracle Direct Path loading, which can be considerably faster than the ODBC sql method.

For Oracle Direct Path loading to work all dates and times must be a character string of the form 'YYYY-MM-DD HH24:MI:SS'.
This is normally achieved via a 'During' load transformation such as
`TO_CHAR(source_date_column,'YYYY-MM-DD HH24:MI:SS')`

Note: Dates must be in the correct format for Oracle Direct Path loading to occur.

The obvious disadvantage in an ODBC based load is the two network transactions required and the overhead placed on the Scheduler server. The UNIX scheduler does not support ODBC based loads, therefore a Windows scheduler must be available to handle ODBC based loads.

Note: A windows scheduler must be available to handle ODBC based loads.

The Properties screens for an ODBC based load are the same as those of a database link load. Refer to the previous section for details.

SSIS LOADER

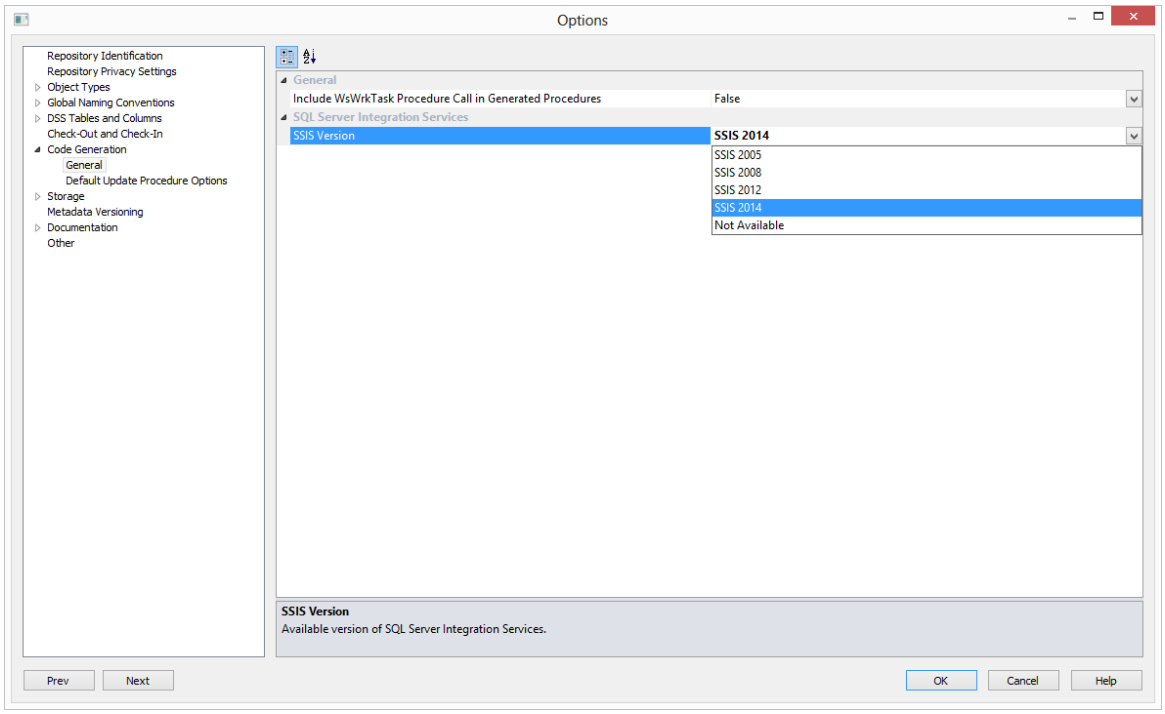
Microsoft SQL Server Integration Services (SSIS) is an Extract Transform and Load (ETL) utility that is supplied and licensed as part of Microsoft SQL Server 2005+. SSIS is built to extract data from data sources using extraction technologies such as OLEDB, transform the data using its own .NET based language and then load it into data destination. SSIS is primarily used for loading data from various sources into SQL Server, but it can be used to extract from most databases and load into any other database.

WhereScape RED manages the extraction and load of data into the data warehouse as part of data warehouse processing. WhereScape RED attempts to utilize the optimum loading technology for each supported database platform. Optimal loading performance from a database source into SQL Server is achieved using SSIS. WhereScape RED provides the ability to leverage the extract and load performance available from SSIS as part of the WhereScape RED processing. WhereScape RED does this by generating and executing an SSIS package dynamically at run time. Any errors or messages resulting from execution are passed back into the WhereScape RED workflow metadata to drive subsequent workflow actions. In the event of an error during execution of the SSIS package, the package will be persisted to disk to assist the developer with problem resolution.

An **Integration Services Load** can have a number of source databases:

- SQL Server
- Oracle
- DB2 for Linux, UNIX and Windows (the DB2 RED can build a warehouse in)
- Teradata
- Sybase
- MySQL
- DB2/400 (the DB2 that runs on AS/400 machines - quite different to above)
- MS Access
- Excel

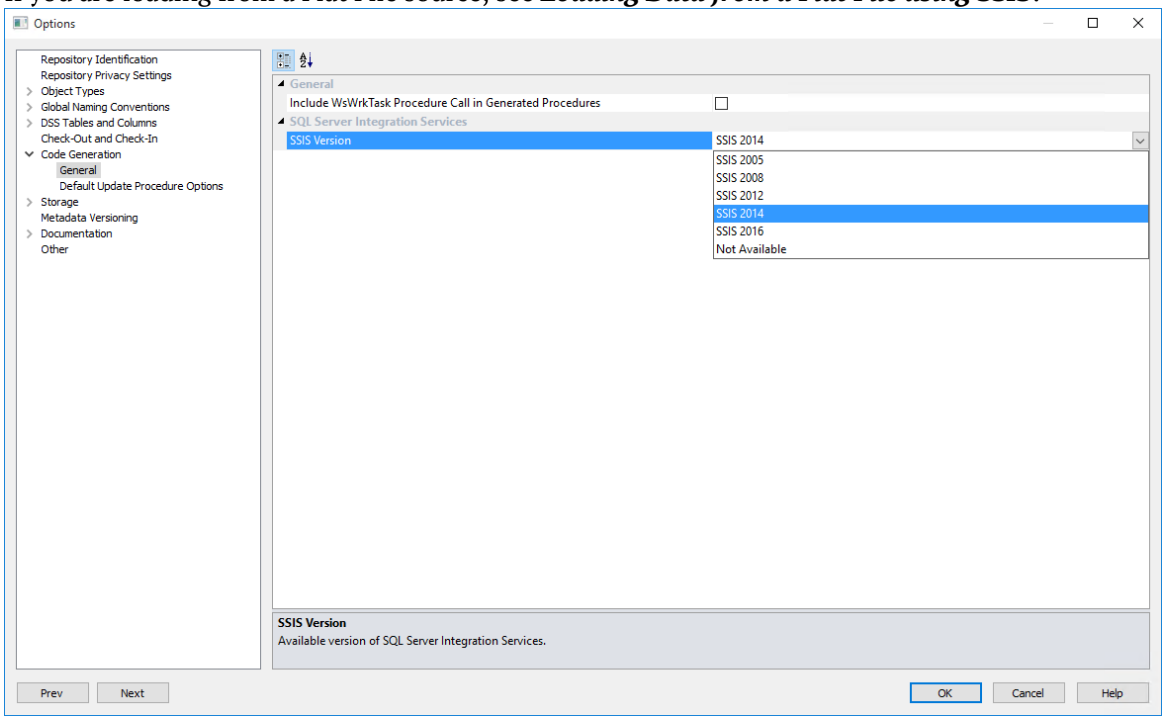
To use SSIS to load data, select the relevant version of SSIS in **Tools/Options/Code Generation/General**.



LOADING DATA INTO RED LOAD TABLES USING SSIS

To use SSIS loading, ensure that SSIS loads are enabled and that the SSIS version is set correctly in **Tools/Options/Code Generation /General**.

If you are loading from a Flat File source, see ***Loading Data from a Flat File using SSIS***.



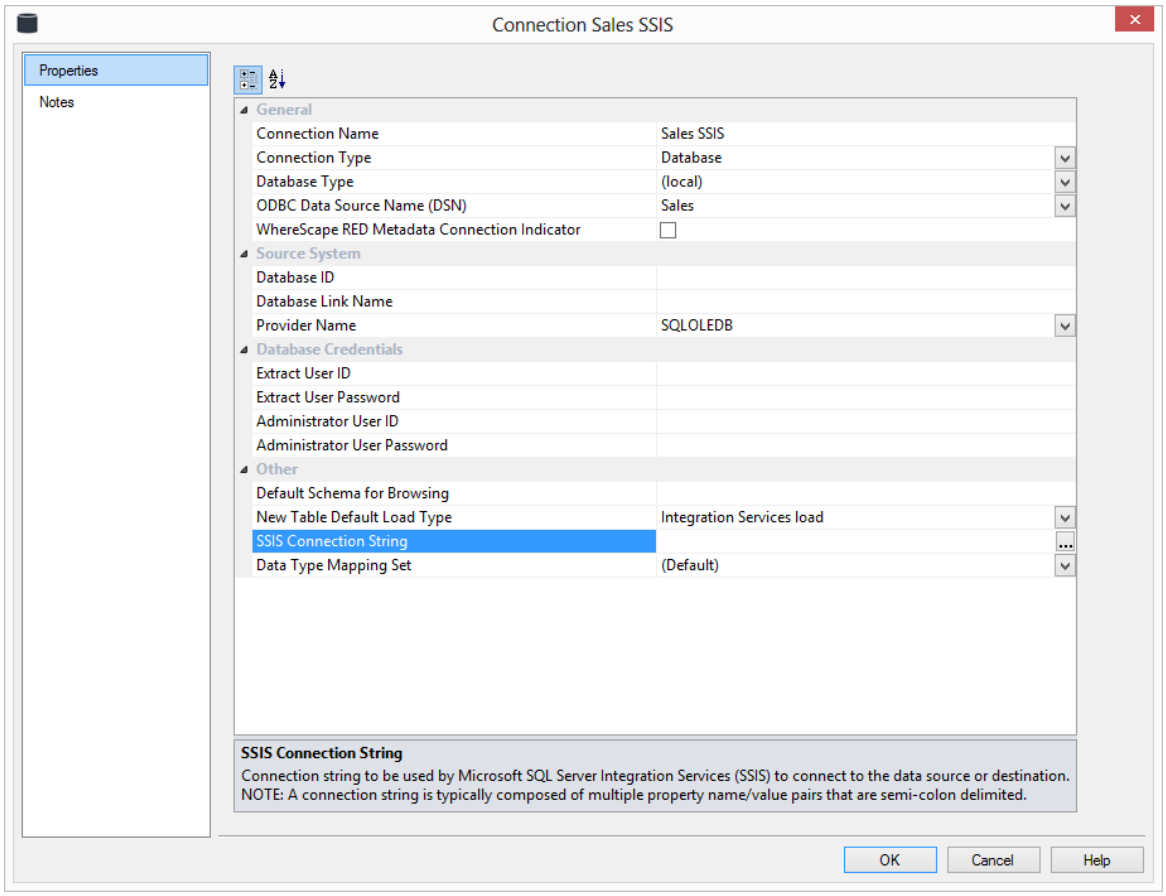
RED can extract and load data using SSIS from database tables or flat files from a Windows connection. As with any load into RED a connection to the source data needs to be created to provide extraction details.
The **SSIS Connection String** is a valid SSIS connection string that can be used to connect to the data source or destination.

Loading Data via SSIS from a database

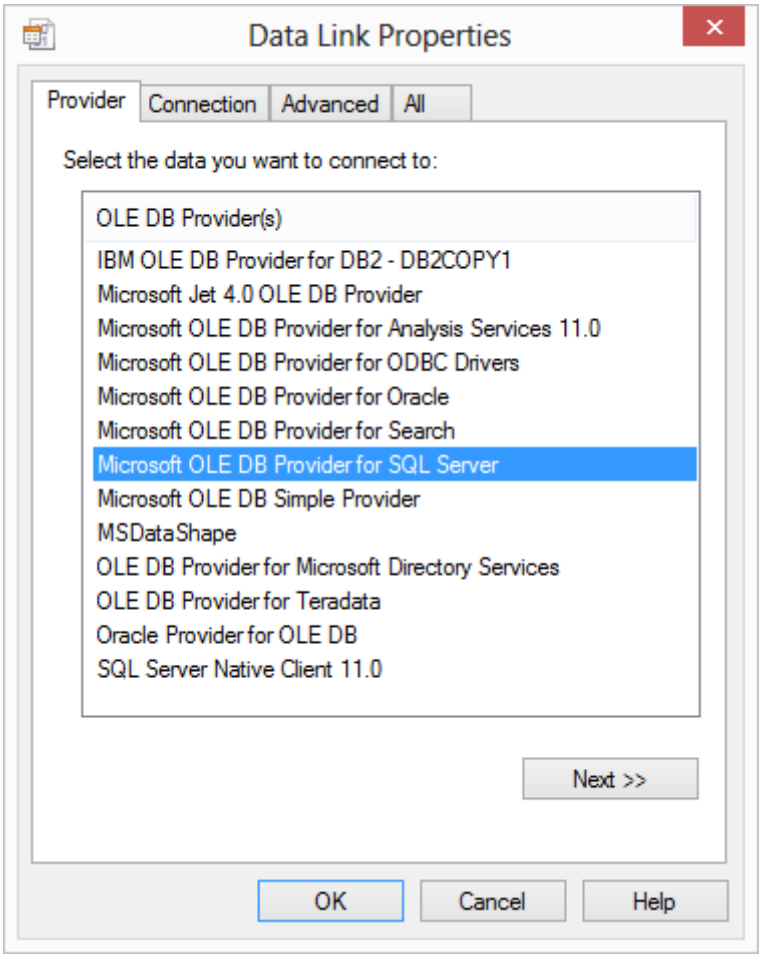
If the connection is a database load, then there is additional connection information that should be supplied to use SSIS as a loading option.
This additional information needs to be supplied on both the source connection and the data warehouse connection.

Creating the SSIS Connection String

1 Click on the **ellipsis** button to the right of the SSIS Connection String field of the relevant connection:



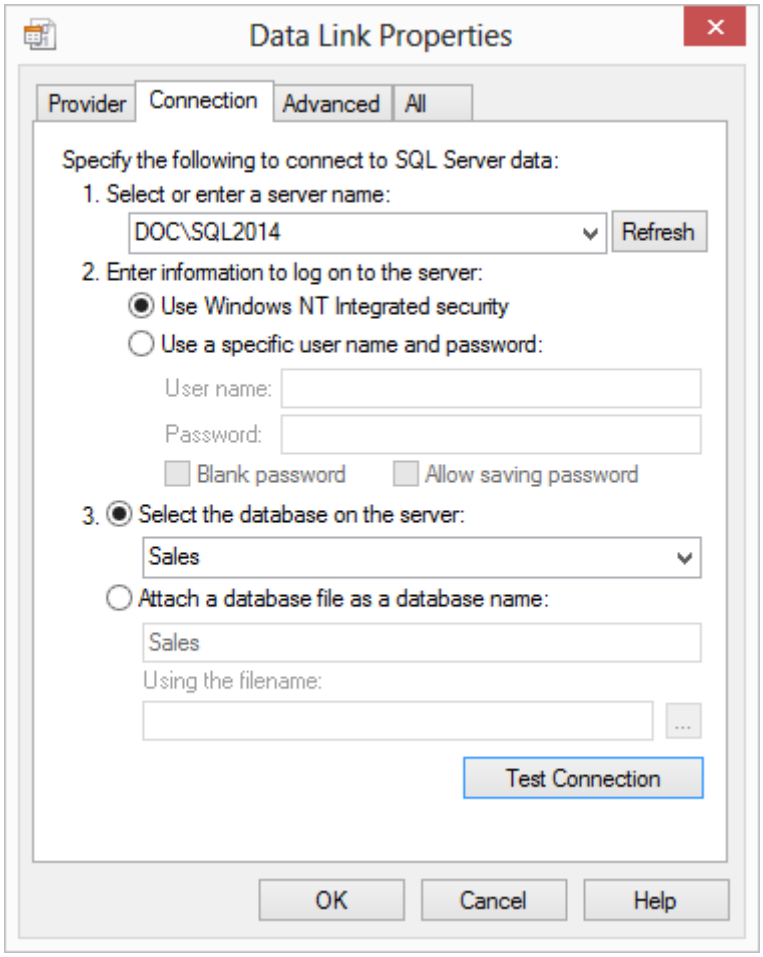
2 On the Provider tab, select the relevant **OLE DB Provider** and click **Next**.



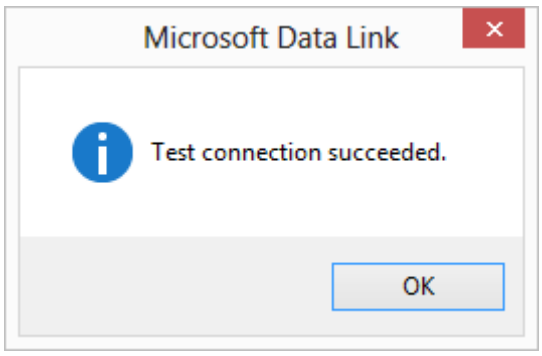
3 On the **Connection** tab, select the **server name**, enter the **information to log on to the server** and select the **database** on the server. Click **Test Connection**.

NOTE: When using a specific **user name** and **password** to connect to the server instead of using Windows integrated security, the **Allow saving password** check-box must be be ticked.

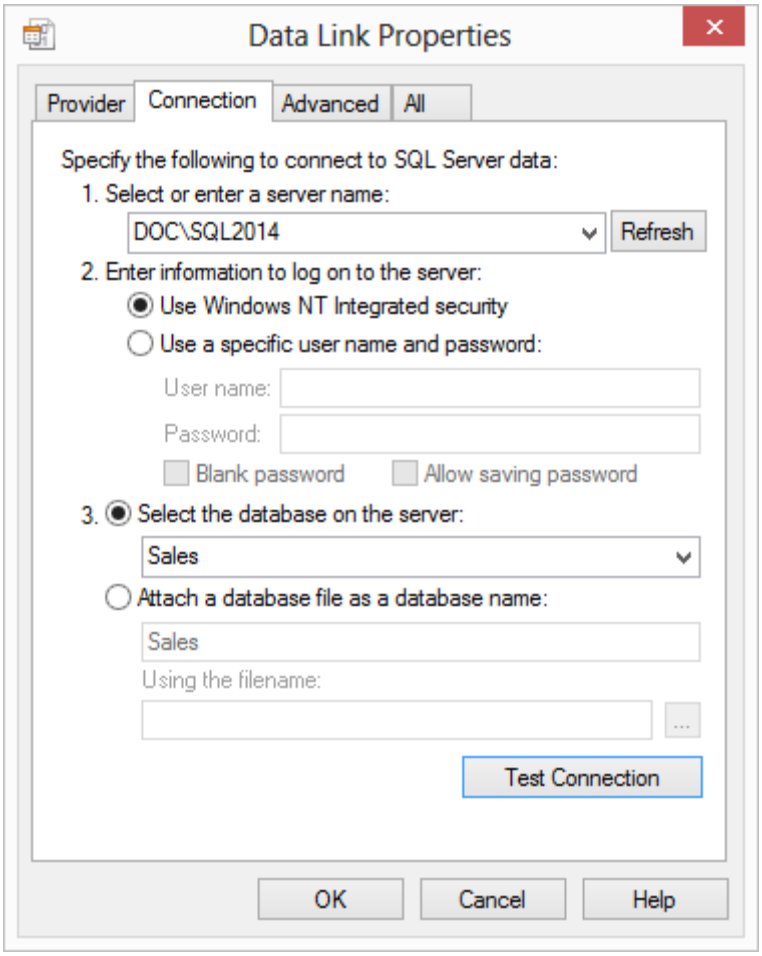
It is also recommended that the password on the SSIS connection string field that is displayed in the connection properties is replaced with the **$PASSWORD$** token that is substituted at runtime.


Data Link Properties
Provider
Connection
Advanced
All
Specify the following to connect to SQL Server data:
1. Select or enter a server name:
DOC\SQL2014
Refresh
2. Enter information to log on to the server:
Use Windows NT Integrated security
Use a specific user name and password:
User name:
Password:
Blank password
Allow saving password
3. Select the database on the server:
Sales
Attach a database file as a database name:
Sales
Using the filename:
Test Connection
OK
Cancel
Help

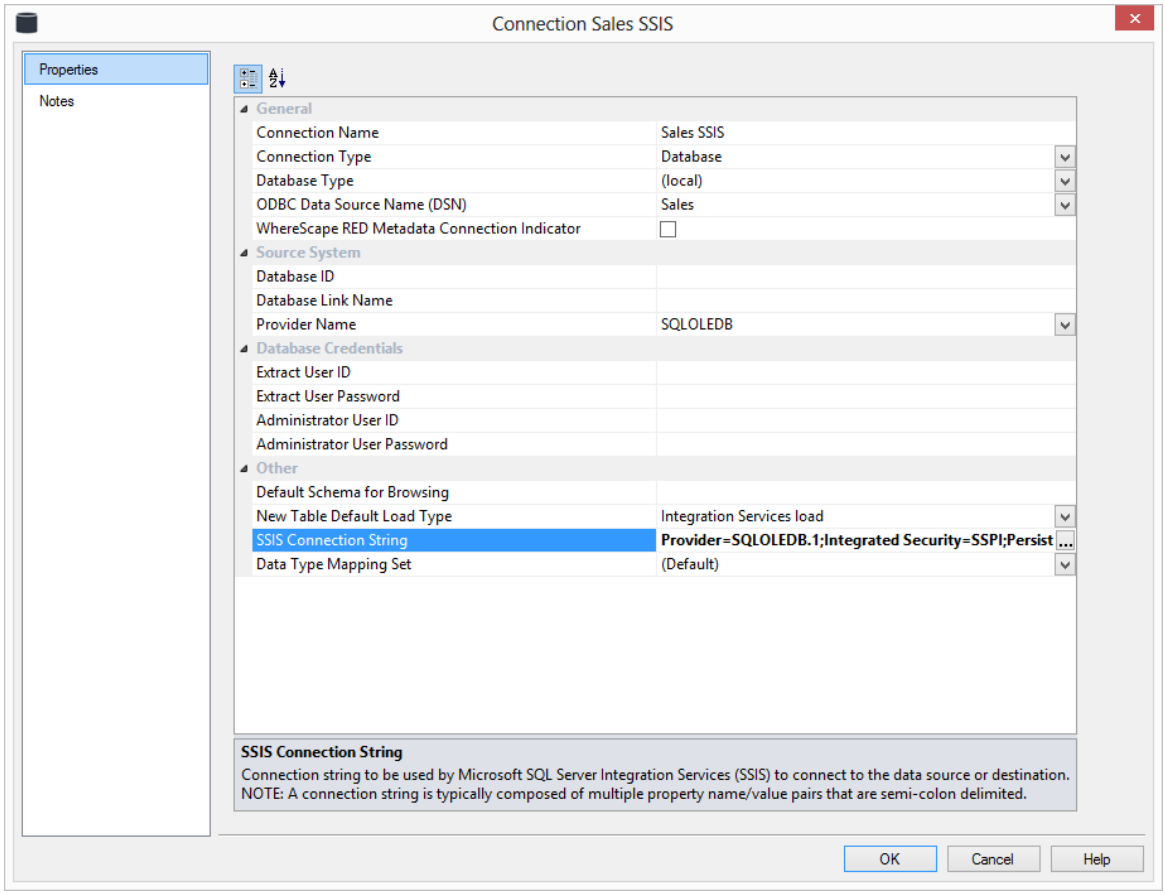
4 Click **OK**.


Microsoft Data Link
Test connection succeeded.
OK

5 Click **OK** on the Data Link Properties dialog to save the SSIS connection string settings.



6 The SSIS connection string is displayed.

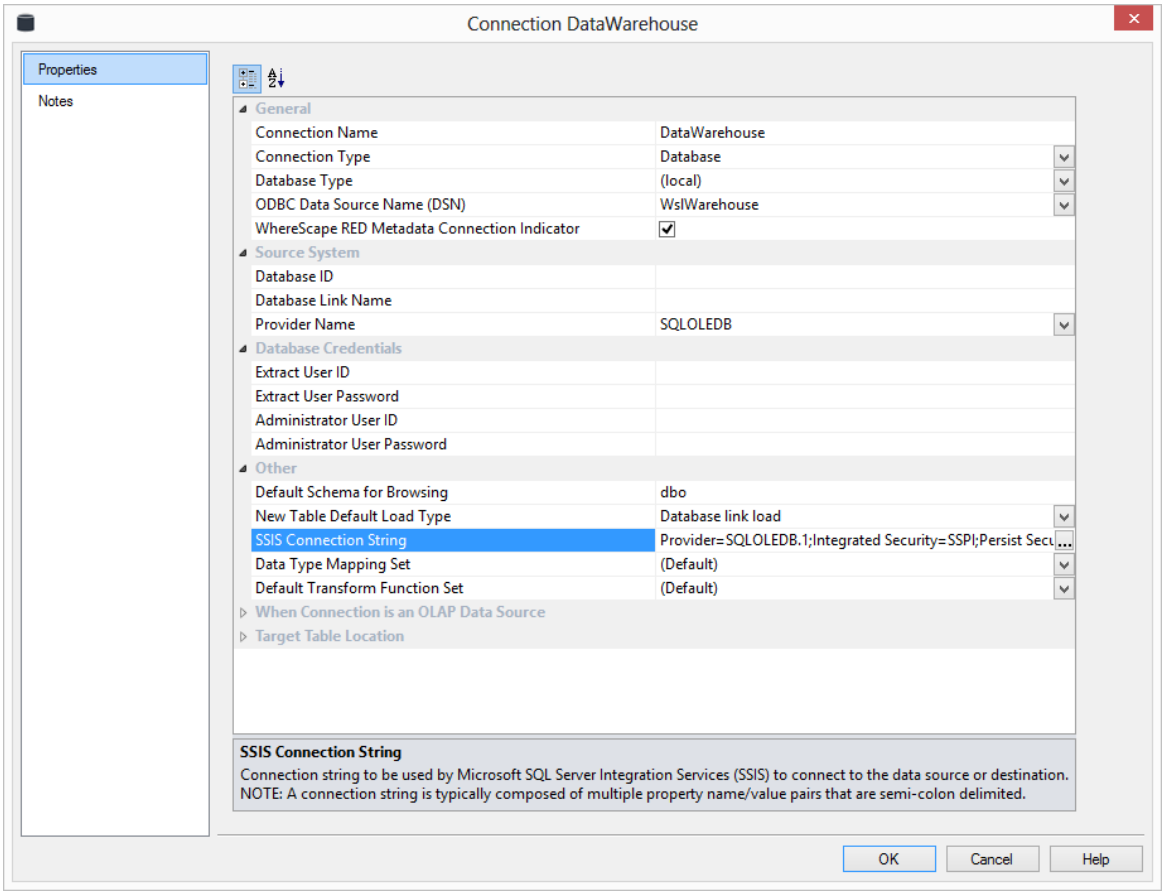


7 Click **OK** to save the connection.

- Right-click on **Sales SSIS** and select **Browse Connection.**
- Accept the defaults and click **OK**.

8 In SSIS terms, you have now defined your **Source** in **SSIS Connection Manager.** Using the same process, you need to add the SSIS Connection String to the data warehouse connection so that you can specify your **Destination** connection:

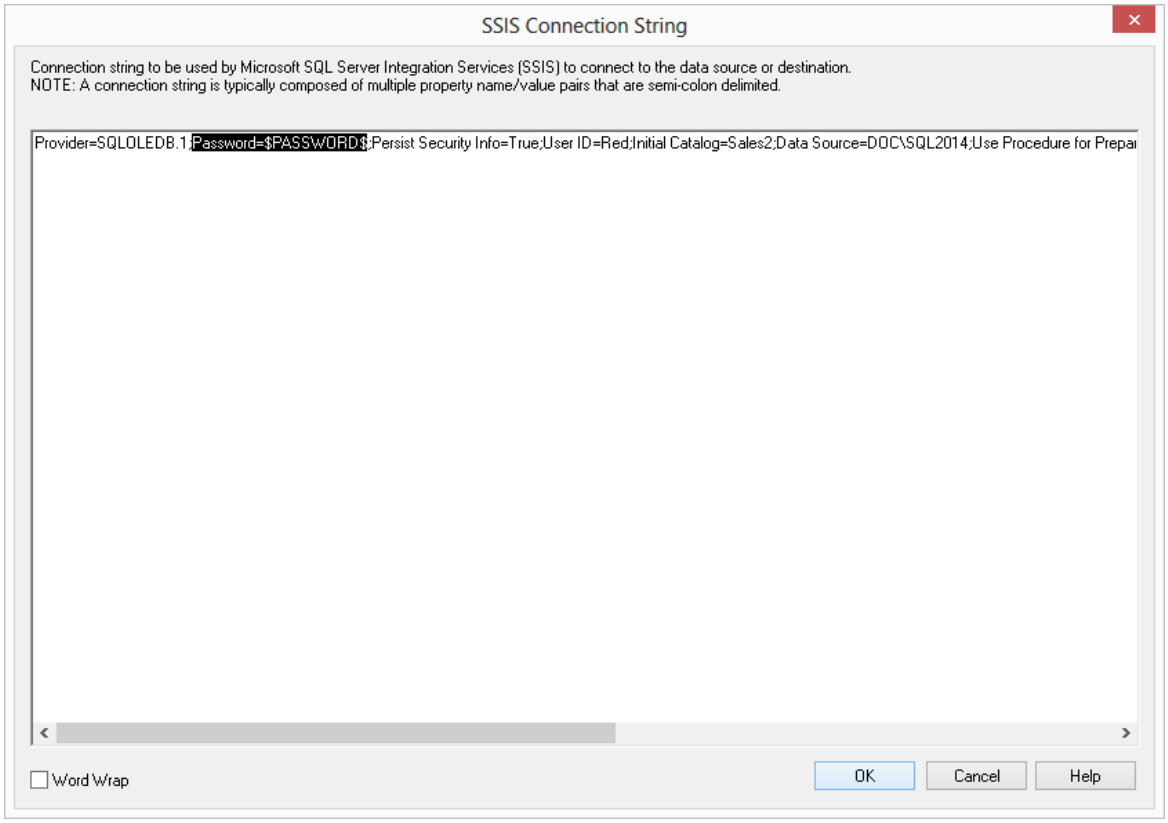
- Double-click on the **DataWarehouse** connection in the object explorer to open the Properties dialog.
- Follow the process above to create the SSIS Connection String.
- Click **OK** to save your connection.



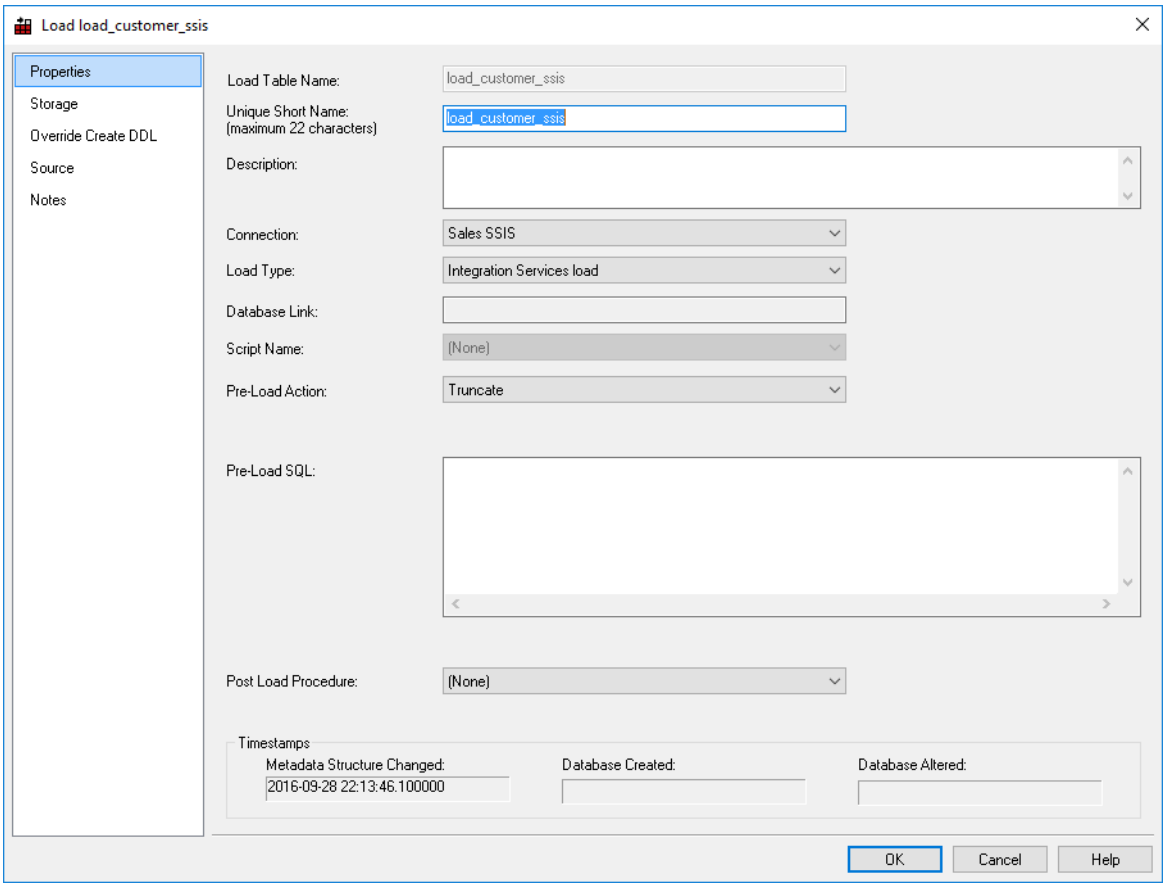
Note1: If the connection string is already set, then the ellipsis button will open an editor dialog.
Edit the connection string and click **OK**.

Note2: For connections that require a username and password, the connection string can also be edited to replace the password with the **$PASSWORD$** token that is substituted at runtime. If the $PASSWORD$ token is used, RED uses the contents of the masked "Extract User Password" property when making the connection.

E.g. "Provider=SQLOLEDB.1;Password=**$PASSWORD$**;..."



9 Once the connection is defined then a load table needs to be created to hold data loaded into the data warehouse by dragging a source table or a flat file to create a load table. (see ***Loading data*** (on page 232) and ***Flat File Load***)
On the load table properties the Load type can be set to Integration Services load. This will create and execute a SSIS package at run time to load data into the data warehouse load table.



The configuration options available on an SSIS load are available on the **Source** tab of the load table's Properties. These options are described in further detail in the next topic SSIS Load Table Source Screen.

SSIS LOAD TABLE SOURCE SCREEN

The configuration options available on an SSIS load are available on the **Source** tab of the load table's Properties. These options are:

Load Type

Method of loading data into the table. The available options are dependent on the **Source Connection**. Defaults to the 'Default Load Type' of the **Source Connection**. Can be specified via the Properties page.

Source Connection

The connection that identifies the source database. Can be specified via the Properties page.

General

Select Distinct Values

Include the DISTINCT keyword in the SQL SELECT statement.

Source Schema

Schema within the source database where the source table resides.

Derive Source Tables(s) and Source Columns

Derive the **Source Table(s)** and **Source Column(s)** properties (of this dialog) from the source details of this table's columns.

Note: The existing property values will be overwritten.

Source Table(s)

Name of the table or tables that the data is sourced from.

Override Source Column/Transformations

Ignore the source and transformation details of this table's columns and instead use the override details specified below. See the section on ***Load Table Transformations*** (on page 318) below for a fuller explanation.

Exit Status when Missing Source Columns

Load exit status when source columns are missing. Only enabled when **Allow Missing Source Columns** is set to True.

Where and Group By Clauses

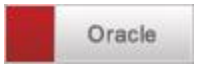
Optional SQL SELECT WHERE-clause and/or GROUP BY-clause. Parameter names can be specified using $Pparameter_name$ (with leading $P and trailing $), which are replaced at run-time by the parameter's value.

TIP: This is where you can build a statement to handle change data.

Override Load SQL

Optional SQL statement to load data into the table, which overrides all other properties. The specified SQL will be executed instead of generated SQL. For a linked database specify a complete INSERT statement. For an ODBC source specify only the SELECT statement.

Allow Missing Source Columns



Oracle only: Allows the load to happen when one or more of the source columns do not exist. (see the section on ***handling missing source columns*** (on page 316)).

SQL Server Integration Services (SSIS)

SSIS Source-Identifier Encapsulation - Characters that are used to enclose source column names. Options are (None), "", [], '', ``

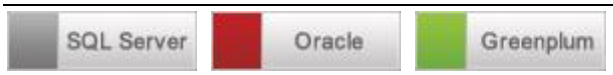
SSIS Source-Identifier Case Conversion - Case-sensitivity conversion applied to Source Object Identifiers (such as table, view, and column names) in RED-generated SSIS packages. If no conversion is applied then the exact case of the identifier defined in the RED metadata is used in SSIS.

SSIS Destination-Identifier Case Conversion - Case-sensitivity conversion applied to Destination Object Identifiers (such as table, view, and column names) in RED-generated SSIS packages. If no conversion is applied then the exact case of the identifier defined in the RED metadata is used in SSIS.

SSIS Source-AlwaysUseDefaultCodePage - Forces the use of the DefaultCodePage property value when describing character data.

SSIS Set Source-Code Page - Enables the SSIS source Code Page property.

SSIS Destination AlwaysUseDefaultCodePage



Forces the use of the DefaultCodePage property value when describing character data.

SSIS Set Destination-Code Page

Enables the SSIS Destination Code Page property.

SSIS Row Count Log - During an SSIS Load include Row Count logging.

SQL Server only options

SSIS Acquire Table Lock – Specifies whether the destination table is locked during the load. This allows SQL Server to reduce lock escalation overheads during loading and will promote minimal logging in SQL 2008+ when a Bulk Logged recovery model is used for the database. The default value of this property is true.

SQL SSIS Commit Interval – Specifies the batch size that the OLE DB destination tries to commit during fast load operations. The default value of 2147483647 indicates that all data is committed in a single batch after all rows have been processed. If you provide a value for this property, the destination commits rows in batches that are the smaller of (a) the Maximum insert commit size, or (b) the remaining rows in the buffer that is currently being processed.

SQL Set SSIS Batch Size – Specifies the number of rows in a batch. The default value of this property is empty, which indicates that no value has been assigned.

PDW only options

SSIS Load Mode – Select the SSIS loading mode from the list. The available options are Append, Fastappend, Reload and Upsert.
When the Upsert option is selected, the user must specify a key to use for the Upsert. The Upsert Key should be a business key for the loaded data.
NOTE: To use the Upsert feature, please ensure the pre-load action in the main properties screen is disabled.

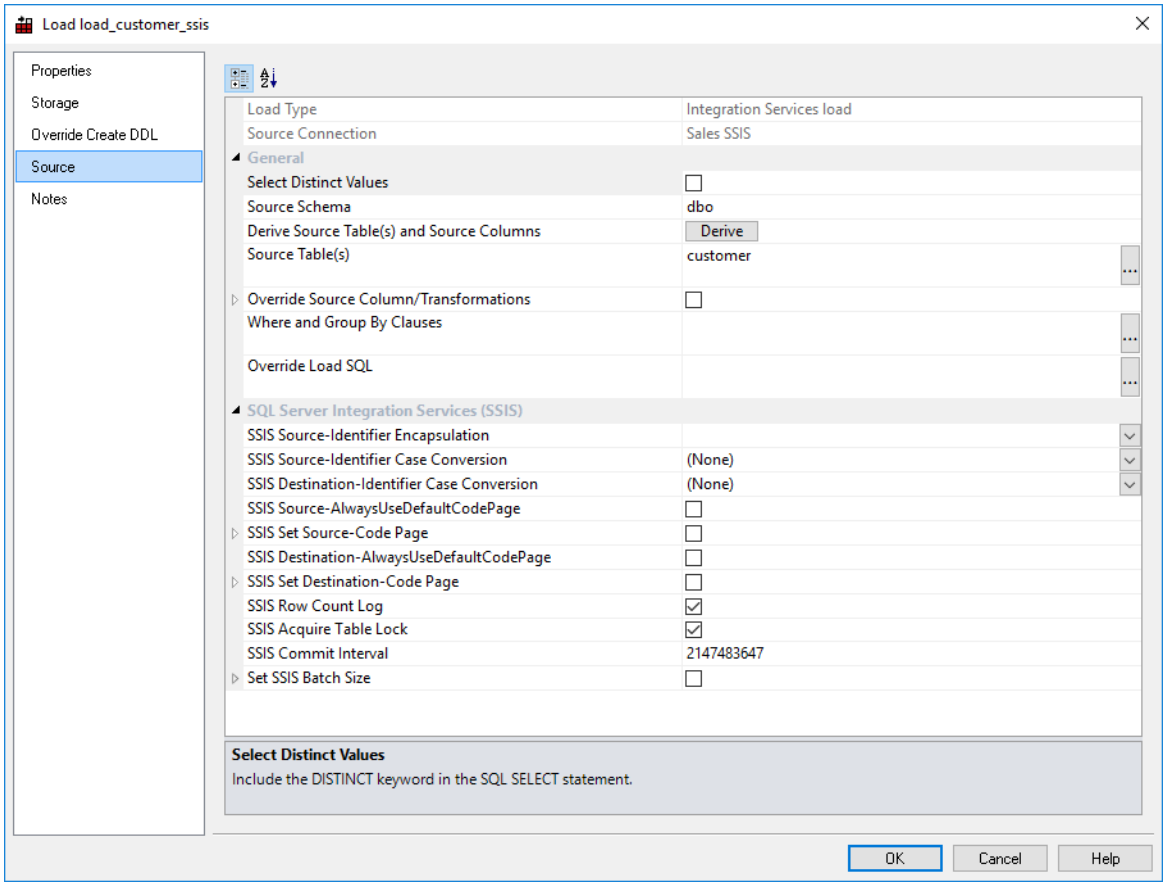
NOTE1: PDW does not allow use of staging DB or Rollback if fastappend is the load mode chosen.
NOTE2: Fastappend cannot be used if table has the storage setting of replicated.

SSIS Upsert Key Columns - Columns selected to make up the Upsert Key. The Upsert Key should be a business key for the loaded data. This can be typed in as comma-separated list of column names or alternatively, users can use the column selection dialog by clicking on the ellipsis button to the right of the SSIS Upsert Key Columns field to select the desired columns.

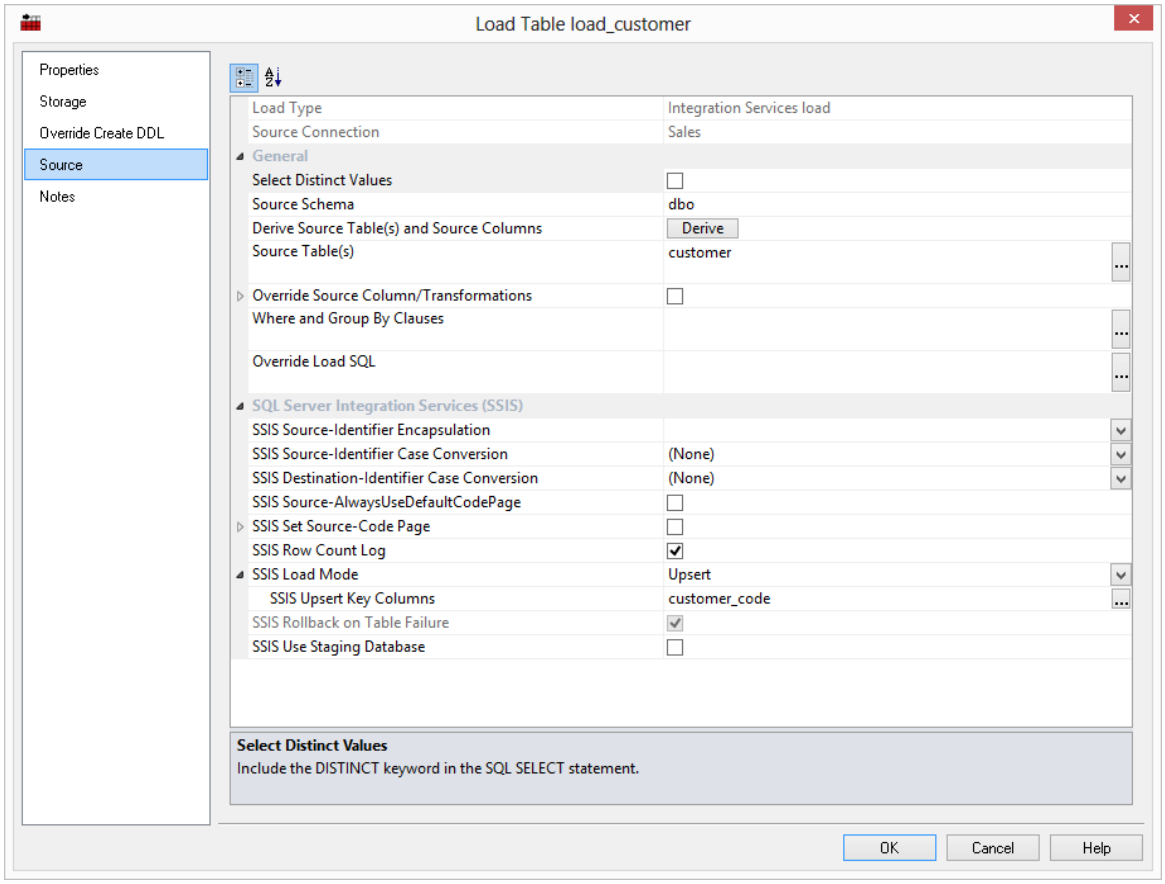
SSIS Rollback on Table Failure – In the event of a SSIS failure, a rollback will be performed.

SSIS Use Staging Database – If this option is set and a staging database is defined on the target connection, then the staging database will be used instead.

SQL Server SSIS load Source screen example:



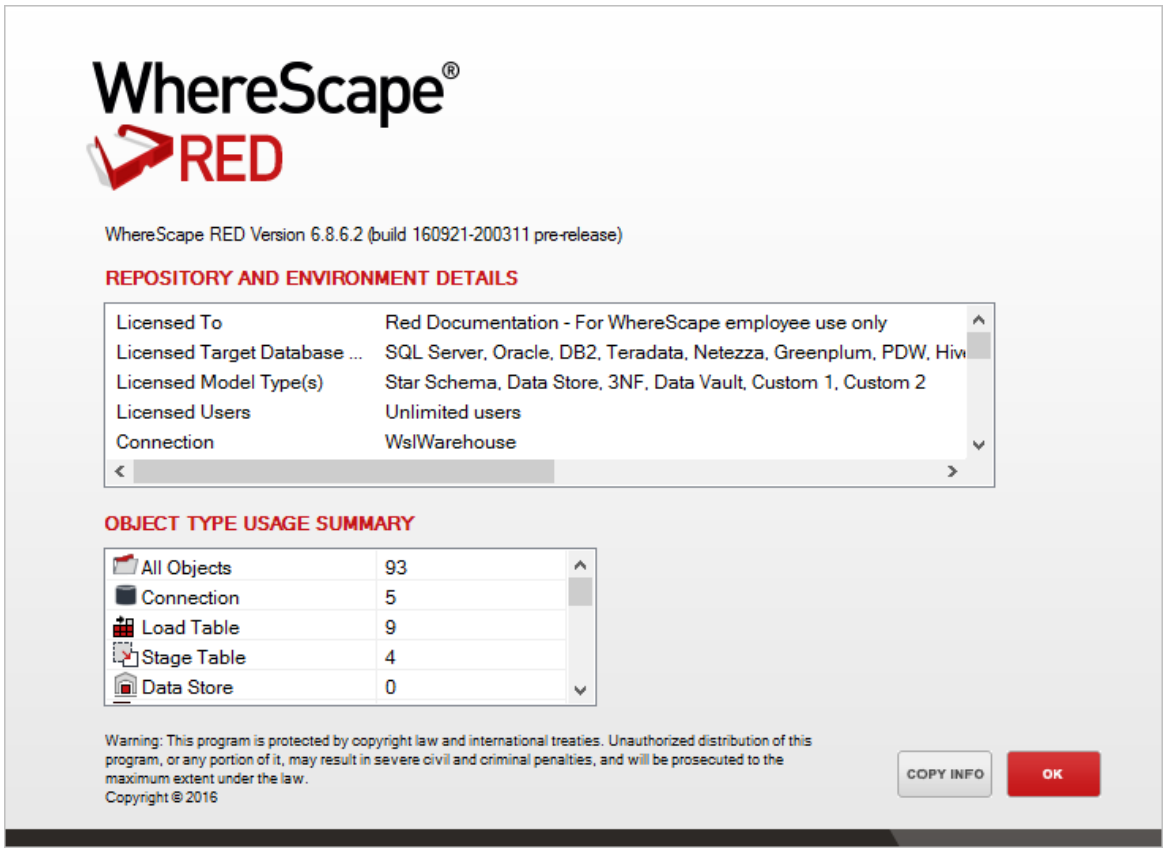
PDW SSIS load Source screen example:



TROUBLESHOOTING

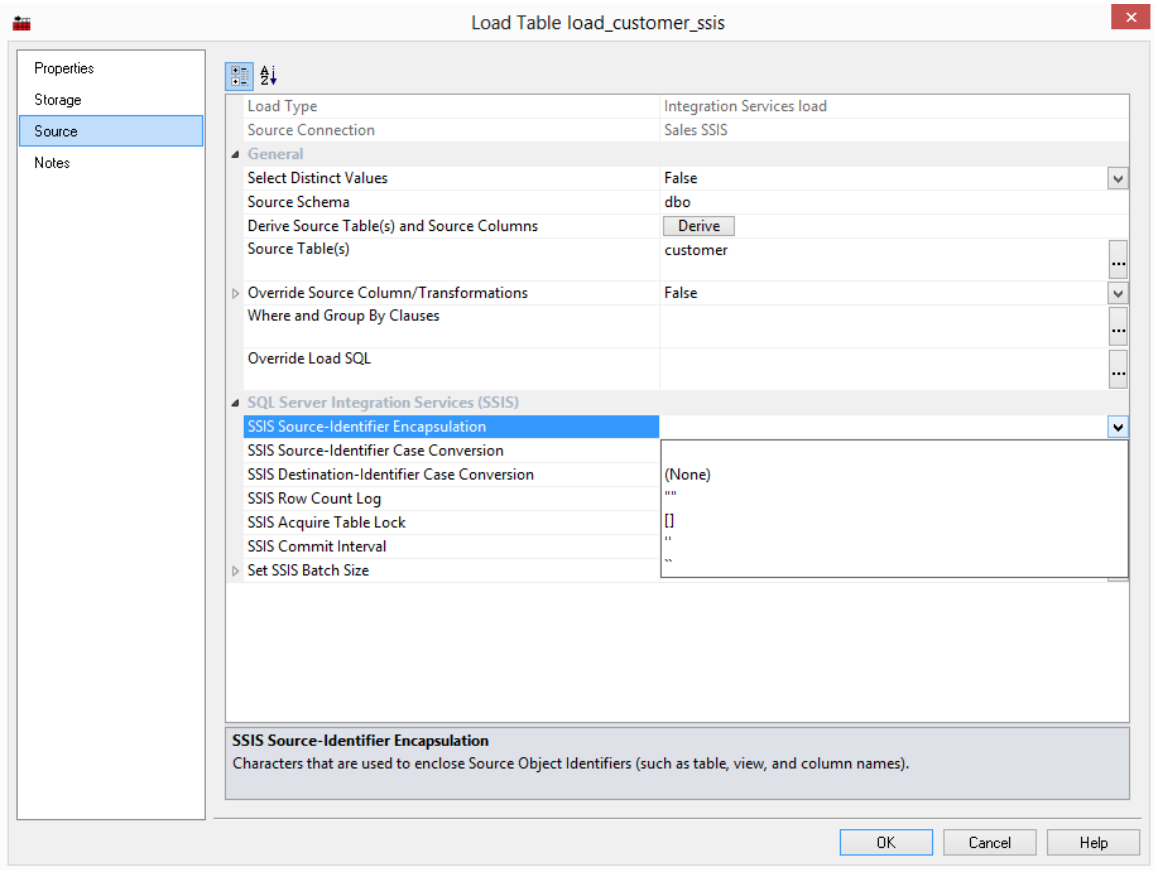
If an SSIS load fails due to incorrect connection strings, then the SSIS package will fail to create at run time. The connection string information in both the source connection and the data warehouse connection (the destination) should be checked to ensure that they are correct.

If the SSIS load fails due to incorrect extraction SQL, due to mismatched destination load table structure or due to constraint violation then the SSIS package will be created in the file system for the developer to check. The SSIS package file will be created in the WhereScape RED Work Directory; shown in the dialog opened from **Help/About WhereScape RED** (refer to dialog below).
This SSIS package will have a DTSX file extension and can be opened using the appropriate version of Microsoft's BI Development Studio add-in to Visual Studio.



The **Integration Services load** in RED does not currently automatically handle spaces in source field names. You need to put [] around the source column on the column Properties of the load table column that contains spaces in Excel. Failure to do so results in SSIS (runtime) crashing and returning an error message. The SSIS package file will be created in the WhereScape RED Work Directory; shown in the dialog opened from **Help/About** WhereScape RED (refer to dialog above).

On the **Source** tab for a load_table, the following drop-down list gives the options for **SSIS Source-Identifier Encapsulation**.
We use encapsulation to ensure that column aliases are not changed during an SSIS load:



NATIVE ODBC BASED LOAD

A Native ODBC based load is similar to an ODBC load, except it provides a faster method to load data from non-linked databases.

A Native ODBC based **interactive load** when using the WhereScape RED tool will use the established ODBC connection to pull the data back to a delimited file on the local PC and then push the data to the data warehouse database via a Bulk Insert in SQL Server, SQL*LOADER in Oracle or LOAD statement in DB2.

A scheduler load will perform in the same way except that the data is loaded into the server that is running the scheduler and then pushed to the data warehouse database.

All loaders require dates in the default format of the target data warehouse database. This is normally achieved via a **During** load transformation using the correct casting function for the source database.

The **Properties** and **Storage** screens for a Native ODBC based load are the same as those of a database link load. Refer to the previous section for details. Details of the source mapping screen follow.

Native Loads in Oracle

Some versions of Oracle do not return the ORACLE_SID correctly when an ODBC call requests it. To ensure that you don't have any issues because of this, please ensure that the following steps are followed:

If you are doing interactive Native Loads in RED into Oracle:

1 Ensure that the **Database id (SID)** field on the Data Warehouse connection Properties screen is set to the actual ORACLE_SID for your database.

This value is used in the sqlldr connection during the native load IF the ORACLE_SID is not returned correctly by the ODBC driver.

If you are using the windows scheduler for Native Loads into Oracle:

1 Change the scheduler to be a direct path load (via maintain scheduler in the WhereScape Administrator).

2 Ensure that the TNS Service is set to the actual ORACLE_SID for your database.

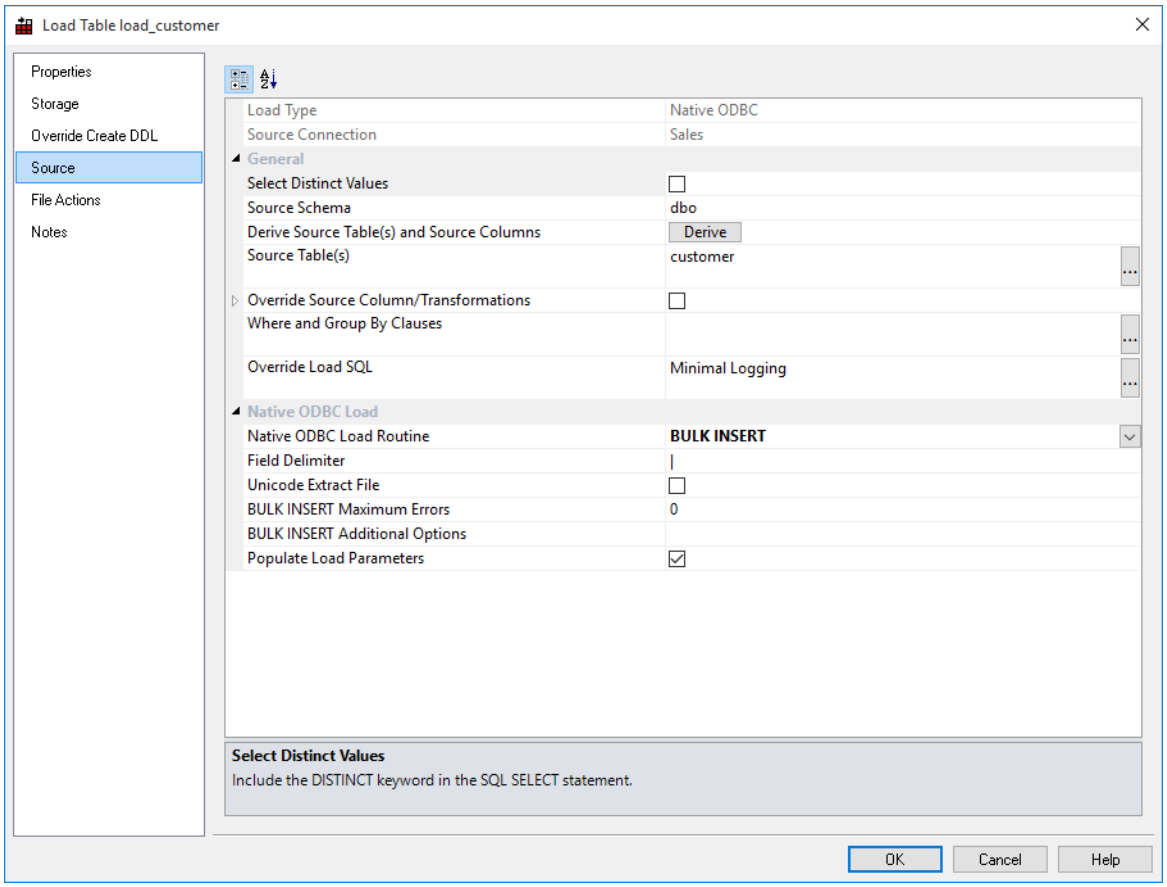
3 Restart the scheduler.

This value is used in the sqlldr connection IF the SID is not returned by the ODBC driver and only IF the scheduler is set to do direct path loads.

NATIVE ODBC BASED LOAD - SOURCE MAPPING

The fields on the Native ODBC Load Source Mapping Screen are described below. **Oracle** and **DB2** Source screen examples can also be found at the bottom of this topic.

SQL Server **Source Screen:**



Load Type

Method of loading data into the table. The available options are dependent on the **Source Connection**. Defaults to the 'Default Load Type' of the **Source Connection**. Can be specified via the Properties page.

Source Connection

The connection that identifies the source database. Can be specified via the Properties page.

General

Select Distinct Values

Include the DISTINCT keyword in the SQL SELECT statement.

Source Schema

Schema within the source database where the source table resides.

Derive Source Tables(s) and Source Columns

Derive the **Source Table(s)** and **Source Column(s)** properties (of this dialog) from the source details of this table's columns.

Note: The existing property values will be overwritten.

Source Table(s)

Name of the table or tables that the data is sourced from.

Override Source Column/Transformations

Ignore the source and transformation details of this table's columns and instead use the override details specified below.

Where and Group By Clauses

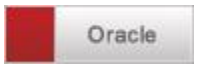
Optional SQL SELECT WHERE-clause and/or GROUP BY-clause. Parameter names can be specified using $Pparameter_name$ (with leading $P and trailing $), which are replaced at run-time by the parameter's value.

 **TIP**: This is where you can build a statement to handle change data.

Override Load SQL

Optional SQL statement to load data into the table, which overrides all other properties. The specified SQL will be executed instead of generated SQL. For a linked database specify a complete INSERT statement. For an ODBC source specify only the SELECT statement.

Source View Name



Oracle only option: Name of the database view to be created by WhereScape RED (if necessary) in the source system that will be used to select the source data. See ***Remote View Extract*** (see "***Remote View Extract - Oracle Databases Only***" on page 245) topic.

Native ODBC Load

Native ODBC Load Routine

File Loader utility/mechanism to use to load the generated extract file.

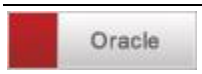
Field Delimiter

Character that separates the fields within each record of the generated extract file. The default value is a | character (pipe). This should be changed if pipes are possible in the source data.

Unicode Extract File

Data will be loaded via a Unicode format file. The default is unchecked and this field is not available FOR DB2.

SQL*Loader Options



Oracle only: use this field to optionally specify an Oracle SQL*Loader OPTIONS clause and/or WHEN clause.
Multiple options can be specified via the OPTIONS clause by separating each option with a comma. The property can be used to specify that SQL*Loader uses the efficient **Direct Path method** to load data into Oracle and/or a WHEN clause to selectively load rows based on a condition.

Examples:
OPTIONS(DIRECT=TRUE) specifies that SQL*Loader uses the Direct Path load method
WHEN (3) = 'Y' specifies that only rows where column 3 contains the value Y are loaded

Note: Use of the the "not" symbol (!) in the WHEN statement is only supported for Linux/UNIX file loads. This symbol is not supported for Windows based loads.

BULK INSERT Maximum Errors

SQL Server only option: Maximum number of errors accepted before a load failure is triggered. The default for this option is 0.

BULK INSERT Additional Options

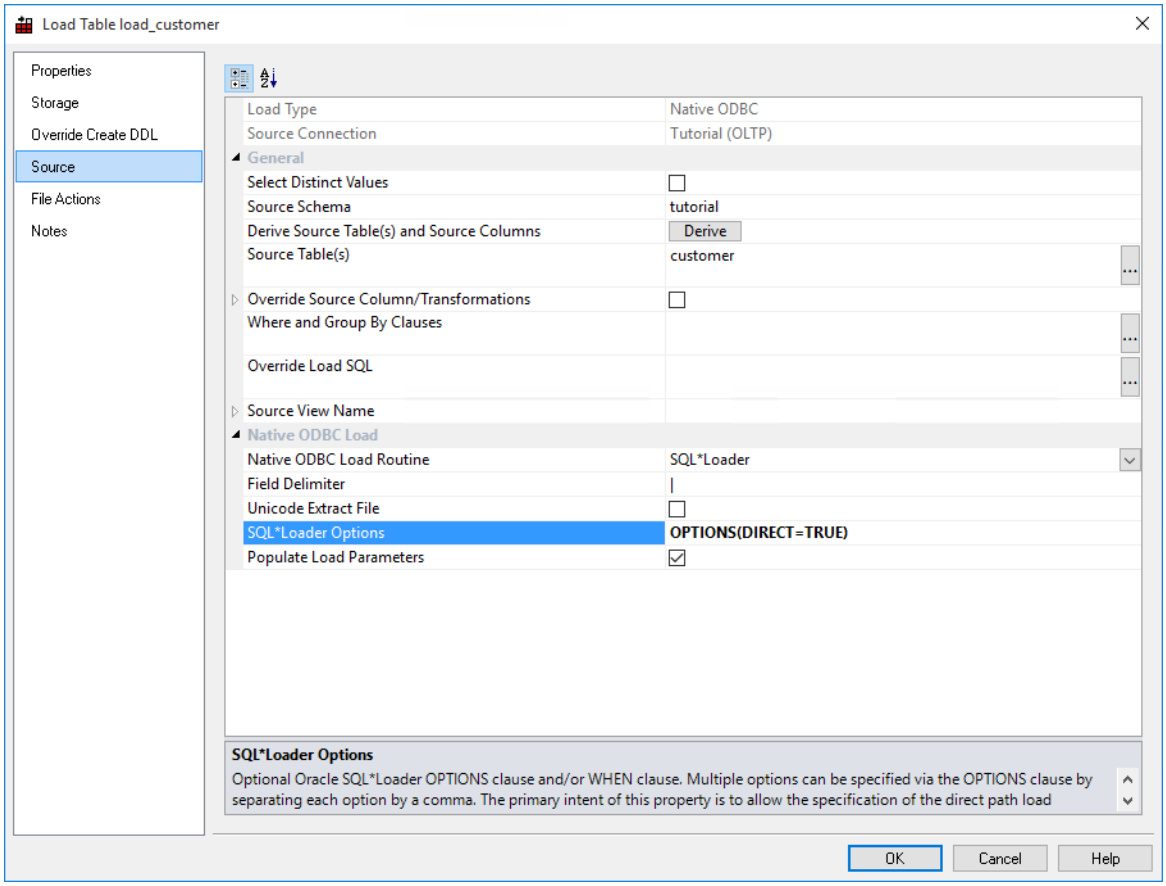
SQL Server only option: Optional comma-delimited list of options to control the behavior of the SQL Server BULK INSERT command.
For example the following options will respectively lock the table; set the batch size to 100,000 rows; and fail the load when the first error occurs: TABLOCK, BATCHSIZE=100000.

Populate Load Parameters

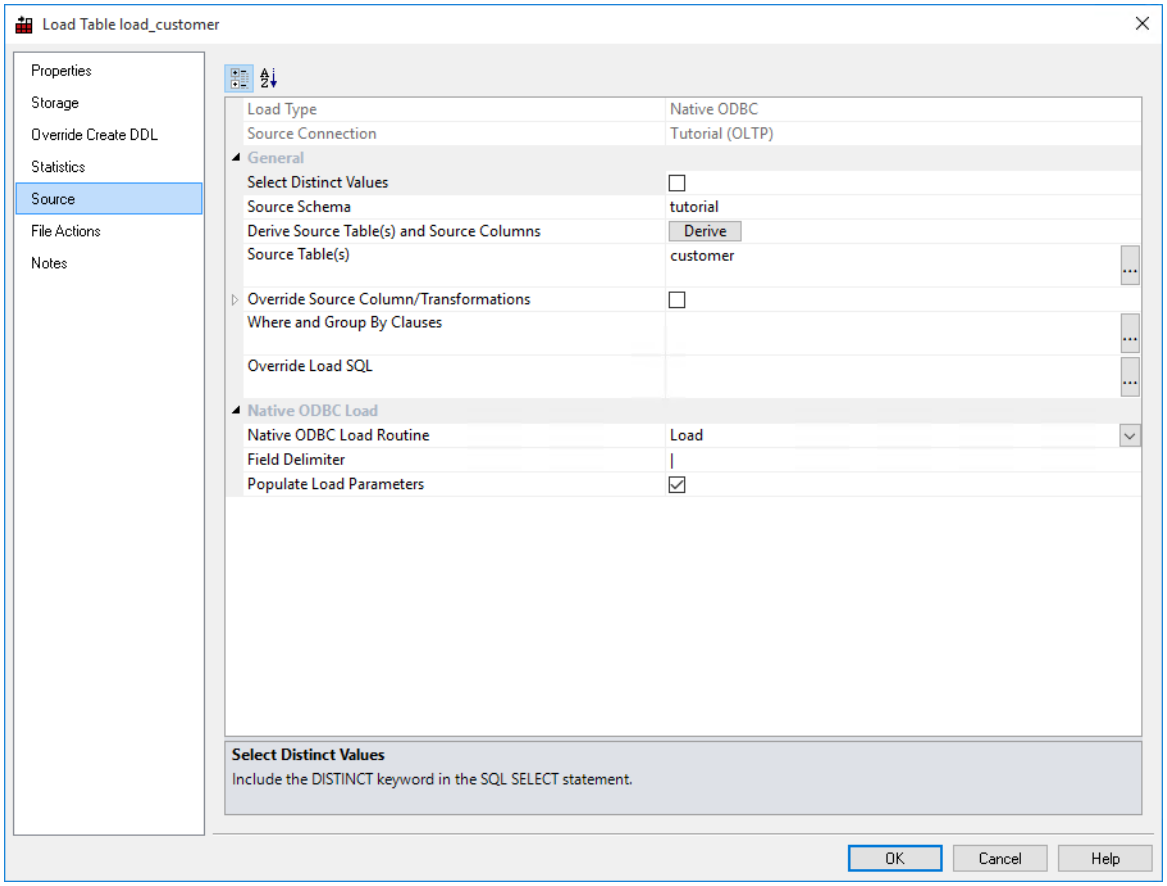
Populate any load-related WhereScape RED parameters, which may be used for validation purposes.

Oracle and DB2 Source screen examples:

Oracle **Source Screen:**



DB2 **Source Screen:**



FILE ACTIONS

A file action defines a copy, move or delete step that happens to **Native ODBC** temporary files after a load is run.
There are six different file action programs:

File Action	Description
copy all files	Copies all temporary files for the load table (including the temporary data file) to another specified directory.
copy data files	Copies the **Native ODBC** load table's temporary data file to another specified directory. Also allows a new file name to be specified.
delete all files	Deletes all temporary files for the load table (including the temporary data file).
delete data files	Deletes the **Native ODBC** load table's temporary data file.
move all files	Moves all temporary files for the load table (including the temporary data file) to another specified directory.
move data files	Moves the **Native ODBC** load table's temporary data file to another specified directory. Also allows a new file name to be specified.

Note: A load table may have more than one file action. File actions are processed in order, based on the action number.

Creating a File Action

To create a file action:

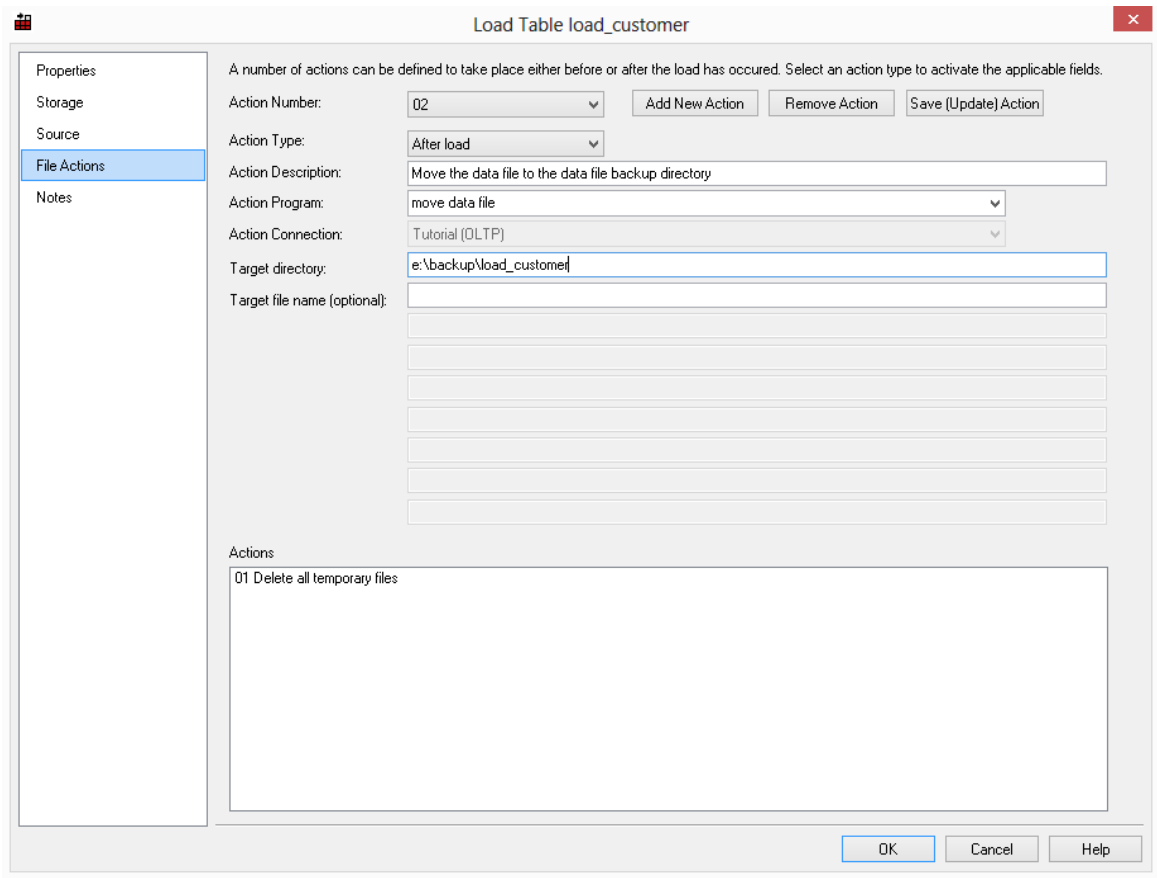
1. Click on the File Actions tab of the load table Properties dialog.
2. Click the **Add new action** button.
3. Choose *After load* from the **Action Type** drop-down list.
4. Enter the **Action Description**.
5. Choose the **Action Program** from the drop-down list (see above for the options).
6. For action programs of copy... and move..., enter the target directory name.
7. For copy data files and move data files, optionally enter the file's new name.
8. Click on **Save/update action**.
9. Repeat if necessary for additional file actions.
10. Click **OK**.

FILE ACTION EXAMPLE

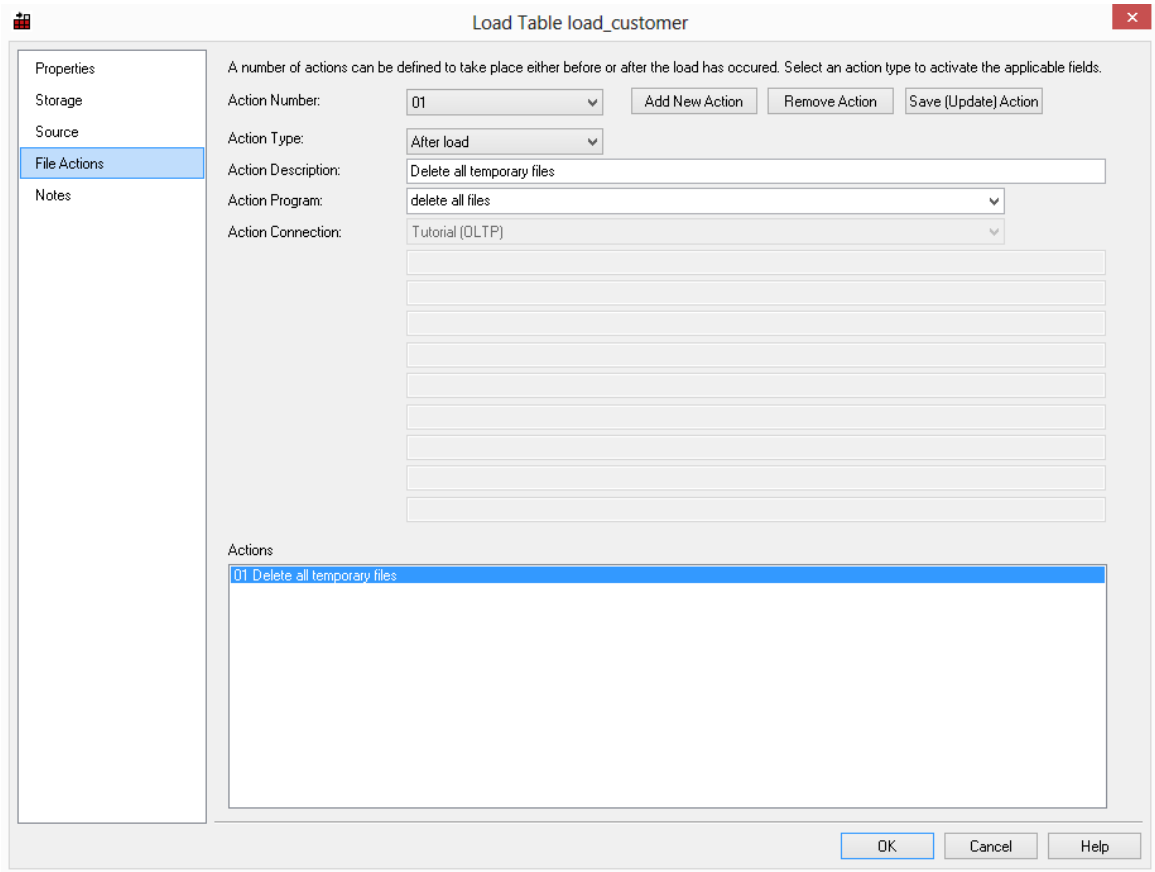
The following example moves the Native ODBC Load data file to another directory then deletes all temporary files.

Note: The data file is not deleted as it has been moved prior to the delete action.

Sample **move data file** file action:



Sample **delete all files** file action:



FLAT FILE LOAD

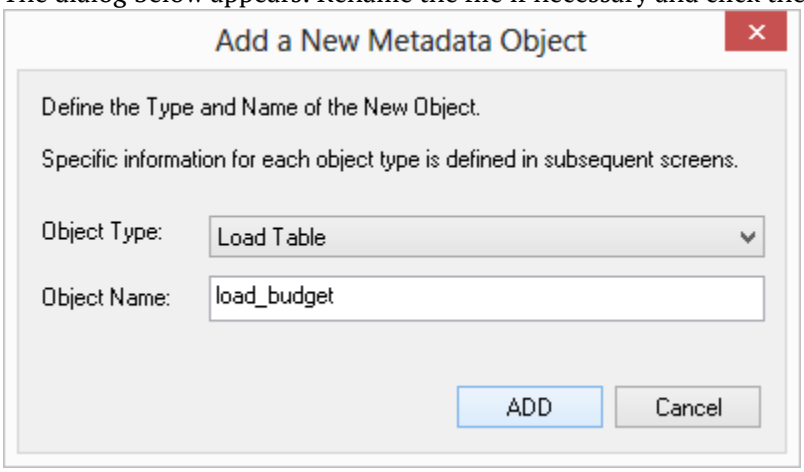
Flat file loads can be performed from either a **Windows**, **UNIX/Linux** or **Hadoop** connection. As with all other load types it is easier to use the drag and drop functionality to create load tables. Flat files can also be loaded using SQL Server Integration Services (SSIS). For Flat File load instructions using SSIS, see the next section – ***Loading Data from a Flat File using SSIS***.

NOTE: To process Greenplum file loads via the Windows Scheduler, please ensure that the **Windows Scheduler user** has the all the necessary permissions including having the **ssh host key** saved.

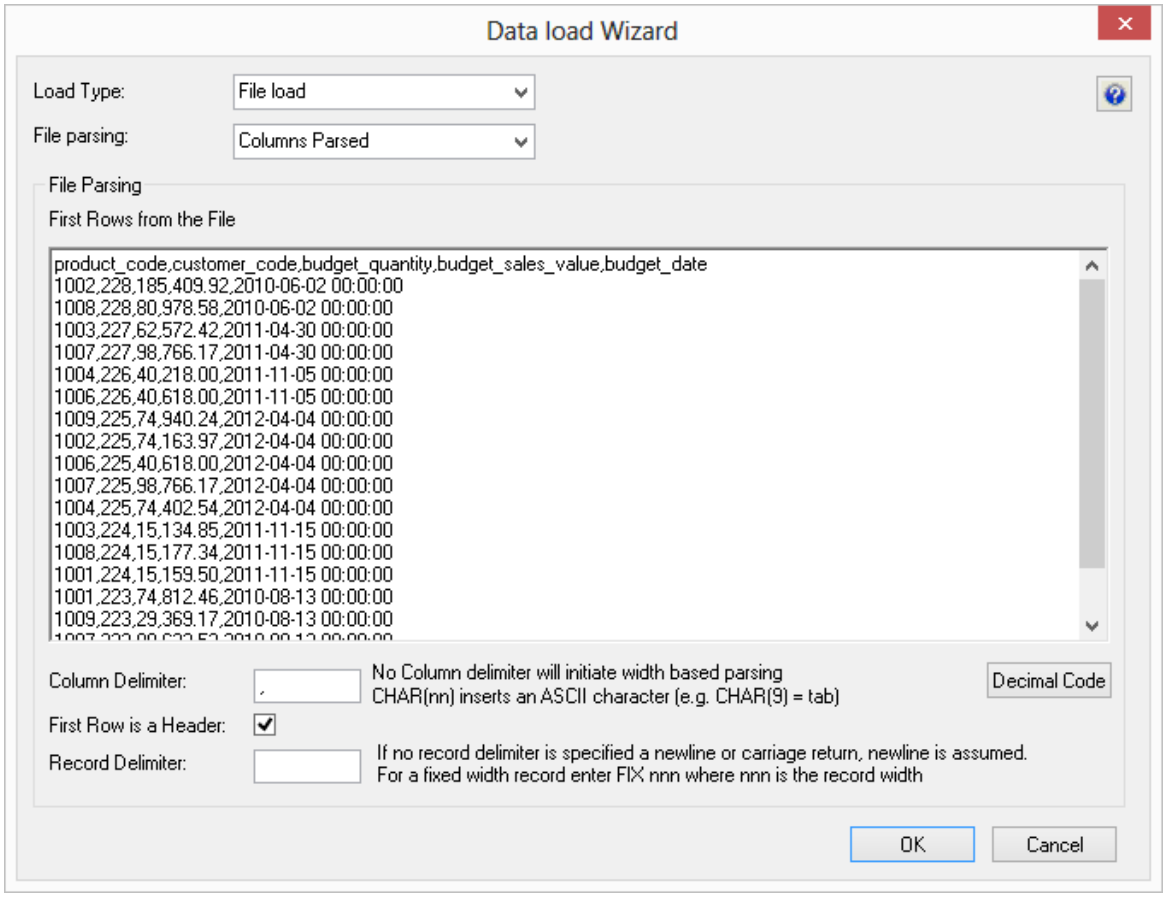
The **drag and drop** process for flat files is as follows:

1. **Browse** to the directory and file via the appropriate ***Connections*** (on page 141).
2. Double-click on the **Load Table** object in the left pane to create a drop target.
3. **Drag** the file from the right pane and **drop** it into the middle pane.

4. The dialog below appears. Rename the file if necessary and click the **ADD** button.



5 The following dialog displays for file loads from **Windows, UNIX/Linux** and **Hadoop** connections.



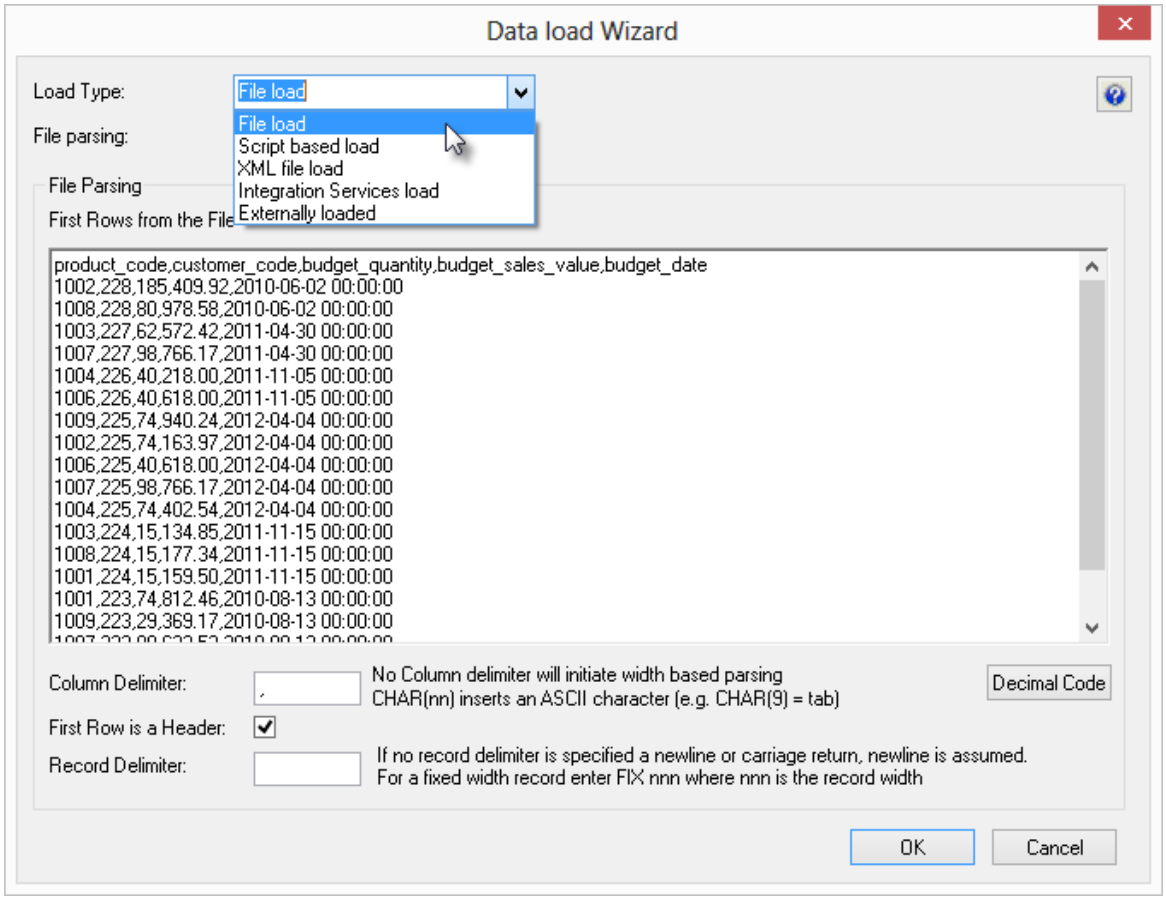
Note: First Row is a Header - To make changes in relational database tables after a table has been defined, users can edit the **First Row is a Header** option in the **Source** tab of the relevant table.

Hive **NOTE:** For File loads into Hive tables, the **First Row is a Header** option is a table option. Please ensure this option is checked in the file load wizard if you want to load files with header rows.
To make any changes after Hive tables have been defined, the **First Row is a Header** option can be found in the **Storage** tab of the relevant Hive table.

---

6 The load type selected in the **New Table Default Load Type** field in the connection dialog will be the pre-selected option in the **Load type** drop-down list.

- To change the desired load type and file parsing, use the **Load type** and **File parsing** drop-down list options.



Load type options

- The **File based load** options results in a load where the bulk of the load management is handled by the scheduler.
- The **Script based load** option will make WhereScape RED generate a host script and the load table will be a **Script based load**. This host script is executed by the scheduler to effect the load. For further details about Scrip based loads see section ***Script based load***.
- The **XML file load** option will only be an available load option from a Windows connection. To see more details about specific XML loads, see section ***XML File load***.
- The **Integration Services load** option will load the file via an Integration Services Package that is generated and executed dynamically at run time. This option is only available from a Windows connection. For specific details about this load option, see section ***Loading Data from a Flat File using SSIS***.
- The **Externally loaded** option will not execute an actual load into the table but will process the actions specified in the Post Load procedure property. Any **After** transformations recorded against any of the columns in an Externally loaded table will also be processed.

File parsing options

- **Single data column** - with this option, the majority of the work in terms of parsing the file must occur in a subsequent procedure within the data warehouse. The data is dumped into a single column. The task of coding a procedure to parse the data must then be undertaken. Three columns are created under Oracle. These include the data column, a sequence column (row_sequence) and the file name column (row_file_name). The file name and sequence columns can be deleted if they are not required for a File based load.
- **Columns parsed** - with this option, WhereScape RED will attempt to parse the columns. You will be asked for details and for the column delimiter. You then step through the columns providing names and data types. WhereScape RED attempts to guess the data type, but it needs to be checked and the field length will probably need to be adjusted. The following screen shot shows the initial file parsed screen.

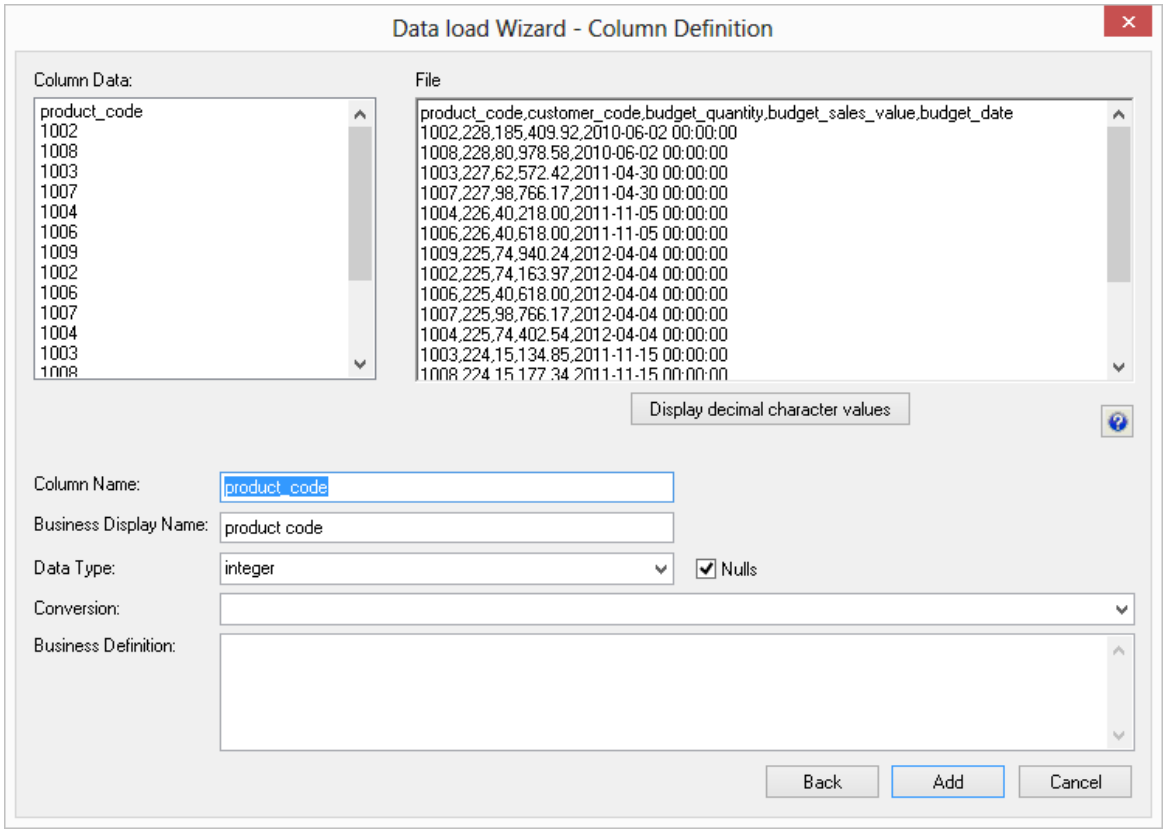
NOTE: DB2 databases do not support data import from files with header row.

NOTE: The **Decimal Code button** will show the decimal value of each character in the lines retrieved from the source file. These decimal codes will be shown below each line and are green.

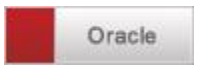
7 Once the selection on the screen above is completed, a screen will display to allow the breakdown of the source data into columns.
If no delimiter is entered, then width based parsing is assumed and an addition width size is prompted for.

The following screen is an example of the file parsing technique.

- Use the **Back** button to revert to the previous column if an incorrect width or delimiter is entered.



Conversion



During the parsing of the columns an Oracle SQL*Loader conversion string can be used and is required in the case of dates for Oracle. Refer to the Oracle Sql loader manual for syntax.
Any Oracle function can be used if enclosed in quotes and with the column name prefixed by a colon. For example: when loading a column called 'product_name', use the following syntax to bring over only the first 30 characters:

```
product_name "substr(:product_name,1,30)"
```

A special variable %FILE_NAME% can be used in an Oracle File based load. This will be substituted with the actual source file name.
To load the full file name (including the path) into a field, enter this transformation:
`"RTRIM('%FILE_NAME%')"`

To remove the path from the full file name on a UNIX or Linux system, enter this transformation:
`"SUBSTR('%FILE_NAME%',INSTR('%FILE_NAME%','/',-1,1)+1)"`

For SQL Server only **After Load transformations** can be processed. The conversion string can be entered in the relevant column conversion field during the parsing of the columns, however, to process the column conversion, users will need to do the following after the table is created and loaded:
1. Go into the **Properties** of the loaded table's relevant column(s) by double-clicking on the column(s) in the middle pane.
2. Click the **Transformation** tab.
3. Select the **After load** option in the **Transformation Stage** drop-down list.
4. Recreate the load table.

Note: If running from the windows scheduler then a wildcard must be used in the file name if you wish to use the %FILE_NAME% variable. If a wild card is not used then the literal file name appears enclosed in double quotes. When a wild card is used Windows looks up the file and returns the actual file name with no enclosing double quotes.

Understanding SQL*Loader

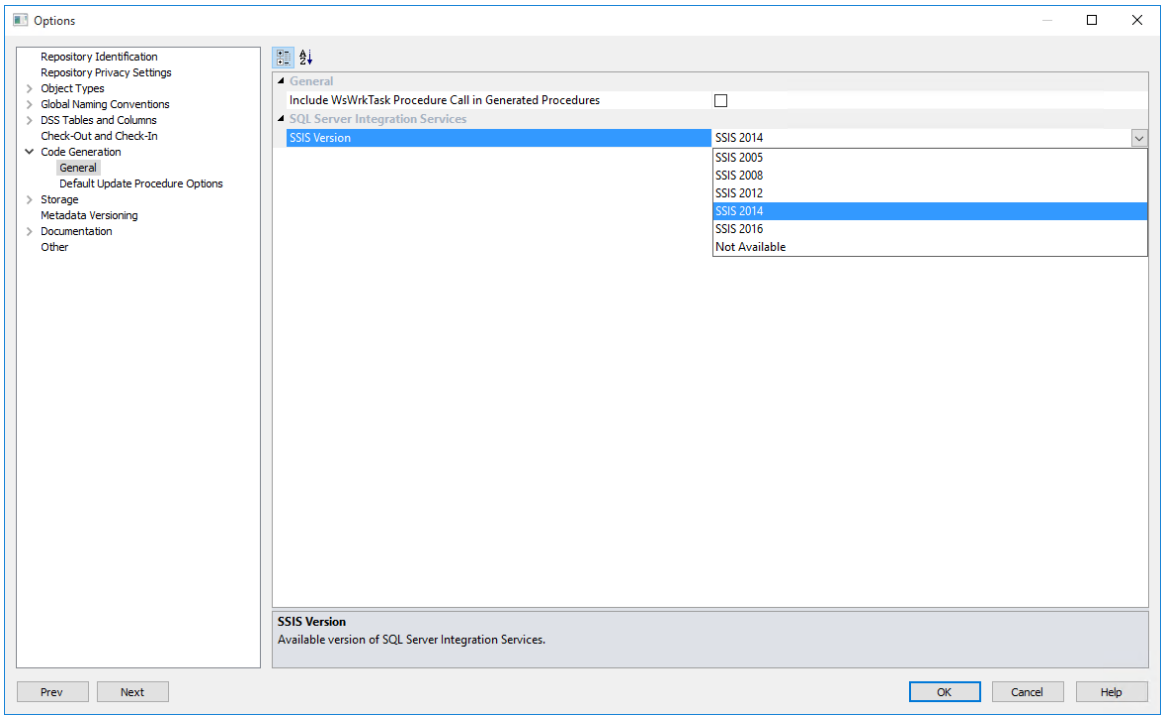
When using either type of loading method **SQL*Loader** is called to perform the actual load. There is therefore a requirement that the executable called **'sqlldr'** be in the path.
Under **UNIX** this means that the UNIX user must be able to execute 'sqlldr' from the command prompt under a normal log on.
Under **Windows** 'sqlldr' must execute when the command is issued from a DOS prompt.
In both the Windows and UNIX environments sqlldr returns a result code indicating if the load worked fully, partially or failed. WhereScape RED makes use of this returned code to report on the results of the load.

LOADING DATA FROM FLAT FILES USING SSIS

Flat files can be loaded into RED from a **Windows** Connection using SQL Server Integration Services (SSIS).

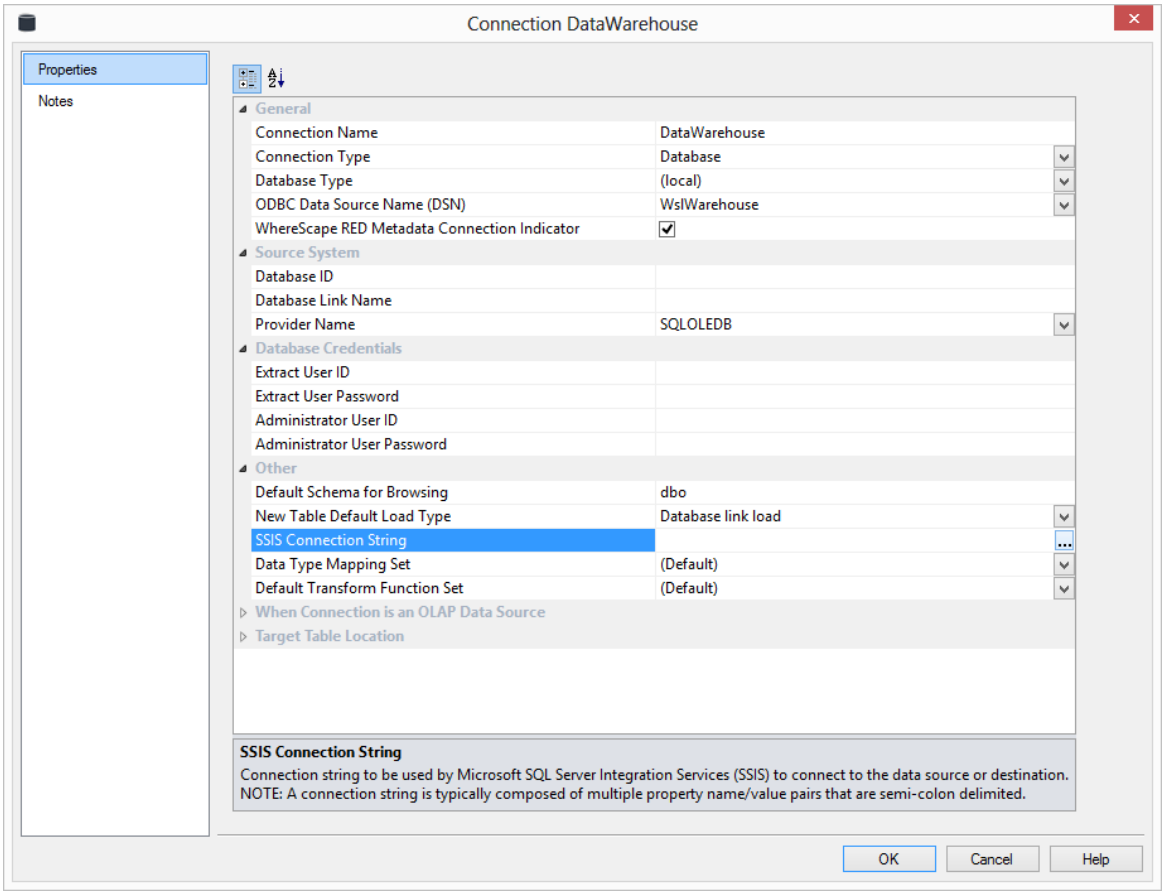
The instructions below detail how to add the SSIS connection string to the data warehouse connection and load flat files using the drag and drop functionality to create load tables. To load files via SSIS, the SSIS connection string must be defined in the DataWarehouse connection.

To use SSIS loading, ensure that SSIS loads are enabled and that the SSIS version is set correctly in **Tools/Options/Code Generation /General**.

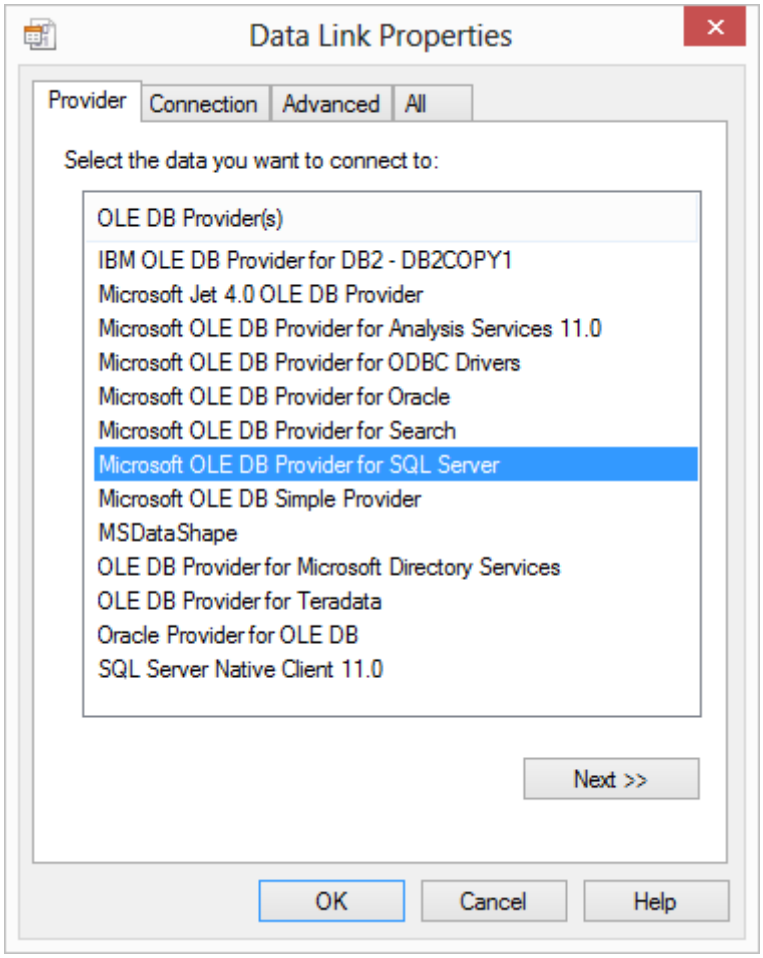


1 To load files via SSIS, the SSIS connection string must be defined in the DataWarehouse connection for the **Destination** connection to be specified:

- Double-click on the **DataWarehouse** connection in the object explorer to open the Properties dialog.
- Click on the ellipsis button to open the wizard and define the SSIS connection string.


Connection DataWarehouse
Properties
Notes
General
Connection Name
DataWarehouse
Connection Type
Database
Database Type
(local)
ODBC Data Source Name (DSN)
WslWarehouse
WhereScape RED Metadata Connection Indicator
Source System
Database ID
Database Link Name
Provider Name
SQLOLEDB
Database Credentials
Extract User ID
Extract User Password
Administrator User ID
Administrator User Password
Other
Default Schema for Browsing
dbo
New Table Default Load Type
Database link load
SSIS Connection String
Data Type Mapping Set
(Default)
Default Transform Function Set
(Default)
When Connection is an OLAP Data Source
Target Table Location
SSIS Connection String
Connection string to be used by Microsoft SQL Server Integration Services (SSIS) to connect to the data source or destination.
NOTE: A connection string is typically composed of multiple property name/value pairs that are semi-colon delimited.
OK
Cancel
Help

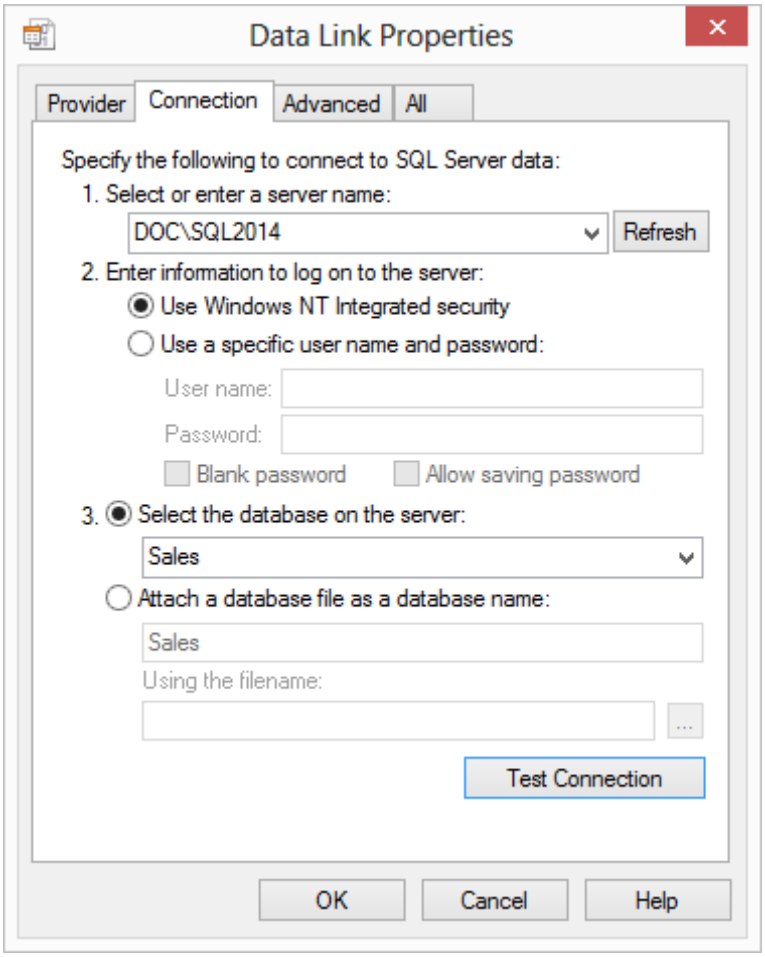
2 On the Provider tab, select the relevant **OLE DB Provider** and click **Next**.



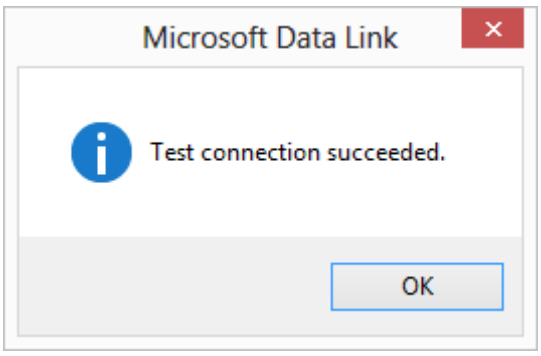
3 On the **Connection** tab, select the **server name**, enter the information to log on to the server and select the **database** on the server. Click **Test Connection**.

NOTE: When using a specific **user name** and **password** to connect to the server instead of using Windows integrated security, the **Allow saving password** check-box must be ticked.

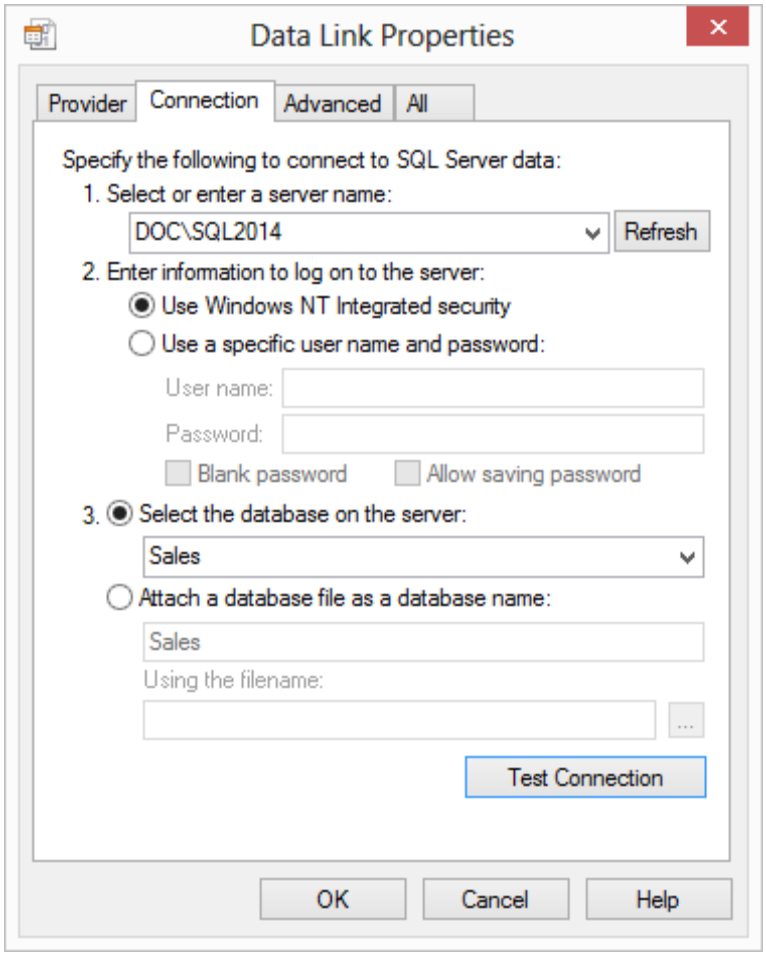
It is also recommended that the password on the SSIS connection string field that is displayed in the connection properties is replaced with the **$PASSWORD$** token that is substituted at runtime.


Data Link Properties
Provider
Connection
Advanced
All
Specify the following to connect to SQL Server data:
1. Select or enter a server name:
DOC\SQL2014
Refresh
2. Enter information to log on to the server:
Use Windows NT Integrated security
Use a specific user name and password:
User name:
Password:
Blank password
Allow saving password
3. Select the database on the server:
Sales
Attach a database file as a database name:
Sales
Using the filename:
Test Connection
OK
Cancel
Help

4 Click **OK**.


Microsoft Data Link
Test connection succeeded.
OK

5 Click **OK** on the Data Link Properties dialog to save the SSIS connection string settings.



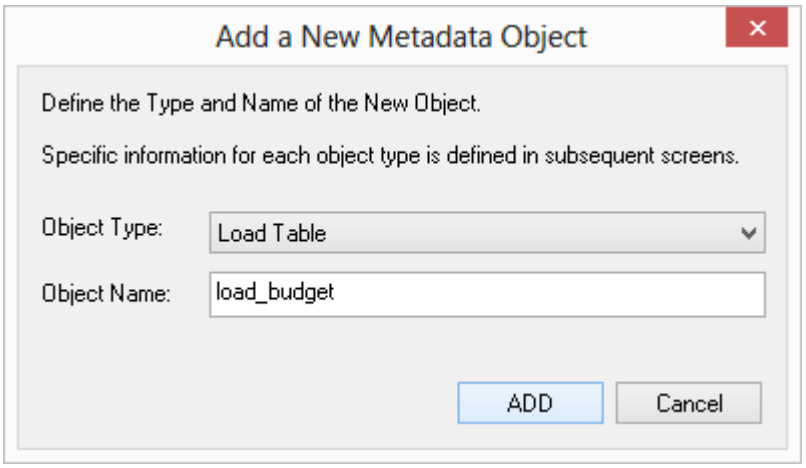
6 Click OK on the Data Warehouse connection properties screen.

7 Browse to the directory and file from the **Windows** connection.

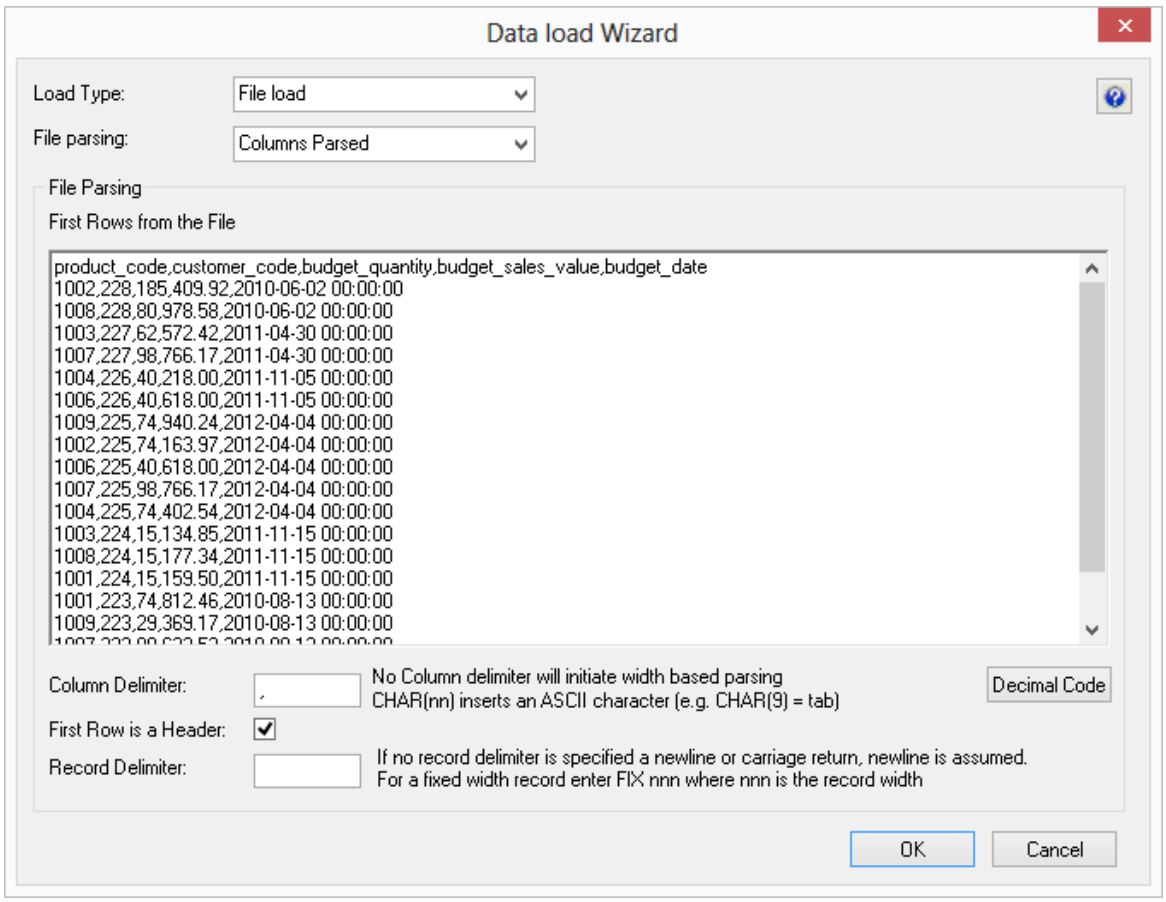
8 Double-click on the **Load Table** object in the left pane to create a drop target.

9 Drag the file from the right pane and drop it into the middle pane. The dialog below appears.

10 Click the **ADD** button.



11 The following dialog displays:



12 There are four options on this screen (buttons at right).

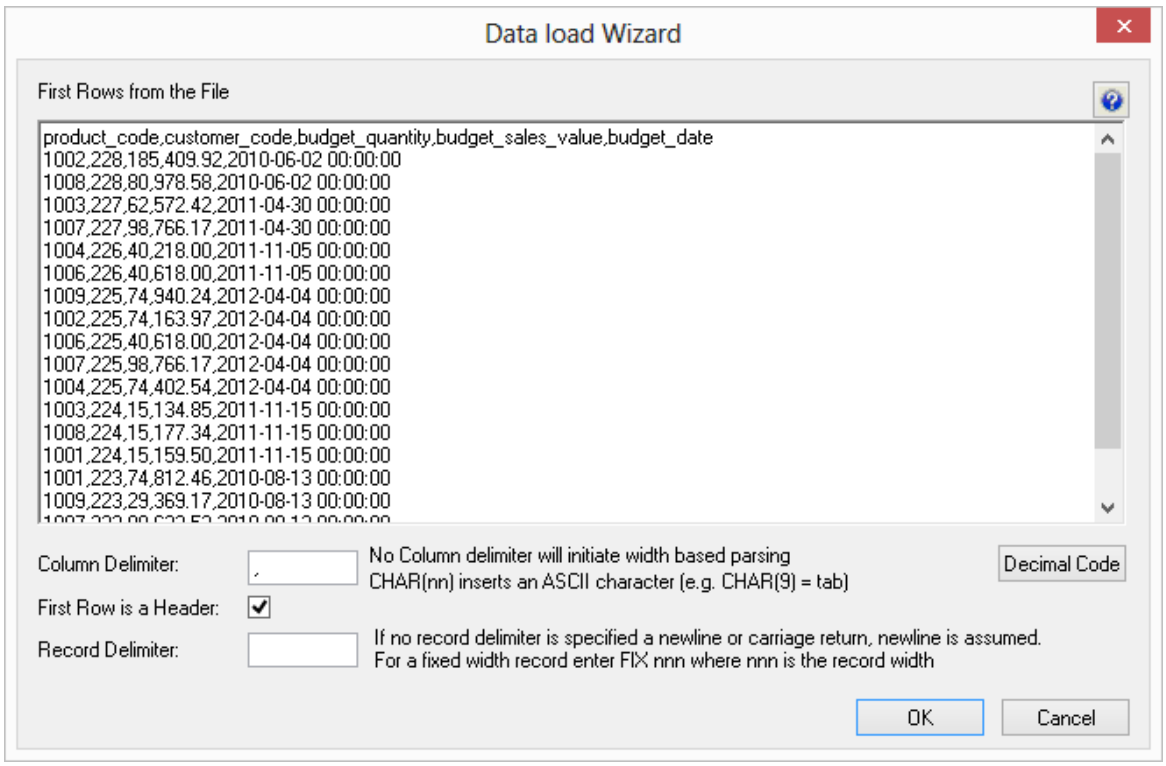
- The first two options result in a **File based load** where the bulk of the load management is handled by the scheduler.
- If you select either of the last two options, then WhereScape RED will generate a host script and the load table will be a **Script based load**. This host script is executed by the scheduler to effect the load.

Single data column

As stated beside the button, the majority of the work in terms of parsing the file must occur in a subsequent procedure within the data warehouse. The data is dumped into a single column. The task of coding a procedure to parse the data must then be undertaken. Three columns are created under Oracle. These include the data column, a sequence column (row_sequence) and the file name column (row_file_name). The file name and sequence columns can be deleted if they are not required for a File based load.

Columns parsed

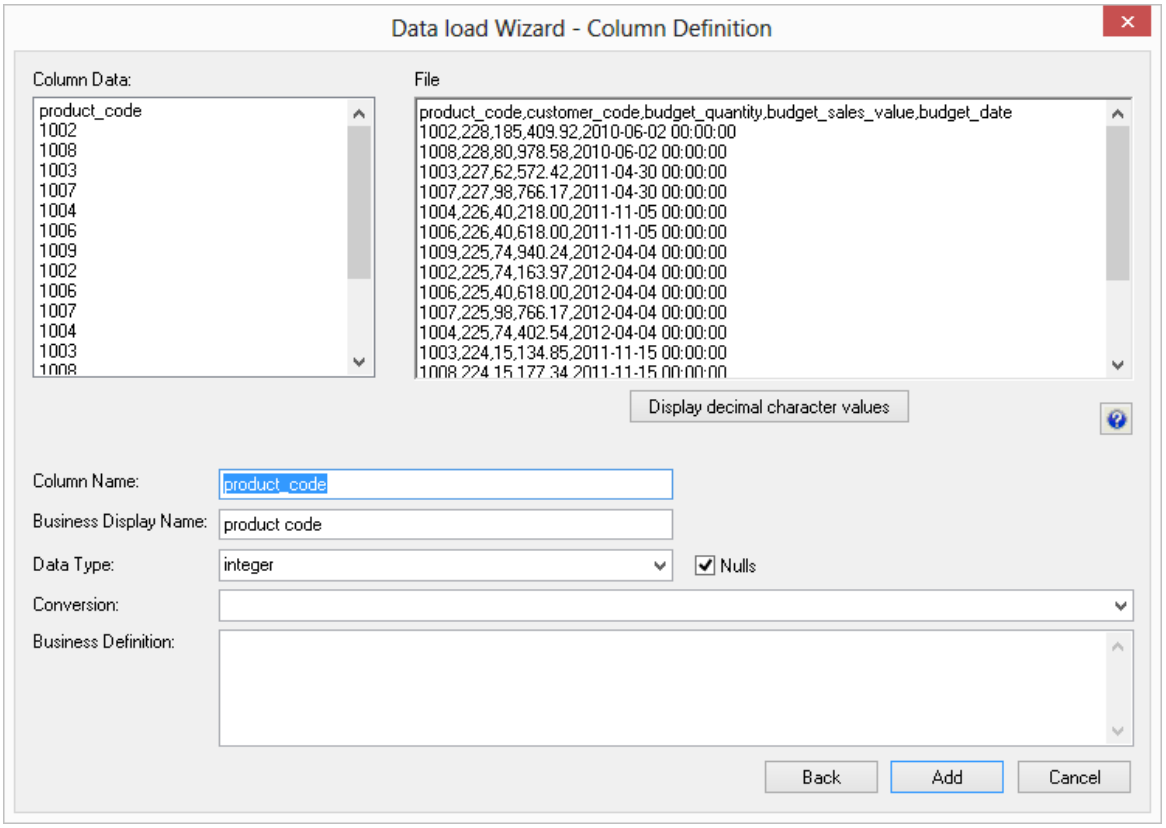
WhereScape RED attempts to parse the columns. You will be asked for details and for the column delimiter.
You then step through the columns providing names and data types. WhereScape RED attempts to guess the data type, but it needs to be checked and the field length will probably need to be adjusted.
The following screen shot shows the initial file parsed screen.



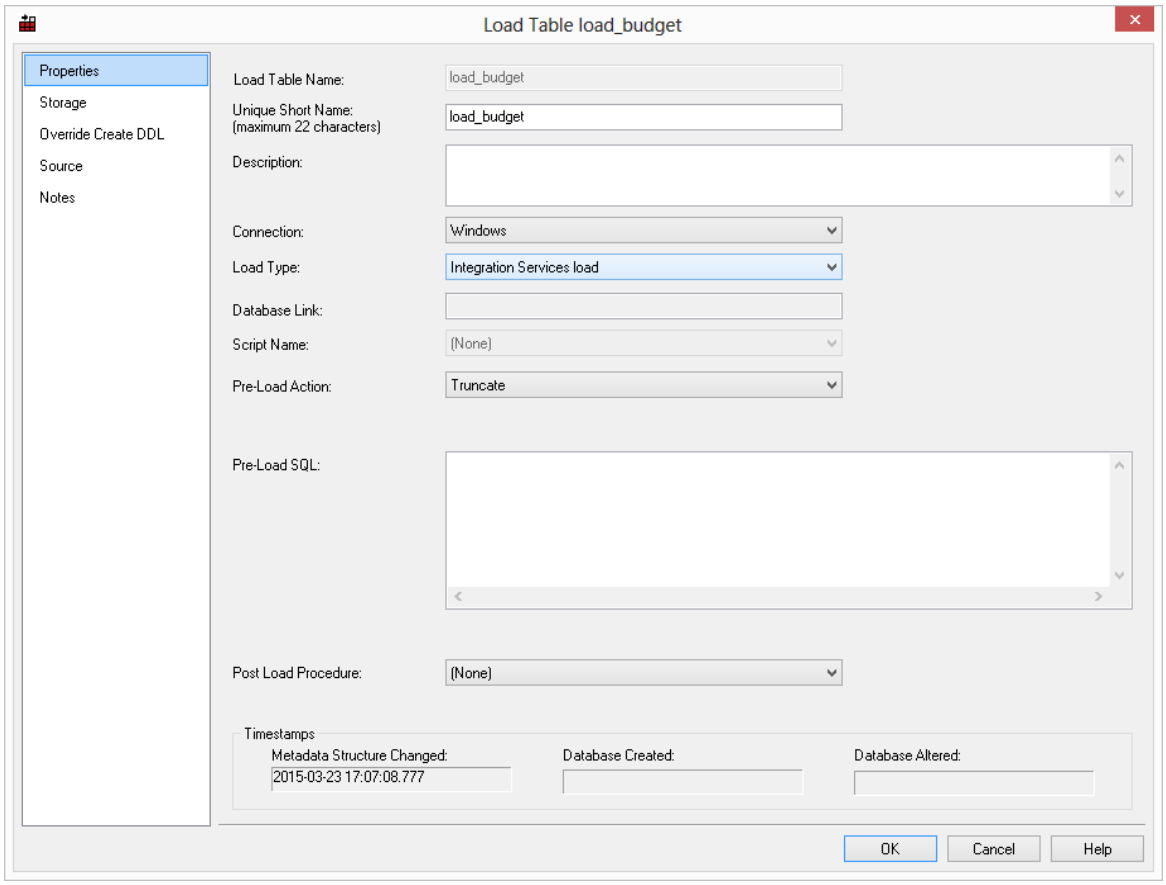
NOTE: DB2 databases do not support data import from files with header row.

The **Decimal Code** button will show the decimal value of each character in the lines retrieved from the source file. These decimal codes will be shown below each line and are green

13 Once the screen above is completed a screen will appear to allow the breakdown of the source data into columns. If no delimiter is entered, then width based parsing is assumed and an addition width size is prompted for.
Use the **Back** button to revert to the previous column if an incorrect width or delimiter is entered.
The following screen is an example of the file parsing technique.



14 On the **Properties** screen for the new load table, select **Integration Services Load** as the Load Type. Click **OK**.
This will create and execute a SSIS package at run time to load data into the data warehouse load table.



NOTE: If the table is changed to an Integration Services load and has been set up using the wizard for the "File load (columns parsed)" flow, some columns might have transformations set up that will not work.

In RED 6.8.3.4 date/time fields have transformations that are invalid for SSIS and will make the load fail.
Since SSIS does not provide any configuration for the parsing of date/time fields, if users have any date/time field special requirements, file or script-based loads provide are a better load option instead.

15 Click **Yes** to Create and Load the table.

FLAT FILE LOAD - SOURCE SCREEN

The fields for the **Flat File Source Screen** are described below. See example source screens in the next section ***File and Script based load Source screens***.

TIP: If the file has been ***dragged and dropped*** (see "***Flat File Load***" on page 274) into the load table (middle pane) then some of the fields on this tab are automatically populated.

Load Type

Method of loading data into the table. The available options are dependent on the **Source Connection**. Defaults to the 'Default Load Type' of the **Source Connection**. Can be specified via the Properties page.

Source Connection

Connection to the data source (database or file system). Can be specified via the **Properties** page.

Load Script Template

Available for script loads only and only if there is a valid template available. Select the template to use when generating a load script, or select (None) to use RED's built-in load script generator. Only templates with the correct Type and Target DB will appear in this drop-down list. For more information on templates, see ***Templates*** (on page 726).

Source File Details

Source File identification and definition information.

Source Directory

The full path (absolute path) of the folder/directory containing the Source File on the Windows or UNIX/Linux system.

Source File Name

The name of the source file containing the data to be loaded.

Source File Character Encoding

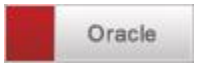
PDW option only: The character encoding of the input datafile for PDW file loads. Select between **ASCII**, **UTF8**, **UTF16** or **UTF16BE**. The **UTF8** encoding cannot be specified for a fixed width source file (i.e. when the Source File Field Delimiter is blank) because UTF8 is a variable length character encoding.

Source File Field Delimiter

Optional character that separates the fields within each record of the Source File. The delimiter identifies the end of each field. Common field delimiters are tab, comma, colon, semi-colon, pipe, tilde. If no field delimiter is specified, the record is regarded as fixed-width.

Note: If an ASCII character value is used this field may show as an unprintable character. To enter a special character enter the uppercase string CHAR with the ASCII value in brackets (e.g. CHAR(9)).

Source File Field Enclosure Delimiter



Oracle and PDW option only: Optional character that delimits BOTH the start & end of a field value i.e. encapsulates a value. A double-quote(") is a common enclosure delimiter. WhereScape RED allows the specification of one enclosure delimiter that is used for both the start & end delimiters. Everything between the start & end enclosure delimiters is considered part of the field value. Enclosing a field's value is OPTIONAL but it must either be enclosed by BOTH a start and an end enclosure delimiter or BOTH of them must be absent. A field value can contain the enclosure delimiter character as embedded data if each embedded instance is "escaped" by the same character.

For **PDW** loads, the delimiter can be specified either as a character or as a hex value (such as 0x22 for a double quote).

Source File Record Terminator

Optional string to identify how each line/record in the Source File is ended/terminated/delineated. The system default is used when not specified. On UNIX/Linux systems, end-of-line is typically line-feed (ASCII 10). On Windows systems, end-of-line is typically carriage-return (ASCII 13) and line-feed (ASCII 10).

Source File has Field Headings/Labels

Indicates whether the first line of the Source File contains a heading/label for each field, which is not regarded as data so it should not be loaded.

Trigger File Details

Optional Trigger File identification and definition information. If a Trigger File is specified the Source File will not be loaded until the Trigger File is available.

Trigger File Path

The purpose of the trigger file is to indicate that the copying/loading of the main file has completed and that it is now safe to load the file. Secondly the trigger file may contain control sums to validate the contents of the main load file. This field should contain the full path name to the directory in which a trigger file is located on the Windows or UNIX systems. If this field and/or the **Trigger Name** field is populated, then the scheduler will look for this file rather than the actual load file.

Trigger File Name

Refers to the name of the file that is used as a trigger to indicate that the main load file is available. A trigger file typically contains check sums. If the trigger file exists then this is the file that is used in the Wait Seconds, and Action on wait expire processing. (see notes under Trigger Path above).

Trigger File Field Delimiter

If the trigger file provides control information then the delimiter identifies the field separation, e.g., or **n** for a return. The data found will be loaded into parameter values whose names will be prefixed by the prefix specified and numbered **0** to **n**.

Trigger Parameter Name Prefix

If a trigger file and delimiter have been specified, then the contents of the trigger file are loaded into parameters. The parameters will be prefixed by the contents of this field and suffixed by an underscore and the parameter number. These parameters can be viewed under **Tools/Parameters** from the WhereScape RED menu bar. The checking of these parameters can be achieved by generating a **Post Load procedure**. An example set of parameters may be budget_0, budget_1 and budget_2 where there are 3 values in the trigger file and the prefix is set to 'budget'.

Wait for Source File or Trigger File

Controls whether the load process waits for the file to arrive when it is NOT available to load. Enabling this allows the wait-related properties to be specified. When a Trigger File is specified the load waits for it rather than the Source File.

Wait Limit (in seconds)

Maximum duration to wait when no file is available, which is specified in seconds e.g. 1800 seconds to wait up to 30 minutes. A value of 0 equates to no wait. If the wait time expires and the specified file cannot be found, then the load will exit with the status defined in Action, e.g., **Default Action = Error**

Action when Wait Limit Reached

Action to take when the Wait Limit has been reached and no file is available. The specified action impacts any remaining tasks in the currently running job. When 'Success' or 'Warning' is specified, the WhereScape RED Scheduler will continue to run any dependent tasks in the running job. In contrast, specifying the 'Error' or 'Fatal Error' actions will cause the scheduler to stop/fail the job and hold any remaining tasks when the "Wait Limit" is reached.

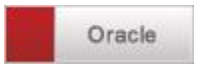
Load Configuration

Configuration details to control the load processing.

BULK INSERT Options

SQL Server only: Optional comma-delimited list of options to control the behavior of the SQL Server BULK INSERT command. For example, the following options will respectively lock the table; set the batch size to 100,000 rows; and fail the load when the first error occurs: TABLOCK, BATCHSIZE= 100000, MAXERRORS=0.

SQL*Loader Options

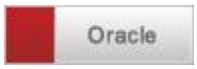


Oracle only: use this field to optionally specify an Oracle SQL*Loader OPTIONS clause and/or WHEN clause.
Multiple options can be specified via the OPTIONS clause by separating each option with a comma. The property can be used to specify that SQL*Loader uses the efficient **Direct Path method** to load data into Oracle and/or a WHEN clause to selectively load rows based on a condition.

Examples:
OPTIONS(DIRECT=TRUE) specifies that SQL*Loader uses the Direct Path load method
WHEN (3) = 'Y' specifies that only rows where column 3 contains the value Y are loaded

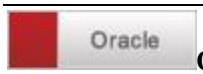
Note: Use of the "not" symbol (!) in the WHEN statement is only supported for Linux/UNIX file loads. This symbol is not supported for Windows based loads.

Hadoop Loader



Oracle only: This option is only available for Script based loads from a Hadoop connection to Oracle. It allows selecting which Hadoop loader RED will use for loading the source file from Hadoop.
Select one of the two Big Data Connectors, **Oracle Loader for Hadoop** or **Oracle SQL Connector for HDFS**.

Character Set



Oracle only: Oracle compliant Character Name Set that tells SQL*Loader the character set of the input datafile.

DB2 Load Command Parameters

DB2 only: Optional Command Parameter Names/Keywords to control the behavior of the DB2 LOAD command/utility. For example, NONRECOVERABLE specifies that the load transaction is to be marked as un-recoverable and it will not be possible to recover it by a subsequent roll-forward action.

DB2 Load Command File Type Modifiers

DB2 only: Optional File Type Modifiers to control the behavior of the DB2 LOAD command/utility. For example, TIMESTAMPFORMAT="YYYY-MM-DD HH:MM:SS.UUUUUU"

Archived File Details

Optional Archived file identification and definition information. Once a file has been successfully loaded or processed it can be optionally archived by moving it and/or renaming it.

Compress Source File when Archive

Optionally compresses the successfully loaded Source File if it is archived.

Archived Source File Path

Optional full path (absolute path) of the folder/directory to MOVE the successfully loaded Source File on the Windows or UNIX/Linux system.

Archived Source File Name

Optional new name to RENAME the successfully loaded Source File to. By default, the original file name is used it is optional to rename it. However, the in-built variable $SEQUENCE$ can be used to include a unique sequence number in the new name. Likewise, the in-built variables $YYYY$, MM, DD, HH, MI and/or SS can be used to include the number of the current year, month, day, hour, minute and/or second respectively. The date/time components can be used separately or can be combined using one set of enclosing $ such as $YYYYMMDD$.

Archived Trigger File Path

Optional full path (absolute path) of the folder/directory to MOVE the successfully processed Trigger File on the Windows or UNIX/Linux system.

Archived Trigger File Name

Optional new name to RENAME the successfully processed Trigger File to. By default, the original file name is used it is optional to rename it. However, the in-built variable $SEQUENCE$ can be used to include a unique sequence number in the new name. Likewise, the in-built variables $YYYY$, MM, DD, HH, MI and/or SS can be used to include the number of the current year, month, day, hour, minute and/or second respectively. The date/time components can be used separately or can be combined using one set of enclosing $ such as $YYYYMMDD$.

PDW File Load Options

Configuration details specific to PDW File Loads.

Load Mode

Select the loading mode from the list. The available options include: Append/Fast Append/Reload/Upsert. Note that selecting the Upsert Load Mode requires a business key to be defined.

Use Staging database

If this field is set and a staging database is defined in the target connection, then the database will be used for staging, otherwise the target connection's database will be used for staging.
To set a Staging Database in the target connection, see ***Connections - Database***.

Additional Command Line Parameters

Optional additional command line parameters (space separated) for PDW file loads which are are not configured elsewhere. Such additional configurations include: certificate verification (-N), skipping loads of empty files (-se), specifying the batch load size (-b <batchsize>), white space trimming around string fields (-c), converting empty strings to null (-E), and load failure options (-rt value|percentage, -rv <reject_value>, -rs <reject_sample_size>).

SQL Server Integration Services (SSIS)

SQL Server Integration Services (SSIS) Attributes.

SSIS Source-Identifier Case Conversion

Case-sensitivity conversion applied to Source Object Identifiers (such as table, view, and column names) in RED-generated SSIS packages. If no conversion is applied, then the exact case of the identifier defined in the RED metadata is used in SSIS.

SSIS Set Source-Code Page

Enables the SSIS source code page property.

SSIS Source Code Page

Use this field to specify the source code page to use when unavailable for the data source.

SSIS Destination-AlwaysUseDefaultCodePage

Forces the use of the Default CodePage property value when describing character data.

SSIS Set Destination-Code Page

Enables the SSIS destination code page property.

SSIS Destination-Code Page

Use this field to specify the destination code page to use when unavailable for the data source.

SSIS Row Count Log

During an SSIS Load include Row Count logging.

SSIS Acquire Table Lock

Specify whether the destination table is locked during the load. This allows SQL Server to reduce lock escalation overheads during loading and will promote minimal logging in SQL 2008+ when a Bulk Logged recovery model is used for the database. The default value of this property is true.

SSIS Commit Interval

Specify the batch size that the OLE DB destination tries to commit during fast load operations. The default value of 2147483647 indicates that all data is committed in a single batch after all rows have been processed. If you provide a value for this property, the destination commits rows in batches that are the smaller of (a) the Maximum insert commit size, or (b) the remaining rows in the buffer that is currently being processed.

Set SSIS Batch Size

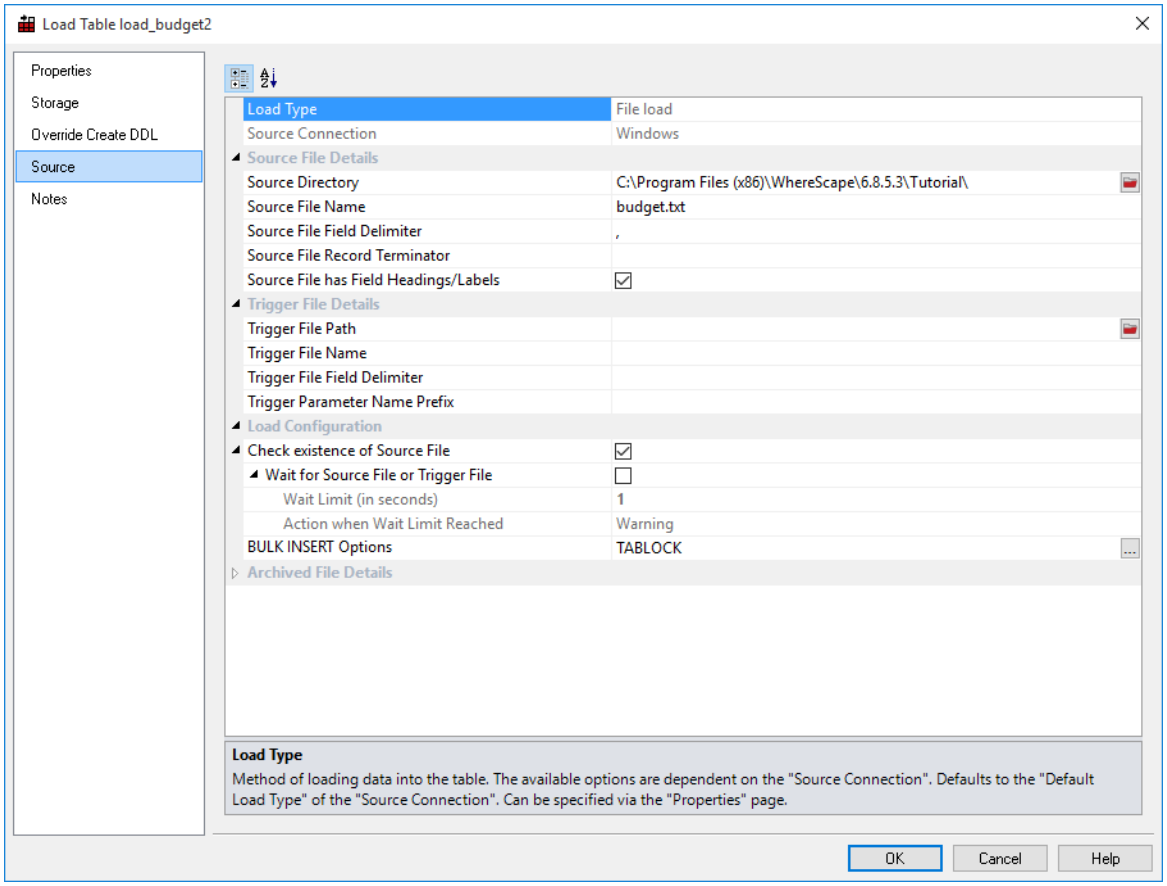
Use this field to specify the number of rows in a batch. The default value of this property is empty, which indicates that no value has been assigned.

SSIS Rows per Batch

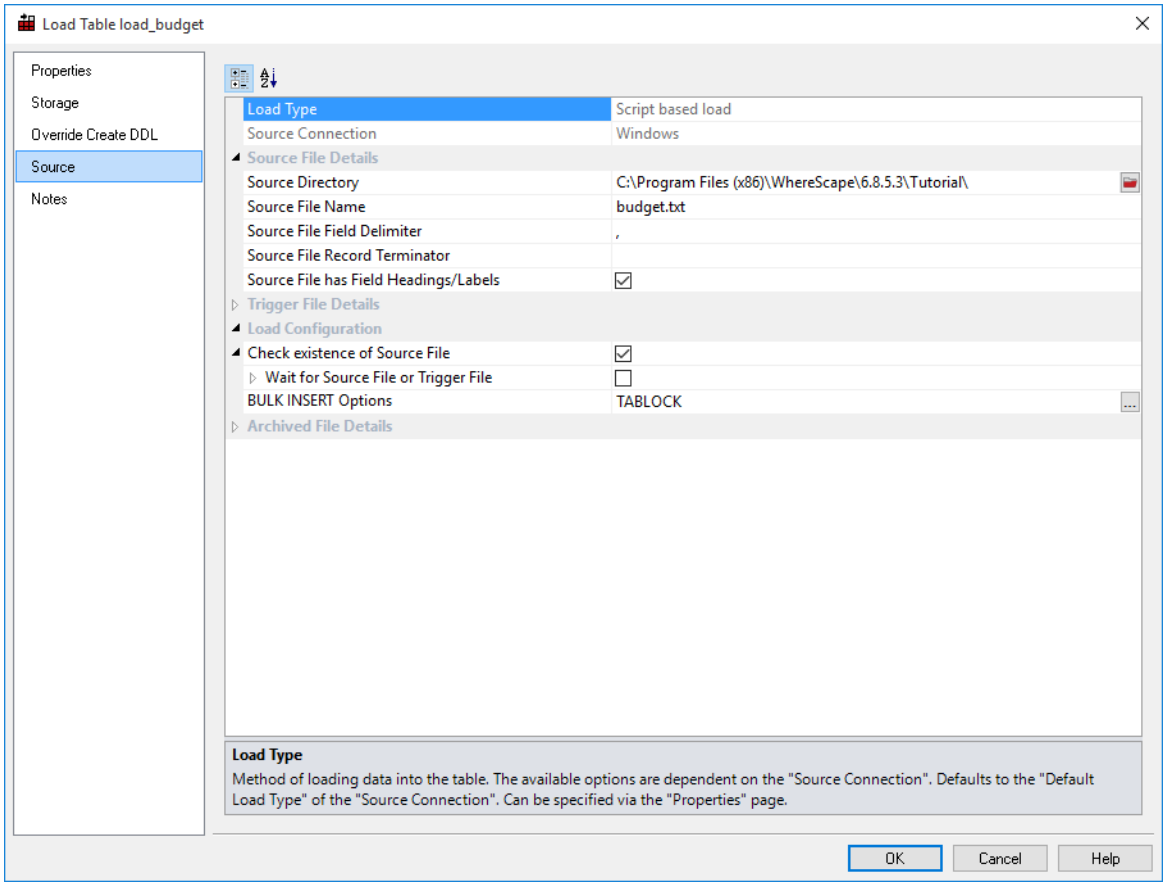
Use this field to define the maximum number of rows in a SSIS batch.

FILE AND SCRIPT BASED LOAD SOURCE SCREENS

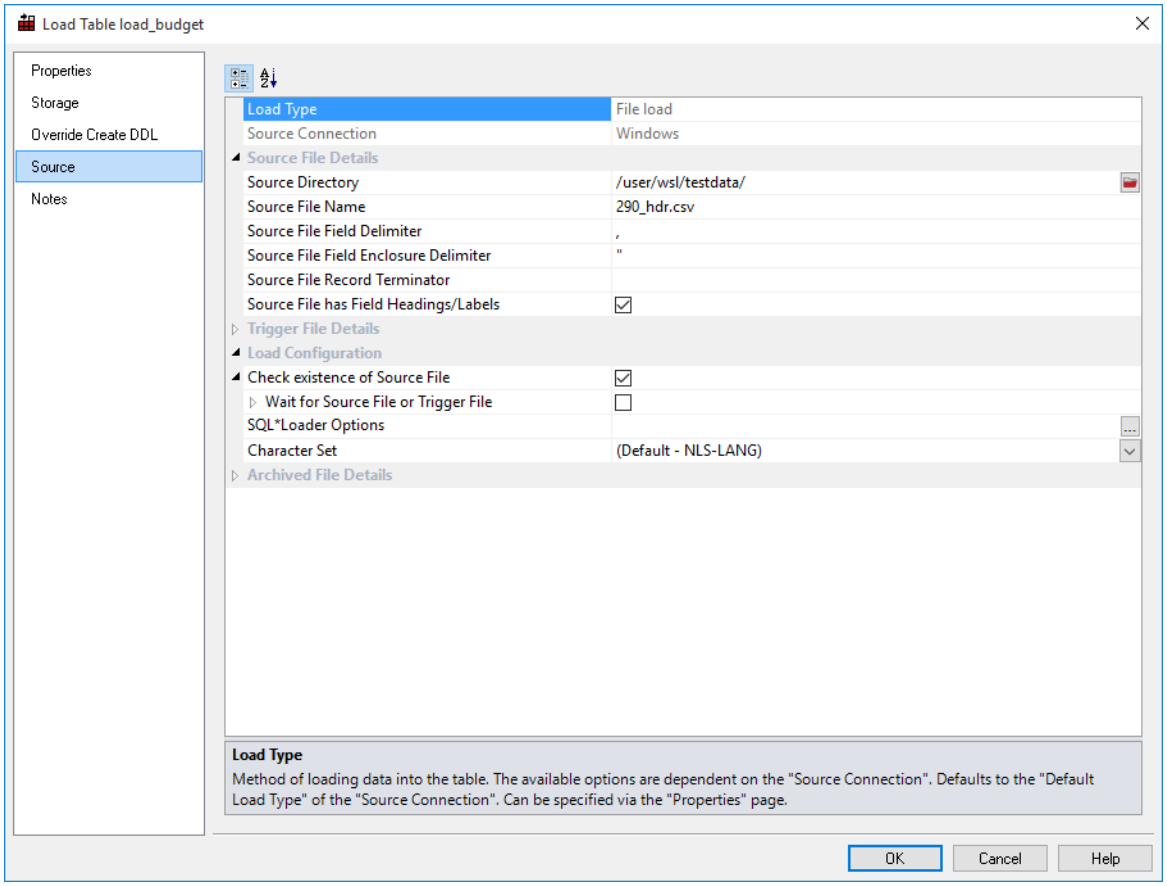
SQL Server **File load Source screen:**



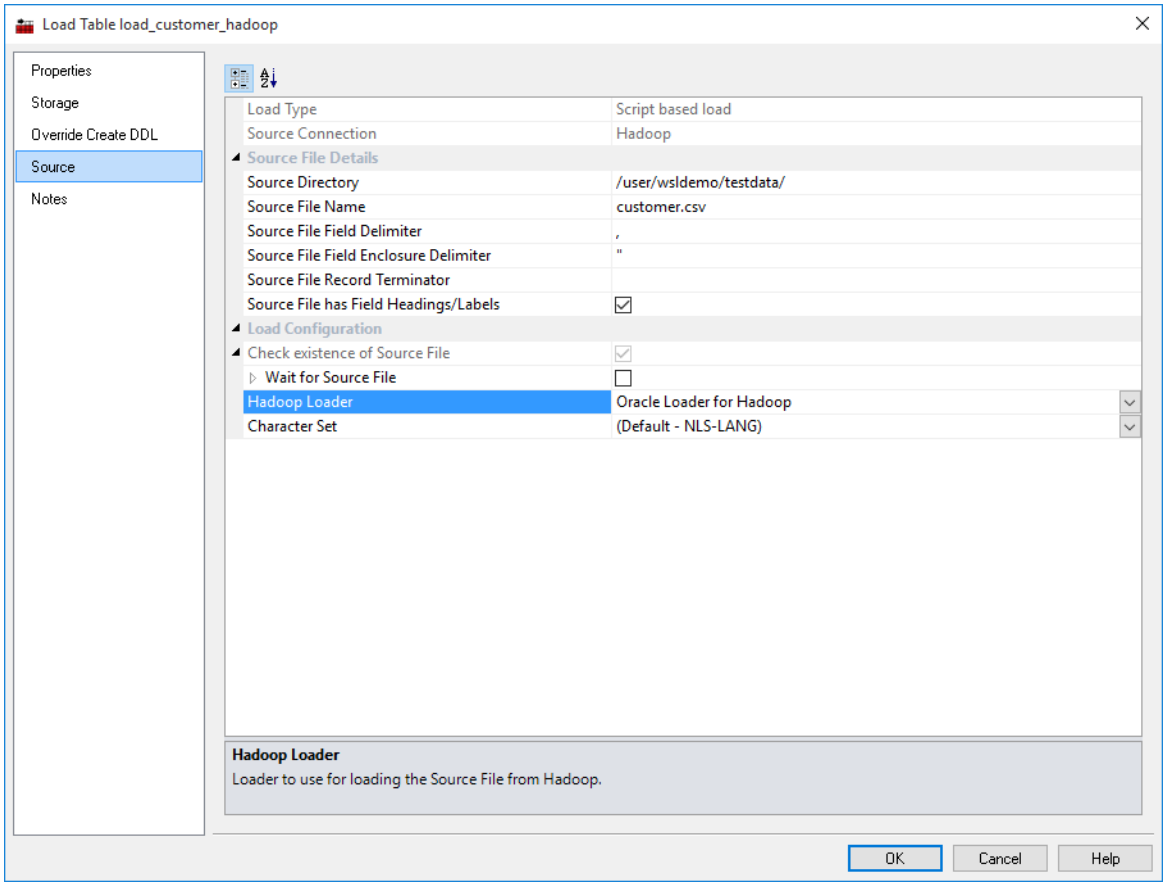
SQL Server **Script load Source screen from a Windows connection:**



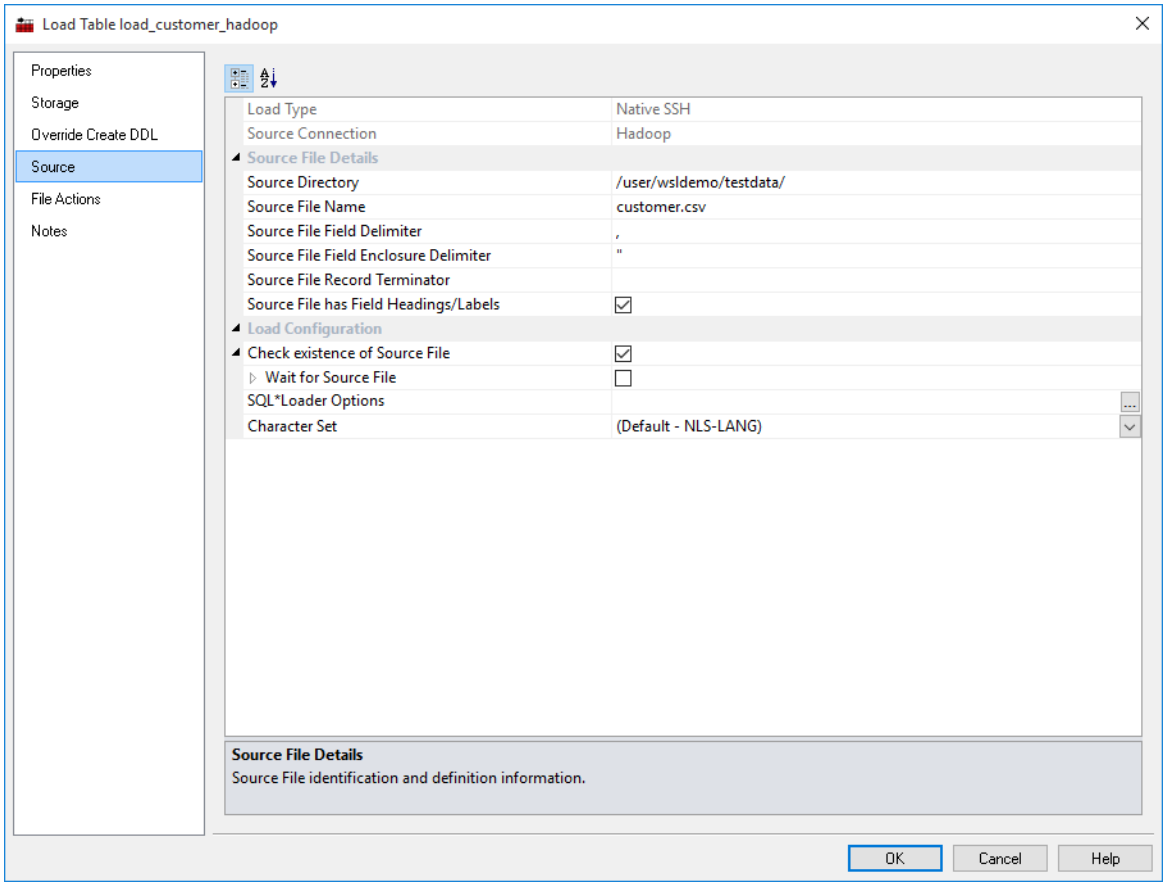
Oracle **File load Source screen:**



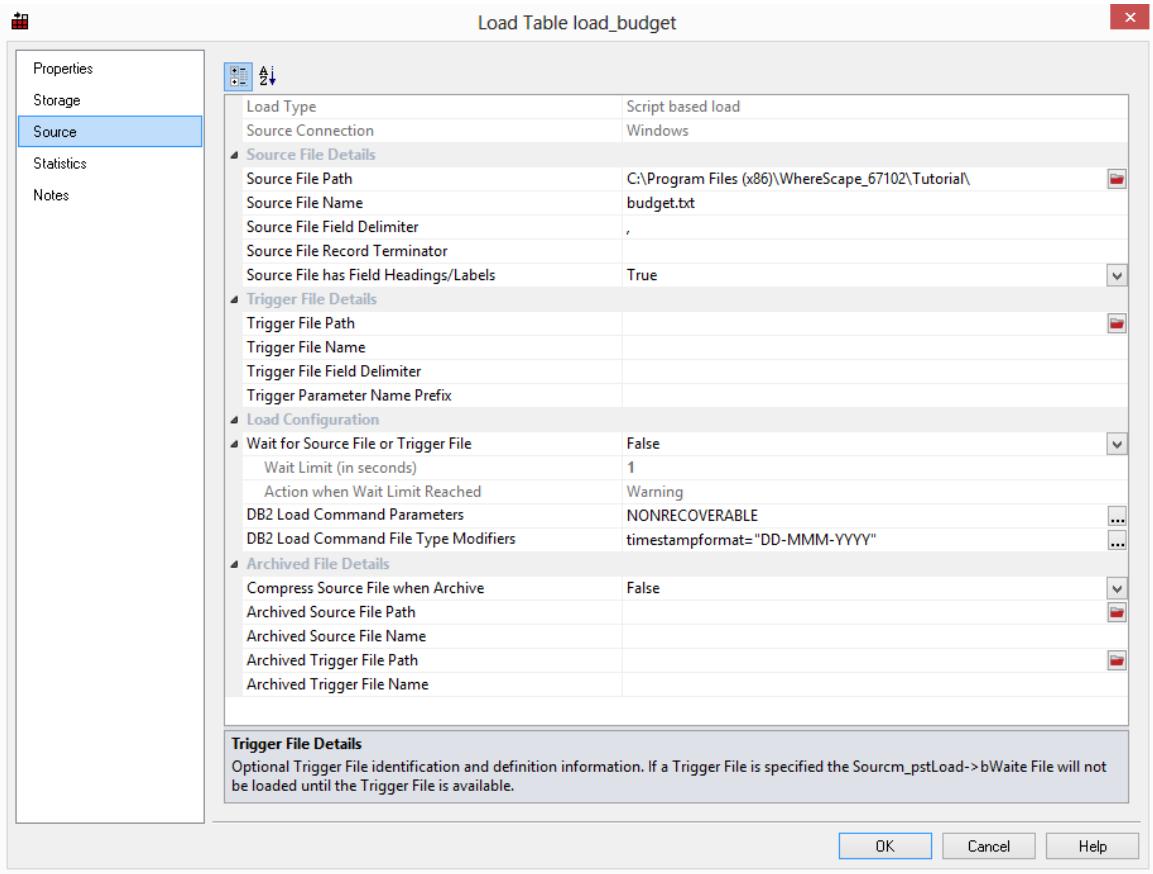
Oracle **Script Load Source screen from a Hadoop connection:**



Oracle **Native SSH Load Source screen from a Hadoop connection:**



DB2 **File load Source screen:**



SCRIPT BASED LOAD

A script based load table will have a Host Script defined. During the load process this host script is executed and the results returned.
During the 'drag and drop' creation of a load table from a UNIX or Windows file a script can be generated by selecting one of the 'Script based' load options. This script can then be edited to more fully meet any requirements.

Note: If the UNIX Scheduler is not being used and a UNIX script is generated and executed interactively the following two variables will be setup as part of the environment.

DSS_USER=username

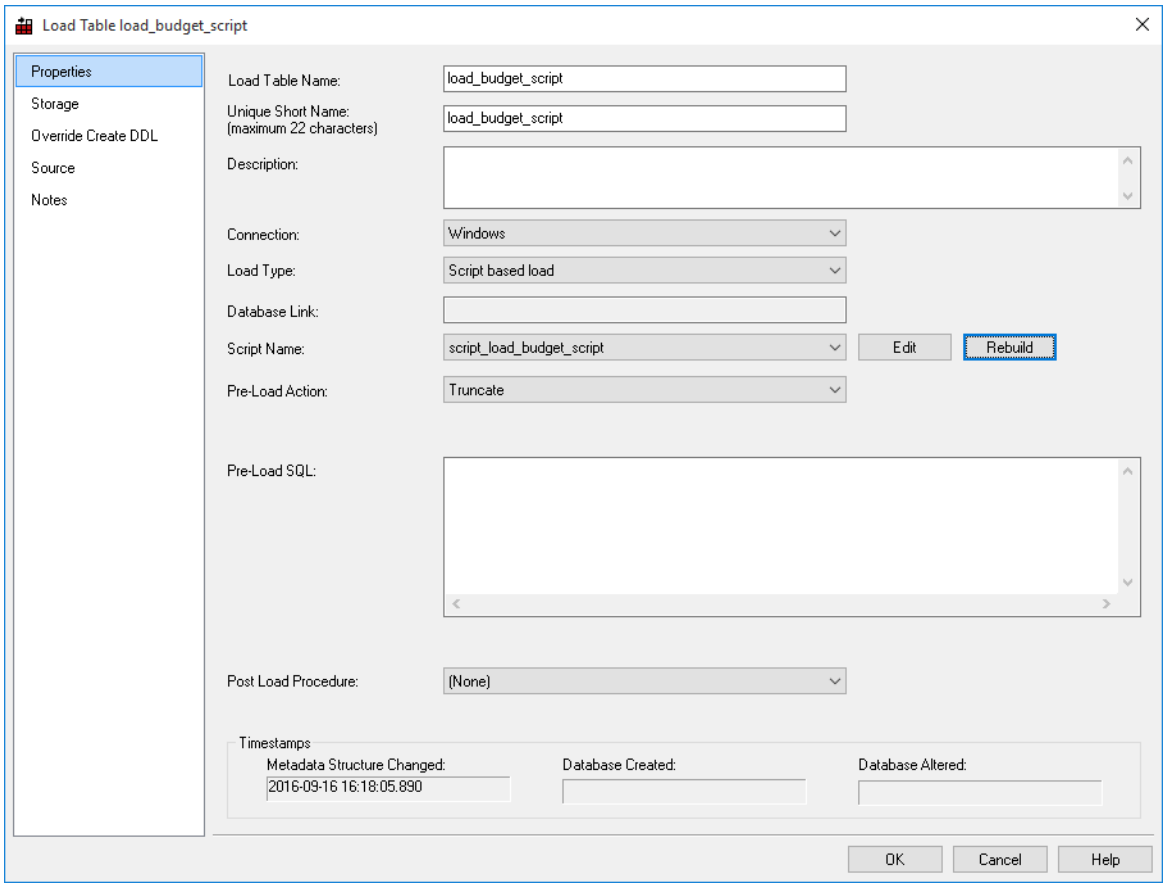
DSS_PWD=password

Where 'username' and 'password' are the entries for the data warehouse database.

There are a number of conventions that must be followed if these host scripts are to be used by the WhereScape scheduler.

1. The first line of data in **standard out** must contain the resultant status of the script. Valid values are '1' to indicate success, '-1' to indicate a Warning condition occurred but the result is considered a success, '-2' to indicate a handled Error occurred and subsequent dependent tasks should be held, -3 to indicate an unhandled Failure and that subsequent dependent tasks should be held.
2. The second line of data in **standard out** must contain a resultant message of no more than 256 characters.
3. Any subsequent lines in **standard out** are considered informational and are recorded in the audit trail. The normal practice is to place a minimum of information in the audit trail. All bulk information should be output to **standard error**
4. Any data output to **standard error** will be written to the error/detail log. Both the audit log and detail log can be viewed from the WhereScape RED tool under the scheduler window.

5 When doing **Script based loads**, it is easy to use the **rebuild** button to the right of the Script-name field to rebuild the scripts.



FILE WILD CARD LOAD

The loading of multiple files in one pass is best undertaken as a host script.

A generated host script contains the code to handle wild card loads and the subsequent loading and renaming of multiple files. The process for setting up wild card loads is as follows:

1. Drag one of the files into a **Load Table** target and select one of the **Script** based options.
2. If necessary edit the generated script and replace the load file name with the required wild card version. (For example, test_result_234.txt renamed to test_result_*.txt. This load file name is defined as a variable at the start of the script.
3. If renaming or moving to another directory is required, then you may need to uncomment the rename phase and enter the desired directory or rename syntax.
4. If only one file at a time is to be processed, but still based on a wild card file name, then you may need to uncomment the break statement at the bottom of the script. This will result in the loading of only the first file encountered that matches the wild card criteria.

In every event check the generated script as it will contain instructions on how to handle wild cards for the appropriate script type (Windows or UNIX).

XML FILE LOAD

XML file loads are only supported from a Windows connection. There are multiple formats for data exchange when using XML. See the following section for details on how an XML file is handled.

To load an XML file located in a Windows directory proceed as follows:

- Create a connection to the Windows system.
- Browse to the connection and locate the XML file.
- Make Load tables the middle pane drop target by double-clicking on the **Load Table** object group in the left pane.
- Drag the XML file into the middle pane.

The only rules concerning the xml file are that the data element tags are the column names and each row of data is a child of the root element. eg

```
<row>
      <dim_customer_key>7</dim_customer_key>
      <code>228</code>
      <name>JOHN AND JOES TOYS</name>
      <address>3700 PARNELL RISE</address>
      <city>BEAVERTON</city>
      <state>OR</state>
      <dss_source_system_key>1</dss_source_system_key>
      <dss_update_time>2003-10-03T10:02:15.310</dss_update_time>
</row>
```

Supported XML Formats

WhereScape RED supports two types of xml file construct. The normal xml standards have the data in the xml file and the table definitions in a separate xsd (xml schema definition) file which is only required when the table is being created or when the xml file is being validated for form. An alternate standard is used by Microsoft. This second standard is an in-line definition which produces one file which contains a Schema element in the data stream where the column names and their approximate data types are defined.

Separate XML and XSD files

The normal XML standards have the data in the xml file and the table definitions in a separate xsd (xml schema definition) file which is only required when the table is being created or when the xml file is being validated for form. The xsd file name is found within the xml file in an xsi (xml schema instance) statement which can include a namespace definition;e.g.

```
<root xmlns="http://www.wherescape.com/wsl-schema"
      xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance"
      xsi:schemaLocation="http://www.wherescape.com/load_table.xsd">
```

or no namespace;e.g.

```
<root xmlns="http://www.wherescape.com/wsl-schema"
      xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance"
      xsi:noNamespaceSchemaLocation="load_table.xsd">
```

The xsd file is an xml file and should be found in the same directory as the xml file that calls it. This xsd file will contain the column definitions for the load table which will be defined during the drag and drop.

The column definitions within the xsd file must be detailed enough to define a load table that the xml file can be loaded into. The data type mapping between the xsd file and the database have been implemented as below:

XSD	SQL Server	Oracle	IBM DB2
string with length	char()	char()	char()
string with maxlength	varchar()	varchar2()	varchar()
integer	integer	integer	integer
decimal with precision and scale	numeric(x,y)	number(x,y)	decimal(x,y)
dateTime (ISO8601)	datetime	date	timestamp
i2	integer	integer	integer
i4	integer	integer	integer
r4	float	number	float
r8	float	number	float
float	float	number	float

These are the ISO-ANSI SQL/XML standards and in the case of integers, datetime and floats the column can be defined with one line. i.e.

```
<xsd:element name="Policy_ID" type="xsd:integer"/>
<xsd:element name="Quote_Date" type="xsd:dateTime"/>
<xsd:element name="Quote_Price" type="xsd:r4"/>
```

In the case of strings and decimals the column requires a bit more detail to produce the correct data type. Strings can be fixed length with padded data by using the length attribute. The following will produce a char(1) column called Excess_Waiver:

```
<xsd:element name="Excess_Waiver">
      <xsd:restriction base="xsd:string">
      <xsd:length value="1"/>
      </xsd:restriction>
</xsd:element>
```

Strings can be of variable length by using the maxLength attribute. The following produces a column of SQL Server varchar(8) or Oracle varchar2(8) called Password:

```
<xsd:element name="Password">
      <xsd:restriction base="xsd:string">
      <xsd:maxLength value="8"/>
      </xsd:restriction>
</xsd:element>
```

Decimal numbers are defined with the precision and scale attributes. If the scale attribute is zero or missing then the column will be a whole number of size precision. The following produces a SQL Server column of numeric(6) or an Oracle column of number(6):

```
<xsd:element name="code" >
      <xsd:restriction base="xsd:decimal">
      <xsd:precision value="6"/>
      <xsd:scale value="0"/>
      </xsd:restriction>
</xsd:element>
```

The following produces a SQL Server column of numeric(8,2) or an Oracle column of number(8,2):

```
<xsd:element name="code" >
      <xsd:restriction base="xsd:decimal">
      <xsd:precision value="8"/>
      <xsd:scale value="2"/>
      </xsd:restriction>
</xsd:element>
```

An example file with most data types would be as follows:

```
<xsd:schema xmlns="http://www.wherescape.com/wsl-schema"
            xmlns:xsd="http://www.wherescape.com/XMLSchema">
<xsd:element name="Col_name1" type="xsd:integer"/>
<xsd:element name="Col_name4" type="xsd:dateTime"/>
<xsd:element name="Col_name5">
      <xsd:restriction base="xsd:string">
      <xsd:maxLength value="100"/>
      </xsd:restriction>
</xsd:element>
<xsd:element name="Col_name6">
      <xsd:restriction base="xsd:string">
      <xsd:length value="100"/>
      </xsd:restriction>
</xsd:element>
<xsd:element name="Col_name7" type="xsd:float"/>
<xsd:element name="Col_name8" >
      <xsd:restriction base="xsd:decimal">
      <xsd:precision value="6"/>
      <xsd:scale value="2"/>
      </xsd:restriction>
</xsd:element>
</xsd:schema>
```

The column order will be the same as the xsd file.

Any columns which are missing from the row will be NULL in the loaded row.

The dateTime format in the xml file is defined as ISO8601 which looks like this:

2003-10-03T10:02:15.310

- WhereScape RED will load this string into **Oracle** as:

```
TO_DATE('20031003100215','YYYYMMDDHH24MI')
```

- Or for **SQL Server**:

```
CONVERT(DATETIME,'2003-10-03T10:02:15.310',126)
```

- Or for **DB2**:

```
TIMESTAMP_FORMAT('20031003100215', 'YYYYMMDDHH24MISS')
```

- The xsd file is only required to create the load table, if the load table is only being loaded then this file is ignored.
- To check that the xml and xsd files are well formed you can open them with any web browser. If the files display with no errors, then they are valid xml files.

In line schema definition

The other supported xml construct allows the use of in line schema definitions as produced by the Microsoft FOR XML AUTO, ELEMENTS, XMLDATA query. This will produce one file which contains a Schema element in which the column names and their approximate data types are defined. Because the supplied data types are not concise enough to define the table columns correctly, this method will produce load tables of either data type text for SQL Server or varchar2(4000) for Oracle. The column names are taken from the <element type="col_name"/> strings within the Schema element. The data elements will be the same as above with the column names making up the start and end tags and the rows being the children of the root element. The file that is produced by the FOR XML query above needs to be changed slightly to comply with the xml standard. Remove everything before the Schema element and then give the file a starting root element and a closing root element.eg <root> and </root>

The xml files can optionally start with an xml pre process statement.eg

```
<?xml version="1.0"?>
```

They may also contain xml comments.eg

```
<!— comments    -->
```

EXTERNAL LOAD

For an externally loaded table the only property that is executed is the Post Load procedure. Any **After** transformations recorded against any of the columns in an Externally loaded table will also be processed.

APACHE SQOOP LOAD

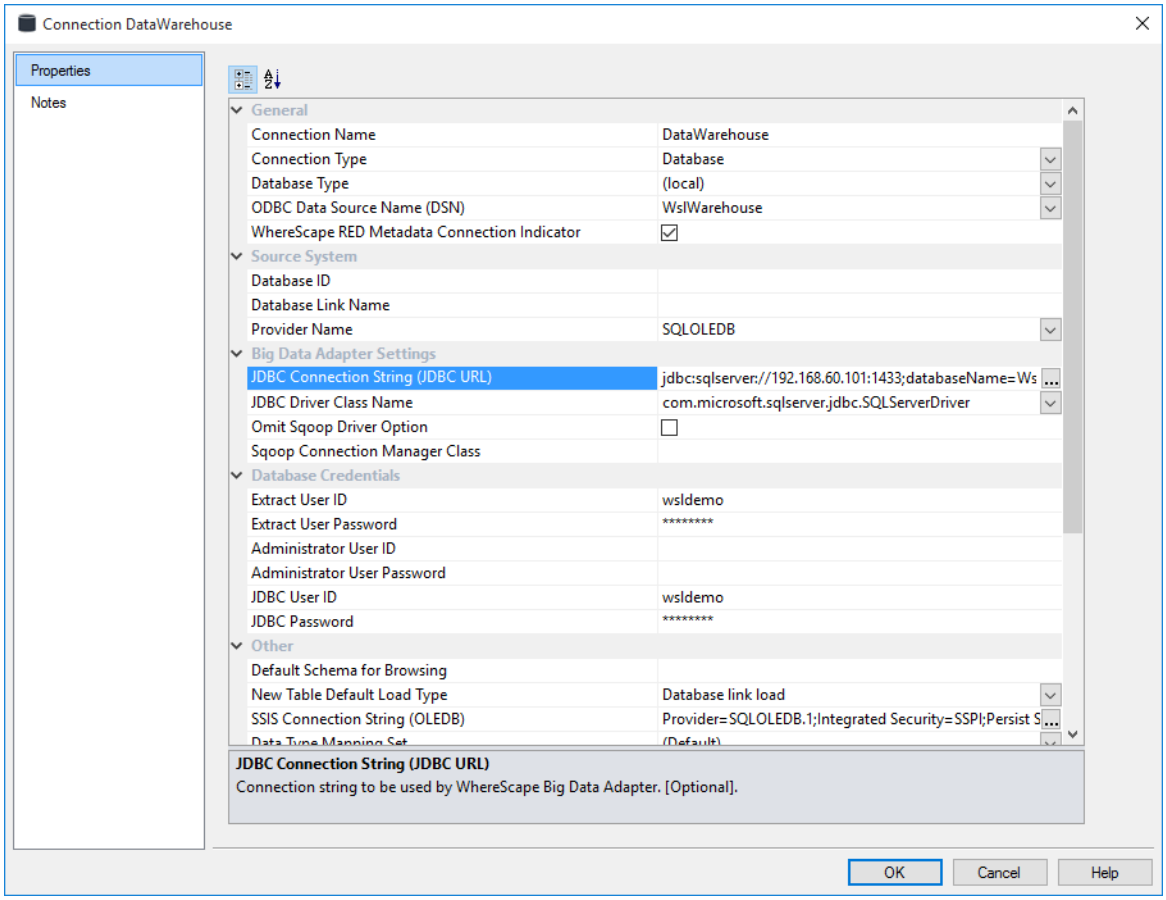
The **Apache Sqoop load** type enables loading data directly from Hive/HDFS to any (non-Hive) targets. This load method provides a non-database specific method of loading data into any database target, which is especially helpful when loading data directly from HDFS or Hive to SQL Server and Oracle target databases.

Apache Sqoop load from a Hive connection to a Hive target are not supported.

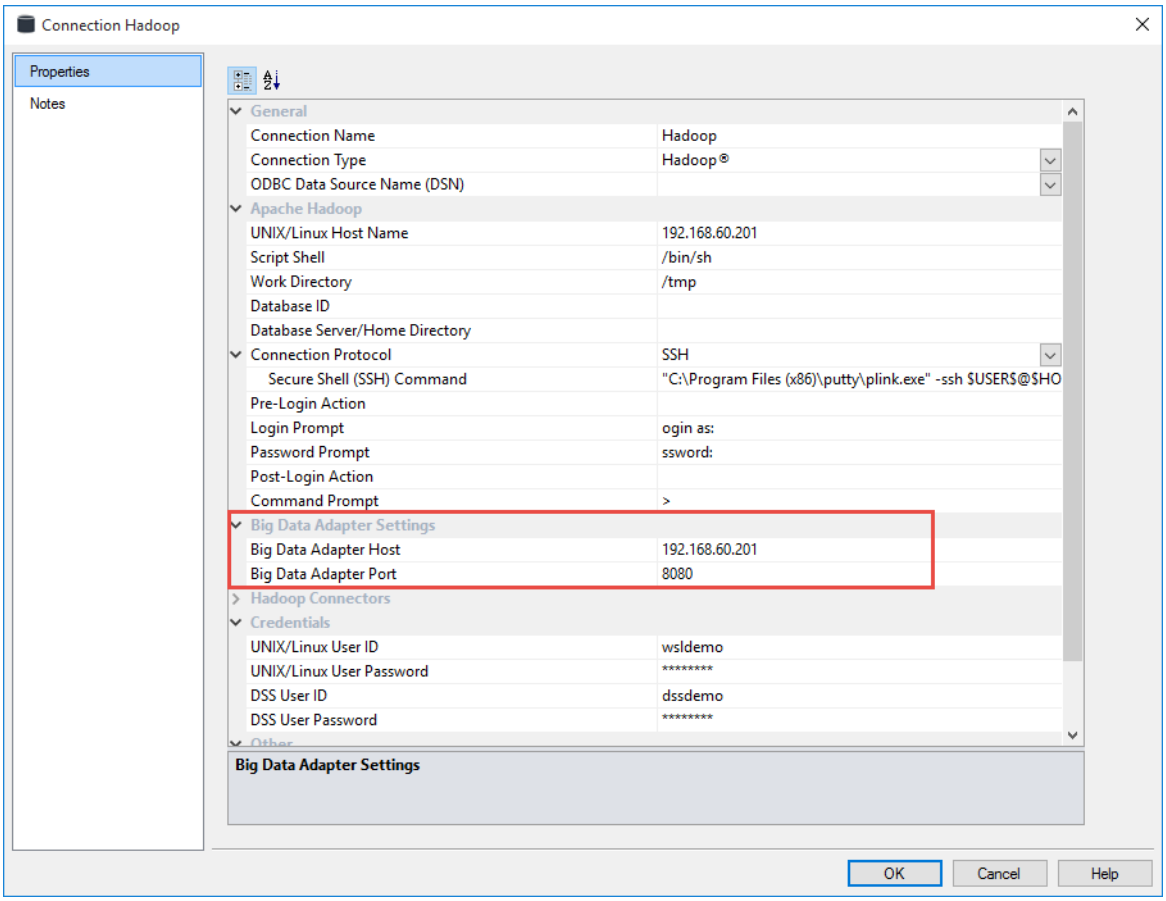
To load tables directly from HDFS/Hive into SQL Server and Oracle target databases:

1 Ensure the relevant **Data Warehouse connection** has the following fields set:

- JDBC Connection string (JDBC URL)
- JDBC Driver Class Name
- Omit Sqoop Driver Option - tick this check-box for loads into Oracle target databases
- JDBC User ID
- JDBC Password



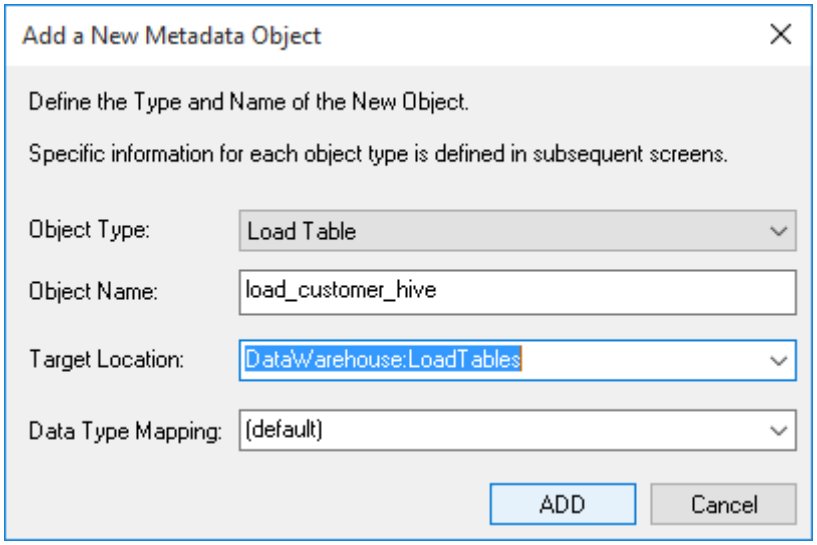
2 When loading from **Hadoop connections**, please ensure the Hadoop connection has its **BDA server host** and **port** fields set in addition to Hive connections.



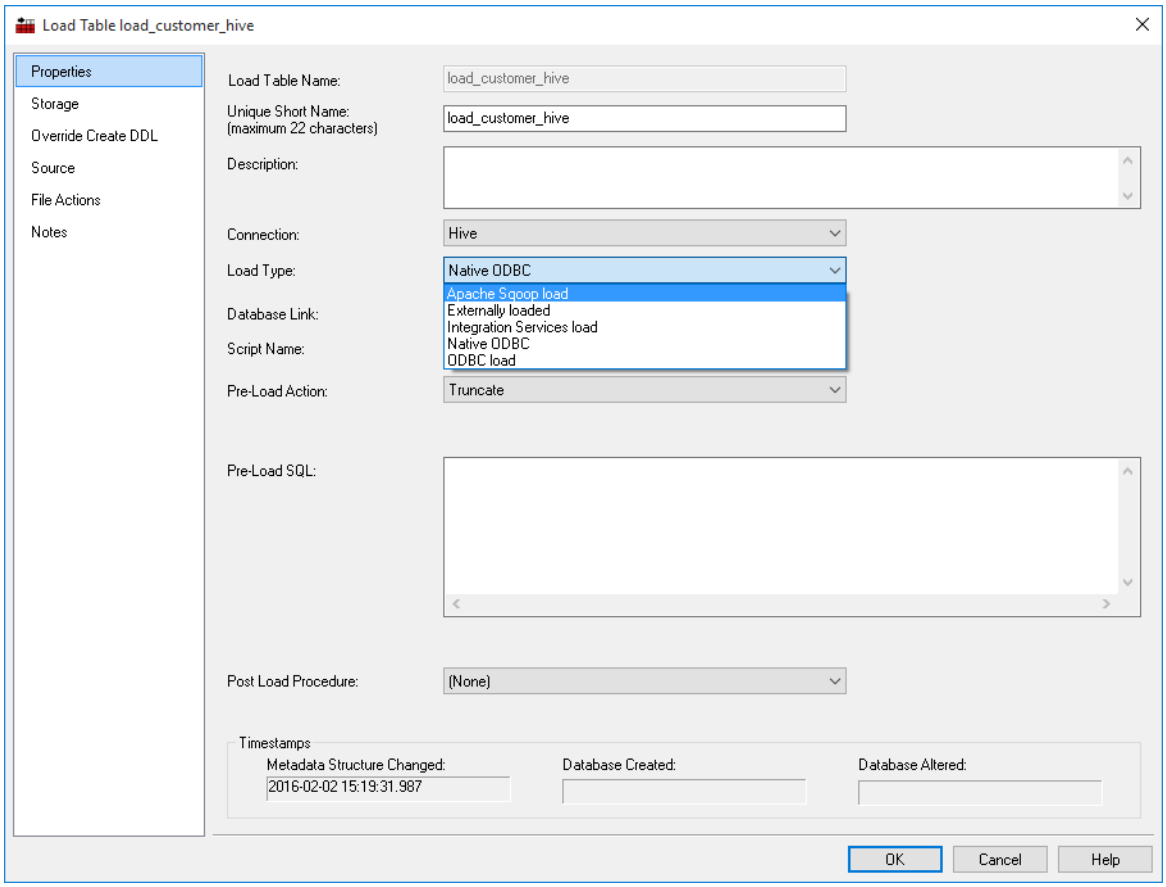
3 Browse the desired **Hadoop/Hive connection**.

4 **Drag and Drop** the table from the Hadoop/Hive connection on the right hand-side into the middle pane.

- Change the table name if necessary and select the relevant target location to place the table from the drop-down list.

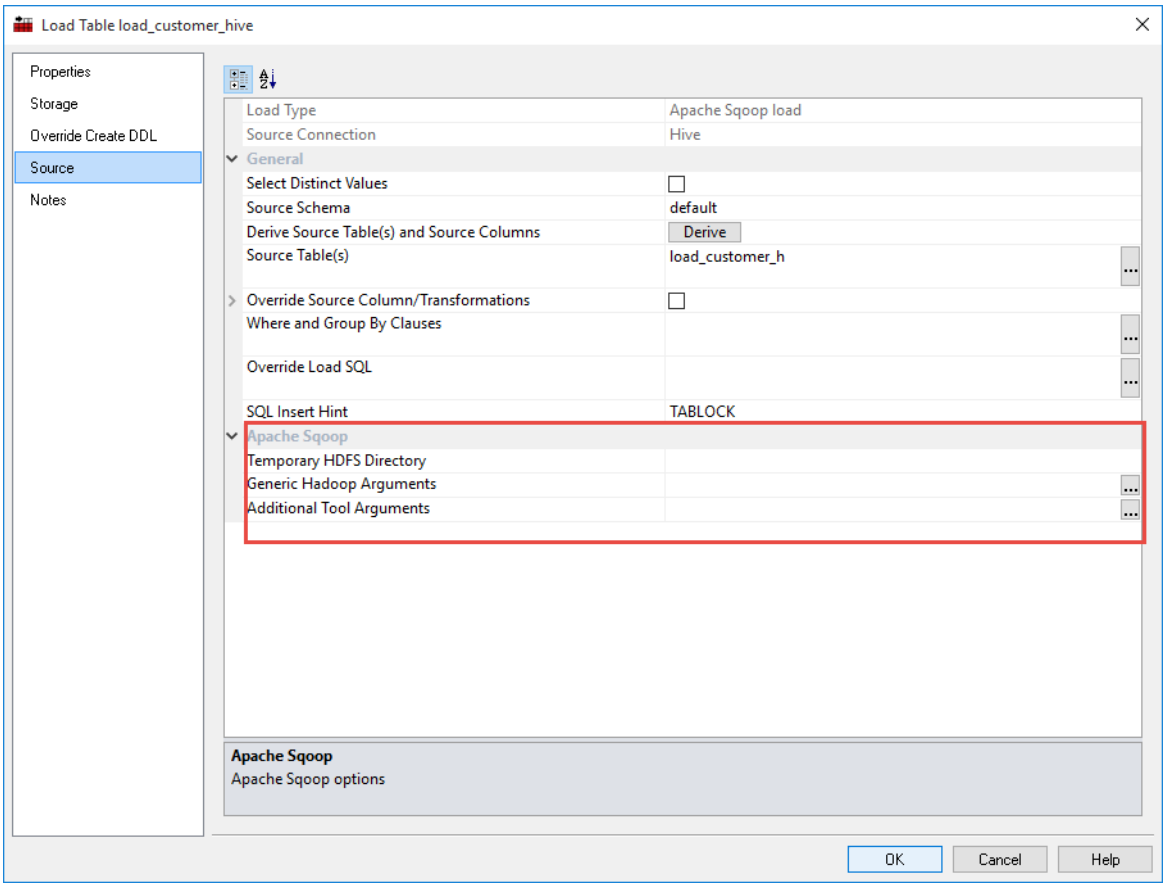


5 Select **Apache Hadoop Load** from the Load Type drop-down list.



6 Click the **Source** tab to add any Apache Sqoop specific options:

- Temporary HDFS Directory - Loading from a Hive source to the data warehouse is implemented in two steps. First, the data is extracted from the Hive table into a temporary HDFS directory. Then, this temporary directory is loaded using "sqoop export". The location of the temporary directory can be configured in RED on the source tab of the load table. When this field is left blank, the default is "/tmp".
- Generic Hadoop Arguments - This field allows adding additional arguments just after the Sqoop command keyword, in this case it is import, in the Sqoop command line.
- Additional Tool Arguments - This field allows adding additional arguments after the generated Sqoop command line.



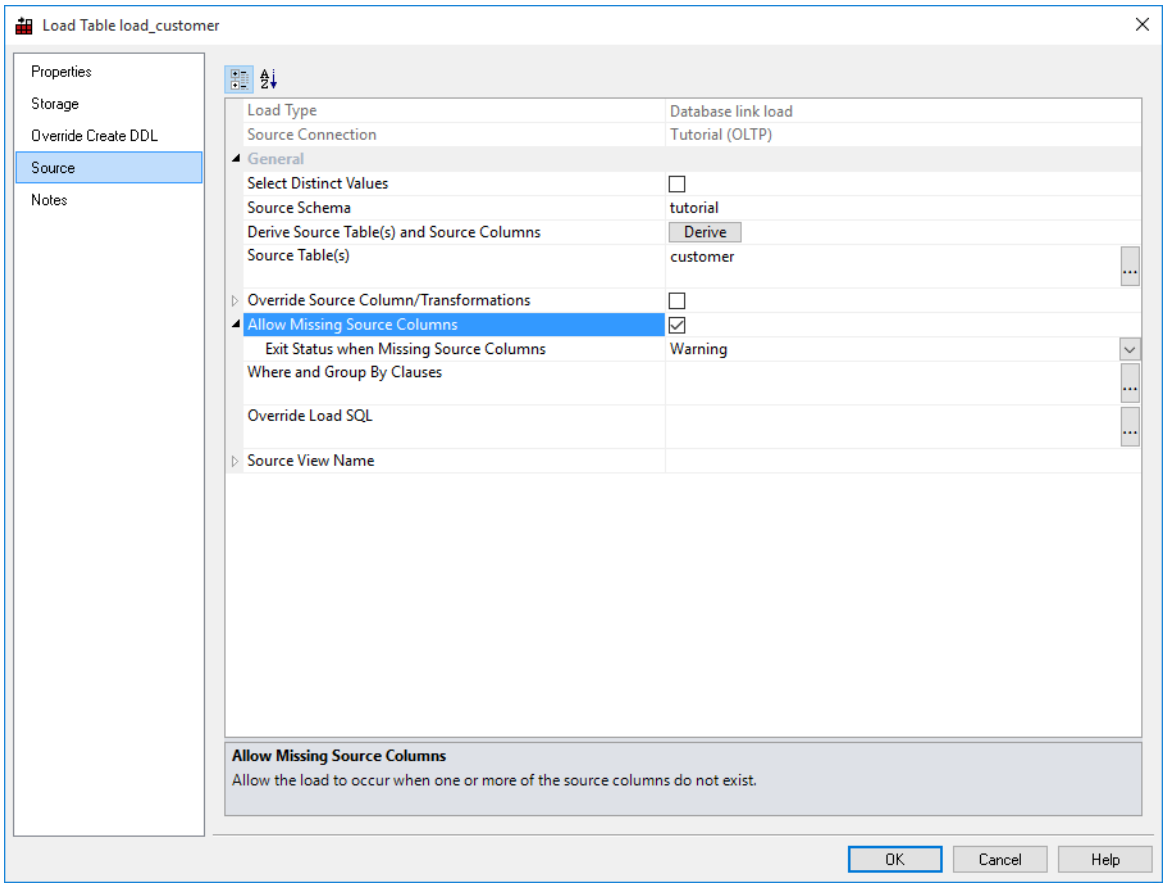
7 Click **Create and Load** to create and load the table.

HANDLING MISSING SOURCE COLUMNS

Note: This option is only available when running an **Oracle** data warehouse.

By default, a load will fail if a source column that is to be loaded does not exist. This default action can be modified by using the **Allow Missing Source Columns** feature of a load table. When this feature is enabled, the load process will check all source columns, and if any are found to be missing will replace those missing columns with a Null value (which can be changed, see below).

On the **Source Mapping** tab of a load tables Properties there are two checkboxes and a drop-down list that we will cover in this section. See the following example:



Override Source Column/Transformation

When this is enabled the **Source Columns** edit window is enabled. With this enabled, a load table uses the contents of the **Source columns** to specify which columns are being extracted during a table load.

When this is unselected, the load process builds up the statement to select from the source table(s) by looking at the two fields source table and source column and any transformations associated with each column in the load table. If a transformation is present, then the transformation overrides the contents of source table/source column. See the following section for more information on transformations.

Note: This box must be unselected to enable **Allow Missing Source Columns** support.

Allow Missing Source Columns

When this checkbox is checked, the load process will examine every column in the load table. Based on the source table/source column fields associated with each column it will check to see if the column exists in the source database. If the column exists normal processing will occur. If the column does not exist then a Null will be substituted for the column, and the load will proceed.

If one or more columns are found to be missing the load process reports this situation. The status level of this reporting can be set via the **Exit status** drop-down. See the following topic. In all cases the load will be deemed to have been successful if no other errors occur.

Often Null values are not desirable in a data warehouse. This Null value can be replaced by some other value by means of a **During** or **After** transformation. For example, a **During** transformation, as shown below, set on a missing column called 'State' will replace the Null with the value 'N/A'.

NVL(state,'N/A')

Exit Status When Missing Columns

If columns are found to be missing as a result of a **Non Mandatory** check then a message is recorded in the Audit trail and against the task if running in the Scheduler.
The drop-down list of the same name allows the choice of action in the event of a missing column or columns. The choices are:

Choice	Impact
Success	The message informing that columns are missing uses an information status.
Warning	The message issued is a warning. The table load will be identified as successful but with warnings.
Error	The message issued is an error. The table load will still complete successfully albeit with an error message.

This drop-down list is only available if the **Non Mandatory Source Columns** check-box is ticked.

Limitations of Missing Column check

The check for missing columns will only examine and check those columns that are recorded in the source table and source column fields of a load table's column Properties. Therefore, if a source column is used in a transformation or join, but is not recorded in the two fields mentioned above it will not be checked.

If a column is used in a transformation, it should have the same usage (name and case) as is found in the source column and source table fields.

This check has no upper limit on the number of missing columns. All columns can be missing and the load will still succeed.

LOAD TABLE TRANSFORMATIONS

Each load table column can have a transformation associated with it.
See ***Transformations*** (on page 650) and ***Load Table Column Transformations*** (on page 654) for more details.

POST-LOAD PROCEDURES

If a procedure name is entered in the post-load procedure field of a load table's Properties, then this procedure will be executed after the load has completed and after any **after transformations** have occurred.

See ***Load Table Column Transformations*** (on page 654) for more details.

CHANGING LOAD CONNECTION AND SCHEMA

The connection associated with a load table can be changed through the Properties of that table.

Connections can also be changed in bulk by using the following process:

1. Double-click on the **Load Table** object group in the left pane. This will display all load tables in the middle pane.
2. Select those load tables that you wish to change using standard Windows selection.
3. Right-click to bring up a menu and select **Change Connect/Schema**.
4. Select the new connection to change all the selected load tables.

Note: You cannot change the connection **type** but it is possible to change from Database to ODBC connections when the following considerations are taken into account.

Switching Connection from ODBC to Database

This switch should be successful in most cases, although it may not provide the most performant load possible. By default, ODBC connections use the source table/column transformation loading method as dates and potentially other data types need to be converted.

When switching to a database link load any transformations will still occur although they may no longer be necessary.

Switching Connection from Database to ODBC

Although it is possible to switch from a Database connection to an ODBC connection, the resultant extract may not function correctly. The main issue is around the conversion of date fields under Oracle. SQL Server tends to handle the date conversion better in most cases. When an extract occurs over an ODBC connection the date is converted in the extract from the source system into a standard ASCII format. This ASCII formatted date is then converted back when it is loaded into the load table. To resolve this problem, place a transformation on any date fields. Examples of typical Oracle transformations are:

to_date('''||to_char(ORDER_DATE,'YYYYMMDD')||''','YYYYMMDD')

to_char(ORDER_DATE,'YYYY-MM-DD HH24:MI:SS')

where ORDER_DATE is the name of the column being extracted. In the first example the conversion string is enclosed within quote marks (') so that it is passed as a literal to the select statement that will be built. Quote marks within the conversion string are protected from premature evaluation by the inclusion of a second quote mark alongside the original.

NOTE: If a load table is created via drag and drop from an ODBC based connection WhereScape RED will build all the required date transformations.

There are several supplied APIs for changing schema and connections programmatically. See ***Callable Routines*** (on page 955) for more details.

CHAPTER 9
DIMENSIONS

IN THIS CHAPTER

DIMENSIONS OVERVIEW

A dimension table is normally defined, for our purposes, as a table that allows us to constrain queries on the fact table.
A dimension is built from the Data Warehouse connection. Unless you are doing a retro-fit of an existing system, dimensions are typically built from one or more load tables.

The normal steps for creating a dimension are defined below and are covered in this chapter. The steps are:

- Identify the source transactional data that will constitute the dimension. If the data is sourced from multiple tables ascertain if a join between the source tables is possible, or if a series of lookups would be a better option.
- Using the 'drag and drop' functionality drag the load table that is the primary source of information for the dimension into a dimension target. See ***Building a Dimension*** (on page 324).
- If only one table is being sourced and most of the columns are to be used (or if prototyping) you can select the auto create option to build and load the dimension and skip the next 4 steps. See ***Building a Dimension*** (on page 324).
- Add columns from other load tables if required. See ***Building a Dimension*** (on page 324).
- Create the dimension table in the database. See ***Building a Dimension*** (on page 324).
- Build the update procedure. See ***Generating the Dimension Update Procedure*** (on page 333).
- Run the update procedure and analyze the results. See ***Dimension Initial Build*** (on page 352).
- Modify the update procedure as required. See ***Dimension Initial Build*** (on page 352).

Dimension Keys

Dimensions have two types of keys that we will refer to frequently. These are the **Business Key** and the **Artificial Key**.
A definition of these two key types follows:

Business Key

The business key is the column or columns that uniquely identify a record within the dimension. Where the dimension maps back to a single or a main table in the source system, it is usually possible to ascertain the business key by looking at the unique keys for that source table. Some people refer to the business key as the **natural key**. Examples of business keys are:

- The product SKU in a product dimension
- The customer code in a customer dimension
- The calendar date in a date dimension
- The 24-hour time in a time dimension (i.e. HHMM) (e.g.1710)
- The airport short code in an airport dimension.

It is assumed that business keys will never be NULL. If a null value is possible in a business key, then the generated code will need to be modified to handle the null value by assigning some default value.

For example, the 'Where' clause in a dimension update may become:

SQL Server: Where isnull(business_key,'N/A') = isnull(v_LoadRec.business_key,'N/A')

Oracle: Where nvl(business_key,'N/A') = nvl(v_LoadRec.business_key,'N/A')

DB2: Where coalesce(business_key,'N/A') = coalesce(v_LoadRec.business_key,'N/A')

NOTE: Business keys are assumed to never be Null. If they could be null it is best to transform them to some value prior to dimension or stage table update. If this is not done an unmodified update will probably fail with a duplicate key error on the business key index.

Artificial Key

The artificial key is the unique identifier that is used to join a dimension record to a fact table. When joining dimensions to fact tables it would be possible to perform the join using the business key. For fact tables with a large number of records this however would result in slow query times and very large indexes. As query time is one of our key drivers in data warehouse implementations the best answer is always to use some form of artificial key. A price is paid in the additional processing required to build the fact table rows, but this is offset by the reduced query times and index sizes. We can also make use of database specific features such as bitmap indexes in Oracle.
The artificial key is an integer and is built sequentially from 1 upwards.

See the section on artificial keys for a more detailed explanation. An artificial key is sometimes referred to as a **surrogate key**.

BUILDING A DIMENSION

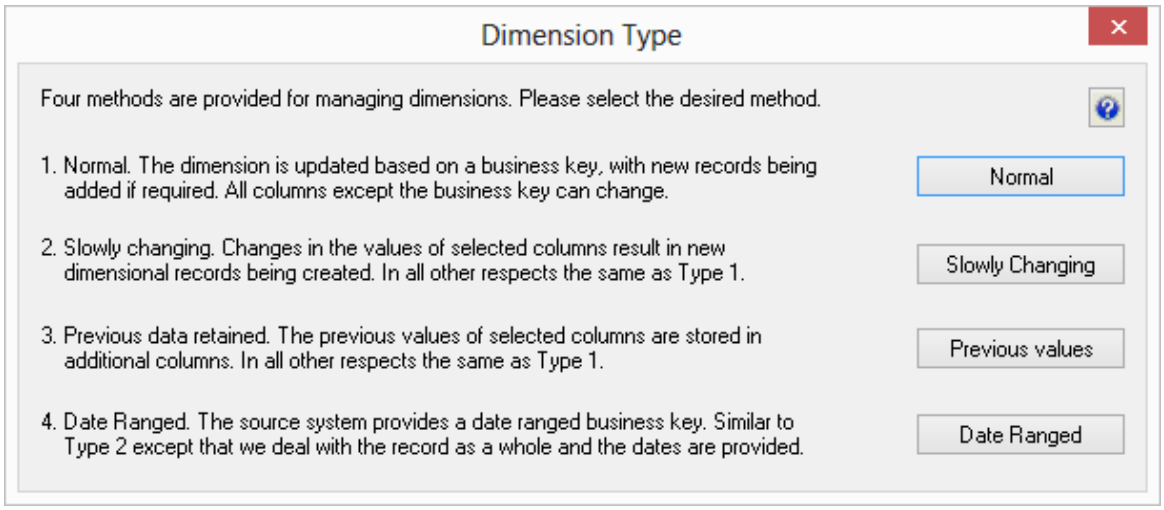
Dimensions are often sourced from one table in the base application. In many cases there are also codes that require description lookups to complete the de-normalization of the dimensional data. The process for building a dimension is the same for most other tables and begins with the drag and drop of the load table that contains the bulk of the dimensional information.

Drag and Drop

- Create a dimension target by double-clicking on the **Dimension group** in the left pane.
- The middle pane will display a list of all existing dimensions. When this list is displayed in the middle pane, the pane is identified as a target for new dimension tables.
- Browse to the Data Warehouse via the Browse/Source Data menu option.
- Drag the load table, that contains the bulk of the dimensional columns, into the middle pane.
- Drop the table anywhere in the pane.
- The new object dialog box will appear and will identify the new object as a Dimension and will provide a default name based on the load table name. Either accept this name or enter the name of the dimension and click **OK** to proceed.

Dimension Type

A dialog will appear as shown below. There are four choices for the default generation of the dimension table and its update procedure.



The first choice being a **normal** dimension where a dimensional record is updated and changed whenever any of the non-business key information changes.
The second choice is a **slowly changing** dimension where new dimension records are created when certain identified columns in the dimension change.
The third choice is a **Previous values** dimension, which allows the storing of the last value of selected fields in secondary columns.
The fourth choice is a **Date Ranged** dimension, which supports source systems that provide start and end dates.

With any dimension, we identify a business key that uniquely identifies the dimension records. For example, in the case of the product dimension from the tutorial the product **code** is deemed to be the business key. The code uniquely identifies each product within the dimension. The product may also have a name or description and various other attributes that distinguish it. (e.g. Size, shape, color etc.). A common question when handling dimensions is what to do when the name or description changes. Do we want to track our fact table records based only on the product code or do we also want to track records based on different descriptions.

An example:

code	**description**	**product_group**	**sub_group**
1235	15oz can of brussel sprouts	canned goods	sprouts

This product has been sold for many years and we consequently have a very good history of sales and the performance of the product in the market. The company does a '20% extra for free' promotion for 3 months during which time it increases the size of the can to 18oz. The description is also changed to be '15 + 3oz can of brussel sprouts'. At the end of the promotion the product is reverted to its original size and the description changed back to its original name.

The question is, do we want to track the sales of the product when it had a different description (slowly changing), or should the description of the product simply change to reflect its current name (normal). For this scenario, a previous value dimension would not provide much advantage, so it is not discussed.

The decision is not a simple one and the advantages and disadvantages of each of the two choices is discussed below.

SLOWLY CHANGING

- Allows the most comprehensive analysis capabilities when just using the product dimension.
- Complicates the analysis. Does not allow a continuous analysis of the product called '15oz can of brussel sprouts' when the description is used. This analysis is however still available through the code which has not changed.
- Adds considerable additional processing requirements to the building of the fact tables that utilize this dimension.
- May track data quality improvements rather than real business change.

NORMAL

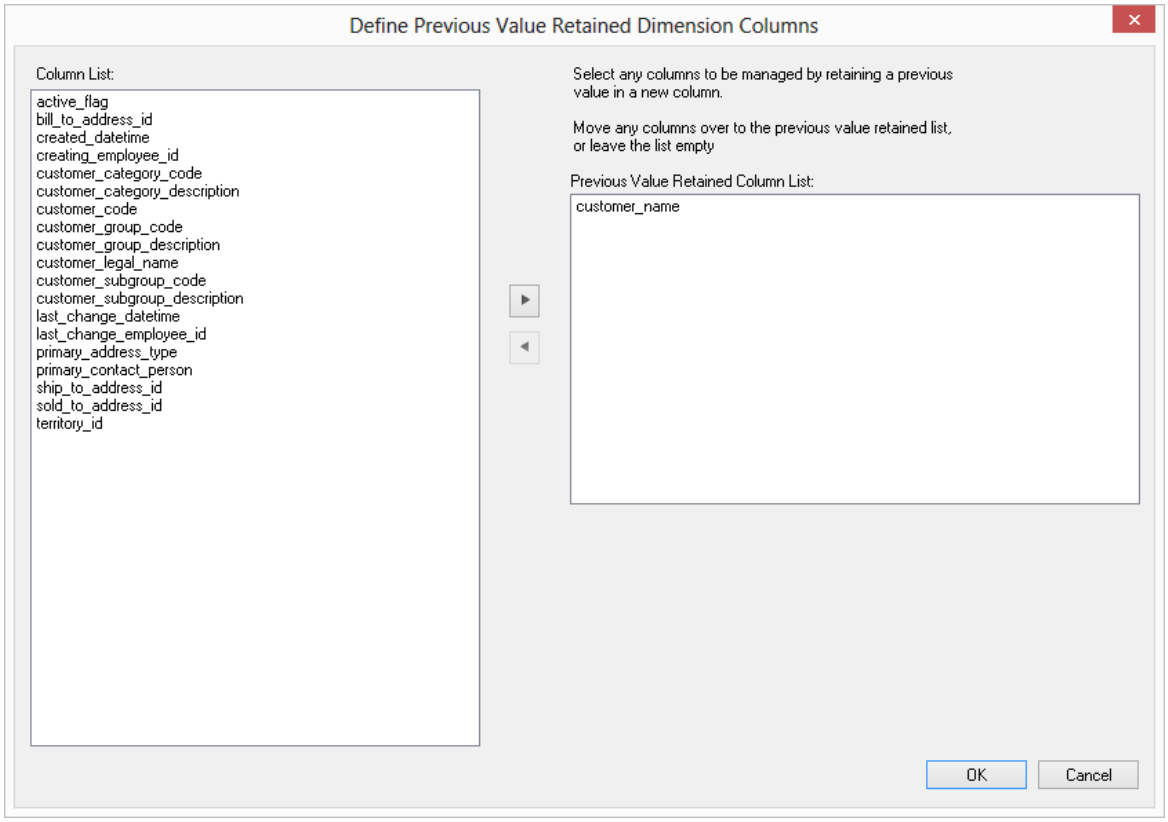
- Does not allow specific analysis of the product during its size change. Note, however that this analysis will probably be available through the combination of a 'promotion' dimension.
- Provides a continuous analysis history for the product called '15oz can of brussel sprouts'. An analysis via description and code will produce the same results.
- Simplifies analysis from an end user's perspective.

As mentioned above the choice is never a simple one. Even amongst experienced data warehouse practitioners there will be a variety of opinions. The decision must be based on the business requirements. In many cases keeping the analysis simple is the best choice, at least in the early stages of a data warehouse development. Slowly changing dimensions do have a place, but there is nearly always an alternate method that provides equal or better results. In the example above, a promotion dimension coupled with the product dimension could provide the same analysis results whilst keeping product only analysis simple and easy to understand.

TIP: Do not over complicate the design of an analysis area. Keep it simple and avoid the unnecessary use of slowly changing dimensions.

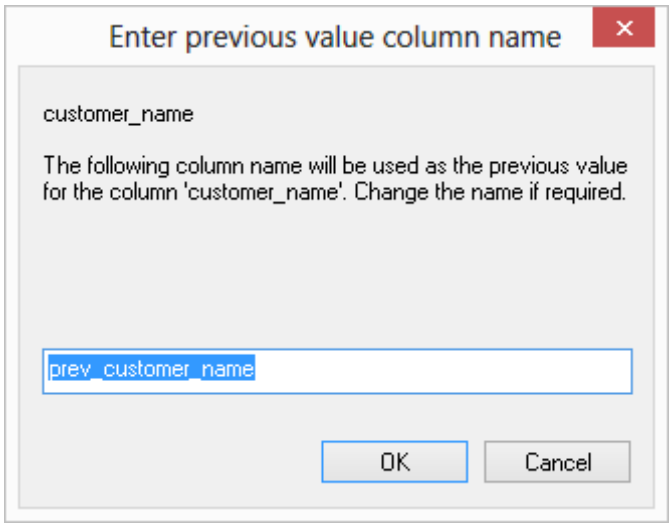
PREVIOUS VALUES DIMENSION TYPE

If you selected a 'Previous values' dimension type, then the following questions will be asked. If one of the other two types where chosen, then proceed directly to **Dimension Properties**.
The following dialog will appear:



It requests the definition of each column that will be managed by storing the previous value in a separate additional column. Note that the business key cannot be used in this manner, as a change to the business key will result in a new record.
Select the columns to be managed and click **OK**.

The following dialog will then appear for each column that has been selected.
Either accept the name of the additional column or enter a new name.



Dimension Properties

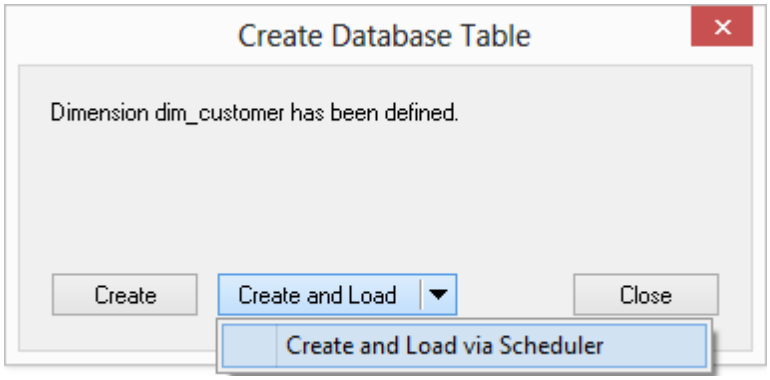
Once the dimension type is chosen the Properties screen will appear. Change the storage options if desired.
If prototyping, and the dimension is simple (i.e. one source table) then it is possible to create, load and update the dimension in a couple of steps.
If you wish to do this select the **(Build Procedure...)** option from the **Update Procedure** drop-down, and answer **Create and Load** to the next question.

Create and Load

The following dialog appears after the Properties screen and asks if you want to create the dimension table in the database and execute the update procedure.

If you are happy with the columns that will be used and do not wish to add additional columns you can click the **Create and Load** button.
In this case, it would be normal practice to select the **(Build Procedure...)** option from the 'Update Procedure' drop-down box in the previous Properties page.
If no procedure has been selected then select the **Create** button as no update will happen in any event.
If **Create** or **Create and Load** is selected and a new procedure creation was chosen proceed directly to the ***Generating the Dimension Update Procedure*** (on page 333) section.
If you have additional columns to add or columns to delete, click **Close** and proceed as follows below.

NOTE: It is possible to create and load the table via the Scheduler; by selecting this option from the drop-down list on the **Create and Load** button:



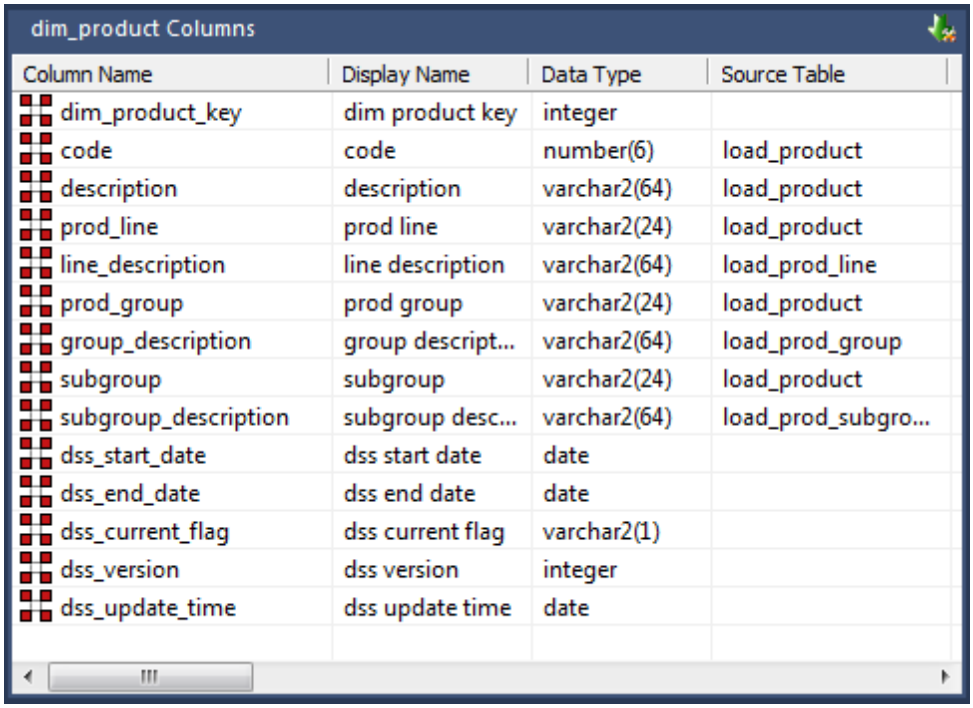
Deleting and Changing columns

The columns defined for the dimension will be displayed in the middle pane. It is possible to delete any unwanted columns by highlighting a column name or a group of names and choosing the **Delete** key.
The name of a column can also be changed by selecting the column and using the right-click menu to edit its properties.
Any new name must conform to the database naming standards. Good practice is to use alphanumerics and the underscore character.
See the section on column Properties for a more detailed description on what the various fields mean.

TIP: When prototyping, and in the initial stages of an analysis area build it is best not to remove columns, nor to change their names to any great extent. This type of activity is best left until after end users have used the data and provided feedback.

Adding additional columns

With the columns of the dimension table displayed in the middle pane, this pane is considered a drop target for additional columns.
It is a simple matter therefore to select columns from other tables and to drag these columns into the middle pane. The following column list is from the product table as supplied in the tutorial data set.



The source table shows where each column was dragged from. Although not the case in the tutorial, it is often common to have columns of the same name coming from different tables. In the example above the description column is acquired from the load_product, load_prod_group and load_prod_subgroup tables. In order that the dimension table be created we need to assign these columns unique names, so for this example the last two columns in question have been renamed to group_description and subgroup_description.
There are a number of columns that do not have a source table. These columns have been added by WhereScape RED, and are added depending on earlier choices.

A description of these columns follows:

Column name	**description**
dim_product_key	The unique identifier (artificial key) for the dimension. This key is used in the joins to the fact table. It is generated via a sequence (Oracle) or identity (SQL Server) associated with the table, except for the date dimension where it has the form YYYYMMDD
ss_start_date	Used for slowly changing dimensions. This column provides a date time stamp when the dimension record came into existence. It is used to ascertain which dimension record should be used when multiple are available.
dss_end_date	Used for slowly changing dimensions. This column provides a date time stamp when the dimension record ceased to be the current record. It is used to ascertain which dimension record should be used when multiple are available.
dss_current_flag	Used for slowly changing dimensions. This flag identifies the current record where multiple versions exist.

Column name	description
dss_source_system_key	Added to support dimensions that cannot be fully conformed, and the inclusion of subsequent source systems. See the ancillary settings section for more details.
dss_version	Used for slowly changing dimensions. This column contains the version number of a dimension record. Numbered from 1 upwards with the highest number being the latest or current version. It forms part of the unique constraint for the business key of a slowly changing dimension.
dss_update_time	Indicates when the record was last updated in the data warehouse.
dss_create_time	Indicates when the record was first created in the data warehouse

Manually adding previous value columns

If a **Previous Value** type of dimension is chosen, or in fact if the dimension is converted to this type, it is possible to manually add any required columns that were not defined as part of the create. The steps are:

1. Add a new column by dragging in the column that is to have a previous value stored.
2. Change the name to a unique name. Typically, by adding the prefix 'prev_' to the column name.
3. Change the source table, to be that of the dimension we are building.
4. Set the Key Type to 4.
5. Having performed these actions WhereScape RED will detect the column and build the appropriate code during the procedure generation phase.

Creating the table

Once the dimension has been defined in the metadata we need to physically create the table in the database. To do this, right-click on the dimension name and choose **Create (ReCreate)** from the pop-up menu.
A results dialog box will appear to show the results of the creation.
The contents of this dialog are a message to the effect that the dimension table was created. A copy of the actual database create statement, the sequence create statement (Oracle only), and if defined the results of any index create statements will be listed. For the initial create no indexes will be defined.
If the table was not created then ascertain and fix the problem. A common problem is a 'Duplicate column' where a column has the same name in two of the source tables.
The best way to find such a column is to double-click on the list heading **Col name**; which will sort the column names into alphabetical order.
Another double-click on the heading will sort the columns back into their create order.

For **Oracle** data warehouses the sequence created as part of the table create is normally called *table_name*_seq unless the default has been overridden. If this sequence already exists a warning is issued. This sequence is used in the assigning of a unique artificial key to the dimension. To change a sequence name while creating a dimension table, go to the Storage tab and change the **Artificial Key Sequence Name** field to the required sequence name. See more details on ***Table Storage Screen - Oracle*** (on page 208).

SQL Server makes use of an identity attribute on the artificial key column.

The next section covers the ***Generating the Dimension Update Procedure*** (on page 333).

GENERATING THE DIMENSION UPDATE PROCEDURE

Once a dimension has been defined in the metadata and created in the database an update procedure can be generated to handle the joining of any tables and the update of the dimension records.

THE ZERO KEY ROW

WhereScape RED will by default always insert a record into the dimension with an artificial key value of zero. This record is used to link any fact records that do not have valid dimension joins. The values of the various columns in this record are acquired from the contents of the field **Zero Key Value** which is set in the Properties screen of each dimension column.

Generating a Procedure

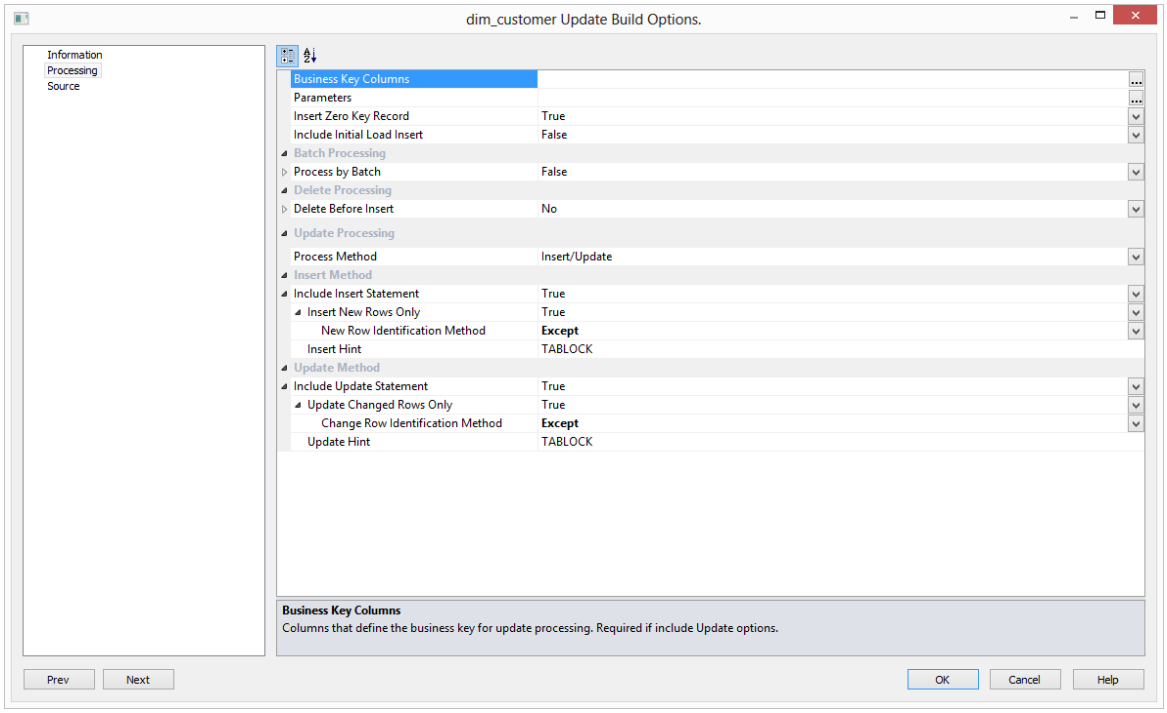
- To generate a procedure, right-click on the dimension name to edit the properties for the dimension.
- From the **Update Procedure** drop-down list, select **(Build Procedure...)**.
- Click **OK** to update the properties and start the process of generating the new procedure.
- A **Procedure Build Type** dialog will appear allowing to select between a **Cursor** and **Set** procedure build types from the drop-down list.

A **Set** based procedure performs one SQL statement to join all the source tables together and then insert this data into the Dimension. This is normally the fastest method of populating a table. See below in this section for different options using the **Cursor** procedure building type.

Set Based Procedure Building types

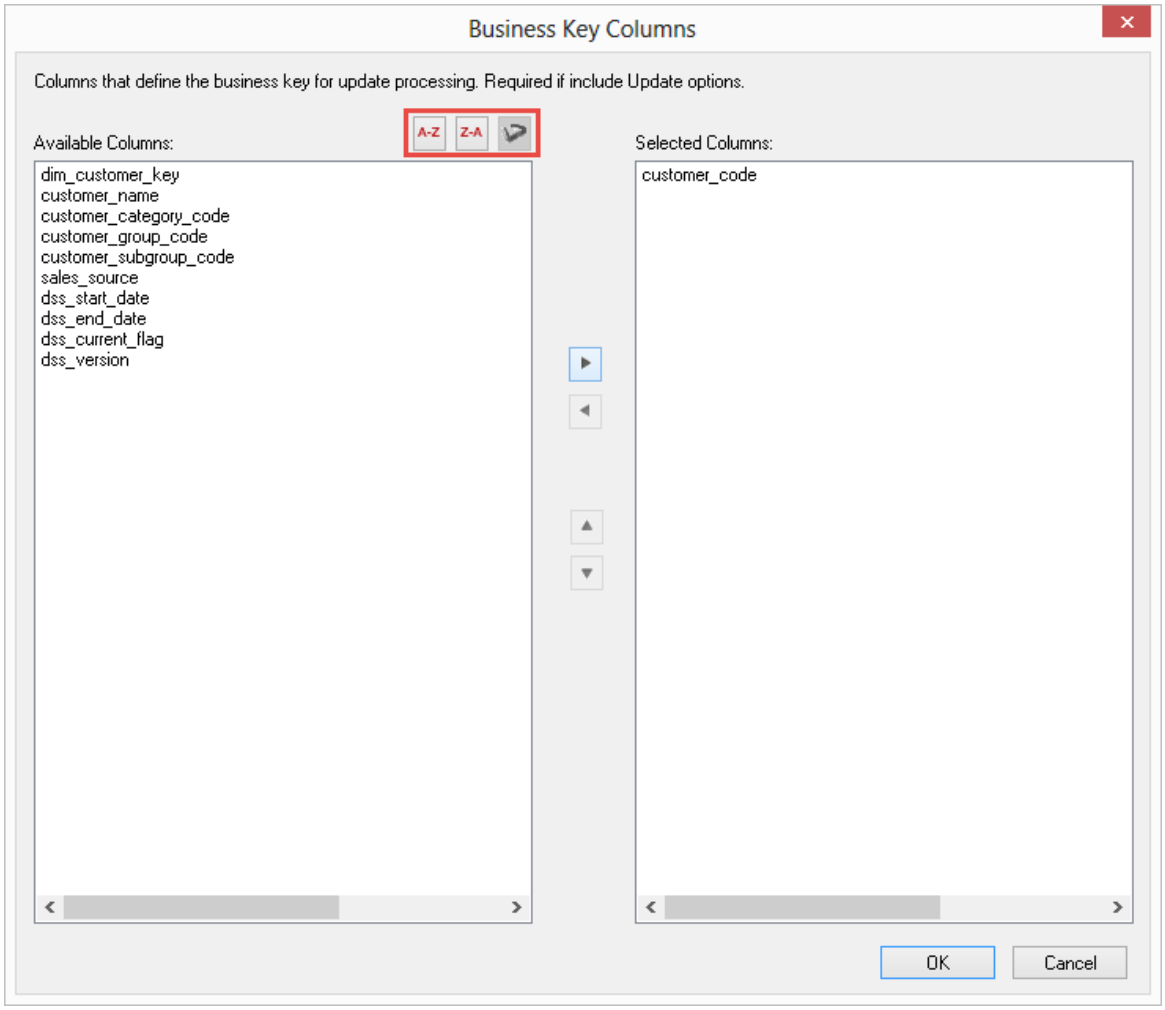
- Select Set from the drop-down menu.
- The following dialog appears to enter the Update Procedure Build Options.
- Click on the **ellipsis** button on the top rightmost corner to select the **Business Key Columns** for the dimension.

PROCESSING TAB



BUSINESS KEY

A business key can be made up of multiple columns, but it must provide a unique identifier. Where multiple columns uniquely and separately identify the dimension, choose one to act as the primary business key.
For example, a source table may have a unique constraint on both a product code and a product description. Therefore, the description as well as the code must be unique. It is of course possible to combine the two columns, but the normal practice would be to choose the code as the business key.

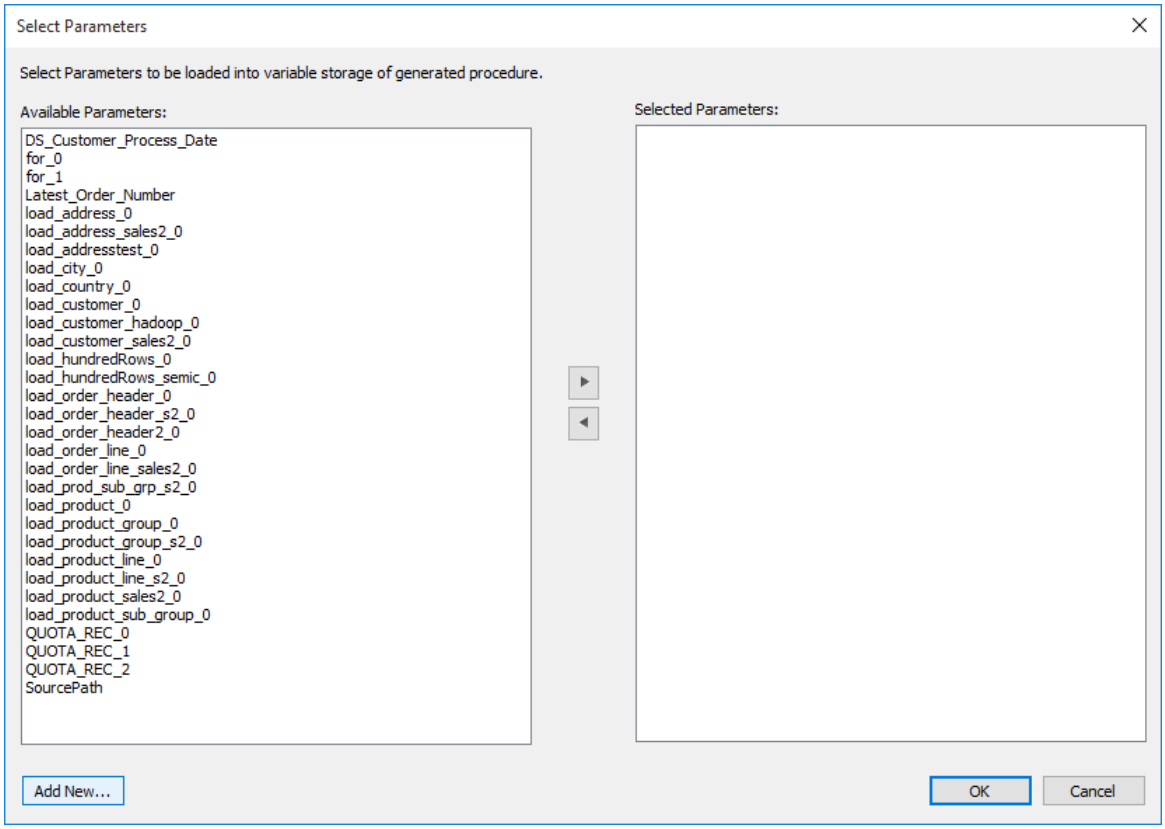


TIP1: Use the column name ascending/descending buttons to sort column names. To revert to the meta column order, click on the meta column order button.

TIP2: NULL Values - none of the columns chosen as the business key should ever contain a NULL value. See the note at the start of the Dimensions chapter.

PARAMETERS

If WhereScape RED parameters exist in the metadata, the following dialog is displayed. Any parameters selected in this dialog (by moving them to the right pane), are included in the generated update procedure as variables.
The procedure will include code to retrieve the value of the parameter at run time and store it in the declared variable.



The variables can also be used in column transformations and in the from/where clause for the update procedure. Some databases have a 30 character limit for variable names. WhereScape RED ensures the variables added for any parameters are less than 30 characters long by creating variable names in the form v_ followed by the first 28 characters of the parameter name.
For example, a parameter called MINIMUM_ORDER_NUMBER_SINCE_LAST_SOURCE_LOAD will be available as the variable v_MINIMUM_ORDER_NUMBER_SINCE_L.

TIP1: WhereScape RED parameters should be unique within the first 28 characters to avoid conflicting variables names.

TIP2: If the desired parameter doesn't exist in the metadata yet, a new parameter can be added by clicking on the **Add New** button on the bottom leftmost corner of the Select Parameters dialog.

See ***Parameters*** (on page 139) for more information on WhereScape RED Parameters.

Oracle **Enable Parallel DML -** adds all the code required to the update procedure to enable Oracle parallel inserts. The default for this option is not set.

Insert Zero Key Record - this option allows you to include the zerokey (unknown record) for tables with an artificial key. The default for this field is set.

Include Initial Load insert - This option adds an additional insert statement to the update procedure that runs if the target Dimension Object is empty. The benefit of this is improved performance inserting into an empty table without performing any checks to see if rows already exist. The default for this field is FALSE (i.e. an initial insert statement is not added to the procedure).

Process by Batch - This field allows the user to select a column to break up the data being processed in a loop based on the distinct values in the field. The update procedure loops on this field and performs the delete, update and/or insert for each value. If the field chosen is a date datatype (date, datetime or timestamp), then the user can specify yearly, monthly, daily or column level looping. The default for this field is False (do not do batch processing).

Batch Processing Field - allows selecting a field to batch process on. If you select a date field, you can process by date part. If you select a join field to process by you can choose and attribute of that related table to group by.

Delete before insert - this option enables a delete statement to be added to the update procedure before any update or insert statement. This is a particularly useful option for purging old data and for updates based on a source system batch number. The default for this field is No which automatically grays out the *Issue Warning if a Delete Occurs* and the *Delete Where Clause fields*.

Issue Warning if a Delete occurs - sets the procedure to a warning state if deletes occur.

Delete Where Clause - The delete where clause is appended to the generated delete statement to constrain the rows deleted.

Process Method - allows selecting whether the table should be updated using an Insert/Update statement or a Merge statement.

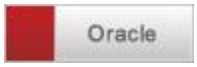
Include Insert Statement - the Include Insert Statement option includes an insert statement in the procedure to insert new rows in the Dimension. If this option is set, the **Insert New Rows Only** option is available. If this option is turned off, the update procedure will not contain an insert statement. The default for this field is set (i.e. an insert statement is included).

Insert New Rows only - the insert new rows only option uses change detection to work out what rows require inserting.

New Row Identification Method - Method used to identify that the records in the source are not currently recorded in the target table.

Insert Hint – enter a database hint to be used in the INSERT statement. This is an Oracle and SQL Server only option. Defaults can be configured in **Tools/Options/Default Update Procedure Options**.

Default is TABLOCK.



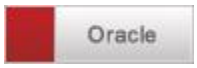
Default is APPEND.

Include Update Statement - update Statement using except/not exists - The Include Update Statement option includes an update statement in the procedure to update changing rows in the Dimension.

Update Changed Rows only - this option uses change detection to work out what rows require updating.

Update Hint – enter a database compliant hint to be used in the UPDATE statement. This is an Oracle and SQL Server only option. Defaults can be configured in **Tools/Options/Default Update Procedure Options**.

Default is TABLOCK.



Default is APPEND.

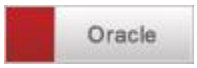
Merge Changed Rows Only – method used to identify that records in the source are recorded differently in the target table.

Delete Rows Not in Source - the Delete Rows Not in Source option allows you to delete all the rows which are not in the source but in the target table as part of the Merge statement. To avoid any accidental deletions, this option is de-activated if **Merge Changed Rows only** is enabled.

New Row Identification Method - method used to identify that records in source have changed from what is currently recorded in the target table.

Merge Hint – enter a database hint to be used in the MERGE statement. This is an Oracle and SQL Server only option. Defaults can be configured in **Tools/Options/Default Update Procedure Options**.

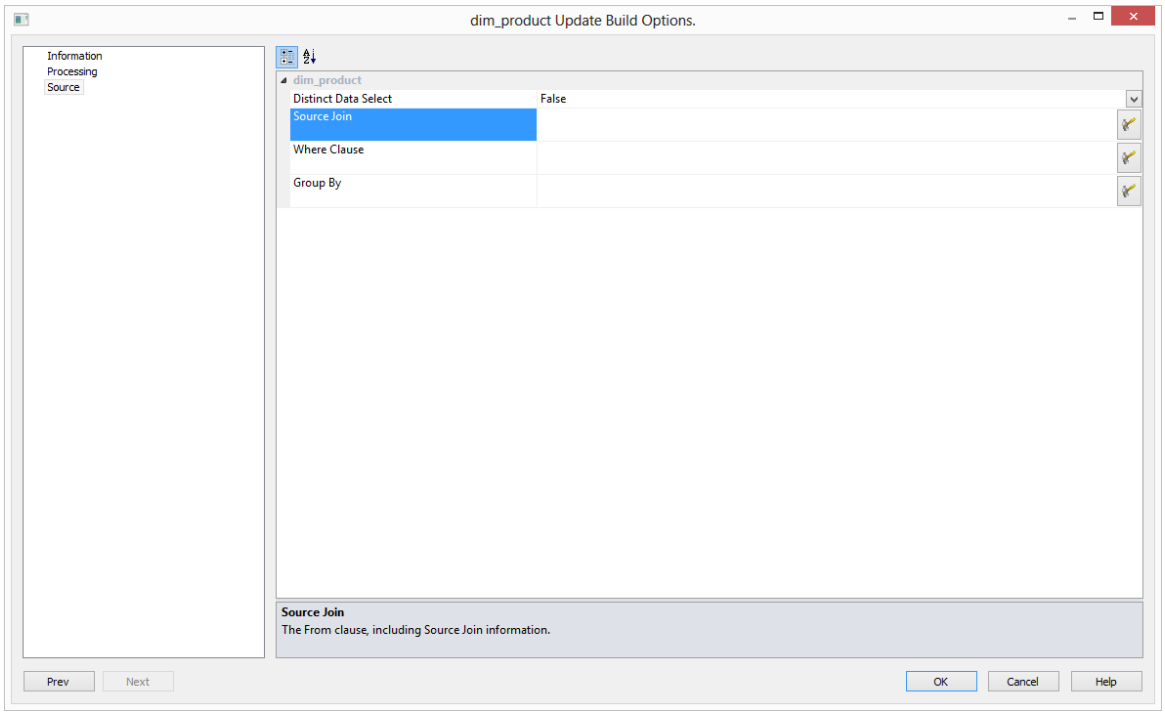
Default is TABLOCK.



Default is APPEND.

SOURCE TAB

If only one source table was used, adding the **Business Key** and checking the above fields is enough to proceed, otherwise use the **Source** tab to **Join** the relevant tables.



Distinct Data Select - The Distinct Data Select option ensures duplicate rows are not added to the Dimension.
This is achieved by the word DISTINCT being added to the source select in the update procedure. The default for this field is off (i.e. duplicates are not removed).

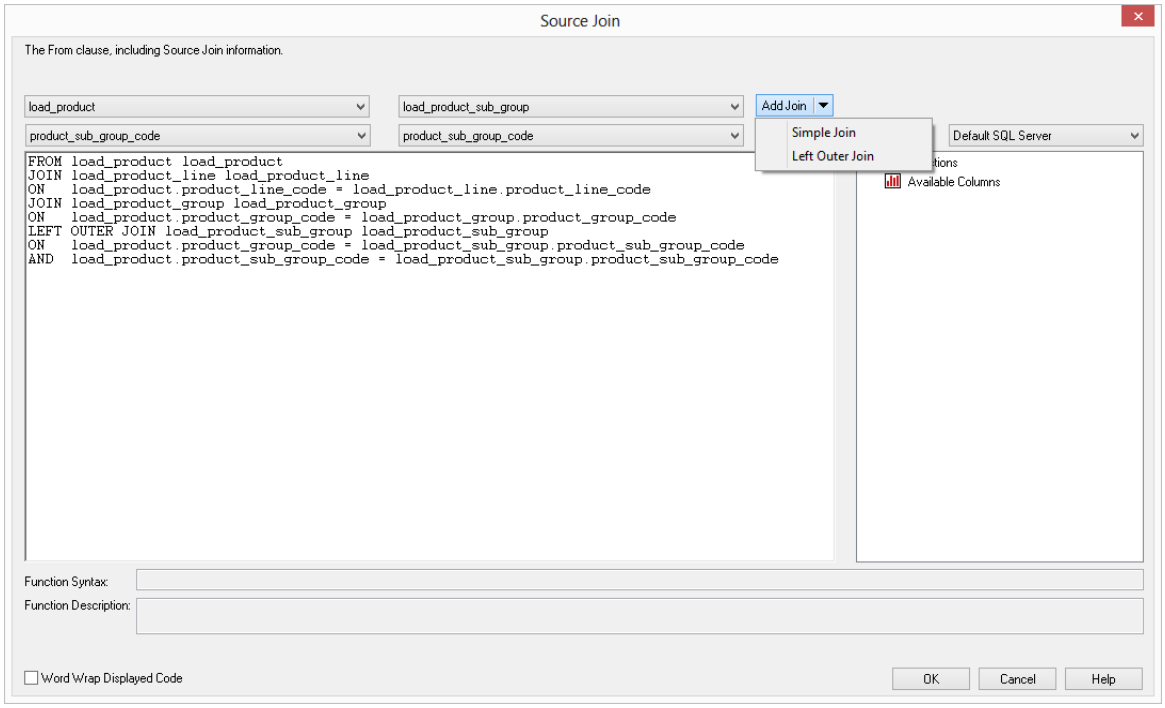
Source Join - The From clause, including Source Join information.

Where Clause - The Where Clause

Group By - The Group by clause.

Joining multiple source tables

Select the column or columns that need to be joined for this dimension and click on the **Add Join** button shown below to add this as a **Simple** or as a **Left Outer Join**.
The columns for the two tables then appear at the top of the dialog and one column is selected from each drop-down list to effect the join between the selected tables.
In the example below the load_product and load_prod_subgroup tables are joined by two columns namely prod_group and subgroup. In such a case two joins are actioned for these two tables, so that both columns can be selected.



Simple Join

A simple join joins the two tables via a 'Where' clause in Oracle. In SQL Server either a where or from clause join can be generated. In DB2, only a from clause join can be generated.
A simple join only returns rows where data is matched in both tables. So for example if table A has 100 rows and table B has a subset of 24 rows. If all the rows in table B can be joined to table A then 24 rows will be returned. The other 76 rows from table A will not be returned.

Outer Join

An outer join joins the two tables via a 'Where' clause in Oracle. In SQL Server either a 'Where' or from clause join can be generated. In DB2, only a from clause join can be generated.
The ANSI standard **From clause** join is recommended as multiple 'Where' clause outer joins will return incorrect results under SQL Server. The outer join returns all rows in the master table regardless of whether they are found in the second table. So if the example above was executed with table A as the master table then 100 rows would be returned. 76 of those rows would have null values for the table B columns.
When WhereScape RED builds up a 'Where' clause join it must place the outer join indicator next to the appropriate column. As this indicator goes on the subordinate table in Oracle, and the master in SQL Server, WhereScape RED needs to know which table is master and which subordinate. Select the join column from the master table first. In the example screen above the table 'load_product' has had its column chosen and the column for the table 'load_prod_subgroup' is currently being chosen. This will result in the 'load_product' table being defined as the master, as per the example statement as shown in the 'Where' clause edit window above. The results of this example select are that a row will be added containing product information regardless of whether a corresponding prod_subgroup entry exists.
As the join columns are selected the join statement is built up in the large edit window on the right. Once all joins have been made the contents of this window can be changed if the join statement is not correct. For example, the Oracle outer join indicator (+) could be removed if required.

NOTE: When upgrading from a RED version before 6.8.2.0 and moving existing objects to a target location, all procedures that reference those objects will need to be rebuilt.
Any **FROM** clauses will also need to be manually regenerated for the table references to be updated to the new [TABLEOWNER] form.

Once satisfied with the join clause click **OK** to proceed to the next step.
For **Oracle** this clause is the 'Where' clause that will be applied to the select statement of the cursor to allow the joining of the various tables.
For **SQL Server** this clause will be either the 'Where' clause or a combined from and 'Where' clause depending on the option chosen. This clause can of course be edited in the procedure that is generated if not correct.
For **SQL Server** data warehouses you have the choice between 'Where' statement joins and ANSI standard joins.
For **DB2** data warehouses, only ANSI standard joins are available.

NOTE: 'Where' joins will not work correctly in SQL Server if more than two tables are used in an outer join.

Change Detection tab

For Slowly Changing columns. If the dimension was defined as a slowly changing dimension the Change Detection tab will display on the Update Build Options dialog.
A change detection field(s) must be selected for this Dimension type.

Change Detection Fields - by clicking on the ellipsis button, this allows to select the change detection fields that are required for the dimension

Null Support - If the Add Null support is selected, the change detect column management will cater for null values in any change detect columns. If this option is not selected and null values are present, errors may occur running the update procedure. The default for this value if off (i.e. null are not catered for).

Update Current Records Only - The update current record only option will only apply changes to non-change detect columns on the current record. If this option is not selected, all past (non-current) rows will also be updated to reflect changes to non-change detect columns. The default for this value is on (i.e. only the current record is updated).

Reset Dates to Initial Values - Resets dss_start date and dss_end_date date values to original values.

Start Date for Initial Member - The start date for initial member field contains the start date for the first version of a particular business key. The value should be specified in an appropriate form, taking into account the default date format in the databases. The date may need to be explicitly cast to the current data type. The default value provided will usually be cast to the correct database and can be treated as a template. The default for this field is 1 January 1900.

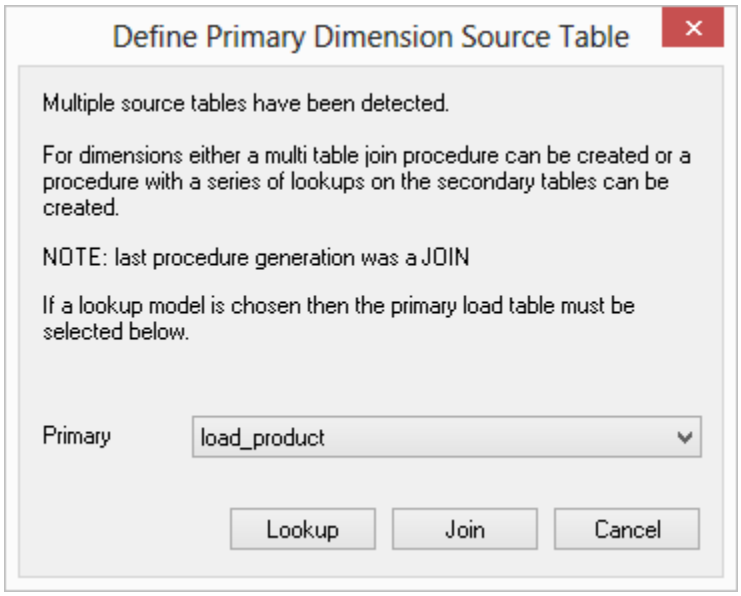
End Date for Current Member - The end date for current member field contains the end date for the current version (the row with a current flag of Y and the maximum version number) of a particular business key. The value should be specified in an appropriate form, taking into account the default date format in the databases. The date may need to be explicitly cast to the current data type. The default value provided will usually be cast to the correct database and can be treated as a template. The default for this field is 31 December 2999.

Start Date for New Member Entry - The start date for new member entry field contains the start date for any subsequent rows added to the history table (not the first row for a particular business key i.e. not version 1). The value should be specified in an appropriate form, taking into account the default date format in the databases. The date may need to be explicitly cast to the current data type. The default value provided will usually be cast to the correct database and can be treated as a template. The default for this field is the current date and time.

End Date for Expiring Member Entry - The end date for the expiring member entry field contains the end date for any rows updated no longer to no longer be the current row in the history table (i.e. rows that are replaced by a new current row). The value should be specified in an appropriate form, taking into account the default date format in the databases. The date may need to be explicitly cast to the current data type. The default value provided will usually be cast to the correct database and can be treated as a template. The default for this field is the current date and time less an arbitrary small amount (for SQL Server this is 0.00000005 of a day, or about 4 thousandth of a second).

SOURCE JOIN / LOOKUP

If columns from multiple source tables were used to create the dimension with a **Cursor** Procedure building type, the dialog below should prompt users to Define the Primary Dimension source table.
The fields will need to be joined on the Source Join field.

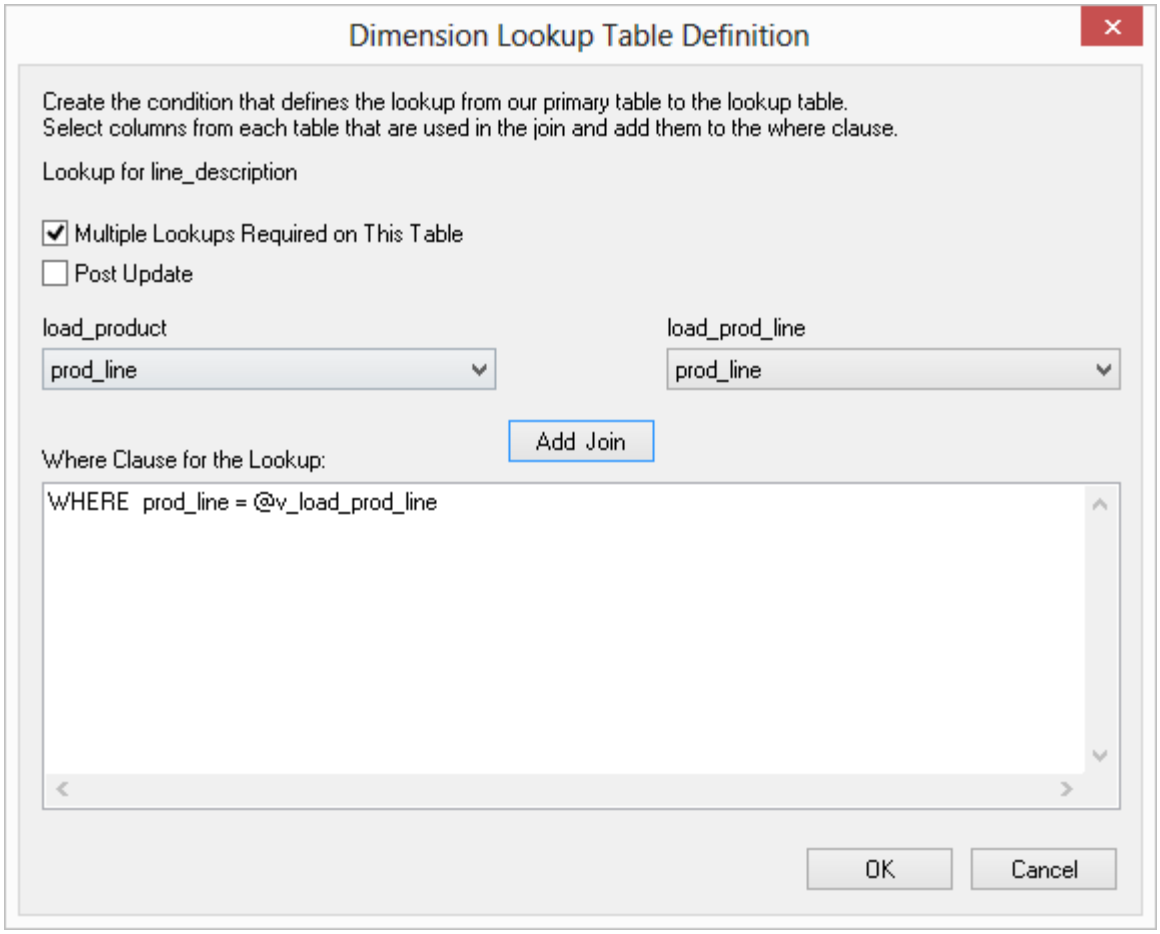


The choice provided is either to create a join that will combine all the information from the various tables into one large select statement, or to do a series of lookups.
The normal process would be to join the tables to create one large select statement. There are however situations where this is not desirable or possible. When a large number of tables are involved then a join will be slower than a series of lookups. In some situations, we may not have all the information to do an initial join. A series of lookups may be required to build up the information.
For complex situations it will be necessary to edit and enhance the generated procedure code.
If Lookup is chosen then the primary table for the dimension must be chosen. The philosophy here is that a series of lookups will be conducted against the secondary tables using information from this primary table.

LOOKUP

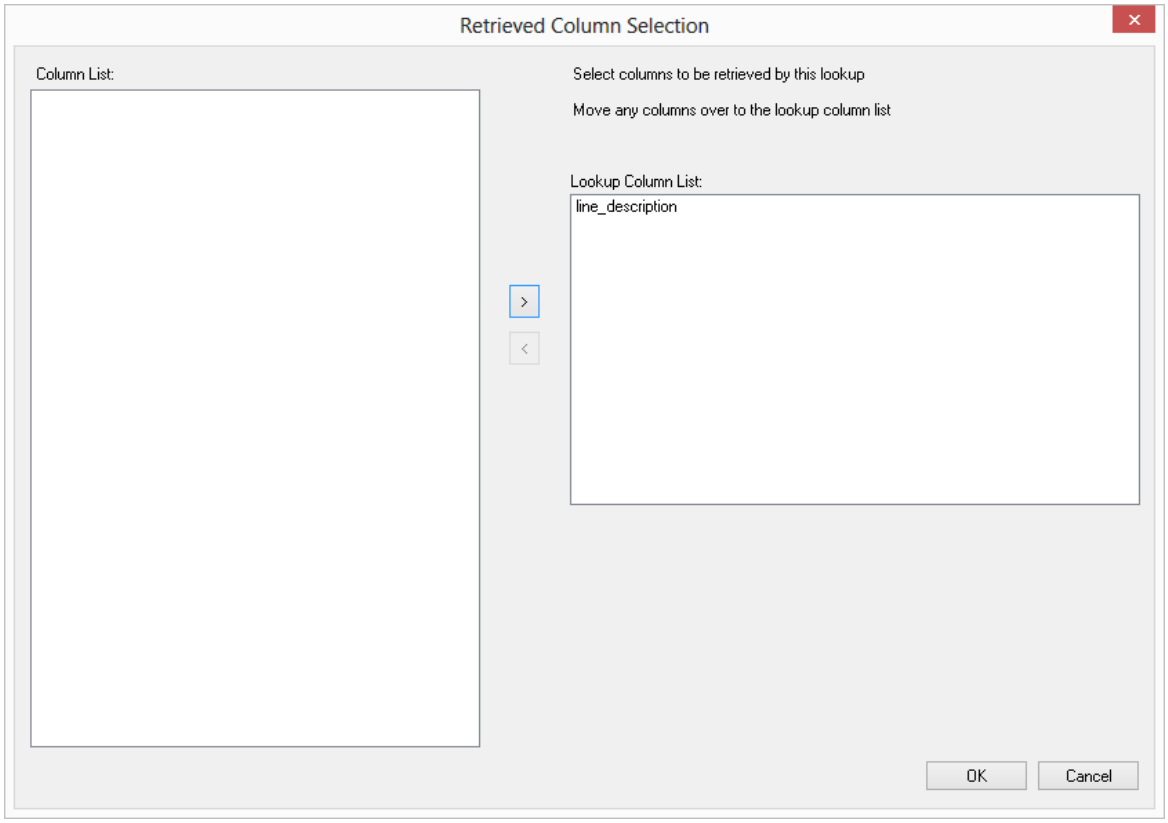
If multiple source tables were used to build the dimension, then an option would have appeared in an earlier dialog to choose between a join or a lookup.
If **Lookup** was chosen, then a dialog box will appear prompting for the joins between the primary table and each secondary (lookup) table.

The example below shows the lookup dialog for two tables from the tutorial data set.
The load_product table is the main table and a lookup is being performed on the load_prod_line table. Columns are selected from the drop-down column list for each table and then the **Add Join** button clicked to add these two columns to the 'Where' clause as shown in the bottom box. This 'Where' clause may be edited and goes to form a **lookup** select statement within the main cursor loop of the update procedure.
In this example the 'Where' clause has had the second line manually added to identify the type of lookup in the code table.



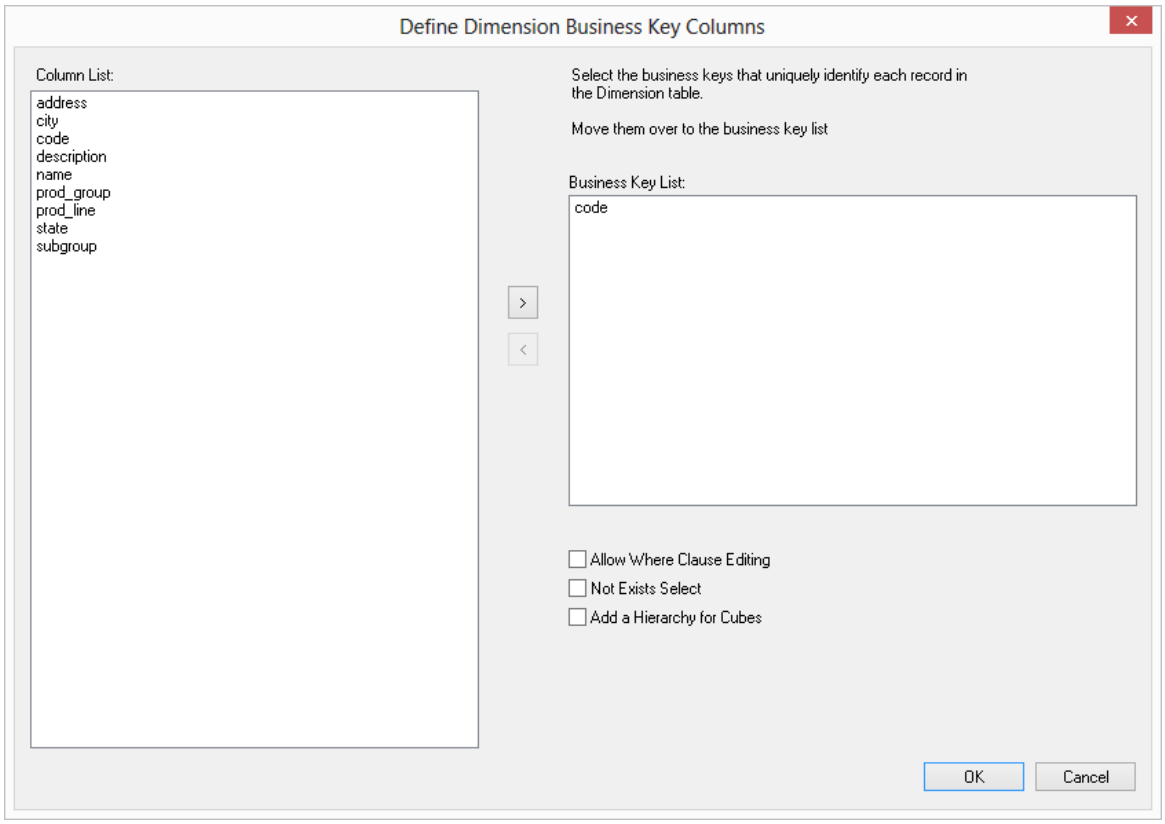
Multiple lookups can occur on the same table. Where we have a generic code table, for example, we may do multiple lookups to get descriptions.
The check-box **Multiple Lookups Required on This Table** will result in repeated lookups against the second table until all columns sourced from this table have been utilized in the lookup statements.

If the **Multiple Lookup** check-box is ticked, then the following dialog will appear to allow selection of the columns that are to be retrieved by the lookup.



BUSINESS KEY

Define the **Business Key** Columns by adding one or more business key columns to the right pane on the Define Business Key Columns dialog.



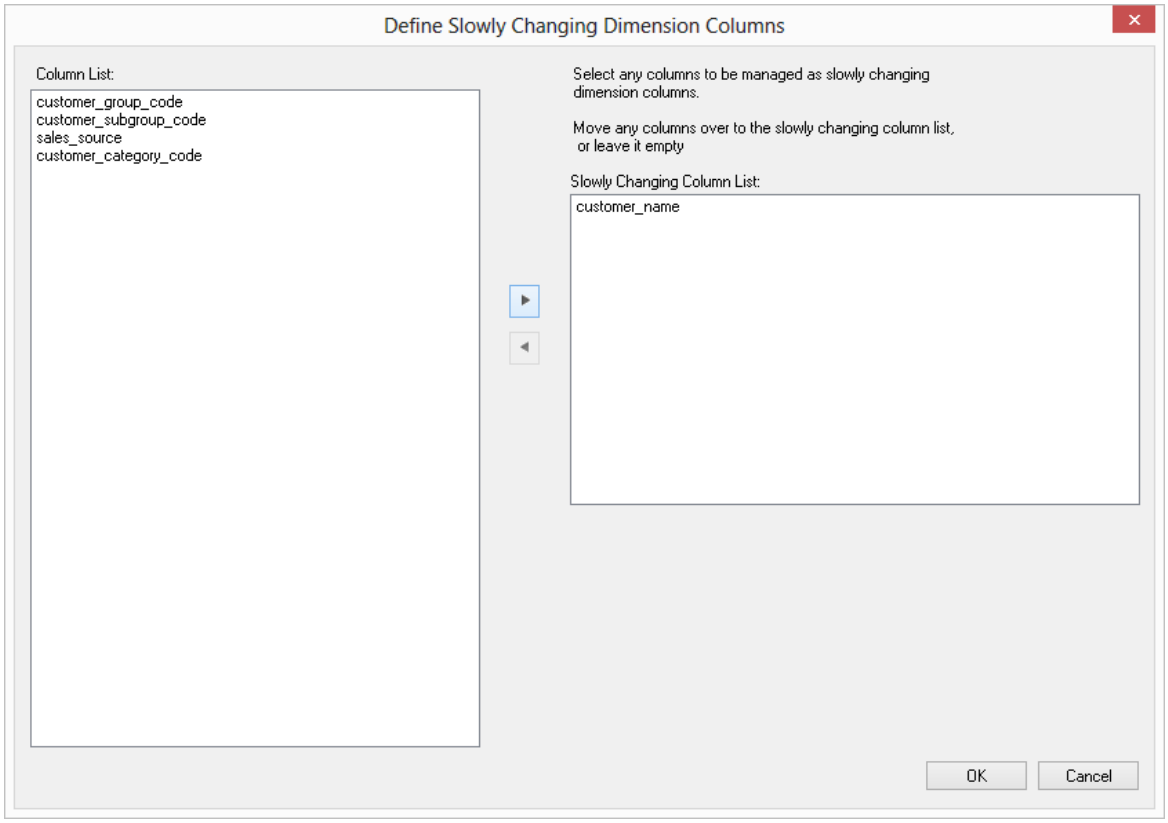
Allow Where Clause Editing

Not Exists Select - when this check-box is ticket, it will add additional code to the main select statement. This code will exclude every row in the source table that has not changed in the dimension table and can result in faster execution of the update procedure for large dimensions.

Add a hierarchy for Cubes - this check-box will produce another dialog that allows the definition of a simple hierarchy structure. This is useful if the hierarchy of the dimension is known and the dimension will be used in the creation of Analysis services cubes. Hierarchies may be defined now or later when required. They may also be defined or modified after the dimension has been created.

SLOWLY CHANGING COLUMNS

If the dimension was defined as a slowly changing dimension, then the following additional dialogs will appear.
The first dialog requests the selection of the columns to be managed as slowly changing dimension columns.
Select the required columns and click **OK** to proceed.
In the example below the customer name is to be managed as a slowly changing column in a customer dimension.



The advantages and disadvantages of slowly changing dimensions are discussed earlier in this chapter, but as a rule try to minimize their usage, as they invariably complicate the processing and end user queries.
The following dialog will appear if one or more columns are chosen for management.

NULL values are the enemy of a successful data warehouse. They result in unreliable query results and can often lead to a lack of confidence in the data.
If a column is considered important enough to be managed as a slowly changing column, then it should not normally contain null values. It is often best to ensure that a Null cannot occur by using a IsNull() (SQL Server), Nvl() (Oracle) or Coalesce() (DB2) transformation when loading the column.

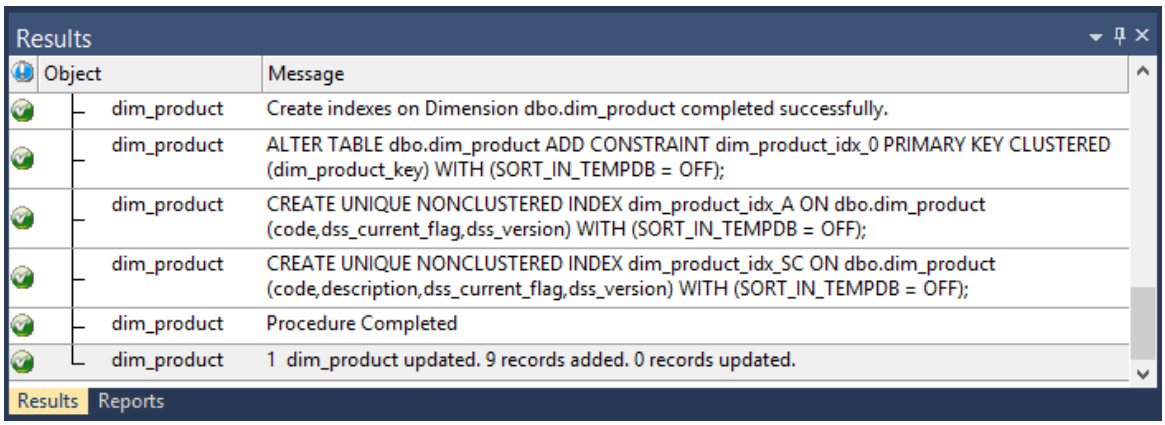
If however, this situation is unavoidable then answering **Yes** to this question will result in additional code to test for Nulls during the update of the dimension records.
If **No** is answered and Nulls are encountered, then a unique constraint violation will occur during the dimension update.

Building and Compiling the Procedure

Once the above questions are completed the procedure is built and compiled automatically.
If the compile fails an error will be displayed along with the first few lines of error messages.
Compile fails typically occur when the physical creation of the table was not done.
If the compile fails for some other reason the best approach is to use the procedure editor to edit and compile the procedure. The procedure editor will highlight all the errors within the context of the procedure.
Once the procedure has been successfully compiled it can either be executed interactively, or passed to the scheduler.

Indexes

By default, a number of indexes will be created to support the dimension. These indexes will be added once the procedure has been built.
An example of the type of indexes created is as follows:



This example shows three indexes being created. They are:

1. A primary key constraint placed on the artificial key for the dimension.
2. A unique index placed on the business key for the dimension.
3. A unique index placed on the business key and a slowly changing column from the dimension.

This third index is only created when a **Slowly Changing** dimension is chosen.
Additional indexes can be added, or these indexes changed. See the chapter on indexes for further details.

DIMENSION ARTIFICIAL KEYS

The artificial (surrogate) key for a dimension is set via a sequence in Oracle and an identity column in SQL Server and DB2.
This artificial key normally, and by default, starts at one and progresses as far as is required.

A WhereScape standard for the creation of special rows in the dimension is as follows:

Key value	Usage
1 upwards	The normal dimension artificial keys are numbered from 1 upwards, with a new number assigned for each distinct dimension record.
0	Used as a join to the dimension when no valid join existed. It is the normal convention in the WhereScape generated code that any dimension business key that either does not exist or does not match is assigned to key 0.
-1 through -9	Used for special cases. The most common being where a dimension is not appropriate for the record. For example, we may have a sales system that has a promotion dimension. Not all sales have promotions. In this situation, it is best to create a specific record in the dimension that indicates that a fact table record does not have a promotion. The stage table procedure would be modified to assign such records to this specific key. A new key is used rather than 0 as we want to distinguish between records that are invalid and not appropriate.
-10 backward	Pseudo records. In many cases, we must deal with different granularities in our fact data. For example, we may have a fact table that contains actual sales at a product SKU level and budget information at a product group level. The product dimension only contains SKU based information. To be able to map the budget records to the dimension we need to create these pseudo keys that relate to product groups. The values -10 and backwards are normally used for such keys. A template called 'Pseudo' is shipped with WhereScape RED to illustrate the generation of these pseudo records in the dimension table.

DIMENSION GET KEY FUNCTION

When WhereScape RED is asked to generate a new procedure to support the updating of a dimension it also generates a **Get Key** function (procedure in SQL Server and DB2).
This function is also generated when dimension views are created, and can be generated for retro-fitted dimensions.

The **Get Key** function is used to return an artificial key when supplied a business key. The normal syntax is to call the function passing the business key and be returned a status indicating the result of the lookup.
On a successful lookup, the artificial key for the dimension record is also returned. If the lookup fails because the business key is not found in the dimension, then an appropriate status is returned and the artificial key is set to 0.

NOTE: Two parameters exist on the generated **Get Key** function that allow manipulation of the process. It is possible to automatically add a new dimension record when a match is not made. It is also possible to record the lookup failure in the detail/error log. By default, both these options are disabled.

Modifying the 'Get Key' function

There may be a situation where a dimension record can have multiple business keys.

For example: You may be able to look up a 'factory' based on the factory code, the name of the factory or its address. The generated **Get Key** function will only support the primary business key which in this case may be the factory code. However, we may have a fact record that only contains the factory name. In such a case, we need to modify the **Get Key** function or add a second function. The advantages of adding a second function are that any code generated in the future that uses this dimension will work without modification. The disadvantages are the multiple points of maintenance required when changing the dimension. Both options are discussed below:

Adding a second 'Get Key' function for a dimension

Choose a name similar to the existing function. If say we have a 'get_dim_factory_key' function then we may choose a name like 'get_factory_name_key'. Proceed to create and populate the function as follows:

Right-click on the **Procedure** group in the left pane and select **New Object** to create a new procedure. Enter the name and click **ADD**.

1. A Properties dialog will appear. Change the type to **Function** for Oracle, leave as a **Procedure** for SQL Server. Enter a description under the **Purpose** column and click the **UPDATE** button.
2. Expand the **Procedure** group in the left pane and double-click on the new object just created. This will start the editor.
3. Select the **Tools/View Procedure or Template** menu option. This will start a dialog box.
4. Select the existing **Get Key** function from the procedure drop-down. A new window will appear on the right showing the existing **Get Key** function.
5. Click the mouse in the existing function window. Right-click and choose **Select All** and then right-click again and select **Copy**.

6 Click the mouse in the main editor window and paste the existing function into the window. The existing function window can now be closed if desired.

7 Using the menu option **Edit/Replace**, change all occurrences of the old function name to the new name.

8 Change the various occurrences of the business key name and variable to match the new one and test it all.

Once a new **Get Key** function has been produced it will obviously be necessary to modify the stage update procedure to utilize the new function.

 **TIP**: An alternative approach is to create a dimension view specifically to generate a different kind of get key function. Then use this view's surrogate key in the stage table instead of the dimension table's surrogate key.
This approach avoids modifying the staging table procedure.

Extending the 'Get Key' function

If you want to keep all the information for the lookup of dimension keys in one location then extend the functionality of the existing **Get Key** function. The normal practice in this case is to include all potential business keys as parameters. Multiple select statements can be employed to select the keys based on the appropriate business key. To choose the correct statement either use Nulls in the unused business keys or include an additional parameter to indicate which business key is being passed.

If this method is used then any stage update procedures will need to be modified to handle the new calling syntax.

DIMENSION INITIAL BUILD

The initial population of a dimension with data can be achieved by generating a custom procedure and then; using the right-click on the dimension, choosing **Execute Custom Procedure via Scheduler**.
The dimension should be analyzed once the custom procedure is completed so that the database query optimizer can make use of the indexes.
For smaller dimensions (i.e. less than 500,000 rows) run the normal **Update** procedure against an empty dimension table. There is however a danger in this action in that the query optimizer will not have any information on the table and hence will do a full table pass when adding each new row. For a very small dimension this will not be an issue, but it will rapidly become a major problem as the row count increases.
The problem with the initial update is that the database does not know to use the index that has been created for the business key, and hence does a full table pass when attempting to update/insert a record.

To prevent this problem the generated code will issue an analyze statement after 1000 rows have been added to the table. The statement used is as follows:

SQL Server: update statistics *dim_table* **with sample 5 percent**
Oracle: analyze table *dim_table* **estimate statistics sample 5 percent for all indexed columns**
DB2: runstats on table *dim_table*
where *dim_table* is the name of the dimension.

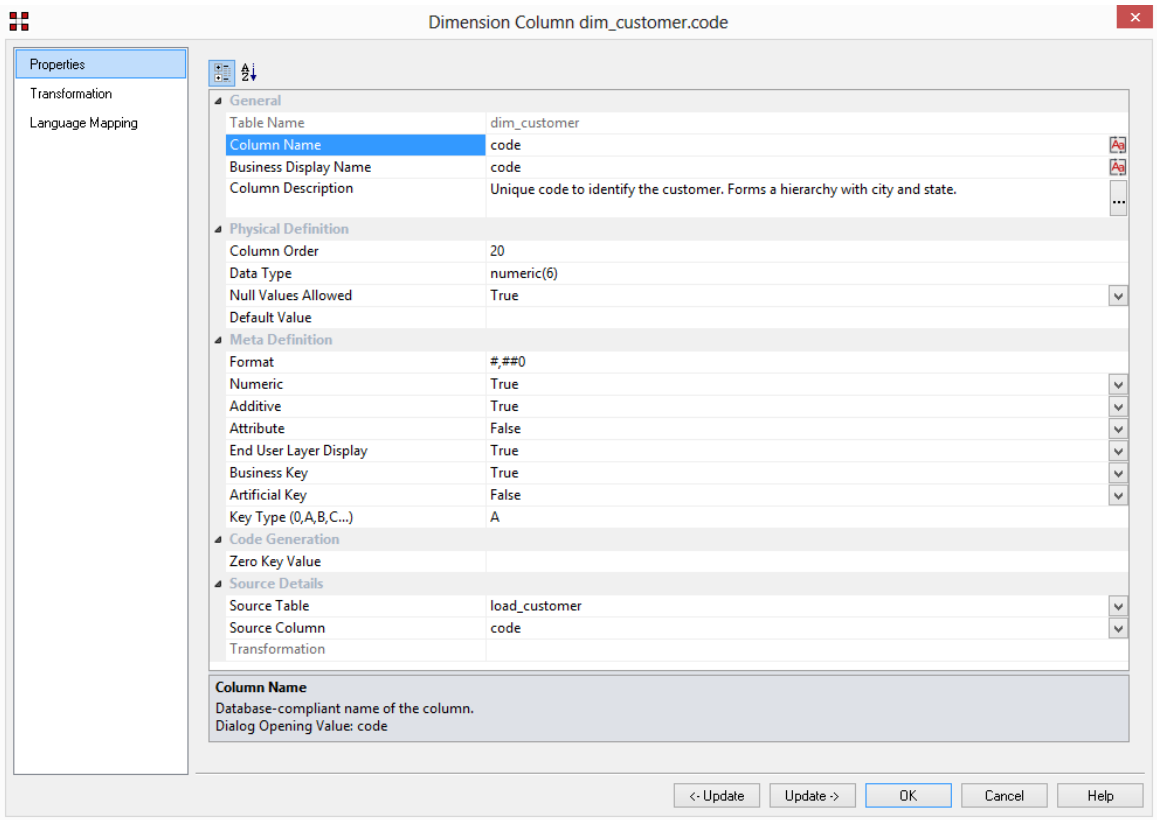
This command will be issued whenever 1000 or more new rows are added to a dimension.
If this is undesirable, then the code should be removed from the update procedure. It may be undesirable if a plan has been locked in for the table.

DIMENSION COLUMN PROPERTIES

Each dimension column has a set of associated properties. The definition of each property is described below:

TIP: If a database table's definition is changed in the metadata then the table will need to be altered in the database. Use the **Validate/Validate Table Create Status** to compare metadata definitions to physical database tables.
The option also provides the ability to alter the database table, through a pop-up menu option from the validated table name. See the example below.

A sample **Properties** screen is as follows:



TIP: The two special **update** keys allow you to update the column and step either forward or backward to the next column's Properties.
ALT-Left Arrow and **ALT-Right Arrow** can also be used instead of the two special update keys.

Table Name

Database-compliant name of the table that contains the column. [Read-only].

Column Name

Database-compliant name of the column. Typically, column-naming standards exclude the use of spaces etc. A good practice is to only use alphanumerics, and the underscore character. Changing this field alters the table's definition.

Note: A case conversion button on the right converts the text between different cases: UPPER CASE, Capitalized Case and lower case. The mode cycles to the next case in the sequence each conversion.

Business Display Name

Name that the business uses to refer to the column, which is included in the RED-generated documentation and can be used in the end user layer of other tools. [Does NOT affect the physical database table]. As such it is a free form entry and any characters are valid.

Note: A case conversion button on the right converts the text between different cases: UPPER CASE, Capitalized Case and lower case. The mode cycles to the next case in the sequence each conversion.

Column Description

This field contains the description for the column. It may be a description from a business user's point of view. This field might additionally contain information on where and how the column was acquired. For example, if the column is sourced from multiple tables or is a composite or derived column then this definition would normally describe the process used to populate the column. This field is used in the documentation and is available via the view **ws_admin_v_dim_col .** This field is also stored as a comment against the column in the database.

Column Order

Numeric value that controls the relative order of columns in the database create statement. The lowest numbered column will appear first in the table. Although this affects the physical table definition no action will be taken unless the table is re-created in the database. The columns can be re-numbered based on the existing order by choosing the **Respace Order Number** pop-up menu option when positioned over any column in the table. This action will number the columns in increments of 10 starting at 10. In addition to a simple change of the order field, the column order can be changed by first displaying the columns in the middle pane and then using drag and drop to move the columns around. This drag and drop process will automatically renumber the columns as required.

Data Type

Database-compliant data type that must be valid for the target database. Typical data types for Oracle are integer, number, char(), varchar2() and date. For SQL Server common types are integer, numeric, varchar() and datetime. See the database documentation for a description of the data types available. Changing this field alters the table's definition.

Null Values Allowed

Determines whether the table column can hold NULL values or whether a value is always mandatory.

Default Value

Initial value that is assigned to the column when a row is inserted into the table but no value is specified for the column.

Format

Optional format mask that can be used in end user tools. [Does NOT affect the physical database table]. As such it is a free form entry and any characters are valid. Typically format masks are only used on numeric fields. Example: #,###0.00. It is not worth the effort of populating this field unless it can be utilized by the end user tools in use.

Numeric

Indicates whether the table column holds values that are numeric. This is normally only relevant for fact tables. It does not affect the physical table definition, but rather provides input to the view **ws_admin_v_dim_col** which can be used to assist in the population of an end user tool's end user layer. The use of this field is not relevant unless it can be utilized by the end user tools.

Additive

Indicates whether the table column holds values that are additive. This implies that the column can be summed when performing data grouping in a query. This is normally only relevant for fact tables. It does not affect the physical table definition, but rather provides input to the view **ws_admin_v_dim_col** which can be used to assist in the population of an end user tool's end user layer. The use of this field is not relevant unless it can be utilized by the end user tools.

Attribute

Indicates whether the table column holds values that are descriptive, and/or are used for grouping/summing. An attribute is defined as a column that is non factual. For example, we may have an order number, or a invoice number stored in the fact table. Such columns are considered attributes, rather than facts. This checkbox is therefore normally only relevant for fact tables. This checkbox does not affect the physical table definition, but rather provides input to the view **ws_admin_v_dim_col** which can be used to assist in the population of an end user tools end user layer. The use of this field is not relevant unless it can be utilized by the end user tools.

End User Layer Display

Indicates whether the table column is available/visible to end users. If set the documentation will include the column in the glossary and in the user documentation. It is also used to decide what columns appear in the view **ws_admin_v_dim_col.** Typically columns such as the artificial key would not be enabled for end user display.

Business Key

Indicates whether the column is part of the BUSINESS key, which is defined during the update procedure build. [Normally maintained automatically]. Multiple columns can form the primary business key.

Artificial Key

Indicates whether the column is the generated artificial/surrogate key (unique identifier) for the table. Only one artificial key per table is supported. [Normally maintained automatically].

Key Type

Key type that is assigned and used when generating the table's update procedure and indexes. [Normally maintained automatically].
It can be altered here, but this should only be done if the consequences are fully understood and tested. The supported values are:

Key type	Meaning
0	The artificial key. Set when the key is added during drag and drop table generation.
1	Component of all business keys. Indicates that this column is used as part of any business key. For example: By default, the dss_source_system_key is added to every dimension table. It is considered part of any lookup on that table and has the key type set to 1. Set when the column is added during drag and drop table generation.
2	Indicates that this column is a dimensional join. Used on fact tables to indicate the dimension keys. Results in an index being built for the column (Bitmap in Oracle). Set during the update procedure generation for a fact table, based on information from the staging table.
3	Slowly changing column indicator. Used on dimension tables to indicate that the column is being managed as a slowly changing column within the context of a slowly changing dimension. Set when a column is identified during the dimension update procedure generation.
4	Previous value column indicator. Used on dimension tables to indicate that the column is being managed as a previous value column. The source column identifies the parent column. Set during the dimension creation.
5	Start date of a date ranged dimension. Used on dimension tables to indicate that the column is defined as the starting date for a source system date ranged dimension. Forms part of the business key. Set during the dimension creation.
6	End date of a date ranged dimension. Used on dimension tables to indicate that the column is defined as the ending date for a source system date ranged dimension. Forms part of the business key. Set during the dimension creation.
A	Indicates that the column is part of the primary business key. Set whenever a business key is defined as part of an update procedure generation.
B-Z	Indicates that the column is part of a secondary business key. Only used during index generation and not normally set.

Zero Key Value

Determines the value populated for the column in the **Invalid Join** or **Unknown record**. By default, NULL is used when a value is not specified. All dimensions that use standard WhereScape RED generated procedures have a row with an artificial key of zero. This row is used to link to the fact records when no match on the dimension can be found. For example, we may have some sales records that come through without a product defined. WhereScape RED will be default associate these fact records with the zero key product dimension entry. So, we might set the zero key value to 'Unknown product' for the name column in the product dimension.

Source Table

Identifies the source table where the column's data comes from. This source table is normally a load table within the data warehouse. If the column was sourced from multiple tables, then the normal practice is to record one of the tables in this field and a comment listing all of the other tables in the **Source Strategy** field. This field is used when generating a procedure to update the dimension. It is also used in the track back diagrams and in the documentation.

Source Column

Identifies the source column where the column's data comes from. Such a column is normally a load table column, which in turn may have been a transformation or the combination of multiple columns. For previous value managed columns, the source column is the column in the table whose previous value is being recorded.

Transformation

Transformation. [Read-only].

Changing a Column Name

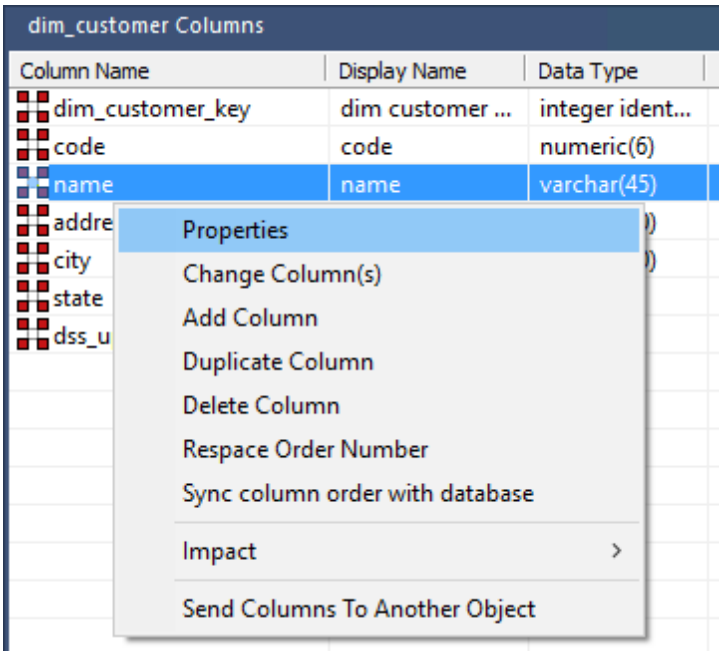
If the **Column name** or **Data type** is changed for a column, then the metadata will differ from the table as recorded in the database.
Use the **Validate/Validate Table Create Status** menu option or the right-click menu to compare the metadata to the table in the database.
|A right-click menu option of **Alter Table** is available when positioned on the table name after the validate has completed. This option will alter the database table to match the metadata definition.

For example: Analysis Services does not like **name** as a column name.

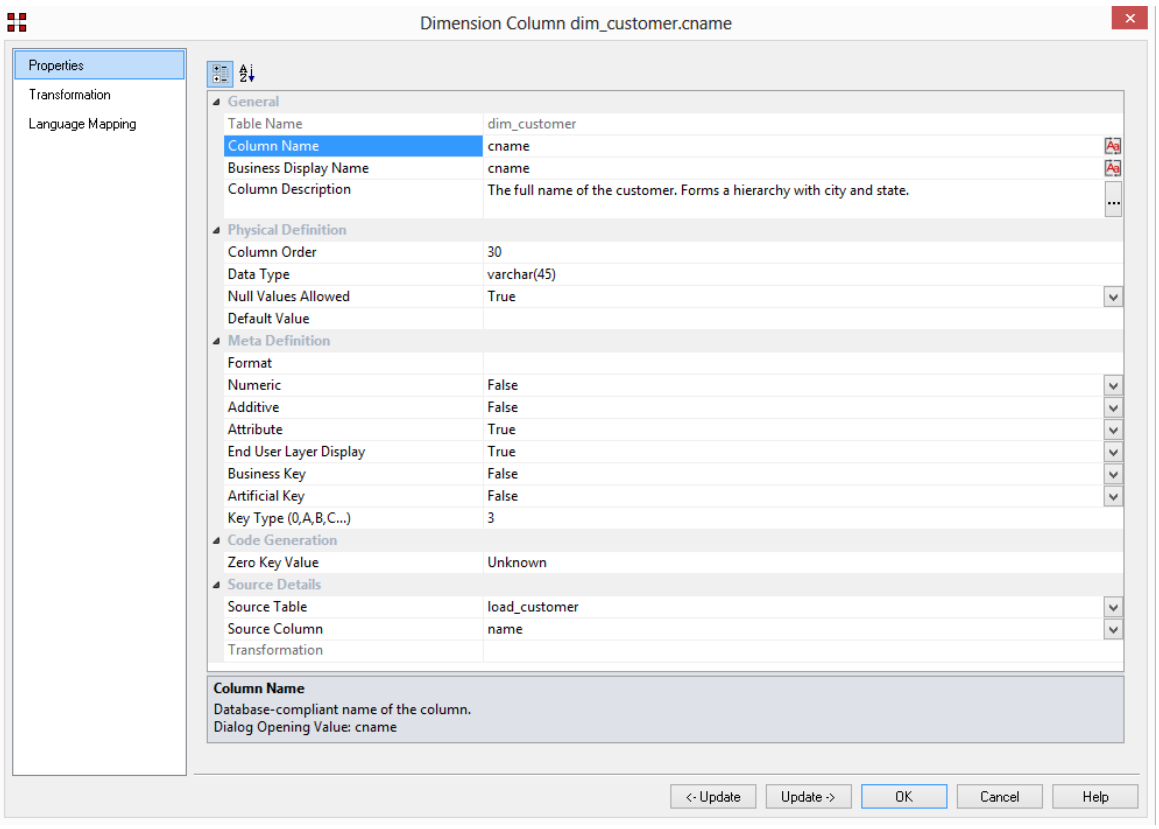
For dim_customer it will therefore be necessary to change the column name from **name** to **cname**.

1 Click on the **dim_customer** object in the left pane to display the dim_customer columns in the middle pane.

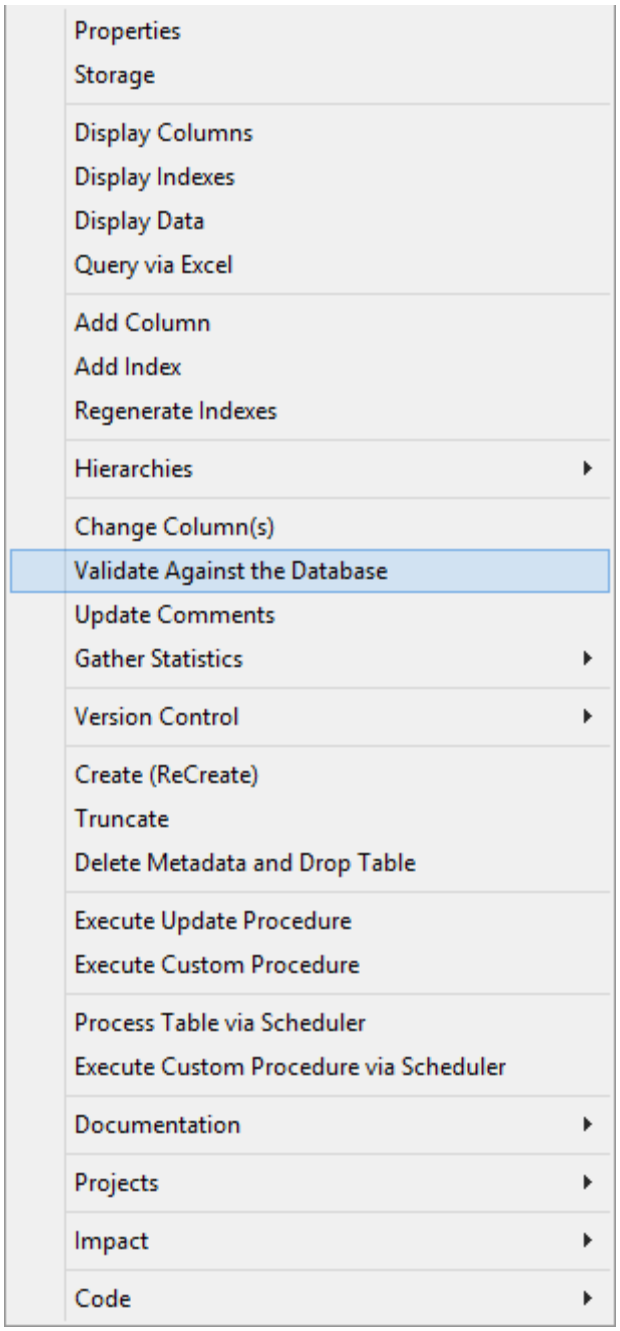
2 When positioned on the column **name** in the middle pane, right-click and select **Properties** from the drop-down menu.



3 Change the column name from **name** to **cname** as shown below. Click **OK** to leave the Properties page.

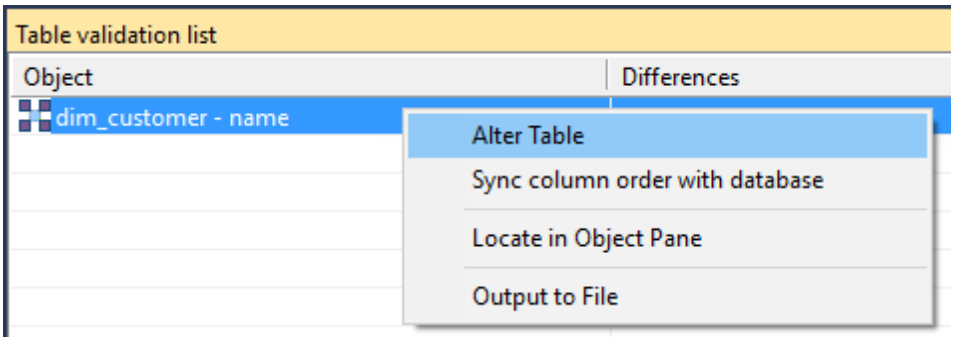


4 Right-click on the **dim_customer** object in the left pane and select **Validate against Database**.



5 The results in the middle pane will show that the metadata has been changed to **cname** while the column name in the database is still **name**.

6 Right-click on **dim_customer** in the middle pane and select **Alter table** from the drop-down list.



7 A warning dialog will appear, displaying the table and column name to be altered. Select **Alter Table**.

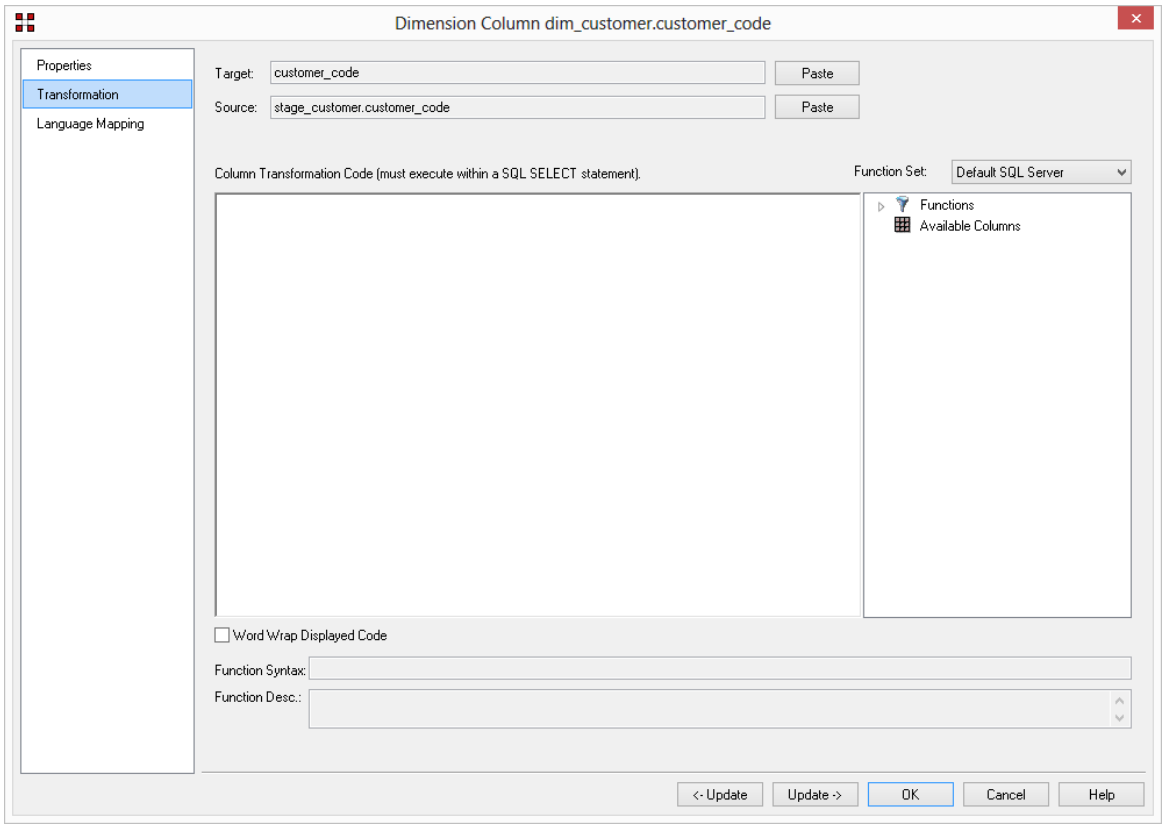
8 A dialog will appear confirming that dim_customer has been altered. Click **OK**.

DIMENSION COLUMN TRANSFORMATIONS

Each dimension table column can have a transformation associated with it. The transformation will be included in the generated procedure and will be executed as part of the procedure update. The transformation must therefore be a valid SQL construct that can be included in a **Select** statement.
For example, we could have a transformation of 'load_order_line.qty * 0.125' to calculate a tax column of 12.5%.

Click the **Transformation** tab to enter a transformation.

The transformation screen is as follows:



NOTE: Transformations are only put into effect when the procedure is re-generated.

Microsoft Analysis Services 2005+ Tabular Mode Tables: For Tabular Mode table column transformations, **Default DAX** is the only applicable Function Set for **after load** transformations.

See ***Transformations*** (on page 650) for more details.

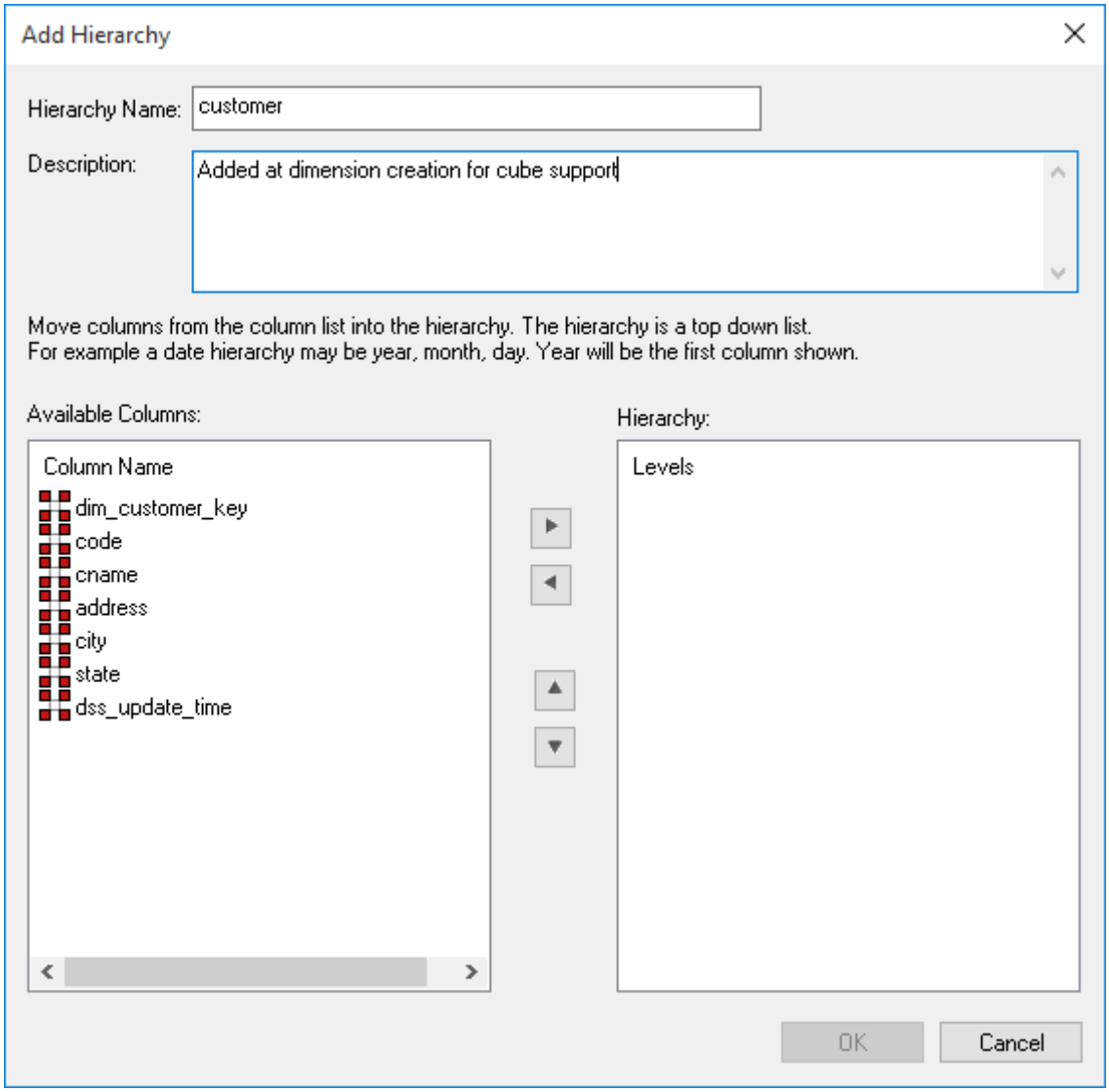
DIMENSION HIERARCHIES

The various hierarchies associated with a dimension can be recorded in the WhereScape RED metadata. These hierarchies are often not used in any form, except to provide documentary completeness.
This section describes the creation of hierarchies using WhereScape RED.

ADDING A DIMENSION HIERARCHY

Any number of hierarchies can be created against a dimension. There is no restriction on the form of the hierarchy.

- To add a new hierarchy, right-click on the dimension table in the left pane and select **Hierarchies/Add Hierarchy**.
- The following dialog will appear.



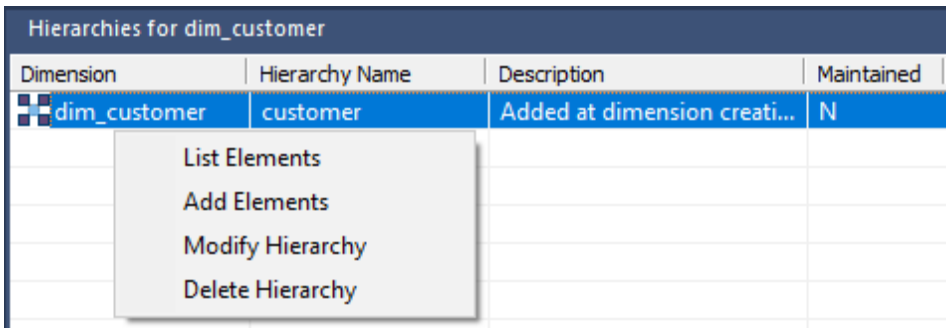
- Enter a meaningful name for the hierarchy.
- Enter a meaningful description for the hierarchy. This description is carried through into the Hierarchy Description field of any OLAP Dimensions that are built from the original Dimension object.

Note: The description text is automatically set to "Added at dimension creation for cube support" but this can be edited to match the user's intended description.

The hierarchy is built with the highest level at the top; for example, a customer dimension may have **category_code** at the highest level, then **group_code**, then **subgroup_code** and finally **name** at the lowest level.

To enter the hierarchy elements, select them, in the required order, from the left pane and click **>** to add them to the right pane.
Once all the hierarchy elements have been added, click **OK**.

A hierarchy and its elements can be edited by listing the hierarchies associated with a dimension and using the right-click menu options available in the middle pane.



Copying Dimension Hierarchies from Source

Hierarchies are automatically copied from a source table when the source table is dragged into the middle pane to create a new Dimension.

To copy the hierarchies from the source objects manually, right-click on a dimension in the Object Pane and select **Hierarchies/Copy Hierarchies from Source**. This feature is useful when the source for a Dimension has been updated to contain new hierarchies.

SNOWFLAKE

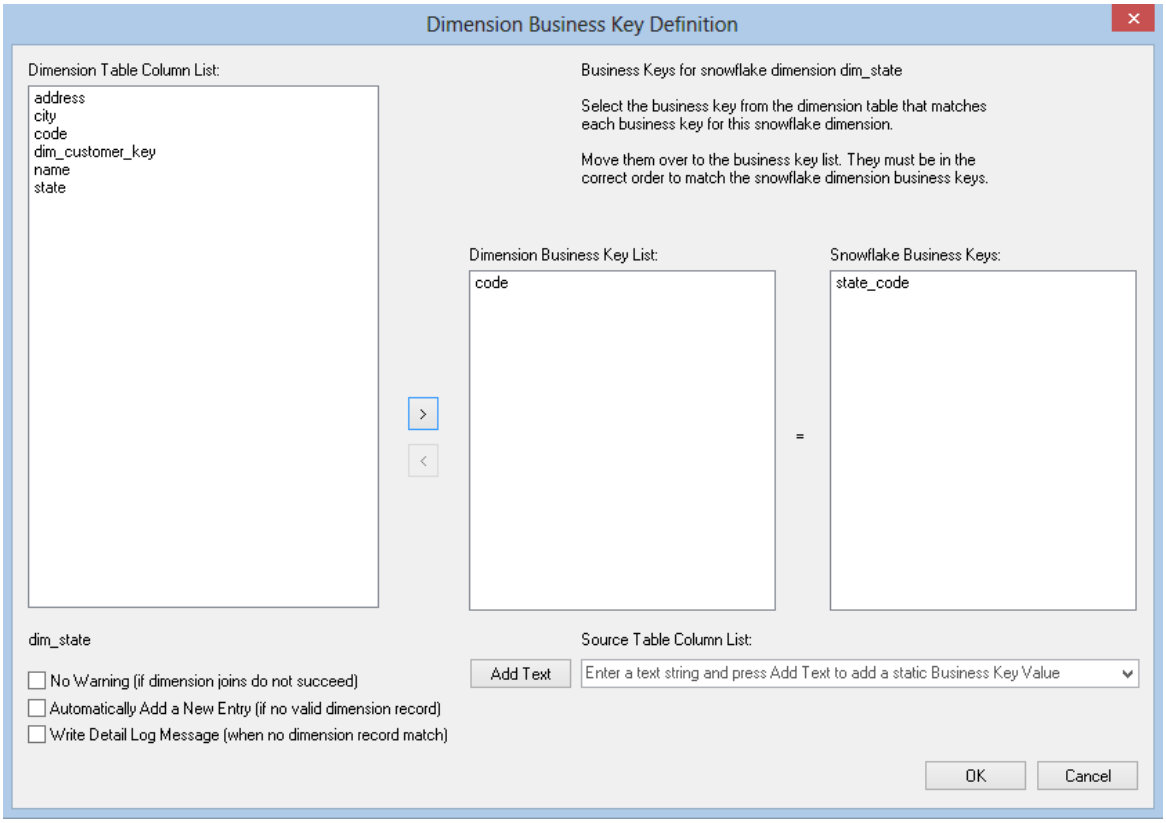
Snowflake schemas normalize dimensions to eliminate redundancy; that is, the dimension data has been grouped into multiple tables instead of one large table.
For example, a product dimension table in a star schema might be EDW 3NF into a products table, a product category table, and a product manufacturer table in a snowflake schema.
While this saves space, it increases the number of dimension tables and requires more foreign key joins.
The result is more complex queries and reduced query performance.

CREATING A SNOWFLAKE

A snowflake dimensional structure is supported by WhereScape RED.
A snowflake can be created for EDW 3NF or partially EDW 3NF dimension tables, by including the surrogate key of the parent dimension in the child dimension.
In the example below, the dim_state table represents the parent dimension.
The column dim_state_key is added to the child dimension dim_customer. Any fact tables that include the dim_customer dimension will inherently have a link to the dim_state dimension.

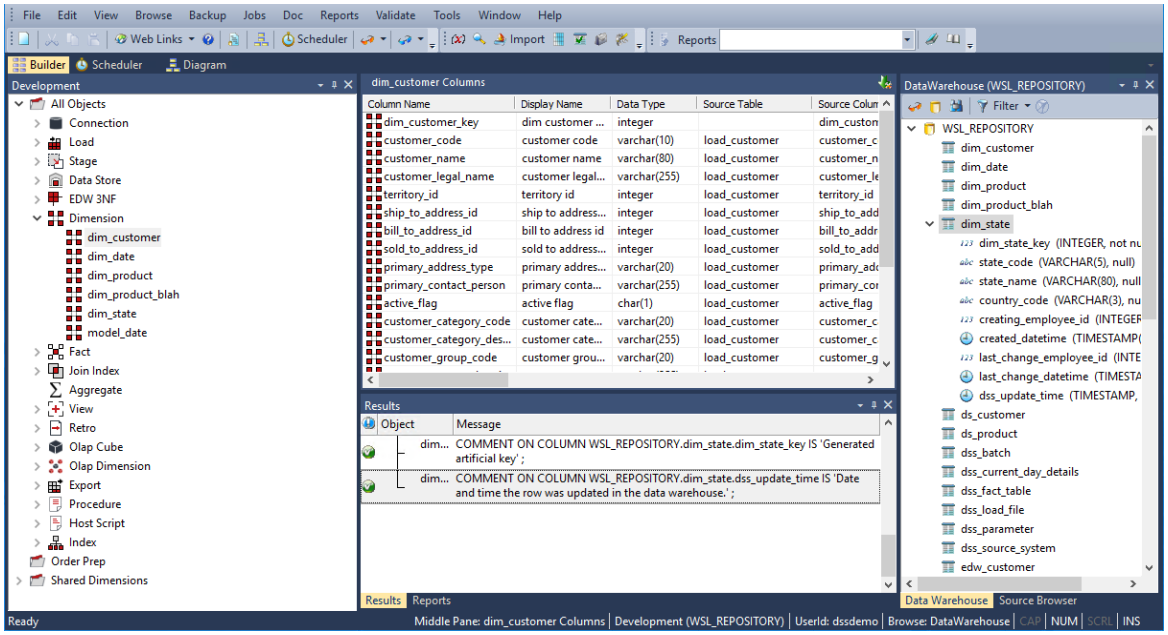
The process for creating a snowflake is as follows:

1. Build both dimensions (see previous sections).
2. Expand **dimensions** in the left pane.
3. Click on the child dimension table in the left pane to display its columns in the middle pane.
4. Browse the data warehouse connection in the right pane.
5. Expand the parent dimension table in the right pane.
6. Drag the surrogate key of the parent dimension table from the right pane to the child dimension's column list in the middle pane.
7. **Create/Recreate** the child dimension.
8. Rebuild the child dimension's update procedure.
9. A dialog will now appear asking for the business key column(s) in the child dimension that matches the business key for the parent dimension:



10 Add the business key column(s) and click **OK**.

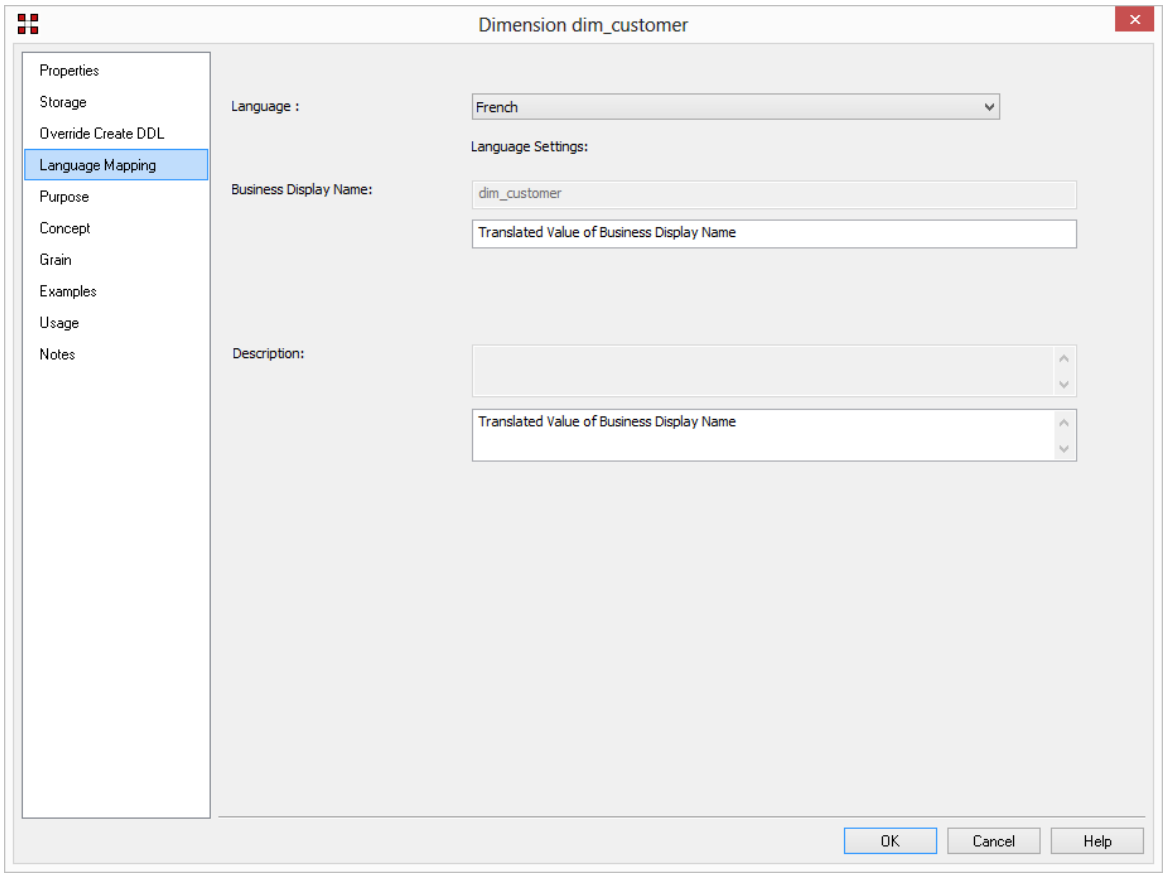
The WhereScape RED screen should look like this:



DIMENSION LANGUAGE MAPPING

The Dimension Properties screen has **Language Mapping** tab.

- Select the language from the drop-down list.
- Enter the translations for the **Business Display Name** and the **Description** in the chosen language.
- The translations for these fields can then be pushed through into OLAP cubes.



To add a language to this list, see ***Settings - Language Options***.

WhereScape®

CHAPTER 10
STAGING

Stage tables are used to transform the data to a star schema or third normal form model. A stage table can be a fact or an EDW 3NF table that only contains change data or a work table. In star schema data warehouses, the stage table brings all the dimensional joins together in preparation for publishing into the fact table.

A stage table is built from the Data Warehouse connection. Unless you are retrofitting an existing system, stage tables are typically built from one or more load or stage tables. They can utilize the surrogate keys from a number of dimension tables.

The normal steps for creating a stage table are defined below and are covered in this chapter. The steps are:

- Identify the source transactional data that will ultimately constitute the fact or EDW 3NF table. If the data is sourced from multiple tables ascertain if a join between the source tables is possible, or if a series of passes will be required to populate the stage table. If the latter option is chosen then bespoke code is needed.
- Using the 'drag and drop' functionality drag the table with the lowest granular data into a stage target. See ***Building the Stage Table*** (on page 368).
- Add columns from other source tables. See ***Building the Stage Table*** (on page 368).
- Add any relevant dimension table or EDW 3NF table keys. See ***Building the Stage Table*** (on page 368).
- Create the stage table in the database. See ***Building the Stage Table*** (on page 368).
- Build the update procedure. See ***Generating the Staging Update Procedure*** (on page 372).
- Test the update procedure and analyze the results. See ***Tuning the Staging Update Process*** (on page 387).
- Modify the update procedure as required. See ***Tuning the Staging Update Process*** (on page 387).

IN THIS CHAPTER

BUILDING THE STAGE TABLE

Building the stage table is potentially the most challenging part of the overall task of building a data warehouse analysis area.
Most of the effort required is in the design phase, in terms of knowing what data needs to come into the fact table that will ultimately be built.
This section assumes that the decision as to what to include has been made.

Multiple data sources

A stage table typically contains the change data for a detail fact table. As such it normally maps to a specific function within the business and in many cases, relates back to one main OLTP table. In many cases however it may be necessary to combine information from a number of tables. One of the decisions required is whether it is practical or even possible to join the data from the different source tables.
In the tutorial, the stage_sales_detail stage table is built by combining information from the order_header table and the order_line table. These two tables are connected via the order_number field which is common to both tables. The tutorial shows the building of a stage table and the joining of these two tables.
We could however also include two additional source tables being invoice_header and invoice_line which contain specific information relating to what was on the invoice. We may want our fact table to contain information from these tables as well. Although these two tables may contain the order_number and hence potentially allow a join with the order tables we may choose not to perform such a join for performance reasons. In this case, we have three obvious choices in terms of how we ultimately update our fact table.

The choices are:

1. Join all four tables using one large join in our staging table.
2. Update the staging table in two passes; one pass updating the order information, the other pass updating the invoice information.
3. Generate two stage tables, one for order and one for invoice. Use these two staging tables to update the one sales_detail fact table.

Although all three options are viable and a normal situation in the WhereScape RED environment, options (2) and (3) will require specific coding and modifications to the generated procedures from the outset.
Given the example provided, option (2) would be the normal approach; although in some cases option (3) would be valid.

Drag and Drop

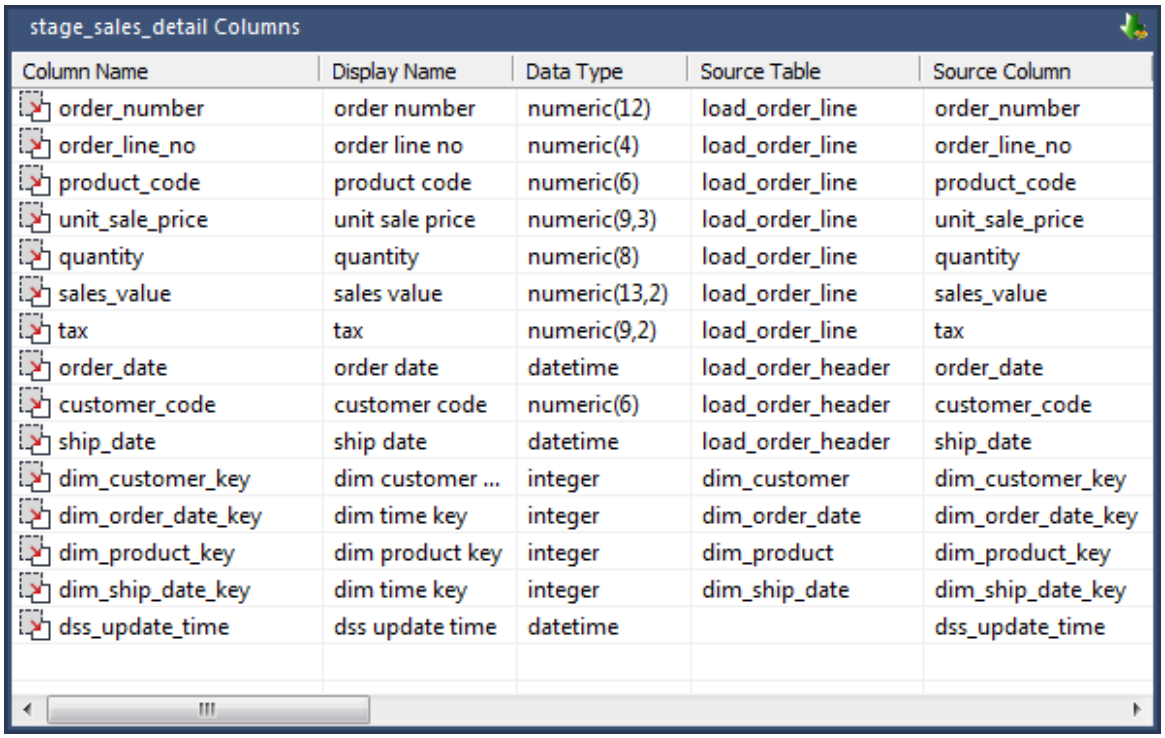
The best approach in creating a stage table is to choose the source table that contains the most fields that we will be using and drag this table into the stage target.
We can then drag specific columns from the other source tables until we have all the source data that is required.

The process for defining the metadata is as follows:

1. Double-click on the **Stage Table** object group in the left pane. This will result in all existing stage tables being displayed in the middle pane. This also sets the middle pane as a **stage drop target**.
2. Browse the Data warehouse connection to display your load tables in the right pane. This is done via the **Browse/Source Tables** menu option, choosing the **Data warehouse connection**.
3. Drag the **primary load table** (i.e. the one with the most columns, or the lowest data granularity) from the right pane and drop it into the middle pane. A dialog will appear to create the new staging object.
4. Leave the object type as 'Stage Table' and change the name to reflect what is being done. For example, in the tutorial the load_order_line table is dropped and a stage table called stage_sales_detail is defined.
5. Once a valid name is entered, the properties for the new stage table are displayed. Normally these would be left unchanged except perhaps for storage settings.
6. Once the Properties dialog is closed the columns for the new stage table are displayed in the middle pane. This middle pane is now considered a *drop target for this specific stage table*. Any additional columns or tables dropped into the middle pane are considered additions to this stage table definition. Any columns that are not required can be deleted.
7. Drag and drop **additional columns** from other source tables if appropriate. In the tutorial, we would now drag the customer_code, order_date and ship_date from the load_order_header table.
8. Drag in the **dimension artificial key** from each dimension that is to be joined to the stage/fact table.
9. You can only join a dimension if a business key exists amongst the stage table columns or if it is possible to derive that business key in some way from the columns or other dimensions.

Note: If a column is being used to join information from two or more source tables, that column must only appear once in the stage table. It is irrelevant which table is used to create the column in the new stage table.

Once completed, your list of columns for the stage table should look something like the list below.
Notice the **Source Table** for each column.



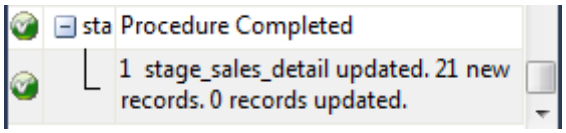
stage_sales_detail Columns

Column Name	Display Name	Data Type	Source Table	Source Column
order_number	order number	numeric(12)	load_order_line	order_number
order_line_no	order line no	numeric(4)	load_order_line	order_line_no
product_code	product code	numeric(6)	load_order_line	product_code
unit_sale_price	unit sale price	numeric(9,3)	load_order_line	unit_sale_price
quantity	quantity	numeric(8)	load_order_line	quantity
sales_value	sales value	numeric(13,2)	load_order_line	sales_value
tax	tax	numeric(9,2)	load_order_line	tax
order_date	order date	datetime	load_order_header	order_date
customer_code	customer code	numeric(6)	load_order_header	customer_code
ship_date	ship date	datetime	load_order_header	ship_date
dim_customer_key	dim customer ...	integer	dim_customer	dim_customer_key
dim_order_date_key	dim time key	integer	dim_order_date	dim_order_date_key
dim_product_key	dim product key	integer	dim_product	dim_product_key
dim_ship_date_key	dim time key	integer	dim_ship_date	dim_ship_date_key
dss_update_time	dss update time	datetime		dss_update_time

The source table (src table) reflects where each column was dragged from.
In the example above, the bulk of the columns came from the load_order_line table, and the customer_code, order_date and ship_date came from the load_order_header columns.
These two load tables will be joined via the order_number column. This order_number column appears in both load tables but is sourced, in this example, from the load_order_line table.
Each dimension artificial key was dragged from its appropriate table. The final column 'dss_update_time' was generated by WhereScape RED and has no source.

Creating the table

- Once the stage table has been defined in the metadata, you need to physically create the table in the database.
- Right-click on the stage table in the left pane and select **Create (ReCreate)** from the pop-up menu.
- A results box will appear to show the results of the creation.
- The following example shows a successful creation.



The contents of this dialog are a message to the effect that the table was created followed by a copy of the actual database create statement, and if defined the results of any index creates. For the initial create no indexes will be defined.

If the table was not created, then ascertain and fix the problem. A common problem is a 'Duplicate column' where a column has been accidentally added twice.

The best way to find such a column is to double-click on the list heading **Column Name**, which will sort the column names into alphabetical order.

Another double-click on the heading will sort the columns back into their create order. Column ordering can be changed by altering the column order value against a column's properties.

TIP: Double-clicking on the heading of a column in a list sorts the list into alphabetical order based on the column chosen.

The next section covers ***Generating the Staging Update Procedure*** (on page 372).

GENERATING THE STAGING UPDATE PROCEDURE

Once a stage table has been defined in the metadata and created in the database an update procedure can be generated to handle the joining of any tables and the lookup of the dimension table artificial keys.

GENERATING A PROCEDURE

- To generate a procedure, right-click on the stage table in the left pane and select **Properties**.
- From the **Update Procedure** drop-down list, select **(Build Procedure...)**.
- Click **OK** to update the properties and start the process of generating the new procedure.
- A series of questions will be asked during the procedure generation to join the tables and link the dimensions.

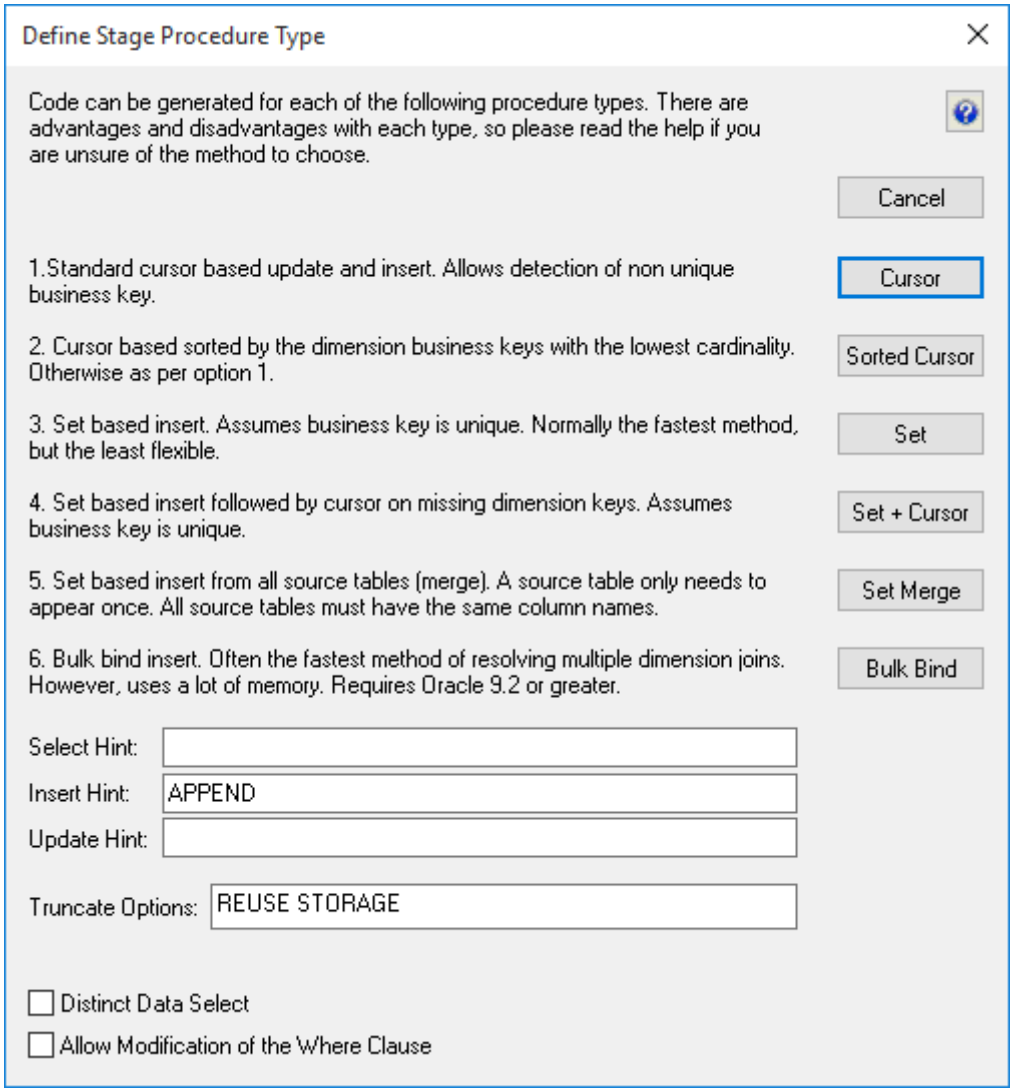
PROCEDURE TYPE

The first dialog box asks for the type of procedure that is to be generated. An example of this follows:

NOTE: Default settings for hint and truncate options are set in **Tools/Options/Code Generation/Default Update Procedure Options**.

ORACLE

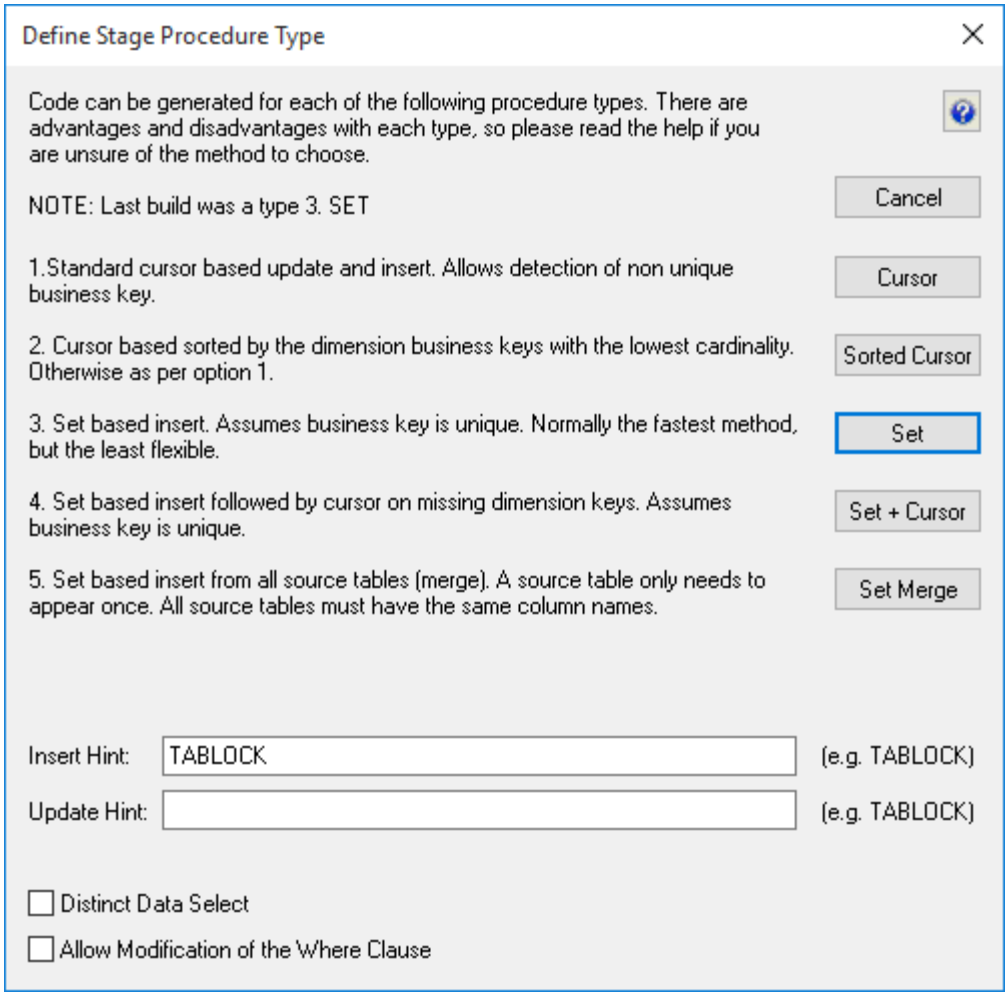
Oracle Only
Select Hint: Enter a database hint to be used in the SELECT statement.
Insert Hint: Enter a database hint to be used in the INSERT statement. Factory default value is APPEND.
Update Hint: Enter a database hint to be used in the UPDATE statement.
Truncate Options: Optional Oracle truncate options such as **REUSE STORAGE** (default value) or **DROP STORAGE**.



SQL SERVER

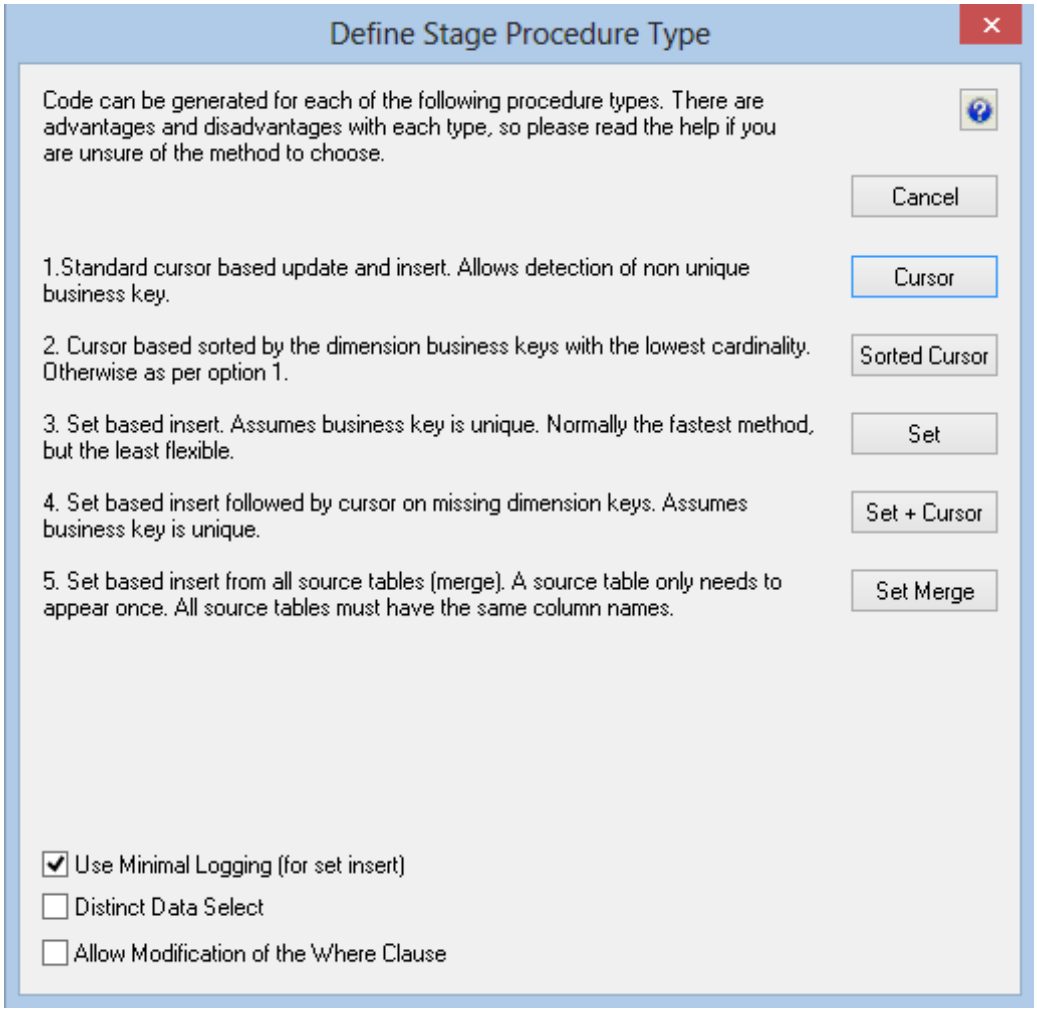
SQL Server Only

Insert Hint: Enter a database hint to be used in the INSERT statement. Factory default value is TABLOCK.
Update Hint: Enter a database hint to be used in the UPDATE statement.



DB2

DB2 Only
Use Minimal Logging (for set insert): The staging procedure uses the **LOAD FROM CURSOR** option in DB2 to minimize database logging.



A number of different types of procedure generation are available. Each type is discussed below.
There is a check-box at the bottom of the dialog box. This check-box will only be present if advanced procedure building features are enabled.
This check-box allows the editing of the 'Where' clause when no table joining is being performed, and hence the 'Where' clause would not be exposed.

Cursor based procedure

This option creates a procedure that loops through each source transaction performing specific tasks on each transaction before writing the transaction to the stage table. The tasks performed include the lookup of each dimension key, and any bespoke value assignments or adjustments.

Sorted cursor procedure

This is a variant on the regular cursor procedure. In this case, dimension key lookups only occur when there is a change in the business key for a given dimension. This can improve the performance of the procedure particularly when one or more of the dimensions has a low cardinality. Where slowly changing dimensions are used, this advantage is lessened as the relative date of the dimension records also comes into play.

Set based procedure

A set based procedure performs one SQL statement to join all the dimensions and source tables together and then insert this data into the stage table. This is normally the fastest method of building the stage table. It is also the least flexible method and is easily prone to incorrect syntax. Care and experience must be used with this option to ensure that all source transactions are processed and that they are processed correctly. This option is only recommended for the experienced practitioner where performance is the overriding requirement. Caution must be taken with regards to **NULL** values in the business keys that are used to make the table joins. The generated code deliberately does not handle such null values.

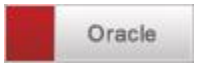
Set + Cursor Procedure

This option is normally used where a number of dimension keys will fail to join and we wish to automatically add new dimension records. The first phase of the generated code will perform a set based join and insert of the data. The dimensions that failed to join will be assigned the zero key value. The second phase of the procedure will use a cursor to cycle through all the stage records that have a zero key dimension join. By setting the appropriate flags in the dimension joins when building this procedure, we can request that any missing dimension records be automatically added.

Set Merge Procedure

This option is to allow the merging of two or more identical tables. The tables to be merged must have exactly the same number of columns and column names. If necessary additional blank columns could be added to one or other of the tables to make them identical. To use this procedure, you must simply have the tables to be merged mentioned at least once in the **Source Table** field of a columns properties. For more details, see ***Set Merge Procedure*** (on page 394).

Bulk Bind Procedure



The bulk bind procedure is only available for Oracle. It uses memory arrays to join all the information before writing it to the table. It can be the fastest method of performing a stage table update when a lot of dimension joins are being used.

There are some other additional options available in this dialog common to SQL Server, Oracle and DB2:

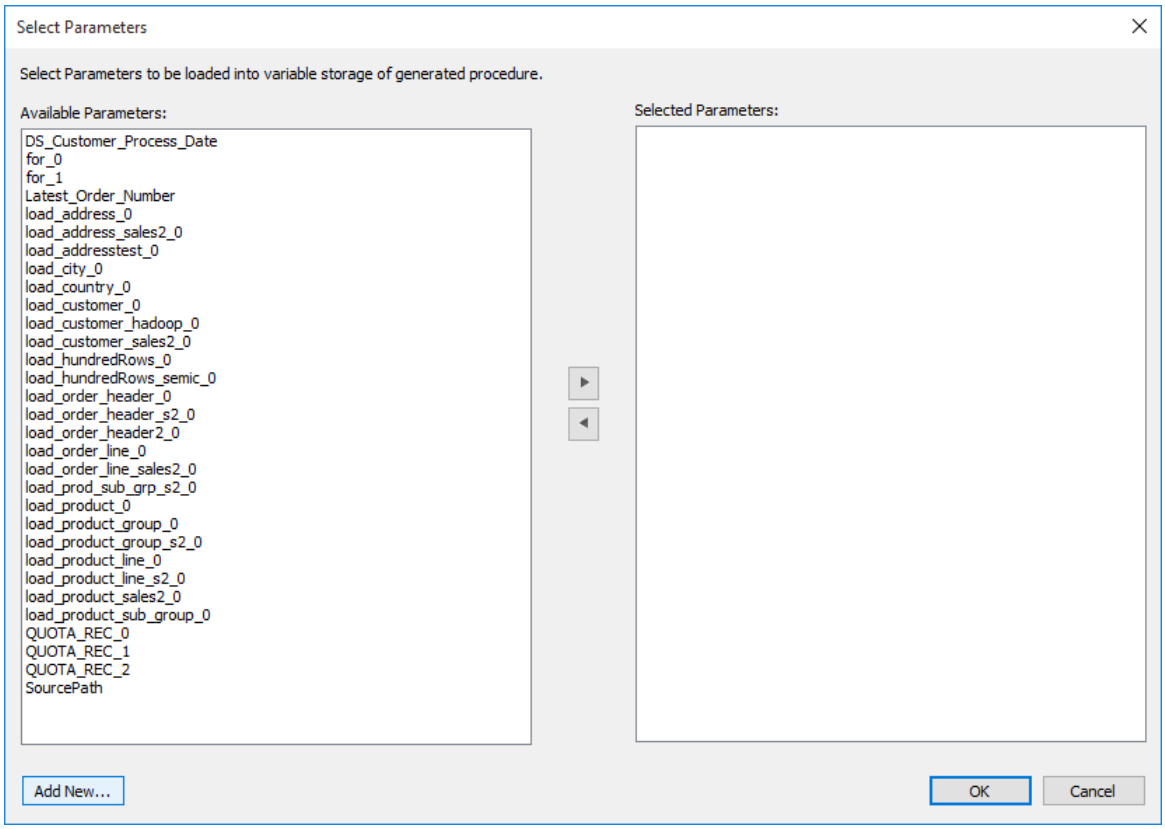
Allow modification of the Where clause: the 'Where' clause dialog is displayed automatically if the table has more than one source table, but not for a single source table stage table unless this option is chosen.

Distinct data select: choosing this option adds the DISTINCT key word to the SELECT statement in the staging procedure.

Parameters

If WhereScape RED parameters exist in the metadata, the following dialog is displayed. Any parameters selected in this dialog (by moving them to the right pane), are included in the generated update procedure as variables.

The procedure will include code to retrieve the value of the parameter at run time and store it in the declared variable.



The variables can also be used in column transformations and in the from/where clause for the update procedure.
Some databases have a 30 character limit for variable names. WhereScape RED ensures the variables added for any parameters are less than 30 characters long by creating variable names in the form v_ followed by the first 28 characters of the parameter name.
For example, a parameter called MINIMUM_ORDER_NUMBER_SINCE_LAST_SOURCE_LOAD will be available as the variable v_MINIMUM_ORDER_NUMBER_SINCE_L.

TIP1: WhereScape RED parameters should be unique within the first 28 characters to avoid conflicting variables names.

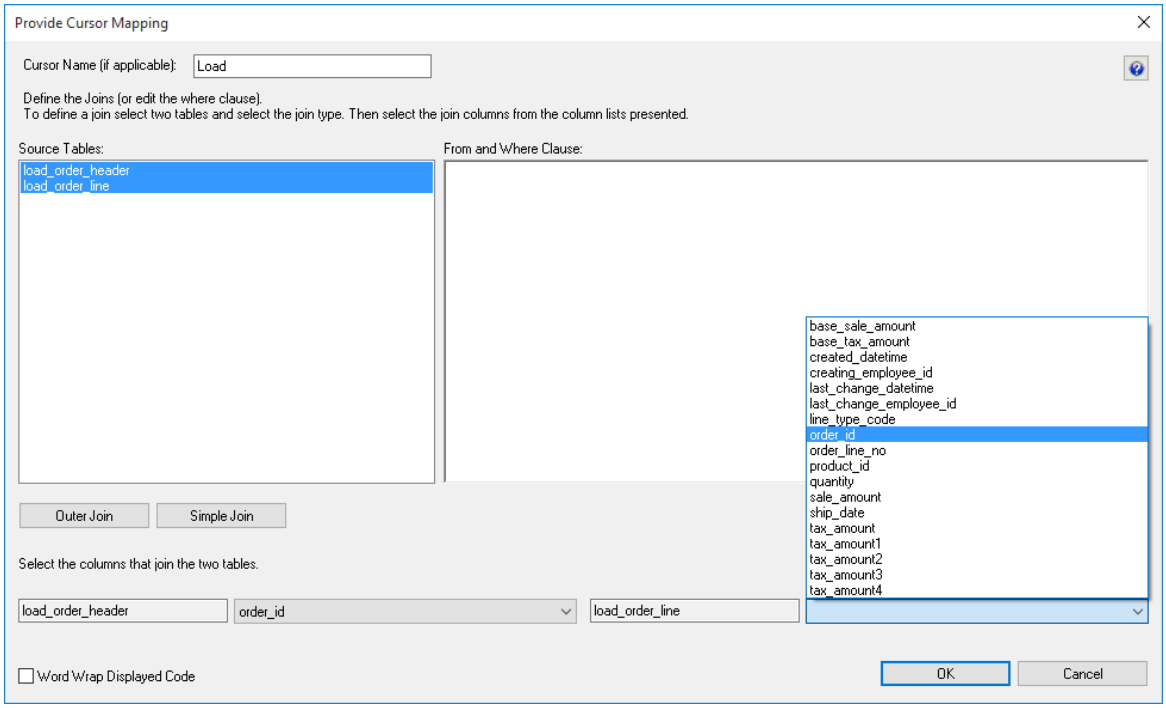
TIP2: If the desired parameter doesn't exist in the metadata yet, a new parameter can be added by clicking on the **Add New** button on the bottom leftmost corner of the Select Parameters dialog.

See ***Parameters*** (on page 139) for more information on WhereScape RED Parameters.

MULTIPLE SOURCE TABLES

If multiple tables are being joined and we did not choose a **Set** based procedure, then an initial dialog will display (as follows) advising that multiple source tables have been detected and that WhereScape RED will only generate one cursor (one join) to handle these tables.
If multiple cursors or passes of the data are required, as in scenario (2) in the previous section then it is best to proceed with one set of joins and manually do the second set after the generation of the procedure has completed.
When multiple source tables are detected a dialog will appear to allow the definition of the joins between the various tables.

NOTE: Joins to dimension tables are not defined in this step. These joins only relate to the source tables.



The joining of the tables will provide either the data for a cursor loop, or part of the construct of the set based update in the update procedure.
If a cursor based procedure is being generated, then the default name for the cursor will be 'Load' unless altered.
If a set based procedure, then the cursor name is ignored.
Only two tables may be joined at a time. To join two tables, select the two tables in the left box and click either the outer join or simple join button. Column lists for both tables will appear at the bottom of the dialog box.
Select the column (or one of the columns) that allows the two tables to be joined. If an outer join is being used, the column for the master table must be chosen first.
If there are multiple columns joining two tables then this action must be repeated for each column. Continue to perform all joins between all tables.
The example below only has two tables with one join column so is a relatively simple case.

For **SQL Server** an additional option is available to allow either an ANSI standard join or a 'Where' clause based join.
In **DB2**, only a from clause join can be generated. The ANSI standard join should be chosen in most situations.
See the example screen in the following section.

Simple Join

A simple join joins the two tables, and only returns rows where data is matched in both tables. So for example if table A has 100 rows and table B has a subset of 24 rows. If all the rows in table B can be joined to table A then 24 rows will be returned. The other 76 rows from table A will not be returned.

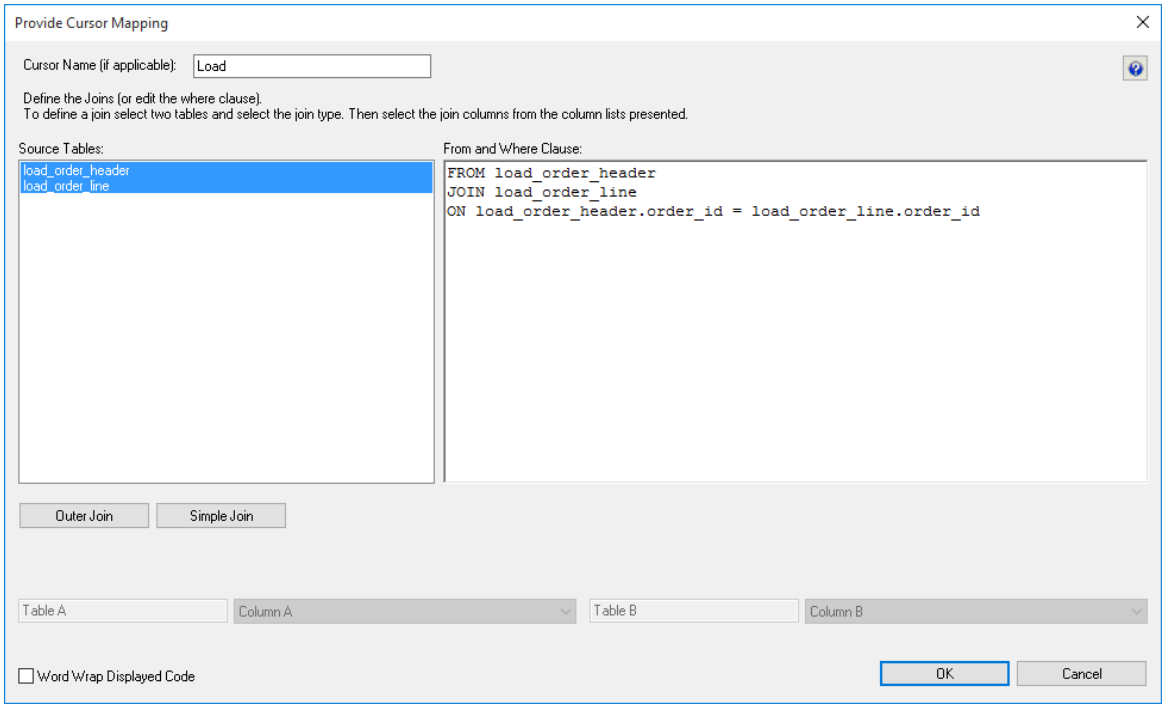
Outer Join

An outer join joins the two tables, and returns all rows in the master table regardless of whether they are found in the second table. So if the example above was executed with table A as the master table then 100 rows would be returned. 76 of those rows would have null values for the table B columns. When WhereScape RED builds up the 'Where' clause for Oracle it places the outer join indicator (+) at the end of each line. As this indicator goes on the subordinate table, it must be that table whose column is selected last. In the example screen above the table 'load_order_line' has had its column chosen and the column for the table 'load_order_header' is currently being chosen. This will result in the statement as shown in the 'Where' clause edit window. The results of this select are that a row will be added containing order_line information regardless of whether an order_header exists.
As the join columns are selected the 'Where' statement is built up in the large edit window on the right. Once all joins have been made the contents of this window can be changed if the join statement is not correct. For example, the Oracle outer join indicator (+) could be removed if required.
Once satisfied with the 'Where' statement click **OK** to proceed to the next step. As indicated in its description this statement is the 'Where' clause that will be applied to the select statement of the cursor to allow the joining of the various source tables. If it is not correct, it can of course be edited in the generated procedure.
For **SQL Server** data warehouses you have the choice between 'Where' statement joins and ANSI standard joins.
For **DB2** data warehouses, only ANSI standard joins are available.

NOTE: 'Where' joins will not work correctly in SQL Server if more than two tables are used in an outer join.

NOTE: When upgrading from a RED version before 6.8.2.0 and moving existing objects to a target location, all procedures that reference those objects will need to be rebuilt.
Any **FROM** clauses will also need to be manually regenerated for the table references to be updated to the new [TABLEOWNER] form.

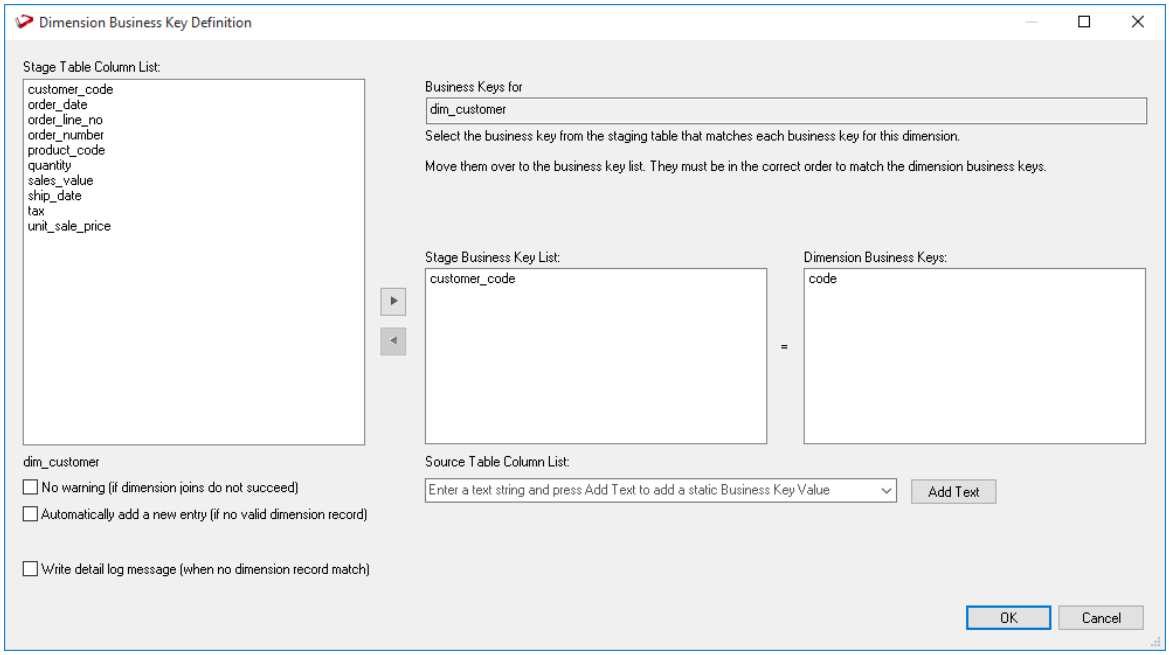
The example below shows the result of an ANSI standard join which takes place in the 'From' statement.



DIMENSION JOINS

For each dimension key a dialog will now appear asking for the business key from the stage table that matches the business key for the dimension.
In the example below we are asked for the stage table business key for the customer dimension. The dimension name is shown both on the first prompt line and at the lower left side of the dialog box.
The customer dimension has a unique business key named **code**. We must provide the corresponding business key from the staging table. In the case of our example, this is the customer_code column.

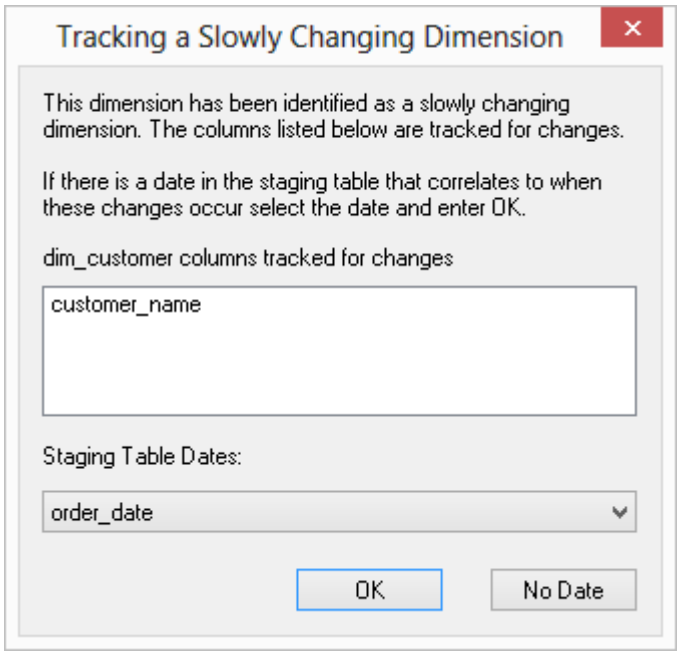
- Click **OK** after the correct business key has been entered.
- If the business key does not exist and will be derived from another dimension or from some form of lookup, then enter any column and edit the procedure once produced.



NOTE: The **Add Text** button and the associated message and edit box are only shown if the user possesses a full license, thus enabling the advanced procedure build options. When the **Add Text** button is clicked any data in the edit box to the right of the button is placed in the stage table column list. In this way, a number or string can be assigned as part or all of a dimension join.

SLOWLY CHANGING DIMENSION INFORMATION

If the dimension being joined was defined as a slowly changing dimension, then an additional dialog will appear, shown below. This dialog asks for a date field in the stage table that enables RED to determine which version of the slowly changing dimension (the **customer_name** field, below) to use based on the specified date range in the Customer Dimension.
Select the appropriate date field for your business needs and click **OK**. If you wish to take the last (or current) version for the dimension, select **No Date**.

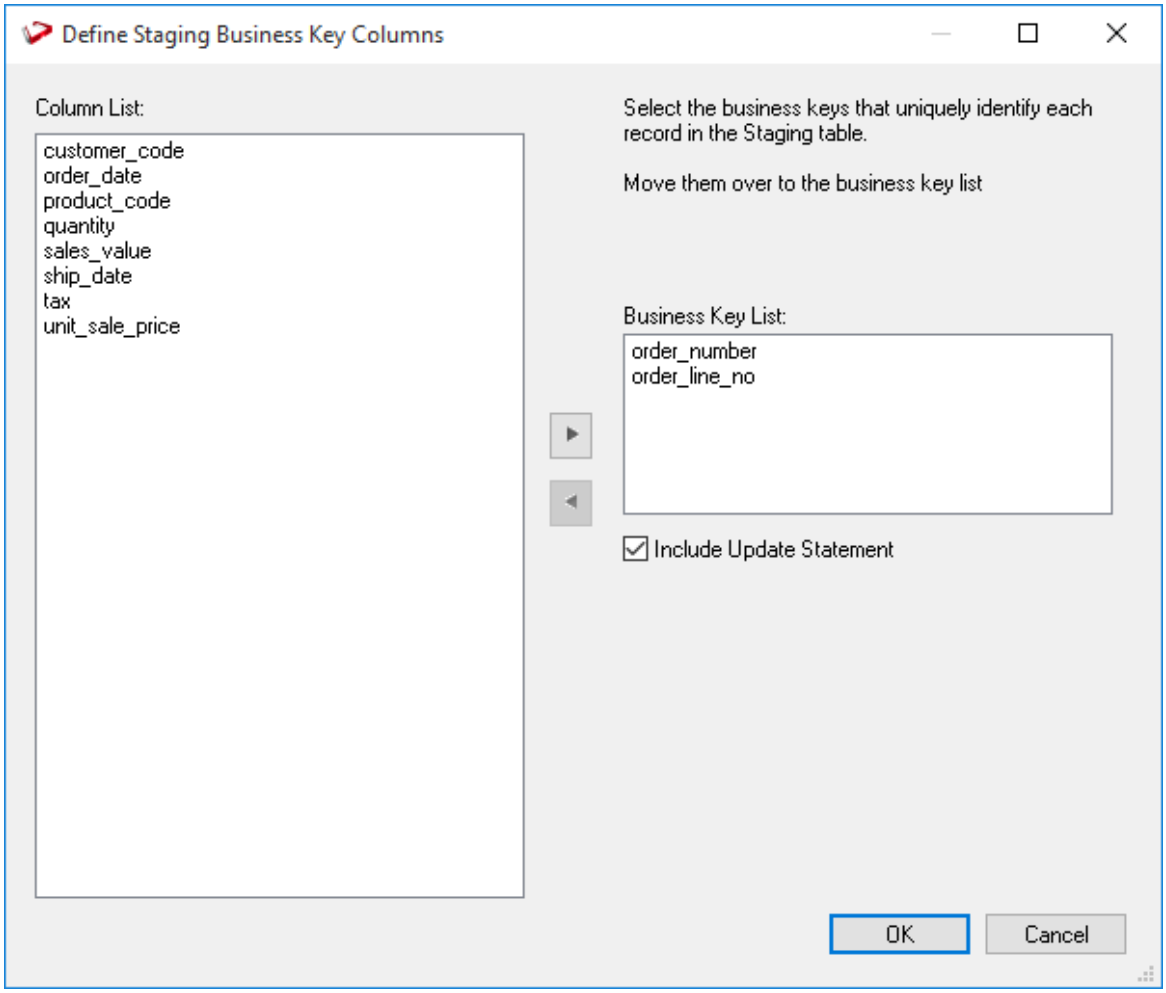


For Example:

As seen in the screenshot above, we have defined the **customer_name** as an item in the dimension that we expect to have versions for over time i.e. each time the data warehouse processing sees a new **customer_name** value, the dimension will record the date range for that version's validity - even though the business key (customer_code in this example) remains the same. This implies we want to create a new dimensional record whenever a customer name is changed even though the customer_code remains the same.
Let's say a customer changes their name on the 5th of the month. If the **Staging Table Dates** field is set to **order_date**, any order received before the 5th of the month is identified under the old customer name and any order received on or after the 5th has the new customer name. Alternatively, by setting the **Staging Table Dates** to **ship_date**, we can specify that any order *shipped* on or after the 5th of the month is shipped with the new name.

STAGING BUSINESS KEY

Once all the dimensional joins have been defined we will be asked to define the unique business key for the staging table. This is the column or columns that allow us to uniquely identify each record in the staging table.
In the example below and in our tutorial the unique business key is a combination of the order_number and the order_line_no.



NOTE1: The order of the columns in the business key list is set to the order that the columns appear in the table. This is done to optimize performance for any indexes.

NOTE2: **NULL VALUES:** None of the columns chosen as the business key should ever contain a NULL value. See the note at the start of the Dimension chapter.

The **Include update statement** check-box provides two options in the generated procedure. If checked then an update/insert combination will be included. If clear, then only an Insert statement will be included in the procedure. If cleared you must be sure that the business key for the table is unique, otherwise either a duplicate key error will occur or records will be lost depending on the type of procedure.

INDEX RE-CREATION

Finally, we're asked if we want to drop and re-create any indexes defined on the table:

- Click **Yes** to drop and re-create.
- **No** to leave existing indexes in place.

BUILDING AND COMPILING THE PROCEDURE

- Once the above questions are completed the procedure is built and compiled automatically.
- If the compile fails, an error will be displayed along with the first few lines of error messages.
- Compilation failures typically occur when the physical creation of the table was not done.
- If the compile fails for some other reason the best approach is to use the procedure editor to edit and compile the procedure.
- The procedure editor will highlight all the errors within the context of the procedure.
- Once the procedure has been successfully compiled it can either be executed interactively or passed to the scheduler.

TUNING THE STAGING UPDATE PROCESS

This section is for **Cursor** based procedures only. It is not relevant for Set based or other types. When generating the cursor based procedures to update the staging table WhereScape RED includes Update/Insert logic. This logic first attempts to update an existing staging record and then inserts a new record if the record does not exist.
This build option should be altered once an understanding of the data you are dealing with is acquired.
For the initial prototype and testing phases of a data warehouse analysis area we need to find out if our business keys for the fact table are genuinely unique.The update/insert logic in the staging procedure will only perform an 'Update' if our business key is not unique.
In this case, we need to decide if this is a failure in the business key, or some form of additive record situation we need to handle and code accordingly.

Removing the Update Component

Once we have resolved any data issues we would normally remove the update component to speed up the processing of the stage data. There are two ways of doing this. First, we can clear the **Include Update Statement** check-box when defining the table business key. This will result in a procedure with only an insert statement. The other option is to do it manually as follows.

There are normally three steps to removing the update component. These being:

1. The update makes use of an index that can also be deleted in most situations.
2. The actual **Update** statement can be removed from the code such that the insert is always executed.
3. The procedure also includes an analyze of the table after 1000 rows have been inserted. This code is present to ensure that the index is used in the update. We can now remove this code. It would normally look like the following (Oracle version):

```
--=======================================
-- If 1000 or more rows inserted then perform an analyze
--=======================================
IF v_insert_count >= 1000 And v_analyze_flag = 0 THEN
      v_analyze_flag := 1;
      v_Sql := 'analyze table stage estimate statistics sample 5 percent for all indexed columns';
      execute immediate v_Sql;
END IF;
```

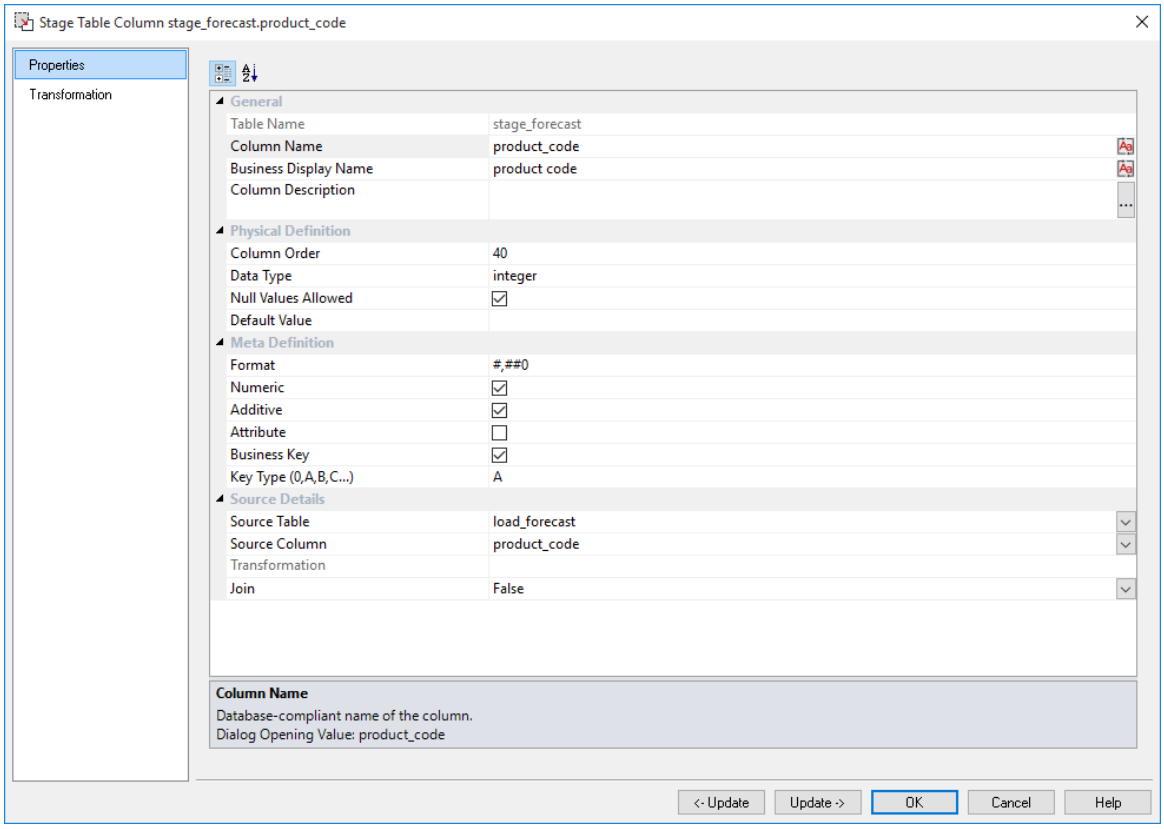
STAGE TABLE COLUMN PROPERTIES

Each stage table column has a set of associated properties. The definition of each property is defined below:

If the **Column name** or **Data type** is changed for a column, then the metadata will differ from the table as recorded in the database.
Use the **Validate/Validate Table Create Status** menu option to compare the metadata to the table in the database.
A right mouse menu option of **Alter Table** is available when positioned on the table name after the validate has completed. This option will alter the database table to match the metadata definition.

TIP: If a database table's definition is changed in the metadata then the table will need to be altered in the database.
Use the **Validate/Validate Table Create Status** to compare metadata definitions to physical database tables. The option also provides the ability to alter the database table, through a pop-up menu option from the validated table name.

A sample **Properties** screen is as follows:



TIP: The two special update keys allow you to update the column and step either forward or backward to the next column's properties.
ALT-Left Arrow and **ALT-Right Arrow** can also be used instead of the two special update keys.

Table Name

Database-compliant name of the table that contains the column. [Read-only].

Column Name

Database-compliant name of the column. Typically, column-naming standards exclude the use of spaces etc. A good practice is to only use alphanumerics, and the underscore character. Changing this field alters the table's definition.

Note: A case conversion button on the right converts the text between different cases: UPPER CASE, Capitalized Case and lower case. The mode cycles to the next case in the sequence each conversion.

Business Display Name

Name that the business uses to refer to the column, which is included in the RED-generated documentation and can be used in the end user layer of other tools. [Does NOT affect the physical database table]. As such it is a free form entry and any characters are valid.

Note: A case conversion button on the right converts the text between different cases: UPPER CASE, Capitalized Case and lower case. The mode cycles to the next case in the sequence each conversion.

Column Description

This field contains the description for the column. It may be a description from a business user's point of view. This field might additionally contain information on where and how the column was acquired. For example, if the column is sourced from multiple tables or is a composite or derived column then this definition would normally describe the process used to populate the column. This field is used in the documentation and is available via the view **ws_admin_v_dim_col .** This field is also stored as a comment against the column in the database.

Column Order

Numeric value that controls the relative order of columns in the database create statement. The lowest numbered column will appear first in the table. Although this affects the physical table definition no action will be taken unless the table is re-created in the database. The columns can be re-numbered based on the existing order by choosing the **Respace Order Number** pop-up menu option when positioned over any column in the table. This action will number the columns in increments of 10 starting at 10. In addition to a simple change of the order field, the column order can be changed by first displaying the columns in the middle pane and then using drag and drop to move the columns around. This drag and drop process will automatically renumber the columns as required.

Data Type

Database-compliant data type that must be valid for the target database. Typical data types for Oracle are integer, number, char(), varchar2() and date. For SQL Server common types are integer, numeric, varchar() and datetime. See the database documentation for a description of the data types available. Changing this field alters the table's definition.

Null Values Allowed

Determines whether the table column can hold NULL values or whether a value is always mandatory.

Default Value

Initial value that is assigned to the column when a row is inserted into the table but no value is specified for the column.

Format

Optional format mask that can be used in end user tools. [Does NOT affect the physical database table]. As such it is a free form entry and any characters are valid. Typically format masks are only used on numeric fields. Example: #,###0.00. It is not worth the effort of populating this field unless it can be utilized by the end user tools in use.

Numeric

Indicates whether the table column holds values that are numeric. This is normally only relevant for fact tables. It does not affect the physical table definition, but rather provides input to the view **ws_admin_v_dim_col** which can be used to assist in the population of an end user tool's end user layer. The use of this field is not relevant unless it can be utilized by the end user tools.

Additive

Indicates whether the table column holds values that are additive. This implies that the column can be summed when performing data grouping in a query. This is normally only relevant for fact tables. It does not affect the physical table definition, but rather provides input to the view **ws_admin_v_dim_col** which can be used to assist in the population of an end user tool's end user layer. The use of this field is not relevant unless it can be utilized by the end user tools.

Attribute

Indicates whether the table column holds values that are descriptive, and/or are used for grouping/summing. An attribute is defined as a column that is non factual. For example, we may have an order number, or a invoice number stored in the fact table. Such columns are considered attributes, rather than facts. This checkbox is therefore normally only relevant for fact tables. This checkbox does not affect the physical table definition, but rather provides input to the view **ws_admin_v_dim_col** which can be used to assist in the population of an end user tools end user layer. The use of this field is not relevant unless it can be utilized by the end user tools.

Business Key

Indicates whether the column is part of the BUSINESS key, which is defined during the update procedure build. [Normally maintained automatically]. Multiple columns can form the primary business key.

Key Type

Key type that is assigned and used when generating the table's update procedure and indexes. [Normally maintained automatically]. It can be altered here, but this should only be done if the consequences are fully understood and tested.
The supported values are:

Key type	Meaning
0	The artificial key. Set when the key is added during drag and drop table generation.
1	Component of all business keys. Indicates that this column is used as part of any business key. For example: By default, the dss_source_system_key is added to every dimension table. It is considered part of any lookup on that table and has the key type set to 1. Set when the column is added during drag and drop table generation.
2	Indicates that this column is a dimensional join. Used on fact tables to indicate the dimension keys. Results in bitmap indexes being built for the columns. Set during the update procedure generation for a fact table, based on information from the staging table.
3	Slowly changing column indicator. Used on dimension tables to indicate that the column is being managed as a slowly changing column within the context of a slowly changing dimension. Set when a column is identified during the dimension update procedure generation.
4	Previous value column indicator. Used on dimension tables to indicate that the column is being managed as a previous value column. The source column identifies the parent column. Set during the dimension creation.
A	Indicates that the column is part of the primary business key. Set whenever a business key is defined as part of an update procedure generation.
B-Z	Indicates that the column is part of a secondary business key. Only used during index generation and not normally set.

Source Table

Identifies the source table where the column's data comes from. This source table is normally a load table, or a dimension table within the data warehouse. If the column was sourced from multiple tables, then the normal practice is to record one of the tables in this field and a comment listing all of the other tables in the **Source strategy** field. This field is used when generating a procedure to update the stage table. It is also used in the track back diagrams and in the documentation.

Source Column

Identifies the source column where the column's data comes from. Such a column is normally a load table column, which in turn may have been a transformation or the combination of multiple columns. This may also be a dimensional key where a dimension is being joined.

Transformation

Transformation. [Read-only].

Join

Indicates whether the table column is used in a table join. [Normally maintained automatically but can be optionally changed to override the default join logic used in the generated update procedure]. The **Source table** and **Source column** fields will provide the dimension table's side of the join. The options for this field are: False, True, Manual and Pre Join.

Setting this field to Manual changes the way the dimension table is looked up during the staging table update procedure build. It allows you to join the dimension manually in the Cursor mapping dialog (where the 'Where' clause is built). The usual dialog for matching the dimension business key to a column or columns in the staging table is not displayed if this option is enabled.

Setting this field to Pre Join activates the **Pre Join Source Table** field and allows you to select a table from the drop-down list.

Pre Join Source Table

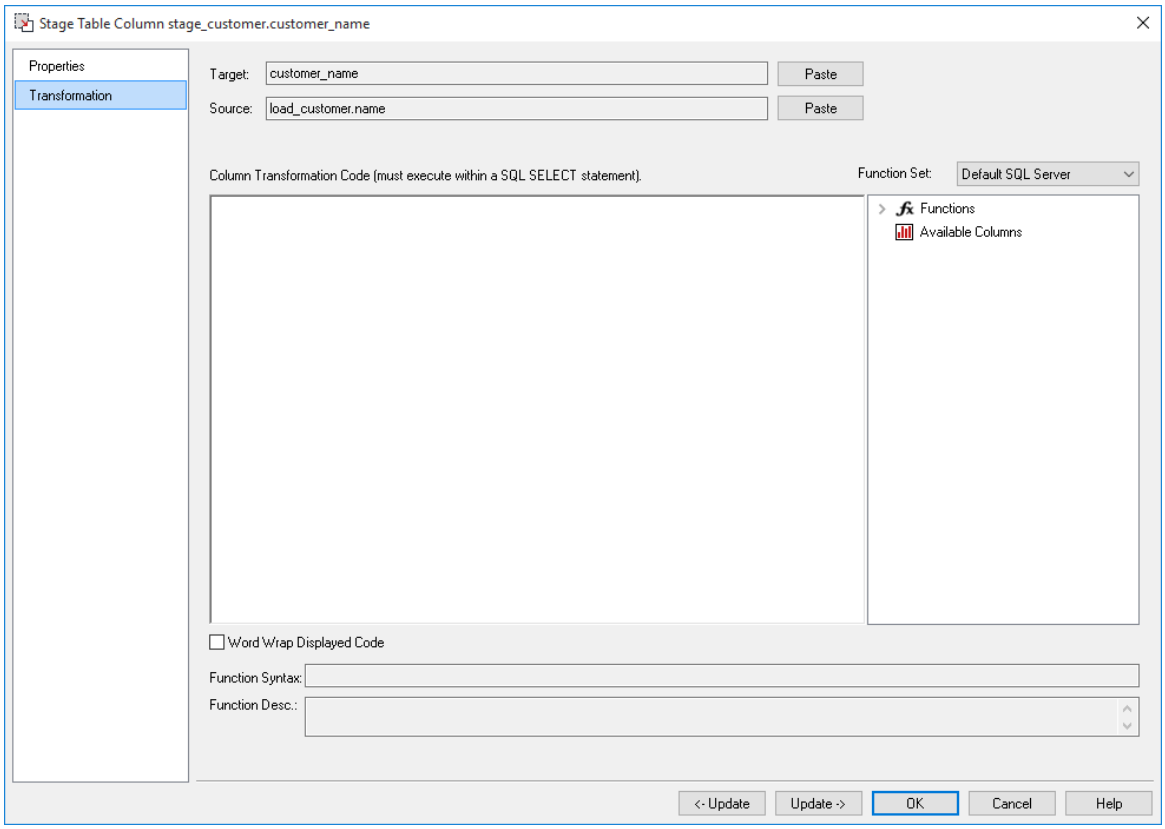
Indicates the table from which the pre joined column was sourced. When the Join option is set to False, this field becomes inactive. When the Join option is set to True or Manual, this field is set to the current table name. When the Join option is set to Pre Join, then you can select the required table from the drop-down list.

STAGE TABLE COLUMN TRANSFORMATIONS

Each stage table column can have a transformation associated with it. The transformation will be included in the generated procedure and will be executed as part of the procedure update.
The transformation must therefore be a valid SQL construct that can be included in a **Select** statement.
For example, we could have a transformation of 'load_order_line.qty * 0.125' to calculate a tax column of 12.5%. Click the **Transformation** tab to enter a transformation.

It is recommended that transformations are not performed on columns that are dimension keys or the business keys for the table.

The transformation screen is as follows:



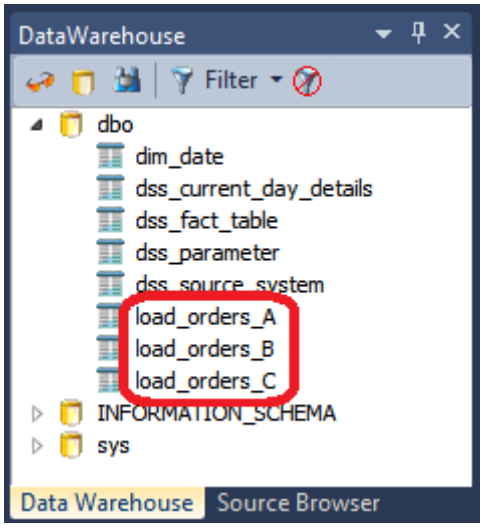
NOTE: Transformations are only put into effect when the procedure is re-generated.

See ***Transformations*** (on page 650) for more details.

SET MERGE PROCEDURE

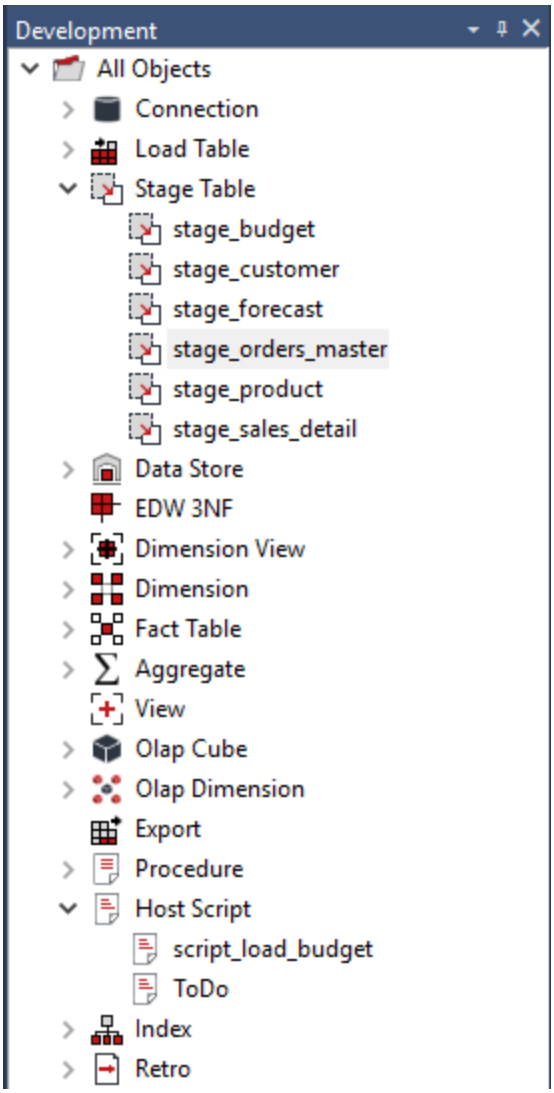
The set merge procedure type allows the merging of two or more identical tables. The tables to be merged must have exactly the same number of columns and column names.
If necessary additional blank columns could be added to one or other of the tables to make them identical.
To use this procedure, you must simply have the tables to be merged mentioned at least once in the **Source Table** field of a columns properties.
In this example, we will merge three load tables into a single stage table.

1 The browser pane shows the three load tables:



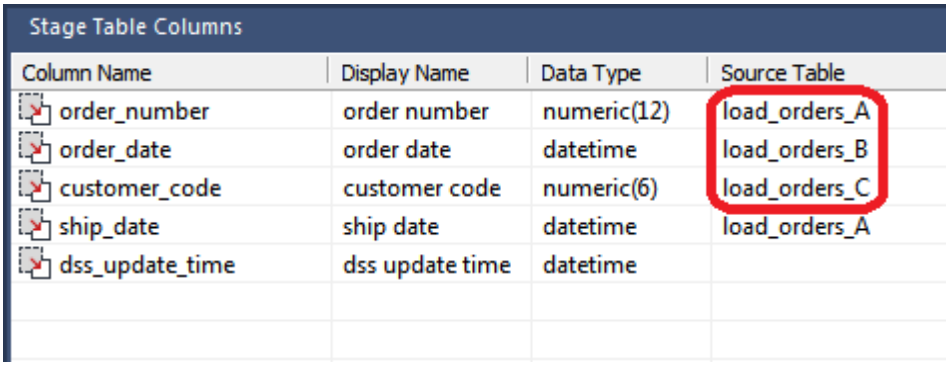
2 Double-click on the stage table object group and then drag one of these load tables from the source pane, into the Stage Object work area.

- Name the stage table, for example **stage_orders_Master**.

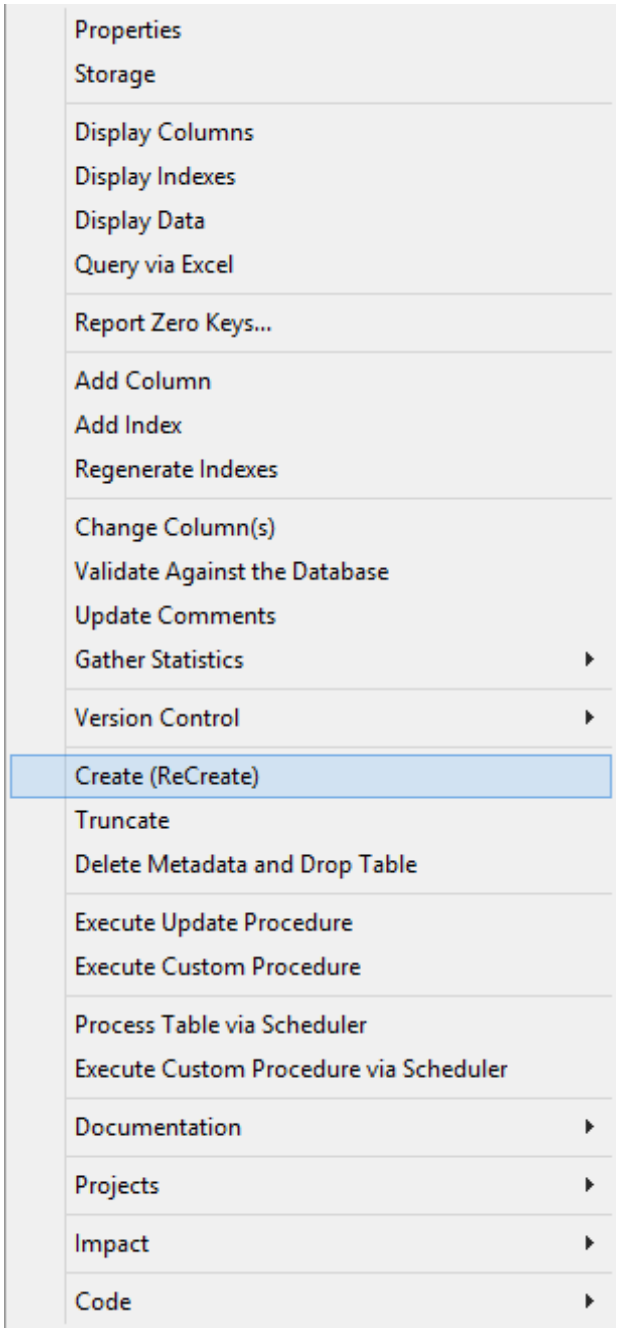


3 Next, modify the source table column to include one instance of each of the three load tables; the order does not matter.

- You can do this either by typing in the table directly, or by going to each of the column's Properties.

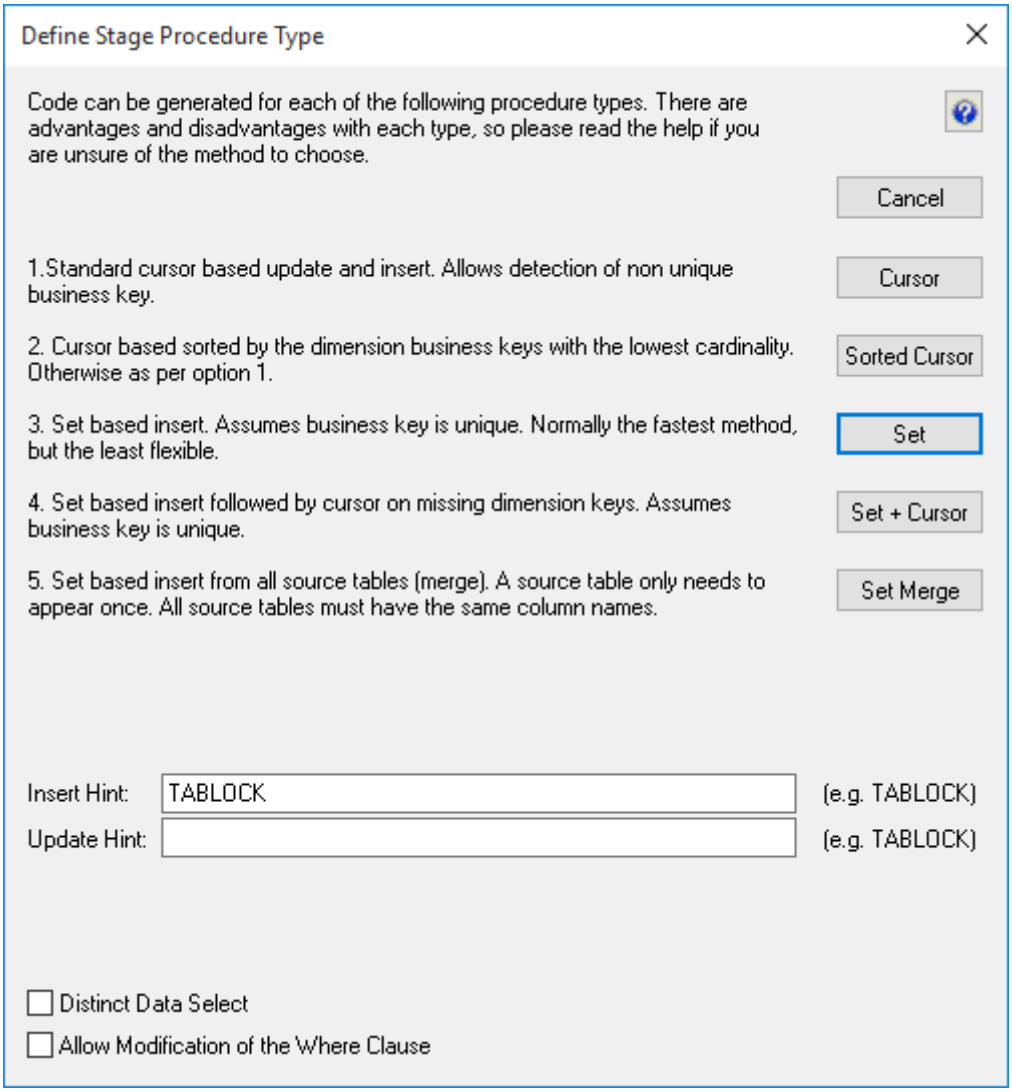


4 Right-click on the stage table and select **Create (ReCreate)**.



5 Double-click on the stage table in the left pane to bring up the **Properties** dialog.

- Click the **Rebuild** button to rebuild the stored procedure.
- Select **Set Merge** as the procedure type.



6 The stored procedure is rebuilt, as follows:

```
-- ================================================================
-- DBMS Name       :    SQL Server
-- Script Name     :    update_stage_orders_Master
-- Description     :    Build the staging table stage_orders_Master
-- Generated by    :    Version 6.5.5.2 (build 111221)
-- Generated for   :    Testing, Tutorials and Documentation
-- Generated on    :    Tuesday, February 07, 2012 at 20:43:10
-- Author          :    QuickStart
-- ================================================================
-- Notes / History
--
CREATE PROCEDURE update_stage_orders_Master
  @p_sequence          integer
, @p_job_name          varchar(256)
, @p_task_name         varchar(256)
, @p_job_id            integer
, @p_task_id           integer
, @p_return_msg        varchar(256) OUTPUT
, @p_status            integer      OUTPUT
AS
  SET XACT_ABORT OFF  -- Turn off auto abort on errors
  SET NOCOUNT ON      -- Turn off row count messages

  -----------------------------------------------------
  -- Control variables used in most programs
  -----------------------------------------------------
  DECLARE
    @v_msgtext         varchar(255)  -- Text for audit_trail
  , @v_sql             nvarchar(255) -- Text for SQL statements
  , @v_set             integer       -- commit set
  , @v_analyze_flag    integer       -- analyze flag
  , @v_step            integer       -- return code
  , @v_update_count    integer       -- no of records updated
  , @v_insert_count    integer       -- no of records inserted
  , @v_count           integer       -- General counter
  , @v_db_code         varchar(10)   -- Database error code
  , @v_db_msg          varchar(100)  -- Database error message

  -----------------------------------------------------
  -- General Variables
  -----------------------------------------------------
  DECLARE
    @v_return_status   integer       -- Update result status
  , @v_getkey_status   integer       -- GetKey procedure status
  , @v_row_count       integer       -- General row count
  , @v_truncate_date   datetime      -- Used to set date to midnight
  , @v_dss_source_system_key integer -- Used for dimension joins
  , @v_dss_current_flag      char(1) -- Used for dimension joins
  , @v_dss_update_time       datetime  -- Used for date insert



  -----------------------------------------------------
  -- MAIN
  -----------------------------------------------------
  SELECT @v_step = 0
  BEGIN TRY

  -- Set initial variable values
  SELECT
    @v_set           = 0
  , @v_analyze_flag  = 0
  , @v_step          = 1
  , @v_update_count  = 0
  , @v_insert_count  = 0
  , @v_dss_source_system_key = 1
  , @v_dss_current_flag = 'Y'
  , @v_dss_update_time = GETDATE()
```

```
--------------------------------------------------------
-- Delete all existing data from the staging table
--------------------------------------------------------
SET @v_sql = N'TRUNCATE TABLE stage_orders_Master'
EXEC @v_return_status = sp_executesql @v_sql
IF @v_return_status <> 0
BEGIN
  SET @v_db_code = CONVERT(varchar, @v_return_status)
  SELECT @v_db_msg = description FROM master.dbo.sysmessages
  WHERE error = @v_return_status
  EXEC WsWrkAudit
      @p_status_code = 'W'
    , @p_job_name    = @p_job_name
    , @p_task_name   = @p_task_name
    , @p_sequence    = @p_sequence
    , @p_message     = 'Failure when truncating table'
    , @p_db_code     = @v_db_code
    , @p_db_msg      = @v_db_msg
    , @p_task_key    = @p_task_id
    , @p_job_key     = @p_job_id

  SET @p_return_msg = ' Failure to truncate stage table. '
    + @v_db_msg
  SET @p_status = -2
  RETURN 0
END

--------------------------------------------------------
-- Insert input records into stage_orders_Master
--------------------------------------------------------
SET @v_step = 100

BEGIN TRANSACTION

INSERT INTO stage_orders_Master WITH ( TABLOCK )
(
  order_number,
  order_date,
  customer_code,
  ship_date,
  dss_update_time
)
SELECT
  load_orders_A.order_number,
  load_orders_A.order_date,
  load_orders_A.customer_code,
  load_orders_A.ship_date,
  @v_dss_update_time
FROM
  load_orders_A

SELECT
  @v_return_status = @@ERROR
, @v_insert_count = @v_insert_count + @@ROWCOUNT

IF @v_return_status <> 0
BEGIN
  ROLLBACK
  SET @p_return_msg = 'Failed Insert into table stage_orders_Master'
  SET @v_db_code = CONVERT(varchar, @v_return_status)
  SELECT @v_db_msg = description FROM master.dbo.sysmessages
  WHERE error = @v_return_status
  EXEC WsWrkAudit 'E', @p_job_name, @p_task_name, @p_sequence,
    @p_return_msg, @v_db_code, @v_db_msg, @p_task_id, @p_job_id
  SET @p_status = -2
  RETURN 0
END
```

```
COMMIT

------------------------------------------------------------
-- Insert input records into stage_orders_Master
------------------------------------------------------------
SET @v_step = 100

BEGIN TRANSACTION

INSERT INTO stage_orders_Master WITH ( TABLOCK )
(
  order_number,
  order_date,
  customer_code,
  ship_date,
  dss_update_time
)
SELECT
  load_orders_B.order_number,
  load_orders_B.order_date,
  load_orders_B.customer_code,
  load_orders_B.ship_date,
  @v_dss_update_time
FROM
  load_orders_B

SELECT
  @v_return_status = @@ERROR
, @v_insert_count = @v_insert_count + @@ROWCOUNT

IF @v_return_status <> 0
BEGIN
  ROLLBACK
  SET @p_return_msg = 'Failed Insert into table stage_orders_Master'
  SET @v_db_code = CONVERT(varchar, @v_return_status)
  SELECT @v_db_msg = description FROM master.dbo.sysmessages
  WHERE error = @v_return_status
  EXEC WsWrkAudit 'E', @p_job_name, @p_task_name, @p_sequence,
    @p_return_msg, @v_db_code, @v_db_msg, @p_task_id, @p_job_id
  SET @p_status = -2
  RETURN 0
END

COMMIT
```

```
------------------------------------------------------
-- Insert input records into stage_orders_Master
------------------------------------------------------
SET @v_step = 100

BEGIN TRANSACTION

INSERT INTO stage_orders_Master WITH ( TABLOCK )
(
  order_number,
  order_date,
  customer_code,
  ship_date,
  dss_update_time
)
SELECT
  load_orders_C.order_number,
  load_orders_C.order_date,
  load_orders_C.customer_code,
  load_orders_C.ship_date,
  @v_dss_update_time
FROM
  load_orders_C

SELECT
  @v_return_status = @@ERROR
, @v_insert_count = @v_insert_count + @@ROWCOUNT

IF @v_return_status <> 0
BEGIN
  ROLLBACK
  SET @p_return_msg = 'Failed Insert into table stage_orders_Master'
  SET @v_db_code = CONVERT(varchar, @v_return_status)
  SELECT @v_db_msg = description FROM master.dbo.sysmessages
  WHERE error = @v_return_status
  EXEC WsWrkAudit 'E', @p_job_name, @p_task_name, @p_sequence,
    @p_return_msg, @v_db_code, @v_db_msg, @p_task_id, @p_job_id
  SET @p_status = -2
  RETURN 0
END

COMMIT


------------------------------------------------------
-- All Done report the results
------------------------------------------------------
```

WhereScape®

CHAPTER 11
DATA STORE OBJECTS

IN THIS CHAPTER

DATA STORE OBJECT OVERVIEW

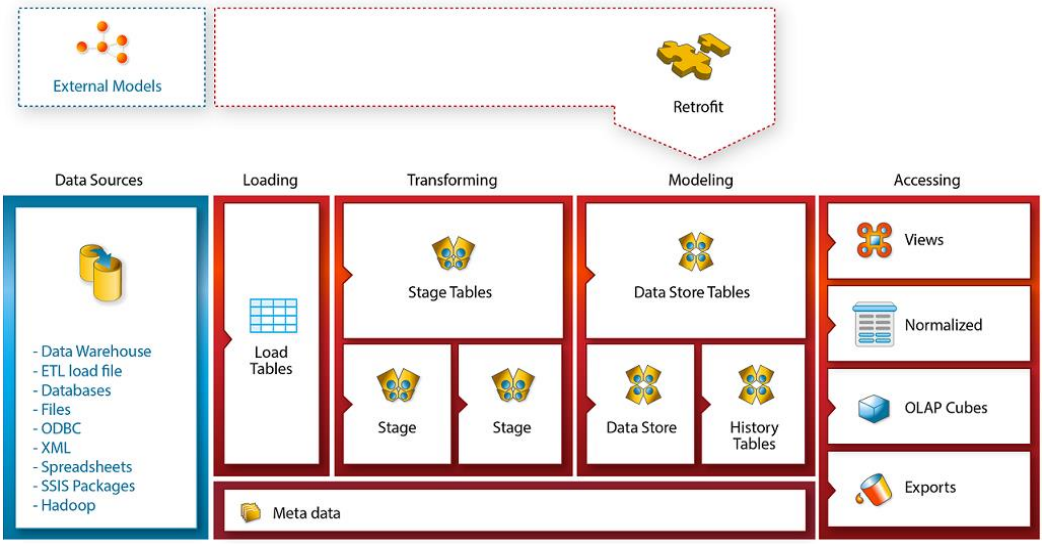
A Data Store Object is a data warehouse object used to store any type of data for later processing. In WhereScape RED, Data Store objects have many of the code generating attributes of stage, dimension and fact tables. Data Store objects can be thought of as a source system for the data warehouse.
Alternatively, they may be reported off directly by users and reporting tools. Data Store Objects can be considered either reference or transactional in nature.
A Data Store Object is built from the Data Warehouse connection. Unless you are retrofitting an existing system, Data Store Objects are typically built from one or more load or stage tables.
The Data Store model may be retrofitted from an enterprise modeling tool. See ***Importing a Data Model*** (on page 1126) for more details.
The usual steps for creating a Data Store model are defined below and are covered in this chapter.

The steps are:

- Identify the source reference or transactional data that will constitute the Data Store Object. If the data is sourced from multiple tables ascertain if a join between the source tables is possible, or if one or more intermediate stage (work) tables would be a better option.
- Using the 'drag and drop' functionality, drag the load or stage table that is the primary source of information for the Data Store Object into a Data Store target. See ***Building a Data Store Object*** (on page 405).
- If there's only one source table and all of the columns from it are being used, you can select the auto create option to build and load the table. This automatically completes the next four steps. See ***Building a Data Store Object*** (on page 405).
- Add columns from other load and/or stage tables if required. See ***Building a Data Store Object*** (on page 405).
- Create the Data Store Object in the database. See ***Building a Data Store Object*** (on page 405).
- Build the update procedure. See ***Generating the Data Store Update Procedure*** (on page 409).
- Run the update procedure and analyze the results.

If necessary, modify the update procedure or create a custom procedure.

WhereScape RED Overview: Data Store



Data Store Object Keys

Data Store Objects have Business Keys, they do not usually have Artificial Keys.

Business Key

The business key is the column or columns that uniquely identify a record within a Data Store Object.

Where the Data Store Object maps back to a single or a main table in the source system, it is usually possible to ascertain the business key by looking at the unique keys for that source table.

The business key is sometimes referred to as the 'natural' key. Examples of business keys are:

- The product SKU in a product table
- The customer code in a customer table
- The IATA airport code in an airport table.

It is assumed that business keys will never be NULL. If a null value is possible in a business key then the generated code will need to be modified to handle the null value by assigning some default value.
In the following examples, the business key column is modified by using a database function and default value:

- **DB2:** COALESCE(business_key,'N/A')
- **SQL Server:** ISNULL(business_key,'N/A')
- **Oracle:** NVL(business_key,'N/A')

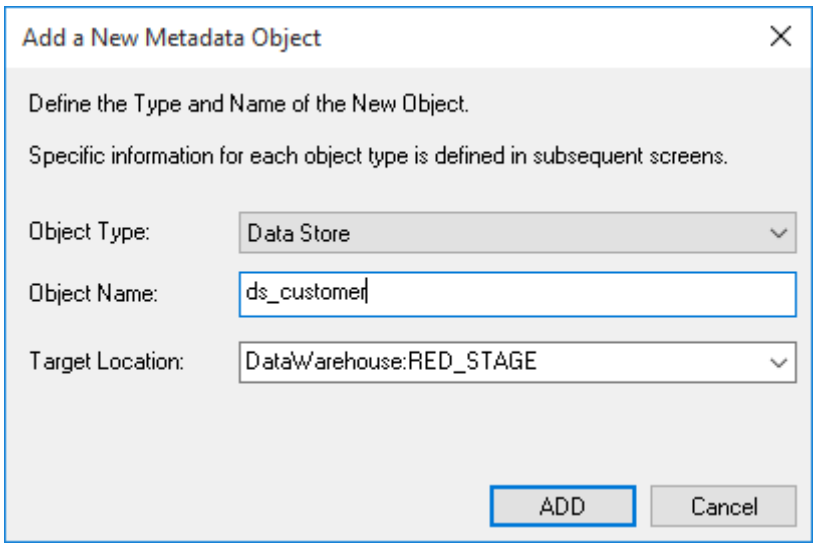
NOTE: Business keys are assumed to never be Null. If they can be null it is best to transform them to some value prior to the Data Store or stage table update.
If this is not done, an un-modified update procedure will probably fail with a duplicate key error on the business key index.

BUILDING A DATA STORE OBJECT

Data Store Objects are often sourced from one table in the base application.
The process for building a Data Store Object begins with the drag and drop of the load or stage table that contains the bulk of the Data Store Object's information.

Drag and Drop

- Create a Data Store Object target by double-clicking on the **Data Store** group in the left pane. The middle pane will display a list of all existing Data Store Objects in the current project. When such a list is displayed in the middle pane, the pane is identified as a target for new Data Store Objects.
- Browse to the Data Warehouse via the **Browse/Source Data** menu option.
- Drag the load or stage table, that contains the bulk of the Data Store Object columns, into the middle pane. Drop the table anywhere in the pane.
- The new object dialog box will appear and will identify the new object as a Data Store Object and will provide a default name based on the load or stage table name.
- Either accept this name or enter the name of the Data Store Object and click **ADD** to proceed:

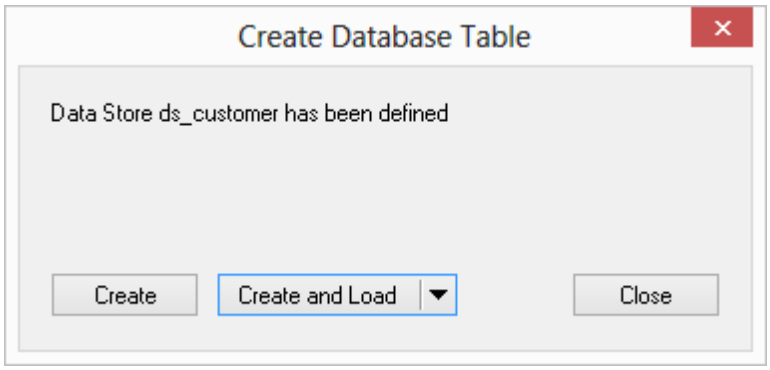


Data Store Object Properties

The table Properties dialog for the new table is now displayed. If required, the Data Store Object can be changed to be a history table by choosing History from the table type drop-down list on the right side of the dialog.
History tables are like slowly changing dimensions in dimensional data warehouses. See ***Building a Dimension*** (on page 324) for more information. Change the storage options if desired.
If prototyping, and the Data Store Object is simple (i.e. one source table) then it is possible to create, load and update the Data Store Object in a couple of steps.
If you wish to do this select the **(Build Procedure...)** option from the **Update Procedure** drop-down, and answer **Create and Load** to the next question.

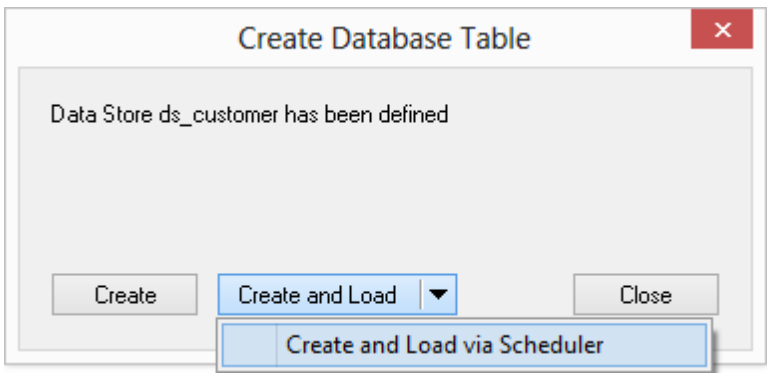
Create and Load

If you chose to build the update procedure the following dialog appears after clicking OK on the Properties page. This dialog asks if you want to create the Data Store table in the database and execute the update procedure.



If **Create** or **Create and Load** is selected and a new procedure creation was chosen proceed directly to the ***Generating the Data Store Update Procedure*** (on page 409) section.
If you have additional columns to add or columns to delete, then select **Close** and proceed as follows below.

Note: It is possible to create and load the table via the Scheduler; by selecting this option from the drop-down list on the **Create and Load** button:



Deleting and Changing columns

The columns defined for the Data Store Object will be displayed in the middle pane. It is possible to delete any unwanted columns by highlighting a column name or a group of names and clicking **Delete**.
The name of a column can also be changed by selecting the column and using the right mouse menu to edit its properties.
Any new name must conform to the database naming standards. Good practice is to use alphanumerics and the underscore character. See the section on column properties for a more detailed description of what the various fields mean.

TIP: When prototyping, and in the initial stages of an analysis area build, it is best not to remove columns, nor to change their names to any great extent. This type of activity is best left until after end users have used the data and provided feedback.

Adding additional columns

With the columns of the Data Store Object displayed in the middle pane, this pane is considered a drop target for additional columns.
It is a simple matter to select columns from other load and/or stage tables and drag these columns into the middle pane. The source table column in the middle pane shows where each column was dragged from.
The column **description** could be acquired from three different tables. Best practice is to rename at least two of the columns, perhaps also adding context to the column name. For example, description could become group_description, and so forth.
There are a number of WhereScape RED ancillary columns that do not have a source table. These columns have been added by WhereScape RED, and are added depending on earlier choices.

A description of these columns follows:

Column name	Description
dss_start_date	Used for history tables. This column provides a date time stamp when the Data Store Object record came into existence. It is used to ascertain which Data Store Object record should be used when multiple is available.
dss_end_date	Used for history tables. This column provides a date time stamp when the Data Store Object record ceased to be the current record. It is used to ascertain which Data Store Object record should be used when multiple is available.
dss_current_flag	Used for Data Store history tables. This flag identifies the current record where multiple versions exist.
dss_source_system_key	Added to support history tables that cannot be fully conformed, and the inclusion of subsequent source systems. See the ancillary settings section for more details.

Column name	Description
dss_version	Used for Data Store history tables. This column contains the version number of a Data Store history tables record. Numbered from 1 upwards with the highest number being the latest or current version. It forms part of the unique constraint for the business key of a Data Store history tables.
dss_update_time	Indicates when the record was last updated in the data warehouse.
dss_create_time	Indicates when the record was first created in the data warehouse

Creating the table

Once the Data Store Object has been defined in the metadata we need to physically create the table in the database.
This is done by right-clicking on the Data Store Object in the left pane and selecting **Create (ReCreate)** from the pop-up menu.
A results dialog box will appear to show the results of the creation. The contents of this dialog are a message to the effect that the Data Store Object was created.
A copy of the actual database create statement and if defined the results of any index create statements will be listed. For the initial create no indexes will be defined.
If the table was not created then ascertain and fix the problem. A common problem is a 'Duplicate column' where a column has the same name in two of the source tables.
The best way to find such a column is to double-click on the list heading **Col name**, which will sort the column names into alphabetical order.
Another double-click on the heading will sort the columns back into their create order.

The next section covers ***Generating the Data Store Update Procedure*** (on page 409).

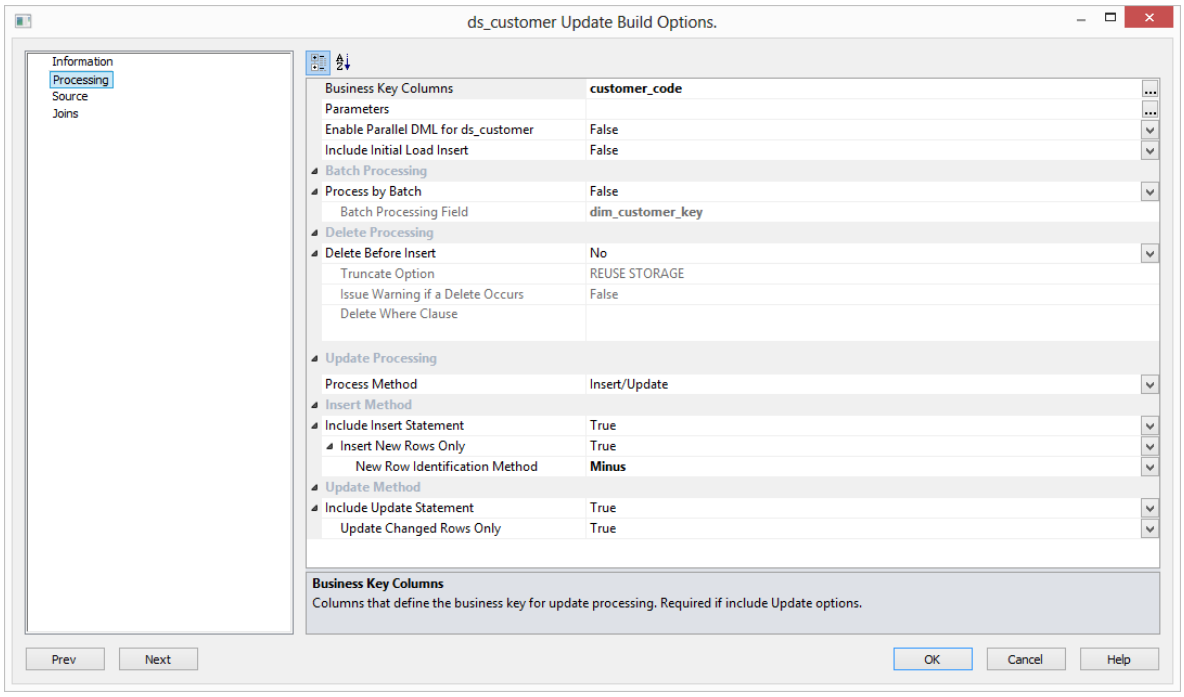
GENERATING THE DATA STORE UPDATE PROCEDURE

Once a Data Store Object has been defined in the metadata and created in the database, an update procedure can be generated to handle the joining of any tables and the update of the Data Store Object.

Generating a Procedure

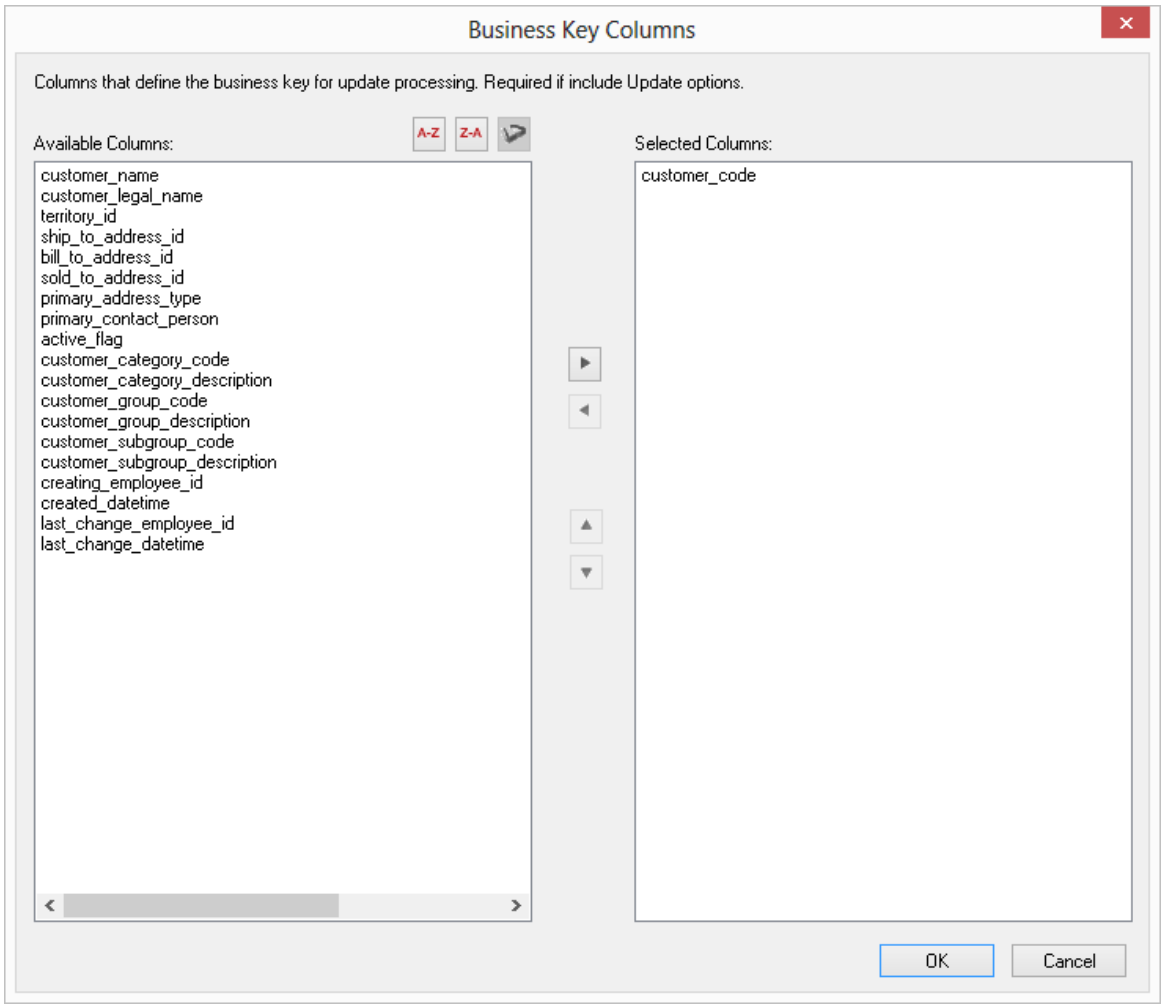
- To generate a procedure, right-click on the Data Store Object in the left pane and select **Properties**.
- Click on the **Rebuild** button to start the process of generating the new procedure.
- A series of options are available.

PROCESSING TAB



Business Key Columns: Columns that define the business key for update processing.

This is required for include Update options.
Clicking on the ellipsis button will bring up the Business Key selection screen.



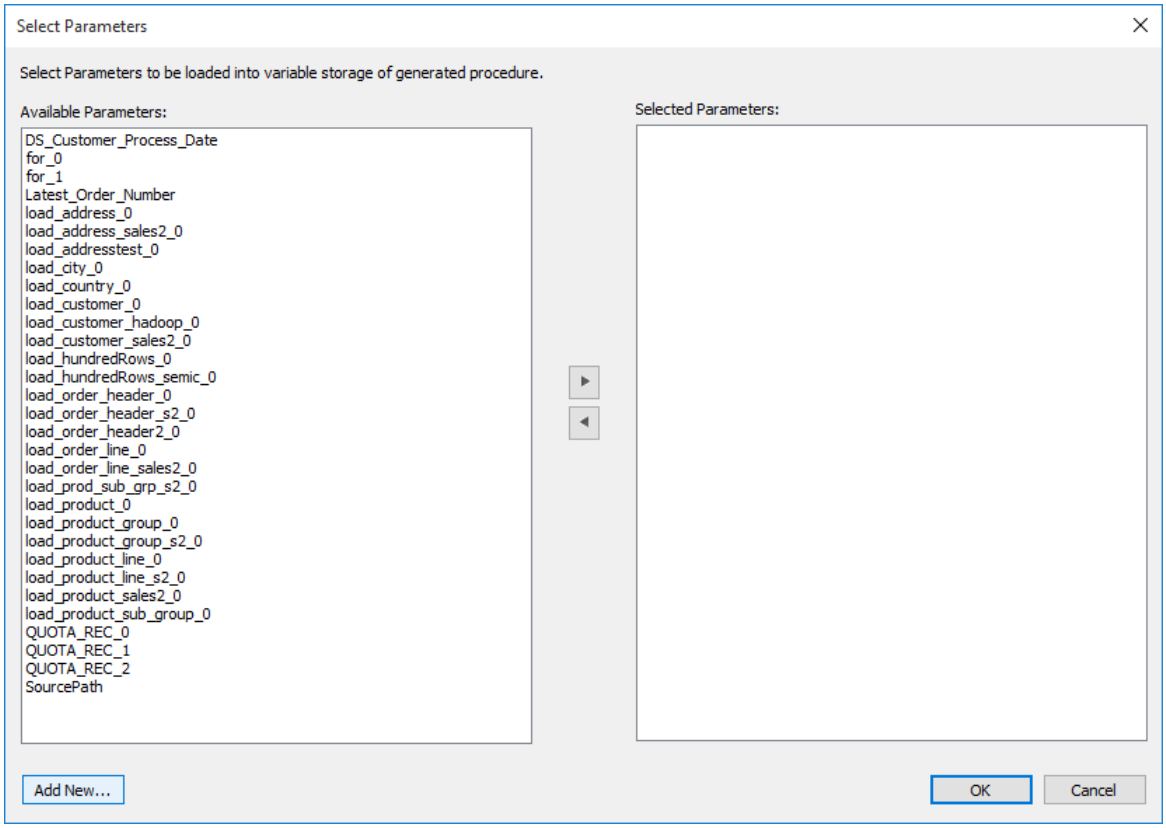
A business key can be made up of multiple columns, but it must provide a unique identifier. Where multiple columns separately uniquely identify rows in the Data Store Object, choose one to act as the primary business key.
For example, a source table may have a unique constraint on both a product code and a product description. Therefore, the description as well as the code must be unique.
It is of course possible to combine the two columns, but the normal practice would be to choose the code as the business key.

TIP1: Use the column name ascending/descending buttons to sort column names. To revert to the meta column order, click on the meta column order button.

TIP2: NULL Values - none of the columns chosen as the business key should ever contain a NULL value. See the note at the start of the Dimensions chapter.

Parameters

Any parameters selected are included in the generated update procedure as variables. The procedure will include code to retrieve the value of the parameter at run time and store it in the declared variable.
Clicking on the ellipsis button will bring up the Parameters selection screen.

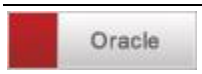


The variables can also be used in column transformations and in the from/where clause for the update procedure. Some databases have a 30 character limit for variable names. WhereScape RED ensures the variables added for any parameters are less than 30 characters long by creating variable names in the form v_ followed by the first 28 characters of the parameter name.
For example, a parameter called MINIMUM_ORDER_NUMBER_SINCE_LAST_SOURCE_LOAD will be available as the variable v_MINIMUM_ORDER_NUMBER_SINCE_L.


TIP1: WhereScape RED parameters should be unique within the first 28 characters to avoid conflicting variables names.


TIP2: If the desired parameter doesn't exist in the metadata yet, a new parameter can be added by clicking on the **Add New** button on the bottom leftmost corner of the Select Parameters dialog.

See ***Parameters*** (on page 139) for more information on WhereScape RED Parameters.

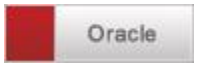


Enable Parallel DML - adds all the code required to the update procedure to enable Oracle parallel inserts. Default for this option is not set.

Include Initial Load Insert: adds an additional insert statement to the update procedure that runs if the target Data Store is empty. The benefit of this is improved performance inserting into an empty table without performing any checks to see if rows already exist. The default for this field is off (i.e. an initial insert statement is not added to the procedure).

Process by Batch: allows users to select a column to drive data processing in a loop based on the distinct ordered values of the selected columns. The update procedure loops on this column and performs the delete, update and/or insert for each value. If the column chosen is a date datatype (date, datetime or timestamp), then the user can specify yearly, monthly, daily or column level looping. The default for this field is off (do not do batch processing).

Delete Before Insert: allows selection of how to process deletes. It enables a delete statement to be added to the update procedure before any update or insert statement. This is a particularly useful option for purging old data and for updates based on a source system batch number. If this option is selected, the following options are also available:



Truncate Option – optional Oracle TRUNCATE clause which is appended to the truncate statement. Add a truncate option such as **REUSE/DROP STORAGE** by typing it in the truncate options box.

Issue Warning if a Delete occurs: this option sets the procedure to a warning state if any deletes occur.

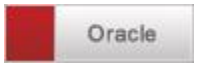
Delete Where Clause: the delete where clause is appended to the generated delete statement to constrain the rows deleted.

Process Method: Select between Insert/Update and Merge which allows you to use the Merge statement instead of two separate Insert and update statements.

Include Insert Statement: includes the insert statement in the procedure to insert new rows in the Data Store.

Insert Hint – enter a database hint to be used in the INSERT statement. This is an Oracle and SQL Server only option. Defaults can be configured in **Tools/Options/Default Update Procedure Options**.

Default is TABLOCK.



Default is APPEND.

Insert New Rows Only: uses change detection to work out what rows require inserting.

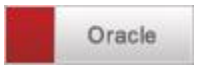
New Row Identification Method: method used to identify that records in source are not currently recorded in the target table.

Existing Data Selection Hint: database-compliant hint to be used for the existing data select statement.

Include Update Statement: includes an update statement in the procedure to update changing rows in the Data Store. If this option is chosen, then the **Update Changed rows only** option is available.

Insert Hint – enter a database hint to be used in the INSERT statement. This is an Oracle and SQL Server only option. Defaults can be configured in **Tools/Options/Default Update Procedure Options**.

Default is TABLOCK.



Default is APPEND.

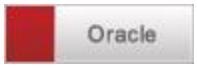
Update Changed Rows Only: uses change detection to work out what rows require updating. Choosing this option, enables the **Change Row identification Method.**

Change Row Identification Method: method used to identify that records in source have changed from what is currently recorded in the target table.

Existing Data Selection Hint: database-compliant hint to be used for the existing data select statement.

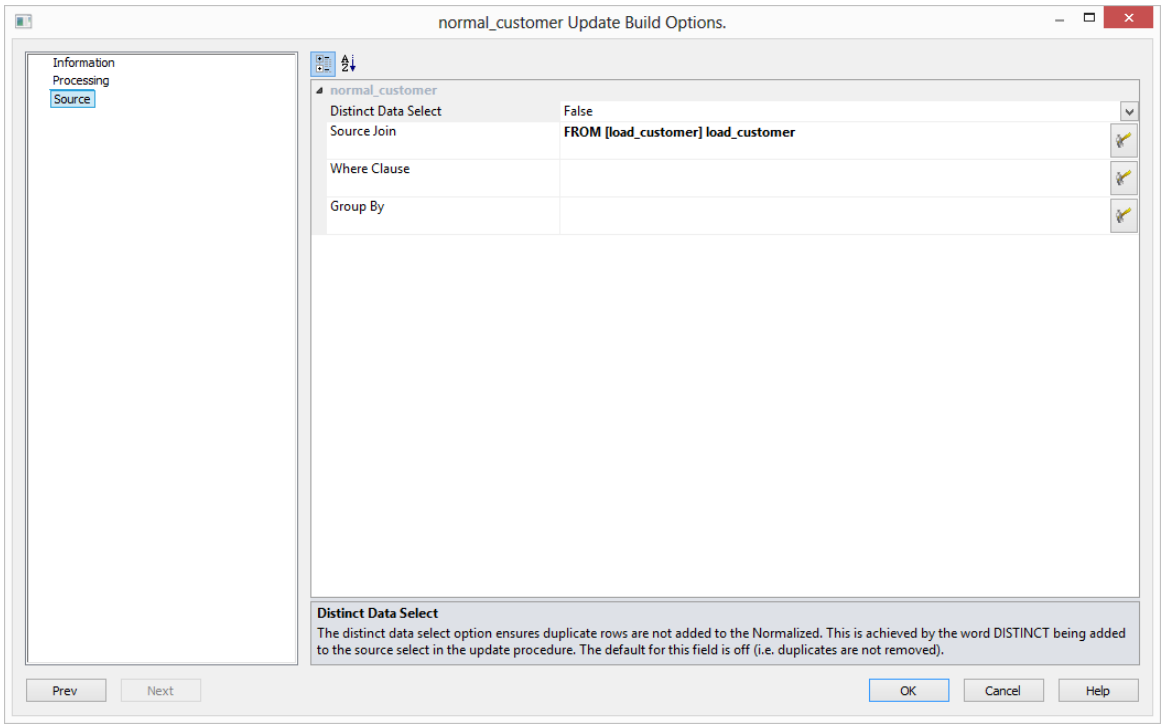
Merge Hint – enter a database hint to be used in the MERGE statement. This is an Oracle and SQL Server only option. Defaults can be configured in **Tools/Options/Default Update Procedure Options**.

Default is TABLOCK.



Default is APPEND.

SOURCE TAB



Distinct Data Select: ensures duplicate rows are not added to the Data Store. This is achieved by the word DISTINCT being added to the source select in the update procedure. The default for this field is not set.

Source Join: The From clause, including Source Join information.

Where Clause: The Where clause.

Group By: The Group By clause.

Simple Join

A simple join only returns rows where data is matched in both tables. So for example if table A has 100 rows and table B has a subset of 24 rows. If all the rows in table B can be joined to table A then 24 rows will be returned. The other 76 rows from table A will not be returned.

Outer Join

The outer join returns all rows in the master table regardless of whether they are found in the second table. So if the example above was executed with table A as the master table then 100 rows would be returned. 76 of those rows would have null values for the table B columns.

NOTE: When WhereScape RED builds up an outer join, it needs to know which table is the master table and which is subordinate. Select the join column from the master table first. In the example screen above the table 'load_order_header' has had its column chosen and the column for the table 'load_order_line' is currently being chosen. This will result in the 'load_order_header' table being defined as the master, as per the example statement above. The results of this example select are that a row will be added containing order information regardless of whether a corresponding load_order_line entry exists.

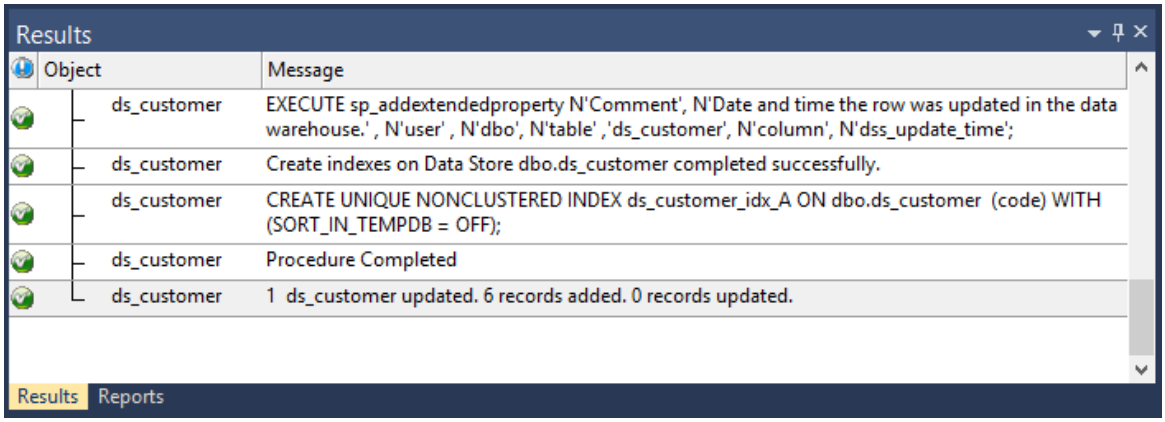
NOTE: When upgrading from a RED version before 6.8.2.0 and moving existing objects to a target location, all procedures that reference those objects will need to be rebuilt.
Any **FROM** clauses will also need to be manually regenerated for the table references to be updated to the new [TABLEOWNER] form.

Building and Compiling the Procedure

- Once the relevant options are completed, click **OK**. The procedure will be built and compiled.
- If the compile fails an error will be displayed along with the first few lines of error messages.
- Compile fails typically occur when the physical creation of the table was not done. If the compile fails for some other reason the best approach is to use the procedure editor to edit and compile the procedure.
- The procedure editor will highlight all the errors within the context of the procedure.
- Once the procedure has been successfully compiled it can either be executed interactively or passed to the scheduler.

Indexes

By default, a number of indexes will be created to support each Data Store Object.
These indexes will be added once the procedure has been built. An example of the type of indexes created is as follows:



Additional indexes can be added, or these indexes changed. See the chapter on indexes for further details.

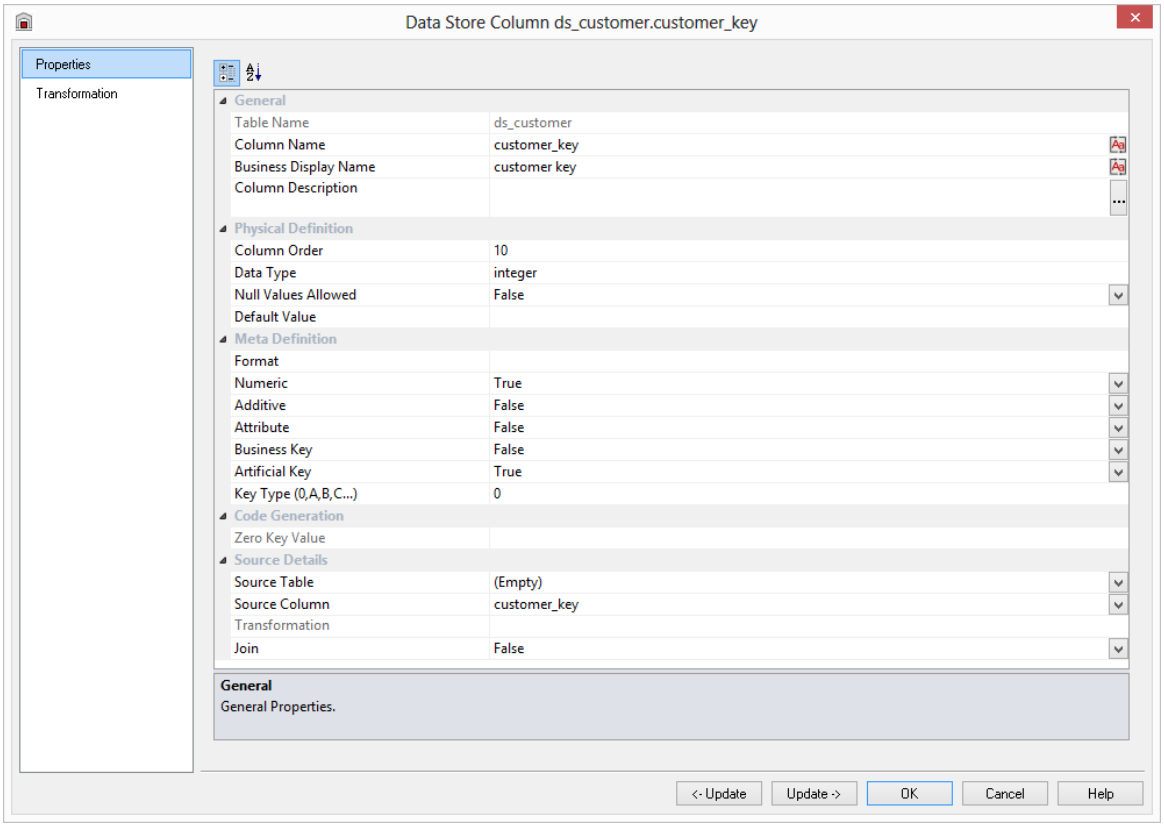
DATA STORE ARTIFICIAL KEYS

By default, Data Store Objects in WhereScape RED do not have an artificial (surrogate) key. Artificial keys can be added manually, but needing to do so could indicate Data Store Objects are not the correct WhereScape RED object for this table (perhaps an EDW 3NF Table would be more appropriate).
The quickest way to do this is to add an extra column to the Data Store Object by using either **Add Column** or **Copy Column**.
Edit the properties of the new column to have the correct name and order, source table and column, datatype, key type and flags.

Specifically:

- The **Column Name** and **Source Column** name should be the same.
- The **Source Table** should be empty.
- The **Data Type** should be:
  - DB2: integer generated by default as identity (start with 1, increment by 1)
  - Oracle: integer
  - SQL Server: integer identity(0,1)
- The **Key Type** should be 0.
- Only the **Numeric** and **Artificial Key** flags should be set on.

The following example shows a manually added artificial key column:



The artificial key for a Data Store Object is set via a sequence in Oracle and an identity column in SQL Server and DB2. This artificial key normally, and by default, starts at one and progresses as far as is required.

A WhereScape standard for the creation of special rows in the data store tables is as follows:

Key value	Usage
1 upwards	The standard artificial keys are numbered from 1 upwards, with a new number assigned for each distinct Data Store Object record.
0	Used as a join to the Data Store Object when no valid join existed. It is the convention in the WhereScape generated code that any EDW 3NF table business key that either does not exist or does not match is assigned to key 0.
-1 through -9	Used for special cases. The most common being where an EDW 3NF table is not appropriate for the record. A new key is used rather than 0 as we want to distinguish between records that are invalid and not appropriate.
-10 backward	Pseudo records. In many cases, we must deal with different granularities in our data. For example, we may have a table that contains actual sales at a product SKU level and budget information at a product group level. The product table only contains SKU based information. To be able to map the budget records to the same table, we need to create these pseudo keys that relate to product groups. The values -10 and backwards are normally used for such keys.

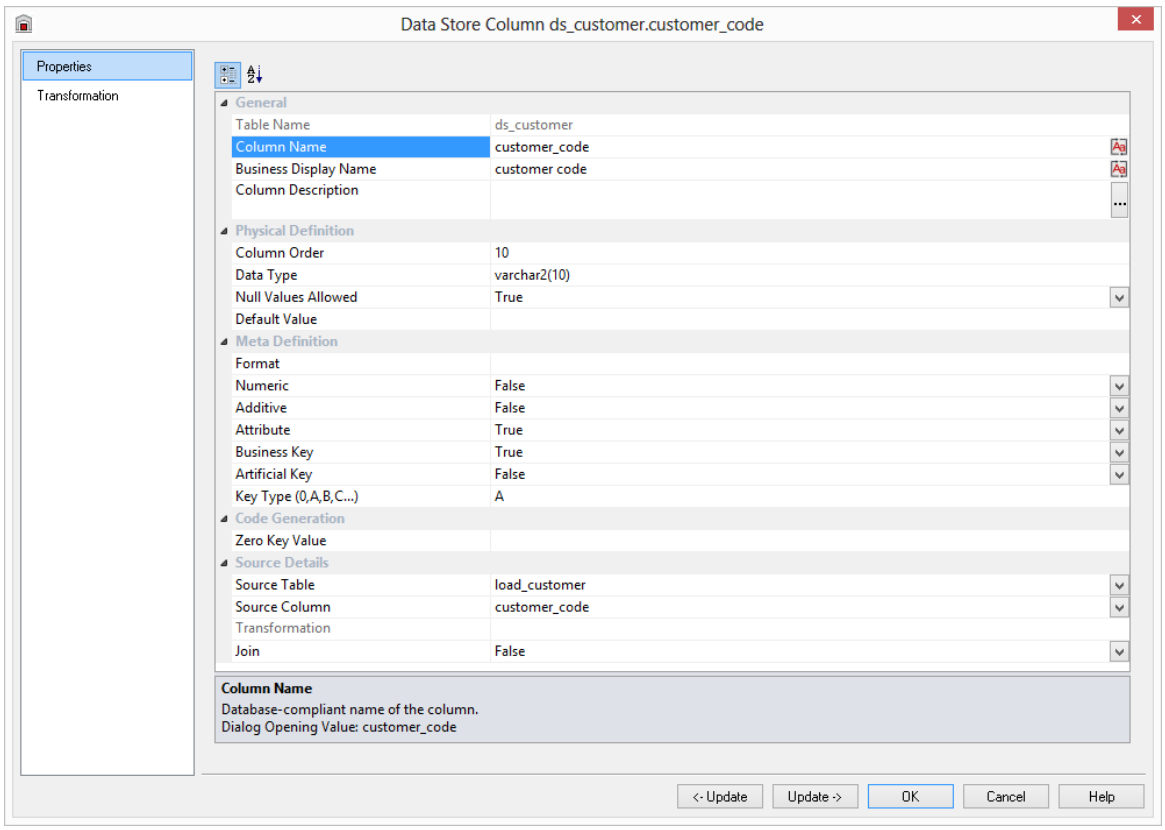
Note: To have a surrogate key auto added for Data Store tables, see ***Global Naming of Key Columns*** (on page 95).

DATA STORE COLUMN PROPERTIES

Each Data Store Object column has a set of associated properties. The definition of each property is described below:
If the **Column name** or **Data type** is changed for a column then the metadata will differ from the table as recorded in the database.
Use the **Validate/Validate Table Create Status** menu option or the right mouse menu to compare the metadata to the table in the database.
A right mouse menu option of **Alter table** is available when positioned on the table name after the validate has completed. This option will alter the database table to match the metadata definition.

TIP: If a database table's definition is changed in the metadata then the table will need to be altered in the database.
Use the **Validate/Validate Table Create Status** to compare metadata definitions to physical database tables. The option also provides the ability to alter the database table, through a pop-up menu option from the validated table name.

A sample **Properties** screen is as follows:



 **TIP:** The two special update keys allow you to update the column and step either forward or backward to the next column's properties.
ALT-Left Arrow and **ALT-Right Arrow** can also be used instead of the two special update keys.

Table Name

Database-compliant name of the table that contains the column. [Read-only].

Column Name

Database-compliant name of the column. Typically, column-naming standards exclude the use of spaces etc. A good practice is to only use alphanumerics, and the underscore character. Changing this field alters the table's definition.

Note: A case conversion button on the right converts the text between different cases: UPPER CASE, Capitalized Case and lower case. The mode cycles to the next case in the sequence each conversion.

Business Display Name

Name that the business uses to refer to the column, which is included in the RED-generated documentation and can be used in the end user layer of other tools. [Does NOT affect the physical database table]. As such it is a free form entry and any characters are valid.

Note: A case conversion button on the right converts the text between different cases: UPPER CASE, Capitalized Case and lower case. The mode cycles to the next case in the sequence each conversion.

Column Description

This field contains the description for the column. It may be a description from a business user's point of view. This field might additionally contain information on where and how the column was acquired. For example if the column is sourced from multiple tables or is a composite or derived column then this definition would normally describe the process used to populate the column. This field is used in the documentation and is available via the view **ws_admin_v_dim_col .** This field is also stored as a comment against the column in the database.

Column Order

Numeric value that controls the relative order of columns in the database create statement. The lowest numbered column will appear first in the table. Although this affects the physical table definition no action will be taken unless the table is re-created in the database. The columns can be re-numbered based on the existing order by choosing the **Respace Order Number** pop-up menu option when positioned over any column in the table. This action will number the columns in increments of 10 starting at 10. In addition to a simple change of the order field, the column order can be changed by first displaying the columns in the middle pane and then using drag and drop to move the columns around. This drag and drop process will automatically renumber the columns as required.

Data Type

Database-compliant data type that must be valid for the target database. Typical data types for Oracle are integer, number, char(), varchar2() and date. For SQL Server common types are integer, numeric, varchar() and datetime. See the database documentation for a description of the data types available. Changing this field alters the table's definition.

Null Values Allowed

Determines whether the table column can hold NULL values or whether a value is always mandatory.

Default Value

Initial value that is assigned to the column when a row is inserted into the table but no value is specified for the column.

Format

Optional format mask that can be used in end user tools. [Does NOT affect the physical database table]. As such it is a free form entry and any characters are valid. Typically format masks are only used on numeric fields. Example: #,###0.00. It is not worth the effort of populating this field unless it can be utilized by the end user tools in use.

Numeric

Indicates whether the table column holds values that are numeric. This is normally only relevant for fact tables. It does not affect the physical table definition, but rather provides input to the view **ws_admin_v_dim_col** which can be used to assist in the population of an end user tool's end user layer. The use of this field is not relevant unless it can be utilized by the end user tools.

Additive

Indicates whether the table column holds values that are additive. This implies that the column can be summed when performing data grouping in a query. This is normally only relevant for fact tables. It does not affect the physical table definition, but rather provides input to the view **ws_admin_v_dim_col** which can be used to assist in the population of an end user tool's end user layer. The use of this field is not relevant unless it can be utilized by the end user tools.

Attribute

Indicates whether the table column holds values that are descriptive, and/or are used for grouping/summing. An attribute is defined as a column that is non factual. For example, we may have an order number, or a invoice number stored in the fact table. Such columns are considered attributes, rather than facts. This checkbox is therefore normally only relevant for fact tables. This checkbox does not affect the physical table definition, but rather provides input to the view **ws_admin_v_dim_col** which can be used to assist in the population of an end user tools end user layer. The use of this field is not relevant unless it can be utilized by the end user tools.

Business Key

Indicates whether the column is part of the BUSINESS key, which is defined during the update procedure build. [Normally maintained automatically]. Multiple columns can form the primary business key.

Artificial Key

Indicates whether the column is the artificial key. Only one artificial key is supported. This indicator is set by WhereScape RED during the initial drag and drop creation of a table, and should not normally be altered.

Key Type

Key type that is assigned and used when generating the table's update procedure and indexes. [Normally maintained automatically]. It can be altered here, but this should only be done if the consequences are fully understood and tested.
The supported values are:

Key type	Meaning
0	The artificial key. Set when the key is added during drag and drop table generation.
1	Component of all business keys. Indicates that this column is used as part of any business key.
A	Indicates that the column is part of the primary business key. Set whenever a business key is defined as part of an update procedure generation.
B-Z	Indicates that the column is part of a secondary business key. Only used during index generation and not normally set.

Source Table

Identifies the source table where the column's data comes from. This source table is normally a load table within the data warehouse. If the column was sourced from multiple tables, then the normal practice is to record one of the tables in this field and a comment listing all of the other tables in the **Source strategy** field. This field is used when generating a procedure to update the Data Store object. It is also used in the track back diagrams and in the documentation.

Source Column

Identifies the source column where the column's data comes from. Such a column is normally a load table column, which in turn may have been a transformation or the combination of multiple columns. This may also be a dimensional key where a dimension is being joined.

Transformation

Transformation. [Read-only].

Join

Indicates whether the table column is used in a table join. [Normally maintained automatically but can be optionally changed to override the default join logic used in the generated update procedure]. The **Source table** and **Source column** fields will provide the dimension table's side of the join. The options for this field are: False, True, Manual and Pre Join.

Setting this field to Manual changes the way the dimension table is looked up during the update procedure build. It allows you to join the dimension manually in the Cursor mapping dialog (where the 'Where' clause is built).

Setting this field to Pre Join activates the **Pre Join Source Table** field and allows you to select a table from the drop-down list.

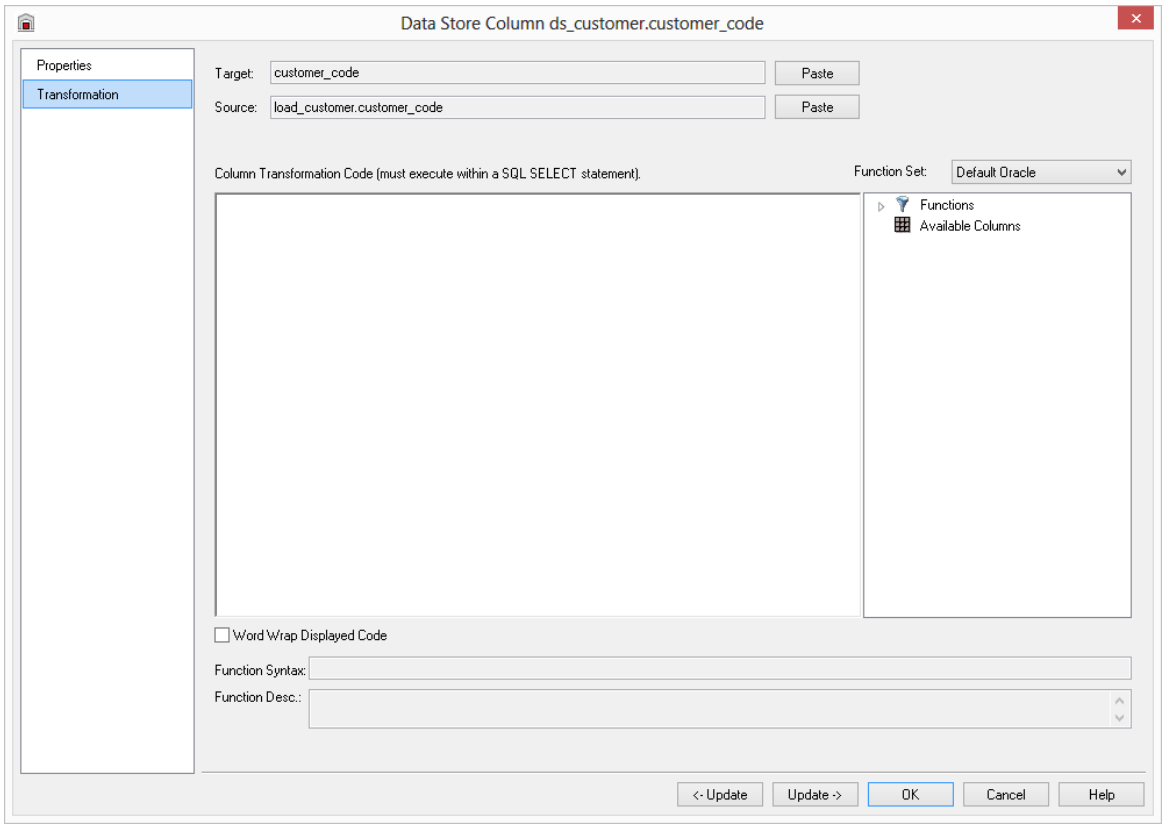
Pre Join Source Table

Indicates the table from which the pre joined column was sourced. When the Join option is set to False, this field becomes inactive. When the Join option is set to True or Manual, this field is set to the current table name. When the Join option is set to Pre Join, then you can select the required table from the drop-down list.

DATA STORE OBJECT COLUMN TRANSFORMATIONS

Each Data Store Object column can have a transformation associated with it. The transformation will be included in the generated procedure and will be executed as part of the procedure update. The transformation must therefore be a valid SQL construct that can be included in a **Select** statement.
For example, we could have a transformation of 'load_order_line.qty * 0.125' to calculate a tax column of 12.5%.

Click the **Transformation** tab to enter a transformation.
The transformation screen is as follows:



Note: Transformations are only put into effect when the procedure is re-generated.

See ***Transformations*** (on page 650) for more details.

WhereScape®

CHAPTER 12
EDW 3NF TABLES

NOTE: EDW 3NF/Normalized Table rename

Former WhereScape RED Normalized Tables have been renamed to EDW 3NF from RED version 6.8.4.3.
However, this change applies only for new metadata repositories, existing metadata repositories will not be affected and will not have its table's naming modified.
All references to Normalized tables have been updated in the RED documentation from version 6.8.4.3 onwards, however, some screenshots of the RED left pane browser might still show instances of the Normalized object type instead of the new EDW 3NF type.

To modify table naming from Normalized to EDW 3NF in existing repositories see ***Object Type Names*** and ***Global Naming Conventions***.
Please note that short name and table prefixes can be overwritten by the **Local Naming conventions** setting in the **User Preferences**. If this is the case, you can disable this option here: ***Local Naming conventions***.

IN THIS CHAPTER

EDW 3NF TABLE OVERVIEW

An EDW 3NF Table is a data warehouse object used to build third normal form enterprise data warehouses.
In WhereScape RED, EDW 3NF objects have many of the code generating attributes of stage, dimension and fact tables.
Third normal form enterprise data warehouses can be thought of as a source system for star schema data marts. Alternatively, they may be reported off directly by users and reporting tools. EDW 3NF tables can be considered either reference or transactional in nature.

An EDW 3NF table is built from the Data Warehouse connection. Unless you are retrofitting an existing system, EDW 3NF Tables are typically built from one or more load or stage tables.
The EDW 3NF model may be retrofitted from an enterprise modeling tool. See ***Importing a Data Model*** (on page 1126) for more details.

The usual steps for creating an EDW 3NF model are defined below and are covered in this chapter. The steps are:

- Identify the source reference or transactional data that will constitute the EDW 3NF Table. If the data is sourced from multiple tables ascertain if a join between the source tables is possible, or if one or more intermediate stage (work) tables would be a better option.
- Using the 'drag and drop' functionality drag the load or stage table that is the primary source of information for the EDW 3NF Table into an EDW 3NF target. See ***Building an EDW 3NF Table*** (on page 429).
- If there's only one source table and all of the columns from it are being used, you can select the auto create option to build and load the table. This automatically completes the next four steps. See ***Building an EDW 3NF Table*** (on page 429).
- Add columns from other load and/or stage tables if required. See ***Building an EDW 3NF Table*** (on page 429).
- Create the EDW 3NF Table in the database. See ***Building an EDW 3NF Table*** (on page 429).
- Build the update procedure. See ***Generating the Update Procedure*** (see "***Generating the EDW 3NF Update Procedure***" on page 433).
- Run the update procedure and analyze the results.

If necessary, modify the update procedure or create a custom procedure.

EDW 3NF Table Keys

EDW 3NF Tables have two types of keys that we will refer to frequently. These are the Business Key and the Artificial Key. A definition of these two key types follows:

Business Key

The business key is the column or columns that uniquely identify a record within an EDW 3NF Table. Where the EDW 3NF Table maps back to a single or a main table in the source system, it is usually possible to ascertain the business key by looking at the unique keys for that source table. The business key is sometimes referred to as the 'natural' key. Examples of business keys are:

- The product SKU in a product table
- The customer code in a customer table
- The IATA airport code in an airport table.

It is assumed that business keys will never be NULL. If a null value is possible in a business key then the generated code will need to be modified to handle the null value by assigning some default value.
In the following examples, the business key column is modified by using a database function and default value:

DB2: COALESCE(business_key,'N/A')

SQL Server: ISNULL(business_key,'N/A')

Oracle: NVL(business_key,'N/A')

Note: Business keys are assumed to never be Null. If they can be null it is best to transform them to some value prior to the EDW 3NF or stage table update.
If this is not done, an un-modified update procedure will probably fail with a duplicate key error on the business key index.

Artificial Key

By default, EDW 3NF Tables in WhereScape RED do not have an artificial key (artificial keys can be added manually, see ***EDW 3NF Table Artificial Keys*** (on page 441) for more details).

An artificial key is the unique identifier that can be used to join an EDW 3NF Table record to other EDW 3NF Tables.
When joining EDW 3NF Tables it would be possible to perform the join using the business key. For EDW 3NF Tables that satisfy one of more of the following conditions, joining with business keys could result in slow query times and excessive use of database storage:

- Multiple column business keys (excessive storage and multiple column joins)
- One or more large character business key columns (excessive storage)
- Very large tables (excessive storage - integer artificial keys often use less space than one small character field)
- History EDW 3NF Tables (complex joins involving a between dates construct)

As query time is one of our key drivers in data warehouse implementations the best answer is often to use some form of artificial key.
A price is paid in the additional processing required doing key lookups, but this is offset by the reduced query times and reduced complexity.

The artificial key is an integer and is built sequentially from 1 upwards.

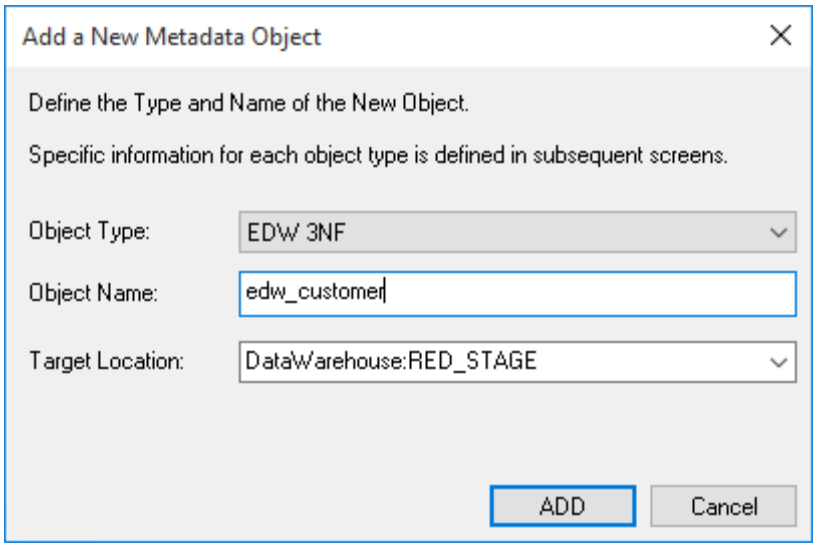
See the section on artificial keys for a more detailed explanation. An artificial key is sometimes referred to as a "surrogate" key.

BUILDING AN EDW 3NF TABLE

EDW 3NF tables are often sourced from one table in the base application. The process for building an EDW 3NF table begins with the drag and drop of the load or stage table that contains the bulk of the EDW 3NF table's information.

Drag and Drop

- Create an EDW 3NF table target by double-clicking on the **EDW 3NF** group in the left pane. The middle pane will display a list of all existing EDW 3NF tables in the current project. When this list is displayed in the middle pane, the pane is identified as a target for new EDW 3NF tables.
- Browse to the Data Warehouse via the Browse/Source Data menu option.
- Drag the load or stage table, that contains the bulk of the EDW 3NF table columns, into the middle pane. Drop the table anywhere in the pane.
- The new object dialog box will appear and will identify the new object as a EDW 3NF table and will provide a default name based on the load or stage table name.
- Either accept this name or enter the name of the EDW 3NF table and click ADD to proceed:

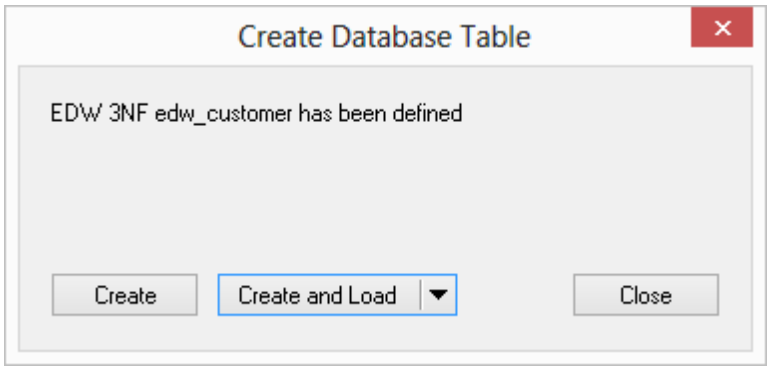


EDW 3NF Table Properties

The table Properties dialog for the new table is now displayed. If required, the EDW 3NF table can be changed to be a history table by choosing History from the table type drop-down list on the right side of the dialog. History tables are like slowly changing dimensions in dimensional data warehouses. See ***Building a Dimension*** (on page 324) for more information. Change the storage options if desired.
If prototyping, and the EDW 3NF table is simple (i.e. one source table) then it is possible to create, load and update the EDW 3NF table in a couple of steps.
If you wish to do this select the **(Build Procedure...)** option from the **Update Procedure** drop-down, and answer **Create and Load** to the next question.

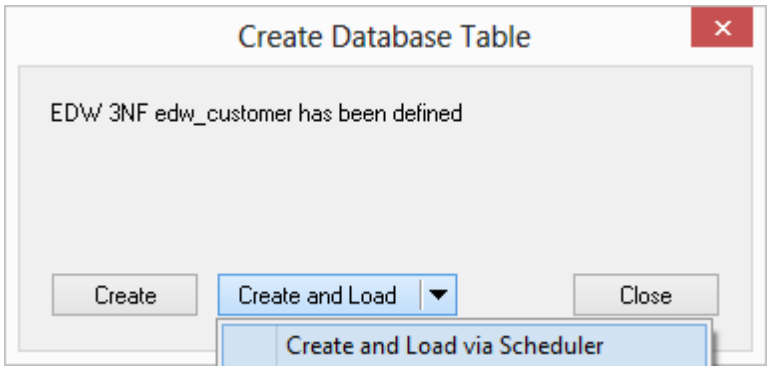
Create and Load

If you chose to build the update procedure the following dialog appears after clicking OK on the Properties page. This dialog asks if you want to create the EDW 3NF table in the database and execute the update procedure.



If you are satisfied with the columns that will be used and do not wish to add any additional columns you can select the **Create and Load** button. Alternatively, the **Create** button creates the table in the repository but does not execute an update, allowing you to change columns before loading data into the table.
If **Create** or **Create and Load** is selected and a new procedure creation was chosen proceed directly to the ***Generating the EDW 3NF Update Procedure*** (on page 433) section.

Note: It is possible to create and load the table via the Scheduler; by selecting this option from the drop-down list on the **Create and Load** button:



If you have additional columns to add or columns to delete, then select **Close** and proceed as follows below.

Deleting and Changing columns

The columns defined for the EDW 3NF table will be displayed in the middle pane. It is possible to delete any unwanted columns by highlighting a column name or a group of names and clicking **Delete**.
The name of a column can also be changed by right-clicking on the column and choosing **Properties** to edit its properties.
Any new name must conform to the database naming standards. Good practice is to use alphanumerics and the underscore character.
See the section on column properties for a more detailed description on what the various fields mean.

TIP: When prototyping, and in the initial stages of an analysis area build it is best not to remove columns, nor to change their names to any great extent. This type of activity is best left until after end users have used the data and provided feedback.

Adding additional columns

With the columns of the EDW 3NF table displayed in the middle pane, this pane is considered a drop target for additional columns.
It is a simple matter to select columns from other load and/or stage tables and drag these columns into the middle pane. The source table column in the middle pane shows where each column was dragged from.
The column **description** could be acquired from three different tables. Best practice is to rename at least two of the columns, perhaps also adding context to the column name. For example, description could become group_description, and so forth.
There are a number of WhereScape RED ancillary columns that do not have a source table. These columns have been added by WhereScape RED, and are added depending on earlier choices.
A description of these columns follows.

Column name	Description
dss_start_date	Used for history tables. This column provides a date time stamp when the EDW 3NF table record came into existence. It is used to ascertain which EDW 3NF table record should be used when multiple are available.
dss_end_date	Used for history tables. This column provides a date time stamp when the EDW 3NF table record ceased to be the current record. It is used to ascertain which EDW 3NF table record should be used when multiple are available.
dss_current_flag	Used for EDW 3NF history tables. This flag identifies the current record where multiple versions exist.
dss_source_system_key	Added to support history tables that cannot be fully conformed, and the inclusion of subsequent source systems. See the ancillary settings section for more details.

Column name	Description
dss_version	Used for EDW 3NF history tables. This column contains the version number of an EDW 3NF history tables record. Numbered from 1 upwards with the highest number being the latest or current version. It forms part of the unique constraint for the business key of an EDW 3NF history tables.
dss_update_time	Indicates when the record was last updated in the data warehouse.
dss_create_time	Indicates when the record was first created in the data warehouse

Creating the table

Once the EDW 3NF table has been defined in the metadata we need to physically create the table in the database.
This is done by right-clicking on the EDW 3NF table in the left pane and selecting **Create (ReCreate).**
A results dialog box will appear to show the results of the creation. The contents of this dialog are a message to the effect that the EDW 3NF table was created. A copy of the actual database create statement and if defined the results of any index create statements will be listed. For the initial create no indexes will be defined.
If the table was not created, then ascertain and fix the problem. A common problem is a 'Duplicate column' where a column has the same name in two of the source tables.
The best way to find such a column is to double-click on the list heading **Col name**; which will sort the column names into alphabetical order.
Another double-click on the heading will sort the columns back into their create order.

The next section covers ***Generating the Update Procedure*** (see "***Generating the EDW 3NF Update Procedure***" on page 433).

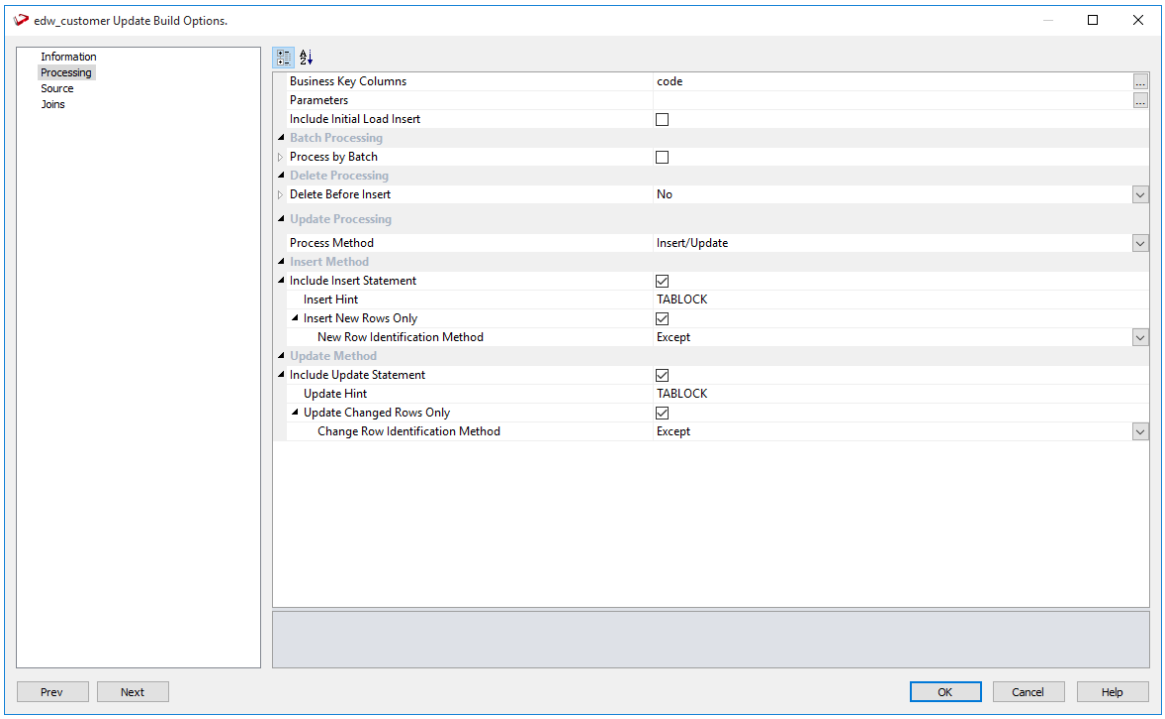
GENERATING THE EDW 3NF UPDATE PROCEDURE

Once an EDW 3NF Object has been defined in the metadata and created in the database, an update procedure can be generated to handle the joining of any tables and the update of the EDW 3NF Object.

Generating a Procedure

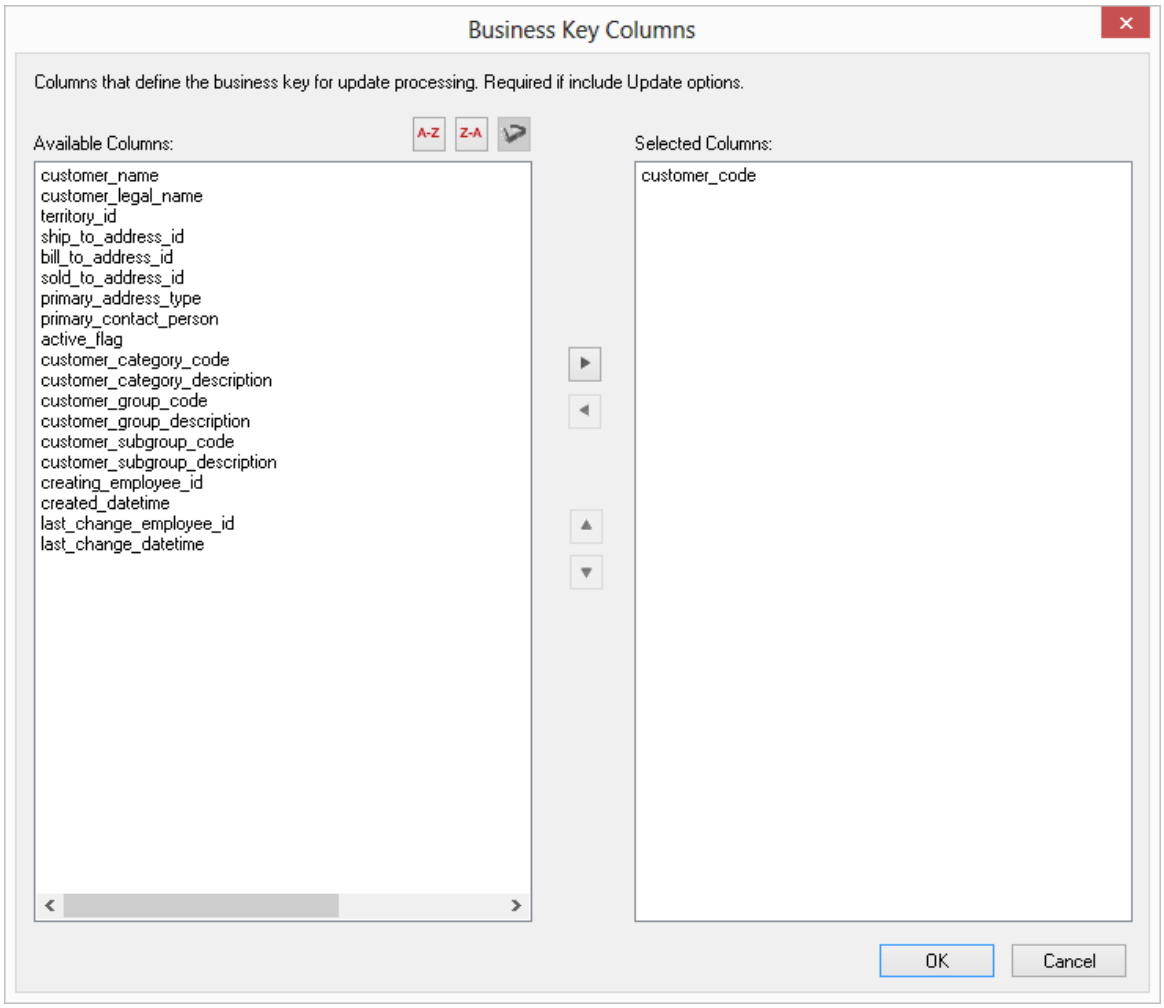
- To generate a procedure, right-click on the EDW 3NF Object in the left pane and select **Properties**.
- Click on the **Rebuild** button to start the process of generating the new procedure.
- A series of options are available.

PROCESSING TAB (ORACLE SCREEN EXAMPLE)



Business Key Columns

Columns that define the business key for update processing. Required for include Update options. Clicking on the ellipsis button will bring up the Business Key selection screen.



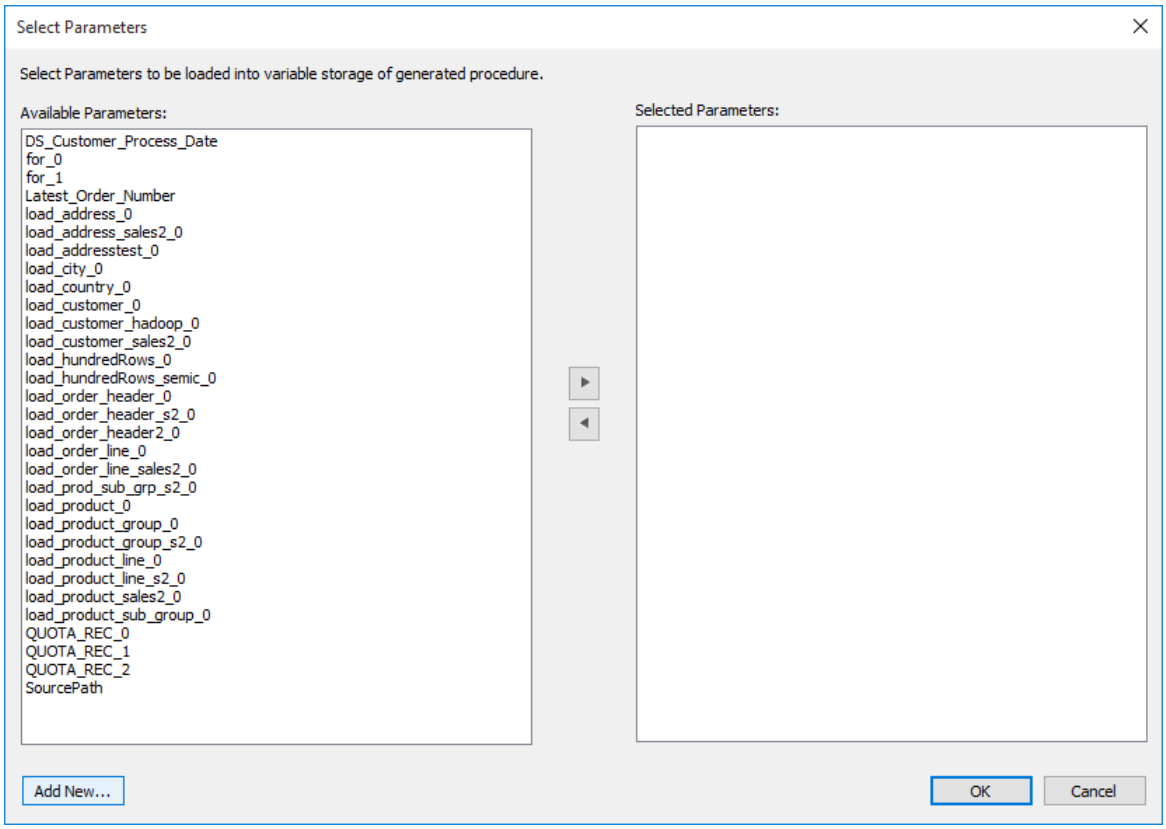
A business key can be made up of multiple columns, but it must provide a unique identifier. Where multiple columns separately uniquely identify rows in the EDW 3NF Object, choose one to act as the primary business key. For example, a source table may have a unique constraint on both a product code and a product description. Therefore, the description as well as the code must be unique. It is possible to combine the two columns, but the normal practice would be to choose the code as the business key.

TIP1: Use the column name ascending/descending buttons to sort column names. To revert to the meta column order, click on the meta column order button.

TIP2: NULL Values - none of the columns chosen as the business key should ever contain a NULL value.

Parameters

Any parameters selected are included in the generated update procedure as variables. The procedure will include code to retrieve the value of the parameter at run time and store it in the declared variable.
Clicking on the ellipsis button will bring up the Parameters selection screen.

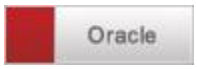


The variables can also be used in column transformations and in the from/where clause for the update procedure. Some databases have a 30 character limit for variable names. WhereScape RED ensures the variables added for any parameters are less than 30 characters long by creating variable names in the form v_ followed by the first 28 characters of the parameter name.
For example, a parameter called MINIMUM_ORDER_NUMBER_SINCE_LAST_SOURCE_LOAD will be available as the variable v_MINIMUM_ORDER_NUMBER_SINCE_L.

TIP1: WhereScape RED parameters should be unique within the first 28 characters to avoid conflicting variables names.

TIP2: If the desired parameter doesn't exist in the metadata yet, a new parameter can be added by clicking on the **Add New** button on the bottom leftmost corner of the Select Parameters dialog.

See ***Parameters*** (on page 139) for more information on WhereScape RED Parameters

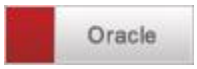


Enable Parallel DML - adds all the code required to the update procedure to enable Oracle parallel inserts. Default for this option is not set.

Include Initial Load Insert: adds an additional insert statement to the update procedure that runs if the target EDW 3NF object is empty. The benefit of this is improved performance inserting into an empty table without performing any checks to see if rows already exist. The default for this field is off (i.e. an initial insert statement is not added to the procedure).

Process by Batch: allows users to select a column to drive data processing in a loop based on the distinct ordered values of the selected columns. The update procedure loops on this column and performs the delete, update and/or insert for each value. If the column chosen is a date datatype (date, datetime or timestamp), then the user can specify yearly, monthly, daily or column level looping. The default for this field is off (do not do batch processing).

Delete Before Insert: allows selection of how to process deletes. It enables a delete statement to be added to the update procedure before any update or insert statement. This is a particularly useful option for purging old data and for updates based on a source system batch number. If this option is selected, the following dialog is displayed:



Truncate Option – optional Oracle TRUNCATE clause which is appended to the truncate statement. Add a truncate option such as **REUSE/DROP STORAGE** by typing it in the truncate options box.

Issue Warning if a Delete occurs: this option sets the procedure to a warning state if any deletes occur.

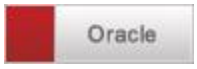
Delete Where Clause: the delete where clause is appended to the generated delete statement to constrain the rows deleted.

Process Method: Select between Insert/Update and Merge which allows you to use the Merge statement instead of two separate Insert and update statements.

Include Insert Statement: includes the insert statement in the procedure to insert new rows in the EDW 3NF Object.

Insert Hint: enter a database compliant hint to be used in the INSERT statement. This is an Oracle and SQL Server only option. Defaults can be configured in **Tools/Options/Default Update Procedure Options**.

Default is TABLOCK.



Default is APPEND.

Insert New Rows Only: uses change detection to work out what rows require inserting.

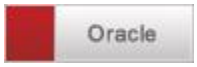
New Row Identification Method: method used to identify that records in source are not currently recorded in the target table.

Existing Data Selection Hint: database-compliant hint to be used for the existing data select statement.

Include Update Statement: includes an update statement in the procedure to update changing rows in the EDW 3NF Object. If this option is chosen, then the **Update Changed rows only** option is available.

Update Hint: enter a database compliant hint to be used in the UPDATE statement. This is an Oracle and SQL Server only option. Defaults can be configured in **Tools/Options/Default Update Procedure Options**.

Default is TABLOCK.



Default is APPEND.

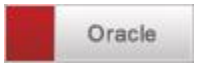
Update Changed Rows Only: uses change detection to work out what rows require updating. Choosing this option, enables the **Change Row identification Method.**

Change Row Identification Method: method used to identify that records in source have changed from what is currently recorded in the target table.

Existing Data Selection Hint: database-compliant hint to be used for the existing data select statement.

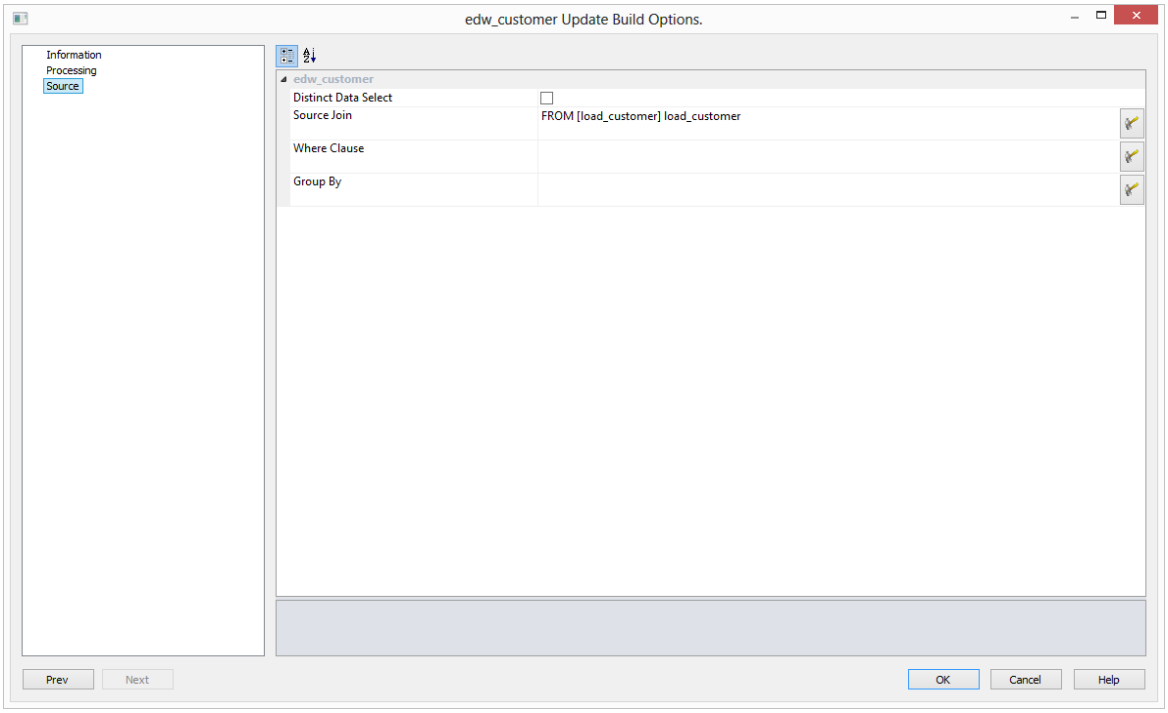
Merge Hint: enter a database hint to be used in the MERGE statement. This is an Oracle and SQL Server only option. Defaults can be configured in **Tools/Options/Default Update Procedure Options**.

Default is TABLOCK.



Default is APPEND.

SOURCE TAB



Distinct Data Select: ensures duplicate rows are not added to the EDW 3NF Object. This is achieved by the word DISTINCT being added to the source select in the update procedure. The default for this field is off.

Source Join: The From clause, including Source Join information.

Where Clause: The Where clause.

Group By: The Group By clause.

Simple Join

A simple join only returns rows where data is matched in both tables. So for example if table A has 100 rows and table B has a subset of 24 rows. If all the rows in table B can be joined to table A then 24 rows will be returned. The other 76 rows from table A will not be returned.

Outer Join

The outer join returns all rows in the master table regardless of whether they are found in the second table. So if the example above was executed with table A as the master table then 100 rows would be returned. 76 of those rows would have null values for the table B columns.

Note: When WhereScape RED builds up an outer join, it needs to know which table is the master table and which is subordinate. Select the join column from the master table first. In the example screen above the table 'load_order_header' has had its column chosen and the column for the table 'load_order_line' is currently being chosen. This will result in the 'load_order_header' table being defined as the master, as per the example statement above. The results of this example select are that a row will be added containing order information regardless of whether a corresponding load_order_line entry exists.

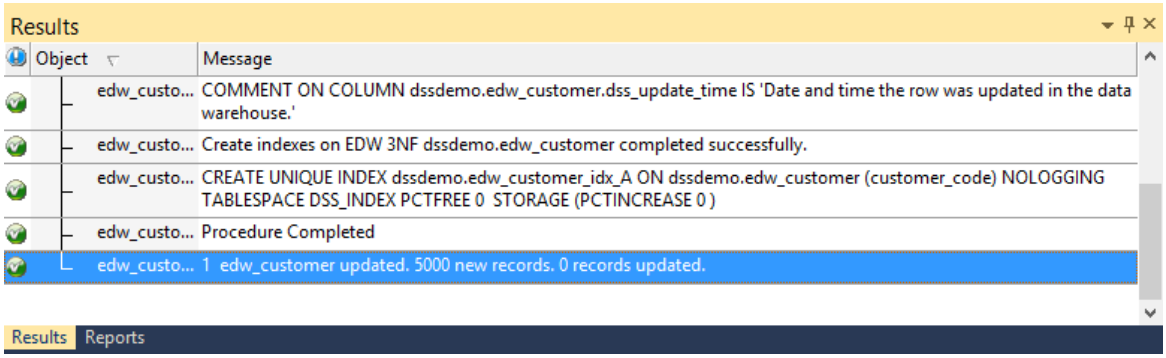
NOTE: When upgrading from a RED version before 6.8.2.0 and moving existing objects to a target location, all procedures that reference those objects will need to be rebuilt.
Any **FROM** clauses will also need to be manually regenerated for the table references to be updated to the new [TABLEOWNER] form.

Building and Compiling the Procedure

Once the relevant options are completed, click **OK**. The procedure will be built and compiled. If the compile fails an error will be displayed along with the first few lines of error messages. Compile fails typically occur when the physical creation of the table was not done. If the compile fails for some other reason the best approach is to use the procedure editor to edit and compile the procedure.
The procedure editor will highlight all the errors within the context of the procedure.
Once the procedure has been successfully compiled it can either be executed interactively or passed to the scheduler.

Indexes

By default, a number of indexes will be created to support each EDW 3NF table. These indexes will be added once the procedure has been built.
An example of the type of indexes created is as follows:



Additional indexes can be added, or these indexes changed. See the chapter on indexes for further details.

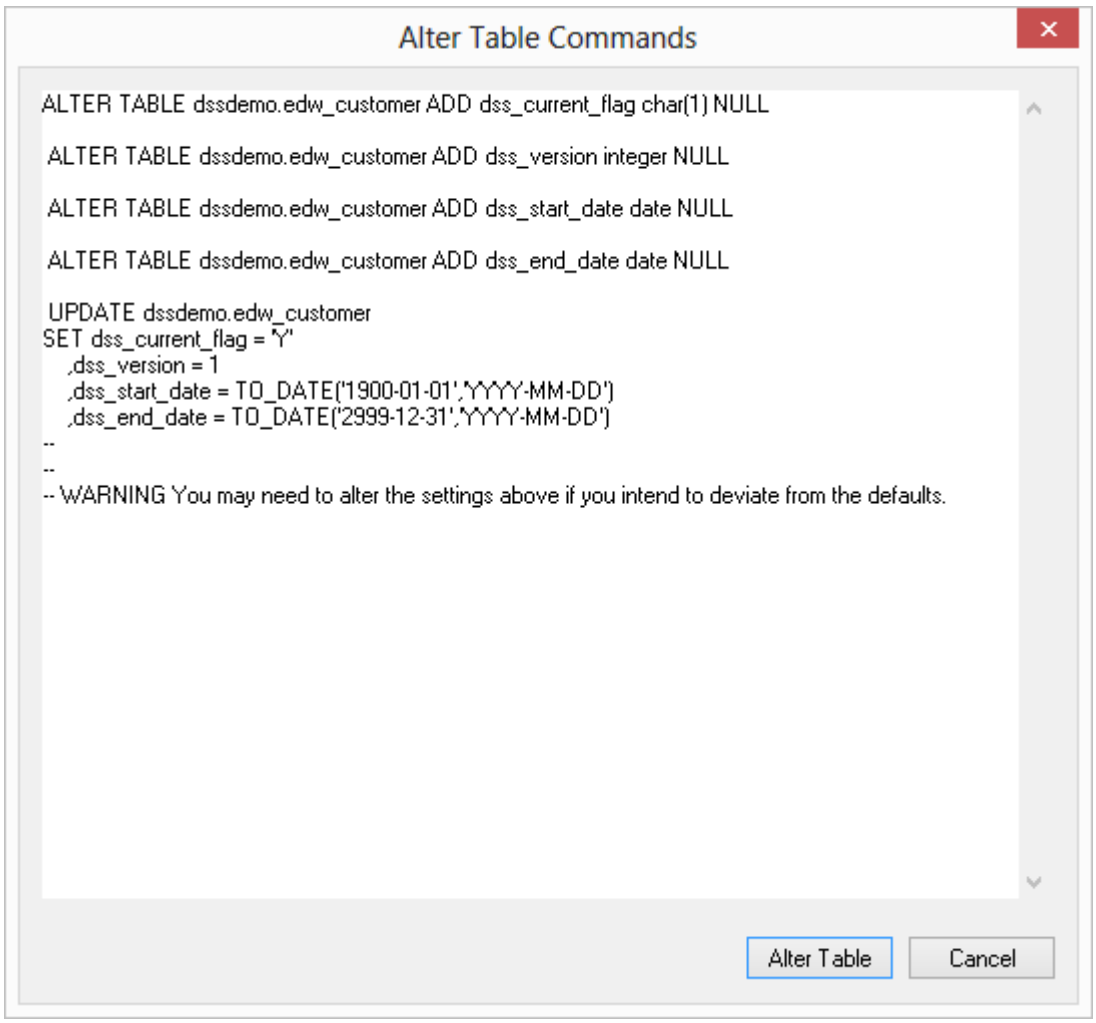
Converting an existing EDW 3NF Table to an EDW 3NF History Table

To convert an EDW 3NF table to an EDW 3NF history table:

- Change the table type to **History** on the Properties dialog.
- Select **(Build Procedure...)** from the **Update Procedure** drop-down list and click **OK**.

If the existing EDW 3NF table is **NOT** to be dropped and recreated, then the following process should be followed:

1. Click **OK** on the Procedure Type dialog.
2. Click **OK** on the Business Key dialog.
3. Click **Alter** on the Adding Additional Columns dialog.
4. Click **Alter Table** on the Alter Table Commands dialog:



Note: The SQL in this dialog can be edited if required.

5. Click **OK** on all remaining dialogs.

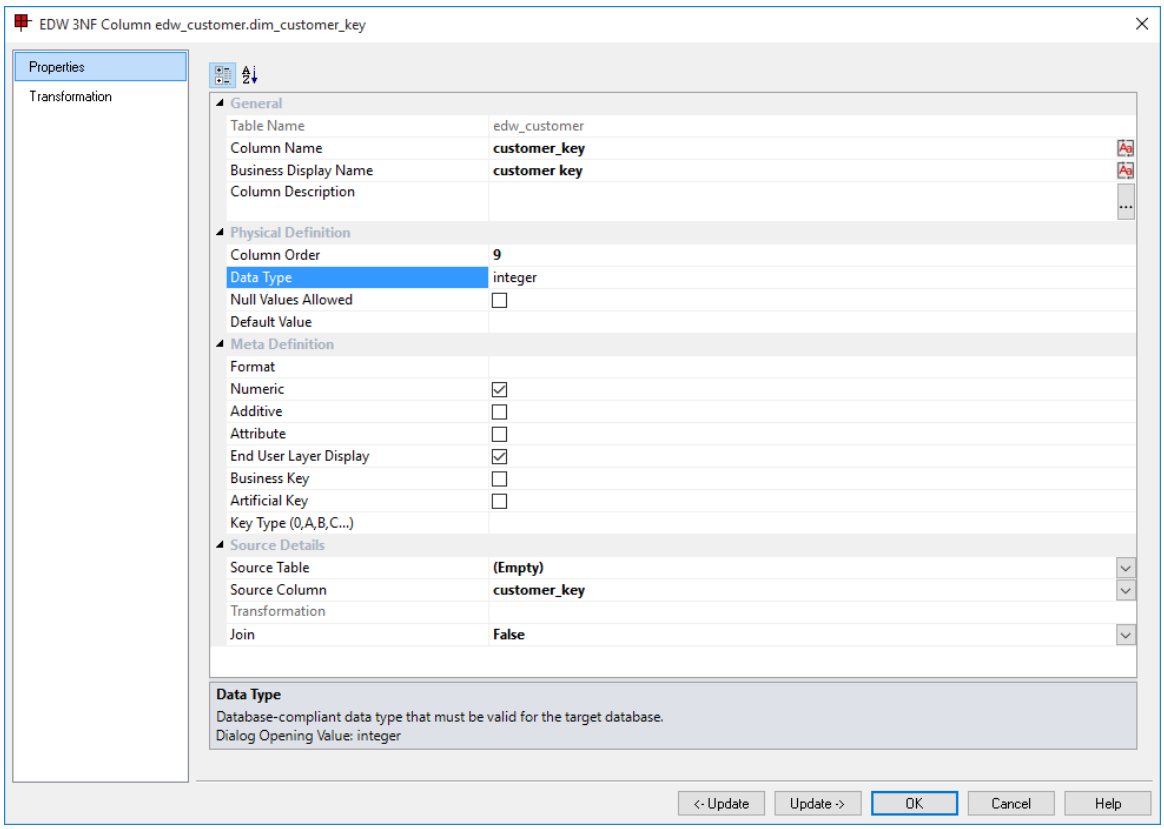
EDW 3NF TABLE ARTIFICIAL KEYS

By default, EDW 3NF tables in WhereScape RED do not have an artificial (surrogate) key. Artificial keys can be added manually.
The quickest way to do this is to add an extra column to the EDW 3NF table by using either **Add Column** or **Copy Column**.
Edit the properties of the new column to have the correct name and order, source table and column, datatype, key type and flags.

Specifically:

- The **Column Name** and **Source Column** should be the same.
- The **Source Table** should be empty.
- The **Data Type** should be:
  - DB2: integer generated by default as identity (start with 1, increment by 1)
  - Oracle: integer
  - SQL Server: integer identity(0,1)
- The **Key Type** should be 0.
- Only the **Numeric** and **Artificial Key** flags should be set on.

The following example shows a manually added artificial key column:



The artificial key for an EDW 3NF table is set via a sequence in Oracle and an identity column in SQL Server and DB2. This artificial key normally, and by default, starts at one and progresses as far as is required.

A WhereScape standard for the creation of special rows in the EDW 3NF tables is as follows:

Key value	Usage
1 upwards	The standard artificial keys are numbered from 1 upwards, with a new number assigned for each distinct EDW 3NF table record.
0	Used as a join to the EDW 3NF table when no valid join existed. It is the convention in the WhereScape generated code that any EDW 3NF table business key that either does not exist or does not match is assigned to key 0.
-1 through -9	Used for special cases. The most common being where an EDW 3NF table is not appropriate for the record. A new key is used rather than 0 as we want to distinguish between records that are invalid and not appropriate.

Key value	Usage
-10 backward	Pseudo records. In many cases, we must deal with different granularities in our data. For example, we may have a table that contains actual sales at a product SKU level and budget information at a product group level. The product table only contains SKU based information. To be able to map the budget records to the same table, we need to create these pseudo keys that relate to product groups. The values -10 and backwards are normally used for such keys.

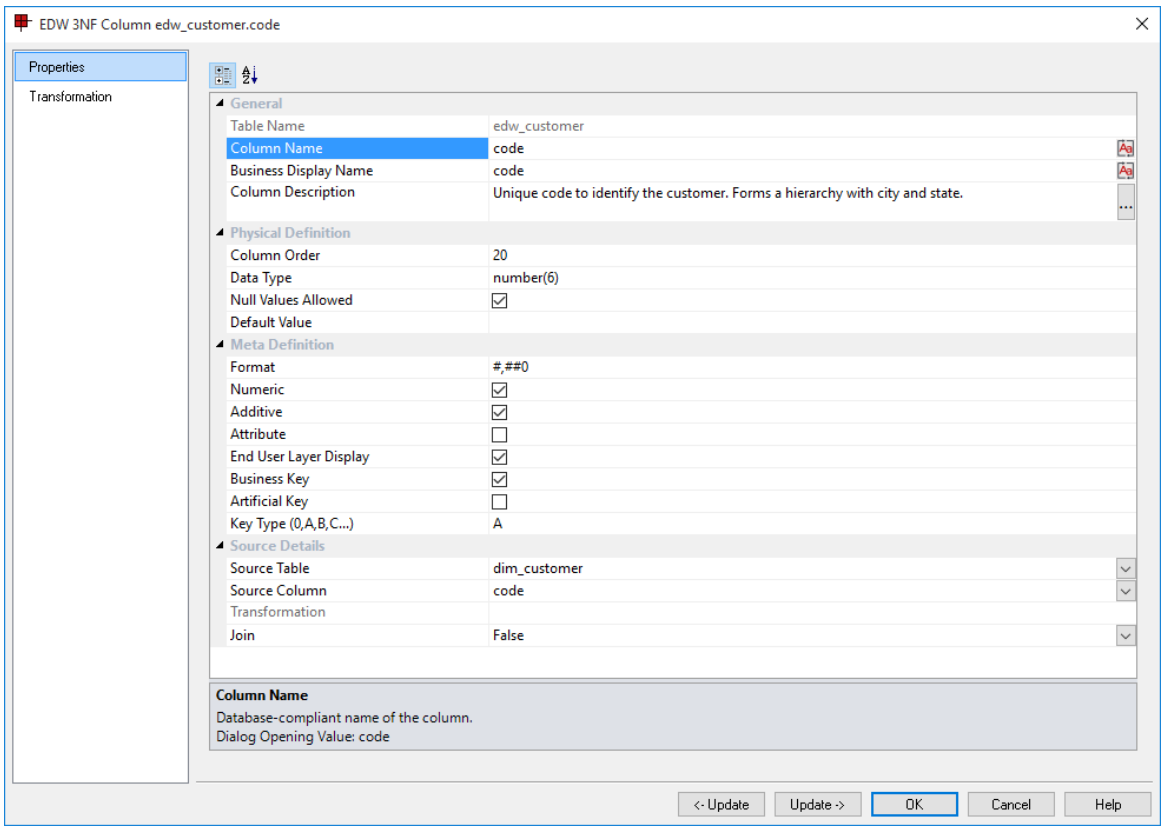
Note: To have a surrogate key auto added for EDW 3NF tables, see ***Global Naming of Key Columns*** (on page 95).

EDW 3NF TABLE COLUMN PROPERTIES

Each EDW 3NF table column has a set of associated properties. The definition of each property is described below:
If the **Column name** or **Data type** is changed for a column then the metadata will differ from the table as recorded in the database.
Use the **Validate/Validate Table Create Status** menu option or the right-click menu to compare the metadata to the table in the database.
A right-click menu option of **Alter table** is available when positioned on the table name after the validate has completed. This option will alter the database table to match the metadata definition.

TIP: If a database table's definition is changed in the metadata then the table will need to be altered in the database.
Use the **Validate/Validate Table Create Status** to compare metadata definitions to physical database tables. The option also provides the ability to alter the database table, through a pop-up menu option from the validated table name.

A sample **Properties** screen is as follows:



TIP: The two special update keys allow you to update the column and step either forward or backward to the next column's properties.
ALT-Left Arrow and **ALT-Right Arrow** can also be used instead of the two special update keys.

Table Name

Database-compliant name of the table that contains the column. [Read-only].

Column Name

Database-compliant name of the column. Typically, column-naming standards exclude the use of spaces etc. A good practice is to only use alphanumerics, and the underscore character. Changing this field alters the table's definition.

Note: A case conversion button on the right converts the text between different cases: UPPER CASE, Capitalized Case and lower case. The mode cycles to the next case in the sequence each conversion.

Business Display Name

Name that the business uses to refer to the column, which is included in the RED-generated documentation and can be used in the end user layer of other tools. [Does NOT affect the physical database table]. As such it is a free form entry and any characters are valid.

Note: A case conversion button on the right converts the text between different cases: UPPER CASE, Capitalized Case and lower case. The mode cycles to the next case in the sequence each conversion.

Column Description

This field contains the description for the column. It may be a description from a business user's point of view. This field might additionally contain information on where and how the column was acquired. For example, if the column is sourced from multiple tables or is a composite or derived column then this definition would normally describe the process used to populate the column. This field is used in the documentation and is available via the view **ws_admin_v_dim_col .** This field is also stored as a comment against the column in the database.

Column Order

Numeric value that controls the relative order of columns in the database create statement. The lowest numbered column will appear first in the table. Although this affects the physical table definition no action will be taken unless the table is re-created in the database. The columns can be re-numbered based on the existing order by choosing the **Respace Order Number** pop-up menu option when positioned over any column in the table. This action will number the columns in increments of 10 starting at 10. In addition to a simple change of the order field, the column order can be changed by first displaying the columns in the middle pane and then using drag and drop to move the columns around. This drag and drop process will automatically renumber the columns as required.

Data Type

Database-compliant data type that must be valid for the target database. Typical data types for Oracle are integer, number, char(), varchar2() and date. For SQL Server common types are integer, numeric, varchar() and datetime. See the database documentation for a description of the data types available. Changing this field alters the table's definition.

Null Values Allowed

Determines whether the table column can hold NULL values or whether a value is always mandatory.

Default Value

Initial value that is assigned to the column when a row is inserted into the table but no value is specified for the column.

Format

Optional format mask that can be used in end user tools. [Does NOT affect the physical database table]. As such it is a free form entry and any characters are valid. Typically format masks are only used on numeric fields. Example: #,###0.00. It is not worth the effort of populating this field unless it can be utilized by the end user tools in use.

Numeric

Indicates whether the table column holds values that are numeric. This is normally only relevant for fact tables. It does not affect the physical table definition, but rather provides input to the view **ws_admin_v_dim_col** which can be used to assist in the population of an end user tool's end user layer. The use of this field is not relevant unless it can be utilized by the end user tools.

Additive

Indicates whether the table column holds values that are additive. This implies that the column can be summed when performing data grouping in a query. This is normally only relevant for fact tables. It does not affect the physical table definition, but rather provides input to the view **ws_admin_v_dim_col** which can be used to assist in the population of an end user tool's end user layer. The use of this field is not relevant unless it can be utilized by the end user tools.

Attribute

Indicates whether the table column holds values that are descriptive, and/or are used for grouping/summing. An attribute is defined as a column that is non factual. For example we may have an order number, or a invoice number stored in the fact table. Such columns are considered attributes, rather than facts. This checkbox is therefore normally only relevant for fact tables. This checkbox does not affect the physical table definition, but rather provides input to the view **ws_admin_v_dim_col** which can be used to assist in the population of an end user tools end user layer. The use of this field is not relevant unless it can be utilized by the end user tools.

End User Layer Display

Indicates whether the table column is available/visible to end users. If set the documentation will include the column in the glossary and in the user documentation. It is also used to decide what columns appear in the view **ws_admin_v_dim_col.** Typically columns such as the artificial key would not be enabled for end user display.

Business Key

Indicates whether the column is part of the BUSINESS key, which is defined during the update procedure build. [Normally maintained automatically]. Multiple columns can form the primary business key.

Artificial Key

Indicates whether the column is the artificial key. Only one artificial key is supported. This indicator is set by WhereScape RED during the initial drag and drop creation of a table, and should not normally be altered.

Key Type

Key type that is assigned and used when generating the table's update procedure and indexes. [Normally maintained automatically]. It can be altered here, but this should only be done if the consequences are fully understood and tested.
The supported values are:

Key type	Meaning
0	The artificial key. Set when the key is added during drag and drop table generation.
1	Component of all business keys. Indicates that this column is used as part of any business key.
A	Indicates that the column is part of the primary business key. Set whenever a business key is defined as part of an update procedure generation.
B-Z	Indicates that the column is part of a secondary business key. Only used during index generation and not normally set.

Source Table

Identifies the source table where the column's data comes from. This source table is normally a load table within the data warehouse. If the column was sourced from multiple tables, then the normal practice is to record one of the tables in this field and a comment listing all of the other tables in the **Source Strategy** field. This field is used when generating a procedure to update the EDW 3NF table. It is also used in the track back diagrams and in the documentation.

Source Column

Identifies the source column where the column's data comes from. Such a column is normally a load table column, which in turn may have been a transformation or the combination of multiple columns. For previous value managed columns, the source column is the column in the table whose previous value is being recorded.

Transformation

Transformation. [Read-only].

Join

Indicates whether the table column is used in a table join. [Normally maintained automatically but can be optionally changed to override the default join logic used in the generated update procedure]. The **Source Table** and **Source Column** fields will provide the other EDW 3NF table's side of the join. The options for this field are: False, True, Manual and Pre Join.

Setting this field to Manual changes the way the dimension table is looked up during the update procedure build. It allows you to join the dimension manually in the Cursor mapping dialog (where the 'Where' clause is built).

Setting this field to Pre Join activates the **Pre Join Source Table** field and allows you to select a table from the drop-down list.

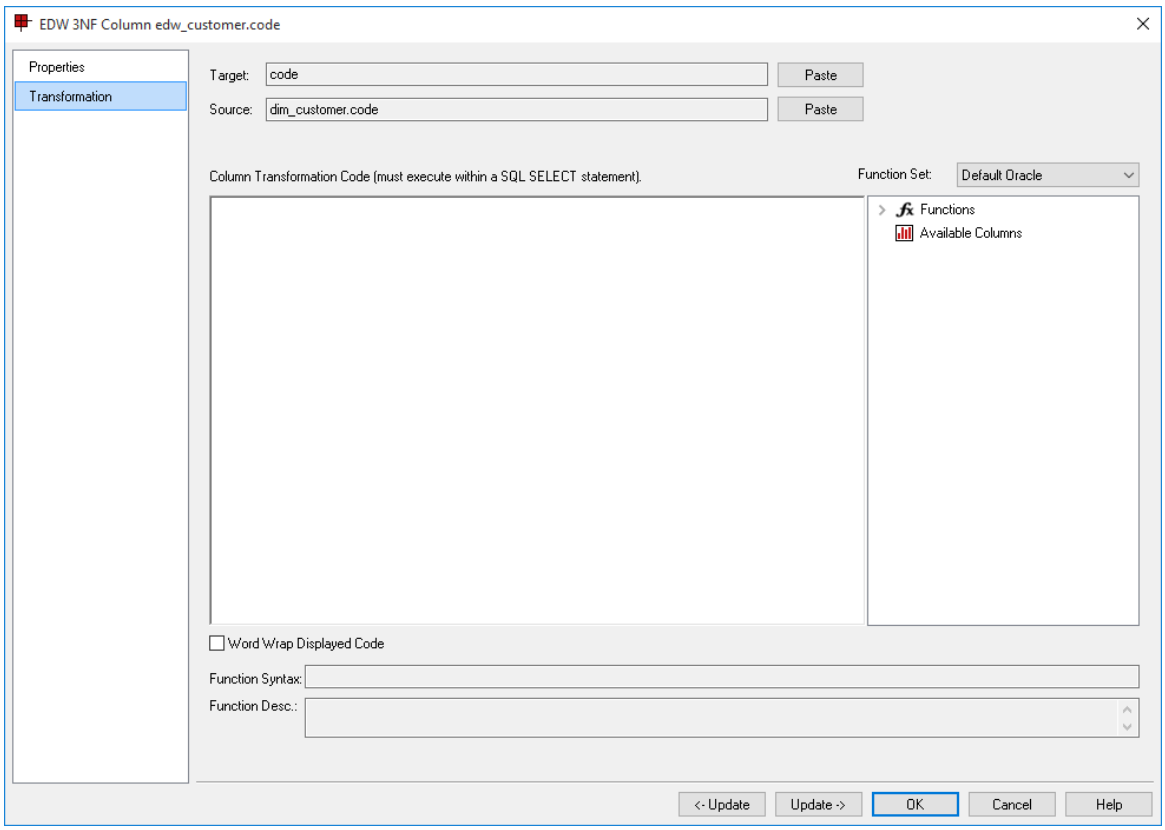
Pre Join Source Table

Indicates the table from which the pre joined column was sourced. When the Join option is set to False, this field becomes inactive. When the Join option is set to True or Manual, this field is set to the current table name. When the Join option is set to Pre Join, then you can select the required table from the drop-down list.

EDW 3NF TABLE COLUMN TRANSFORMATIONS

Each EDW 3NF table column can have a transformation associated with it. The transformation will be included in the generated procedure and will be executed as part of the procedure update. The transformation must therefore be a valid SQL construct that can be included in a **Select** statement.
For example, we could have a transformation of 'load_order_line.qty * 0.125' to calculate a tax column of 12.5%.

Click the **Transformation** tab to enter a transformation.
The transformation screen is as follows:



NOTE: Transformations are only put into effect when the procedure is re-generated.

Microsoft Analysis Services 2005+ Tabular Mode Tables: For Tabular Mode table column transformations, **Default DAX** is the only applicable Function Set for **after load** transformations.

See ***Transformations*** (on page 650) for more details.

WhereScape®

CHAPTER 13
DATA VAULTS

To import a Data Vault into RED, WhereScape suggests using the following process:

1. Design the Data Vault in WhereScape 3D,
2. Import the Data Vault from 3D to RED, and
3. Physically create the tables in RED.

For more information on Data Vault design, refer to *Building a Scalable Data Warehouse With Data Vault 2.0* by *Daniel Linstedt and Michael Olschimke*.

IN THIS CHAPTER

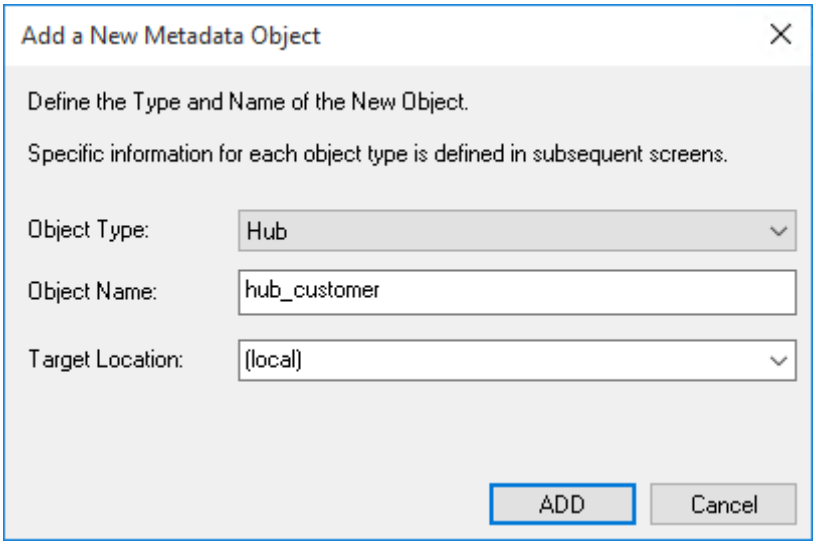
BUILDING A DATA VAULT

WhereScape recommends using WhereScape 3D to design your Data Vault, but this can also be completed manually in RED.

Hubs, **Links** and **Satellites** are often sourced from one table in the base application. The process for building a Data Vault table begins with the drag and drop of the load or stage table that contains the bulk of the table's information.

Drag and Drop

- Click on the **Hub**, **Link** or **Satellite** group in the left pane. The middle pane will display a list of all existing tables in the current project.
- Browse to the Data Warehouse via the Browse/Source Data menu option in the Browser Pane (right pane).
- Drag the load or stage table, that contains the bulk of the Data Vault table columns, into the middle pane. Drop the table anywhere in the pane.
- The new object dialog box will appear and will identify the new object as a Data Vault table and will provide a default name based on the load or stage table name.
- Either accept this name or enter a new name for the Data Vault table and click ADD to proceed:



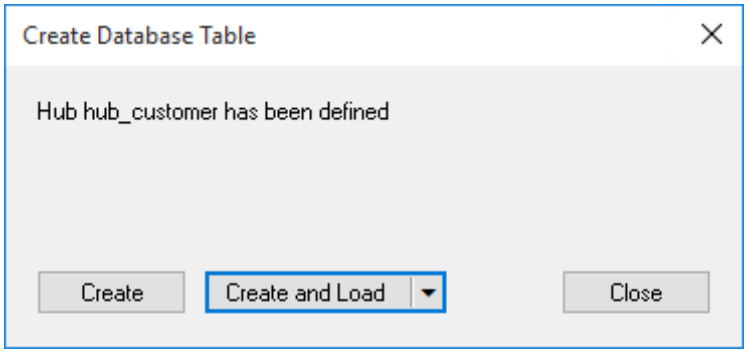
Data Vault Table Properties

The table Properties dialog for the new table is now displayed. If required, the Data Vault table can be changed to be a history table by choosing History from the table type drop-down list on the right side of the dialog. **Hub** and **Link** tables should usually be set to *Detail* and **Satellite** tables should be set to *History*. History tables hold a traceable history of changes of specific attributes (change detection columns) for a stored item, detail tables only record the current snapshot of an item.

If you wish to create the update procedure now, select the **(Build Procedure...)** option from the **Update Procedure** drop-down, click OK in the Properties dialog.

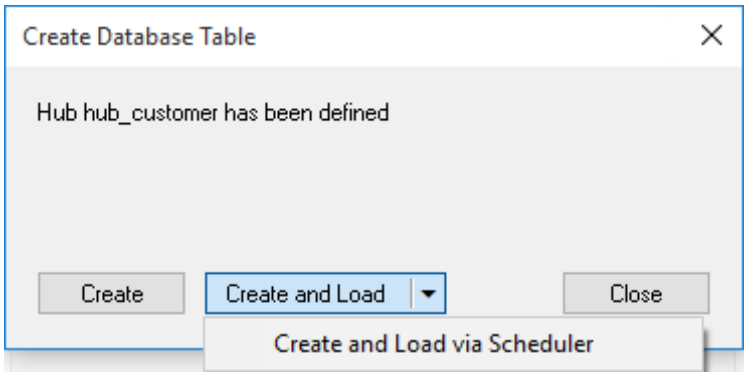
Create and Load

If you chose to build the update procedure the following dialog appears after clicking OK on the Properties page. This dialog asks if you want to create the Data Vault table in the database and execute the update procedure.



If you are satisfied with the columns that will be used and do not wish to add any columns you can select the **Create and Load** button. Alternatively, the **Create** button creates the table in the repository but does not execute an update, allowing you to change columns before loading data into the table.
If **Create** or **Create and Load** is selected and a new procedure creation was chosen proceed directly to the ***Generating the Data Vault Update Procedure*** (on page 456) section.

Note: It is possible to create and load the table via the Scheduler; by selecting this option from the drop-down list on the **Create and Load** button:



If you have additional columns to add or columns to delete, then select **Close** and proceed as follows:

Deleting and Changing columns

The columns defined for the Data Vault table will be displayed in the middle pane. It is possible to delete any unwanted columns by highlighting a column name or a group of names and pressing **Delete**.
The name of a column can also be changed by right-clicking on the column and choosing **Properties** to edit its properties.
Any new name must conform to the database naming standards. Good practice is to use alphanumerics and the underscore character.
See the section on column properties for a more detailed description on what the various fields mean.

 **TIP**: When prototyping, and in the initial stages of an analysis area build it is best not to remove columns, nor to change their names to any great extent. This type of activity is best left until after end users have used the data and provided feedback.

Adding additional columns

With the columns of the Data Vault table displayed in the middle pane, this pane is considered a drop target for additional columns.
It is a simple matter to select columns from other load and/or stage tables and drag these columns into the middle pane. The source table column in the middle pane shows where each column was dragged from.
There are a number of WhereScape RED ancillary columns that do not have a source table. These columns have been added by WhereScape RED or 3D, and are added depending on earlier choices. A description of these columns follows.

Column name	Description
dss_start_date	Used Data Vault table record came into existence. It is used to ascertain which Data Vault table record should be used when multiple are available.
dss_end_date	Used for history tables, for example Satellite tables. This column provides a date time stamp when the Data Vault table record ceased to be the current record. It is used to ascertain which Data Vault table record should be used when multiple are available.
dss_current_flag	Used for history tables. This flag identifies the current record where multiple versions exist.
dss_source_system	Added to support history tables that cannot be fully conformed, and the inclusion of subsequent source systems. See the ancillary settings section for more details.
dss_version	Used for history tables. This column contains the version number of a Data Vault history tables record. Numbered from 1 upwards with the highest number being the latest or current version. It forms part of the unique constraint for the business key of a Data Vault history table.
dss_update_time	Indicates when the record was last updated in the data warehouse.
dss_create_time	Indicates when the record was first created in the data warehouse.

Creating the table

Once the Data Vault table has been defined in the metadata we need to physically create the table in the database.
This is done by right-clicking on the Data Vault table in the left pane and selecting **Create (ReCreate).**
The results of the creation are shown in the Results pane. For the initial create no indexes will be defined.
If the table was not created, then ascertain and fix the problem. A common problem is a 'Duplicate column' where a column has the same name in two of the source tables.
The best way to find such a column is to double-click on the list heading **Column name**; which will sort the column names into alphabetical order.
A double-click on the **Order** column will sort the columns back into their create order.

GENERATING THE DATA VAULT UPDATE PROCEDURE

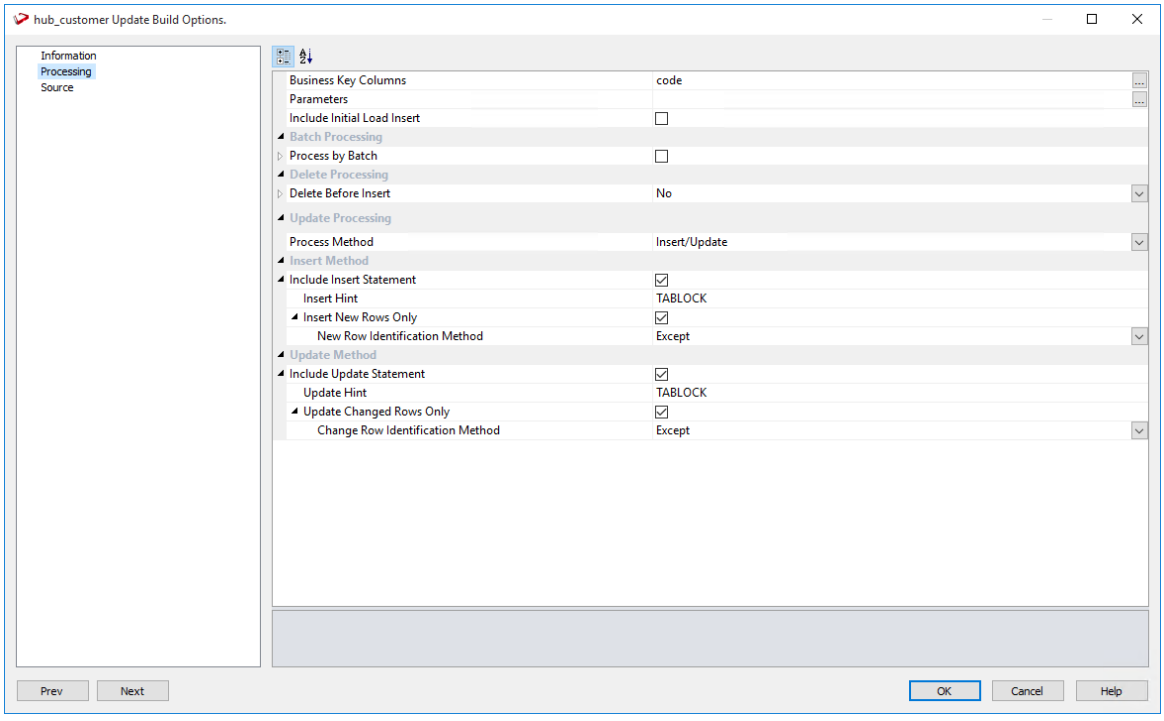
Once a Data Vault Object has been defined in the metadata and created in the database, an update procedure can be generated to handle the joining of any tables and the update of the Data Vault Object.

NOTE: There is no code generation specific to Data Vault object types, RED generates EDW 3NF history and non-history procedures for Data Vault objects. WhereScape recommends importing Data Vault models from WhereScape 3D to establish correct relationships between Data Vault objects in RED.

Generating a Procedure

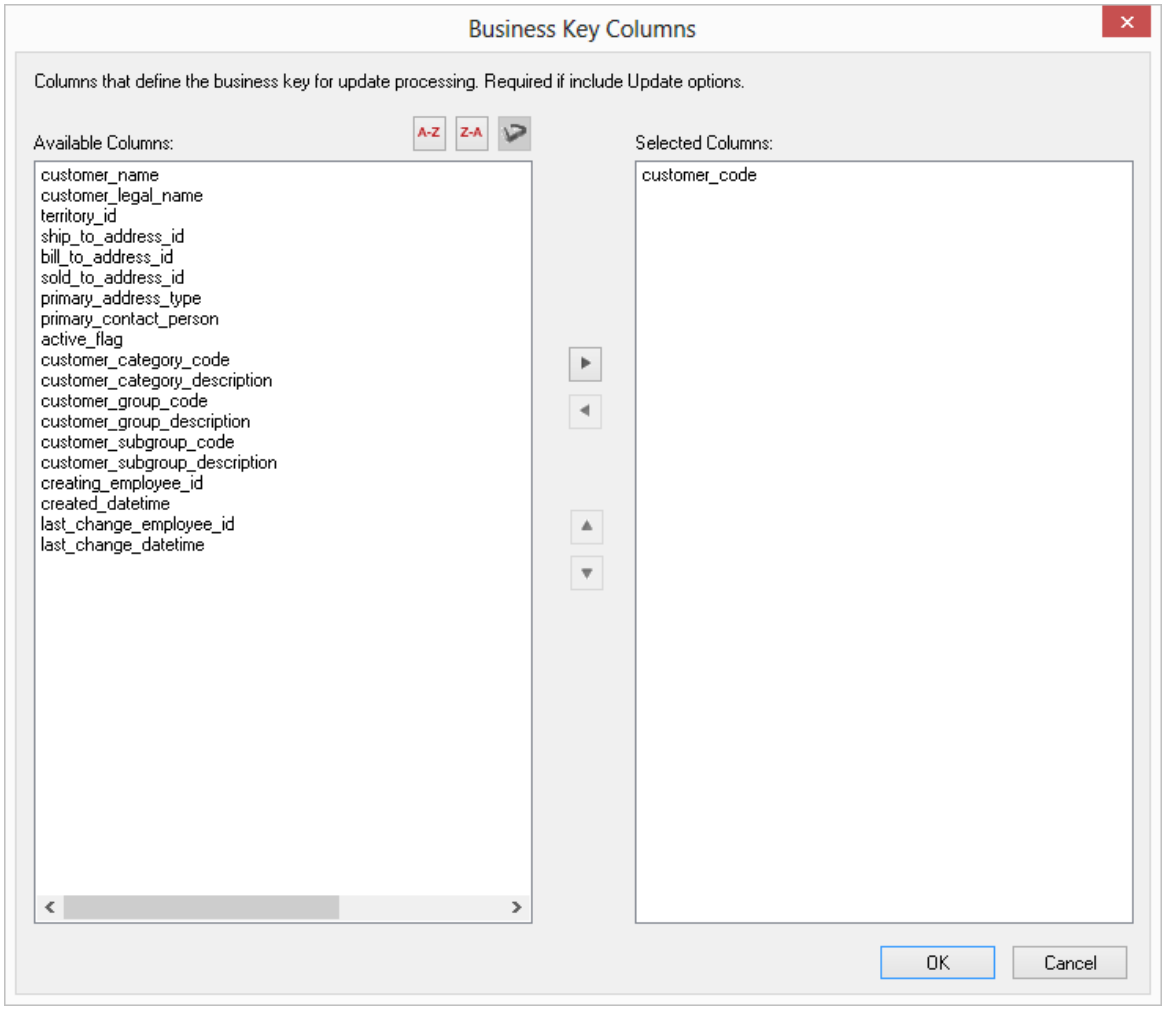
- To generate a procedure, right-click on the Data Vault Object in the left pane and select **Properties**.
- Click on the **Rebuild** button to start the process of generating the new procedure.
- A series of options are available.

PROCESSING TAB



Business Key Columns

Columns that define the business key for update processing. Required for include Update options. Clicking on the ellipsis button will bring up the Business Key selection screen.



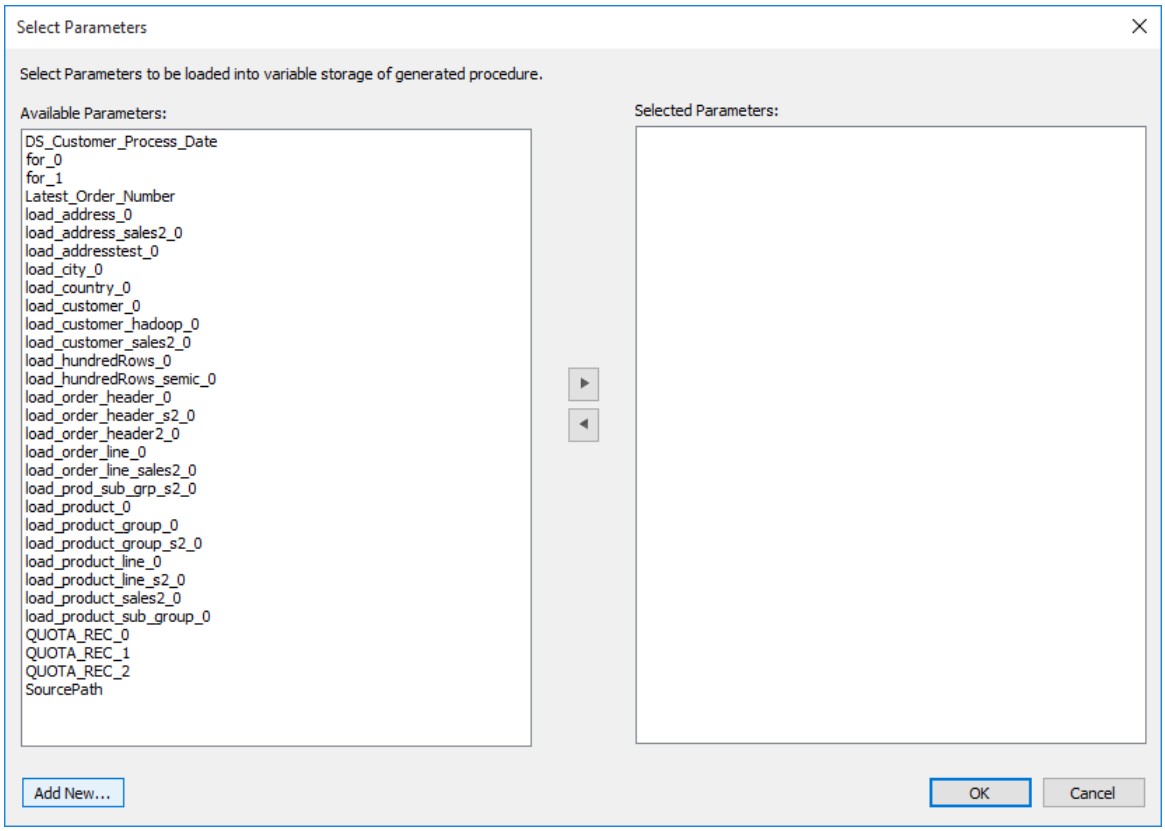
A business key can be made up of multiple columns, but it must provide a unique identifier. Where multiple columns separately uniquely identify rows in the Data Vault Object, choose one to act as the primary business key. For example, a source table may have a unique constraint on both a product code and a product description. Therefore, the description as well as the code must be unique. It is possible to combine the two columns, but the normal practice would be to choose the code as the business key.

TIP1: Use the column name ascending/descending buttons to sort column names. To revert to the meta column order, click on the meta column order button.

TIP2: NULL Values - none of the columns chosen as the business key should ever contain a NULL value.

Parameters

Any parameters selected are included in the generated update procedure as variables. The procedure will include code to retrieve the value of the parameter at run time and store it in the declared variable.
Clicking on the ellipsis button will bring up the Parameters selection screen.

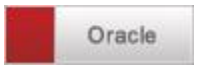


The variables can also be used in column transformations and in the from/where clause for the update procedure. Some databases have a 30 character limit for variable names. WhereScape RED ensures the variables added for any parameters are less than 30 characters long by creating variable names in the form v_ followed by the first 28 characters of the parameter name.
For example, a parameter called MINIMUM_ORDER_NUMBER_SINCE_LAST_SOURCE_LOAD will be available as the variable v_MINIMUM_ORDER_NUMBER_SINCE_L.

TIP1: WhereScape RED parameters should be unique within the first 28 characters to avoid conflicting variables names.

TIP2: If the desired parameter doesn't exist in the metadata yet, a new parameter can be added by clicking on the **Add New** button on the bottom leftmost corner of the Select Parameters dialog.

See ***Parameters*** (on page 139) for more information on WhereScape RED Parameters.



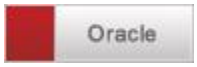
Enable Parallel DML - adds all the code required to the update procedure to enable Oracle parallel inserts. Default for this option is not set.

Insert Zero Key Record - Adds an insert statement for an unknown record with an artifical key of zero. Only applicable to tables with an artificial key.

Include Initial Load Insert: adds an additional insert statement to the update procedure that runs if the target Data Vault object is empty. The benefit of this is improved performance inserting into an empty table without performing any checks to see if rows already exist. The default for this field is off (i.e. an initial insert statement is not added to the procedure).

Process by Batch: allows users to select a column to drive data processing in a loop based on the distinct ordered values of the selected columns. The update procedure loops on this column and performs the delete, update and/or insert for each value. If the column chosen is a date datatype (date, datetime or timestamp), then the user is able to specify yearly, monthly, daily or column level looping. The default for this field is off (do not do batch processing).

Delete Before Insert: allows selection of how to process deletes. It enables a delete statement to be added to the update procedure before any update or insert statement. This is a particularly useful option for purging old data and for updates based on a source system batch number. If this option is selected, the following dialog is displayed:



Truncate Option: optional Oracle TRUNCATE clause which is appended to the truncate statement. Add a truncate option such as **REUSE/DROP STORAGE** by typing it in the truncate options box.

Issue Warning if a Delete occurs: this option sets the procedure to a warning state if any deletes occur.

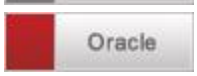
Delete Where Clause: the delete where clause is appended to the generated delete statement to constrain the rows deleted.

Process Method: Select between Insert/Update and Merge which allows you to use the Merge statement instead of two separate Insert and update statements.

Include Insert Statement: includes the insert statement in the procedure to insert new rows in the Data Vault Object.

Insert Hint: enter a database compliant hint to be used in the INSERT statement. This is an Oracle and SQL Server only option. Defaults can be configured in **Tools/Options/Default Update Procedure Options**.

Default is TABLOCK.



Default is APPEND.

Insert New Rows Only: uses change detection to work out what rows require inserting.

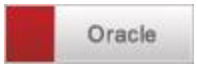
New Row Identification Method: method used to identify that records in source are not currently recorded in the target table.

Existing Data Selection Hint: database-compliant hint to be used for the existing data select statement.

Include Update Statement: includes an update statement in the procedure to update changing rows in the Data Vault Object. If this option is chosen, then the **Update Changed rows only** option is available.

Update Hint: enter a database compliant hint to be used in the UPDATE statement. This is an Oracle and SQL Server only option. Defaults can be configured in **Tools/Options/Default Update Procedure Options**.

Default is TABLOCK.



Default is APPEND.

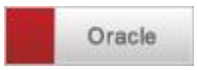
Update Changed Rows Only: uses change detection to work out what rows require updating. Choosing this option, enables the **Change Row identification Method.**

Change Row Identification Method: method used to identify that records in source have changed from what is currently recorded in the target table.

Existing Data Selection Hint: database-compliant hint to be used for the existing data select statement.

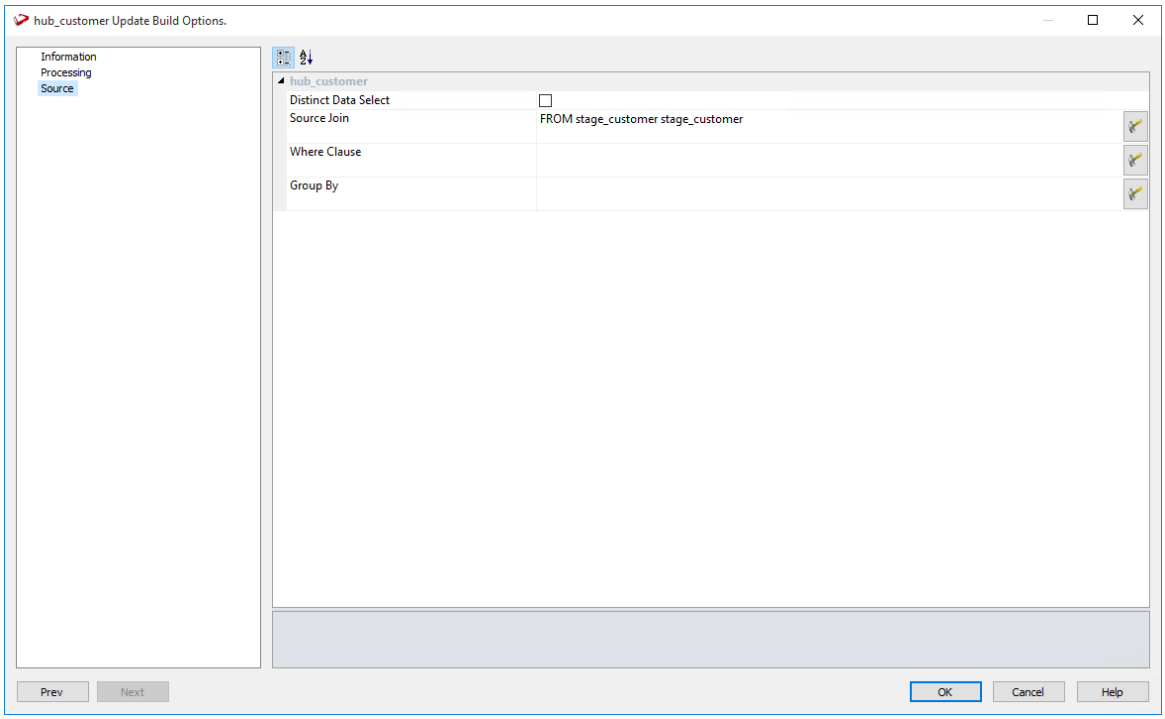
Merge Hint: enter a database hint to be used in the MERGE statement. This is an Oracle and SQL Server only option. Defaults can be configured in **Tools/Options/Default Update Procedure Options**.

Default is TABLOCK.



Default is APPEND.

SOURCE TAB



Distinct Data Select: ensures duplicate rows are not added to the Data Vault Object. This is achieved by the word DISTINCT being added to the source select in the update procedure. The default for this field is off.

Source Join: The From clause, including Source Join information.

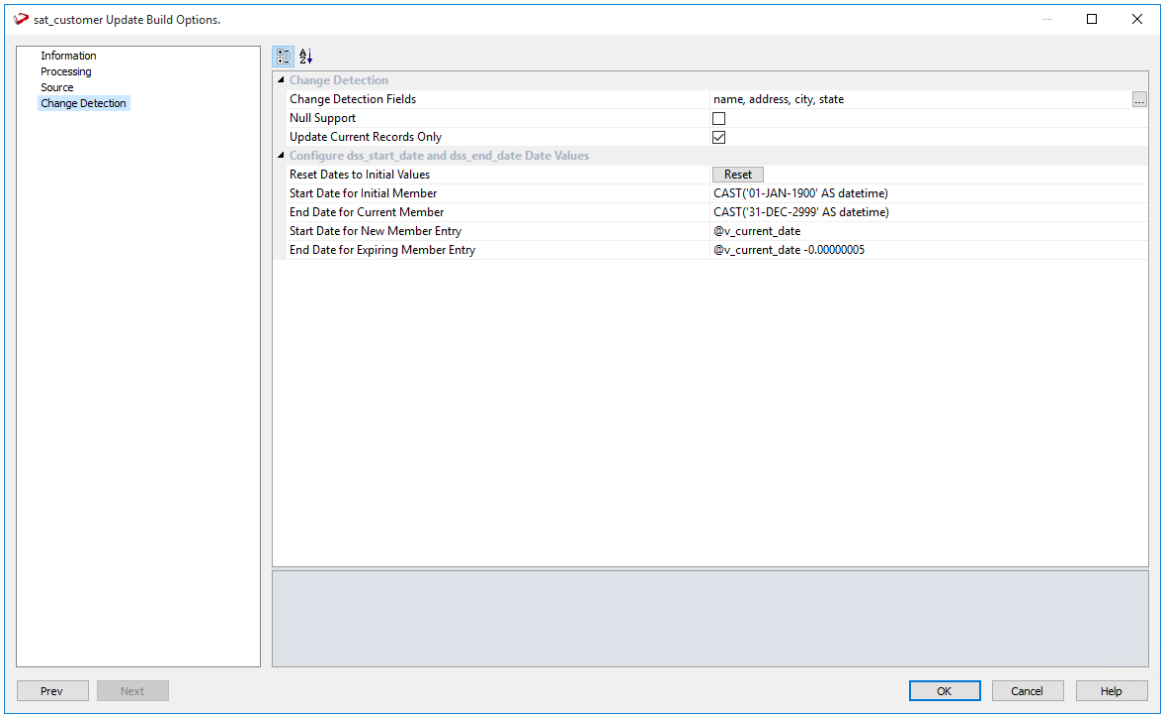
Where Clause: The Where clause.

Group By: The Group By clause.

NOTE: When upgrading from a RED version before 6.8.2.0 and moving existing objects to a target location, all procedures that reference those objects will need to be rebuilt.
Any **FROM** clauses will also need to be manually regenerated for the table references to be updated to the new [TABLEOWNER] form.

CHANGE DETECTION TAB

For *History* type tables, such as **Satellite** tables. A change detection field(s) must be selected for this table type.



Change Detection Fields - by clicking on the ellipsis button, this allows the user to select the change detection fields that are required for the dimension

Null Support - If this option is selected, the change detect column management will cater for null values in any change detect columns. If this option is not selected and null values are present, errors may occur running the update procedure. The default for this value if off (i.e. null are not catered for).

Update Current Records Only - The update current record only option will only apply changes to non-change detect columns on the current record. If this option is not selected, all past (non-current) rows will also be updated to reflect changes to non-change detect columns. The default for this value is on (i.e. only the current record is updated).

Reset Dates to Initial Values - Resets dss_start date and dss_end_date date values to original values.

Start Date for Initial Member - The start date for initial member field contains the start date for the first version of a particular business key. The value should be specified in an appropriate form, taking into account the default date format in the databases. The date may need to be explicitly cast to the current data type. The default value provided will usually be cast to the correct database and can be treated as a template. The default for this field is 1 January 1900.

End Date for Current Member - The end date for current member field contains the end date for the current version (the row with a current flag of Y and the maximum version number) of a particular business key. The value should be specified in an appropriate form, taking into account the default date format in the databases. The date may need to be explicitly cast to the current data type. The default value provided will usually be cast to the correct database and can be treated as a template. The default for this field is 31 December 2999.

Start Date for New Member Entry - The start date for new member entry field contains the start date for any subsequent rows added to the history table (not the first row for a particular business key i.e. not version 1). The value should be specified in an appropriate form, taking into account the default date format in the databases. The date may need to be explicitly cast to the current data type. The default value provided will usually be cast to the correct database and can be treated as a template. The default for this field is the current date and time.

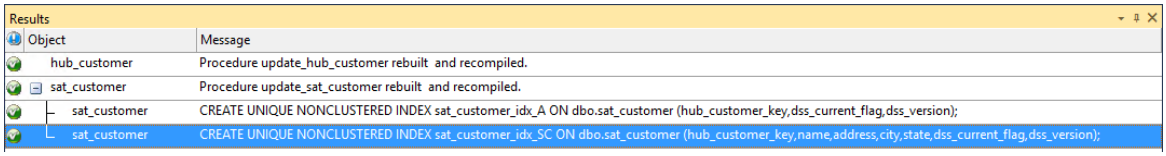
End Date for Expiring Member Entry - The end date for the expiring member entry field contains the end date for any rows updated no longer to no longer be the current row in the history table (i.e. rows that are replaced by a new current row). The value should be specified in an appropriate form, taking into account the default date format in the databases. The date may need to be explicitly cast to the current data type. The default value provided will usually be cast to the correct database and can be treated as a template. The default for this field is the current date and time less an arbitrary small amount (for SQL Server this is 0.00000005 of a day, or about 4 thousandth of a second).

Building and Compiling the Procedure

Once the relevant options are completed, click **OK**. The procedure will be built and compiled. If the compile fails an error will be displayed along with the first few lines of error messages. Compile fails typically occur when the physical creation of the table was not done. If the compile fails for some other reason the best approach is to use the procedure editor to edit and compile the procedure.
The procedure editor will highlight all the errors within the context of the procedure.
Once the procedure has been successfully compiled it can either be executed interactively or passed to the scheduler.

Indexes

By default a number of indexes will be created to support each Data Vault table. These indexes will be added once the procedure has been built.
An example of the type of indexes created is as follows:



Additional indexes can be added, or these indexes changed. See the chapter on indexes for further details.

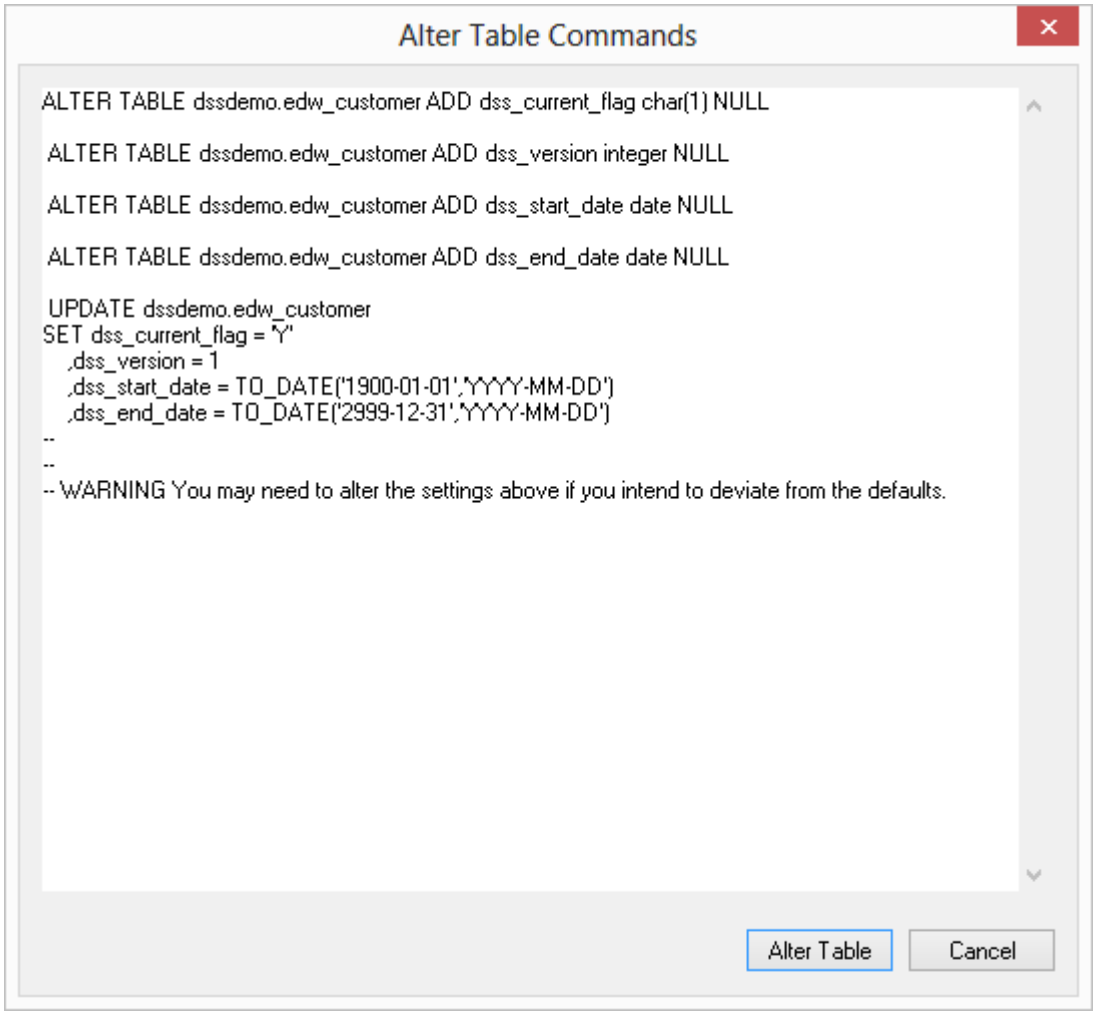
Converting an existing Data Vault table to a Data Vault History Table

To convert a Data Vault table to a Data Vault history table:

- Change the table type to **History** on the Properties dialog.
- Select **(Build Procedure...)** from the **Update Procedure** drop-down list and click **OK**.

If the existing Data Vault table is **NOT** to be dropped and recreated, then the following process should be followed:

1. Click **OK** on the Procedure Type dialog.
2. Click **OK** on the Business Key dialog.
3. Click **Alter** on the Adding Additional Columns dialog.
4. Click **Alter Table** on the Alter Table Commands dialog:



Note: The SQL in this dialog can be edited if required.

5. Click **OK** on all remaining dialogs.

DATA VAULT HASH KEYS

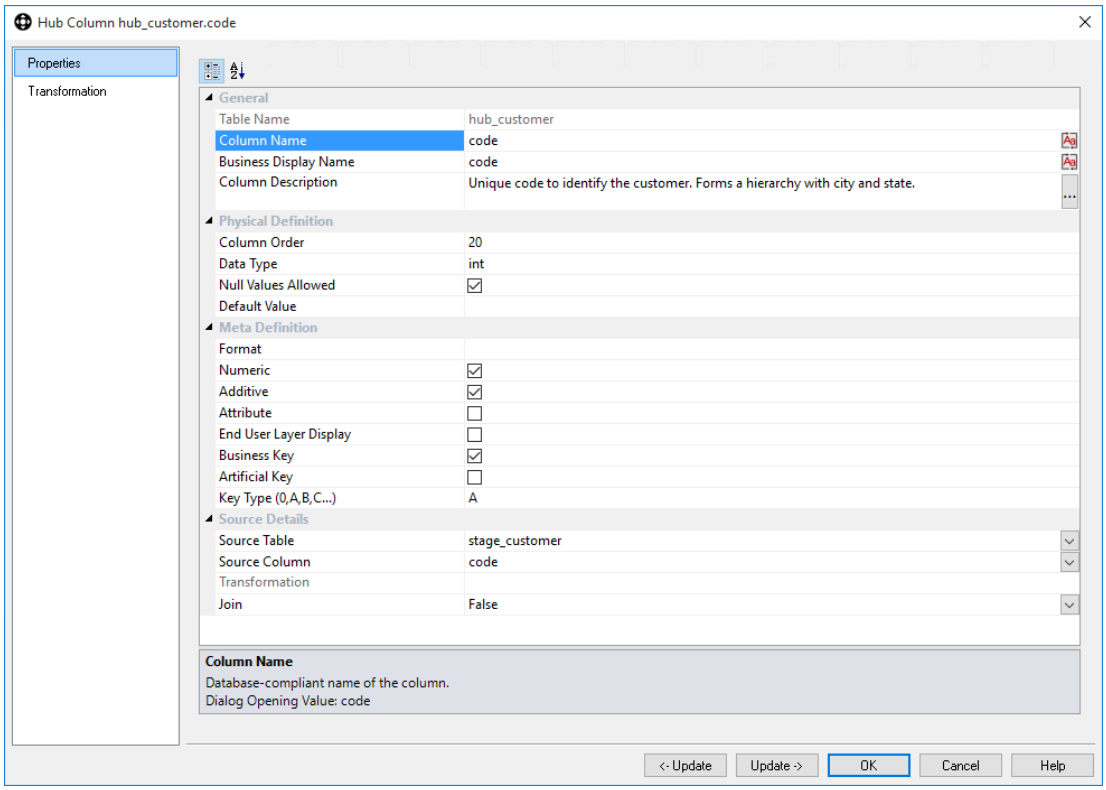
Data Vault tables in WhereScape RED should have a hash key. It is best practice to create these Hash columns in the Stage (source) table for each Data Vault table. For more information on transformation columns, see ***Transformations*** (on page 650).

DATA VAULT TABLE COLUMN PROPERTIES

Each Data Vault table column has a set of associated properties. The definition of each property is described below:
If the **Column name** or **Data type** is changed for a column then the metadata will differ from the table as recorded in the database.
Use the **Validate/Validate Table Create Status** menu option or the right-click menu to compare the metadata to the table in the database.
A right-click menu option of **Alter table** is available when positioned on the table name after the validate has completed. This option will alter the database table to match the metadata definition.

TIP: If a database table's definition is changed in the metadata then the table will need to be altered in the database.
Use the **Validate/Validate Table Create Status** to compare metadata definitions to physical database tables. The option also provides the ability to alter the database table, through a pop-up menu option from the validated table name.

A sample **Properties** screen is as follows:



TIP: The two special update keys allow you to update the column and step either forward or backward to the next column's properties.
ALT-Left Arrow and **ALT-Right Arrow** can also be used instead of the two special update keys.

Table Name

Database-compliant name of the table that contains the column. [Read-only].

Column Name

Database-compliant name of the column. Typically, column-naming standards exclude the use of spaces etc. A good practice is to only use alphanumerics, and the underscore character. Changing this field alters the table's definition.

Note: A case conversion button on the right converts the text between different cases: UPPER CASE, Capitalized Case and lower case. The mode cycles to the next case in the sequence each conversion.

Business Display Name

Name that the business uses to refer to the column, which is included in the RED-generated documentation and can be used in the end user layer of other tools. [Does NOT affect the physical database table]. As such it is a free form entry and any characters are valid.

Note: A case conversion button on the right converts the text between different cases: UPPER CASE, Capitalized Case and lower case. The mode cycles to the next case in the sequence each conversion.

Column Description

This field contains the description for the column. It may be a description from a business user's point of view. This field might additionally contain information on where and how the column was acquired. For example, if the column is sourced from multiple tables or is a composite or derived column then this definition would normally describe the process used to populate the column. This field is used in the documentation and is available via the view **ws_admin_v_dim_col .** This field is also stored as a comment against the column in the database.

Column Order

Numeric value that controls the relative order of columns in the database create statement. The lowest numbered column will appear first in the table. Although this affects the physical table definition no action will be taken unless the table is re-created in the database. The columns can be re-numbered based on the existing order by choosing the **Respace Order Number** pop-up menu option when positioned over any column in the table. This action will number the columns in increments of 10 starting at 10. In addition to a simple change of the order field, the column order can be changed by first displaying the columns in the middle pane and then using drag and drop to move the columns around. This drag and drop process will automatically renumber the columns as required.

Data Type

Database-compliant data type that must be valid for the target database. Typical data types for Oracle are integer, number, char(), varchar2() and date. For SQL Server common types are integer, numeric, varchar() and datetime. See the database documentation for a description of the data types available. Changing this field alters the table's definition.

Null Values Allowed

Determines whether the table column can hold NULL values or whether a value is always mandatory.

Default Value

Initial value that is assigned to the column when a row is inserted into the table but no value is specified for the column.

Format

Optional format mask that can be used in end user tools. [Does NOT affect the physical database table]. As such it is a free form entry and any characters are valid. Typically format masks are only used on numeric fields. Example: #,###0.00. It is not worth the effort of populating this field unless it can be utilized by the end user tools in use.

Numeric

Indicates whether the table column holds values that are numeric. This is normally only relevant for fact tables. It does not affect the physical table definition, but rather provides input to the view **ws_admin_v_dim_col** which can be used to assist in the population of an end user tool's end user layer. The use of this field is not relevant unless it can be utilized by the end user tools.

Additive

Indicates whether the table column holds values that are additive. This implies that the column can be summed when performing data grouping in a query. This is normally only relevant for fact tables. It does not affect the physical table definition, but rather provides input to the view **ws_admin_v_dim_col** which can be used to assist in the population of an end user tool's end user layer. The use of this field is not relevant unless it can be utilized by the end user tools.

Attribute

Indicates whether the table column holds values that are descriptive, and/or are used for grouping/summing. An attribute is defined as a column that is non factual. For example we may have an order number, or a invoice number stored in the fact table. Such columns are considered attributes, rather than facts. This checkbox is therefore normally only relevant for fact tables. This checkbox does not affect the physical table definition, but rather provides input to the view **ws_admin_v_dim_col** which can be used to assist in the population of an end user tools end user layer. The use of this field is not relevant unless it can be utilized by the end user tools.

End User Layer Display

Indicates whether the table column is available/visible to end users. If set the documentation will include the column in the glossary and in the user documentation. It is also used to decide what columns appear in the view **ws_admin_v_dim_col.** Typically columns such as the artificial key would not be enabled for end user display.

Business Key

Indicates whether the column is part of the BUSINESS key, which is defined during the update procedure build. [Normally maintained automatically]. Multiple columns can form the primary business key.

Artificial Key

Indicates whether the column is the artificial key. Only one artificial key is supported. This indicator is set by WhereScape RED during the initial drag and drop creation of a table, and should not normally be altered.

Key Type

Key type that is assigned and used when generating the table's update procedure and indexes. [Normally maintained automatically]. It can be altered here, but this should only be done if the consequences are fully understood and tested.
The supported values are:

Key type	Meaning
0	The artificial key. Set when the key is added during drag and drop table generation.
1	Component of all business keys. Indicates that this column is used as part of any business key.
A	Indicates that the column is part of the primary business key. Set whenever a business key is defined as part of an update procedure generation.
B-Z	Indicates that the column is part of a secondary business key. Only used during index generation and not normally set.

Source Table

Identifies the source table where the column's data comes from. This source table is normally a stage table within the data warehouse. If the column was sourced from multiple tables, then the normal practice is to record one of the tables in this field and a comment listing all of the other tables in the **Source Strategy** field. This field is used when generating a procedure to update the Data Vault table. It is also used in the track back diagrams and in the documentation.

Source Column

Identifies the source column where the column's data comes from. Such a column is normally a stage table column, which in turn may have been a transformation or the combination of multiple columns. For previous value managed columns the source column is the column in the table whose previous value is being recorded.

Transformation

Transformation. [Read-only].

Join

Indicates whether the table column is used in a table join. [Normally maintained automatically but can be optionally changed to override the default join logic used in the generated update procedure]. The **Source Table** and **Source Column** fields will provide the other Data Vault table's side of the join. The options for this field are: False, True, Manual and Pre Join.

Setting this field to Manual changes the way the dimension table is looked up during the update procedure build. It allows you to join the dimension manually in the Cursor mapping dialog (where the 'Where' clause is built).

Setting this field to Pre Join activates the **Pre Join Source Table** field and allows you to select a table from the drop-down list.

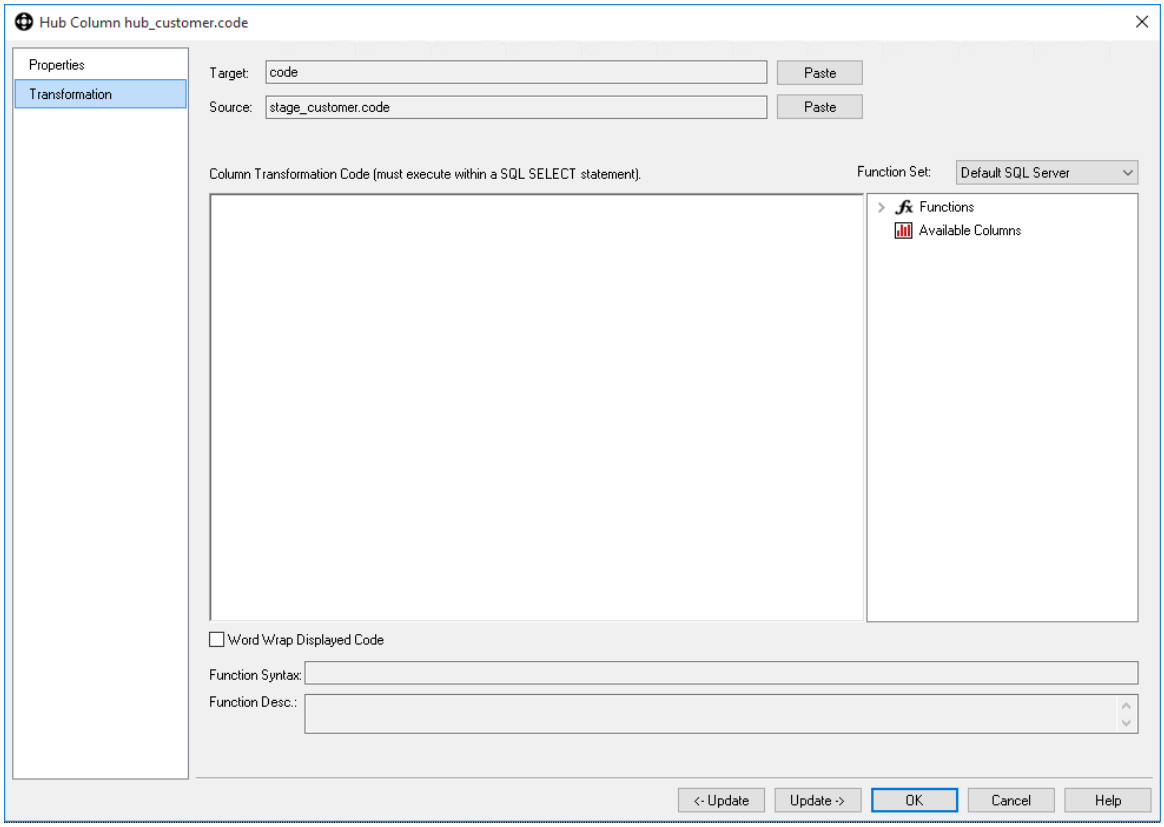
Pre Join Source Table

Indicates the table from which the pre joined column was sourced. When the Join option is set to False, this field becomes inactive. When the Join option is set to True or Manual, this field is set to the current table name. When the Join option is set to Pre Join, then you can select the required table from the drop-down list.

DATA VAULT TABLE COLUMN TRANSFORMATIONS

Each Data Vault table column can have a transformation associated with it. The transformation will be included in the generated procedure and will be executed as part of the procedure update. The transformation must therefore be a valid SQL construct that can be included in a **Select** statement.
For example, we could have a transformation of 'load_order_line.qty * 0.125' to calculate a tax column of 12.5%.

Click the **Transformation** tab to enter a transformation.
The transformation screen is as follows:



NOTE: Transformations are only put into effect when the procedure is re-generated.

Microsoft Analysis Services 2005+ Tabular Mode Tables: For Tabular Mode table column transformations, **Default DAX** is the only applicable Function Set for **after load** transformations.

See ***Transformations*** (on page 650) for more details.

WhereScape®

CHAPTER 14
CUSTOM OBJECTS

Custom1 and **Custom2** objects are user defined objects. These Object Types can be renamed in the **Tools/Options/Object Types/Object Names** menu.

A Custom object license is required for these object types.

Custom objects have the same options and properties as EDW 3NF tables, for more information see ***EDW 3NF Tables*** (on page 426).

WhereScape®

CHAPTER 15
FACT TABLES

A Fact Table is normally defined, for our purposes, as a table with facts (measures) and dimensional keys that allow the linking of multiple dimensions.
It is normally illustrated in the form of a Star Schema with the central fact table and the outlying dimensions.

The ultimate goal of the Fact Table is to provide business information to the end user community. In many cases, different types of fact tables are required to address different end user requirements.
For simplicity, the different types of fact table will be grouped together under the following headings:

- ***Detail*** (see "***Detail Fact Tables***" on page 473)
- ***Rollup*** (see "***Rollup or Combined Fact Tables***" on page 477)
- ***KPI*** (see "***KPI Fact Tables***" on page 480)
- ***Snapshot*** (see "***Snapshot Fact Tables***" on page 493) (essentially a type of Rollup fact table)
- ***Slowly Changing*** (see "***Slowly Changing Fact Tables***" on page 494)
- ***Partitioned*** (see "***Partitioned Fact Tables***" on page 495)

IN THIS CHAPTER

DETAIL FACT TABLES

A detail fact table is normally a transactional table that represents the business data at its lowest level of granularity. In many ways, these tables reflect the business processes within the organization.
These fact tables are usually large and are focused on a specified analysis area.

There may be quite a large number of detail fact tables in a data warehouse implementation, of which only a few are used on any regular basis by the end user community. The disadvantage of such fact tables is that they provide isolated pools of information. Although joined by conformed dimensions, it is still often difficult to answer queries across the various analysis areas.
They do however provide the ultimate drill down for all information and also the platform on which to build higher level and more complex fact tables. In terms of the time dimension detail fact tables are typically at a daily or even hourly granular level.

An example of a detail fact table may be the sales, or orders fact tables that have a daily granularity, and show all sales by product, by customer etc. on a given day.

CREATING DETAIL FACT TABLES

A detail fact table is typically created by dragging a staging table onto a fact table list. Once released, the dialog to create the fact table will commence.

When a staging table is dragged into the Drop Target Pane (middle pane) all fact tables are by default *detail fact tables*.
If manually creating a table, then the table type can be selected under the Properties of the fact table.

Detail Fact Columns

The columns for a detail fact table are typically those of the staging table. Such fact tables typically contain a wide range of measures and attributes as well as the business keys used to look up the artificial dimension keys.
These business keys should be included whenever possible as they provide a means of rebuilding the dimensional link. If size prohibits their inclusion it will probably be necessary to backup or archive all source data to ensure that the fact table can be rebuilt.
These fact tables normally contain a large number of attributes such as dates, which have not been converted to dimensions. Also contained would be information such as order numbers, invoice numbers etc.

See the first tutorial for an example of a fact table creation.

GENERATING THE DETAIL FACT UPDATE PROCEDURE

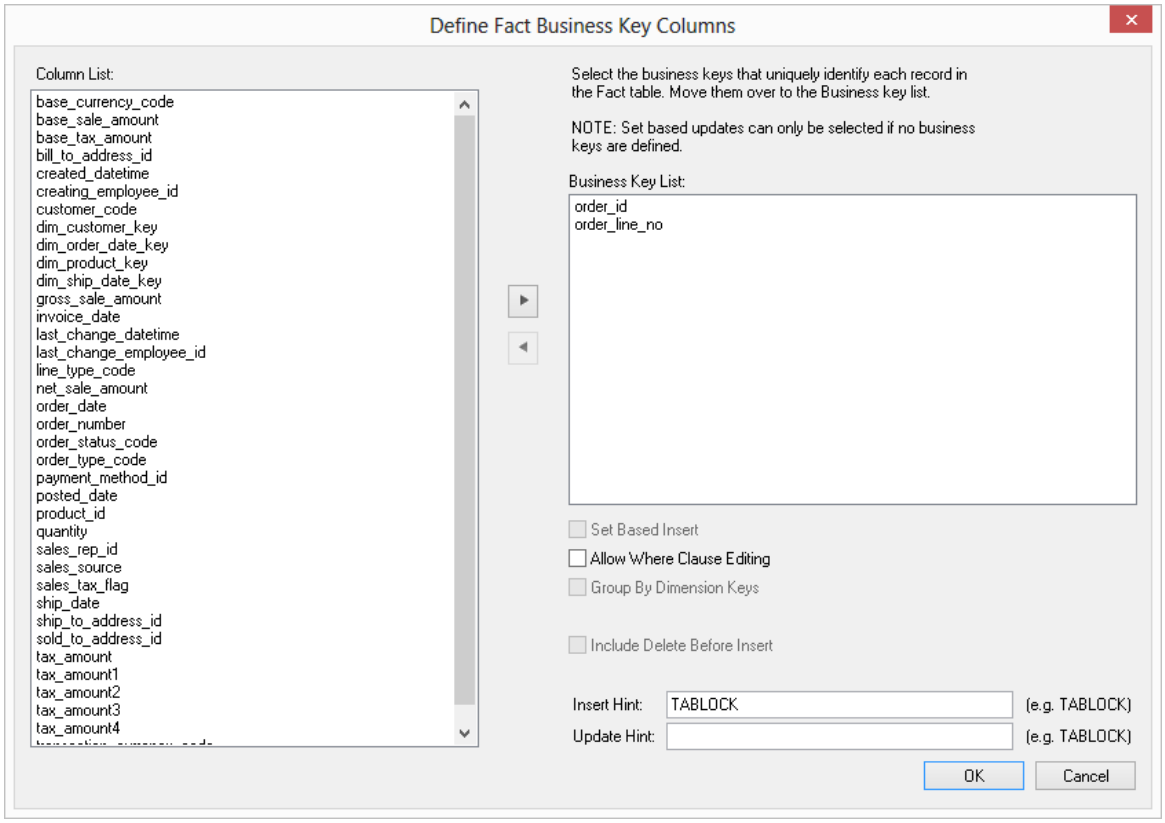
Once a detail fact table has been defined in the metadata and created in the database, an update procedure can be generated to handle the update of the fact table.

Generating a Procedure

- To generate a procedure, right-click on the fact table in the left pane and select **Properties**.
- Click on the **Rebuild** button to start the process of generating the new procedure.
- A series of questions will be asked during the procedure generation based on the type of load information.

DEFINE FACT BUSINESS KEY COLUMNS

The first dialog displayed when generating a detail fact table update procedure is the define fact business key columns dialog, asking for the business key that will uniquely identify each fact table record.
The source table from which the fact table is derived would normally have some form of unique constraint applied.
In most cases this will be the business key. In the example below the order_id and order_line_no are selected for the business key list.



A business key can be made up of multiple columns, but it must provide a unique identifier. Where multiple columns separately uniquely identify rows in the fact table, choose one to act as the primary business key.
For example, a source table may have a unique constraint on both a product code and a product description. Therefore, the description as well as the code must be unique.

None of the columns chosen as the business key should ever contain a NULL value.

Allow Where Clause Editing

If the **Allow Where Clause Editing** option is selected, then the next dialog to be displayed is the **Provide Cursor Mapping** dialog, else this dialog is skipped.

Insert Hint

Enter a database hint to be used in the INSERT statement.

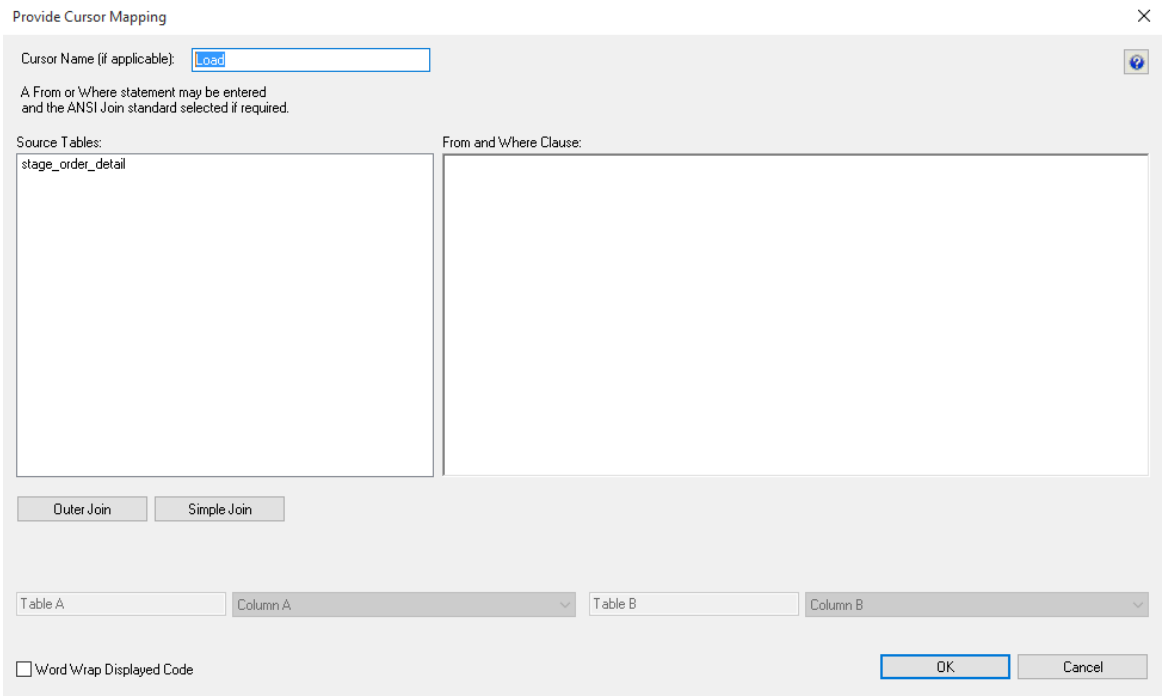
Update Hint

Enter a database hint to be used in the UPDATE statement.

Oracle Only
If the **Set Based Insert** and **Include Delete Before insert** options are selected, the **Definition for a Delete Statement before the Fact Table Insert** will appear allowing for options such as **Truncate** and **Multiple Pass Delete**.

PROVIDE CURSOR MAPPING

This dialog is used to join source tables, add 'Where' clauses and specify group by clauses.



As source table joins should have been performed in the stage table, see ***Generating the Staging Update Procedure*** (on page 372) for more details.
Since most of the real work is done in the stage table, this is all that is needed to build the Detail Fact Update Procedure.

ROLLUP OR COMBINED FACT TABLES

Rollup fact tables are typically the most heavily used fact tables in a data warehouse implementation. They provide a smaller set of information at a higher level of granularity. These fact tables typically have a monthly granularity in the time dimension and reflect figures for the month or month-to-date. They often combine multiple analysis areas (detail fact tables) into one super set of information. This allows simpler comparisons or joining of otherwise isolated pools of information.

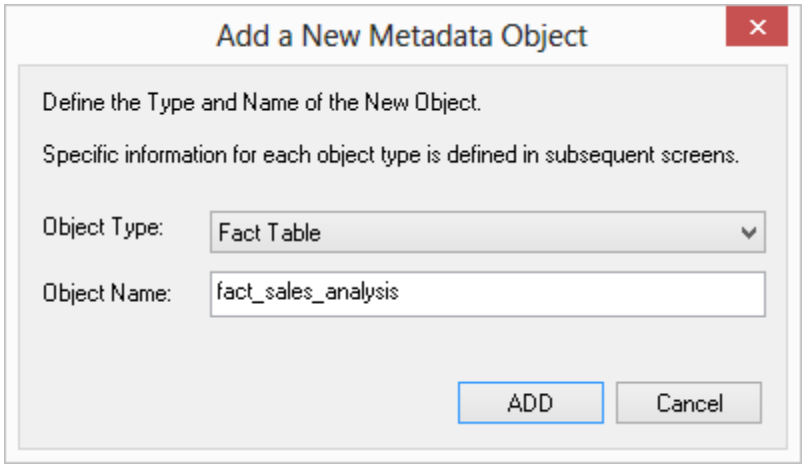
An example may be the combination of daily sales, weekly demand forecasts, and monthly budgets, into one rollup fact table. Such a fact table provides the ability to compare monthly or month-to-date sales against budget and forecast.

These rollup/combined fact tables may and often do contain different levels of granularity, as well as different sets of dimensions.
For example, it may be possible to look at sales and forecasts by promotion, but the budget information does not have a populated promotion dimension, so we will not get a budgeted value for the promotion.

CREATING ROLLUP FACT TABLES

A rollup or combined fact table is typically created by dragging a fact table onto a fact table list. In the example screenshot below a double-click on the Fact Table object group under the Sales project produced a list in the middle pane showing the existing fact tables.

A **fact_sales_analysis** rollup fact table is being created after having selected the **fact_sales_detail** table in the right panel, holding down the left mouse button and dragging the table into the middle pane.
The initial object create dialog will appear. Note that the name implies that this is a rollup fact table.
The name will be changed to the one chosen above.
A list of columns is displayed and those not required are deleted (everything except additive measures and dimensions).



Drag and drop additional columns from other fact tables into the middle pane if required. Where columns have the same name these will need to be changed.
For example, if we acquired a quantity column from **fact_sales_detail** and a column of the same name from **fact_forecast** we may choose to change the names to **actual_quantity** and **forecast_quantity**.
Once all columns have been defined:

1. Create the table.
2. Edit the properties of the table and request that a new procedure be created.
3. Click **OK** and the dialog to create a template procedure will commence. Included in this process will be the definition of the date dimension granularity. Monthly rollup is the norm.
4. To just combine fact tables with no rollup select the date dimension key as directed in the dialog. Other forms of dimension rollup will require some alteration to the produced template procedure.

Rollup Fact Columns

The columns for a rollup fact table are typically the dimensions, and only those key measures that are added.
For example in the sales world, measures may simply be quantity, kilograms, $value, forecast_quantity, forecast_$value, budget_quantity, budget_$value.
See the second tutorial for an example of rollup fact table creation.

GENERATING THE ROLLUP FACT UPDATE PROCEDURE

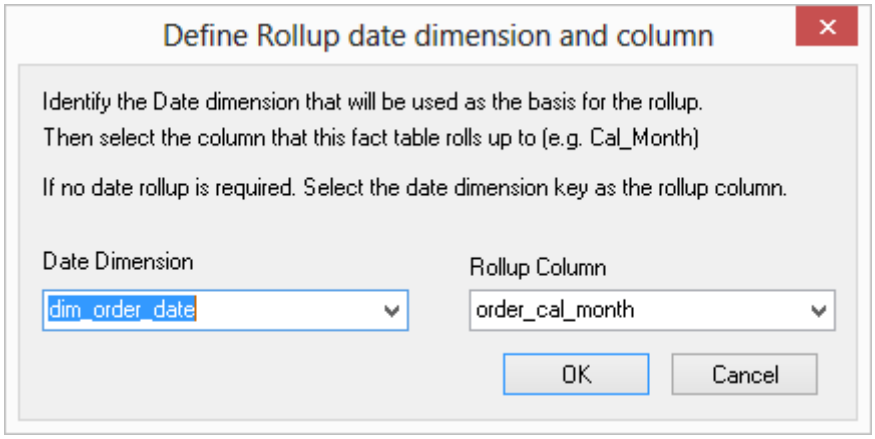
Once a rollup fact table has been defined in the metadata and created in the database, an update procedure can be generated to handle the update of the fact table.

Generating a Procedure

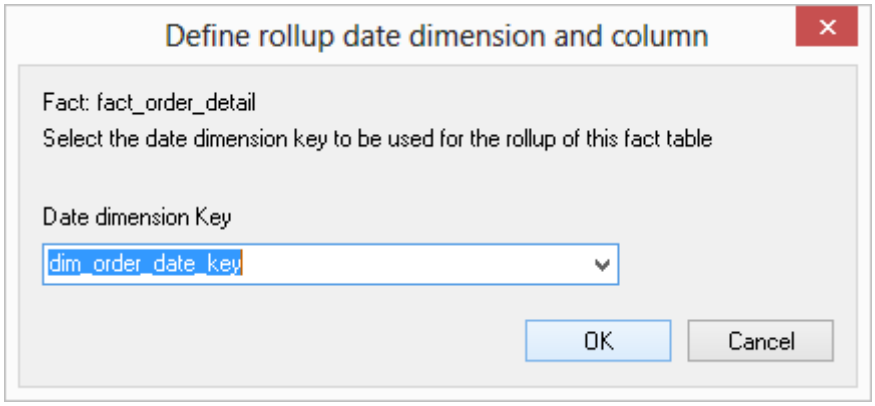
- To generate a procedure, right-click on the fact table in the left pane and select **Properties** to edit the properties for the fact table.
- Click on the **Rebuild** button to start the process of generating the new procedure.
- A series of questions will be asked during the procedure generation based on the type of load information.

DEFINE ROLLUP DATE DIMENSION AND COLUMN

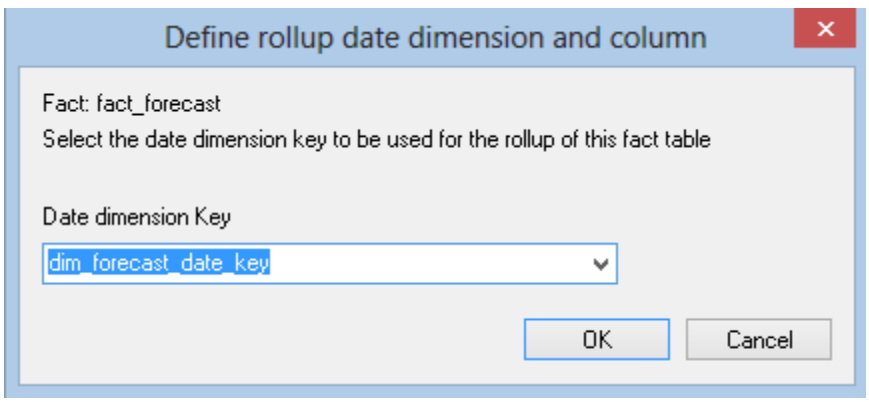
The first dialog displayed when generating a detail fact table update procedure is the define rollup date dimension and column dialog:



RED needs to know which date is driving the rollup and to which level of detail we wish to set as our base.
In this example, we have chosen **dim_order_date** as our date dimension and **order_cal_month** as our rollup column to set the level to summarize all transactions to. Click **OK**.
RED also needs to understand which date field in the actuals should be used to base the rollup; we have chosen **dim_order_date_key** and click **OK**.



Once again, select the date dimension key and click **OK**.



KPI FACT TABLES

Key Performance Indicator (KPI) fact tables share a lot of similarities with the rollup fact tables in that they are typically at a monthly level of granularity and they combine information from multiple detail fact tables. They differ in that they provide specific sets of pre-defined information (KPIs). Such KPIs are often difficult to answer and require the implementation of some form of business rule. Also, they often provide pre-calculated *year-to-date*, *same month last year*, and *previous year-to-date* values for easier comparisons. They will have a KPI dimension which provides a list of all the specific statistics or performance indicators we are tracking.
An example from our previous sales examples may be a set of KPIs dealing with customer acquisition and loss. We may have a specific KPI that records the number of new customers during the period and the $value of their business during that period. Another KPI may record the percentage growth of all customers over the last six months. Another may show the number of customers who closed their accounts in the period, and the value of their business over the preceding 12 months.
KPI fact tables require the specific definition of each KPI.
It is strongly recommended that the KPI tutorial be undertaken. This tutorial will give an understanding of what can be achieved via a KPI fact table.

CREATING THE KPI DIMENSION

The KPI dimension essentially provides a record of each KPI or statistic we are tracking. There should be one row in this table for each such element.
The KPI fact table requires that this dimension contain at the least an artificial dimension key and a business key. We will refer to the business key as the KPI Identifier.
The following example covers the manual creation of a basic KPI dimension:

1. Right-click on the **Dimension** object group in the left pane and select **New Object**.
2. Give the new dimension a name for example **dim_kpi**.
3. Select a **Normal** dimension and click **OK** on the Properties screen to define the dimension table.
4. Right-click on the dimension name in the left pane and select **Add Column**, add the three columns defined in the following table.

Field	Column 1	Column 2	Column 3
Column Name	dim_kpi_key	kpi_identifier	kpi_name
Business display	kpi artificial key	kpi unique id	kpi name
Datatype - Oracle - SQL Server/DB2	 integer integer	 varchar2(12) varchar(12)	 varchar2(256) varchar(256)
Nulls	Clear	Clear	Checked

Field	Column 1	Column 2	Column 3
Key Type	0	A	Clear
Artificial Key	Checked	Clear	Clear
Primary Business	Clear	Checked	Clear

The first two columns are required, and should be set up as shown above, although a different data type can be used for the second column, and the names can be chosen to suit the requirements.

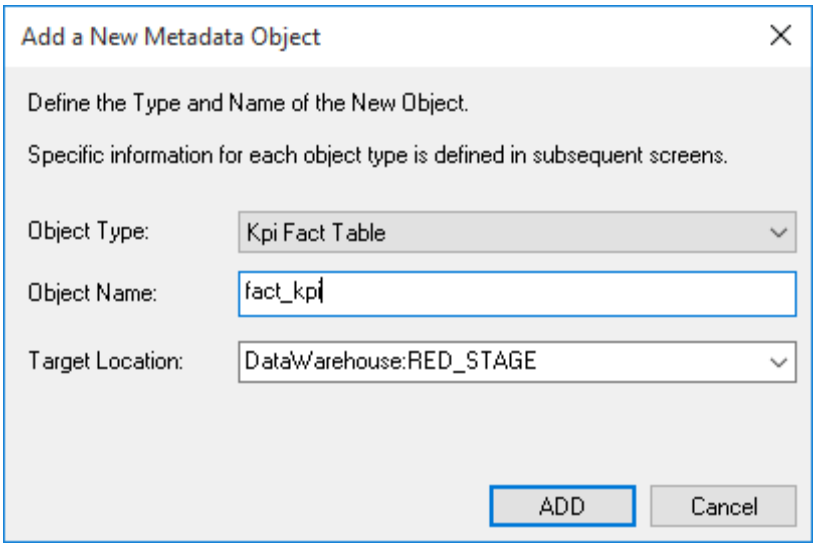
CREATING KPI FACT TABLES

A KPI fact table is normally created manually.

1 Right-click on the Fact Table object group in the left pane and select **New Object**.

2 A dialog displays as shown below.

Note: The default object type is **Fact Table**. This must be changed to **KPI Fact Table** and a table name entered.



Adding columns

Once created the KPI fact table will need to have columns added. New columns can be added by dragging columns from the browser window, or by manually adding from the right mouse menu. Dimension keys should normally be dragged to ensure that all the properties are correctly set.

The columns for a KPI fact table are typically the dimensions and generic measures. For example, valid measures could be **kpi_count**, **kpi_quantity** and **kpi_value**. The interpretation of these measures will depend on the KPI or group of KPIs being queried. As with the rollup fact table it is normal and desirable to keep the number of measures as small as possible. Attributes should not be included in these fact tables. These measures must allow Null values for the generated update procedure to function correctly.

NOTES:

1. The KPI measure column order is important. For example, if we have mtd count, quantity and value in that order then the prev month, prev ytd and ytd figures also need count first followed by quantity and finally value. If this order is not adhered to then the generated procedure will load the wrong values into the wrong columns.

2. A kpi fact table should also have the column dss_update_time added. This column is required if the kpi table is to be used to populate higher level fact tables, aggregates or cubes. This column should be populated with the current date time whenever the particular row is updated.

SETUP OF KPI FACT TABLES

The set-up of a KPI fact table is somewhat more complex than the other types of fact table. Once all columns have been defined the following steps need to be taken:

1. Create and populate a KPI dimension that provides at least an id (unique identifier) for each KPI and a description. The key from this dimension must be one of the columns in the fact table.
2. Physically create the KPI fact table in the database once all columns including the KPI dimension key have been added.
3. Define the column type and default value for ALL columns in the table. There must be at least one date dimension key and one KPI dimension key. These attributes are set by selecting a column and modifying its properties.
4. Create the update procedure for the fact table.
5. Set-up each individual KPI.

KPI SETUP

KPI Dimension

The KPI dimension provides a means of selecting a specific KPI. This dimension needs to be built in advance of the KPI fact table. It does not need to be populated, as the KPI fact table will add entries to this table as required. A key and some form of unique identifier for the KPI is required, e.g, they may be numbered or given short names.

Create the KPI Fact Table

Right-click on the KPI table and select **Create/Recreate** to physically create the table in the database.

Defining column types and default values

1. List all the columns for the KPI fact table.
2. Amend the properties for each column. The column types can be one of the following:

Type	Description
Dimension Key	Any dimension that is not the time or KPI dimension
Time Dimension Key	Time dimension key
KPI Dimension Key	KPI dimension key
Month-to-date Measure	Used to identify the normal KPI measures. These are calculated to populate the KPI during each run.
Year-to-date Measure	Used to identify year-to-date measures for the KPI. Optionally calculated or derived from existing rows and measures in the KPI fact table.
Last year same month Measure	Used to identify measures for same month last year comparisons. Derived from existing rows and measures in the KPI fact table.
Previous year-to-date Measure	Used to identify previous year-to-date measures. Derived from existing rows and measures in the KPI fact table.
Fact Table Key	Not normally used
Date	Not normally used
Attribute	Not normally used
Measure	Not normally used

TIP: If an **Attribute** or **Measure** column is specified then this column will just be set to its default value. In this way, an update_time column can be set by making its default value **sysdate**.

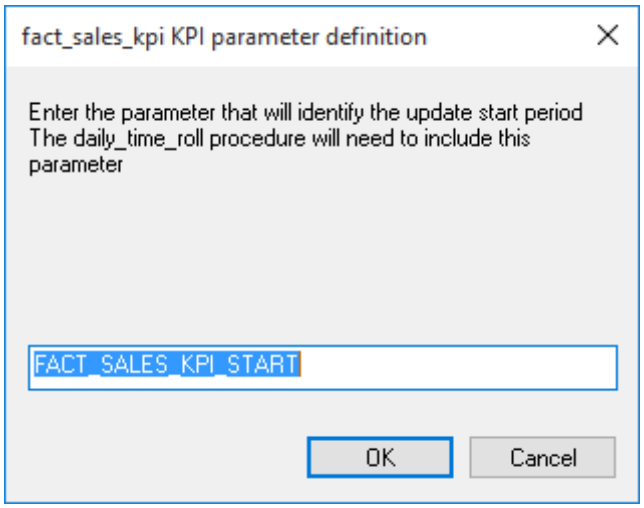
3 For each column set a default value.

- For measures and dimension keys this would normally be zero (0). Nulls can be represented as two single quotation marks, i.e. ' '
- **sysdate** can be used for a default date value
- These default values populate the columns where and when no other information is available. For example, if a KPI only makes use of 4 of the 6 dimensions available then the remaining 2 dimensions will be set to their default values.

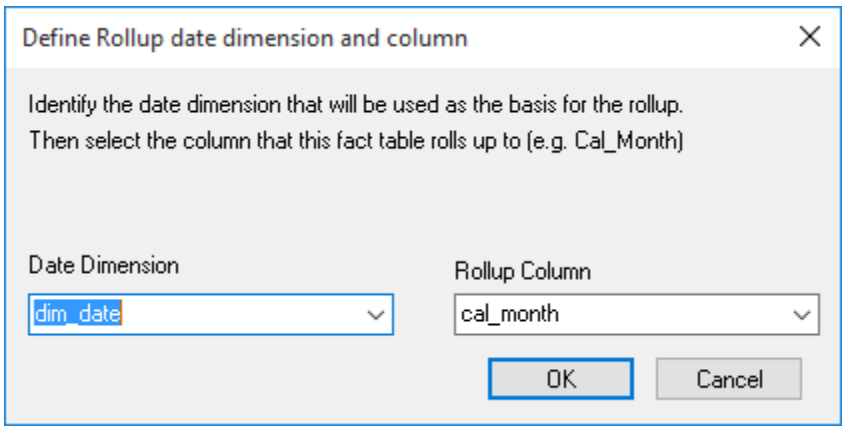
Create the 'Update' procedure

Right-click on the KPI fact table name and select **Properties**. In the Properties dialog box, set the update procedure list to the **(Build Procedure...)** option. Click **OK** to initiate a series of dialogs to create a procedure used to update the KPI fact table.

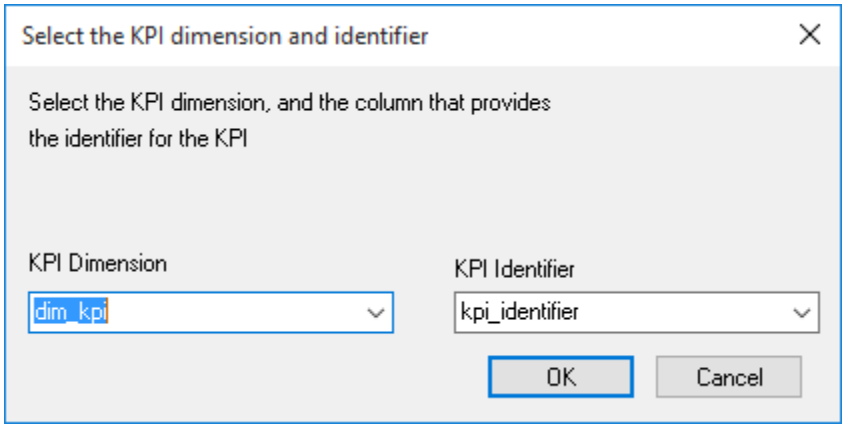
This first two dialogs ask for parameter names to define the start and end periods for which the KPIs will be updated. The default naming convention is fact_table_name_START and fact_table_name_END. Accept these default names if possible. These two parameters will need to be created, and will need to be updated once the KPIs have been defined and tested. Under normal operation these parameters should probably be set daily from the daily_time_roll procedure.



The date dimension and column for rollup is required. Normally this would be cal_month or fin_month.
This assumes that most KPIs are monthly or annual. Subsequently a rollup year and date will be required from the same date dimension.



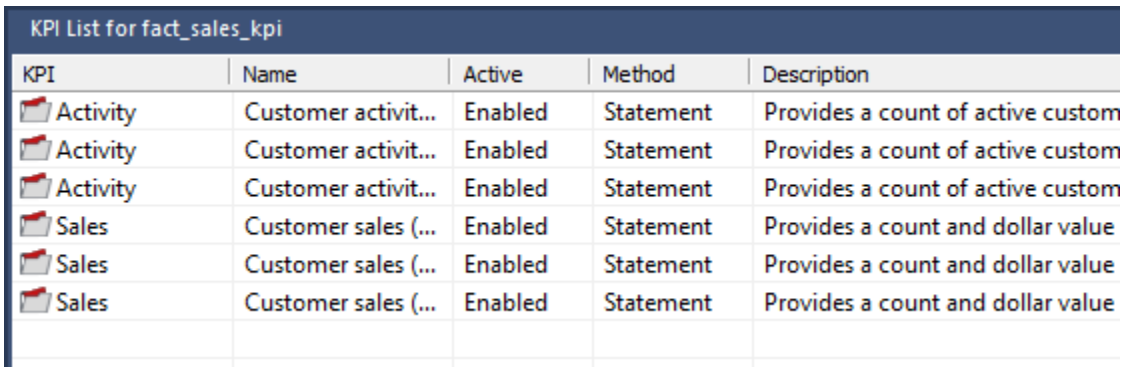
The KPI dimension is required. Also, the identifier column (not the key) that uniquely identifies the KPI. This would normally be a number or short name (max 64 chars).



The procedure is then created and compiled.

Define each KPI

Once the procedure has successfully compiled we are ready to start setting up each of the KPIs. Right-click on the KPI table name to get the popup menu.
This menu is similar to the other fact table menus except for the additional options to display and add KPIs. A display of the KPIs will list any KPIs that are currently defined. The result set is as appears below:



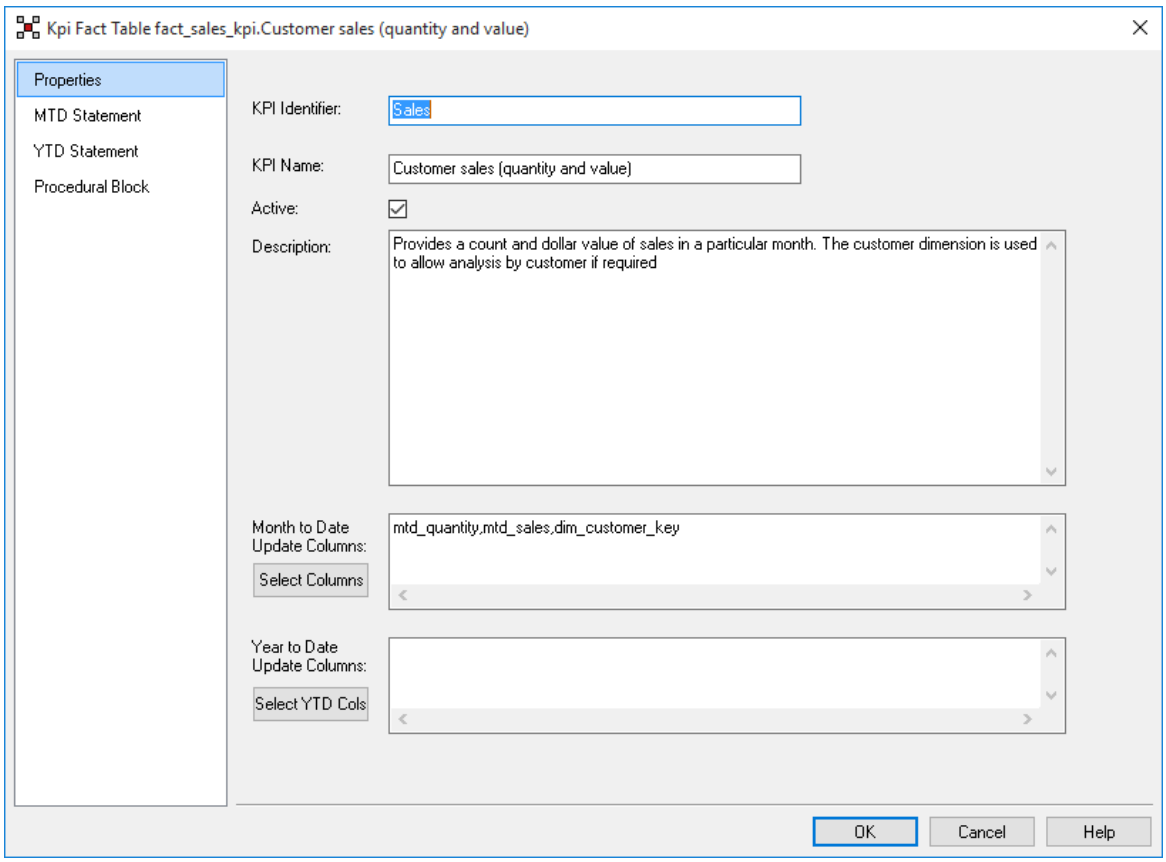
KPI	Name	Active	Method	Description
Activity	Customer activit...	Enabled	Statement	Provides a count of active custom
Activity	Customer activit...	Enabled	Statement	Provides a count of active custom
Activity	Customer activit...	Enabled	Statement	Provides a count of active custom
Sales	Customer sales (...	Enabled	Statement	Provides a count and dollar value
Sales	Customer sales (...	Enabled	Statement	Provides a count and dollar value
Sales	Customer sales (...	Enabled	Statement	Provides a count and dollar value

The 'Id' is the same unique identifier that will be found in the kpi dimension. The Name is a name entered in the properties of the KPI and does not relate to the dimension in any way. The Active flag indicates if the KPI will be processed when the **Update** or **Process** menu options are selected. Each KPI can be defined as a SQL Statement, a Procedural Block or a Procedure. The current method used to update the KPI is one of these three and is displayed under Method. The description is a comment or description field entered in the properties of the KPI and in no way, relates in the KPI dimension.

A right-click anywhere in the List Window will allow a KPI add. If a right mouse button is depressed while positioned over the 'Id' then a menu popup will provide options to copy, modify, delete, add and view the KPI statement.

When the add KPI option is selected a Properties screen will appear, as shown below. The id and name fields are required. If the **Active** checkbox is checked the procedure will be enabled for update.

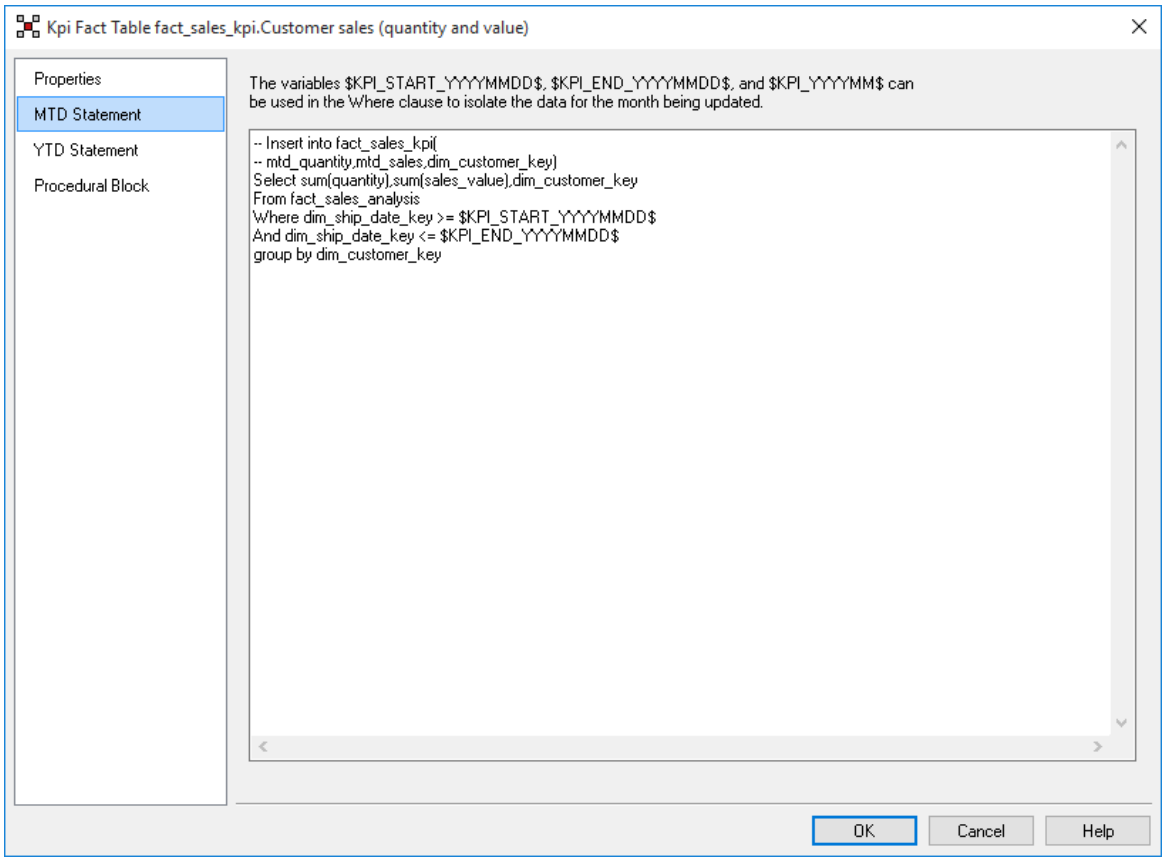
The **Select Columns** button allows the selection of the columns that this KPI will update during the month to date update pass. The date and KPI dimensions will be updated for you so are not provided as choices. The only choices provided are the 'Month to date measures' and the other 'Dimensions'. Select those measures and dimensions that will be updated. All non selected measures and dimensions will be set to their default values. The resultant choice is shown in the Update Columns window. This window should only be updated via the **Select Columns** button.



Once these initial values are defined the **Statement** tabs can be selected to define the update SQL statement. If a procedural block or procedure is required for the update (i.e. too complex for a single SQL statement) then select the Procedural Block tab.

The KPI Statement

The **Statement** tabs present edit windows that allow the input of a select statement that will return the values for the columns being updated.
They must return the values for the columns selected in the Properties tab in the order shown in the **Update Columns** window. An example of a completed KPI statement follows:



The first two lines are comments inserted (when modifying) to provide a reminder of the columns and order required by the select statement. Anything prior to the initial select statement is ignored.

The statement must make use of at least one of the variables provided to select the appropriate period. This variable will be replaced with the values for the period being updated at run time. These values are derived from the start and end parameters defined for the KPI fact table.

In order to group by the non selected columns and any attributes etc. WhereScape RED must be able to break this statement into its component Select, From, Where, Group By, and Having components (where present). If a complex statement is used that has more than one of these key words, then the primary one must be in UPPER case. For example, if a sub query is being used then the main SELECT and FROM key words must be in upper case and must be the only occurrence of these key words in uppercase. WhereScape RED will check and issue a warning if it is unable to resolve the main key words.

To test a statement, right-click on the KPI Id and select **View Statement**. This will provide a window with the full statement with the default columns inserted and the parameters converted. This statement can be cut and run in Sql Admin (shipped with WhereScape RED) or in a database specific utility for testing.

Procedural Block

The **Procedural Block** tab presents an edit window that allows the input of an Oracle anonymous block. If there is anything in this window it will be actioned instead of the statement. A button in the lower right allows a rudimentary template block to be built. If used this block must do the actual insert statement rather than just the select. It must also handle ALL dimension, month to date measure and non standard columns. In addition, it will need to put default values in columns not actively being updated. The KPI dimension and date dimension keys must also be updated. The output from a 'View Statement' option for another KPI will give an indication of what the block must do. The same month last year, year to date and previous year to date measures will still be handled automatically. The same $KPI_ variables as shown above the statement edit window can and must be used.

Procedure

If for some reason it is not possible to implement the KPI in either the statement or procedural block, then a procedure may be created. The name of such a procedure can be entered in the window at the bottom of the Procedural Block tab. If any value is in this window, then the procedure will take precedence over both the statement and block.

This procedure must adhere to the standard set of parameters as with all other procedures to be executed by the scheduler. It must undertake the same tasks as defined above in the Procedural Block section, and as with those tasks it does not need to set the year to date, last year same month and previous year to date values.

Copying KPIs

Right-click on the KPI Id and select **Insert Copy** to insert an exact duplicate of the KPI. This copy can then have its Id and Name changed along with its statement to create a new KPI. This is particularly useful when only small differences exist between a number of KPIs.

Testing/Development.

During testing, it would be normal to DISABLE all KPIs except the one currently being developed. In this way, the update or process options will only utilize that KPI. As mentioned above the 'View Statement' option of the KPI popup menu displays, and allows the cutting of, the statement that will be executed to acquire the KPI information. This statement can be tested in sql*plus or some other environment if problems are encountered.

GENERATING THE KPI FACT UPDATE PROCEDURE

Once a detail fact table has been defined in the metadata and created in the database, an update procedure can be generated to handle the update of the fact table.

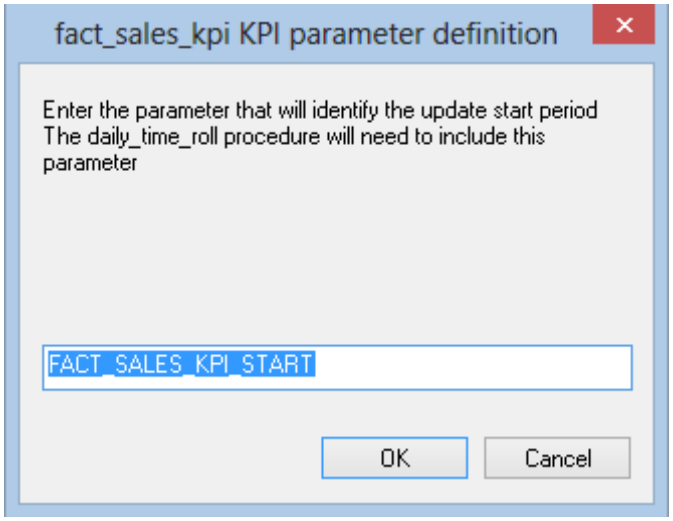
Generating a Procedure

To generate a procedure, right-click on the fact table in the left pane and select **Properties**. Click on the **Rebuild** button to start the process of generating the new procedure.

A series of questions will be asked during the procedure generation based on the type of load information.

KPI PARAMETER DEFINITION

The first dialog displayed when generating a detail fact table update procedure is the KPI parameter definition dialog:



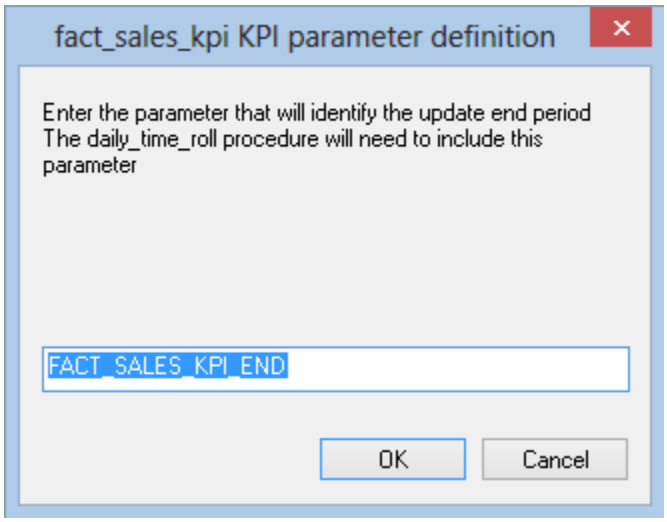
The **KPI parameter definition** dialog box will prompt for a parameter to hold the start period that we are interested in. This period sets the first period that will be processed when the KPI fact table is updated. Once entered, click **OK**.

NOTE: If this parameter is not set, the default current period will be set from dim_date. The format is YYYYMM.

TIP: The **dim_date** custom update procedure **daily_date_roll** can be updated to set these parameters automatically if you do not wish to control them manually. This is good practice

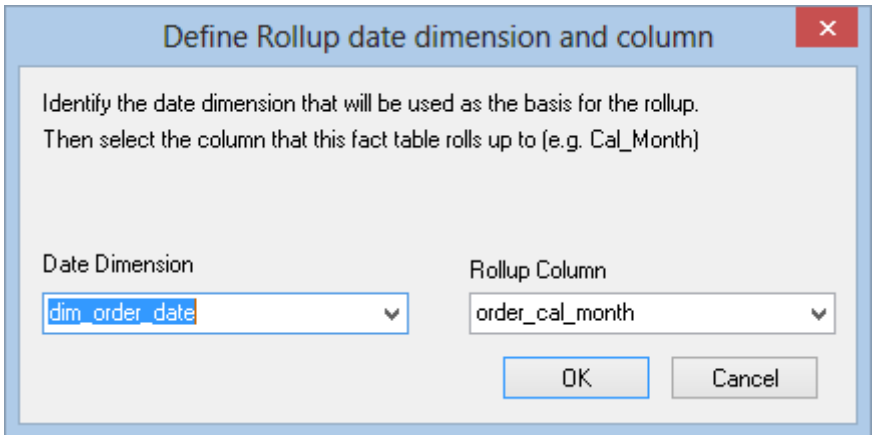
---

Similarly, a **KPI parameter definition** dialog will prompt for a parameter to hold the end period. Once entered, click **OK**.

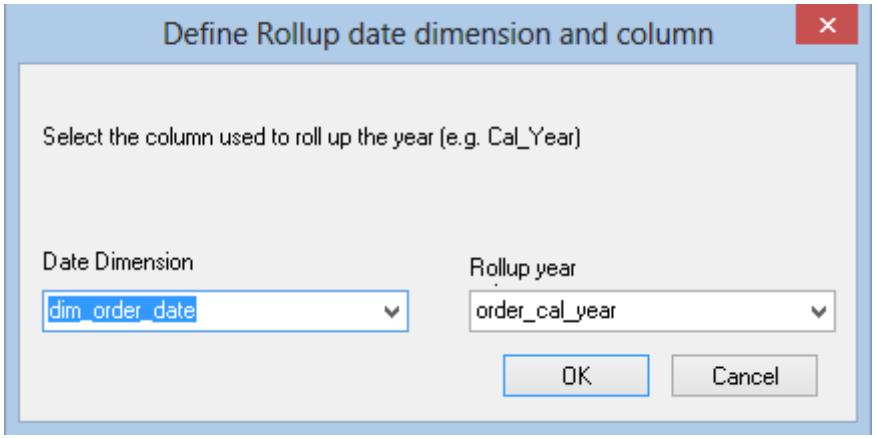


Define Rollup date dimension and column

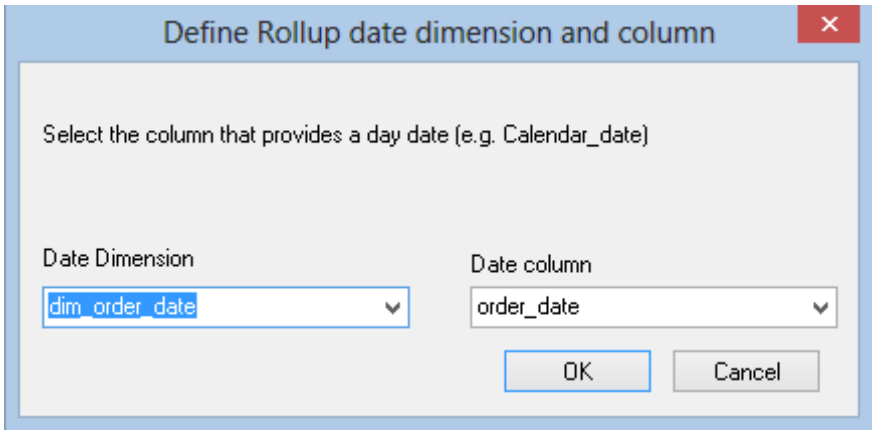
As with Rollup tables, we now need to specify which date in the transactional data we will use to control the rollups and to which level – in this example, for the Monthly level rollup, we choose **dim_order_date** as our **Date Dimension** and **order_cal_month** as our **Rollup Column** and click **OK**.



Once again for our Yearly rollup, in this example we choose **dim_order_date** as our **Date Dimension** and **order_cal_year** as our **Rollup Year** and click **OK**.

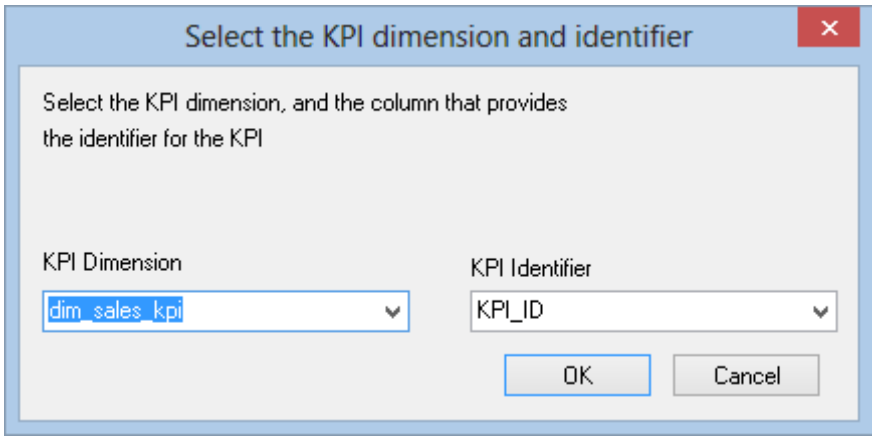


The next screen sets the Daily level for the update procedure to understand the lower level detail, we choose **dim_order_date** as our **Date Dimension** and **order_date** as our **Date Column** and click **OK**.



Select the KPI dimension and identifier

Finally, we need to specify where the KPIs are defined and how they are defined, so in this example we choose **dim_sales_kpi** as the **KPI Dimension** and **KPI_ID** as the **KPI Identifier**. Click **OK.**



SNAPSHOT FACT TABLES

Snapshot fact tables provide an ‘as at’ view of some part of the business. They always have a time dimension that represents the ‘as at’ date. For example, we may have a weekly or monthly snapshot of our inventory information. Such a snapshot would allow us to see what inventory we had at a particular point in time, and allow us to compare our holdings over time. Snapshots are often used when dealing with people profiles or client/customer profiles to enable us to view the state of our relationship with an entity at a given point in time e.g. to see a monthly trend of a client's outstanding balance.

CREATING SNAPSHOT FACT TABLES

The creation of snapshot fact tables is very dependent on the method used to create the snapshot. In some cases, they may be created in exactly the same way as detail fact tables. These cases are typically where the OLTP application provides the ‘as at’ state. For example, an inventory system may provide a weekly stock take table which is the basis for our snapshot fact table. In such cases the table is created as per a detail fact table, but with additional tailoring around the time dimension.
In other situations, the snapshot is built from an existing transactional fact table, where all information up to a specific date is summarized or frozen to create the snapshot. In these instances, the snapshot can be created as a rollup fact table.
Regardless of whether a detail or rollup method is used for the creation of the snapshot table, a significant amount of work will be required to tailor the update procedure to reflect the particular requirements of the snapshot table.

Snapshot Fact Columns

The columns for a snapshot fact table vary to a large degree. A date dimension is always present to reflect the ‘as at’ date. Some snapshots may have a large number of measures reflecting $values quantities etc., whereas others may be rich in attribute (non additive) facts.

GENERATING THE SNAPSHOT FACT UPDATE PROCEDURE

Once a snapshot fact table has been defined in the metadata and created in the database, an update procedure can be generated to handle the update of the fact table.

Generating a Procedure

- To generate a procedure, right-click on the fact table in the left pane and select **Properties**.
- Click on the **Rebuild** button to start the process of generating the new procedure.
- A series of questions will be asked during the procedure generation based on the type of load information.
- If a **detail** method was used for the creation of the snapshot table, refer to ***Generating the Detail Fact Update Procedure*** (on page 474)
- However, if a **rollup** method was used for the creation of the snapshot table, refer to ***Generating the Rollup Fact Update Procedure*** (on page 478)

SLOWLY CHANGING FACT TABLES

Slowly changing fact tables provide a complete history of changes for some part of the business. Typically, such fact tables are used for situations where a relatively small number of changes occur over time, but we need to track all of these changes.
A good example is an inventory stock holding that does not change very often. In this case a slowly changing fact table can record both the current inventory levels as well as providing a means of seeing the state of inventory holdings at any point in time.

CREATING SLOWLY CHANGING FACT TABLES

WhereScape RED does not provide any automated way for creating a slowly changing fact table. The best method is to proceed as follows.

1. Drag and drop the stage table into a dimension target and create a dummy slowly changing dimension with the name we will be using for our fact table. (i.e. fact_...).
2. Answer the questions to the pop-up dialog boxes. Select a join if asked, but do not bother to join any of the tables. Select the columns to be managed as slowly changing. Normally these will be the measures such as quantity and all of the dimension keys. The generated procedure will fail to compile.
3. Inspect the additional columns added for the dimension and make a note of them. Specifically, dss_end_date, dss_start_date, dss_current_flag and dss_version.
4. Inspect the indexes created to help manage the slowly changing dimension and make a note of their columns.
5. Delete the dummy dimension you have created.
6. Delete the get_.._key procedure created for the dimension. Do not delete the update procedure.
7. Create a detail fact table in the normal manner.
8. Add in the special columns as created for the dimension. Namely dss_end_date, dss_start_date, dss_current_flag and dss_version.
9. Recreate the table
10. Create indexes as per the dimension table to help maintain the slowly changing fact table.
11. Modify the update procedure for the fact table. Bring up the old version of the procedure that was created for the dimension of the same name. This will have been automatically versioned out when the latest procedure was created and can be seen through the procedure viewer.
12. This new procedure will need to merge the code from the old procedure, which will provide a guide in how to build the code for a slowly changing fact table.

PARTITIONED FACT TABLES

Partitioned fact tables are normally used for performance reasons. They allow us to break a large fact table into a number of smaller tables (partitions).
WhereScape RED supports partitioned fact tables. The following section explains their creation and support.

CREATING A PARTITIONED FACT TABLE IN SQL SERVER

In WhereScape RED a partitioned fact table can only be created from an existing fact table. The process therefore is to create a fact table normally, including the update procedure and indexes. The table is then partitioned through the **Storage** screen of the fact table's properties dialog. WhereScape RED will assist in creating the exchange table, modifying the indexes and building the procedure to handle the update of the partitions.

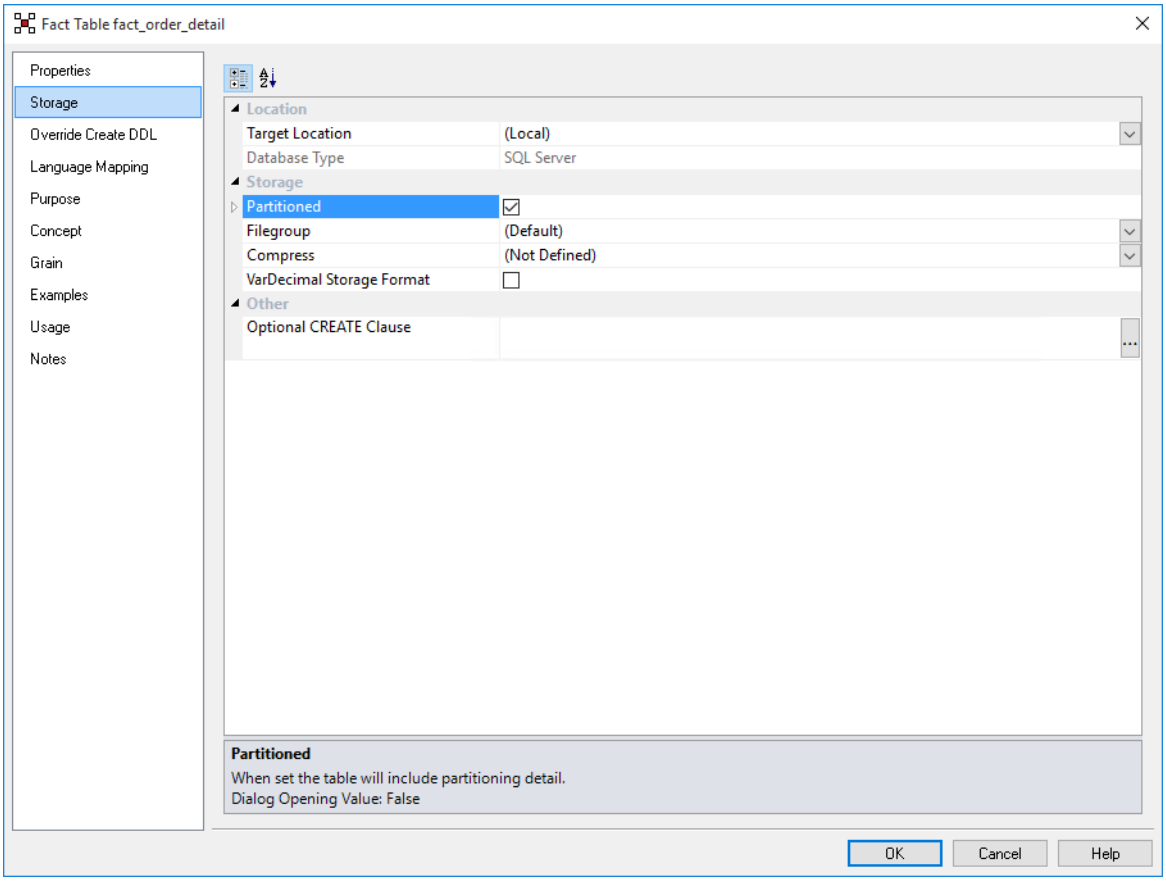
Create a detail or rollup/combined fact table

- Create the fact table that you wish to partition in the usual way.
- You must create an update procedure but do not need to load any data. The process of creating the update procedure will also build any indexes for the fact table.
- RED will be using a partition exchange table to move data to and from the partitioned fact table. This exchange table must have the same columns, column order and indexes as the fact table. It is therefore recommended to get the fact table into its final form before partitioning it.
- Add any indexes to avoid extra work later.

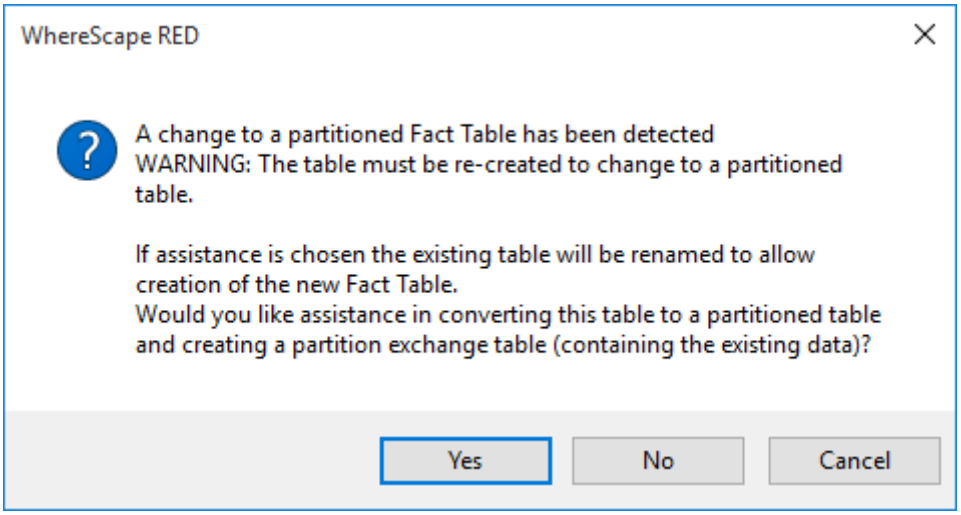
Convert to a partitioned fact table

1. Double-click on the fact table that you want to partition to bring up its properties screen.
2. Click on the **Storage** tab and tick the **Partitioned** check-box. (see example in the screenshot below)
3. A prompt will be displayed warning you that this table will need to be recreated to change the current fact to a partitioned table and offering assistance to create the partitioned table. The exchange table is then used to replace the relevant period in the fact table. The fact table remains in a queryable state throughout the process.
4. Click **Yes** to launch the wizard that will assist you through the partitioning of the table.

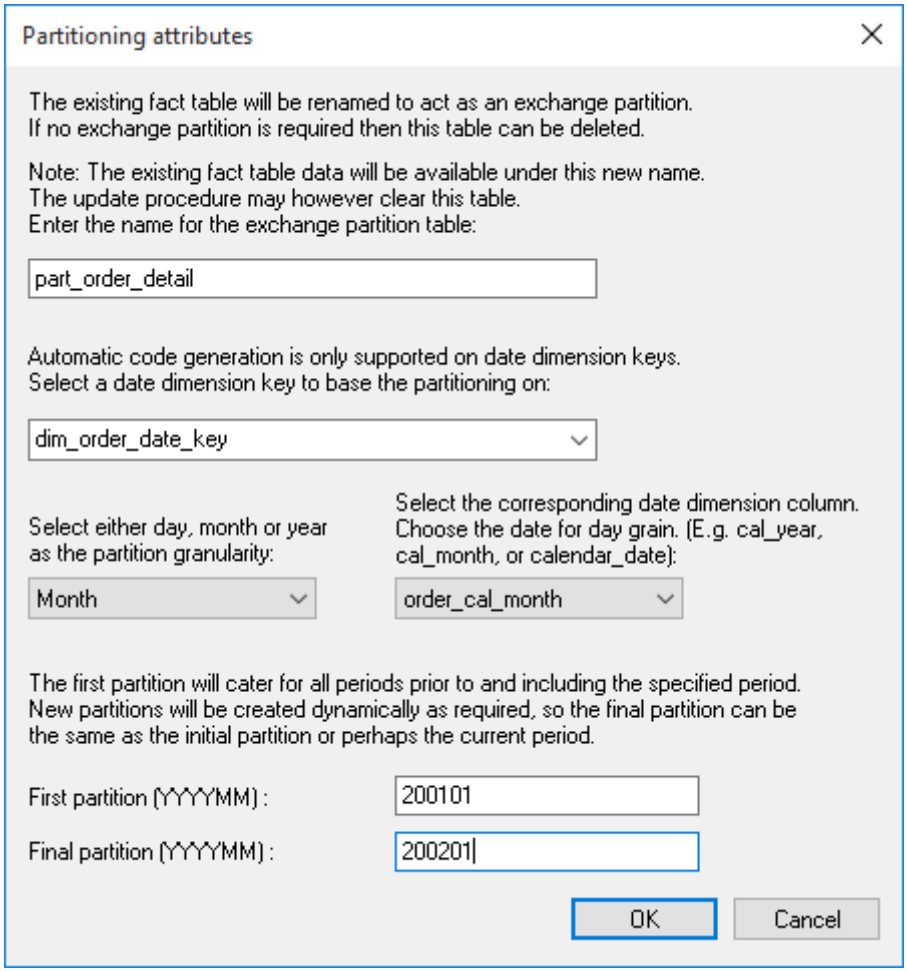
Note: Converting an existing non-partitioned table to a partitioned table cannot be done using a deployment application.
To convert non-partitioned to partitioned tables using deployment applications some manual intervention may be required to update the target databases to match the new metadata.



5 As you click the Partitioned check-box, a dialog will pop-up as follows asking for confirmation that the table is to be converted to a partitioned fact table.
If you click **Yes**, the existing fact table will be renamed to become the exchange partition table. Any current data will remain in this table.



6 The parameters for the partitioned fact table are then prompted for and are described below:



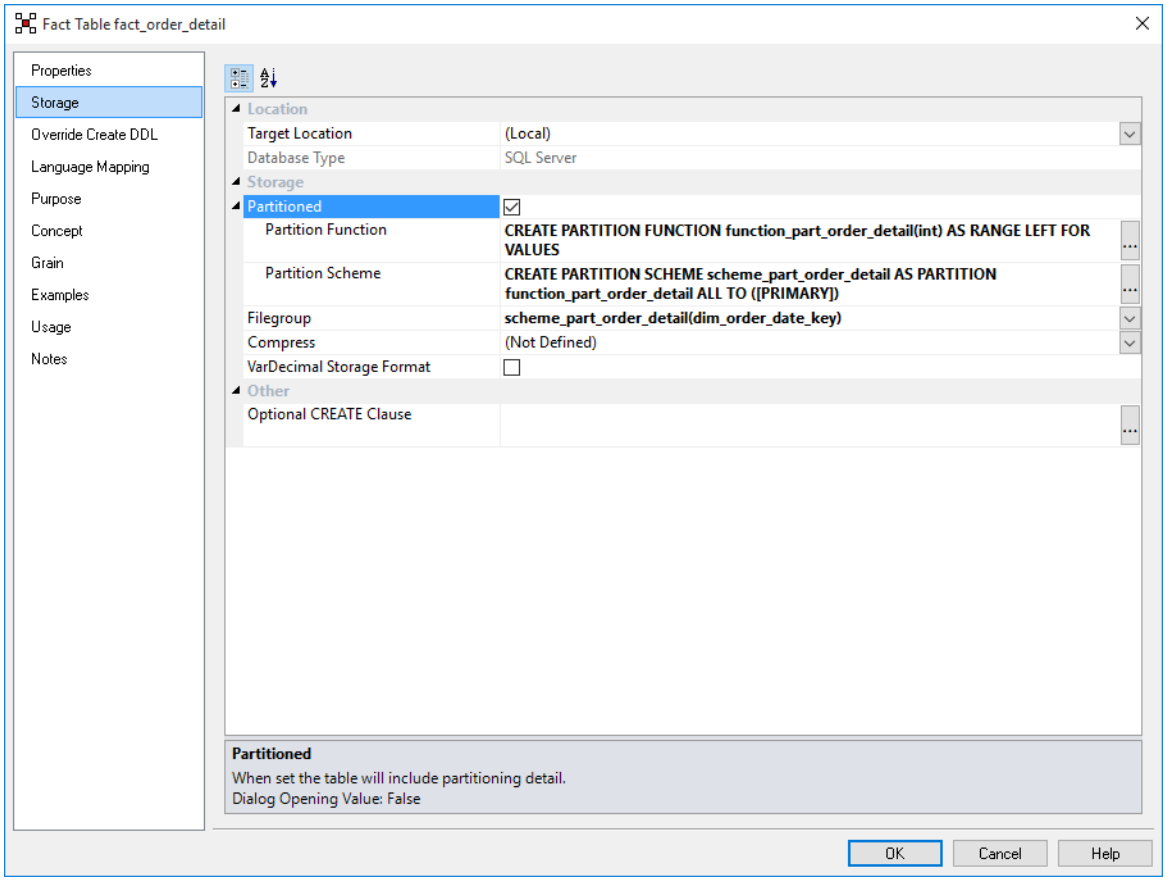
7 The partition exchange table name is prompted for. This table is identical to the fact table and is used to swap partitions to and from the fact table. Enter a **name** for the exchange partitioned table.

- The fact table will be partitioned by the date dimension key. Select a **date dimension key** that will be used as the basis for the partitioning of the fact table. It is assumed that this key has the standard WhereScape format of YYYYMMDD.
- Select a partition granularity of day, month or year along with the corresponding column in the date dimension.
  - For a **daily partition** a column with a date data type must be chosen. The date dimension key will actually be used for the partitioning but the date column is required.
  - For a **monthly partition** a column with the format YYYYMM must be chosen. In the standard WhereScape date dimension this is the column cal_month.
  - For a **yearly partition** a column with the format YYYY must be chosen. In the standard WhereScape date dimension this is the column cal_year.
- The first and last partition should then be selected. The last partition is not that important as additional partitions will be created as required. Normally just select a partition a few on from the first. The first partition is however important as any data that is dated prior to the first partition will be loaded into this first partition. The partition must be entered in the format shown. For example, monthly partitions require the format YYYYMM.
- After all the relevant fields have been completed, click **OK** and the conversion process will populate the **Partition Function, Partition Scheme** and **Table Partition Scheme** fields of the table's storage properties.

Note: This field is limited in size, so if too many partitions are chosen only the first few will be added to this field. The other partitions will be created dynamically by the update procedure.

---

8 Click the **OK** button on the table's **Storage** screen to begin the process of converting the fact table, creating the new fact table and building the new update procedure.
Examine the results screen to see what has been done.



The exchange partition table will now have any current data, to populate the fact table, copy the data from partition table to the fact table, drop and recreate fact index.
The date dimension key used to base the partitioning on is not allowed to have nulls. WhereScape RED has modified the table property, however the partition table needs to be validated and recreated to be valid.
Then the fact table can be processed.

The partitioned fact table update procedure

The generated procedure will handle the creation of new partitions as required.

Any data for periods prior to the first partition is placed in the first partition. If data is received for a period after the last partition then new partitions are created.

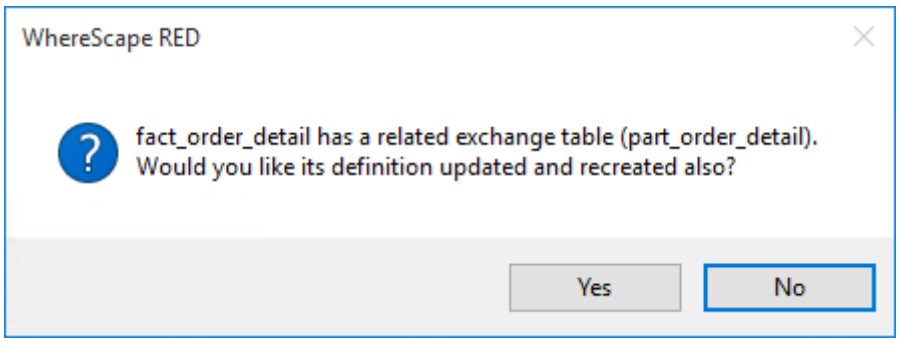
Empty partitions are created if necessary to skip periods where no data is present. For example, if we have a table partitioned by month and the latest partition is 200204 (april/2002). If data is then received for say september 2002 the update procedure will build partitions for may, june, july, august and september.

There is a variable at the start of the update procedure called v_max_skip_periods. This variable is by default set to 12. This defines the maximum number of continuous partitions we will leave empty.

From our previous example if our latest partition was april 2002 and we received data for july 2003 with no interim data then the update procedure will fail as it would have to skip 14 partitions. This check is present to prevent erroneous data from creating hundreds of partitions into the future.

NOTE: Re-Creating or Altering Partitioned Fact Tables

Recreating or Altering a partitioned fact table prompts with an offer to resync and recreate the related exchange table and indexes. This will resync the columns and indexes from the fact table to its exchange partition table so that partition swapping works correctly.



CREATING A PARTITIONED FACT TABLE IN ORACLE

In WhereScape RED a partitioned fact table can only be created from an existing fact table. The process therefore is to create a fact table normally, including the update procedure and indexes. The table is then partitioned through the **Storage** screen of the fact table's properties dialog. WhereScape RED will assist in creating the exchange table, modifying the indexes and building the procedure to handle the update of the partitions.

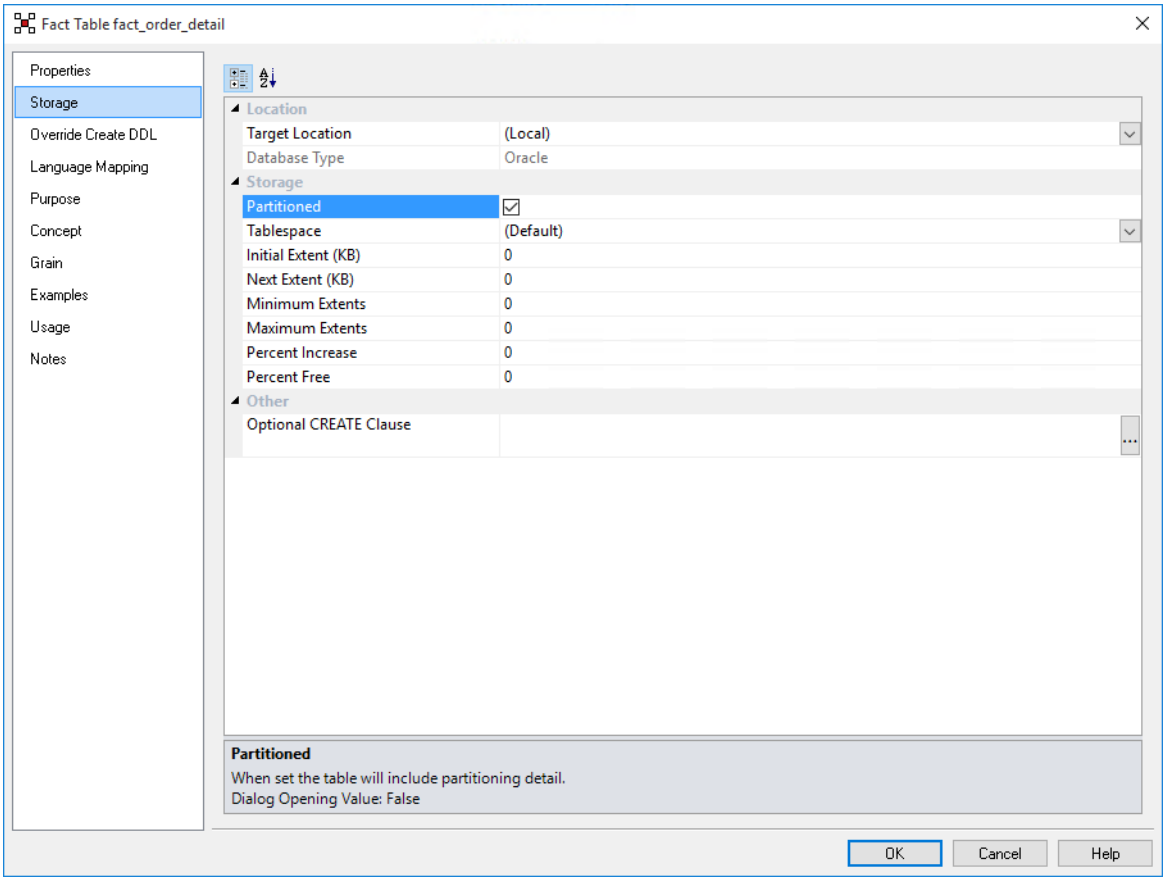
Create a detail or rollup/combined fact table

- Create the fact table that you wish to partition in the usual way.
- You must create an update procedure but do not need to load any data. The process of creating the update procedure will also build any indexes for the fact table.
- RED will be using a partition exchange table to move data to and from the partitioned fact table. This exchange table must have the same columns, column order and indexes as the fact table. It is therefore recommended to get the fact table into its final form before partitioning it.
- Add any indexes to avoid extra work later.

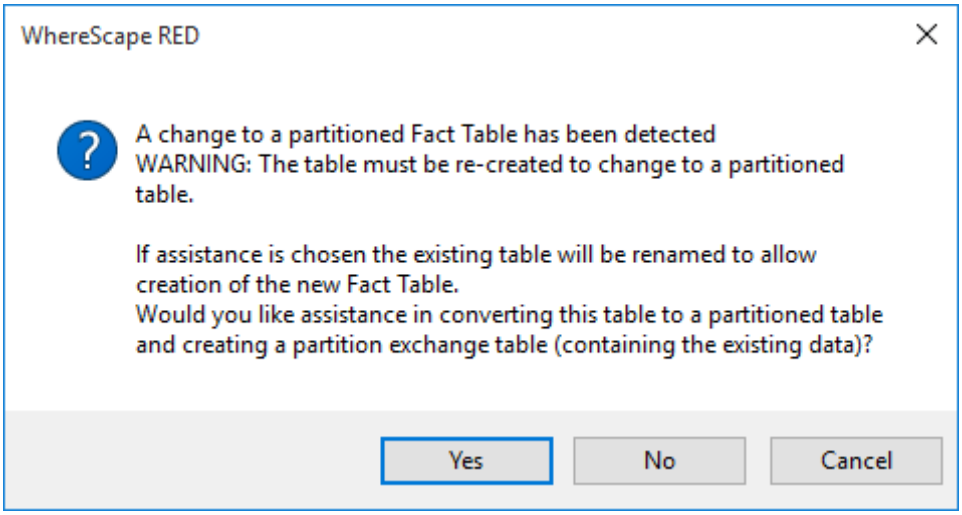
Convert to a partitioned fact table

1. Double-click on the fact table that you want to partition to bring up its properties screen.
2. Click on the **Storage** tab and tick the **Partitioned** check-box. (see example in the screenshot below)
3. A prompt will be displayed warning you that this table will need to be recreated to change the current fact to a partitioned table and offering assistance to create the partitioned table. The exchange table is then used to replace the relevant period in the fact table. The fact table remains in a queryable state throughout the process.
4. Click **Yes** to launch the wizard that will assist you through the partitioning of the table.

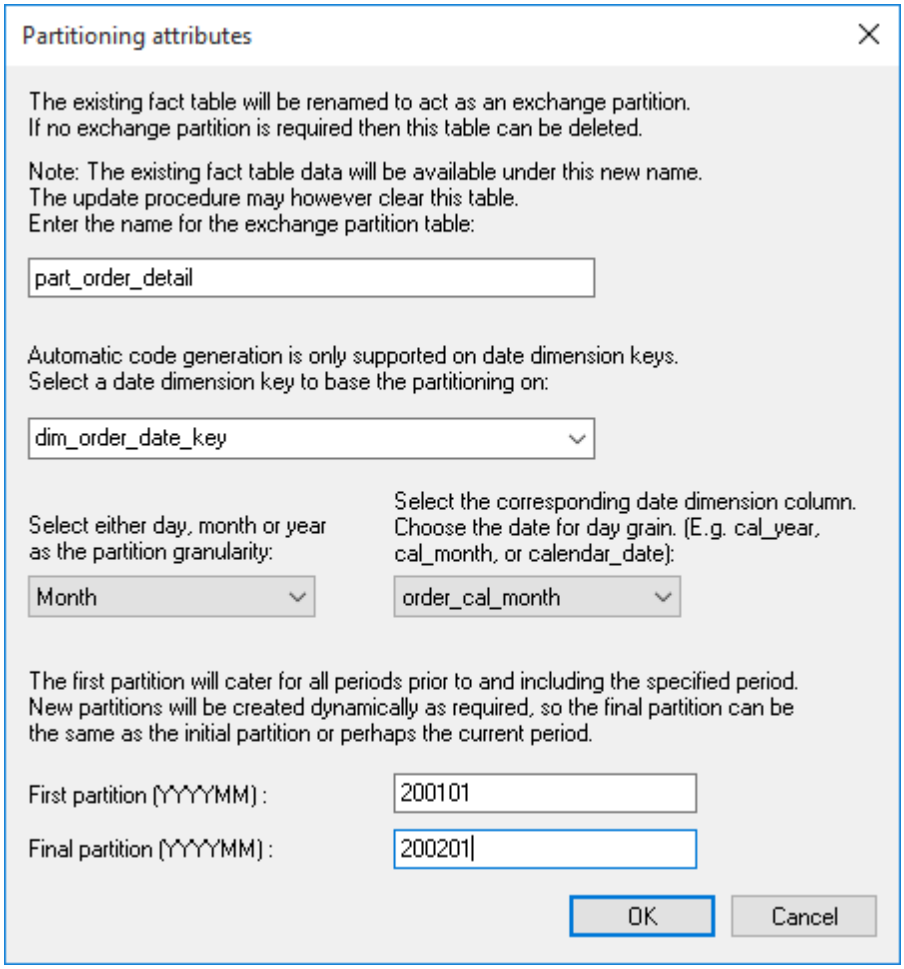
Note: Converting an existing non-partitioned table to a partitioned table cannot be done using a deployment application.
To convert non-partitioned to partitioned tables using deployment applications some manual intervention may be required to update the target databases to match the new metadata.



5 A dialog will pop-up as follows, asking for confirmation that the table is to be converted to a partitioned fact table.
If you click **Yes**, the existing fact table will be renamed to become the partition exchange table. Any current data will remain in this table.



6 The parameters for the partitioned fact table are then prompted for and are described below:



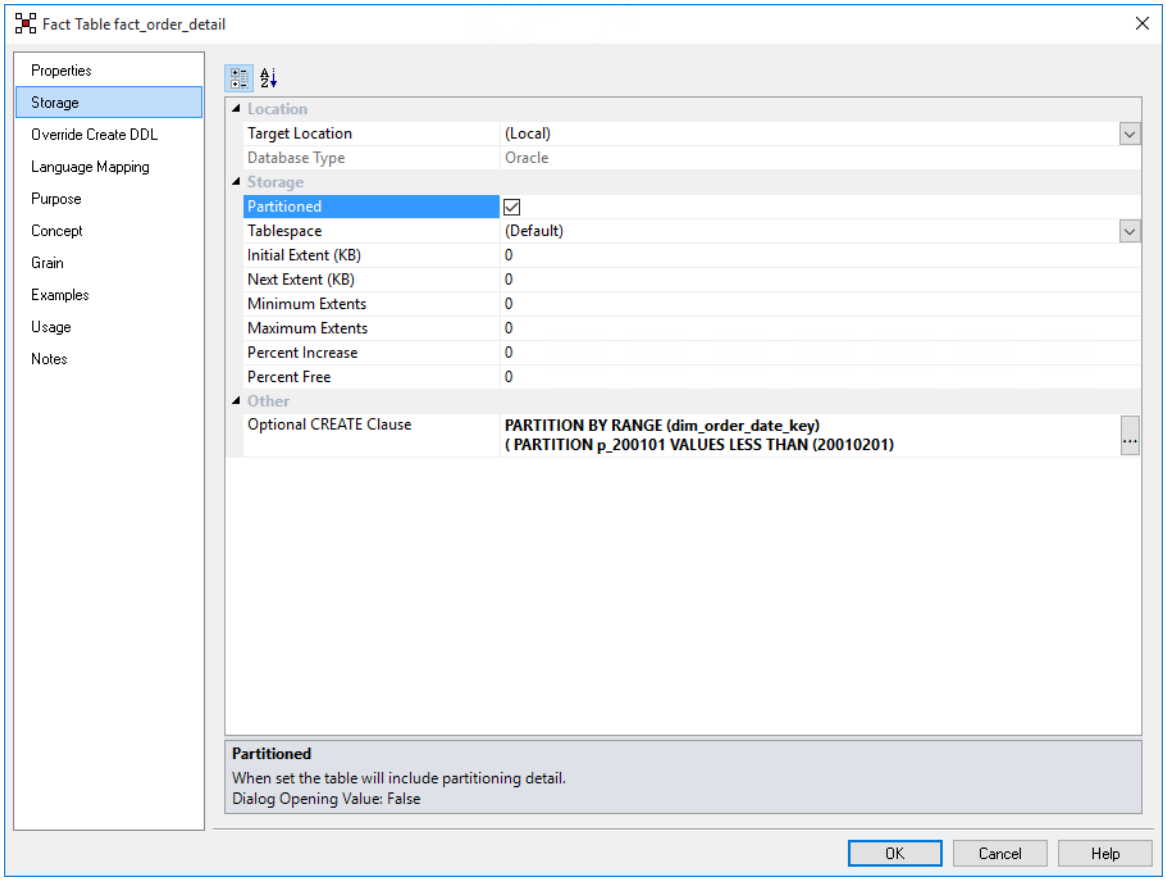
7 The partition exchange table name is prompted for. This table is identical to the fact table and is used to swap partitions to and from the fact table. Enter a **name** for the exchange partitioned table.

- The fact table will be partitioned by the date dimension key. Select a **date dimension key** that will be used as the basis for the partitioning of the fact table. It is assumed that this key has the standard WhereScape format of YYYYMMDD.
- Select a partition granularity of day, month or year along with the corresponding column in the date dimension.
  - For a **daily partition** a column with a date data type must be chosen. The date dimension key will actually be used for the partitioning but the date column is required.
  - For a **monthly partition** a column with the format YYYYMM must be chosen. In the standard WhereScape date dimension this is the column cal_month.
  - For a **yearly partition** a column with the format YYYY must be chosen. In the standard WhereScape date dimension this is the column cal_year.
- The first and last partition should then be selected. The last partition is not that important as additional partitions will be created as required. Normally just select a partition a few on from the first. The first partition is however important as any data that is dated prior to the first partition will be loaded into this first partition. The partition must be entered in the format shown. For example monthly partitions require the format YYYYMM.
- After all the relevant fields have been completed, click **OK** and the conversion process will populate the **Optional CREATE Clause** field of the table's storage properties.

Note: This field is limited in size, so if too many partitions are chosen only the first few will be added to this field. The other partitions will be created dynamically by the update procedure.

---

8 Click the **OK** button on the table's **Storage** screen to begin the process of converting the fact table, creating the new fact table and building the new update procedure.
Examine the results screen to see what has been done.



The exchange partition table will now have any current data, to populate the fact table, copy the data from partition table to the fact table, drop and recreate fact index.
The date dimension key used to base the partitioning on is not allowed to have nulls. WhereScape RED has modified the table property, however the partition table needs to be validated and recreated to be valid.
Then the fact table can be processed.

The partitioned fact table update procedure

The generated procedure will handle the creation of new partitions as required.

Any data for periods prior to the first partition is placed in the first partition. If data is received for a period after the last partition then new partitions are created.

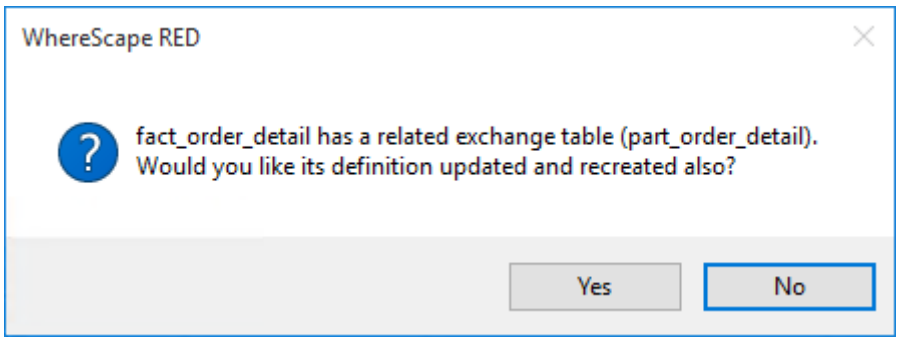
Empty partitions are created if necessary to skip periods where no data is present. For example, if we have a table partitioned by month and the latest partition is 200204 (april/2002). If data is then received for say september 2002 the update procedure will build partitions for may, june, july, august and september.

There is a variable at the start of the update procedure called v_max_skip_periods. This variable is by default set to 12. This defines the maximum number of continuous partitions we will leave empty.

From our previous example if our latest partition was april 2002 and we received data for july 2003 with no interim data then the update procedure will fail as it would have to skip 14 partitions. This check is present to prevent erroneous data from creating hundreds of partitions into the future.

NOTE: Re-Creating or Altering Partitioned Fact Tables

Recreating or Altering a partitioned fact table prompts with an offer to resync and recreate the related exchange table and indexes. This will resync the columns and indexes from the fact table to its exchange partition table so that partition swapping works correctly.



CREATING A PARTITIONED FACT TABLE IN DB2

In WhereScape RED a partitioned fact table can only be created from an existing fact table. The process therefore is to create a fact table normally, including the update procedure and indexes. The table is then partitioned through the **Storage** screen of the fact table's properties dialog. WhereScape RED will assist in creating the table, modifying the indexes and building the procedure to handle the update of the partitions.

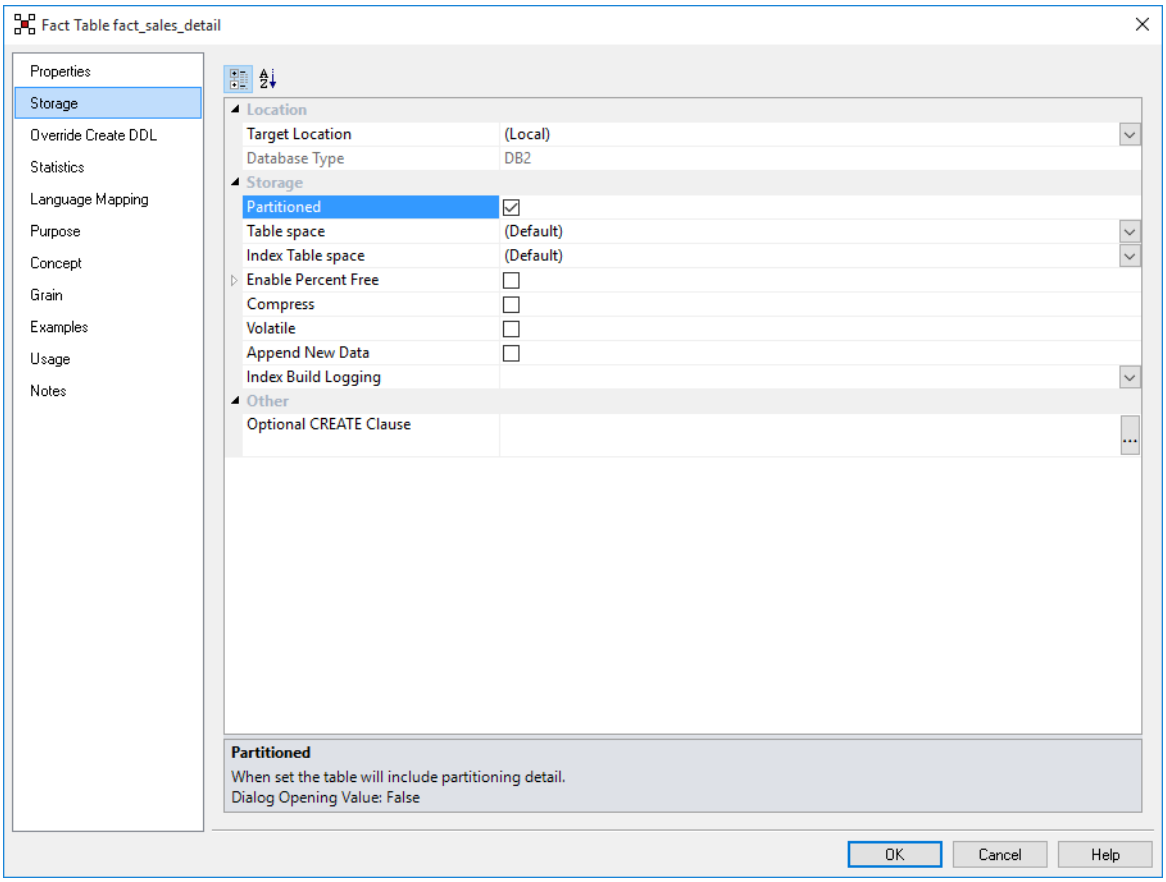
Create a detail or rollup/combined fact table

- Create the fact table that you wish to partition in the normal way.
- You must create an update procedure but do not need to load any data. The process of creating the update procedure will also build any indexes for the fact table.

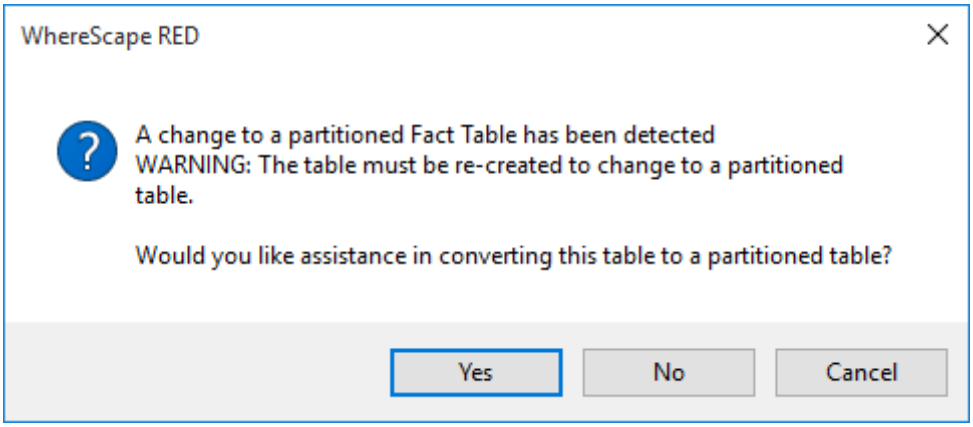
Convert to a partitioned fact table

1. Double-click on the fact table that you want to partition to bring up its properties screen.
2. Click on the **Storage** tab and tick the **Partitioned** check-box. (see example in the screenshot below)
3. A prompt will be displayed warning you that this table will need to be recreated to change the current fact to a partitioned table and offering assistance to create the partitioned table.
4. Click **Yes** to launch the wizard that will assist you through the partitioning of the table.

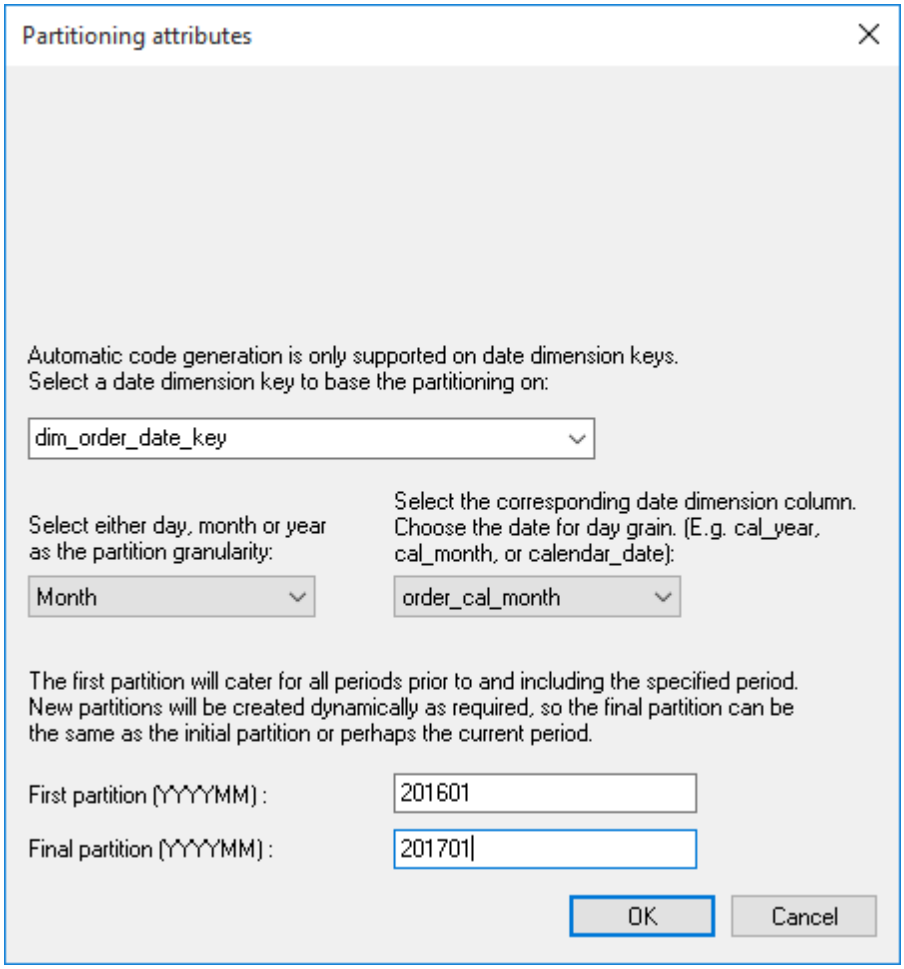
Note: Converting an existing non-partitioned table to a partitioned table cannot be done using a deployment application.
To convert non-partitioned to partitioned tables using deployment applications some manual intervention may be required to update the target databases to match the new metadata.



5 A dialog will pop-up as follows, asking for confirmation that the table is to be converted to a partitioned fact table. If **Yes** is chosen, then the existing fact table will be dropped and any current data will be lost.



6 The parameters for the partitioned fact table are then prompted for:



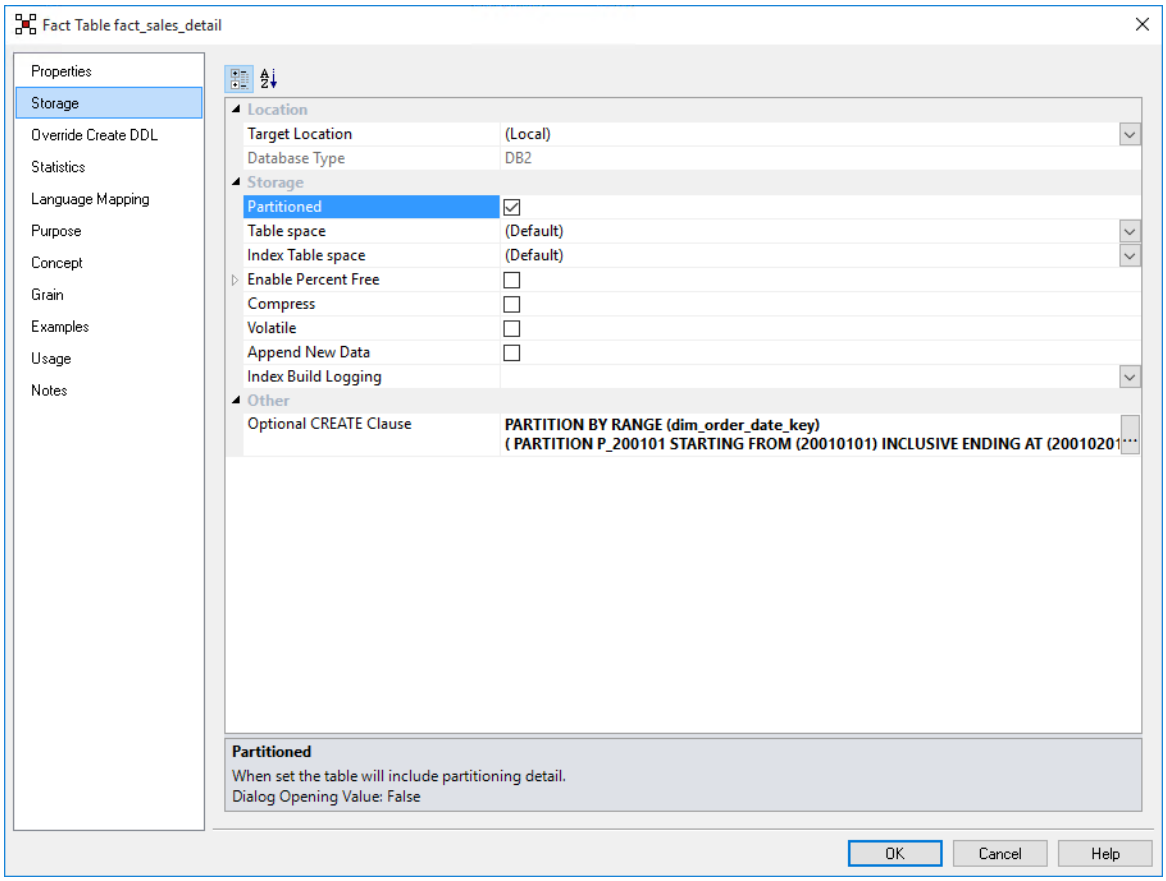
7 The fact table will be partitioned by the date dimension key. Select a **date dimension key** that will be used as the basis for the partitioning of the fact table. It is assumed that this key has the standard WhereScape format of YYYYMMDD.

- Select a partition granularity of day, month or year along with the corresponding column in the date dimension.
  - For a **daily partition** a column with a date data type must be chosen. The date dimension key will actually be used for the partitioning but the date column is required.
  - For a **monthly partition** a column with the format YYYYMM must be chosen. In the standard WhereScape date dimension this is the column cal_month.
  - For a **yearly partition** a column with the format YYYY must be chosen. In the standard WhereScape date dimension this is the column cal_year.
- The first and last partition should then be selected. The last partition is not that important as additional partitions will be created as required. Normally just select a partition a few on from the first. The first partition is however important as any data that is dated prior to the first partition will be loaded into this first partition. The partition must be entered in the format shown. For example monthly partitions require the format YYYYMM.
- After all the relevant fields have been completed, click **OK** and the conversion process will populate the **Optional CREATE Clause** field of the table's storage properties.

Note: This field is limited in size, so if too many partitions are chosen only the first few will be added to this field. The other partitions will be created dynamically by the update procedure.

---

8 Click the **OK** button on the table's **Storage** screen to begin the process of converting the fact table to be partitioned.
This drops and re-creates the fact table in DB2. The new update procedure is also generated and compiled. Examine the results screen to see what has been done.



The partitioned fact table update procedure

The generated procedure will handle the creation of new partitions as required.

Any data for periods prior to the first partition is placed in the first partition. If data is received for a period after the last partition, then new partitions are created.

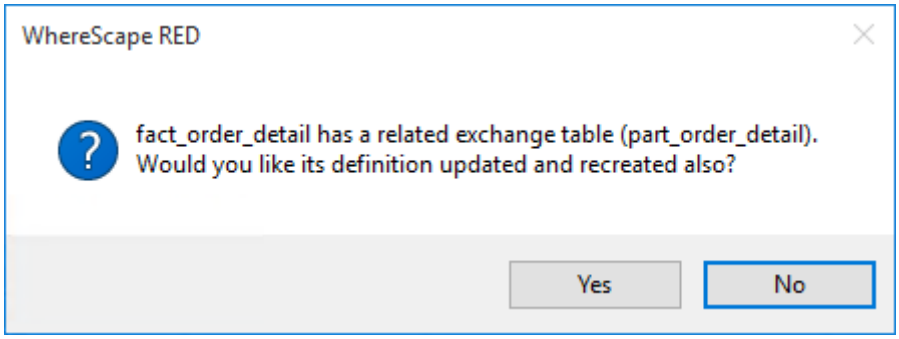
Empty partitions are created if necessary to skip periods where no data is present. For example, if we have a table partitioned by month and the latest partition is 200204 (april/2002). If data is then received for say september 2002 the update procedure will build partitions for may, june, july, august and september.

There is a variable at the start of the update procedure called v_max_skip_periods. This variable is by default set to 12. This defines the maximum number of continuous partitions we will leave empty.

From our previous example if our latest partition was april 2002 and we received data for july 2003 with no interim data then the update procedure will fail as it would have to skip 14 partitions. This check is present to prevent erroneous data from creating hundreds of partitions into the future.

NOTE: Re-Creating or Altering Partitioned Fact Tables

Recreating or Altering a partitioned fact table prompts with an offer to resync and recreate the related exchange table and indexes. This will resync the columns and indexes from the fact table to its exchange partition table so that partition swapping works correctly.



GENERATING THE PARTITIONED FACT UPDATE PROCEDURE

Once a partitioned fact table has been defined in the metadata and created in the database, an update procedure can be generated to handle the update of the fact table.

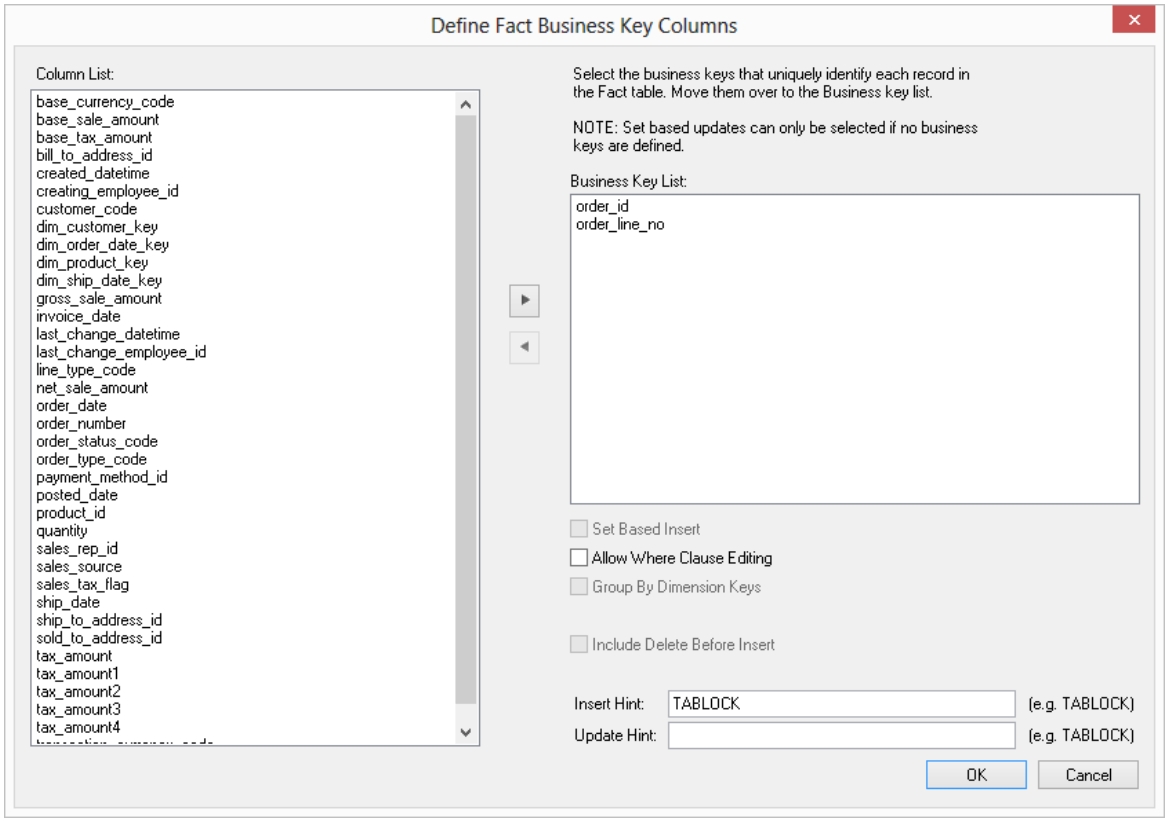
Generating a Procedure

- To generate a procedure, right-click on the fact table in the left pane and select **Properties**.
- Click on the **Rebuild** button to start the process of generating the new procedure.

A series of questions will be asked during the procedure generation based on the type of load information.

Define Fact Business Key Columns

The first dialog displayed when generating a partitioned fact table update procedure is the define fact business key columns dialog, asking for the business key that will uniquely identify each fact table record.
The source table from which the fact table is derived would normally have some form of unique constraint applied. In most cases this will be the business key.
In the example below the order_id and order_line_no are selected for the business key list.



A business key can be made up of multiple columns, but it must provide a unique identifier. Where multiple columns separately uniquely identify rows in the fact table, choose one to act as the primary business key.
For example, a source table may have a unique constraint on both a product code and a product description. Therefore, the description as well as the code must be unique.

None of the columns chosen as the business key should ever contain a **NULL** value.

Allow Where Clause Editing

If the **Allow Where Clause Editing** option is selected, then the next dialog to be displayed is the **Provide Cursor Mapping** dialog, otherwise this dialog is skipped.

Insert Hint

Enter a database hint to be used in the INSERT statement.

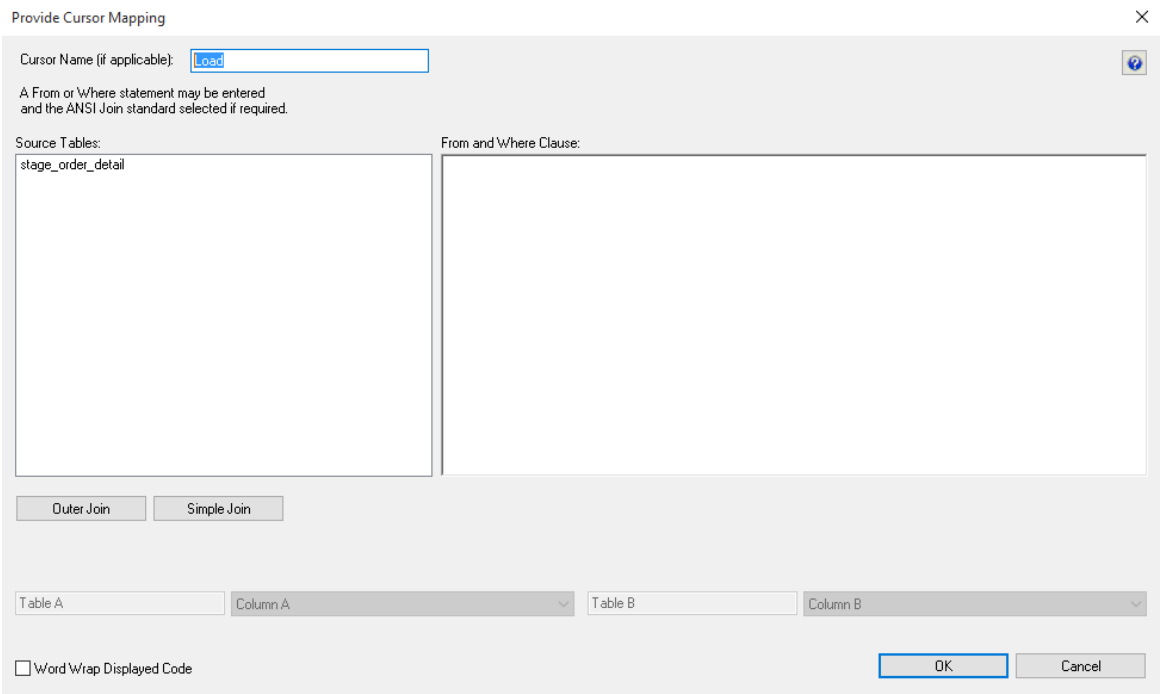
Update Hint

Enter a database hint to be used in the UPDATE statement.

Enter a database hint to be used in the UPDATE statement.

PROVIDE CURSOR MAPPING

This dialog is used to join source tables, add 'Where' clauses and specify group by clauses.



As source table joins should have been performed in the stage table, see ***Generating the Staging Update Procedure*** (on page 372) for more details.

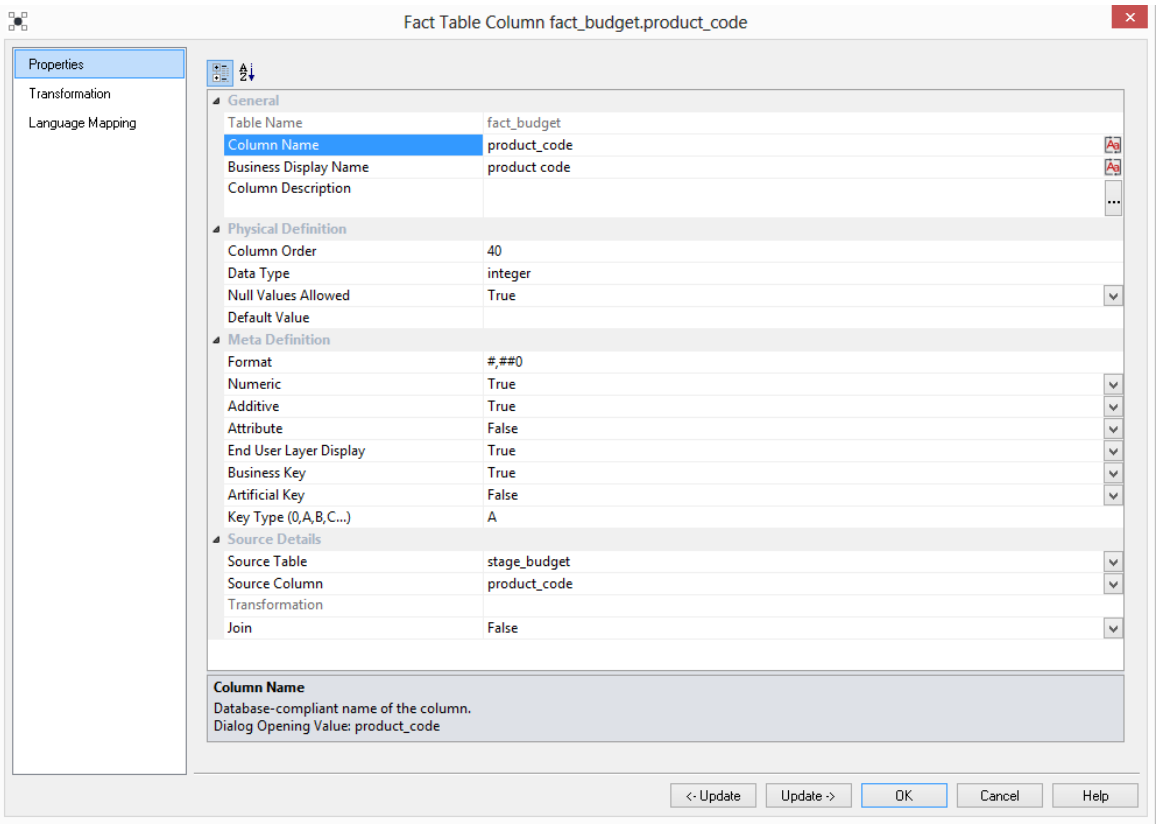
Since most of the real work is done in the stage table, this is all that is needed to build the Partitioned Fact Update Procedure.

FACT TABLE COLUMN PROPERTIES

Each fact table column has a set of associated properties. The definition of each property is defined below:
If the **Column name** or **Data type** is changed for a column then the metadata will differ from the table as recorded in the database.
Use the **Validate/Validate Table Create Status** menu option to compare the metadata to the table in the database. A right mouse menu option of Alter table is available when positioned on the table name after the validate has completed. This option will alter the database table to match the metadata definition.

TIP: If a database table's definition is changed in the metadata then the table will need to be altered in the database.
Use the **Validate/Validate Table Create Status** to compare metadata definitions to physical database tables. The option also provides the ability to alter the database table, through a pop-up menu option from the validated table name.

A sample **Properties** screen is as follows:



TIP: The two special update keys allow you to update the column and step either forward or backward to the next column's properties.
ALT-Left Arrow and **ALT-Right Arrow** can also be used instead of the two special update keys.

Table Name

Database-compliant name of the table that contains the column. [Read-only].

Column Name

Database-compliant name of the column. Typically, column-naming standards exclude the use of spaces etc. A good practice is to only use alphanumerics, and the underscore character. Changing this field alters the table's definition.

Note: A case conversion button on the right converts the text between different cases: UPPER CASE, Capitalized Case and lower case. The mode cycles to the next case in the sequence each conversion.

Business Display Name

Name that the business uses to refer to the column, which is included in the RED-generated documentation and can be used in the end user layer of other tools. [Does NOT affect the physical database table]. As such it is a free form entry and any characters are valid.

Note: A case conversion button on the right converts the text between different cases: UPPER CASE, Capitalized Case and lower case. The mode cycles to the next case in the sequence each conversion.

Column Description

This field contains the description for the column. It may be a description from a business user's point of view. This field might additionally contain information on where and how the column was acquired. For example, if the column is sourced from multiple tables or is a composite or derived column then this definition would normally describe the process used to populate the column. This field is used in the documentation and is available via the view **ws_admin_v_dim_col .** This field is also stored as a comment against the column in the database.

Column Order

Numeric value that controls the relative order of columns in the database create statement. The lowest numbered column will appear first in the table. Although this affects the physical table definition no action will be taken unless the table is re-created in the database. The columns can be re-numbered based on the existing order by choosing the **Respace Order Number** pop-up menu option when positioned over any column in the table. This action will number the columns in increments of 10 starting at 10. In addition to a simple change of the order field, the column order can be changed by first displaying the columns in the middle pane and then using drag and drop to move the columns around. This drag and drop process will automatically renumber the columns as required.

Data Type

Database-compliant data type that must be valid for the target database. Typical data types for Oracle are integer, number, char(), varchar2() and date. For SQL Server common types are integer, numeric, varchar() and datetime. See the database documentation for a description of the data types available. Changing this field alters the table's definition.

Null Values Allowed

Determines whether the table column can hold NULL values or whether a value is always mandatory.

Default Value

Initial value that is assigned to the column when a row is inserted into the table but no value is specified for the column.

Format

Optional format mask that can be used in end user tools. [Does NOT affect the physical database table]. As such it is a free form entry and any characters are valid. Typically format masks are only used on numeric fields. Example: #,###0.00. It is not worth the effort of populating this field unless it can be utilized by the end user tools in use.

Numeric

Indicates whether the table column holds values that are numeric. This is normally only relevant for fact tables. It does not affect the physical table definition, but rather provides input to the view **ws_admin_v_dim_col** which can be used to assist in the population of an end user tool's end user layer. The use of this field is not relevant unless it can be utilized by the end user tools.

Additive

Indicates whether the table column holds values that are additive. This implies that the column can be summed when performing data grouping in a query. This is normally only relevant for fact tables. It does not affect the physical table definition, but rather provides input to the view **ws_admin_v_dim_col** which can be used to assist in the population of an end user tool's end user layer. The use of this field is not relevant unless it can be utilized by the end user tools.

Attribute

Indicates whether the table column holds values that are descriptive, and/or are used for grouping/summing. An attribute is defined as a column that is non factual. For example, we may have an order number, or a invoice number stored in the fact table. Such columns are considered attributes, rather than facts. This checkbox is therefore normally only relevant for fact tables. This checkbox does not affect the physical table definition, but rather provides input to the view **ws_admin_v_dim_col** which can be used to assist in the population of an end user tools end user layer. The use of this field is not relevant unless it can be utilized by the end user tools.

End User Layer Display

Indicates whether the table column is available/visible to end users. If set the documentation will include the column in the glossary and in the user documentation. It is also used to decide what columns appear in the view **ws_admin_v_dim_col.** Typically columns such as the artificial key would not be enabled for end user display.

Business Key

Indicates whether the column is part of the BUSINESS key, which is defined during the update procedure build. [Normally maintained automatically]. Multiple columns can form the primary business key.

Key Type

Key type that is assigned and used when generating the table's update procedure and indexes. [Normally maintained automatically]. It can be altered here, but this should only be done if the consequences are fully understood and tested.
The supported values are:

Key type	Meaning
0	The artificial key. Set when the key is added during drag and drop table generation.
1	Component of all business keys. Indicates that this column is used as part of any business key. For example: By default, the dss_source_system_key is added to every dimension table. It is considered part of any lookup on that table and has the key type set to 1. Set when the column is added during drag and drop table generation.
2	Indicates that this column is a dimensional join. Used on fact tables to indicate the dimension keys. Results in bitmap indexes being built for the columns. Set during the update procedure generation for a fact table, based on information from the staging table.
3	Slowly changing column indicator. Used on dimension tables to indicate that the column is being managed as a slowly changing column within the context of a slowly changing dimension. Set when a column is identified during the dimension update procedure generation.
4	Previous value column indicator. Used on dimension tables to indicate that the column is being managed as a previous value column. The source column identifies the parent column. Set during the dimension creation.
A	Indicates that the column is part of the primary business key. Set whenever a business key is defined as part of an update procedure generation.
B-Z	Indicates that the column is part of a secondary business key. Only used during index generation and not normally set.

KPI Column Type

Only used by **KPI** fact tables. This field defines the column type for the KPI Fact Table. Refer to the KPI table creation section for more details on this field.

Source Table

Identifies the source table where the column's data comes from. This source table is normally a stage table within the data warehouse. If the column was sourced from multiple tables, then the normal practice is to record one of the tables in this field and a comment listing all of the other tables in the **Source strategy** field. This field is used when generating a procedure to update the fact table. It is also used in the track back diagrams and in the documentation.

Source Column

Identifies the source column where the column's data comes from. Such a column is normally a stage table column, which in turn may have been a transformation or the combination of multiple columns. This may also be a dimensional key where a dimension is being joined.

Transformation

Transformation. [Read-only].

Join

Indicates whether the table column is used in a table join. [Normally maintained automatically but can be optionally changed to override the default join logic used in the generated update procedure]. The **Source table** and **Source column** fields will provide the dimension table's side of the join. The options for this field are: False, True, Manual and Pre Join.

Setting this field to Manual changes the way the dimension table is looked up during the update procedure build. It allows you to join the dimension manually in the Cursor mapping dialog (where the 'Where' clause is built).

Setting this field to Pre Join activates the **Pre Join Source Table** field and allows you to select a table from the drop-down list.

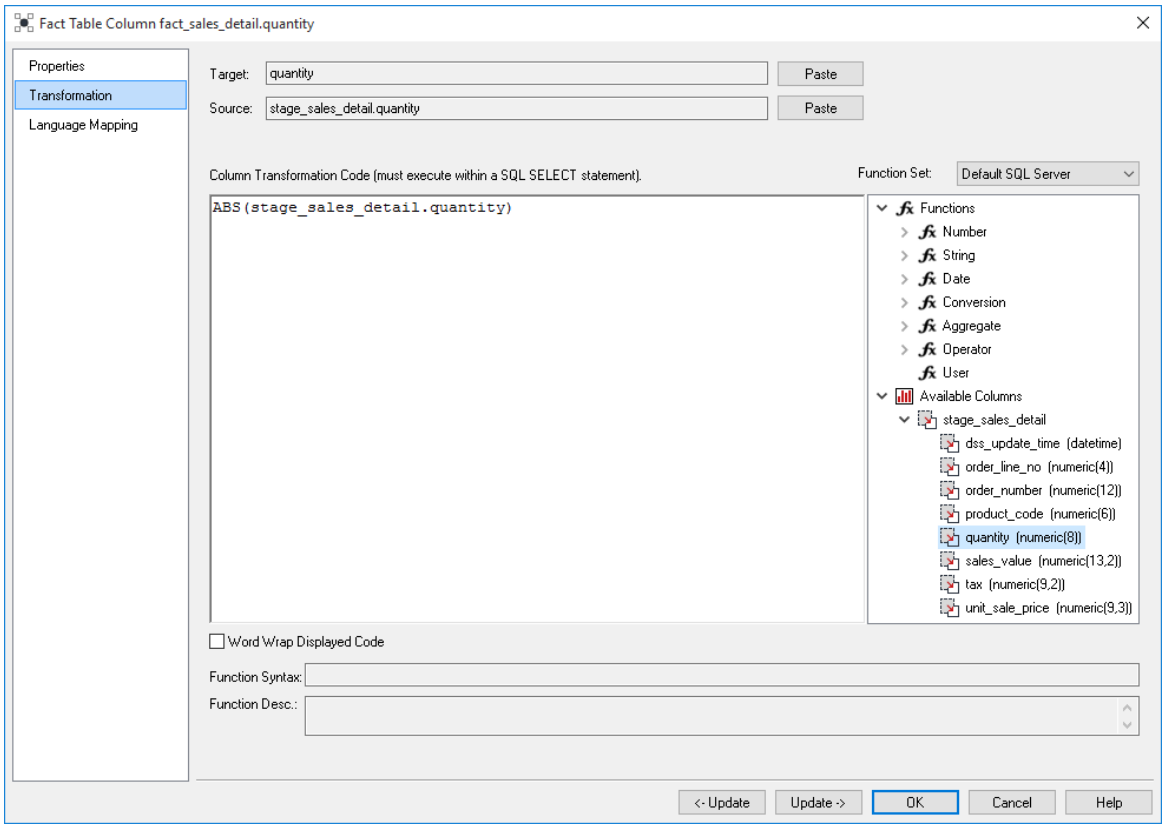
Pre Join Source Table

Indicates the table from which the pre joined column was sourced. When the Join option is set to False, this field becomes inactive. When the Join option is set to True or Manual, this field is set to the current table name. When the Join option is set to Pre Join, then you can select the required table from the drop-down list.

FACT TABLE COLUMN TRANSFORMATIONS

Each fact table column can have a transformation associated with it. The transformation will be included in the generated procedure and will be executed as part of the procedure update.
The transformation must therefore be a valid SQL construct that can be included in a **Select** statement.
For example, we could have a transformation of 'load_order_line.qty * 0.125' to calculate a tax column of 12.5%.

Click the **Transformation** tab to enter a transformation.



Note: Transformations are only put into effect when the procedure is re-generated.

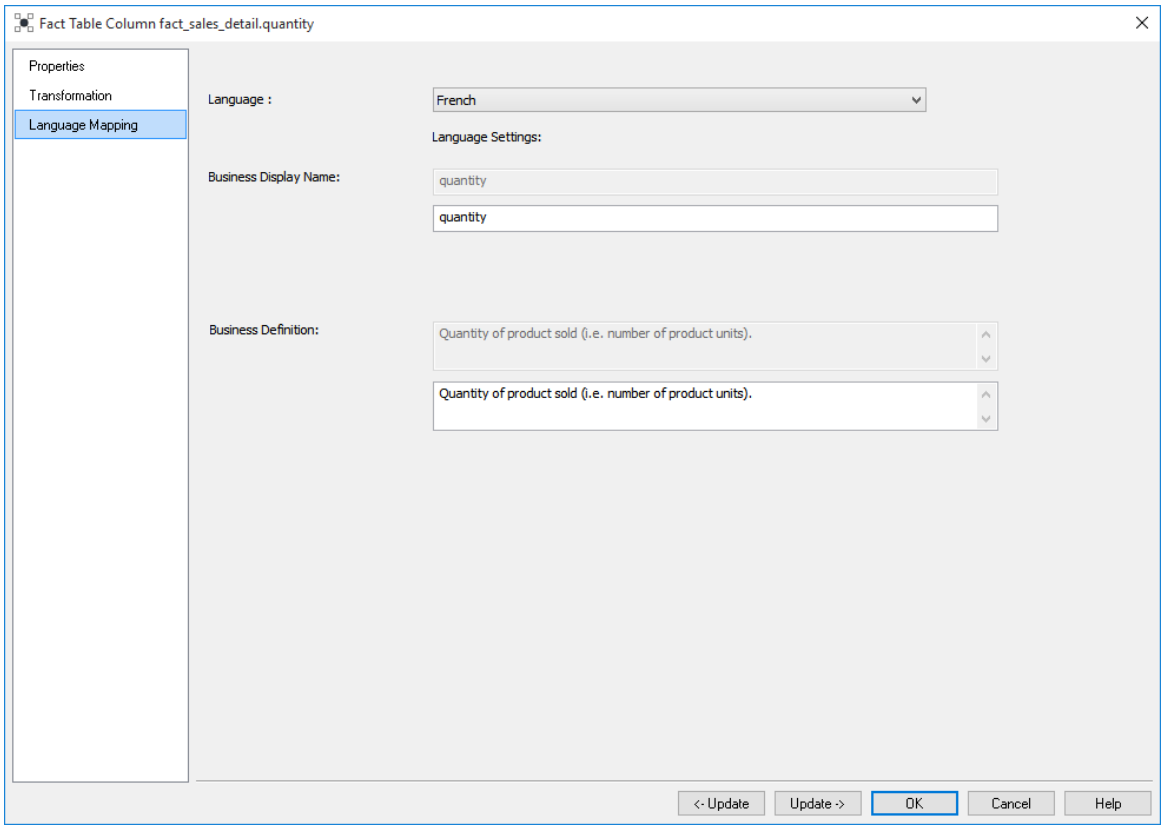
Microsoft Analysis Services 2005+ Tabular Mode Tables: For Tabular Mode table column transformations, **Default DAX** is the only applicable Function Set for **after load** transformations.

See ***Transformations*** (on page 650) for more details.

FACT TABLE LANGUAGE MAPPING

The Fact Properties screen has a tab called **Language Mapping**.

Select the language from the drop-down list and then enter the translations for the **Business Display Name** and the **Description** in the chosen language.
The translations for these fields can then be pushed through into OLAP cubes.



CHAPTER 16
AGGREGATION

Two types of aggregate tables are discussed.

The first is where all non-additive facts and one or more dimensions are removed from a fact table. Typically, this results in a smaller table that can answer a subset of the queries that could be posed against the fact table. This aggregate table still maintains full integrity to the remaining dimensions, and consequently reflects all changes to those dimensions.

The second type, we will call an aggregate summary, or summary table. This table includes additive measures and in some cases hierarchical elements of one or more of the dimensions providing a rolled-up summary of the fact table data. For example, we may choose to deal at product group level rather than product SKU which is the granularity of the dimension.

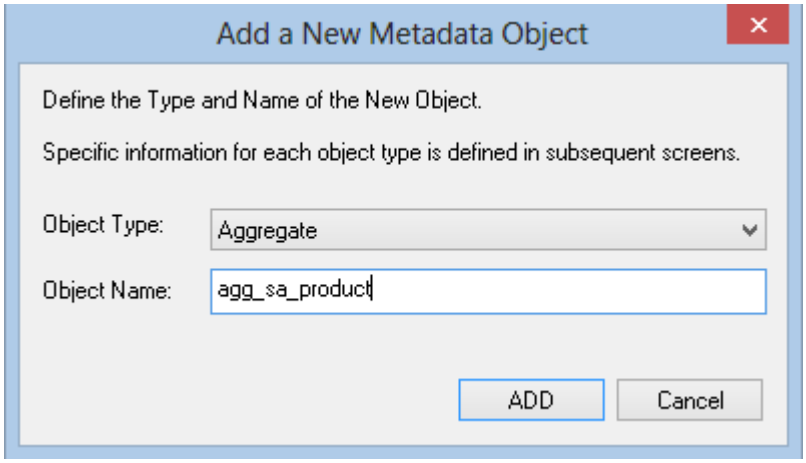
When using the Oracle database, the aggregate tables are seen as pre-built materialized views with reduced precision and query rewrite enabled.

IN THIS CHAPTER

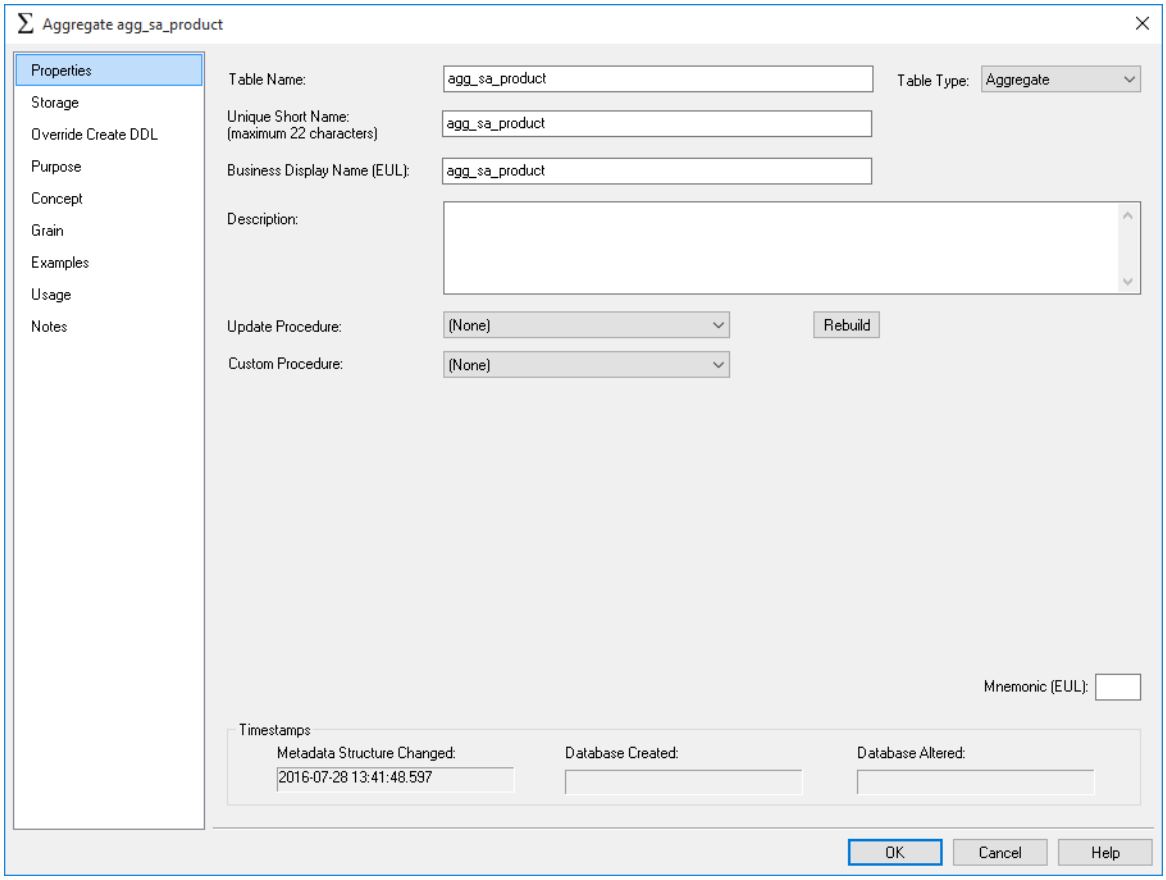
CREATING AN AGGREGATE TABLE

1 In the left pane double-click on the **aggregate** group to list the aggregates in the middle pane and set aggregates as the drop target.

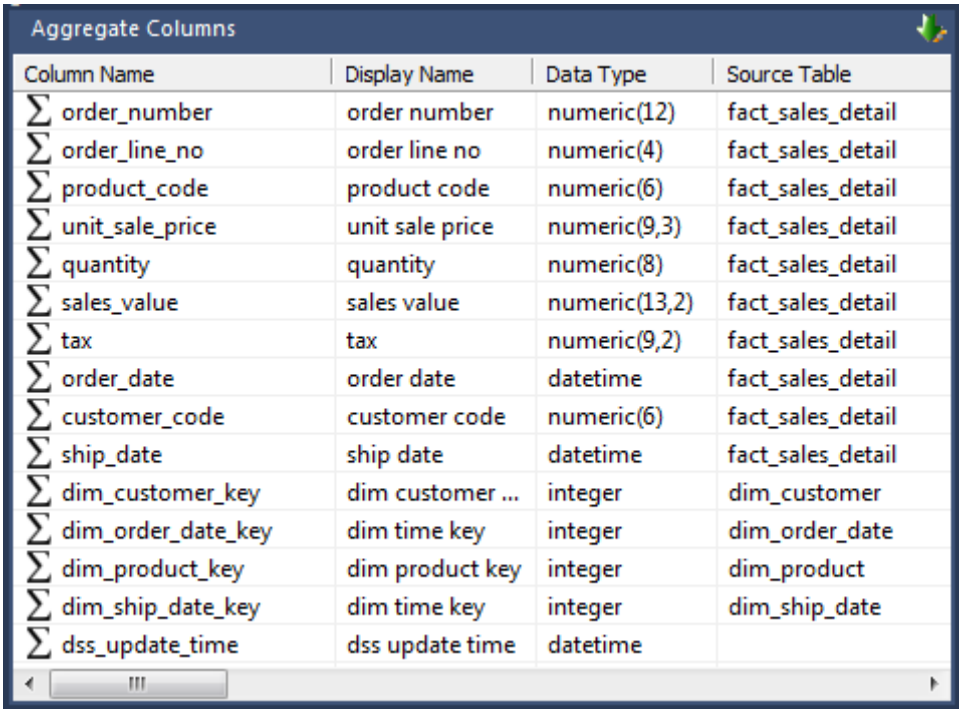
2 From the Data Warehouse browse (right) pane drag a fact table into the middle pane and enter the aggregate name. Click **ADD**.



3 The aggregate properties are displayed. Click **OK**.

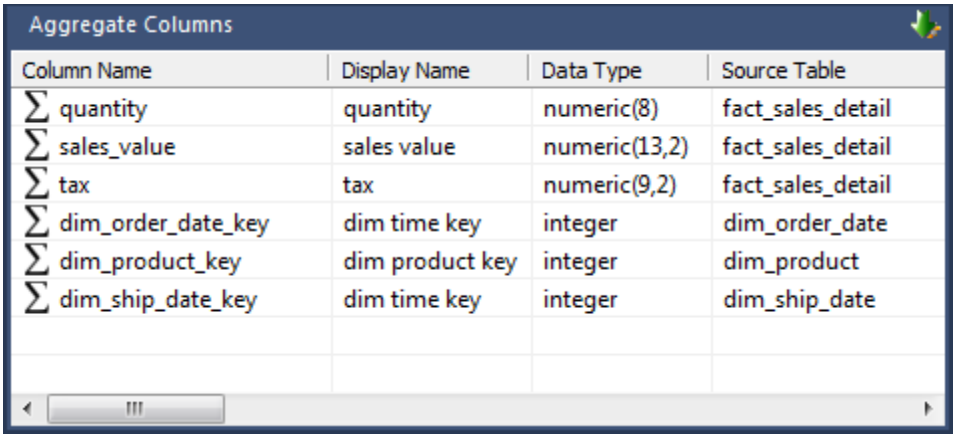


4 The list of columns in the Aggregate is displayed in the middle pane:



Column Name	Display Name	Data Type	Source Table
order_number	order number	numeric(12)	fact_sales_detail
order_line_no	order line no	numeric(4)	fact_sales_detail
product_code	product code	numeric(6)	fact_sales_detail
unit_sale_price	unit sale price	numeric(9,3)	fact_sales_detail
quantity	quantity	numeric(8)	fact_sales_detail
sales_value	sales value	numeric(13,2)	fact_sales_detail
tax	tax	numeric(9,2)	fact_sales_detail
order_date	order date	datetime	fact_sales_detail
customer_code	customer code	numeric(6)	fact_sales_detail
ship_date	ship date	datetime	fact_sales_detail
dim_customer_key	dim customer ...	integer	dim_customer
dim_order_date_key	dim time key	integer	dim_order_date
dim_product_key	dim product key	integer	dim_product
dim_ship_date_key	dim time key	integer	dim_ship_date
dss_update_time	dss update time	datetime	

5 Remove any columns that will not make sense at an aggregated level. For example, dss_fact_table_key, any business keys, any non-additive facts, any measures that relate to detail (e.g, unit price) and any dimension keys not required:



Column Name	Display Name	Data Type	Source Table
quantity	quantity	numeric(8)	fact_sales_detail
sales_value	sales value	numeric(13,2)	fact_sales_detail
tax	tax	numeric(9,2)	fact_sales_detail
dim_order_date_key	dim time key	integer	dim_order_date
dim_product_key	dim product key	integer	dim_product
dim_ship_date_key	dim time key	integer	dim_ship_date

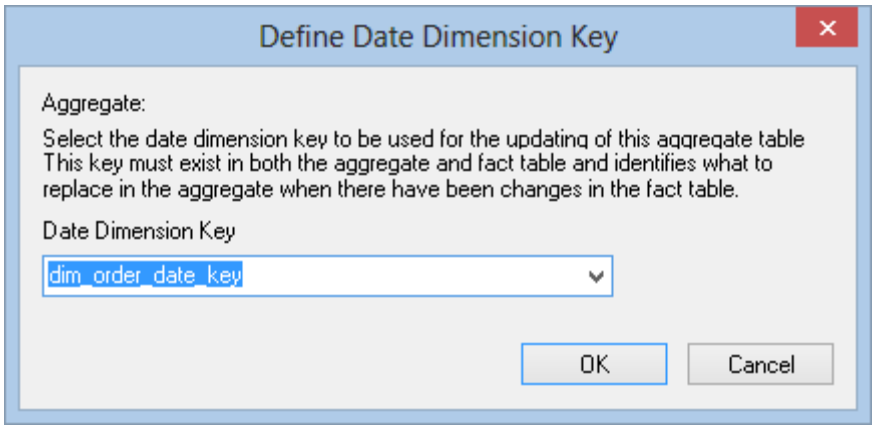
Note: All aggregate table columns (other than dss columns) should always have a source table specified, even if a column transformation is in use.

6 Create the aggregate table in the database by right-clicking on the aggregate and selecting **Create(ReCreate)**.

7 Create a procedure to update the aggregate by right-clicking on the aggregate, selecting **Properties** and selecting **(Build Procedure...)** in the Update Procedure field.

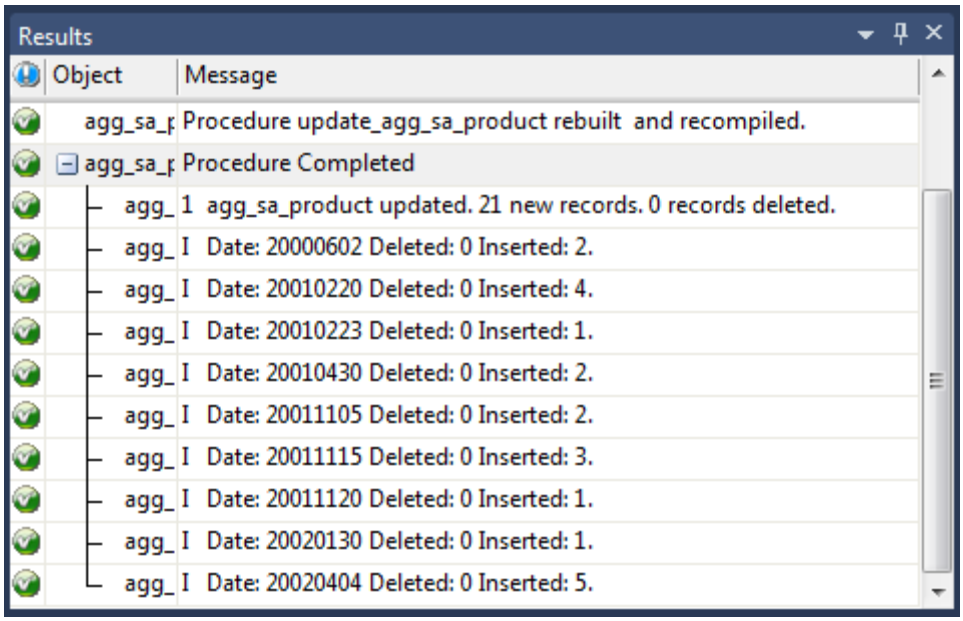
Note: WhereScape RED generated update procedures for aggregates are incremental. Incremental updates are based on a date dimension key and number of look back days. The aggregate update process looks at any records that have been updated in the fact table in the last 7 days (by default).

To support this, a date dimension key must be selected. The columns chosen must be in both the source fact table and the aggregate. Select this column and click **OK**.



8 Update the table by right-clicking and choosing **Execute Update Procedure.**

Note: Any column transformations added to a measure column of an aggregate table must always include an aggregate function, usually **SUM**. For example, an ISNULL added to forecast_quantity should be entered as: SUM(ISNULL(forecast_quantity,0)).



Oracle: When the procedure runs it first removes the materialized view, then updates any changes from the fact table into the aggregate. It then re-establishes the materialized view, enabling query rewrite.

DB2: When the procedure runs it first removes the materialized query table, then updates any changes from the fact table into the aggregate. It then re-establishes the materialized query table, enabling query rewrite.

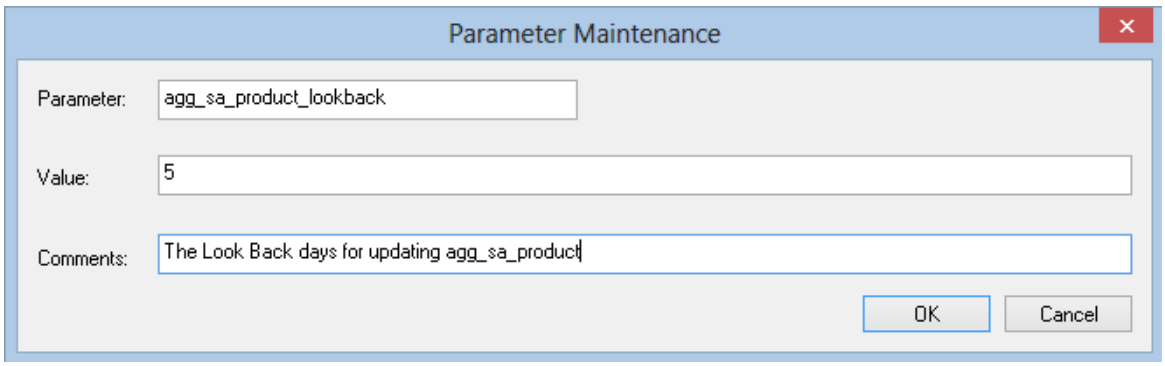
CHANGE AN AGGREGATES DEFAULT LOOK BACK DAYS

WhereScape RED generate update procedures for aggregates that are incremental. Incremental updates are based on a date dimension key and number of look back days.

The aggregate update process looks at any records that have been updated in the fact table in the last number of look back days.

The default number of look back days is 7. The default is set in each update procedure. To change the number of look back days for an aggregate table, create a WhereScape RED parameter called AggregateName_lookback and enter the required number of look back days as the parameter value.

For Example, add the following parameter to change the look back days to 5 for agg_sa_product:

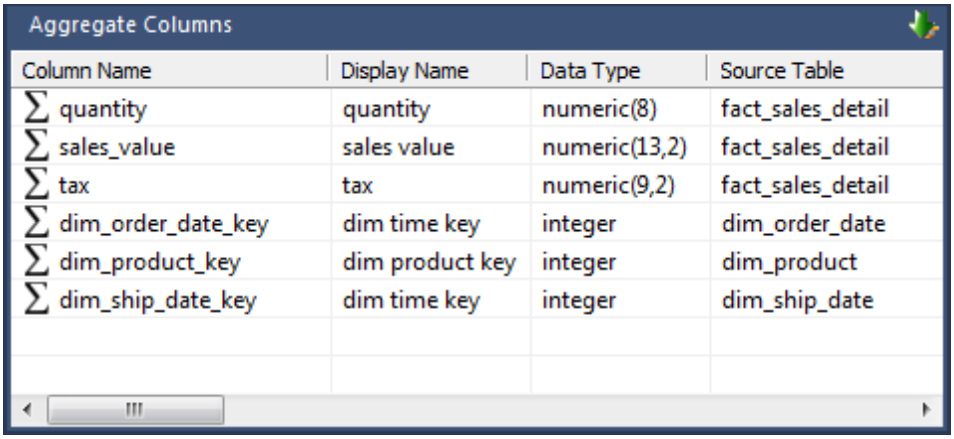


See ***Parameters*** (on page 139) for more information on WhereScape RED parameters.

CREATING AN AGGREGATE SUMMARY TABLE

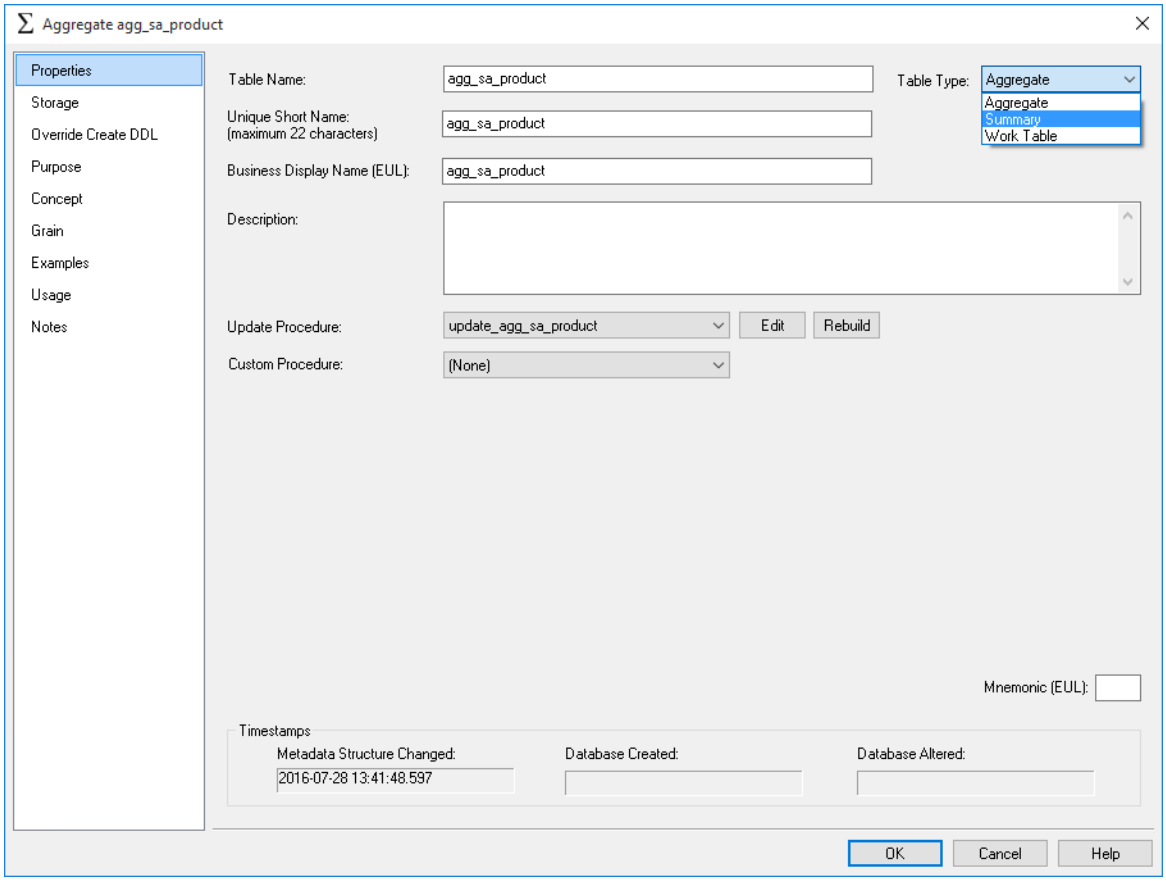
The creation of a summary table proceeds initially in the same way as an aggregate table.

1. In the left pane double-click on the **aggregate** group to list the aggregates in the middle pane and set aggregates as the drop target.
2. From the Data Warehouse browse (right) pane drag a fact table into the middle pane. Remove any columns that will not make sense at an aggregated level. For example, dss_fact_table_key, any business keys, any non-additive facts, any measures that relate to detail (e.g. unit price).
3. Drag over columns from dimensions that are linked to the fact table. Delete the dimension keys to allow a rollup to the level of the dimension elements.



Note: All aggregate summary table columns (other than dss columns) should always have a source table specified, even if a column transformation is in use.

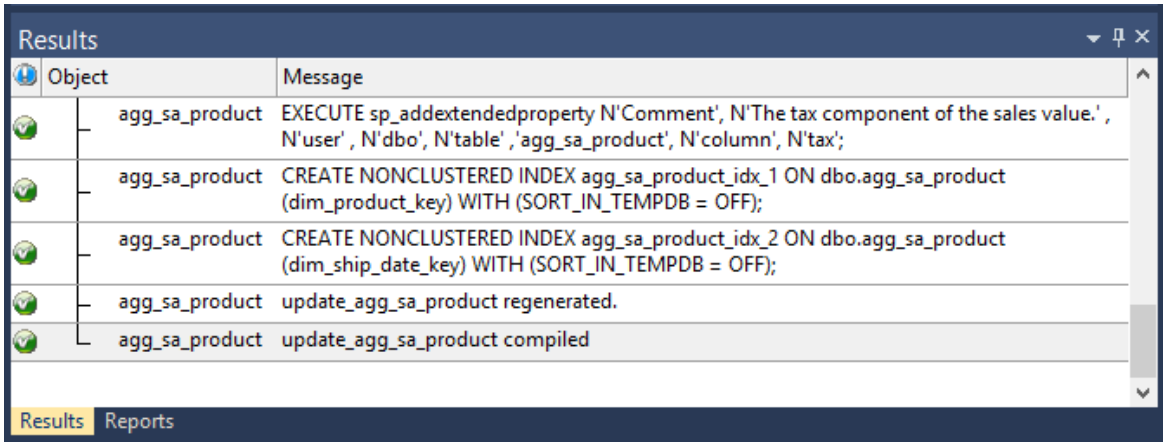
4 In the Properties of the aggregate table change the **Table Type** to **Summary**:



5 Create the aggregate summary table in the database by right-clicking on the aggregate and selecting **Create(ReCreate)**.

6 Create a procedure to update the aggregate summary by right-clicking on the aggregate, selecting **Properties** and selecting **(Build Procedure...)** in the Update Procedure field. The aggregate summary table is totally rebuilt each time the procedure is executed.

Note: Any column transformations added to a measure column of an aggregate summary table must always include an aggregate function, usually **SUM**. For example, an ISNULL added to forecast_quantity should be entered as: SUM(ISNULL(forecast_quantity,0)).



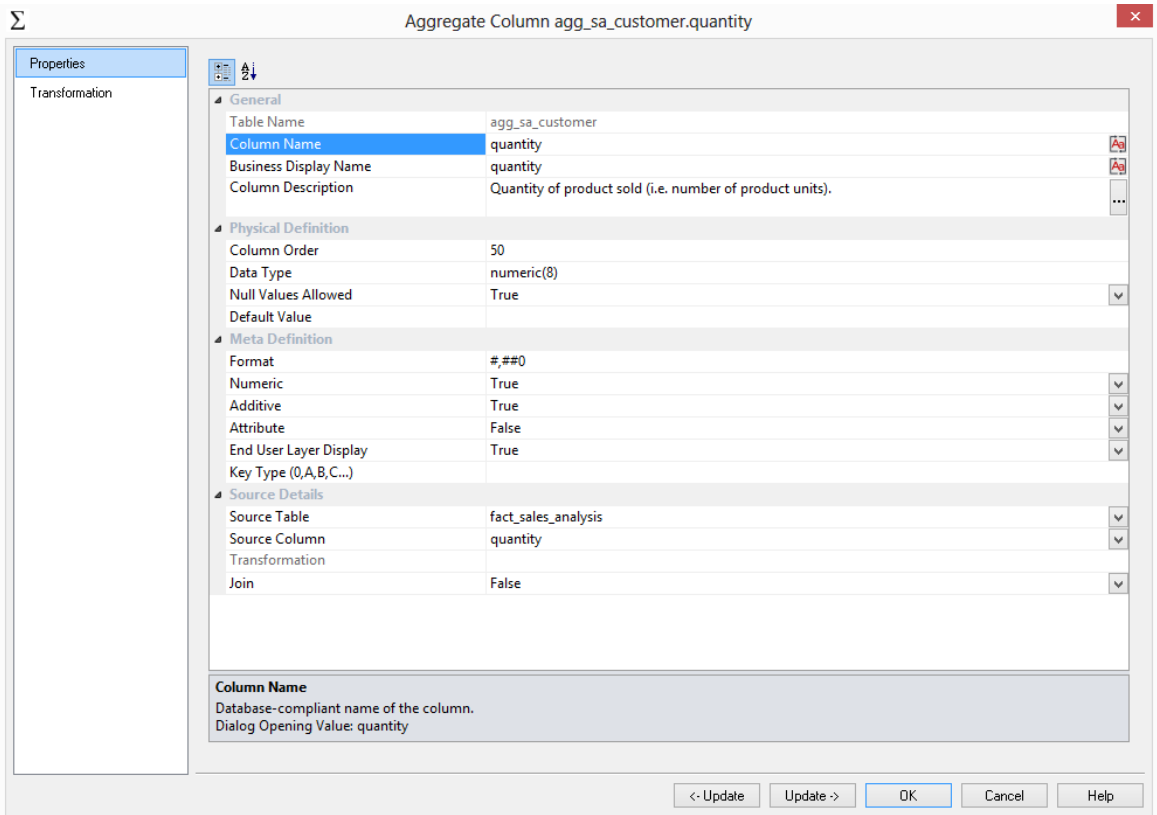
AGGREGATE TABLE COLUMN PROPERTIES

Each aggregate table column has a set of associated properties. The definition of each property is defined below:

If the **Column name** or **Data type** is changed for a column then the metadata will differ from the table as recorded in the database. Use the **Validate/Validate Table Create Status** menu option to compare the metadata to the table in the database. A right mouse menu option of **Alter Table** is available when positioned on the table name after the validate has completed. This option will alter the database table to match the metadata definition.

TIP: If a database table's definition is changed in the metadata then the table will need to be altered in the database. Use the **Validate/Validate Table Create Status** to compare metadata definitions to physical database tables. The option also provides the ability to alter the database table, through a pop-up menu option from the validated table name.

A sample **Properties** screen is as follows:



The two special update keys allow you to update the column and step either forward or backward to the next column's properties. **ALT-Left Arrow** and **ALT-Right Arrow** can also be used instead of the two special update keys.

Table Name

Database-compliant name of the table that contains the column. [Read-only].

Column Name

Database-compliant name of the column. Typically, column-naming standards exclude the use of spaces etc. A good practice is to only use alphanumerics, and the underscore character. Changing this field alters the table's definition.

Note: A case conversion button on the right converts the text between different cases: UPPER CASE, Capitalized Case and lower case. The mode cycles to the next case in the sequence each conversion.

Business Display Name

Name that the business uses to refer to the column, which is included in the RED-generated documentation and can be used in the end user layer of other tools. [Does NOT affect the physical database table]. As such it is a free form entry and any characters are valid.

Note: A case conversion button on the right converts the text between different cases: UPPER CASE, Capitalized Case and lower case. The mode cycles to the next case in the sequence each conversion.

Column Description

This field contains the description for the column. It may be a description from a business user's point of view. This field might additionally contain information on where and how the column was acquired. For example, if the column is sourced from multiple tables or is a composite or derived column then this definition would normally describe the process used to populate the column. This field is used in the documentation and is available via the view **ws_admin_v_dim_col .** This field is also stored as a comment against the column in the database.

Column Order

Numeric value that controls the relative order of columns in the database create statement. The lowest numbered column will appear first in the table. Although this affects the physical table definition no action will be taken unless the table is re-created in the database. The columns can be re-numbered based on the existing order by choosing the **Respace Order Number** pop-up menu option when positioned over any column in the table. This action will number the columns in increments of 10 starting at 10. In addition to a simple change of the order field, the column order can be changed by first displaying the columns in the middle pane and then using drag and drop to move the columns around. This drag and drop process will automatically renumber the columns as required.

Data Type

Database-compliant data type that must be valid for the target database. Typical data types for Oracle are integer, number, char(), varchar2() and date. For SQL Server common types are integer, numeric, varchar() and datetime. See the database documentation for a description of the data types available. Changing this field alters the table's definition.

Null Values Allowed

Determines whether the table column can hold NULL values or whether a value is always mandatory.

Default Value

Initial value that is assigned to the column when a row is inserted into the table but no value is specified for the column.

Format

Optional format mask that can be used in end user tools. [Does NOT affect the physical database table]. As such it is a free form entry and any characters are valid. Typically format masks are only used on numeric fields. Example: #,###0.00. It is not worth the effort of populating this field unless it can be utilized by the end user tools in use.

Numeric

Indicates whether the table column holds values that are numeric. This is normally only relevant for fact tables. It does not affect the physical table definition, but rather provides input to the view **ws_admin_v_dim_col** which can be used to assist in the population of an end user tool's end user layer. The use of this field is not relevant unless it can be utilized by the end user tools.

Additive

Indicates whether the table column holds values that are additive. This implies that the column can be summed when performing data grouping in a query. This is normally only relevant for fact tables. It does not affect the physical table definition, but rather provides input to the view **ws_admin_v_dim_col** which can be used to assist in the population of an end user tool's end user layer. The use of this field is not relevant unless it can be utilized by the end user tools.

Attribute

Indicates whether the table column holds values that are descriptive, and/or are used for grouping/summing. An attribute is defined as a column that is non factual. For example, we may have an order number, or a invoice number stored in the fact table. Such columns are considered attributes, rather than facts. This checkbox is therefore normally only relevant for fact tables. This checkbox does not affect the physical table definition, but rather provides input to the view **ws_admin_v_dim_col** which can be used to assist in the population of an end user tools end user layer. The use of this field is not relevant unless it can be utilized by the end user tools.

End User Layer Display

Indicates whether the table column is available/visible to end users. If set the documentation will include the column in the glossary and in the user documentation. It is also used to decide what columns appear in the view **ws_admin_v_dim_col.** Typically columns such as the artificial key would not be enabled for end user display.

Key Type

Key type that is assigned and used when generating the table's update procedure and indexes. [Normally maintained automatically]. It can be altered here, but this should only be done if the consequences are fully understood and tested. The supported values are:

Key type	Meaning
0	The artificial key. Set when the key is added during drag and drop table generation.
1	Component of all business keys. Indicates that this column is used as part of any business key. For example: By default, the dss_source_system_key is added to every dimension table. It is considered part of any lookup on that table and has the key type set to 1. Set when the column is added during drag and drop table generation.
2	Indicates that this column is a dimensional join. Used on fact tables to indicate the dimension keys. Results in bitmap indexes being built for the columns. Set during the update procedure generation for a fact table, based on information from the staging table.
3	Slowly changing column indicator. Used on dimension tables to indicate that the column is being managed as a slowly changing column within the context of a slowly changing dimension. Set when a column is identified during the dimension update procedure generation.
4	Previous value column indicator. Used on dimension tables to indicate that the column is being managed as a previous value column. The source column identifies the parent column. Set during the dimension creation.
A	Indicates that the column is part of the primary business key. Set whenever a business key is defined as part of an update procedure generation.
B-Z	Indicates that the column is part of a secondary business key. Only used during index generation and not normally set.

Source Table

Identifies the source table where the column's data comes from. This source table is normally a fact table or a dimension table within the data warehouse. If the column was sourced from multiple tables, then the normal practice is to record one of the tables in this field and a comment listing all of the other tables in the **Source strategy** field. This field is used when generating a procedure to update the aggregate table. It is also used in the track back diagrams and in the documentation.

Source Column

Identifies the source column where the column's data comes from. Such a column is normally a fact table column or a dimension table column, which in turn may have been a transformation or the combination of multiple columns.

Transformation

Transformation. [Read-only].

Join

Indicates whether the table column is used in a table join. [Normally maintained automatically but can be optionally changed to override the default join logic used in the generated update procedure]. The **Source Table** and **Source Column** fields will provide the dimension table's side of the join. The options for this field are: False, True, Manual and Pre Join.

Setting this field to Manual changes the way the dimension table is looked up during the update procedure build. It allows you to join the dimension manually in the Cursor mapping dialog (where the 'Where' clause is built).

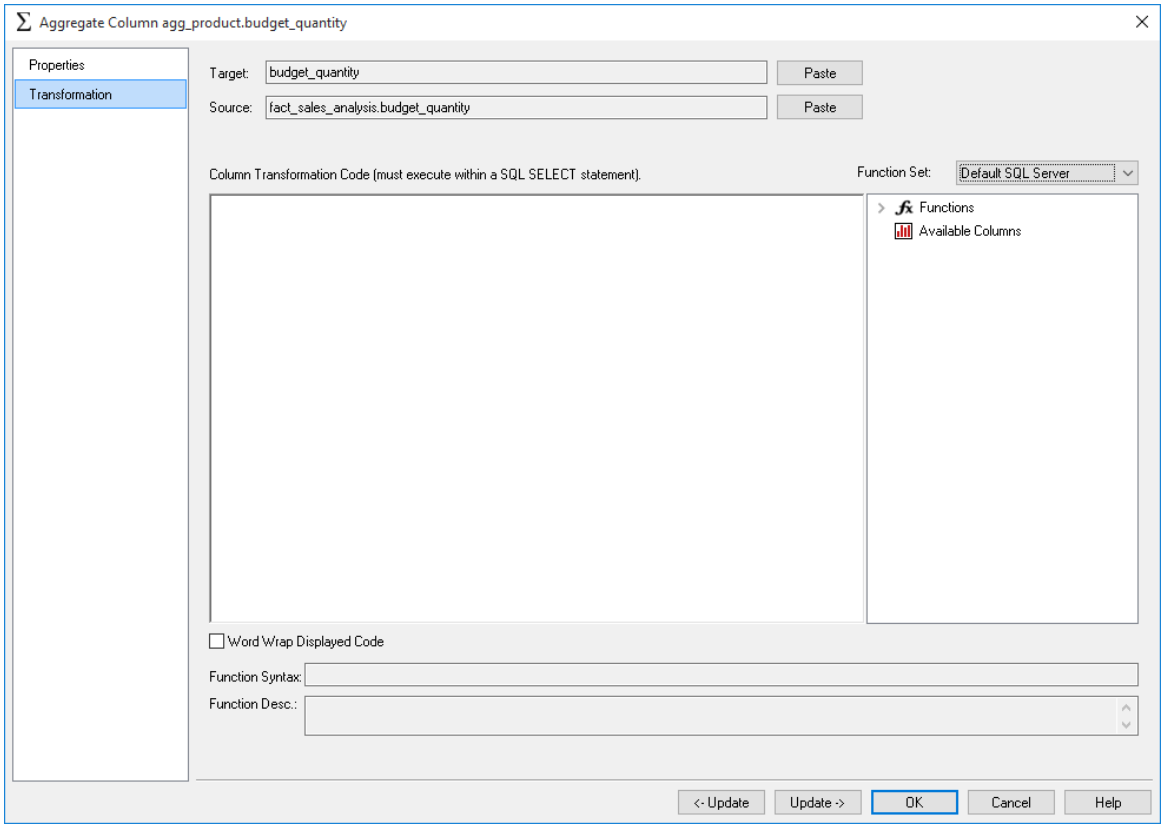
Setting this field to Pre Join activates the **Pre Join Source Table** field and allows you to select a table from the drop-down list.

Pre Join Source Table

Indicates the table from which the pre joined column was sourced. When the Join option is set to False, this field becomes inactive. When the Join option is set to True or Manual, this field is set to the current table name. When the Join option is set to Pre Join, then you can select the required table from the drop-down list.

AGGREGATE TABLE COLUMN TRANSFORMATIONS

Each aggregate table column can have a transformation associated with it. The transformation will be included in the generated procedure and will be executed as part of the procedure update. The transformation must therefore be a valid SQL construct that can be included in a **Select** statement. For example, we could have a transformation of 'load_order_line.qty * 0.125' to calculate a tax column of 12.5%. Click the **Transformation** tab to enter a transformation.



Note: Transformations are only put into effect when the procedure is re-generated.

See ***Transformations*** (on page 650) for more details.

WhereScape®

CHAPTER 17
VIEWS

Views are normally created from dimensions. These views are then used to define dimensions in a fact table where the dimension appears multiple times. A classic example is the date dimension. In our tutorials, we create a fact table that has two date dimensions - an order date and a ship date dimension. To facilitate the creation of the fact table we create two dimension views to allow the date dimension to be joined to the fact table twice.

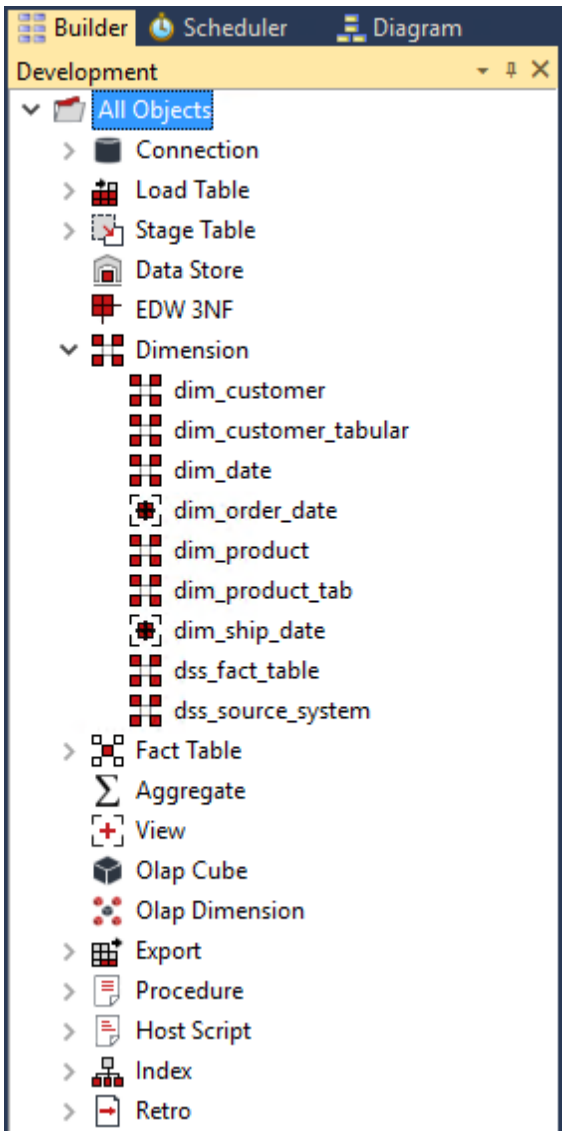
Views can also be created from any other table type. By default, views are displayed as part of the dimension object group. If fact table views or other views are being used then views can be displayed as a separate object type by selecting this option on the relevant **Tools/Options** menu item.

IN THIS CHAPTER

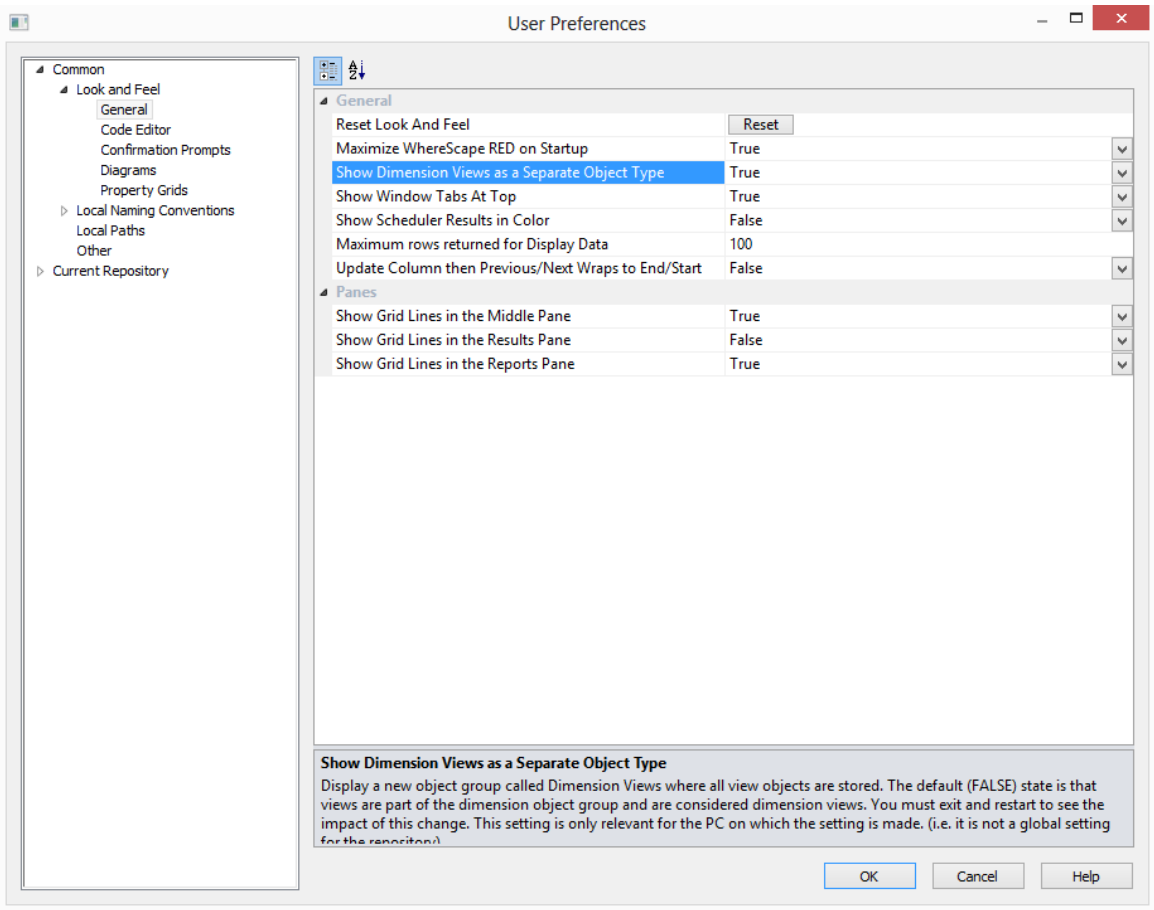
CREATING A DIMENSION VIEW

A dimension view is a database view of a dimension table. It may be a full or partial view. It is typically used in such cases as date dimensions where multiple date dimensions exist for one fact table.
In many data warehouses dimension views are built as part of the end user layer, but creating them in the data warehouse means they are available regardless of the end user tools used.

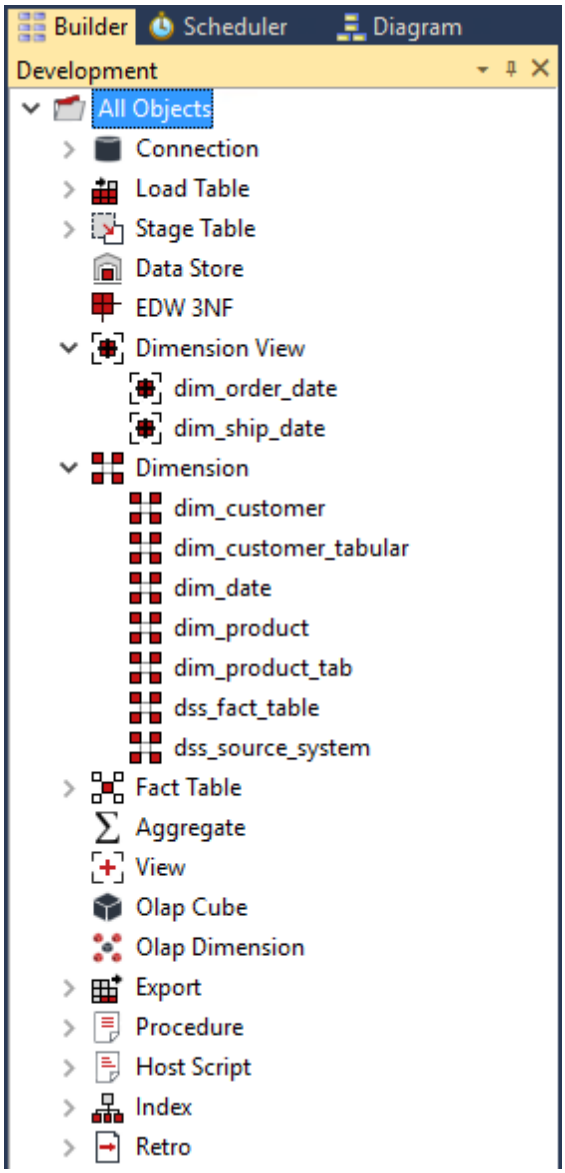
Views are displayed in WhereScape RED either as part of the **Dimension** object group or in a separate object group - **Dimension View**.
The default is to view dimensions and dimension views together:



This visualization option is set from the **Tools/User Preferences** menu item:

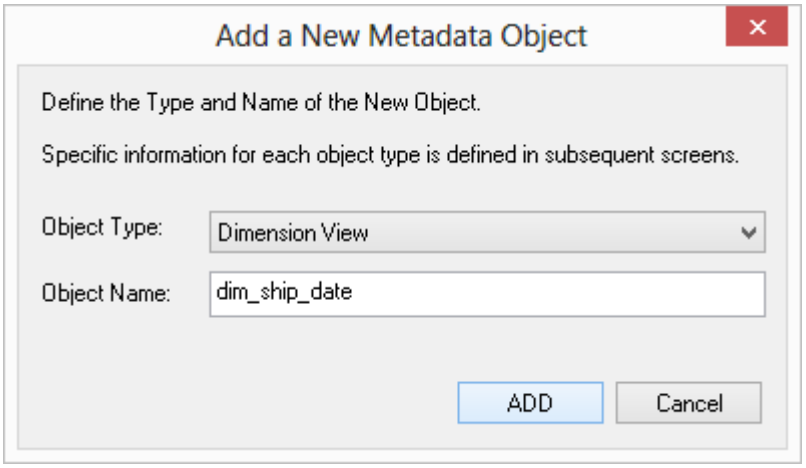


This example shows **Dimensions** and **Dimension Views** as separate object types:



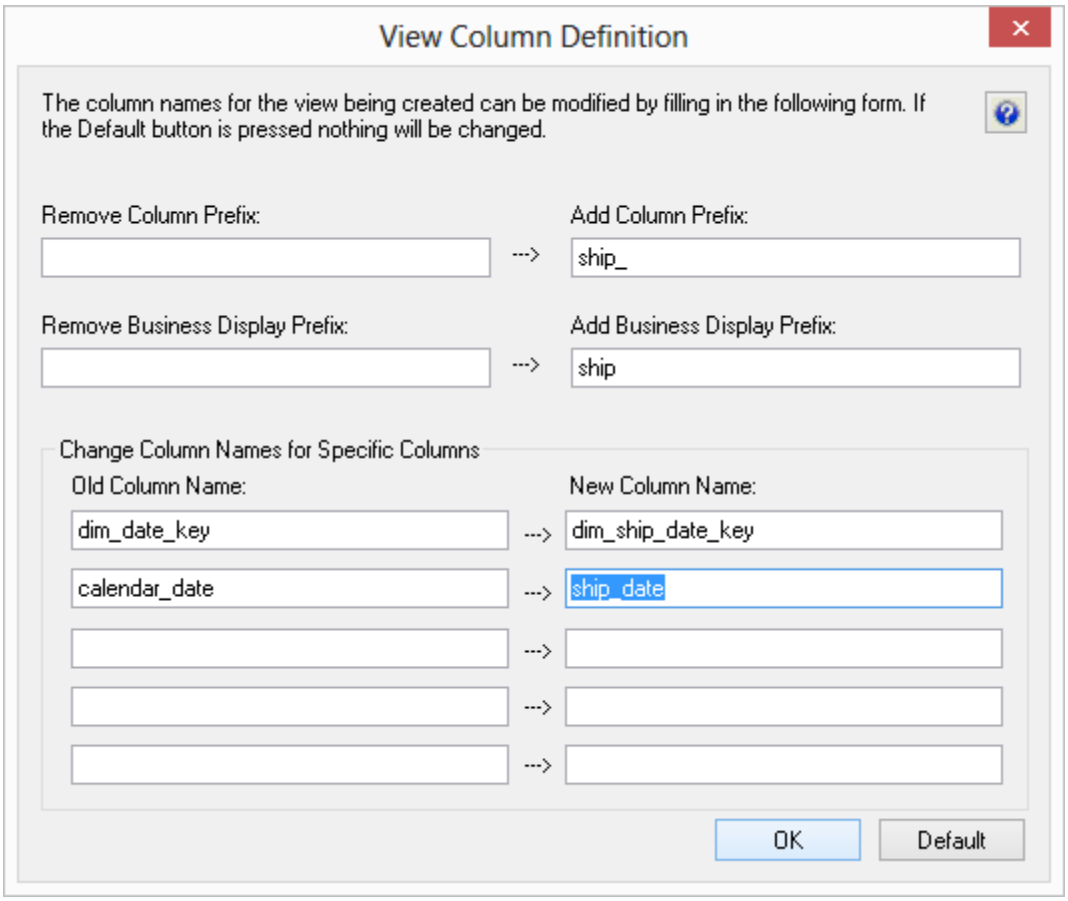
Creating a Dimension View:

1. Double-click on Dimension (or Dimension View) in the left pane.
2. Browse the data warehouse in the right pane.
3. Drag a table from the right pane into the middle pane.
   - The dialog box defaults the object type to a dimension view
   - Change the view name as required, and click **Add**.



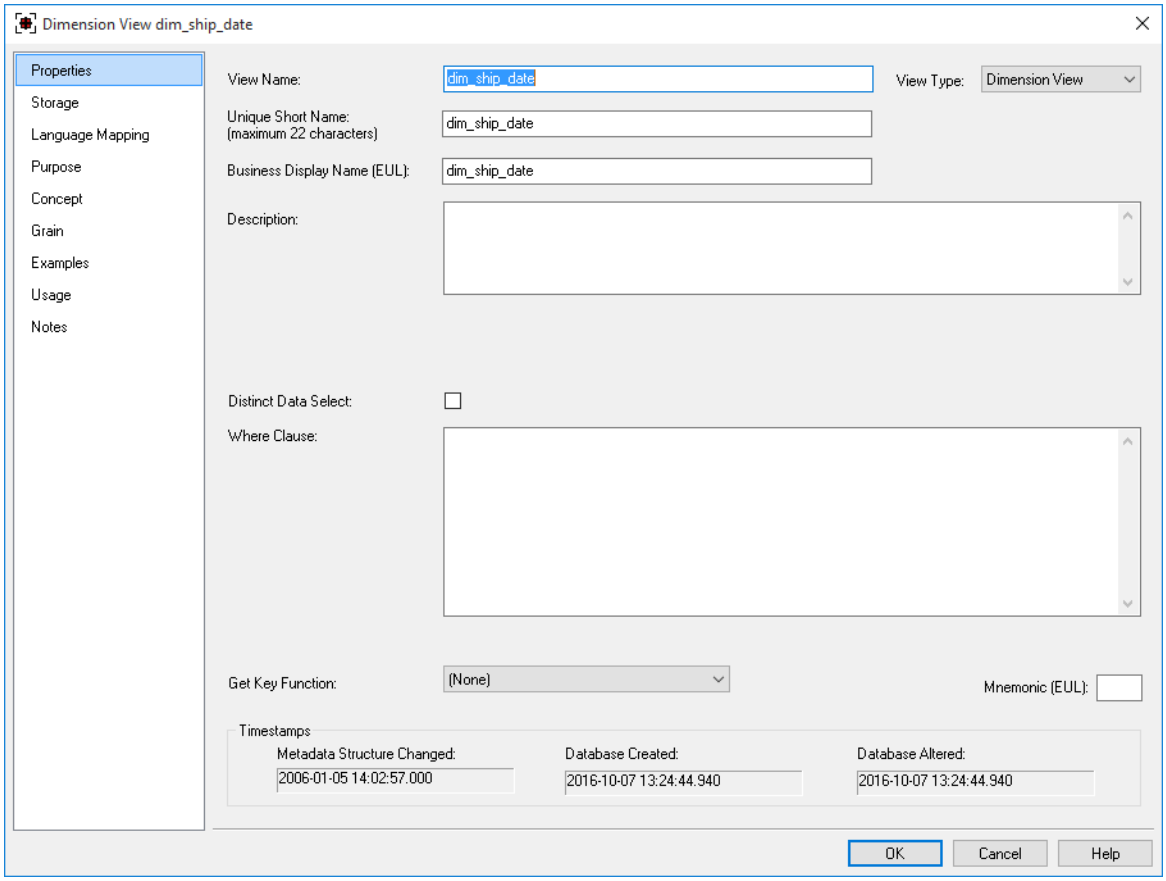
4 A dialog box displays to provide a means of re-mapping some of the column names in the view if required.

A description of the fields in this dialog are provided in the section that follows.



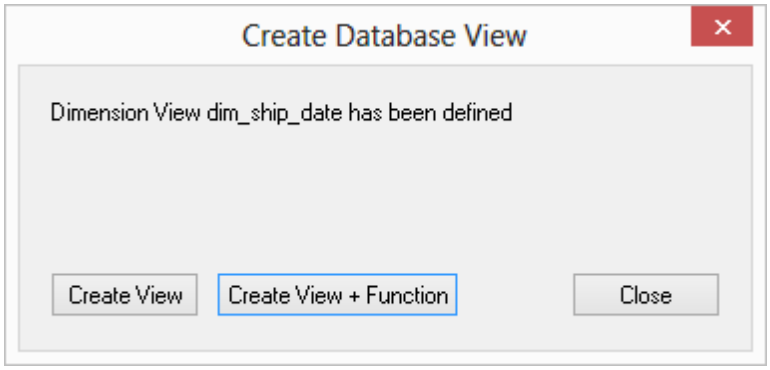
5 The view property defaults dialog will appear.

- Tick the **Distinct Data Select** check-box if you want the view to return only different values.
- A '**Where**' clause could be entered, but this can be done later. Normally you would just click **OK**.

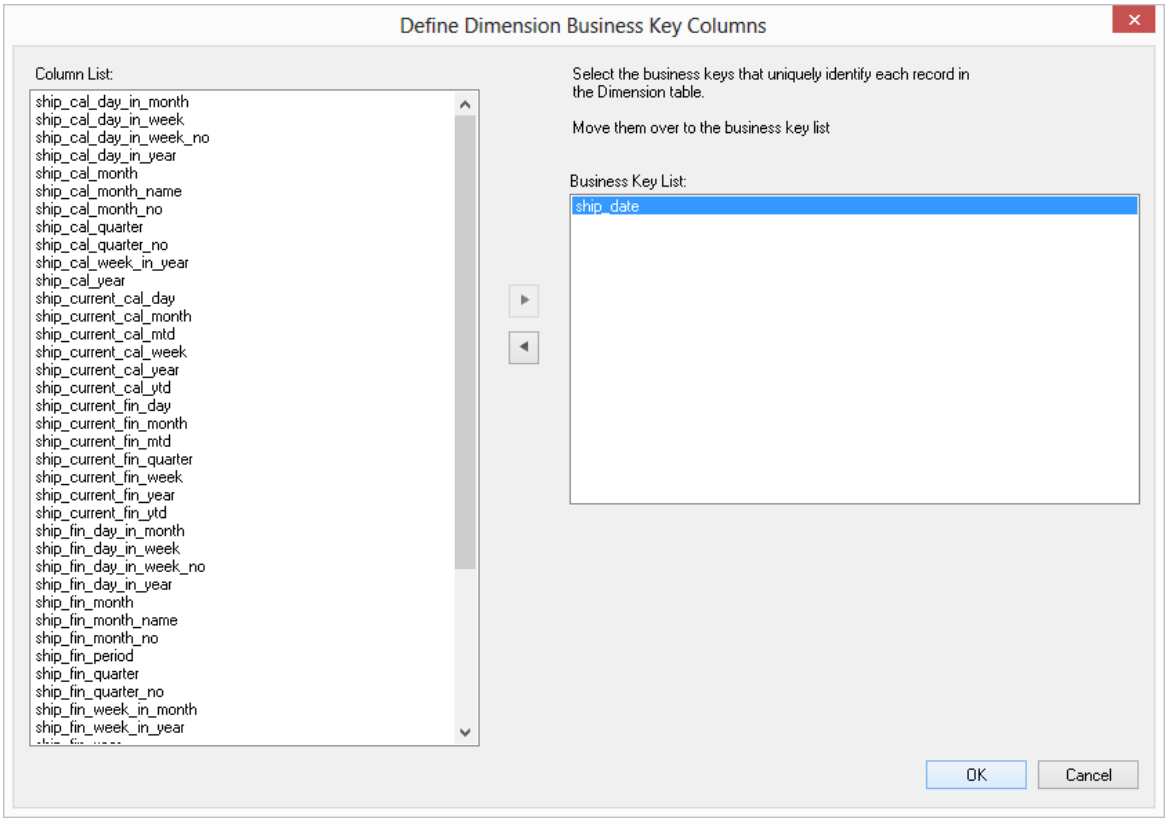


6 A dialog box indicates that the view has been defined.

- If this is a dimension view, then you will be given an option to create both the view and an associated Get Key function. For any other view type, a Get Key function is not created.



7 If you click **Create View + Function**, the following dialog displays allowing you to select a business (natural) key for the dimension view.

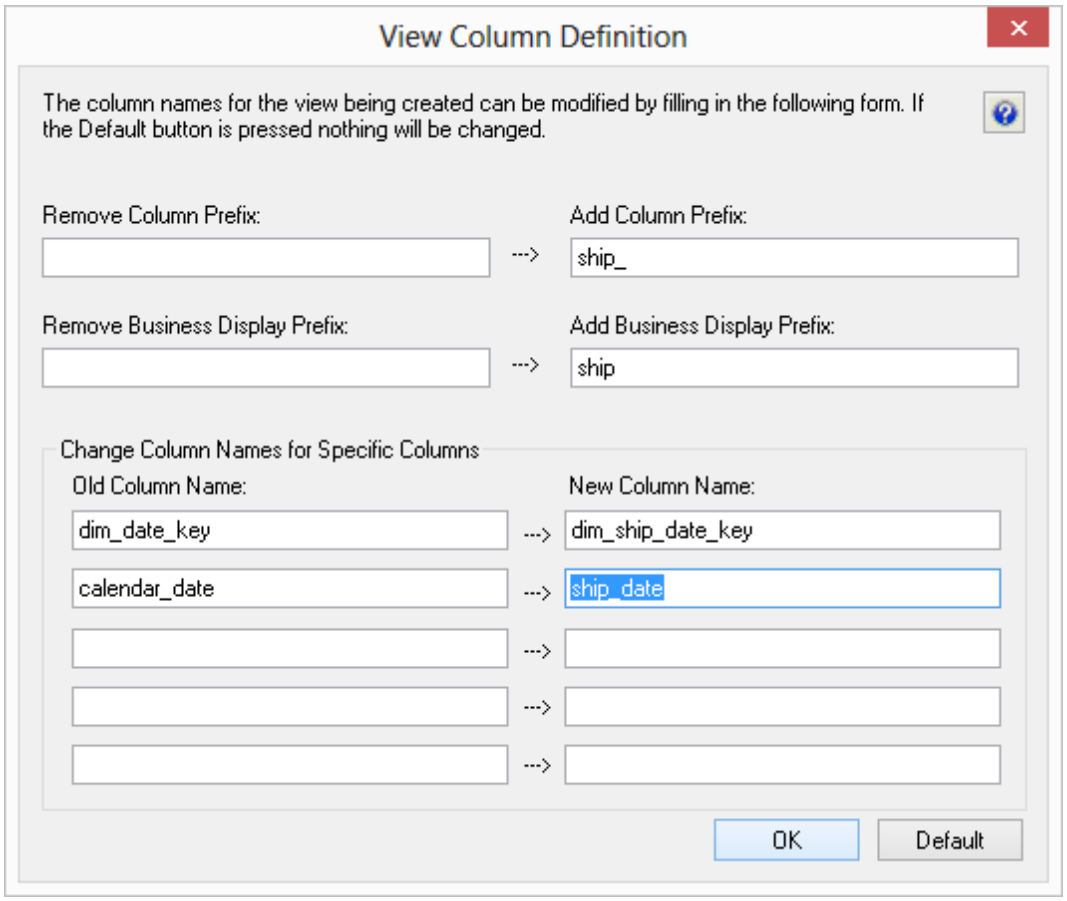


VIEW COLUMN RE-MAPPING

The view column definition dialog allows for the automated re-mapping of certain column names. It provides an easy method for changing the column names for a large number of columns when creating a view.

The various actions undertaken as a result of entries in this dialog can all be done or reversed manually by changing the individual column properties.

The various fields are described below:



Remove column prefix

If the columns in the source table are prefixed, then that prefix can be removed by entering it in this field. An example may be a date dimension which has each column prefixed with date_ (e.g. date_month, date_year etc.). If this field is left blank, then no removal action is taken.

Add column prefix

If required a prefix can be added to each column name. This is particularly useful when defining a date dimension view where you would like each column to be prefixed by the date type.

Remove business display prefix

As per the column names it may be required to remove a prefix from the business display fields. If so enter the prefix to remove in this column.

Add business display prefix

The business display fields are used in the creation of the glossary. It is therefore quite useful to prefix these display fields with an identifier for the view being created. It is assumed that these business display names will be carried forward to the end user layer. Enter a value in this field to prefix the business display name fields for each column. It is normal to include a space at the end of this field.

Old column name

Up to five individual column names can be re mapped. Enter the column name as it appears in the source table in one of the **Old Column Name** fields to re map that column name. The business display name is also changed to match.

New column name

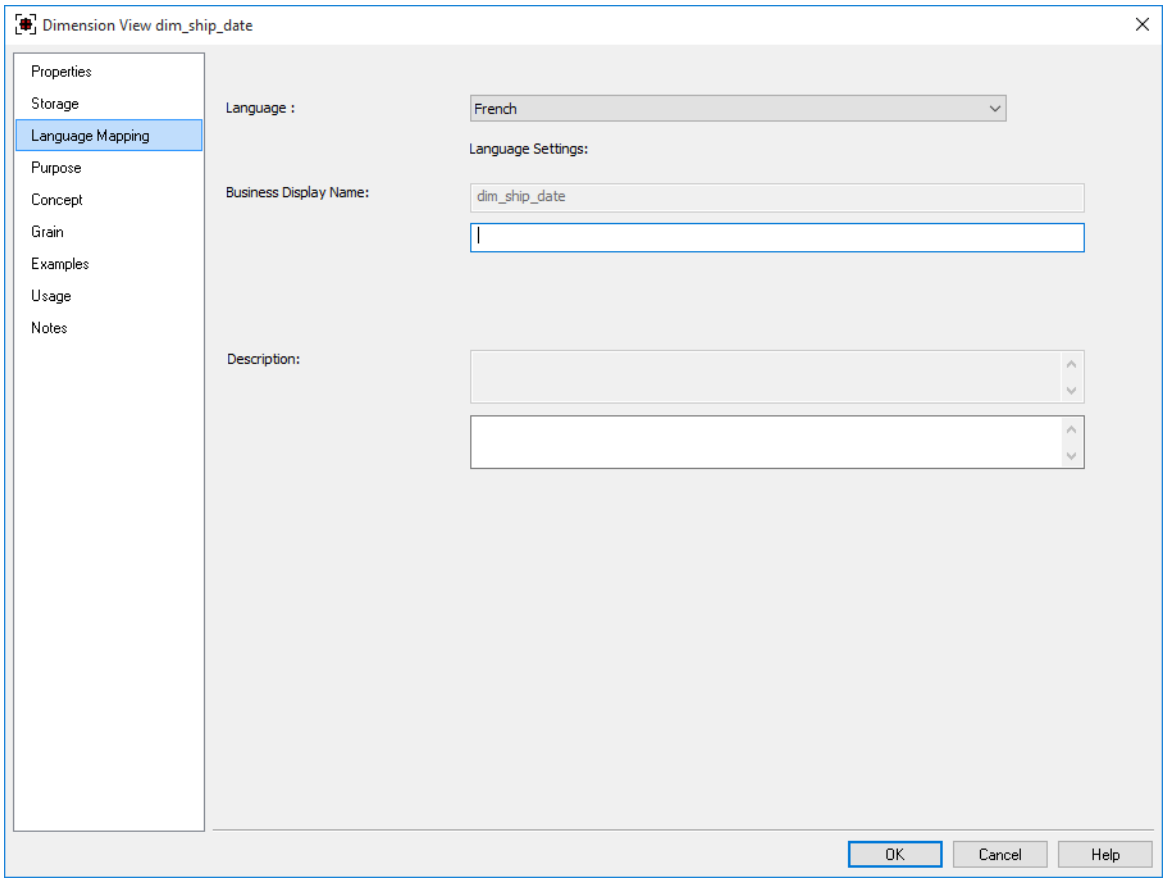
Place a new column name alongside any existing column name you wish to re map. In the example dialog above a column named **calendar_date** is being renamed to **order_date** in the view.

DIMENSION VIEW LANGUAGE MAPPING

The Dimension View Properties screen has a **Language Mapping** tab.

1 Select the language from the drop-down list and then enter the translations for the **Business Display Name** and the **Description** in the chosen language.

2 The translations for these fields can then be pushed through into OLAP cubes.



TIP: Languages can be added and maintained from the **Tools/Language Options** menu. See ***Language Settings*** (see "***Language Options***" on page 137) for more details on how to add a new language.

Business Display Name

Translated Value of Business Display Name.

Description

Translated Value of Description.

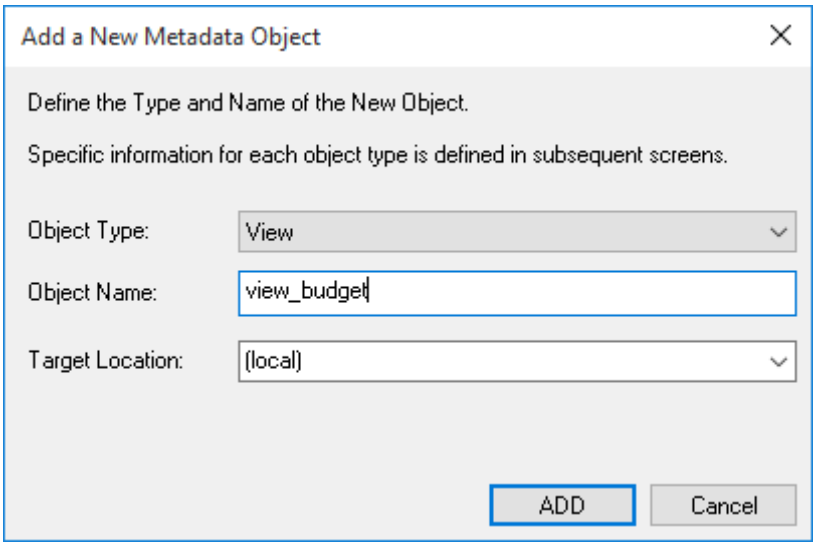
NON-DIMENSION OBJECT VIEWS

WhereScape RED also supports creating views of objects other that dimensions (load tables, facts, stage tables, aggregates, ...).
A fact view is a database view of a fact table. It may be a full, partial view, or a view incorporating both the fact data and some dimensional attributes.
It is typically used in cases where a subset of information is to be provided for specific end user communities. It is common that the 'Where' clause be populated in fact views, either to join the dimension tables or restrict the data that the view represents.
Similarly, views may be created over staging tables or any other **table** object.

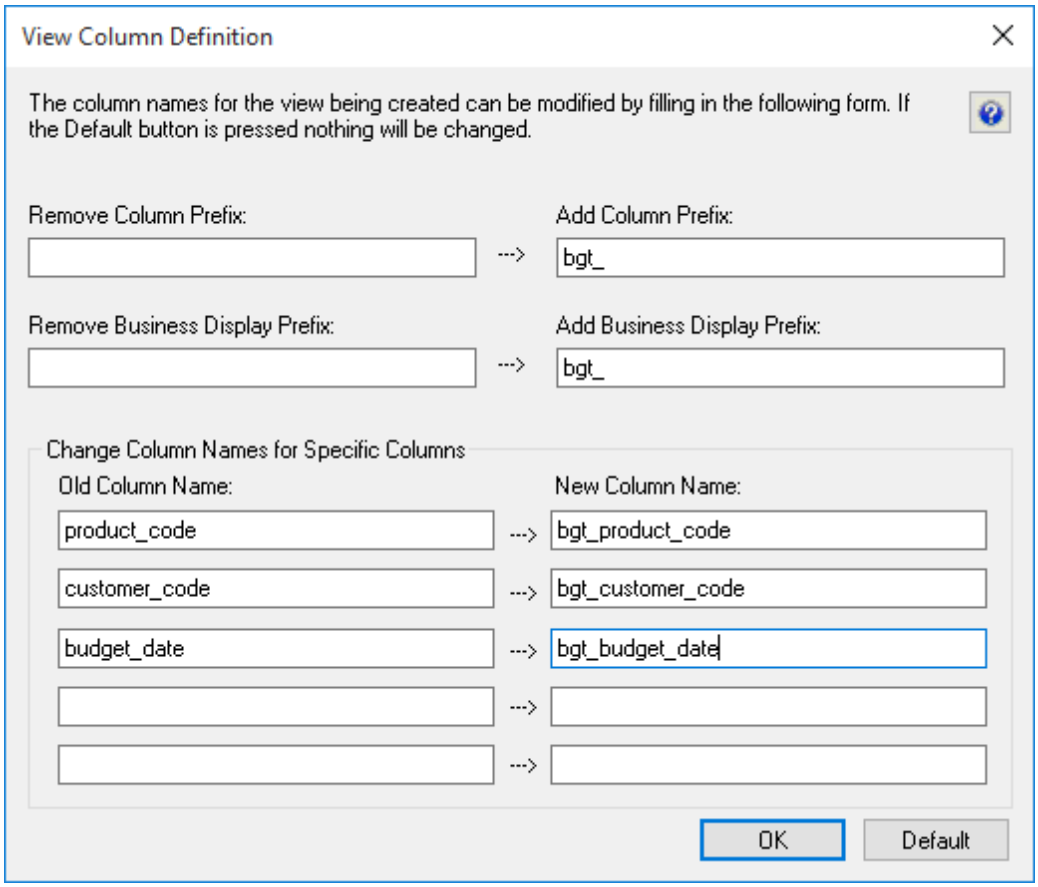
The creation of non-dimension table views is the same process as that for dimension views:

1 Drag a table from the right pane into the middle pane.

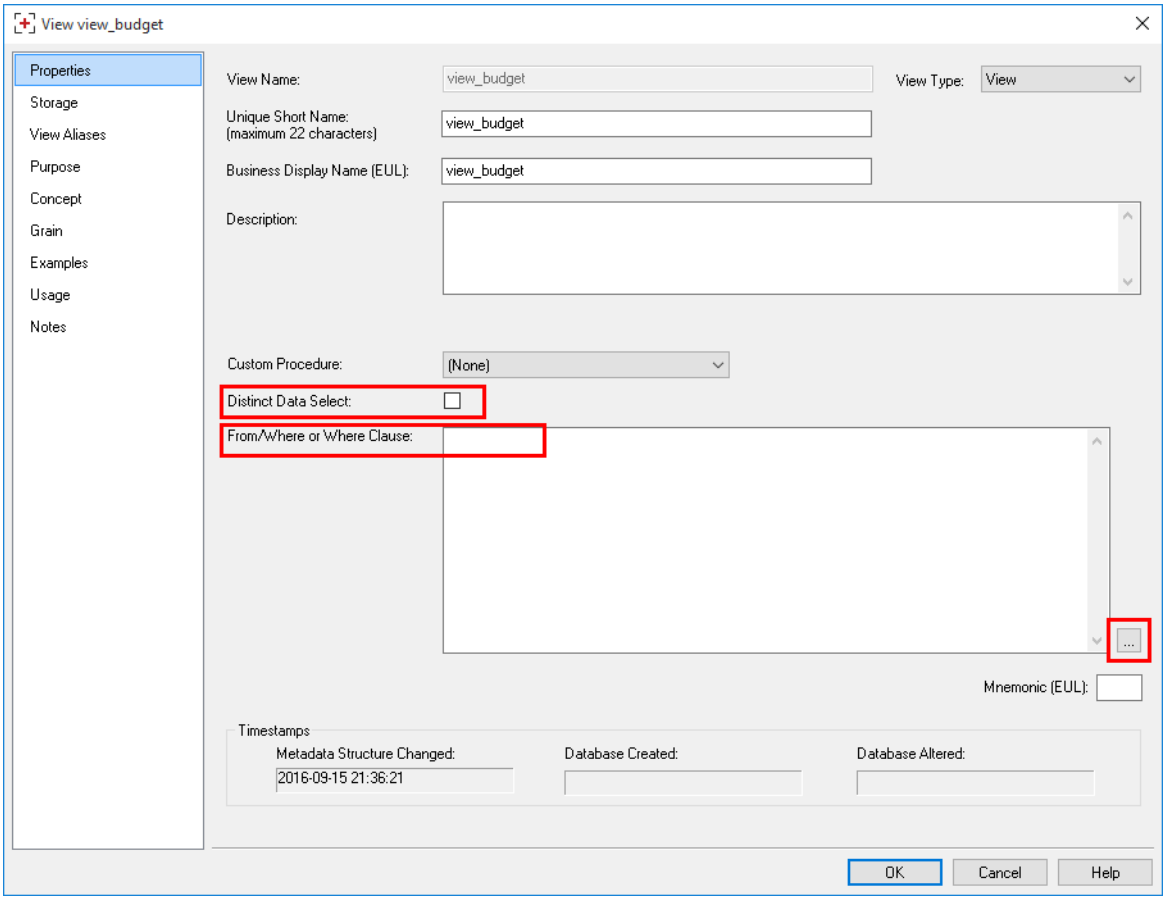
- The dialog box defaults the object type to a view.
- Change the view name as required, and click **Add**.



2 The view column definition dialog allows for the automated re-mapping of certain column names.
It provides an easy method for changing the column names for a large number of columns when creating a view.

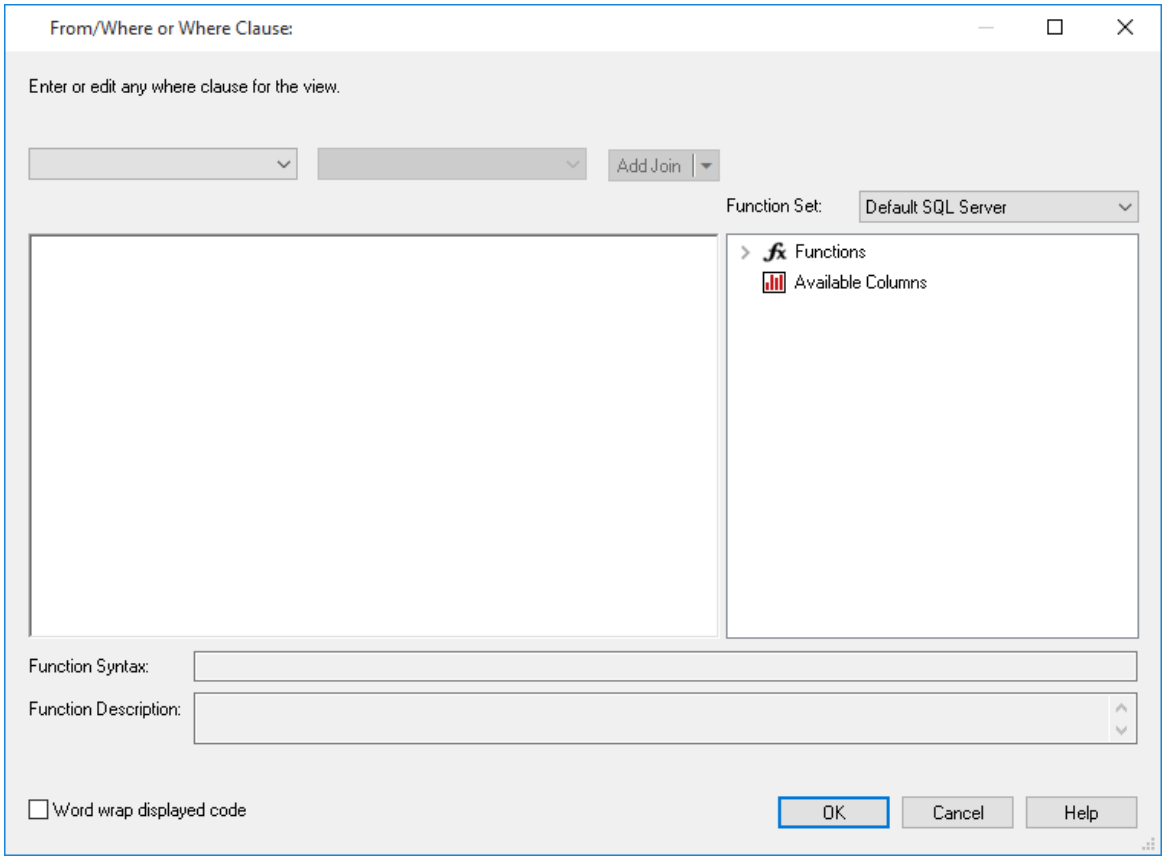


3 The View properties dialog displays:



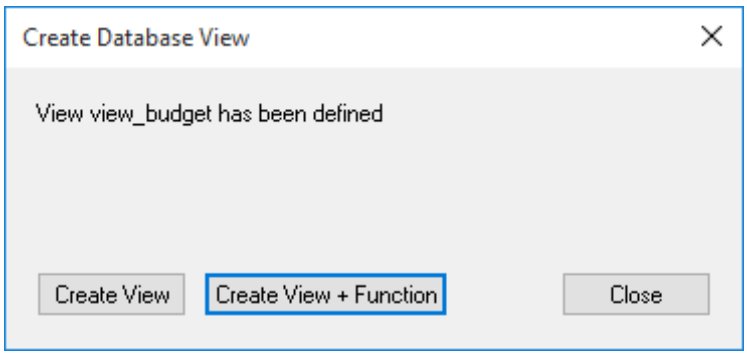
Change the following properties, if desired:

- Tick the **Distinct Data Select** check-box if you want the view to return only distinct values.
- A **From/Where** clause could be entered but this can also be done later. The ellipses button can be used to open the From/Where Clause editor dialog:

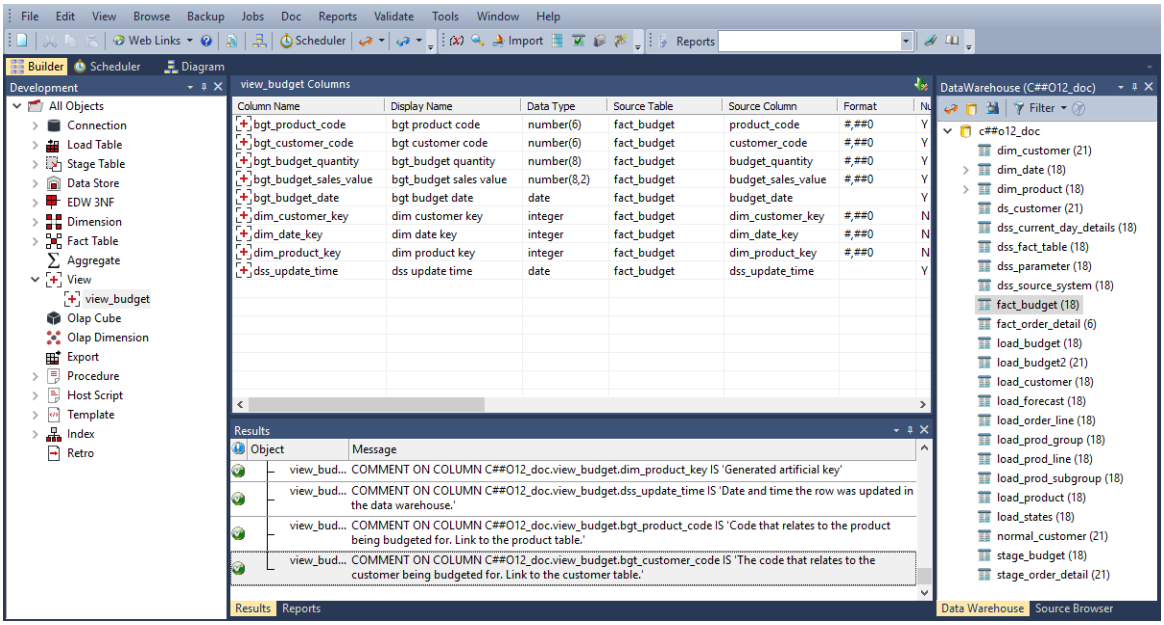


4 Click **OK**.

5 A dialog box indicates that the view has been defined, click **Create View**.



6 The columns of the view table will be displayed in the middle pane.



Note: If dimension attributes are being added (eg: to a fact view) then drag these attributes into the views column list, create the view and edit the 'Where' clause for the view.

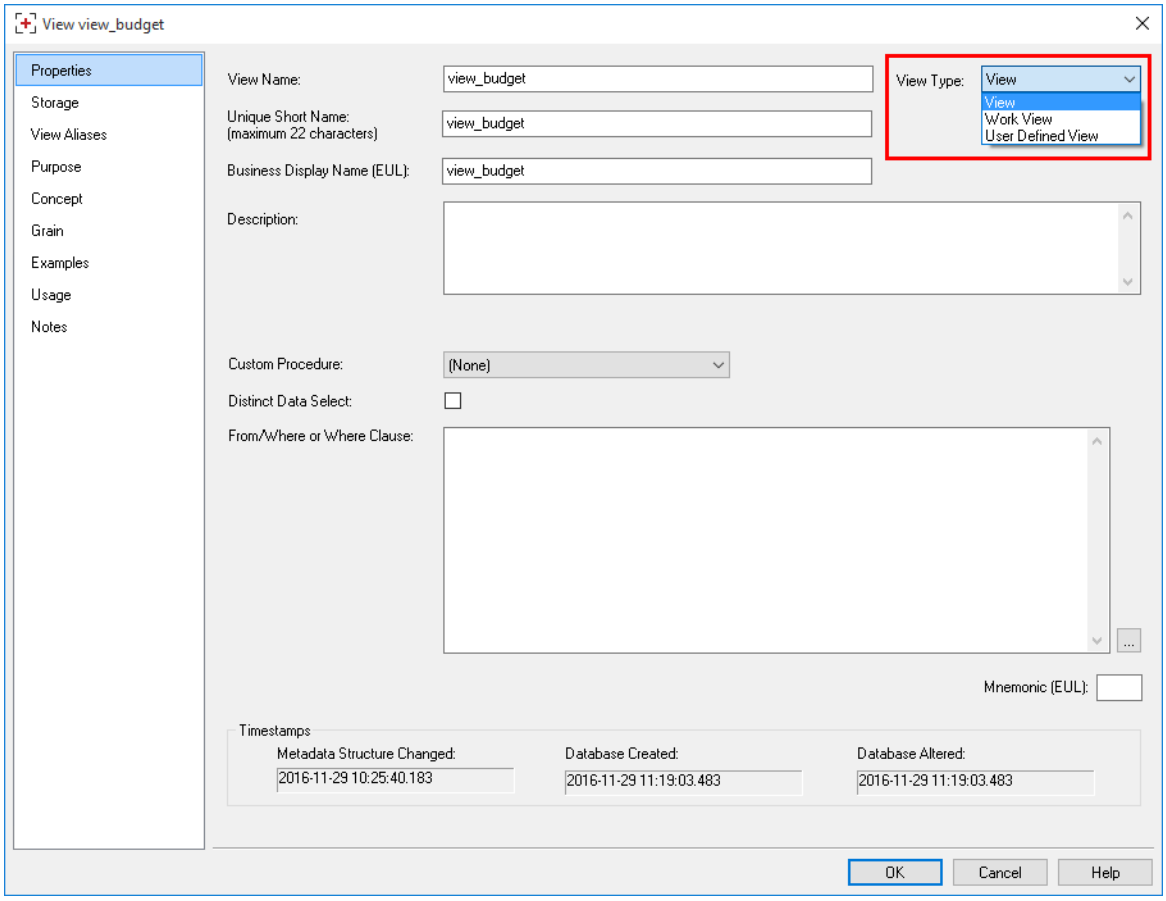
CREATING A CUSTOM VIEW

A custom view can be created within WhereScape RED to handle views that are not strictly one to one such as where multiple tables are joined or where a complex condition is placed on the view. There are two options for custom views, the first where the columns are defined in WhereScape RED and the 'Select' component of the view is customized. The second option is where the view is totally customized and no columns need to be defined in WhereScape RED, although it is good practice to still define the columns for documentation purposes.

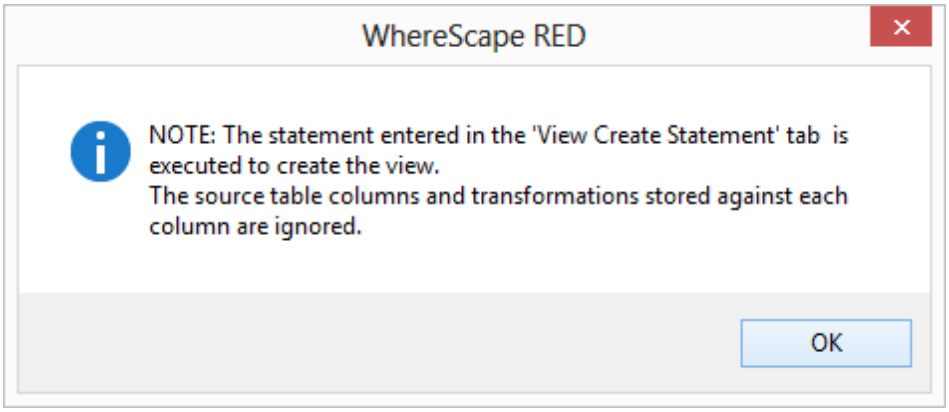
Creating a Custom or 'User Defined' view:

1 Create a view in the normal manner either by dragging a table in or adding a new object.

2 Change the table type to **User Defined View** in the View type drop-down box.

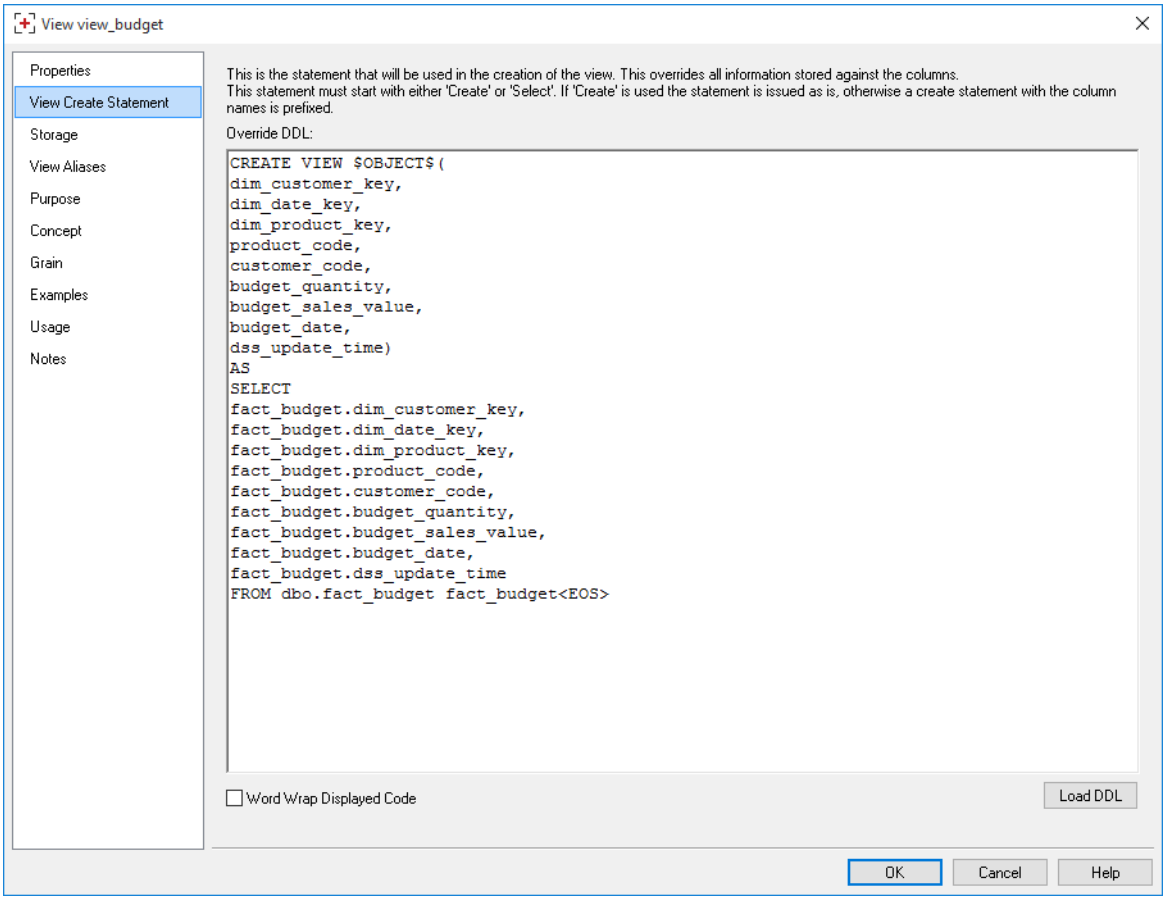


3 The following message is displayed. Click **OK**.



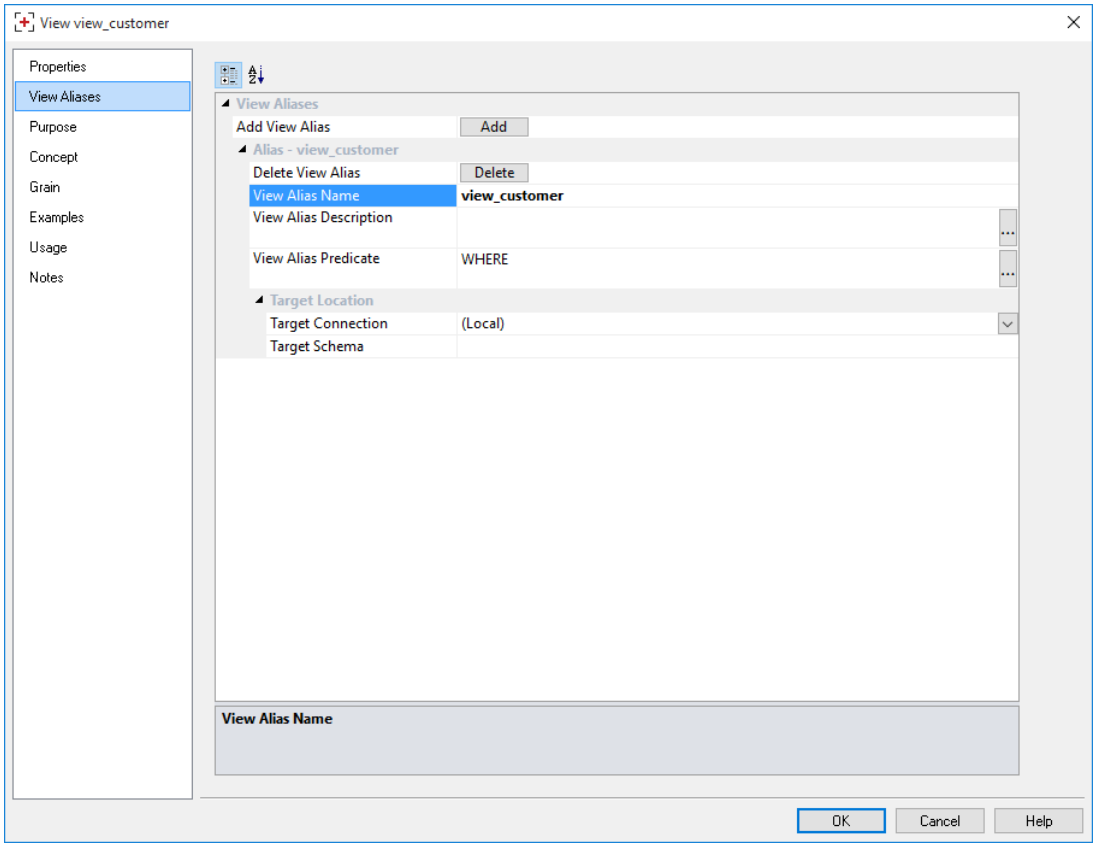
4 Edit the new tab **View Create Statement** and insert the SQL Statement that will be used to create the view.

- This SQL Statement must start with either **Create** or **Select**.
- If **Create** is used, then the columns in the view are ignored and the statement will be issued to create the view.
- If the statement starts with **Select,** then WhereScape RED will build up a view create statement from the column names and the supplied **Select** clause.
- There is also a **Load DDL** button on the right corner to get a sample **Select** statement based on the columns in the view and any transformations.



VIEW ALIASES

A View Alias provides the ability to create security views on SQL Server in an alternate schema. The **View Aliases** tab enables you to define additional/replica views.



Add button

Enables you to add a View Alias.

Delete button

Enable you to delete a View Alias.

View Alias Name

The view alias name.

View Alias Description

Description of the view alias.

View Alias Predicate

Optional 'Where' clause to include in the alternate view definition.

Target Schema

The target schema.

WhereScape®

CHAPTER 18
ANALYSIS SERVICES OLAP CUBES/TABULAR MODELS

IN THIS CHAPTER

OLAP CUBES

OLAP OVERVIEW

A **cube** is a set of related measures and dimensions that is used to analyze data.

- A **measure** is a transactional value or measurement that a user may want to aggregate. The source of measures are usually columns in one or more source tables. Measures are grouped into **measure groups**.
- A **dimension** is a group of attributes that represent an area of interest related to the measures in the cube and which are used to analyze the measures in the cube. For example, a customer dimension might include the attributes:
  - Customer Name
  - Customer Gender
  - Customer City

These would enable measures in the cube to be analyzed by Customer Name, Customer Gender, and Customer City. The source of attributes are usually columns in one or more source tables. The attributes within each dimension can be organized into hierarchies to provide paths for analysis.

A cube is then augmented with **calculations**, key performance indicators (generally known as **KPIs**), **actions**, **partitions**, **perspectives**, and **translations**.

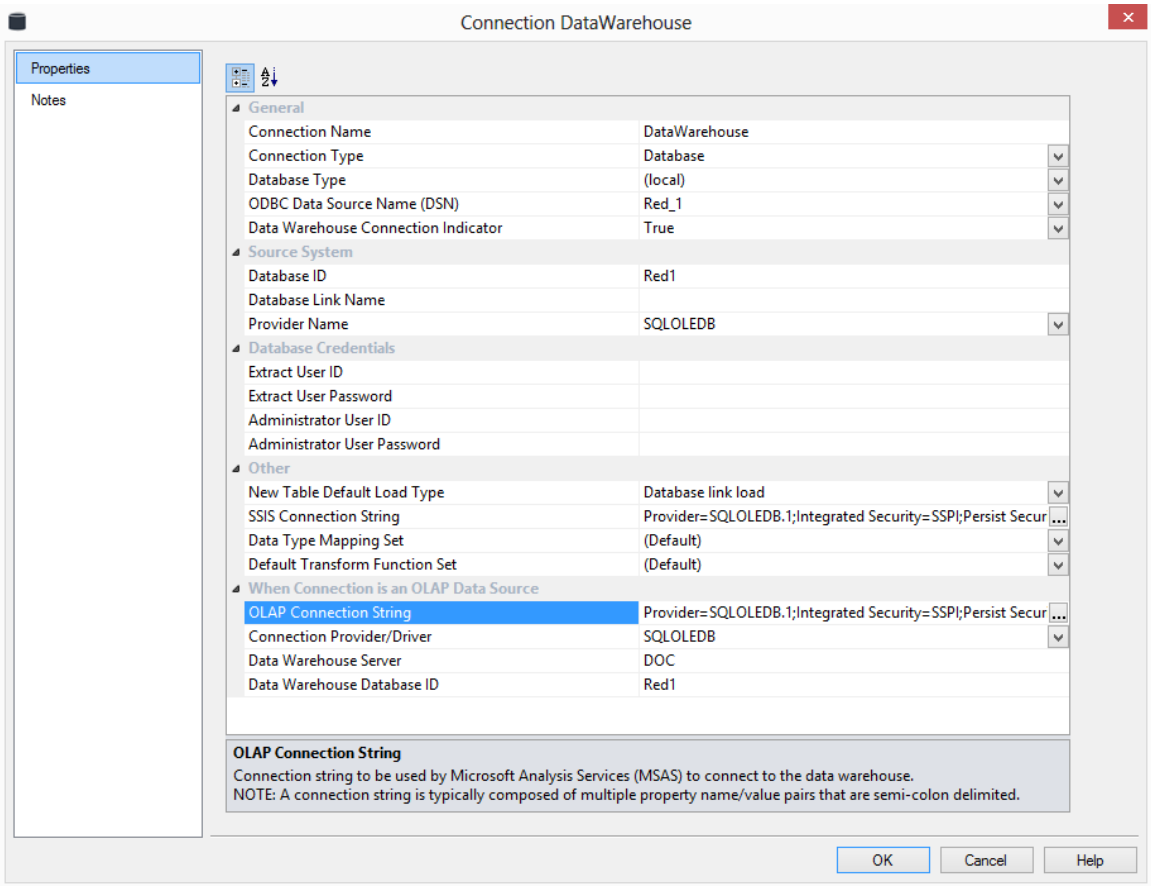
The information required to build and support an Analysis Services cube and its surrounding structure is reasonably complex and diverse. In attempting to automate the building of Analysis Services cubes WhereScape RED has simplified and restricted many of the functions available to the cube designer. WhereScape RED includes most of the commonly used capabilities and the components that logically fit into the methodology incorporated within WhereScape RED.

WhereScape RED broadly provides functionality to manage all of the above, except for perspectives and translations. These can be created outside of WhereScape RED, scripted in xmla and executed from within WhereScape RED. Features of cubes that are not supported in WhereScape RED can be added to the cube via the Microsoft tools. These altered cubes can still be processed through the WhereScape RED scheduler, and the cube should be documented within WhereScape RED to explain the post creation phases required.

As a general rule, once a cube or a component of a cube is created on the Analysis Services server it cannot be altered through WhereScape RED. The OLAP object can be dropped and recreated easily using RED. New OLAP objects defined in RED (e.g. additional calculations or measures) can be added by recreating the cube.

OLAP DEFINING THE DATA SOURCE FOR THE OLAP CUBE

Before we can create an OLAP cube, we first need to set up the data warehouse to be used as a source for Analysis Services cubes.
On the **Datawarehouse** Properties screen, the fields in the section **When Connection is an OLAP Data Source** are required.



OLAP Connection String

Connection string to be used by Microsoft Analysis Services (MSAS) to connect to the data warehouse.

Note: A connection string is typically composed of multiple property name/value pairs that are semi-colon delimited.

Connection Provider/Driver

Name of the Connection Provider/Driver to use to connect to the data warehouse database when it is used as the data source for OLAP cubes. Set to **SQLOLEDB**.

Data Warehouse Server

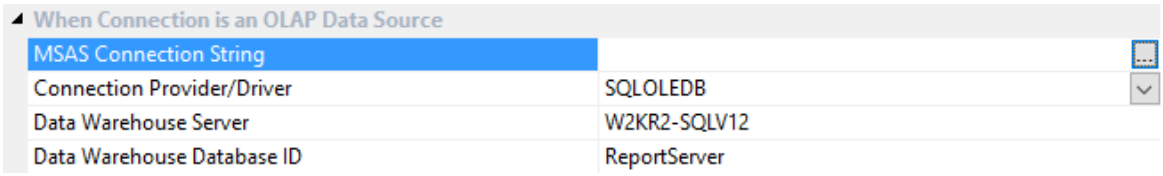
Data Warehouse Server Name, which is used when the data warehouse is used as the data source for OLAP cubes.

Data Warehouse Database ID

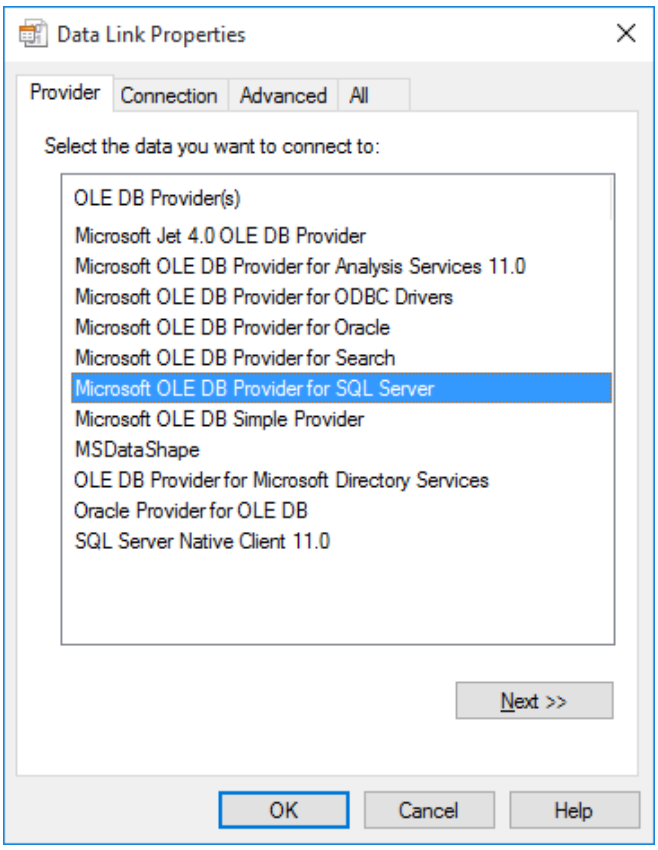
Data Warehouse Database Identifier (e.g. Oracle SID or TNS Name, Teradata TDPID) or Database Name (e.g. as in DB2 or SQL Server), which is used when the data warehouse is used as the data source for OLAP cubes.

Building the OLAP Connection String

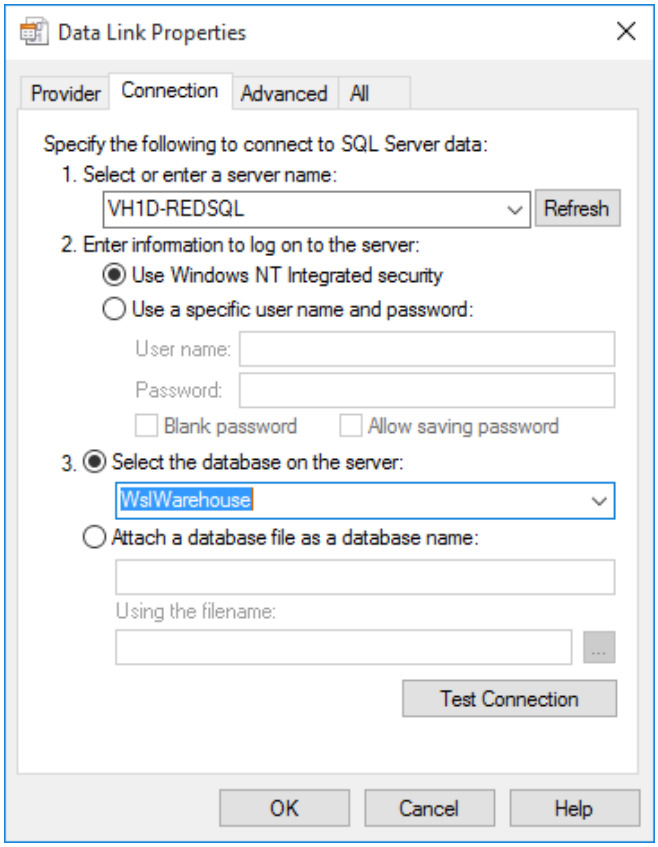
1 The **OLAP Connection String** is built using a wizard. To activate the wizard, click on the expander button.



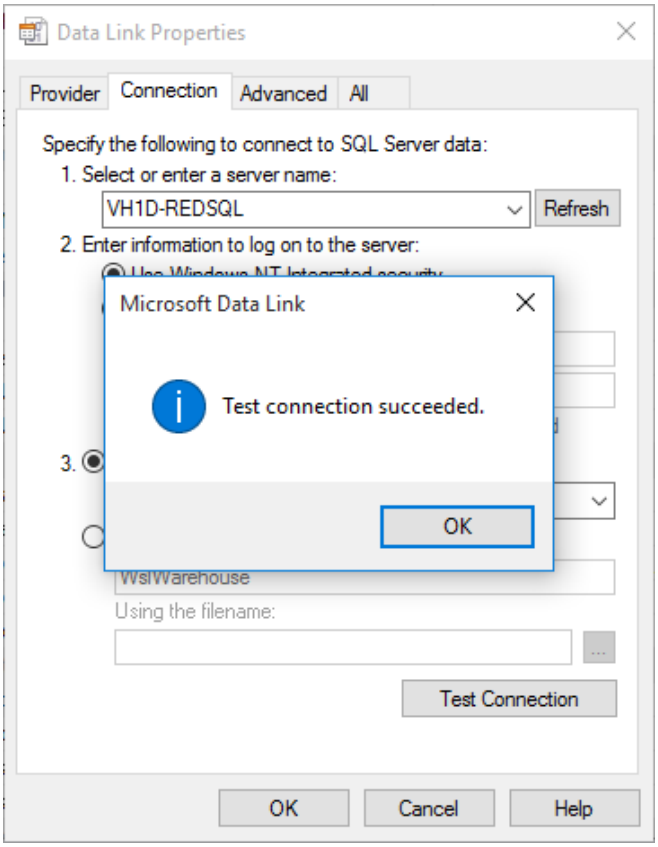
2 On the Provider tab, select the **OLE DB Provider** and click **Next**.



3 Enter the connection details and click Test Connection.



4 Click **OK** on the success message and then **OK** again on the Data Link Properties screen.



5 The OLAP connection string will be displayed.

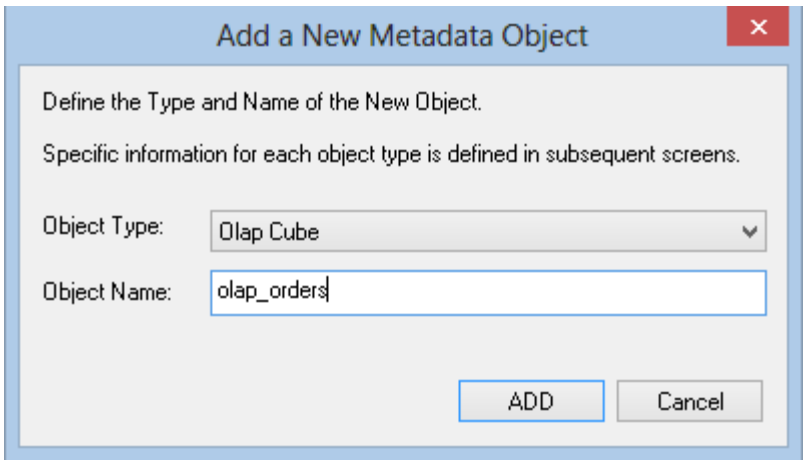
OLAP DEFINING AN OLAP CUBE

OLAP Cubes can be created from fact, fact view or aggregate objects. A single cube can contain dates from multiple source star schemas, each defined with a measure group. An OLAP Cube consists of many parts namely, measure groups, measures, calculations, actions, dimensions, dimension hierarchies, dimension attributes and dimension attribute relationships. It is strongly recommended that drag and drop is used to create an OLAP Cube in order that all the components are set up correctly. OLAP Cubes can utilize a hierarchical structure in the dimensions to facilitate user queries. Therefore, each dimension present in an OLAP Cube should have either a hierarchy of levels or attributes and relationships. The hierarchies are defined against the underlying dimensional attributes which can be inherited from the source dimension metadata. Individual attributes can be added to the dimension after the OLAP Cube or OLAP Dimension metadata has been created.

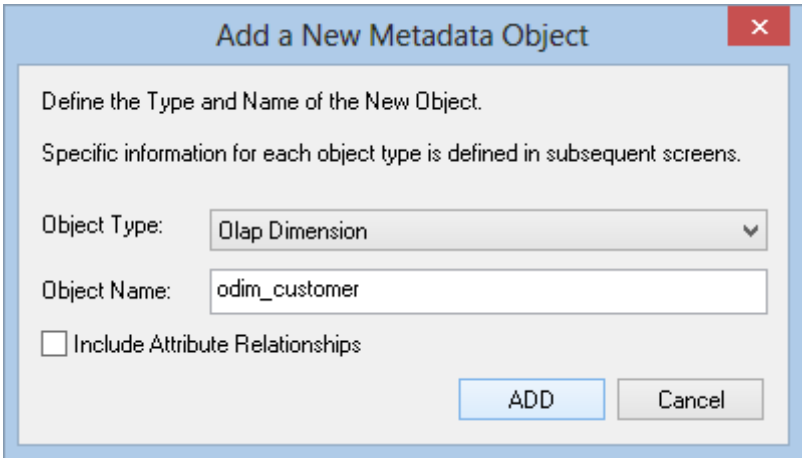
Building a New OLAP Cube

To create an OLAP Cube proceed as follows:

1. Double-click on the **OLAP Cube object group** to make the middle pane a cube drop target.
2. Select the **data warehouse connection** to browse in the source pane. The connection can be selected by right-clicking the Data Warehouse connection in the Object pane and choosing Browse Source System.
3. Drag a **fact table** from the source pane into the target pane.
4. Set the cube name in the **Create new metadata Object** dialog box and click **ADD**.

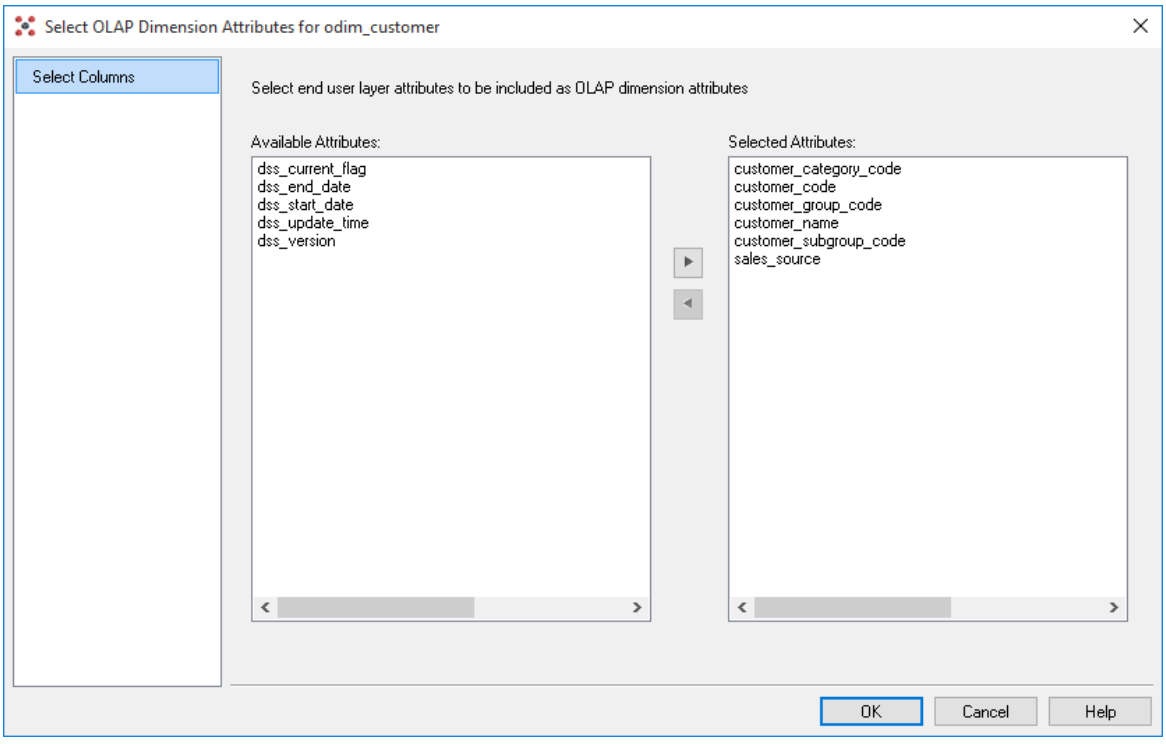


5 A dialog box will prompt for any OLAP Dimensions that do not already exist that are required for this cube (based on the fact table metadata). Set the dimension name in the **Add a new metadata Object** dialog box and click **ADD**.



Note: If you wish to include Attribute Relationships in Analysis Services for this dimension, click on the **Include Attribute relationships** checkbox.

6 The following dialog appears, prompting you to select the attributes to be included in the *Customer* OLAP dimension.

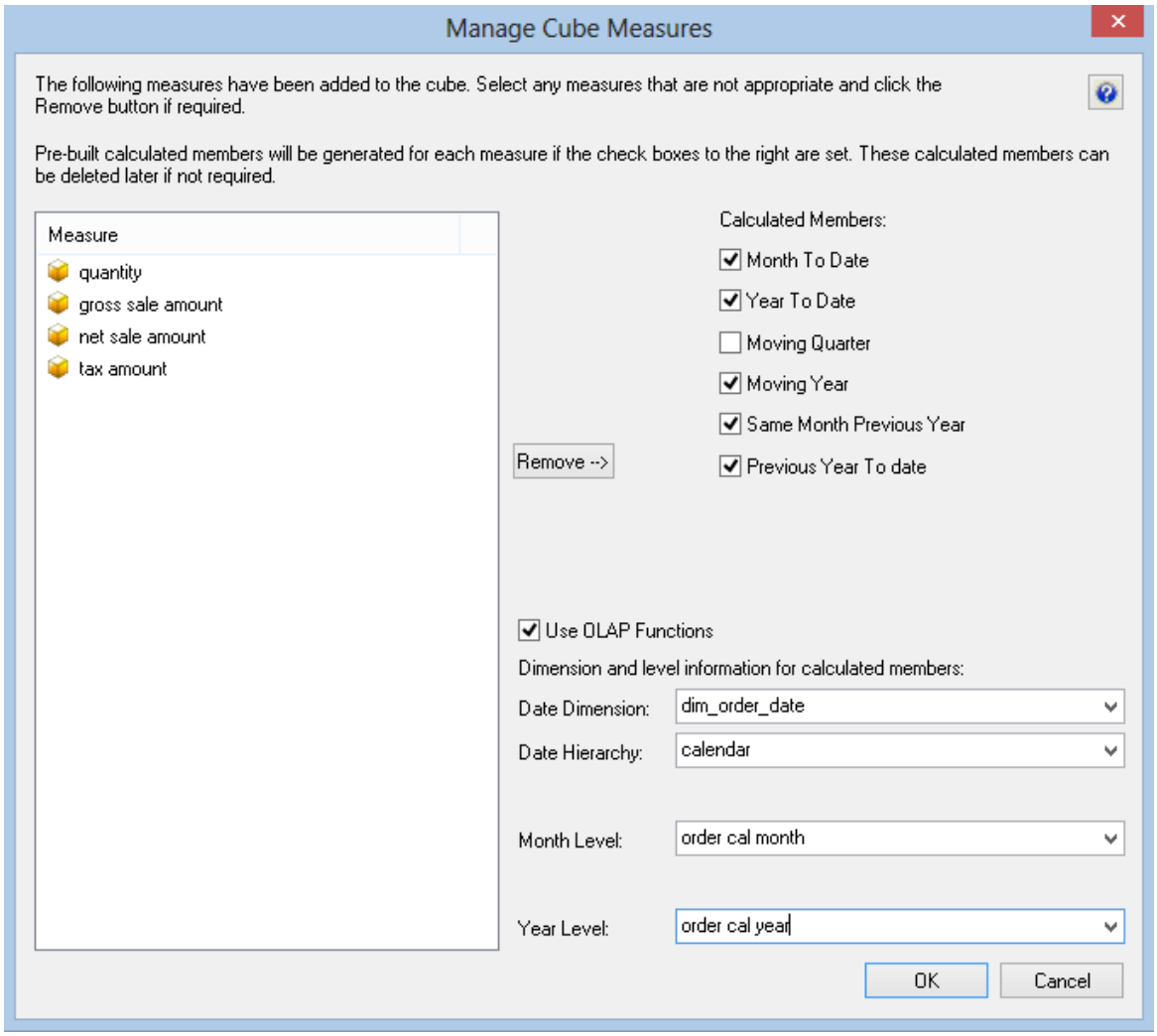


The attributes available for selection are in the left column. To select an attribute, click on the attribute in the left column and click **>**. This will move the attribute to the right column.

To de-select an attribute, click on the attribute in the right column and click **<**. This will move the attribute to the left column.

Repeat Step 5 for each dimension as required.

7 A dialog will appear with a list of the fields that are to be added as measures. Remove any columns that are not measures. A measure is a column that uses the sum, count, min or max of the column. Calculations can be chosen if required at this point. A date dimension must be present along with a hierarchy to allow the definition of these calculated members.



8 During cube creation, the **Adding Cube Measures** dialog is shown. In this dialog the following options are provided:

- Measure, provides a list of measures that can be aggregated (e.g. using Sum, Count, Min, Max or Distinct Count). By default, WhereScape RED will show all attributes in the fact table that are defined as numeric and additive. Those attributes that should not be considered measures can be removed using the **Remove** button.
- Calculated Member options will allow the user to add some predefined date based calculated member definitions to be built against the cube. The standard calculations:
  - Month to date
  - Year to date
  - Moving Quarter
  - Moving Year
  - Same Month Previous Year
  - Previous Year to date

  These will define a calculated measure based on the associated drop-down boxes. There are 2 different ways that WhereScape RED will implement these calculations which is dictated by the Use OLAP functions checkbox:

- Using OLAP Functions - will implement the calculations using MDX Expressions within the cube using date based MDX functions. These calculations are efficiently executed by Analysis Services.
- Without using OLAP functions - will implement the calculations using an MDX Filter function built over date dimension attributes. This option leverages the flags from the relational date dimension and ensures that a query using the calculations in the Cube will match an equivalent query against the star schema and is particularly useful if non-standard date periods are used.

9 The cube and dimensions will be created in WhereScape RED metadata and the cube measures will be displayed.

Setting Cube Properties

The properties of the cube must be completed before we can create the cube in the Analysis services database. Most of the elements in the properties screen will be defaulted, but each of the following columns will probably need to be completed.

1 The **Connection** to the Analysis services server must be defined within the cube properties. This connection is a connection object. If no such connection exists then a new connection of type must be created and configured. SQL Server 2005 or 2008 Analysis Services use a connection type of "Analysis Server 2005+". This connection name must then be chosen in the cubes properties.

2 A **Cube Database Name** must be selected. A new database name can be created by selecting **(Define New Cube Database...)** from the drop-down list. This database name is the database that the cubes will reside in on the Analysis services server.

3 The **Data Source Connection** must be defined and the three derived values shown under this connection must be present. If there is nothing in the three fields below the data source connection, then the connection object will need to be modified. The provider type will normally be MSDAORA if the data warehouse is an Oracle database or SQLOLEDB if the data warehouse is SQL Server. The Server is either the SQL Server database server, or the Oracle TNS name. The database is the data warehouse database name.

OLAP INSPECTING AND MODIFYING ADVANCED CUBE PROPERTIES

Now that the basic OLAP Cube has been defined, various properties of the OLAP Cube can be inspected or modified:

Measure Groups

1. Display the Measure Groups by right-clicking on the cube name and selecting **Display Measure Groups**.
2. Change the Measure Group properties by right-clicking on the measure group and selecting **Properties**.

Measures

1. Display all of the Measures associated with a cube by right-clicking on the cube name and selecting **Display Measures**.
2. Change the measure properties by right-clicking on the measure name and selecting **Properties**.

Calculations

1. Display all of the Calculated members defined on the cube by right-clicking on the cube name and selecting **Display Calculations**.
2. Change the Calculated members by right-clicking on a calculation and selecting **Properties**.

KPIs

1. Display all of the KPIs defined on the cube by right-clicking on the cube name and selecting **Display KPIs**.
2. Change the KPIs by right-clicking on the KPI name and selecting **Properties**.

Actions

1. Display all of the Actions defined on the cube by right-clicking on the cube name and selecting **Display Actions**.
2. Actions can be changed by right-clicking on the Action name and selecting **Properties**.

Partitions

1. Display all of the Partitions defined on the Measure Groups that are associated with the cube by right-clicking on the cube name and selecting **Display Partitions**.
2. Change Partitions by right-clicking on the Partition name and selecting **Properties**.

Dimensions

1. Display all of the OLAP Dimensions associated with the cube by right-clicking on the cube name and selecting **Display Dimensions**.
2. Change the customizable OLAP Dimension properties by right-clicking on the OLAP Dimension name and selecting **Properties**.

Measure Group Dimensions

1. Display the relationship of OLAP Dimensions to Measure Groups defined against the cube by right-clicking on the cube name and selecting **Display Measure Group Dimensions**.
2. Change the customizable properties of the relationship of the OLAP Dimension to Measure Group by right-clicking on the OLAP Dimension name and selecting **Properties**.

OLAP CREATING AN OLAP CUBE ON THE ANALYSIS SERVICES SERVER

If all of the tasks above are completed, then it should be possible to now create the cube on the Analysis Services server. When positioned on the **OLAP Cube** name, right-click and select **Create (Alter) Cube**.
WhereScape RED will check that key components of the cube are correct before it proceeds to issue the create command.

The create cube menu option will perform the following tasks on the Analysis Services server:

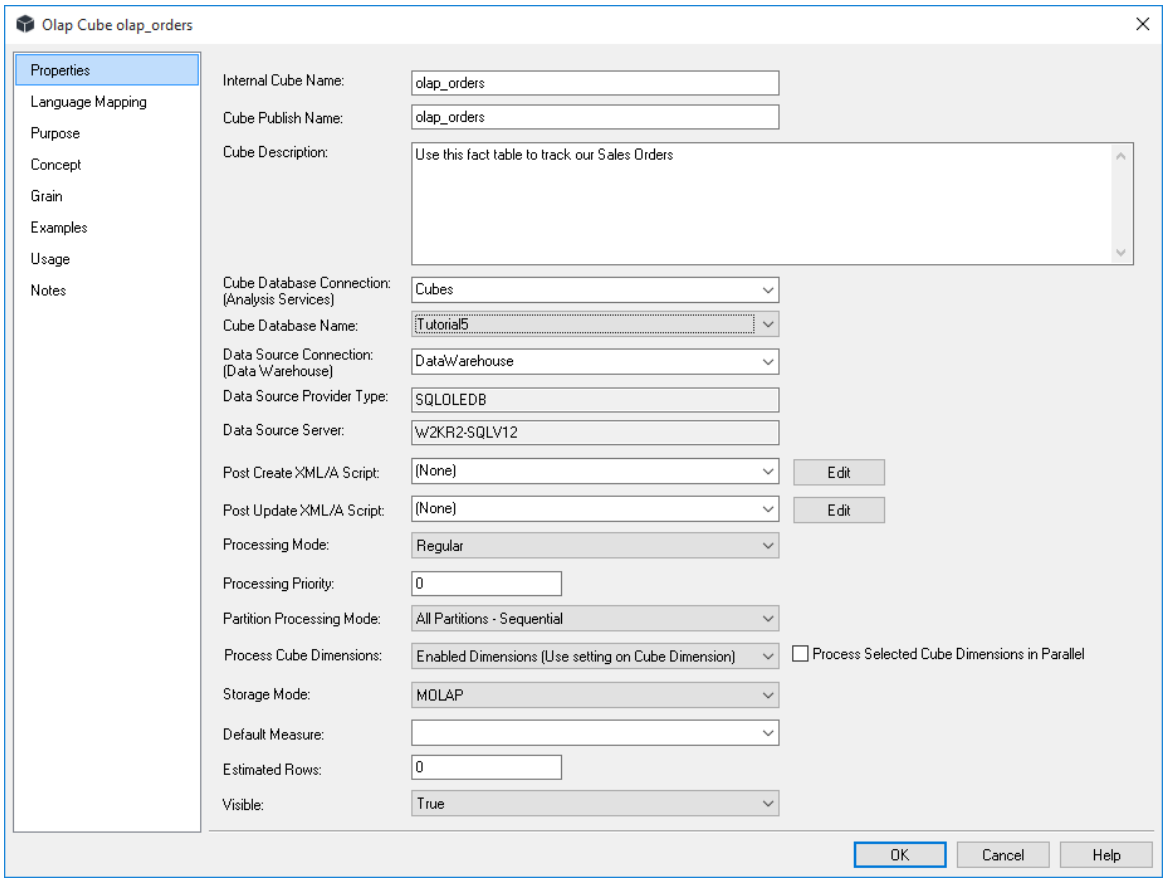
- Create an Analysis Services database if the name specified is not already present.
- Create a Data Source with connection information back to the data warehouse based on the cube source information in the Data Warehouse connection.
- Create a Data Source View to support the cube objects defined
- Create the dimensions used by the cube as database shared dimensions if they do not already exist.
- Create the cube if it does not exist
- Create a partition for the cube.

OLAP CUBE OBJECTS

OLAP CUBE PROPERTIES

The properties associated with a cube are described below. These properties relate both to the cube environment and to the cube itself.

There are seven tabs in the cube Properties screen.
The first is the main properties, the second the processing and partitioning options and the rest are for documentation stored in the WhereScape RED metadata and displayed in the generated WhereScape RED documentation.
In order to see the cube properties, right-click on the cube name and select **Properties**.



Internal Cube Name

This is the name by which the cube is known within WhereScape RED. This name is limited to 64 characters in length and may contain only alphanumeric characters and underscores.

Cube Publish Name

This is the name that the cube will have in the Analysis Services server. It is not constrained in its content except for the limitations imposed on it by Analysis Services.

Cube Description

A description of the cube. This is used both in the WhereScape RED documentation and is also stored against the cube in Analysis Services.

Cube Database Connection

This field allows the selection of one of the existing connections defined within WhereScape RED. The connection must be of type 'Microsoft Analysis Server 2005+'. If no such connection exists then a new connection object must be created. This connection object is used to point to the Analysis Services server.

Cube Database Name

An Analysis Services server must have databases associated with it. Each database can contain one or more cubes, dimensions, data sources etc. Select the name of an existing database on the server from the drop-down list. To create a new Database name, select **(Define New Cube Database)** from the drop-down list and the dialog that follows will allow you to register the name within the WhereScape RED metadata. Once registered, the name can then be selected as the database.

Data Source Connection

In Analysis Services the data source is the location of the data to be used to build the cube. It also defines the path for any drill through operations. This field provides a drop-down of the existing connections. The Data Warehouse connection must be chosen.

Data Source Provider Type

This field essentially defines what type of database the Data Warehouse is. The three options are MSDAORA where the data source is an Oracle database. OLESQLDB where the data source is a SQL Server database, or MSDASQL to define an ODBC source. The MSDASQL provider type could be used for either Oracle or SQL Server data warehouses, but performance may be compromised. This field is a read only field in the Properties screen. Its value is set in the properties of the data source connection.

Data Source Server

The data source server is also a read only field being sourced from the properties of the data source connection. For SQL Server, it defines the server on which the data warehouse database runs. For Oracle, it is the TNS names entry for the database.

Data Source Database

The data source database is also a read only field being sourced from the properties of the data source connection. For SQL Server, it defines the database in which the data warehouse runs. For Oracle, it is also the database name.

Post Create XML/A Script

This is an XML/A script that is run on the cube database when the cube is created. This script allows Analysis Services features to be added to the cube or cube database that have been built outside of WhereScape RED - for example security roles that has been defined for the cube can be recreated from the script when the cube is created (or recreated).

Post Update XML/A Script

This is an XML/A script that is run on the cube database when the cube is updated or processed via the scheduler. This script allows Analysis Services features to be added to the cube or cube database that have been built outside of WhereScape RED - for example security roles that has been defined for the cube can be recreated from the script when the cube is updated or processed.

Processing Mode

Gets or sets the index and aggregation settings for cube processing. The value indicates whether processed data is available after all required data aggregation has been completed (Regular) or immediately after the data has been loaded (Lazy Aggregations). This setting will be used as the default for new measure groups and partitions created for the cube.

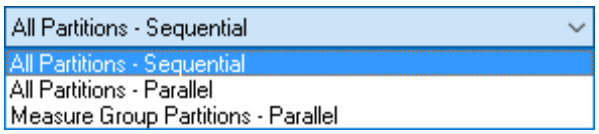
Processing Priority

Gets or sets the processing priority for the cube.

Partition Processing Mode

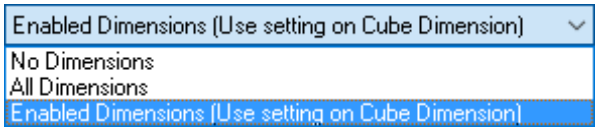
This option determines how partitions are updated when a cube is updated.

All Partitions - Sequential will update each cube partition sequentially.
All Partitions - Parallel will have all the cube partitions updated in parallel.
Measure Group Partitions - Parallel will process each measure group sequentially with all partitions of that same measure group being updated in parallel.



Process Cube Dimensions

Will determine whether to process the OLAP Dimensions related to the cube as part of the processing of the cube. The options are to process the **enabled dimensions** only, to process **no dimensions**, or to process **all the dimensions**.
Processing of specific Dimensions with the Cube can be enabled or disabled on the **Process Cube Dimension With Cube** check-box of each Dimension's Properties screen. See ***OLAP Cube Dimensions***.



Process Selected Cube Dimensions in Parallel

Ticking this check-box will allow for all dimensions within the cube to be updated in parallel instead of being updated sequentially.

Storage Mode

This field allows two options; MOLAP - Multidimensional OLAP or ROLAP - Relational OLAP. At the cube properties level, setting this field will determine the defaults for the Storage Mode field on its related Measure Groups and partitions.

Default Measure

Specifies the measure used to resolve MDX expressions if a measure is not explicitly referenced in the expression. If no default measure is specified an arbitrary measure is used as the default measure.

Estimated Rows

Specifies the estimated number of rows in the fact tables. Enter the size of the fact or source table if known, otherwise leave as zero.

Visible

Indicates whether the cube is visible to client applications.

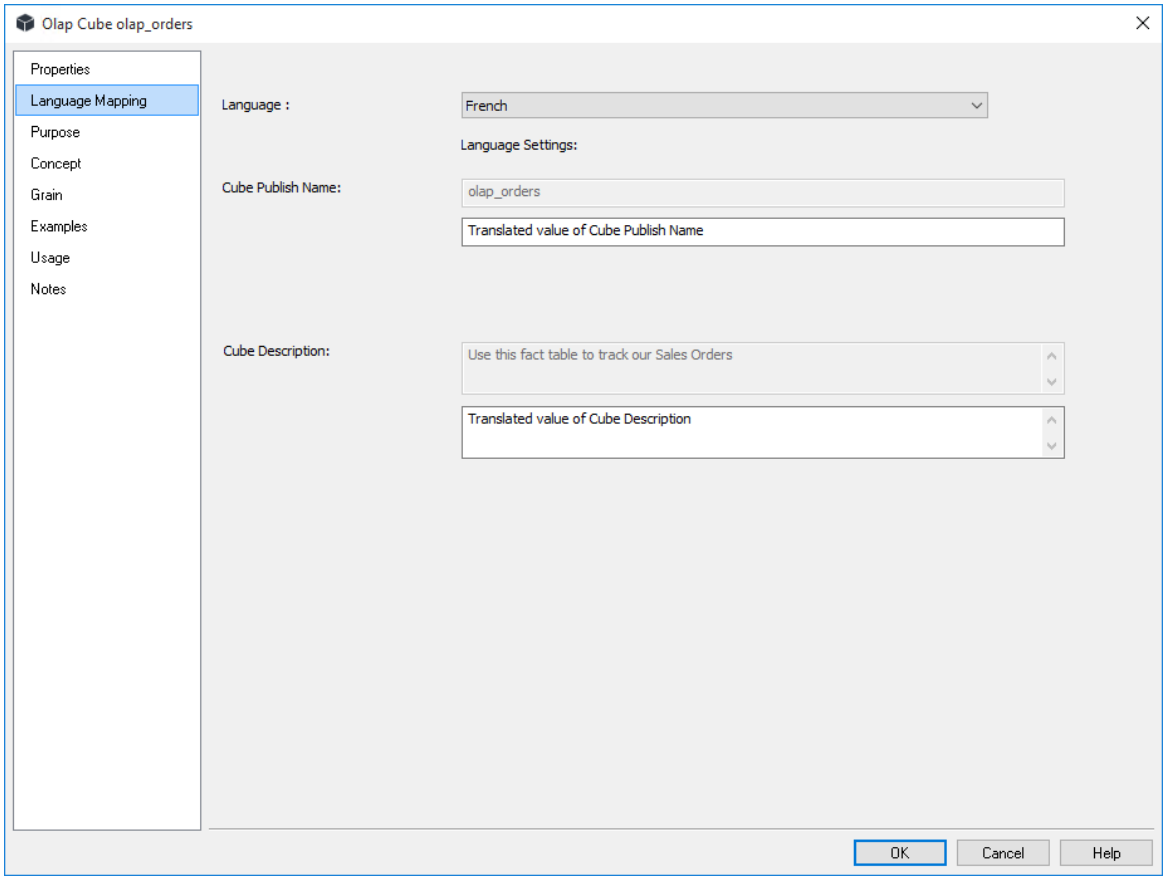
ROLAP stands for Relational Online Analytical Processing.

ROLAP is an alternative to the MOLAP (Multidimensional OLAP) technology. While both ROLAP and MOLAP analytic tools are designed to allow analysis of data through the use of a multidimensional data model, ROLAP differs significantly in that it does not require the pre-computation and storage of information. Instead, ROLAP tools access the data in a relational database and generate SQL queries to calculate information at the appropriate level when an end user requests it. With ROLAP, it is possible to create additional database tables (summary tables or aggregations) which summarize the data at any desired combination of dimensions.

While ROLAP uses a relational database source, generally the database must be carefully designed for ROLAP use. A database which was designed for OLTP will not function well as a ROLAP database. Therefore, ROLAP still involves creating an additional copy of the data. However, since it is a database, a variety of technologies can be used to populate the database.

Language Mapping

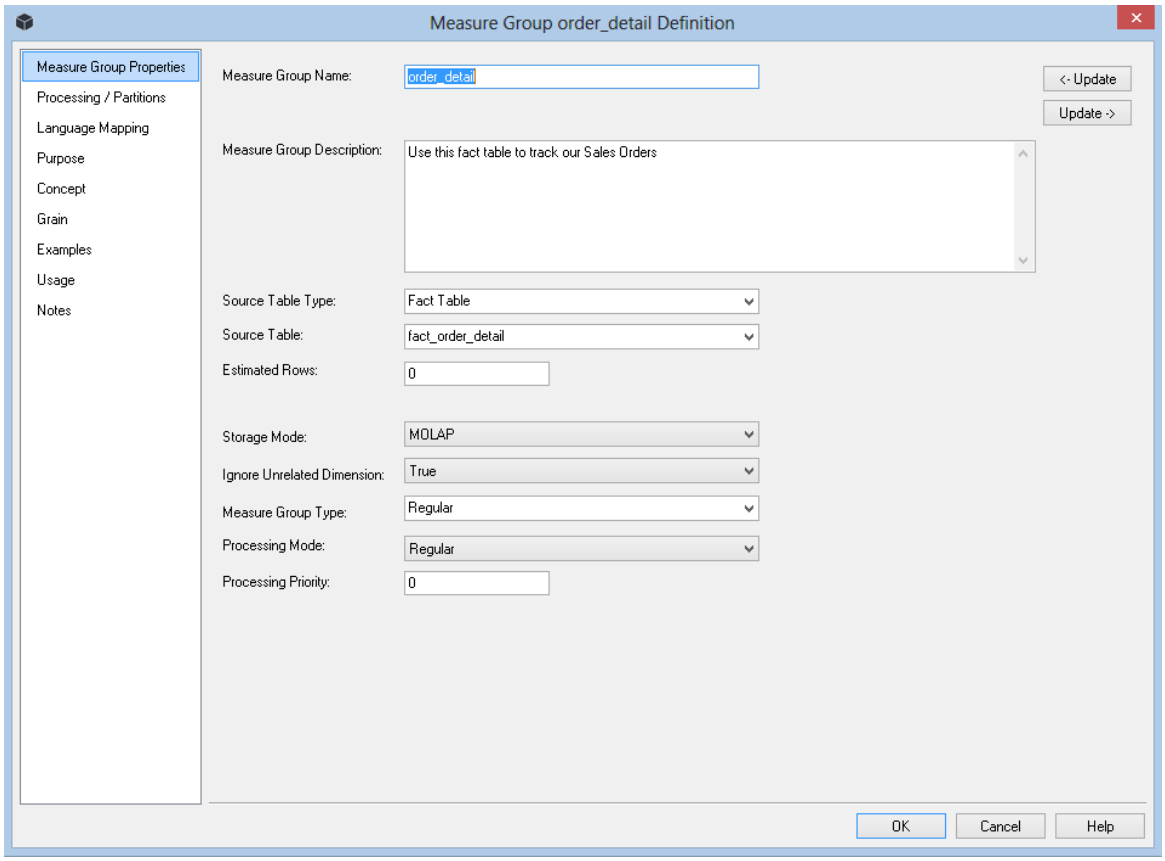
The OLAP Cube Properties screen has a tab called **Language Mapping**.
Select the language from the drop-down list and then enter the translations for the **Cube Publish Name** and the **Cube Description** in the chosen language.



See ***Language Settings*** (see "***Language Options***" on page 137) for more details on how to add languages for translation.

OLAP CUBE MEASURE GROUPS

A cube can contain multiple measure groups. In WhereScape RED each measure group can belong to a single cube, and each measure group relates to a single star schema. The Measure Groups Cube processing is defined on the tab entitled **Measure Group Properties**.
The Measure Group's Properties are shown by right-clicking on the Measure Group and choosing **Properties**. The Measure Group Properties associated with cubes are described below.



Measure Group Name

Specifies the name of the Measure Group in Analysis Services

Measure Group Description

Specifies the Metadata description of the Measure Group. This description is stored in Analysis Services and is also used in the WhereScape RED auto-generated documentation.

Update Buttons

The Update Buttons: **Update <-** and **Update ->** are used to move from the current Measure Group to the previous and next Measure Group respectively for the current OLAP cube. The alternative is to exit the Measure Group and choose the next Measure Group Properties screen.

Source Table Type

Specifies the type of table from which the Measure Group has been built in WhereScape RED.

Source Table

Specifies the table from which the Measure Group has been built and populated in WhereScape RED.

Estimated Rows

Specifies the estimated number of rows in the source table. If it is unknown then leave this value as 0.

Storage Mode

This field allows two options; MOLAP - Multidimensional OLAP or ROLAP - Relational OLAP. The default value is inherited from the value set at the Cube level. Again, setting this field at the Measure Group level will determine the default for the Storage Mode field on the related Partitions.

Ignore Unrelated Dimensions

Indicates whether dimensions that are unrelated to the Measure Group are forced to their top level when their members are included in a query.

Measure Group Type

Specifies the type of information the Measure Group contains. In some cases, this enables specific treatment by the server and client applications.

Processing Mode

Indicates whether processed data is available after all the aggregations have been computed (Regular) or immediately after the data has been loaded (Lazy Aggregations).

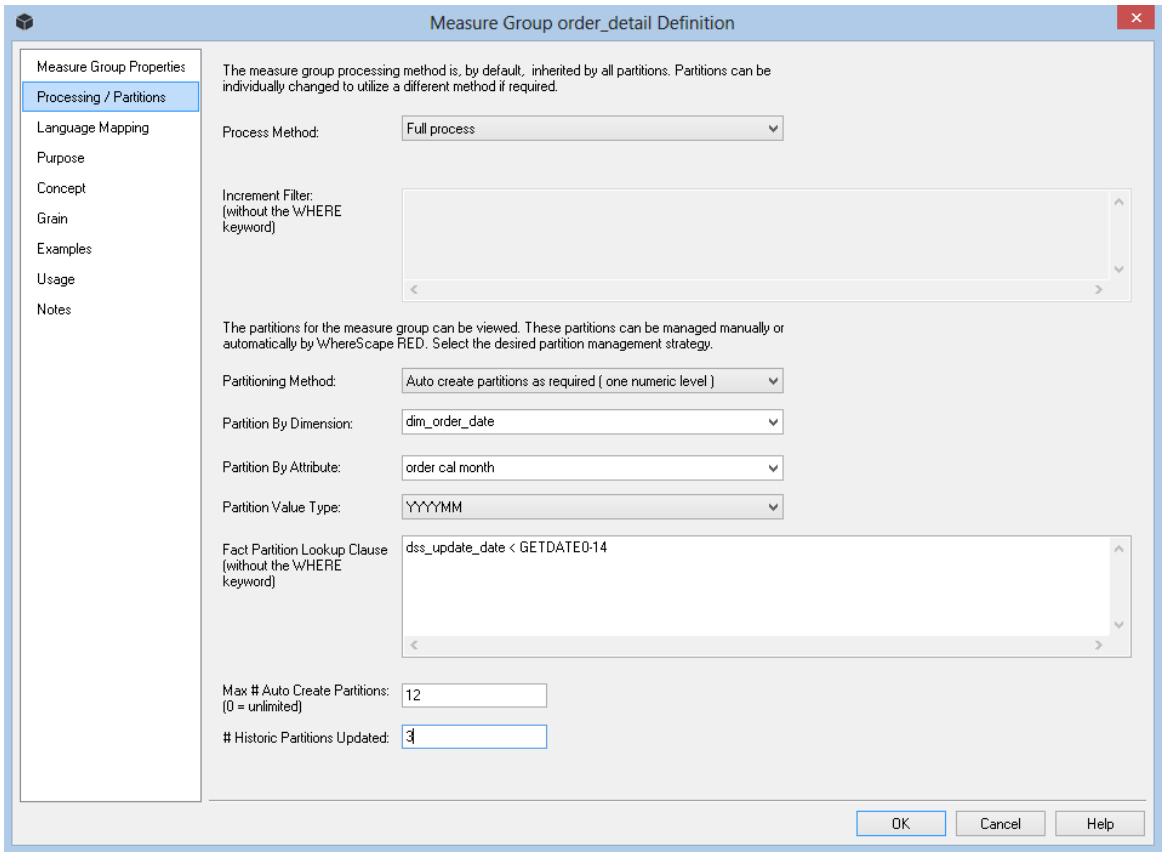
Processing Priority

Specifies the priority for processing the measure group.

OLAP CUBE MEASURE GROUP PROCESSING/PARTITIONS

Partitions define separately manageable data slices of the measure group data. Partitions can be created, processed and deleted independently within a Measure Group. Each Measure Group needs at least one partition to be defined to allow the Measure Group to be processed.

Partitions are managed through the **Processing/Partitions tab** of the Measure Group's Properties within WhereScape RED.



Process Method

The processing or updating method of a Measure Group is an area that requires careful consideration. The default option is **Full process** which will result in the Measure Group being rebuilt. This is in many ways the safest option, but processing time may mean other options must be chosen. The valid options are:

The following describes the processing methods that are available in Analysis Services for Measure Groups:

- **Default Process**

  Detects the process state of the measure group, and performs processing necessary to deliver a fully processed state.

- **Full Process**

  Processes an Analysis Services Measure Group regardless of state. When Process Full is executed against a Measure Group that has already been processed, Analysis Services drops all data, and then processes it.

- **Incremental Process**

  Adds newly available fact data and process only to the relevant partitions. To use this option, you must set the 'incremental filter' field with a statement that will result in only new data being selected. Failure to do so will result in duplicated data in the cube. In many data warehouses a fact table undergoes changes to the data as well as the addition of new data, and so the incremental update option is not possible. A validation regime should be put in place to compare cube data to the fact data if incremental is used. This validation regime should be used to notify the administrator in the event that duplicate data is inserted into the cube.
- **Update Data**

  Processes data only without building aggregations or indexes. If there is data in the partitions, it will be dropped before re-populating the partition with source data. This processing option is supported for dimensions, cubes, measure groups, and partitions.
- **Build Structure**

  If the cube is unprocessed, Analysis Services will process, if it is necessary, all the cube's dimensions. After that, Analysis Services will create only cube definitions. The build structure option just builds the cube structure without populating it. This can be useful if you have a very large cube and want to validate the design.

Increment Filter

If an incremental processing option is chosen, then a filter statement must be selected to only return those rows that are to be added to the Measure Group. As mentioned above, care must be taken when using incremental updates. For example, if the Measure Group is accidentally processed twice and the filter is based on date, then duplicate data will be inserted into the Measure Group without any warning.

Partitioning

Partitioning is useful for handling large datasets where it is impractical to reprocess the entire Measure Group. In such a case the full process option would probably be chosen but only selected partitions would be processed. See the section on partitioning for more information. The default process will perform a full process on the first pass followed by incremental updates on subsequent processing runs. Care should be taken when choosing default for the cube.

Partitioning Method

Three options are provided for handling Measure Group partitions. They are:

1. **One partition only**

   When this option is selected the partition information for the Measure Group is ignored and one partition is created and processed for the Measure Group. This would be the normal situation unless performance issues require an alternate strategy.
2. **Manually managed multiple partitions**

   With this option the partition information stored for the Measure Group is used in the creation and processing of the Measure Group.

3 Automatic partition handling

This option is available if the Measure Group is to be partitioned by one numeric value. The partitioning should preferably be on something like day, month or year. (i.e. YYYY, YYYYMM or YYYYDDD). If this option is chosen together with one of the date formats described above, then WhereScape RED will automatically create partitions as required and process only those partitions that are marked for processing.

Partition by Dimension

This field is only available if automatic partition handling is chosen. Select the dimension in the Measure Group that we will partition by. This would normally be a date dimension.

Partition by Attribute

This field is only available if automatic partition handling is chosen. Select the attribute that we are to partition by. This would normally be a year or maybe a month level. (e.g. cal_year, fin_year from the WhereScape date dimension).

Partition by Value Type

This field is only available if automatic partition handling is chosen. Select the type of level we are dealing with. Choose YYYY for a year partition and YYYYMM for a month partition. This format must correspond with the column in the date dimension. WhereScape RED only supports partitioning by Year, Quarter, month or day.

Fact Partition Lookup Clause

This field is only available if automatic partition handling is chosen. To know when to create a new partition WhereScape RED executes a query against the fact table and the date dimension to acquire each unique period. When dealing with a large fact table, such a query may take a long time to complete. This field can be used to include the components of the 'Where' clause to restrict the amount of data examined. For example we may enter 'dss_update_time < GETDATE()-14' to only look at fact table records that have been inserted or updated in the last 14 days. This should still allow us to catch any new partitions and add them. The first time a cube is converted to auto partitioning handling, a full pass of the fact table should occur to allow inclusion of every partition. This field should therefore only be populated once the cube has been initially built with all partitions intact.

Max Number of Auto Created Partitions

This field is only available if automatic partition handling is chosen. You can specify an upper limit for automatically created partitions. The default is zero, or no limit. This limit may be useful if your source system can get erroneous data. If set, then the processing of the Measure Group will fail if a new partition will exceed the counter.

Number of Historic Partitions Updated

This field is only available if automatic partition handling is chosen. This field allows you to restrict the partition updating to the latest nnn partitions. If for example, we were partitioning by year and we set this value to 2, we would process the current and previous years only. WhereScape RED turns off partition processing after it does a partition update, so the first pass will still update all partitions.

To Display Measure Groups

To display a list of measure groups defined against a cube, right-click on a cube and select **Display Measure Groups**.

To Add a Measure Group

To add a measure group, display the measure groups in the middle pane and either:

- Drag over a new fact table into the target pane - this will automatically create a new measure group in the cube. Any additional dimensions required to support analysis of the Measure Group will be added to the cube.
- Right-click on the cube in the object pane and select Add Measure Group and fill in the Measure Group properties.

To Delete a Measure Group

To delete a measure group, display the measure groups in the middle pane and right-click, select delete measure group.

Displaying Measures

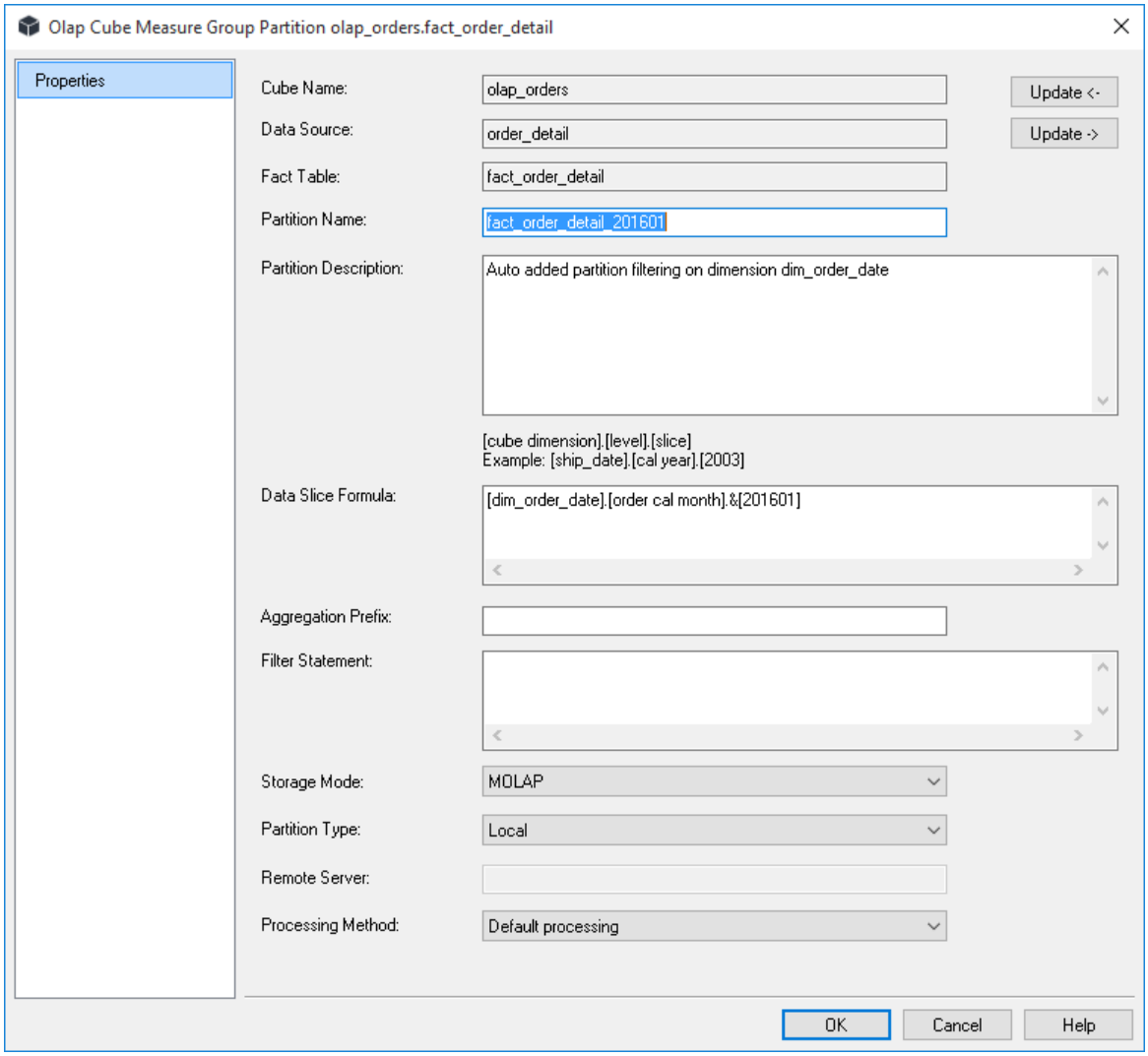
Measures can be displayed or added while viewing measure groups in the middle pane. Right-click on a measure group and select the appropriate option.

Displaying Partitions

Partitions can be displayed or added while viewing measure groups in the middle pane. Right-click on a measure group and select the appropriate option.

OLAP CUBE MEASURE GROUP PARTITIONS

The Measure Group Partition's properties are shown by right-clicking on the Measure Group Partition and selecting **Properties**. The partition's properties associated with a measure group are described below.



Cube Name

The name of the cube that the partition belongs to.

Data Source

The data source for the partitions. This will be inherited from the cube and cannot be changed. You cannot have partitions with different data sources or different fact tables in WhereScape RED. If you need to support either scenario, then the partition must be created directly within Analysis Services. In such a case, it can still be managed in terms of processing through WhereScape RED.

Fact Table

The fact table that the data is derived from. This is inherited from the cube and cannot be changed. See the notes above under data source.

Update Buttons

The Update Buttons: **Update <-** and **Update ->** are used to move from the current Measure Group Partition to the previous and next Measure Group Partition respectively for the current Measure Group. The alternative is to exit the Measure Group Partition and choose the next Measure Group Partition Properties screen.

Partition Name

Where only one partition exists, it is normally given the same name as the cube. If manually creating, then a unique name must be assigned for each partition. If auto partitioning is chosen, then WhereScape RED will use the cube name plus the level value to make the partition name.

Partition Description

A description of the partition for documentation purposes.

Data Slice Formula

This field defines the range of data stored in the partition. It is a very simplified version of what can be done in Analysis Services. If a more complex partitioning algorithm is required, then the partition will need to be created in Analysis Services. The format for the formula is as follows:

The brackets must surround each name and a full stop must separate the three parts of the formula. For example, a cube that is partitioned by year on its date dimension would have the following formula for the 2003 year. [dim_date].[cal_year].[2003]

Aggregation Prefix

By default, any cube aggregation will be prefixed with the partition name. An alternate name can be entered here. See Analysis Services for more details.

Filter Statement

Not implemented.

Storage Mode

This field allows two options; MOLAP - Multidimensional OLAP or ROLAP - Relational OLAP. This determines how the OLAP cube is processed. The default value is inherited from the value set at the Measure Group level.

Partition Type

The partition can be either Local or Remote. Local means that the partition resides on the same Analysis Services server as the cube. If Remote is chosen than a server must be specified where the partition will be located.

Remote Server

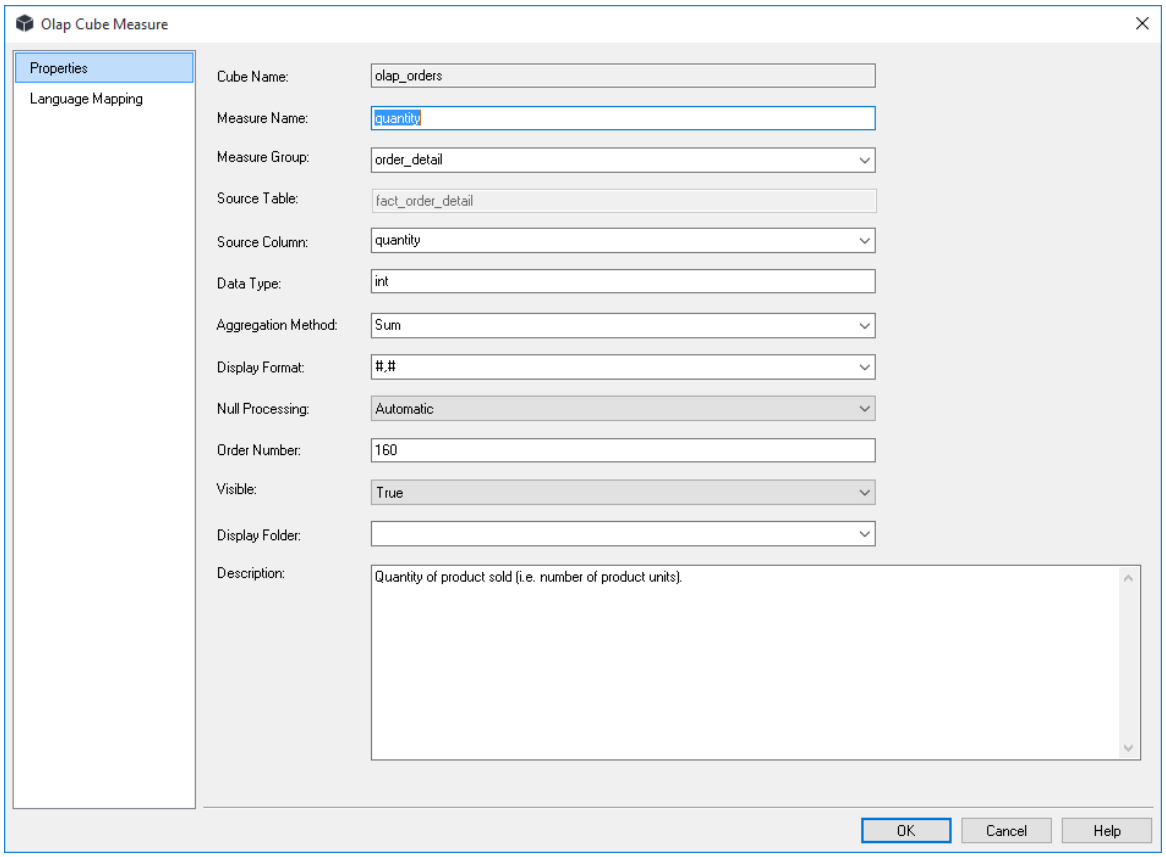
If a Remote partition, then the name of the remote Analysis Services server must be entered here.

Processing Method

A partition is either enabled for processing or disabled. This field can be set to Always process or Never process. If left as default and in automatic mode, then WhereScape RED will disable the processing once the partition has been aged out.

OLAP CUBE MEASURES

Measures represent the numeric attributes of a fact table that are aggregated using an OLAP aggregate function defined against each Measure. Each Measure is defined against a Measure Group, which is defined against a cube. The properties of a measure are shown by right-clicking on a Measure and choosing **Properties**. In more detail:



Cube Name

A Read Only field that indicates against which OLAP Cube the measure is defined.

Measure Name

Specifies the Analysis Services name defined for the measure.

Measure Group

Specifies against which Measure Group the Measure is defined. This is related to the fact table of which the Measure is an attribute.

Update Buttons

The Update Buttons: **Update <-** and **Update ->** are used to move from the current Measure to the previous and next Measure respectively for the current OLAP Cube Measure Group. The alternative is to exit the Measure and choose the next Measure Properties screen.

Source Table

A read only field that indicates the fact table that is related to the Measure Group.

Source Column

Specifies from which numeric attribute of the underlying fact table the Measure is built. This is a drop-down list populated from REDs metadata definition of the fact table associated with the Measure Group above.

Data Type

Specifies the data type used by Analysis Services. The data type specified for this property is inherited from the fact table attribute defined in WhereScape RED metadata but can be different (typically a larger data type is used in the cube to cope with the larger numbers generated by aggregating the source data).

Aggregation Method

Specifies the OLAP function used to aggregate measure values. The default options are:

- Sum
- Count
- Distinct Count - only one distinct count is allowed per Measure Group and can have query performance implications.
- Min - Minimum
- Max - Maximum

Display Format

Specifies the format used by clients when displaying the measure value.

Null Processing

Specifies the processing null values. Setting this property to Automatic means that Analysis Services uses default behavior.

Order Number

The order in which the measures appear in the cube is dictated by their order number.

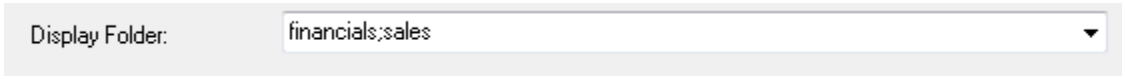
Visible

Measures are normally visible, but some measures used in calculations may be hidden. In such a case clear this checkbox.

Display Folder

Cube Measures can be organized into user-defined folders to view and manage these attributes within the Analysis Services user interface more easily. Enter the display folder name.

Note: One object can be in multiple display folders, for example:



Description

A description of the measure which is stored in the cube. This description will by default be acquired from the source column.

To View measures

The measures can be viewed by clicking on an OLAP Cube in the left pane which will display the measures in the middle pane.

To Add a New Measure

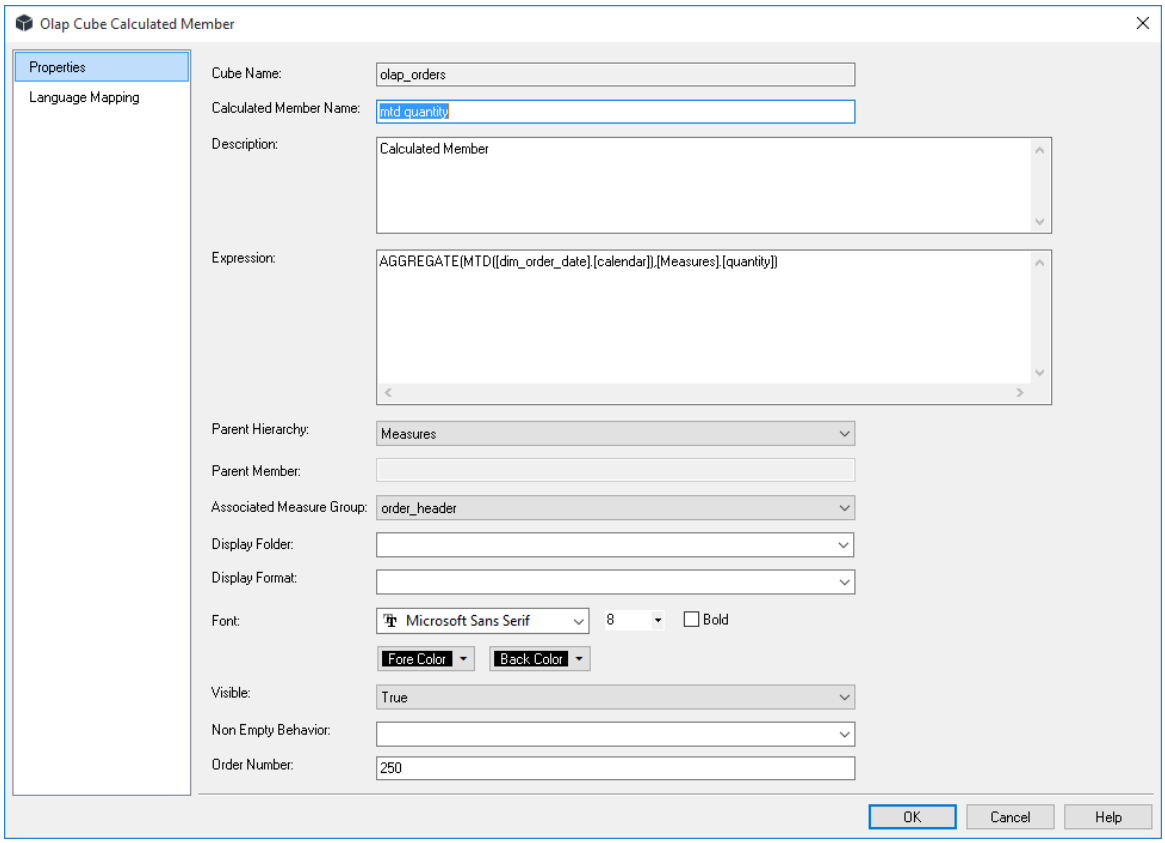
To add a new measure, view the measures in the middle pane, right-click in the middle pane and select add measure. This can also be done when viewing measure groups in the middle pane. Alternatively, to create measure which is very similar to an existing measure, view the measures in the middle pane and right-click, select **Copy Measure**. The same dialog box appears as for **Add Measure** with most of the fields filled in. Notice that the measure name has the suffix "- Copy". Change the name in the Measure Name field and make any other alterations and click **OK**.

To Delete a measure

Display the measures in the middle pane, select the measure to delete, right-click and select **Delete**.

OLAP CUBE CALCULATIONS

Calculations provide the ability to define the derivation of a value at query time within a cube. The calculation is most typically a numeric derivation based on measures, but can be defined against any dimension. The calculation is defined in MDX (Multi-Dimensional eXpressions). The definition of a Calculation is shown by right-clicking a **Calculation** and choosing **Properties**. The following Cube Calculated Member Definition dialog is shown:



Cube Name

A Read Only field that indicates against which OLAP Cube the Calculation is defined.

Calculated Member Name

Specifies the Analysis Services name defined for the Calculation.

Update Buttons

The Update Buttons: **Update <-** and **Update ->** are used to move from the current Calculation to the previous and next Calculation respectively for the current OLAP Cube. The alternative is to exit the Calculation and choose the next Calculation Properties screen.

Description

A business description of the Calculation that is used to populate WhereScape RED documentation.

Expression

Specifies the MDX expression that defines the calculation.

Parent Hierarchy

Specifies where the calculation is displayed for use. By default, and in most cases, this will be 'Measures'. This means that the calculated member will be displayed to the end user of the cube as a measure, otherwise known as a calculated measure. Alternatively, you can include the calculated member in a dimension instead of in the measures. Hierarchies are descriptive categories of a dimension by which the measures in a cube can be separated for analysis. A calculated member provides a new label in the parent dimension you select.

Parent Member

This is not available if you select **Measures** as your parent hierarchy, or if you select a one-level hierarchy. Hierarchies are divided into levels that contain members. Each member produces a heading in the cube. While browsing a cube, users can drill down to subordinate headings. The heading for the calculated member will be added directly below the selected Parent Member.

Associated Measure Group

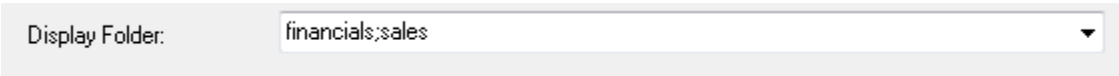
Specifies against which Measure Group the Measure is defined. This is related to the fact table of which the Measure is an attribute.

Note: SSAS 2008+ uses the property **Associated Measure Group** but previous versions of SSAS do not and can result in errors when creating the cube. It is possible to set this attribute in RED to **(Undefined)** for previous versions of RED, but this is not necessary for the current version of RED as this attribute will only be used when appropriate.

Display Folder

Cube Calculations can be organized into user-defined folders to view and manage these attributes within the Analysis Services user interface more easily. Enter the display folder name.

Note: One object can be in multiple display folders, for example:



Display Format

Specifies the format used by clients when displaying the measure value.

Visible

Specifies whether the calculation is visible to client tools.

Non Empty Behavior

Determines the non-empty behavior associated with the calculation

Order Number

The create order of the member in the dimension hierarchy.

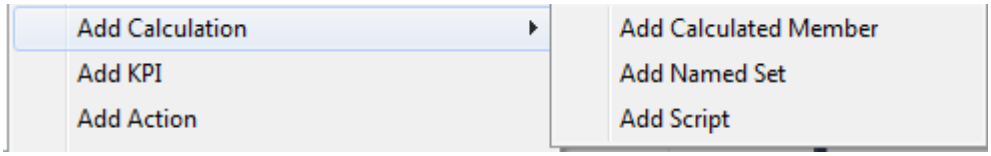
To View Calculations

To view the list of calculations (sometimes called a calculated measure), right-click on an OLAP Cube in the left pane and select **Display Calculations**.

To Add a Calculation

1 To add a calculation, right-click on an OLAP Cube in the left pane, select **Add Calculation** and then choose between:

- Add Calculated Member
- Add Named Set
- Add Script



This can also be achieved by displaying calculations in the middle pane, right-clicking and selecting **Add Calculated Member**, **Add Named Set** or **Add Script**. Fill out the dialog box with the relevant details.

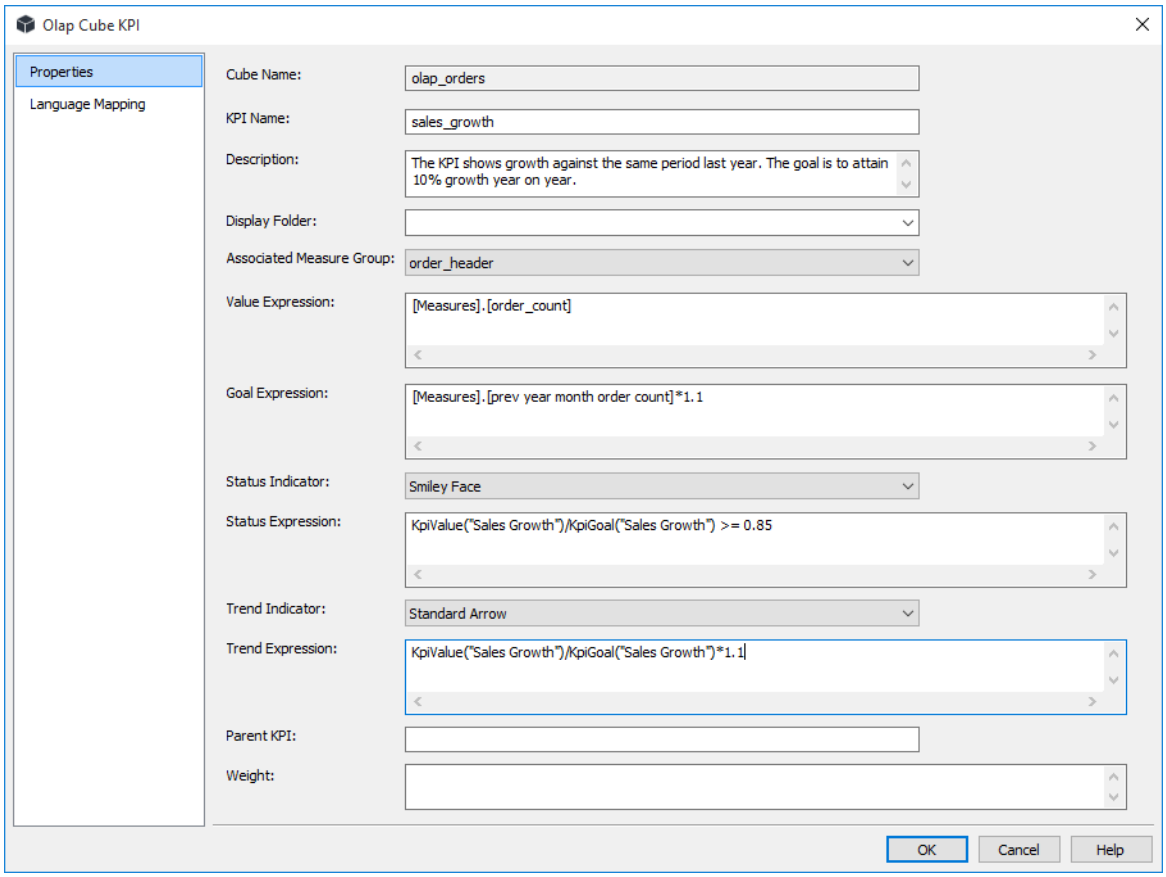
2 To create a calculation that is similar to an existing calculation, display the calculations in the middle pane and select **Copy Calculation**. The same dialog box appears as for **Add Calculation** with most of the fields filled in. Notice that the calculated member name has the suffix "- Copy". Change the name in the Measure Name field and make any other alterations and click **OK**.

To Delete a calculation

Display the calculations in the middle pane, select the calculation to delete, right-click and select **Delete**.

OLAP CUBE KEY PERFORMANCE INDICATORS

In Analysis Services, a KPI is a collection of calculations that are associated with a measure group in a cube that are used to evaluate business success. Typically, these calculations are a combination of Multidimensional Expressions (MDX) expressions or calculated members. KPIs also have additional metadata that provides information about how client applications should display the results of the KPI's calculations. The definition of a Calculation is shown by right-clicking a Calculation and choosing Properties.
The following Cube KPI Definition dialog is shown below:



Cube Name

A Read Only field that indicates against which OLAP Cube the KPI is defined.

KPI Name

Specifies the name of the KPI defined in Analysis Services.

Description

A business description of the KPI that is used to populate WhereScape RED documentation.

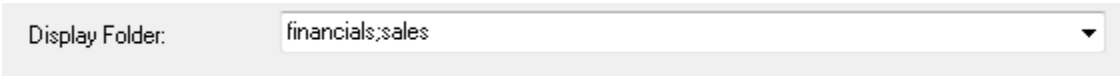
Update Buttons

The Update Buttons: **Update <-** and **Update ->** are used to move from the current KPI to the previous and next KPI respectively for the current OLAP Cube. The alternative is to exit the KPI and choose the next KPI Properties screen.

Display Folder

KPIs can be organized into user-defined folders to view and manage these attributes within the Analysis Services user interface more easily. Enter the display folder name.

Note: One object can be in multiple display folders, for example:



Associated Measure Group

Specifies the Measure Group against which the KPI is defined.

Value Expression

An MDX numeric expression that returns the actual value of the KPI.

Goal Expression

An MDX numeric expression or a calculation that returns the target value of the KPI.

Status Indicator

A visual element that provides a quick indication of the status of a KPI. The display of the element is determined by the value of the MDX expression that evaluates status.

Status Expression

An MDX expression that represents the state of the KPI at a specified point in time.

The status MDX expression should return a normalized value between -1 and 1. Values equal to or less than -1 will be interpreted as "bad" or "low." A value of zero (0) is interpreted as "acceptable" or "medium." Values equal to or greater than 1 will be interpreted as "good" or "high."

An unlimited number of intermediate values can optionally be returned and can be used to display any number of additional states, if supported by the client application.

Trend Indicator

A visual element that provides a quick indication of the trend for a KPI. The display of the element is determined by the value of the MDX expression that evaluates trend.

Trend Expression

An MDX expression that evaluates the value of the KPI over time. The trend can be any time-based criterion that is useful in a specific business context.

The trend MDX expression enables a business user to determine whether the KPI is improving over time or degrading over time.

Parent KPI

A reference to an existing KPI that uses the value of the child KPI as part of computation of the parent KPI. Sometimes, a single KPI will be a computation that consists of the values for other KPIs. This property facilitates the correct display of the child KPIs underneath the parent KPI in client applications.

Weight

An MDX numeric expression that assigns a relative importance to a KPI. If the KPI is assigned to a parent KPI, the weight is used to proportionally adjust the results of the child KPI value when calculating the value of the parent KPI.

To View KPIs

To view the list of KPIs, right-click on an OLAP Cube in the left pane and select **Display KPIs**.

To Add a KPI

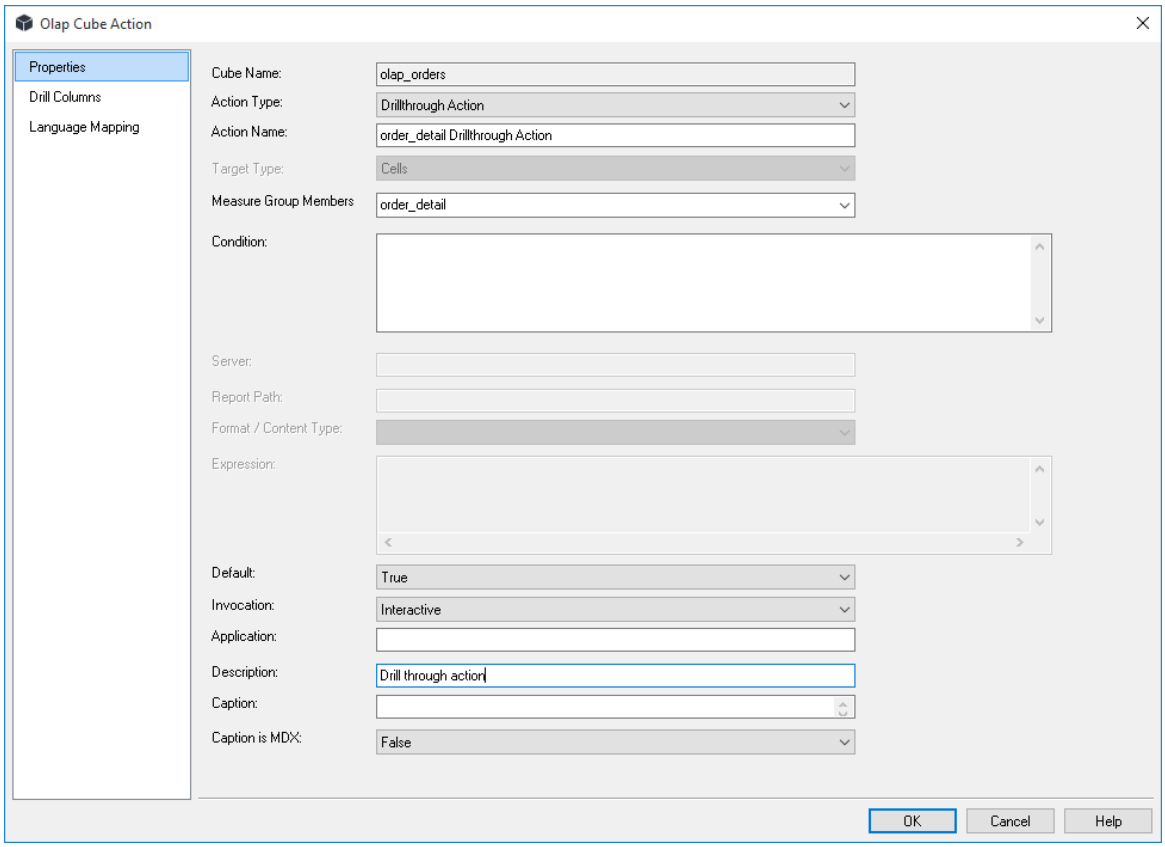
1. To add a calculation, right-click on an OLAP Cube in the left pane and select **Add KPI**. This can also be achieved by displaying KPIs in the middle pane, right-clicking and selecting **Add KPI**. Fill out the dialog box with the relevant details.
2. To create a KPI that is similar to an existing KPI, display the KPIs in the middle pane and select **Copy KPI**. The same dialog box appears as for **Add KPI** with most of the fields filled in. Notice that the KPI name has the suffix "- Copy". Change the name in the KPI Name field and make any other alterations and click **OK**.

To Delete a KPI

Display the KPIs in the middle pane, select the KPI to delete, right-click and select **Delete**.

OLAP CUBE ACTIONS

An action provides information to a client application to allow an action to occur based on the property of a clicked dimensional member. Actions can be of different types and they have to be created accordingly.
To view the definition of an Action, right-click on the **Action** and select **Properties**. The following Cube Action Definition dialog displays:



Cube Name

A Read Only field that indicates against which OLAP Cube the Calculation is defined.

Action Type

Actions can be of the following types:

- **Drill through actions** which return the set of rows that represents the underlying data of the selected cells of the cube where the action occurs. When this option is chosen, an additional tab is enabled that allows drill through columns to be chosen.

- **Reporting actions** which return a report from Reporting Services that is associated with the selected section of the cube where the action occurs.

- **Standard actions** (Action), which return the action element (URL, HTML, DataSet, RowSet, and other elements) that is associated with the selected section of the cube where the action occurs.

The action type chosen determines which action specific fields are enabled within the dialog.

Action Name

Defines the name of the Action in the Cube.

Update Buttons

The Update Buttons: **Update <-** and **Update ->** are used to move from the current Action to the previous and next Action respectively for the current OLAP Cube. The alternative is to exit the Action and choose the next Action Properties screen.

Target Type

Select the object to which the action is attached. Generally, in client applications, the action is displayed when end users select the target object; however, the client application determines which end-user operation displays actions. For Target type, select from the following objects:

- Attribute members
- Cells
- Cube
- Dimension members
- Hierarchy
- Hierarchy members
- Level
- Level members
-

Target Object:

The cube object of the designated target type against which the action is defined.

Condition

Specify an optional Multidimensional Expressions (MDX) expression that resolves to a Boolean value. If the value is True, the action is performed on the specified target. If the value is False, the action is not performed.

Report Action: Server

The name of the computer running report server.

Report Action: Report Path

The path exposed by report server.

Format/Content Type

Select the type of action. The following table summarizes the available types.

- Data Set - Retrieves a dataset
- Proprietary - Performs an operation by using an interface other than those listed in this table.
- Row Set - Retrieves a rowset.
- Statement - Runs an OLE DB command.
- URL - Displays a page in an Internet Browser.

Expression

Specifies the parameters that are passed when the action is run. The syntax must evaluate to a string, and you must include an expression written in MDX. MDX expressions are evaluated before the parameters are passed.

Default

An additional true/false drop-down list is enabled for Drill through actions. This gets or sets the current DrillThroughAction as the default action when multiple drillthrough actions are defined. This is important for users of Excell 2007 to browse the Olap Cube because Excel will only invoke the Drill through Action marked as the default.

Invocation

Specifies how the action is run. Interactive, the default, specifies that the action is run when a user accesses an object. The possible settings are:

- Batch
- Interactive
- On Open

Application

Describes the application of the action.

Description

Describes the action.

Caption

Provides a caption that is displayed for the action.

Caption is MDX

If the caption is MDX, specify True, if not specify False.

To View Actions

To view the list of Actions, right-click on an OLAP Cube in the left pane and select **Display Actions**.

To Add an Action

1. To add an Action, right-click on an OLAP Cube in the left pane and select **Add Action**. This can also be achieved by displaying Actions in the middle pane, right-clicking and selecting **Add Action**. Fill out the dialog box with the relevant details.
2. To create an Action that is similar to an existing Action, display the Actions in the middle pane and select **Copy Action**. The same dialog box appears as for **Add Action** with most of the fields filled in. Notice that the Action name has the suffix "- Copy". Change the name in the Action Name field and make any other alterations and click **OK**.

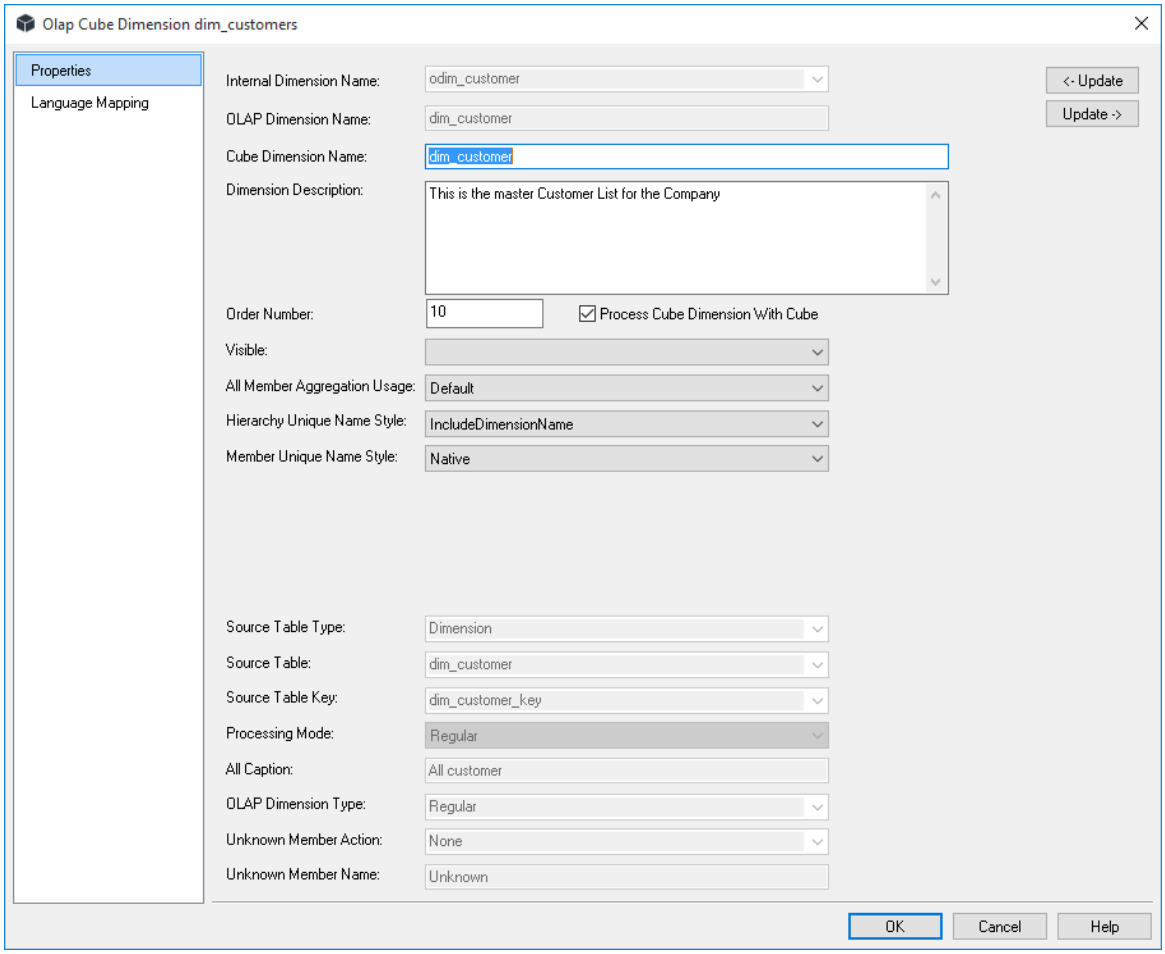
To Delete an Action

Display the Actions in the middle pane, select the Action to delete, right-click and select **Delete**.

OLAP CUBE DIMENSIONS

OLAP Dimensions are associated automatically with a cube when a cube is created in WhereScape RED based on the underlying star schema. OLAP Dimensions that are associated with a cube can be displayed, or additional OLAP Dimensions can be manually added from the list of OLAP Dimensions defined in WhereScape RED.

Once an OLAP Dimension is associated with a cube a relationship is created with the relevant Measure Groups within the cube - these relationships are defined automatically with WhereScape RED, and they can also be added. The properties of an OLAP Dimension associated with a cube are shown by right-clicking the cube dimensions listed in the middle pane and selecting **Properties**. The properties are shown below:



Internal Dimension Name

A read only field displaying the name of the OLAP Dimension in WhereScape RED.

OLAP Dimension Name

Specifies the name of the OLAP Dimension as a Dimension in Analysis Services.

Cube Dimension Name

Specifies the exposed name of the Dimension when associated with a cube (this can be different from the OLAP Dimension Name).

Update Buttons

The Update Buttons: **Update <-** and **Update ->** are used to move from the current OLAP Dimension to the previous and next OLAP Dimension respectively for the current OLAP Cube. The alternative is to exit the OLAP Dimension and choose the next OLAP Dimension Properties screen.

Dimension Description

A description of the dimension when associated with the cube.

Order Number

The order number.

Process Cube Dimension With Cube

Using this check-box you can enable the dimension to be processed with the OLAP Cube.

Visible

Determines whether the dimension is visible to client applications.

All member Aggregation Usage

Specifies how aggregations will be designed by the BIDS Storage Design Wizard if it is used to design cube aggregations.

Hierarchy Unique Name Style

Indicates whether the dimension name will be included in the name of the hierarchies. If set to Default, then the system will apply a default behavior and will include the dimension name in the case where there is more than one usage of the same dimension.

Member Unique Name Style

Indicates how member unique names will be formed.

Other Read only fields that are displayed in this dialog are configurable against the OLAP Dimensions' properties and cannot be changed in this dialog, including:

- Source Table Type
- Source Table
- Source Table Key
- Processing Mode
- All caption
- OLAP Dimension Type
- Unknown Member Action
- Unknown Member Name

To View Cube Dimensions

To view the list of Dimensions associated with a cube, right-click on an OLAP Cube in the left pane and select **Display Dimensions**.

To Add a Cube Dimension

To add an existing OLAP Dimension, right-click on an OLAP Cube in the left pane and select **Add Dimension**. This can also be achieved by displaying Dimensions in the middle pane, right-clicking and selecting **Add Dimension**. Fill out the dialog box with the relevant details.

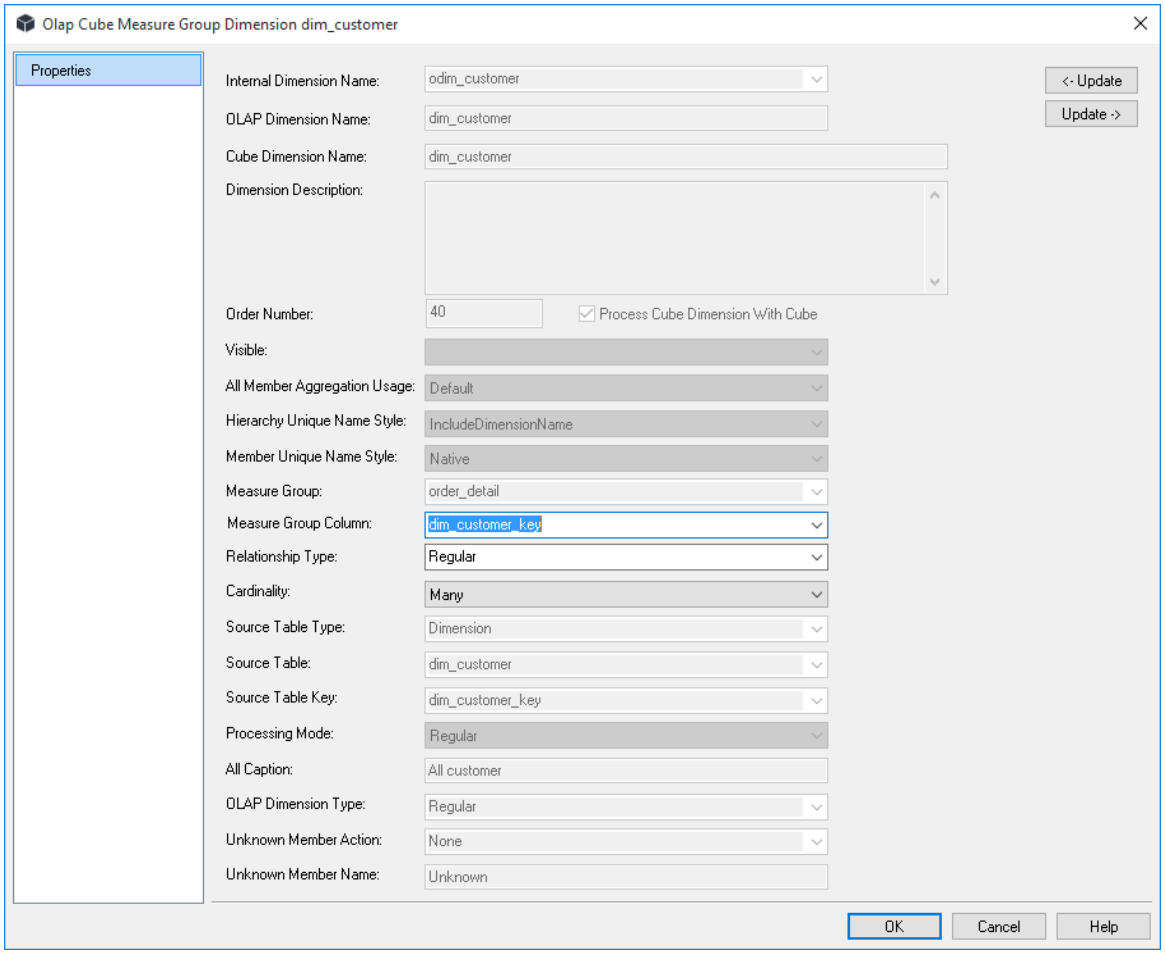
To Remove a Cube Dimension

Display the Cube Dimensions in the middle pane, select the Dimension to remove, right-click and select **Remove Dimension from Cube**. This action removes the association of the OLAP Dimension from the OLAP Cube.

OLAP CUBE MEASURE GROUP DIMENSIONS

Measure group dimensions are the relationships between cube Measure Groups and OLAP Dimensions. In WhereScape RED this equates to the relationships between fact tables and dimensions in the underlying star schema.

The **Properties** are shown below:



Internal Dimension Name

A read only field displaying the name of the OLAP Dimension in WhereScape RED.

OLAP Dimension Name

A read only field displaying the name of the OLAP Dimension as a Dimension in Analysis Services.

Cube Dimension Name

A read only field displaying the name of the Dimension when associated with a cube (this can be different from the OLAP Dimension Name).

Update Buttons

The Update Buttons: **Update <-** and **Update ->** are used to move from the current OLAP Measure Group Dimension to the previous and next OLAP Measure Group Dimension respectively for the current OLAP Cube. The alternative is to exit the OLAP Measure Group Dimension and choose the next OLAP Measure Group Dimension Properties screen.

Dimension Description

A description of the dimension when it is associated with the cube.

Order Number

The order number.

Update Dimension with Cube

A read only checkbox showing whether the dimension is processed when the cube is processed.

Visible

A read only field displaying the visibility of the dimension on the cube.

All Member Aggregation Usage

A read only field displaying how aggregations will be designed by the BIDS Storage Design Wizard if it is used to design cube aggregations.

Hierarchy Unique Name Style

A read only field which indicates whether the dimension name will be included in the name of the hierarchies.

Member Unique Name Style

A read only field which indicates how member unique names will be formed. A read only field.

Measure Group

A read only field displaying the Measure Group which is being referenced by the Measure Group Dimension relationship.

Measure Group Column

Specifies which fact table key joins to the dimension key.

Relationship Type

Defines the relationship type for the relationship between the Dimension and Measure Group. In WhereScape RED this option can be Regular, which means that the relationship is based on a dimension key join, or No Relationship between the Measure Group and Dimension.

Cardinality

Indicates whether the measure group has a many to one or one to one relationship with the dimension.

Source Table Type

A read only field that displays the type of table from which the OLAP Dimension was created.

Source Table

A read only field that displays the name of the table from which the OLAP Dimension was created.

Source Table Key

A read only field that displays the key (typically the primary key) that relates the dimension to the fact in the underlying star schema.

Processing Mode

A read only field that indicates whether processed data is available after all aggregations have been computed or immediately after the data has been loaded.

All Caption

A read only field that displays the name of the (All) member. This applies to all hierarchies in the dimension that have an (All) member.

OLAP Dimension Type

A read only field that displays the type of information contained by the dimension.

Unknown Member Action

A read only field that displays the existence of an Unknown member and whether that member is visible or hidden.

Unknown Member Name

A read only field that displays the caption for the unknown member.

OLAP DIMENSION OBJECTS

OLAP DIMENSION OVERVIEW

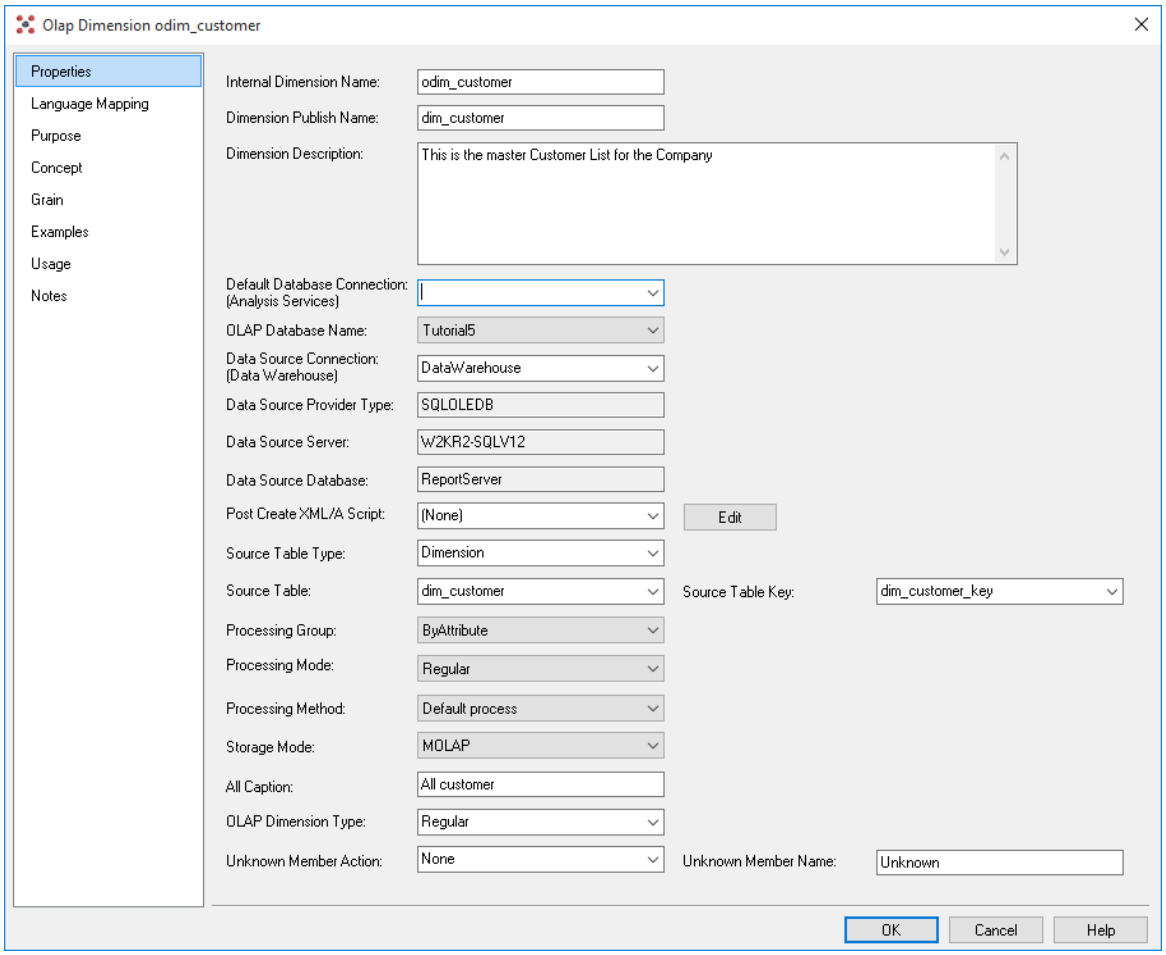
OLAP Dimensions are dimensions that get created in an Analysis Services database.

An OLAP Dimension is a collection of related attributes which can be used to provide information about fact data in one or more cubes. By default, attributes are visible as attribute hierarchies and can be used to understand the fact data in a cube. Attributes can be organized into user-defined hierarchies that provide navigational paths to assist users when browsing the data in a cube.

They are typically created and populated from a relational dimension.

One or more OLAP Dimensions are defined automatically by WhereScape RED when a fact table is dragged over to create a cube or measure group. WhereScape RED will take the relational dimension tables and related metadata (including hierarchies) defined in the star schemas and create OLAP Dimensions automatically. They can also be defined manually in WhereScape RED.

The properties of an OLAP Cube dimension are shown by right-clicking on the **OLAP Dimension** and choosing **Properties**. The following dialog is shown:



Internal Dimension Name

Specifies the name of the dimension in WhereScape RED.

Dimension Publish Name

Specifies the name of the dimension as created in Analysis Services.

Dimension Description

A business description of the OLAP Dimension for use in documentation - this description also gets created in the analysis services metadata.

Default Database Connection and OLAP Database Name

The WhereScape RED connection that is an OLAP connection to an analysis services server. These fields only need to be populated when the OLAP Dimension needs to be created in Analysis Services separately from a cube. If these fields are blank this dimension can only be created in the same Analysis Services server and database as the related cubes when the cubes get created.

Data Source Connection

Defines the WhereScape RED connection that points to the relational dimensional table(s) used to populate the OLAP Dimension - typically the Data Warehouse connection. When the connection is defined, the following read only fields are populated with the connection information:

- Data Source Provider Type
- Data Source Server
- Data Source Database

Post Create XML/A Script

Specifies XML/A script that is executed after the object is created.

Source Table Type

Specifies the type of table from which the OLAP Dimension was created.

Source Table

Specifies the name of the table from which the OLAP Dimension was created.

Source Table Key

Specifies the key (typically the primary key) that relates the dimension to the fact in the underlying star schema.

Processing Group

Specifies the processing group for processing the dimension. This determines how much data is read into memory during dimension processing at any one time.

Processing Mode

Indicates whether processed data is available after all aggregations have been computed (Regular) or immediately after the data has been loaded (Lazy aggregations).

Processing Method

Indicates which processing method should be used for populating the dimension:

- Process Default - Detects the process state of an object, and performs processing necessary to deliver unprocessed or partially processed objects to a fully processed state.
- Process Full - Processes an Analysis Services object and all the objects that it contains. When Process Full is executed against an object that has already been processed, Analysis Services drops all data in the object, and then processes the object. This kind of processing is required when a structural change has been made.
- Rebuild Data - Processes data only without building aggregations or indexes.

Storage Mode

This field allows two options; MOLAP - Multidimensional OLAP or ROLAP - Relational OLAP. This determines how the OLAP cube is processed.

All Caption

Specifies the name of the (All) member. This applies to all hierarchies in the dimension that have an (All) member.

OLAP Dimension Type

Specifies the type of information contained by the dimension. Some client tools can treat the dimension differently based on this information.

Unknown Member Action

Specifies the existence of an Unknown member and whether that member is visible or hidden. Fact data not associated with a member can be associated with the unknown member.

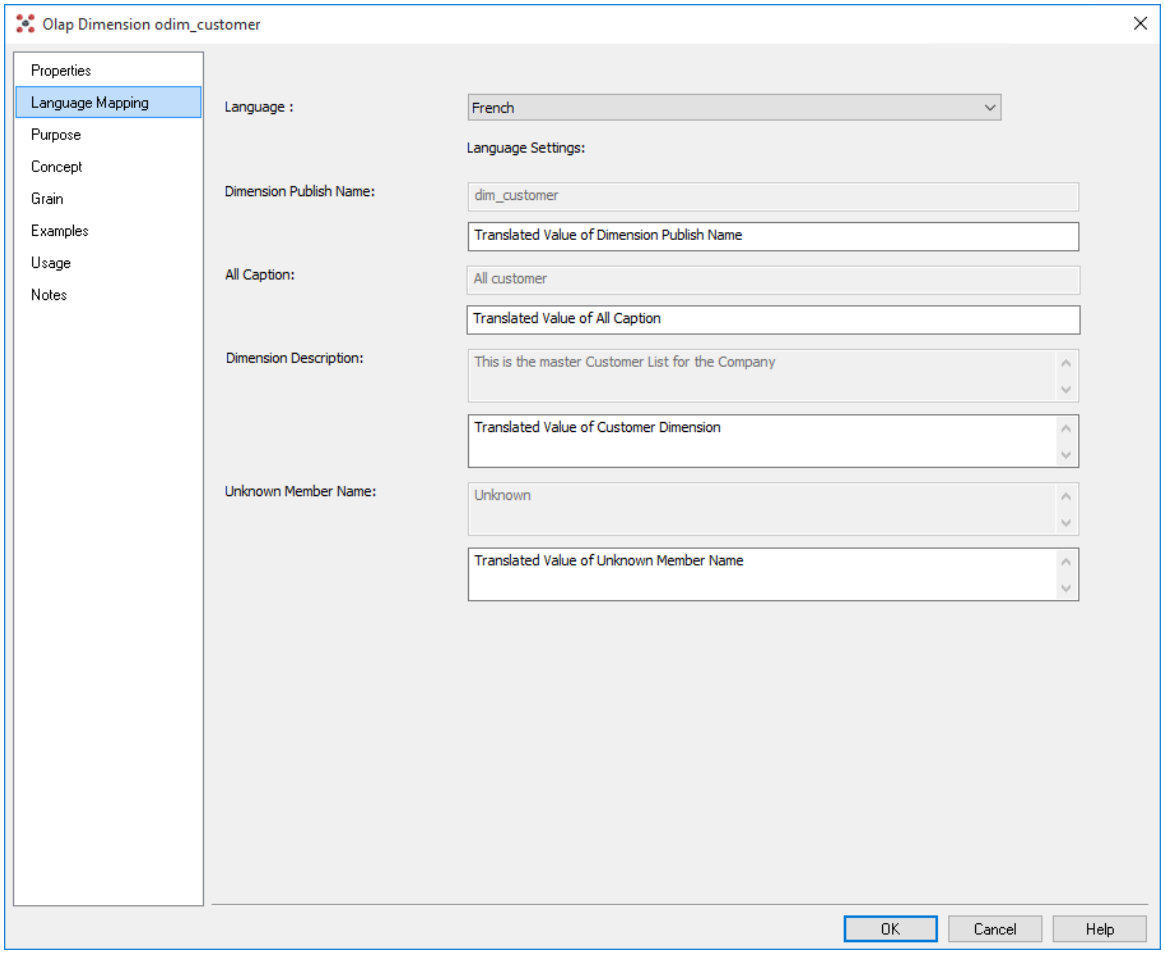
Unknown Member Name

Specifies the caption for the unknown member.

Language Mapping

The OLAP Dimension Properties screen has a tab called **Language Mapping**.

Select the language from the drop-down list and then enter the translations for the **Dimension Publish Name**, **All Caption**, **Dimension Description** and the **Unknown Member Name**.

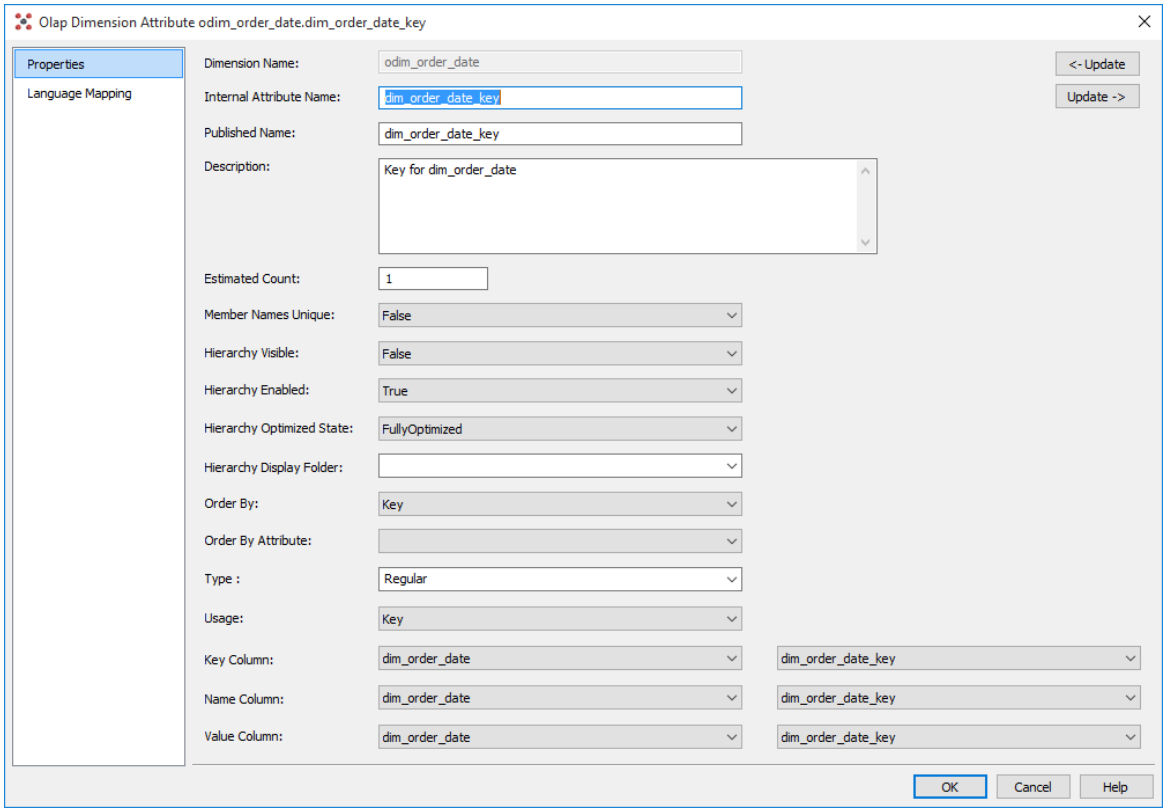


See ***Language Settings*** (see "***Language Options***" on page 137) for more details on how to add languages for translation.

OLAP DIMENSION ATTRIBUTES

Dimensional attributes contain information about the Dimension object. Attributes are exposed in the cube to provide the ability to navigate and aggregate the data in the cube.
User defined hierarchies can be built over attributes to provide drill paths through the data and to aid aggregation.

The properties of an attribute can be displayed by right-clicking an attribute in the middle pane and choosing **Properties**. The following dialog is displayed:



Dimension Name

A read only field to display the dimension which the attribute is related to.

Internal Attribute Name

The name of the attribute in WhereScape RED.

Published Name

The name of the attribute created in Analysis Services.

Description

A business name that is stored in WhereScape RED for documentation and stored in the Analysis Services metadata.

Estimated Count

Specifies the number of members in the attribute. This number is either the amount last counted by Analysis Services or a user provided estimate of the member count.

Member Names Unique

Indicates whether member names are unique for this attribute.

Hierarchy Visible

Indicates whether the attribute hierarchy is visible to client applications. Even if the attribute hierarchy is not visible it can still be used in a user defined hierarchy .

Hierarchy Enabled

Indicates whether an attribute hierarchy is enabled for this attribute. If the attribute hierarchy is not enabled then the attribute cannot be used in a user defined hierarchy.

Hierarchy Optimized State

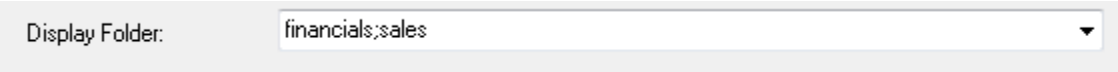
Specifies the level of optimization applied to the attribute hierarchy.

Hierarchy Display Folder

Attribute hierarchies can be organized into user-defined folders to view and manage these attributes within the Analysis Services user interface more easily. Enter the display folder name.

Notes:

1. One object can be in multiple display folders, for example:



2. Dimension Attributes and Dimension Hierarchies have the same display folder drop-down list.

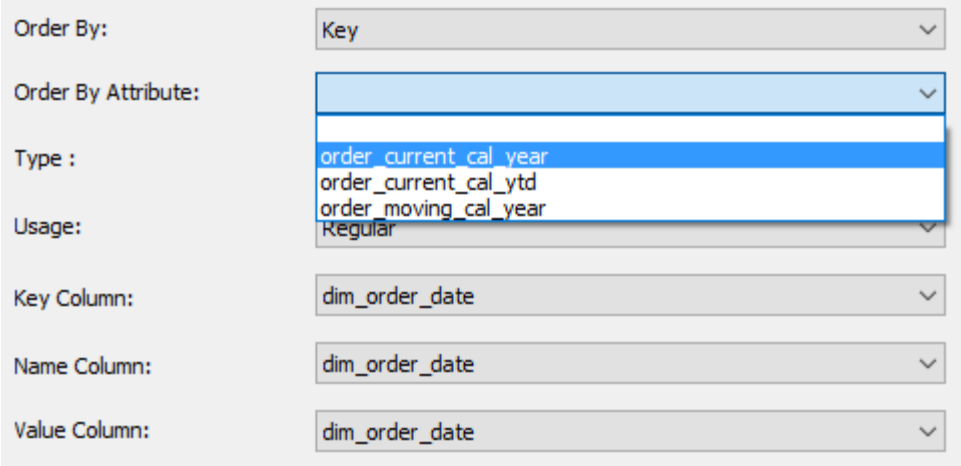
Order by

Specifies the method used to order the members of the attribute.

Order by attribute

Specifies the attribute used to order the members of the attribute hierarchy

If the **Order By** property is set to 'AttributeKey' or 'AttributeName' then **Order By Attribute** cannot be empty. It must be populated with values from attribute relationships.



Type

Specifies the type of information contained by the attribute.

Usage

Specifies the usage of the attribute.

Key Column

Specifies the details of the binding to the column containing the member key.

Name Column

Specifies the details of the binding to the column containing the member name.

Value Column

Specifies the details of the binding to the column containing the member value.

Using the Value Column OLAP cube attribute setting for Excel date filtering
In the relevant OLAP Date dimension ensure the OLAP Dimension Type property is set to "Time", then for the Key Attribute of the OLAP Date Dimension (e.g. dim_date_key) set the Value Column property to a date data type column (e.g. calendar_date). Usually it will be useful to set the Name Value property for the Key Attribute to a column containing a textual date format (e.g. dates presented in dd/mm/yy format). After publishing and processing the OLAP cube use Microsoft Office Excel PivotTables to expose date-specific filtering options for this dimension's hierarchies instead of label filtering options.

To View Attributes

To view the list of Attributes, right-click on an OLAP Dimension in the left pane and select **Display Attributes**.

To Add an Attribute

1. Display the attributes of an OLAP Dimension in the middle pane. Display the columns of the dimension in the right pane then drag over a column from the underlying relational dimension into the middle pane.
2. To add an Attribute, right-click on an OLAP Dimension in the left pane and select **Add Attribute**. This can also be achieved by displaying Attributes in the middle pane, right-clicking and selecting **Add Attribute**. Fill out the dialog box with the relevant details.
3. To create an Attribute that is similar to an existing Attribute, display the Attributes in the middle pane and select **Copy Attribute**. The same dialog box appears as for **Add Attribute** with most of the fields filled in. Notice that the Attribute name has the suffix "- Copy". Change the name in the Attribute Name field and make any other alterations and click **OK**.

To Delete an Attribute

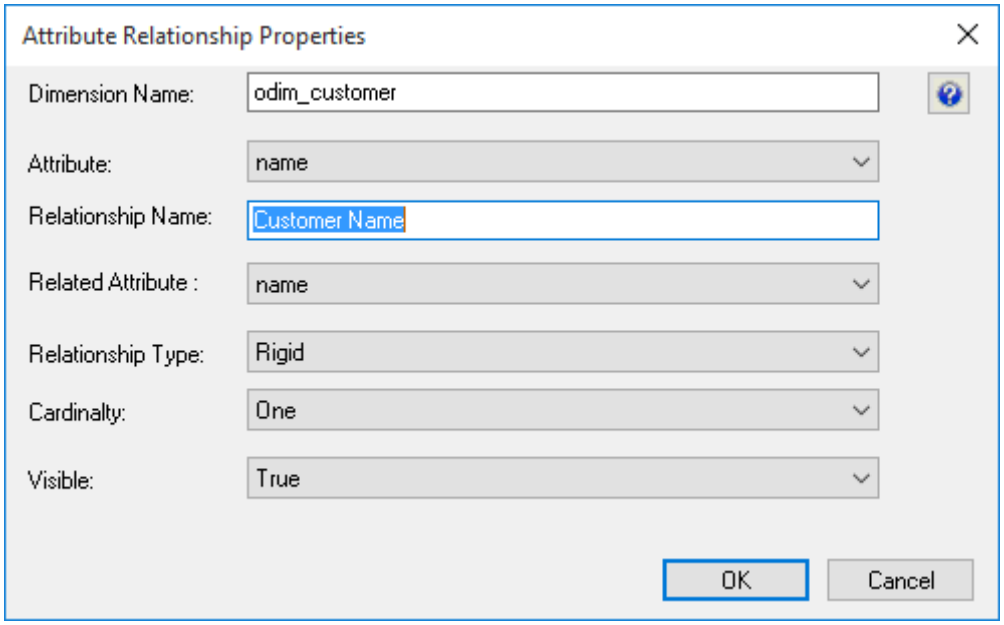
Display the Attributes in the middle pane, select the Attribute to delete, right-click and select **Delete**.

OLAP DIMENSION ATTRIBUTE RELATIONSHIPS

Attribute relationships define functional dependencies between attributes. If attribute A has a related attribute B in a hierarchical relationship, then an attribute relationship can be defined that assists Analysis Services to aggregate data.

Note: Attribute relationships require unique members to function correctly - if there are duplicate values participating in an attribute relationship then cube data can be incorrectly aggregated and displayed.

To view the **Properties** of an attribute relationship right-click on an attribute relationship and select **Properties** to show the dialog below:



Dimension Name

A read only field indicating the dimension associated with the attribute relationship.

Attribute

Specifies the attribute on which the attribute relationship is based.

Relationship Name

Specifies the name of the attribute relationship.

Related Attribute

Specifies the name of the related attribute.

Relationship Type

Indicates whether the relationship between attributes can change over time.

Cardinality

Indicates if the related attribute has a many to one or a one to one relationship with this attribute.

Visible

Indicates whether the attribute relationship is visible to the client applications.

To View Attribute Relationships

To view the list of Attribute Relationships, right-click on an OLAP Dimension in the left pane and select **Display Attribute Relationships**.

To Add an Attribute Relationship

1. Attribute relationships are defined automatically when a User Defined hierarchy is inherited from an underlying relational dimension.
2. To add an Attribute Relationship, right-click on an OLAP Dimension in the left pane and select **Add Attribute Relationship**. Fill out the dialog box with the relevant details.

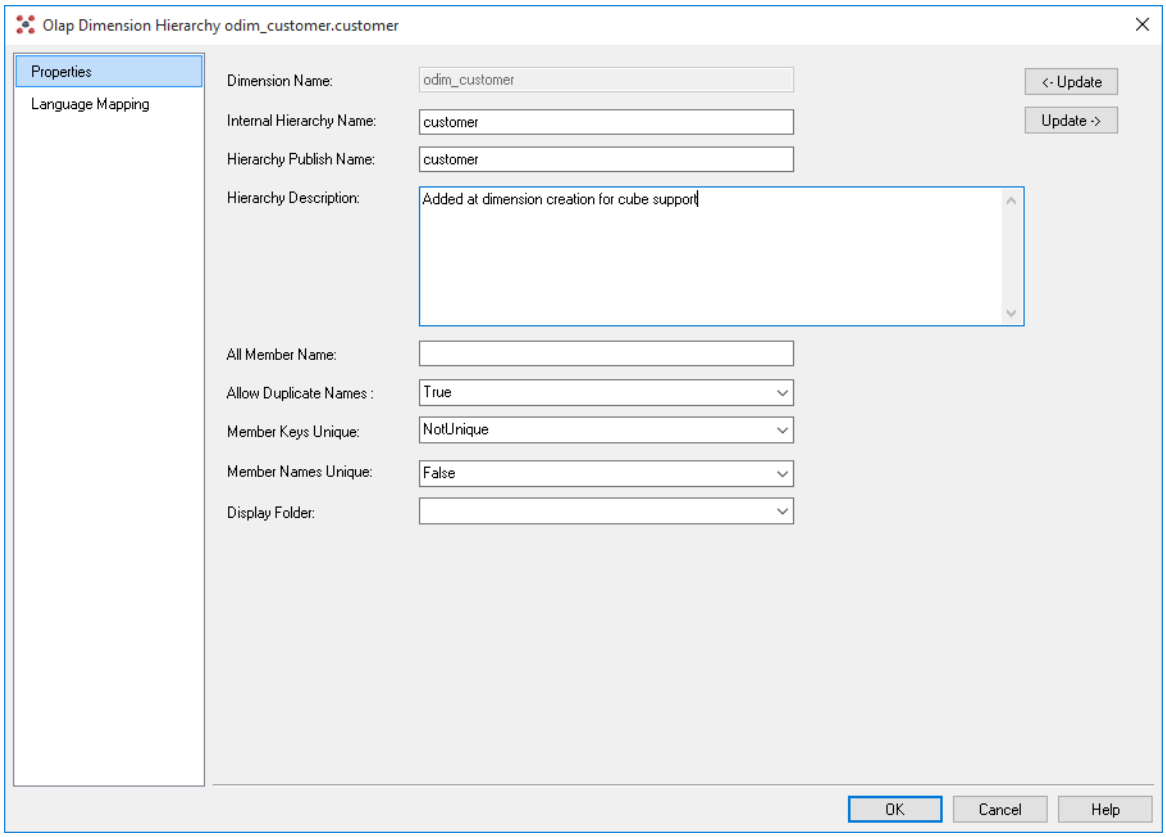
To Delete an Attribute Relationship

Display the Attribute Relationships in the middle pane, select the Attribute Relationship to delete, right-click and select **Delete**.

OLAP DIMENSION HIERARCHIES

User defined hierarchies define hierarchical relationships between related dimensional attributes (e.g. Geographical or time based attributes). These related attributes are defined as levels within the hierarchy.

To view the properties of a user defined Hierarchy right-click on the **Hierarchy** and select **Properties**.



Dimension Name

A read only field indicating the dimension associated with the attribute relationship.

Internal Hierarchy Name

The name of the hierarchy within WhereScape RED.

Hierarchy Publish Name

The name of the hierarchy as created within Analysis Services.

Hierarchy Description

A business name that is stored in WhereScape RED for documentation and stored in the Analysis Services metadata.

All Member Name

Specifies the name of the member in the All level

Allow duplicate Names

Indicates whether the members under a common parent can have the same name.

Member Keys Unique

Indicates whether member keys are unique for this hierarchy.

Note: If you are using a version of Analysis Services earlier than service pack 2, the only value allowed for **Member Keys Unique** is 'NotUnique'. If the value 'Unique' is used, Analysis Services will return an error and the cube will not be created.

Member Names Unique

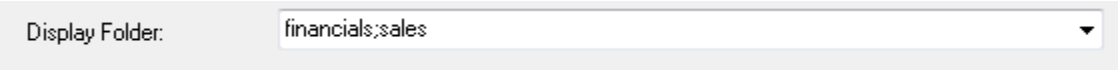
Indicates whether member names are unique for this hierarchy.

Display Folder

Dimension hierarchies can be organized into user-defined folders to view and manage these attributes within the Analysis Services user interface more easily. Enter the display folder name.

Notes:

1. One object can be in multiple display folders, for example:



2. Dimension Attributes and Dimension Hierarchies have the same display folder drop-down list.

To view User Defined Hierarchies

To view the list of User Defined Hierarchies, right-click on an OLAP Dimension in the left pane and select **Display Hierarchies**.

To Add a User Defined Hierarchy

To add a User Defined Hierarchy, right-click on an OLAP Dimension in the left pane and select **Add Hierarchy**. Fill out the dialog box with the relevant details.

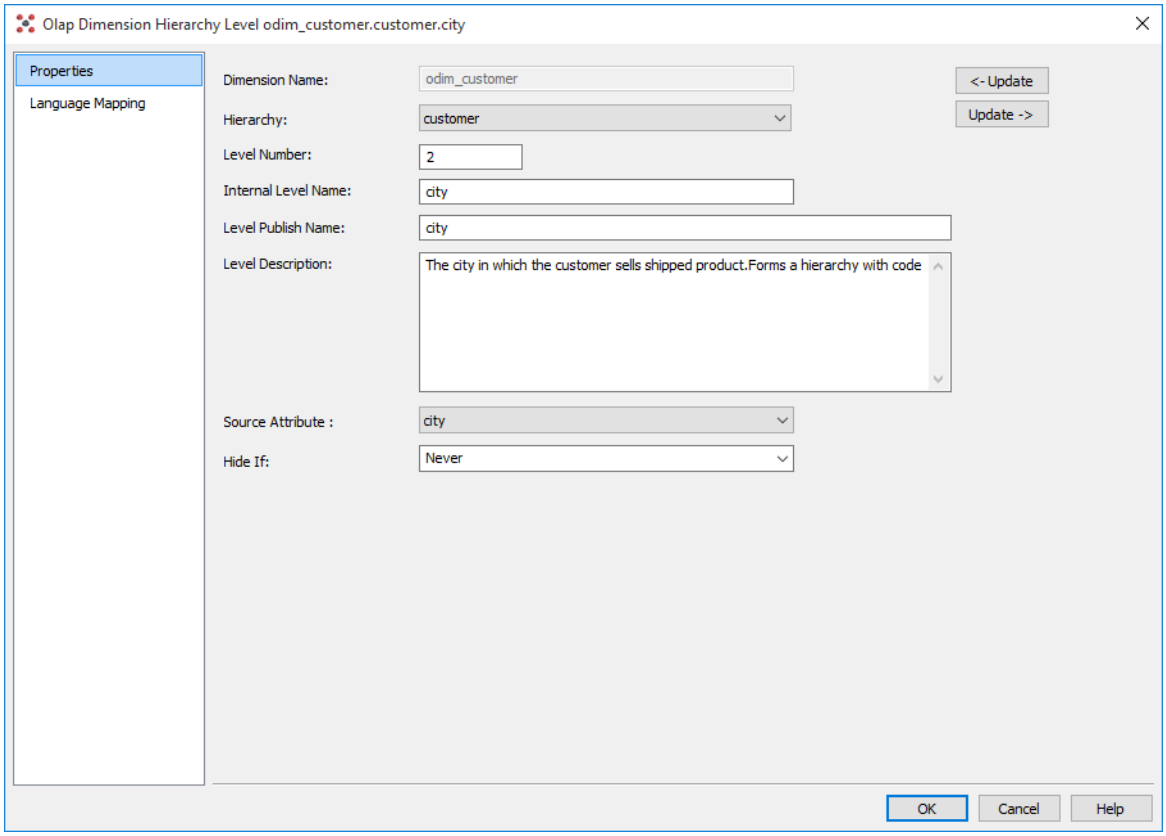
To Delete a User Defined Hierarchy

Display the User Defined Hierarchies in the middle pane, select the User Defined Hierarchy to delete, right-click and select **Delete**.

OLAP DIMENSION USER DEFINED HIERARCHY LEVELS

The levels specify the drill path over a set of related attributes. The classic hierarchy levels are Year, Month, Date in a Calendar based hierarchy in the date dimension.

To view the Properties of a user defined Hierarchy Level right-click on a user defined Hierarchy Level and select **Properties**.



Dimension Name

A read only field indicating the dimension associated with the attribute relationship.

Hierarchy

The User Defined Hierarchy that contains the level.

Level Number

The number of levels from the top most hierarchy level. These level numbers must start at 1 for the top level and provide continuous numbering to the bottom level.

Internal Level Name

The name of the Level within WhereScape RED.

Level Publish Name

The name of the Level as created within Analysis Services.

Level Description

A business name that is stored in WhereScape RED for documentation and stored in the Analysis Services metadata.

Source Attribute

Specifies the source attribute on which the level is based.

Hide If

Specifies which members are hidden. This property supports ragged hierarchies containing logical gaps between members.

To View User Defined Hierarchy Levels

To view the list of User Defined Hierarchy Levels, right-click on an OLAP Dimension in the left pane and select **Display Hierarchy Levels**.

To Add a User Defined Hierarchy

1. To add a User Defined Hierarchy level, right-click on an OLAP Dimension in the left pane and select **Add Hierarchy Level**. Fill out the dialog box with the relevant details.
2. Alternatively right-click on a User Defined Hierarchy in the middle pane and select **Add Hierarchy Level**. Fill out the dialog box with the relevant details.

To Delete a User Defined Hierarchy Level

Display the User Defined Hierarchy Levels in the middle pane, select the User Defined Hierarchy Level to delete, right-click and select **Delete**.

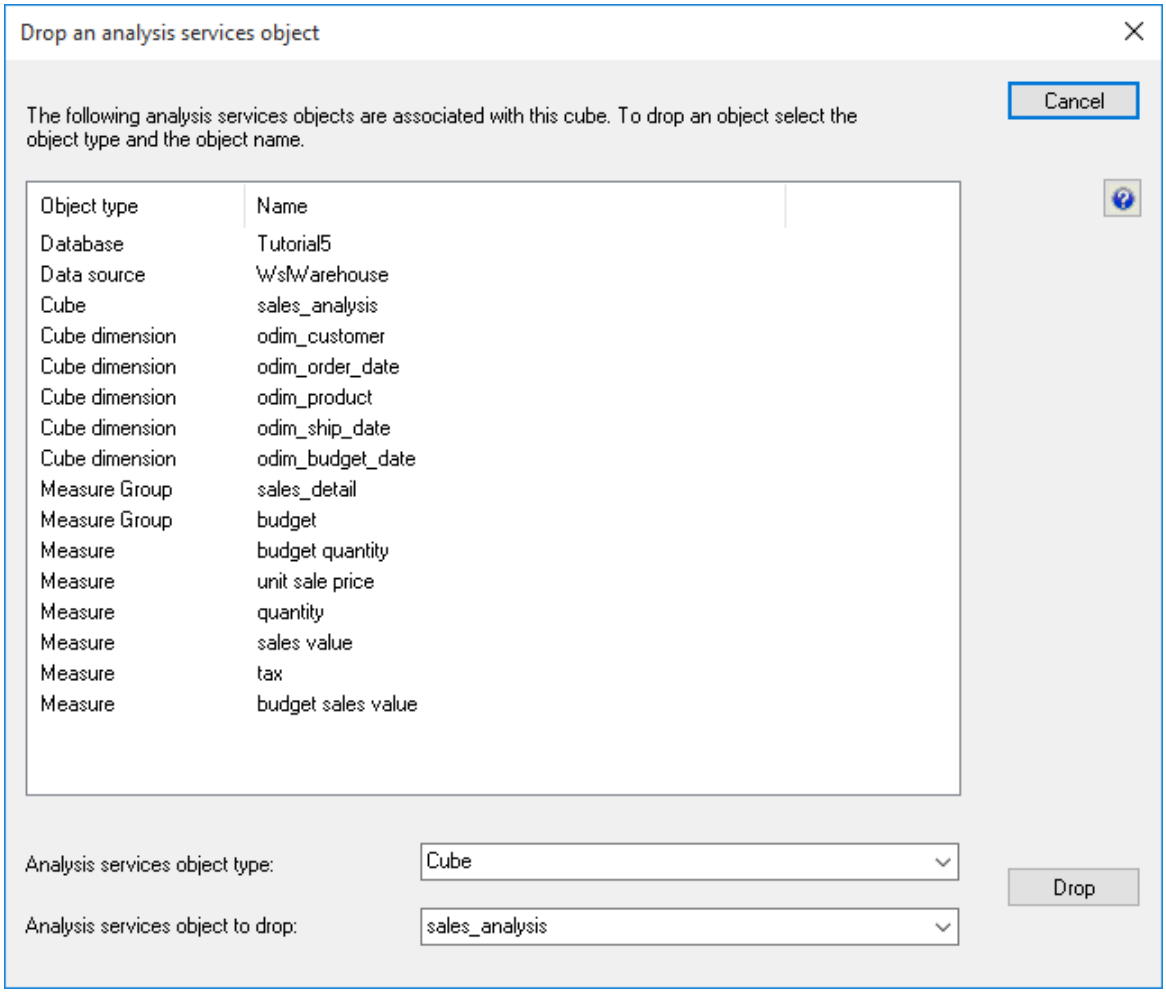
OLAP CHANGING OLAP CUBES

An understanding of the dependency of objects within Analysis Services is the key to figuring out what needs to be dropped or recreated in a cube database using WhereScape RED.

Changes to the underlying relational star schema can cause cube processing to fail as the star schema is frozen in the Data Source View (DSV) of the cube database. Minor changes such as the addition of table columns, or altered data types (e.g. changing a char to varchar) will not break the cube, but renaming a source table or column that is used as a source to the cube will invalidate the DSV and cause processing to fail.

The solution to this issue is to drop and recreate the Cube database from RED to recreate the DSV or manually update the DSV using Microsoft BIDS.

If an object needs to be dropped and recreated in RED, then these are two separate actions. For example, to drop the OLAP database, right-click an OLAP cube within that database in RED and select **Drop Analysis Services Object**, then either double-click on an object in the list or use the drop-down boxes in the Drop Analysis Services Object dialog to choose the object to drop, and click **Drop**. This will drop the object from Analysis Services.



A Create action on an Analysis Services object in RED will be different depending on whether the object already exists in Analysis Services:

- If the object does not already exist in Analysis Services, then RED will create the object (and any related objects e.g. OLAP database and DSV).
- If the object does already exist in Analysis Services, then RED will try to detect any changes or additional features that need to be added to the object and add or alter the existing Analysis Services object.

Some objects need to be dropped and recreated to be changed (eg dimension structures), and some only need to be recreated (e.g. calculations).

Changes to cube runtime objects do not require the cube database to be dropped. For example, a new or changed definition of a calculation or KPI will not require the cube to be dropped and recreated (so data is retained). By Recreating the cube the definition of these runtime objects will be updated and available immediately to cube users.

A summary of the hierarchy of objects and the remedial action is shown below:

Cube Object	Change	Action
Data Source	This changes the source database connection. It is defined in the Data Warehouse connection in RED.	OLAP database needs to be dropped and recreated.
Data Source View (underlying relational star)	The DSV reflects the design of the relational star. So the DSV would need to be updated if any changes are made to tables or views that are used to build OLAP objects.	Changes to the underlying relational star that affect an existing OLAP Object requires that the OLAP Database is dropped and recreated to regenerate the DSV.
OLAP Dimension	The addition or deletion of attributes or hierarchies to an existing OLAP dimension.	The OLAP dimension plus any OLAP cubes associated with the dimension need to be dropped and recreated.
OLAP Cube Measure Group	Delete or Add a Measure Group based on a fact that already exists in the DSV.	Recreate the cube in RED and reprocess.
OLAP Cube Measure Group	Add a Measure Group based on a fact that does not exist in the DSV.	Recreate the OLAP cube database and reprocess.
OLAP Cube Measures	Delete or Add measures based on columns that already exist in the DSV.	Recreate the cube in RED and reprocess.
OLAP Cube Measures	Add measures that are based on new columns that do not exist in the DSV.	Recreate the OLAP cube database and reprocess.
OLAP Cube Calculations, KPIs, Actions	Add, change or delete definition on the cube.	Recreate the cube in RED (a reprocess is not necessary because just the calculation definition is stored in the cube - the result is calculated at query time).

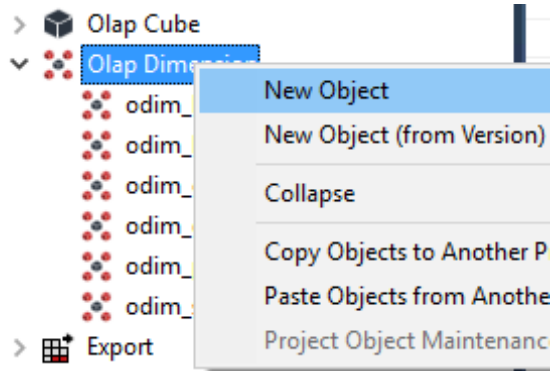
OLAP RETROFITTING AN OLAP OBJECT

WhereScape RED provides the functionality for retrofitting OLAP cubes from Analysis Services.

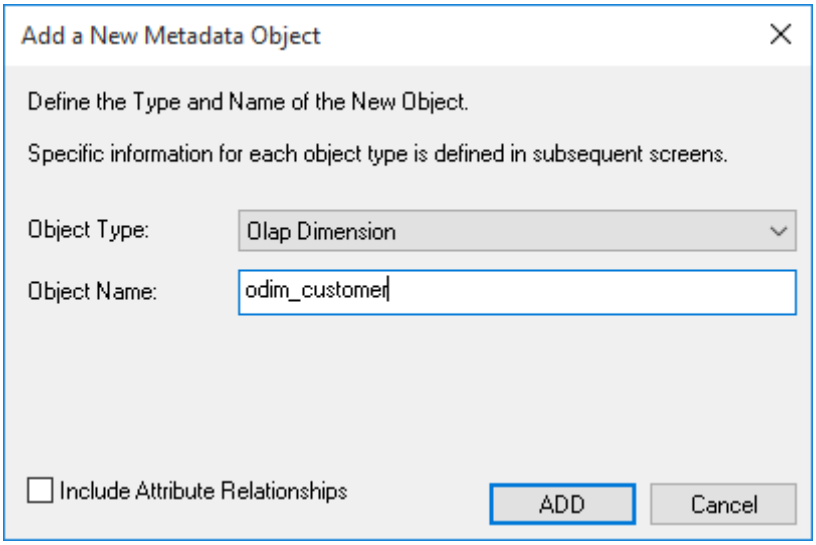
Note: Before you can retrofit an OLAP cube, you must first retrofit any and all of the OLAP dimensions used by the OLAP cube.

The process to retrofit an OLAP dimension is as follows:

1 Right-click on the **OLAP Dimension** object heading in the left pane and select **New Object**.



Enter any name for the object name and click **ADD**.



2 On the Properties dialog:
For the **Internal dimension name** enter the name that matches exactly the name of the OLAP Dimension in Analysis Services that you want to retrofit.

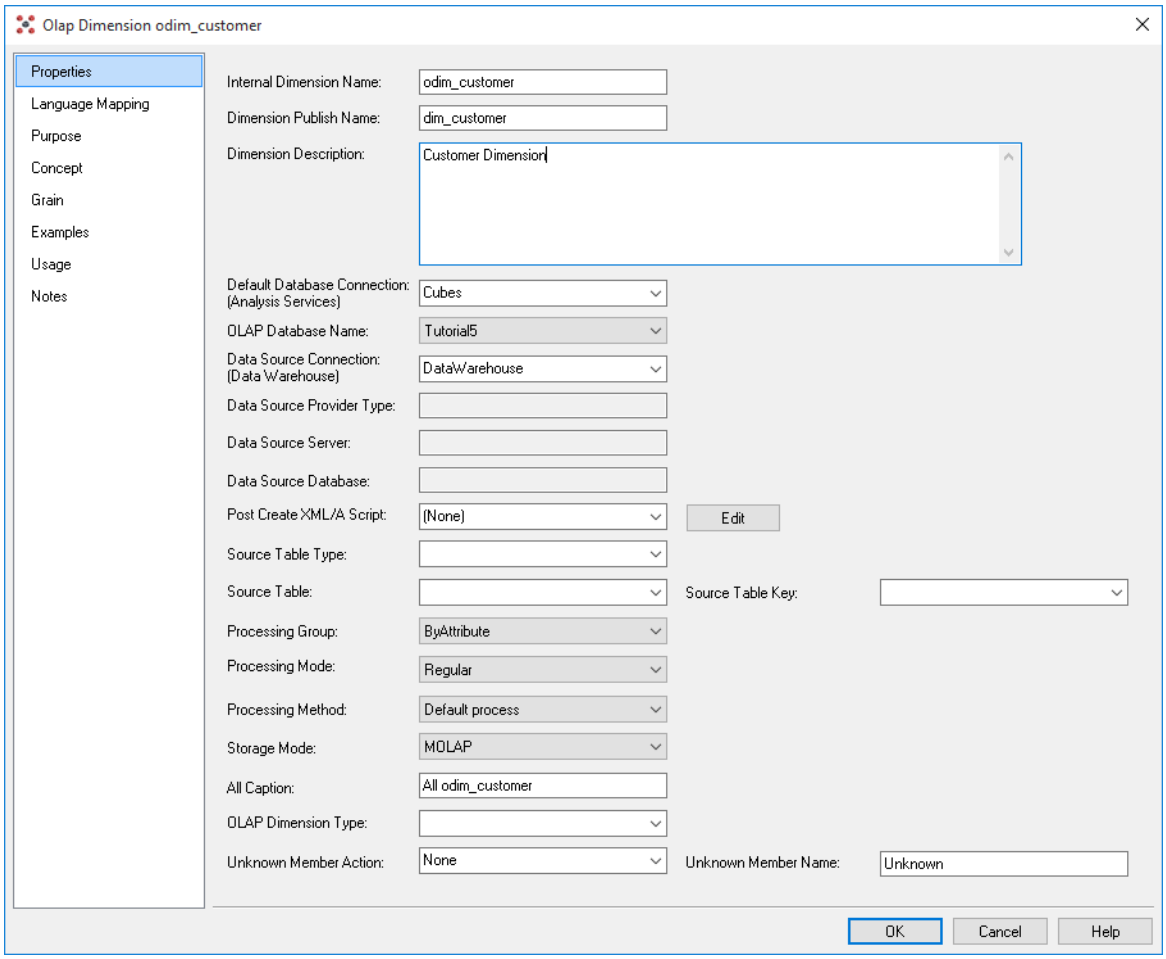
For the **Dimension publish name** enter a name for the dimension.

For the **Dimension description** enter a description for the dimension.

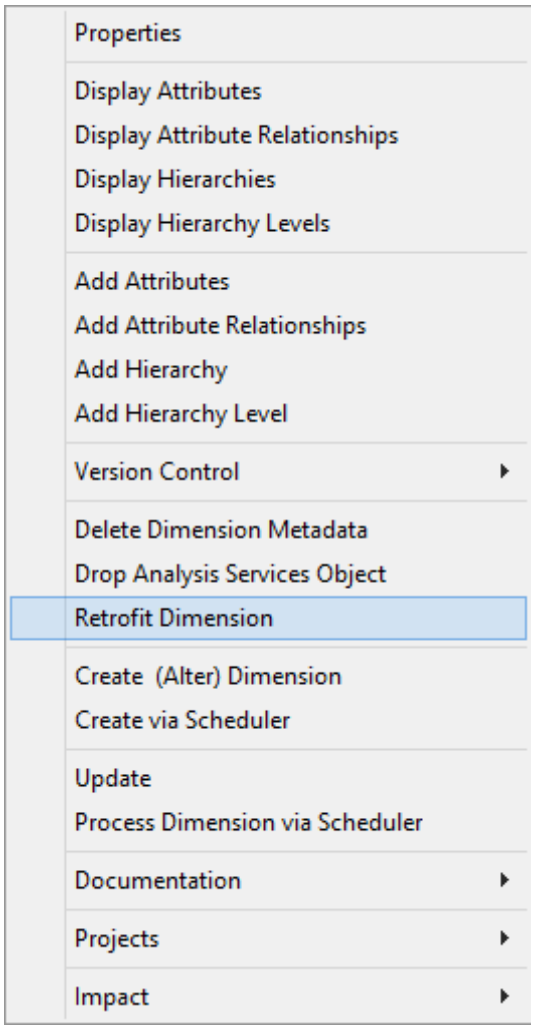
For the **Default database connection** select the required Analysis Services connection.

For the **OLAP database name** select the database name in Analysis Services.

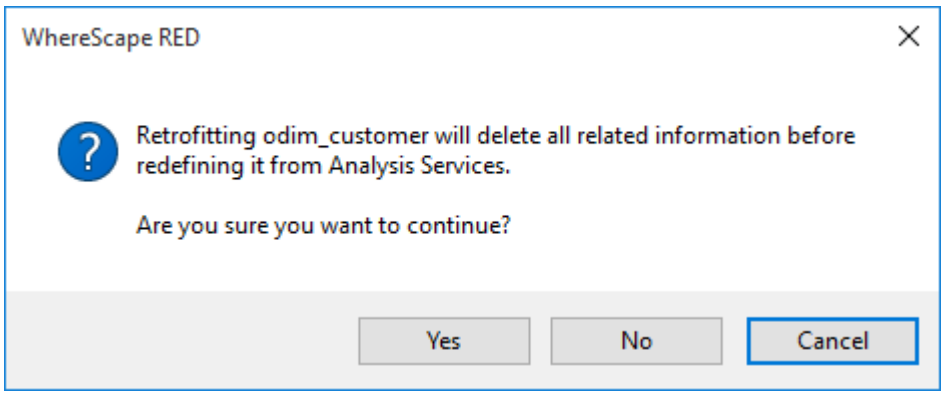
For the **Data source connection** select the relevant data source connection.



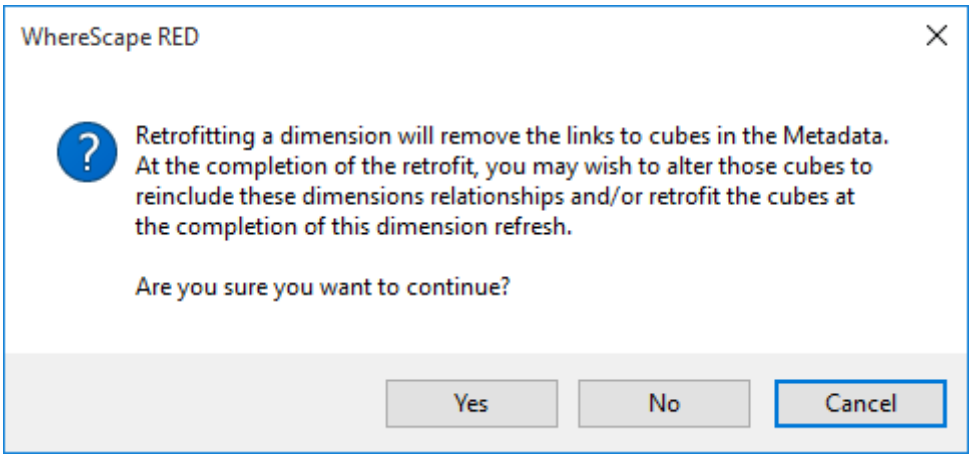
3 The OLAP Dimension can now be retrofitted by right-clicking the object in the left pane and choosing **Retrofit Dimension**.



4 Two warning dialogs now appear. The first dialog warns that the existing information will be deleted before being redefined from Analysis Services. Select **Yes**.



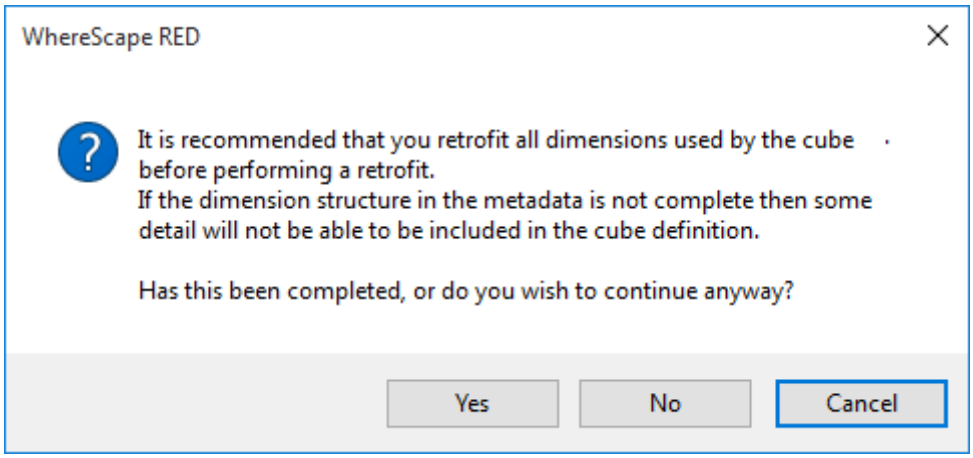
5 The second dialog warns that retrofitting a dimension will remove the links to cubes in the Metadata. Select **Yes**.



6 The results of the retrofit are displayed in the results panel at the bottom of the screen. If it was successful the new object should now have all the attributes, hierarchies and hierarchy levels (as well as all of their properties) as set in Analysis Services.

Once all of the dimensions have been retrofitted you can retrofit the **OLAP Cube**. Follow steps 1-6 above to retrofit the OLAP Cube as for the OLAP dimensions.

The final dialog is a reminder to only retrofit an OLAP cube once the dimensions used by the cube have been retrofitted. If all relevant dimensions have been retrofitted, select **Yes**.



Once again, the results of the retrofit are displayed in the results panel at the bottom of the screen.

TABULAR MODELS

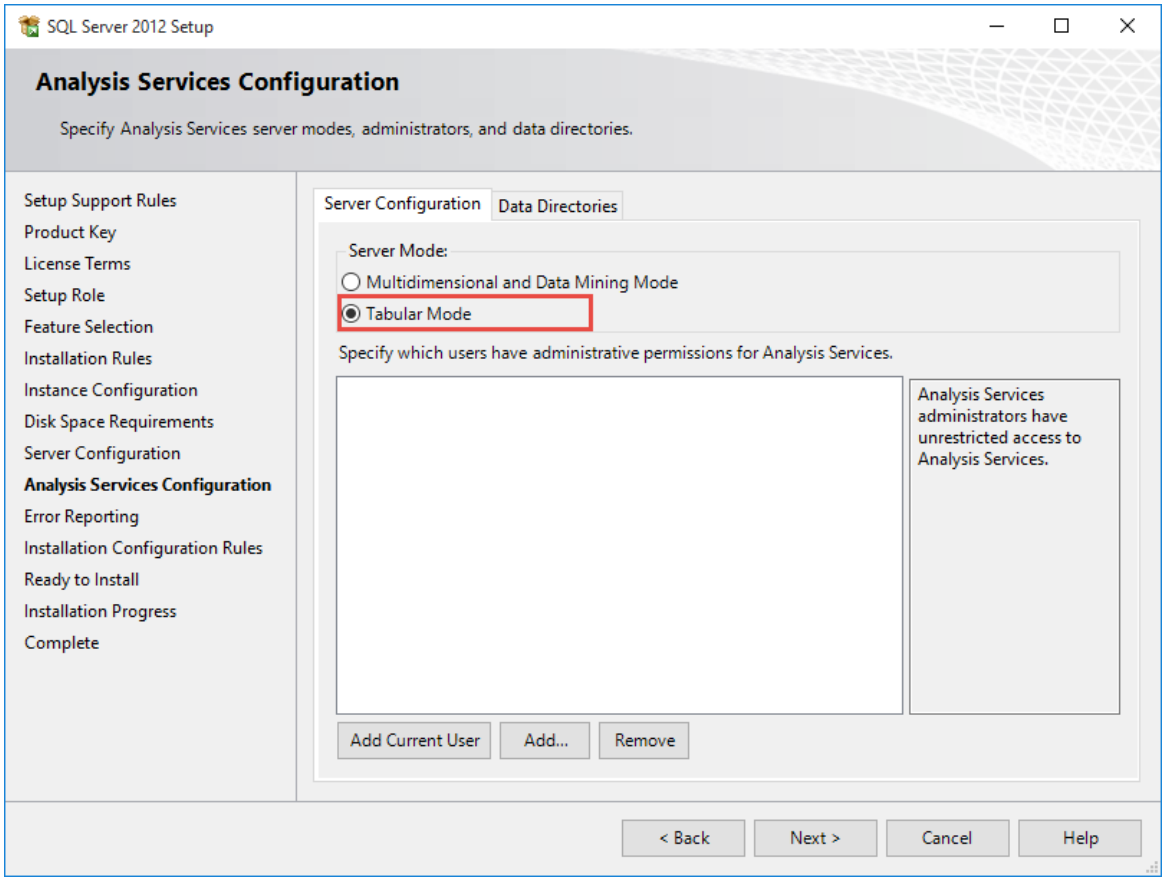
WhereScape RED MSAS's Tabular Mode functionality is enabled for SQL Server, Oracle, Teradata and DB2 repositories, available only for Dimension, Fact and EDW3NF objects.
To use this RED functionality, users will need to ensure they have a Microsoft Analysis Services Server running in Tabular mode available.

The following sections provide an overview of the Tabular functionality in RED, detailing the following:

- Ensuring the installation of a Microsoft Analysis Services Server is running in Tabular mode
- The creation of a new Tabular connection with associated tabular targets
- Setup of an MSAS connection string in the DataWarehouse connection
- Creating Fact, Dimension or EDW3NF tables by Dragging and Dropping, then choosing a Tabular Target Location.
- Querying the results via Excel

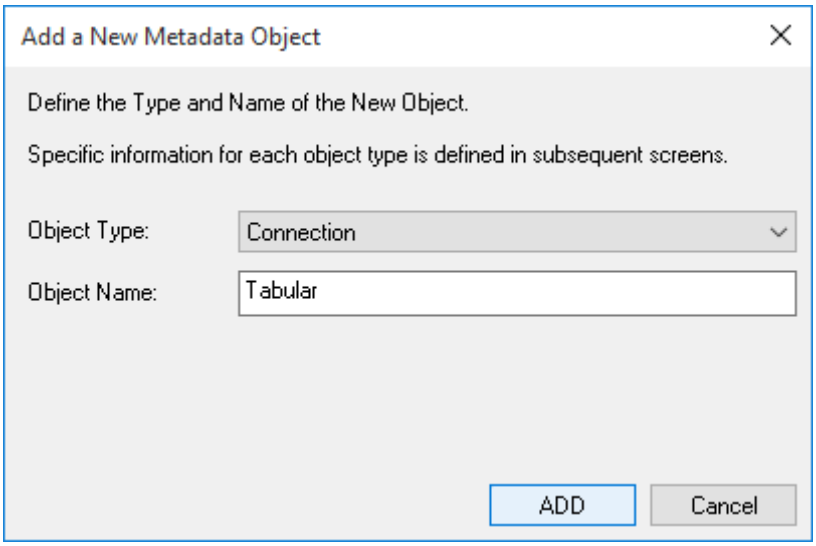
DEFINING THE DATA SOURCE AND TARGETS FOR TABULAR MODELS

1 To use the MSAS Tabular mode functionality in RED, start by following your SQL Server setup's Installation to "New SQL Server stand-alone installation or add features to an existing installation" option to ensure that your Analysis Server Configuration has been installed for **Tabular Mode**.



2 In RED, create a new Tabular **Connection**.

- Give your connection a relevant name for your system. In this example, we have named the connection "Tabular".
- Click **Add**.

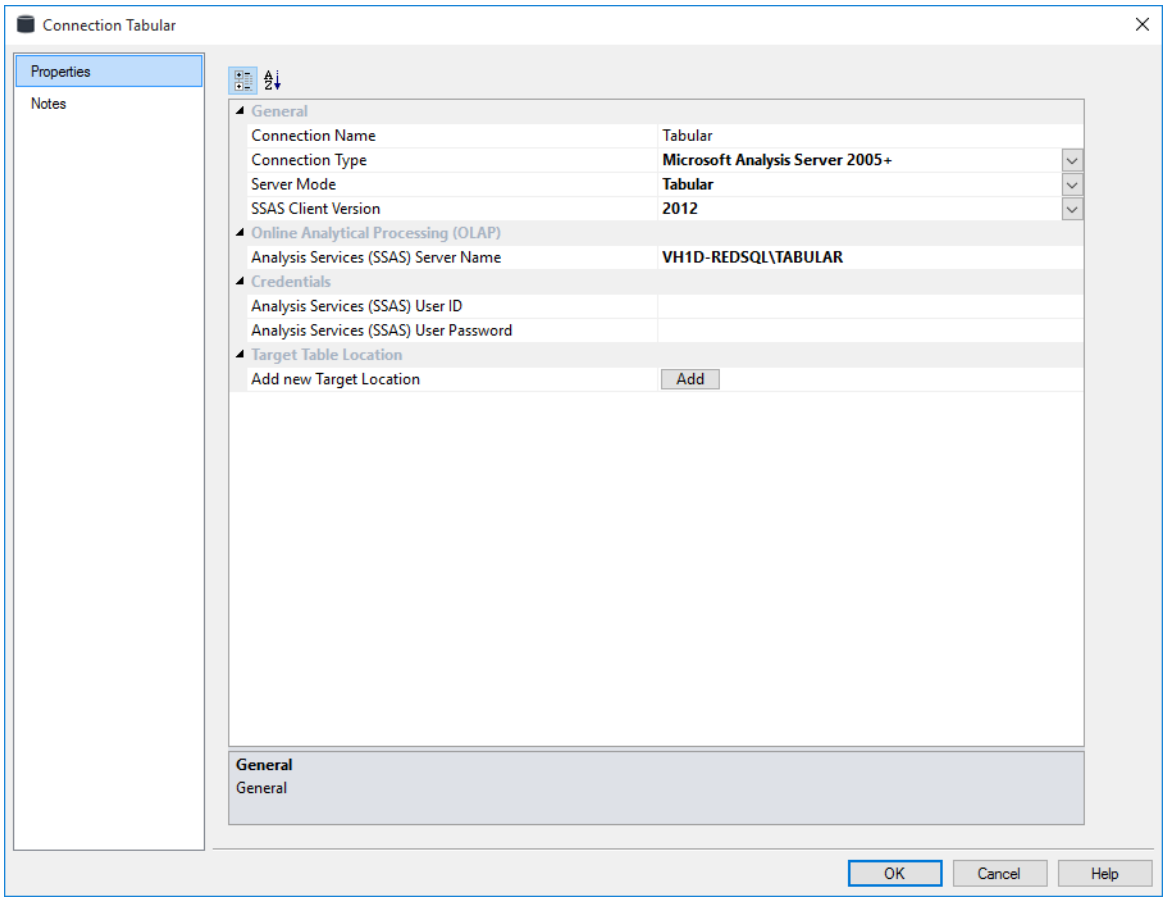


3 Select "Microsoft Analysis Server 2005+" for your **Connection Type**.

- Select "Tabular" for your **Server Mode**.
- Select the **SSAS Client Version** to connect for your SSAS database.

Please note it is recommended that the client version matches the database version.

- For the **Analysis Services (SSAS) Server Name**, enter the same server name used when installing Microsoft Server Analysis Services in Tabular Mode.



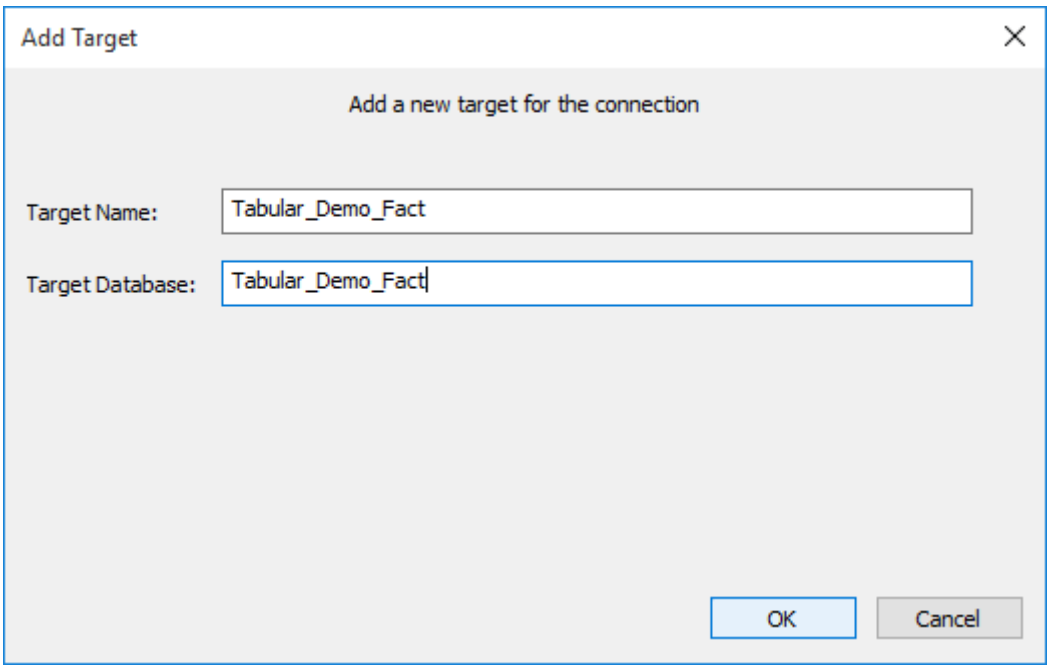
Notes:

If the required SQL Server Analysis Management Objects (AMO) are missing, see the following article for more information:
https://msdn.microsoft.com/en-us/library/dn141152.aspx
(***https://msdn.microsoft.com/en-us/library/dn141152.aspx***)

You may need to specify the port number of the Analysis Services instance. To find your port number, follow the procedure documented in this Microsoft article:
https://support.microsoft.com/en-us/kb/2466860
(***https://support.microsoft.com/en-us/kb/2466860***).
An example of your Analysis Server (SSAS) Server Name using the port number in RED would be: VH1D-REDSQL:49449\TABULAR

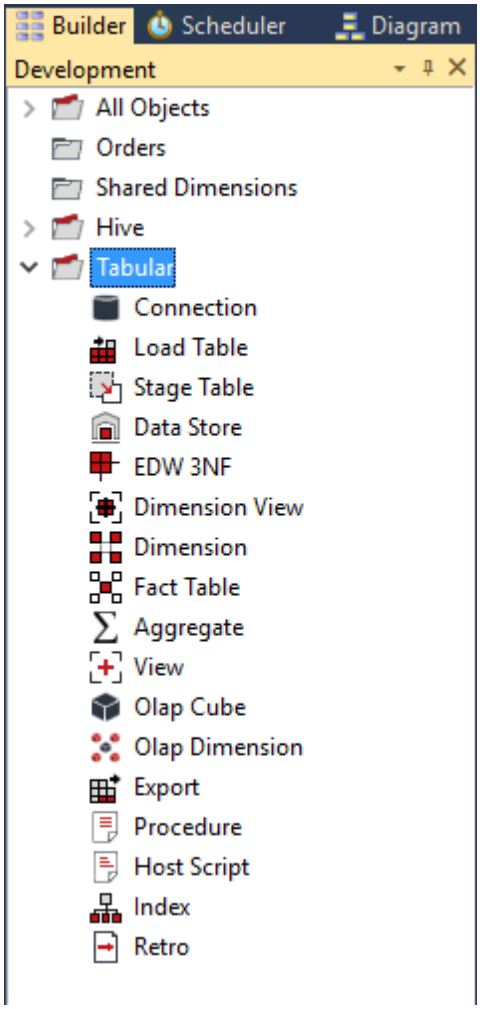
4 Create your Tabular targets by clicking the **Add** button and entering a name for the Target and the Target Database.

- The **Target Name** will be the relevant Tabular Database's name displayed in RED.
- The **Target Database** will be the relevant Tabular Database's name displayed in Analysis Services.



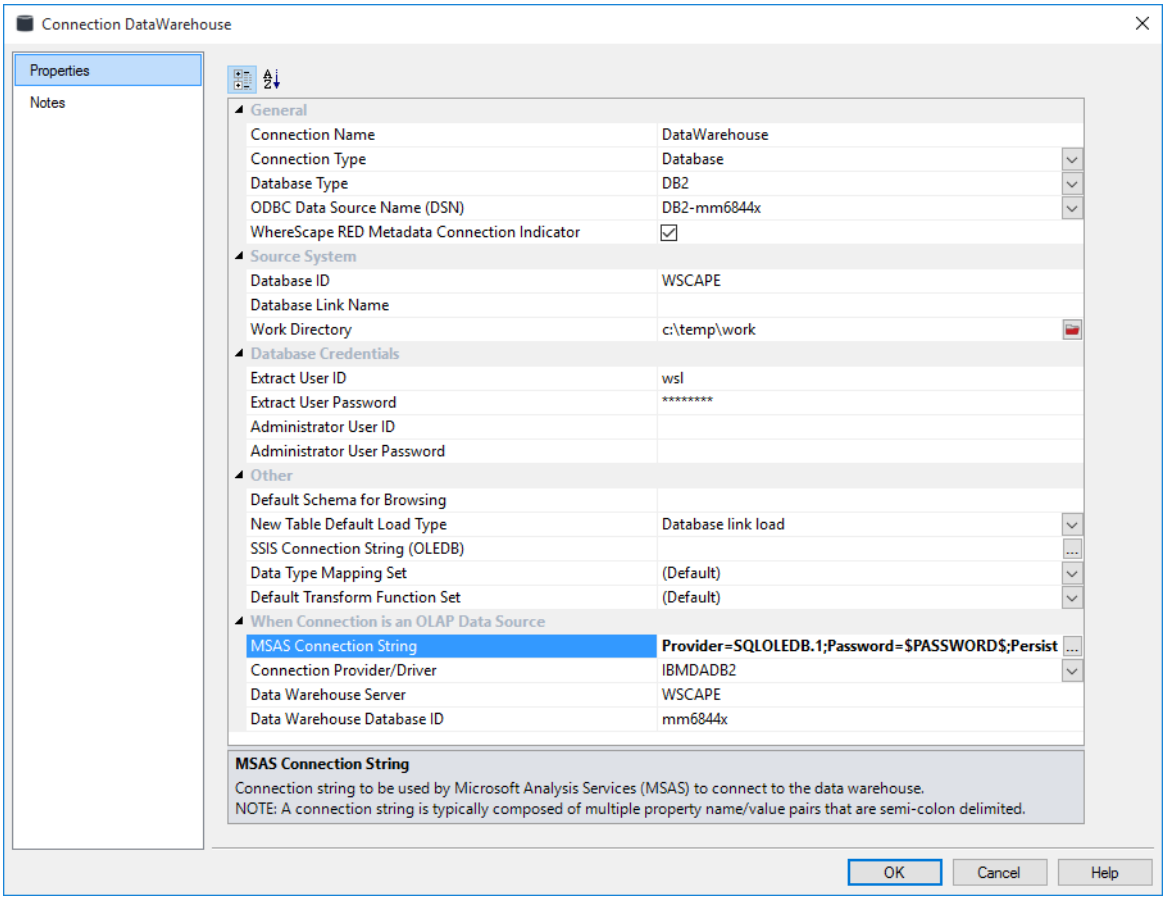
5 For the ease of exploring the new Tabular functionality in RED and differentiate it from the existent database tables, we have created another Project and named it "Tabular".

NOTE: To show unused object types in the browser pane enable **Project Properties/Unused Object Types in Object Browser**.



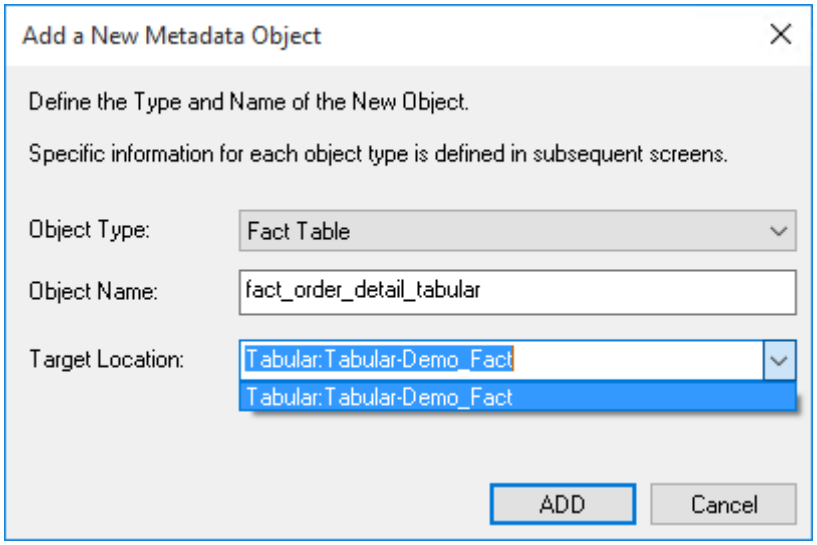
6 Click on your **Datawarehouse** connection and ensure you have set your MSAS Connection String and other relevant Analysis Server required fields.

If you have already set the MSAS Connection String for use with Multidimensional (OLAP) Cubes, you should not need to change it.
For more information about setting up this connection string, refer to ***Defining the Data Source for OLAP Cube*** (see "***OLAP Defining the Data Source for the OLAP Cube***" on page 558).


Connection DataWarehouse
Properties
Notes
General
Connection Name
DataWarehouse
Connection Type
Database
Database Type
DB2
ODBC Data Source Name (DSN)
DB2-mm6844x
WhereScape RED Metadata Connection Indicator
Source System
Database ID
WSCAPE
Database Link Name
Work Directory
c:\temp\work
Database Credentials
Extract User ID
wsl
Extract User Password
****
Administrator User ID
Administrator User Password
Other
Default Schema for Browsing
New Table Default Load Type
Database link load
SSIS Connection String (OLEDB)
Data Type Mapping Set
(Default)
Default Transform Function Set
(Default)
When Connection is an OLAP Data Source
MSAS Connection String
Provider=SQLOLEDB.1;Password=$PASSWORD$;Persist
Connection Provider/Driver
IBMDADB2
Data Warehouse Server
WSCAPE
Data Warehouse Database ID
mm6844x
MSAS Connection String
Connection string to be used by Microsoft Analysis Services (MSAS) to connect to the data warehouse.
NOTE: A connection string is typically composed of multiple property name/value pairs that are semi-colon delimited.
OK
Cancel
Help

DEFINING A TABULAR MODEL

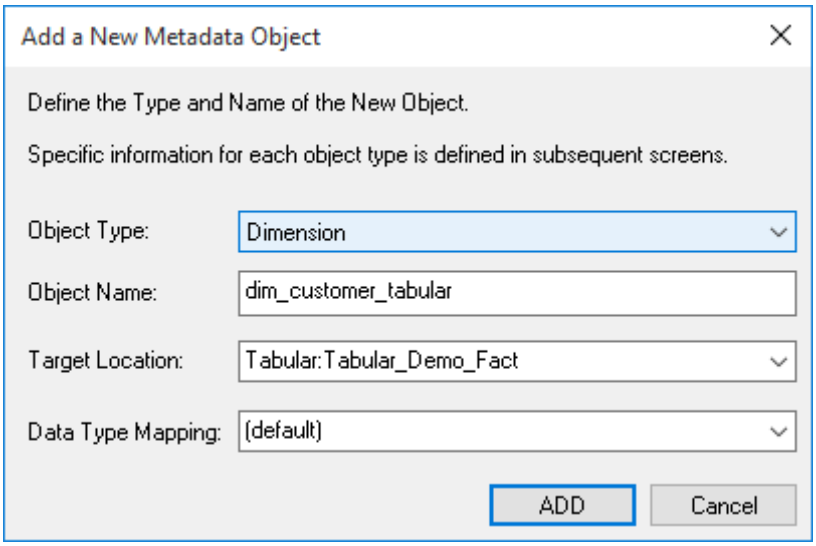
1. Browse your **Datawarehouse** connection.
2. Within your Tabular Project, click on the **Fact Table** object group on the left pane to set your work pane to be your Fact Tables.
3. Then from the right pane, drag a **Fact Table** into the middle pane, rename it with a tabular or relevant reference and select the Tabular target you want to place the table in.
   - We have renamed this table to "fact_order_detail_tabular" and selected the "Tabular_Demo_Fact target" to place it in.



4. Click **OK** on the Fact Table's Properties screen.
5. Right click on the table **fact_order_detail_tabular** from the left pane and select **Create(Recreate)**.
6. After the table has been recreated, right click again and select **Update** to update the table.
7. Browse your Datawarehouse connection again, set your work pane to be Dimension tables and drag your Customer Dimension from the right pane into the middle pane Dimension group.

8 Change the **Object Type** from Dimension View to **Dimension** and give the table a relevant name.

- Select a tabular target location to place the table.



9 Click **OK** on the Properties screen.

10 Right click on the dim_customer_tabular table from the left pane and select **Create(Recreate)**.

11 After the table has been recreated, right-click again and select **Update** to update the table.

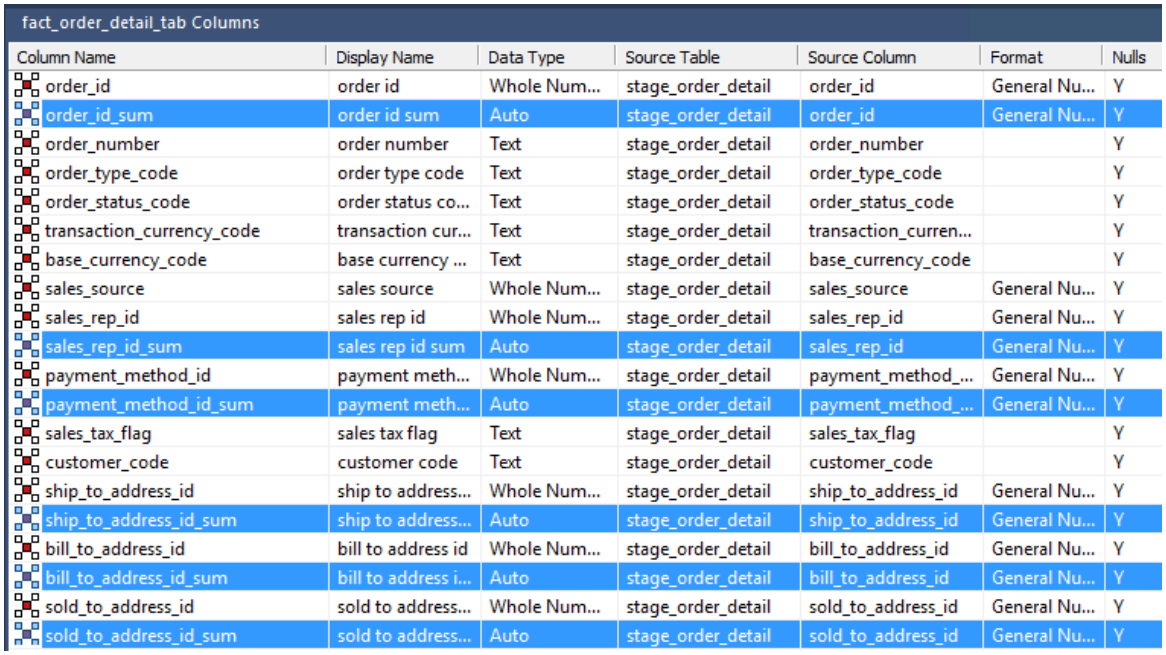
12 Repeat steps 7 to 11 for all other dimension tables joined to your tabular fact table.

13 Right-click the **fact_order_detail_tabular** and select **Query Via Excel** to view the table in Excel.

NOTE: RED operations such as Validate Against Database, Generate Statistics, Table Row Count and Regenerate Indexes are currently not supported with Tabular mode.

TABULAR MODEL MEASURES

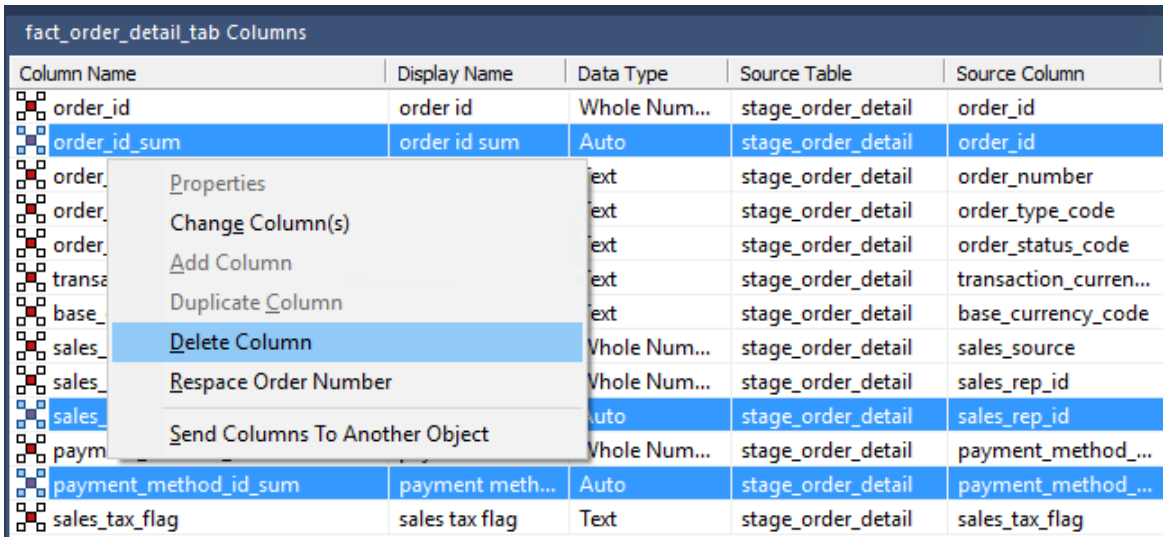
1 When a new Tabular fact table is created, all numeric columns that are market as additive (apart from key columns) will have auto-created measure totals.



fact_order_detail_tab Columns

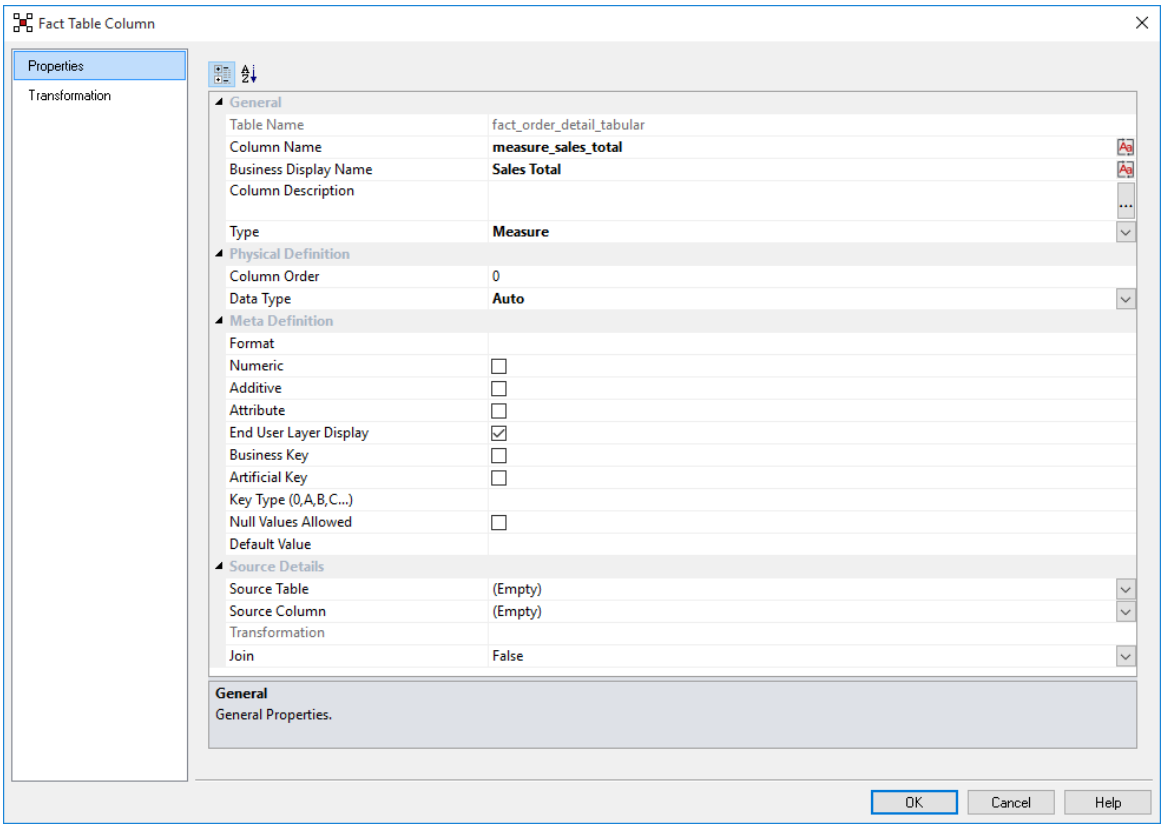
Column Name	Display Name	Data Type	Source Table	Source Column	Format	Nulls
order_id	order id	Whole Num...	stage_order_detail	order_id	General Nu...	Y
order_id_sum	order id sum	Auto	stage_order_detail	order_id	General Nu...	Y
order_number	order number	Text	stage_order_detail	order_number		Y
order_type_code	order type code	Text	stage_order_detail	order_type_code		Y
order_status_code	order status co...	Text	stage_order_detail	order_status_code		Y
transaction_currency_code	transaction cur...	Text	stage_order_detail	transaction_curren...		Y
base_currency_code	base currency ...	Text	stage_order_detail	base_currency_code		Y
sales_source	sales source	Whole Num...	stage_order_detail	sales_source	General Nu...	Y
sales_rep_id	sales rep id	Whole Num...	stage_order_detail	sales_rep_id	General Nu...	Y
sales_rep_id_sum	sales rep id sum	Auto	stage_order_detail	sales_rep_id	General Nu...	Y
payment_method_id	payment meth...	Whole Num...	stage_order_detail	payment_method_...	General Nu...	Y
payment_method_id_sum	payment meth...	Auto	stage_order_detail	payment_method_...	General Nu...	Y
sales_tax_flag	sales tax flag	Text	stage_order_detail	sales_tax_flag		Y
customer_code	customer code	Text	stage_order_detail	customer_code		Y
ship_to_address_id	ship to address...	Whole Num...	stage_order_detail	ship_to_address_id	General Nu...	Y
ship_to_address_id_sum	ship to address...	Auto	stage_order_detail	ship_to_address_id	General Nu...	Y
bill_to_address_id	bill to address id	Whole Num...	stage_order_detail	bill_to_address_id	General Nu...	Y
bill_to_address_id_sum	bill to address i...	Auto	stage_order_detail	bill_to_address_id	General Nu...	Y
sold_to_address_id	sold to address...	Whole Num...	stage_order_detail	sold_to_address_id	General Nu...	Y
sold_to_address_id_sum	sold to address...	Auto	stage_order_detail	sold_to_address_id	General Nu...	Y

2 If the created columns are not relevant for the table, they can be deleted by using the right-click option to **Delete Column**.

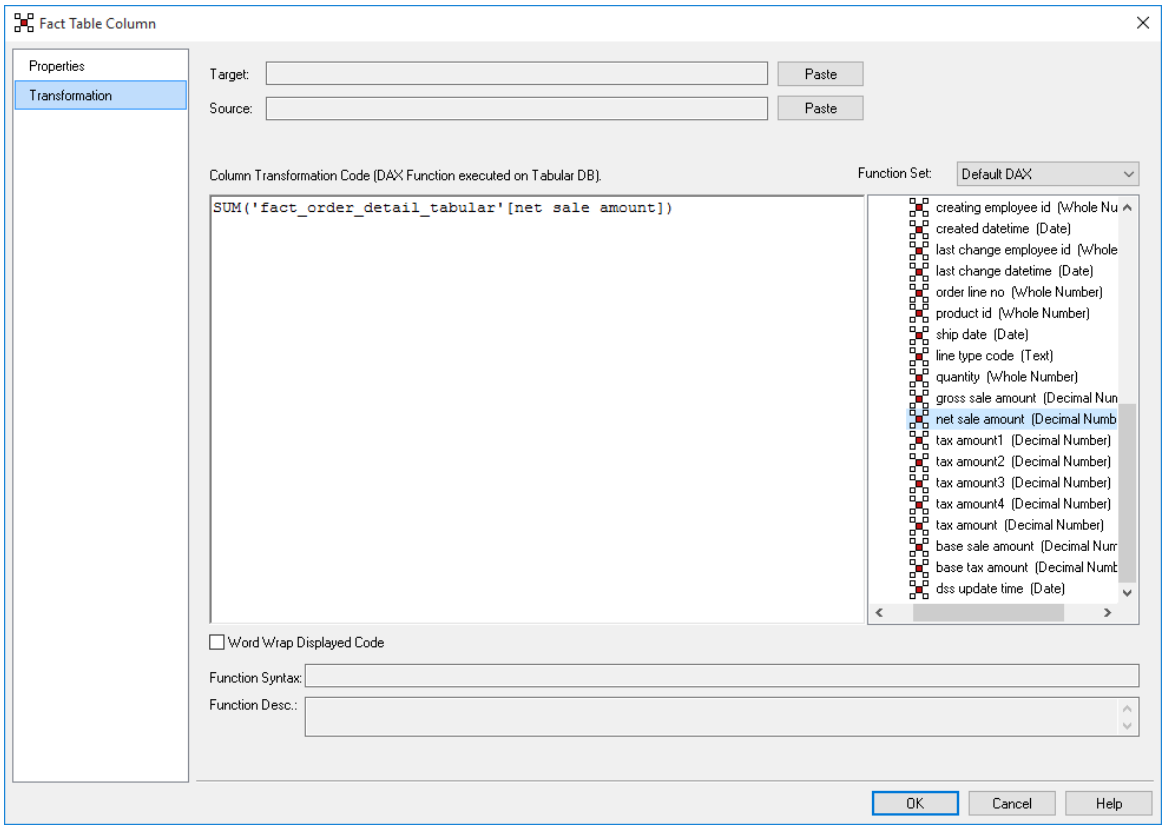


3 To add new **measures** to your Tabular Fact table, right-click on a column in the middle pane and click **Add Column**. In the table column properties, you can set the following fields:

- **Column name:** "measure_sales_total"
- **Business Display Name:** "Sales Total"
- **Type:** Measure



4 Click the **Transformation** tab, add **SUM** to the Column Transformation Code and then select a sales column from the list. Click **OK**.



5 Right click on the table **fact_order_detail_tabular** from the left pane and select **Create(Recreate)**.

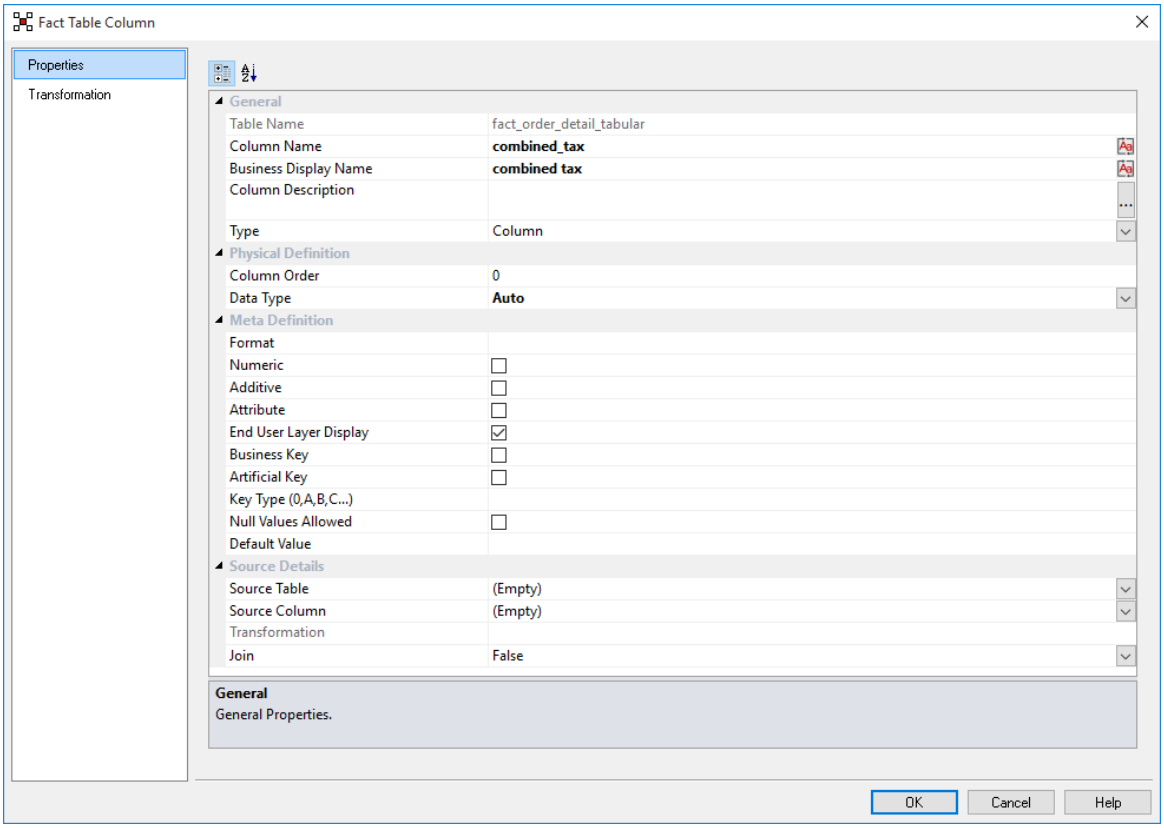
6 After the table has been recreated, right click again and select **Update** to update the table.

TABULAR MODEL CALCULATED COLUMNS

To add Calculated Columns to a Tabular Model:

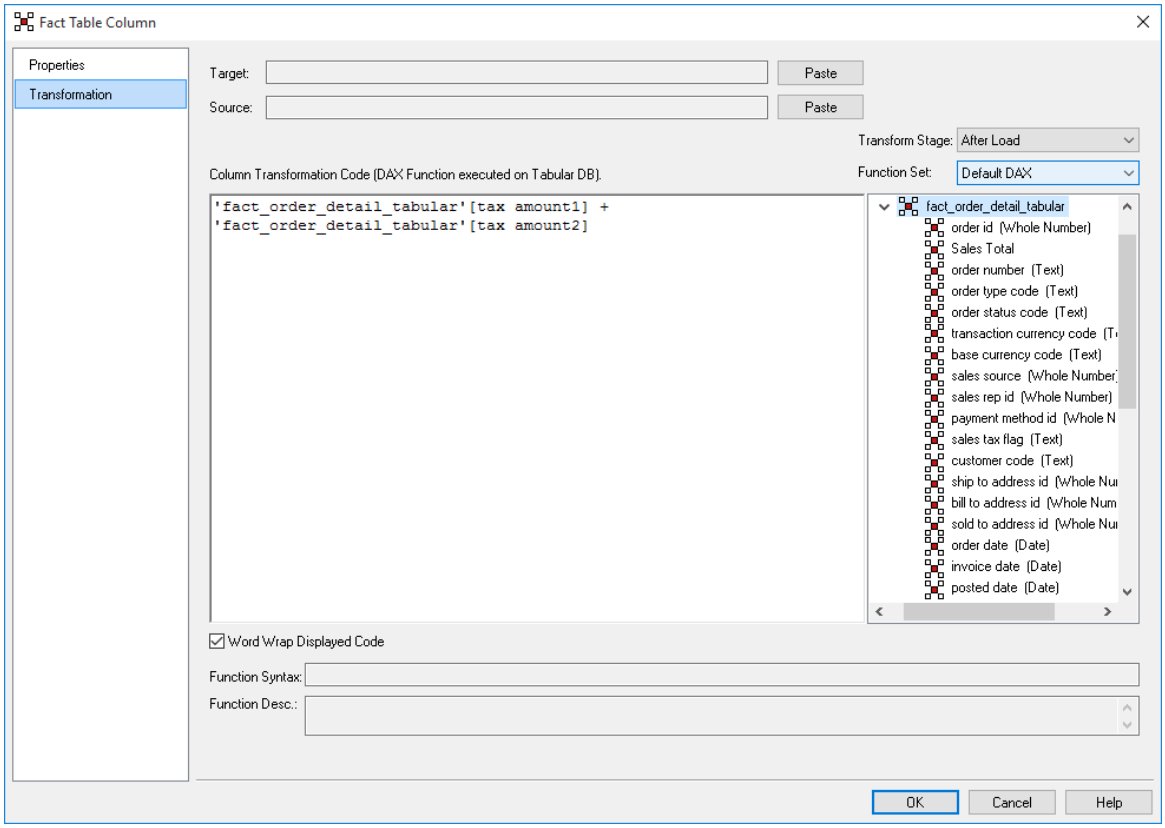
1 Go to to the Fact Table's relevant column properties' screen to set the following fields:

- **Column name:** "combined_tax"
- **Business Display Name:** "Combined Tax"
- **Type:** Column



2 Click the **Transformation** tab and select the following:

- **After Load** as as the Transformation Stage
- **Default DAX** as the Function Set
- Add a **SUM** and select the tax amount columns from the list of available columns on the right
- Click **OK** on the fact column transformation screen



Microsoft Analysis Services 2005+ Tabular Mode Tables: For Tabular Mode table column transformations, **Default DAX** is the only applicable Function Set for **after load** transformations.

3 Right click on the table **fact_order_detail_tabular** from the left pane and select **Create(Recreate)**.

4 After the table has been recreated, right click again and select **Update** to update the table.

WhereScape®

CHAPTER 19
EXPORTING DATA

Export objects are used in WhereScape RED to produce ascii files from a single database table or view for a downstream feed. Some or all of the columns in a table or view can be exported. There are three ways of performing exports in RED:

- **File export** - an export where most of the processing is managed and controlled by the scheduler
- **Script-based export** - an export where a Windows script file is executed to perform the load.
- **Integration Services Export** - an export processed using a Windows connection where the processing is handled via an Integration Services Package that is generated and executed dynamically at run time.
  SSIS exports to UNIX/Linux connections and processed via the UNIX/Linux scheduler are currently not supported.

IN THIS CHAPTER

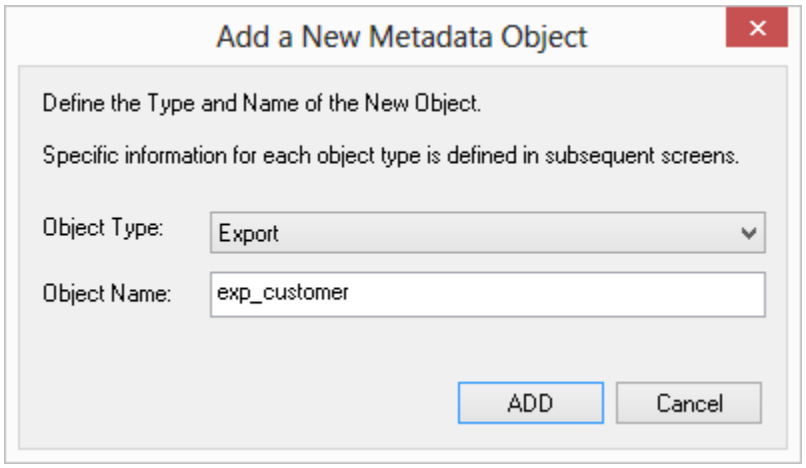
BUILDING AN EXPORT OBJECT

TIP1: The relevant client software for each database type must be installed on each machine you want to export to.
TIP2: For **Integration Services** exports, ensure the **SSIS Version** is selected in **Tools>Options>Code Generation>SSIS Version.**

Greenplum **File Exports** are the only export type currently supported in Greenplum from both Windows and UNIX/Linux connections. Script-based exports are currently unsupported. To process Greenplum exports via the Windows Scheduler, please ensure that the **Windows Scheduler user** has the all the necessary permissions including having the **ssh host key** saved.

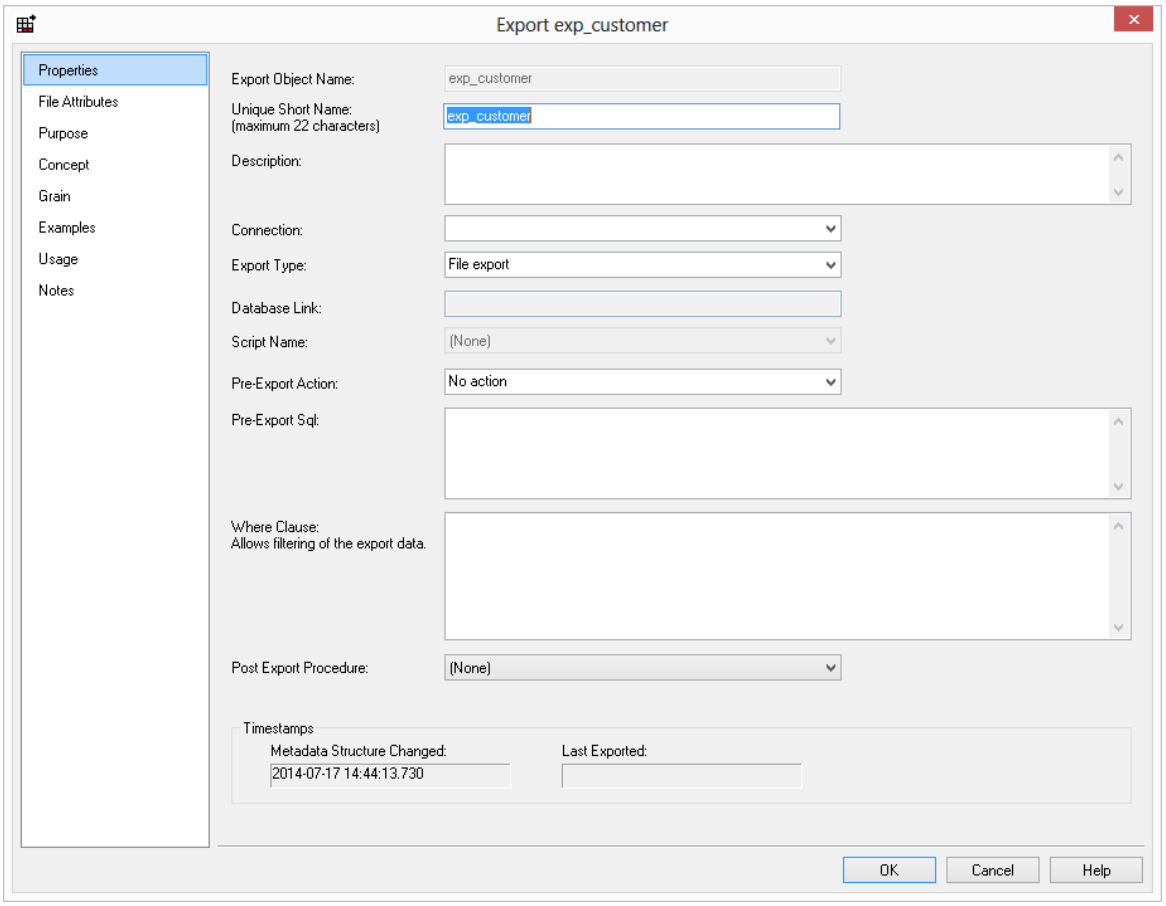
The simplest way to create an export object is to use the **drag and drop** functionality of WhereScape RED.

1. Browse to the data warehouse connection (Browse/Source Tables).
2. Create a drop target by double-clicking on the **Export object** group in the left pane. The middle pane should have a column heading of **Export Objects** for the leftmost column.
3. Select a table or view in the right pane and drag it into the middle pane. Drop the table or view anywhere in the middle pane. The following dialog appears:



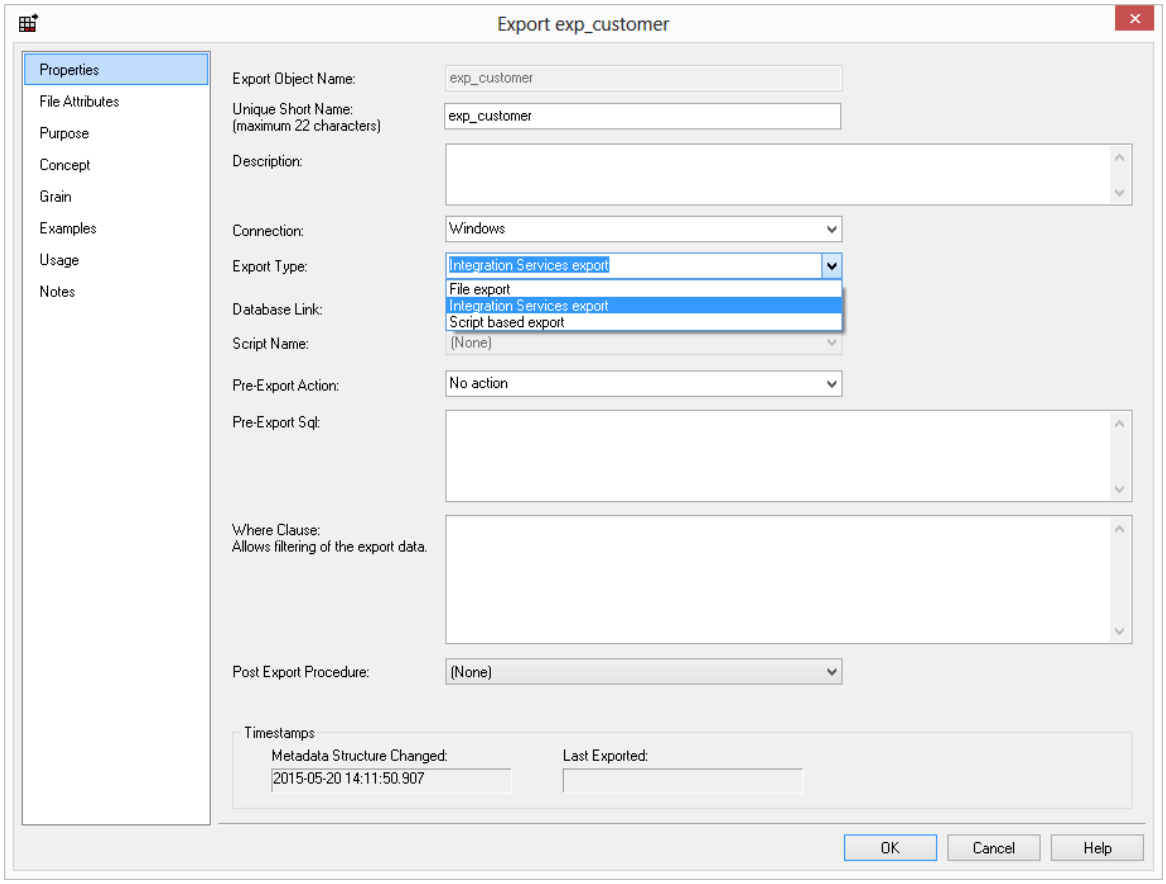
4 If the export object needs to be renamed, rename it and then click the **ADD** button.

5 The following dialog appears.



6 Select the **Connection** that you want to perform the export, from the Connection drop-down list. In this example, the Connection is **Windows**.

- Select the preferred Export type: **File export**, **Script-based export**
- **Integration Services Export** is also an available export type when exporting objects using a **Windows** connection.



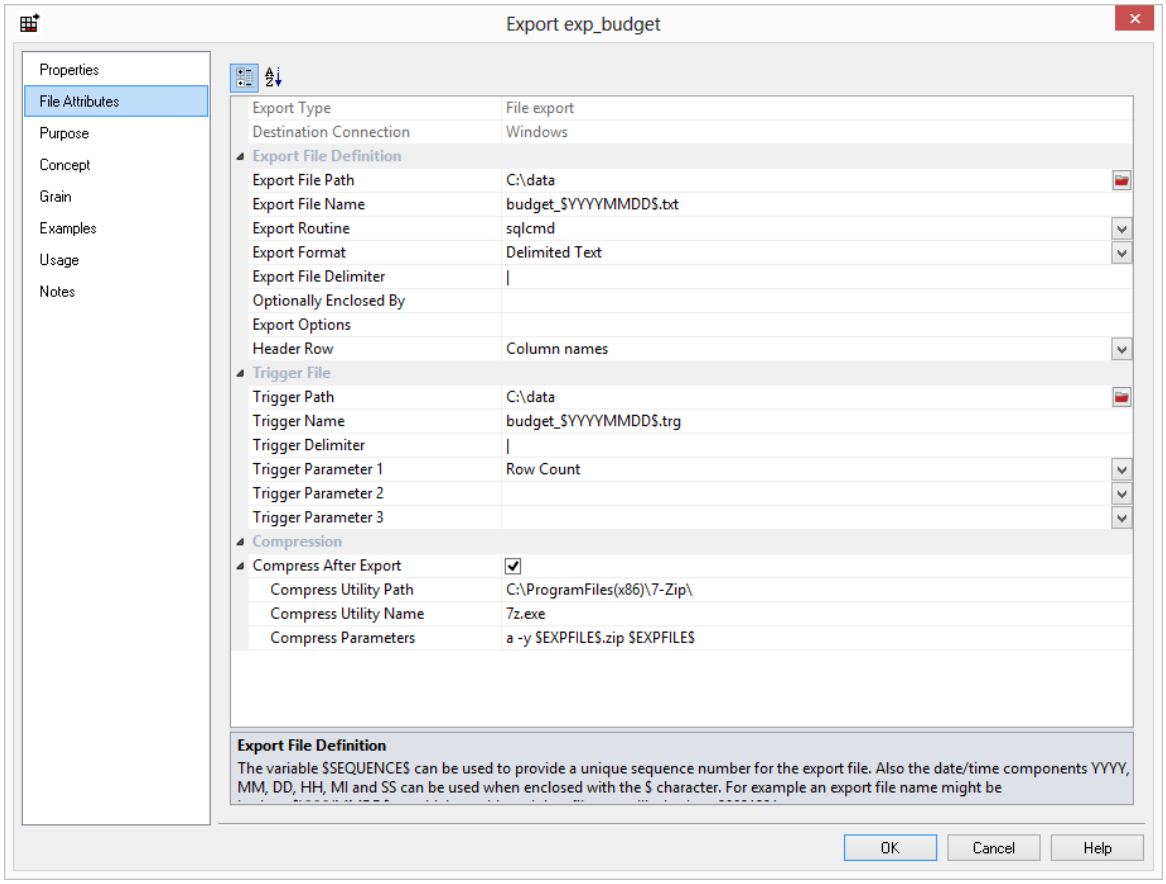
7 Click on the **File Attributes** tab and fill in the fields as described in the next section.

8 Finally, run the export by right-clicking on it and selecting **Export**.

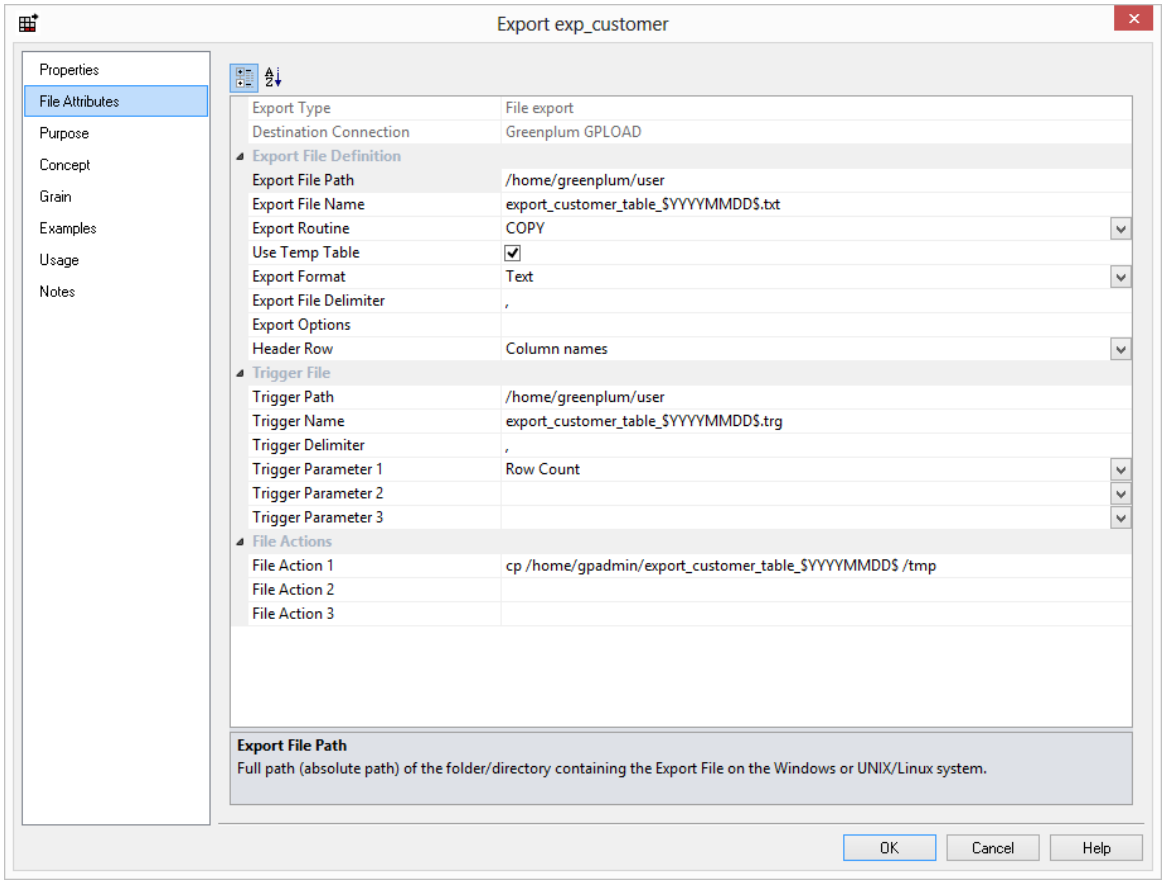
FILE ATTRIBUTES

The following fields are available to define the location, name and contents of the exported data file:

SQL Server File Attributes example screen:



Greenplum File Attributes example screen:



Export Type

Method of exporting data from the table. The available options are dependent on the Destination connection that can be specified via the Properties page.

Destination Connection

Destination to the file system to which data will be exported. The destination connection can be specified on the Properties screen.

Export File Definition

Export File Path

The full path (absolute path) of the folder/directory where the File is to be created on the Windows or UNIX/Linux system.

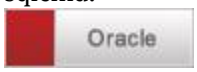
Export File Name

Name of the Export File to which the data will be exported. The variable $SEQUENCE$ can be used to provide a unique sequence number for the export file. Also the data/file components YYYY, MM, HH, MI, SS can be used when enclosed with the $ character.For example an export file name might be budget_$YYYYMMDD$.txt which would result in a file name like budget_20150520.txt.

Export Routine

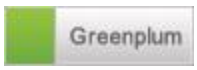
Database-specific routine to use to export the data.

In SQL Server, this is BCP or isql. In later versions of SQL Server this is BCP or sqlcmd.



For Oracle, this is SQL*Plus.

For DB2 the supported export routine is EXPORT.



For Greenplum, the supported export routine is COPY.

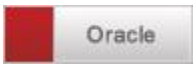
For PDW, the supported export routine is sqlcmd.

Use Temp Table

This option is only available for **Greenplum** and is enabled by default. Ticking this check-box creates a temporary table to export the data into. The temporary table is then deleted once the data has been exported.

Export Format

Routine-specific format to use to export the data.



This is the format of the data in the export file and is either Delimited Text or Width Fixed Text in Oracle and SQL Server.

In DB2, the supported formats are DEL (delimited), WSF (worksheet format) and IXF(intergrated exchange format).

In Greenplum, the supported formats are Text and CSV.

Export File Delimiter

Character that separates the fields within each record of the Export File for Delimited formats. The delimiter identifies the end of which field. Common field delimiters are tab, comma, colon, semi-colon, pipe. To enter a special character enter the uppercase string CHAR with the ASCII value in brackets (e.g. CHAR(9)). This is only available if the Export Format is Delimited Text.

Optionally Enclosed by

Character that brackets text fields within each record of the Export File for Delimited formats. A common example is ". This is only available if the Export Format is Delimited Text.

Header Row in Export

If a header line is required, choose business names or column names from this drop-down list. This option is not available in DB2.

Export Options

Allows the entry of export utility options. If more than one option is required, then a semi-colon should be used between options. This option is not available in DB2.

Export Modifier

DB2 only. This allows the entry of EXPORT modifiers in DB2.

Trigger file

If a trigger file is specified, then this file is created after the export file. It will normally contain a row count and a check sum.

Trigger Path

The purpose of the trigger file is to indicate that the export to the main file has completed and that it is now safe to load the file. Secondly the trigger file may contain control sums to validate the contents of the main load file. This field should contain the full path name to the directory in which a trigger file is to be generated on the destination system.

Trigger Name

Refers to the name of the file that is to be created as a trigger file. A trigger file typically contains check sums (row count or the sum of a numeric column). The variable $SEQUENCE$ can be used to provide a unique sequence number for the trigger file. Also, the data/file components YYYY, MM, HH, MI, SS can be used when enclosed with the $ character. For example, a trigger file name might be budget_$YYYYMMDD$.txt which would result in a file name like budget_20150520.txt.

Trigger Delimiter

Multiple fields in the trigger file are to be separated by the trigger delimiter.

Trigger Parameter 1,2,3

The checksums to be put in the trigger file. One of the row count and the sum of any numeric fields in the source data.

Compress After Export
Check this box if you want to compress the export file after it has been created.

Compress Utility Path
The directory in which the compress utility exists.

Compress Utility Name
The name of the compression utility executable.

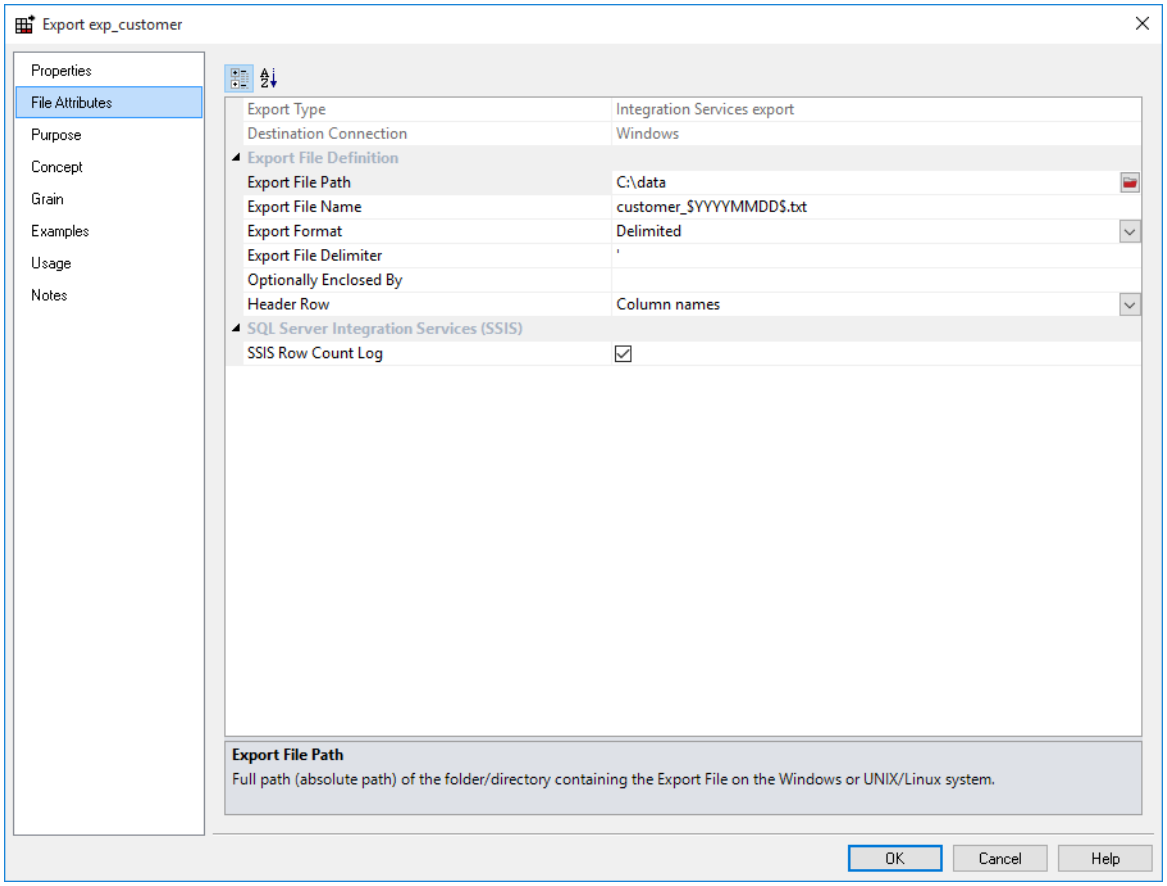
Compress Parameters
The name of the file to be compressed (using the WhereScape RED variable $EXPFILE$) and any commands or switches required to make the compression utility work. These parameters will depend on the compression utility used.

File Action 1, 2, 3
File Actions are only available for Greenplum. Use these fields to enter any command lines to run after the export such as copying files to another machine or deleting files.

FILE ATTRIBUTES - SSIS EXPORTS

The following fields below are available to define the location, name and definitions of the exported data file:

SQL Server File Attributes example screen



Export Type

Method of exporting data from the table. The available options are dependent on the Destination connection that can be specified via the Properties page.

Destination Connection

Destination to the file system to which data will be exported. The destination connection can be specified on the Properties screen.

Export File Definition

Export File Path

The full path (absolute path) of the folder/directory/ where the File is to be created on the Windows or UNIX/Linux system.

Export File Name

Name of the Export File to which the data will be exported. The variable $SEQUENCE$ can be used to provide a unique sequence number for the export file. Also the data/file components YYYY, MM, HH, MI, SS can be used when enclosed with the $ character. For example an export file name might be customer_$YYYYMMDD$.txt which would result in a file name like customer_20150520.txt.

Export Format

Routine-specific format to use to export the data.

Export File Delimiter

Character that separates the fields within each record of the Export File for Delimited formats. The delimiter identifies the end of which field. Common field delimiters are tab, comma, colon, semi-colon, pipe. To enter a special character enter the uppercase string CHAR with the ASCII value in brackets (e.g. CHAR(9)). This is only available if the Export Format is Delimited Text.

Optionally Enclosed by

Character that brackets text fields within each record of the Export File for Delimited formats. A common example is ". This is only available if the Export Format is Delimited Text.

Header Row

If a header line is required, choose business names or column names from this drop-down list. This option is not available in DB2.

SQL Server Integration Services (SSIS)

SSIS Row Count Log

When enabled, this option includes Row Count logging on SSIS exports.

EXPORT COLUMN PROPERTIES

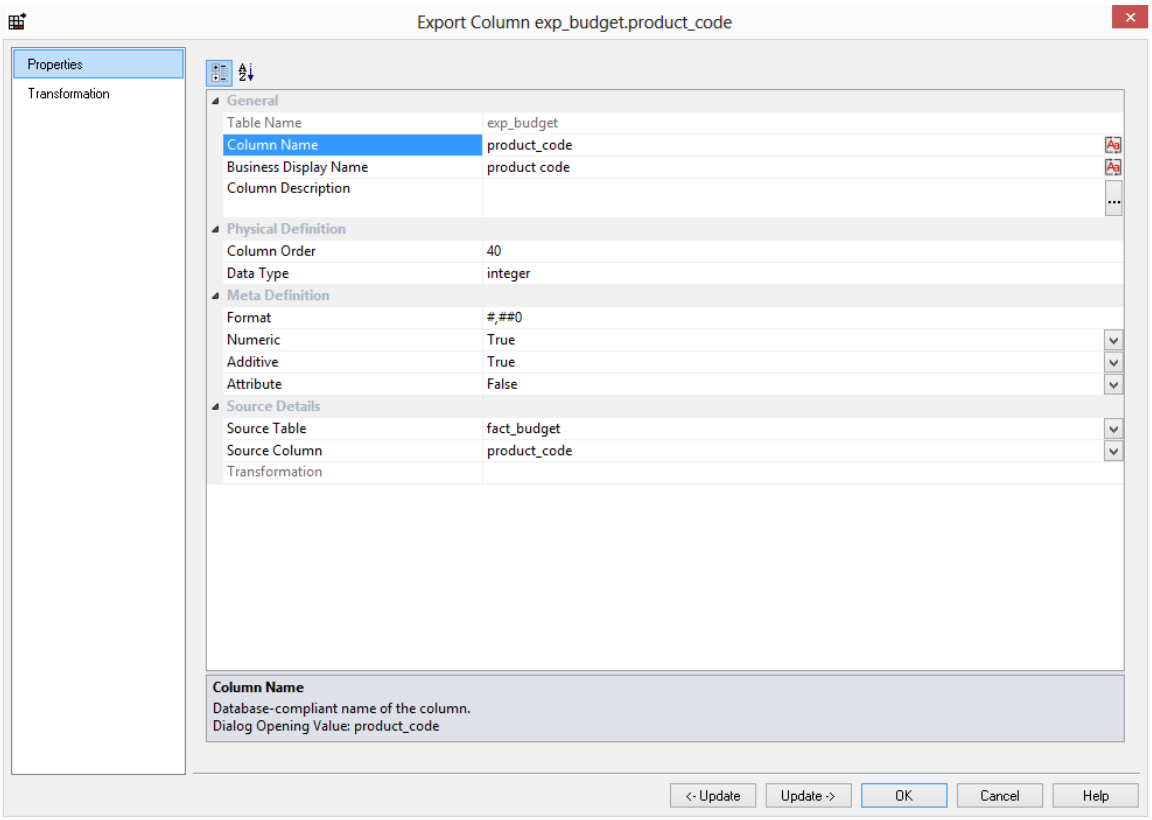


Table Name

Database-compliant name of the table that contains the column. [Read-only].

Column Name

Database-compliant name of the column. Typically, column-naming standards exclude the use of spaces etc. A good practice is to only use alphanumerics, and the underscore character. Changing this field alters the table's definition.

Note: A case conversion button on the right converts the text between different cases: UPPER CASE, Capitalized Case and lower case. The mode cycles to the next case in the sequence each conversion.

Business Display Name

Name that the business uses to refer to the column, which is included in the RED-generated documentation and can be used in the end user layer of other tools. [Does NOT affect the physical database table]. As such it is a free form entry and any characters are valid.

Note: A case conversion button on the right converts the text between different cases: UPPER CASE, Capitalized Case and lower case. The mode cycles to the next case in the sequence each conversion.

Column Description

This field contains the description for the column. It may be a description from a business user's point of view. This field might additionally contain information on where and how the column was acquired. For example, if the column is sourced from multiple tables or is a composite or derived column then this definition would normally describe the process used to populate the column. This field is used in the documentation and is available via the view **ws_admin_v_dim_col .** This field is also stored as a comment against the column in the database.

Column Order

Numeric value that controls the relative order of columns in the database create statement. The lowest numbered column will appear first in the table. Although this affects the physical table definition no action will be taken unless the table is re-created in the database. The columns can be re-numbered based on the existing order by choosing the **Respace Order Number** pop-up menu option when positioned over any column in the table. This action will number the columns in increments of 10 starting at 10. In addition to a simple change of the order field, the column order can be changed by first displaying the columns in the middle pane and then using drag and drop to move the columns around. This drag and drop process will automatically renumber the columns as required.

Data Type

Database-compliant data type that must be valid for the target database. Typical data types for Oracle are integer, number, char(), varchar2() and date. For SQL Server common types are integer, numeric, varchar() and datetime. See the database documentation for a description of the data types available. Changing this field alters the table's definition.

Format

Not relevant for Export Objects.

Numeric

Not relevant for Export Objects.

Additive

Not relevant for Export Objects.

Attribute

Not relevant for Export Objects.

Source Table

Identifies the source table where the column's data comes from. This source table is normally a stage table within the data warehouse. If the column was sourced from multiple tables, then the normal practice is to record one of the tables in this field and a comment listing all of the other tables in the **Source Strategy** field. This field is used when generating a procedure to update the fact table. It is also used in the track back diagrams and in the documentation.

Source Column

Identifies the source column where the column's data comes from. Such a column is normally a stage table column, which in turn may have been a transformation or the combination of multiple columns. This may also be a dimensional key where a dimension is being joined.

Transformation

Transformation. [Read-only].

SCRIPT BASED EXPORTS

A script based export object will have a Host Script defined. During the export process this host script is executed and the results returned.
During the **drag and drop** creation of a export object from a single table or view, a script can be generated by selecting one of the 'Script based' export options. This script can then be edited to more fully meet any requirements.

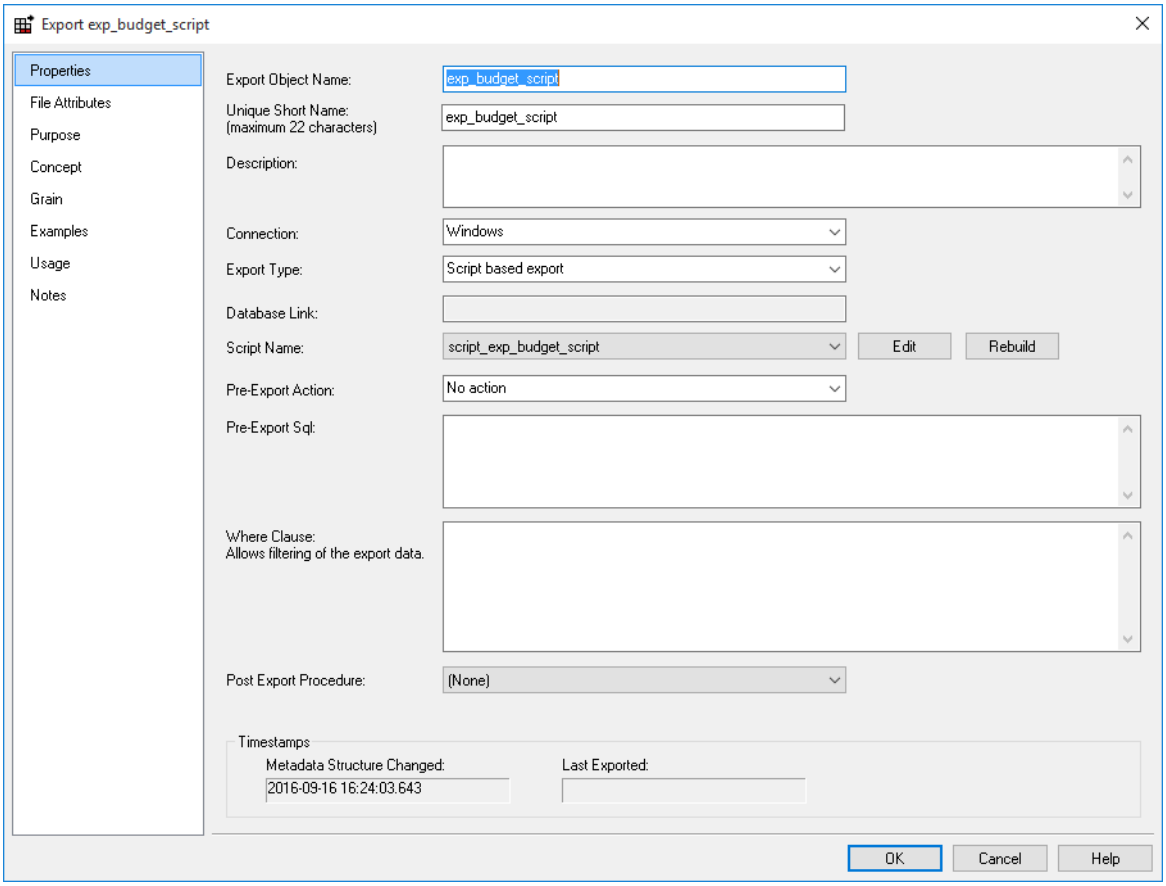
There are a number of conventions that must be followed if these host scripts are to be used by the WhereScape scheduler.

1. The first line of data in **standard out** must contain the resultant status of the script. Valid values are '1' to indicate success, '-1' to indicate a Warning condition occurred but the result is considered a success, '-2' to indicate a handled Error occurred and subsequent dependent tasks should be held, -3 to indicate an unhandled Failure and that subsequent dependent tasks should be held.
2. The second line of data in **standard out** must contain a resultant message of no more than 256 characters.

3 Any subsequent lines in **standard out** are considered informational and are recorded in the audit trail. The normal practice is to place a minimum of information in the audit trail. All bulk information should be output to **standard error.**

4 Any data output to **standard error** will be written to the error/detail log. Both the audit log and detail log can be viewed from the WhereScape RED tool under the scheduler window.

5 When doing **Script based exports**, it is easy to use the **rebuild** button to the right of the Script-name field to rebuild the scripts.



WhereScape®

CHAPTER 20
TRANSFORMATIONS

Standard ***column transformations*** (on page 651) can be used in WhereScape RED to perform calculations, change data types or format data.

Re-using complex transformations can save a significant amount of time. These can be achieved in two ways with WhereScape RED:

- Database Functions
- WhereScape RED Re-usable Transformations

IN THIS CHAPTER

COLUMN TRANSFORMATIONS

Each table or export object column can have a transformation associated with it. For all table types, except for load tables, the transformation will be included in the generated procedure for the table. These are executed as part of the procedure update. The transformation must therefore be a valid SQL construct that can be included in a **Select** statement. For example, we could have a transformation of 'load_order_line.qty * 0.125' to calculate a tax column of 12.5%. Click the **Transformation** tab on the column properties to enter a transformation.

Note: Transformations added to an existing table that have an update procedure are only put into effect when the procedure is re-generated and re-compiled.

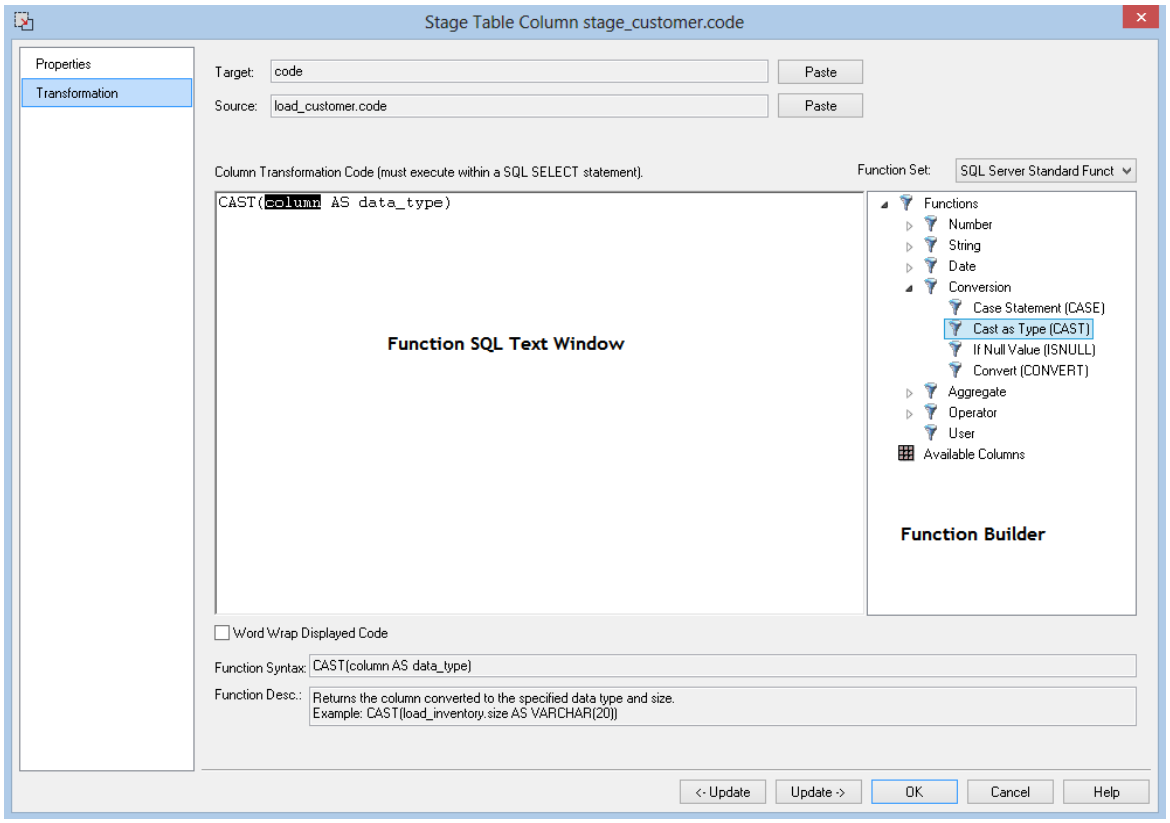
Column transformations on load tables are more complex, due to the unique nature of load tables. See ***Load Table Column Transformations*** (on page 654) for more details.

Dimension View transformations are included in the Database Definition Language (DDL) that creates the view in the database. Any changes to Dimension View column transformations require the view to be dropped and recreated.

Export object column transformations are dynamically applied for file exports. If the export object is executed via a host script, then the script needs to be regenerated for changes to transformations to take effect.

COLUMN TRANSFORMATION PROPERTIES

An example below shows the transformation property screen with a simple transformation:



The two special update keys allow you to update the column and step either forward or backward to the next column's properties. **ALT-Left Arrow** and **ALT-Right Arrow** can also be used instead of the two special update keys.

Function SQL Text Window

The **Function SQL Text Window** contains the SQL used in the transformation. It can be directly entered, built up using the **Function Builder** and **Add** buttons or a combination of both.

Function Builder

The **Function Builder** contains a list of standard database functions, operators, user defined functions and all columns belonging to all source tables.

Expanding the **Function Heading** displays the **Function Groups** (Number, String, Data, Conversion, etc.) and the **Re-usable Function Heading**. Similarly, expanding the **Data Heading** displays **Source Tables**.

Each function group or source table can in turn be expanded to show individual **Functions** and **Source Columns**.

Double-clicking on a function adds the **Function Model** to the Function SQL Text Window. The first variable (almost always the source column) is left highlighted in the Function SQL Text Window. This allows additional Functions or Source Columns to be added to the correct place in the Function SQL Text Window by double-clicking on the required Function or Source Column in the Function Builder.

Target Paste Button

The **Target Paste** button adds the current column in the form ColumnName to the Function SQL Text Window at the location of the cursor.

Source Paste Button

The **Source Paste** button adds the source table and column in the form TableName.ColumnName to the Function SQL Text Window at the location of the cursor.

Transform Stage

Only visible on load table column transformations. See ***Load Table Column Transformations*** (on page 654) for more information.

Function Set

This drop-down list enables the user to select which set of functions are to be displayed in the tree view when creating a transformation on a column of a table.

Update Buttons

The Update Buttons: **Update <-** and **Update ->** are used to move from the current column to previous and next columns respectively in the current table. The alternative is to exit the Column Transformation Properties, choose the next column, re-enter the Column Properties and choose the **Transformation** tab.

Function Syntax

The syntax guide for the **Function,** visible when you click on the function. Essentially the same as the Function model; loaded into the **Function SQL Text Window** when you click on the function. Read only.

Function Desc

The description of the **Function** visible when the function is clicked. Read only.

Function Model

The model (template SQL code) for a **Re-usable Transformations**. This is only visible for Re-usable Transformations. Read only.

Localize Transformation

The **Localize transformation** button breaks the link between a **column transformation** and the **Re-usable Transformation** it is based on. If this button is clicked, changes to the underlying re-usable transformation cannot be automatically propagated to the column transformation. This is only visible for Re-usable Transformations.

LOAD TABLE COLUMN TRANSFORMATIONS

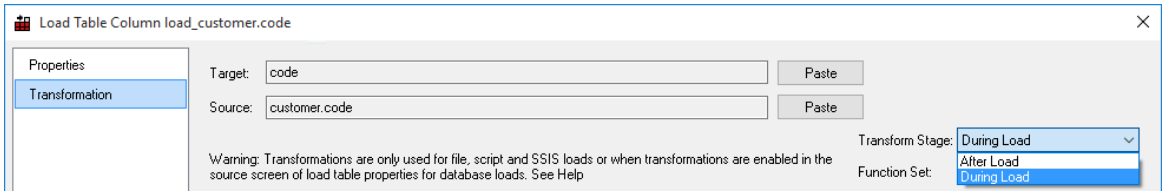
Overview

Data entering the data warehouse can be manipulated if required. This manipulation can occur at any stage, but is supported via a number of methods during the **Load** stage. Load tables provide options to transform data. If multiple pass transformations are required then a load table can be created from another load table, i.e. multiple load tables can be supported in the data flow path.

The options available differ depending on the type of load but in most cases the **after** transformation and post load procedure can be utilized. Specifically:

Database Link Load	• During Load transformations • After Load transformations • Post Load procedure
ODBC Based Load	• During Load transformations • After Load transformations • Post load procedure
File Based Load	• During Load transformations • After Load transformations • Post load procedure
Integration Services Load	• During Load transformations • After Load transformations
Script Based Load	• During Load transformations • After Load transformations • Post load procedure
Externally Loaded	• After Load transformations • Post Load procedure

The **Transformation** tab of a column's properties is used to define **during** and **after** load transformations. It can only be one or the other for a specific column. One column can be used to build another, so an **after** can be based on the results of a **during**, if different columns are used.



Note: The **During** transformations use **Source Table** columns. The **After** transformations use the **Load Table** columns.

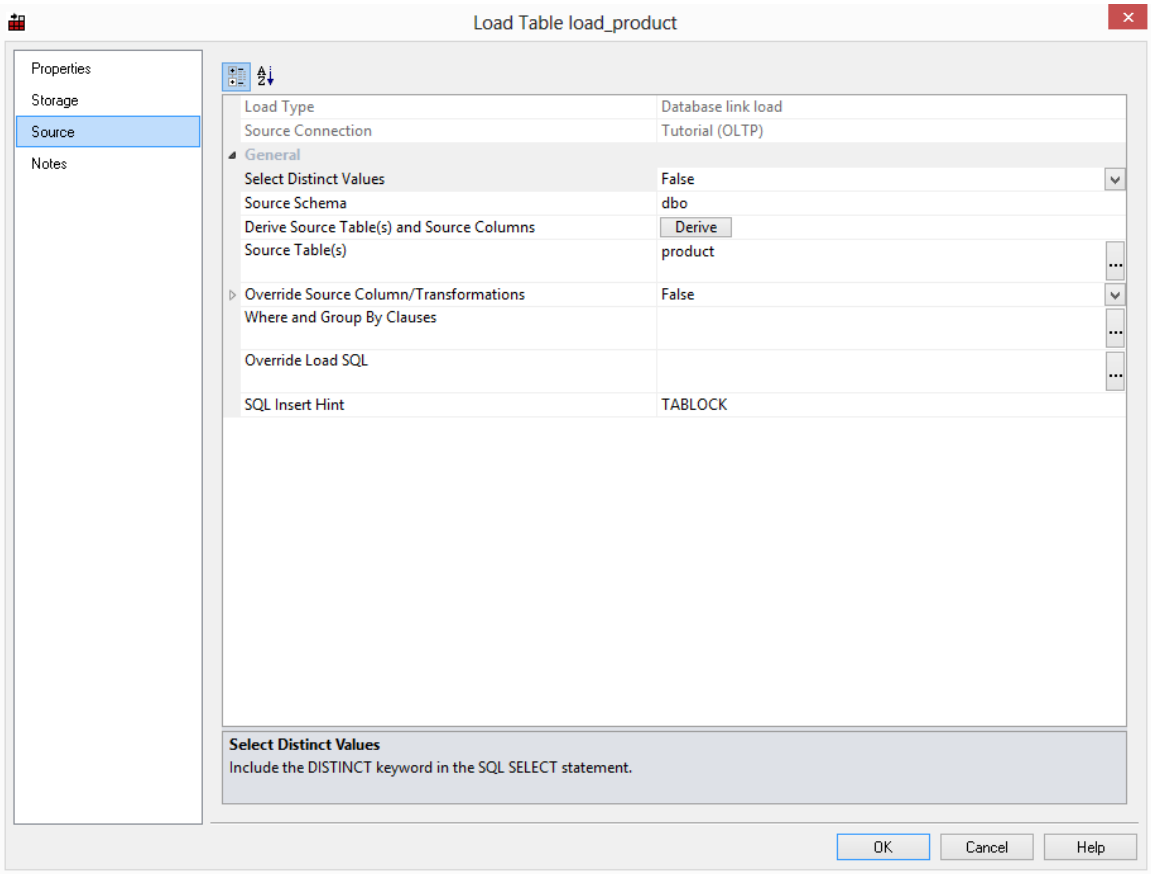
All **After** transformations take place in the native SQL language of the data warehouse database. The **During** transformations differ in terms of which language is used. This is particularly true for file based loads. Normally the 'During' transformation will occur in the native SQL language of the source database.

For **Flat file loads** using SSIS, **After Load transformations** use the SQL syntax of the target database but **During Load transformations** use SSIS expression syntax that can be referred to on the Microsoft Developer Network website : https://msdn.microsoft.com/en-us/library/ms137547(v=sql.110).aspx.

DATABASE LINK DURING LOAD TRANSFORMATIONS

The **during** load transformation allows the manipulation of column data as it enters the data warehouse.

By default, **Database link loads** and **ODBC based loads** have **During** and **After** transformations enabled.



When transformations are enabled, the contents of a source table/source column for each column are used as the basis of the loading statement.

- If source table and source column are null, then a null is used.
- If data exists in the **Transformation** tab of a column's properties, then this transformation data is used instead of the source table/source column combination.

Example

The following load table columns will generate the load sql statement if no transformation data is present against these columns.

load_product Columns

Column Name	Display Name	Data Type	Source Table	Source Column
code	code	numeric(6)	product	code
description	description	varchar(64)	product	description
prod_line	prod line	varchar(24)	product	prod_line
prod_group	prod group	varchar(24)	product	prod_group
subgroup	subgroup	varchar(24)	product	subgroup

The SQL code from the load results is:

Results

Object	Message
load_product	Load Complete
load_product	Table dbo.load_product truncated.
Load	INSERT INTO dbo.load_product WITH (TABLOCK)
Load	SELECT
Load	product.code,product.description,product.prod_line,product.prod_group,product....
Load	From [Tutorial].dbo.product product
Load	;
Load	9 rows loaded

Results Reports

If the Column has a transformation defined as follows:

UPPER(substring(description,1,1)) + LOWER(substring(description,2,1))

then the following SQL statement will be executed.

Results

Object	Message
load_product	Table dbo.load_product truncated.
Load	INSERT INTO dbo.load_product WITH (TABLOCK)
Load	SELECT
Load	product.code,UPPER(substring(description,1,1)) + LOWER(substring(description,
Load	2,1)),product.prod_line,product.prod_group,product.subgroup
Load	From [Tutorial].dbo.product product
Load	;
Load	9 rows loaded

Results Reports

FILE DURING LOAD TRANSFORMATIONS

The loading of flat files is performed using the database specific loader. For Oracle, this is SQL*LOADER. For SQL Server, it is the Bulk Insert Transact SQL statement. The contents of the **Transformation** tab in the column's Properties are the functions and conversions supported by the database loader.

Example

SQL*LOADER performs transformations such as:

- DATE 'DD-MMM-YYYY' converts from a value such as 23-Mar-1999 to an Oracle date.
- "TO_NUMBER(:COL)" converts the data to a number
- "NVL(TRIM(:COL),0)" trims leading and trailing white characters and inserts zero if null.

AFTER LOAD TRANSFORMATIONS

After transformations, will initiate a pass of the load table after the load has occurred. They allow manipulation of the load table columns using the database and SQL functions.

Example

The following **after** transformation set for the column code in the table load_product

lpad(code,5,'0')

would result in the following SQL statement being executed after the load:

update load_product set code = lpad(code,5,'0');

DATABASE FUNCTIONS

Database functions can be created in the data warehouse database (as procedure objects in WhereScape RED) and used in column transformation in WhereScape RED. See the Oracle and SQL Server documentation on functions and ***Procedures and Scripts*** (on page 667) for more information on creating functions inside WhereScape RED.

RE-USABLE TRANSFORMATIONS

WhereScape RED Re-usable Transformations allow a complex SQL transformation using standard database functions (including User Defined Functions) to be built once and reused in multiple column transformations. Functionality is included to manage and propagate changes to re-usable transformations.

CREATING A NEW RE-USABLE TRANSFORMATION

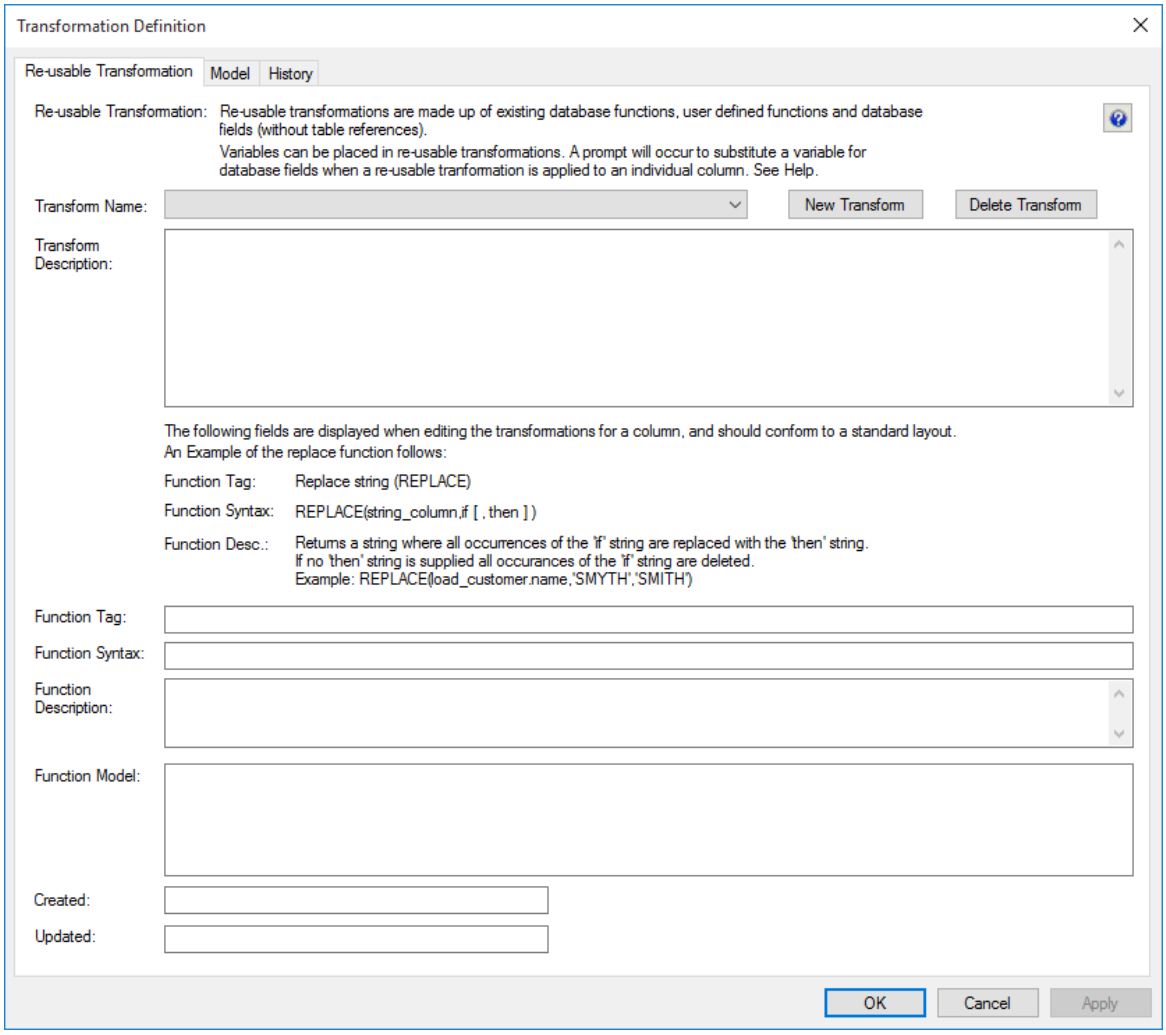
New **re-usable transformations** are created from the **Tools/Define Re-Usable Transformations** menu.

Creating a new re-usable transformation is a three-step process:

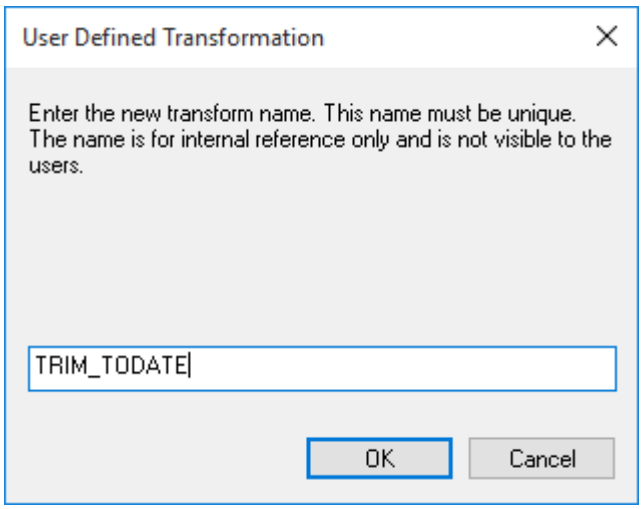
- Specify the name of the transformation
- Enter metadata for the transformation
- Define the transformation

SPECIFY THE NAME OF THE TRANSFORMATION

After selecting **Define Re-Usable Transformations** from the **Tools** menu the following dialog is displayed:



Click **New Transform** and enter a name for the re-usable transformation:



Note: This is the internal WhereScape RED name for the transformation, not the name developers reference when utilizing the transformation on column transformations.

Click **OK**.

ENTER RE-USABLE TRANSFORMATION METADATA

Enter the following metadata for the transformation to describe the transformation for developers.

Transform Description

A general description of the transformation.

Function Tag

This is the function name visible to the users, which can be selected from the function builder when building column transformations.

Function Syntax

The syntax guide for the function. This is visible in the function builder when clicking on the User defined function.

Function Description

The description of the function visible in the function builder when clicking on the User defined function.

DEFINE THE TRANSFORMATION MODEL

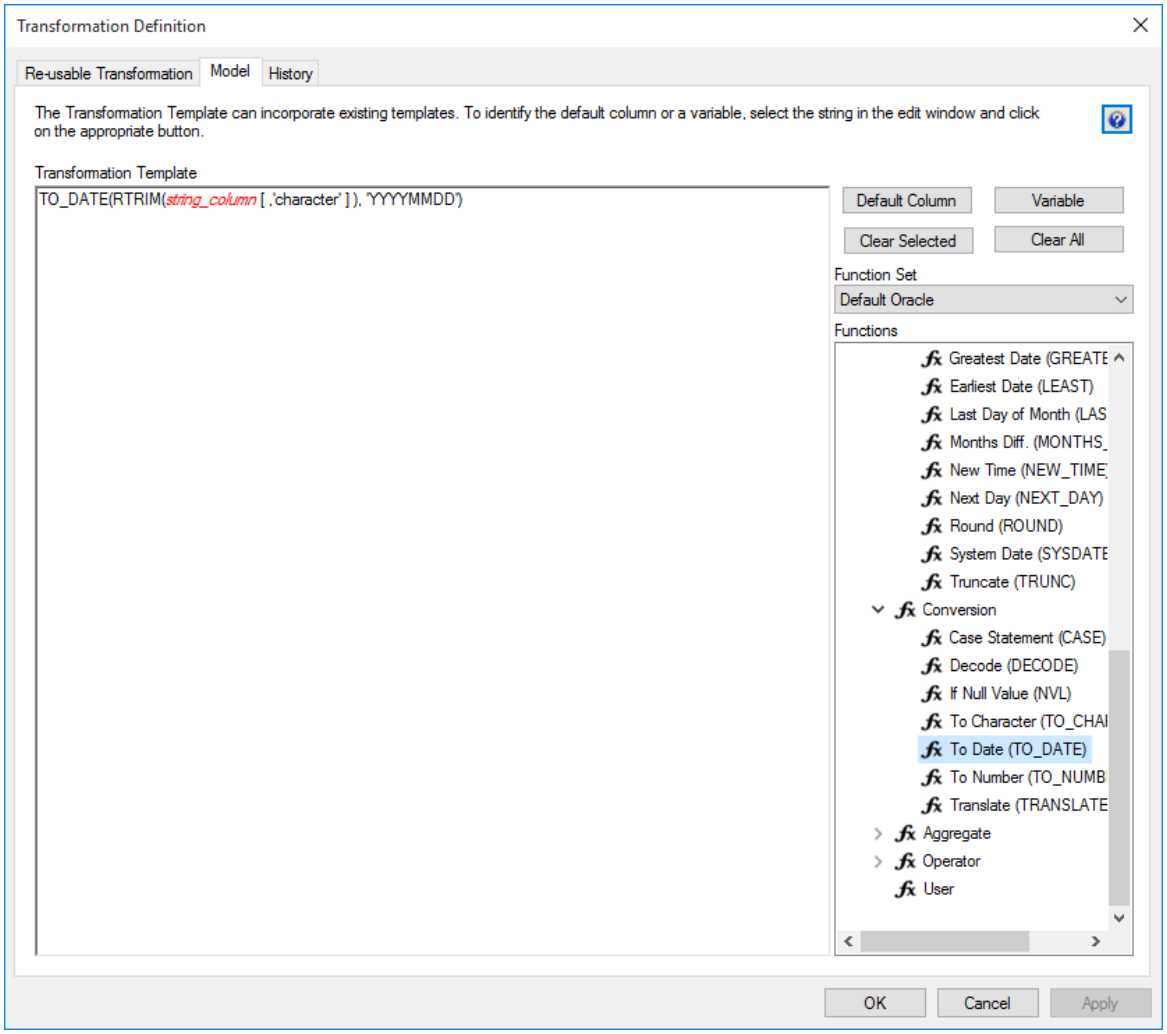
Once the transformation has been created and the metadata entered, the actual SQL code used by the transformation can be defined. The SQL code can be entered directly or via the **Function Builder** on the right side. To use the function builder, expand the tree to find the required function.

Oracle Example of Building a Transformation Model

To build a model TO_DATE, converting a trimmed string in YYYYMMDD format to a date, do the following:

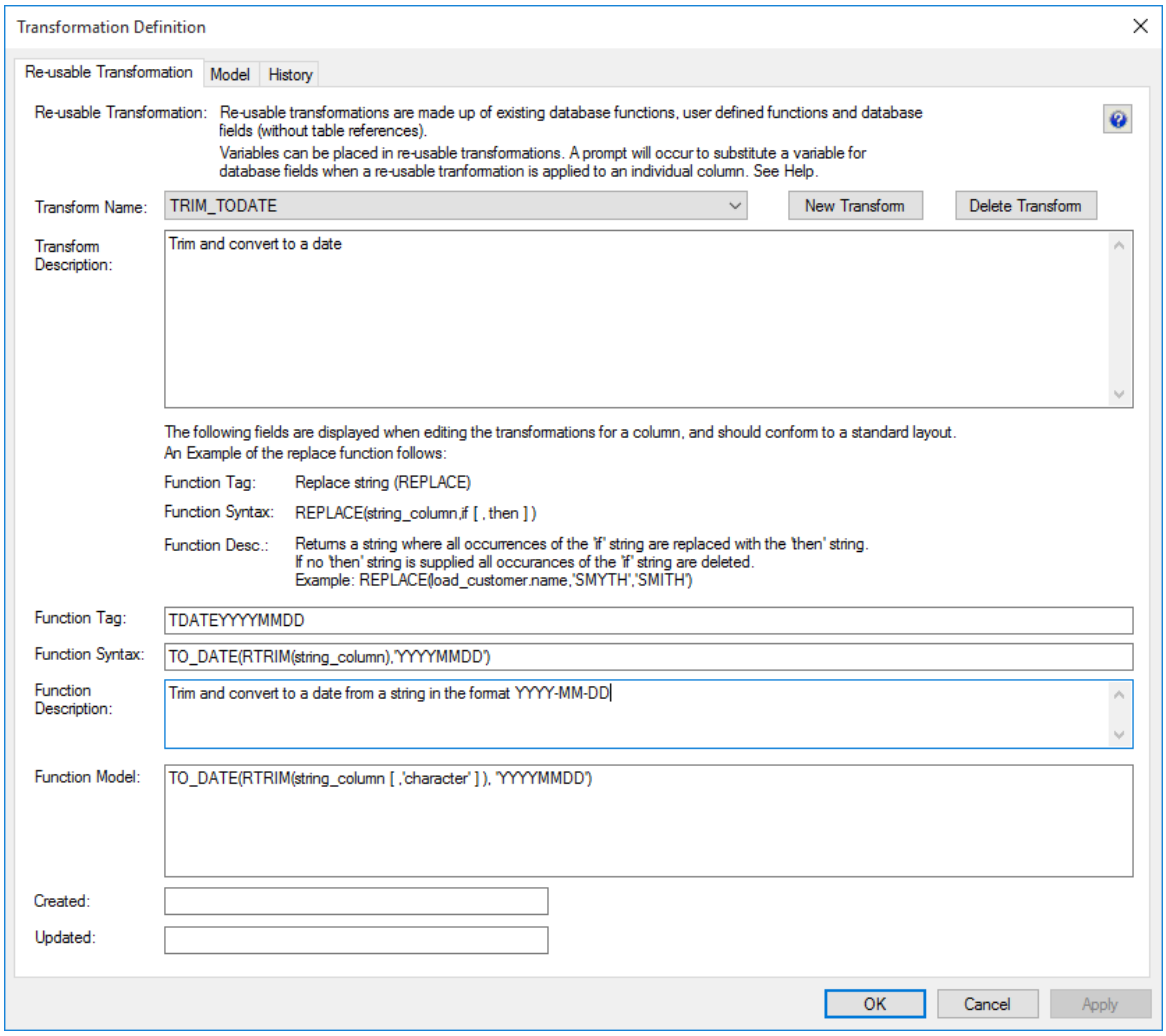
- Click on the **Model** tab
- Expand **Functions** heading
- Expand the **Conversion** heading
- Double-click on **To Date (TO_DATE)**
- Highlight **numeric_column**
- Expand the **String** heading
- Double-click on **Right Trim (RTRIM)**
- Highlight **[format]** and type '**YYYYMMDD**'

You should see the following:



Now click **OK**.

COMPLETED RE-USABLE TRANSFORMATION



CHANGING A RE-USABLE TRANSFORMATION

To change a re-usable transformation:

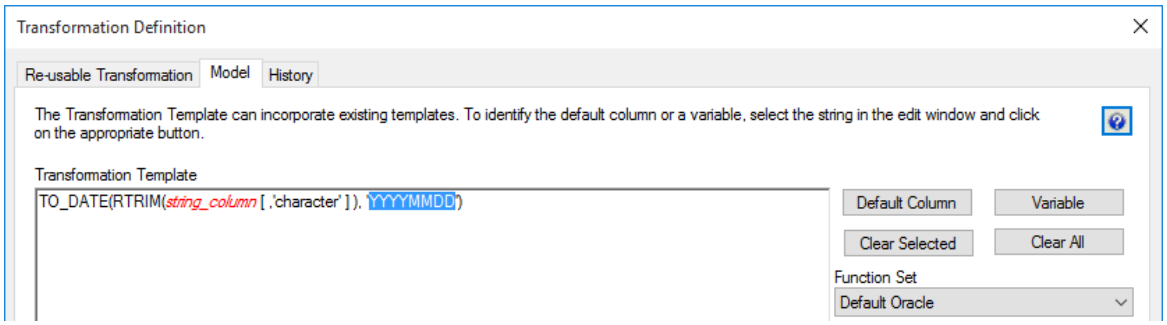
- Select **Re-Usable Transformations** from the **Tools** menu.
- Choose the transformation from the **Transform Name** drop-down list.
- Click on the **Model** Tab.
- Change the SQL as required.
- Click **OK.**

Example of a change to the Model SQL

In the example used in ***Creating a New Re-usable Transformation*** (on page 659), the SQL was:

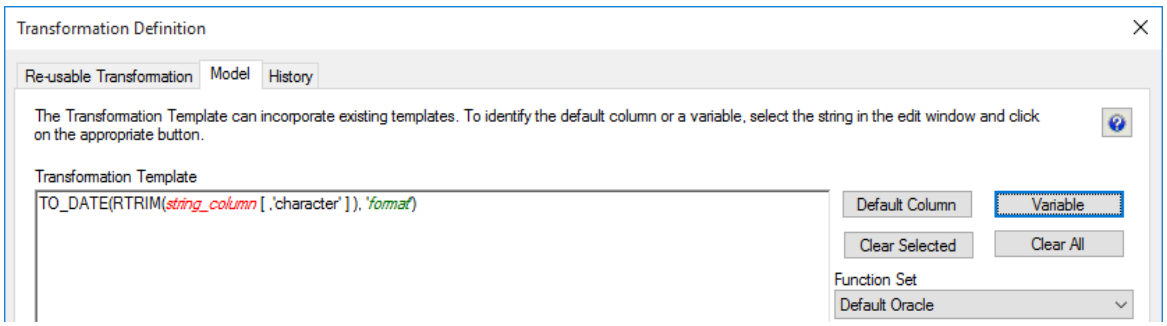
TO_DATE(RTRIM(string_column),'YYYYMMDD')

Change the SQL to allow the format to be specified when the transformation is used by changing **YYYYMMDD** to **format**.



Then highlight the word **format** and click on the **Variable** button. This makes the word **format** a variable than can be substituted when the Re-usable Transformation is used.

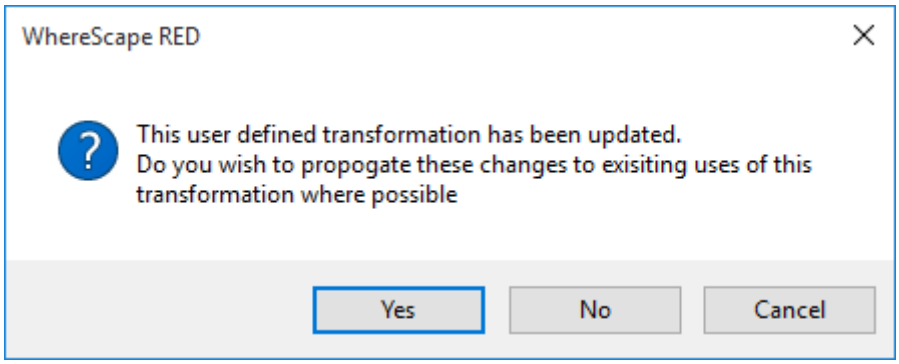
Now **format** is green and in italics:



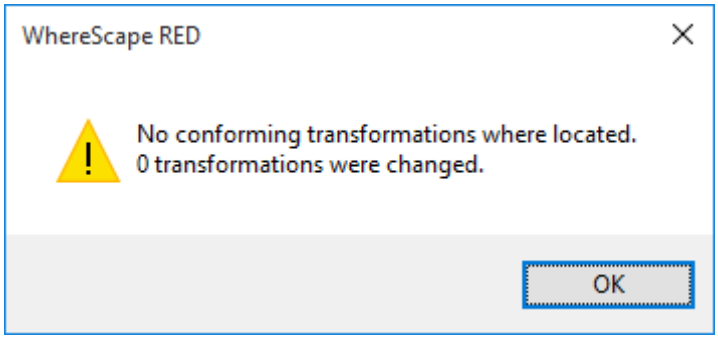
Click **OK**.

APPLYING CHANGES TO DEPENDANT TRANSFORMATIONS

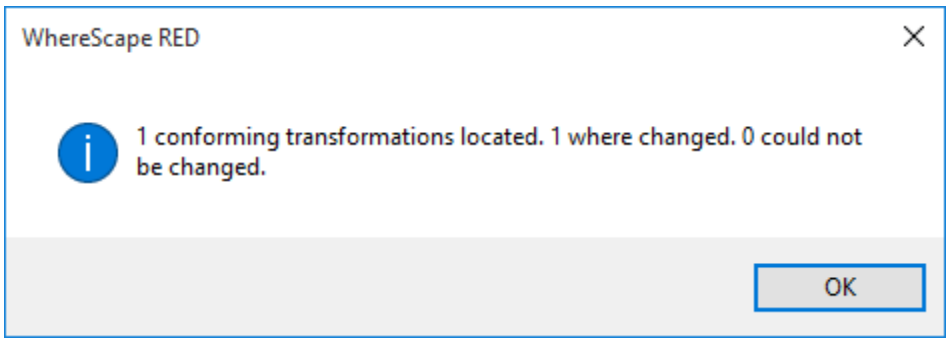
After changing a **Re-usable Transformation**, a dialog appears asking to confirm that changes should be applied to individual columns using the transformation, where possible:



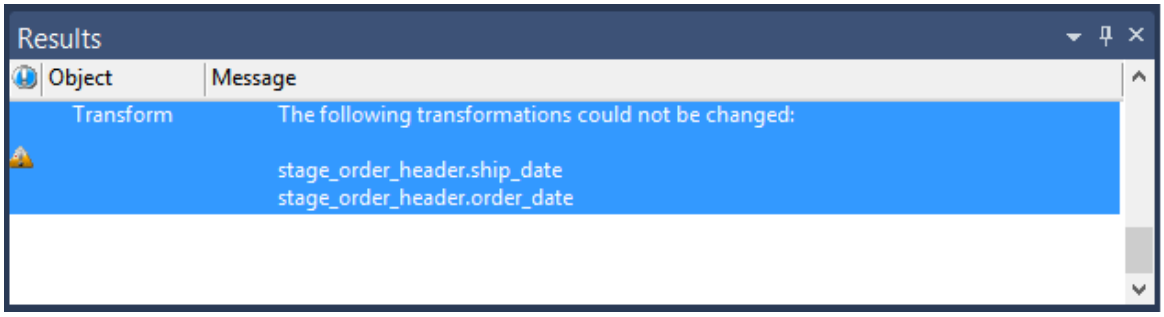
If the **Re-usable Transformation** doesn't have any dependant columns, then the following message is displayed:



If the **Re-usable Transformation** has been used for dependant columns then this message is displayed:



When an attempt is made to update a dependant Transformation, and the transformation has been modified in such a way as to make it impossible for the changes to the Re-usable Transformation to be applied, the error message above will include a count of the failures. The results window will detail the columns (and tables) where the update failures have occurred:



USING RE-USABLE TRANSFORMATIONS

Re-usable transformations are used exactly the same way as any standard database **Function**. They can be used on any object type. See ***Column Transformation Properties*** (on page 652)

WhereScape®

CHAPTER 21
PROCEDURES AND SCRIPTS

WhereScape RED has a **Procedure** object group for database stored procedures and a **Script** object group for host system scripts, such as UNIX shell scripts or Windows batch files.

Procedures

WhereScape RED generates the bulk of the procedures during a prototype build, but these procedures can be customized. In fact, it would be normal practice once the prototype phase is completed to modify these procedures to meet specific requirements. The procedure object group refers to the concept of database stored procedures. Specific objects may in fact be functions, procedures or packages.

Scripts

Host scripts are generated when a script based file load is chosen during a file drag and drop from a Windows or UNIX connection. Scripts can also be created manually provided the rules for their inclusion into the scheduling process are followed.

This chapter covers the generation, editing and compilation of procedures, as well as the generation, editing and testing of host scripts as well as explaining the components required to allow host scripts to work in the scheduler environment.

IN THIS CHAPTER

PROCEDURE GENERATION

WhereScape RED generates template procedures to assist in the various phases of the data warehouse build process. A procedure is generated by selecting the **(Build Procedure...)** option from a drop-down box that is used to display the update, initial build, or post load procedure in a table's properties.

For **Load** tables a post load procedure can be generated by selecting the option above. This post load procedure is designed to assist in the management of file loads where a trigger file has been used.

For **Dimension**, **Stage**, **Fact** and **Aggregate** tables an Update or Initial Build procedure can be generated by selecting the option above from the appropriate drop-down box. When an 'Update' procedure generation is selected for a Dimension a **get_key function** is also automatically created.

If a new procedure is created from scratch (i.e. not auto generated) then an outline of the syntax required by the WhereScape scheduler can be generated by selecting the **Tools/Create Procedure Outline** menu option when in the procedure editor.

Wrapper procedures

In some cases, multiple procedures will be required to update a table. In such cases, it is best to create a top level procedure that is seen by the scheduler as the 'Update' procedure. This procedure can in turn call other procedures.

Example
We may have a fact table that is updated from multiple stage tables. This Oracle wrapper procedure calls two child procedures, one for each stage table that is to update the fact table. A status is reported back to the audit trail for each stage and an overall status ascertained for the fact table update.
The Oracle wrapper procedure may look as follows:

```
CREATE OR REPLACE procedure update_fact_sales_detail
    (
    v_sequence          IN  number,
    v_job_name          IN  varchar2,
    v_task_name         IN  varchar2,
    v_job_id            IN  number,
    v_task_id           IN  number,
    v_return_msg        OUT varchar2,
    v_status            OUT number
    )
    AS

    --------------------------------------------
    -- Control variables used in most programs
    --------------------------------------------
    v_MsgText          varchar2(255); -- Text for audit_trail
    v_step             integer := 0;  -- return code
    v_invoice_msg     varchar2(256);  -- from invoice update
    v_order_msg       varchar2(256);  -- from order update
    v_invoice_status  number  := 0;   -- from invoice update
    v_order_status    number  := 0;   -- from order update
    v_retcode         varchar2(1);

BEGIN

    ----------------------------------------
    -- Update fact table with info from orders stage table
    -- Report the results to the audit trail.
    ----------------------------------------
    v_step := 10;
    update_fact_sales_detail_o(v_sequence,v_job_name,v_task_name,
                v_job_id, v_task_id,v_order_msg, v_order_status);
    If v_order_status = 1 Then
        v_retcode := 'S';
    ElsIf v_order_status = -1 Then
        v_retcode := 'W';
    ElsIf v_order_status = -2 Then
        v_retcode := 'E';
    ElsIf v_order_status = -3 Then
        v_retcode := 'F';
    End If;
    v_status := WsWrkAudit(v_retcode,v_job_name,v_task_name, v_sequence,
              v_order_msg,NULL,NULL,v_task_id, v_job_id);

    ----------------------------------------
    -- If The previous update (orders) was successful
    -- Update fact table with info from invoice stage table
    ----------------------------------------
    If v_order_status > -2 Then
```

```
        v_step := 20;
        update_fact_sales_detail_i(v_sequence,v_job_name,v_task_name,
                    v_job_id, v_task_id,v_invoice_msg, v_invoice_status);
        If v_invoice_status = 1 Then
            v_retcode := 'S';
        ElsIf v_invoice_status = -1 Then
            v_retcode := 'W';
        ElsIf v_invoice_status = -2 Then
            v_retcode := 'E';
        ElsIf v_invoice_status = -3 Then
            v_retcode := 'F';
        End If;
        v_status := WsWrkAudit(v_retcode,v_job_name,v_task_name, v_sequence,
                    v_invoice_msg,NULL,NULL,v_task_id, v_job_id);
    Else
        v_invoice_status := 0;
    End If;

    ---------------------------------------
    -- All Done report the results
    -- If any of our called procedures failed or gave
    -- warnings then report the final status as such
    ---------------------------------------
    v_step := 90;
    v_status := v_order_status;
    If v_invoice_status < v_order_status Then
        v_status := v_invoice_status;
    Else
        v_status := v_order_status;
    End If;
    v_return_msg := 'fact_sales_detail update ';
    If v_status = 1 Then
        v_return_msg := v_return_msg||'completed.';
    ElsIf v_status = -1 Then
        v_return_msg := v_return_msg||'completed with warnings.';
    Else
        v_return_msg := v_return_msg||'Failed.';
    End If;

    RETURN;

EXCEPTION
  WHEN OTHERS THEN
    v_return_msg := 'Unhandled Exception in '||'update_fact_sales_detail. '||
                ' Step ' ||v_step || ' '||SQLERRM;
    v_status := WsWrkAudit('F',v_job_name,v_task_name, v_sequence,
                v_return_msg,SQLCODE,SQLERRM,v_task_id, v_job_id);
    v_status := -3;

    RETURN;
END update_fact_sales_detail;
```

Procedure Placeholders

Procedure placeholders can help in moving procedures between environments without the necessity of regenerating those same procedures. In <PRODUCT, the purpose of these placeholders is to automatically substitute the corresponding strings, which is needed for a specific environment.
The following procedure placeholders described below can be found in the update_xxxx_xxxx procedure.

[TABLEOWNER] is used as a placeholder to replace the schema name defined in the connection or target.
For targets, the **[TABLEOWNER]** placeholder is derived from the Target Location Database/Schema in the connection. The target can be changed in the table's Storage screen, on the Target drop-down list. For more information about Target Location Database/Schema in connections and table's storage screens, see ***Connections - Database***, ***Connections - ODBC*** and ***Table Storage Properties***.
When moving tables between environments, the **[TABLEOWNER]** placeholder is determined by the individual connection of the target environment.

Example
During the compilation process of the procedure, [TABLEOWNER.tablename] will be replaced with PRODUSER.tablename if the table owner is PRODUSER in the destination environment.

```
-- ============================================================================
-- DBMS Name       :    Oracle
-- Script Name     :    update_dim_customer
-- Description     :    Update the dimension dim_customer
-- Generated by    :    WhereScape RED Version 6.8.4.4 ( Build 151111-200341 pre-release )
-- Generated for   :    RED Documentation
-- Generated on    :    Tuesday, November 17, 2015 at 09:05:24
-- Author          :    WhereScape Documentation
-- ============================================================================
-- Notes / History
--

CREATE OR REPLACE PROCEDURE update_dim_customer
  (
  p_sequence         IN  number,
  p_job_name         IN  varchar2,
  p_task_name        IN  varchar2,
  p_job_id           IN  number,
  p_task_id          IN  number,
  p_return_msg       OUT varchar2,
  p_status           OUT number
  )
  AS

  ------------------------------------------------------
  -- Control variables used in most programs
  ------------------------------------------------------
  v_msgtext          varchar2(255); -- Text for audit_trail
  v_sql              varchar2(255); -- Text for SQL statements
  v_set              integer := 0;  -- commit set
  v_analyze_flag     integer := 0;  -- analyze flag
  v_step             integer := 0;  -- return code
  v_update_count     integer := 0;  -- no of records updated
  v_insert_count     integer := 0;  -- no of records inserted
  v_count            integer := 0;  -- General counter;

  ------------------------------------------------------
  -- Variables
  ------------------------------------------------------
  v_DimRec           [TABLEOWNER].[dim_customer]%ROWTYPE;

  ------------------------------------------------------
  -- Select all input columns for all dimension rows
  ------------------------------------------------------
```

WhereScape RED Tip: dim_date

The TABLEOWNER placeholder is especially useful in update procedures when the related table is moved to a different schema or environment. For example, when moving dim_date to other schemas, [TABLEOWNER] will be replaced with the schema of the table when the procedure is compiled.

To do this, prefix the table name in the procedure to [TABLEOWNER].table_name (e.g. [TABLEOWNER].dim_date). Then it is only necessary to do a recompile instead of rebuilding or regenerating the procedure.

Note: For Oracle and DB2 databases [TABLEOWNER] has already been added to the sample dim_date update procedures.

[FUNCTIONOWNER] is used as a placeholder for **Greenplum** databases to enable the creation of procedures in different targets without the need of procedure regeneration.

```
--------------------------------------------------------------------------------
-- DBMS Name       :    Greenplum
-- Script Name     :    update_stage_order_header_gp
-- Description     :    Update the Stage  table stage_order_header_gp
-- Generated by    :    WhereScape RED Version 6.8.4.4 ( Build 151111-200341 pre-release )
-- Generated for   :    RED Documentation
-- Generated on    :    Tuesday, November 17, 2015 at 09:46:12
-- Author          :    WhereScape Documentation
--------------------------------------------------------------------------------
-- Notes / History
--

CREATE OR REPLACE FUNCTION [FUNCTIONOWNER].[update_stage_order_header_gp]
  (
  p_sequence      IN    integer,
  p_job_name      IN    varchar(256),
  p_task_name     IN    varchar(256),
  p_job_id        IN    integer,
  p_task_id       IN    integer,
  p_parameters    INOUT varchar[][],
  p_return_msg    INOUT varchar(256),
  p_status        INOUT integer
  )
RETURNS RECORD
AS $BODY$
DECLARE
  --------------------------------------------------------------------------------
  -- Control variables used in most programs
  --------------------------------------------------------------------------------
```

[SEQUENCE.tablename] is used as a placeholder for **Oracle** and **Greenplum** database sequences where the Sequence name can be defined for artificial keys in a table's storage screen.
This placeholder can be edited in the table's storage screen "Artificial Key Sequence Name" field, if that table contains an artificial key. For more information about replacing the sequence name in Oracle see ***Table Storage Screen - Oracle*** (on page 208).

```
--=====================================================
-- If not found then add it if we have the flag set.
--=====================================================
WHEN NO_DATA_FOUND THEN
  IF p_auto_add = 'Y' THEN
    INSERT INTO [TABLEOWNER].[dim_customer]
    (
      dim_customer_key,
      customer_code,
      dss_update_time
    )
    VALUES
    (
      [SEQUENCE.dim_customer].NEXTVAL,
      p_customer_code,
      SYSDATE
    )
    RETURNING dim_customer_key INTO p_dim_customer_key;
```

[SCHEMA.tablename] is used as a placeholder for **Oracle** database schemas in partitioned fact tables, to facilitate moving partitioned fact table procedures between environments without having to regenerate the procedures.

```
BEGIN

  --------------------------------------------------------
  -- Get the current partition tablespace in case we need
  -- to add new partitions
  --------------------------------------------------------
  v_step := 100;
  SELECT MAX(tablespace_name) INTO v_part_tablespace
  FROM   all_tab_partitions
  WHERE  UPPER(table_owner) = UPPER('[SCHEMA.fact_customer]')
  AND    UPPER(table_name) = UPPER('fact_customer');

  --------------------------------------------------------
  -- Get the first partition name and maximum value as we
  -- will place all prior periods in this partition
  --------------------------------------------------------
  v_step := 110;
  SELECT partition_name, high_value
  INTO   v_min_part, v_min_high_value
  FROM   all_tab_partitions
  WHERE  UPPER(table_owner) = UPPER('[SCHEMA.fact_customer]')
  AND    UPPER(table_name) = UPPER('fact_customer')
  AND    partition_position = (
    SELECT MIN(partition_position)
    FROM   all_tab_partitions
    WHERE  UPPER(table_owner) = UPPER('[SCHEMA.fact_customer]')
    AND    UPPER(table_name) = UPPER('fact_customer'));

  -- Get the part high value and period
  v_min_part_value := TO_NUMBER(v_min_high_value);
  v_min_part_period  := TO_NUMBER(SUBSTR(v_min_part,3,8));

  --------------------------------------------------------
  -- Loop through each partition we need to modify
  --------------------------------------------------------
```

PROCEDURE EDITING

WhereScape RED includes a procedure editor which allows the maintenance of the various procedures, functions and packages within the data warehouse. The editor is invoked by double-clicking on a procedure name in the left pane or by right-clicking on the procedure name and selecting **Edit the Procedure**.

A procedure can be compiled by selecting the **Compile/Compile** menu option. See the section on compiling procedures for more information.

This section will discuss some of the features of the procedure editor.

In the following sections reference is made to a selected block of text. A selected block of text is a normal windows selection where the text in question is highlighted. Normally achieved by holding down the left mouse button and moving the cursor.

Indenting code

The **tab** character inserts four spaces into the text. A **shift/tab** removes four spaces.

Cut, Copy, Paste and Delete

The normal Windows cut, copy, paste and delete functions are available either through the toolbar or via the right mouse pop-up menu.

Indenting a block of text

A selected block of text can be indented by four spaces by depressing the **tab** character. Each tab will indent by a further four spaces. A **shift/tab** will remove four spaces from the front of each line in the selected block.

Commenting out a block of text

A selected block of text can be turned into a comment by right-clicking and selecting **Comment**. The editor will place two dashes '--' at the front of each line in the selected block. In the same way a block of text can be uncommented by choosing the **Uncomment** option. Note that only lines that start with two dashes in the left most column can be uncommented.

Inserting Steps

The right-click menu option **Insert Step** will insert a code line of the format 'v_step := 1000;'. Each subsequent insert will add 100 to the step value. The Alt/Z key can also be used to insert a step line.

The v_step variable is used in the event of an unhandled exception. In such a case the current v_step value is reported, and it may be possible to ascertain the code that failed based on the value of this v_step variable.

Note: If a step is inserted via this method then the step numbering will be automatically reset for all steps numbering from 100 in increments of 100 when the procedure is compiled or saved.

Inserting Audit Message

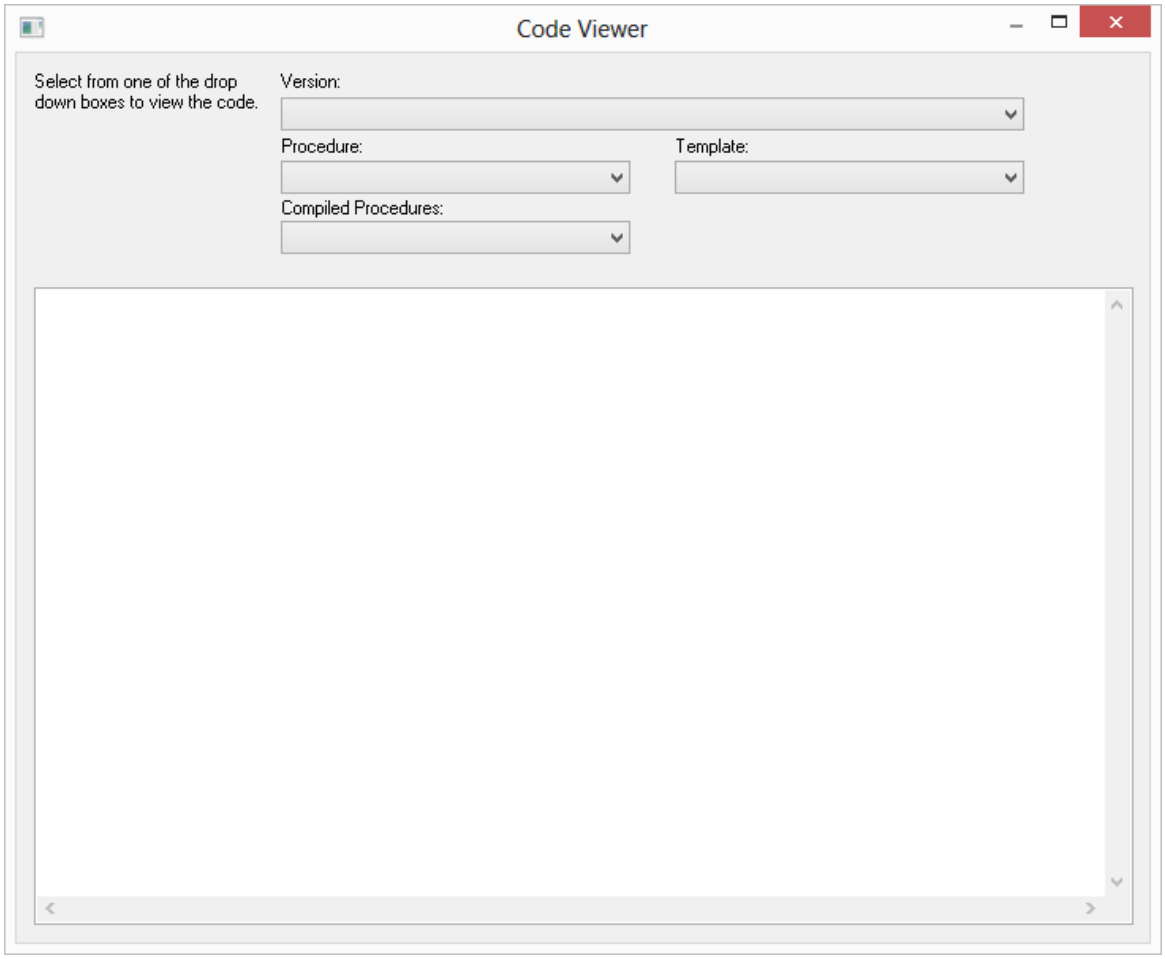
The right-click menu option **Insert Audit Log Call** inserts several lines of code into the procedure which calls WsWrkAudit. This API is used to log messages to the audit log in the WhereScape RED scheduler. By default, the message is generated to log the current step number. It can be changed by altering the first added line. The Alt/A key can also be used instead of the right-click menu option.

Inserting Detail/Error Message

The right-click menu option **Insert Detail/Error Log Call** inserts several lines of code into the procedure which calls WsWrkError. This API is used to log messages to the detail/error log in the WhereScape RED scheduler. By default, the message is generated to log the current step number. It can be changed by altering the first added line. The Alt/D key can also be used instead of the right-click menu option.

Viewing other procedural code

During the editing process, it is possible to pop up a window containing other procedural code. This window allows cut and paste operations. In such a way it can be used as a work area or as a source of code. Select the **Tools/View Procedure or Template** menu option to bring the viewer window up. A dialog will appear as follows:

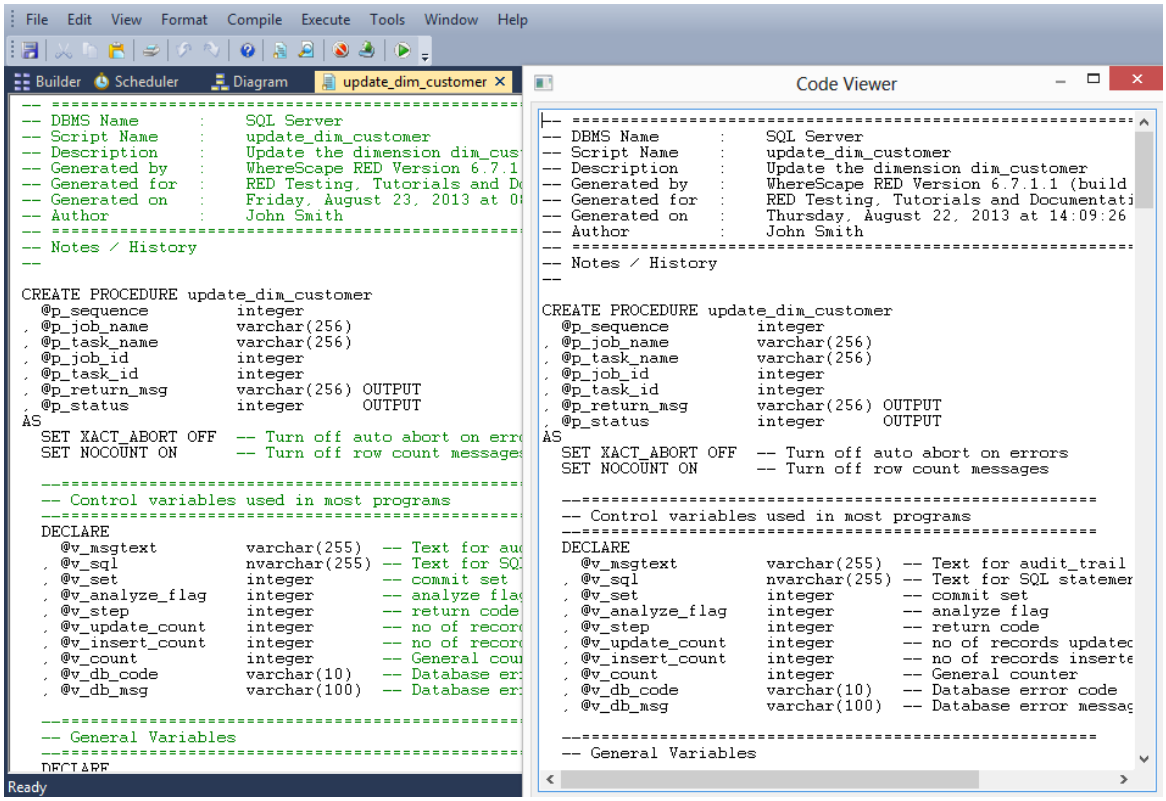


There are a number of drop-down list boxes to choose from. Once an item is selected the viewer loads the code and moves to the right side of the edit window. The various options are:

- Version: A previously saved version of this procedure. The description and time are shown.
- Procedure: Some other procedure stored within the WhereScape metadata.
- Template: A template procedure as defined in the WhereScape metadata.
- Compiled procedure: One of the currently compiled procedures existing in the database.

Once an item is chosen, the viewer appears on the right side of the edit window. The viewer or the main edit window can receive the focus by clicking the mouse within the appropriate window.

Code can be cut from one window and pasted into the other. Any changes made in the viewer window cannot be saved. An example of an active view window is as follows:



NOTE: Editing, deleting or compiling Locked for Edit or opened procedures/scripts
Procedures or scripts cannot be deleted if they are Locked for Edit by any user, checked out by another user or if there is another object that has the same associated procedures or scripts. Saving or Compiling in the procedure or script edit window cannot be performed if the procedures or scripts become Locked for Edit by other users after the edit window was opened. Procedures or scripts cannot be deleted or modified after the edit window has been opened, unless the Edit Lock has been released. Edit Locks can be released by any user in the Script or Procedure Properties screen.

To prevent updates, deletes and modifications to certain procedures or scripts, it is best to use the Check Out functionality instead. For more information about this functionality, please see section ***Check Outs and Check Ins***.

PROCEDURE LOADING AND SAVING

Procedures are normally stored in the WhereScape RED metadata tables. When a procedure is opened within WhereScape RED then the data is retrieved from the meta tables. Likewise, when the procedure is saved it overwrites the existing data in the meta tables.

When a procedure is compiled it is also written to the database system tables. In the case of Oracle, a compiled procedure can be viewed via the 'User_source' view (e.g. select text from user_source where name = 'MY_PROCEDURE' order by line;). For SQL Server, the command is sp_helptext 'MY_PROCEDURE'.

Loading data

As mentioned above when a procedure is opened for editing the information stored in the metadata is loaded. Additional text can be loaded into the procedure edit window by selecting the **File/Insert from File** menu option which allows the browsing and inserting of a PC based file. Also paste buffer data can be inserted in the normal manner. In the previous 'Editing' section the viewer window was discussed. This window can also be a source of data via cut and paste.

Saving the Procedure

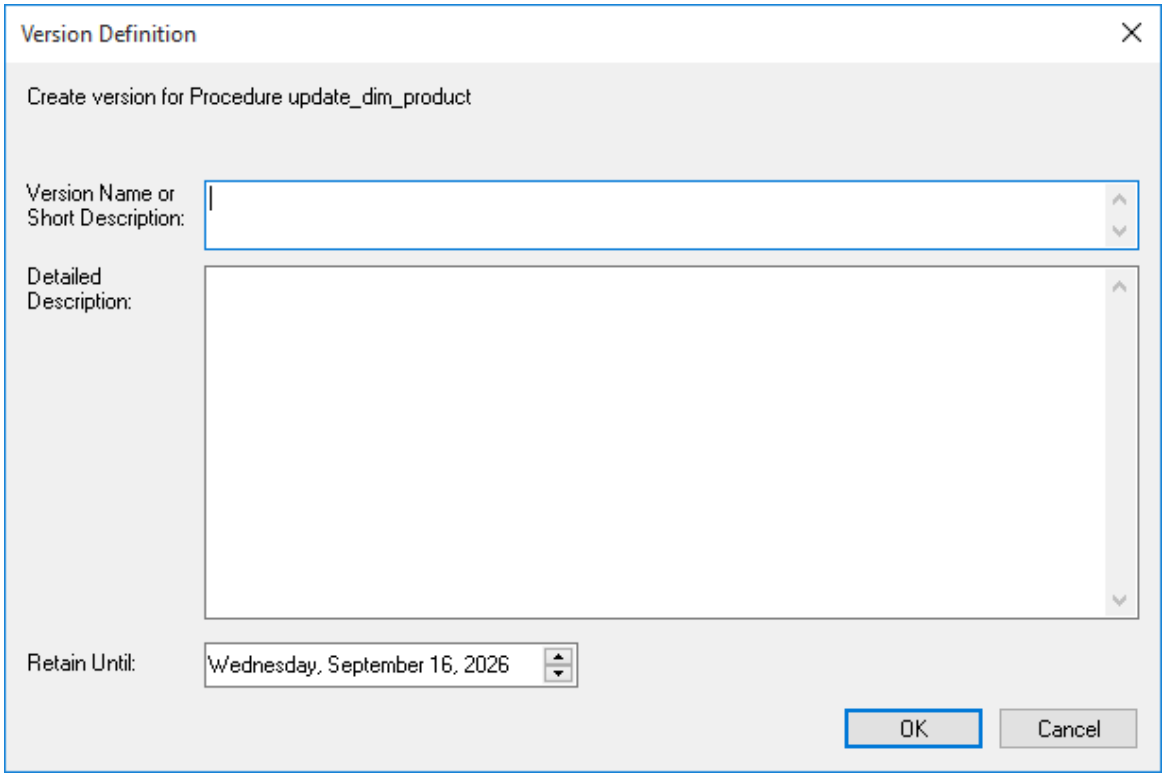
The default **File/Save Procedure** menu option overwrites the existing procedure in the metadata. In addition, a procedure can be saved to a Windows disk by selecting the **File/Write procedure to disk** menu option. All procedures can be written to disk from the main Builder menu option **Backup/Save procedures to disk**. This option allows the selection of a directory. The procedures are then written individually into that directory. All procedures are also written to one text file as part of the **Backup/Export the metadata** menu option.

Versions

Multiple versions of a procedure can be stored. Once a version is created that version may be read but may not be updated. Only the current procedure can be edited. There are a number of ways of creating a version of a procedure. These are:

1. By setting **Auto-Version before Procedure Compile** under **Tools/Options/Versioning.** If set to true, a new version of a procedure will be created whenever the procedure is compiled.
2. The Procedure Editor menu option **File/Save and Version** will save a procedure and create a version at the same time.
3. By selecting the **Version Control/New Version** menu option from the pop-up menu when positioned on a procedure in the main Builder window.
4. By selecting the **Tools/Version All** menu option.

When a version is created via method (2) or (3) above the following screen will appear, to allow the definition of the version. If an auto version is created, then the person creating the version is recorded along with the reason for the version. (e.g. Version on compile, Version on procedure delete)

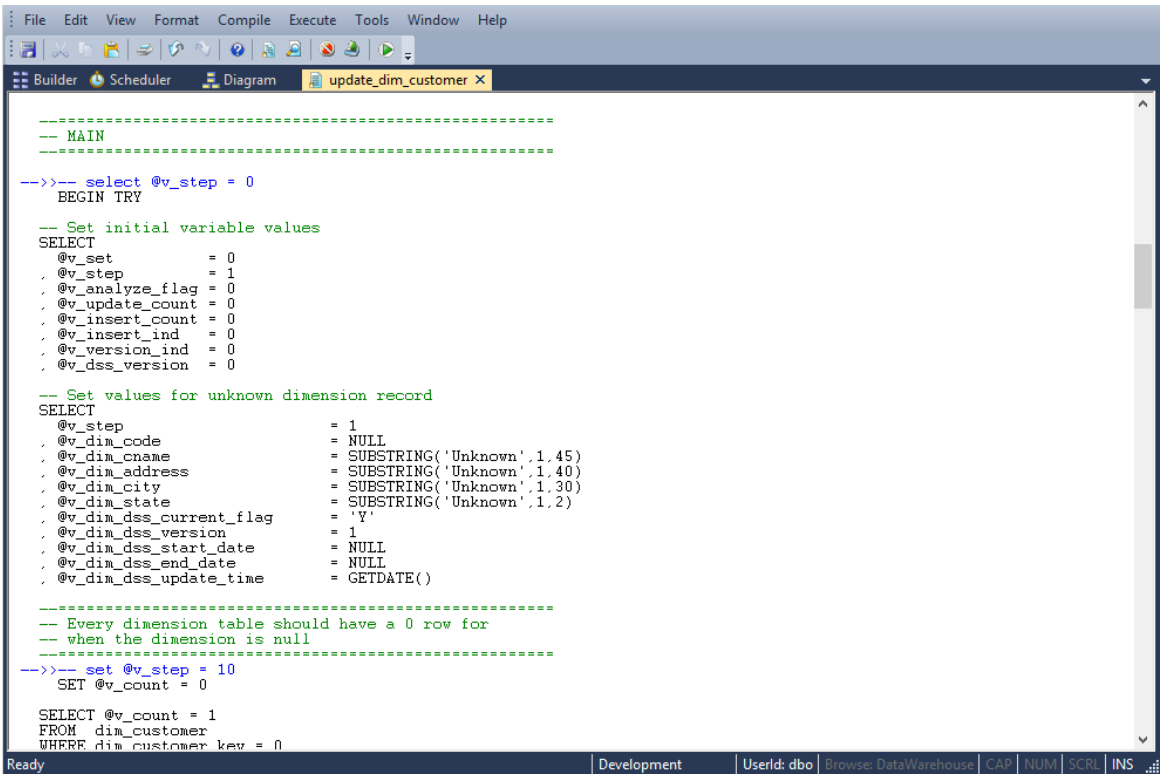


- The version name/description appears when the versions are subsequently browsed.
- The Retain until date is set ten years in the future by default.
- The automated deletion of versions is not supported.

PROCEDURE COMPARISONS

A procedure can be compared to either an earlier version, or to the currently running code as compiled/stored in the database. The menu option **Tools/Compare to Compiled Source** allows the comparison of the procedure being edited with the code currently compiled and running in the database. If a viewer window is open (see the procedure editing section) then the **Tools/Compare to Viewer** menu option will compare the contents of the viewer window with the current code. Therefore, to compare against an older version, we first load the viewer window with the older version and perform a **Compare to Viewer**.

The comparison will highlight the differences, as shown in the example below:

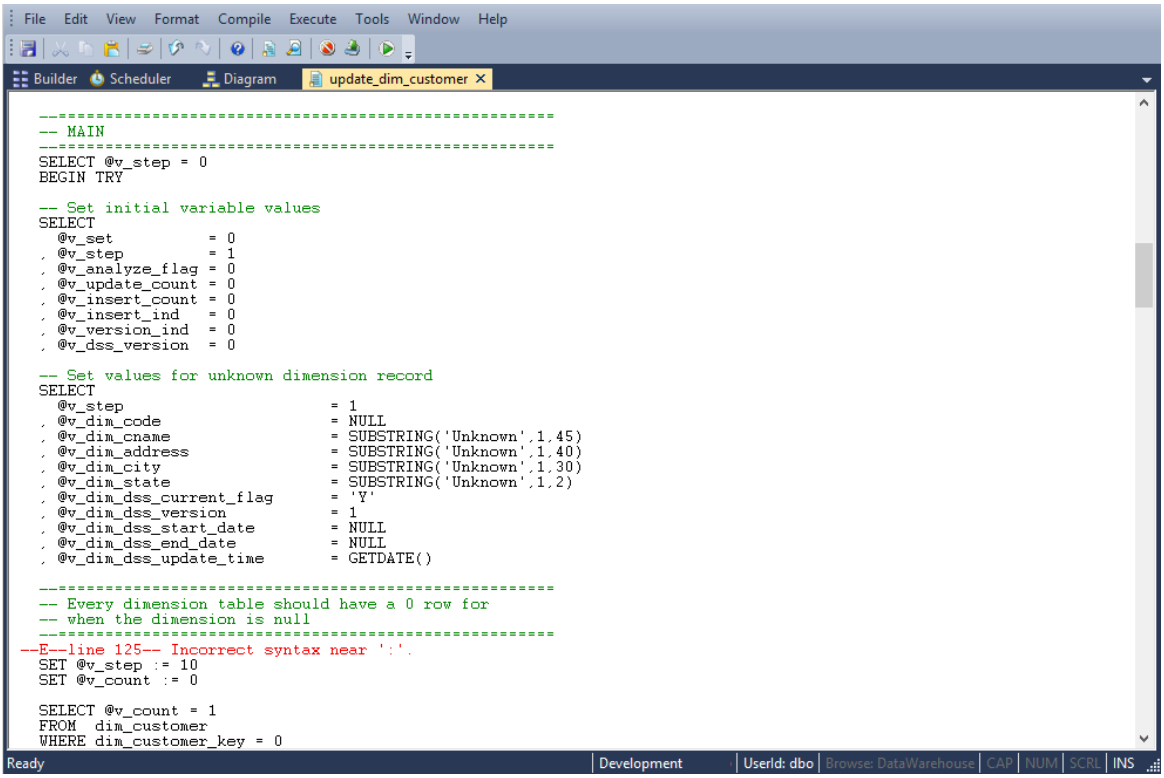


In this example the lines **select @v_step = 0** and **set @v_step = 10** have been removed from the current code in the edit window.

Once the comparison has been completed you can either remove the compare comments or accept the compare changes. The menu option **Tools/Remove Compare Comments** will remove the added blue comments and code. The menu option **Tools/Accept Compare Changes** will implement the changes highlighted. For the above example the line **select @v_step = 0** and **set @v_step = 10** would be added.

PROCEDURE COMPILATION

From within the procedure editor a procedure can be compiled by selecting the menu option **Compile/Compile** or by clicking the **Compile** icon. If the procedure compiles successfully a dialog box will appear notifying of a successful compile. If the compile fails then error message comments will be inserted into the procedure code. In the following example the error messages are in red and begin with --E--.



Error comments will be inserted at each error point. *A compile will delete any previous error comments*. Error comments can also be removed through the menu option **Compile/Delete Error messages.**

Note: In some instances, the error comments may not be positioned on the correct line. This can occur as the result of one or more procedure lines being wrapped. Therefore, ensure the procedure editor window is maximized when dealing with compile errors.

If a procedure fails to compile then it is invalidated in the database, and will not run until successfully compiled.

PROCEDURE RUNNING

Only procedures that conform to the WhereScape scheduler syntax can be executed from within the procedure editor. Select the **Execute/Execute** menu option or click the **Execute** icon to run the procedure. A procedure must have been compiled in order to run.

The results of the procedure will be displayed in a dialog box. The result code and result message will be displayed as well as any additional messages.

PROCEDURE SYNTAX

The procedures managed by the WhereScape scheduler require the following standards. If a function or procedure is being developed that is not called directly by the scheduler then it does not need to conform with this standard. If however such a procedure or function wants to log messages to the audit or error logs then it will need the input parameters included in its parameter list.

Parameters

The procedure must have the following parameters in the following order:

Parameter name	Input/Output	Oracle Type	SQL Server/DB2 Type
p_sequence	Input	Number	Integer
p_job_name	Input	Varchar2	Varchar(256)
p_task_name	Input	Varchar2	Varchar(256)
p_job_id	Input	Number	Integer
p_task_id	Input	Number	Integer
p_return_msg	Output	Varchar2	Varchar(256)
p_status	Output	Number	Integer

The input parameters are passed to the procedure by the scheduler. If the procedure is called outside the scheduler then the normal practice is to pass zero (0) in the sequence, job_id and task_id. A description of the run can be passed in the job name and the task name is typically the name of the procedure.

The output parameters must be populated by the procedure on completion. The return_msg can be any string up to 256 characters long that describes the result of the procedures execution. The status must be one of the following values:

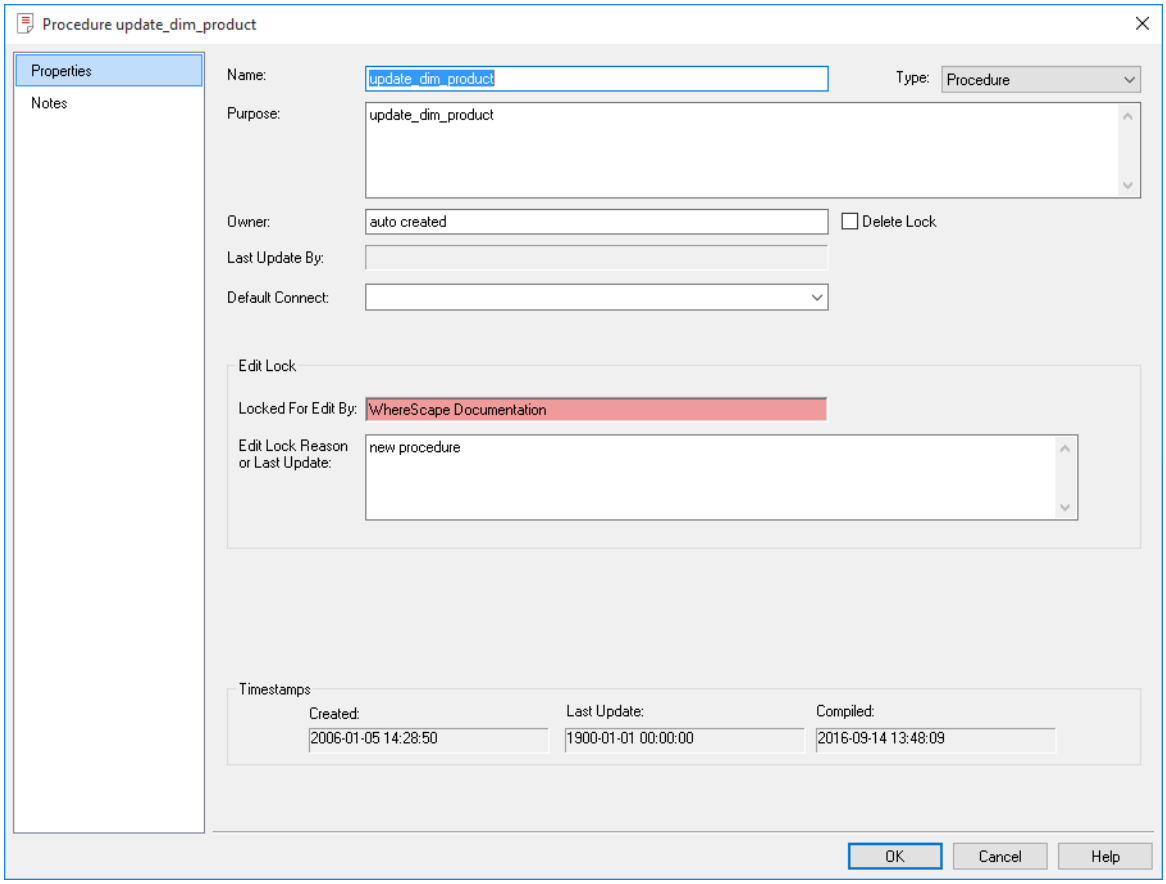
Status	Meaning	Description
1	Success	Normal completion
-1	Warning	Completion with warnings
-2	Error	Hold subsequent tasks dependent on this task
-3	Fatal Error	Hold all subsequent tasks

Note: Multiple SQL statements can be separated using the "end of statement" indicator. This is <EOS> by default but can be configured in **Tools/Options/Code Generation/General**.

PROCEDURE PROPERTIES

The properties screen for procedures and scripts is the same. A procedure can be renamed by changing the name field.

If a procedure is renamed, then it will also be necessary to change the procedure name within the actual code. The purpose and owner fields are purely informational.



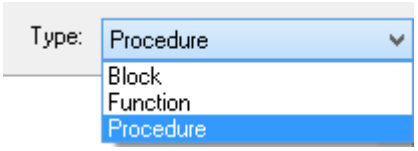
In the example above the **Delete Lock** check-box is not checked. Ticking this check-box will prevent the procedure being deleted through the **Delete** menu option. It will also prevent the procedure being overwritten if a new procedure generation is requested.

Also in the example above the procedure is currently being edited, and is shown as being Locked for Edit by "WhereScape Documentation". If procedures or scripts have already been opened for editing, they can only subsequently be opened for viewing.
Check Out and delete operations for procedures and scripts as well as the regeneration and drop of procedures are not permitted if the object is currently Locked For Edit by another user.

The **Release Edit Lock** button to the right of the edit lock message allows the edit lock to be cleared. If WhereScape RED, the database or the PC crashes when a procedure is open, then the check out will need to be cleared through this screen.

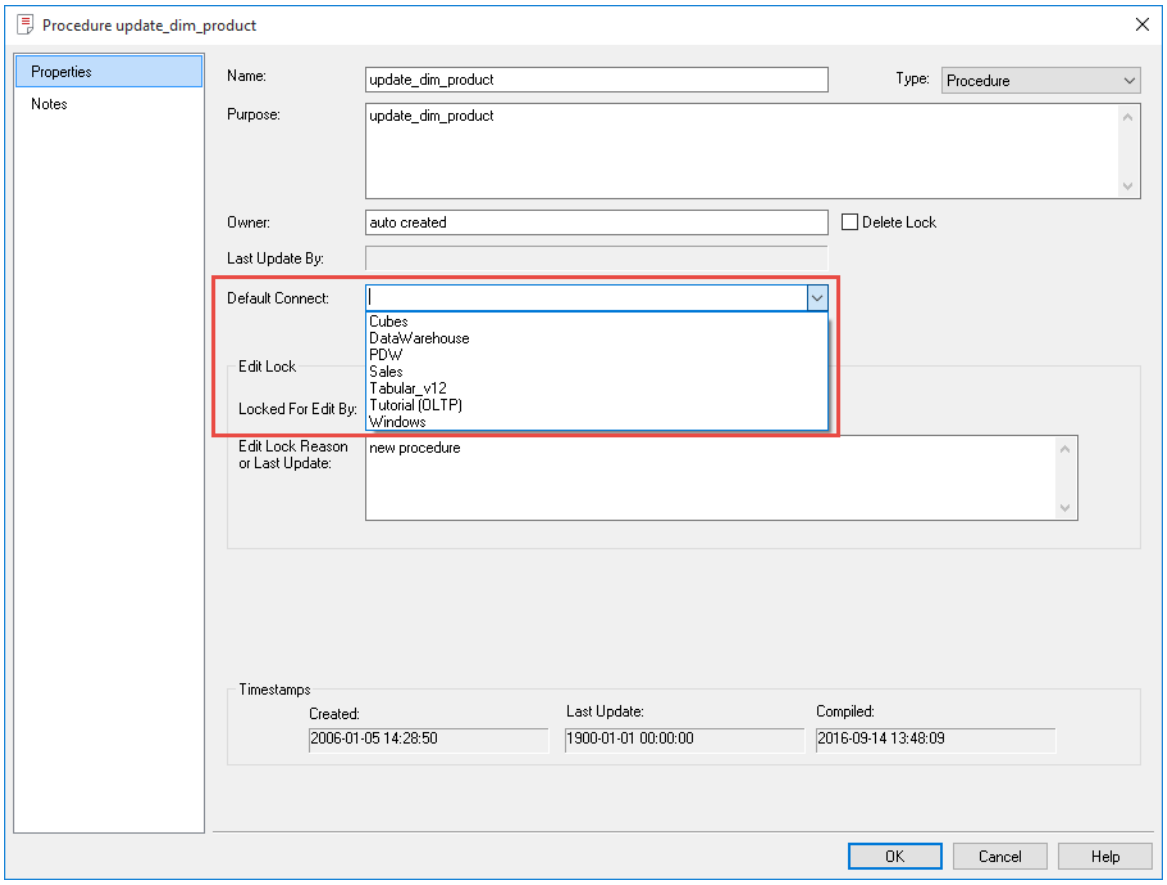
The **Edit Lock Reason** is for information only, and can be used as another comment field if desired.

The **Type** drop-down list allows the selection of **Block**, **Function** or **Procedure**:



Selecting a type of **Block** will allow you to execute a SQL block against another connection.

- An additional field on the Properties screen - **Default Connect** - then allows you to select the connection against which the SQL block will be executed.



SCRIPT GENERATION

WhereScape RED generates template scripts to assist in the loading of textual files from UNIX or Windows. These scripts are generated when a UNIX or Windows file is dragged into a load table target and one of the two 'Script based' file load options is chosen. Typically, script loads are used when some complexity prevents the use of the standard file based load. In such a case the script will probably need modification.

Note: UNIX scripts are only supported for an Oracle data warehouse running on a UNIX platform.

The following sample scripts are covered in the remainder of this section:

1. Windows script for SQL.
2. UNIX script for Oracle.
3. Windows script for Oracle.
4. Windows script for DB2.

SCRIPT GENERATION (WINDOWS/SQL SERVER)

A sample Windows script for SQL Server is as follows. The key components of the script are described below:

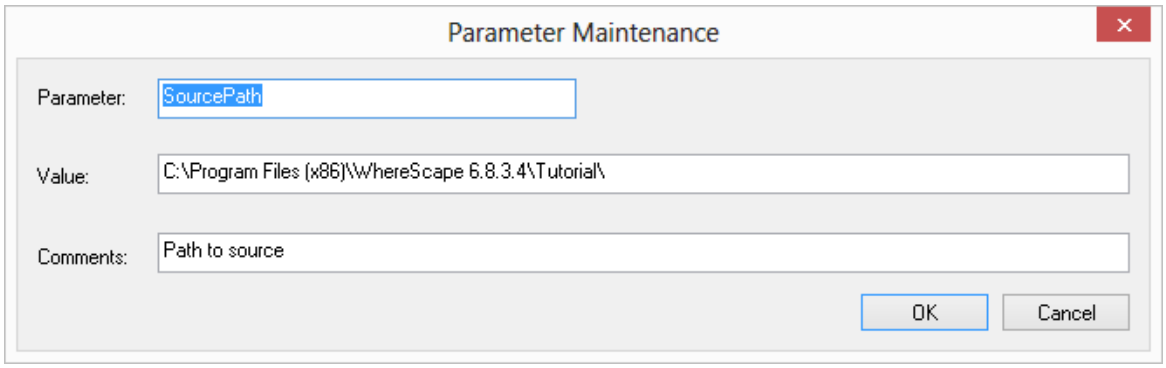
```
@echo off
setlocal enabledelayedexpansion
setlocal enableextensions
REM ************************************************************
REM ***** Load of D:\forecast.txt into load_forecast_txt
REM ************************************************************

REM ************************************************************
REM ***** NOTE: The following environment variables will be set
REM ***** WSL_LOAD_TABLE = the load table name
REM ***** WSL_SEQUENCE = a unique sequence number for the scheduler
REM ***** WSL_WORKDIR = the work directory defined in the connection
REM ***** WSL_SERVER = the server defined in the connection
REM ***** WSL_DATABASE = the database defined in the connection
REM ***** WSL_USER = the dss user defined in the connection
REM ***** WSL_PWD = the dss password defined in the connection
REM ***** WSL_PARAMnnn = Any parameters that start with the load
REM ***** table name. For example: A table called load_abc has a
REM ***** parameter called load_abc_server defined. In this case
REM ***** a variable called WSL_PARAM_SERVER will be created
REM ************************************************************

SET LOAD_FILE=D:\forecast.txt
SET TRIG_FILE=D:\forecast.trg

if defined WSL_LOAD_TABLE (SET LOAD_TABLE=%WSL_LOAD_TABLE%) else SET LOAD_TABLE=load_forecast_txt
if defined WSL_SERVER (SET SERVER=%WSL_SERVER%) else SET SERVER=PMSDEV\PMSDEV
if defined WSL_DATABASE (SET DATABASE=%WSL_DATABASE%) else SET DATABASE=DataWarehouse
if defined WSL_USER (SET USER=%WSL_USER%) else SET USER=sa
if defined WSL_USER (SET PWD=%WSL_PWD%) else SET PWD=9986
if defined WSL_WORKDIR (SET WORKDIR=%WSL_WORKDIR%) else goto LABEL_FILE_DEFAULT
if defined WSL_SEQUENCE (SET SEQUENCE=%WSL_SEQUENCE%) else goto LABEL_FILE_DEFAULT
SET FILECTL=%WORKDIR%\wsl%SEQUENCE%.ctl
SET FILELOG=%WORKDIR%\wsl%SEQUENCE%.log
SET FILEAUD=%WORKDIR%\wsl%SEQUENCE%.aud
goto LABEL_WAIT
:LABEL_FILE_DEFAULT
SET FILECTL=c:\temp\wsl39.ctl
SET FILELOG=c:\temp\wsl39.log
SET FILEAUD=c:\temp\wsl39.aud
:LABEL_WAIT
```

WhereScape RED TIP: Parameters

Parameters can also be added to Scripts to facilitate deployment processes or environment changes without the need to regenerate scripts. Parameters can be added to scripts of Load and Export tables.

For example: add $P<ParameterName>$ to the script where P is the parameter indicator as show below.

Before adding the Parameter to the script, create the desired parameter in **Tools->Parameters->Add Parameter.**




```
SET LOAD_FILE="$PSourcePath$\forecast.txt"
SET TRIG_FILE="$PSourcePath$\forecast.trg"
```

The script makes use of a number of environmental variables. These variables are acquired from both the table and connection properties. These variables are established in the environment by either WhereScape RED or the scheduler. If the script is to be executed outside of WhereScape RED or scheduler control, then these variables will need to be assigned.

The first section of the script defines the variables. The second section provides a timed wait for the load file to arrive. By default, the WAITSECS variable is set to zero, so that no wait occurs. This can be set to a number of seconds that the script is to wait for the file to arrive.

```
REM ************************************************************
REM ************** W A I T     T I M E R ***********************
REM ************************************************************
REM If a wait time specified then loop looking for the file
REM or trigger file to arrive until the wait time expires.
SET /A WAITSECS=0
:LABEL_WAITLOOP
if %WAITSECS% LEQ 0 goto LABEL_FILECHECK
REM see if the file exists and if so skip the wait check
if exist %TRIG_FILE% goto LABEL_FILECHECK
Call WslSleep to wait for 30 seconds. It will return remaining seconds
D:\Pms\Release\WslSleep %WAITSECS% 30
SET /A WAITSECS=%errorlevel%
goto LABEL_WAITLOOP
REM
REM Finished our wait loop. See if we have the file
REM ************************************************************
REM ************** F I L E    C H E C K ************************
REM ************************************************************
:LABEL_FILECHECK
if exist %TRIG_FILE% goto LABEL_TRIGGER
echo -1
echo File %TRIG_FILE% was not found.
exit
REM
```

Once the wait has completed, either through a time expiry or through the location of the file, we check that the file is present, and if not found report back a warning. This warning can be changed to an error by changing the first echo statement to "-2". See the syntax section for more information.

When a trigger file is specified the script looks for a trigger file, and will exit with the specified status if the file is not found. The following code is included if a trigger file is present.

```
REM
REM ************************************************************
REM ************* T RI G G E R ********************************
REM ************************************************************
:LABEL_TRIGGER
REM
REM First clear any existing parameters
REM
isql -S %SERVER% -d %DATABASE% -U%USER% -P%PWD% -n -Q "UPDATE dss_parameter SET dss_parameter_value = NULL WHERE dss_parameter_name like 'FORECAST_%%'" -o %FILELOG%
REM
REM Get the first four and all remaining parameters
REM
for /F "tokens=1,2,3,4* delims=," %%a in (%TRIG_FILE%) do SET A=%%a&SET B=%%b&SET C=%%c&SET D=%%d&SET E=%%e
REM
REM For each parameter write it to the parameter table
REM
if defined A isql -S %SERVER% -d %DATABASE% -U%USER% -P%PWD% -n -Q "EXEC WsParameterWrite 'FORECAST_0','%A%','script load'"
if defined B isql -S %SERVER% -d %DATABASE% -U%USER% -P%PWD% -n -Q "EXEC WsParameterWrite 'FORECAST_1','%B%','script load'"
if defined C isql -S %SERVER% -d %DATABASE% -U%USER% -P%PWD% -n -Q "EXEC WsParameterWrite 'FORECAST_2','%C%','script load'"
if defined D isql -S %SERVER% -d %DATABASE% -U%USER% -P%PWD% -n -Q "EXEC WsParameterWrite 'FORECAST_3','%D%','script load'"
if defined E isql -S %SERVER% -d %DATABASE% -U%USER% -P%PWD% -n -Q "EXEC WsParameterWrite 'FORECAST_4','%E%','script load'"
REM
```

Such a file (trigger) contains control information about the main file to be loaded and arrives after the main file to indicate that the main file transfer has completed and that it is OK to load.

This section loads the contents of the trigger file into the Parameters table, so that the table can be validated. See the section on Post Load procedures for an explanation on how trigger files are used to validate a load file.

```
REM ************************************************************
REM ************* L O A D    T H E   D A T A ******************
REM ************************************************************
echo load of %LOAD_TABLE% > %FILEAUD%
SET /A RESULT_CODE=1
SET RESULT_MSG=Load Completed
:LABEL_LOAD
SET FILE_NAME=NO_MORE_FILES
for %%A in (%LOAD_FILE%) do if "!FILE_NAME!"=="NO_MORE_FILES" SET FILE_NAME=%%A
if %FILE_NAME%==NO_MORE_FILES goto LABEL_EXIT
echo loading %FILE_NAME%  >> %FILEAUD%
echo exit(  > %FILECTL%
echo   BULK INSERT  >> %FILECTL%
echo     %LOAD_TABLE% >> %FILECTL%
echo   FROM  >> %FILECTL%
echo     '%FILE_NAME%' >> %FILECTL%
echo   WITH >> %FILECTL%
echo   ( >> %FILECTL%
echo       DATAFILETYPE='char' >> %FILECTL%
echo     , CODEPAGE='raw' >> %FILECTL%
echo     , FIELDTERMINATOR = ',' >> %FILECTL%
echo     , FIRSTROW = 2 >> %FILECTL%
echo   ) >> %FILECTL%
echo ) >> %FILECTL%
rem
rem In the following isql statement use either -E for a trusted connection
rem or -Uxxx -Pyyy  where xxx is the username and ppp the password
rem
isql -S %SERVER% -d %DATABASE% -U%USER% -P%PWD% -n -i %FILECTL% -o %FILELOG%
SET ERRLEV=%errorlevel%
IF %errorlevel% EQU 0 GOTO LABEL_OKAY
:LABEL_FAIL
echo -2
echo Load failed with return status %ERRLEV% See error log for details
type %FILEAUD% >&2
type %FILELOG% >&2
exit
:LABEL_OKAY
type %FILELOG% >> %FILEAUD%
REM
```

This section calls isql to invoke Bulk Insert to load the file. It makes use of a temporary file to build as a control file and then calls isql to load the data. Note that the load is actually in a for loop. Wild card file names can be used to load multiple files. Each file to be loaded must have the same format.

Note that the data being loaded is appended to the database table. As part of the scheduler run the load table is truncated if the property for truncation is set. In this way multiple files can be loaded into the database table.

TIP: If this script is to be executed outside the control of the WhereScape RED scheduler then a truncate statement may need to be performed on the database load table. This would normally be placed before the 'for loop' and would look something like the following:

```
isql -S %SERVER% -d %DATABASE% -U%USER% -P%PWD% -n -Q "truncate table load_forecast"
-o %FILELOG%
```

This next section handles the rename and potential looping. The first block of code renames the file and also the trigger file if appropriate. This code is only generated if the rename fields in the file attributes are populated.

The goto label_load statement 9 lines from the end can be used if all the files in a wild card file load are required. Simply uncomment this goto statement and the script will load each file in the wild card.

```
REM ************************************************************
REM ************** R E N A M E ********************************
REM ************************************************************
:LABEL_RENAME
REM
REM rename the file
move %FILE_NAME% D:\loaded\
echo %FILE_NAME% moved to D:\loaded\ >> %FILEAUD%
REM rename the trigger file
move %TRIG_FILE% D:\loaded\
echo %TRIG_FILE% moved to D:\loaded\ >> %FILEAUD%
REM
REM ************************************************************
REM ************** N E X T   F I L E **************************
REM ************************************************************
REM If multiple files are being processed via a wildcard
REM uncomment the goto LABEL_LOAD statement
REM WARNING: Do not loop back unless the file name contains a wildcard
REM WARNING: You must rename the file that has been loaded if
REM          looping back otherwise the script will loop for
REM          ever loading the same file.
:LABEL_NEXTFILE
REM goto LABEL_LOAD
REM ************************************************************
REM ************** E X I T ************************************
REM ************************************************************
:LABEL_EXIT
echo %RESULT_CODE%
echo %RESULT_MSG%
type %FILEAUD%
exit
```

SCRIPT GENERATION (UNIX/ORACLE)

A sample UNIX script for Oracle is as follows. The key components of the script are described below:

```
#!/bin/sh
#*******************************************************************************
#****** Load of /home/dssadm/budget.txt into table load_budget_txt
#*******************************************************************************
#
LOAD_FILE=/home/dssadm/budget.txt
#
#*******************************************************************************
#************************* W A I T     T I M E R ***********************
#*******************************************************************************
# If a wait time specified then loop looking for the file or trigger
# to arrive until the wait period expires.
WAITSECS=0
if [ "$WAITSECS" -gt "0" ]
then
  # Get the current time and work out the end of the wait
  #
  NOW_HH=`date +%H`
  NOW_MM=`date +%M`
  NOW_SS=`date +%S`
  let NOW_HH="$NOW_HH * 3600"
  let NOW_MM="$NOW_MM * 60"
  let NOW_TIME="$NOW_HH + $NOW_MM + $NOW_SS"
  let TILL_TIME="$NOW_TIME + $WAITSECS"
  let START_TIME="$NOW_TIME"
  #
  # Loop checking for the file until the wait is complete
  #
  while [ "$NOW_TIME" -lt "$TILL_TIME" ]
  do
    FCOUNT=`find $LOAD_FILE -print 2>/dev/null | wc -l`
    if [ "$FCOUNT" -gt "0" ]
    then
      let NOW_TIME="$TILL_TIME"
    else
      #
      # if the file is not present sleep 30 seconds
      # and recalculate the time. If less than the start
      # then we have gone past midnight so add 86400
      sleep 30
```

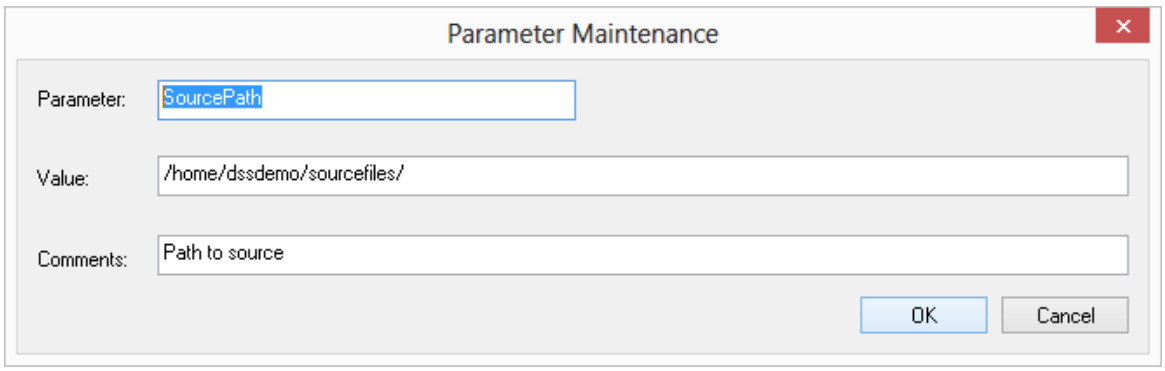
```
      NOW_HH=`date +%H`
      NOW_MM=`date +%M`
      NOW_SS=`date +%S`
      let NOW_HH="$NOW_HH * 3600"
      let NOW_MM="$NOW_MM * 60"
      let NOW_TIME="$NOW_HH + $NOW_MM + $NOW_SS"
      if [ "$NOW_TIME" -lt "$START_TIME" ]
      then
        let NOW_TIME="$NOW_TIME + 86400"
      fi
    fi
  done
fi
#
# Finished our wait loop. See if we have the file
#
#*****************************************************************************
#************************* F I L E   C H E C K  ************************
#*****************************************************************************
FCOUNT=`find $LOAD_FILE -print 2>/dev/null | wc -l`
if [ "$FCOUNT" -eq "0" ]
then
  #
  # file not present so exit with the requested return code
  #
  echo "-1"
  echo "File /home/dssadm/budget.txt was not found"
  exit
fi
```

WhereScape RED TIP: Parameters

Parameters can also be added to Scripts to facilitate deployment processes or environment changes without the need to regenerate scripts. Parameters can be added to scripts of Load and Export tables.

For example: add $P<ParameterName>$ to the script where P is the parameter indicator as shown below.

Before adding the Parameter to the script, create the desired parameter in **Tools->Parameters->Add Parameter.**




```
LOAD_FILE=$PSourcePath$/forecast.txt
TRIG_FILE=$PSourcePath$/forecast.trg
```

The script makes use of two environmental variables called DSS_USER and DSS_PWD. These variables are established in the environment by the scheduler. If the script is to be executed outside of scheduler control, then these two variables will need to be assigned after the LOAD_FILE variable.

The first section of the script defines the load_file variable. The second section provides a timed wait for the load file to arrive. By default, the WAITSECS variable is set to zero, so that no wait occurs. This can be set to a number of seconds that the script is to wait for the file to arrive.

Once the wait has completed, either through a time expiry or through the location of the file, we check that the file is present, and if not found report back a warning. This warning can be changed to an error by changing the first echo statement to "-2". See the syntax section for more information.

When the file based load is used instead of the script based load, the scheduler looks for a trigger file if defined, and will exit with the specified status if the file is not found.

```
#*************************************************************************
#************************ T R I G G E R  *****************************
#*************************************************************************
#
# Clear any existing Trigger file parameters
#
### sqlplus -s <<EOF >/dev/null
### $DSS_USER/$DSS_PWD
### set sqlprompt "";
### set heading off;
### set pagesize 0
### set linesize 256
### set trimspool on
### set echo off;
### update dss_parameter set dss_parameter_value = NULL
### where dss_parameter_name like 'budget.txt_%';
### exit;
### EOF
### if [ "$?" -ne "0" ]
### then
###    echo "-1"
###    echo "Parameter clear Sqlplus returned a non standard return code of $?"
### fi
#
# Load the contents of the trigger file into parameters
#
### cat /home/dssadm/budget.trg | tr "\012" "\012" > /tmp/wsl2.trg
### TRIGNO=`cat /tmp/wsl2.trg | wc -l | tr -d " "`
### if [ "$TRIGNO" != "0" ]
### then
###   ROWNUM=0
###   while [ "$ROWNUM" -lt "$TRIGNO" ]
###   do
###     let ROWNUM="$ROWNUM+1"
###     TRIG_NO="budget.txt_$ROWNUM"
###     TRIG_MSG=`cat /tmp/wsl2.trg | head -$ROWNUM | tail -1 | sed "s/'/''/g"`
###     sqlplus -s <<EOF1 >/dev/null
###     $DSS_USER/$DSS_PWD
###     variable x number;
###     exec :x := WsParameterWrite('$TRIG_NO','$TRIG_MSG','load of /home/dssadm/budget.txt');
###     exit;
### EOF1
###   done
### fi
```

This section of the script is commented out by default. It is used when a trigger file is present. Such a file contains control information about the main file to be loaded and arrives after the main file to indicate that the main file transfer has completed and that it is OK to load.

This section loads the contents of the trigger file into the Parameters table, so that the table can be validated. See the section on Post Load procedures for an explanation on how trigger files are used to validate a load file.

```
#*************************************************************************
#************************ L O A D   T H E   D A T A ***********************
#*************************************************************************
#
# Loop through all files found loading them into the table
# Will need to use append in the sql load in case there are multiple files
#
FLIST=`find $LOAD_FILE -print`
for ACTFILE in $FLIST
do
  # Build the sql loader control script
  echo "load data" >/tmp/wsl2.ctl
  echo "infile '$ACTFILE'" >>/tmp/wsl2.ctl
  echo "badfile '/tmp/wsl2.bad'" >>/tmp/wsl2.ctl
  echo "append" >>/tmp/wsl2.ctl
  echo "into table load_budget_txt" >>/tmp/wsl2.ctl
  echo "fields terminated by \",\"" >>/tmp/wsl2.ctl
  echo "optionally enclosed by '\"'" >>/tmp/wsl2.ctl
  echo "trailing nullcols" >>/tmp/wsl2.ctl
  echo "(" >>/tmp/wsl2.ctl
  echo "order_number," >>/tmp/wsl2.ctl
  echo "order_line_number," >>/tmp/wsl2.ctl
  echo "product_code," >>/tmp/wsl2.ctl
  echo "customer_code," >>/tmp/wsl2.ctl
  echo "budget_quantity," >>/tmp/wsl2.ctl
  echo "budget_sales," >>/tmp/wsl2.ctl
  echo "budget_date date 'yyyy-mm-dd'" >>/tmp/wsl2.ctl
  echo ")" >>/tmp/wsl2.ctl
  ORACLE_SID=0817
  export ORACLE_SID
  sqlldr userid=$DSS_USER/$DSS_PWD control=/tmp/wsl2.ctl skip=1 silent=header,feedback log=/tmp/wsl2.log
  retcode=`echo $?`
  rows=`cat /tmp/wsl2.log| grep Rows | grep success`
```

This section calls sql*loader to load the file. It makes use of a temporary file to build as a control file and then calls sqlldr to load the data. Note that the load is actually in a for loop. Wild card file names can be used to load multiple files. Each file to be loaded must have the same format.

Note that the data being loaded is appended to the database table. As part of the scheduler run the load table is truncated if the property for truncation is set. In this way, multiple files can be loaded into the database table.

TIP: If this script is to be executed outside the control of the WhereScape RED scheduler then a truncate statement may need to be performed on the database load table. This would normally be placed before the 'for loop' and would look something like the following:

```
echo "truncate table load_budget_txt;" >/tmp/wsl2.sql
echo "commit;" >>/tmp/wsl2.sql
echo "exit" >>/tmp/wsl2.sql
sqlplus $DSS_USER/$DSS_PWD @/tmp/wsl2.sql >/dev/null
```

Note that the first echo has a single output '>' whereas the subsequent lines have two '>>'. The first output command creates or re-creates the file and the subsequent lines append to it. We put the output of the command to the null device as we do not want data in the output stream that does not match our syntax.

```
case "$retcode" in
0) echo "1";
   echo "load_budget_txt completed. $rows from $ACTFILE"
   ;;
2) echo "-1";
   echo "Not all rows loaded. $rows. see error trail" ;
   cat /tmp/wsl2.log| grep Rows
   cat /tmp/wsl2.log>&2
   ;;
1) echo "-2";
   echo "SQL*Loader execution exited with EX_FAIL, see error trail";
   cat /tmp/wsl2.log>&2
    ;;
3) echo "-3";
   echo "SQL*Loader execution encountered a fatal error"
   cat /tmp/wsl2.log>&2
   ;;
*) echo "-3";
   echo "SQL*Loader unknown return code"
   cat /tmp/wsl2.log>&2
   ;;
esac
```

This section of the script checks the return code from the sql*loader command. Depending on the code it either identifies success, a warning or an error. In any non success situation it puts the log file out to the error stream '>&2'. This will result in the information ending up in the Detail/Error log when this script is executed through the scheduler.

```
  #
  #*****************************************************************************
  #************************* R E N A M E *****************************************
  #*****************************************************************************
  #
  # If we had a success or a warning then do a rename
  ### if [ "$retcode" -eq "0" -o "$retcode" -eq "2" ]
  ### then
    #
    # Rename the file
    #
  ###   mv $ACTFILE /home/dssadm/budget.txt
  ###   echo "$ACTFILE renamed to /home/dssadm/budget.txt"
    #
    # Rename the trigger
    #
  ###   mv /home/dssadm/budget.trg /home/dssadm/budget.trg
  ###   echo "/home/dssadm/budget.trg renamed to /home/dssadm/budget.trg"
  ### fi
#*******************************************************************************
#************************* E N D    F I L E   L O A D   L O O P ********
#*******************************************************************************
#
# Uncomment the break statement if you only wish to process one file at
# a time in a wildcard file load. The break will case the script to finish
# after the first file that matches the wildcard has been loaded
#
# break
done
exit
```

This final section is largely commented out. It ends with the closing of the for loop that processes each file in a wild card file load.

The first block of commented out code renames the file and also the trigger file if appropriate. If this action is required enter the rename path etc. and un-comment this code.

The break statement 3 lines from the end can be used if just the first file in a wild card file load is required. Simply uncomment this break statement and the script will end after the first file has been loaded.

SCRIPT GENERATION (WINDOWS/ORACLE)

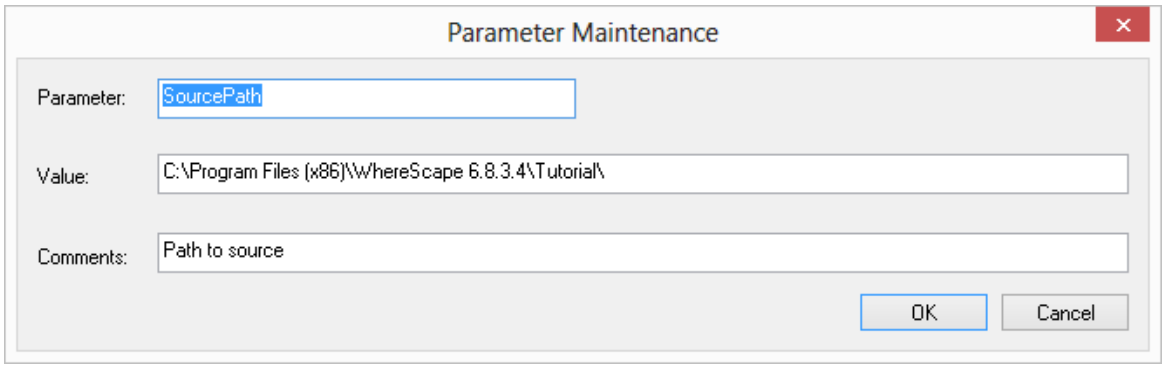
A sample Windows script for Oracle is as follows. The key components of the script are described below:

```
@echo off
setlocal enabledelayedexpansion
setlocal enableextensions
REM ************************************************************
REM ***** Load of D:\forecast.txt into load_forecast_txt
REM ************************************************************

REM ************************************************************
REM ***** NOTE: The following environment variables will be set
REM ***** WSL_LOAD_TABLE = the load table name
REM ***** WSL_SEQUENCE = a unique sequence number for the scheduler
REM ***** WSL_WORKDIR = the work directory defined in the connection
REM ***** WSL_DBHOME = the database home defined in the connection
REM ***** WSL_DATABASE = the database defined in the connection
REM ***** WSL_USER = the dss user defined in the connection
REM ***** WSL_PWD = the dss password defined in the connection
REM ***** WSL_PARAMnnn = Any parameters that start with the load
REM ***** table name. For example: A table called load_abc has a
REM ***** parameter called load_abc_server defined. In this case
REM ***** a variable called WSL_PARAM_SERVER will be created
REM ************************************************************

SET LOAD_FILE=D:\forecast.txt
SET TRIG_FILE=D:\forecast.trg

if defined WSL_LOAD_TABLE (SET LOAD_TABLE=%WSL_LOAD_TABLE%) else SET LOAD_TABLE=load_forecast_txt
if defined WSL_DBHOME (SET DBHOME=%WSL_DBHOME%) else SET DBHOME=
if defined WSL_DATABASE (SET DATABASE=%WSL_DATABASE%) else SET DATABASE=
if defined WSL_USER (SET USER=%WSL_USER%) else SET USER=dssadm
if defined WSL_USER (SET PWD=%WSL_PWD%) else SET PWD=wsl
if defined WSL_WORKDIR (SET WORKDIR=%WSL_WORKDIR%) else goto LABEL_FILE_DEFAULT
if defined WSL_SEQUENCE (SET SEQUENCE=%WSL_SEQUENCE%) else goto LABEL_FILE_DEFAULT
SET FILECMD=%WORKDIR%\wsl%SEQUENCE%.cmd
SET FILECTL=%WORKDIR%\wsl%SEQUENCE%.ctl
SET FILELOG=%WORKDIR%\wsl%SEQUENCE%.log
SET FILEAUD=%WORKDIR%\wsl%SEQUENCE%.aud
SET FILEBAD=%WORKDIR%\wsl%SEQUENCE%.bad
SET FILELO2=%WORKDIR%\wsl%SEQUENCE%.lo2
goto LABEL_WAIT
:LABEL_FILE_DEFAULT
SET FILECMD=c:\temp\wsl22.cmd
SET FILECTL=c:\temp\wsl22.ctl
SET FILELOG=c:\temp\wsl22.log
SET FILEAUD=c:\temp\wsl22.aud
SET FILEBAD=c:\temp\wsl22.bad
SET FILELO2=c:\temp\wsl22.lo2
:LABEL_WAIT
```

WhereScape RED TIP: Parameters

Parameters can also be added to Scripts to facilitate deployment processes or environment changes without the need to regenerate scripts. Parameters can be added to scripts of Load and Export tables.
For example: add $P<ParameterName>$ to the script where P is the parameter indicator as show below.

Before adding the Parameter to the script, create the desired parameter in **Tools->Parameters-> Add Parameter.**




```
SET LOAD_FILE="$PSourcePath$\forecast.txt"
SET TRIG_FILE="$PSourcePath$\forecast.trg"
```

The script makes use of a number of environmental variables. These variables are acquired from both the table and connection properties. These variables are established in the environment by either WhereScape RED or the scheduler. If the script is to be executed outside of WhereScape RED or scheduler control, then these variables will need to be assigned.

The first section of the script defines the variables. The second section provides a timed wait for the load file to arrive. By default, the WAITSECS variable is set to zero, so that no wait occurs. This can be set to a number of seconds that the script is to wait for the file to arrive.

```
REM ************************************************************
REM ************* W A I T     T I M E R ***********************
REM ************************************************************
REM If a wait time specified then loop looking for the file
REM or trigger file to arrive until the wait time expires.
SET /A WAITSECS=0
:LABEL_WAITLOOP
if %WAITSECS% LEQ 0 goto LABEL_FILECHECK
REM see if the file exists and if so skip the wait check
if exist %TRIG_FILE% goto LABEL_FILECHECK
Call WslSleep to wait for 30 seconds. It will return remaining seconds
D:\Pms\Release\WslSleep %WAITSECS% 30
SET /A WAITSECS=%errorlevel%
goto LABEL_WAITLOOP
REM
REM Finished our wait loop. See if we have the file
REM ************************************************************
REM ************* F I L E    C H E C K ************************
REM ************************************************************
:LABEL_FILECHECK
if exist %TRIG_FILE% goto LABEL_TRIGGER
echo -1
echo File %TRIG_FILE% was not found.
exit
```

Once the wait has completed, either through a time expiry or through the location of the file, we check that the file is present, and if not found report back a warning. This warning can be changed to an error by changing the first echo statement to "-2". See the syntax section for more information.

When a trigger file is specified the script looks for a trigger file, and will exit with the specified status if the file is not found. The following code is included if a trigger file is present.

```
REM
REM ************************************************************
REM ************* T RI G G E R ********************************
REM ************************************************************
:LABEL_TRIGGER
REM
REM First clear any existing parameters
REM
echo UPDATE dss_parameter SET dss_parameter_value = NULL > %FILECMD%
echo WHERE dss_parameter_name like 'FORECAST_%%'; >> %FILECMD%
echo variable X varchar2 >> %FILECMD%
REM
REM Get the first four and all remaining parameters
REM
for /F "tokens=1,2,3,4* delims=," %%a in (%TRIG_FILE%) do SET A=%%a&SET B=%%b&SET C=%%c&SET D=%%d&SET E=%%e
REM
REM For each parameter write it to the parameter table
REM
if defined A echo EXEC :X := WsParameterWrite('FORECAST_0','%A%','script load'); >> %FILECMD%
if defined B echo EXEC :X := WsParameterWrite('FORECAST_1','%B%','script load'); >> %FILECMD%
if defined C echo EXEC :X := WsParameterWrite('FORECAST_2','%C%','script load'); >> %FILECMD%
if defined D echo EXEC :X := WsParameterWrite('FORECAST_3','%D%','script load'); >> %FILECMD%
if defined E echo EXEC :X := WsParameterWrite('FORECAST_4','%E%','script load'); >> %FILECMD%
REM
REM Execute the sql command
REM
echo exit; >> %FILECMD%
sqlplus -s %USER%/%PWD%@%DATABASE% @%FILECMD% > %FILELO2% 2>&1
REM
```

Such a file (trigger) contains control information about the main file to be loaded and arrives after the main file to indicate that the main file transfer has completed and that it is OK to load.

This section loads the contents of the trigger file into the Parameters table, so that the table can be validated. See the section on Post Load procedures for an explanation on how trigger files are used to validate a load file.

```
REM ************************************************************
REM ************* L O A D    T H E   D A T A *****************
REM ************************************************************
echo load of %LOAD_TABLE% > %FILEAUD%
SET /A RESULT_CODE=1
SET RESULT_MSG=Load Completed
:LABEL_LOAD
SET FILE_NAME=NO_MORE_FILES
for %%A in (%LOAD_FILE%) do if "!FILE_NAME!"=="NO_MORE_FILES" SET FILE_NAME=%%A
if %FILE_NAME%==NO_MORE_FILES goto LABEL_EXIT
echo loading %FILE_NAME% >> %FILEAUD%
echo load data > %FILECTL%
echo infile '%FILE_NAME%' >> %FILECTL%
echo badfile '%FILEBAD%' >> %FILECTL%
echo into table %LOAD_TABLE% >> %FILECTL%
echo fields terminated by "," >> %FILECTL%
echo optionally enclosed by '^"' >> %FILECTL%
echo trailing nullcols >> %FILECTL%
echo ( >> %FILECTL%
echo  product_code,>> %FILECTL%
echo  customer_code,>> %FILECTL%
echo  forecast_quantity,>> %FILECTL%
echo  forecast_sales_value,>> %FILECTL%
echo  forecast_date date 'dd-mon-yyyy'>> %FILECTL%
echo ) >> %FILECTL%
sqlldr userid=%USER%/%PWD%@%DATABASE% control=%FILECTL% skip=1 silent=header,feedback log=%FILELOG%
SET /A ERRLEV=%errorlevel%
```

This section calls sql*loader (bulk insert for SQL Server) to load the file. It makes use of a temporary file to build as a control file and then calls sqlldr to load the data. Note that the load is actually in a for loop. Wild card file names can be used to load multiple files. Each file to be loaded must have the same format.

Note that the data being loaded is appended to the database table. As part of the scheduler run the load table is truncated if the property for truncation is set. In this way, multiple files can be loaded into the database table.

TIP: If this script is to be executed outside the control of the WhereScape RED scheduler then a truncate statement may need to be performed on the database load table. This would normally be placed before the 'for loop' and for Oracle would look something like the following:

```
echo truncate table load_budget_txt; >/tmp/wsl2.sql
echo commit; >>/tmp/wsl2.sql
echo exit >>/tmp/wsl2.sql
sqlplus %USER%/%PWD%@%DATABASE% @/tmp/wsl2.sql > %FILELO2 2>&1
```

Note that the first echo has a single output '>' whereas the subsequent lines have two '>>'. The first output command creates or re-creates the file and the subsequent lines append to it. We put the output of the command to the null device as we do not want data in the output stream that does not match our syntax.

This next section handles the return status from the load. It returns the appropriate status and messages to the scheduler.

```
if %ERRLEV% GTR 3 goto LABEL_BADRESULT
if %ERRLEV% LSS 0 goto LABEL_BADRESULT
if %ERRLEV% EQU 0 goto LABEL_OKAY
if %ERRLEV% EQU 1 goto LABEL_WARNING
if %ERRLEV% EQU 2 goto LABEL_ERROR
if %ERRLEV% EQU 3 goto LABEL_FATAL
:LABEL_BADRESULT
echo -3
echo SQL*Loader execution encountered an unexpected fatal error
type %FILELOG% >&2
exit
:LABEL_ERROR
echo -2
echo SQL*Loader execution exited with EX_FAIL, see error trail
type %FILELOG% >&2
exit
:LABEL_FATAL
echo -3
echo SQL*Loader execution encountered a fatal error
type %FILELOG% >&2
exit
:LABEL_WARNING
SET /A RESULT_CODE=-1
SET RESULT_MSG=WARNING: Not all rows loaded. See error trail
type %FILELOG% >>&2
:LABEL_OKAY
type %FILELOG% >>&2
REM
```

This final section is as follows:

```
REM ************************************************************
REM ************** R E N A M E **********************************
REM ************************************************************
:LABEL_RENAME
REM
REM rename the file
move %FILE_NAME% D:\old_data
echo %FILE_NAME% moved to D:\old_data >> %FILEAUD%
REM rename the trigger file
move %TRIG_FILE% D:\old_data
echo %TRIG_FILE% moved to D:\old_data >> %FILEAUD%
REM
REM ************************************************************
REM ************** N E X T   F I L E ****************************
REM ************************************************************
REM If multiple files are being processed via a wildcard
REM uncomment the goto LABEL_LOAD statement
REM WARNING: Do not loop back unless the file name contains a wildcard
REM WARNING: You must rename the file that has been loaded if
REM          looping back otherwise the script will loop for
REM          ever loading the same file.
:LABEL_NEXTFILE
REM goto LABEL_LOAD
REM ************************************************************
REM ************** E X I T **************************************
REM ************************************************************
:LABEL_EXIT
echo %RESULT_CODE%
echo %RESULT_MSG%
type %FILEAUD%
exit
```

The first block of code renames the file and also the trigger file if appropriate. This code is only generated if the rename fields in the file attributes are populated.

The goto label_load statement 9 lines from the end can be used if all the files in a wild card file load are required. Simply uncomment this goto statement and the script will load each file in the wild card.

SCRIPT GENERATION (WINDOWS/DB2)

A sample Windows script for DB2 is as follows. The key components of the script are described below:

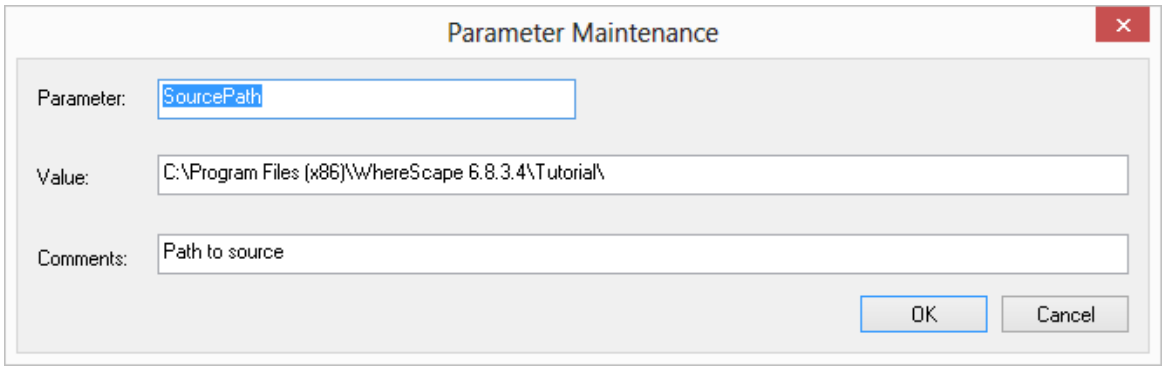
```
@echo off
setlocal enabledelayedexpansion
setlocal enableextensions
REM ************************************************************
REM ***** Load of c:\temp\forecast.txt into load_forecast
REM ************************************************************

REM ************************************************************
REM ***** NOTE: The following environment variables will be set
REM ***** WSL_LOAD_TABLE = the load table name
REM ***** WSL_LOAD_SCHEMA = the load table schema
REM ***** WSL_SEQUENCE = a unique sequence number for the scheduler
REM ***** WSL_WORKDIR = the work directory defined in the connection
REM ***** WSL_SERVER = the server defined in the connection
REM ***** WSL_DATABASE = the database defined in the connection
REM ***** WSL_USER = the dss user defined in the connection
REM ***** WSL_PWD = the dss password defined in the connection
REM ***** WSL_PARAMnnn = Any parameters that start with the load
REM ***** table name. For example: A table called load_abc has a
REM ***** parameter called load_abc_server defined. In this case
REM ***** a variable called WSL_PARAM_SERVER will be created
REM ************************************************************
REM If a wild card is used in the file name then %FILE_NAME%
REM in the control file must be enclosed in single quotes,
REM  otherwise it must not be enclosed in quotes.

SET LOAD_FILE="c:\temp\forecast.txt"
SET TRIG_FILE="c:\temp\forecast.trg"

if defined WSL_LOAD_TABLE (SET LOAD_TABLE=%WSL_LOAD_TABLE%) else SET LOAD_TABLE=load_forecast
if defined WSL_LOAD_SCHEMA (SET LOAD_SCHEMA=%WSL_LOAD_SCHEMA%) else SET LOAD_SCHEMA=RED
if defined WSL_SERVER (SET SERVER=%WSL_SERVER%) else SET SERVER=
if defined WSL_DATABASE (SET DATABASE=%WSL_DATABASE%) else SET DATABASE=RED
if defined WSL_METABASE (SET METABASE=%WSL_METABASE%) else SET METABASE=RED
if defined WSL_USER (SET USER=%WSL_USER%) else SET USER=
if defined WSL_USER (SET PWD=%WSL_PWD%) else SET PWD=
if defined WSL_WORKDIR (SET WORKDIR=%WSL_WORKDIR%) else goto LABEL_FILE_DEFAULT
if defined WSL_SEQUENCE (SET SEQUENCE=%WSL_SEQUENCE%) else goto LABEL_FILE_DEFAULT
SET FILECTL=%WORKDIR%\wsl%SEQUENCE%.ctl
SET FILELOG=%WORKDIR%\wsl%SEQUENCE%.log
SET FILEAUD=%WORKDIR%\wsl%SEQUENCE%.aud
goto LABEL_WAIT
:LABEL_FILE_DEFAULT
SET FILECTL=c:\temp\wsl74.ctl
SET FILELOG=c:\temp\wsl74.log
SET FILEAUD=c:\temp\wsl74.aud
:LABEL_WAIT
```

WhereScape RED TIP: Parameters

Parameters can also be added to Scripts to facilitate deployment processes or environment changes without the need to regenerate scripts. Parameters can be added to scripts of Load and Export tables.

For example: add $P<ParameterName>$ to the script where P is the parameter indicator as show below.

Before adding the Parameter to the script, create the desired parameter in **Tools->Parameters-> Add Parameter.**




```
SET LOAD_FILE="$PSourcePath$\forecast.txt"
SET TRIG_FILE="$PSourcePath$\forecast.trg"
```

The script makes use of a number of environmental variables. These variables are acquired from both the table and connection properties. These variables are established in the environment by either WhereScape RED or the scheduler. If the script is to be executed outside of WhereScape RED or scheduler control then these variables will need to be assigned.

The first section of the script defines the variables. The second section provides a timed wait for the load file to arrive. By default the WAITSECS variable is set to zero, so that no wait occurs. This can be set to a number of seconds that the script is to wait for the file to arrive.

```
REM ************************************************************
REM ************* W A I T     T I M E R ************************
REM ************************************************************
REM If a wait time specified then loop looking for the file
REM or trigger file to arrive until the wait time expires.
SET /A WAITSECS=0
:LABEL_WAITLOOP
if %WAITSECS% LEQ 0 goto LABEL_FILECHECK
REM see if the file exists and if so skip the wait check
if exist %TRIG_FILE% goto LABEL_FILECHECK
REM Call WslSleep to wait for 30 seconds. It will return remaining seconds
"C:\Program Files\WhereScape\WslSleep" %WAITSECS% 30
SET /A WAITSECS=%errorlevel%
goto LABEL_WAITLOOP
REM
REM Finished our wait loop. See if we have the file
REM ************************************************************
REM ************* F I L E    C H E C K ************************
REM ************************************************************
:LABEL_FILECHECK
if exist %TRIG_FILE% goto LABEL_TRIGGER
echo -1
echo File %TRIG_FILE% was not found.
exit
```

Once the wait has completed, either through a time expiry or through the location of the file, we check that the file is present, and if not found report back a warning. This warning can be changed to an error by changing the first echo statement to "-2". See the syntax section for more information.

When a trigger file is specified the script looks for a trigger file, and will exit with the specified status if the file is not found. The following code is included if a trigger file is present.

```
REM
REM ********************************************************
REM ************* T R I G G E R ****************************
REM ********************************************************
:LABEL_TRIGGER
REM
REM First clear any existing parameters
REM
REM
SET SQL=UPDATE %METABASE%.dss_parameter SET dss_parameter_value = NULL WHERE dss_parameter_name like 'FORECAST_%%';
echo %SQL% > %FILECTL%
REM
REM Get the first four and all remaining parameters
REM
for /F "tokens=1,2,3,4* delims=," %%a in ('type %TRIG_FILE%') do SET A=%%a&SET B=%%b&SET C=%%c&SET D=%%d&SET E=%%e
REM
REM For each parameter write it to the parameter table
REM
if defined A SET SQL=CALL %METABASE%.WsParameterWrite ('FORECAST_0','%A%','script load');
if defined A echo %SQL% >> %FILECTL%
if defined B SET SQL=CALL %METABASE%.WsParameterWrite ('FORECAST_1','%B%','script load');
if defined B echo %SQL% >> %FILECTL%
if defined C SET SQL=CALL %METABASE%.WsParameterWrite ('FORECAST_2','%C%','script load');
if defined C echo %SQL% >> %FILECTL%
if defined D SET SQL=CALL %METABASE%.WsParameterWrite ('FORECAST_3','%D%','script load');
if defined D echo %SQL% >> %FILECTL%
if defined E SET SQL=CALL %METABASE%.WsParameterWrite ('FORECAST_4','%E%','script load');
if defined E echo %SQL% >> %FILECTL%
rem
db2batch -d %DATABASE% -f %FILECTL% -r %FILELOG% -time off -o p 0
SET ERRLEV=%errorlevel%
```

Such a file (trigger) contains control information about the main file to be loaded and arrives after the main file to indicate that the main file transfer has completed and that it is OK to load.

This section loads the contents of the trigger file into the Parameters table, so that the table can be validated. See the section on Post Load procedures for an explanation on how trigger files are used to validate a load file.

```
REM ************************************************************
REM ************* L O A D    T H E   D A T A ******************
REM ************************************************************
echo load of %LOAD_TABLE% > %FILEAUD%
SET /A RESULT_CODE=1
SET RESULT_MSG=Load Completed
:LABEL_LOAD
SET FILE_NAME=NO_MORE_FILES
for %%A in (%LOAD_FILE%) do if "!FILE_NAME!"=="NO_MORE_FILES" SET FILE_NAME=%%A
if "!FILE_NAME!"=="NO_MORE_FILES" goto LABEL_EXIT
echo loading %FILE_NAME%  >> %FILEAUD%
echo CALL SYSPROC.ADMIN_CMD('LOAD FROM %FILE_NAME%  OF DEL > %FILECTL%
echo MODIFIED BY COLDEL, timestampformat="DD-MMM-YYYY" >> %FILECTL%
echo METHOD P (1, 2, 3, 4, 5) >> %FILECTL%
echo MESSAGES ON SERVER INSERT INTO %LOAD_SCHEMA%.%LOAD_TABLE%( >> %FILECTL%
echo product_code >> %FILECTL%
echo , customer_code >> %FILECTL%
echo , forecast_quantity >> %FILECTL%
echo , forecast_sales_value >> %FILECTL%
echo , forecast_date >> %FILECTL%
echo ) >> %FILECTL%
echo NONRECOVERABLE'); >> %FILECTL%
rem
rem In the following sql statement no username/password is required for a trusted connection
rem or -axxx/yyy  where xxx is the username and ppp the password. The slash is required
rem
db2batch -d %DATABASE% -f %FILECTL% -r %FILELOG% -time off -o p 0
SET ERRLEV=%errorlevel%
```

This section calls the DB2batch command to load the file. It makes use of a temporary file to build as a control file which includes a call to the LOAD statement. It then runs the temporary files using DB2batch to load the data. Note that the load is actually in a for loop. Wild card file names can be used to load multiple files. Each file to be loaded must have the same format.

Note that the data being loaded is appended to the database table. As part of the scheduler run the load table is truncated if the property for truncation is set. In this way, multiple files can be loaded into the database table.

This next section handles the return status from the load. It returns the appropriate status and messages to the scheduler.

```
SET ERRLEV=%errorlevel%
IF %errorlevel% EQU 0 GOTO LABEL_OKAY
:LABEL_FAIL
echo -2
echo Load failed with return status %ERRLEV% See error log for details
type %FILEAUD% >&2
type %FILELOG% >&2
exit
:LABEL_OKAY
type %FILELOG% >> %FILEAUD%
```

This final section is as follows:

```
REM ************************************************************
REM ************* R E N A M E ********************************
REM ************************************************************
:LABEL_RENAME
rem
SET YYYY=%DATE:~-4%
SET MM=%DATE:~-10,2%
SET DD=%DATE:~-7,2%
rem rename the file
move %FILE_NAME% "c:\temp\loadedforecast_%YYYY%%MM%%DD%.txt" >> %FILEAUD%
rem
SET ERRLEV=%errorlevel%
rem
IF %ERRLEV% EQU 0 GOTO LABEL_MOVE3
echo -3
echo %FILE_NAME% NOT MOVED >> %FILEAUD%
exit
GOTO LABEL_AFTER_MOVE3
:LABEL_MOVE3
rem
echo %FILE_NAME% moved to c:\temp\loadedforecast_%YYYY%%MM%%DD%.txt >> %FILEAUD%
:LABEL_AFTER_MOVE3
rem
rem rename the trigger file
move %TRIG_FILE% "c:\temp\loadedforecast_%YYYY%%MM%%DD%.trg" >> %FILEAUD%
rem
SET ERRLEV=%errorlevel%
IF %ERRLEV% EQU 0 GOTO LABEL_MOVE4
echo -3
rem
echo %TRIG_FILE% NOT MOVED >> %FILEAUD%
exit
GOTO LABEL_AFTER_MOVE4
:LABEL_MOVE4
rem
echo %TRIG_FILE% moved to c:\temp\loadedforecast_%YYYY%%MM%%DD%.trg >> %FILEAUD%
:LABEL_AFTER_MOVE4
REM
REM ************************************************************
REM ************* N E X T   F I L E **************************
REM ************************************************************
REM If multiple files are being processed via a wildcard
REM uncomment the goto LABEL_LOAD statement
REM WARNING: Do not loop back unless the file name contains a wildcard
REM WARNING: You must rename the file that has been loaded if
REM          looping back otherwise the script will loop for
REM          ever loading the same file.
:LABEL_NEXTFILE
REM goto LABEL_LOAD
REM ************************************************************
REM ************* E X I T ************************************
REM ************************************************************
:LABEL_EXIT
echo %RESULT_CODE%
echo %RESULT_MSG%
type %FILEAUD%
exit
```

The first block of code renames the file and also the trigger file if appropriate. This code is only generated if the rename fields in the file attributes are populated.

The goto label_load statement 9 lines from the end can be used if all the files in a wild card file load are required. Simply uncomment this goto statement and the script will load each file in the wild card.

SCRIPT EDITING

WhereScape RED includes a script editor which allows the maintenance of any host scripts within the data warehouse. The editor is invoked by double-clicking on a script name in the left pane or by right-clicking on the script name and selecting **Edit the Script**.

Indenting code

The tab character inserts four spaces into the text. A shift/tab removes four spaces.

Cut, Copy, Paste and Delete

The normal Windows cut, copy, paste and delete functions are available either through the toolbar or via the right mouse popup menu.

Indenting a block of text

A selected block of text can be indented by four spaces by depressing the tab character. Each tab will indent by a further four spaces. A shift/tab will remove four spaces from the front of each line in the selected block.

Viewing other scripts

During the editing process, it is possible to pop up a window containing other scripts. This window allows cut and paste operations. In such a way, it can be used as a work area or as a source of code. Select the **Tools/View Script or Template** menu option to bring the viewer window up. A dialog will appear.

A number of drop-down list boxes can be chosen from. Once an item is selected the viewer loads the code and moves to the right side of the edit window. The various options are:

- Version: A previously saved version of this script. The description and time are shown.
- Script: Some other script stored within the WhereScape metadata.
- Template: A template script as defined in the WhereScape metadata.

Once an item is chosen, the viewer appears on the right side of the edit window. The viewer or the main edit window can receive the focus by clicking the mouse within the appropriate window. Code can be cut from one window and pasted into the other. Any changes made in the viewer window cannot be saved.

NOTE: Editing, deleting or compiling Locked for Edit or opened procedures/scripts
Procedures or scripts cannot be deleted if they are Locked for Edit by any user, checked out by another user or if there is another object that has the same associated procedures or scripts. Saving or Compiling in the procedure or script edit window cannot be performed if the procedures or scripts become Locked for Edit by other users after the edit window was opened. Procedures or scripts cannot be deleted or modified after the edit window has been opened, unless the Edit Lock has been released. Edit Locks can be released by any user in the Script or Procedure Properties screen.

To prevent updates, deletes and modifications to certain procedures or scripts, it is best to use the Check Out functionality instead. For more information about this functionality, please see section ***Check Outs and Check Ins***.

SCRIPT TESTING

When a host script is scheduled, it is run in the scheduler environment. Therefore, a UNIX scheduler must be available to run a UNIX script and only a Windows scheduler can run a Windows script.

It is possible to test a script interactively. For a Windows script this is achieved by running the script on the current PC. For a UNIX script WhereScape RED will attempt to start a telnet window to the UNIX host (as defined in the default connection for the script) and run the script within this telnet window.

A script is invoked via the **Execute/Execute the Script** menu option. The output from the script is shown in a pop-up dialog box.

UNIX Scripts

UNIX scripts have the additional execution option being the menu option **Execute/ Execute and retain**. This option will create a Telnet window to the UNIX host, execute the script, display the results and leave the Telnet window up. Additional commands or a review of what occurred can then be undertaken directly in the Telnet window. When completed the Telnet window should be closed to return to the script editor.

Note: A connection must be correctly set-up to support UNIX script testing. See the **Installation and Administration Guide** and the section on UNIX connections.

SCRIPT SYNTAX

There are a number of conventions that must be followed if a host script is to be used by the WhereScape scheduler. These conventions are:

1. The first line of data in 'standard out' must contain the resultant status of the script. Valid values are '1' to indicate success, '-1' to indicate a warning condition occurred but the result is considered a success, '-2' to indicate a handled error occurred and subsequent dependent tasks should be held, -3 to indicate an unhandled Failure and that subsequent dependent tasks should be held.
2. The second line of data in 'standard out' must contain a resultant message of no more than 256 characters.
3. Any subsequent lines in 'standard out' are considered informational and are recorded in the audit trail. The normal practice is to place a minimum of information in the audit trail. All bulk information should be output to 'standard error'.
4. Any data output to 'standard error' will be written to the error/detail log. Both the audit log and detail log can be viewed from the WhereScape RED tool under the scheduler window.

Windows Example for Oracle:

In the following example the first line '@echo off' prevents unwanted information from being reported to standard out. A Sql*loader script file is built up (echo statements). The sqlldr command is then executed with the 'silent-header,feedback' option once again to prevent unwanted output.

```
@echo off
echo load data >c:\temp\wsl1.ctl
echo infile 'C:\Temp\budget.txt' >>c:\temp\wsl1.ctl
echo badfile 'c:\temp\wsl1.bad' >>c:\temp\wsl1.ctl
echo into table load_budget_txt >>c:\temp\wsl1.ctl
echo fields terminated by "," >>c:\temp\wsl1.ctl
echo optionally enclosed by '^"' >>c:\temp\wsl1.ctl
echo trailing nullcols >>c:\temp\wsl1.ctl
echo ( >>c:\temp\wsl1.ctl
echo  product_code,>>c:\temp\wsl1.ctl
echo  customer_code,>>c:\temp\wsl1.ctl
echo  budget_quantity,>>c:\temp\wsl1.ctl
echo  budget_sales_value,>>c:\temp\wsl1.ctl
echo  budget_date date 'dd-mon-yyyy'>>c:\temp\wsl1.ctl
echo ) >>c:\temp\wsl1.ctl
sqlldr userid=dssadm/wsl@ORCL control=c:\temp\wsl1.ctl skip=1
silent=header,feedback log=c:\temp\wsl1.log
goto answer%errorlevel%
:answer0
echo 1
echo Load completed successfully
type c:\temp\wsl1.log >&2
exit
:answer2
echo -1
echo Not all rows loaded.  See error trail
type c:\temp\wsl1.log >&2
exit
:answer1
echo -2
echo SQL*Loader execution exited with EX_FAIL, see error trail
type c:\temp\wsl1.log >&2
exit
:answer3
echo -3
echo SQL*Loader execution encountered a fatal error
type c:\temp\wsl1.log >&2
```

exit

SCRIPT ENVIRONMENT VARIABLES

The following environment variables are available for all script loads and script exports, both Windows and UNIX/Linux.

All load scripts

The following variables are available in all load scripts:

<table>
<tr><th>Windows variable</th><th>UNIX/Linux variable</th><th>Description</th></tr>
<tr><td>WSL_LOAD_FULLNAME</td><td>LOAD_FULLNAME</td><td>The fully-qualified load table name.</td></tr>
<tr><td>WSL_LOAD_TABLE</td><td>LOAD_TABLE</td><td>The unqualified load table name.</td></tr>
<tr><td>WSL_LOAD_SCHEMA</td><td>LOAD_SCHEMA</td><td>The schema for the load table.
Note: A trailing dot is appended for SQL Server or Oracle due to historical usage.
A trailing dot is not appended for any other database type due to historical usage.
However, it is better not to assume the trailing dot is or isn't appended by using the variable like this, when it is not empty:
<table>
<tr><th>OS</th><th>If no trailing dot is wanted</th><th>If a trailing dot is wanted</th></tr>
<tr><td>Windows</td><td>!WSL_LOAD_SCHEMA:.=!</td><td>!WSL_LOAD_SCHEMA:.=!.</td></tr>
<tr><td>UNIX/ Linux</td><td>${LOAD_SCHEMA%.}</td><td>${LOAD_SCHEMA%.}.</td></tr>
</table></td></tr>
<tr><td>WSL_LOAD_DB</td><td>LOAD_DB</td><td>The name of the database for the load table.</td></tr>
<tr><td>WSL_TEMP_DB</td><td>TEMP_DB</td><td>Teradata: The name of the database for load temporary tables.
PDW: The name of the staging database for the load.
Others: Not Used.</td></tr>
<tr><td>WSL_TGT_DSN</td><td>TGT_DSN</td><td>The ODBC data source name (DSN) for the load table's storage connection.</td></tr>
<tr><td>WSL_TGT_SERVER</td><td>TGT_SERVER</td><td>The server for the load table's storage connection.</td></tr>
<tr><td>WSL_TGT_DBPORT</td><td>TGT_DBPORT</td><td>The database port for the load table's storage connection.</td></tr>
<tr><td>WSL_TGT_DBID</td><td>TGT_DBID</td><td>The Database ID property of the load table's storage connection.</td></tr>
</table>

Windows variable	UNIX/Linux variable	Description
		For Teradata this is the Teradata Director Program ID (TDPID). **For Oracle** this is the Oracle SID or TNS Name.
WSL_TGT_USER	TGT_USER	The user id for the load table's storage connection.
WSL_TGT_PWD	TGT_PWD	The password for the load table's storage connection.

All load scripts from Database or ODBC connections

In **addition** to the variables in the previous table, the following variables are available in all load scripts from Database or ODBC connections:

Windows variable	UNIX/Linux variable	Description
WSL_SRC_DSN	SRC_DSN	The ODBC data source name (DSN) for the source connection.
WSL_SRC_SERVER	SRC_SERVER	The server for the source connection.
WSL_SRC_DBPORT	SRC_DBPORT	The database port for the source connection.
WSL_SRC_DBID	SRC_DBID	The *Database ID* property of the source connection. **For Teradata** this is the Teradata Director Program ID (TDPID). **For Oracle** this is the Oracle SID or TNS Name.
WSL_SRC_DB	SRC_DB	The name of the database for the source connection.
WSL_SRC_SCHEMA	SRC_SCHEMA	The Source Schema property of the load. **Note:** The property is fetched without modification, so there may or may not be a trailing dot depending on how it is configured. However, it is better not to assume the trailing dot is or isn't appended by using the variable like this, when it is not empty:
WSL_SRC_USER	SRC_USER	The user id for the source connection.
WSL_SRC_PWD	SRC_PWD	The password for the source connection.

OS	If no trailing dot is wanted	If a trailing dot is wanted
Windows	!WSL_SRC_SCHEMA:.=!	!WSL_SRC_SCHEMA:.=!.
UNIX/ Linux	${SRC_SCHEMA%.}	${SRC_SCHEMA%.}.

All export scripts

The following variables are available in all export scripts:

Windows variable	UNIX/Linux variable	Description
WSL_EXP_NAME	EXP_NAME	The export object name.
WSL_EXP_FULLNAME	EXP_FULLNAME	The fully-qualified export table name.
WSL_EXP_TABLE	EXP_TABLE	The unqualified export table name. **Note:** For Windows script exports from PDW, this variable is initialized with the fully-qualified export table name, due to historical usage. To enable this variable to be uniformly described and used as the unqualified export table name, an additional variable WSL_EXP_SIMPLENAME is created. This allows the following command to be explicitly added to the top of the script by the script author, to be executed before all other processing: `if defined WSL_EXP_SIMPLENAME (SET WSL_EXP_TABLE=!WSL_EXP_SIMPLENAME!) else SET WSL_EXP_TABLE=` After such a command is executed, the variable WSL_EXP_TABLE will contain the unqualified export table name.
WSL_EXP_SCHEMA	EXP_SCHEMA	The schema for the export table. **Note:** A trailing dot is appended for SQL Server or Oracle due to historical usage. A trailing dot is not appended for any other database type due to historical usage. However, it is better not to assume the trailing dot is or isn't appended by using the variable like this, when it is not empty:

OS	If no trailing dot is wanted	If a trailing dot is wanted
Windows	`!WSL_EXP_SCHEMA:.=!`	`!WSL_EXP_SCHEMA:.=!.`
UNIX/ Linux	`${EXP_SCHEMA%.}`	`${EXP_SCHEMA%.}.`

WSL_EXP_DB	EXP_DB	The name of the database for the export table.
WSL_TEMP_DB	TEMP_DB	**Teradata:** The name of the database for export temporary tables. **Others:** Not used.

Windows variable	UNIX/Linux variable	Description
WSL_SRC_DSN	SRC_DSN	The ODBC data source name (DSN) for the export table's storage connection.
WSL_SRC_SERVER	SRC_SERVER	The server for the export table's storage connection.
WSL_SRC_DBPORT	SRC_DBPORT	The database port for the export table's storage connection.
WSL_SRC_DBID	SRC_DBID	The *Database ID* property of the export table's storage connection. **For Teradata** this is the Teradata Director Program ID (TDPID). **For Oracle** this is the Oracle SID or TNS Name.
WSL_SRC_USER	SRC_USER	The user id for the export table's storage connection.
WSL_SRC_PWD	SRC_PWD	The password for the export table's storage connection.

All scripts

In **addition** to the specific variables in the previous tables, the following variables are available in all scripts:

Windows variable	UNIX/Linux variable	Description
WSL_META_DSN	META_DSN	The ODBC data source name (DSN) for the meta-repository connection.
WSL_META_SERVER	META_SERVER	The server for the meta-repository connection.
WSL_META_DBID	META_DBID	The *Database ID* property of the meta-repository connection. **For Teradata** this is the Teradata Director Program ID (TDPID). **For Oracle** this is the Oracle SID or TNS Name.
WSL_META_DB	META_DB	The name of the database for the meta-repository connection.

<table>
<tr><th>Windows variable</th><th>UNIX/Linux variable</th><th>Description</th></tr>
<tr><td>WSL_META_SCHEMA</td><td>META_SCHEMA</td><td>The meta-repository table qualifier, with a trailing dot.
For SQL Server and DB2 this is the schema for the meta-repository.
For Teradata and Oracle this is not actually a schema name.
Note: A trailing dot is appended due to historical usage.
However, it is better not to assume the trailing dot is or isn't appended by using the variable like this, when it is not empty:
<table>
<tr><th>OS</th><th>If no trailing dot is wanted</th><th>If a trailing dot is wanted</th></tr>
<tr><td>Windows</td><td>!WSL_META_SCHEMA:.=!</td><td>!WSL_META_SCHEMA:.=!.</td></tr>
<tr><td>UNIX/ Linux</td><td>${META_SCHEMA%.}</td><td>${META_SCHEMA%.}.</td></tr>
</table>
</td></tr>
<tr><td>WSL_META_USER</td><td>META_USER</td><td>The user id for the meta-repository connection.</td></tr>
<tr><td>WSL_META_PWD</td><td>META_PWD</td><td>The password for the meta-repository connection.</td></tr>
<tr><td>WSL_WORKDIR</td><td>WORKDIR</td><td>Windows: The work directory defined in the Windows connection.
UNIX/Linux: The work directory defined in the UNIX/Linux or Hadoop connection.</td></tr>
<tr><td>WSL_SEQUENCE</td><td>SEQ</td><td>A unique sequence number for the load or export task.</td></tr>
<tr><td>WSL_PARAMnnn</td><td>PARAMnnn</td><td>Any parameters that start with the load table or export object name.
Example:
A table called load_abc has a parameter called load_abc_server defined.
In this case, a variable called WSL_PARAM_SERVER (Windows) or PARAM_SERVER (UNIX/Linux) will be created.</td></tr>
</table>

CALLING A BATCH FILE FROM A SCRIPT

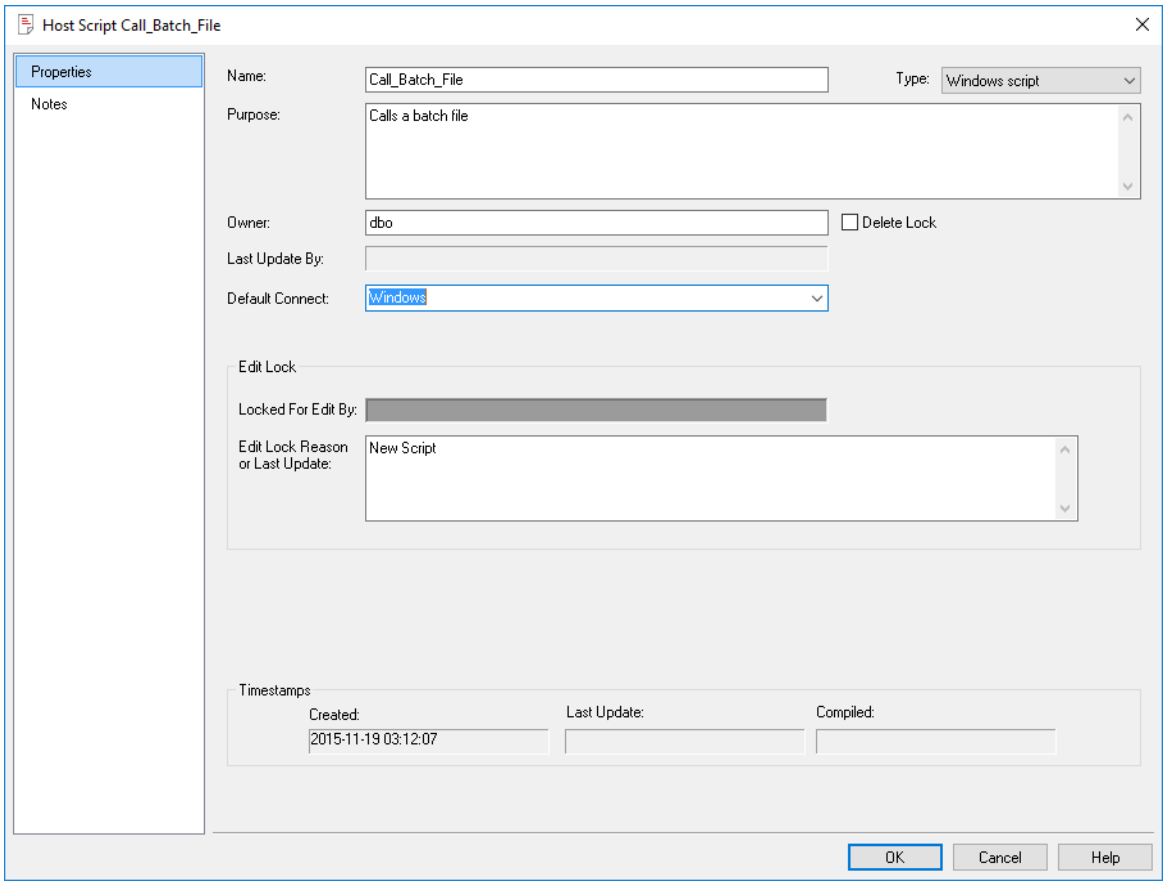
Below is an example RED host script which calls a batch file:

```
@ECHO OFF
SETLOCAL ENABLEDELAYEDEXPANSION
SETLOCAL ENABLEEXTENSIONS
CALL c:\temp\MyBatchFile.bat > c:\temp\MyBatchFile.log 2>&1
IF %ERRORLEVEL% EQU 0 GOTO LABEL_OKAY
ECHO -2
ECHO Batch file returned an error code of %ERRORLEVEL%
TYPE c:\temp\MyBatchFile.log
EXIT
:LABEL_OKAY
ECHO 1
ECHO Batch file completed successfully
TYPE c:\temp\MyBatchFile.log
```

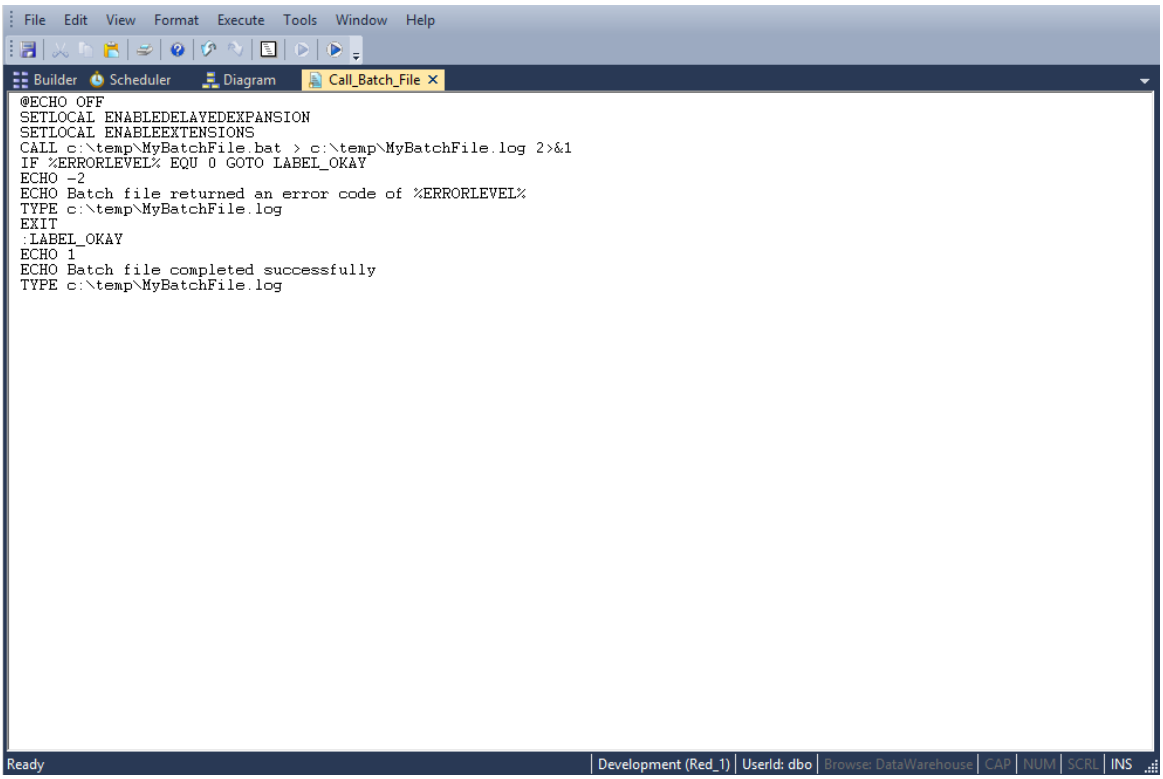
Where "c:\temp\MyBatchFile.bat" contains this:

```
ECHO Hello
SET ERRORLEVEL=0
```

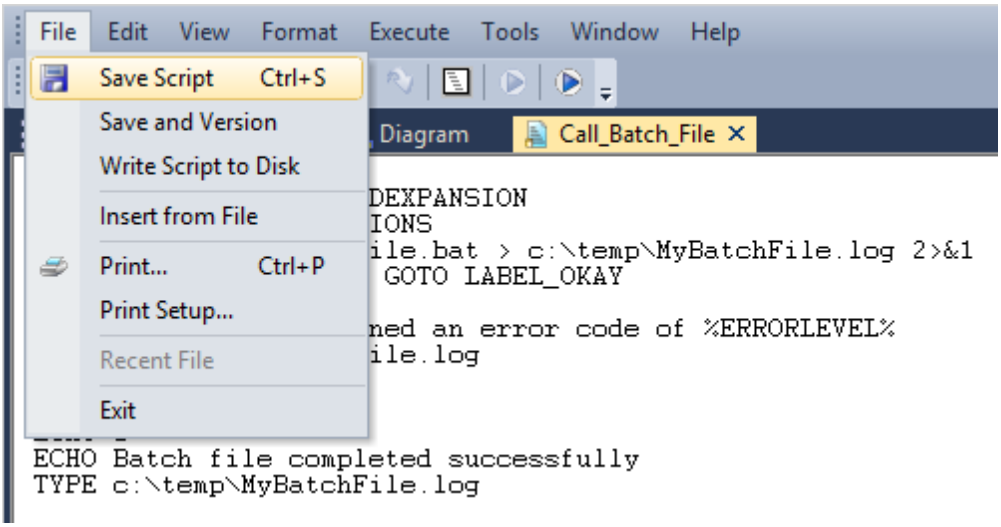
Create the Host Script in RED:



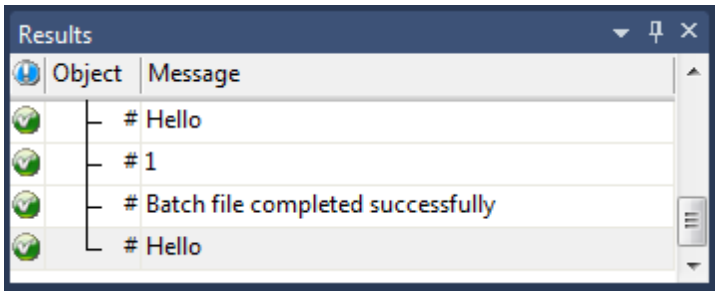
Edit the Script and enter the following:



Save the Script:



When the script is executed, you will see the following results:

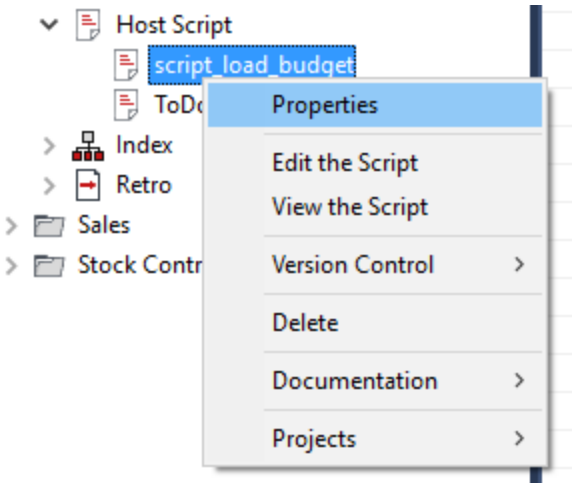


SCHEDULING SCRIPTS

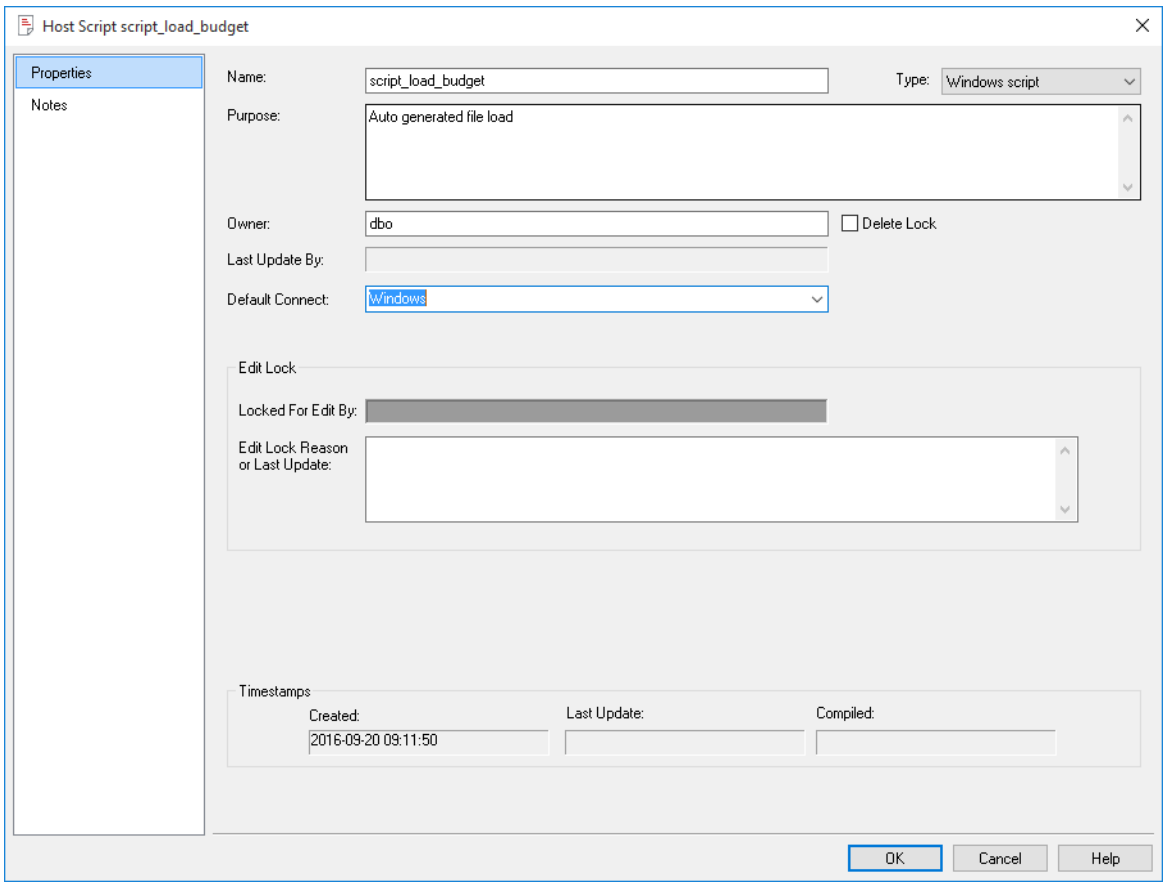
When a host script is scheduled, it is run in the scheduler environment. Therefore, a UNIX scheduler must be available to run a UNIX script and only a Windows scheduler can run a Windows script.

It is important to set the **default connection** on the Properties screen for that script.

Right-click on the host script in the left pane and select **Properties**.



Set **Default Connect** to either **Windows** or **Unix** and click **OK**.



Note: If you fail to set the default connection for the host script, you will receive a return message of **Invalid Host Type** when the host script is executed.

There are a number of conventions that must be followed if a host script is to be used by the WhereScape scheduler. These conventions are:

1. The first line of data in 'standard out' must contain the resultant status of the script. Valid values are '1' to indicate success, '-1' to indicate a warning condition occurred but the result is considered a success, '-2' to indicate a handled error occurred and subsequent dependent tasks should be held, -3 to indicate an unhandled Failure and that subsequent dependent tasks should be held.
2. The second line of data in 'standard out' must contain a resultant message of no more than 256 characters.
3. Any subsequent lines in 'standard out' are considered informational and are recorded in the audit trail. The normal practice is to place a minimum of information in the audit trail. All bulk information should be output to 'standard error'.
4. Any data output to 'standard error' will be written to the error/detail log. Both the audit log and detail log can be viewed from the WhereScape RED tool under the scheduler window.

MANUALLY CREATED SCRIPTS

Individual scripts can also be manually created in RED to perform and schedule tasks that are not related to load tables.
The example below shows a minimal script that will run successfully.

```
@echo off

REM ****************************************************************
REM ***** Parameter Example
REM ****************************************************************


SET /A RESULT_CODE=1
SET RESULT_MESSAGE="Success."

SET FILEAUD=%WORKDIR%\wsl%SCRIPT%%SEQUENCE%.aud
SET FILE_PROCESS_LOG=c:\temp\process.log

echo $PDS_Customer_Process_Date$ >> %FILE_PROCESS_LOG%


echo %RESULT_CODE%
echo %RESULT_MSG%

exit
```

Please note that you need to use the following codes to determine the script's results meaning. It is important that one of these codes is the first output of the script.

Output	**Description**
Result Number	Output Result Number: 1 Success. **-1 Warning.** -2 Error. -3 Fatal/Unexpected Error.

WhereScape®

CHAPTER 22
TEMPLATES

Templates provide the ability to customize automatically generated code within RED. This feature is most suited to users that would like to customize automatically generated code or would like to expand RED to support non-native database platforms.

Creating templates is an advanced function that requires intimate knowledge of RED operations and metadata structure. WhereScape recommends that you contact our consulting team to assist with this feature. However, should you wish to use this feature independently, example templates and up-to-date reference information is available on our website:

https://www.wherescape.com/support/software-downloads-documentation/wherescape-red/templates/

Some templates may be included in your RED installation, depending on your license.

Each template is assigned a type and a target database, these properties are used to assist with filtering when associating table operations to templates. RED supports templates for the following operations:

Operation	Database	Template Type
Create DDL	All database types	DDL
Export Script	All database types	Windows Script
		Unix Script
		Olap/XMLA Script
Load Script	All database types	Windows Script
		Unix Script
		Olap/XMLA Script
Update Procedure	Custom	Block
	Hive	Block
	PDW	Block
		Procedure

Utility type templates can contain common code for use by other templates.

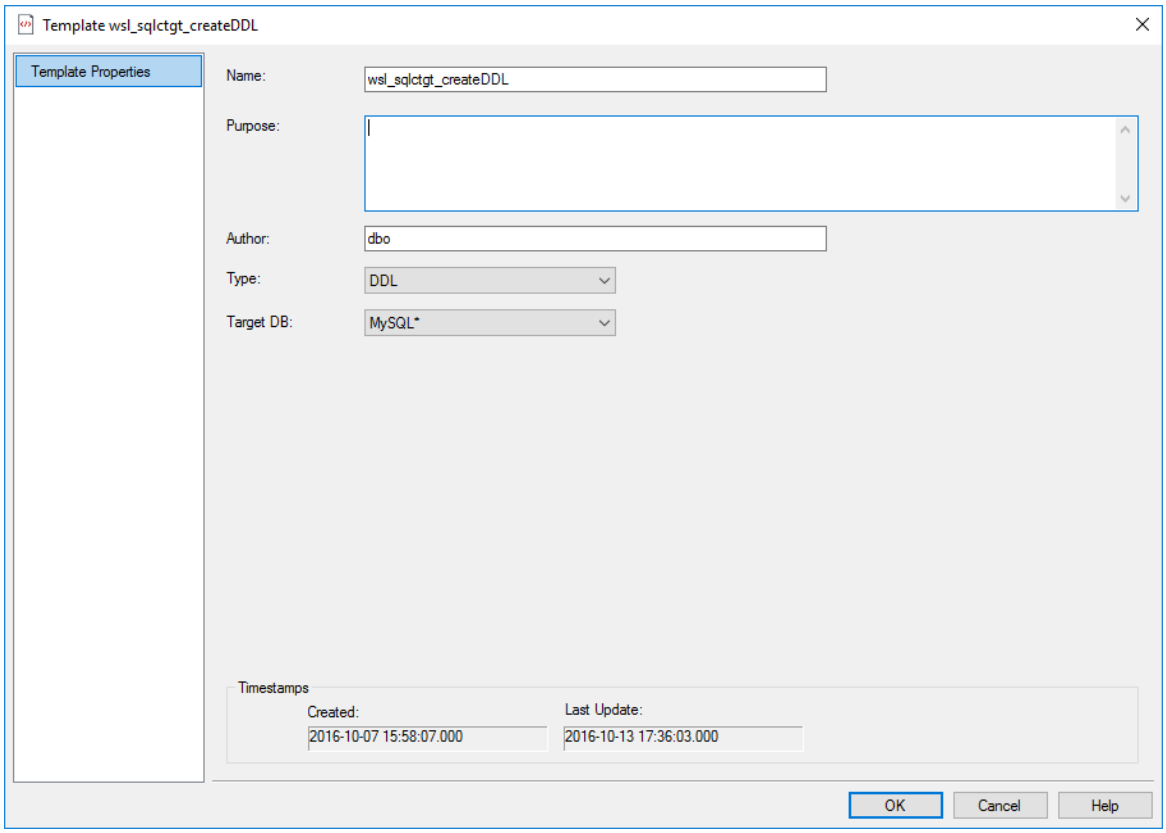
Templates are written in the Pebble template language, for more information on Pebble see **http://www.mitchellbosecke.com/pebble/documentation**.

 **TIP:** Detailed logs can be produced during template evaluation by typing *FULLLOG* in the Notes of the relevant connection.

IN THIS CHAPTER

TEMPLATE PROPERTIES

The properties screen for a template is shown below.



Name, **Purpose** and **Author** fields should be completed to provide background information on the template; these fields are purely informational.

Created and **Last Update** fields provide date information on the template.

The **Type** field informs RED what this template can be used for. This can be set to one of the following:

- Block
- DDL
- Function
- OLAP (XML/A) Script
- Procedure
- Unix Script
- Utility
- Windows Script

The **Target DB** sets the type of database connections for this template. The template will only be selectable for an operation when the **Target DB** field matches that of the object. Target DB is restricted based on your license.

- Common (applies to all databases)
- Custom
- DB2
- Greenplum
- Hive
- Netezza
- Oracle
- PDW
- SQL Server
- Tabular
- Teradata

NOTE: Hive and Custom update procedure templates only support Block update procedures so you should create a block template for these.

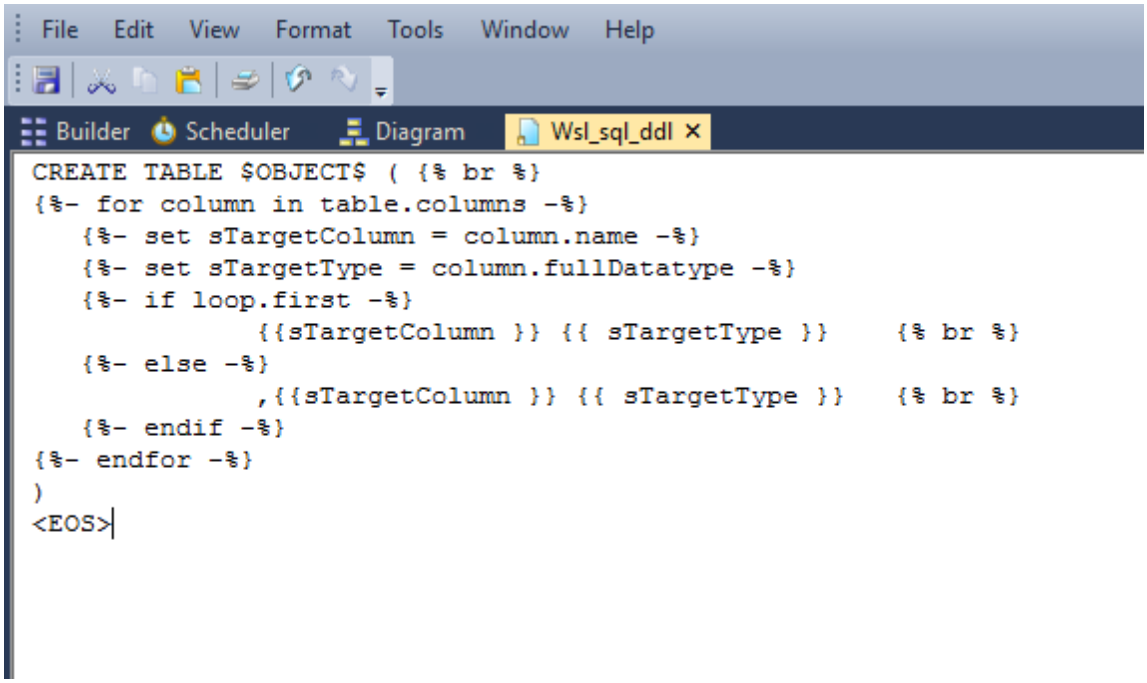
Properties Screen Banner

The banner message will vary depending on the state of the template object.

- *<TEMPLATE NAME> - NO UPDATE: This template is READ ONLY*
  indicates that this is a shipped master template that may not be changed.
- *<TEMPLATE NAME> - NO UPDATE: Checked out by <USER>*
  indicates that the template has been checked out by a user and thus may not be edited until checked back in.
- <TEMPLATE NAME> - NO UPDATE: Checked out by <USER>
  indicates that the template is in use by procedures and may thus not be altered. The **Type:** field is also locked.

TEMPLATE EDITOR

Right-click on a template and select '**Edit Template**' or '**View Template**' to open the Template Editor.



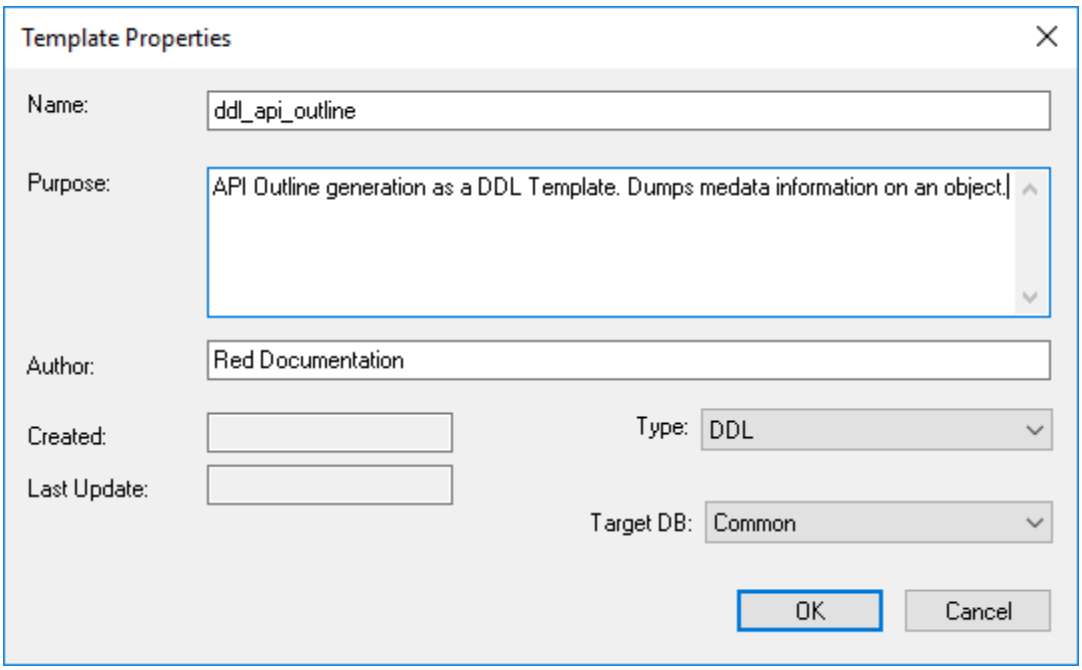
EVALUATING AN API OUTLINE TEMPLATE

An API Outline Template is available to output all object properties relevant to the current object. Upon evaluation of this template, the status of each property is generated and printed to the script or procedure file.

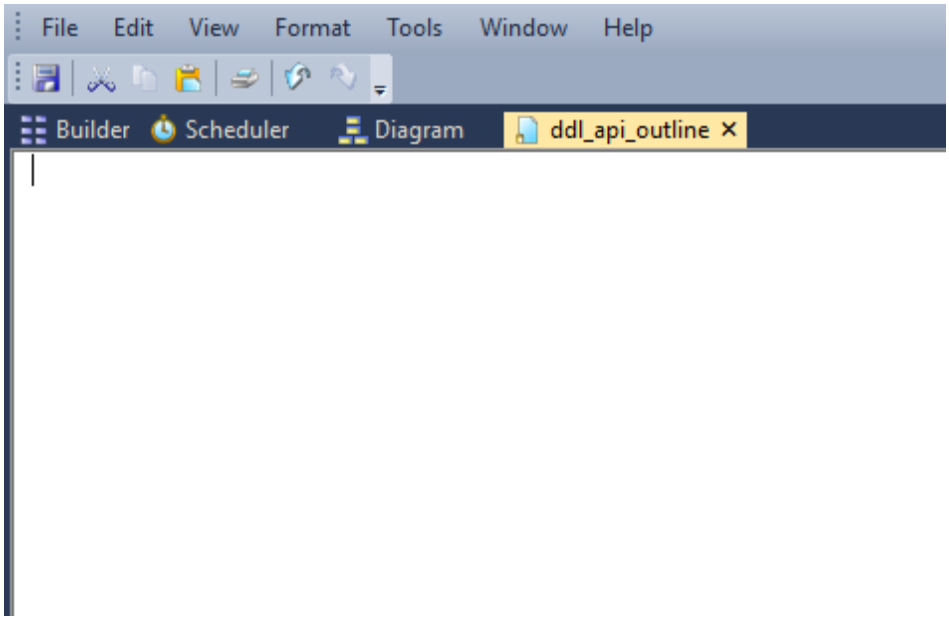
NOTE: Template evaluation usually generates a script or procedure file, but the API Outline Template generates plain text. The output of this template is intended to be viewed or copied to a text file, it cannot be executed as a script.

To evaluate an API Outline Template:

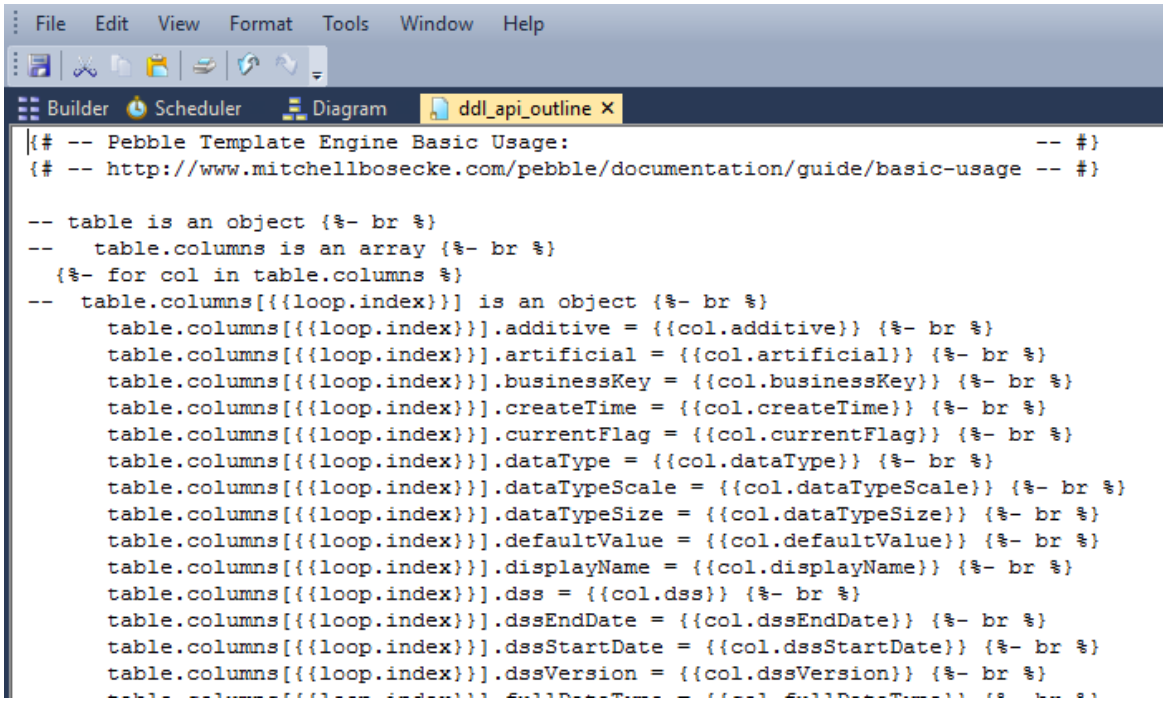
1 Create a **new template**. The template can normally be of any type but in this example, we'll use the DDL template type as viewing the evaluation of DDL templates is simple.
Set **Target DB** to your source connection database type. In this example, we will set the Target DB to SQL Server because the load table we are evaluating is stored on a SQL Server connection.



2 Open the template in the Template Editor.



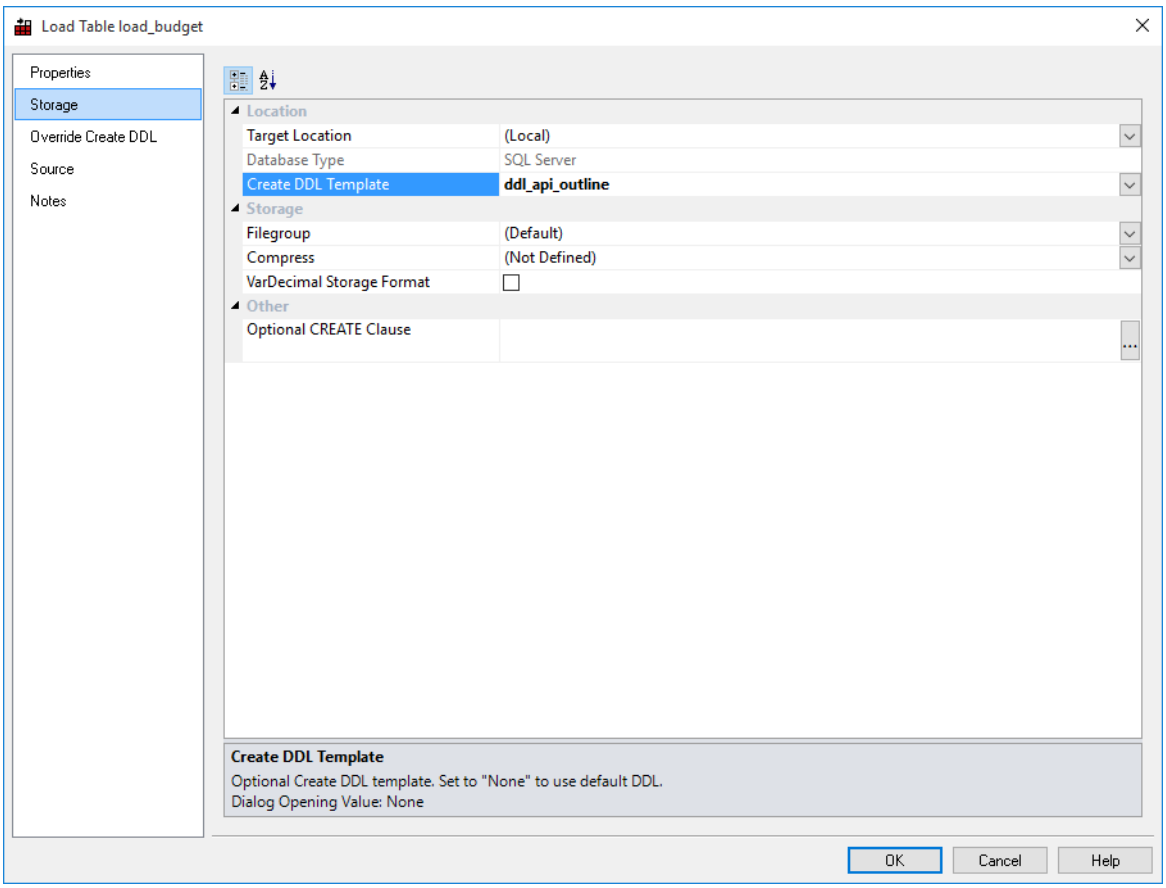
3 Click **Tools > Create API Example Outline**. The API Example Outline text is added to the blank template.



4 **Save** and **close** the template.

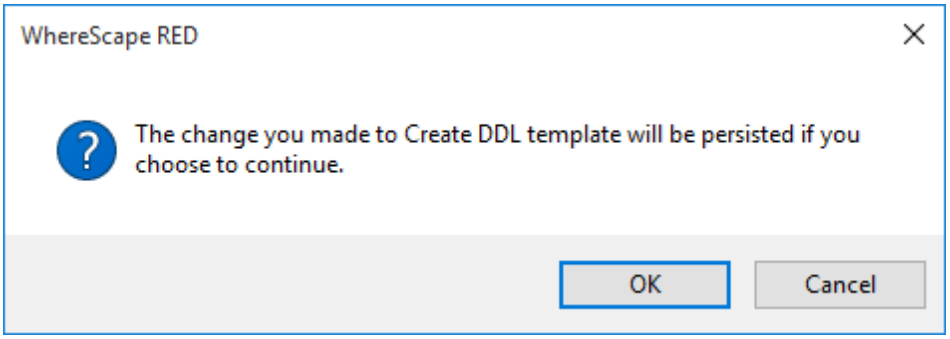
5 Open the Properties dialog for the **Load Table** you wish to evaluate.

6 In the **Storage** tab, select the template you created in the **Create DDL Template** drop-down box.

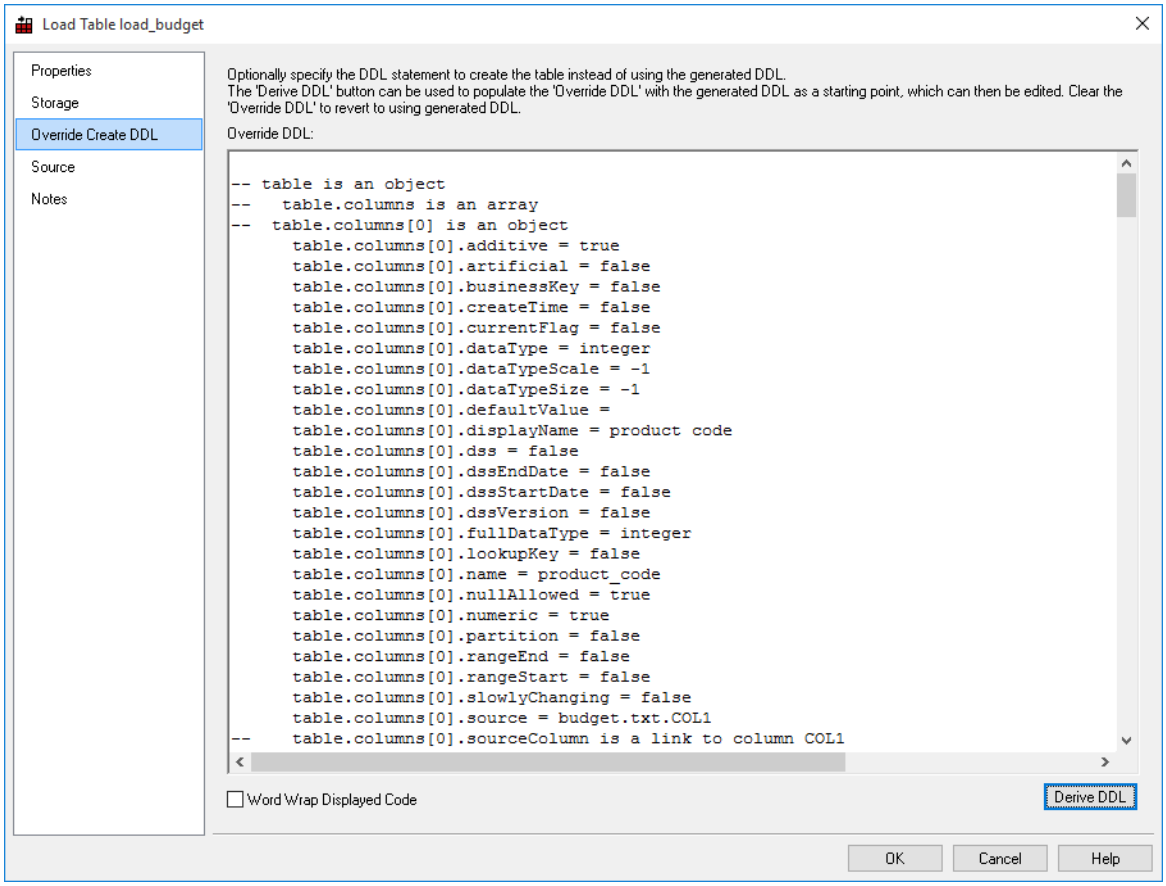


7 Open the **Override Create DDL** tab. If the **Override DDL** field is populated with a custom DDL statement, copy and paste this statement to a text file for backup purposes.

8 Click the **Derive DDL** button. A warning dialog box displays informing you that the association of the DDL template with this table will be saved to the metadata, click **OK**.



9 The results of the API Outline Template are printed to the **Override DDL** text box. Cut or copy this text to a text editor and save as a text file for reference purposes.



10 Click **Cancel** in the Load Table properties dialog.

NOTE: Ensure the Load Table properties are returned to their previous state. The default value for **Storage>Create DDL Template** is `None` and the **Overide Create DDL>Override DDL** field is left blank to use the automatically generated DDL statement or ensure your custom DDL statement remains.

TEMPLATE USAGE

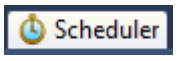
If a template exists of the correct **Type** and **Target DB**, templates can be specified and evaluated as follows:

Operation	Location to Specify the Template	Notes
Create DDL	**Table Properties dialog>Storage>Create DDL Template**	DDL will be evaluated for the object at runtime if a template is specified in the Storage tab. Alternatively, clicking the Derive DDL button in the Override Create DDL tab will generate Override DDL based on the specified template. **IMPORTANT:** If Override DDL is specified, the Override DDL will be used at runtime.
Export Script	**Export Object Properties dialog>File Attributes>Export Script Template**	
Load Script	**Load Table Properties>Source>Load Script Template**	
Update Procedure	When building the Update Procedure, specify the Template field on the **Update Build Options dialog>Processing tab**.	If a **Block** template is specified, a SQL block will be generated. If a **Procedure** template is specified an update procedure will be generated.

To check which Operations are supported by Templates on your Target DB, see ***Templates*** (on page 726).

WhereScape®

CHAPTER 23
SCHEDULER

The scheduler is accessible by clicking the **Scheduler** button 

on the toolbar. It is also available as a stand alone utility. In this way, operators can be given access to the scheduler without gaining full access to the data warehouse.

The scheduler runs on either a UNIX host or under Windows (as a system service). It processes predefined jobs, recording the success, failure or otherwise of the job.

Audit trail

Specific information relating to the tasks in the job are recorded to the audit trail. Generally only summary information is written to this audit trail. The contents of the audit trail are maintained even after a job is deleted.

Error trail

Detail or error information is written to the error trail. The contents of the error trail are deleted when the job is deleted.

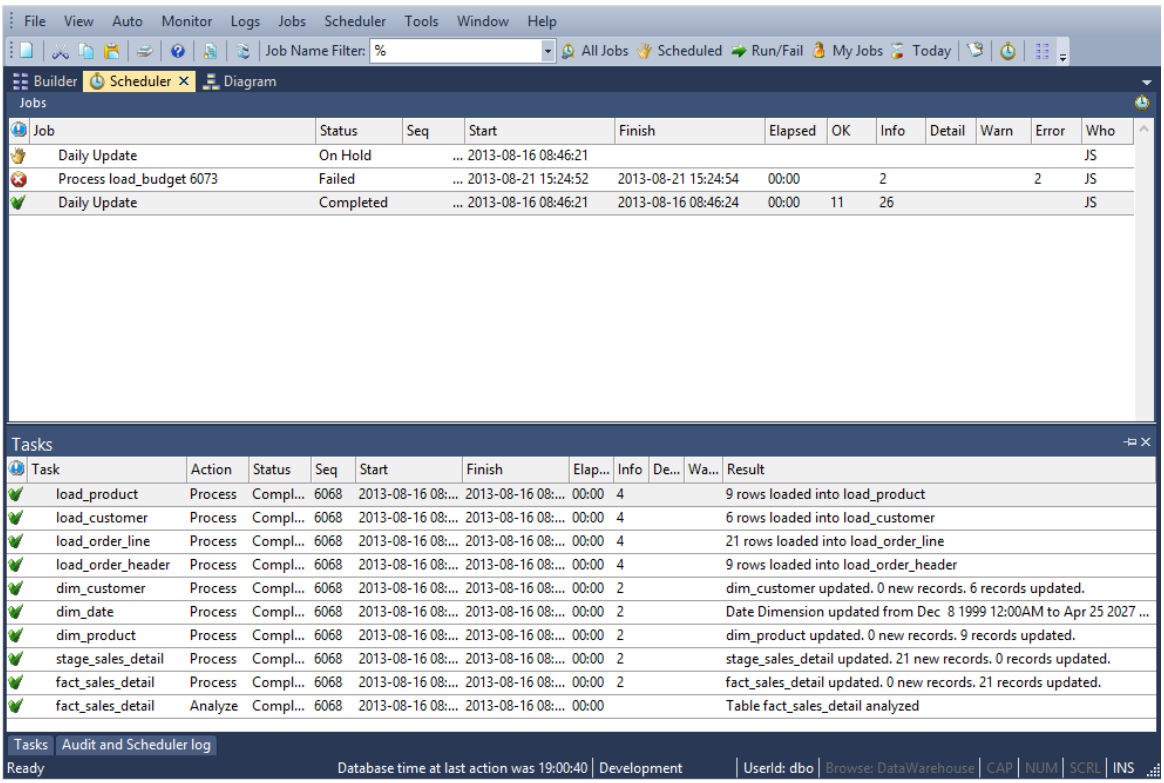
Administration Views

It is possible to view the status of a job without using the WhereScape RED product. Three views are provided to assist in this undertaking. They are ws_admin_v_audit, ws_admin_v_error and ws_admin_v_sched. Queries can be issued using these views to see the results or status of a job.

IN THIS CHAPTER

SCHEDULER OPTIONS

An example of the **Scheduler** screen is shown below.



Toolbar/Jobs menu

Quick access to some job categories are in the toolbar. The complete options are listed under the Jobs menu and while most are self-explanatory they are described below:

Object	Description
All jobs	All jobs are listed in the middle pane
Scheduled jobs	In the middle pane lists those jobs that are waiting to run or are on hold.
Last 24 hours	Lists the jobs that have run or started to run during the last 24 hours.
Prior 24 hours	Lists the jobs that ran during the previous 24 hours.
This weeks Jobs	Lists the jobs that have run or are scheduled to run during the current week
Last weeks Jobs	Lists the jobs that ran during the last week.
My Jobs	Lists the jobs you have scheduled or have run.

Object	Description
Job Name Filter	Lists the jobs whose names match the filter supplied. This filter only works as an appended filter to the main filter selected under **Jobs**. ie first enter a filter for **Job Name Filter** or select a filter from the drop-down list; and then choose your main filter under **Jobs** - eg All Jobs, Scheduled Jobs etc.
Recent audit trail Today's audit trail	Provides listings from the audit trail. The information is useful when a job fails to start or enters some other unknown state. Generally the audit trail entries for a job can be found by drilling down into the job itself.
Scheduler status	Lists all schedulers and displays their current status. The status is updated every few minutes, and a right menu option allows the polling of a scheduler for status, and the termination of a scheduler.

Top pane

The top pane shows the details of the jobs. Information covers:

Column	Description
Job name	The name given to the job when created.
Status	The status of the job. Refer to the following section for the various status values.
Sequence	This is a unique number assigned to each job iteration and job. If you enter a new job it will acquire a new sequence number. In normal daily processing when no new jobs have been created sequence numbers will be sequential.
Start and Finish times	As the names suggest, the start and finish dates and times for the job.
Elapsed time	The time that elapses from start to finish of a job.
Ok	These are success messages written to the audit trail.
Info	Informational messages written to the audit trail about the the running of the job.
Detail	Lines written to the detail or error logs
Warn	The number of warnings written to the audit trail.
Error	The number of error messages written to the audit trail.
Who	The initials of the person who scheduled the job.

Additional fields can be added via the **Tools/Select Job Report Fields** option

Middle pane

The middle pane shows the tasks related to a selected job. Task information available includes:

Column	Description
Task	The object name
Action	The action to be done to the object
Status	Status of the task
Sequence	This is a unique number assigned to each job iteration and job. If you enter a new job it will acquire a new sequence number. In normal daily processing when no new jobs have been created sequence numbers will be sequential.
Start and Finish times	As the names suggest, the start and finish dates and times for the task
Elapsed time	The time that elapses from start to finish of a task.
Info	Informational messages written to the audit trail about the the running of the job.
Detail	Lines written to the detail or error logs
Warning	The number of warnings written to the audit trail.
Result	Result of the task

Additional fields can be added via the **Tools/Select Task Report Fields** option

Bottom pane

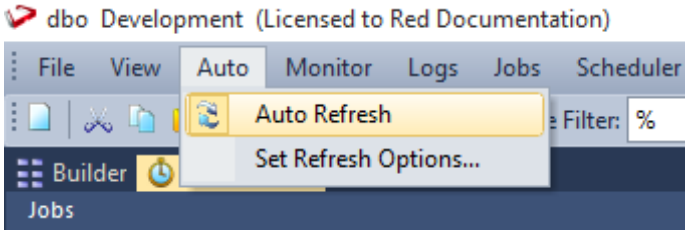
The bottom pane shows the audit trail of a selected task/job. Audit trail information includes:

Column	Description
Task	The object name
Status	The status of the message
Sequence	This is a unique number assigned to each job iteration and job. If you enter a new job it will acquire a new sequence number. In normal daily processing when no new jobs have been created sequence numbers will be sequential.
Timestamp	Time of message output
Message	The message
DB Message	Message from database
Job	Job relating to this message

AUTO

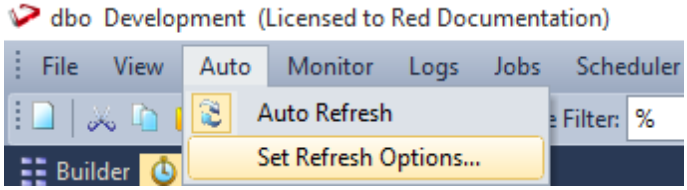
Auto Refresh

Use auto refresh to automatically refresh all jobs.



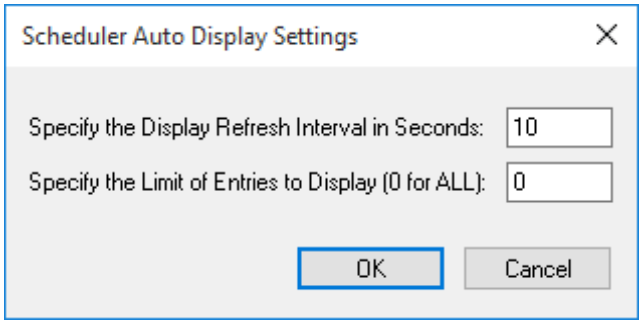
Set Refresh Options

Use this setting to control the maximum number of rows that are displayed via the Auto Refresh option as well as the display refresh interval.



When clicking the **Set Refresh Options**, the settings dialog allows adding a display limit. The jobs displayed when on auto refresh will stop at the selected number of rows, so if the limit is set to 100, the refresh will stop after the first 100 rows (jobs) are returned.

If **0** is selected in the Specify the Limit of Entries to Display, then all rows (jobs) will be displayed.

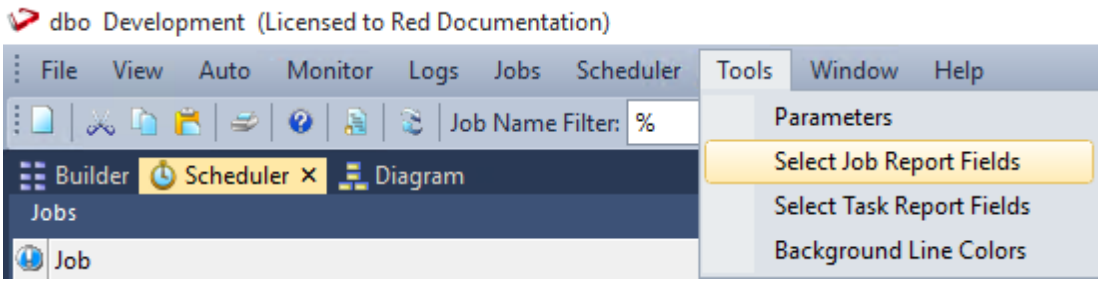


TOOLS

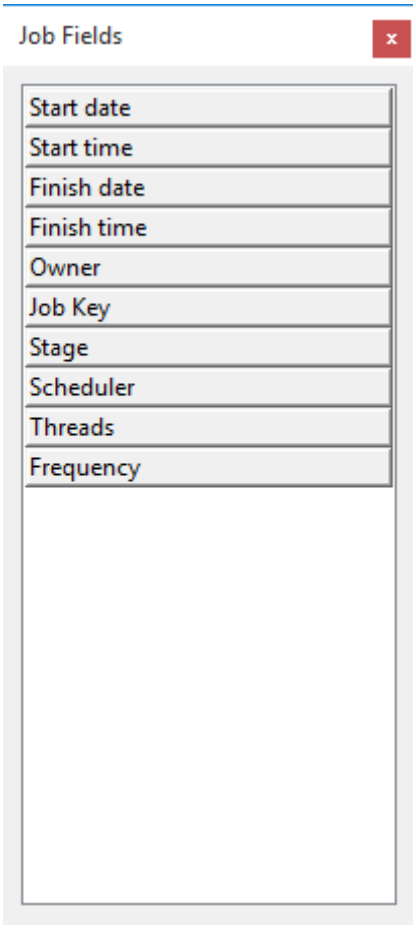
SELECT JOB REPORT FIELDS

The **Select Job Report Fields** menu option in the scheduler pane enables users to select extra fields such as Scheduler, Threads and Frequency fields into the job report.

1 To make these fields available, click **Tools->Select job report fields** from the top pane.



2 Select the fields you want to add to your job report from the menu and drag them to where you want to place them on the report.



NOTE: The "Frequency" field is only populated for scheduled jobs. If selected, it will return blank results for running or completed jobs.

SCHEDULER STATES

A scheduled **job** can have the following states:

- Hold
- Waiting
- Blocked
- Pending
- Running
- Failed
- Failed - Aborted
- Completed

State	Description
Hold	The job is on hold. It can be edited and its state changed to release the job.
Waiting	The job is waiting to start, or waiting for its scheduled time to arrive, or waiting for a scheduler to become available.
Blocked	The job is blocked as a previous instance of the same job is still running. When the running instance completes, this job will start.
Pending	This is the first stage of a running job. The scheduler has identified the job as ready to start and has allocated a thread, or sub task to process the job. A job is in this state until the thread or sub task begins processing. If a job stays in this state, then the scheduler thread has failed for some reason. The logs can be checked on either the UNIX or Windows server on which the scheduler is running.
Running	The job is currently running. Double-click on the job name in the right pane to drill down into the specific tasks.
Failed	A failed job is one that had a problem. It can be restarted from the point of failure and is considered to be running unless subsequently aborted.
Failed - Aborted	The job has been aborted after having failed. Once in this state a job cannot be restarted. The job exists then only as a log of what occurred and is no longer regarded as a job.
Completed	The job has successfully completed, possibly with warnings. Once in this state a job cannot be restarted. The job exists then only as a log of what occurred and is no longer regarded as a job.

Note: When a job fails and drilling down does not show any errors against the tasks, right-click on the job and "View Audit Trail". The job may have failed because of an error in the JOB level.

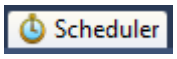
A scheduled **task** can have the following states:

- Waiting or Blank
- Held
- Running
- Failed
- Completed
- Error Completion
- Bad Return Status

State	Description
Held	The task has been held due to a prior dependency failure. The problem must be rectified and the job restarted.
Waiting (Blank)	Tasks that are waiting to run either due to a shortage of threads, or prior dependencies normally have a blank status.
Running	The task is currently running.
Failed	The task has had a fatal error. Any dependencies on this task will be held. Double-click on the task to see more detail error information or review the audit and error/detail log for the job.
Completed	The task has completed successfully.
Error Completion	The task has completed with a handled Error. Any dependent tasks will be held, and the job must be restarted when the problem is rectified.
Bad Return Status	The task has returned an unknown status. This normally occurs with script files that produce unexpected information. The rule for scripts is that the first line returned must be a status of either 1, -1, -2, or -3. The second line is a message detailing the result. If the first line does not contain one of these four values then this status will be returned and dependent tasks held. Run the script manually to view the output or check the logs.

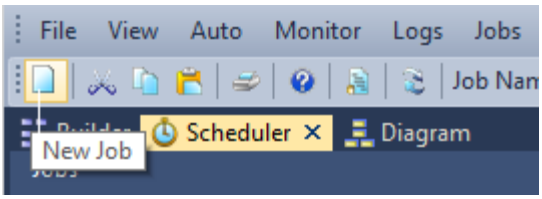
SCHEDULING A JOB

To schedule a job

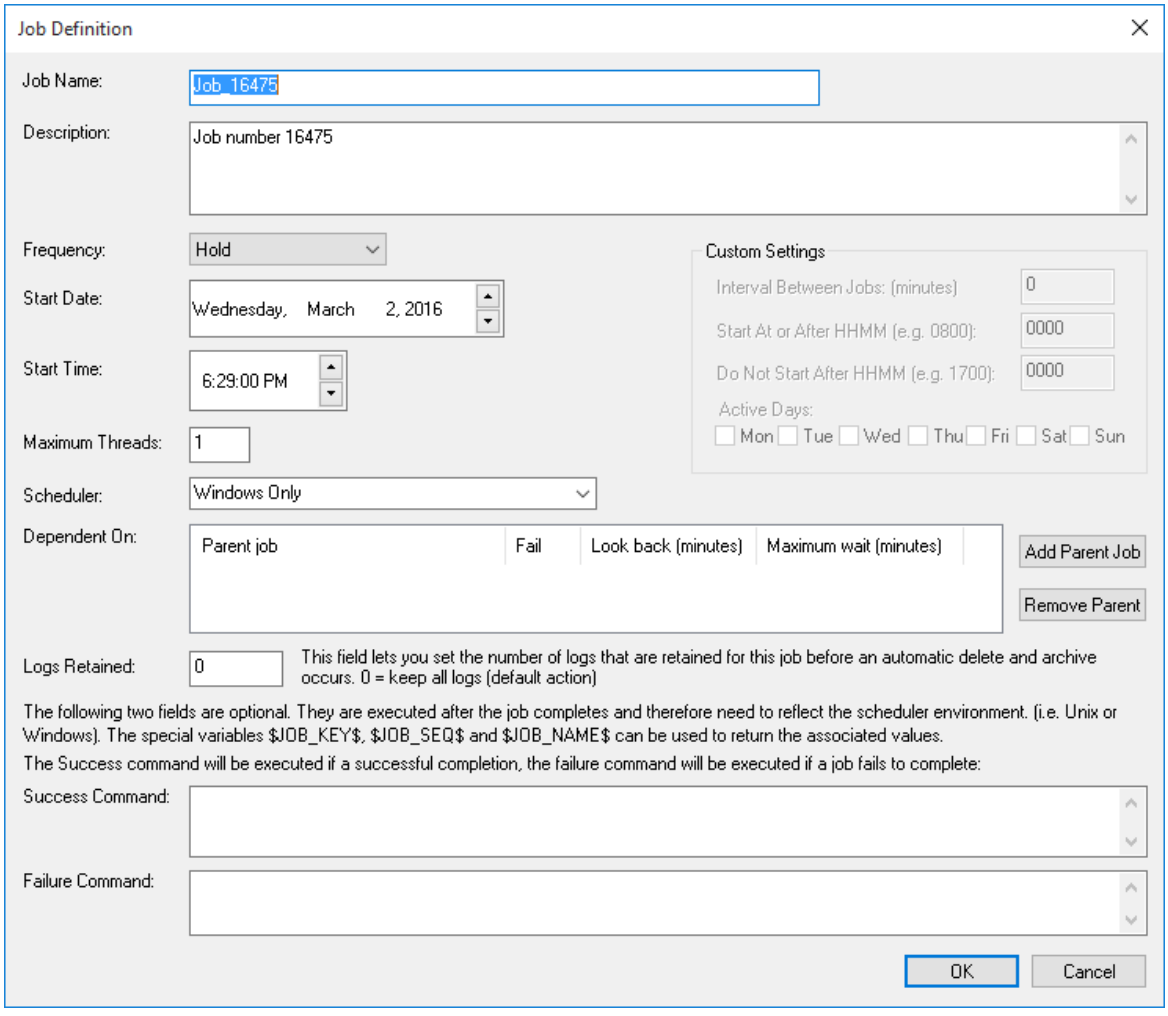
Firstly access the scheduler by clicking the **Scheduler** button 

on the toolbar.

Select **File/New Job** from the menu strip at the top of the screen, or click the **New Job** button on the toolbar.

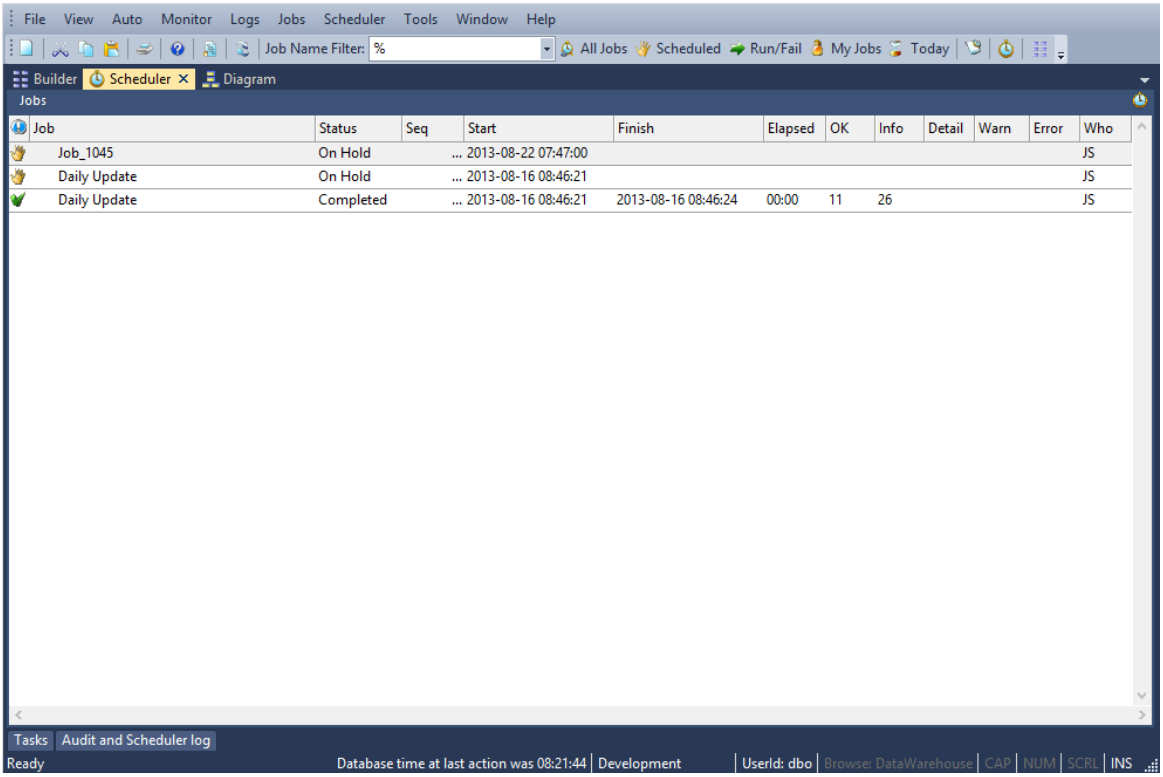


The following **Job Definition** dialog is displayed.



See ***Creating a Job*** (on page 751) for more details on how to create a new job.

Once the job has been created, click on the **All Jobs** button on the toolbar. The newly created job will now be displayed in the scheduler window.



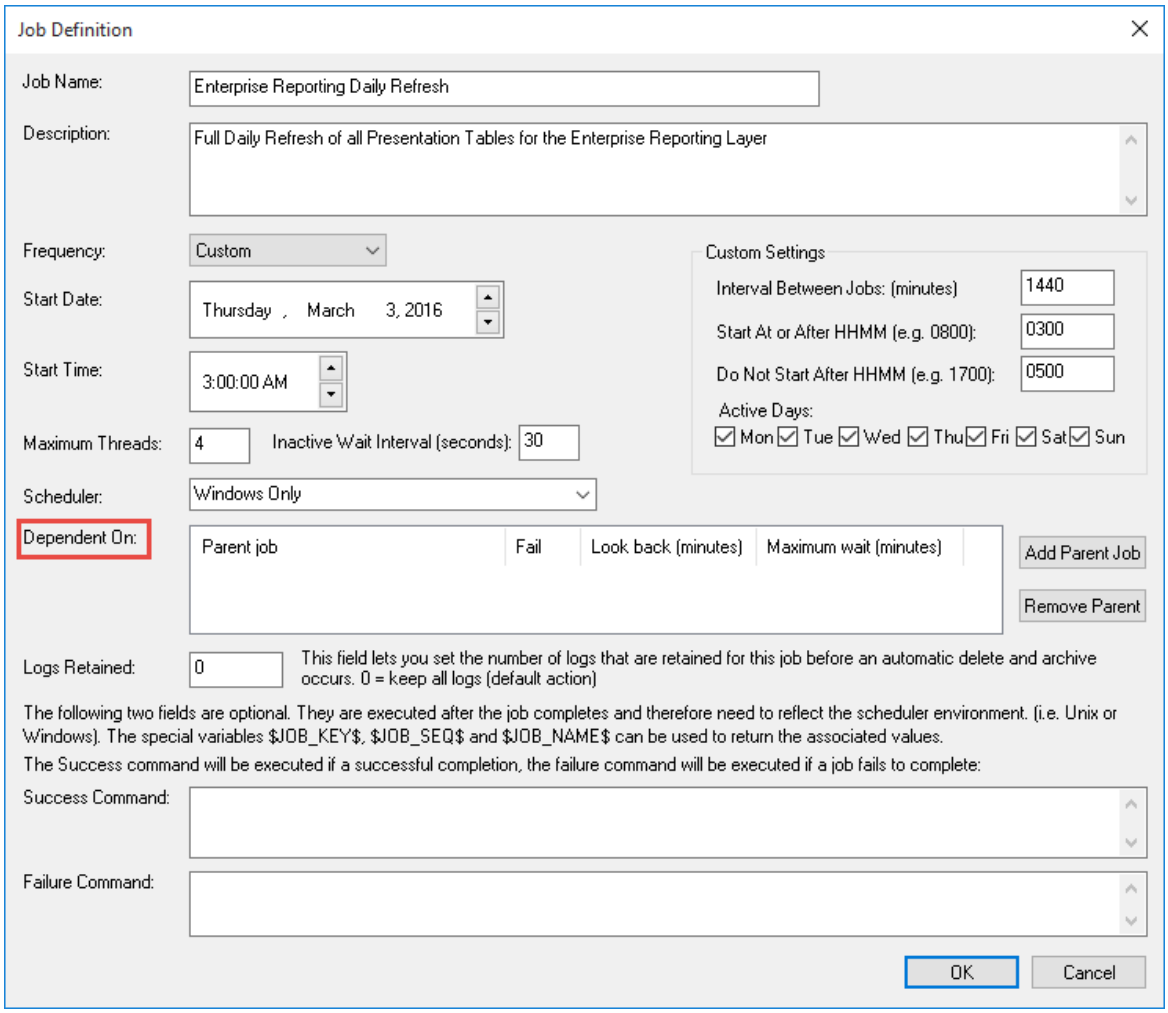
To create a job within a job

It is possible to schedule one job from another job. There are however some limitations and rules that must be understood when doing this.

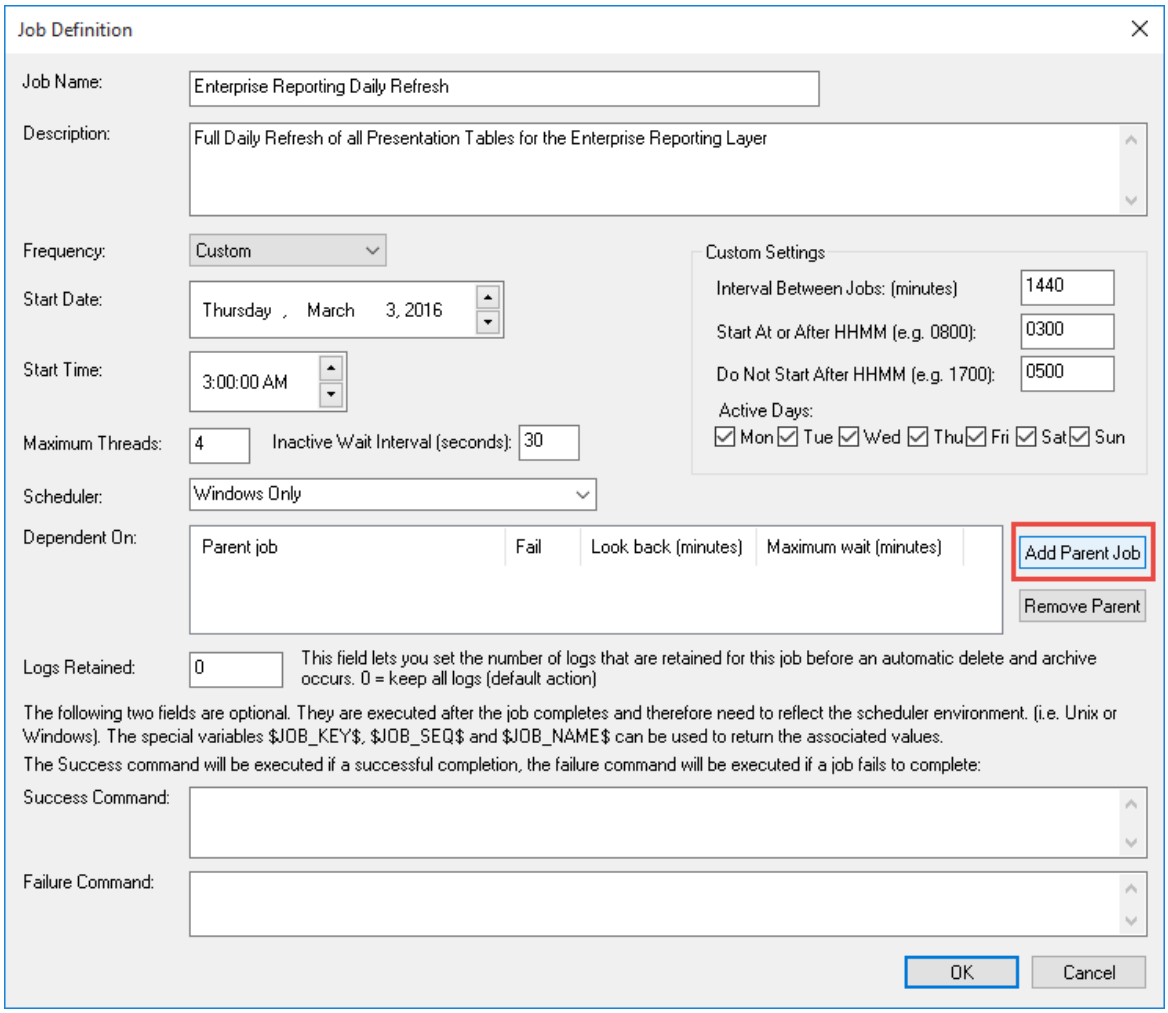
1. A job that is called from another job is only ever allocated one thread. All tasks within the called job will therefore run sequentially.
2. A job can only have one running iteration. Therefore, a called job will be blocked if that job is already running independently or as part of another job.
3. Any job dependencies for the called job are ignored. The parent's job dependencies are the only ones that are used.
4. A called job essentially runs as a separate job, so that if it fails both it and the parent job will show in a failed state. Once the problem is fixed the parent job should be restarted which will restart the called job.

To create a job dependency

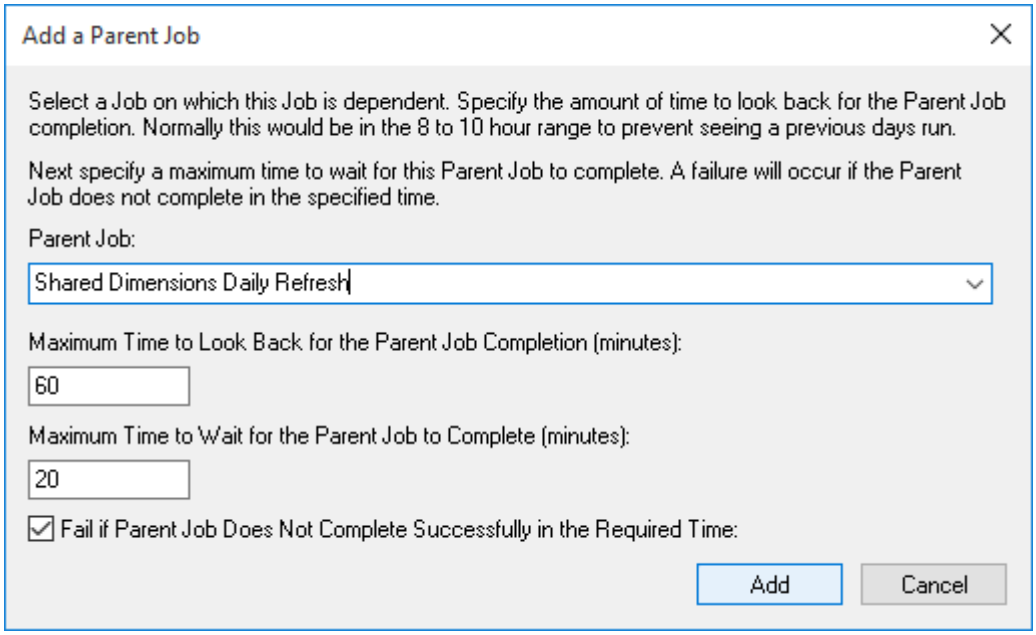
It is possible to make a job dependent on another job, using the **Dependent On** field in the **Job Definition** dialog.



Click on the **Add Parent Job** button.



In the dialog that follows, select the Parent Job from the drop-down list. In our case, we will choose the job **Shared Dimensions Daily Refresh**.



The **Maximum Time to Look Back for the Parent Job Completion** field prevents older iterations of the parent job as being identified as a completion. In our example, we are starting both jobs at 3am, so we don't need to look too far back to ensure that the dimension refresh has run. We have therefore set the look back minutes to 60 to allow for any delays in starting this job.

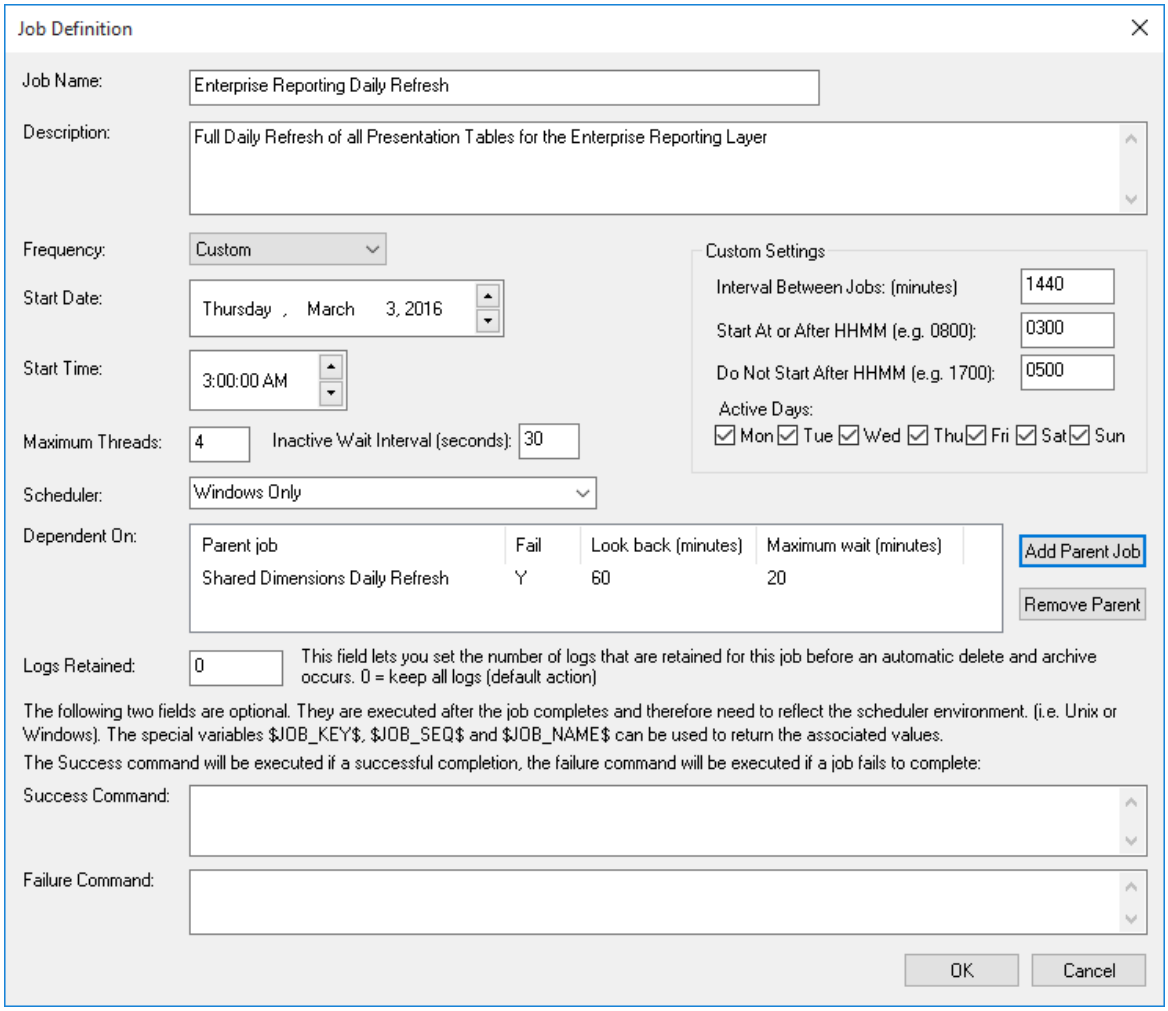
The **Maximum Time to Wait for the Parent Job to Complete** specifies how long to await a successful completion of the parent job. In our example, we know that the dimension refresh only takes a few minutes, but we should allow for the occasional slow network or resource drains making the dimension refresh take longer; so we have set the maximum wait to 20 minutes. This means that our job will wait 20 minutes from its own scheduled start time for the parent job to complete.

The checkbox to fail if the parent job does not complete in time will prevent this job from running if the parent job (dimension refresh) does not complete successfully. As we do not wish for the transactional data in our fact deliveries to be flagged with 'Unknown' dimensional item(s), we can leave this check-box checked to ensure that this job does not run.

Click **Add**.

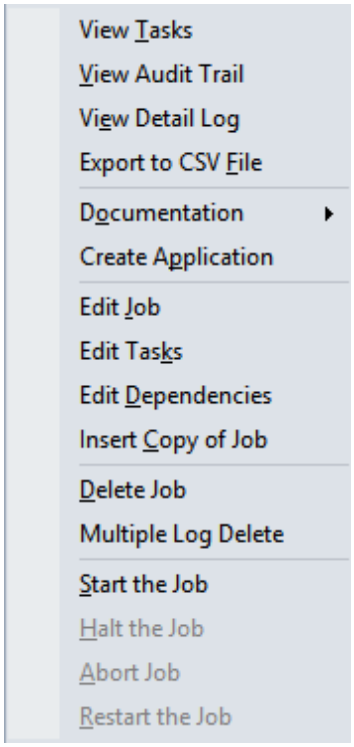
NOTE: Clearing the check-box to fail if the parent fails will simply ensure that this job waits for the completion of the dimension refresh and, irrespective of the dimension's refresh success or failure, starts.

Click **OK** to link this data job to the parent dimensional job. In this way, the job **Enterprise Reporting Daily Refresh** cannot run until the parent job **Shared Dimensions Daily Refresh** has completed successfully; thus the facts will have the latest dimensional keys associated with them.



WORKING WITH JOBS

When positioned on a Job in the scheduler window, the right-click pop-up menu provides a number of options for working with the job. Some of the options are discussed in more detail in the chapters that follow, however a brief overview of the menu options follows:



The **View Tasks** menu option enables you to view the tasks of a job.

The **View Audit Trail** option enables you to view the audit trail of a job.

The **View Detail Log** option enables you to view a detail log of a job.

The **Export to CSV File** option enables you to export a job to a CSV file.

The **Documentation** option enables you to create documentation for a job.

The **Edit Job** option enables you to edit a job. See ***Editing a Job*** (on page 764)

The **Edit Tasks** option enables you to edit the tasks of a job. See ***Editing Tasks*** (see "***Editing Tasks in a Job***" on page 767)

The **Edit Dependencies** option enables you to edit the task dependencies of a job. See ***Editing Dependencies*** (see "***Editing Task Dependencies***" on page 777)

The **Insert Copy of Job** option enables you to insert a copy of a job. See ***Inserting a Copy of a Job*** (on page 784)

The **Delete Job** option enables you to delete a job. See ***Deleting a Job*** (on page 786)

The **Multiple Log Delete** option enables you to delete multiple logs of a job. See ***Deleting Job Logs*** (on page 787)

The **Start the Job** option enables you to start a job. See ***Starting a Job*** (on page 789)

The **Halt the Job** option enables you to halt a job. See ***Halting a Job*** (on page 790)

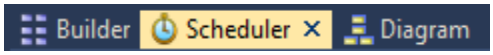
The **Abort the Job** option enables you to abort a job. See ***Aborting a Job*** (on page 791)

The **Restart the Job** option enables you to restart a job. See ***Restarting a Job*** (on page 795)

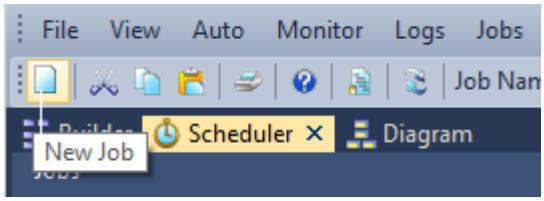
CREATING A JOB

To create a job

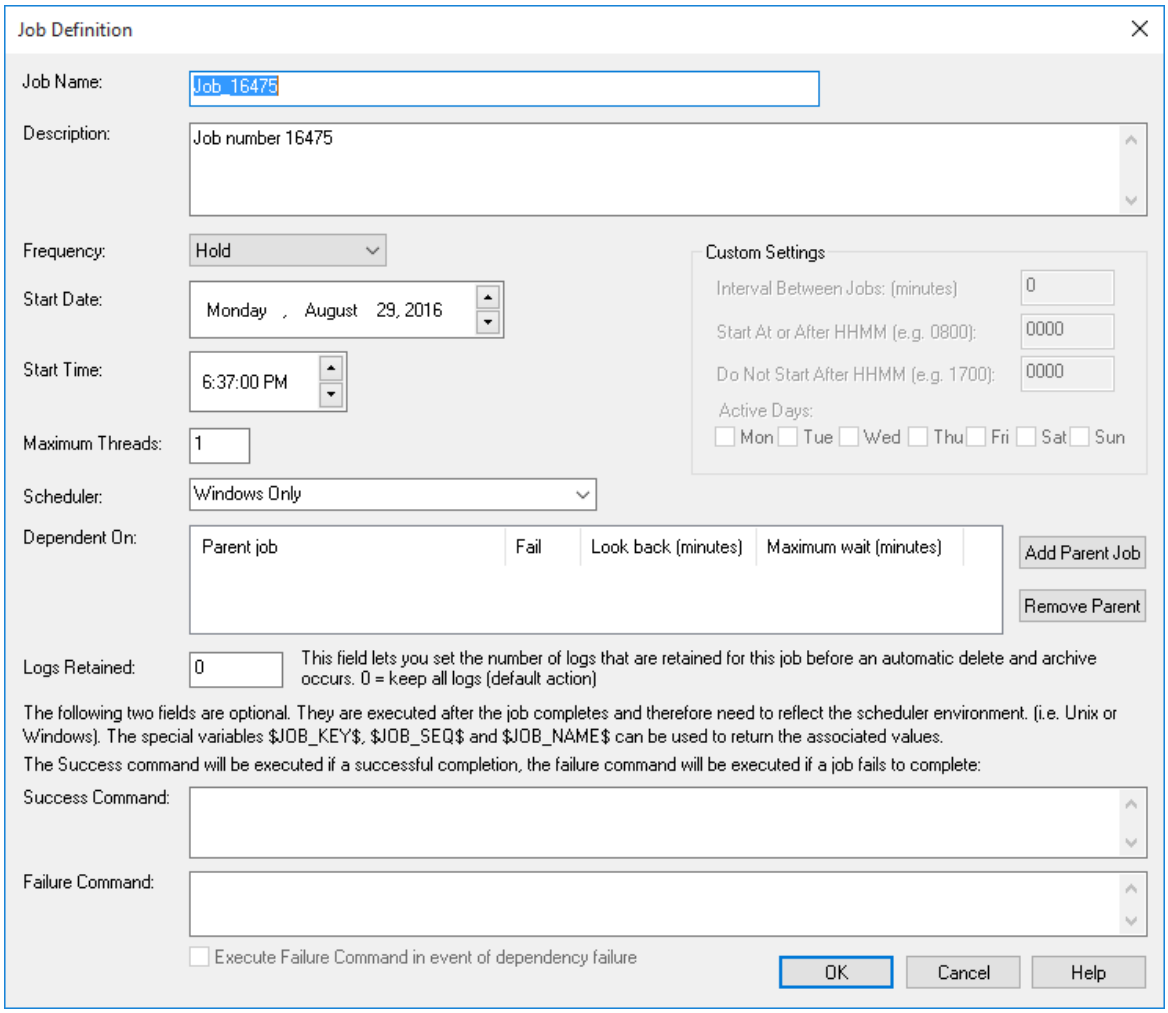
Click on the **Scheduler** tab to open the scheduler window.



Click on the **New Job** button to create a new job.



A **Job Definition** dialog is displayed.



Complete the fields and click **OK**. The main fields are described in the following table:

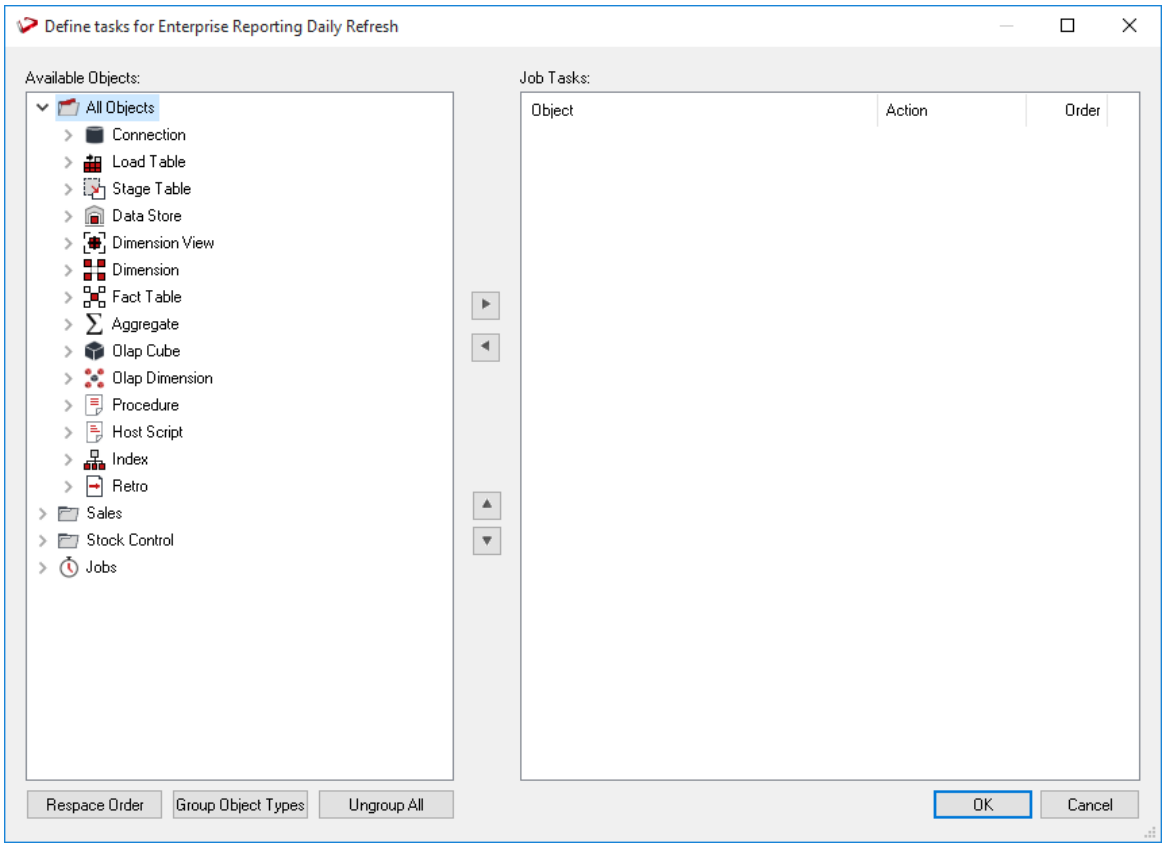
Field	Description
Job Name	The Scheduler defaults to the next job number in the sequence. You can alter this to any alphanumeric. **Note**: Only alphanumerics, spaces and the underscore are supported in the name. WARNING: on some UNIX systems long job names can cause jobs to be canceled (see Knowledge Base article 67), so where possible keep the name short.
Description	A description of the job
Frequency	When the job runs. The options available in the drop-down list box are: • Once Only - job is deleted on completion • Once and Hold - runs and puts another copy of the job on hold • Hold - puts the job on hold for manual release • Daily - runs the job daily • Custom - enables custom definition • Weekly - runs the job weekly • Monthly - runs the job monthly • Annually - runs the job annually
Start Date and Start Time	• The date and time for the job to start.
Max Threads	The maximum number of threads allocated to run the job, e.g, if some tasks can run in parallel then if more than one thread is allocated then they will run in parallel.
Scheduler	Certain types of jobs will only run in a specific environment. For example ODBC based loads can only be handled by the Windows' scheduler. It is possible to have multiple schedulers running. Select the desired scheduler from this drop-down. The valid options are: UNIX Preferred, UNIX Only, Windows Preferred, Windows Only, or the name of a specific scheduler can be entered (e.g. WIN0002)
Dependent On	A job can be dependent on the successful completion of one or more other jobs. Click the **Add Parent Job** button to select a job that this job will be dependent on. The maximum time to look back for parent job completion field prevents older iterations of the parent job as being identified as a completion. The maximum time to wait specifies how long to await a successful completion of the parent job. The action if that wait expires can also be set. See the Job Dependency example in ***Scheduling a Job*** (on page 744).
Logs Retained	Specify the number of logs to retain for the job. By default all logs are retained. This field can be used to reduce the build up of scheduler logs by specifying a number of logs to retain.

Field	Description
OK command and Failure command	These are either UNIX or Windows shell commands depending on which scheduler is used. They are executed if the condition is met. Typically, these commands would mail or page on success or failure. **Notes**: 1. The RED scheduler does not check return codes from called commands, scripts and programs. 2. It is recommended that all output from commands, scripts and programs is redirected to a log file. For example, add this to the end of any SUCCESS/FAILURE commands: >> c:\scheduler\success_failure_prod.log 2>&1

The following fields are available if a frequency of **Custom** is chosen:

Field	Description
Interval between jobs (Minutes)	Specify the number of minutes between iterations of the job. For example, to run a job every 30 minutes set this value to 30. If a job is to run only once but on selected days set this value to 1440 (daily)
Start at or after HHMM	The time that the job may run from. To run anytime set to 0000.
Do not start after HHMM	If multiple iterations are being done, then this is the time after which a new iteration will not be started. For example, if a job is running every 10 minutes it will continue until this time is reached. To run till the end of day set to 2400.
Active on the days	Select each day of the week that the custom job is to be active on.

Once the job itself has been defined, tasks then need to be added to the job. The **Define tasks** window is shown below.



The screen has two main areas. The right pane shows the tasks to be run for this job and the left pane lists all the objects.

To add a task

Double-click on an object in the left pane to add it to the task list in the right pane. Normally objects such as load or fact tables are scheduled rather than procedures.

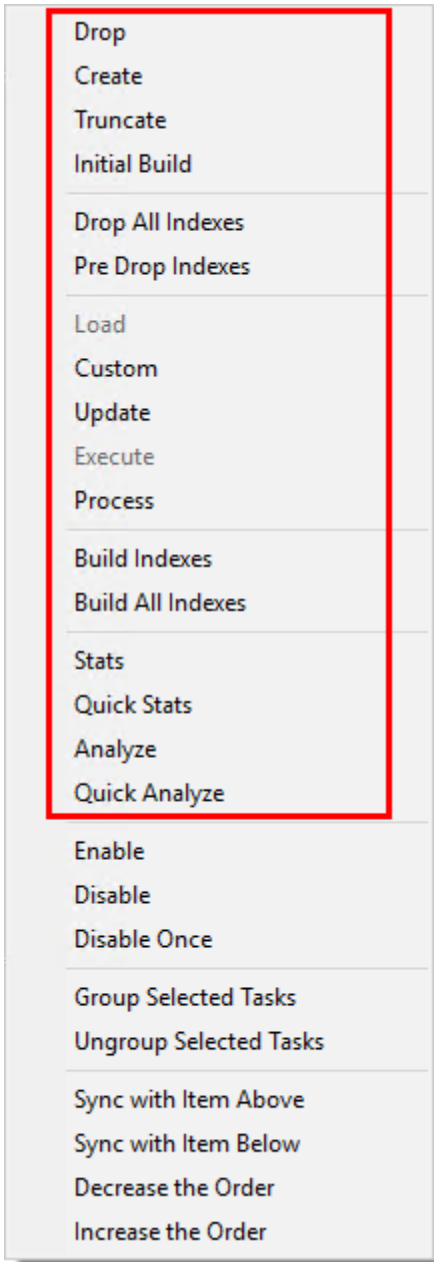
To set the action on a task

Each task can have a specific action that is to be performed on its object.

The default action for **load tables** is **process**. This means that when the task is actioned it will drop any indexes that are due to be dropped, or have **pre-drop** set, then load the table and perform any post-load procedures or transformations and then re-create any dropped indexes.

The default action for all other tables is the same as above, except it will execute the *update* procedure rather than loading the table.

You can change the action on a task by right-clicking on the task in the right pane. The menu options are shown below.



The following task actions are available:

Action	**Description**
Drop	Drop table, view or index.
Create	Create table, view or index.
Truncate	Delete all rows from the table.
Initial Build	**Drop All Indexes** then **Custom** then **Build All Indexes**.

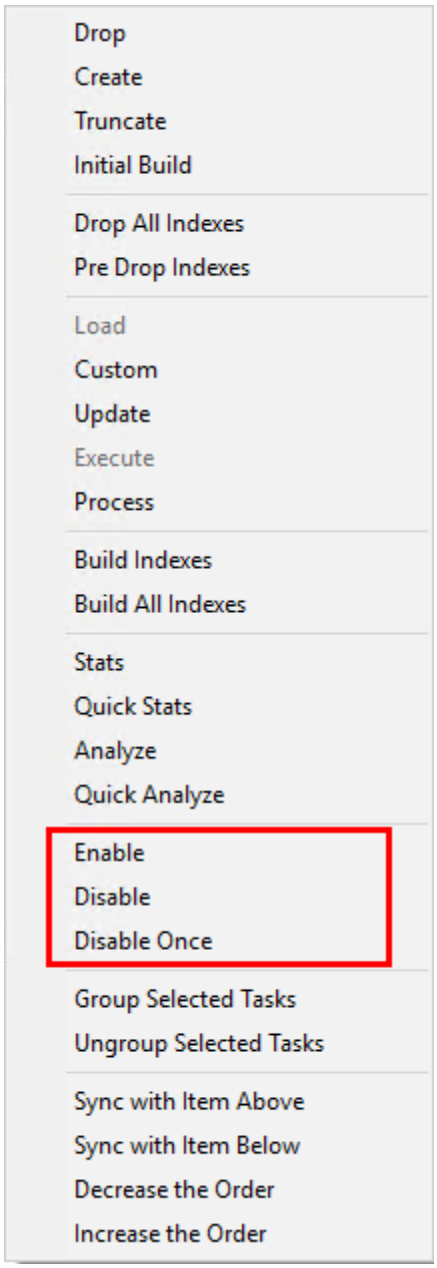
Action	Description
Drop All Indexes	Drop all indexes on the table.
Pre Drop Indexes	Drop all indexes on the table marked as "Pre Drop".
Load	Load the table (Load tables only).
Custom	Run the custom procedure on the table.
Update	Run the update procedure on the table.
Execute	Execute the procedure or host script.
Process	**Pre Drop Indexes** then **Update** and then **Build Indexes**.
Process and Statistics	**Process** then Default Stats as defined on Table Properties/ Statistics/Process and statistics method (DB2 only).
Build Indexes	Build the indexes on the table marked as "Pre Drop".
Build All Indexes	Build all indexes on the table.
DB2: Analyze	Performs a custom analyze if defined in: • Table Properties/Statistics/Analyze else for tables performs: • **RUNSTATS ON TABLE tabname WITH DISTRIBUTION AND DETAILED INDEXES ALL** otherwise (for indexes): • **RUNSTATS ON TABLE indname FOR INDEXES**
DB2: Quick Analyze	Performs a custom analyze if defined in: • Table Properties/Statistics/Quick Analyze else for tables performs: • **RUNSTATS ON TABLE tabname** otherwise (for indexes): • **RUNSTATS ON TABLE indname FOR INDEXES**
DB2: Stats	Performs a custom stats if defined in: • Table Properties/Statistics/Stats else for tables performs: • **RUNSTATS ON TABLE tabname WITH DISTRIBUTION AND DETAILED INDEXES ALL** otherwise (for indexes): • **RUNSTATS ON TABLE indname FOR INDEXES**
DB2: Quick Stats	Performs a custom analyze if defined in: • Table Properties/Statistics/Quick Stats else for tables performs: • **RUNSTATS ON TABLE tabname** otherwise (for indexes): • **RUNSTATS ON TABLE indname FOR INDEXES**

Action	Description
Oracle: Analyze	Performs a custom analyze if defined in: • Tools-Options-Statistics-Table/Index Analyze Full otherwise performs: • **ANALYZE TABLE / INDEX name COMPUTE STATISTICS**
Oracle: Quick Analyze	Performs a custom analyze if defined in: • Tools-Options-Statistics-Table/Index Analyze Quick otherwise performs: • **ANALYZE TABLE / INDEX name ESTIMATE STATISTICS SAMPLE 3 PERCENT**
Oracle: Stats	Performs a custom stats if defined in: • Tools-Options-Statistics-Table/Index Stats Full otherwise performs: • **DBMS_STATS.GATHER_ TABLE / INDEX _STATS()** using the ownname and tabname / indname parameters with cascade.
Oracle: Quick Stats	Performs a custom analyze if defined in: • Tools-Options-Statistics-Table/Index Stats Quick otherwise performs: • **DBMS_STATS.GATHER_ TABLE / INDEX _STATS()** using the ownname and tabname/indname parameters with cascade and estimate_percent = 3.
SQL Server: Analyze	Performs: • **UPDATE STATISTICS name WITH FULLSCAN**
SQL Server: Quick Analyze	Performs: • **UPDATE STATISTICS name WITH SAMPLE 3 PERCENT**
SQL Server: Stats	Same as Analyze.
SQL Server: Quick Stats	Same as Quick Analyze.

Note: Not all actions are available on all object types.

To set the state of a task

Each task can be set to a state:

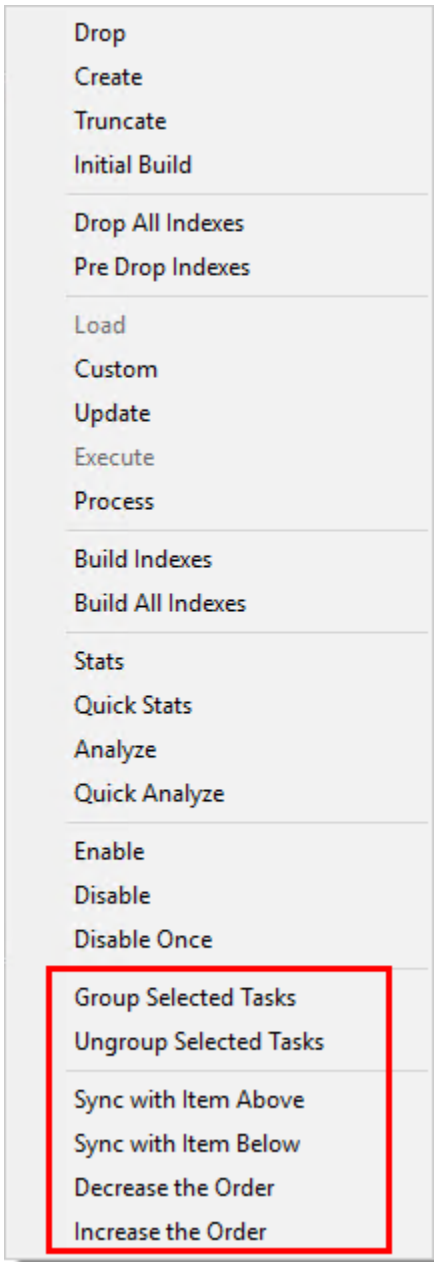


The following states are available:

State	Description
Enable	Job Task is enabled.
Disable	Job Task is disabled.
Disable Once	Job Task is disabled once and reverts to enabled next time the Job is released by the Scheduler.

To create dependencies between tasks

It is possible to create dependencies between tasks in the list by selecting one or more tasks and right-clicking to bring up the dependency options.



The following task dependency options are available from the menu:

Task Option	Description
Group Selected Tasks	Groups two or more selected tasks to have the same order value, allowing them to run in parallel if the maximum threads setting allows.
Ungroup Selected Tasks	Un-group selected tasks.

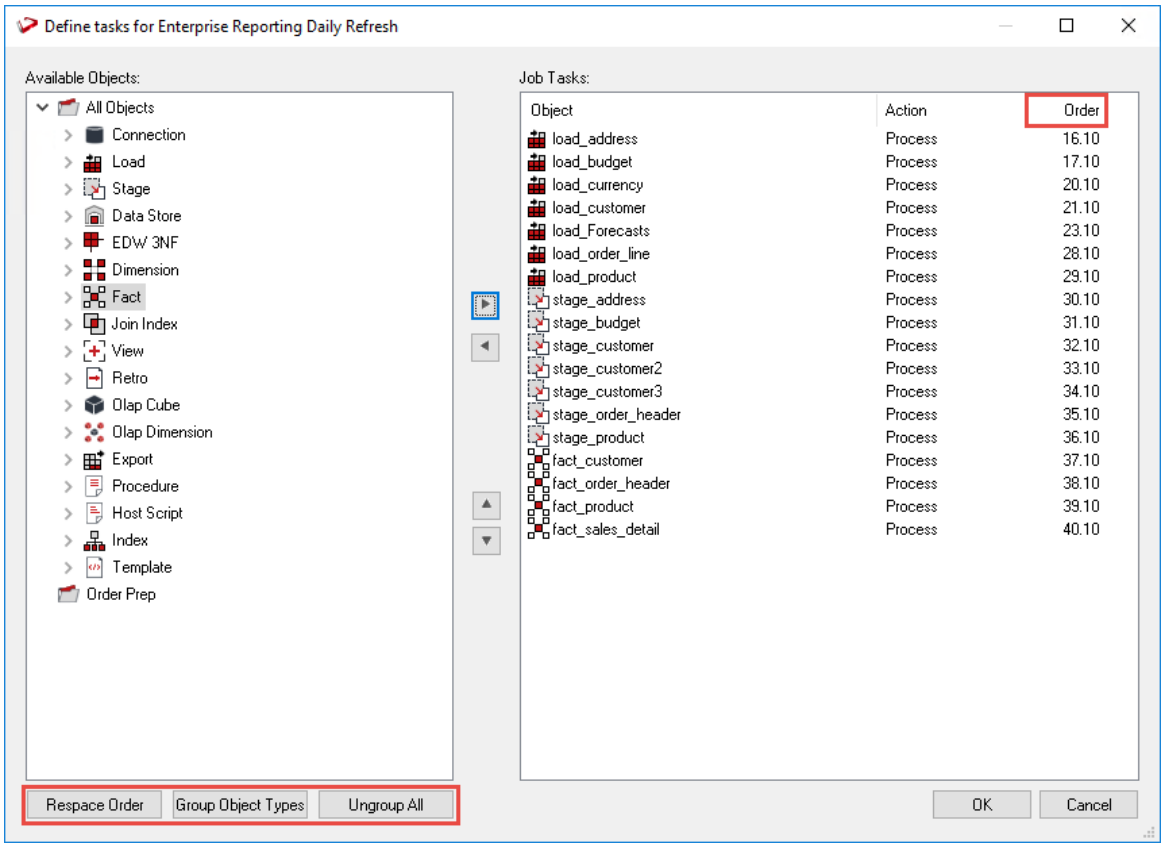
Task Option	Description
Sync with Item Above	Changes a selected task to have the same order value as the task above it, allowing them to run in parallel if the maximum threads setting allows.
Sync with Item Below	Changes a selected task to have the same order value as the task below it, allowing them to run in parallel if the maximum threads setting allows.
Decrease the Order	Changes a selected task to an order number one less than its current value. The task will now run immediately before it would have previously.
Increase the Order	Changes a selected task to an order number one more than its current value. The task will now run immediately after it would have previously.

To order or group the tasks

The **Order** column shows the order in which the tasks are to be run, e.g. 20.20 If the two numbers are the same as another task then those tasks can run in parallel. If the two numbers are different, then those tasks run sequentially. This is an initial definition of dependencies. These dependencies can be altered specifically once the job has been created.

Tasks can be moved up or down by selecting the task and clicking the **Move Up**  or **Move Down**  buttons.

To respace the order of the tasks; to group or ungroup object types, use the **buttons** at the bottom of the **Define tasks** dialog.



- **Respace Order**

  This button will respace the order numbers. The existing dependency structure and groupings are retained. The purpose of this button is simply to allow room between tasks to fit new tasks. So for example if we have two tasks that have an order of 20.19.5 and 20.20.6 and we want to add a task between these two tasks we can click the **Respace Order** button to open up a gap between the two tasks.

- **Group Object Types**

  This option will put all objects of the same type into groups. For example all load tables will be able to run in parallel, all dimensions etc.

- **Ungroup All**

  This button will remove all groupings and make all tasks sequential. New groupings can be made by selecting a range of sequentially listed tasks in the left pane and using the right-click menu option **Group Selected Tasks**. Tasks that are grouped have the same first two numbers in the order and can execute at the same time if the job has multiple threads.

Upon completion of adding tasks, click **OK**.

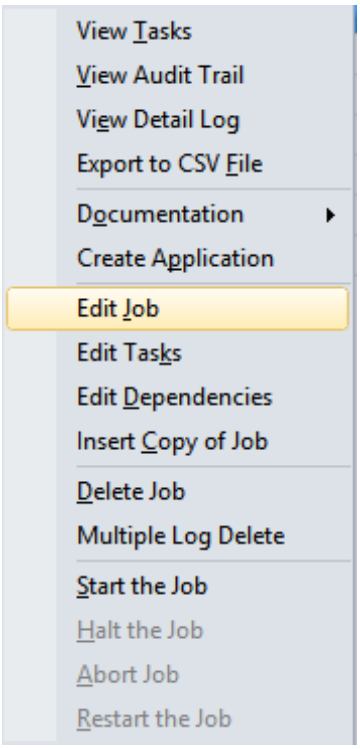
EDITING A JOB

Once jobs have been created they can be edited.

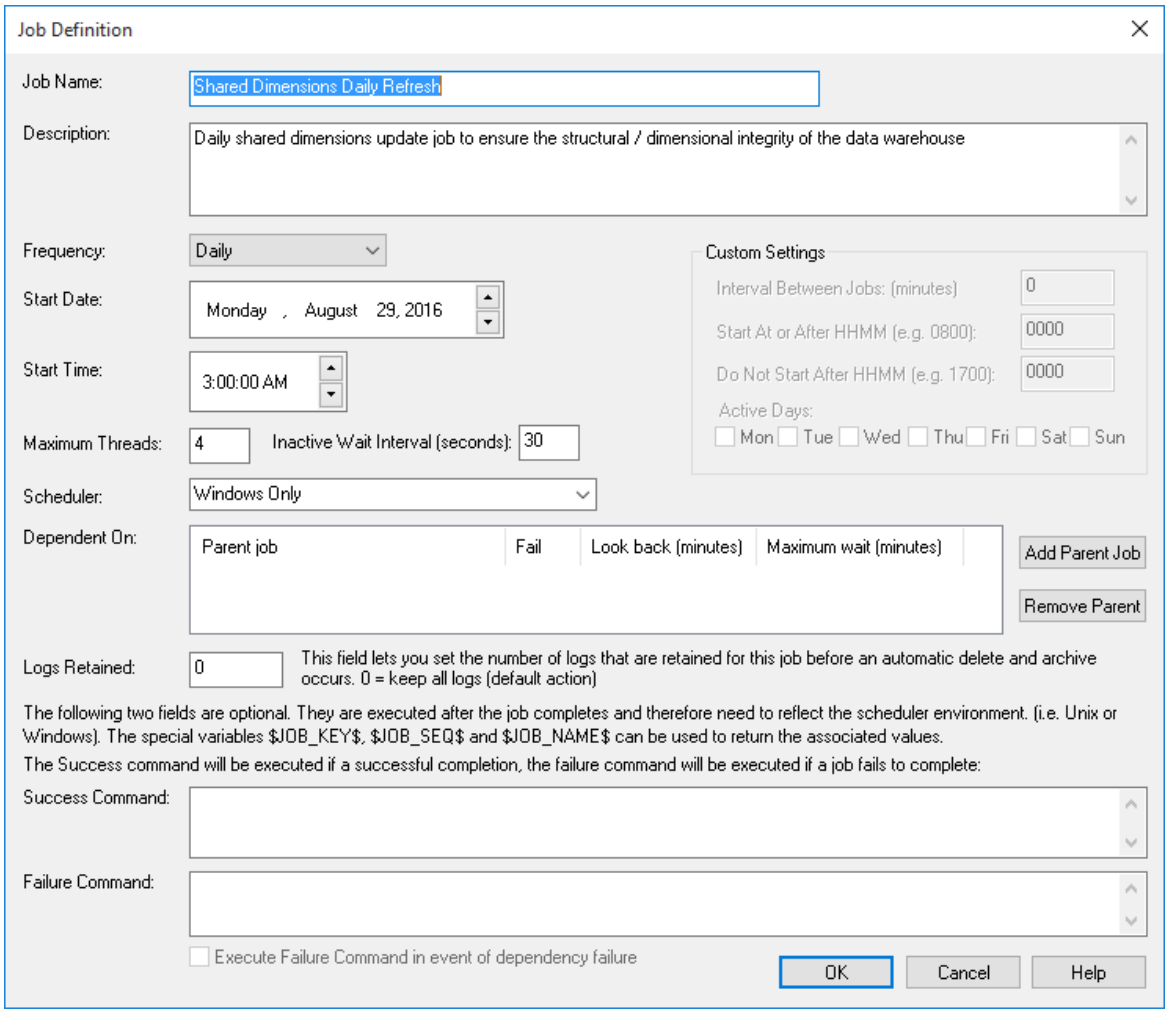
Note: A job can only be edited when it is not in a running state and only if the job is a scheduled job. Completed jobs remain in the list but only logs remain.

To edit a job

Select the job from the scheduler middle pane. Right-click on the job and select **Edit Job** from the drop-down list.



The **Job Definition** will be displayed.



Edit the fields as required and click **OK**. The main fields are described in the following table:

Field	Description
Job Name	The Scheduler defaults to the next job number in the sequence. You can alter this to any alphanumeric. **Note**: Only alphanumerics, spaces and the underscore are supported in the name. WARNING: on some UNIX systems, long job names can cause jobs to be canceled (see Knowledge Base article 67), so where possible keep the name short.
Description	A description of the job

Field	Description
Frequency	When the job runs. The options available in the drop-down list box are: • Once Only - job is deleted on completion • Once and Hold - runs and puts another copy of the job on hold • Hold - puts the job on hold for manual release • Daily - runs the job daily • Custom - enables custom definition • Weekly - runs the job weekly • Monthly - runs the job monthly • Annually - runs the job annually
Start Date and Start Time	• The date and time for the job to start.
Max Threads	The maximum number of threads allocated to run the job, e.g, if some tasks can run in parallel then if more than one thread is allocated then they will run in parallel.
Scheduler	Certain types of job will only run in a specific environment. For example, ODBC based loads can only be handled by the Windows scheduler. It is possible to have multiple schedulers running. Select the desired scheduler from this drop-down. The valid options are: UNIX Preferred, UNIX Only, Windows Preferred, Windows Only, or the name of a specific scheduler can be entered (e.g. WIN0002)
Dependent On	A job can be dependent on the successful completion of one or more other jobs. Click the **Add Parent Job** button to select a job that this job will be dependent on. The maximum time to look back for parent job completion field prevents older iterations of the parent job as being identified as a completion. The maximum time to wait specifies how long to await a successful completion of the parent job. The action if that wait expires can also be set. See the Job Dependency example in ***Scheduling a Job*** (on page 744).
Logs Retained	Specify the number of logs to retain for the job. By default, all logs are retained. This field can be used to reduce the build up of scheduler logs by specifying a number of logs to retain.
OK command and Failure command	These are either UNIX or Windows shell commands depending on which scheduler is used. They are executed if the condition is met. Typically, these commands would mail or page on success or failure. **Notes**: 1. The RED scheduler does not check return codes from called commands, scripts and programs. 2. It is recommended that all output from commands, scripts and programs is redirected to a log file. For example, add this to the end of any SUCCESS/FAILURE commands: >> c:\scheduler\success_failure_prod.log 2>&1

The following fields are available if a frequency of **Custom** is chosen:

Field	Description
Interval between jobs (Minutes)	Specify the number of minutes between iterations of the job. For example, to run a job every 30 minutes set this value to 30. If a job is to run only once but on selected days set this value to 1440 (daily)
Start at or after HHMM	The time that the job may run from. To run anytime set to 0000.
Do not start after HHMM	If multiple iterations are being done, then this is the time after which a new iteration will not be started. For example, if a job is running every 10 minutes it will continue until this time is reached. To run till the end of day set to 2400.
Active on the days	Select each day of the week that the custom job is to be active on.

EDITING TASKS IN A JOB

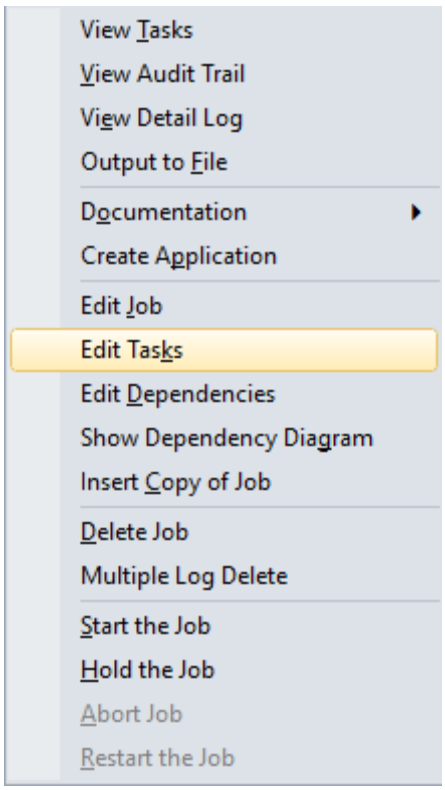
Once jobs have been created, you can edit their tasks.

Note: A job can only be edited when it is not in a running state and only if the job is a scheduled job. Completed jobs remain in the list but only logs remain.

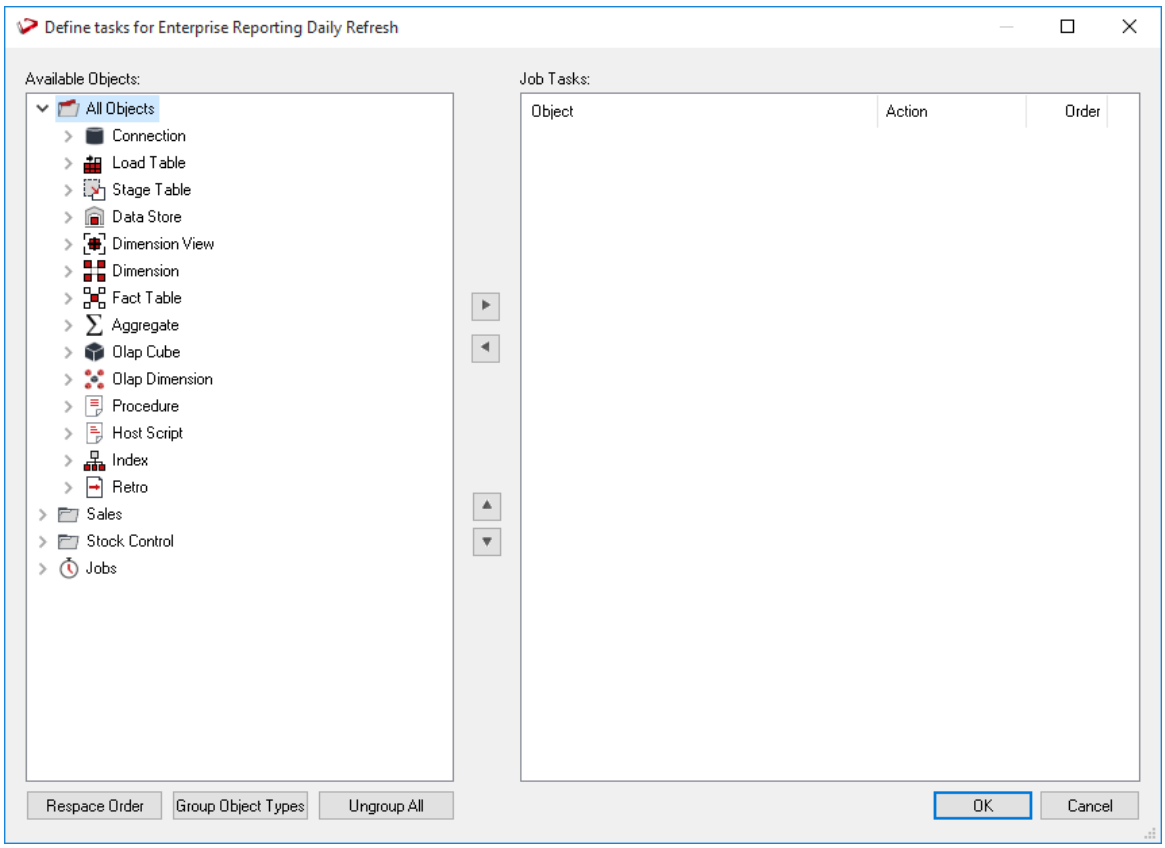
JOB TASK LIMIT - There is maximum number of **999 tasks** that can be added to a job.

To edit the tasks of a job

Select the job from the scheduler middle pane. Right-click on the job and select **Edit Tasks** from the drop-down list.



The **Define tasks** window is shown below.



The screen has two main areas. The right pane shows the tasks to be run for this job and the left pane lists all the objects.

To add a task

Double-click on an object in the left pane to add it to the task list in the right pane. Normally objects such as load or fact tables are scheduled rather than procedures.

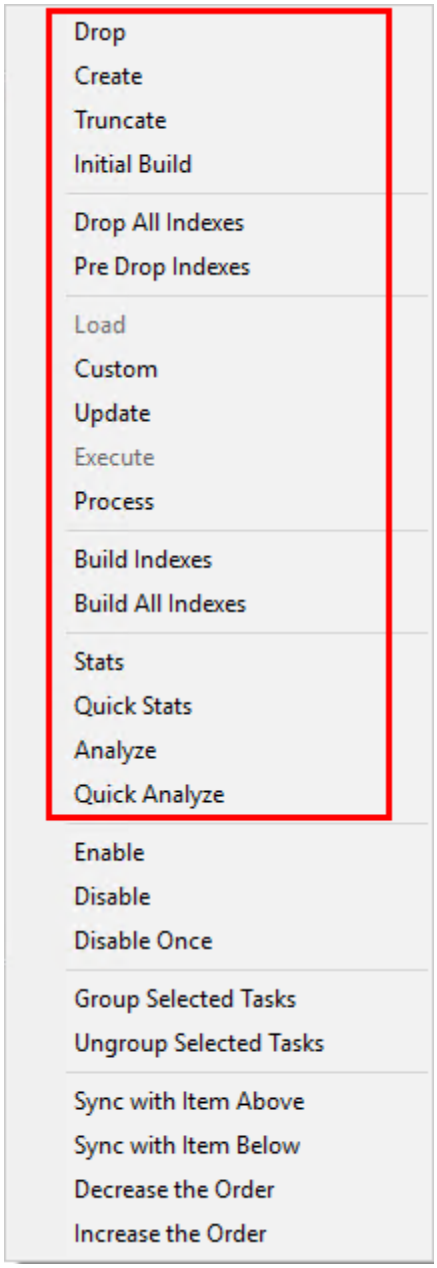
To set the action on a task

Each task can have a specific action that is to be performed on its object.

The default action for **load tables** is **process**. This means that when the task is actioned it will drop any indexes that are due to be dropped, or have **pre-drop** set, then load the table and perform any post-load procedures or transformations and then re-create any dropped indexes.

The default action for all other tables is the same as above, except it will execute the *update* procedure rather than loading the table.

You can change the action on a task by right-clicking on the task in the right pane. The menu options are shown below.



The following task actions are available:

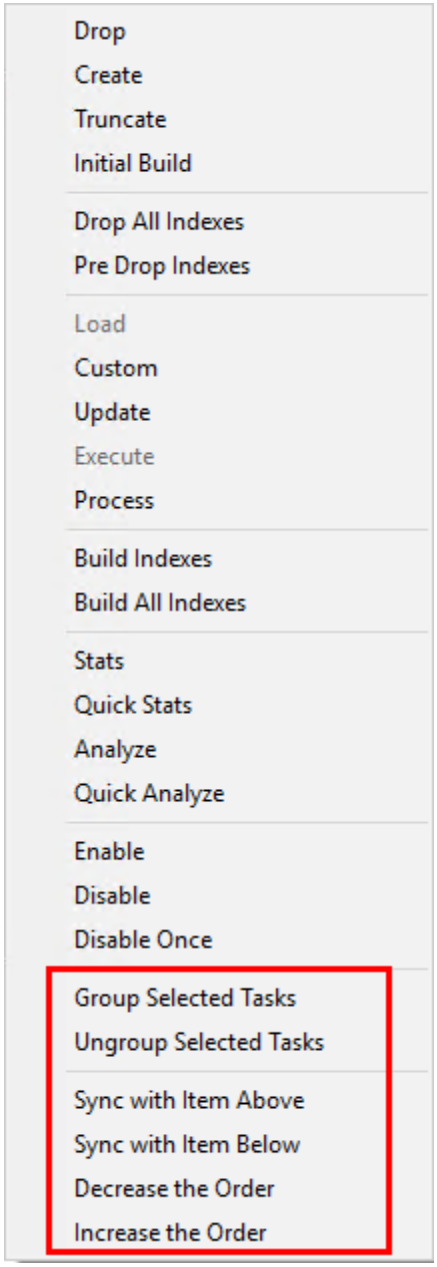
Action	Description
Drop	Drop table, view or index.
Create	Create table, view or index.
Truncate	Delete all rows from the table.
Initial Build	**Drop All Indexes** then **Custom** then **Build All Indexes**.
Drop All Indexes	Drop all indexes on the table.
Pre Drop Indexes	Drop all indexes on the table marked as "Pre Drop".
Load	Load the table (Load tables only).
Custom	Run the custom procedure on the table.
Update	Run the update procedure on the table.
Execute	Execute the procedure or host scripts.
Process	**Pre Drop Indexes** then **Update** and then **Build Indexes**.
Process and Statistics	**Process** then Default Stats as defined on Table Properties/ Statistics/Process and statistics method (DB2 only).
Build Indexes	Build the indexes on the table marked as "Pre Drop".
Build All Indexes	Build all indexes on the table.
DB2: Analyze	Performs a custom analyze if defined in: • Table Properties/Statistics/Analyze else for tables performs: • **RUNSTATS ON TABLE tabname WITH DISTRIBUTION AND DETAILED INDEXES ALL** otherwise (for indexes): • **RUNSTATS ON TABLE indname FOR INDEXES**
DB2: Quick Analyze	Performs a custom analyze if defined in: • Table Properties/Statistics/Quick Analyze else for tables performs: • **RUNSTATS ON TABLE tabname** otherwise (for indexes): • **RUNSTATS ON TABLE indname FOR INDEXES**
DB2: Stats	Performs a custom stats if defined in: • Table Properties/Statistics/Stats else for tables performs: • **RUNSTATS ON TABLE tabname WITH DISTRIBUTION AND DETAILED INDEXES ALL** otherwise (for indexes): • **RUNSTATS ON TABLE indname FOR INDEXES**

Action	Description
DB2: Quick Stats	Performs a custom analyze if defined in: • Table Properties/Statistics/Quick Stats else for tables performs: • **RUNSTATS ON TABLE tabname** otherwise (for indexes): • **RUNSTATS ON TABLE indname FOR INDEXES**
Oracle: Analyze	Performs a custom analyze if defined in: • Tools-Options-Statistics-Table/Index Analyze Full otherwise performs: • **ANALYZE TABLE / INDEX name COMPUTE STATISTICS**
Oracle: Quick Analyze	Performs a custom analyze if defined in: • Tools-Options-Statistics-Table/Index Analyze Quick otherwise performs: • **ANALYZE TABLE / INDEX name ESTIMATE STATISTICS SAMPLE 3 PERCENT**
Oracle: Stats	Performs a custom stats if defined in: • Tools-Options-Statistics-Table/Index Stats Full otherwise performs: • **DBMS_STATS.GATHER_ TABLE / INDEX _STATS()** using the ownname and tabname / indname parameters with cascade.
Oracle: Quick Stats	Performs a custom analyze if defined in: • Tools-Options-Statistics-Table/Index Stats Quick otherwise performs: • **DBMS_STATS.GATHER_ TABLE / INDEX _STATS()** using the ownname and tabname/indname parameters with cascade and estimate_percent = 3.
SQL Server: Analyze	Performs: • **UPDATE STATISTICS name WITH FULLSCAN**
SQL Server: Quick Analyze	Performs: • **UPDATE STATISTICS name WITH SAMPLE 3 PERCENT**
SQL Server: Stats	Same as Analyze.
SQL Server: Quick Stats	Same as Quick Analyze.

Note: Not all actions are available on all object types.

To create dependencies between tasks

It is possible to create dependencies between tasks in the list by selecting one or more tasks and right-clicking to bring up the dependency options.



The following task dependency options are available from the menu:

Task Option	Description

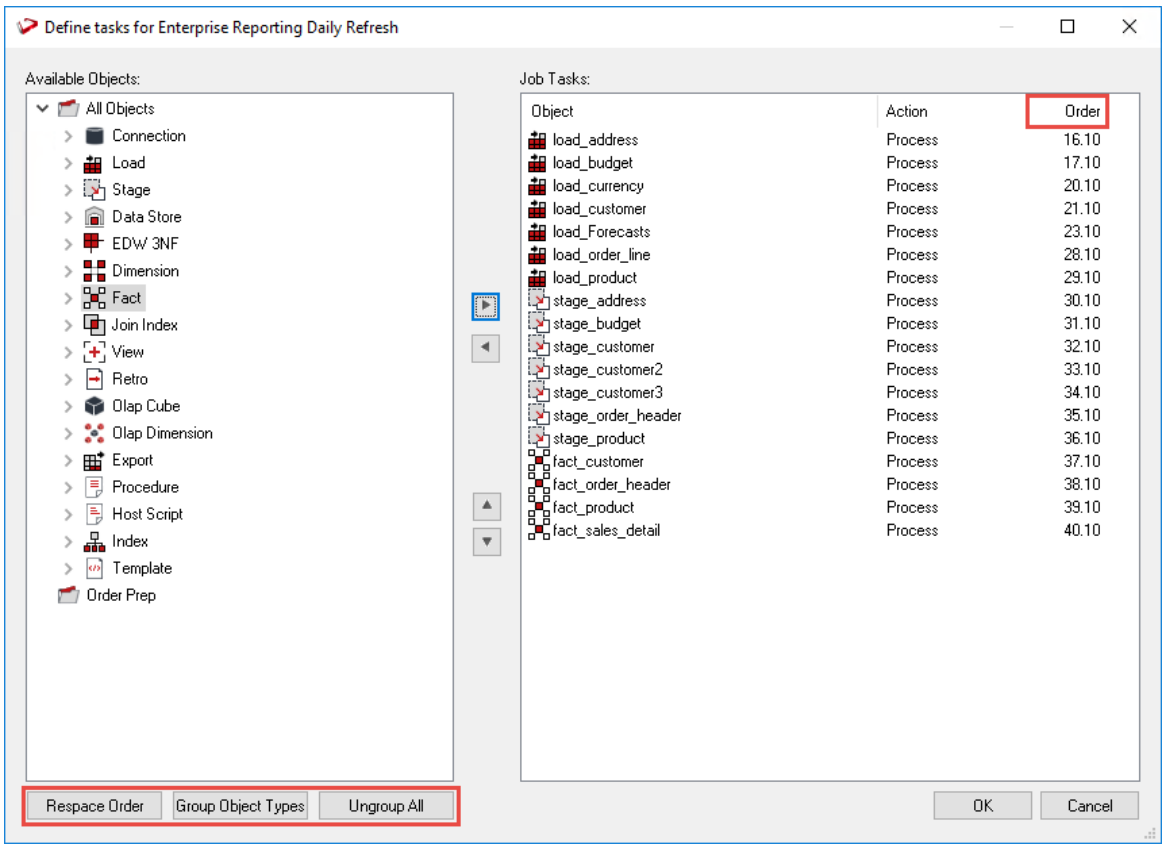
Task Option	Description
Group Selected Tasks	Groups two or more selected tasks to have the same order value, allowing them to run in parallel if the maximum threads setting allows.
Ungroup Selected Tasks	Un-group selected tasks.
Sync with Item Above	Changes a selected task to have the same order value as the task above it, allowing them to run in parallel if the maximum threads setting allows.
Sync with Item Below	Changes a selected task to have the same order value as the task below it, allowing them to run in parallel if the maximum threads setting allows.
Decrease the Order	Changes a selected task to an order number one less than its current value. The task will now run immediately before it would have previously.
Increase the Order	Changes a selected task to an order number one more than its current value. The task will now run immediately after it would have previously.

To order or group the tasks

The **Order** column shows the order in which the tasks are to be run, e.g. 20.20 If the two numbers are the same as another task then those tasks can run in parallel. If the two numbers are different then those tasks run sequentially. This is an initial definition of dependencies. These dependencies can be altered specifically once the job has been created.

Tasks can be moved up or down by selecting the task and clicking the **Move Up**  or **Move Down**  buttons.

To re-space the order of the tasks; to group or un-group object types, use the **buttons** at the bottom of the **Define tasks** dialog.



- **Respace Order**

  This button will respace the order numbers. The existing dependency structure and groupings are retained. The purpose of this button is simply to allow room between tasks to fit new tasks. So for example if we have two tasks that have an order of 20.19.5 and 20.20.6 and we want to add a task between these two tasks we can click the **Respace Order** button to open up a gap between the two tasks.

- **Group Object Types**

  This option will put all objects of the same type into groups. For example all load tables will be able to run in parallel, all dimensions etc.

- **Ungroup All**

  This button will remove all groupings and make all tasks sequential. New groupings can be made by selecting a range of sequentially listed tasks in the left pane and using the right-click menu option **Group Selected Tasks**. Tasks that are grouped have the same first two numbers in the order and can execute at the same time if the job has multiple threads.

Upon completion of editing tasks, click **OK**.

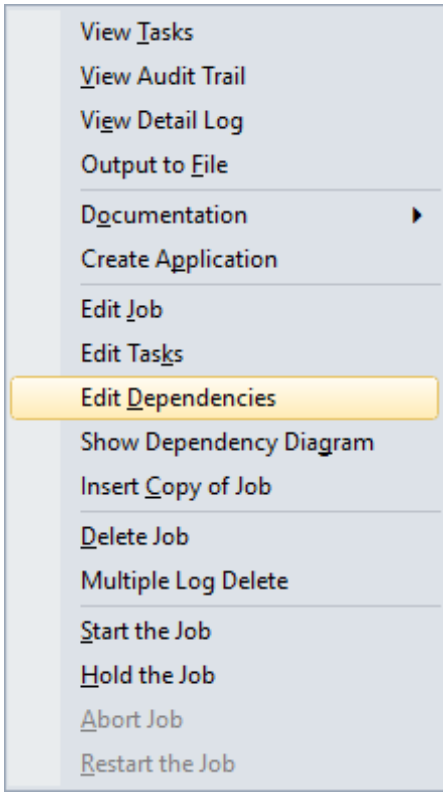
EDITING TASK DEPENDENCIES

Once jobs have been created they can be edited.

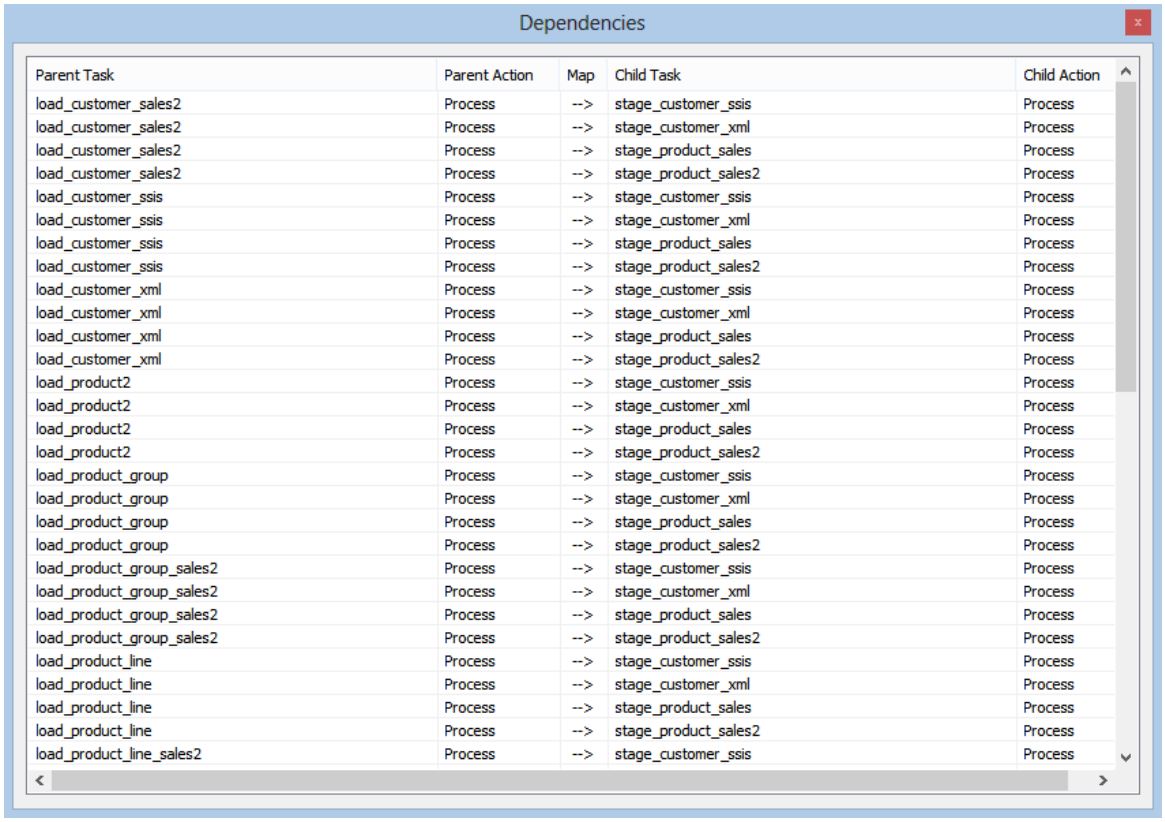
Note: A job can only be edited when it is not in a running state and only if the job is a scheduled job. Completed jobs remain in the list but only logs remain.

To edit task dependencies

Select the job from the scheduler middle pane. Right-click on the job and select **Edit Dependencies** from the drop-down list.



The **Dependencies** dialog will be displayed, showing the dependencies between the tasks of the job. The list consists of **Parent Tasks** on the left and **Child Tasks** on the right. A child task is thus dependent on its parent task in that it cannot run until its parent has run.

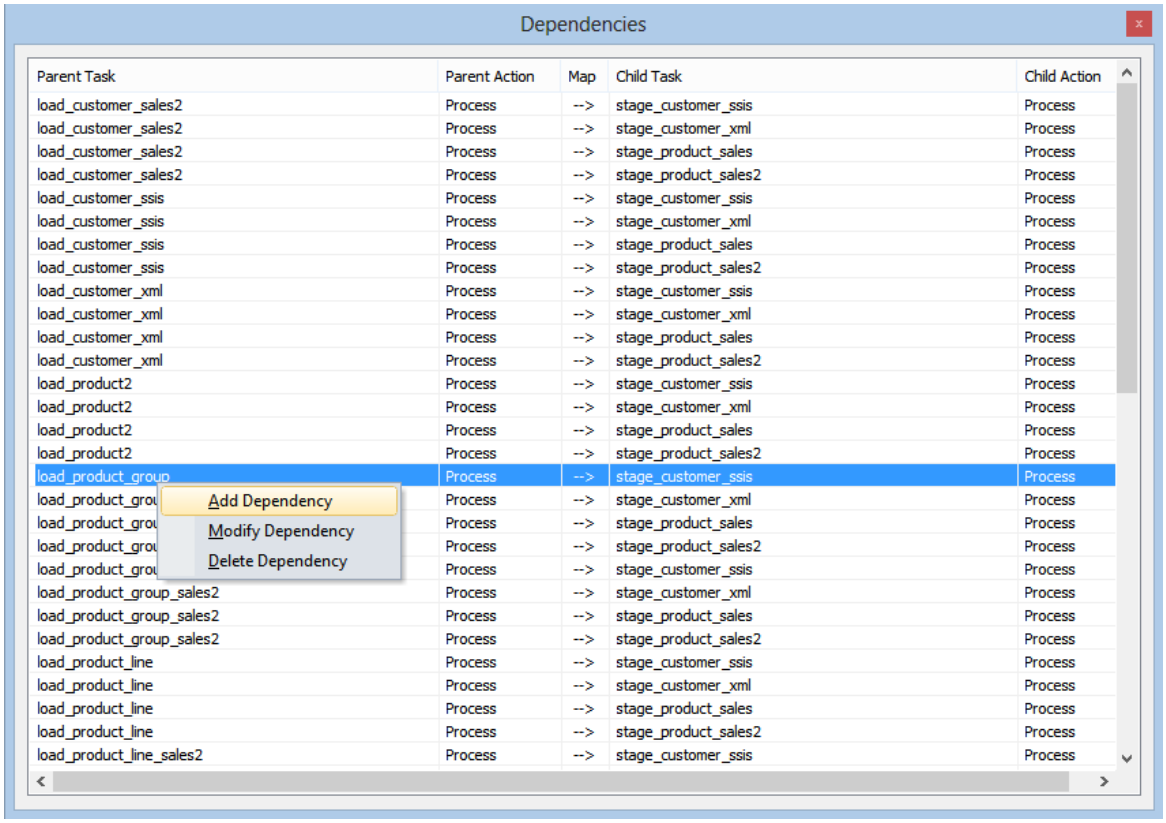


Parent Task	Parent Action	Map	Child Task	Child Action
load_customer_sales2	Process	-->	stage_customer_ssis	Process
load_customer_sales2	Process	-->	stage_customer_xml	Process
load_customer_sales2	Process	-->	stage_product_sales	Process
load_customer_sales2	Process	-->	stage_product_sales2	Process
load_customer_ssis	Process	-->	stage_customer_ssis	Process
load_customer_ssis	Process	-->	stage_customer_xml	Process
load_customer_ssis	Process	-->	stage_product_sales	Process
load_customer_ssis	Process	-->	stage_product_sales2	Process
load_customer_xml	Process	-->	stage_customer_ssis	Process
load_customer_xml	Process	-->	stage_customer_xml	Process
load_customer_xml	Process	-->	stage_product_sales	Process
load_customer_xml	Process	-->	stage_product_sales2	Process
load_product2	Process	-->	stage_customer_ssis	Process
load_product2	Process	-->	stage_customer_xml	Process
load_product2	Process	-->	stage_product_sales	Process
load_product2	Process	-->	stage_product_sales2	Process
load_product_group	Process	-->	stage_customer_ssis	Process
load_product_group	Process	-->	stage_customer_xml	Process
load_product_group	Process	-->	stage_product_sales	Process
load_product_group	Process	-->	stage_product_sales2	Process
load_product_group_sales2	Process	-->	stage_customer_ssis	Process
load_product_group_sales2	Process	-->	stage_customer_xml	Process
load_product_group_sales2	Process	-->	stage_product_sales	Process
load_product_group_sales2	Process	-->	stage_product_sales2	Process
load_product_line	Process	-->	stage_customer_ssis	Process
load_product_line	Process	-->	stage_customer_xml	Process
load_product_line	Process	-->	stage_product_sales	Process
load_product_line	Process	-->	stage_product_sales2	Process
load_product_line_sales2	Process	-->	stage_customer_ssis	Process

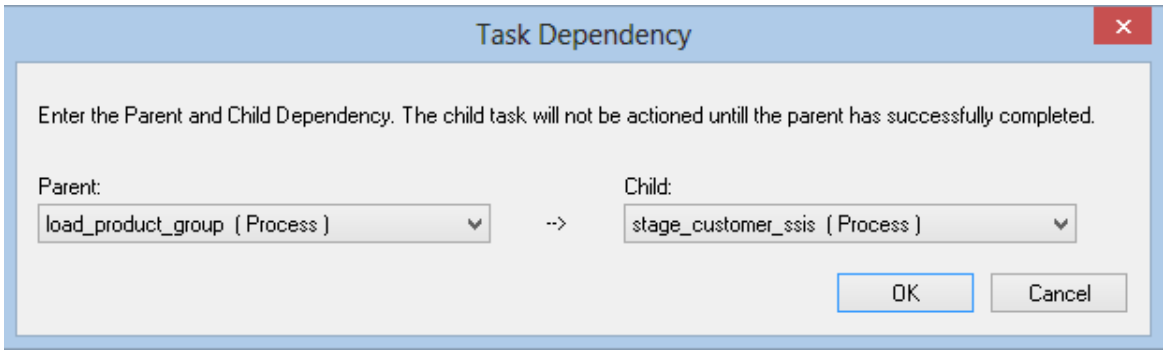
Edit the dependencies and close the dialog.

To add a task dependency

To add a task dependency, right-click anywhere in the Dependencies pane and select **Add Dependency**.

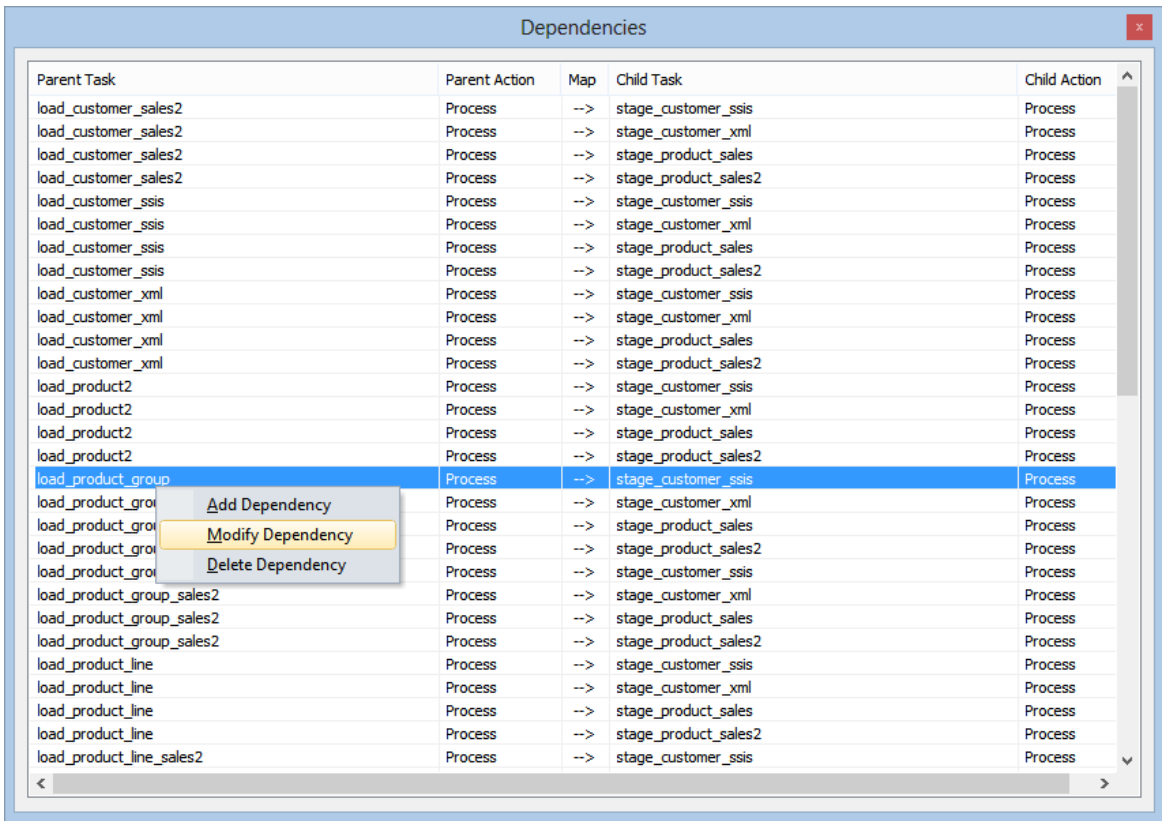


Select the **Parent** and **Child** tasks from the drop-down lists to create the dependency and click **OK**.

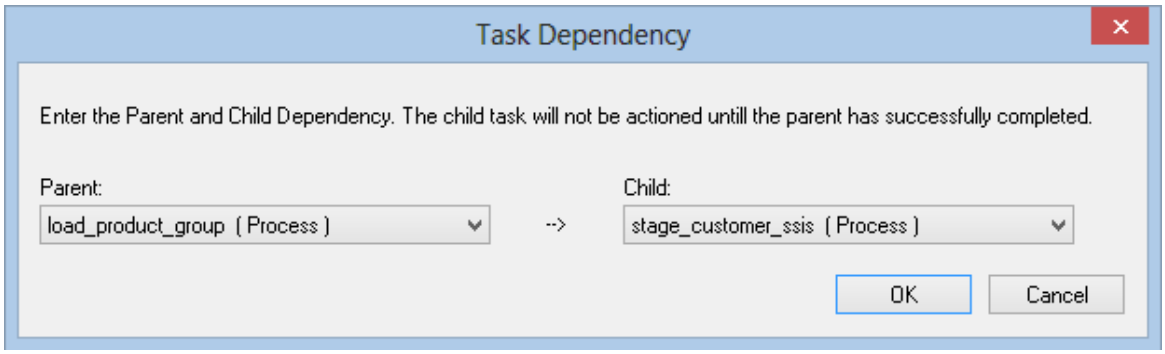


To modify a task dependency

To modify a task dependency, right-click on the dependency in the Dependencies pane and select **Modify Dependency**.

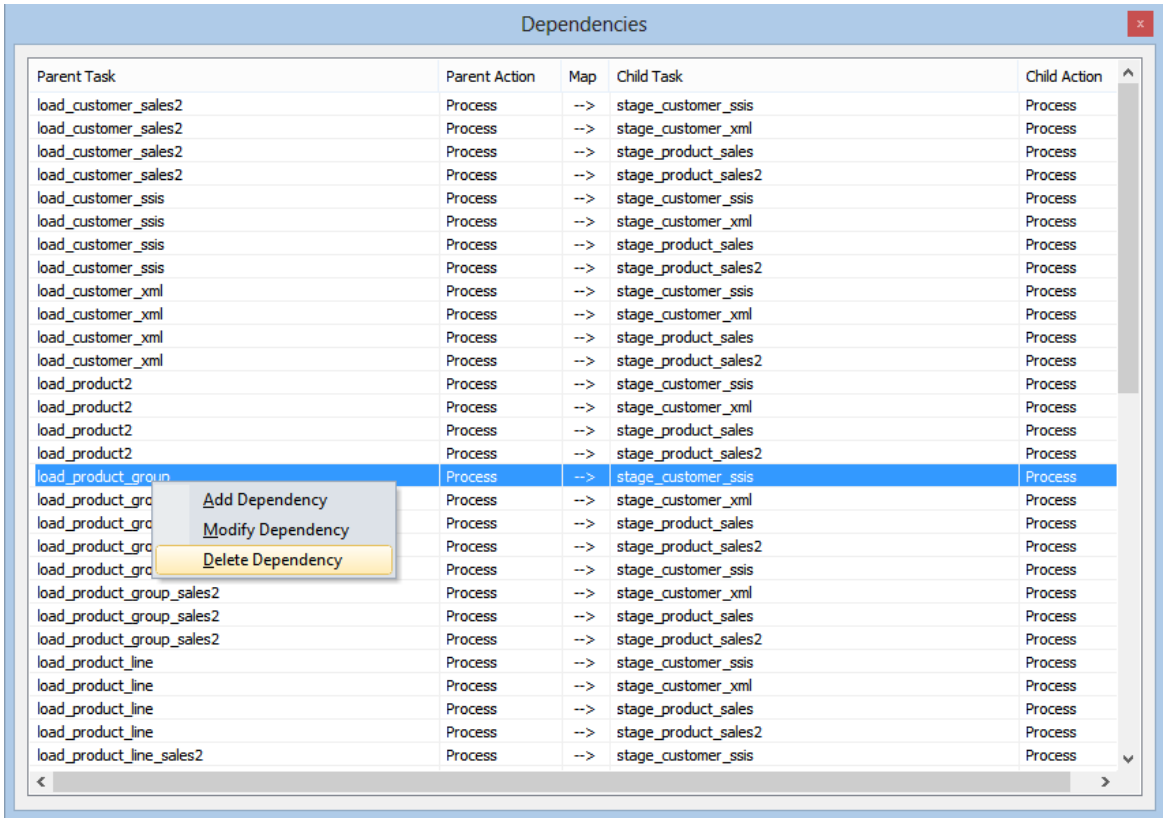


Change the **Parent** and **Child** tasks to modify the dependency and click **OK**.



To delete a task dependency

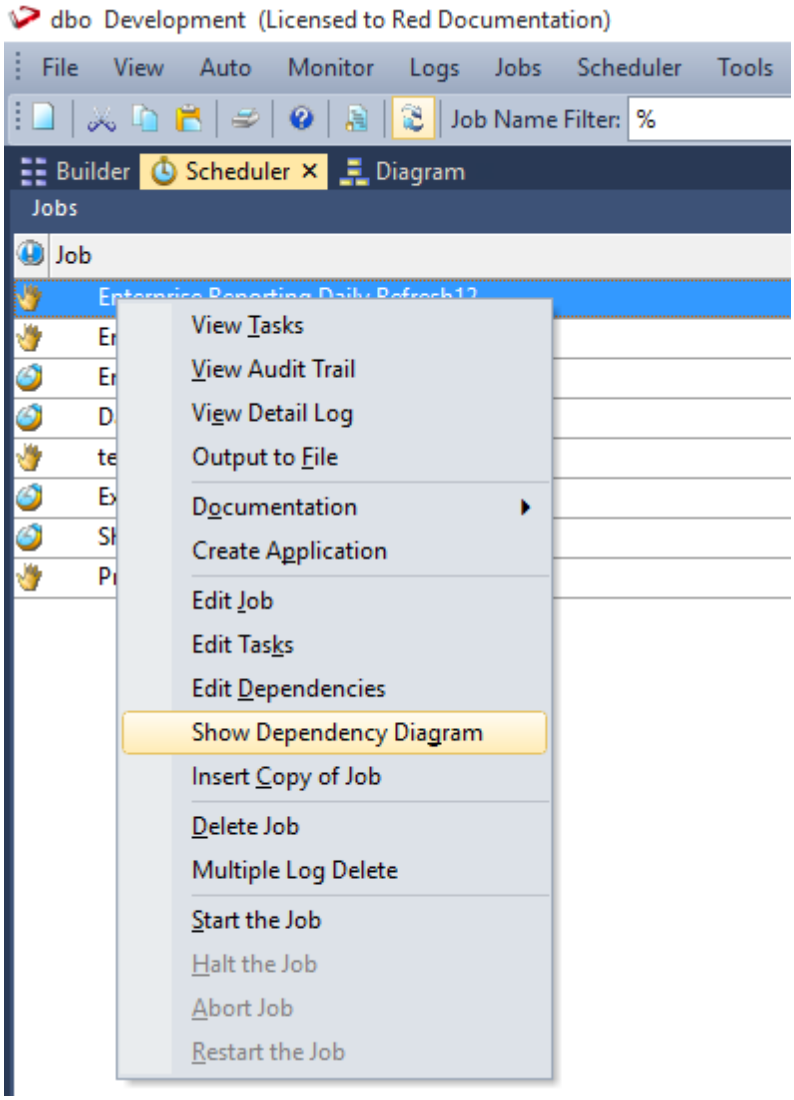
To delete a task dependency, right-click on the dependency in the Dependencies pane and select **Delete Dependency**.



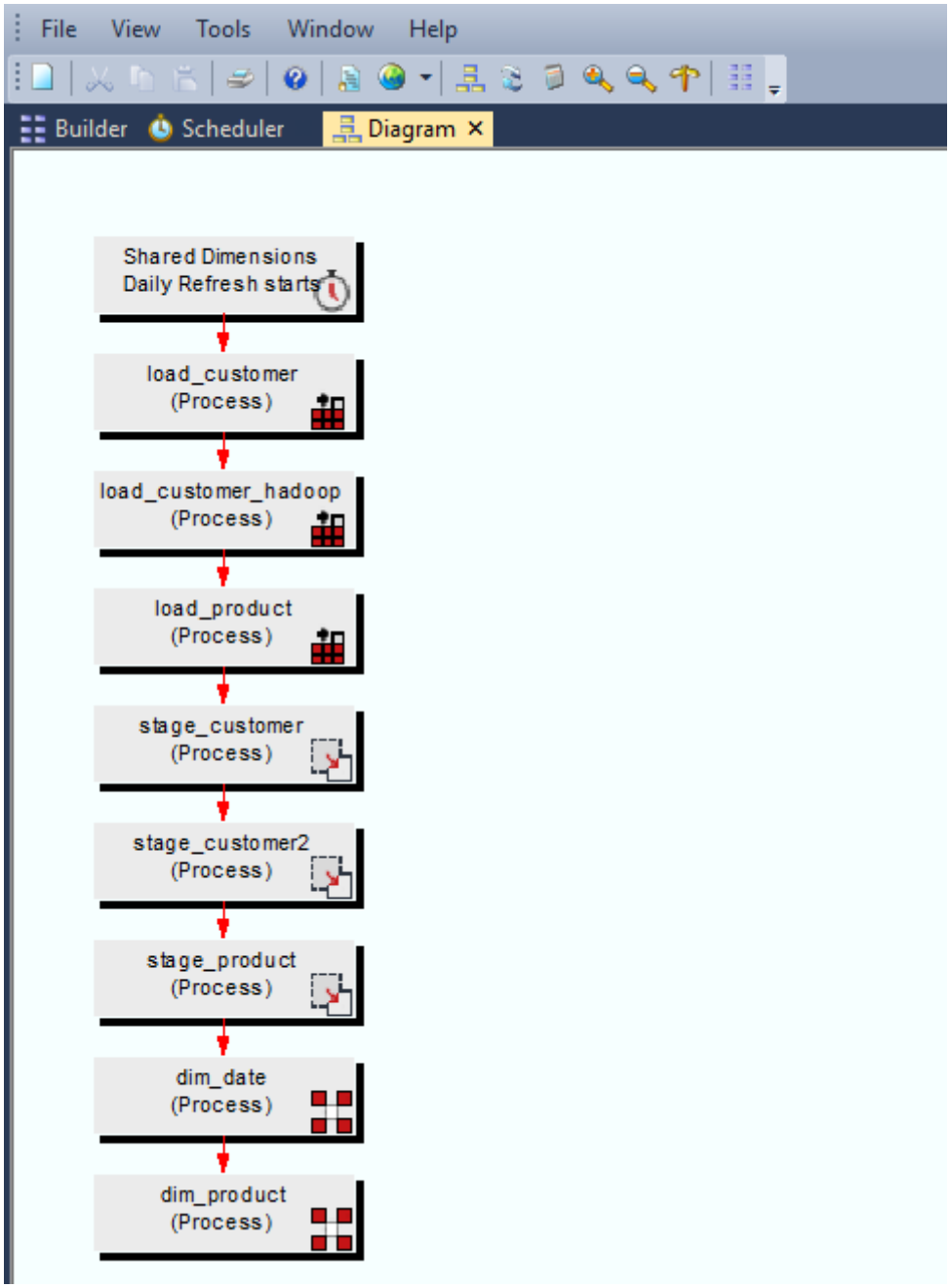
The dependency will be deleted without warning.

SHOW DEPENDENCIES DIAGRAM

Select the **Show Dependency Diagram** option from the right-click menu of any job to see all job dependencies displayed as a Diagram from RED's Diagram view tab.



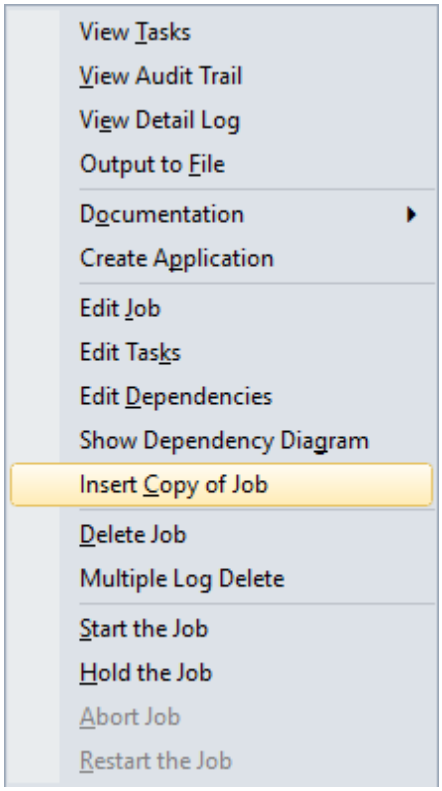
Job Dependency Diagram view


File
View
Tools
Window
Help
Builder
Scheduler
Diagram
Shared Dimensions
Daily Refresh starts
load_customer
(Process)
load_customer_hadoop
(Process)
load_product
(Process)
stage_customer
(Process)
stage_customer2
(Process)
stage_product
(Process)
dim_date
(Process)
dim_product
(Process)

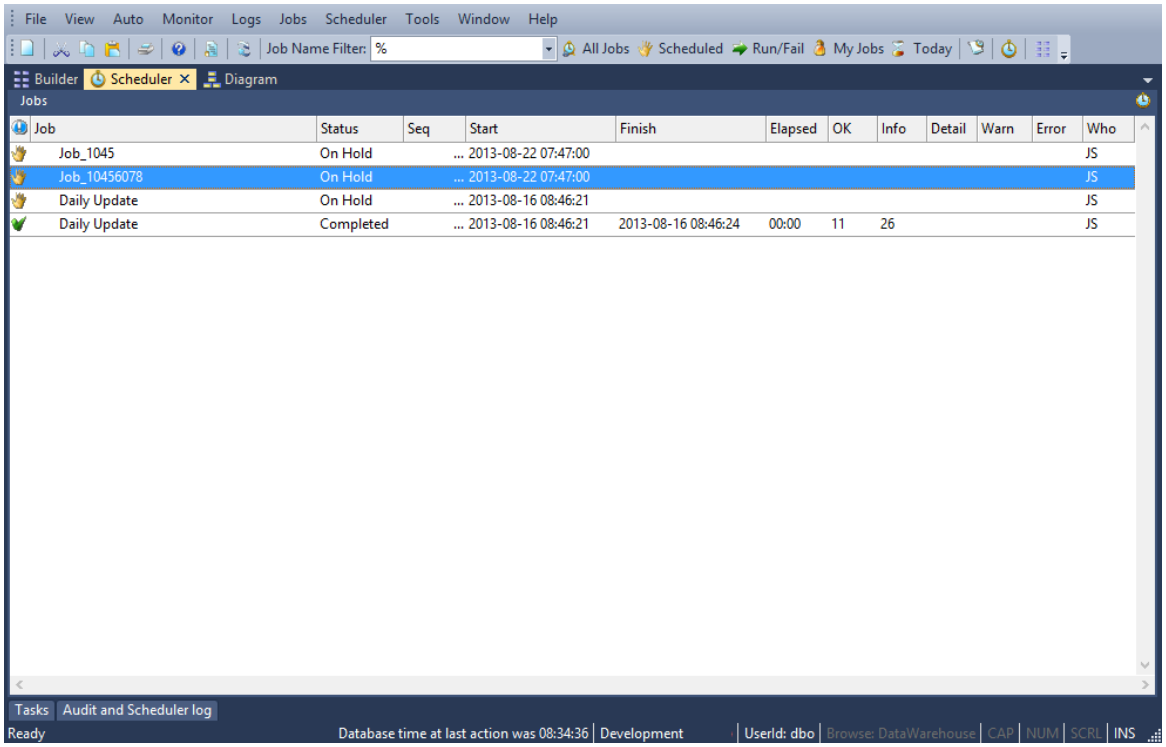
INSERTING A COPY OF A JOB

To insert a copy of a job

A copy of a job can be inserted by right-clicking on the job and choosing **Insert Copy of Job**.



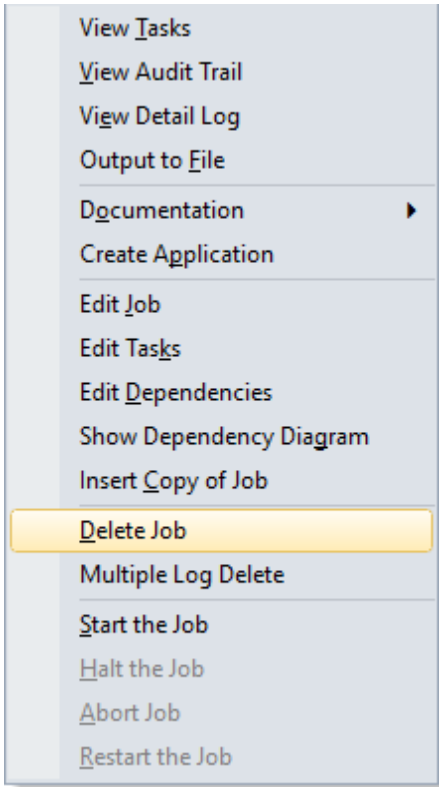
The new job will immediately be visible and the Status will be **On Hold**.



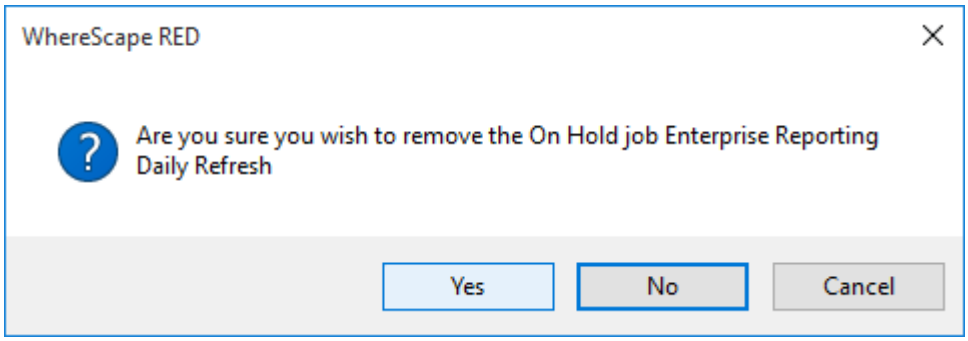
DELETING A JOB

To delete a job

A job can be deleted by right-clicking on a job in the scheduler window and choosing **Delete Job**.



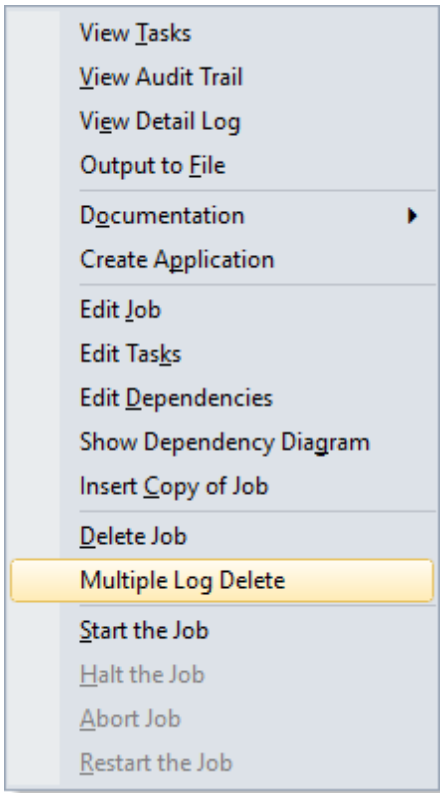
A warning message will be displayed; click **Yes** to delete.



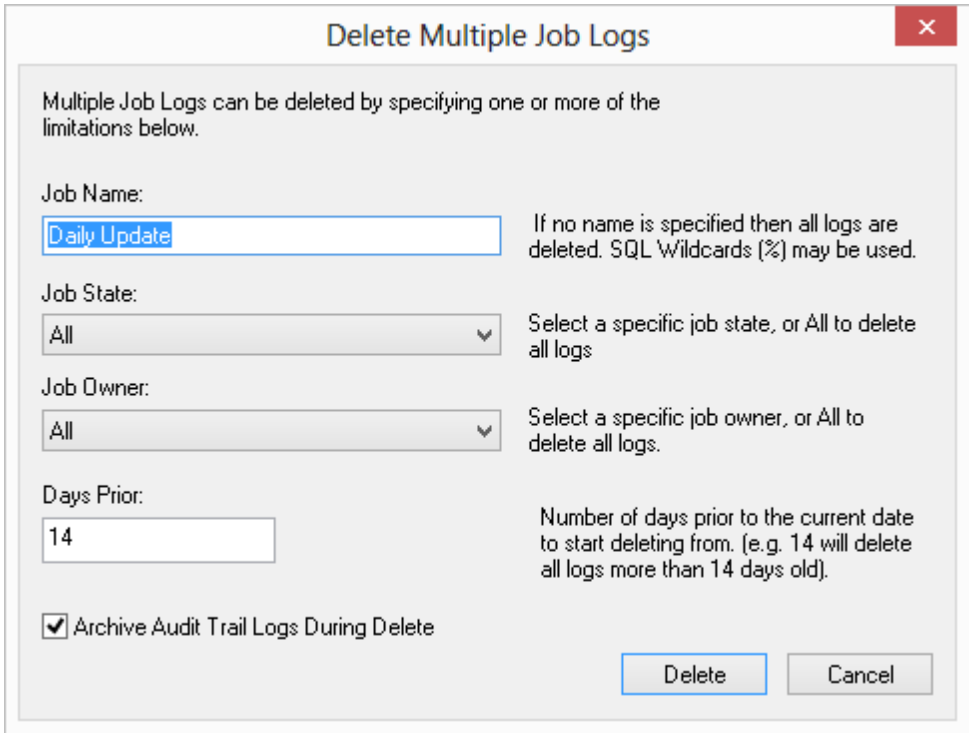
DELETING JOB LOGS

To delete multiple job logs

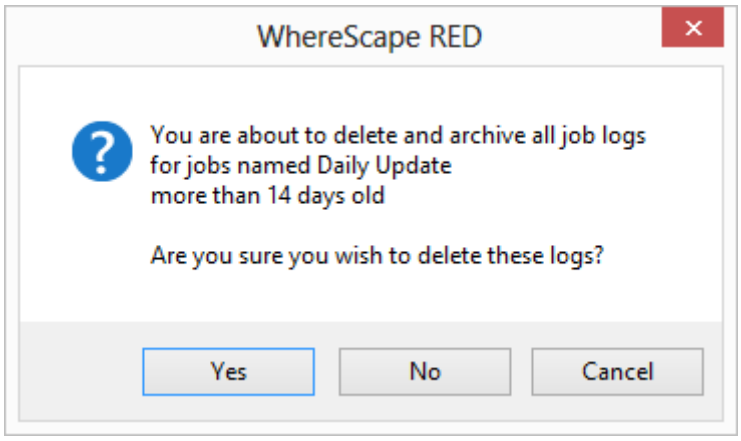
Multiple job logs can be deleted by right-clicking on a job in the scheduler window and choosing **Multiple Log Delete**.



The **Delete Multiple Job Logs** dialog will be displayed. Select or enter the appropriate options to delete the range of job logs required.



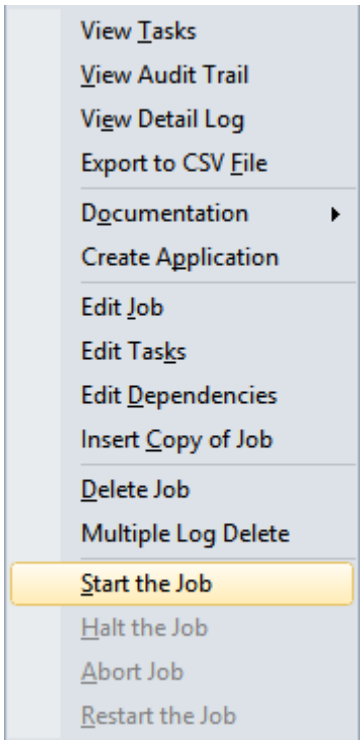
A warning message will be displayed. Click **Yes** to delete.



STARTING A JOB

To start a job

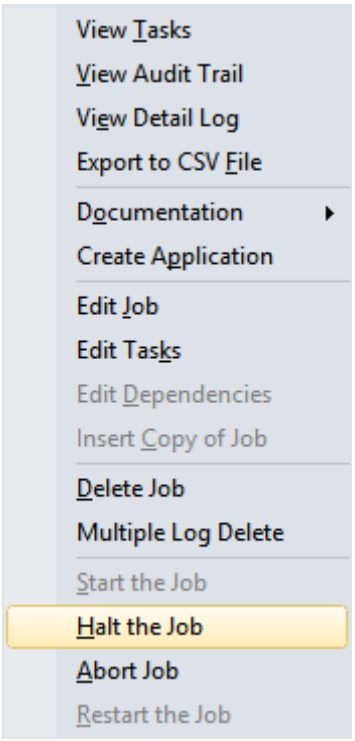
A job can be started by right-clicking on a job in the scheduler window and choosing **Start the Job**.



HALTING A JOB

To halt a job

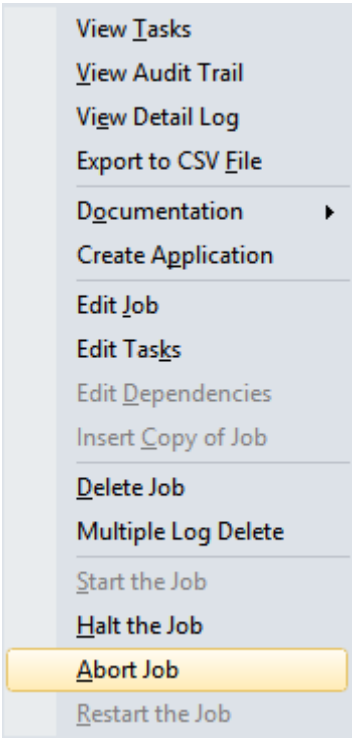
A job can be halted by right-clicking on a job in the scheduler window and choosing **Halt the Job**.



ABORTING A JOB

To abort a job

A job can be aborted by right-clicking on the job in the scheduler window and choosing **Abort Job**.



Once in this state, a job cannot be restarted. The job now exists only as a log of what occurred and is no longer regarded as a job.

EFFECTS OF ABORTING A JOB

The effects of aborting a job depend on the Operating System the Scheduler is running on, the target platform and the action being run.

Note: Rolling back the current transaction may take a considerable amount of time.

Windows Scheduler

SQL Server

Object	Action	Effect
Load	Native Load, SSIS Load	Does not stop process
Load	ODBC Load, Database Link Load, File/Script Windows Load	Stops process
All objects using Procedures	Update	Stops process and rolls back current transaction
Cube	Update	Does not stop process

Oracle

Object	Action	Effect
Load	Database Link Load	Stops process
Load	All other loads	Does not stop process
All objects using Procedures	Update	Stops process and rolls back current transaction
Cube	Update	Does not stop process

DB2

Object	Action	Effect
Load	All Loads	Does not stop process
All objects using Procedures	Update	Stops process and rolls back current transaction
Cube	Update	Does not stop process

Greenplum (target)

Load and update processes are not stopped for all objects.

Hive (target)

Load and update processes are not stopped for all objects.

Netezza (target)

Load and update processes are not stopped for all objects.

PDW (target)

Object	Action	Effect
Load	Native Load, SSIS Load	Does not stop process
Load	ODBC Load, Database Link Load, File/Script Windows Load	Stops process
All objects using Procedures (Local targets)	Update	Stops process and rolls back current transaction
All objects using Procedures (remote targets)	Update	Does not stop process
Cube	Update	Does not stop process

Unix Scheduler

Oracle

Object	Action	Effect
Load	Database Link Load	Stops process and rolls back current transaction
Load	File/Script Unix Load	Does not stop process

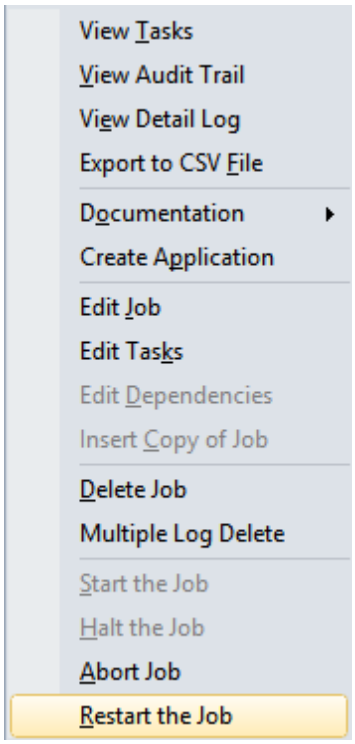
Hive (target)

File/Hadoop Load processes are not stopped on Hive target platforms.

RESTARTING A JOB

To restart a job

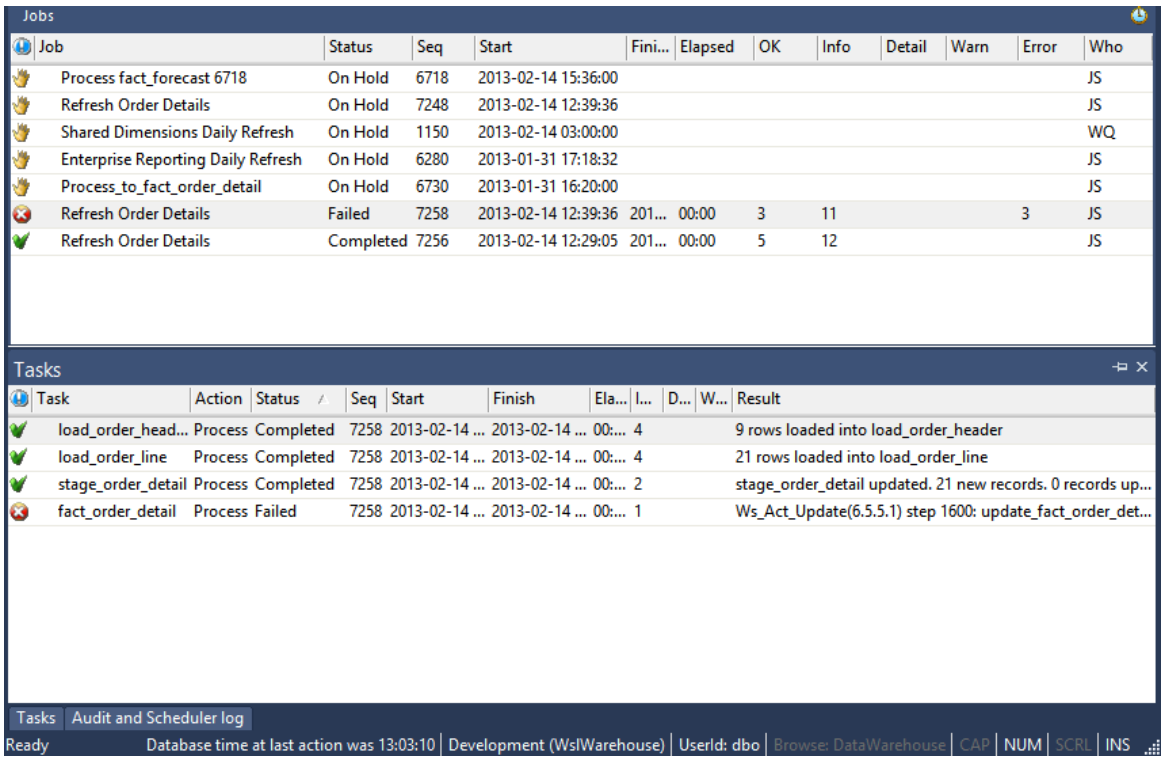
A job can be restarted by right-clicking on a job in the scheduler window and choosing **Restart the Job**.



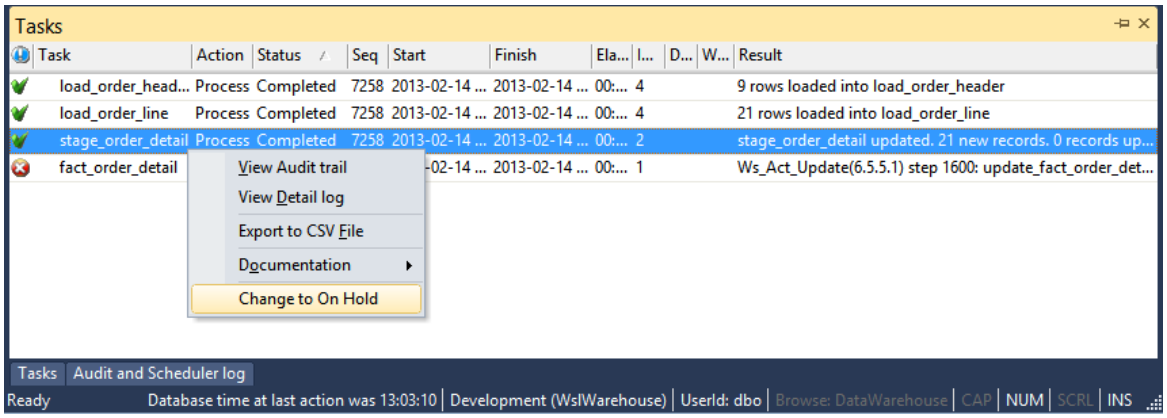
Before restarting a job, it is possible to edit the **status** of the job tasks so that only certain tasks will be **run again** or be **skipped over**.

To run a task again

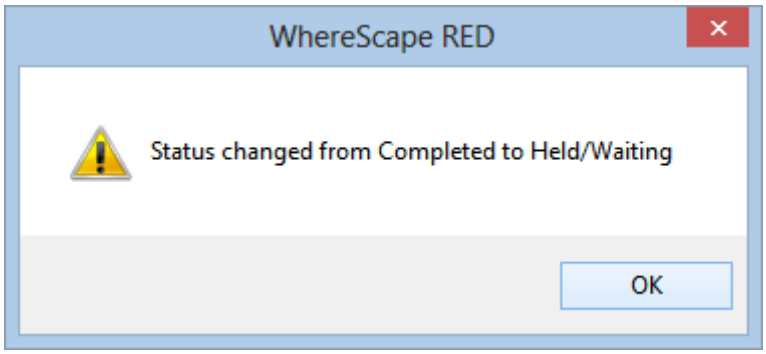
View the job tasks by double-clicking on the failed job. The tasks will be displayed in the bottom pane.



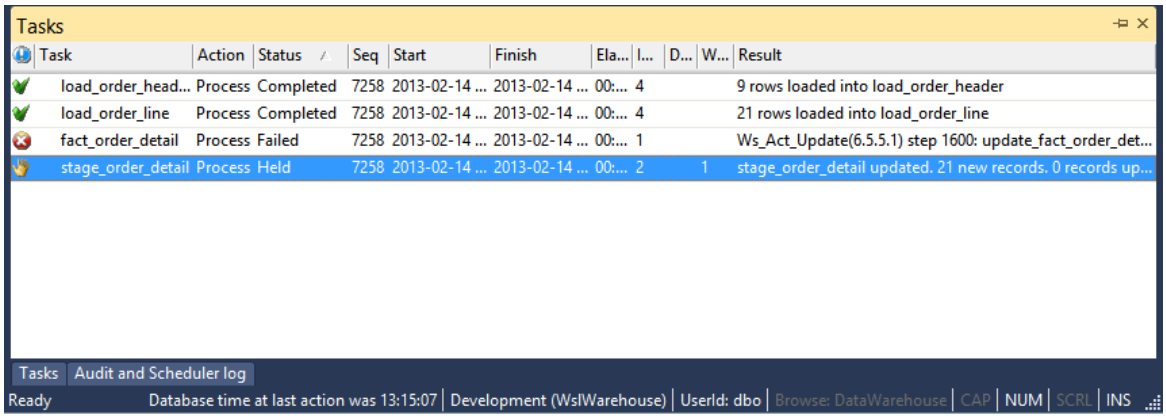
To rerun a task, right-click on the completed task and select **Change to On Hold**



Click **OK** on the message dialog.

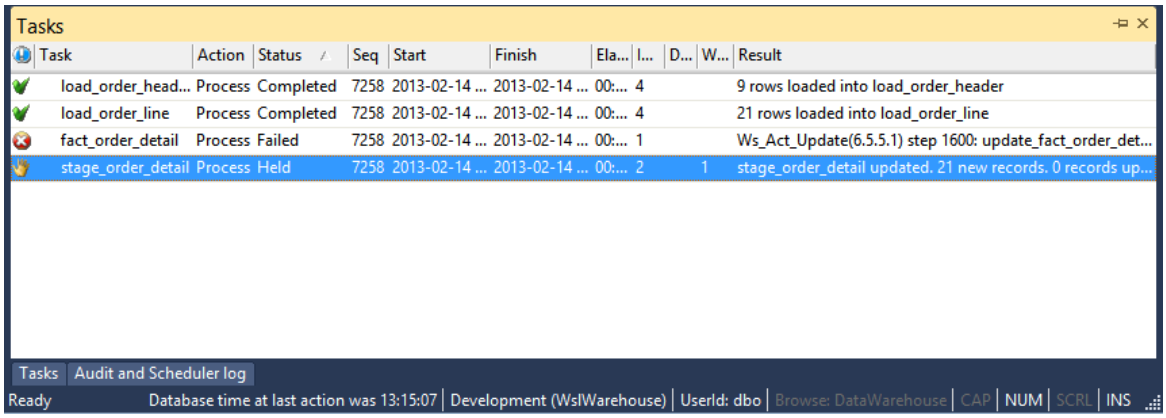


Double-click on the job again to display the tasks. You will see that the selected task now has a status of **Hold** and will thus be rerun when you restart the job.

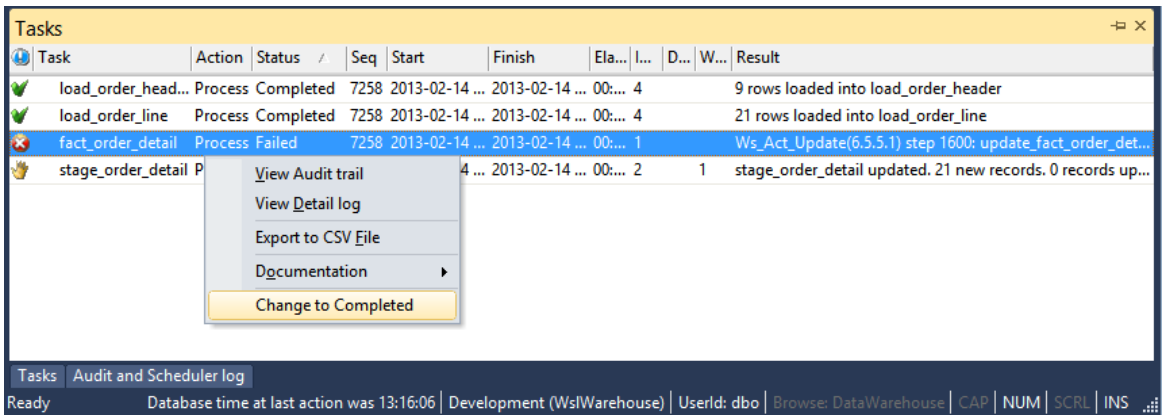


To skip over a task

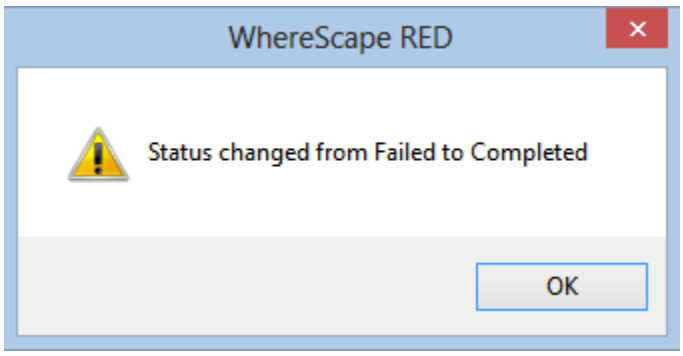
View the job tasks by double-clicking on the failed job. The tasks will be displayed in the bottom pane.



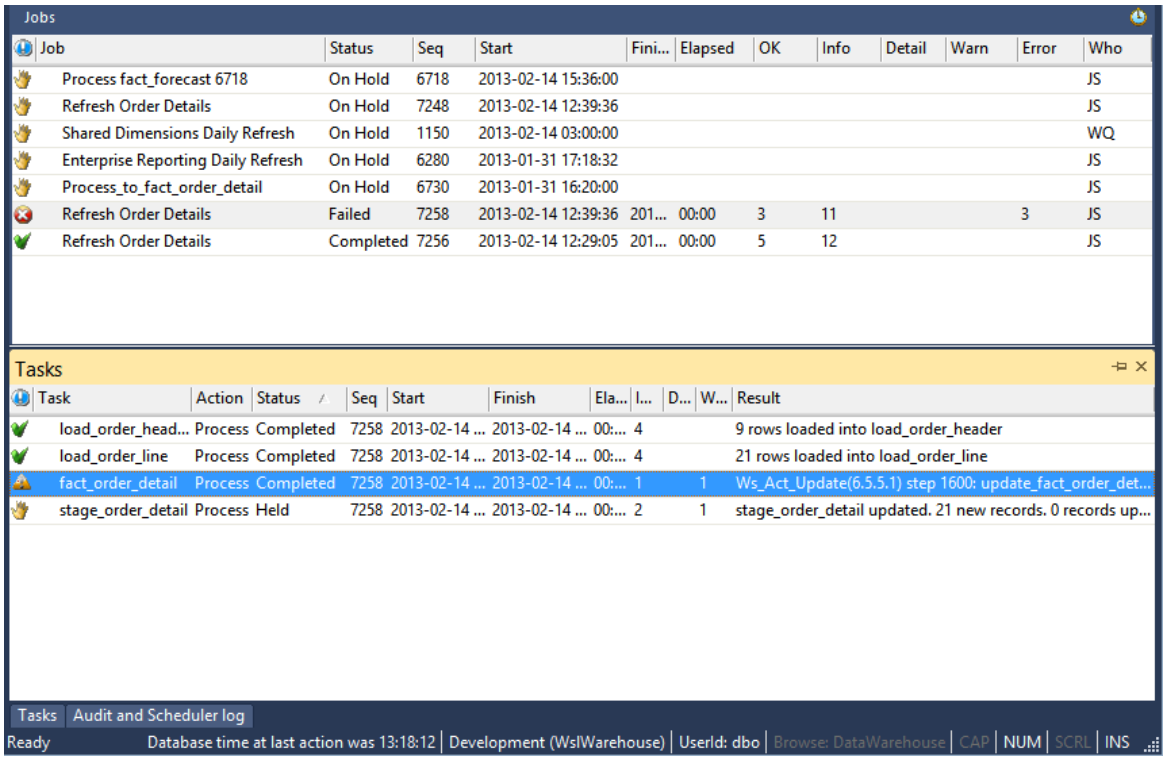
To skip over a task, right-click on the task and select **Change to Completed**.



Click **OK** on the message dialog.



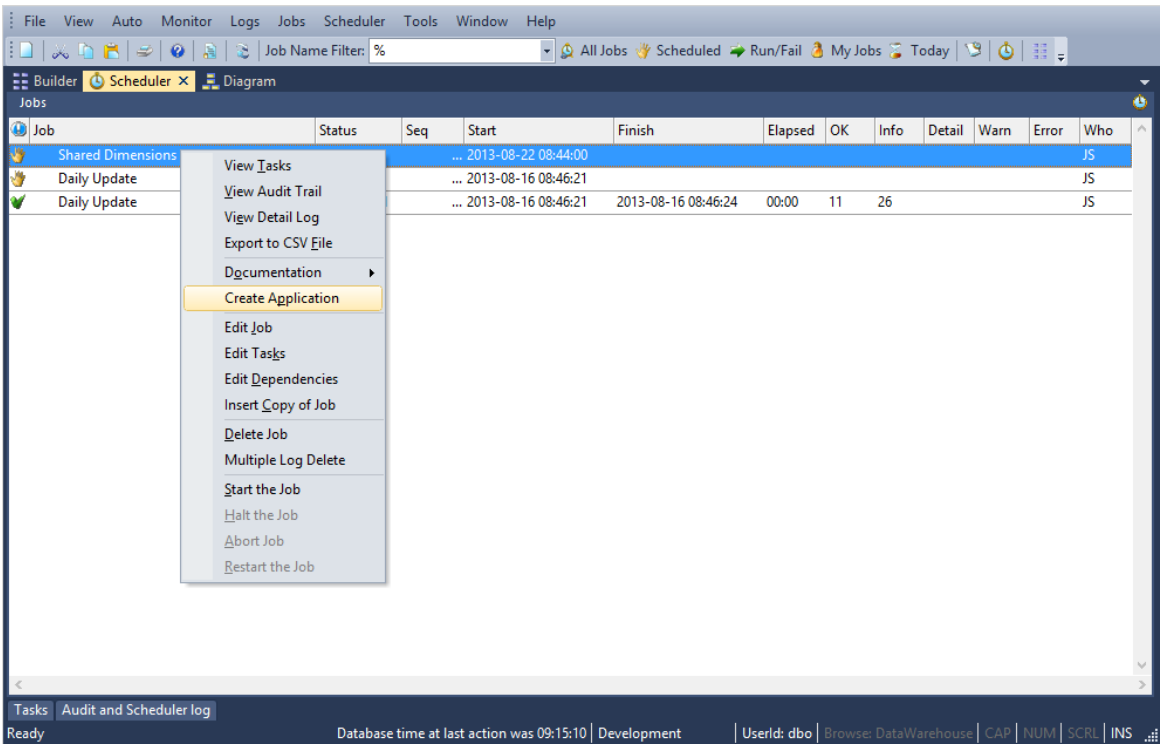
Double-click on the job again to display the tasks. You will see that the selected task now has a status of **Completed** and will thus be skipped when you restart the job.



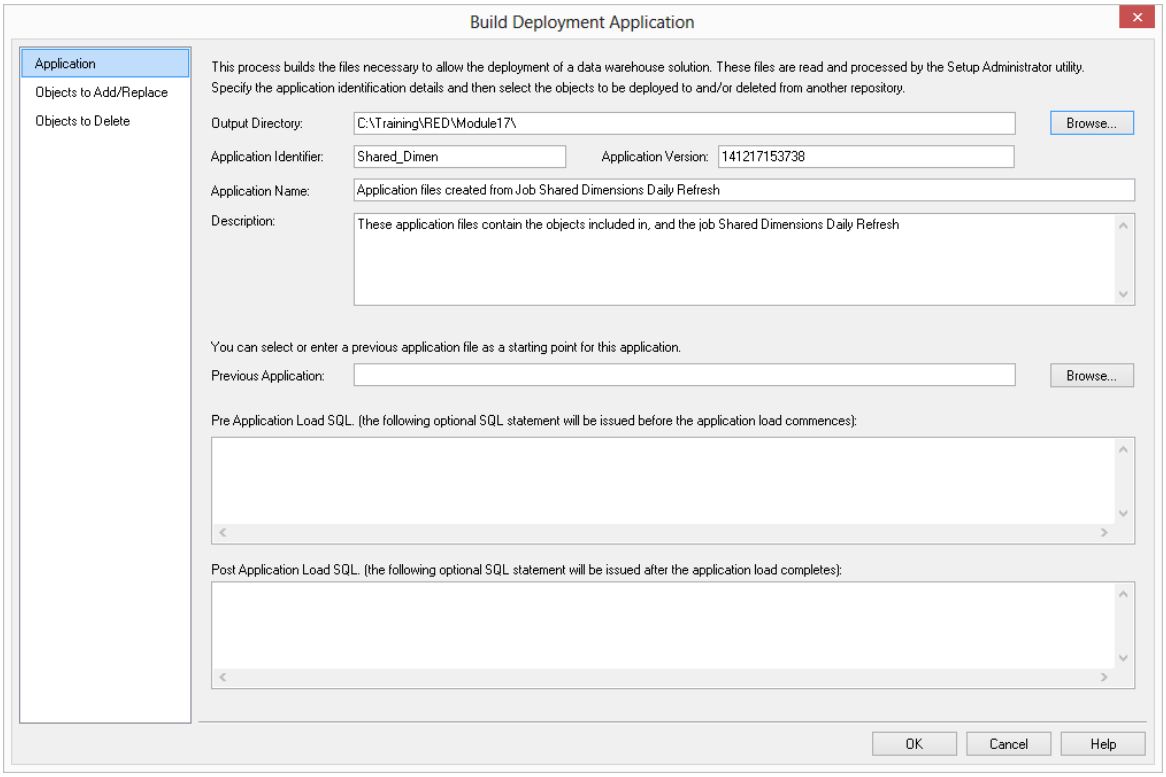
CREATING AN APPLICATION FROM A JOB

To Create an Application from a Job

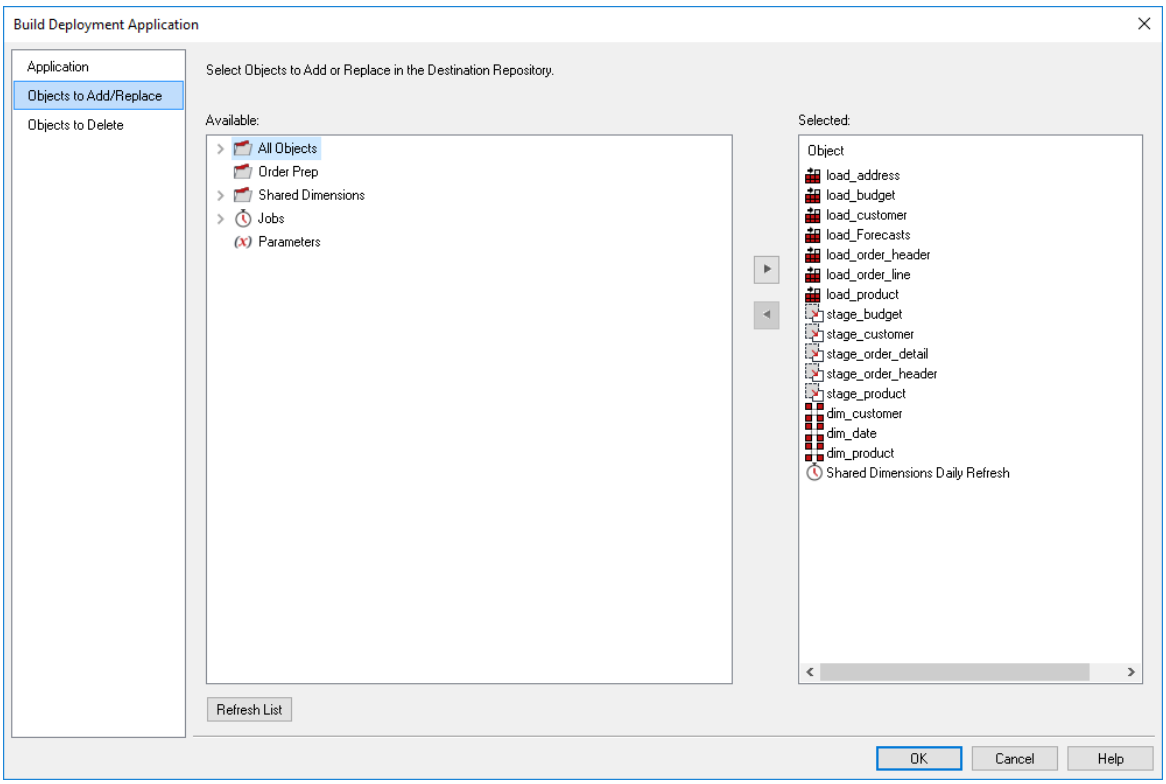
1 Right-click on the job in the scheduler window and select **Create Application**.



2 Edit the application as required.



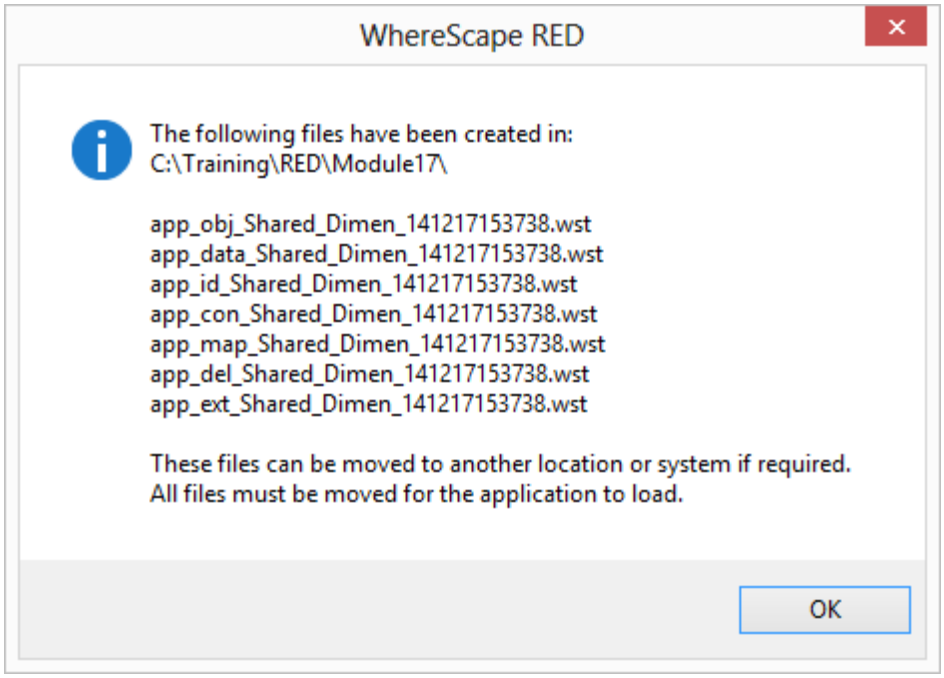
3 Edit the objects to add or replace as required.



Note: Creating an application from a job will save the objects in the job and the job, but not the associated objects.

4 Click **OK** when finished.

5 A dialog will display, confirming the creation of the application files. Click **OK**.



MONITORING THE DATABASE AND JOBS

Note: Oracle on UNIX and Linux only.

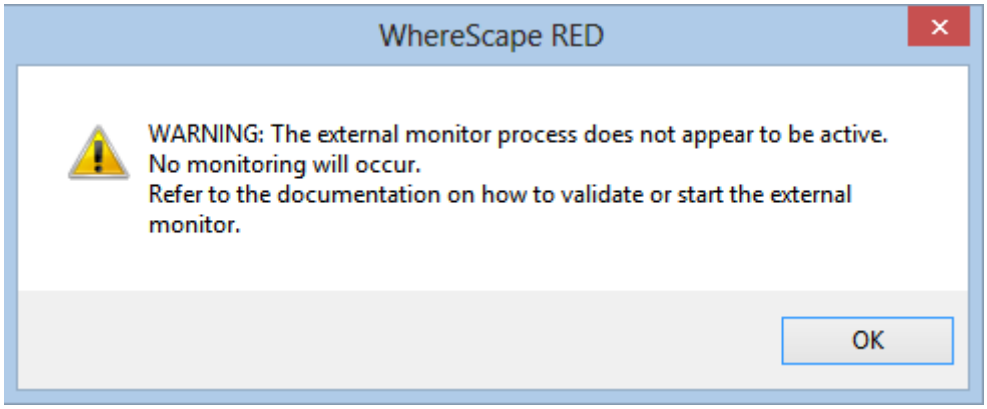
WhereScape RED provides a methodology for monitoring both the database and selected jobs. In the event that a job fails or the database is not available a notification message can be sent to an administrator. The monitoring process requires the establishment of an external process. This process must run on a UNIX platform. It does not have to run on the data warehouse database platform, and in fact it is often preferable that it runs on a separate system. The monitor requires network access to the database. Refer to the Setup/Administrator guide for information on how to establish the external monitor process. This external monitor process must be running in order to utilize the features discussed in the following sections.

The following sections cover both database monitoring and job monitoring. The normal practice would be to setup database monitoring to cover those times when critical jobs are running. The individual monitoring of critical jobs is then defined.

Note: Although it is possible to setup job monitoring without database monitoring this is not recommended. If database monitoring is not in place and the database fails, then no job monitoring will occur and no notifications regarding the jobs will be issued. Therefore, database monitoring is required to provide a comprehensive job monitoring regime.

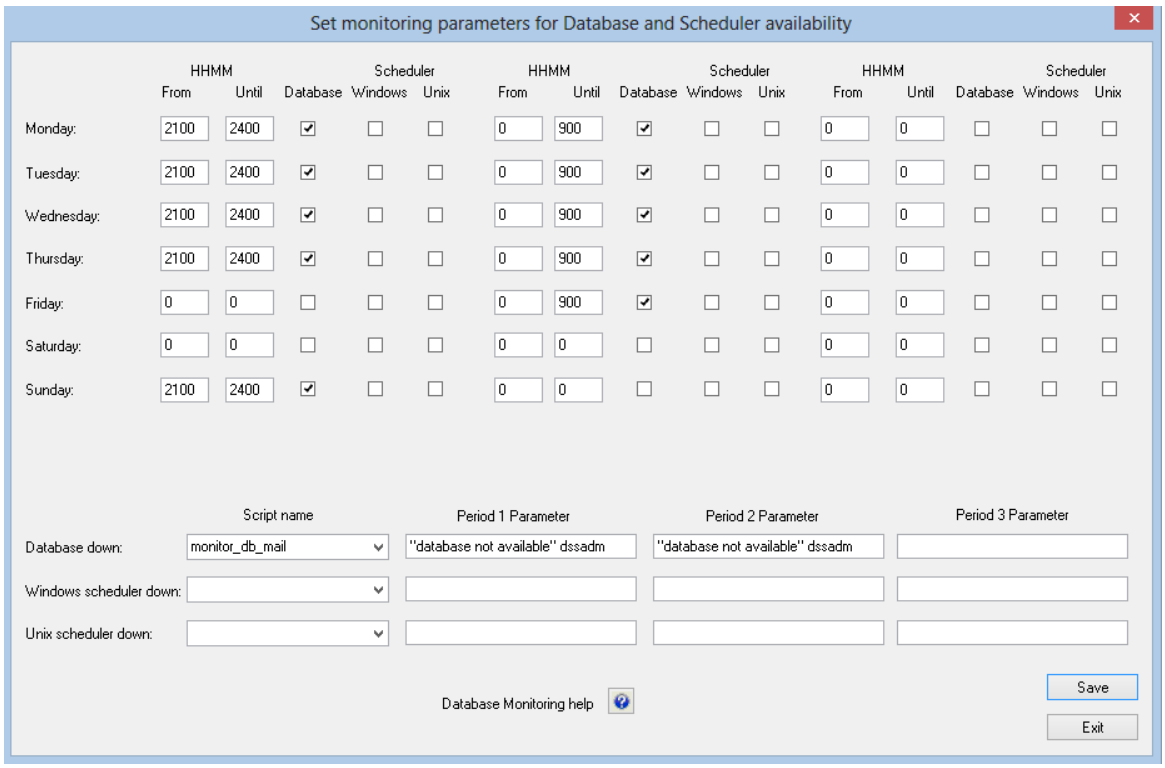
DATABASE MONITORING

Database monitoring is defined by selecting the **Monitor/Database monitoring** menu option from the scheduler window, or from the stand-alone scheduler maintenance program. When this menu option is selected a check is made to ensure that the external monitor process is active. If the external process has not reported in the last 24 hours the following dialog will appear:



In such a case the external monitoring process on the UNIX platform needs to be checked or started. Refer to the **Setup/Administrator** guide for instructions on how to maintain the external monitoring process.

The following example dialog will then appear:



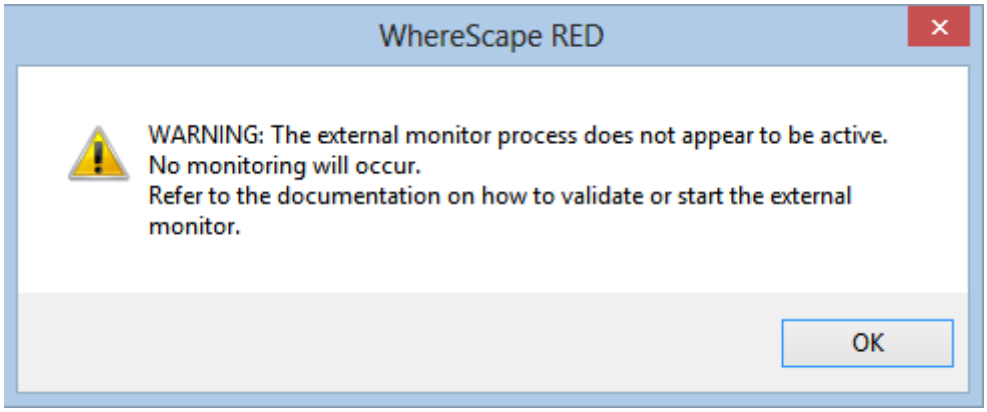
In this example the database is being monitored between the hours of 9:00pm and 9:00am to cover processing for business days monday through friday. If the database is not available during these times the host script called 'monitor_db_mail' will be called and it will be passed two positional parameters. The first parameter being "database not available" and the second being "dssadm". The parameters required are specific to the script being invoked.

In this example, we have defined two monitoring periods. The first is for 9:00pm until midnight and the second is from midnight to 9:00am. As a result, there are two sets of parameters defined for a database notification, one for the first period and one for the second period. In this case, both are the same.

JOB MONITORING

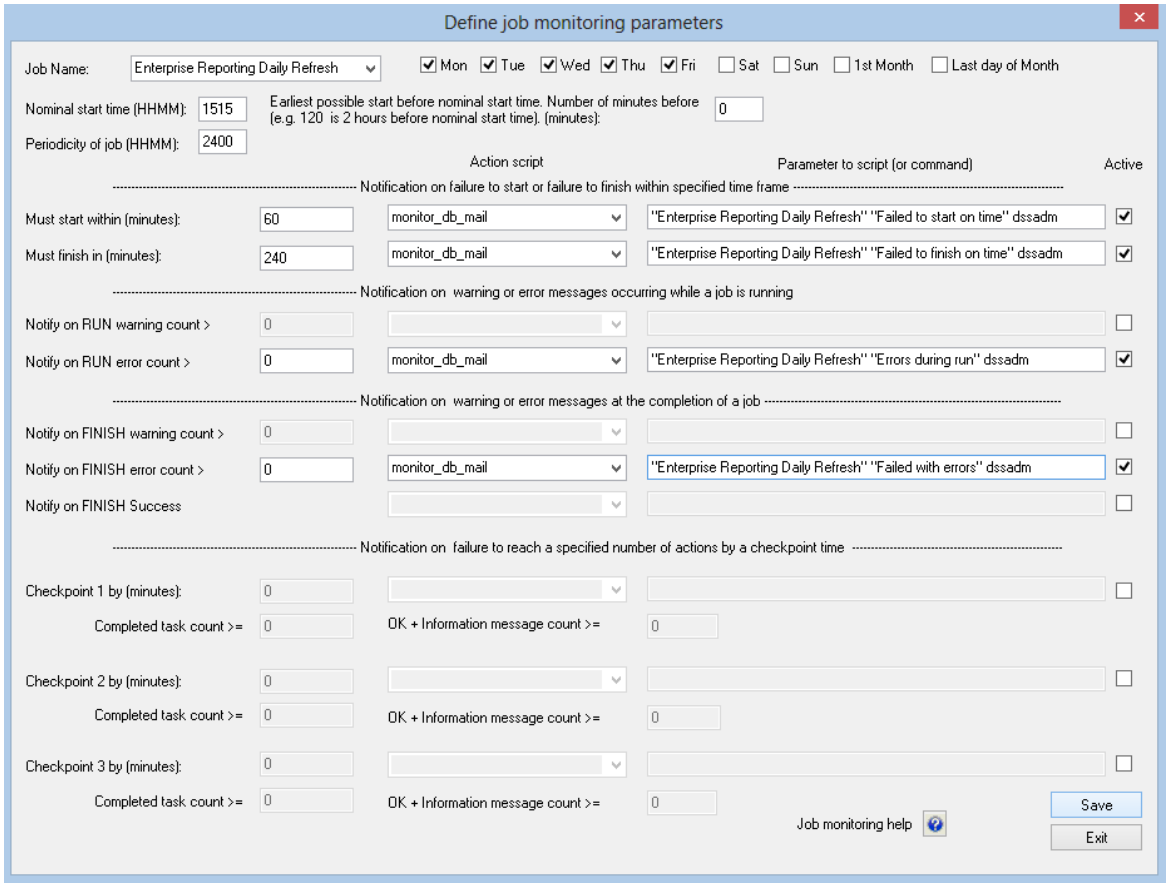
The external monitoring process regularly monitors jobs running in the scheduler and provides notifications when certain conditions occur. By default, the interval between the monitoring of jobs is 15 minutes. This interval can be overridden during the setup of the external job monitor. See the **Setup/Administration** guide.

Job monitoring is defined by selecting the **Monitor/Job monitoring** menu option from the scheduler window, or from the stand-alone scheduler maintenance program. When this menu option is selected, a check is made to ensure that the external monitor process is active. If the external process has not reported in the last 24 hours the following dialog will appear:



In such a case the external monitoring process on the UNIX platform needs to be checked or started. Refer to the Setup/Administrator guide for instructions on how to maintain the external monitoring process.

The following example dialog will then appear:



In this example the job called 'Sales Daily' is being monitored. The days Monday through Friday are checked, so monitoring will occur on these days. The nominal start time is 04:00am. Notifications will occur if the following conditions occur.

1. The job fails to start within 60 minutes of this nominal start time.
2. The job fails to finish within 240 minutes (4 hours) of the nominal start time.
3. Errors occur while the job is running
4. The job fails with errors.

In all conditions that could cause a notification we will invoke the host script 'monitor_job_mail'. This script requires three parameters. The first parameter is the job name, the second a message, and the third the user or users to mail.

Monitor Days

When defining monitoring for a job one or more days need to be selected. As well as the normal week days, there is also the first and last day of the month. Using these two special days it is possible to monitor a job that is critical at the end or start of a month. If Monday through Sunday are checked then it would not be necessary to set the special days. The monitoring day is the day on which the job starts (or could start).

Note: The monitoring day is the day of the absolute minimum start time of a job. For example, if we wish to monitor a job that starts at 00:01, and which has an earliest possible start of 5 minutes earlier (i.e. 23:56) then we need to set monitoring for the day of the earliest possible start. Therefore, to monitor the Monday morning run we would need to have the Sunday checkbox set.

Nominal Start Time

The nominal start time is the time at which the job will normally start. This field must be populated for all jobs being monitored even if not used. Monitoring is optimized for daily jobs, so this field is normally set to the start time of the job.

Earliest Possible Start

The earliest possible start field allows for the setting of a possible start time before the nominal start time. In most situations, this field will be set to 0. When setting to a value other than zero the impact of the potential start time on the days being monitored needs to be considered. See the note under the Monitor Days section above. The nominal start time less the earliest possible start time provides an **Absolute Minimum Start Time** which is used by most of the notification conditions, as well as the monitor days.

Periodicity of Job

The periodicity defines the interval between iterations of the job. For a daily job this would be 24 hours and 0 minutes or 2400. This periodicity is used by the monitoring software to ascertain if a job being examined is the current iteration of the job or a previous iteration.

Notification Order

Notifications are processed in the following order. If a notification is sent for a job, then no further notification checks are made for that job.

1. Finish Error
2. Finish Warning
3. Finish Success
4. Must Finish In
5. Run Error
6. Run Warning
7. Checkpoint 3
8. Checkpoint 2
9. Checkpoint 1
10. Must Start Within

To fully cover all possible events, it is usually necessary to set multiple notifications.

Must Start Within (Notification)

The **Must Start Within** notification allows notification when a job fails to start within a specified number of minutes of the absolute minimum start time. In the example dialog above the job must start within 60 minutes of the start time.

If this criteria is not met, then the monitoring software will look at the **Action Script** and **Parameter to Script** fields to ascertain how to process the notification. If an action script has been defined, then this script will be executed in the UNIX environment and will be passed the defined parameters. If no script is defined, then the **Parameter to Script** field will be executed in the UNIX environment.

An example notification script is included as a template called 'unix_mon_job_mail'.

Must Finish In (Notification)

The **Must Finish In** notification allows notification when a job fails to finish within a specified number of minutes of the absolute minimum start time. In the example dialog above the job must finish within 4 hours of the start time. See the notes under **Must Start Within** for the action taken if the criteria is met.

Run Warning Count (Notification)

The **Run Warning Count** notification allows notification when a job exceeds a specified number of warning messages whilst running. This notification *will only occur when a job is running*. If the job has failed to start, has finished or failed before the external monitor checks the number of warnings this notification will be ignored. See the notes under **Must Start Within** for the action taken if the criteria is met.

Run Error Count (Notification)

The **Run Error Count** notification allows notification when a job exceeds a specified number of error messages whilst running. This notification *will only occur when a job is running*. If the job has failed to start, has finished or failed before the external monitor checks the number of errors this notification will be ignored. See the notes under **Must Start Within** for the action taken if the criteria is met.

Finish Warning Count (Notification)

The **Finish Warning Count** notification allows notification when a job has exceeded a specified number of warning messages and has completed or failed. This notification *will only occur when a job has completed or failed*. If the job has failed to start or is still running when the external monitor checks the number of warnings this notification will be ignored. See the notes under **Must Start Within** for the action taken if the criteria is met.

Finish Error Count (Notification)

The **Finish Error Count** notification allows notification when a job has exceeded a specified number of error messages and has failed. This notification *will only occur when a job has failed*. If the job is still running when the external monitor checks the number of warnings, this notification will be ignored. If the job has been restarted and gone on to complete normally, this notification will be ignored even when errors have occurred. See the notes under **Must Start Within** for the action taken if the criteria is met.

Finish Success (Notification)

The **Finish Success** notification allows notification when a job completes. This notification can be used for example to mail a log of the completed job to a specified user. See the notes under **Must Start Within** for the action taken if the criteria is met.

Checkpoint 1 (Notification)

The **Checkpoint 1** notification allows notification when a job fails to achieve either a specified number of task completions, or a specified number of information/OK messages within a specified elapsed time. The elapsed time is the time from the absolute minimum start time. This notification *will only occur when the job is running*. If the job has failed to start, completed or failed when the external monitor checks the checkpoint, this notification will be ignored. See the notes under **Must Start Within** for the action taken if the criteria is not met.

Checkpoint 2 (Notification)

The **Checkpoint 2** notification allows notification when a job fails to achieve either a specified number of task completions, or a specified number of information/OK messages within a specified elapsed time. The elapsed time is the time from the absolute minimum start time. This notification *will only occur when the job is running*. If the job has failed to start, completed or failed when the external monitor checks the checkpoint, this notification will be ignored. See the notes under **Must Start Within** for the action taken if the criteria is not met.

Checkpoint 3 (Notification)

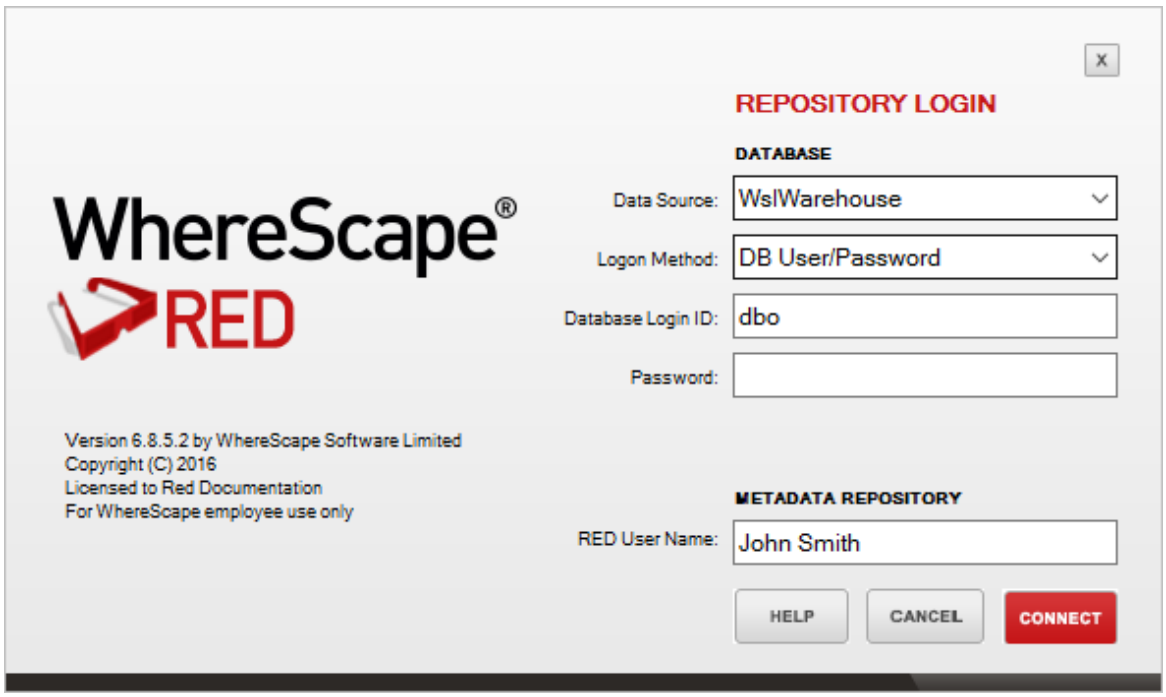
The **Checkpoint 3** notification allows notification when a job fails to achieve either a specified number of task completions, or a specified number of information/OK messages within a specified elapsed time. The elapsed time is the time from the absolute minimum start time. This notification *will only occur when the job is running*. If the job has failed to start, completed or failed when the external monitor checks the checkpoint, this notification will be ignored. See the notes under **Must Start Within** for the action taken if the criteria is not met.

STAND ALONE SCHEDULER MAINTENANCE

WhereScape RED includes a stand alone scheduler maintenance screen. This screen provides all the scheduler control functionality found in the main WhereScape RED utility, but with no access to the main metadata repository.

Scheduler maintenance logon

The logon screen differs in that the user name and password do not have to be that of a valid metadata repository. This user name/password combination can be any valid database user. There is an additional entry required being **Meta database**. This is the schema of the metadata repository that you wish to connect to in terms of scheduler maintenance.



For **SQL Serve**r the Meta database must be set to **dbo**.

For **Oracle** data warehouses the 'Meta database' owner must have made the following grants to the specified user before the scheduler maintenance utility can be used.

Scheduler maintenance grants

Repository access grants:

- grant select on ws_meta to dsssched;
- grant select on ws_meta_tables to dsssched;
- grant select on ws_obj_type to dsssched;
- grant select,insert on ws_user_adm to dsssched;
- grant select,insert,update on ws_user_adm to dsssched;
- grant select,alter on ws_wrk_job_dependency_seq to dsssched;
- grant select on ws_table_attributes to dsssched;

Object access (job create) grants:

- grant select on ws_obj_object to dssscheduler;
- grant select on ws_obj_pro_map to dssscheduler;
- grant select on ws_obj_project to dssscheduler;
- grant select on ws_obj_group to dssscheduler;
- grant select on ws_pro_gro_map to dssscheduler;

Scheduler status grants:

- grant select on ws_wrk_audit_log to dssscheduler;

Scheduler status, and job deletion grants:

- grant select,delete on ws_wrk_error_log to dssscheduler;

Scheduler status, poll grants:

- grant select,update on ws_wrk_scheduler to dssscheduler;

Job creation grants:

- grant select,insert,update,delete on ws_wrk_dependency to dssscheduler;
- grant select,insert,update,delete on ws_wrk_job_ctrl to dssscheduler;
- grant select,alter on ws_job_seq to dssscheduler;
- grant select,alter on ws_task_seq to dssscheduler;

Job maintenance grants:

- grant select,insert,delete on ws_wrk_job_log to dssscheduler;
- grant select,update,delete on ws_wrk_job_run to dssscheduler;
- grant select,insert,update,delete on ws_wrk_dependency to dssscheduler;
- grant select,insert,update,delete on ws_wrk_job_dependency to dssscheduler;
- grant select,delete on ws_wrk_job_thread to dssscheduler;

Task maintenance grants:

- grant select,insert,update,delete on ws_wrk_task_ctrl to dssscheduler;
- grant select,update,delete on ws_wrk_task_run to dssscheduler;
- grant select,insert,delete on ws_wrk_task_log to dssscheduler;
- grant select,insert,update on dss_parameter to dssscheduler;

Right-click **Used by** option in parameters listing grants:

- grant select on ws_pro_header to dssscheduler;

- grant select on ws_pro_line to dssched;
- grant select on ws_scr_header to dssched;
- grant select on ws_scr_line to dssched;
- grant select on ws_load_tab to dssched;
- grant select on ws_load_col to dssched;
- grant select on ws_stage_tab to dssched;
- grant select on ws_stage_col to dssched;
- grant select on ws_dim_tab to dssched;
- grant select on ws_dim_col to dssched;
- grant select on ws_fact_tab to dssched;
- grant select on ws_fact_col to dssched;
- grant select on ws_agg_tab to dssched;
- grant select on ws_agg_col to dssched;

A sample script to grant these privileges is shipped with WhereScape RED. This script is called 'grant_sched_access.sql' and can be found in the WhereScape program directory.

The scheduler maintenance utility does not require a WhereScape license key. The WhereScape RED software can be installed onto a PC, and this utility utilized without having to use the WhereScape **Setup Administrator** utility.

SQL TO RETURN SCHEDULER STATUS

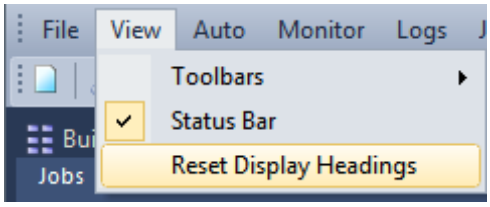
This SQL returns the scheduler status:

```
SELECT CASE
WHEN ws_stop_date IS NOT NULL
THEN 'STOPPED'
WHEN ((DATEDIFF(mi,ws_active_date,GETDATE()) -
CONVERT(INTEGER,DATEDIFF(mi,ws_active_date,GETDATE())/60)*60) > 15 )
OR (CONVERT(INTEGER,DATEDIFF(mi,ws_active_date,GETDATE())/60)>0)
THEN 'NOT ACTIVE'
WHEN (((DATEDIFF(mi,ws_active_date,GETDATE()) -
CONVERT(INTEGER,DATEDIFF(mi,ws_active_date,GETDATE())/60)*60)>((ws_inter
val/60)+10)
OR CONVERT(INTEGER,DATEDIFF(mi,ws_active_date,GETDATE())/60)>0)
AND ws_poll_flag=1)
THEN 'NOT ACTIVE'
ELSE 'Running'
END
FROM dbo.ws_wrk_scheduler
WHERE ws_name = 'YourSchedulerName'
```

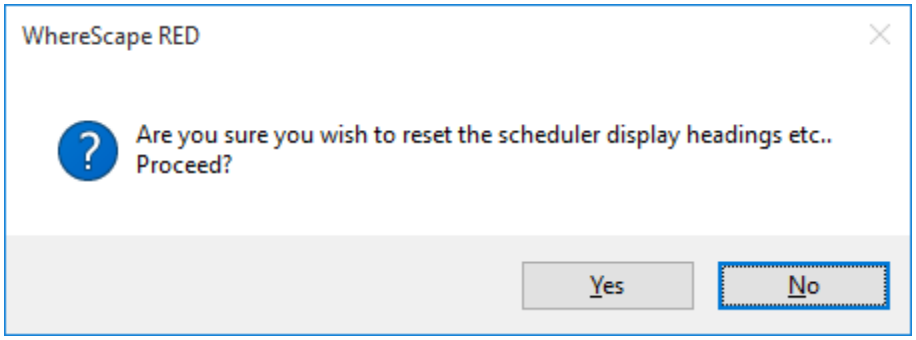
The procedure sets the status in the metadata.

RESET COLUMNS IN JOB AND TASK VIEW

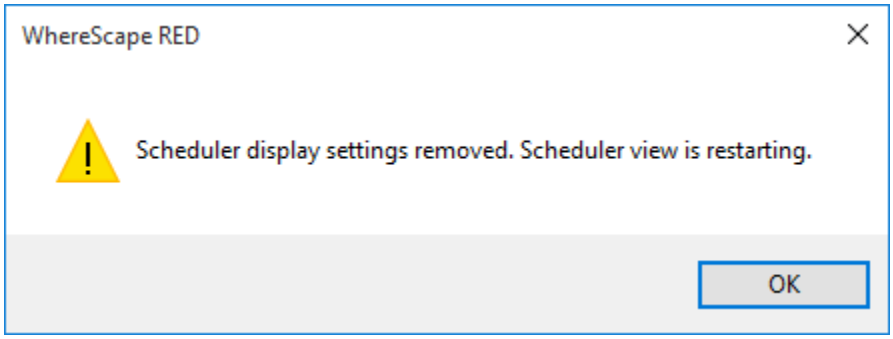
Job and Task Report headings can be reset by selecting the **View/Reset Display Headings** menu option from the scheduler window. The short-cut keys are Alt+V-R.



A dialog will ask you to confirm the request.

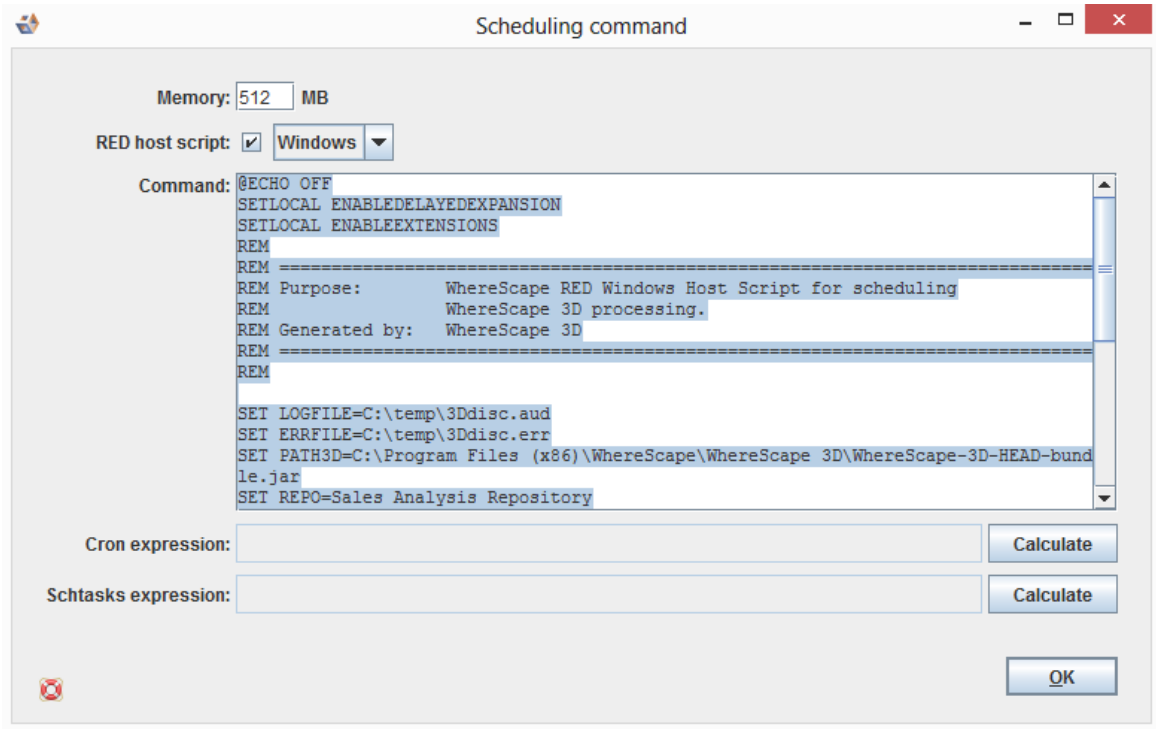


If you selected **Yes** to reset the display settings, then a dialog will confirm once the reset has occurred.

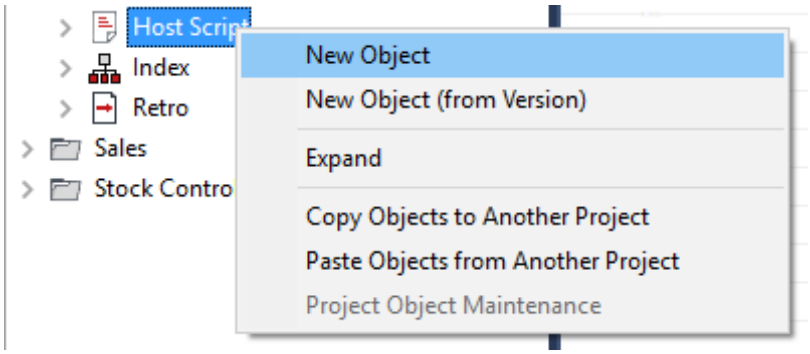


SCHEDULING A RED HOST SCRIPT FROM 3D

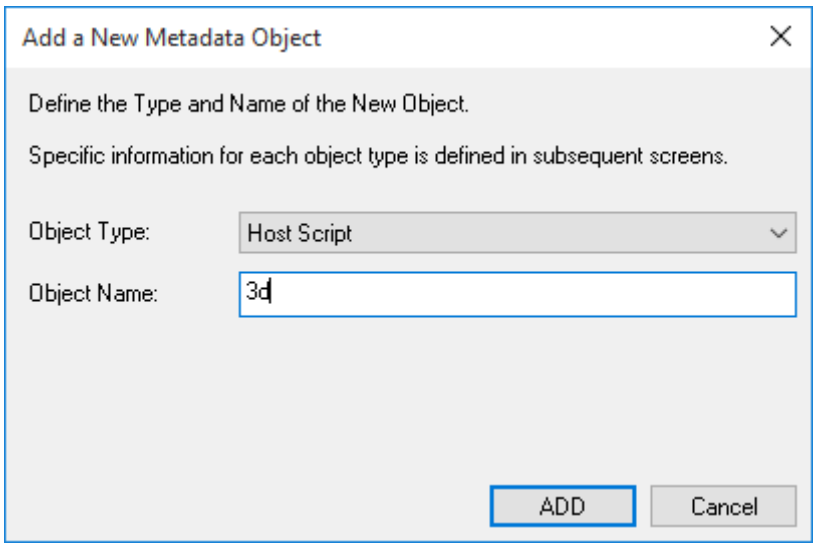
Create the RED host script in 3D. Use Ctrl-C to copy the script.



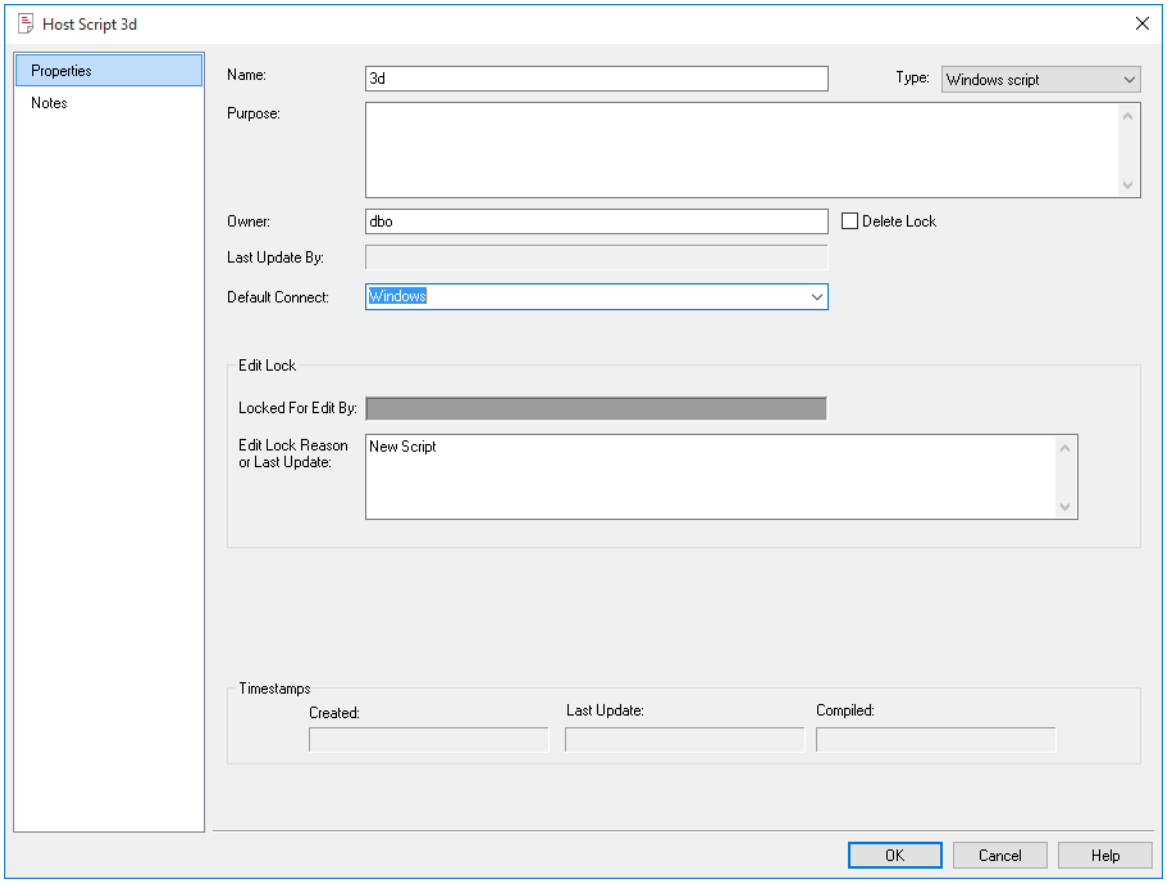
In RED, create a host script by right-clicking on the **Host Script** object group and choosing **New Object**.



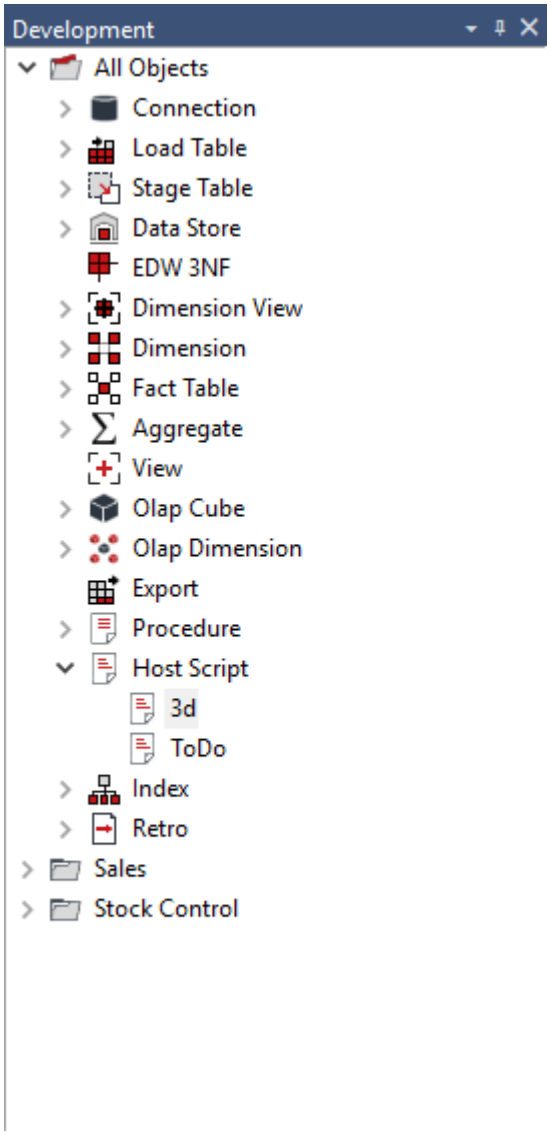
Enter a name for the host script and click **ADD**.



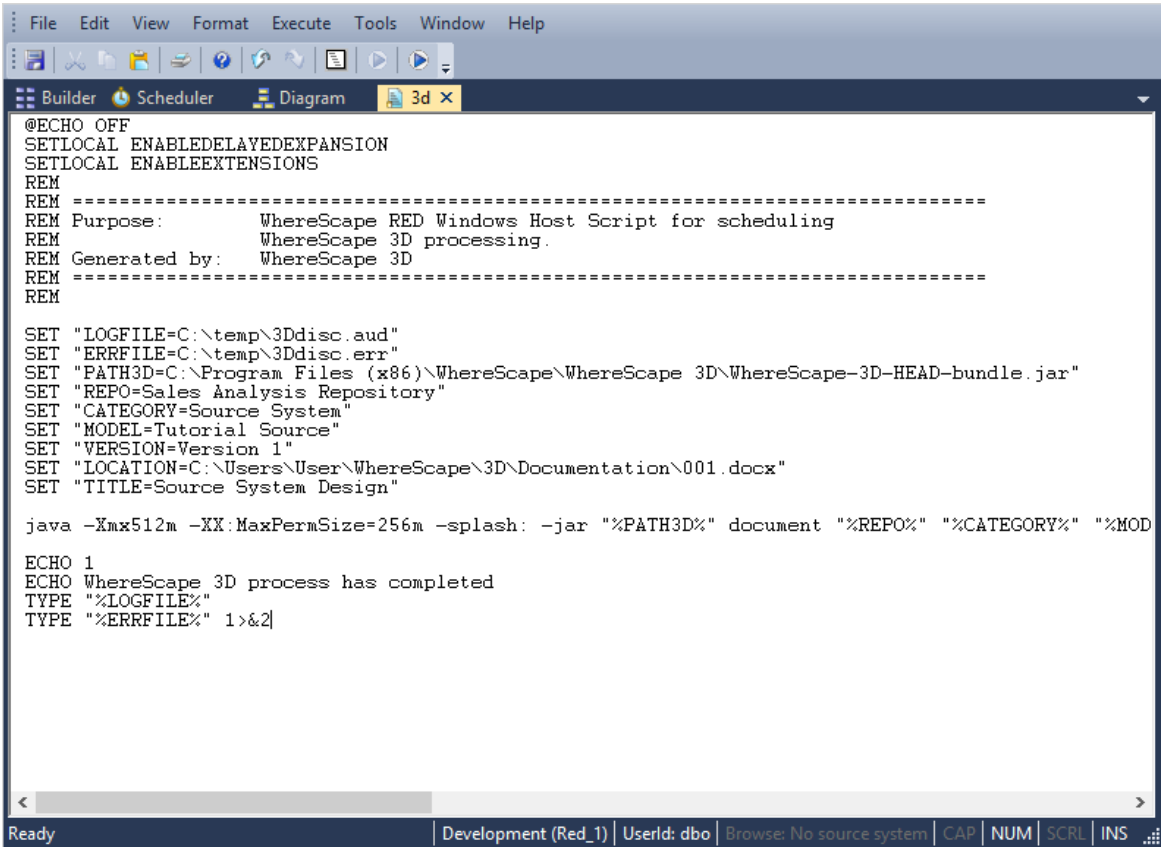
Set the default connection to Windows and click **OK**.



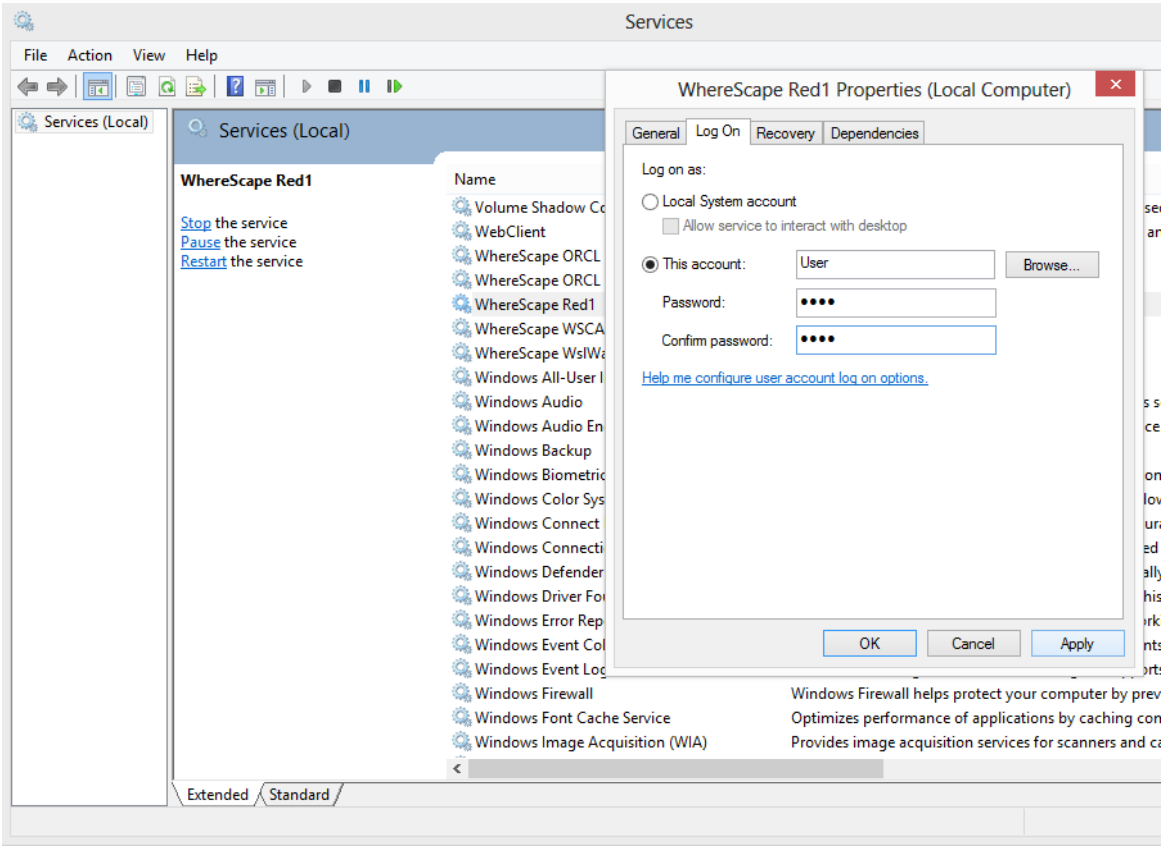
Double-click on the 3d host script in the left pane.



Now paste the script from 3D and click **Save**; then close the host script window.

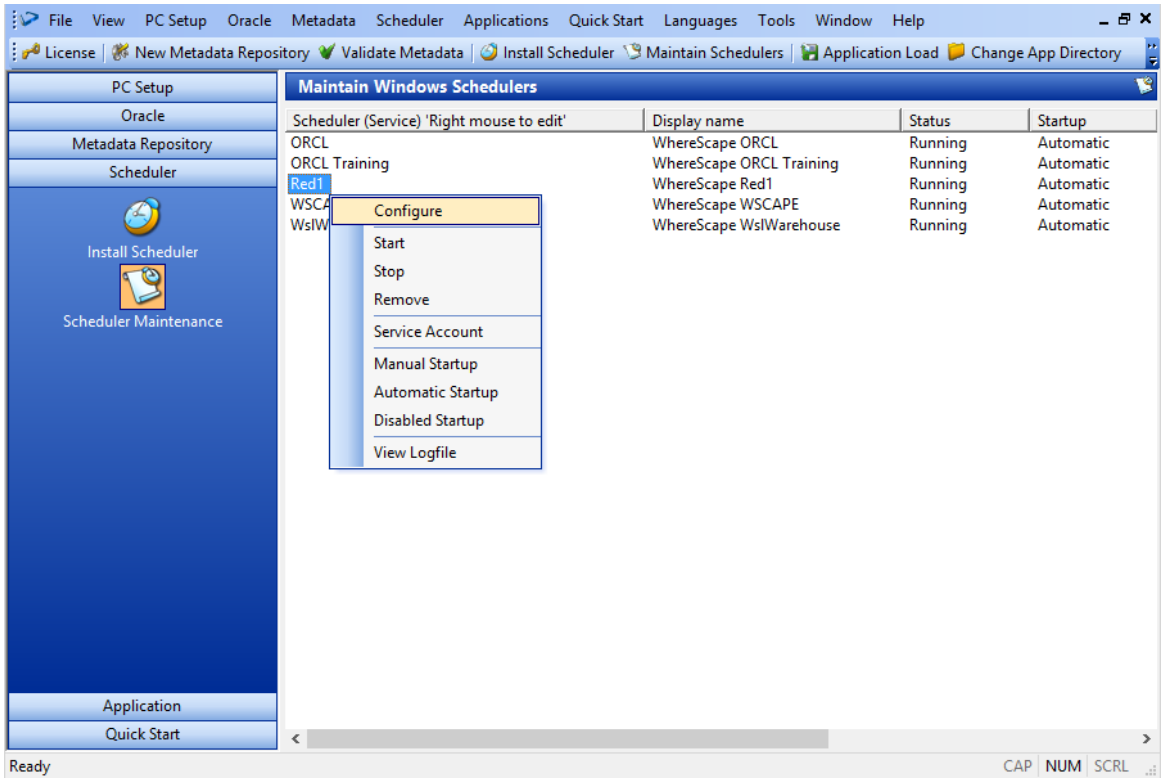


Go to **Control Panel > Administrative Tools > Services** and right-click on the relevant service name. Choose **Properties** and then the **Log On** tab. Change the scheduler to run as the current user. Click **Apply** and then **OK**.

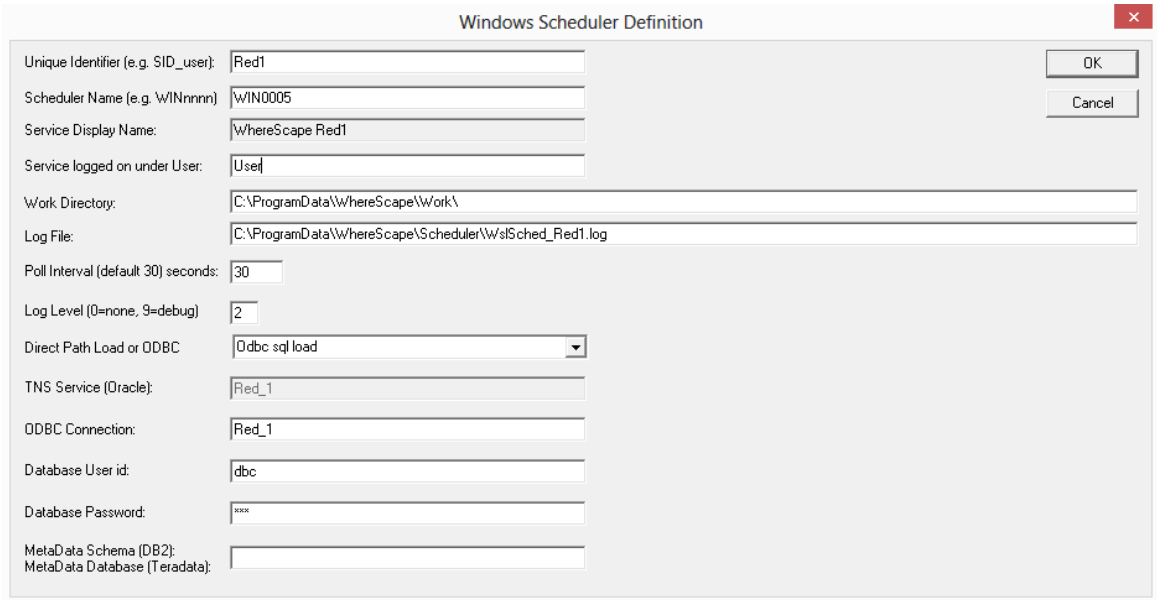


Stop and start the scheduler and then close the **Services** window.

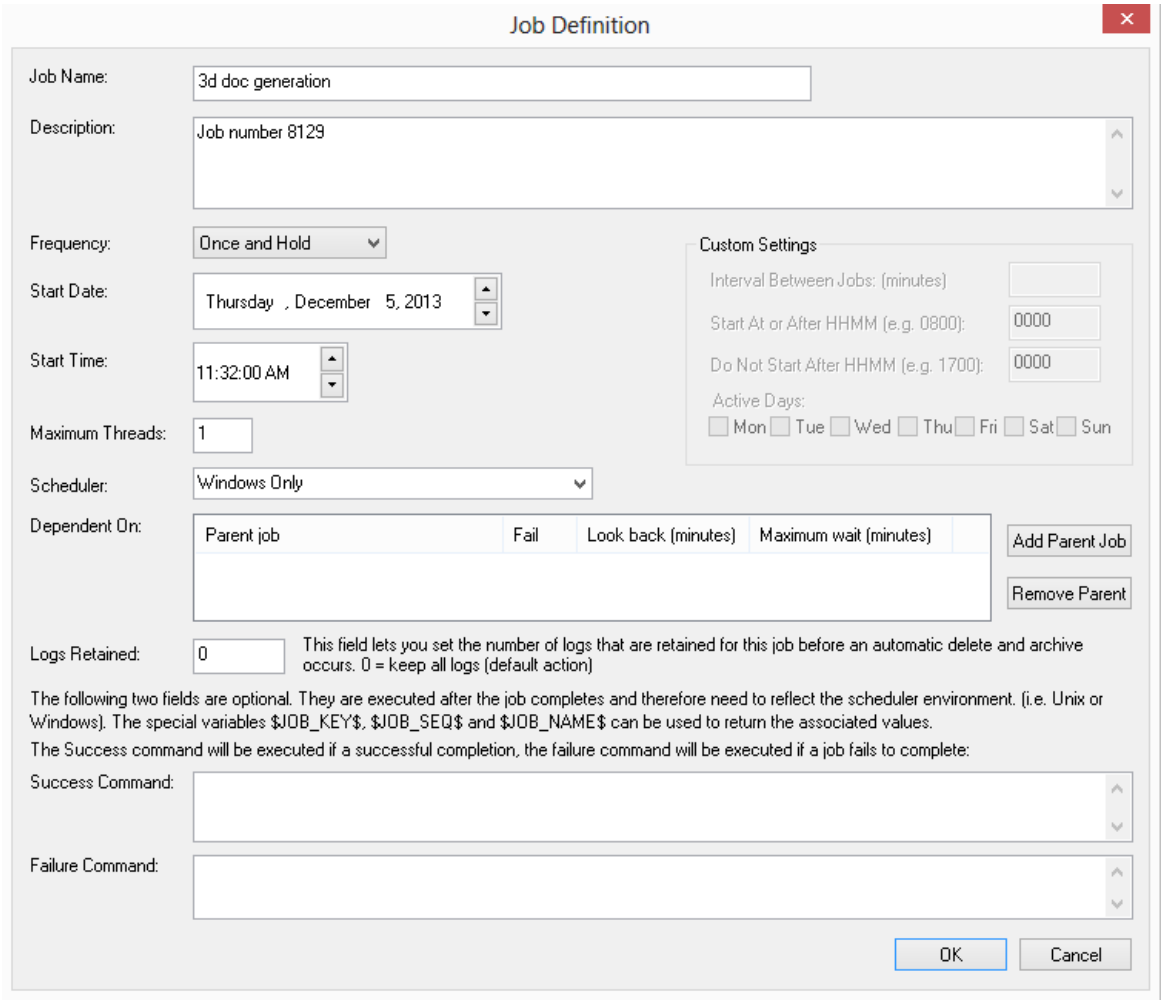
Click on **Scheduler Maintenance** in WhereScape Administrator; right-click on the scheduler and choose **Configure**.



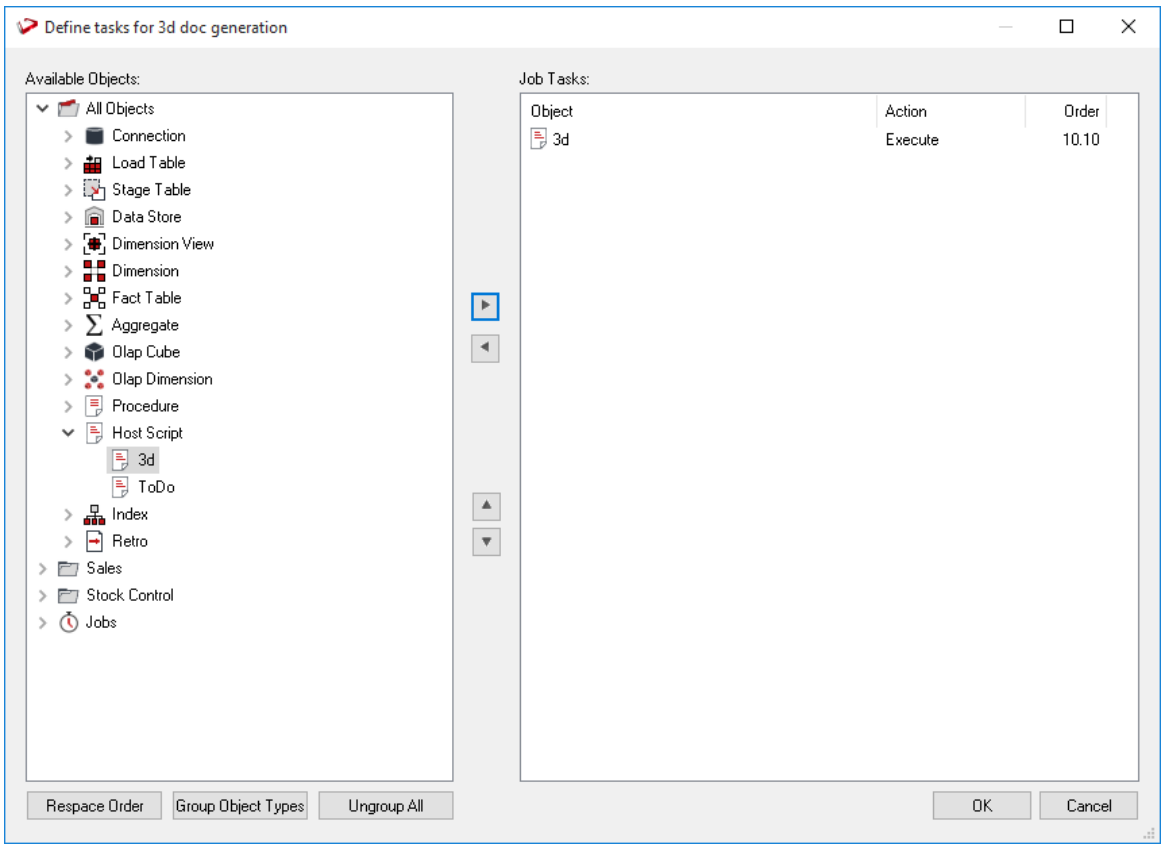
Set the **Service logged on under User** field to the current user. Click **OK** and close WhereScape Administrator.



In RED, click on the **Scheduler** tab and create a new job. Click **OK**.



Select the host script to be run and click **OK**.



Run the job.

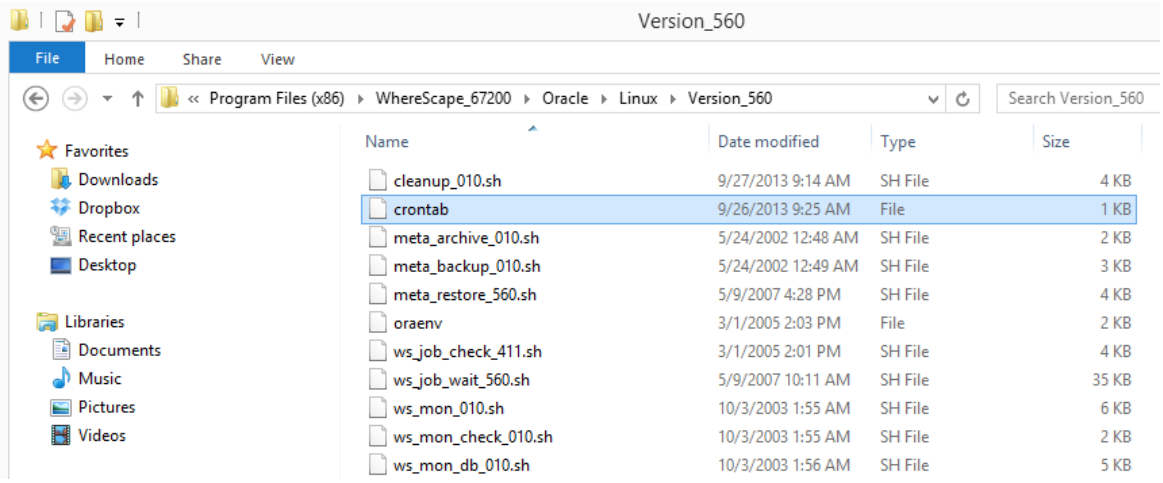
STOPPING A LINUX/UNIX SCHEDULER FROM WITHIN RED

To stop a Linux/UNIX Scheduler from within RED, follow the steps below:

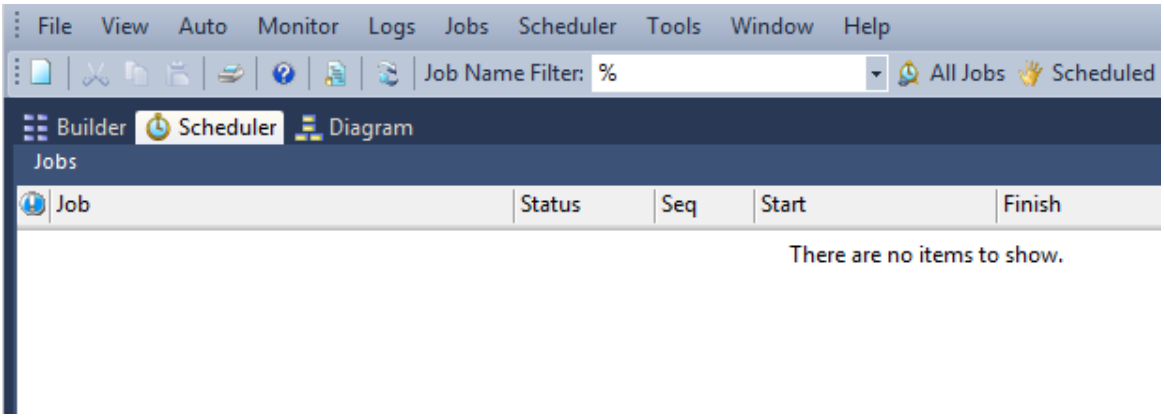
1 Edit the **crontab** and comment out the ws_sched_check_nnn.sh entry. This will stop the scheduler restarting within the next 20 minutes.

```
## backup of the meta data
## workdays backup at end of day
## on the weekends at 7:00pm
##
00 20 * * 1-5 _HOME_/wsl/bin/meta_backup_010.sh
oraenv >/dev/null 2>&1
00 19 * * 0,6 _HOME_/wsl/bin/meta_backup_010.sh
oraenv >/dev/null 2>&1
##
## each day archive (tar) the meta backups and compress
##
00 22 * * * _HOME_/wsl/bin/meta_archive_010.sh >/dev/null 2>&1
##
## each day cleanup log files etc.
##
00 18 * * * _HOME_/wsl/bin/cleanup_010.sh oraenv >/dev/null 2>&1
#
## check and if required start the scheduler
##
0,20,40 * * * * _HOME_/wsl/bin/ws_sched_check_560.sh
oraenv >>_HOME_/wsl/sched/log/sched_oraenv.log 2>&1
#
## check and if required start the monitor
##
0,30 * * * * _HOME_/wsl/bin/ws_mon_check_010.sh
oraenv >>_HOME_/wsl/sched/log/mon_oraenv.log 2>&1
```

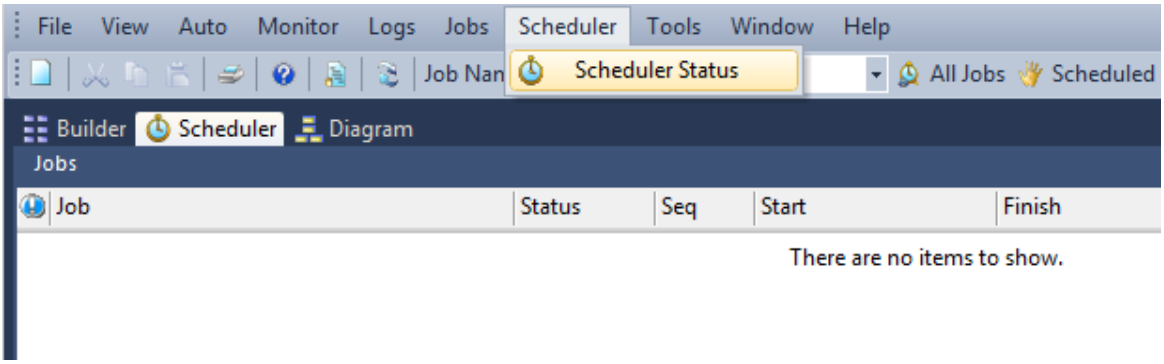
Note: There may be several different versions of the scheduler files for a given database and platform (UNIX or Linux). For example, there may be different folders in ...\WhereScape\Oracle\Linux\: Version_010, Version_411 and Version_560. The highest version number script less than or equal to the version of RED in use should always be used.



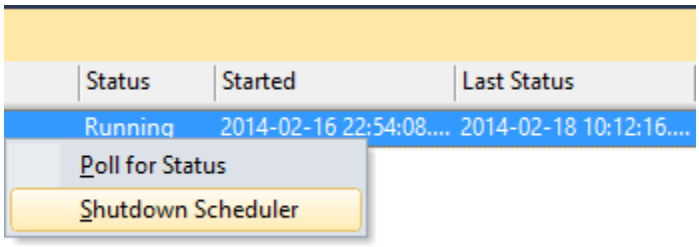
2 Start RED and click on the **Scheduler** tab.



3 Click on Scheduler in the toolbar and then select **Scheduler Status**.



4 Right-click on the displayed UNIX/Linux scheduler entry and choose **Shutdown Scheduler**.



Sometime within the next poll interval of the scheduler, the scheduler will gracefully stop.

CHAPTER 24
INDEXES

Indexes may exist on any table defined in WhereScape RED.

By default WhereScape RED will auto-generate a number of indexes during the drag and drop process and when building procedures.

These indexes can be altered or deleted. New indexes can be created as desired.

Note: The maintenance of the indexes is performed as part of the normal scheduler processing.

In the left pane right-click on a table to:

- Display indexes
- Add indexes

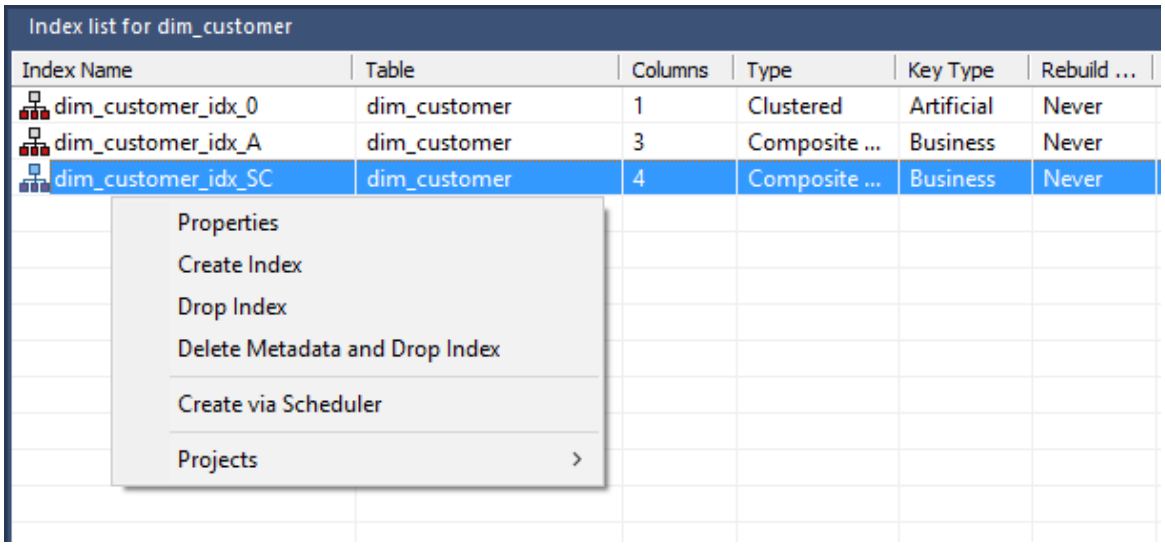
IN THIS CHAPTER

INDEX DEFINITION

By right-clicking on a table in the left pane and selecting **Display Indexes** the middle pane will display the indexes for that table. Alternatively, you can double-click on the Index object type in the left pane to display all indexes in the repository or a specific group or project.

In the middle pane, right-click on an index and the following options are available:

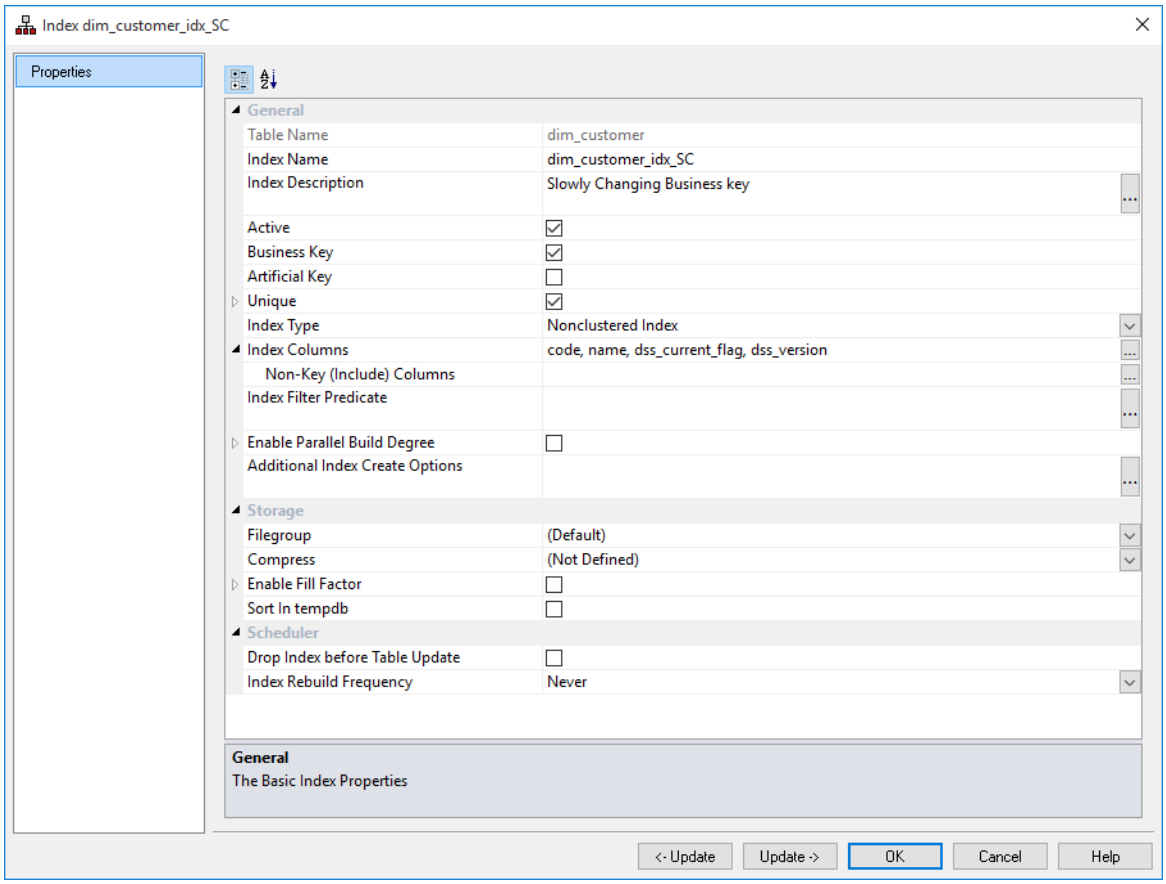
- Properties
- Create Index
- Drop Index
- Delete Metadata and Drop Index
- Create via Scheduler
- Projects



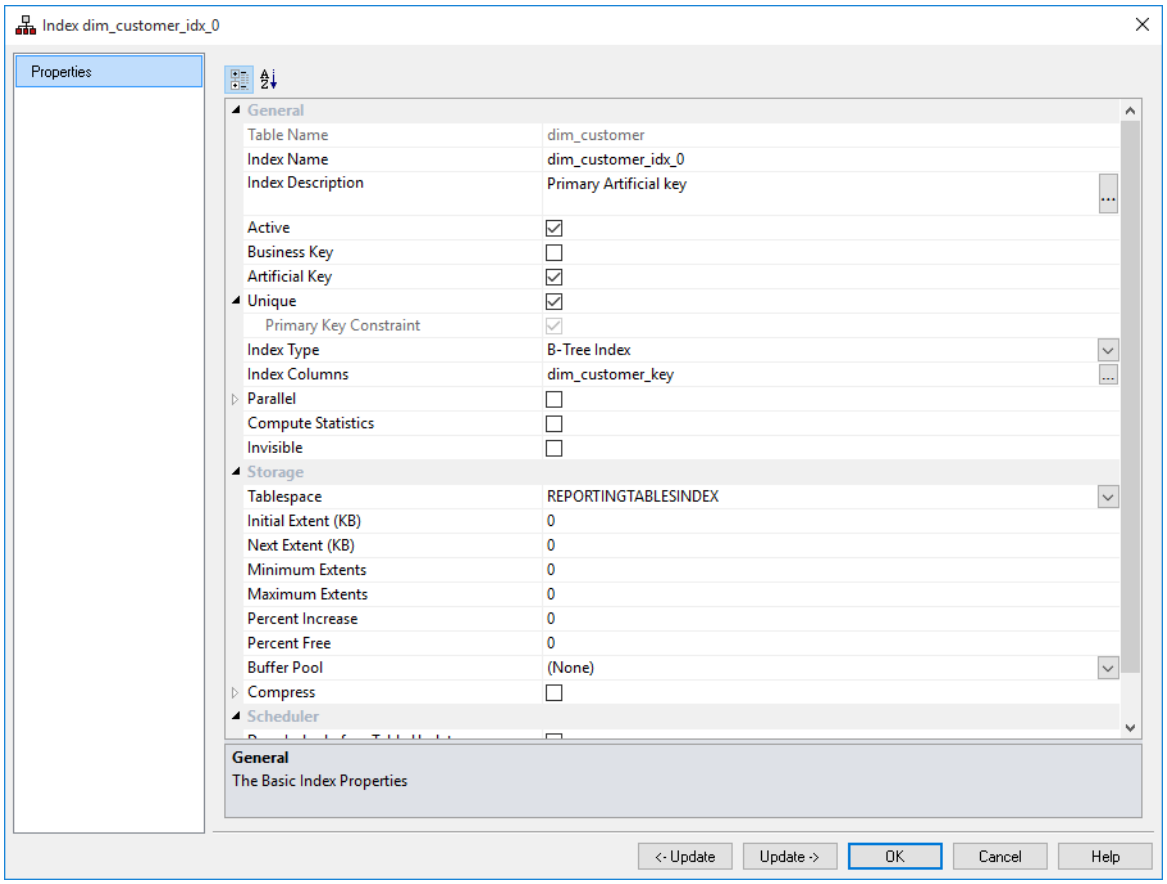
Properties

The **Properties** screen (see example below) can be selected via the right-click menu when positioned on an index name in the middle pane. The Update Buttons: **Update <-** and **Update ->** are used to move to the previous and next index respectively. The Update Buttons are not available when browsing all indexes in a group, project or repository.

- **SQL Server** example:



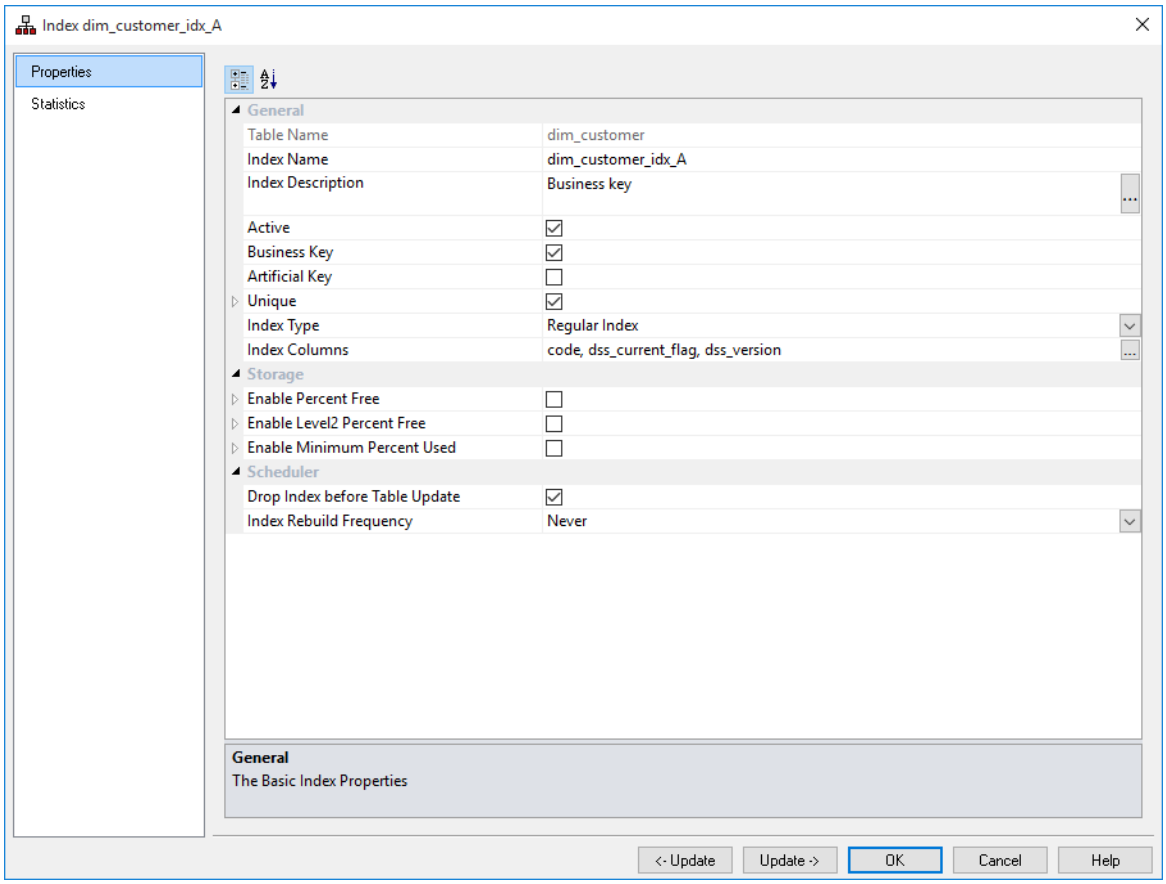
- **Oracle** example:



NOTE: Oracle 11g now allows for Indexes to be marked as **invisible**.

Invisible Indexes are maintained like any other index but they are ignored by the optimizer unless the OPTIMIZER_USE_INVISIBLE_INDEXES parameter is set to true at the instance or the session level. The default of this option is not set.

- IBM **DB2** example:



The fields in the **General** section are described below:

Field	Description
Index Name	Typically the table short name followed by: • **_idx_0** indicating primary key • **_idx_n** where n = any number from 1 indicating dimensional keys • **_idx_x** where x = any letter a thru z indicating business keys.
Index Description	• Free flow description of the index
Active	• When enabled means the index is in use. • When disabled means the index is not managed by the scheduler.
Business Key	Denotes a business key.
Artificial Key	When enabled indicates that this is the surrogate (artificial) key generated by the system.

Field	Description
Unique	Specifies that the index is a unique index **Note**: If both **unique and artificial** are set it is assumed to be a primary key constraint and it is added as such.
Index Type	Database-specific type of index On **SQL Server** the options are: • Nonclustered Index • Clustered Index • ColumnStore Index available for SQL2012 On **Oracle** the options are: • B-Tree Index • Bitmap Index • Function-based Index
Index Columns	Shows the columns in the order they will be applied to the index. The order can be changed using the up/down buttons at the left.
Non-Key (Include) Columns	Extend a Nonclustered Index to include Non-Key columns in addition to the key columns.
Index Filter Predicate	A filtered index is an optimized Nonclustered index, which uses a filter predicate to
Enable Parallel Build Degree	Parallelize the creation /build of the index. For unsupported database versions (Pre SQL2005) leave disabled.
Additional Index Create Options	Database-specific-and-compliant options to include in the generated CREATE INDEX statement.
Compute Statistics	**For Oracle**. When enabled, RED automatically collects statistics during index creation and rebuild. NOTE: Deprecated since 10g (but supported for backwards compatibility) as now statistics are always collected during index creation and rebuild.

The fields in the **Storage** section are described below:

Field	Description
Filegroup/Tablespace	Default Filegroup/Tablespace of the index that determines the storage location it is created in. Select (Default) to use the default.
Initial Extent (KB)	**For Oracle**. Size of the first extent allocated to the index when it is created. The default value of 0 will instead use the value from the Tablespace in which the index is stored.
Next Extent (KB)	**For Oracle**. Size of the next extent allocated to the index, unless Percent Increase is specified.

Field	Description
Minimum Extents	**For Oracle**. Number of extents to allocate when the index is created. The default value of 0 will instead use the value from the Tablespace in which the index is stored.
Maximum Extents	**For Oracle**. Total number of extents that can be allocated to the index. The default value of 0 will instead use the value from the Tablespace in which the index is stored.
Percent Increase	**For Oracle**. Percent by which the third and subsequent grow over the preceding extent. The default value of 0 will instead use the value from the Tablespace in which the index is stored.
Percent Free	**For Oracle**. Percentage [0-99] of space in each data block of the index to reserve for future updates. The default value of 0 will instead use the value from the Tablespace in which the index is stored.
Buffer Pool	**For Oracle**. Default buffer pool used to cache the data blocks of the index.
Compress	Compress index to reduce disk and memory use. For unsupported database versions (Pre SQL2008) select (Not Defined).
Enable Fill Factor	Setting this will enable setting of the Fill Factor. For unsupported database versions (Pre SQL2005) leave disabled.
Sort In tempdb	Use the tempdb to store the intermediate sort results that are used to build the index, which may reduce the index build time if tempdb is on different disks to the index.
Enable Percent Free	**For DB2.** Setting this will enable setting of the Percent Free to something other than the Database defaults.
Percent Free	**For DB2.** The amount of free space to leave in each extent/page of the index.
Enable Level2 Percent Free	**For DB2.** Setting this will enable setting of the Level2 Percent Free to something other than the Database defaults.
Level2 Percent Free	**For DB2.** The amount of free space in each index level 2 page.
Enable Minimum Percent Used	**For DB2.** Setting this will enable setting of the Minimum Percent Used to something other than the Database defaults.
Minimum Percent Used	**For DB2.** The threshold of the minimum percentage of space used on an index leaf page.

The fields in the **Scheduler** section are described below:

Field	Description
Drop Index before Table Update	Drop the index before running the table update procedure and recreate the index after the update procedure has completed.

Field	Description
Index Rebuild Frequency	Optional frequency that the index is automatically rebuilt by the Scheduler.

Indexes are normally managed by the scheduler as part of the normal processing of a table.

Refer to ***Ws_Maintain_Indexes*** (on page 1046). This function allows the control of index drop and creation from within a procedure. Typically, this function is called when using partitioned tables.

WhereScape®

CHAPTER 25
DOCUMENTATION AND DIAGRAMS

WhereScape RED includes the ability to document the data warehouse based on the information stored against the metadata for all the tables and columns in the data warehouse.

The documentation will only be meaningful if information is stored in the metadata. The business definition and a description should be stored against all columns that will be visible to end users.

The following sections describe how to ***generate*** (see "***Creating Documentation***" on page 834) and ***read*** (see "***Reading the Documentation***" on page 839) the documentation.

Also included is a section on information that is available to assist in the connection of end user ***query tools*** (see "***Query Tool Access***" on page 863).

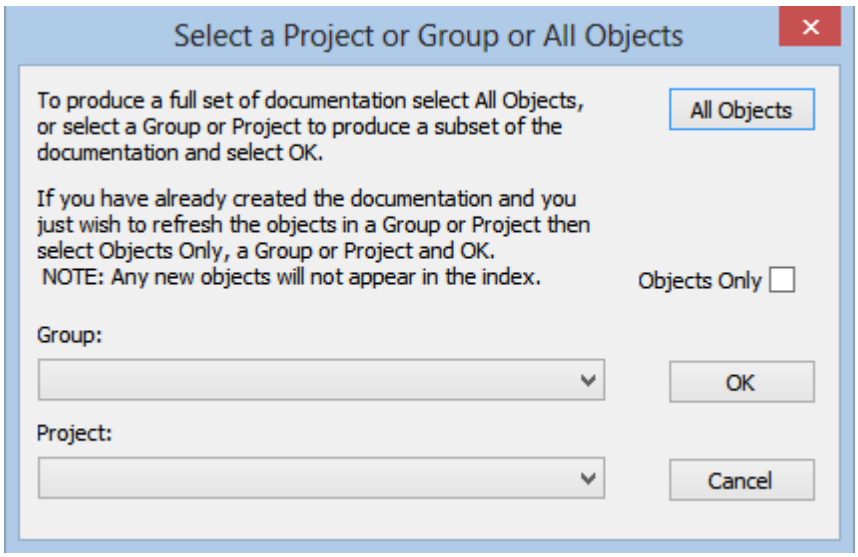
IN THIS CHAPTER

CREATING DOCUMENTATION

Create documentation

To create the documentation for the components of the data warehouse, select **Doc** from the builder menu bar, then **Create Documentation**.

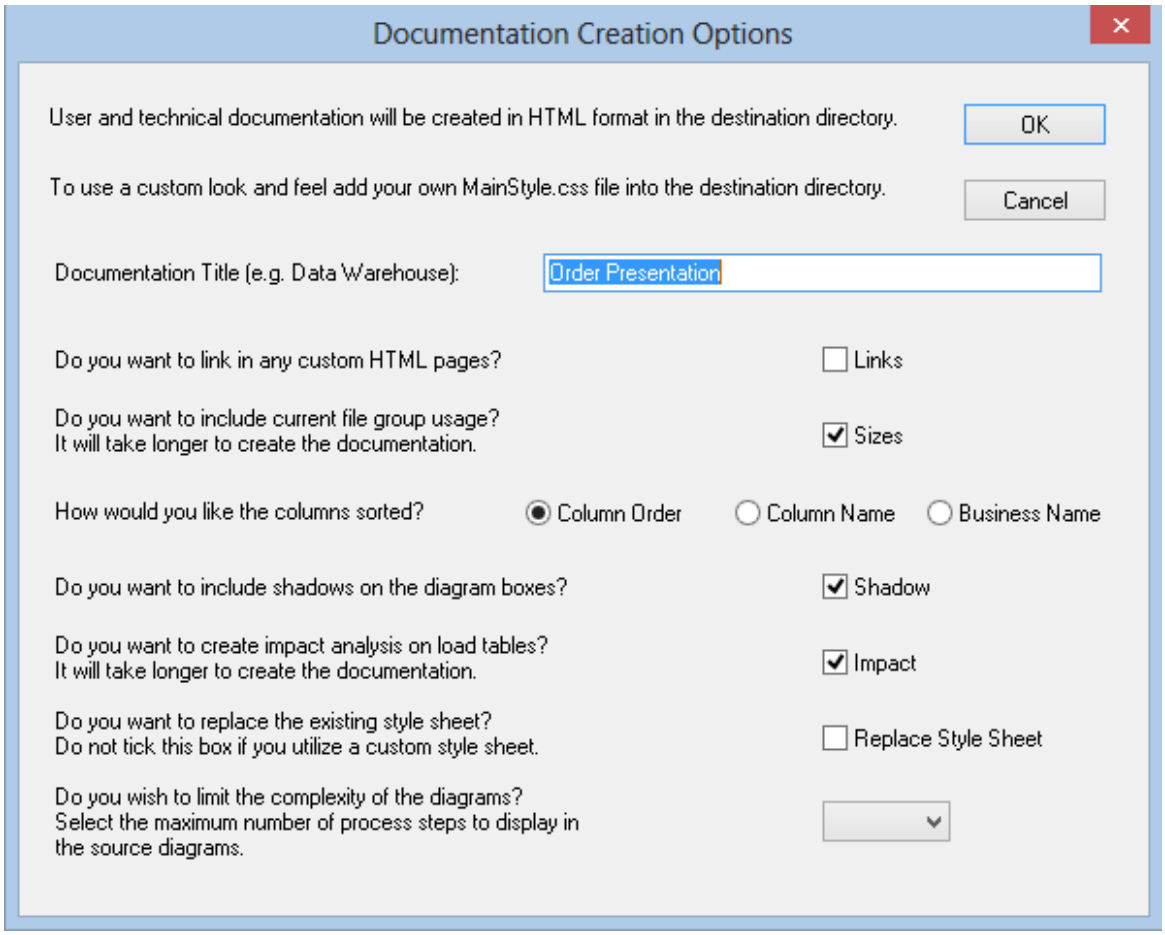
If the repository has projects or groups in use then the following dialog will appear to allow the selection of a specific group or project. The default is all objects in the repository.



A file dialog will appear next. Select a file path (directory) under which to save the HTML files that will be produced.

A style sheet called **mainstyle.css** is created if it does not exist. If this style sheet exists then it is not overwritten. The style sheet can therefore be modified to match an existing intranet standard.

The next screen allows for the inclusion of a banner and user defined links. It also provides some build options:



The size checkbox instructs the documentation generator to examine the size of all tables and indexes in the database. This process may be slow in some situations, thus should normally be used only for final documentation.

The sorted checkbox sorts the columns within the tables into alphabetical order. By default, the columns are in the order in which they appear within the tables.

Creating a header

If you check the banner frame option, then a banner (heading) will be created at the top of each page in the documentation. You will be prompted for height of the banner frame in pixels (default is 60), an image file (jpg or gif) and any banner text. It is recommended that any image be relatively small (60 pixels high or approximately 1/2 an inch) as it will appear on every page.

Adding Links

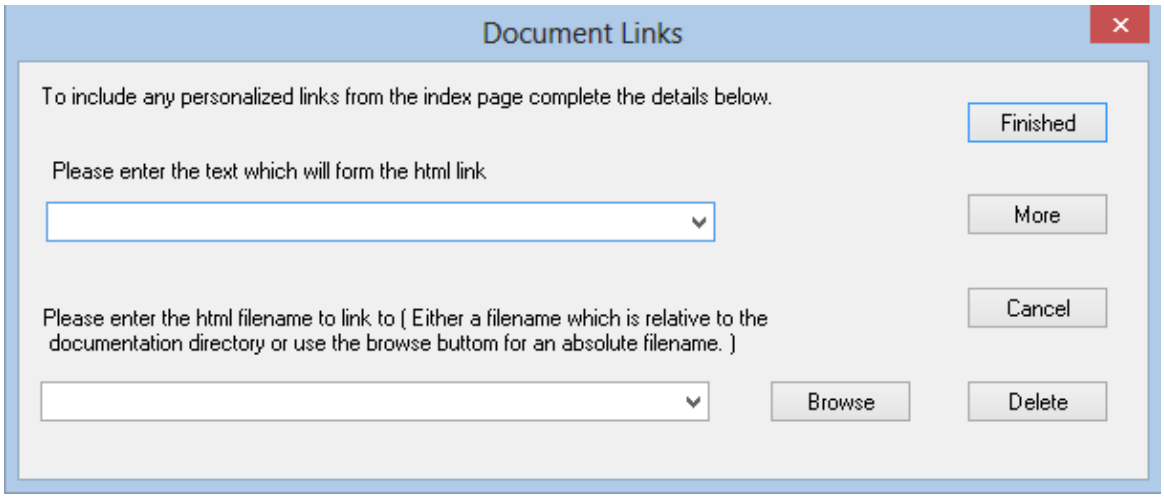
Custom information can be linked into the generated documentation.

This means that every time the documentation is regenerated, custom information will be included. In this way, the complete documentation for the data warehouse can be viewed in one location.

If you check the add links option, then you will be prompted to include personalized links from index pages. These links must be to previously created HTML files.

Index pages (linkage points) are available at three points:

- index - initial page
- techindex - technical documentation initial page
- indexuser - user documentation initial page



Multiple links can be added to each index page by using the More option.

Adding glossary elements

As part of the user documentation a glossary is produced defining all the business terms and column names used in the data warehouse. This glossary is primarily based on the columns in the dimension and fact tables. Additional information can, however, be added via the 'Ws_Api_Glossary' procedure defined in the procedures chapter. This procedure allows the manual inclusion of glossary elements that will be stored in the metadata repository and added to the glossary whenever the documentation is recreated.

BATCH DOCUMENTATION CREATION

WhereScape RED includes the ability to document the data warehouse based on the information stored against the metadata for all the tables and columns in the data warehouse. In a larger environment, it may be a good idea to generate documentation in batch mode.

The following syntax chart illustrates the options available:

```
med.exe /BD { /U UserName { /P Password } } /C OdbcSource { /M Schema } {
/N FullName } /D Directory { /G GroupName | ProjectName } /S NumHops
```

Note: {} indicates an optional parameter and | indicates alternative values.

Parameter Descriptions

The following parameters are available:

Parameter	Specify Value?	Mandatory?	Description
BD	No	Yes	Indicates batch documentation mode.
U	Yes	Sometimes *1	Username parameter.
P	Yes	Sometimes *1	Password parameter.
C	Yes	Yes	ODBC data source parameter.
M	Yes	Sometimes *2	Metadata schema parameter.
N	Yes	No	Full user name parameter, only for logging purposes.
D	Yes	Yes	Directory name where documentation is created.
G	Yes	No	Group or Project name if specified. All Objects if not included.
S	Yes	Yes	Number of processes/hops in the source diagrams.

Notes:
*1. User Name and Password are not required when using a trusted connection or operating system authentication.

*2. Metadata Schema is required for DB2, but is not required for Oracle and SQL Server.

Example

The following example connects to a SQL Server repository called WslWarehouse, using a trusted connection into the C:\Temp\my_doco directory with 4 hops in diagrams:

```
med /BD /C "WslWarehouse" /D "C:\Temp\my_doco" /S "4"
```

READING THE DOCUMENTATION

To read the documentation you have created, select **Doc** from the builder menu bar, then **Display Documentation**. This will launch a browser and display the contents of index.html. Alternatively you can access the HTML pages directly from their saved location.

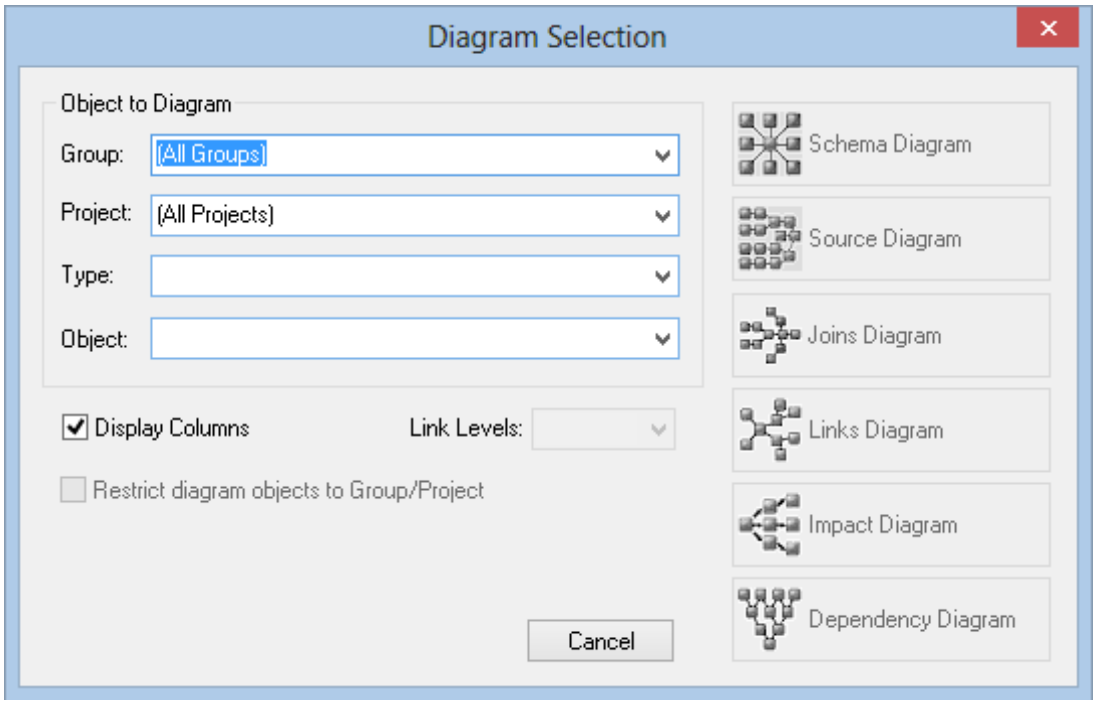
DIAGRAMS

TYPES OF DIAGRAMS

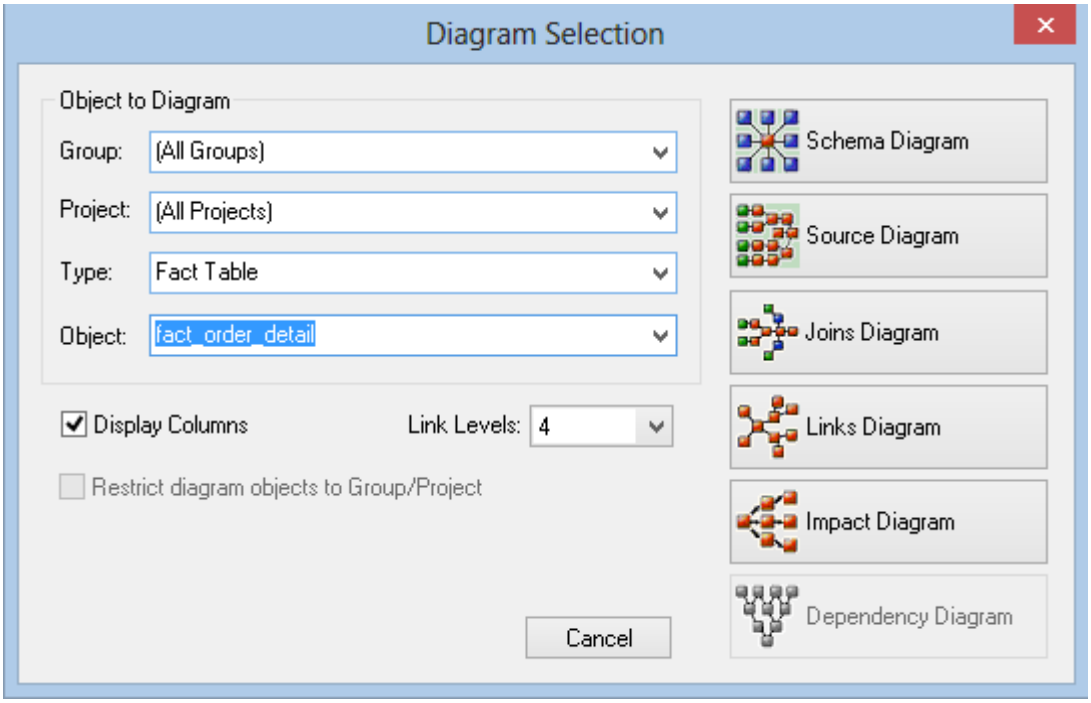
Six types of diagrams are provided to give visual representation of what has been created. These are

- The ***Schema Diagram*** (on page 842)
- The ***Source Diagram*** (on page 844)
- The ***Joins Diagram*** (on page 848)
- The ***Links Diagram*** (on page 849)
- The ***Impact Diagram*** (on page 850)
- The ***Dependency Diagram*** (on page 851)

1 To display the **Diagram Selection** dialog, click on the diagrammatic view button 



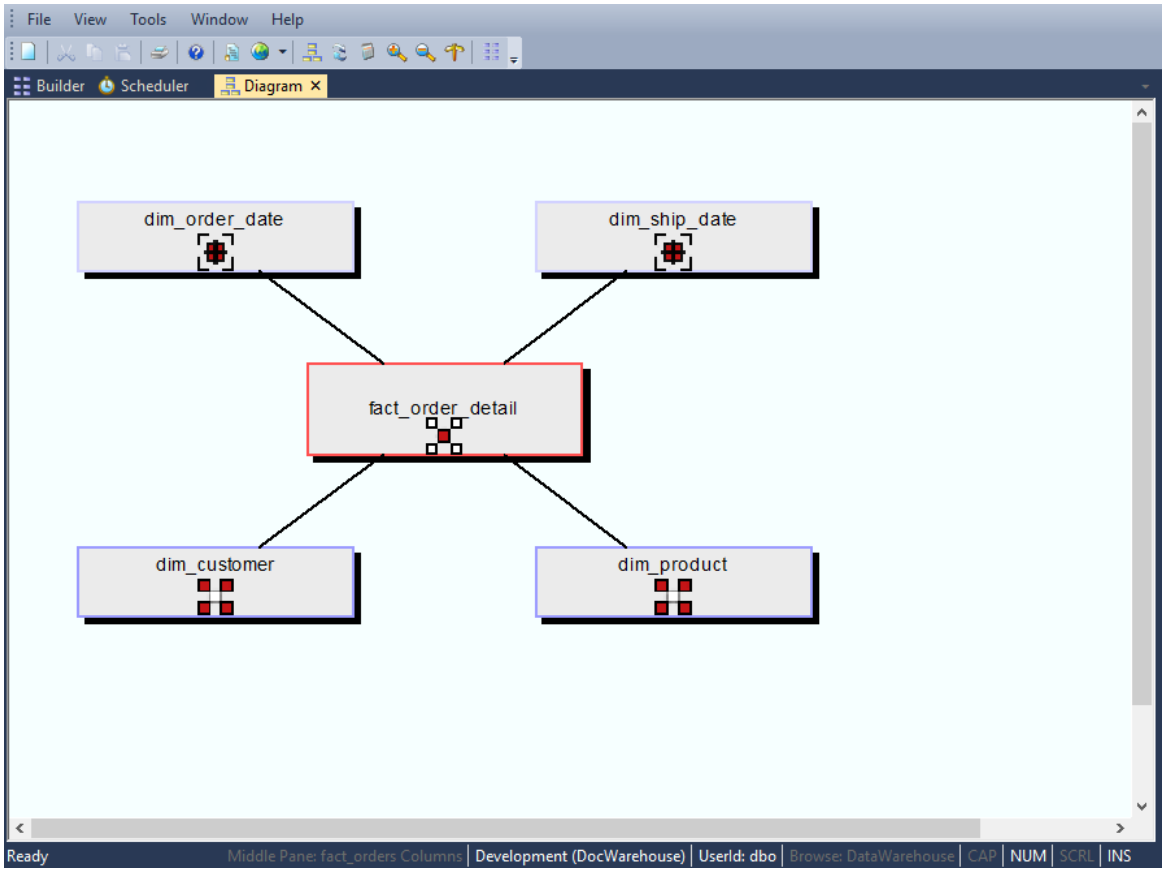
2 Choose the object to diagram by optionally choosing the **Type** to limit the selection list; and then selecting the **Object**. The diagram type buttons on the right will then become active and you can choose the type of diagram to display.



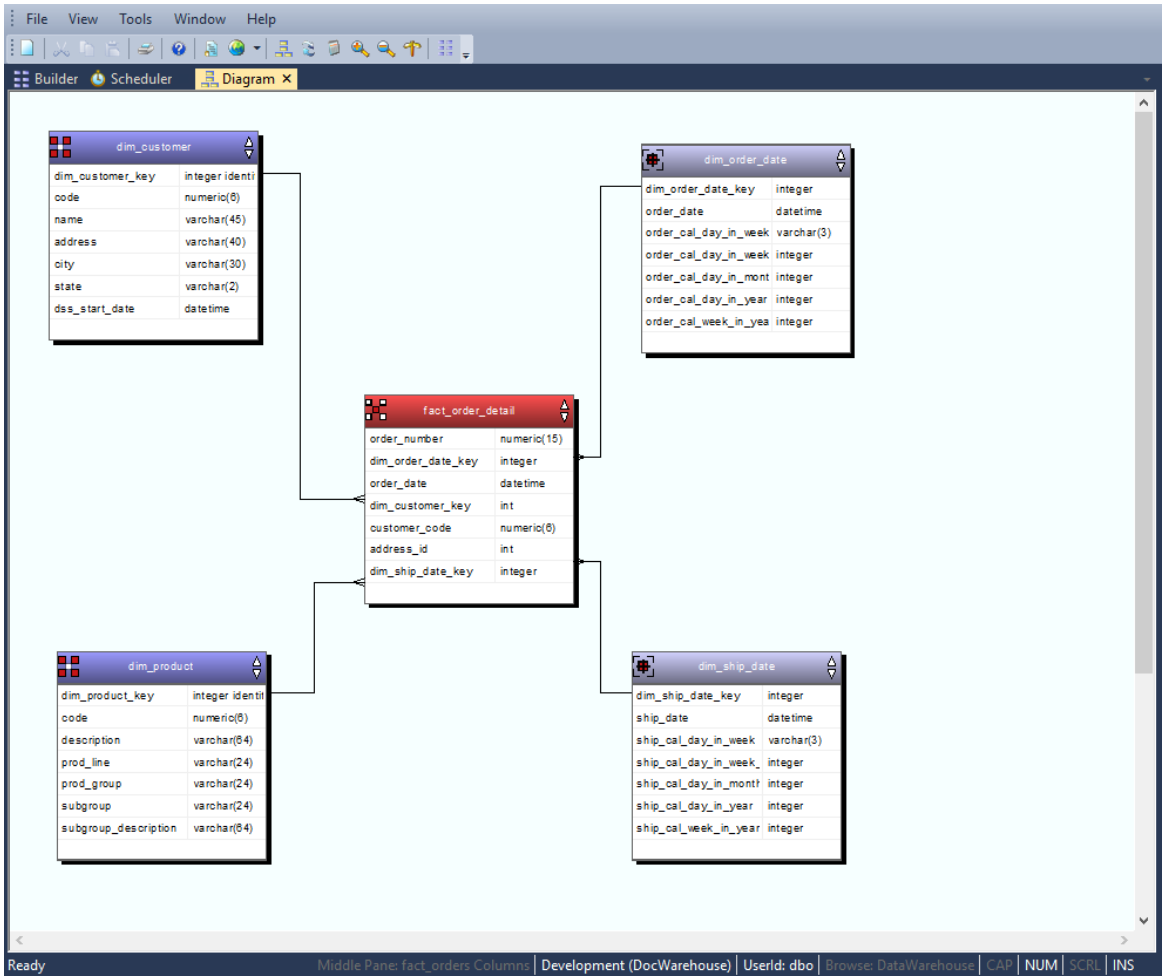
SCHEMA DIAGRAM

A star schema diagram can be displayed for a fact table, an aggregate table and an OLAP cube. It shows the central table with the outlying dimensions.

An example of a **Schema** diagram in **Standard Diagram** format is displayed below.



An example of a **Schema** diagram in **Detail Diagram** format is displayed below.

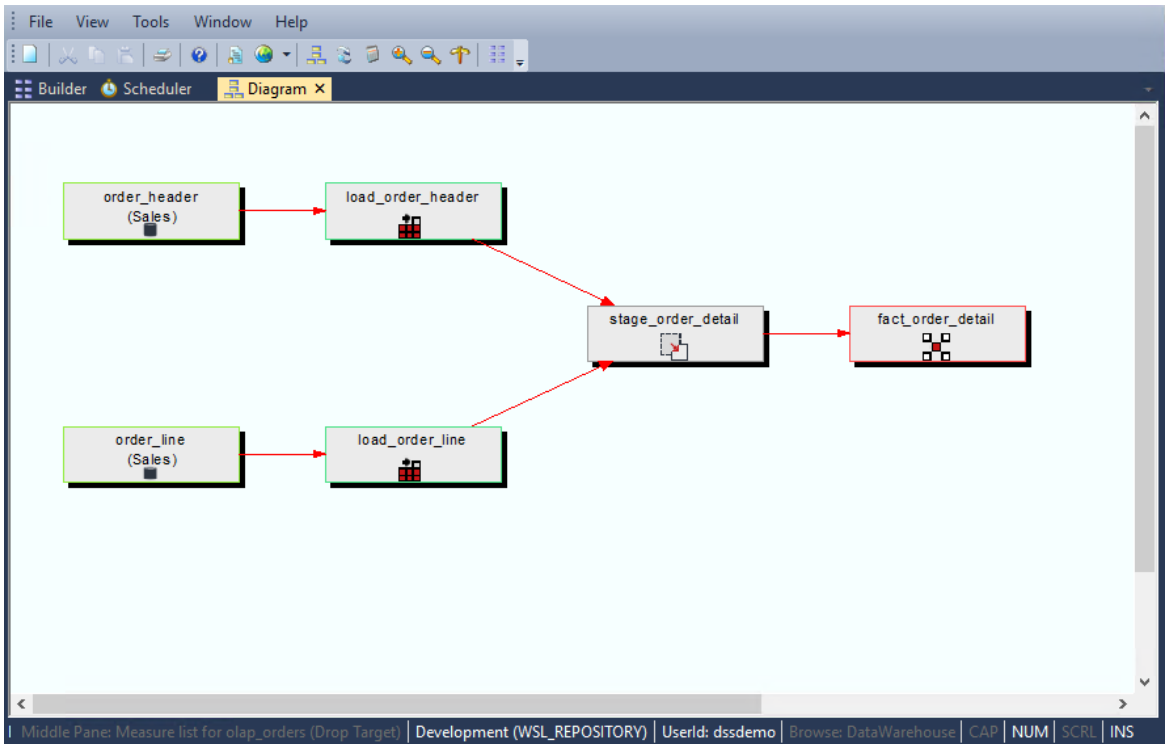


The star schema diagrams are produced in **Standard Diagram** format as part of the user and technical documentation when you select **Doc > Create Documentation** from the main builder window.

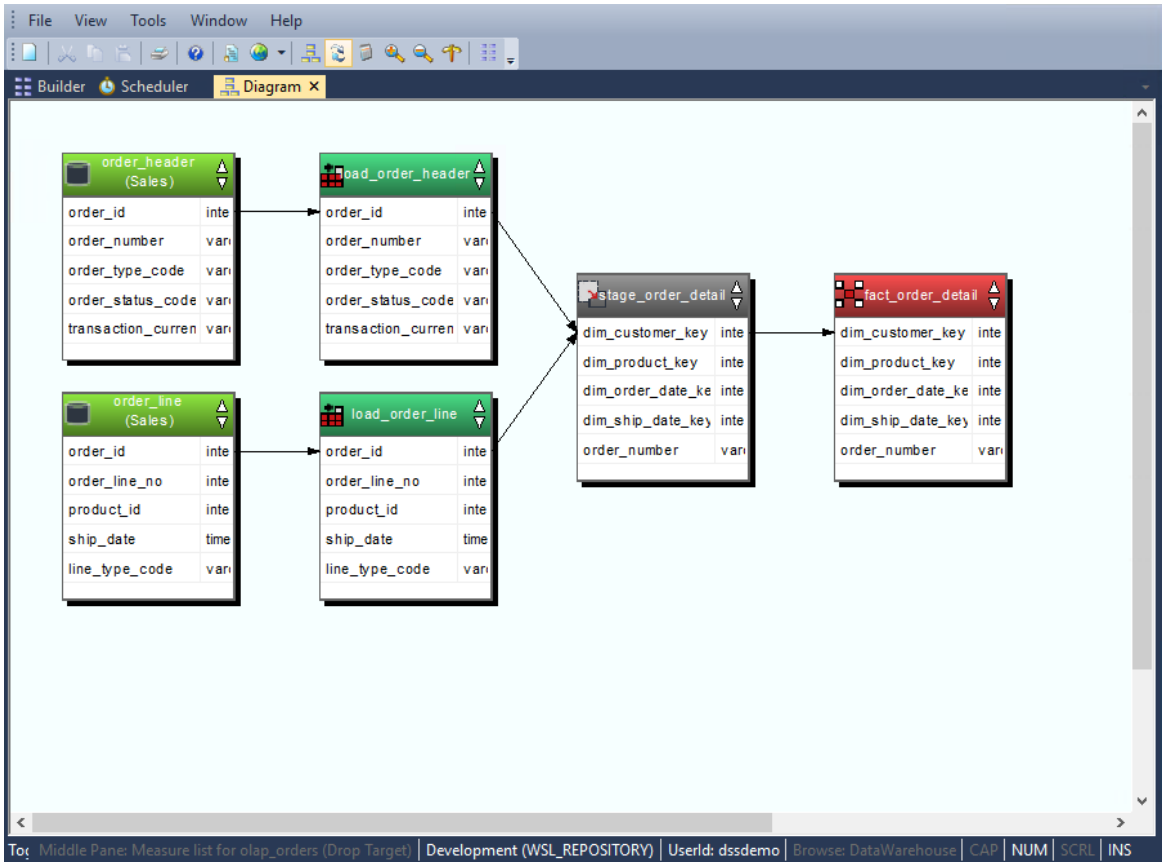
SOURCE DIAGRAM

A source tracking diagram can be displayed for any table. It shows connections back from the chosen table to the source tables from which information was derived. Hovering the cursor over a line shows additional information. For lines going into load tables, the source of the data will be displayed; while for other lines in the diagram, the procedure used to move data between two tables is displayed.

An example of a **Source** diagram in **Standard Diagram** format is displayed below.



An example of a **Source** diagram in **Detail Diagram** format is displayed below.

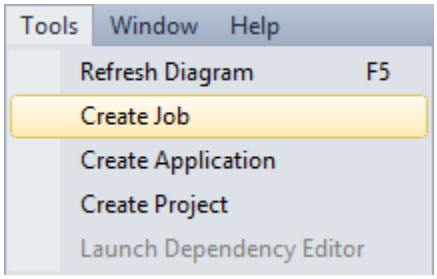


The Source diagrams are produced in **Standard Diagram** format as part of the technical documentation when you select **Doc > Create Documentation** from the main builder window.

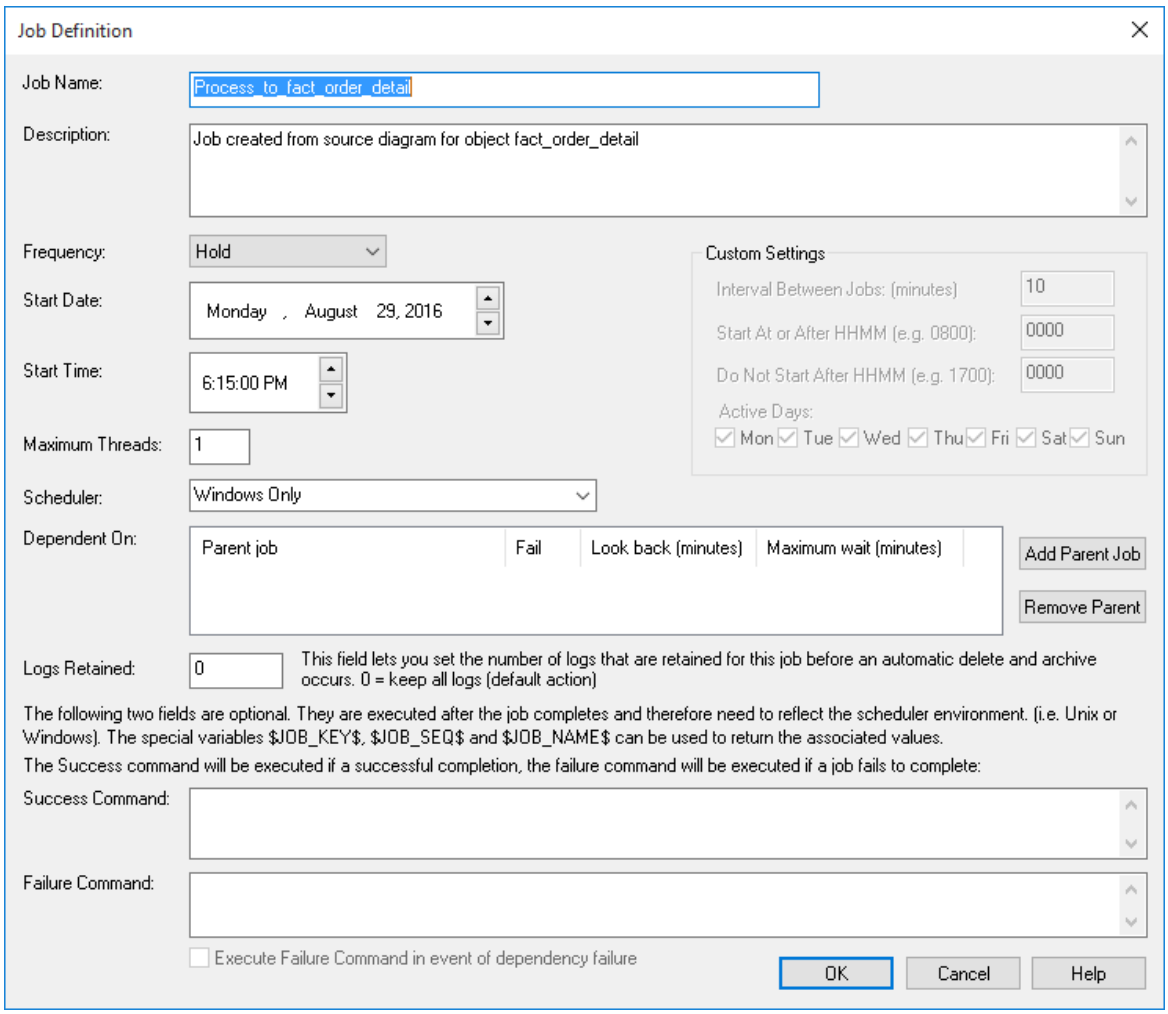
Creating a Job from a Source Diagram

Once a source tracking diagram has been created for a table, a scheduler job can be generated from the diagram. This job will be called **Process_to_table_name**, where table_name is the name of the table the track back diagram was run for.

To create a Job, select **Create Job** from the **Tools** menu after the diagram is displayed:

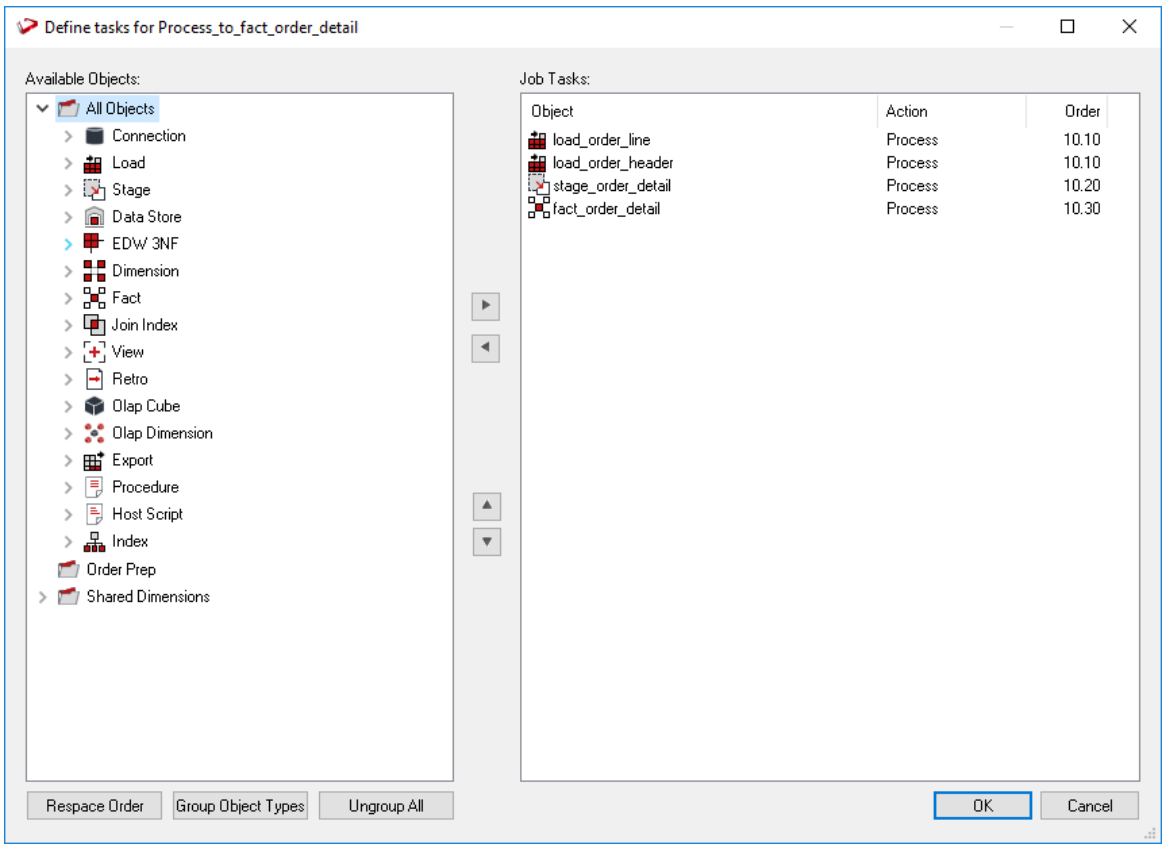


The Job Definition is then displayed:

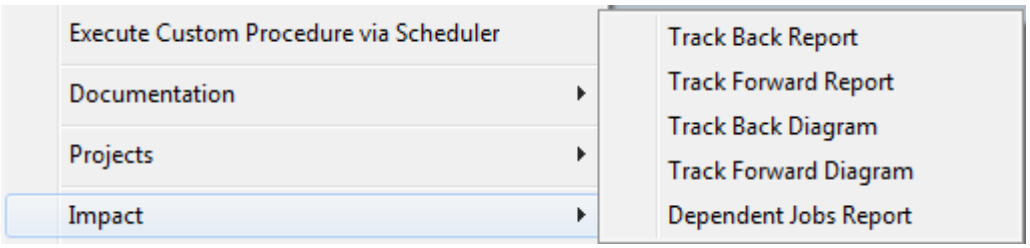


Make any changes here that are required and click **OK**.

For the diagram above, a job is created with the following **tasks**:



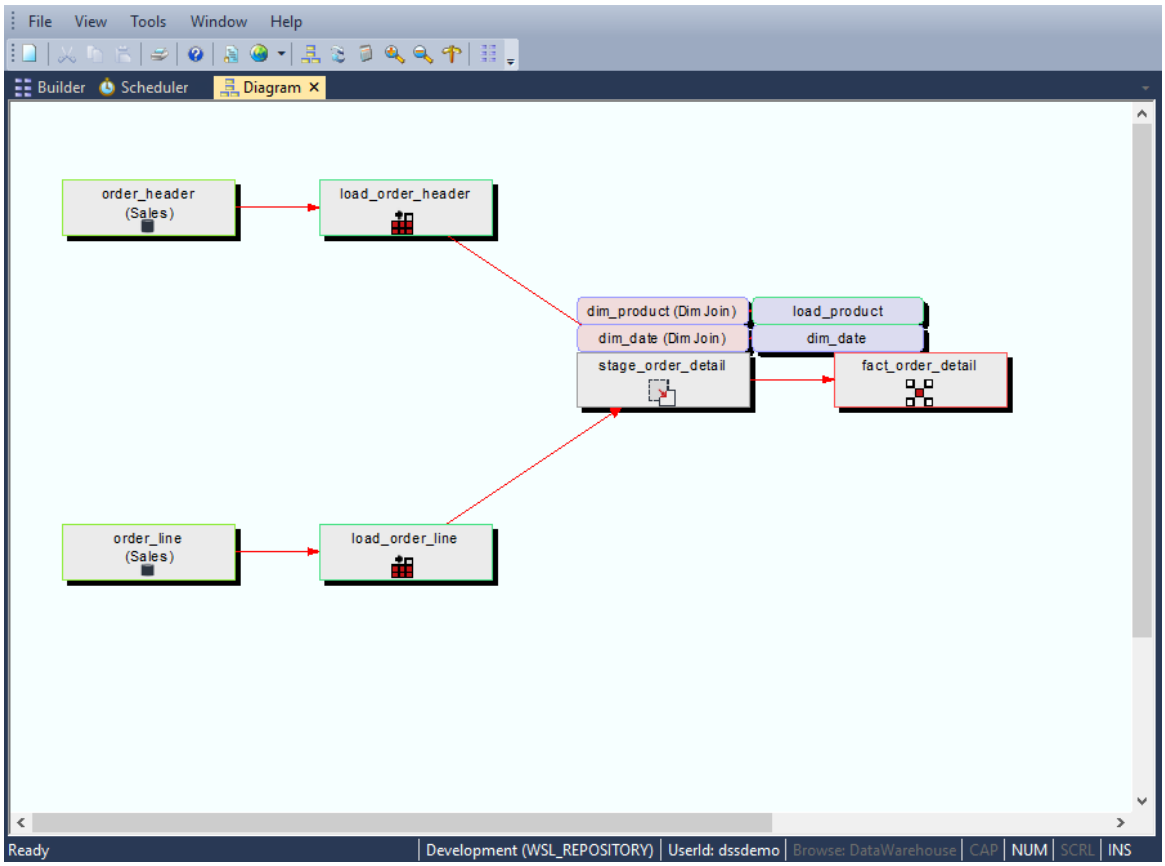
Note: It is also possible to display the source diagram by right-clicking on a table and choosing **Impact/Track Back Diagram:**



JOINS DIAGRAM

A data join track back diagram can be displayed for any table. It shows connections back from the chosen table to the source tables from which information was derived and includes dimension table joins. Hovering the cursor over a line shows additional information. For lines going into load tables, the source of the data will be displayed; while for other lines in the diagram, the procedure used to move data between two tables is displayed.

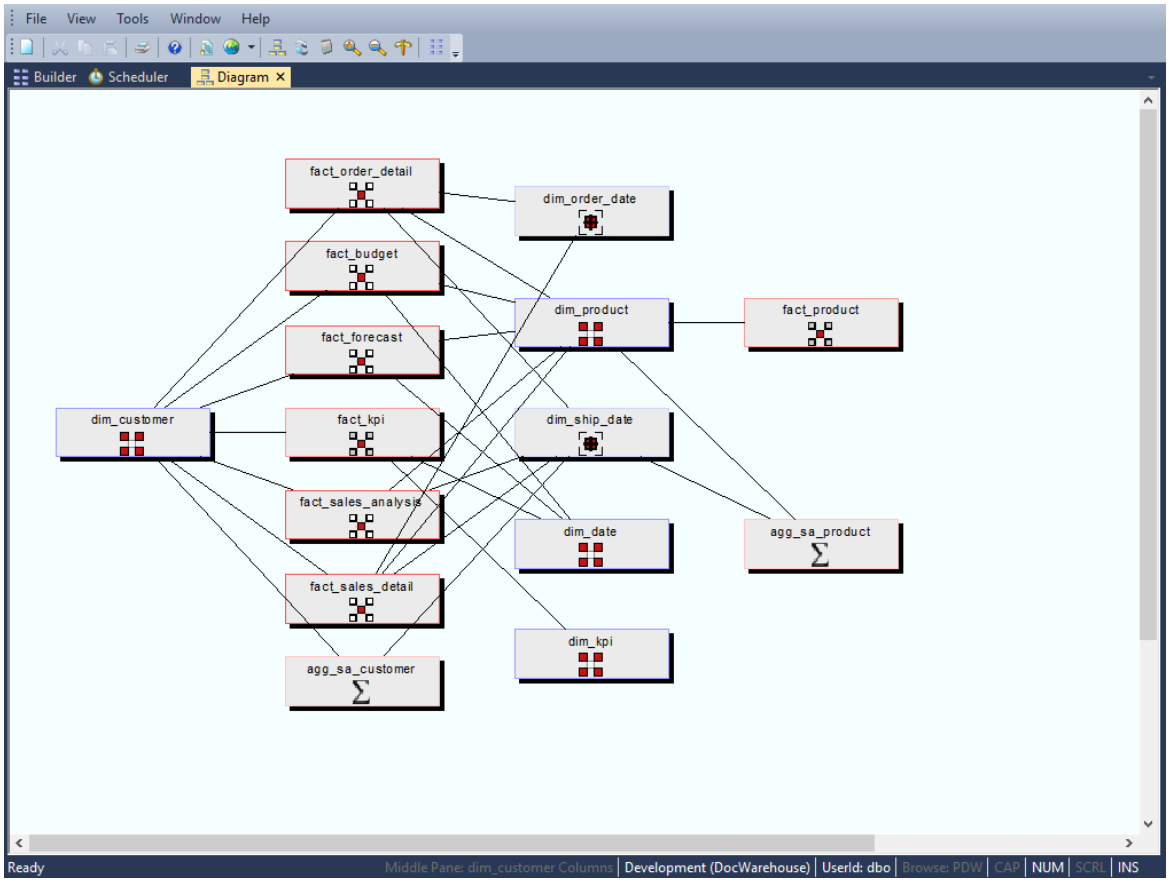
An example of a **Joins** diagram in **Standard Diagram** format is displayed below.



LINKS DIAGRAM

A linked tables diagram can be shown for any table. It shows relationships between tables, looking out from the chosen table a selected number of hops. The number of hops is determined by table relationships and source and target relationships.

An example of a **linked tables** diagram in **Standard Diagram** format is displayed below.

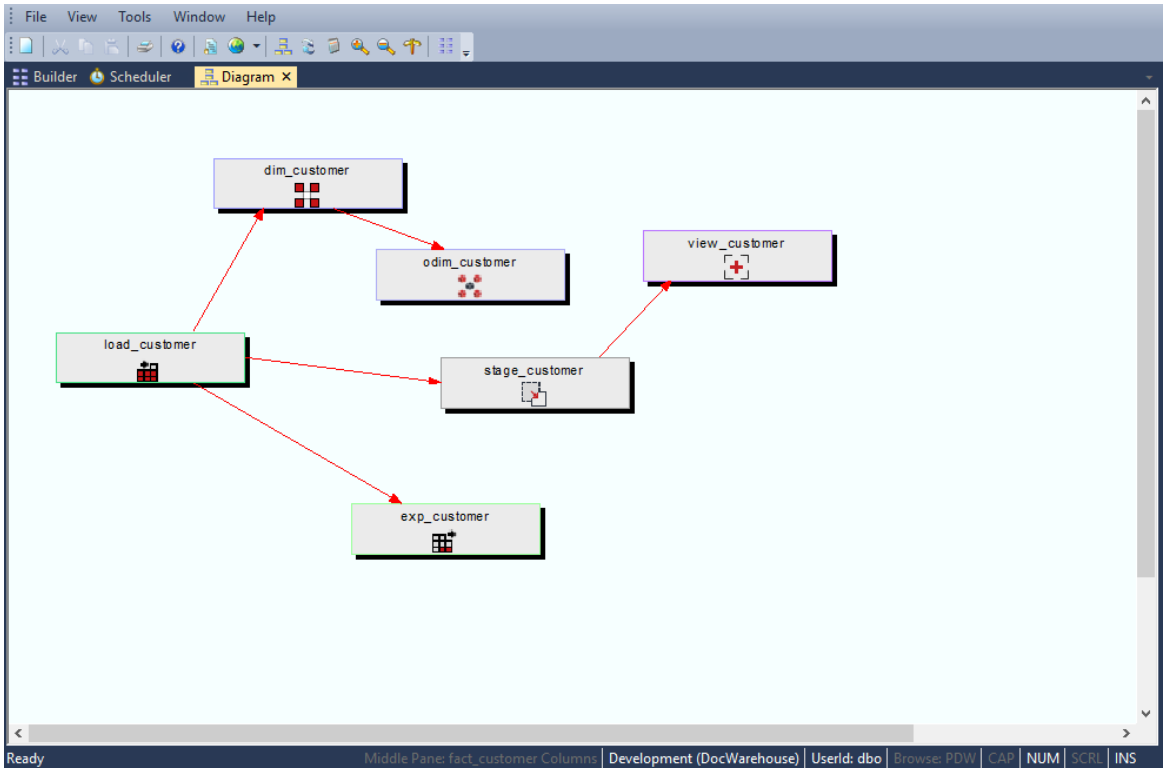


IMPACT DIAGRAM

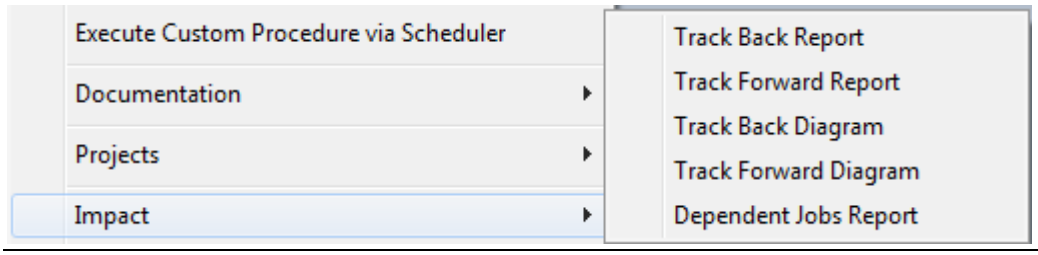
A track forward impact diagram can be displayed for any table. It shows connections forward from the chosen table to the subsequent tables built with columns from this table.

A track back impact diagram can be displayed for any table. It shows connections backwards from the chosen table to the previous tables.

An example of an **Impact** diagram in **Standard Diagram** format is displayed below.



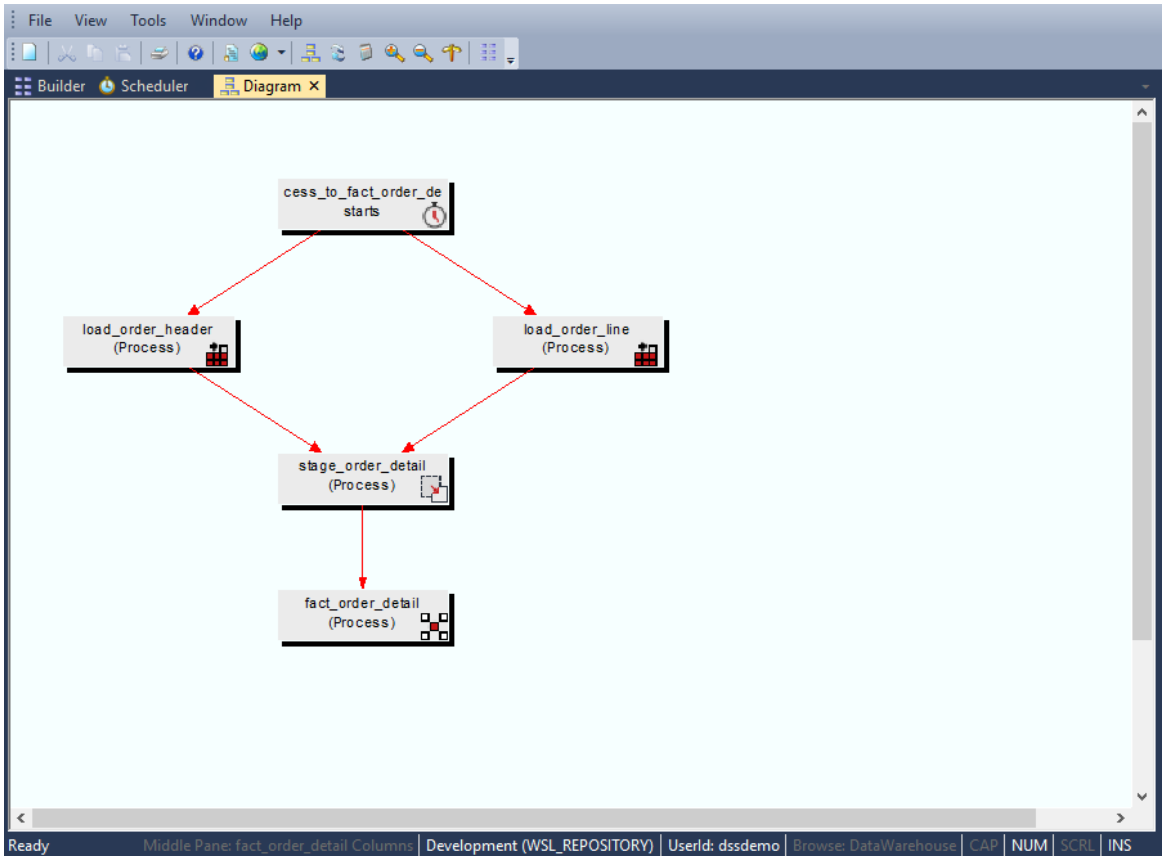
Note: It is also possible to display the track back / forward diagram by right-clicking on a table and choosing **Impact > Track Back Diagram or Impact > Track Forward Diagram:**



DEPENDENCY DIAGRAM

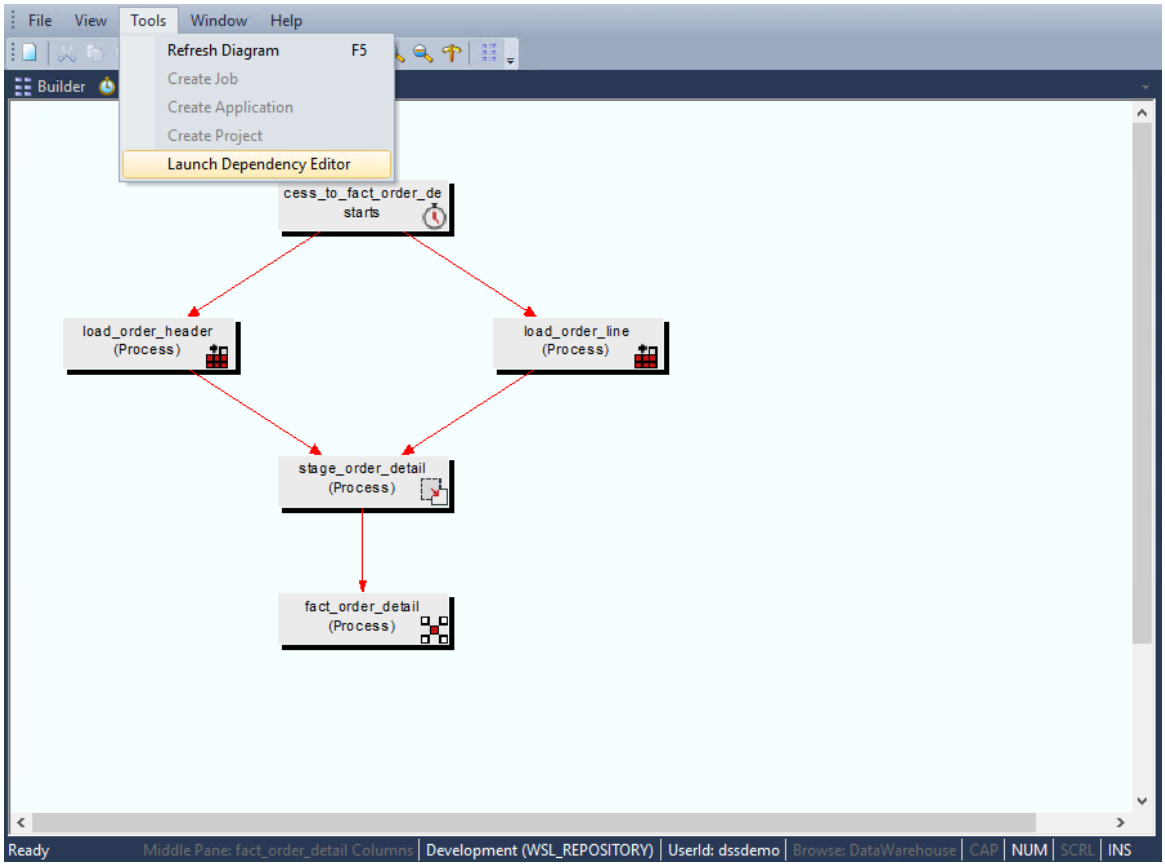
A job dependency diagram can be displayed for any job defined in the WhereScape RED scheduler. It shows the parent and child relationships between tasks within a job.

An example of a **Dependency** diagram in **Standard Diagram** format is displayed below.

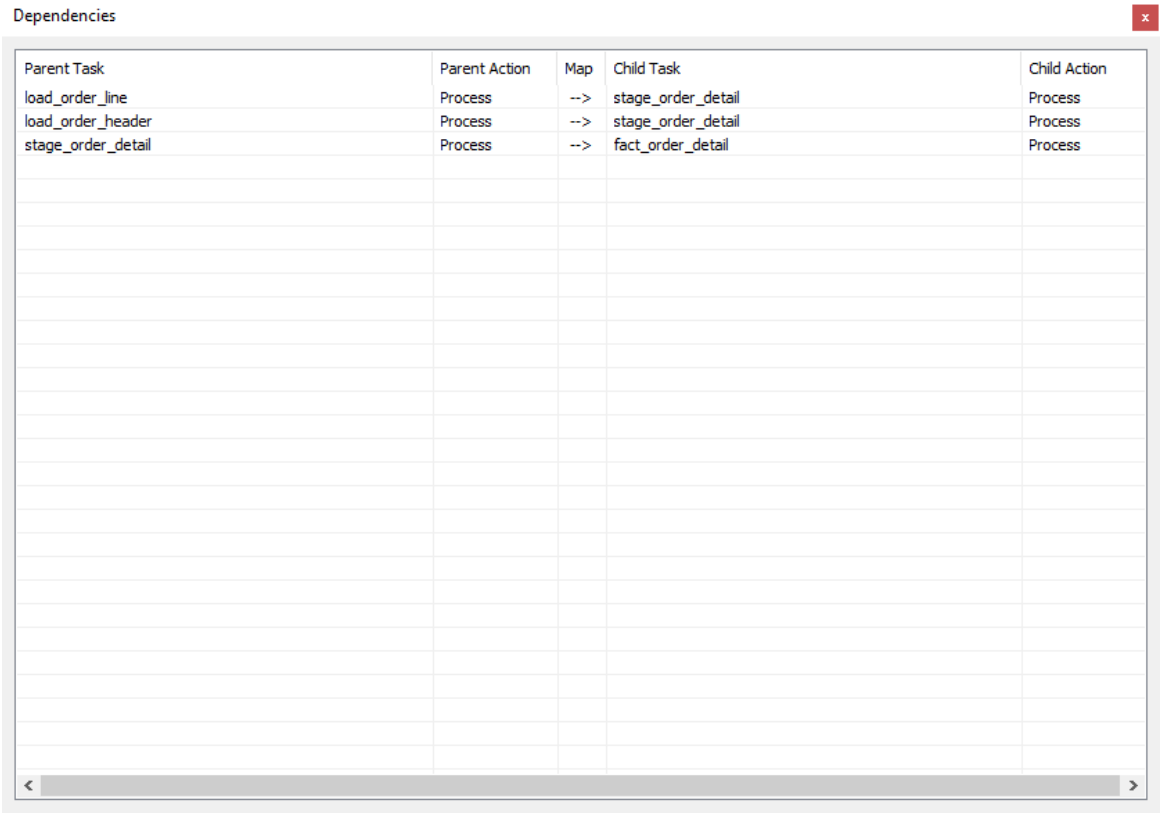


Editing a Job's Dependencies from a Job Dependency diagram

Once a Job Dependency diagram has been created for a job, its dependencies can be edited from the diagram. To do this, select **Launch Dependency Editor** from the **Tools** menu:



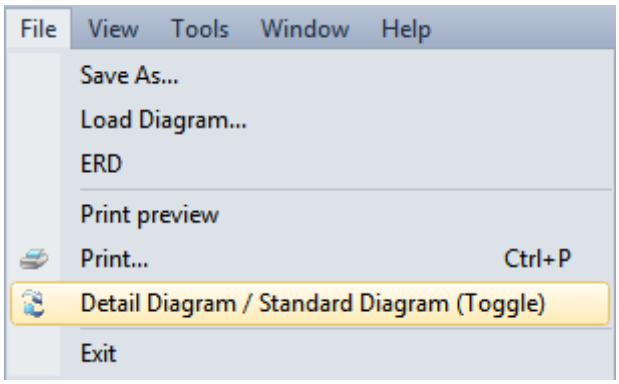
The dialog that follows allows you to edit the dependencies within a job:



WORKING WITH DIAGRAMS

Diagram Display

Once displayed there are two modes for diagram display; standard and detailed. To move between displays, select **File > Detail Diagram / Standard Diagram (Toggle)**

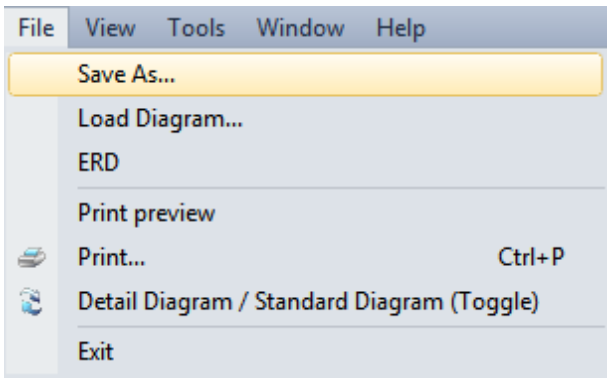


or use the **toggle** button.

Diagram Save

A diagram can be saved either as a meta file or as a jpeg image. If saved as a meta file it can be subsequently reloaded and re-edited. A jpeg cannot be reloaded into WhereScape RED.

The diagram can be saved by selecting **File > Save As ...**



Note: By default the diagram is saved as a meta file.

Diagram Load

If a diagram has been saved as a meta file, it can be reloaded. To reload a saved meta file switch to diagrammatic view and select the menu option **File > Load Diagram**. A dialog box will allow you to choose a windows meta file (*.wmf). If the meta file had previously been saved from WhereScape RED then the diagram will be loaded.

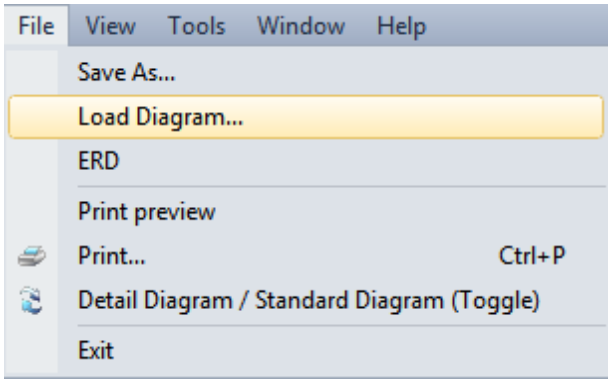


Diagram Print

A diagram can be printed by selecting **File > Print ...**

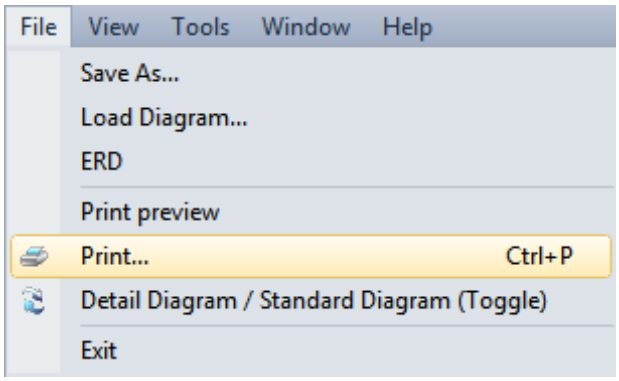
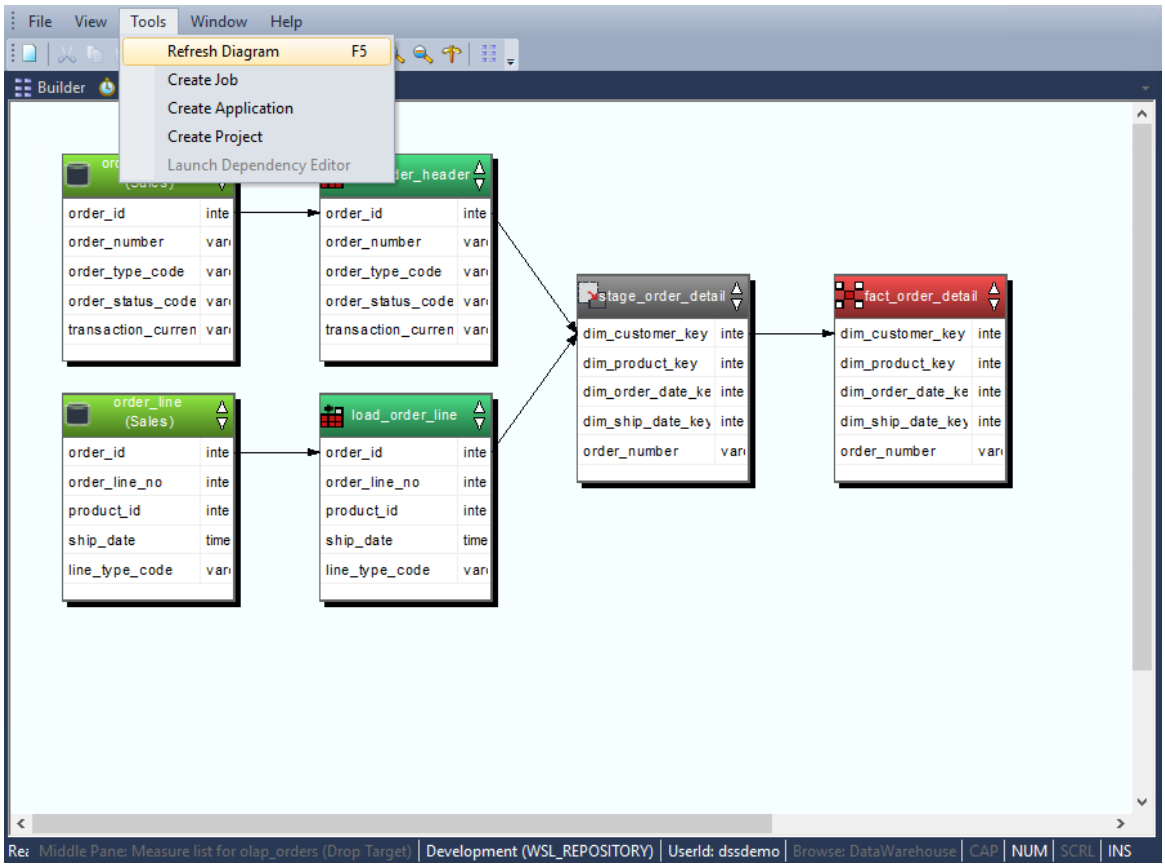


Diagram Refresh

Once a diagram has been displayed, it can be refreshed by choosing **Tools/Refresh Diagram**, or by pressing **F5**.

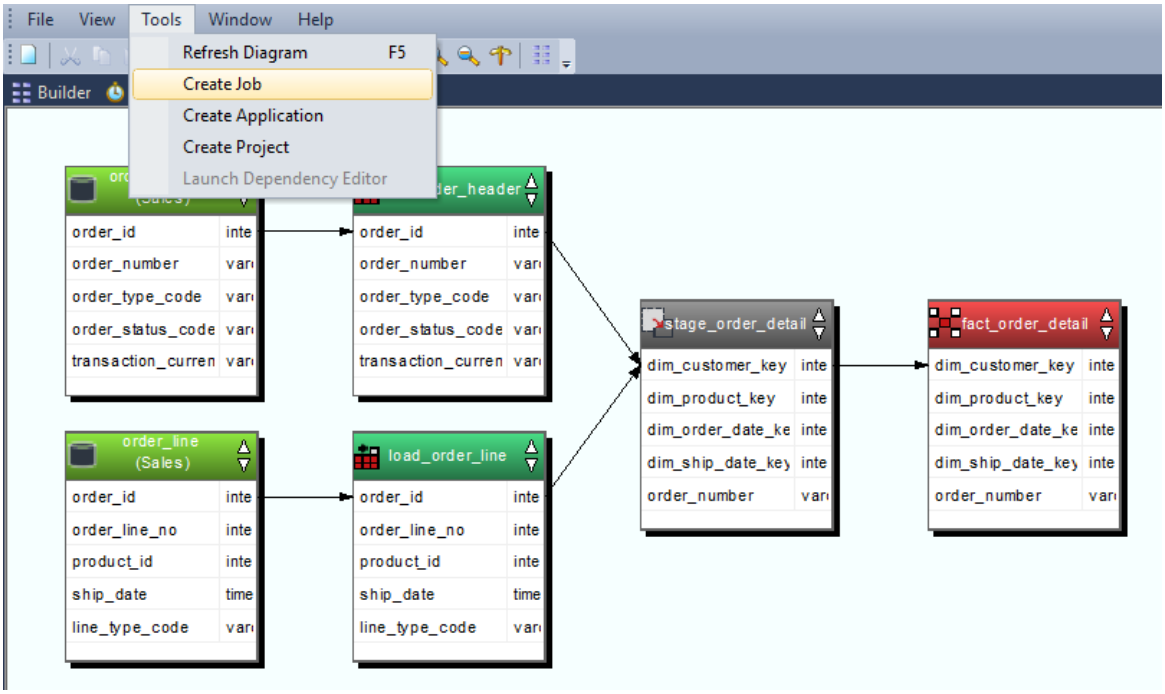


CREATING A JOB FROM A DIAGRAM

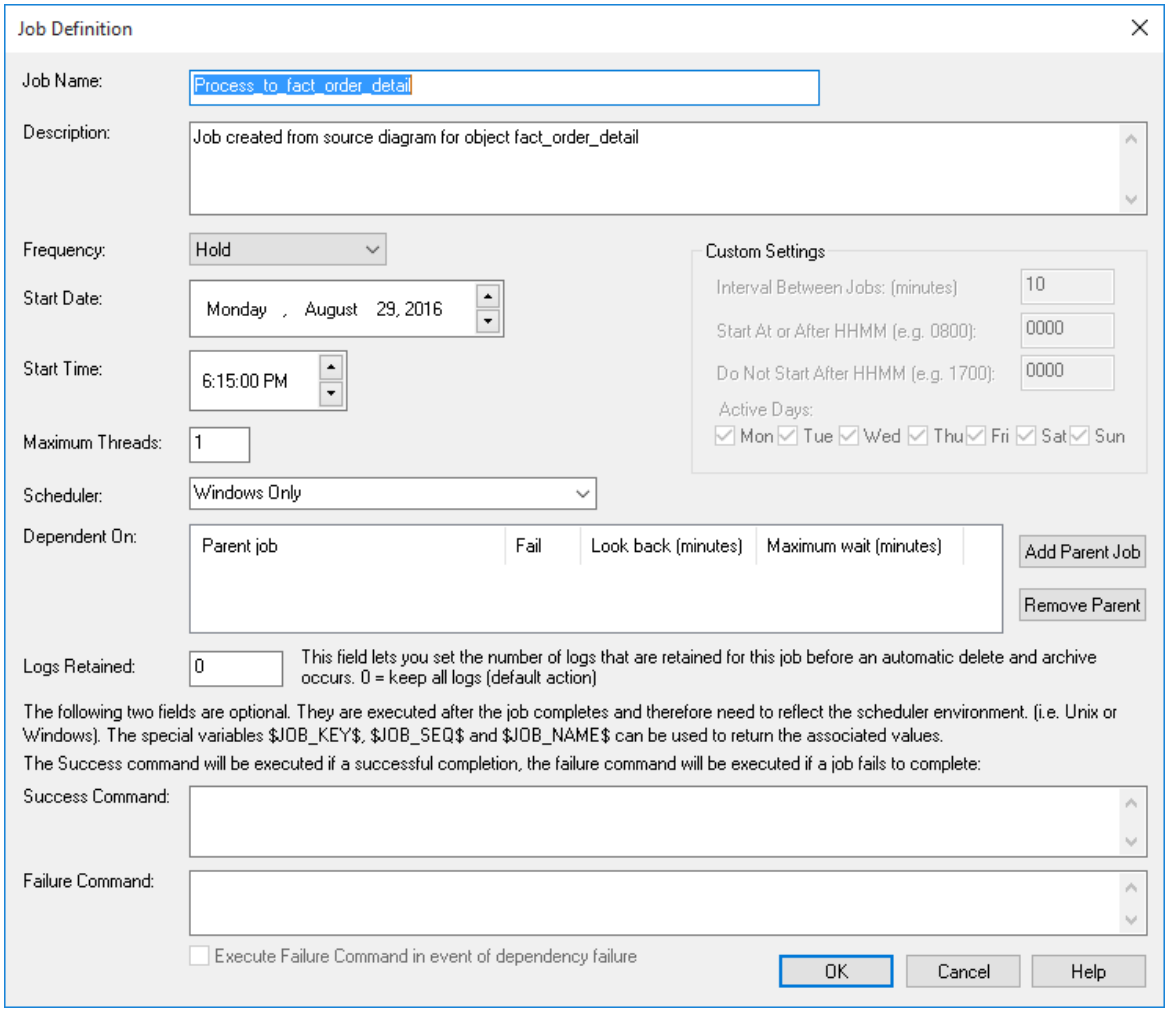
A job can be created from a **Source** diagram or a **Joins** diagram.

To Create a Job from a Diagram

1 Once the diagram is displayed, select **Tools/Create Job**.



2 Edit the job as required and click **OK**.

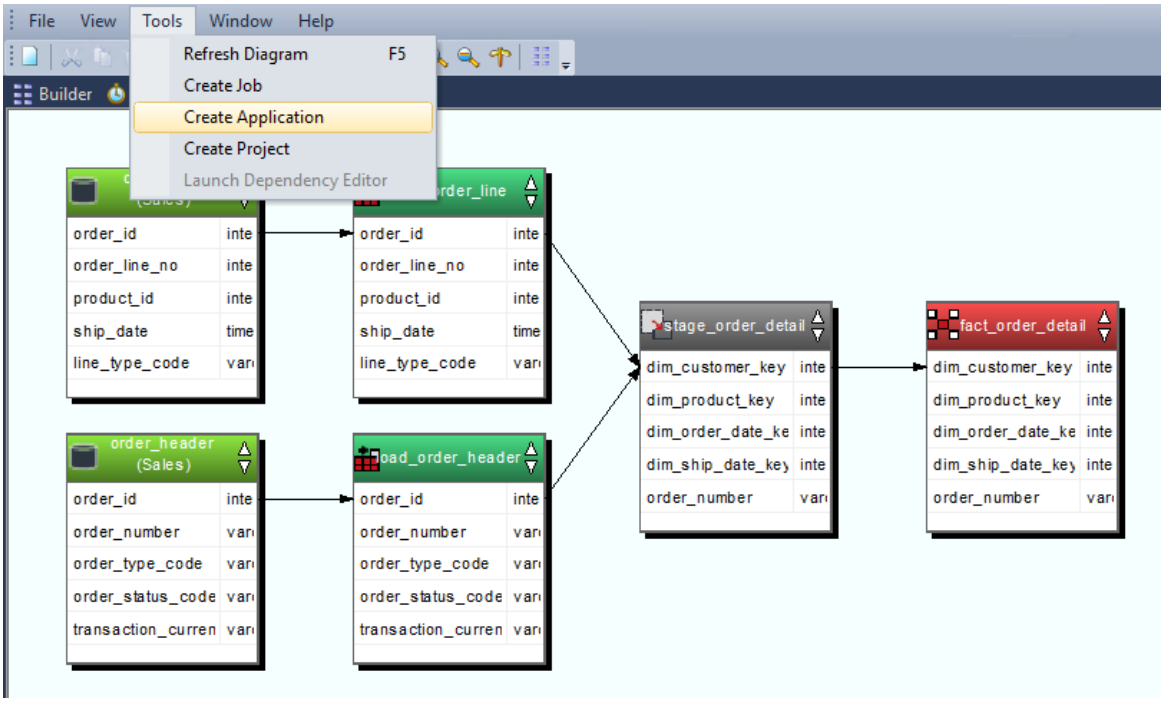


CREATING AN APPLICATION FROM A DIAGRAM

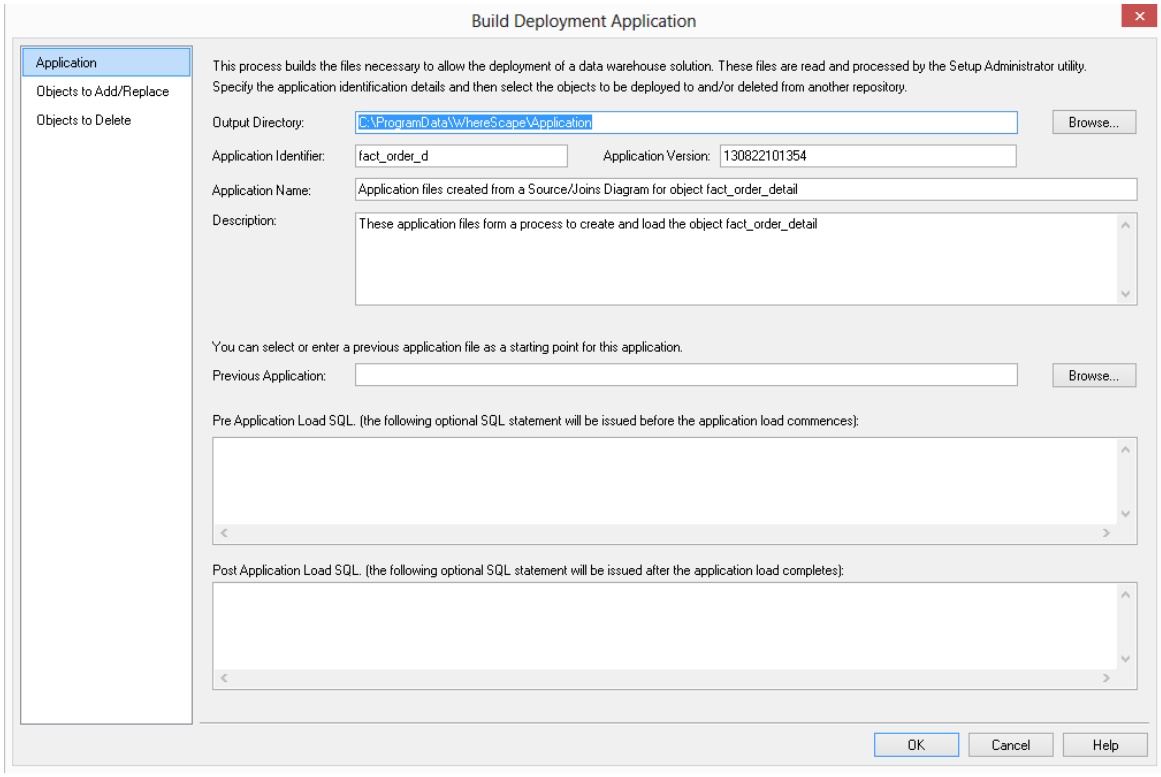
An application can be created from a **Source** diagram or a **Joins** diagram.

To Create an Application from a Diagram

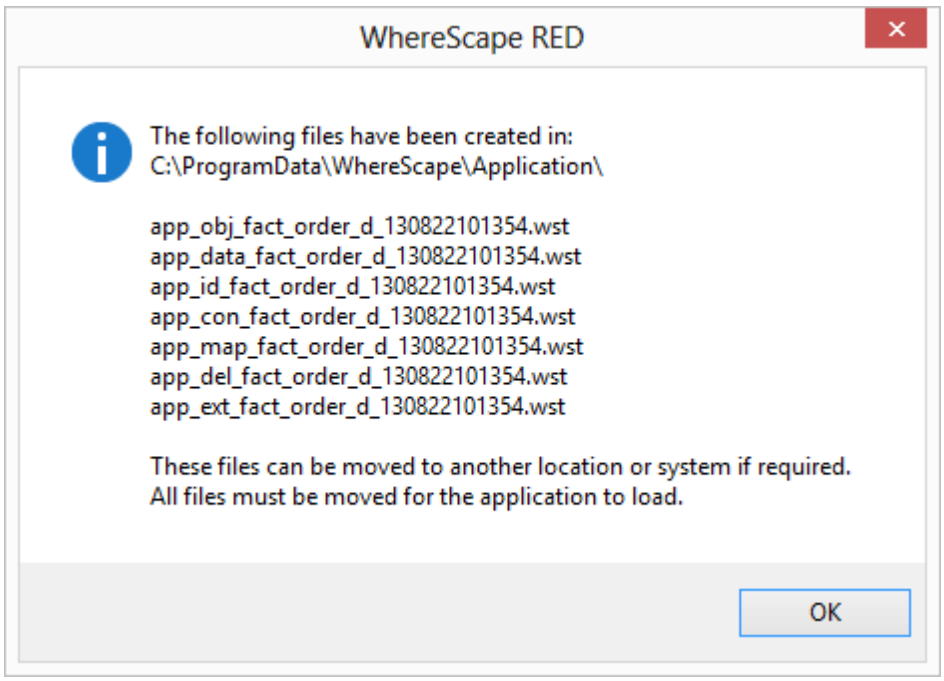
1 Once the diagram is displayed, select **Tools/Create Application**.



2 Edit the application as required and click **OK**.



3 A dialog will display, confirming the creation of the application files. Click **OK**.



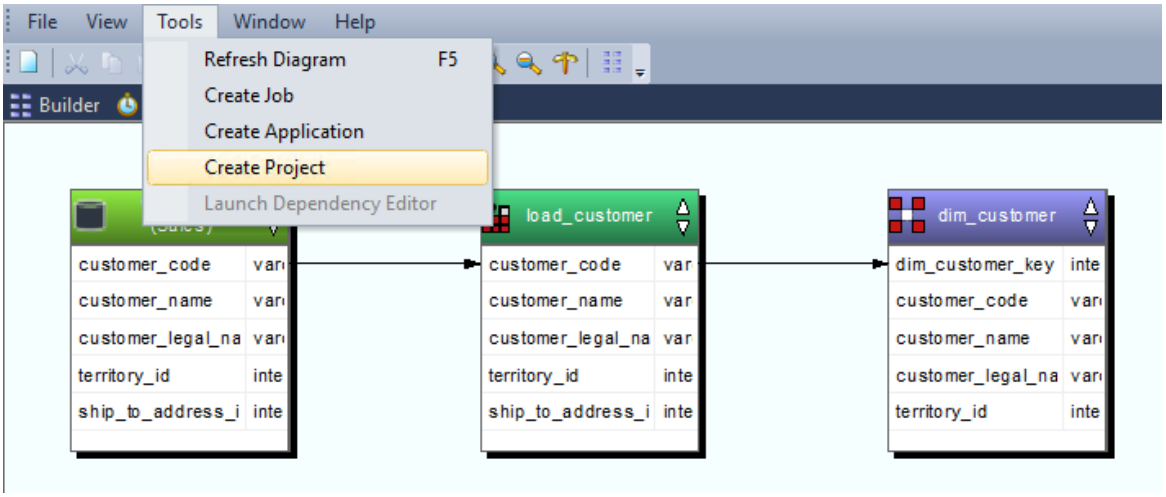
Note: Creating an application from a diagram will save the objects in the diagram and the associated objects, including indexes.

CREATING A PROJECT FROM A DIAGRAM

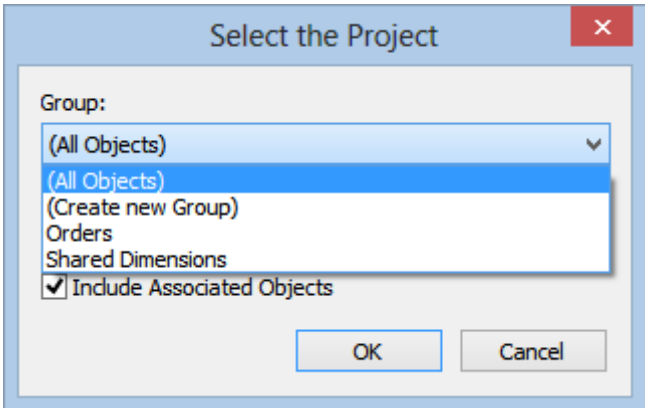
A project can be created from a **Source** diagram or a **Joins** diagram.

To Create a Project from a Diagram

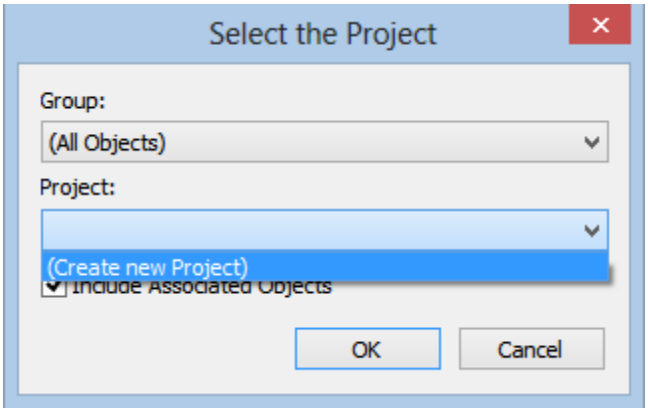
1 Once the diagram is displayed, select **Tools/Create Project**.



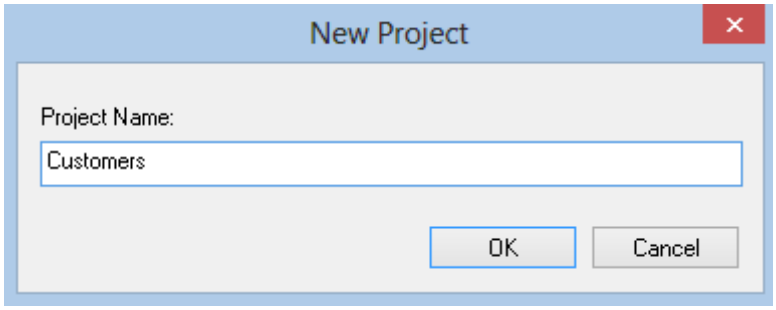
2 Select an existing group or create a new group.



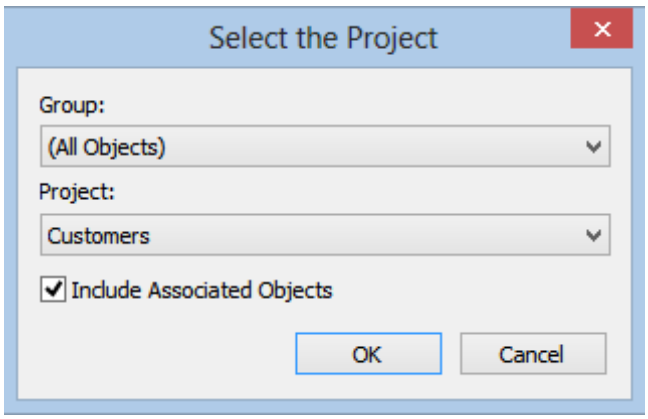
3 Select to **Create new Project**.



4 Enter the name of the new project and click **OK**.



5 Click **OK**.



The objects in the diagram will be moved into the selected project. If the **Include Associated Objects** checkbox is selected, this will include all associated procedures, scripts and indexes. The default for this checkbox is on.

QUERY TOOL ACCESS

Where the business definition has been set in the properties of either a fact or dimension column WhereScape RED will attempt to record this information in the database. For the Oracle database, this business definition is stored as a column comment. Many end user query tools can make use of these Oracle column comments. They can also be examined via the Oracle database system views all_col_comments and user_col_comments. For the SQL Server database, the business definition is also stored as a column comment via the sp_addextendedproperty procedure with a property name of **Comment**.

In addition, a number of metadata views exist to make the task of connecting an end user query tool to the data warehouse simpler. Many end user tools can make use of external data to populate their end-user-layer (eul) definitions. Using these views end user query tools can utilize WhereScape RED's business metadata. The views supplied are:

Dimension tables and columns

The view **ws_admin_v_dim_col** provides a listing of all columns in all dimensions that have the 'Eul visible' flag set. Included in the information provided is the business definition and description of the column as defined within WhereScape RED.

Fact tables and columns

The view **ws_admin_v_fact_col** provides a listing of all columns in all fact tables that have the 'Eul visible' flag set. Included in the information provided is the business definition and description of the column as defined within WhereScape RED.

Fact - Dimension Joins

The view **ws_admin_v_fact_join** provides a listing of all fact tables and all the dimensions that each fact table joins to. Both the fact table dimension key column and the dimension key column are included in the view.

WhereScape®

CHAPTER 26
REPORTS

WhereScape RED includes reports for analyzing columns, tables, procedures, indexes,objects and jobs.

When these reports are run, the results are displayed in a separate tab in the bottom pane of the RED screen.

The following sections describe the purpose, parameters and results for each report.

IN THIS CHAPTER

DIMENSION-FACT MATRIX

This report shows dimension tables used by each fact and aggregate table in the metadata as a matrix.

Objects Included

The following WhereScape RED object types are included in this report:

- Dimension Tables
- Fact Tables
- Aggregate Tables

Parameters

There are no parameters for this report.

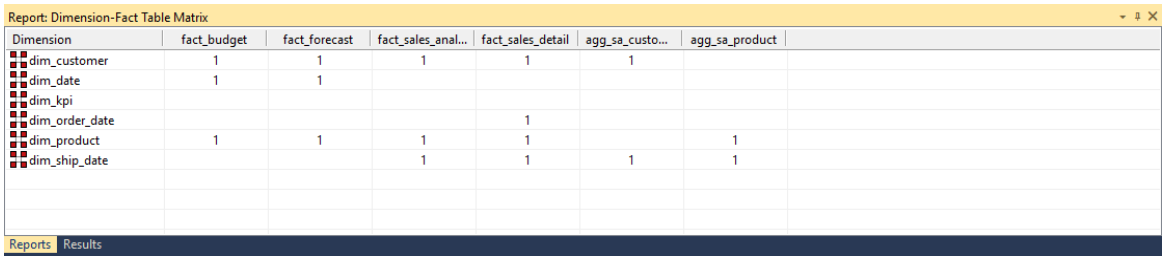
Results

The results of this report are displayed as a list of refreshed objects in the metadata repository with the following columns:

- Dimensions *(the dimension name)*
- Fact/Aggregate Table 1
- Fact/Aggregate Table 2
- ...
- Fact/Aggregate Table *n*

The cells in the crosstab have a 1 to indicate the dimension is used by the fact or aggregate table and blank otherwise. The result set is not sortable.

Report Example



Sending Results to Microsoft Excel

Right-click on the result set and click **Output to File** to send the results of this report to Microsoft Excel.

OLAP DIMENSION-CUBE MATRIX

This report shows the relationships between cube measure groups and OLAP dimensions in the metadata as a matrix.

Objects Included

The following WhereScape RED object types are included in this report:

- OLAP Dimensions
- OLAP Measure Groups

Parameters

There are no parameters for this report.

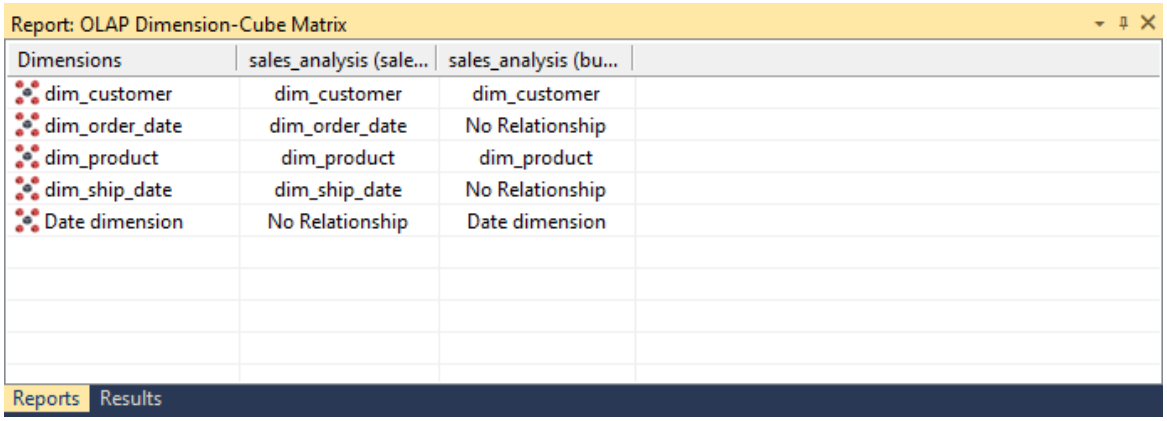
Results

The results of this report are displayed as a list of refreshed objects in the metadata repository with the following columns:

- Dimensions *(the dimension name)*
- Measure Group 1
- Measure Group 2
- ...
- Measure Group *n*

The cells in the crosstab have a value to indicate the relationship else 'No Relationship' if no relationship exists between the Measure Group and the OLAP Dimension. The result set is not sortable.

Report Example



Sending Results to Microsoft Excel

Right-click on the result set and click **Output to File** to send the results of this report to Microsoft Excel.

DIMENSION VIEWS OF A SPECIFIED DIMENSION

This report shows dimension views built on a specified dimension table.

Objects Included

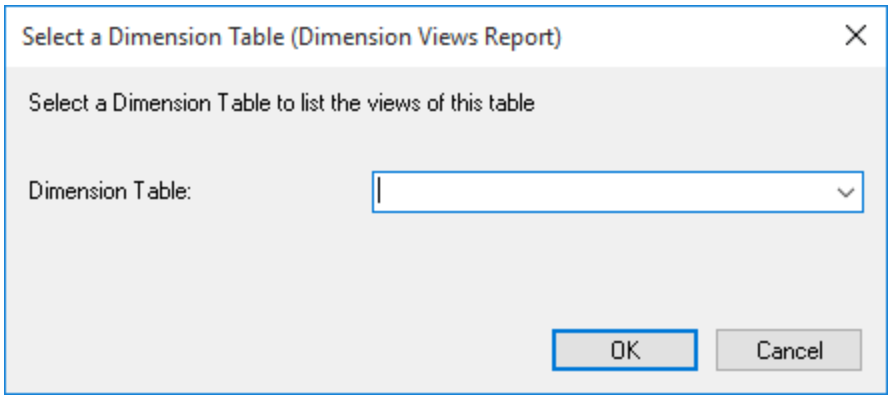
The following WhereScape RED object types are included in this report:

- Dimensions (Dimension View table type only)

Parameters

This report has one parameter:

- Dimension table name



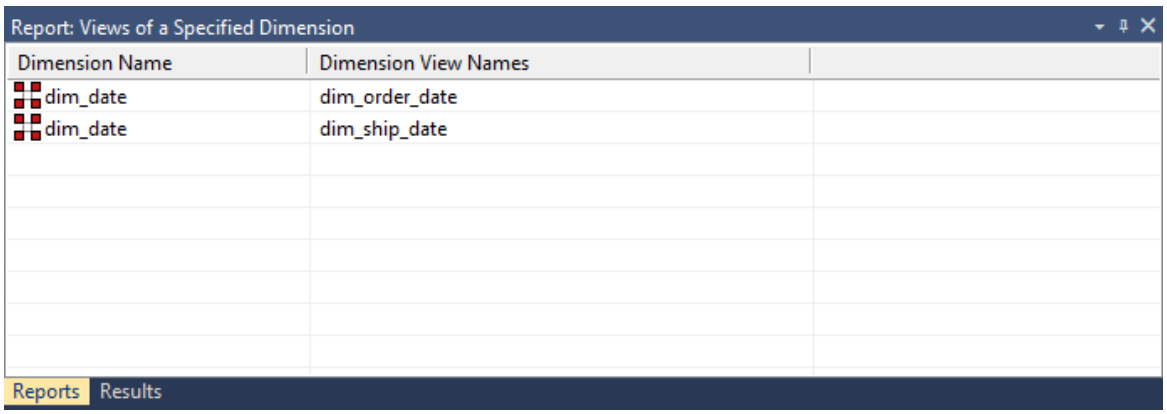
Results

The results of this report are displayed as a list of objects in the metadata repository with the following columns:

- Dimension Name *(the name of the dimension table)*
- Dimension View Names

The result set is sortable by clicking on the appropriate column heading.

Report Example



Sending Results to Microsoft Excel

Right-click on the result set and click **Output to File** to send the results of this report to Microsoft Excel.

COLUMN REPORTS

There are five reports for analyzing Columns:

- Columns Without Comments
- All Column Transformations
- Re-Usable Column Transformations
- Column Track-Back
- Column Track-Forward

COLUMNS WITHOUT COMMENTS

This report shows user facing **table** objects columns in the metadata that don't have descriptions.

Objects Included

The following WhereScape RED object types are included in this report:

- Dimension Tables and Views
- Fact Tables
- Aggregate Tables
- Hubs, Satellites and Links

Parameters

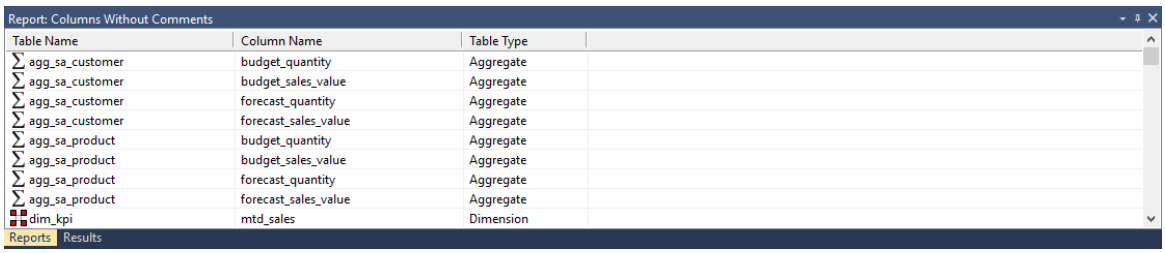
There are no parameters for this report.

Results

The results of this report are displayed as a list of objects and columns in the metadata missing comments with the following columns:

- Table name *(the name of the table)*
- Column Name
- Table type

Report Example



Sending Results to Microsoft Excel

Right-click on the result set and click **Output to File** to send the results of this report to Microsoft Excel.

ALL COLUMN TRANSFORMATIONS

This report shows all columns that have a column transformation on them and the details of the transformation.

Objects Included

The following WhereScape RED object types are included in this report:

- Load Tables
- Stage Tables
- Dimension Tables and Views
- Fact Tables
- Aggregate Tables
- Exports
- Hubs, Satellites and Links

Parameters

There are no parameters for this report.

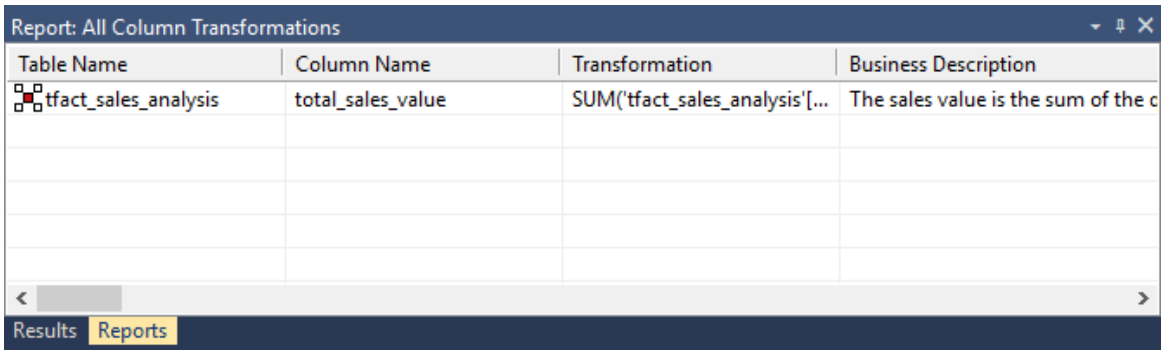
Results

The results of this report are displayed as a list of objects in the metadata repository with the following columns:

- Table name *(the name of the table)*
- Column name
- Transformation

The result set is sortable by clicking on the appropriate column heading.

Report Example



Sending Results to Microsoft Excel

Right-click on the result set and click **Output to File** to send the results of this report to Microsoft Excel.

RE-USABLE COLUMN TRANSFORMATIONS

This report shows all reusable transformations as defined via **Tools/Reusable Transformations**.

Parameters

There are no parameters for this report.

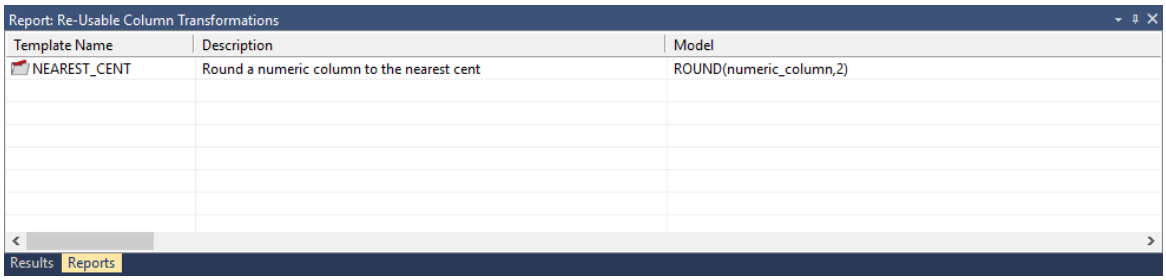
Results

The results of this report are displayed as a list of Re-Usable transformations with the following columns:

- Template Name
- Description

The result set is not sortable.

Report Example



Sending Results to Microsoft Excel

Right-click on the result set and click **Output to File** to send the results of this report to Microsoft Excel.

COLUMN TRACK-BACK

This report lists the origins of the selected objects.

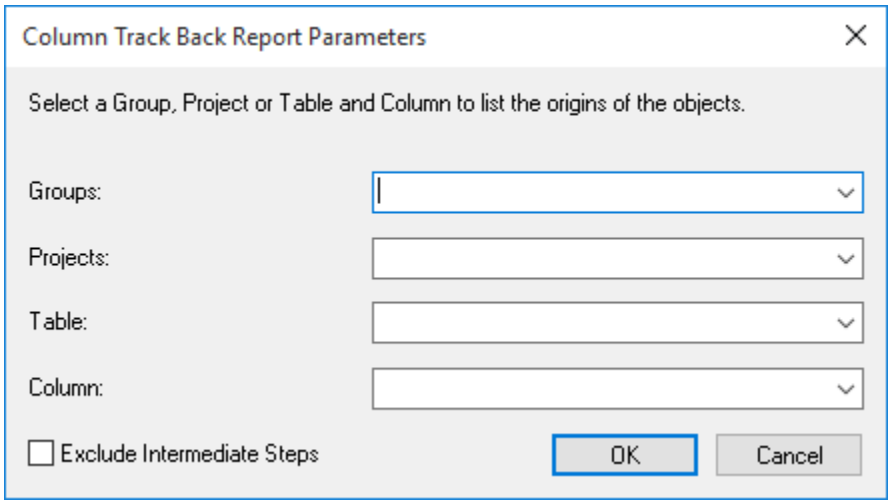
Objects Included

The following WhereScape RED object types are included in this report:

- Load Tables
- Stage Tables
- Dimension Tables and Views
- Fact Tables
- Aggregate Tables
- Exports
- Hubs, Satellites and Links

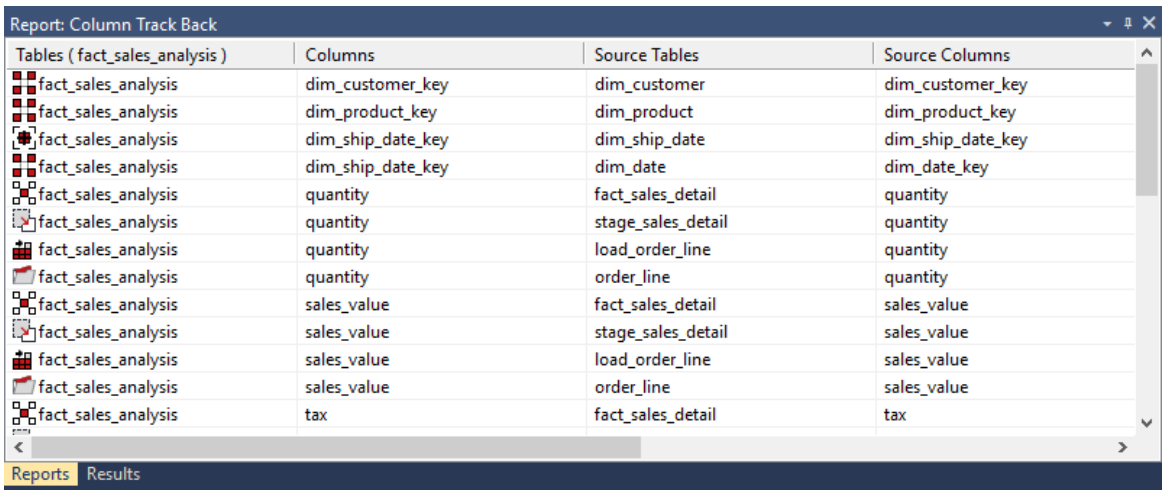
Parameters

- Groups
- Projects
- Table
- Column

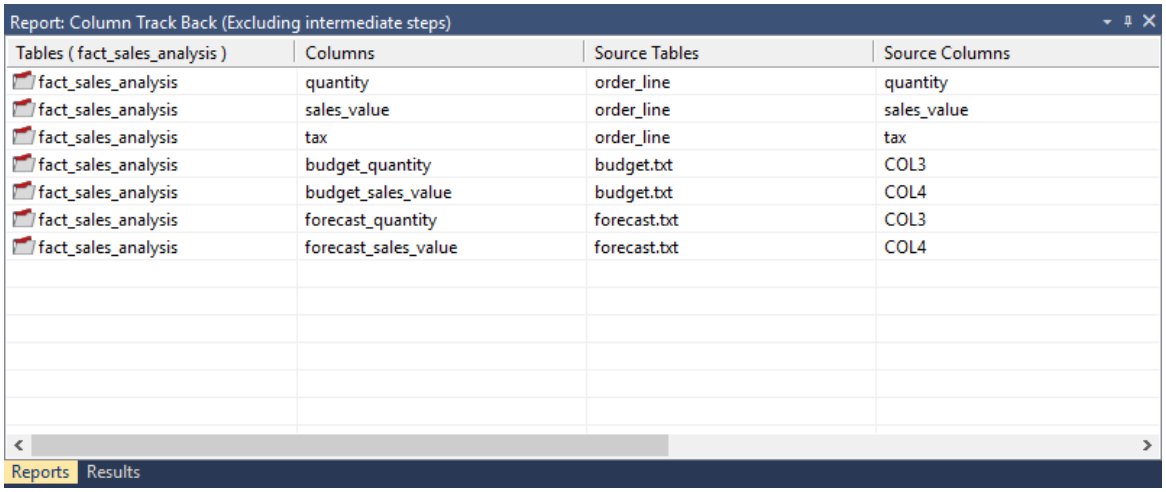


Results

If you left the **Exclude Intermediate Steps** checkbox **unchecked**, then the results screen will be as follows, showing the line of origins for the selected tables:



If however, you selected **Exclude Intermediate Steps**, then the results screen will be as follows, showing only the original source table for the selected tables:



The results of this report are displayed as a list of source tables and columns, the order of the result set determining the immediate lineage. The report includes the following columns:

- Tables *(the name of a selected table)*
- Columns *(the name of a selected column)*
- Source Tables
- Source Columns

The result set is **NOT** sortable, as the order of the result set determines the immediate lineage.

Sending Results to Microsoft Excel

Right-click on the result set and click **Output to File** to send the results of this report to Microsoft Excel.

COLUMN TRACK-FORWARD

This report lists the columns derived from the selected objects.

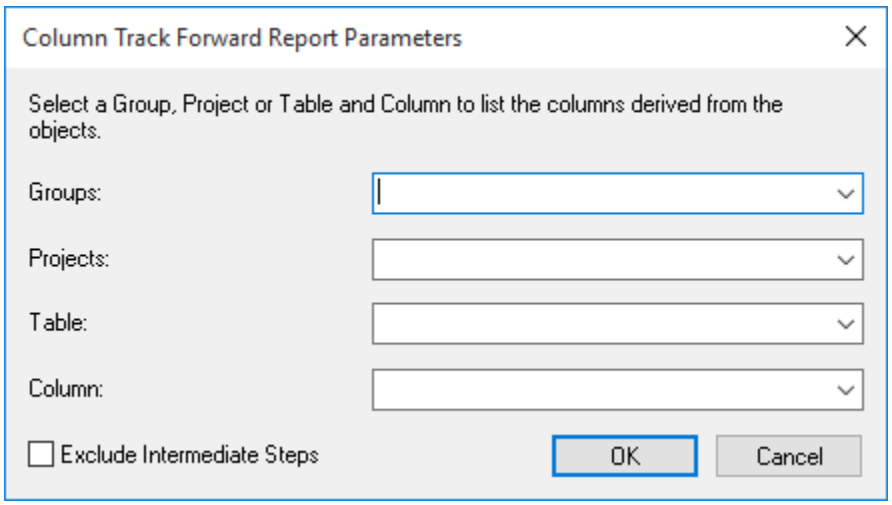
Objects Included

The following WhereScape RED object types are included in this report:

- Load Tables
- Stage Tables
- Dimension Tables and Views
- Fact Tables
- Aggregate Tables
- Exports
- Hubs, Satellites and Links

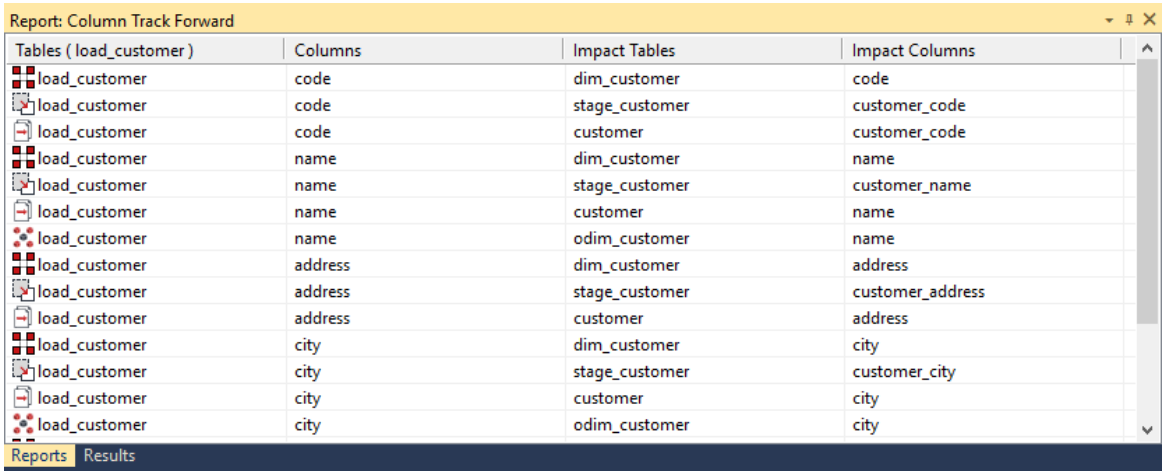
Parameters

- Groups
- Projects
- Table
- Column

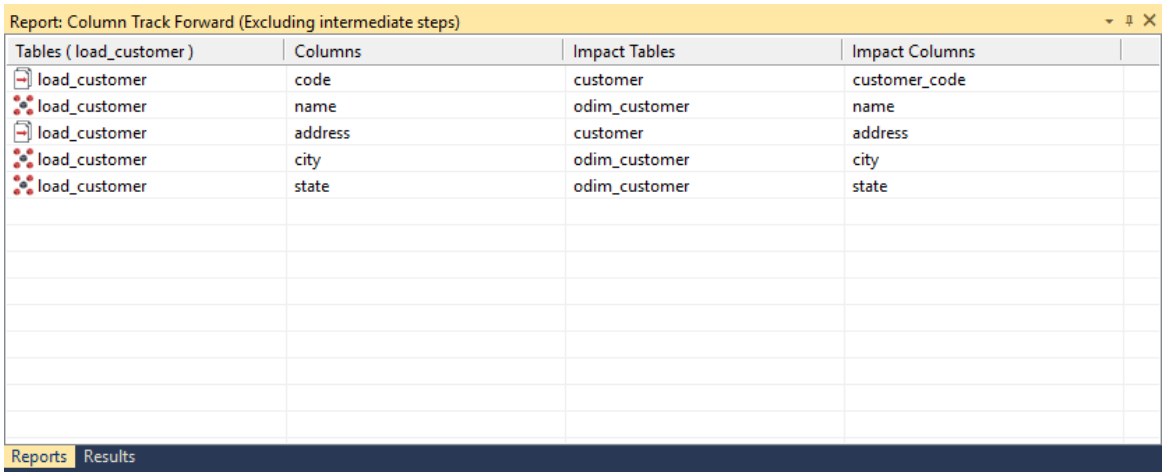


Results

If you left the **Exclude Intermediate Steps** checkbox **unchecked**, then the results screen will be as follows, showing the impacted tables for the selected tables:



If however, you **selected Exclude Intermediate Steps**, then the results screen will be as follows, showing only the final impacted table for the selected tables:



The results of this report are displayed as a list of impacted tables and columns, the order of the result set determining the immediate impact. The report includes the following columns:

- Tables *(the names of selected tables)*
- Columns *(the names of selected columns)*
- Impact Tables
- Impact Columns

The result set is **NOT** sortable, as the order of the result set determines the immediate lineage.

Sending Results to Microsoft Excel

Right-click on the result set and click **Output to File** to send the results of this report to Microsoft Excel.

TABLE REPORTS

There are five reports for analyzing Tables:

- Tables Without Comments
- Load Tables by Connection
- Export Objects by Connection
- Records that Failed a Dimension Join
- External source Table/Files

TABLES WITHOUT COMMENTS

This report shows user facing **table** objects in the metadata that don't have descriptions.

Objects Included

The following WhereScape RED object types are included in this report:

- Dimension Tables and Views
- Fact Tables
- Aggregate Tables

Parameters

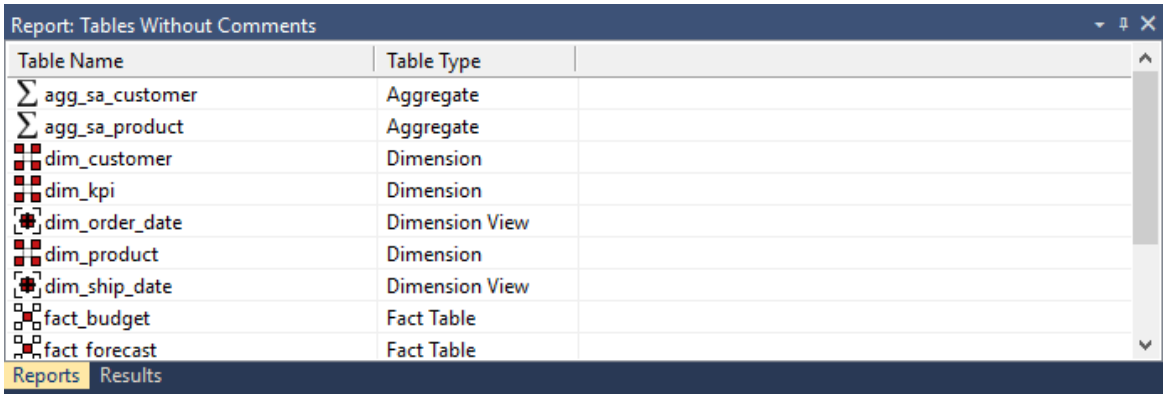
There are no parameters for this report.

Results

The results of this report are displayed as a list of objects in the metadata missing comments with the following columns:

- Table name *(the name of the table)*
- Table type

Report Example



Sending Results to Microsoft Excel

Right-click on the result set and click **Output to File** to send the results of this report to Microsoft Excel.

LOAD TABLES BY CONNECTION

This report shows load tables in the metadata repository with their Connection and Source schema or database.

Objects Included

The following WhereScape RED object types are included in this report:

- Load Tables

Parameters

There are no parameters for this report.

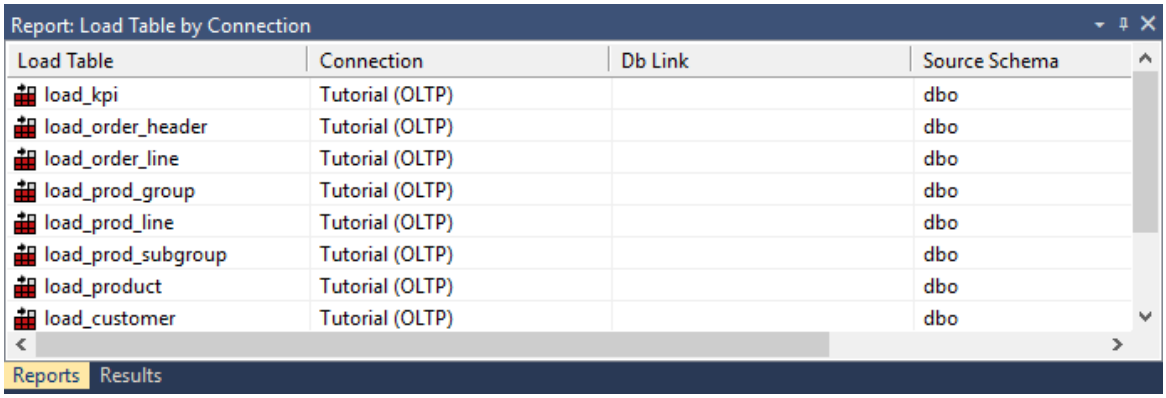
Results

The results of this report are displayed as a list of objects in the metadata repository with the following columns:

- Load table *(the name of the load table)*
- Connections
- Source Schema (*the name the database or schema the load table is source from - blank for files)*

The result set is sortable by clicking on the appropriate column heading.

Report Example



Sending Results to Microsoft Excel

Right-click on the result set and click **Output to File** to send the results of this report to Microsoft Excel.

EXPORT OBJECTS BY CONNECTION

This report shows export tables in the metadata repository with their Connection and Source schema or database.

Objects Included

The following WhereScape RED object types are included in this report:

- Export Tables

Parameters

There are no parameters for this report.

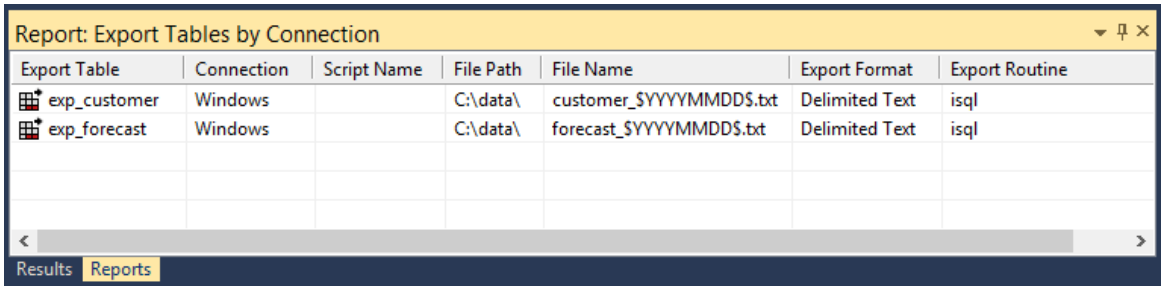
Results

The results of this report are displayed as a list of objects in the metadata repository with the following columns:

- Export Table *(the name of the export table)*
- Connection
- Script Name
- File Path
- File Name
- Export Format
- Export Routine

The result set is sortable by clicking on the appropriate column heading.

Report Example



Sending Results to Microsoft Excel

Right-click on the result set and click **Output to File** to send the results of this report to Microsoft Excel.

RECORDS THAT FAILED A DIMENSION JOIN

This report shows the dimension business key(s) that could not be found in a specified dimension when a specified staging table was last updated. This report will show null values, blank values and business keys not found in the dimension.

Objects Included

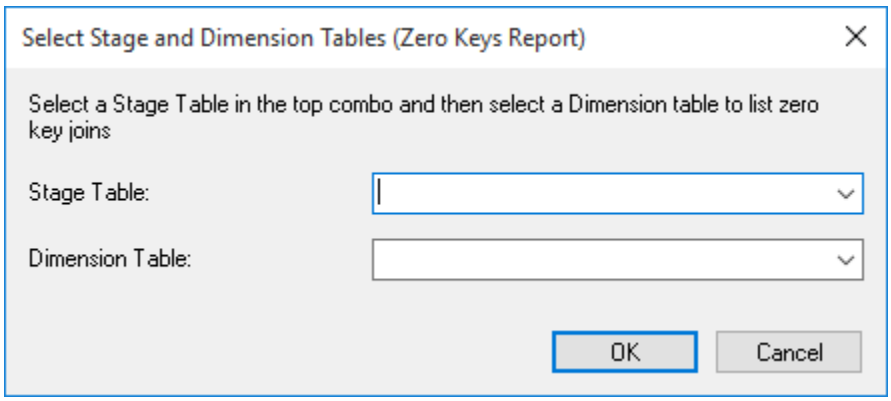
The following WhereScape RED object types are included in this report:

- Stage Tables

Parameters

This report requires two parameters to be specified:

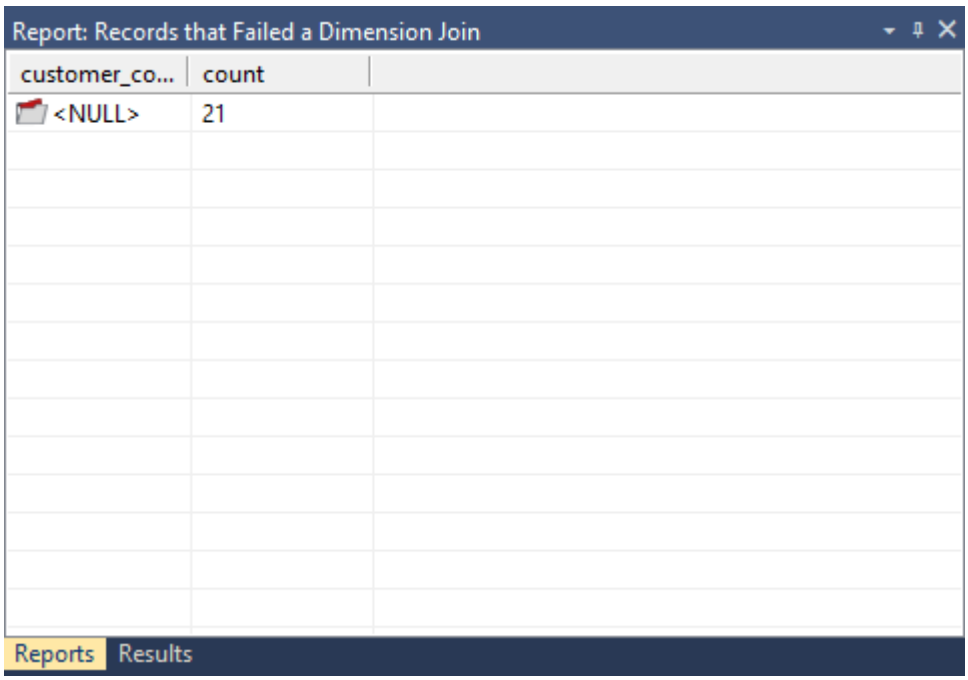
- Stage Table Name *(the staging table to be checked)*
- Dimension Table Name *(the dimension table of the dimension key in the staging table selected first)*



Results

The report contains a list of values not found in the dimension and a count for each value. Each column is sortable by clicking on the appropriate column heading.

Report Example



Sending Results to Microsoft Excel

Right-click on the result set and click **Output to File** to send the results of this report to Microsoft Excel.

EXTERNAL SOURCE TABLE/FILES

This report shows external sources for load tables in the metadata repository.

Objects Included

The following WhereScape RED object types are included in this report:

- Load Tables

Parameters

There are no parameters for this report.

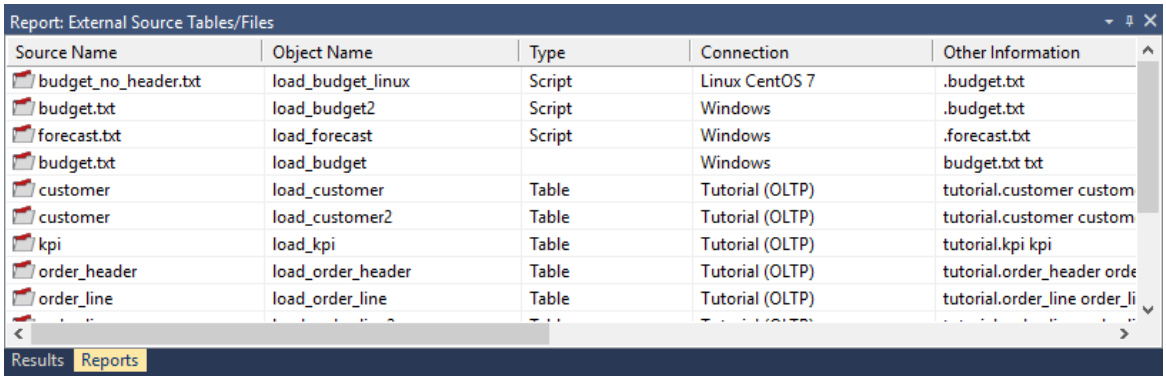
Results

The results of this report are displayed as a list of objects in the metadata repository with the following columns:

- Source name *(the name of the object's source)*
- Object name *(the name of the object)*
- Type *(the type of object the source is: Table or File)*
- Connection
- Other information *(for tables, the source schema/database.source table; for files, the file name)*

The result set is sortable by clicking on the appropriate column heading.

Report Example



Sending Results to Microsoft Excel

Right-click on the result set and click **Output to File** to send the results of this report to Microsoft Excel.

PROCEDURE REPORTS

There are two reports for analyzing Procedures:

- Modified Procedures
- Custom Procedures

MODIFIED PROCEDURES

This report shows **modified** procedures in the metadata repository with their creation and modification dates.

Objects Included

The following WhereScape RED object types are included in this report:

- Procedures

Parameters

There are no parameters for this report.

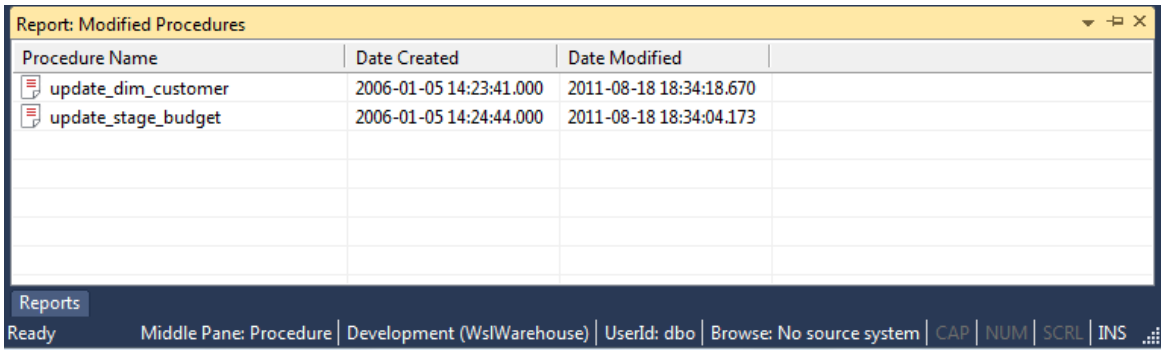
Results

The results of this report are displayed as a list of modified procedures in the metadata repository with the following columns:

- Procedure Name *(the name of the object)*
- Dated Created
- Date Modified *(last modification date)*

The result set is sortable by clicking on the appropriate column heading.

Report Example



Sending Results to Microsoft Excel

Right-click on the result set and click **Output to File** to send the results of this report to Microsoft Excel.

CUSTOM PROCEDURES

This report shows **custom** procedures in the metadata repository with their creation and modification dates.

Note: Custom procedures are procedures attached to any table object as a **Custom Procedure**.

Objects Included

The following WhereScape RED object types are included in this report:

- Procedures

Parameters

There are no parameters for this report.

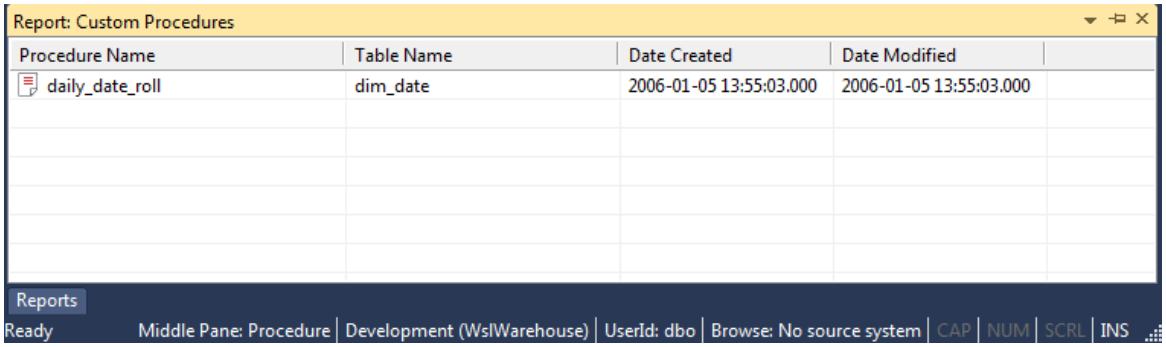
Results

The results of this report are displayed as a list of custom procedures in the metadata repository with the following columns:

- Procedure Name *(the name of the object)*
- Table Name *(the table object the procedure is attached to)*
- Dated Created
- Date Modified *(last modification date)*

The result set is sortable by clicking on the appropriate column heading.

Report Example



Sending Results to Microsoft Excel

Right-click on the result set and click **Output to File** to send the results of this report to Microsoft Excel.

INDEX REPORTS

There is one report for analyzing Indexes:

- Modified Indexes

MODIFIED INDEXES

This report shows indexes in the metadata repository with their creation and modification dates.

Objects Included

The following WhereScape RED object types are included in this report:

- Indexes

Parameters

There are no parameters for this report.

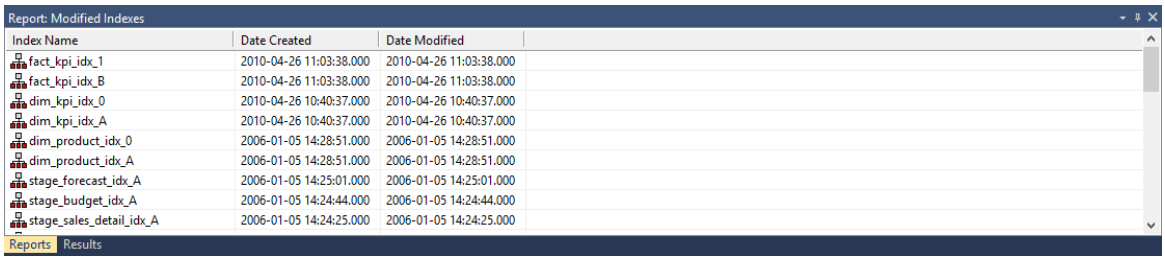
Results

The results of this report are displayed as a list of indexes in the metadata repository with the following columns:

- Index Name *(the name of the index)*
- Dated Created
- Date Modified *(last modification date)*

The result set is sortable by clicking on the appropriate column heading.

Report Example



Sending Results to Microsoft Excel

Right-click on the result set and click **Output to File** to send the results of this report to Microsoft Excel.

OBJECT REPORTS

There are four reports for analyzing Objects:

- Objects-Project Matrix
- Modified Objects (excluding indexes)
- Objects Checked-out
- Loaded or Imported Objects

OBJECTS-PROJECT MATRIX

This report shows the objects that exist in projects other than **All Objects**.

Objects Included

All object types are included in this report.

Parameters

There are no parameters for this report.

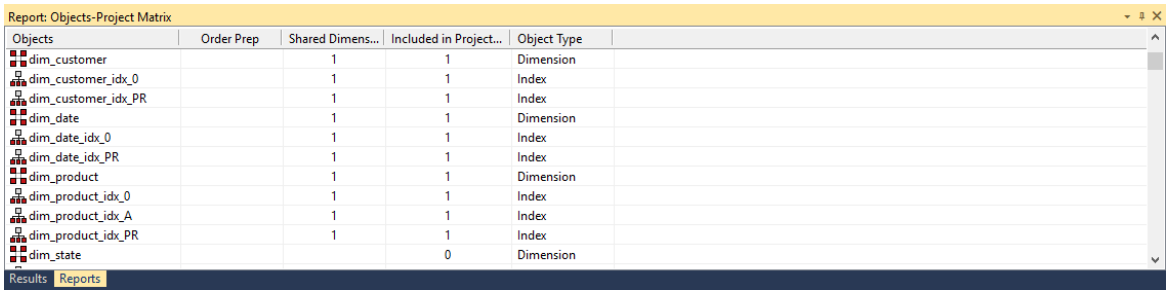
Results

The results of this report are displayed as a list of objects in the metadata repository that are in one or more projects (other than All Objects) with the following columns as a grid:

- Objects *(the name of the object)*
- Project Name 1 *(heading is the name of the first project, value is a 1 to indicate the object is in this project, blank otherwise)*
- Project Name 2 *(heading is the name of the second project, value is a 1 to indicate the object is in this project, blank otherwise)*
- ...
- Project Name n *(heading is the name of the nth project, value is a 1 to indicate the object is in this project, blank otherwise)*

The result set is sortable by clicking on the appropriate column heading.

Report Example



Sending Results to Microsoft Excel

Right-click on the result set and click **Output to File** to send the results of this report to Microsoft Excel.

MODIFIED OBJECTS (EXCLUDING INDEXES)

This report shows objects in the metadata repository with their creation and modification dates and indicates if they have been modified.

Objects Included

The following WhereScape RED object types are included in this report:

- Load Tables
- Stage Tables
- Dimension Tables and Views
- Fact Tables
- Aggregate Tables
- Exports
- Procedures
- Host Scripts

Parameters

There are no parameters for this report.

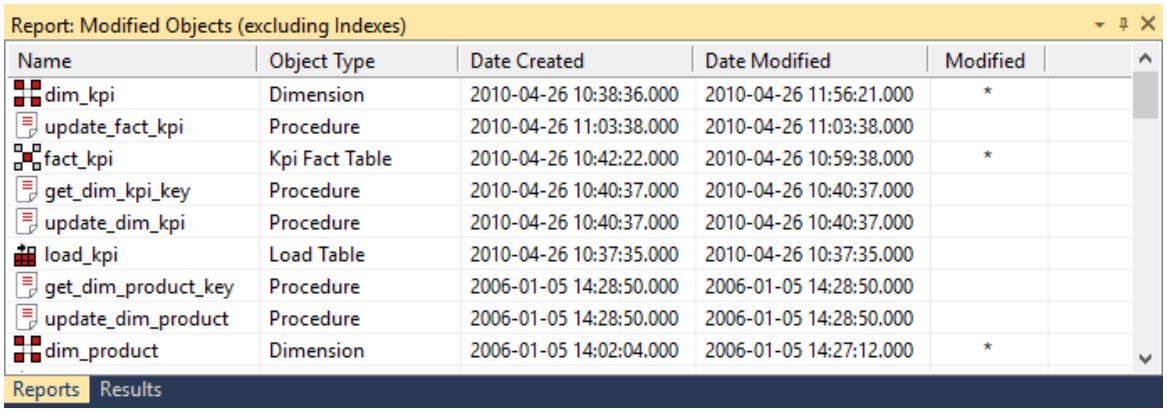
Results

The results of this report are displayed as a list of objects in the metadata repository with the following columns:

- Name *(the name of the object)*
- Object Type
- Dated Created
- Date Modified *(last modification date)*
- Modified *(a star for modified objects, blank for objects that have not been modified)*

The result set is sortable by clicking on the appropriate column heading.

Report Example



Sending Results to Microsoft Excel

Right-click on the result set and click **Output to File** to send the results of this report to Microsoft Excel.

OBJECTS CHECKED-OUT

This report lists all objects currently checked out.

Objects Included

All object types and data warehouse tables can be included in this report.

Parameters

There are no parameters for this report.

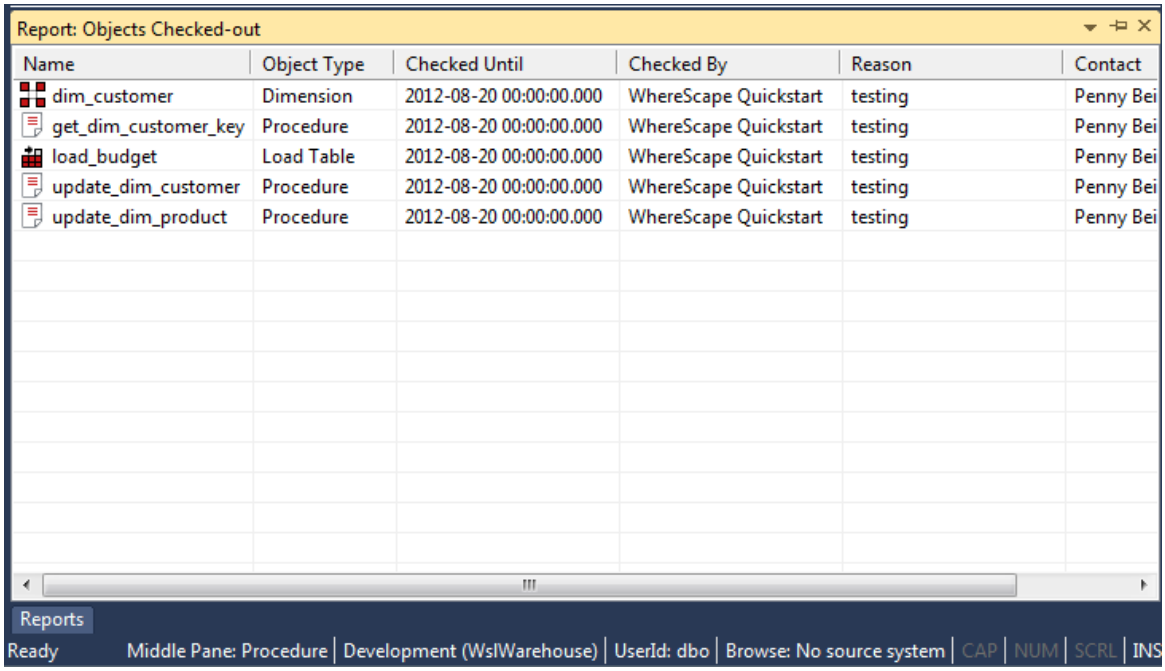
Results

The results of this report are displayed with the following columns:

- Name *(The name of the object)*
- Object Type *(The type of the object, e.g. Fact, Dimension, etc)*
- Checked Until *(The date the object will be automatically checked back in)*
- Checked By *(The name of the WhereScape RED user who checked out the object)*
- Reason *(The reason provided for checking out the object)*
- Contact *(The contact details provided when the object was checked out)*

The result set is sortable by clicking on the appropriate column heading.

Report Example



Sending Results to Microsoft Excel

Right-click on the result set and click **Output to File** to send the results of this report to Microsoft Excel.

LOADED OR IMPORTED OBJECTS

This report shows objects in the metadata repository that have been refreshed or imported from another repository.

Objects Included

The following WhereScape RED object types are included in this report:

- Load Tables
- Stage Tables
- Dimension Tables and Views
- Fact Tables
- Aggregate Tables
- Join Indexes
- Views
- Cubes
- Exports
- Procedures
- Host Scripts

Note: Indexes are not included.

Parameters

There are no parameters for this report.

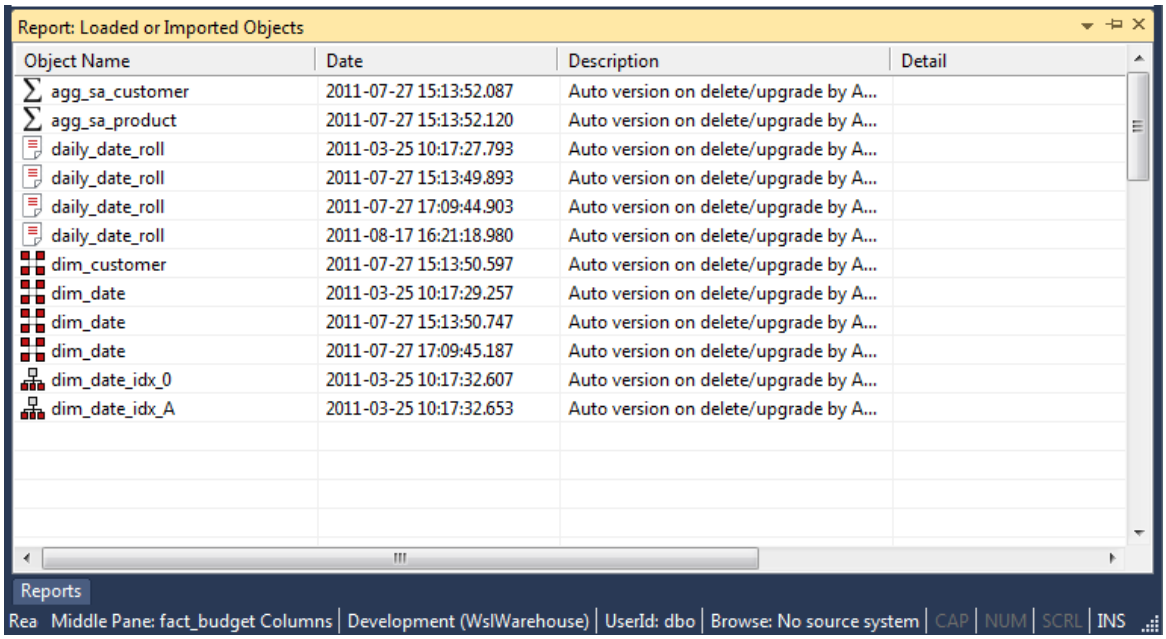
Results

The results of this report are displayed as a list of refreshed objects in the metadata repository with the following columns:

- Object Name
- Date *(of last refresh or import)*
- Description *(the kind of import or refresh)*
- Detail *(not currently used)*

The result set is sortable by clicking on the appropriate column heading.

Report Example



Sending Results to Microsoft Excel

Right-click on the result set and click **Output to File** to send the results of this report to Microsoft Excel.

JOB REPORTS

There are three reports for analyzing Jobs:

- Object-Job Matrix
- Jobs with an Object
- Tasks of a Job

OBJECT-JOB MATRIX

This report shows all jobs and objects as well as the object actions. Table and Cube objects not in any jobs are displayed with a job name of **No Job**.

Objects Included

All object types are included in this report.

Parameters

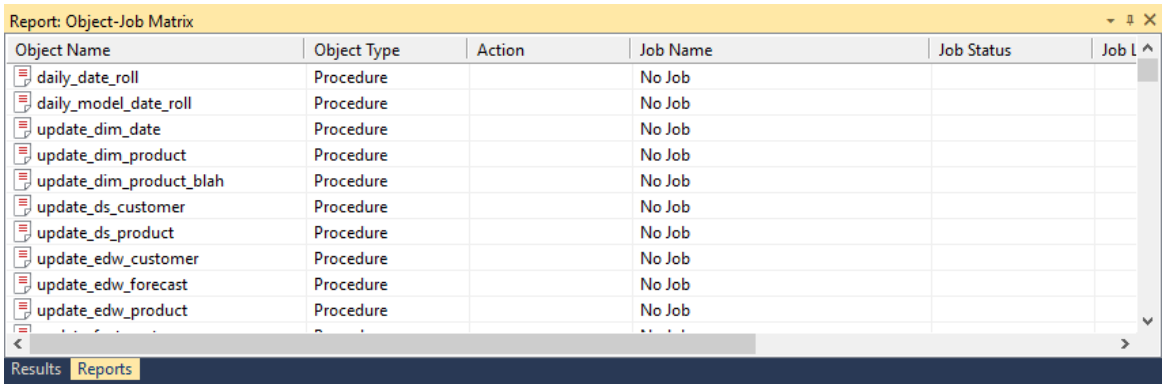
There are no parameters for this report.

Results

The results of this report are displayed as a list of jobs with the following columns:

- Table Name *(the name of the table or cube)*
- Action
- Job Name *(the name of the job)*
- Job Status
- Job Last Run
- Job Next Run

Report Example



Sending Results to Microsoft Excel

Right-click on the result set and click **Output to File** to send the results of this report to Microsoft Excel.

JOBS WITH AN OBJECT

This report shows all jobs containing a selected object and lists the job action.

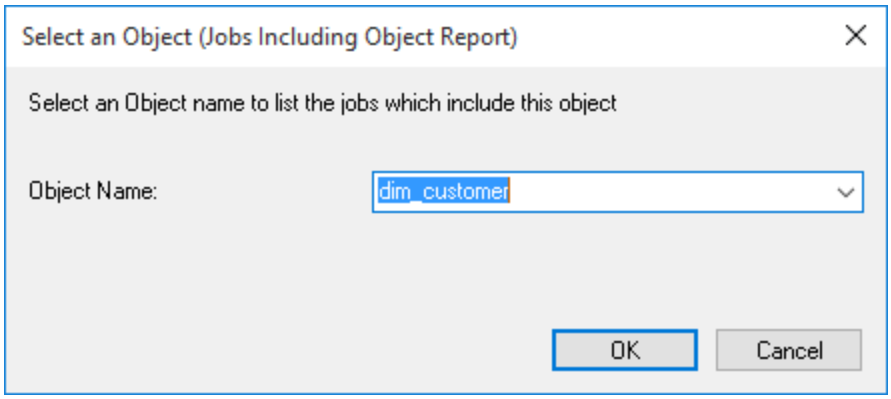
Objects Included

All object types are included in this report.

Parameters

This report has one parameter:

- Object Name

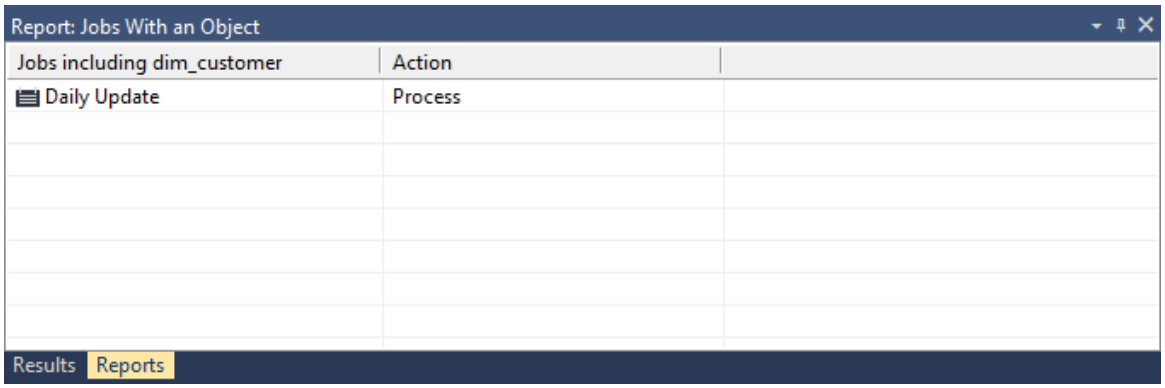


Results

The results of this report are displayed as a list of jobs with the following columns:

- Jobs including **object_name**
- Action

Report Example



Sending Results to Microsoft Excel

Right-click on the result set and click **Output to File** to send the results of this report to Microsoft Excel.

TASKS OF A JOB

This report shows all tasks for a selected job including dependencies.

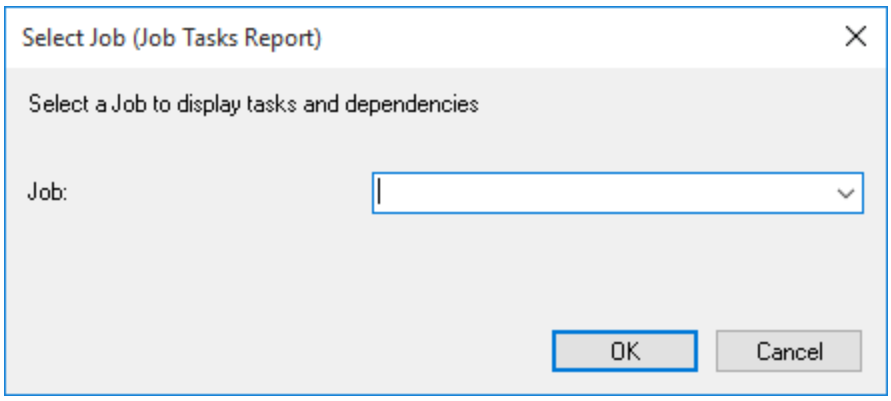
Objects Included

All object types are included in this report.

Parameters

This report has one parameter:

- Job Name

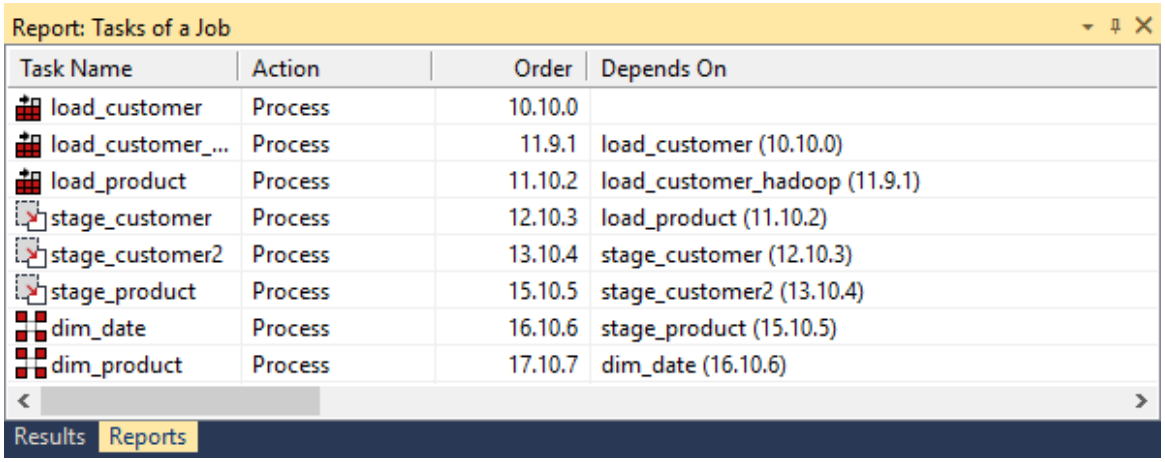


Results

The results of this report are displayed as a list of task dependencies with the following columns:

- Task name *(the table, Index, procedure or script name)*
- Action
- Order *(the order number as shown in the edit tasks dialog in the scheduler)*
- Depends On *(the task(s) and order number this task depends on)*

Report Example



Sending Results to Microsoft Excel

Right-click on the result set and click **Output to File** to send the results of this report to Microsoft Excel.

OPERATIONAL REPORTS

There are four Operational Reports:

- Object Performance History
- Job Performance History
- Task Performance History
- Fragmentation

OBJECT PERFORMANCE HISTORY

This report shows the audit trail from the scheduler logs for a selected object.

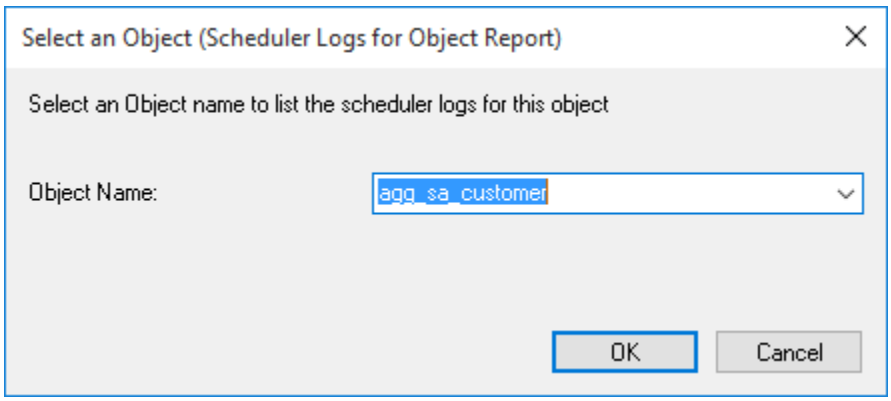
Objects Included

All object types are included in this report.

Parameters

This report has one parameter:

- Object Name

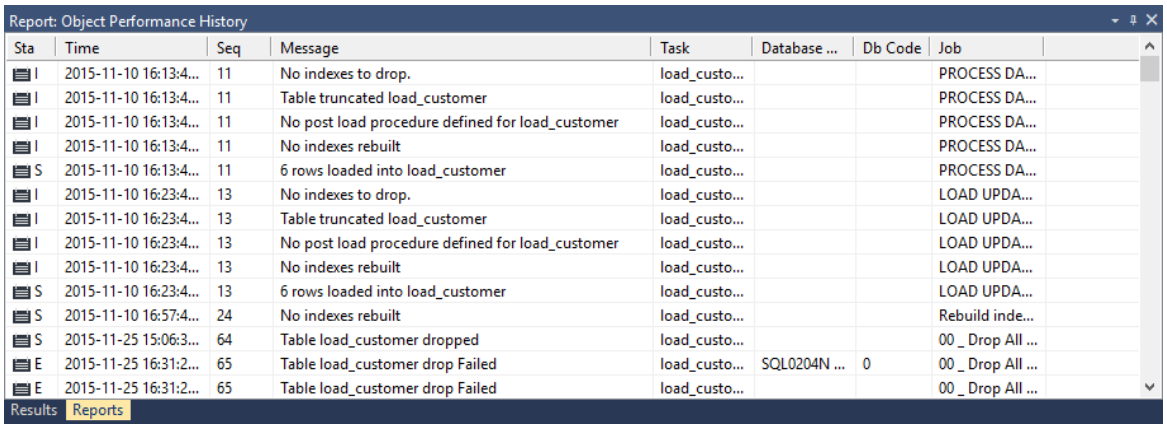


Results

The results of this report are displayed as a list of audit log entries with the following columns:

- Sta *(the type of audit log entry)*
- Time *(the date and time the audit log entry was written)*
- Seq *(the job sequence of the job writing the audit log entry)*
- Message *(the message in the Audit log)*
- Task *(the object name)*
- Job *(the name of the job that ran the task)*

Report Example



Sending Results to Microsoft Excel

Right-click on the result set and click **Output to File** to send the results of this report to Microsoft Excel.

JOB PERFORMANCE HISTORY

This report shows the performance (duration) over time of a selected job.

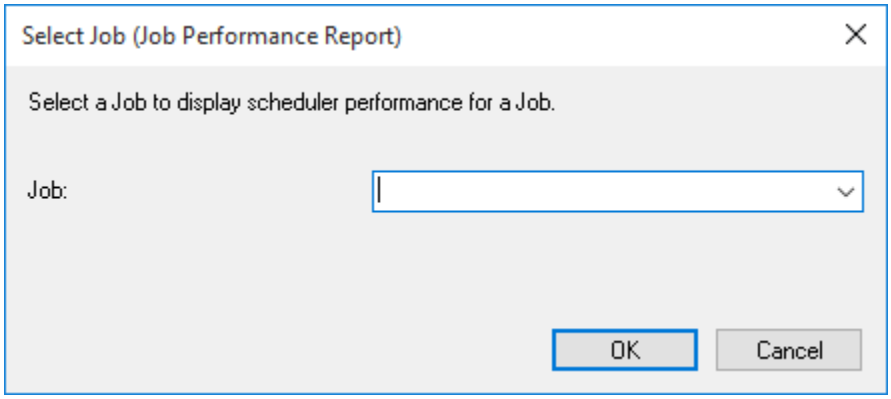
Objects Included

All object types are included in this report.

Parameters

This report has one parameter:

- Job Name

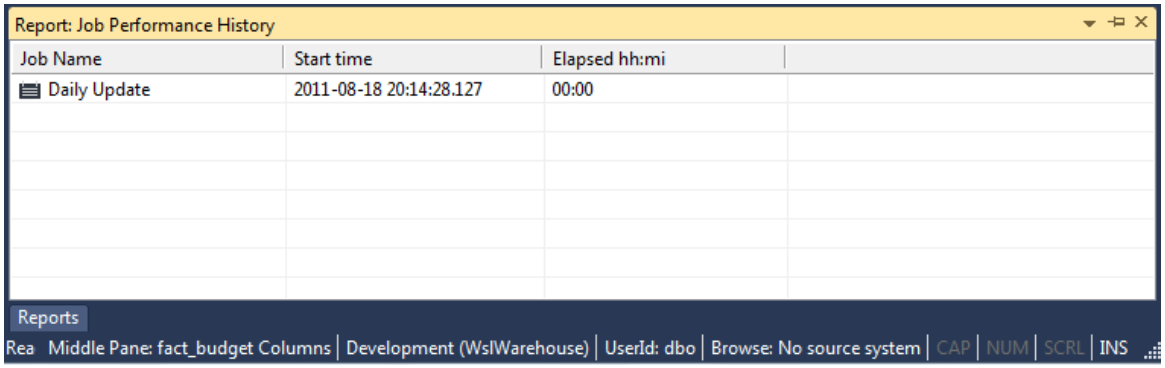

Select Job (Job Performance Report)
Select a Job to display scheduler performance for a Job.
Job:
OK
Cancel

Results

The results of this report are displayed as a list of job instances with the following columns:

- Job name *(the name of the job)*
- Start time *(the date and time the job started)*
- Elapsed hh:mi *(the duration of the job)*

Report Example



Sending Results to Microsoft Excel

Right-click on the result set and click **Output to File** to send the results of this report to Microsoft Excel.

TASK PERFORMANCE HISTORY

This report shows the performance (duration) over time of a selected task within a selected job.

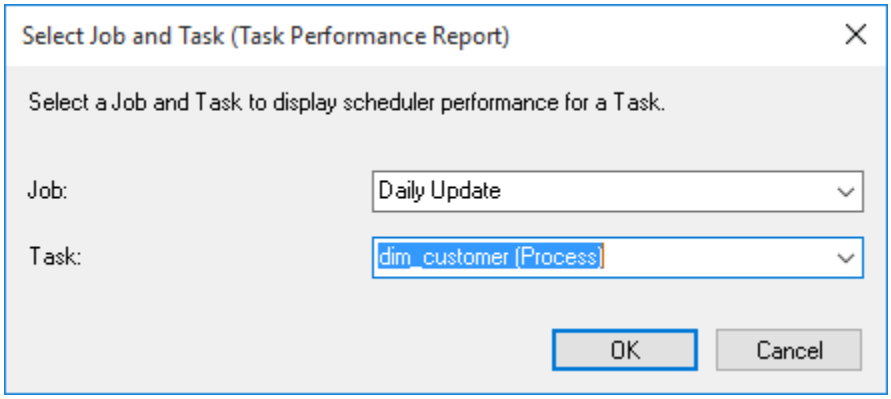
Objects Included

All object types are included in this report.

Parameters

This report has two parameters:

- Job Name
- Task Name (including action)

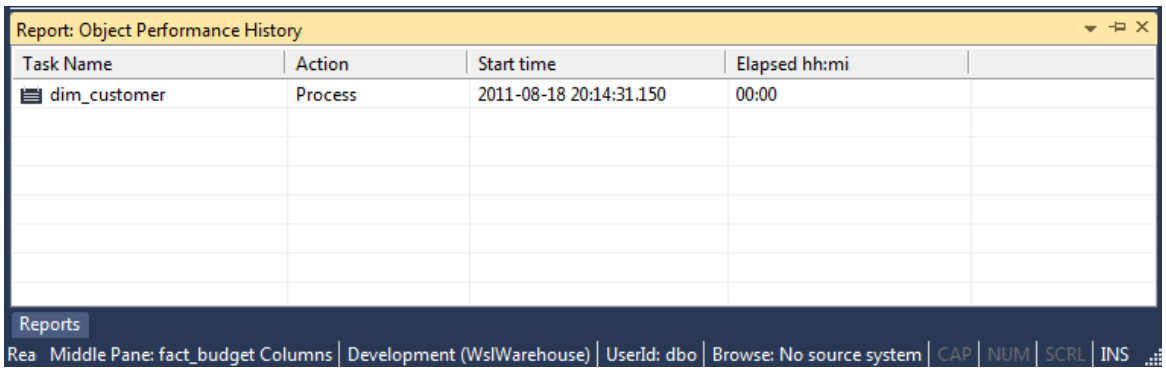


Results

The results of this report are displayed as a list of task instances for the selected job with the following columns:

- Task name *(the table, Index, procedure or script name)*
- Action
- Start time *(the date and time the task started)*
- Elapsed hh:mi *(the duration of the task)*

Report Example



Sending Results to Microsoft Excel

Right-click on the result set and click **Output to File** to send the results of this report to Microsoft Excel.

FRAGMENTATION

This report is only available on a SQL Server database.

Objects Included

All object types are included in this report.

Parameters

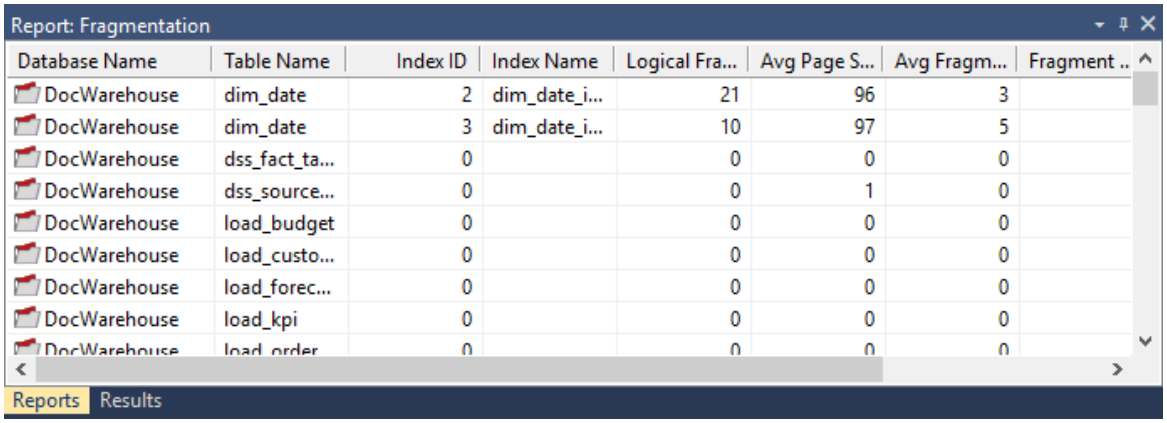
This report has no parameters.

Results

The results of this report are displayed as a list of tables within the database with the following columns:

- Database
- Table Name
- Index ID
- Index Name
- Logical Fragmentation
- Avg Page Space Used
- Avg Fragment Size in Pages
- Fragment Count
- Page Count
- Size in GB

Report Example



Sending Results to Microsoft Excel

Right-click on the result set and click **Output to File** to send the results of this report to Microsoft Excel.

WhereScape®

CHAPTER 27
VALIDATE

When these Validate processes are run, the results are displayed in the middle pane of the RED screen; the results of the other reports are displayed in a separate tab in the bottom pane of the RED screen.

IN THIS CHAPTER

VALIDATE META-DATA

This process validates the Meta data. If problems are encountered, the results are displayed in the middle pane.

Use the right-click option against each identified issue to apply a repair.

Sending Results to Microsoft Excel

Right-click on the result set and click **Output to File** to send the results of this report to Microsoft Excel.

VALIDATE WORKFLOW DATA

This process validates the Workflow data. If problems are encountered, the results are displayed in the middle pane.

Use the right-click option against each identified issue to apply a repair.

Sending Results to Microsoft Excel

Right-click on the result set and click **Output to File** to send the results of this report to Microsoft Excel.

VALIDATE TABLE CREATE STATUS

This process validates a table structure in the meta data against the table in the database.

- Select one or more tables and click the **Validate Selected** button or click the **Validate All** button to validate all the tables.

1. If a table is found to be different then it can be altered by using the right-click menu option when positioned over the table name.
2. If the update date and the modified in database date imply a change that is not apparent, then these dates can be re-synced in the same way.

Sync Column order with database

Right click on the result set and select **Sync Column order with database** to reorder the metadata columns to match the column order in the database table.

Sending Results to Microsoft Excel

Right-click on the result set and click **Output to File** to send the results of this report to Microsoft Excel.

VALIDATE LOAD TABLE STATUS

This process compares a load table in the meta data with the table in the source system. It compares the columns and column data types.

Sending Results to Microsoft Excel

Right-click on the result set and click **Output to File** to send the results of this report to Microsoft Excel.

VALIDATE INDEX STATUS

This process validates an index structure in the meta data against the index in the database. Select one or more indexes and click the **Validate Selected** button or click the **Validate All** button to validate all indexes.

Sending Results to Microsoft Excel

Right-click on the result set and click **Output to File** to send the results of this report to Microsoft Excel.

VALIDATE PROCEDURE STATUS

This process compares a procedure in the meta data with the compiled version of the same procedure stored within the database. The subsequent display will report either a match or a difference.

If a procedure is found to differ then you can use the procedure editor to examine the exact differences by selecting the **Tools/Compare to Compiled Source** option.

Sending Results to Microsoft Excel

Right-click on the result set and click **Output to File** to send the results of this report to Microsoft Excel.

LIST META-DATA TABLES NOT IN THE DATABASE

This report shows database **table** objects in the metadata that don't exist in the data warehouse database.

Objects Included

The following WhereScape RED object types are included in this report:

- Load Tables
- Stage Tables
- Dimension Tables and Views
- Fact Tables
- Aggregate Tables
- Retro Copies (but not Retro Definitions)

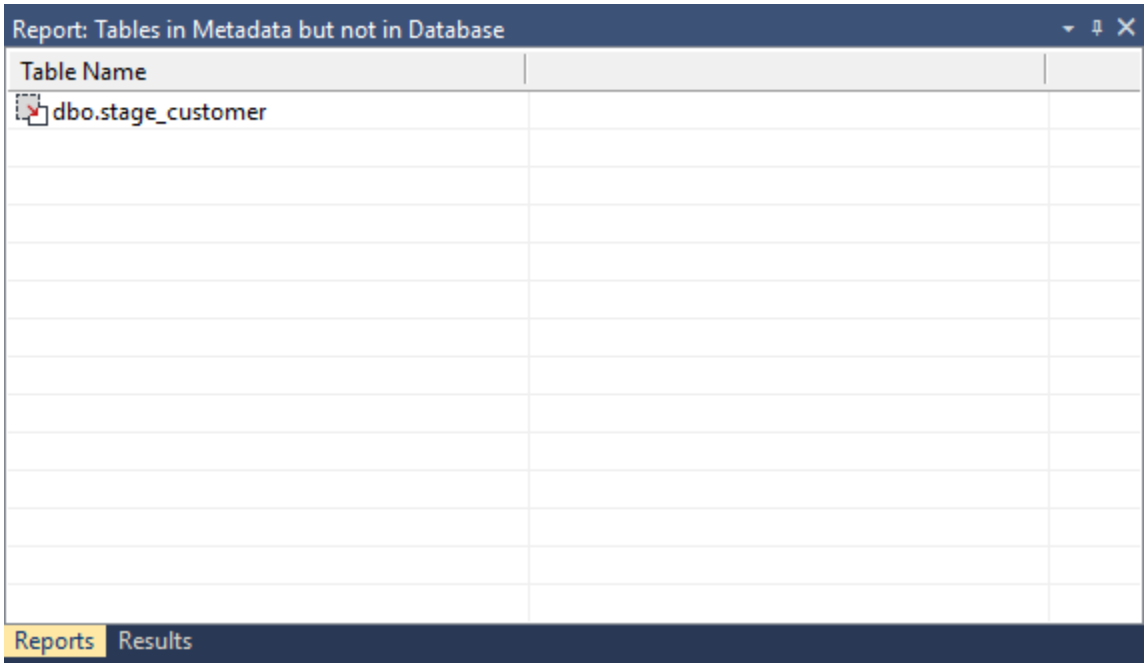
Parameters

There are no parameters for this report.

Results

The results of this report are displayed as a list of objects in the metadata not in the data warehouse database.

Report Example



Sending Results to Microsoft Excel

Right-click on the result set and click **Output to File** to send the results of this report to Microsoft Excel.

LIST DATABASE TABLES NOT IN THE META-DATA

This report shows database **table** objects that exist in the data warehouse database but not in the metadata.

Objects Included

The following WhereScape RED object types are included in this report:

- Load Tables
- Stage Tables
- Dimension Tables and Views
- Fact Tables
- Aggregate Tables
- Retro Copies (but not Retro Definitions)

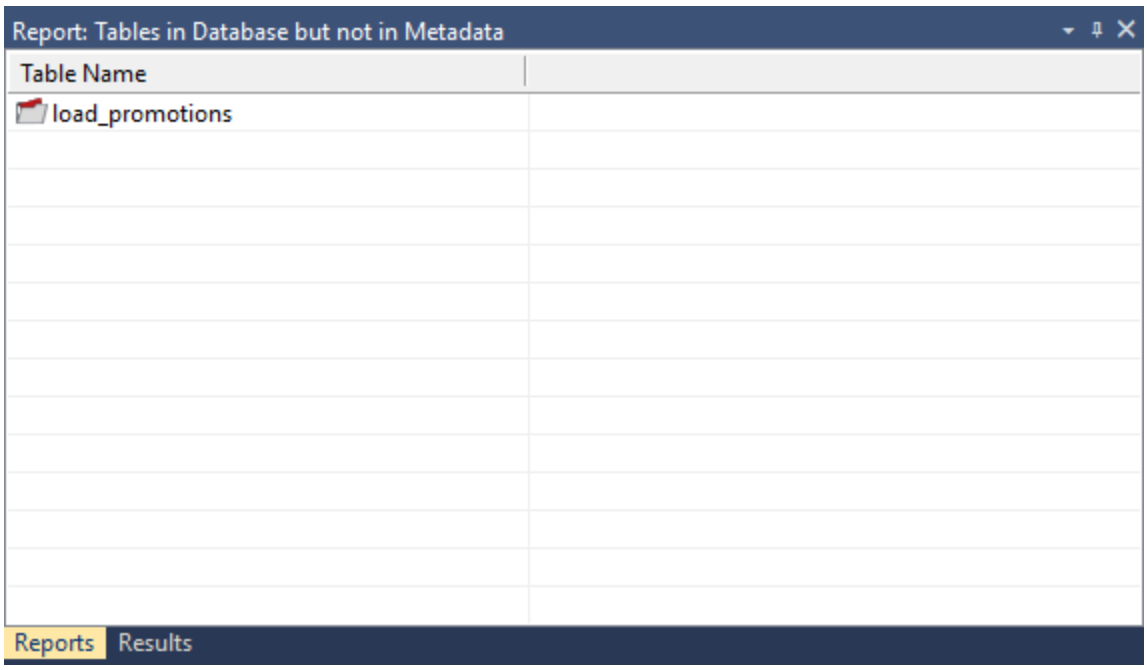
Parameters

There are no parameters for this report.

Results

The results of this report are displayed as a list of objects in the data warehouse database not in the metadata.

Report Example



Sending Results to Microsoft Excel

Right-click on the result set and click **Output to File** to send the results of this report to Microsoft Excel.

WhereScape®

LIST TABLES WITH NO RELATED PROCEDURES OR SCRIPTS

This report shows all table objects (certain types of objects only - see below) in the metadata repository that don't have an associated update procedure.

Objects Included

The following WhereScape RED object types are included in this report:

- Load Tables *(script based loads only)*
- Stage Tables
- Dimension Tables and Views
- Fact Tables
- Aggregate Tables
- Data Store Tables
- EDW 3NF Tables

Parameters

There are no parameters for this report.

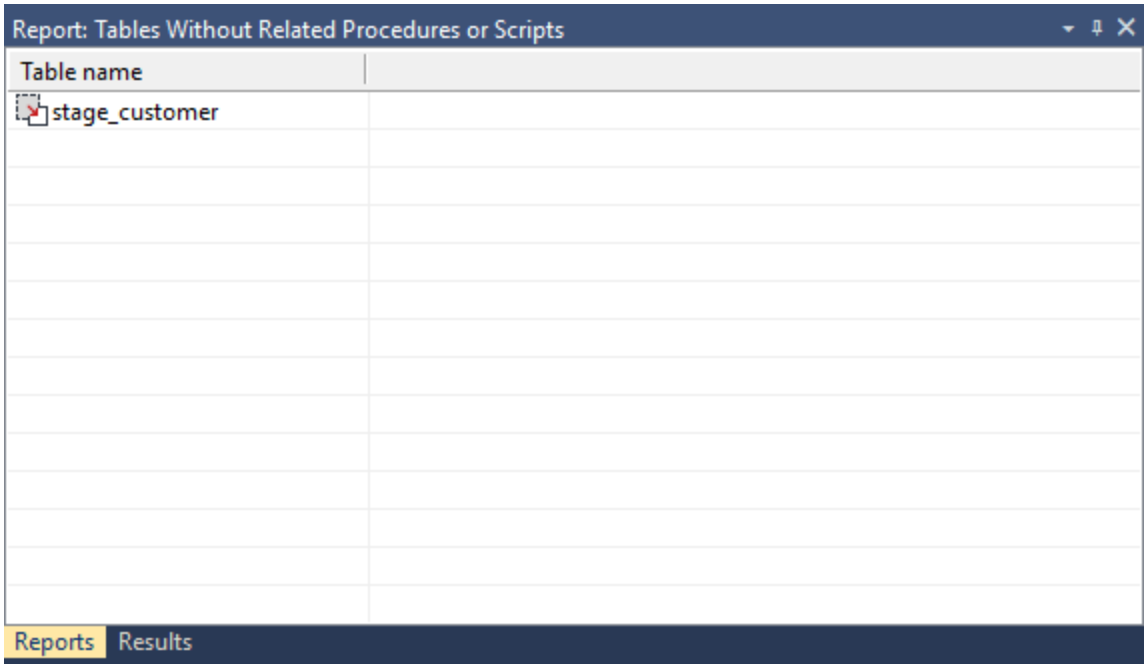
Results

The results of this report are displayed as a list of table objects in the metadata repository with the following columns:

- Table name

The result set is sortable by clicking on the appropriate column heading.

Report Example



Sending Results to Microsoft Excel

Right-click on the result set and click **Output to File** to send the results of this report to Microsoft Excel.

LIST PROCEDURES NOT RELATED TO A TABLE

This report shows all procedures and host scripts in the metadata repository that aren't associated with a table object.

Objects Included

The following WhereScape RED object types are included in this report:

- Procedures
- Host Scripts

Parameters

There are no parameters for this report.

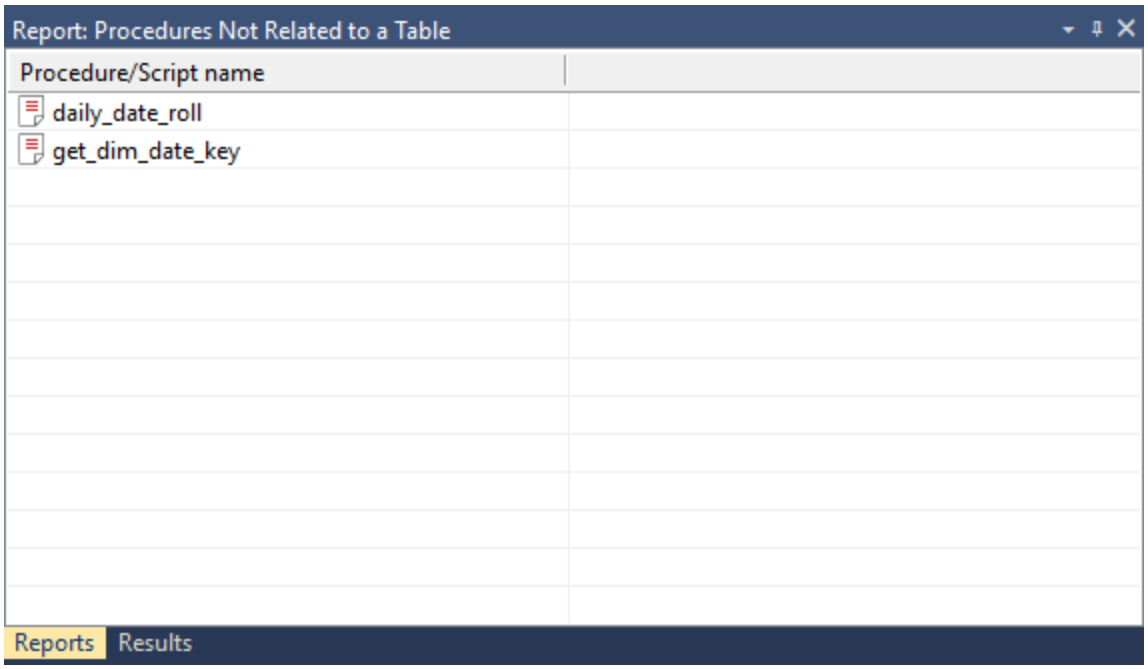
Results

The results of this report are displayed as a list of code objects in the metadata repository with the following columns:

- Procedure/Script name

The result set is sortable by clicking on the appropriate column heading.

Report Example



Sending Results to Microsoft Excel

Right-click on the result set and click **Output to File** to send the results of this report to Microsoft Excel.

COMPARE META-DATA REPOSITORY TO ANOTHER

This report compares the current metadata repository to a remote repository.

Objects Included

All object types are included in this report.

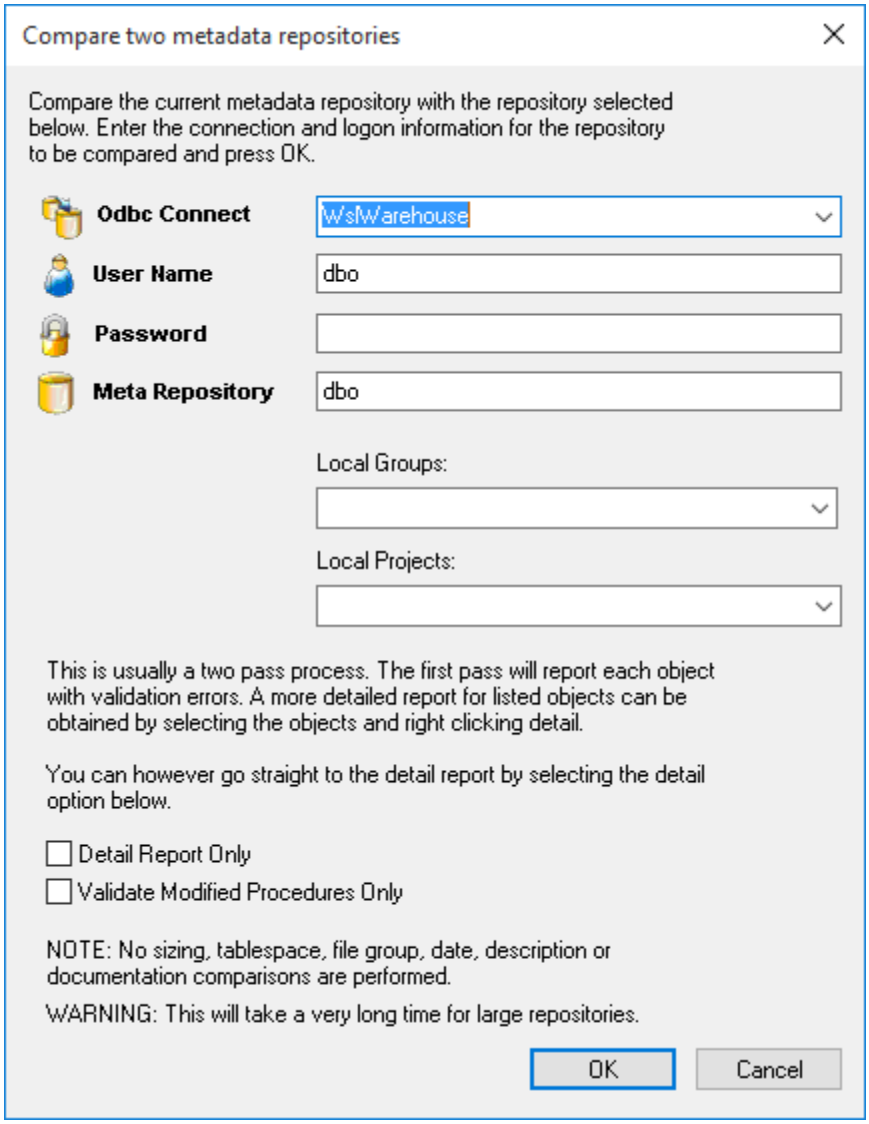
Parameters

This report requires connection information for the remote repository to be entered. Specifically:

- Odbc Connect *(the odbc source for the other repository)*
- User Name *(user name for the remote repository)*
- Password *(password for the remote repository)*
- Meta Repository *(database/user the remote repository metadata is stored in)*

Four additional parameters can be specified:

- Optional filter on Local Groups
- Optional filter on Local Projects
- Do detail report
- Only validate procedures that have been modified



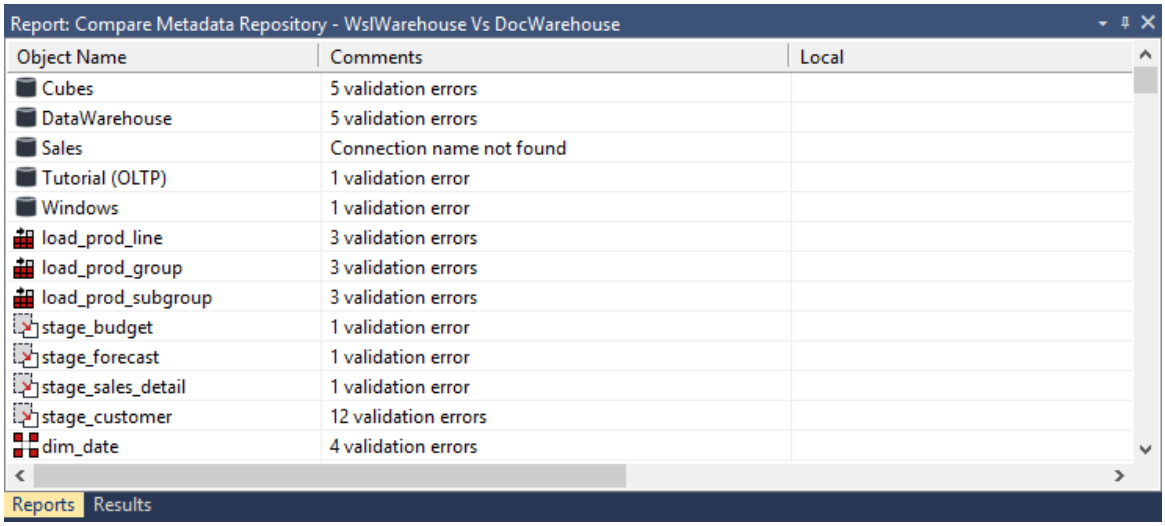
Results

The results of this report are displayed as a list of differences with the following columns:

- Object Name
- Comments *(Summary difference between current and selected repository)*
- Local
- Remote

The result set is sortable by clicking on the appropriate column heading.

Report Example



Checking result details:

- Right-click on the object name with validation errors.
- Select **Detail.**
- This will rerun the report just for the selected object and will display more details about the errors in the Comments, Local and/or Remote Column(s).

Sending Results to Microsoft Excel

- Right-click on the result set and click **Output to File** to send the results of this report to Microsoft Excel.

COMPARE META-DATA INDEXES TO DATABASE

This report shows database indexes in the data warehouse database that don't exist in the metadata.

Objects Included

The following WhereScape RED object types are included in this report:

- Indexes

Parameters

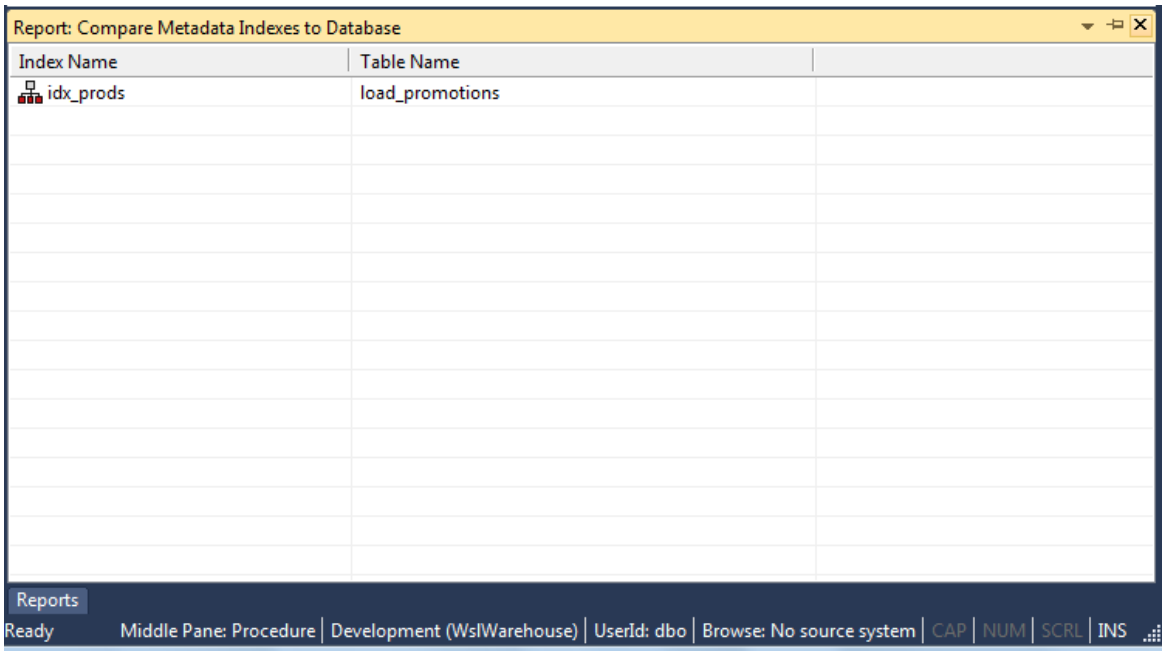
There are no parameters for this report.

Results

The results of this report are displayed as a list of indexes in the data warehouse database not in the metadata with the following columns:

- Index Name *(the name of the index)*
- Table Name

Report Example



Sending Results to Microsoft Excel

Right-click on the result set and click **Output to File** to send the results of this report to Microsoft Excel.

LIST DUPLICATE BUSINESS KEY COLUMNS

This report shows any columns that are the business (natural) key of more than one table.

Objects Included

The following WhereScape RED object types are included in this report:

- Stage Tables
- Dimension Tables and Views
- Fact Tables
- Aggregate Tables

Parameters

There are no parameters for this report.

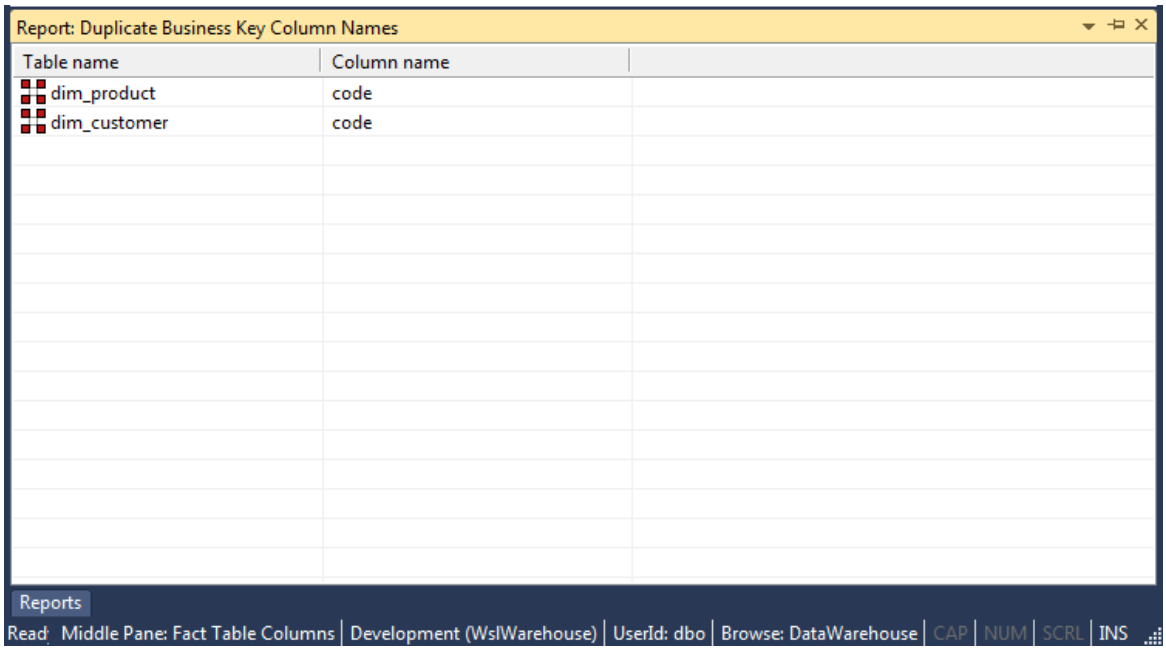
Results

The results of this report are displayed as a list of table columns with the following columns:

- Table name
- Column name

The result set is sortable by clicking on the appropriate column heading.

Report Example



Sending Results to Microsoft Excel

Right-click on the result set and click **Output to File** to send the results of this report to Microsoft Excel.

QUERY DATA WAREHOUSE OBJECTS

This report allows SQL queries to be run as the user signed into the repository.

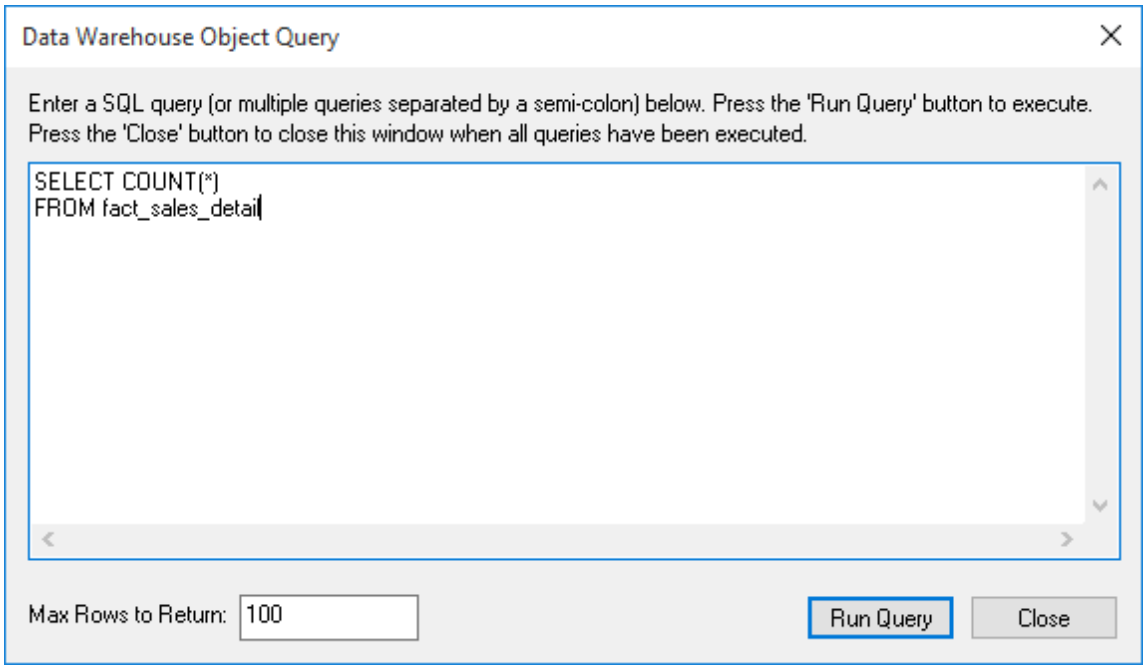
Objects Included

All object types and data warehouse tables can be included in this report.

Parameters

This report has one parameter:

- The SQL Query



Results

The results of this report are displayed with the following columns:

- First SQL column SELECTed
- Second SQL column SELECTed
- ...
- nth SQL column SELECTed

The result set is sortable by clicking on the appropriate column heading.

Report Example

Sending Results to Microsoft Excel

Right-click on the result set and click **Output to File** to send the results of this report to Microsoft Excel.

WhereScape®

CHAPTER 28
PROMOTING BETWEEN ENVIRONMENTS

This chapter covers the promotion of metadata objects between environments. Various methods exist for getting new or changed metadata from one repository into another repository.
It is of course possible and in fact desirable to have multiple metadata repositories. At the very least we would normally have a development and a production environment.
In some situations it may also be desirable to have multiple child development repositories with one master repository where all elements are brought together. WhereScape RED supports this type of structure but does not include source control or co-ordination of the repositories. It is up to the data warehouse manager to manually ensure that the various objects are kept in sync and coordinated.
As with any software system there are issues around how code is moved from a development environment into a testing or production environment.
This promotion of objects can be achieved via a number of different methods. Each is discussed below.
In summary they are:

1. Updating a repository with an application or application patch.
2. Importing objects from another repository.
3. Restoring a full metadata set into a repository (see ***Backup and Restore*** (see "***Backing Up and Restoring Metadata***" on page 932)).

IN THIS CHAPTER

APPLICATIONS

The definition of an application is discussed in the following section on applications and the loading and updating of applications is discussed at some length in the **Installation and Administration Guide**. Only the concepts of the use of applications will be covered here.

An application is defined for our purposes as a group of objects. An application is a method of loading objects into a metadata repository. It can be used to upgrade or provide patches to an existing metadata repository. As such an application can be used to distribute and remotely maintain a specific data warehousing solution.
An application consists of a series of Windows files, which can be distributed to remote sites.
A list of the applications that have been applied to the metadata repository can be acquired via the **Tools/List Loaded Deployment Applications** menu option.

An application is created through the **Tools/Build Deployment Application** menu option. This application can then update a metadata repository through the Setup/Administration utility. In this manner the application model can be used to update a metadata repository in an ordered and controlled fashion. Loading an application inserts various objects into the chosen metadata repository. An application is best defined as a set of objects that are shipped to allow inclusion of those objects in a remote repository.

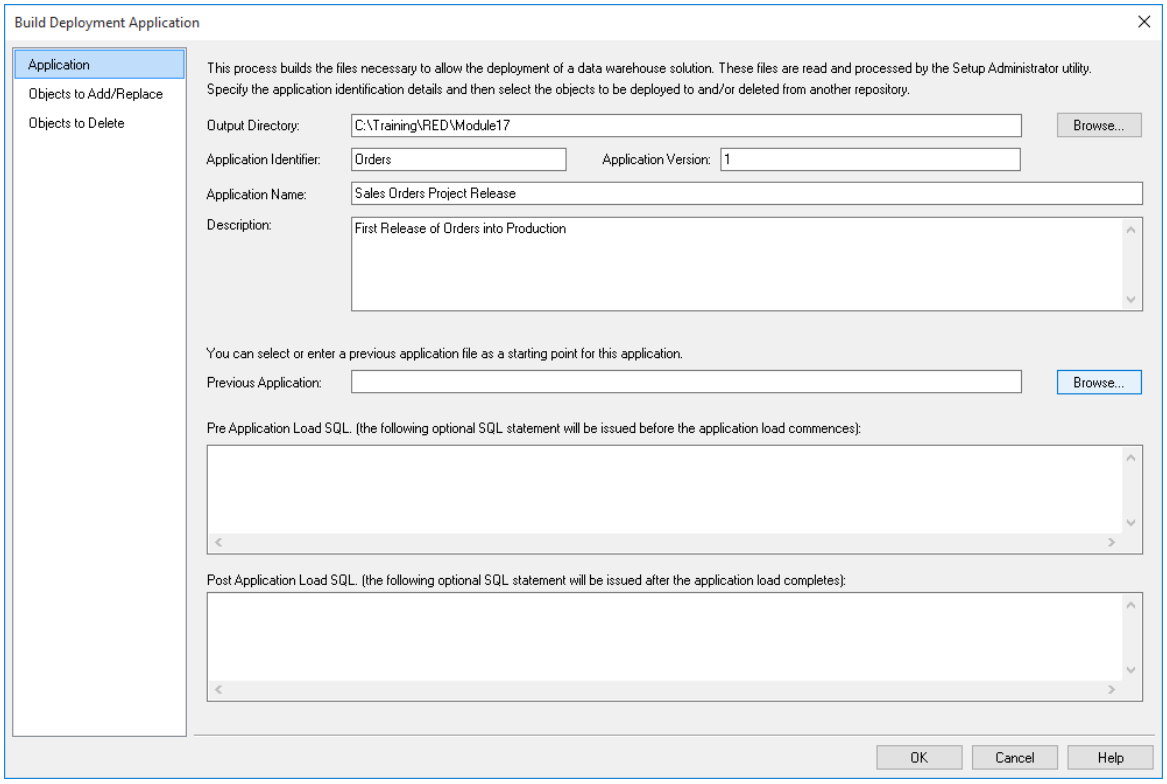
Note: An application can only be loaded into a metadata repository running on the same database type as that of the application creator. (e.g. An Oracle application cannot be loaded into a SQL Server metadata repository).

APPLICATION CREATION

Creating an Application

An application is created by selecting the **Tools/Build Deployment Application** menu option. The following dialog box is displayed. Once the application is defined and the objects selected, the application files are generated when the **OK** button is clicked. If procedures are compiled as part of the subsequent application load, the compiles occur in the order in which they are listed in the application. If procedure dependencies exist, ensure that their ordering in the application object list is correct.
There are three tabs in the **Build Deployment Application** screen. The first tab defines the application, the second lists the objects to add or replace in the destination repository and the third tab lists the objects to delete in the destination repository.

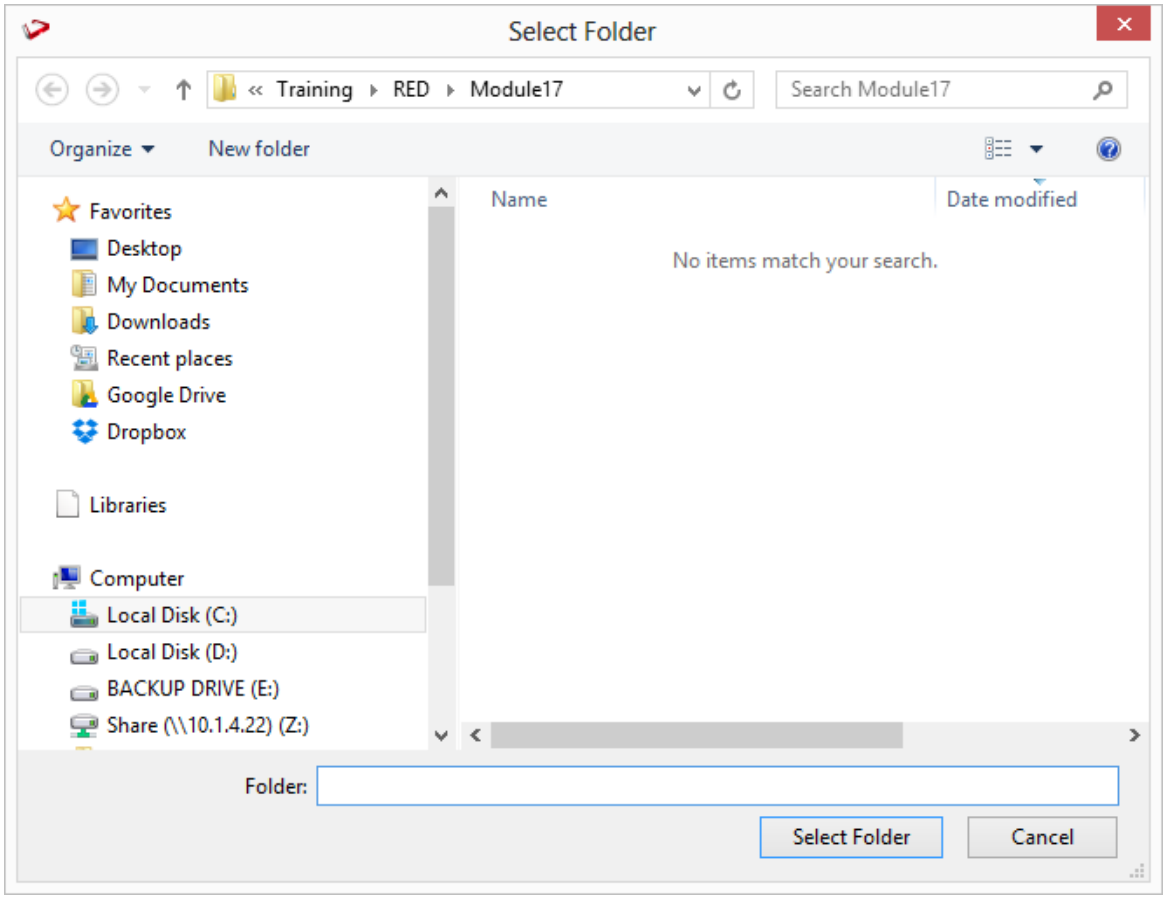
Define an Application distribution



Output Directory

The directory to which the application files will be written. By default this will be the WhereScape program directory.

Note: To browse for the required folder, click on the **Browse...** button.



The **Make New Folder** button allows you to create a new folder in the currently selected directory.

Application Identifier

The application identifier is a four character code used to uniquely identify the application. This identifier is used in the naming of the files that are created to contain the application data.

Application Version

The version is a character string that provides a version number for reference purposes. This version number is displayed when applications are being loaded, and is used in the naming of the files that are created to contain the application data. As such it must contain characters that are valid in a Windows file name.

Application Name

The name by which the application is known. This name is displayed during the choosing of an application to load and is recorded in the metadata of the repository into which an application is loaded. It is not used apart from documentation purposes.

Description

This description is displayed during the choosing of an application to load. It is not used at any other point apart from documentation purposes.

Application Files

When an application is created the following files are built, where XXXX is the application identifier and NNNNN is the application version.

File	Purpose
App_data_XXXX_NNNNN.wst	This file contains the scripts and data required to rebuild the objects in the new metadata repository.
App_id_XXXX_NNNNN.wst	This control file identifies the application and its version.
App_obj_XXXX_NNNNN.wst	This file contains control information and each object in the application.
App_con_XXXX_NNNNN.wst	A list of all the connections either in the application or used by objects in the application.
App_map_XXXX_NNNNN.wst	A list of all the project and group mappings for the objects in the application.

Previous Application

Click on the **Browse** button next to **Previous application** to choose a previously built application to use as a list of objects to include in the new application. After using a previous application as a starting point for this application, additional objects can be added or removed from the application.

Pre application load SQL

This box allows the entry of a SQL Statement that will be executed before the application is loaded. For example we may wish to drop the date dimension before loading the application because we have changed the primary key constraint. In such a case we would enter 'drop table dim_date' in this field to have the table dropped before the application is loaded.

Post application load SQL

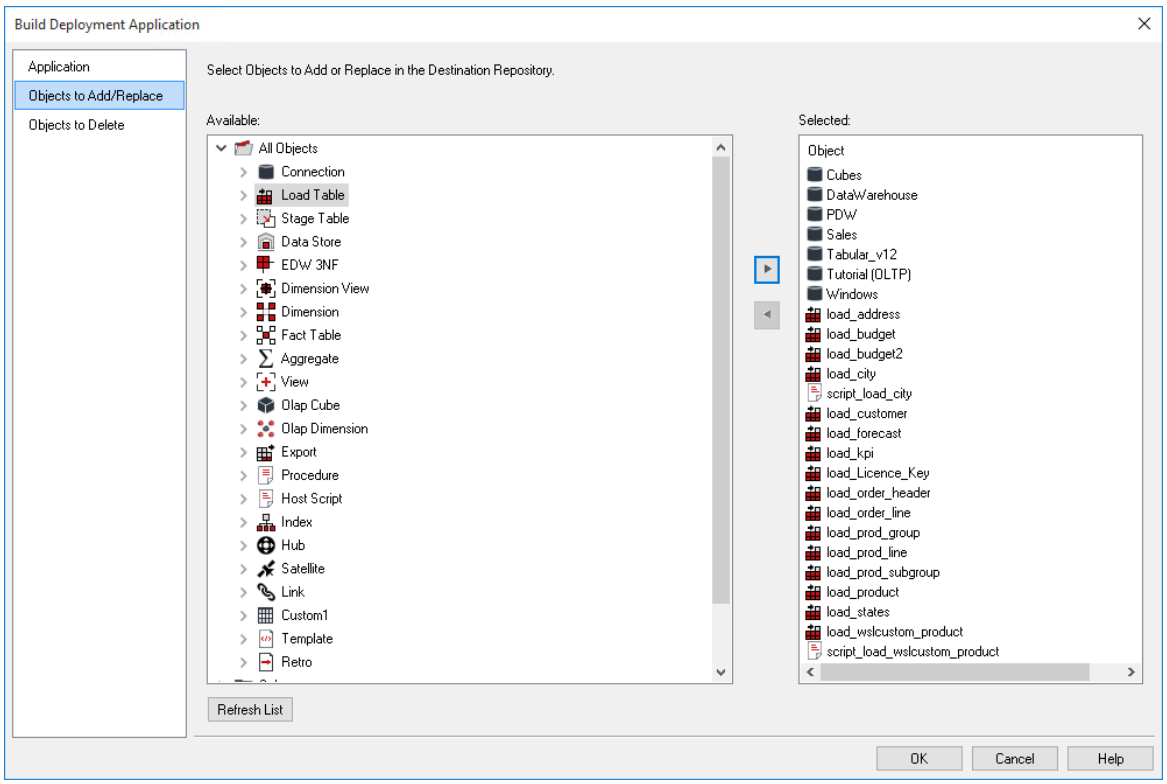
This box allows the entry of a SQL Statement that will be executed after the application is loaded. For example we could execute a function to populate a table.

Objects to Add/Replace

Objects can be moved from the left object tree by double-clicking on an object name or by using the > button. This tab allows you to select the objects to add or replace in the destination repository.

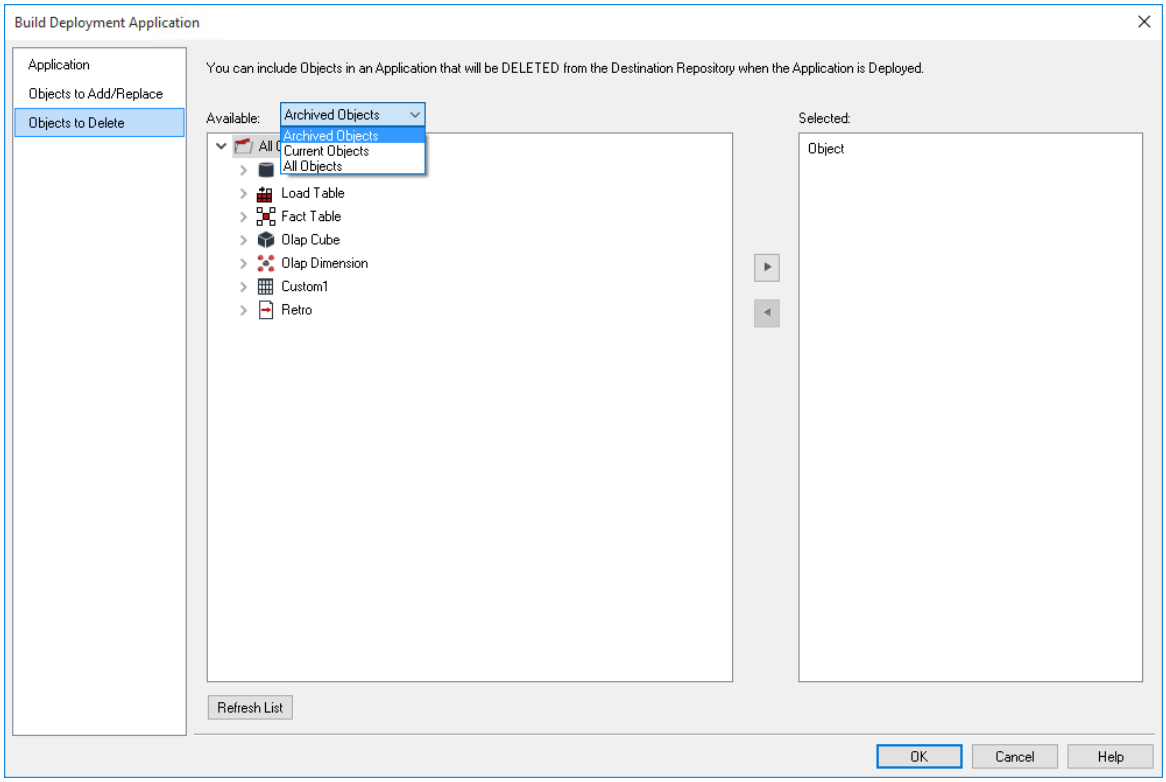
NOTE: Maximum number of objects in an application

5000 objects (including jobs)
2000 source views of views
1000 jobs



Objects to Delete

Objects can be moved from the left object tree by double-clicking on an object name or by using the **>** button. This tab allows you to select the objects to delete in the destination repository.



NOTE: To set the objects available for selection, choose from the **Available** drop-down list. The options are Archived Objects, Current Objects and All Objects. The default is Archived Objects.

APPLICATION LOADING

Note: Applications can only be loaded into the same relational database type from which they where created. (For Example an Oracle application can only be loaded into an Oracle database).

Applications are loaded via the Setup Administrator utility. The normal process for implementing an application would be as follows:

1. Run the **Setup Administrator** utility.
2. Change the application directory to the application's location.
3. Turn on logging in the Setup Administrator utility using **Tools/Start Logging**.
4. Load the application via the Setup Administrator utility.
5. Choose the level of metadata application. There are several levels, from load metadata only through to load metadata and apply changes to all tables.
6. Resolve any connections and tablespaces/filegroups to those present in the target environment.
7. Create/Re-create/Alter database tables, if selected in (5).
8. Compile database procedures, if selected in (5).
9. Turn off logging.
10. Review the output in the Setup Administrator utility.
11. Review the log file.

Note: Some database operations, such as converting an existing non-partitioned table to a partitioned table, cannot be done using a deployment application. In these cases some manual intervention may be required to update the target databases to match the new metadata.

Refer to the **Setup Administrator** manual for more information about loading an application.

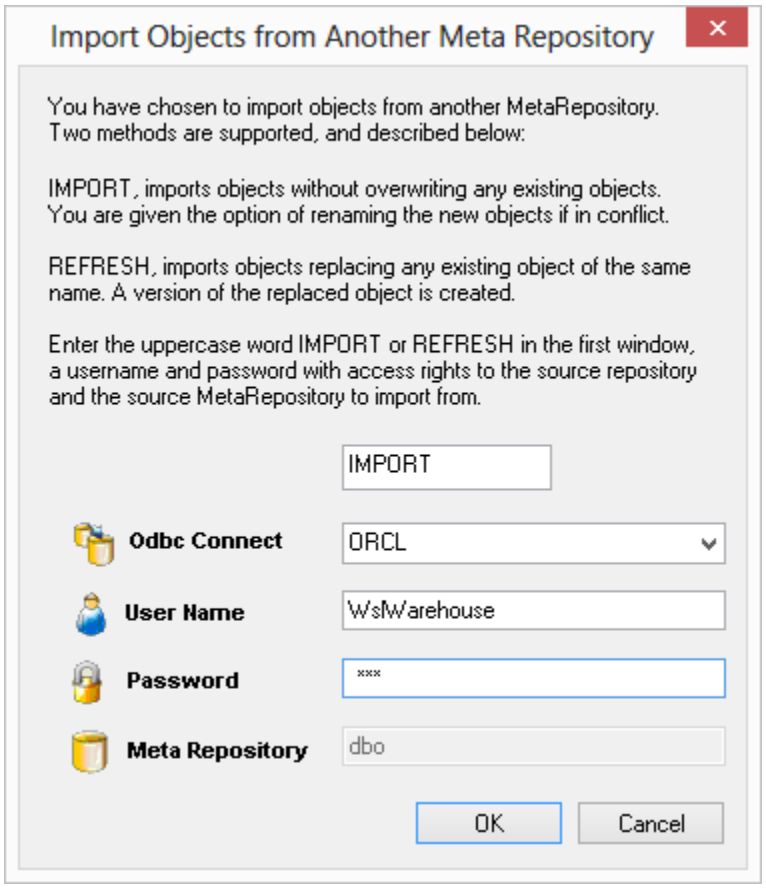
CREATING AND LOADING APPLICATIONS FROM THE COMMAND LINE

It is possible to create and load applications from the command line by running a bat file.
For more detailed instructions, please see section **12.2 Creating and Loading Applications from the Command Line** in the **RED Installation Guide**.

IMPORTING OBJECT METADATA

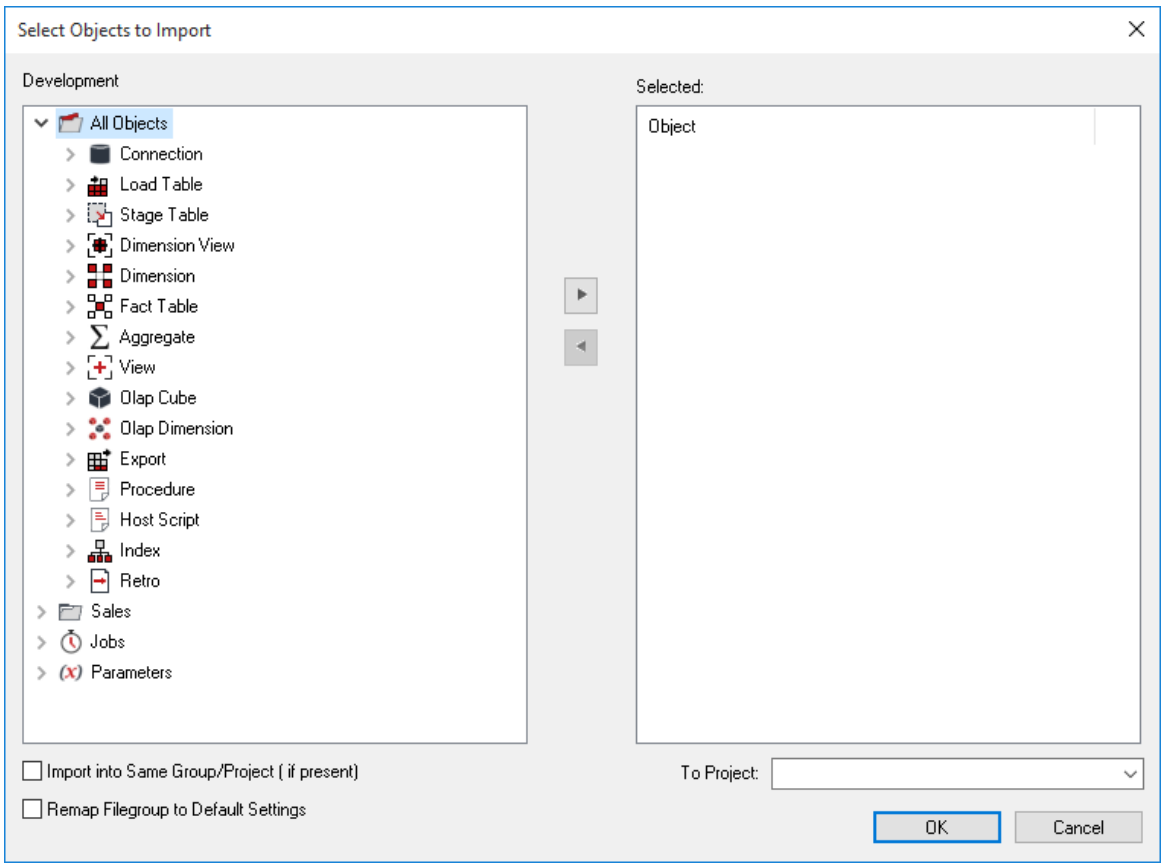
Any group of objects can be imported into the current metadata repository from another repository. If an object already exists in the target repository then it is either skipped or replaced depending on the type of import undertaken. If an object is to be replaced as part of an import, a version of the object is created prior to its replacement.

To import an object or group of objects select the **Tools/Import Metadata Objects** menu option. A dialog as below will appear. The two options are IMPORT or REFRESH. An import will not replace an existing object of the same name. A refresh will version and replace any existing object of the same name.



Enter the connection, and a database user name and password that has access to the source metadata repository.
Finally, enter the user name of the metadata repository you want to import from. In most situations, the **User Name** and **Meta Repository** would be the same.
However, if you only have read access to a meta repository then it may be necessary to login to the database under a different user name from that of the repository you are trying to import from.
You are not permitted to select the current meta repository in the **Meta Repository** field, but are permitted to login using the existing repository username.
Once a successful logon is completed the contents of the source repository are loaded and the following dialog appears.

Select an object by double-clicking on it or by selecting and using the **>** button. If an object such as a table is chosen then any related scripts, procedures and indexes are also selected.
They can be removed if required. A target project can be selected.



Once all required objects are selected the import will commence when the **OK** button is clicked.
On completion a dialog box will appear notifying of the number of each type of object imported, and skipped.

Note: The repository from which you are importing should be the same metadata version as the target repository.

IMPORTING LANGUAGE FILES

Note: Applications can only be loaded into the same relational database type from which they were created. (For Example, an Oracle application can only be loaded into an Oracle database).

Language Files are loaded via the Setup Administrator utility. The normal process for implementing a Language file would be as follows:

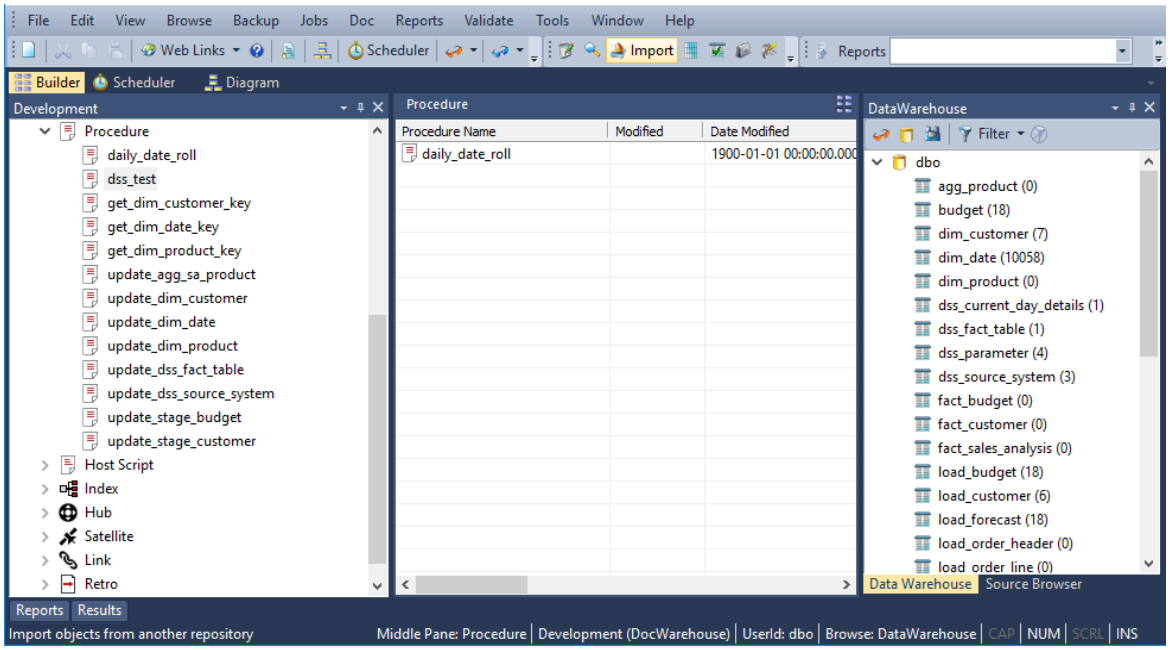
1. Run the **Setup Administrator** utility.
2. Go to the Languages menu item in the top command bar and select **Load Languages**.
3. Right-click on the Language file to be loaded and select **Install Language**.
4. Select the ODBC data source and Log on to the target meta repository.
5. Select the language to be updated.
6. Review the output in the Setup Administrator utility.

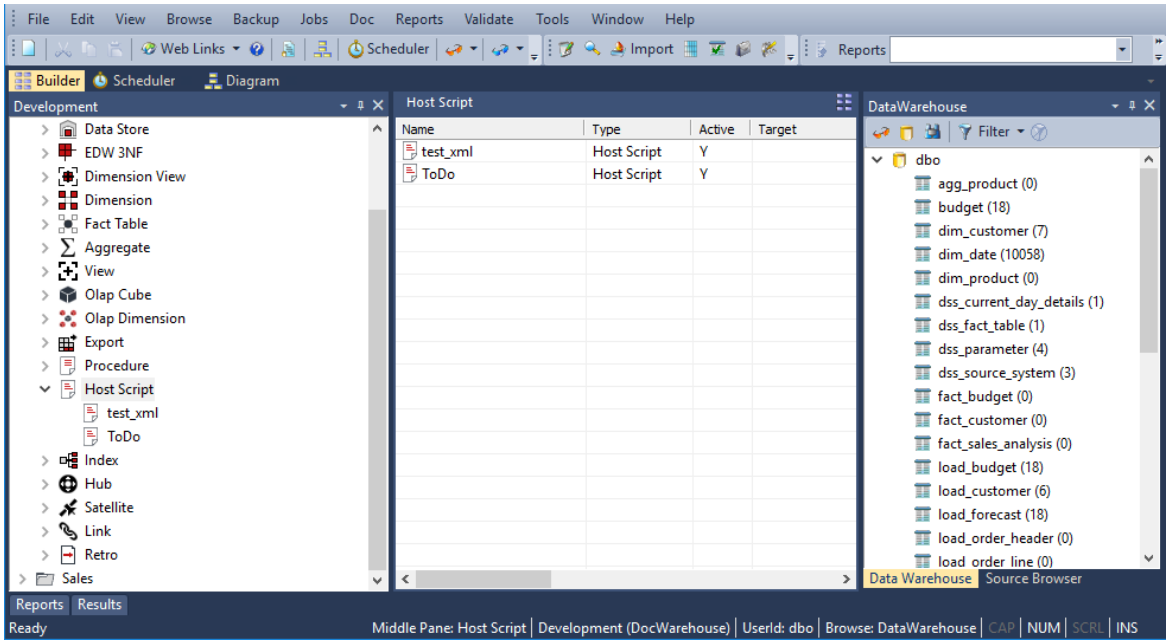
Refer to the **Setup Administrator** manual for more information about loading a Language File.

DATA WAREHOUSE TESTING

Note: Applications can only be loaded into the same relational database type from which they were created. (For Example, an Oracle application can only be loaded into an Oracle database).

Testing applications are loaded via the Setup Administrator utility. Refer to the Setup Administrator manual for more information on how to load an application.
A testing application set consists of a Procedure and an XML script and provides the ability to define a series of tests against data warehouse objects; either comparing them to an expected value or to the results of a query.
Once the application set has been loaded, the Procedure and the XML script will be visible in the left pane.



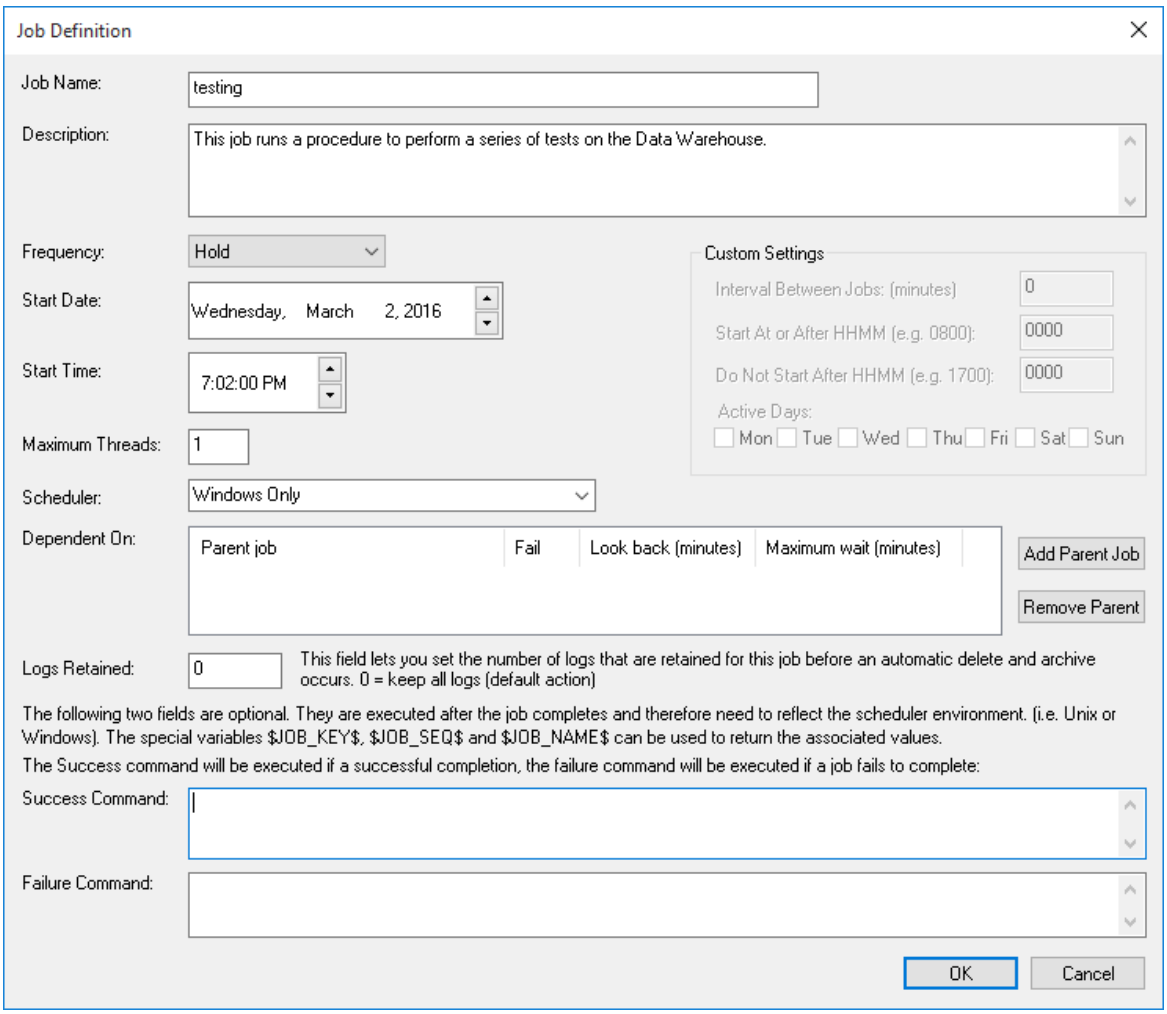


The XML script contains the test definitions. Each test is a new XML node in the comparison query. The procedure simply runs the test and determines whether the tests are passed or not. This is most likely to be run as a scheduled job within WhereScape RED. To create a job:

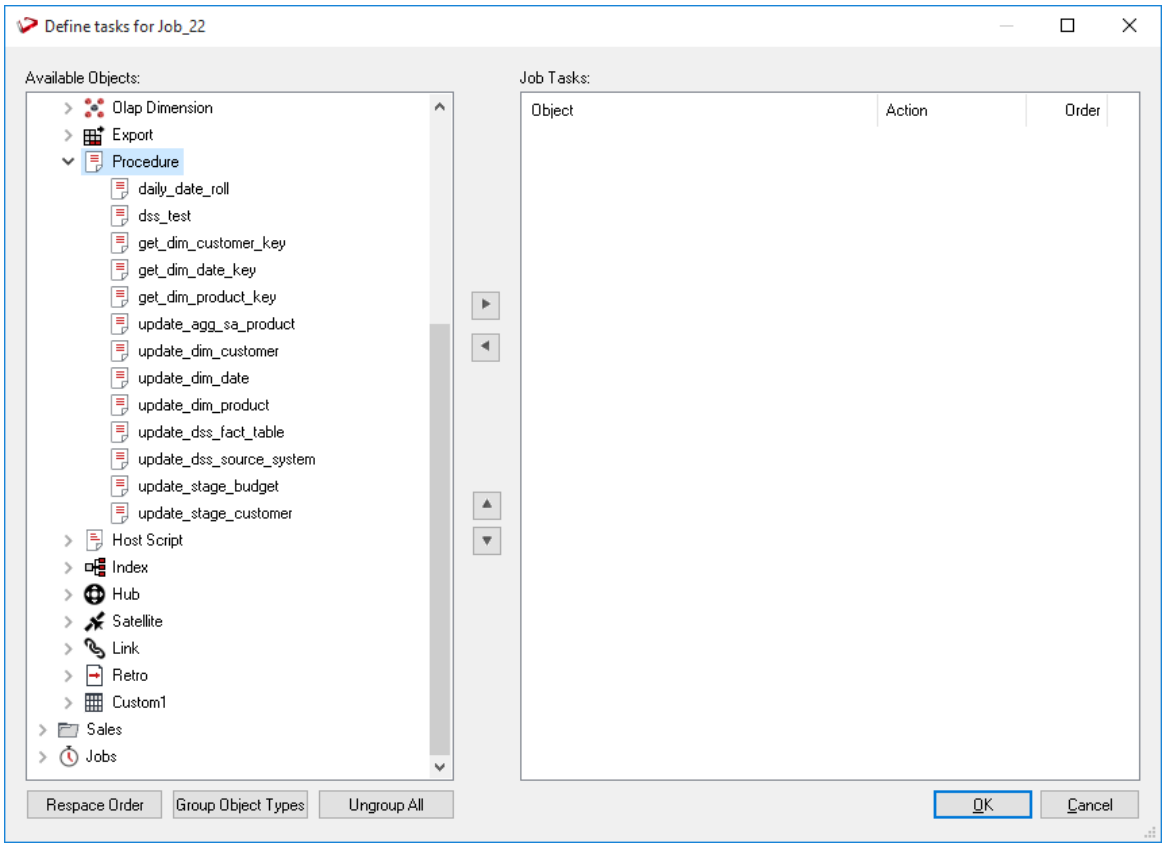
1 Click the **Scheduler** Button.

2 Choose **File** and then **New Job**.

3 Enter the definition of the job.



4 To select the test procedure as a task, open the Procedure object heading in the left pane. Choose **dss_test** and the **>** button. Click **OK**.



5 To run the job, click on the **All Jobs** button and then right-click on the job and select **Start the Job**.

WhereScape®

CHAPTER 29
BACKING UP AND RESTORING METADATA

This chapter covers the saving and reloading of metadata repository objects. The backup section describes the methods for backing up the metadata repository. It can also be backed up via normal database backup procedures. The restore section covers the metadata restoration functions available.

WhereScape RED provides two different methods for backing up and restoring metadata:

1. Backup and Restore using the native backup tools available in the data warehouse database.
2. WhereScape RED unload and load.

IN THIS CHAPTER

BACKUP USING DB ROUTINES

The backup of the metadata repository can be undertaken as a separate exercise from the general backup of the data warehouse. The metadata backup does not backup any of the actual data warehouse tables. It only saves the table definitions, index definitions, procedures etc., so is not normally large.

The backup includes any tables that begin with "dss_". In this way key metadata tables such as dss_parameter, dss_source_system and dss_fact_table are included in the backup. These tables are required if the restored metadata is to function correctly.

It is recommended that the metadata is backed up at least daily when possible. There are two methods for backing up the meta repository. These are through a scheduled UNIX task, and through the main WhereScape RED tool.

UNIX Backups

Where the data warehouse database is running on a UNIX server, it is possible to schedule a regular backup of the metadata. The UNIX scripts shipped with WhereScape RED include a script called from cron to daily backup the metadata. These backups are in turn compressed and saved for 60 days by default. Refer to the **Installation and Administration Guide** for details on these scripts and their setup.

Windows Backups

Two main methods of Windows backup exist within WhereScape RED. The first is a database independent backup which is designed purely for moving the meta repository, and should not be used for regular backups. The other method is the database specific backup. See the following sections for the appropriate database.

Oracle Windows Backups

An Oracle export based backup can be taken from the WhereScape RED tool by selecting the **Backup/Export the metadata (Oracle Export)** menu option. This option assumes that the Oracle Client on the PC is the same version as that of the Oracle database where the WhereScape RED meta repository is stored. The backup may not work if the versions differ.

When executed this menu option attempts to locate the Oracle export utility. This utility is normally called 'exp' and resides in the Oracle bin directory. If WhereScape RED cannot locate this utility or it has not been loaded onto the PC then the export does not proceed and an error message will be displayed.

A pop-up window asks for a file name for the export. A directory should be chosen and a name entered. When the **Save** button is clicked the export will start. The following files are created (where 'file' is the name of the chosen file). A dialog box appears to show the results of the backup.

File name	Purpose
file.bat	Windows command file containing the export command.
file.dbl	Command file for the re-creation of any database links.
file.drp	Command file for dropping all metadata tables. (Used in the restore prior to a reload).
file.exp	Oracle export file.
file.log	Log file of the export.
file.par	Parameter file for the export. Contains an entry for each meta table to be exported.
file.pro	A file containing each procedure stored in the metadata.
file.seq	Command file to re-create the sequences used by both the metadata and the data warehouse tables.

If problems are encountered with the backup it may be possible to manually run the generated .bat script file to ascertain what has gone wrong.

SQL Server Windows Backups

A SQL Server bcp based backup can be taken from the WhereScape RED tool by selecting the **Backup/Export the metadata (Sql server bcp)** menu option. This option assumes that the SQL Server bcp utility is available on the PC.

When executed this menu option attempts to locate the SQL Server bcp utility. This utility is normally called 'bcp' and resides in the SQL Server bin directory. If WhereScape RED cannot locate this utility or it has not been loaded onto the PC then the export does not proceed and an error message will be displayed.

A pop-up window asks for a directory name for the export. A directory should be chosen or created. When the **Save** button is clicked the export will start. The following files are created. A dialog box appears to show the results of the backup.

File name	Purpose
wsl_bcp_out.bat	Windows batch file used to run all the bcp table exports
wsl_bcp_out.err	Errors from the bcp export of each table
wsl_bcp_out.log	Log file of the bcp table exports
*.bcp	Each metadata table is stored with a .bcp extension. This is the export of each table.
wsl_bcp.par	Parameter file for the export. Contains an entry for each meta table to be exported. Unused by SQL Server.
wsl_bcp.drp	Command file for dropping all metadata tables. (Used in the restore prior to a reload).

File name	Purpose
wsl_bcp.pro	A file containing each procedure stored in the metadata.
wsl_bcp.seq	Unused by SQL Server.
wsl_bcp.dbl	Command file for the rebuild of all the linked servers. Used during the restore.
wsl_bcp.bld	Basis of the command file that will be used to restore the metadata. This file is modified to include the target directory prior to restore.

If problems are encountered with the backup it may be possible to manually run the generated .bat script file to ascertain what has gone wrong.

DB2 Windows Backups

A DB2 IMPORT based backup can be taken from the WhereScape RED tool by selecting the **Backup/Export the metadata (DB2 export)** menu option. This option assumes that the IBM DB2 EXPORT command is available on the PC.

When executed this menu option attempts to locate the SQL Server bcp utility. This utility is normally called **bcp** and resides in the SQL Server bin directory. If WhereScape RED cannot locate this utility or it has not been loaded onto the PC then the export does not proceed and an error message will be displayed.

A pop-up window asks for a directory name for the export. A directory should be chosen or created. When the **Save** button is clicked the export will start. The following files are created. A dialog box appears to show the results of the backup.

File name	Purpose
RED.bat	Windows batch file used to run the DB2 EXPORT for each metadata table.
RED.cmd	Control file for the export. Contains an entry for each meta table to be exported.
RED.log	Log file of the EXPORT.
RED.pro	A file containing each procedure stored in the metadata.
*.ixf	Each metadata table is stored in a file with a ixf extension. This is the export of each table.
restore.drp	Command file for dropping all metadata tables. (Used in the restore prior to a reload).
restore.lst	Basis of the command file that will be used to restore the metadata. This file is modified to include the target directory prior to restore.

If problems are encountered with the backup it may be possible to manually run the generated RED.bat script file to ascertain what has gone wrong.

RESTORING DB BACKUPS

WhereScape RED metadata can be restored from a prior backup. For Oracle data warehouses the restore can take place either from UNIX or through the WhereScape RED tool. The same set of files as defined in the backup section are used for both restores and can be moved between the Windows and UNIX environments. For SQL Server data warehouses the restore must take place in a Windows environment either through the WhereScape RED tool or manually.

Note: If transferring the export files via ftp, ensure that the file.exp and .bcp files are transferred in binary mode and that the other files are transferred in ASCII mode.

Restoring metadata

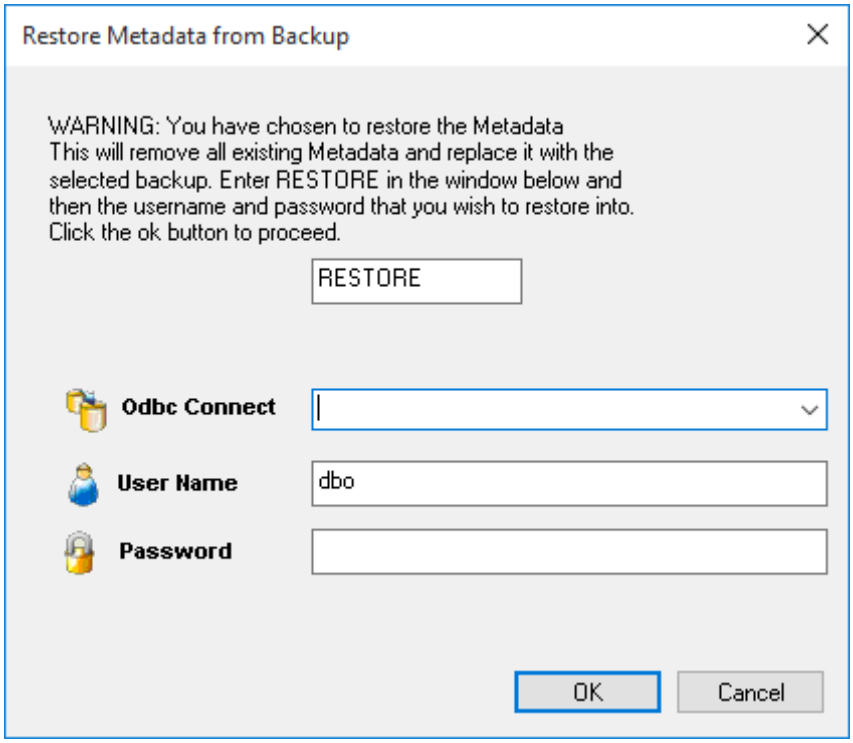
A backup of a metadata repository can be restored over the top of an existing repository. This action replaces the existing repository in its entirety. The restore of repositories is covered in an earlier section, but its uses in the promoting of code are discussed here. This restore process does not affect any existing database tables or any compiled procedures, so it can be used as a means of updating a metadata repository, albeit by replacing that repository. It is often a good method to choose when first establishing a new repository. For example, if we have a development repository and want to create a production environment the steps may be:

1. Backup the development repository, using either the UNIX or Windows repository backup. See the previous sections.
2. Create a new repository using the **Setup/Administrator** utility.
3. Log on to the newly created repository.
4. Restore the backup of the development repository into the new repository. For Oracle, this requires the database privilege 'imp_full_database', or the dba role.
5. Modify any connections, set any default values, alter any table sizes or extents.
6. Create the tables.
7. Compile all procedures.

Oracle Windows Restore

Select the menu option **Backup/Restore Metadata (from Oracle Export)** to begin the restore process.

A dialog box will appear. The word **RESTORE** needs to be entered along with the **User Name** and **Password** where the metadata is to be restored to. The **User Name** does not have to be that of the current metadata repository, but it must be a valid repository. If a restore is being performed to a repository that differs from that of the backup, then the 'imp_full_database' database privilege may be required.



Once the **OK** button is clicked a new dialog box will appear asking for a selection of the export file. Browse to the directory where the export is located and select the export file. Once the export file is selected the import will begin. A dialog box will appear to show the results of the import.

Oracle UNIX restore

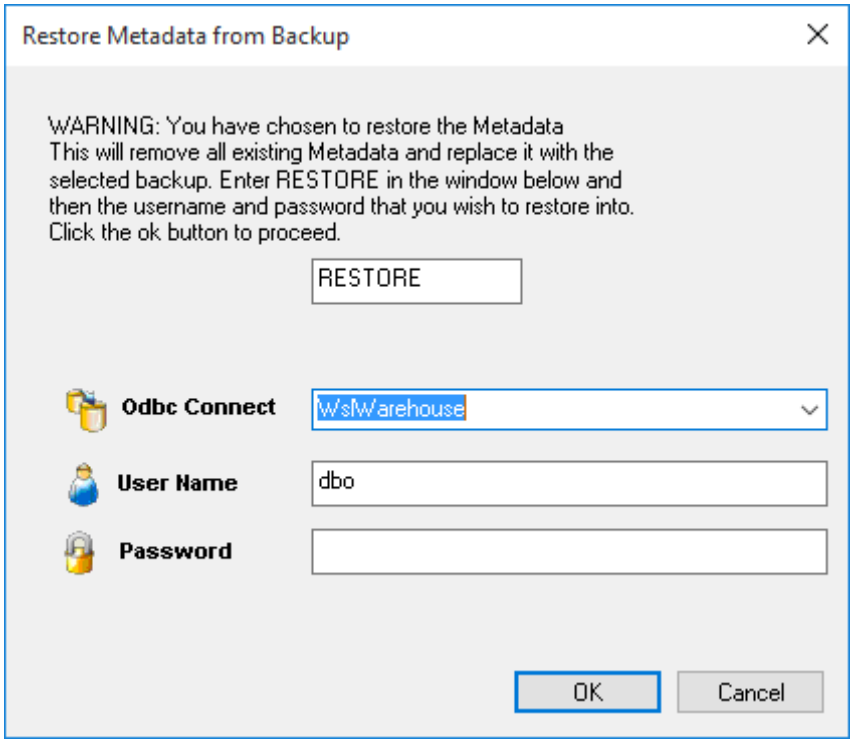
The UNIX restore is actioned via the meta_restore_010.sh shell script. This script expects the export file to be located in the wsl/export directory, so it may need to be un compressed and moved to this directory if not already present.

Execute the meta_restore_010.sh shell script. You will be prompted to enter the backup file name. Once successfully entered a username and password to be restored into are requested and the restore commences.

SQL Server Windows Restore

Select the menu option **Backup/Restore the metadata (from Sql server bcp)** to begin the restore process.

A dialog box will appear. The word **RESTORE** needs to be entered along with the connection, **User Name** and **Password** where the metadata is to be restored to. The metadata will always be restored to the dbo schema. The connection does not have to be that of the current metadata repository, but it must be a valid repository.

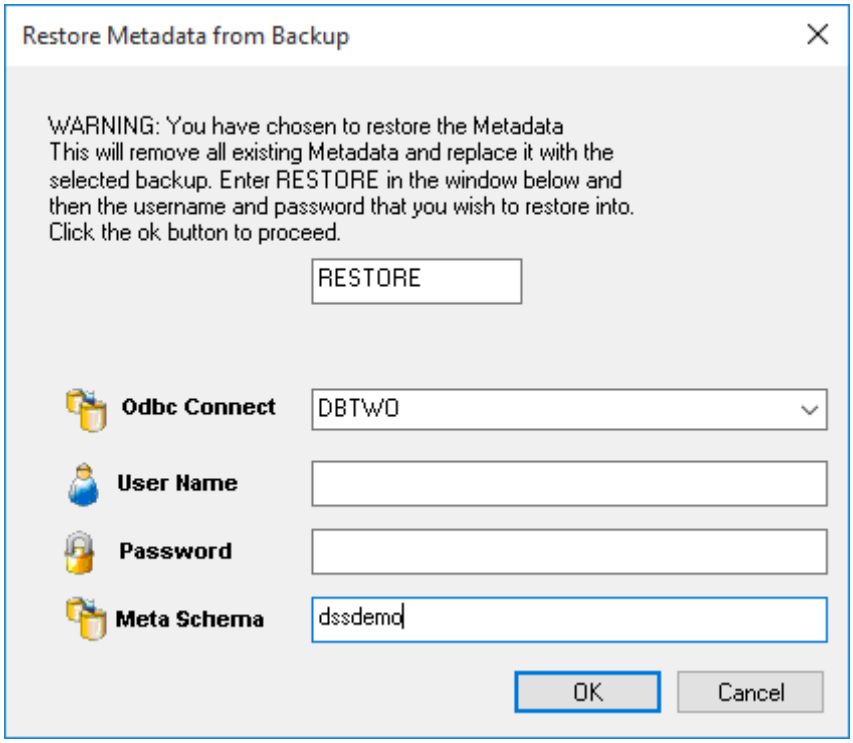


Once the **OK** button is clicked a new dialog box will appear asking for a selection of the export directory. Browse to the contents of the directory where the export is located. Once the export directory is selected the import will begin. A dialog box will appear to show the results of the import.

DB2 Windows Restore

Select the menu option **Backup/Restore metadata (from DB2 export)** to begin the restore process.

A dialog box will appear. The word **RESTORE** needs to be entered with the **User Name** and **Password** for DB2. If using a trusted connection to DB2, the username and password should be left blank. The metadata schema should also be entered. The schema does not have to be that of the current metadata repository, but it must contain a valid repository.



Once the **OK** button is clicked a new dialog box will appear asking for a selection of the export file. Browse to the directory where the export is located and select the export file. Once the export file is selected the import will begin. A dialog box will appear to show the results of the import.

UNLOADING METADATA

WhereScape RED provides a generic unload utility for backing up the metadata. The advantage of this backup is that it is database, and database version independent. It can be used to backup the metadata regardless of the version of the database client running on the PC. It is possible to transport the metadata from one database platform to another using unload and load. For example, the metadata from a SQL Server unload can be loaded into an Oracle database.

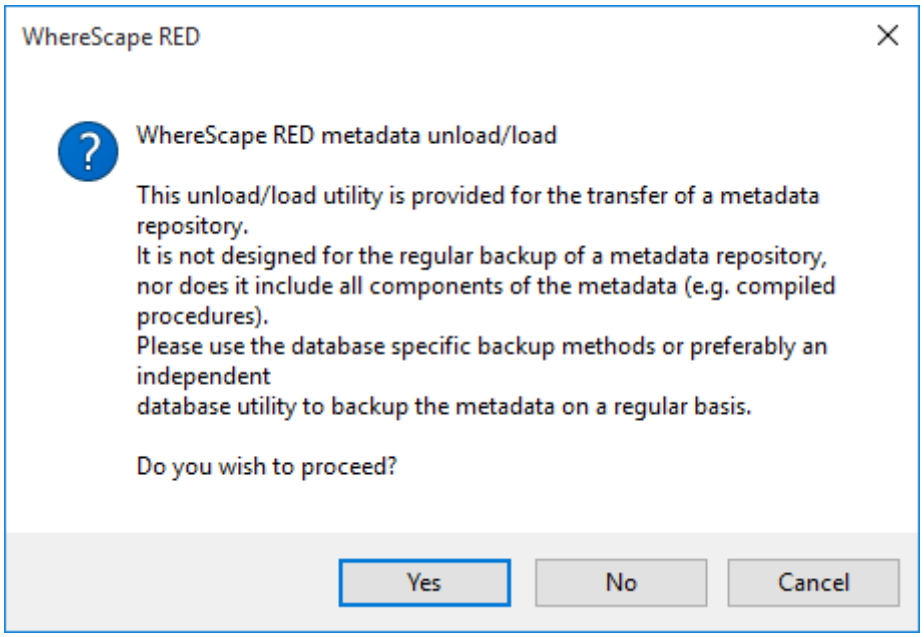
After performing a cross platform unload and load:

- Transformations must be altered manually to use the correct syntax for the new database.
- Generated procedures regenerated.
- Modified and custom procedures must be changed to use the procedure language of the new database platform.

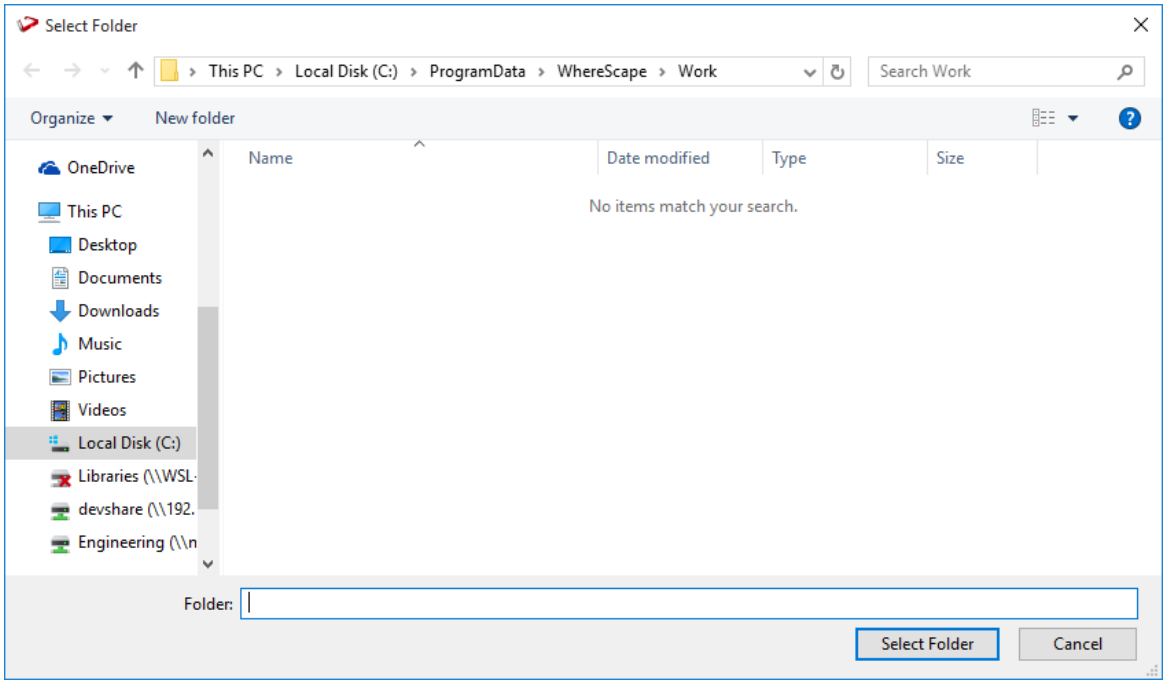
Performing an Unload

An unload can be performed within the WhereScape RED tool by selecting the **Backup/Unload the metadata to disk** menu option.

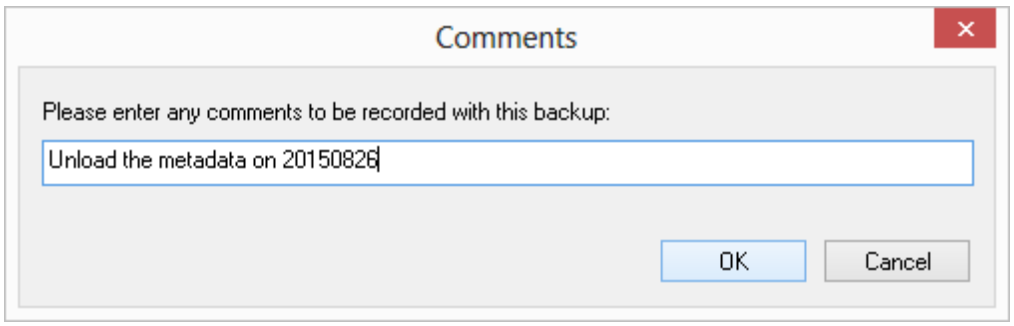
A popup window asks for confirmation to proceed with the unload. Click **Yes**.



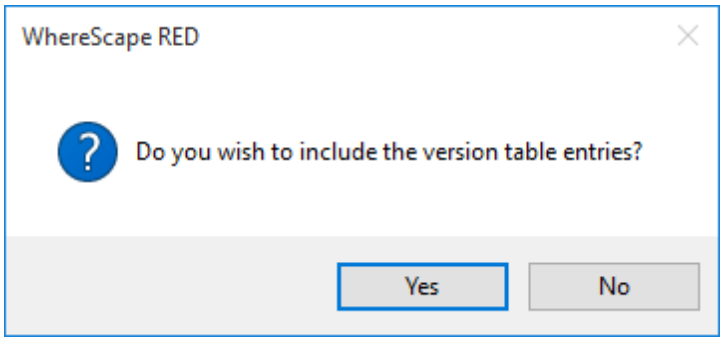
An additional pop-up window asks for the folder for the metadata to be exported to. Select a directory and click **Save**.



Enter a comment for the unload and click **OK**.



Finally, click **Yes** or No on the include version history dialog:



This either includes or excludes version history metadata in the unload.

The unload starts, and indicates progress with a progress bar.

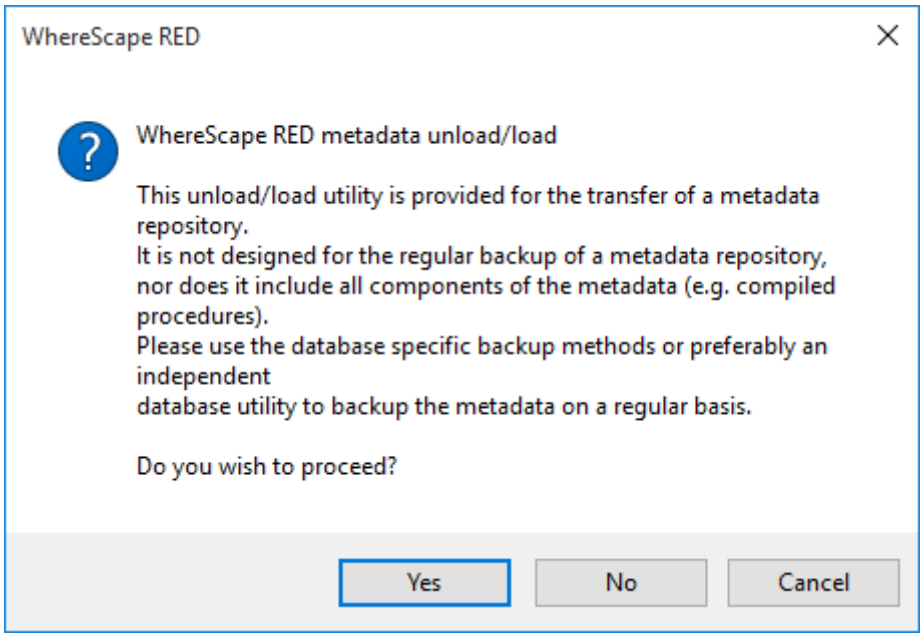
LOADING AN UNLOAD

WhereScape RED metadata can be restored from a prior unload.

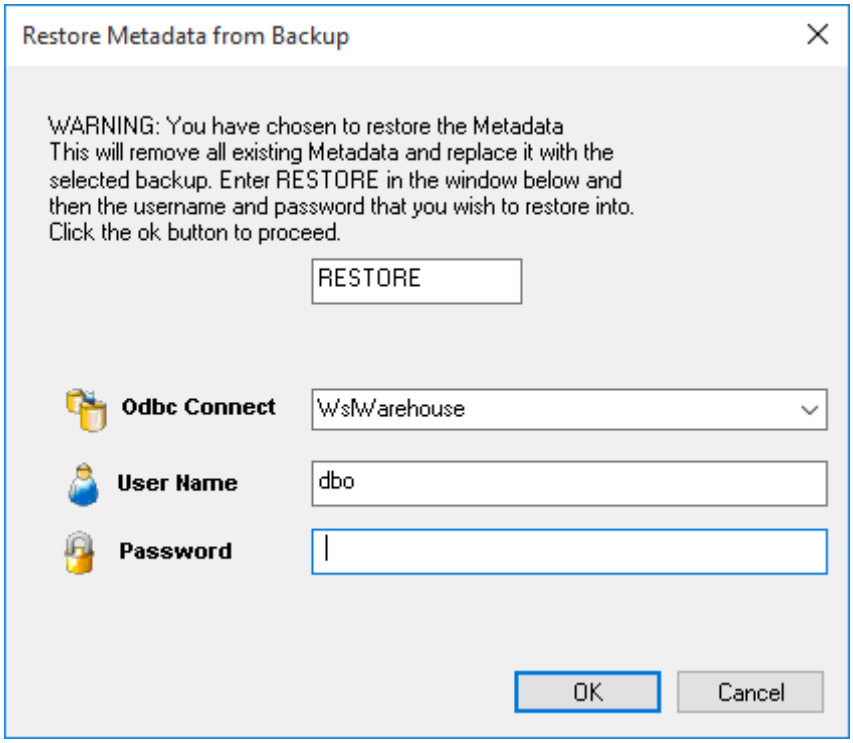
Performing a Load

An unload of a metadata repository can be loaded over the top of an existing repository. This action replaces the existing repository in its entirety. To load an unload, select the menu option **Backup/Load the Metadata from disk** to begin the load process.

A popup window asked for confirmation a load is intended. Click **Yes**.

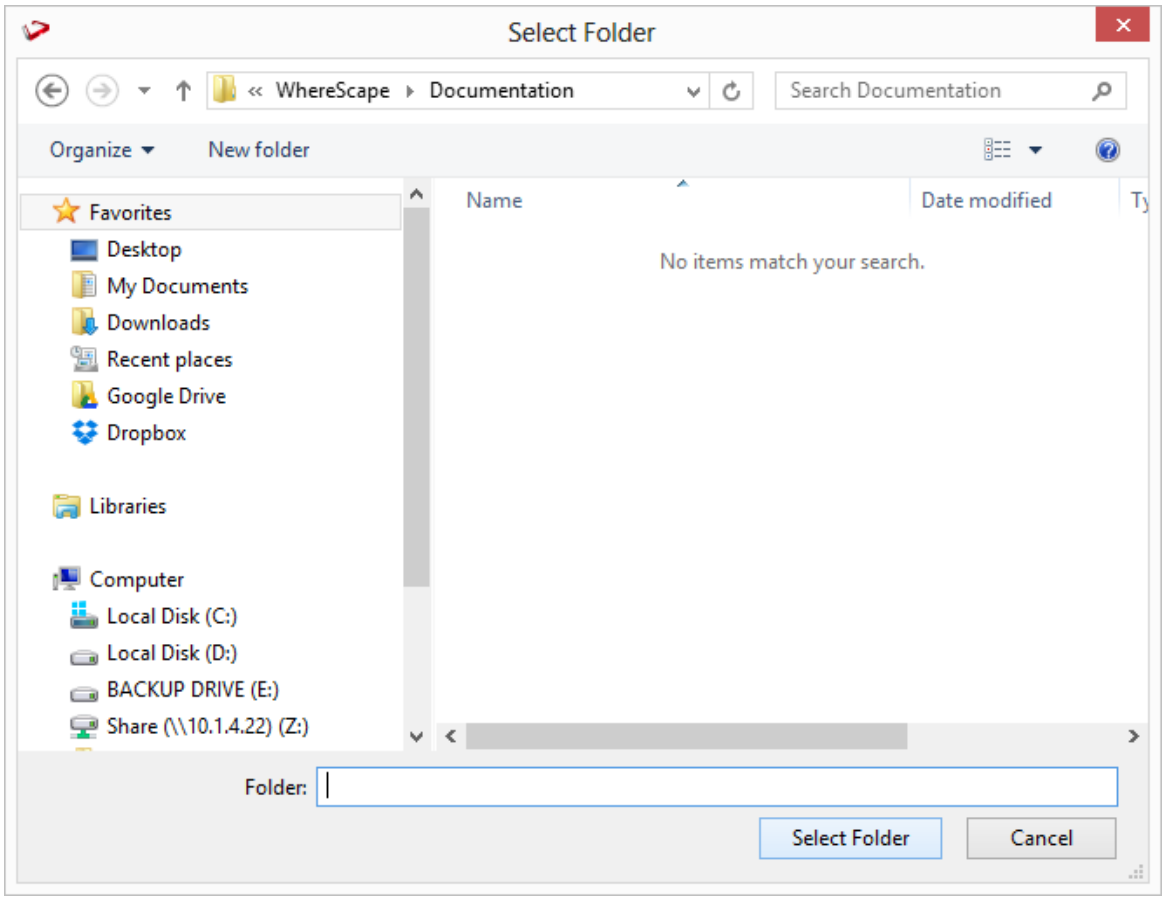


A dialog box appears. The word **RESTORE** needs to be entered. The odbc connection needs to be chosen, along with the **User Name** and **Password** where the metadata is to be restored to. The username does not have to be that of the current metadata repository, but it must be a valid repository. Click **OK**.

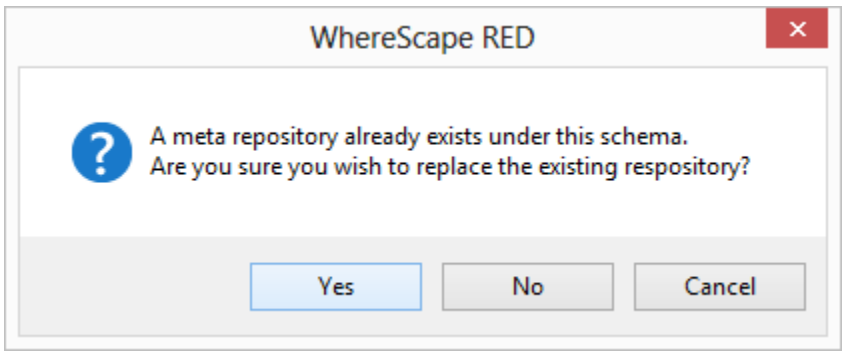


Note: The dialog box above will include an additional field for *Schema* if loading into DB2.

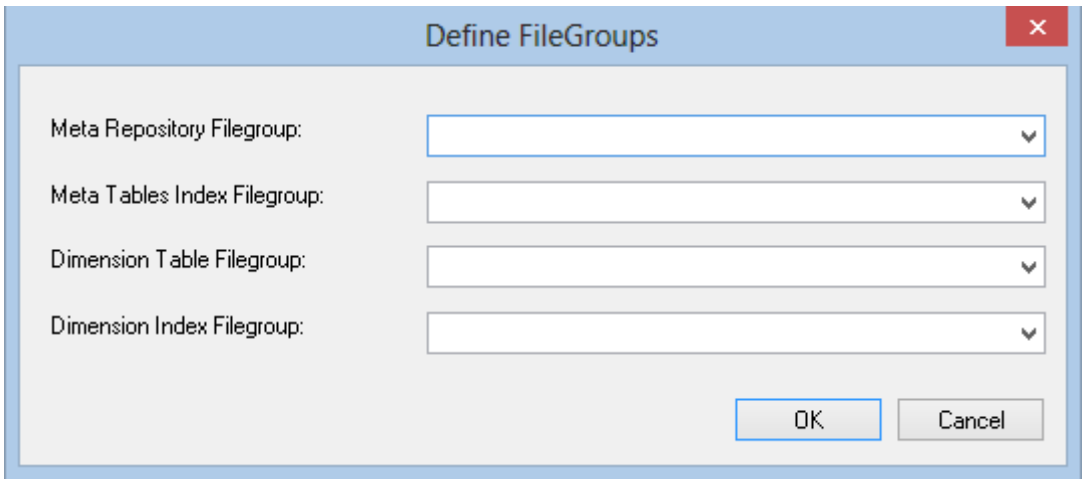
The next dialog box asks for the folder for the metadata to be loaded from. Browse to the contents of the directory where the unload is located and click **Select Folder**.



Confirm the load will overwrite the existing repository by clicking **Yes**:

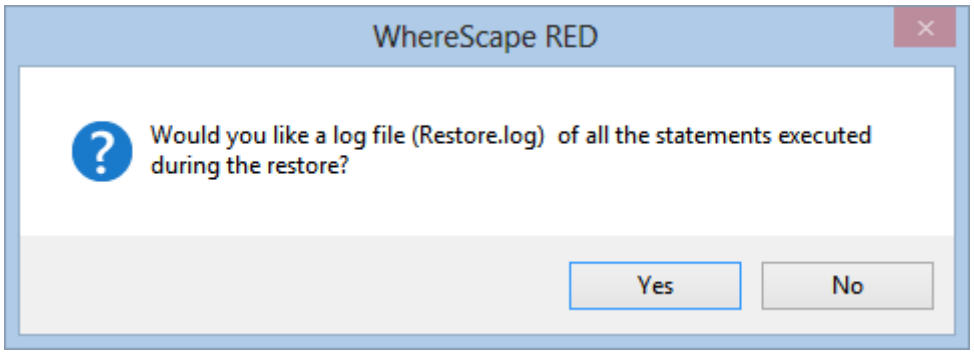


Confirm the metadata and dimension tablespaces/filegroups for the metadata load and click **OK**:

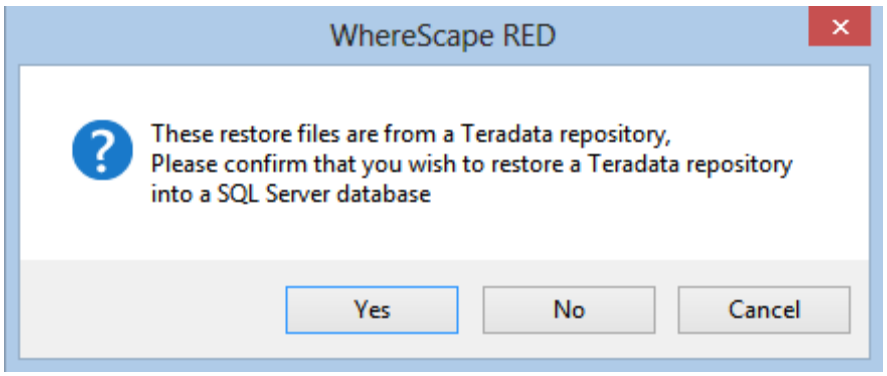


Note: The dialog above was displayed when loading into SQL Server. It may differ slightly if loading into Oracle or DB2.

A pop-up window asks if a restore log is required. Click **Yes** for a log, or No otherwise.



If a cross platform load is being performed, the following dialog is displayed. Click **Yes**.



The metadata load now runs. Once the load has completed, start WhereScape Administrator and validate the metadata repository that has just been loaded.

WhereScape®

CHAPTER 30
ALTERING METADATA

This chapter provides information on how to change and manipulate the data warehouse once it has been established.

New source columns or changes to the source systems from which the data warehouse is built will require modifications to both the metadata and the data warehouse tables and procedures.

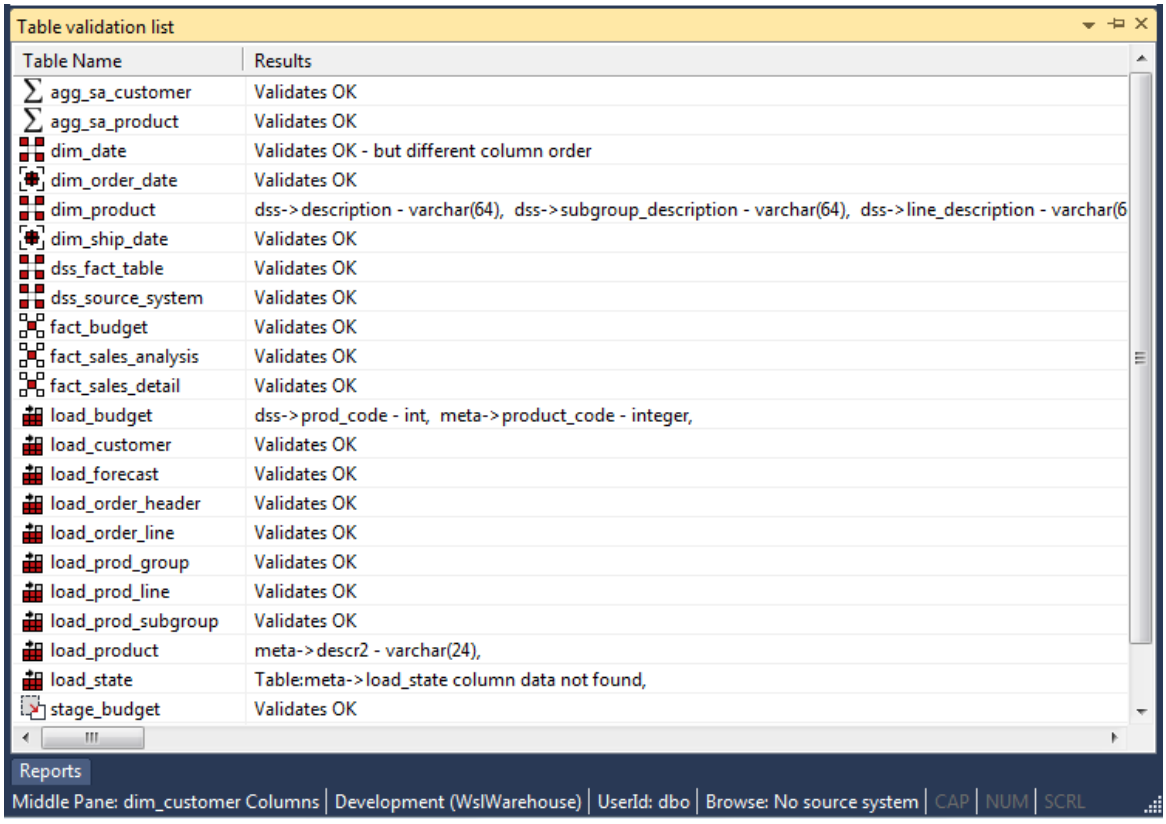
IN THIS CHAPTER

VALIDATING TABLES

The metadata as stored and maintained by WhereScape RED does not necessarily reflect the actual tables and procedures in use in the data warehouse. For example, if a new column is added to the metadata for a table then that change is not automatically made in the actual physical table residing in the data warehouse. Likewise, if a column is deleted from the metadata then that column may still exist in the physical database table.

This situation may be particularly apparent after an application patch or upgrade. The menu option **Validate/Validate Table Create Status**, and the right-click menu options in either the left or middle panes all provide a means of comparing the metadata to the physical tables in the database. A table, range of tables or all tables can be chosen. Each chosen table is a table in the metadata and it is compared against the physical database table if it exists.

The following example is the output from a validation.



In this example we see five different scenarios.

The metadata for table **agg_sa_customer** matches the physical table in the database.

The table **dim_date** has the same columns in both the metadata and the physical table, but the column order is different. This is probably not an issue for most tables, but may be a problem for some type of load tables, where the column order is important. This could be the result of a previous altering of the table. The table must be re-created if the order is important.

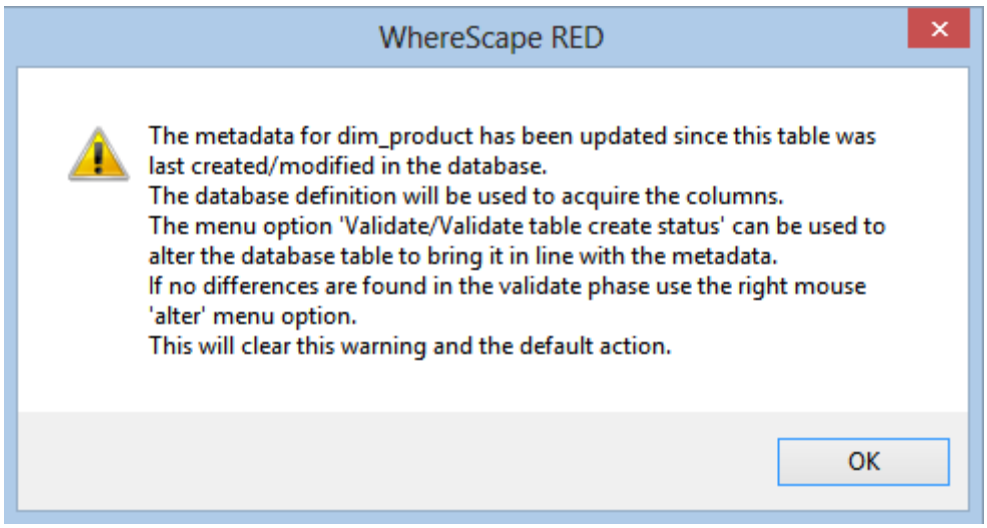
The physical database table **dim_product** has additional columns not found in the metadata. The columns are 'subgroup_description' and 'line_description'. The table can be altered if desired. See the next section on Altering Tables.

The metadata for the table **load_product** does not match the physical table. The metadata has an additional column called 'state'. This column was not found in the physical table. The table can be altered if desired. See the next section on Altering Tables.

The table **load_state** is defined in the metadata but has not been physically created in the database. The table can be created in the normal manner.

Using outdated metadata in drag and drop

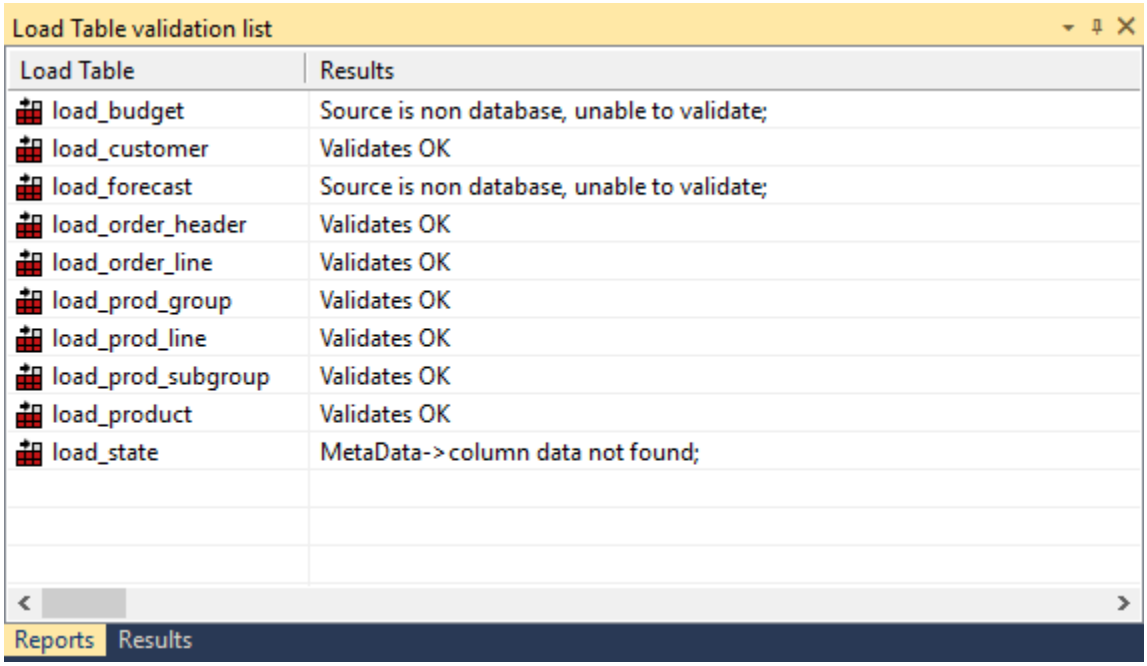
When dragging from a data warehouse table to create another data warehouse table (e.g. load table to create dim table) a check is made to ensure that the metadata matches the database table.
If the two are found to be out of sync the following message will appear:



If a subsequent validate of the table in question shows that it validates, this message will mean that the dates are somehow out of sync. This can occur for example after an import where the metadata has been replaced, but the underlying table still matches the metadata. Another common occurrence is where a new column is added and then deleted. To prevent the message from reoccurring in such a situation proceed as follows. Use the right-click menu and select **Alter Table** when positioned on the table name in the validate results screen (event though the table validates OK). The metadata update time will be set back to that of the last database table create.

VALIDATING SOURCE (LOAD) TABLES

Changes to the source systems from which the data warehouse is built can be detected to a limited degree. The menu option **Validate/Validate Load Table Status** allows a comparison between load tables and the source tables from which they were built. This comparison is not available for flat file or script based loads. A load table or group of load tables are selected and the results are displayed in the middle pane. An example screen from a load table validate is as follows:



The tables **load_budget** and **load_forecast** are Windows file loads and as such cannot be validated.

If a table shows additional columns in the source table; such a scenario will not cause problems for the continued operation of the data warehouse. It simply means that more columns are present in the source table than have been loaded into the data warehouse. This may have been the result of an initial partial selection or as a result of new columns. Further investigation of the source table would be required to ascertain if there was new information available.

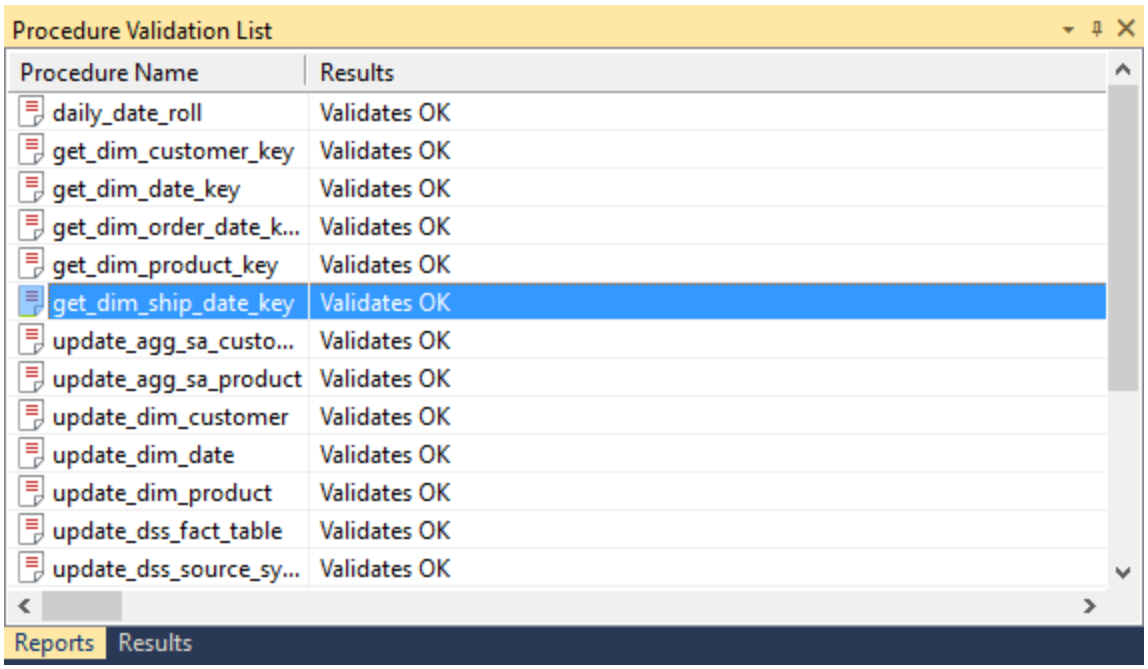
The table **load_state** reflects a problem for the continued operation of the data warehouse. The source table does not have a column that was previously identified as having come from that table. This will probably cause the load of that table to fail. This scenario would also require an investigation into the source table. The resolution may be to delete the column. The potential impact on later tables (dimension, stage and fact) and procedures in the data warehouse can be ascertained by using the right-click menu when positioned over a load table name.

VALIDATING PROCEDURES

The menu option **Validate/Validate Procedure Status** compares procedures as stored in the metadata with those compiled and running in the data warehouse. This option provides a listing in the middle pane of each selected procedure and its status. The status will be **Validates OK**, **Not compiled**, or **Compare failed**.

Where a procedure is marked as **Not compiled** this means that the procedure exists in the metadata but has not been compiled into the database.

Where a procedure fails to compare, the Procedure Editor must be used to find the actual differences. Start the editor by double-clicking on the procedure name in the left pane. Use the **Tools/Compare to Compiled Source** menu option to display the differences between the procedure in the metadata and that compiled in the database.

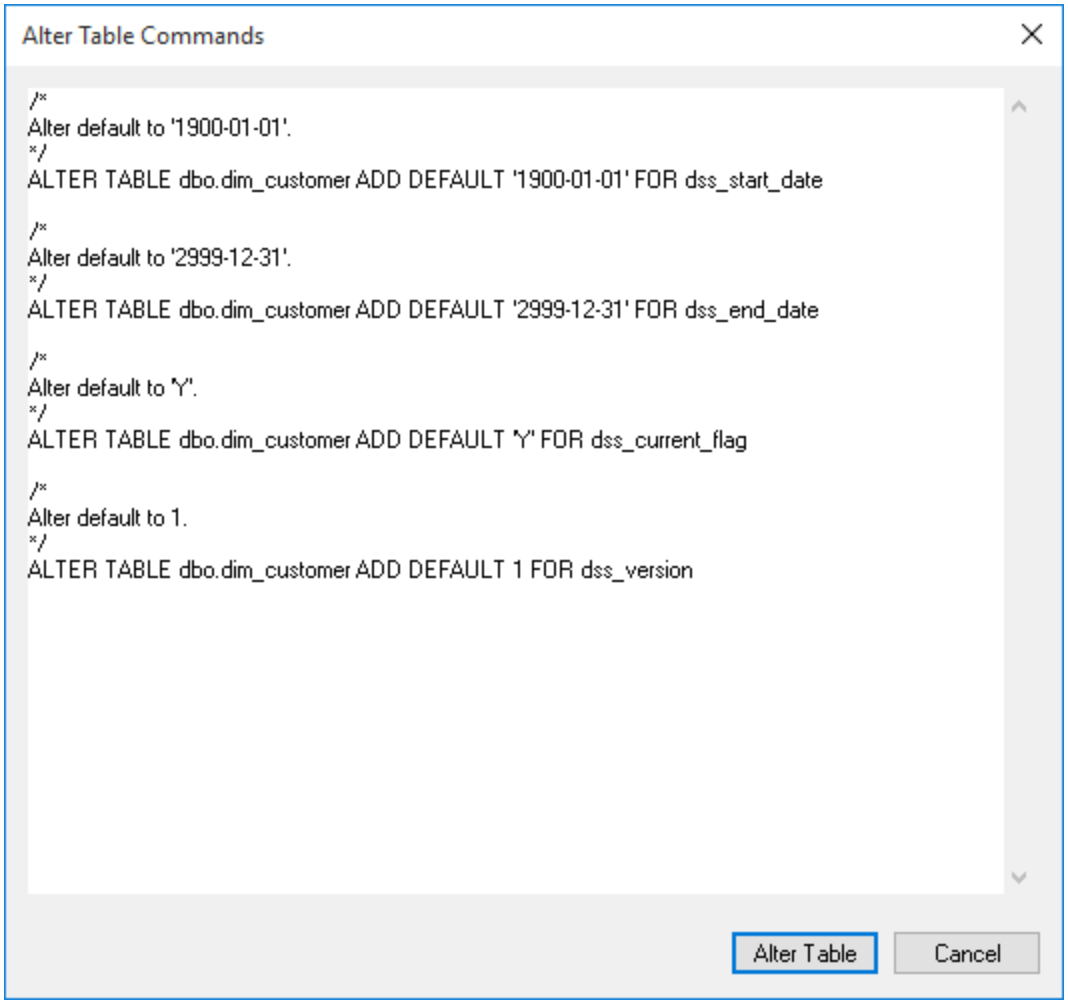


ALTERING TABLES

The previous section covered the process of validating a table as defined in the metadata with the physical table as defined in the database. If a table is found to be different it is possible to alter the table.

Note: Care should be taken when altering large data warehouse tables. A number of factors such as the time required to perform the alter, access to the table and the optimum storage of the table come into play.

To alter a table, first validate the table through the **Validate/Validate Table Create Status** menu option, or use the right-click menu option from the object name. Then in the middle pane (the validation listing) right-click on the table that has not validated and select **Alter table**. A screen similar to the one below will appear advising of the planned changes.



In this example the **dim_product** table is to be altered. Comments at the top of the screen shown the reason(s) for the alteration and the actual alter command(s) follow.

The alter table window is an edit window. The command to be executed can be changed or additional commands entered. The command may also be cut to be executed in some other environment or at some later stage.

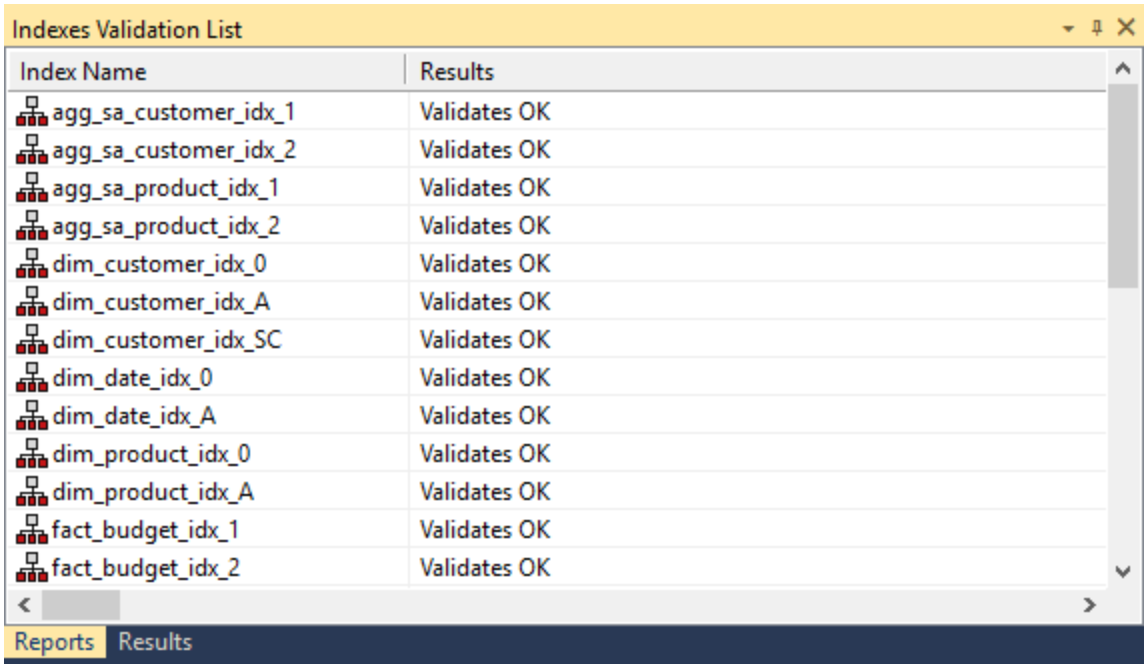
Pressing the **Alter Table** button will proceed to alter the table. In effect, it will execute any command in the window.

VALIDATING INDEXES

The menu option **Validate/Validate Index Status** checks the metadata definition of an index against that in use in the database. It checks to ensure that the index exists and that the index type and columns are the same.

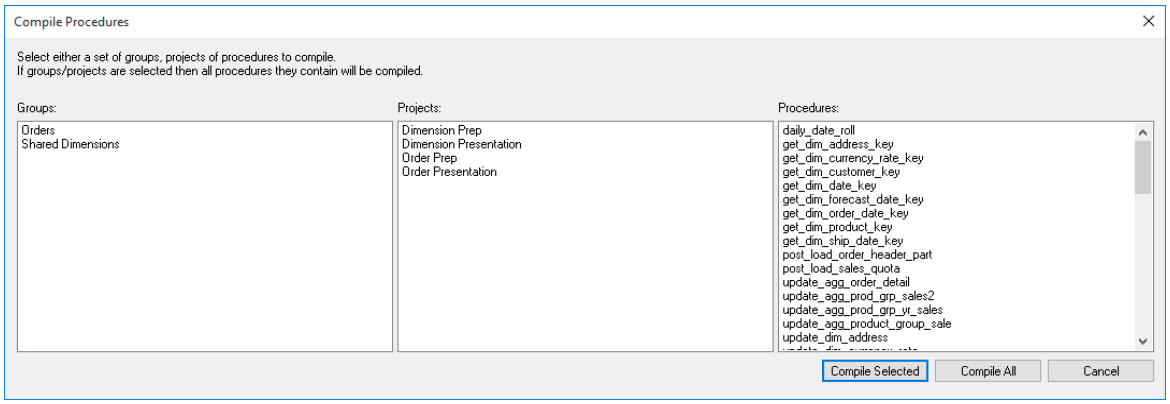
The results are listed in the middle pane with the status of each selected index shown.

If an index differs, then the only option is to drop and create the index either via the scheduler or through the right-click menu options for the index.



RECOMPILING PROCEDURES

Procedures can be invalidated as a result of changes to underlying tables, or child procedures. A procedure can be recompiled through the procedure editor or via the menu option **Tools/Compile Procedures**.
This menu option will load the following dialog box:



All procedures in the metadata can be compiled, or a selected group of procedures may be compiled. Selection is done via standard Windows multi selection using the shift and CTRL keys. By selecting a project or group it is possible to compile all procedures associated with the project or group.

As each procedure is compiled it is displayed in the middle pane of the builder window. If a procedure fails to compile a failure message is displayed along side the procedure name. Procedures that fail to compile will need to be investigated through the procedure editor, as no specific error information is provided by this bulk compiler.

Re-validate Procedures (Oracle only)

The menu option **Tools/Re-compile Invalid Procedures** will attempt to validate the procedure by issuing an 'alter procedure xxx compile' command. This command will be executed for any procedure or function that is in an invalid state in the schema. Any procedures or functions that are still invalid after the commands have been completed are shown in the results dialog. WhereScape RED does not drop procedures from the database even when their metadata is deleted. Therefore, there may be old procedures present that are in an invalid state. These procedures may be redundant.

WhereScape®

CHAPTER 31
CALLABLE ROUTINES

IN THIS CHAPTER

INTRODUCTION TO CALLABLE ROUTINES

CALLABLE ROUTINES API

WhereScape RED callable routines provide an Application Program Interface (API) to the WhereScape RED metadata using the following SQL-invoked routines:

Routine Name	Description
Ws_Api_Glossary (on page 965)	Adds an entry to the documentation glossary.
Ws_Connect_Replace (on page 969)	Replaces the contents of a connection with details from another connection.
Ws_Job_Abort (on page 974)	Aborts a job if it is in a running state.
Ws_Job_Clear_Archive (on page 977)	Purges archived job logs that are older than the specified age in days.
Ws_Job_Clear_Logs (on page 983)	Archives job logs when the maximum number of current logs to retain is exceeded.
Ws_Job_Clear_Logs_By_Date (on page 988)	Archives job logs that are older than the specified age in days.
Ws_Job_Create (on page 993)	Creates a job based on an existing job and optionally starts it immediately.
Ws_Job_CreateWait (on page 1002)	Creates a job based on an existing job and schedules it to start later.
Ws_Job_Dependency (on page 1011)	Adds or removes a child-to-parent dependency between two jobs to control the child job.
Ws_Job_Release (on page 1017)	Starts a job if it is a holding or waiting state.
Ws_Job_Restart (on page 1022)	Starts a job if it is in a failed state.
Ws_Job_Schedule (on page 1027)	Schedules a job if it is in a holding or waiting state.
Ws_Job_Status (on page 1032)	Returns the current status of a job.
Ws_Load_Change (on page 1041)	Changes the Connection or Schema of a load table.
Ws_Maintain_Indexes (on page 1046)	Drops and/or builds database indexes that are defines in the WhereScape RED metadata.
Ws_Version_Clear (on page 1052)	Purges metadata versions for all objects that do not meet the specified retention criteria.

Routine Name	Description
WsParameterRead (see "***WsParameterReadF***" on page 1060)	Returns the value and comment (for most RDBMS) of a WhereScape RED metadata Parameter.
WsParameterReadF (on page 1060)	Returns the value of a WhereScape RED metadata Parameter. **[SQL Server only]**
WsParameterReadG (on page 1063)	Returns the value of a "global" WhereScape RED metadata Parameter that relates to a load table.
WsParameterWrite (on page 1068)	Updates the value and comment of a WhereScape RED metadata Parameter or creates it.
WsWrkAudit (on page 1071)	Records a message in the Audit Log.
WsWrkAuditBulk (on page 1077)	Records multiple messages in the Audit Log.
WsWrkError (on page 1083)	Records a message in the Error/Detail Log.
WsWrkErrorBulk (on page 1089)	Records multiple messages in the Error/Detail Log.
WsWrkTask (on page 1095)	Updates row counts for a task in the Task Log.

CALLABLE ROUTINES PER RDBMS

Each WhereScape RED Callable Routine exists in the database as either a Stored Procedure or a User-defined Function that can be invoked from SQL. A Callable Procedure is invoked by a SQL call/execute statement and a Callable Function is invoked in a SQL SELECT statement or a value expression. The Callable Routines are typically implemented as **procedures in most database systems** due to RDBMS limitations of user-defined functions when the WhereScape RED API was originally developed.

Routine Name	SQLServer	Teradata	Oracle	DB2
Ws_Api_Glossary (on page 965)	Procedure	Procedure	Function	Procedure
Ws_Connect_Replace (on page 969)	Procedure	Procedure	Function	Procedure
Ws_Job_Abort (on page 974)	Procedure	Procedure	Function	Procedure
Ws_Job_Clear_Archive (on page 977)	Procedure	Procedure	Function	Procedure
Ws_Job_Clear_Logs (on page 983)	Procedure	Procedure	Function	Procedure

Routine Name	SQLServer	Teradata	Oracle	DB2
Ws_Job_Clear_Logs_By_Date (on page 988)	Procedure	Procedure	Function	Procedure
Ws_Job_Create (on page 993)	Procedure	Procedure	Function	Procedure
Ws_Job_CreateWait (on page 1002)	Procedure	Procedure	Function	Procedure
Ws_Job_Dependency (on page 1011)	Procedure	Procedure	Function	Procedure
Ws_Job_Release (on page 1017)	Procedure	Procedure	Function	Procedure
Ws_Job_Restart (on page 1022)	Procedure	Procedure	Function	Procedure
Ws_Job_Schedule (on page 1027)	Procedure	Procedure	Function	Procedure
Ws_Job_Status (on page 1032)	Procedure	Procedure	Function	Procedure
Ws_Load_Change (on page 1041)	Procedure	Procedure	Function	Procedure
Ws_Maintain_Indexes (on page 1046)	Procedure	Procedure	Function	Procedure
Ws_Version_Clear (on page 1052)	Procedure	Procedure	Function	Procedure
WsParameterRead (see "***WsParameterReadF***" on page 1060)	Procedure	Procedure	Function	Procedure
WsParameterReadF (on page 1060)	Procedure	Procedure	Function	Procedure
WsParameterReadG (on page 1063)	Procedure	Procedure	Function	Procedure
WsParameterWrite (on page 1068)	Procedure	Procedure	Function	Procedure
WsWrkAudit (on page 1071)	Procedure	Procedure	Function	Procedure
WsWrkAuditBulk (on page 1077)	Procedure	Procedure	Function	Procedure
WsWrkError (on page 1083)	Procedure	Procedure	Function	Procedure
WsWrkErrorBulk (on page 1089)	Procedure	Procedure	Function	Procedure
WsWrkTask (on page 1095)	Procedure	Procedure	Function	Procedure

CALLABLE ROUTINES NAMES QUALIFIER

The WhereScape RED Callable Routines can be invoked using the unqualified routine name for SQL Server and Oracle. However, for Teradata and DB2 it is necessary to qualify the routine name with the owner/schema of the WhereScape RED metadata repository. All RED-generated procedures in a Teradata or DB2 repository that invoke a WhereScape RED Callable Routine do so by qualifying the routine name with **[METABASE]** e.g. **[METABASE]**.routine_name. When RED creates/compiles a procedure in a Teradata or DB2 database it automatically replaces the **[METABASE]** "token" with the repository owner/schema.

When you edit a RED-generated procedure or create your own custom procedure in Teradata or DB2 you can qualify each callable routine name by either hard-coding the owner/schema or by using the **[METABASE]** "token" that RED will replace when the procedure is created/compiled in the database. The Teradata and DB2 examples in this chapter use the **[METABASE]** "token" but if you invoke a WhereScape RED Callable Routine interactively or from outside a RED-compiled procedure/function then the actual owner/schema must be specified because RED won't have the chance to replace the "token".

CALLABLE ROUTINES COMMON INPUT

The following input parameters are common to most of the WhereScape RED Callable Routines, which are primarily used for integration with the WhereScape RED Scheduler.

Input	Parameter Name	Description
Job Instance Identifier	p_sequence	Unique identifier of the **running job** (i.e. the running instance of a held or scheduled job) that executed the routine. When invoked from a WhereScape RED Scheduler the routine will be passed the parameter argument. When invoked manually or externally to the WhereScape RED Scheduler, then any integer value can be used.
Job Name	p_job_name	Name of the **running job** that executed the routine. When invoked from a WhereScape RED Scheduler the routine will be passed the parameter argument. When invoked manually or externally to the WhereScape RED Scheduler, then any name can be used.

Input	Parameter Name	Description
Task Name	p_task_name	Name of the **running task** (of a running job) that executed the routine. When invoked from a WhereScape RED Scheduler the routine will be passed the parameter argument. When invoked manually or externally to the WhereScape RED Scheduler, then any name can be used.
Job Identifier	p_job_id	Unique identifier of the **held or scheduled job** that the running job is a specific instance of. When invoked from a WhereScape RED Scheduler the routine will be passed the parameter argument. When invoked manually or externally to the WhereScape RED Scheduler, it is recommended to use 0 (zero).
Task Identifier	p_task_id	Unique identifier of the **running task** (of a running job) that executed the routine. When invoked from a WhereScape RED Scheduler the routine will be passed the parameter argument. When invoked manually or externally to the WhereScape RED Scheduler, it is recommended to use 0 (zero).

Note: Typically, the **parameter names** of the WhereScape RED Callable Routines use a p_ prefix as indicated but in some routines a v_ prefix is used instead. In addition, for SQL Server all parameter names are also prefixed by @. The RDBMS-specific parameter names are included in the subsequent details of each Callable Routine.

CALLABLE ROUTINES INVOCATION

This chapter includes examples of each WhereScape RED Callable Routine to illustrate how to invoke the routine for each RDBMS along with the necessary variable declarations. Typically, the routines are executed from a Stored Procedure so the routine-specific examples illustrate that scenario and use parameters/arguments that are passed by position (as in a RED-generated procedure). However, it can be useful to execute a Callable Routine using other methods such as the following.

ALTERNATIVE INVOCATION METHODS

Invocation using Named Parameters

SQL Server, Oracle, and DB2 support the passing of parameter arguments by **name** as an alternative to passing them by position. Typically, for most RDBMS when one parameter argument is supplied by name then all subsequent parameter arguments must be supplied in the same manner. The order of the parameter names doesn't matter but for clarity the following alternative examples pass the parameter names in the same order that they are declared in the routine. When parameter arguments are not passed by name the values must be supplied in the identical order (left to right) as the parameters are declared in the routine.

SQL Server example of invoking the Ws_Connect_Replace PROCEDURE using named parameters:

```
DECLARE @v_return_code            varchar(1)
DECLARE @v_return_msg             varchar(256)
DECLARE @v_result_num             integer
EXEC Ws_Connect_Replace
  @p_sequence    = 0
, @p_job_name    = 'Test Job Name'
, @p_task_name   = 'Test Task Name'
, @p_job_id      = 0
, @p_task_id     = 0
, @p_action      = 'REPLACE'
, @p_source      = 'Connection1'
, @p_target      = 'Connection2'
, @p_return_code = @v_return_code OUTPUT
, @p_return_msg  = @v_return_msg  OUTPUT
, @p_result      = @v_result_num  OUTPUT
```

Oracle example of invoking the Ws_Connect_Replace FUNCTION using named parameters via an anonymous PL/SQL block:

```
DECLARE
  v_return_code                   varchar2(1);
  v_return_msg                    varchar2(256);
  v_result_num                    integer;
BEGIN
  v_result_num := Ws_Connect_Replace
( p_sequence    => 0
  , p_job_name    => 'Test Job Name'
  , p_task_name   => 'Test Task Name'
  , p_job_id      => 0
  , p_task_id     => 0
  , p_action      => 'REPLACE'
  , p_source      => 'Connection1'
  , p_target      => 'Connection2'
  , p_return_code => v_return_code
  , p_return_msg  => v_return_msg
  );
END;
/
```

DB2 example of invoking the Ws_Connect_Replace PROCEDURE using named parameters via another procedure:

```
-- The owner/schema is DSSDEMO.
DROP PROCEDURE DSSDEMO.invoke_DB2_Ws_Connect_Replace;
CREATE PROCEDURE DSSDEMO.invoke_DB2_Ws_Connect_Replace
  LANGUAGE SQL
BEGIN
  DECLARE v_return_code           varchar(1);
  DECLARE v_return_msg            varchar(256);
  DECLARE v_result_num            integer;
  CALL DSSDEMO.Ws_Connect_Replace
  ( p_sequence    => 0
  , p_job_name    => 'Test Job Name'
  , p_task_name   => 'Test Task Name'
  , p_job_id      => 0
  , p_task_id     => 0
  , p_action      => 'REPLACE'
  , p_source      => 'Connection1'
  , p_target      => 'Connection2'
  , p_return_code => v_return_code
  , p_return_msg  => v_return_msg
  , p_result      => v_result_num
  );
END
/
CALL DSSDEMO.invoke_DB2_Ws_Connect_Replace();
```

Invocation via ODBC

The following examples illustrate how to invoke a WhereScape RED Callable Routine via an ODBC connection (using a tool such as WhereScape SQL Admin), which uses both input and output parameters. The examples work for a SQL Server RED repository but for another RDBMS the routine name may need to be qualified with the appropriate owner/schema. Refer to the detailed description of the Ws_Connect_Replace routine for an explanation of the parameters/arguments.

The following invocations via ODBC are equivalent and the only difference is the formatting as the first command uses a single line while the second command is formatted across multiple lines:

-- **OBDC Example 1 (single line)** .

```
{ CALL Ws_Connect_Replace( 0,'Test Job Name', 'Test Task Name', 0, 0,
'REPLACE', 'Connection1', 'Connection2', ?, ?, ?) };
```

-- **OBDC Example 2 (same command formatted using multiple lines)** .

```
{ CALL Ws_Connect_Replace
  ( 0
  , 'Test Job Name'
  , 'Test Task Name'
  , 0
  , 0
  , 'REPLACE'
  , 'Connection1'
  , 'Connection2'
  , ?
  , ?
  , ?
  )
};
```

The result of the Ws_Connect_Replace invocation can be confirmed by checking the target connection. In addition, a log entry is created using the specified job and task names that can be viewed via the Logs – Recent Audit Trail Logs menu item of the WhereScape RED Scheduler.

Invocation via the Command-Line

Each WhereScape RED Callable Routine can also be invoked from the command-line using an RDBMS-specific tool such as:

RDBMS	SQL Command-Line-Interface (CLI) Tool
SQL Server	SQL Server sqlcmd Utility i.e. sqlcmd.exe.
Teradata	Basic Teradata Query (BTEQ) i.e. bteq.
Oracle	Oracle SQL*Plus i.e. sqlplus.
DB2	DB2 Command Line Processor (CLP) i.e. db2 or db2cmd.

These tools can be used to invoke a WhereScape RED Callable Routine by connecting to the database, executing SQL statements, and disconnecting from the database. Refer to the RDBMS-specific documentation for details of how to connect and execute SQL via the relevant CLI tool as well as details of tool's return codes.

Typically, the CLI command will include options to specify the database, connection credentials, and the SQL to execute. Multiple SQL statements can typically be executed by terminating/delimiting each statement using a semi-colon (;). In most cases, the SQL needs to be quoted (typically double-quoted) when specified on the command-line and if the SQL statements include embedded quotes (such as single-quotes around literals) then they may need to be "escaped" depending on the CLI tool and the platform. Most of the tools also allow the SQL commands to be read from a file instead of from standard input.

WS_API_GLOSSARY

Synopsis

Adds an entry to the documentation glossary.

Description

Adds the specified entry to the documentation glossary, which is included in documentation that is subsequently generated by WhereScape RED.

Input

Input	Description
Object Name	Item that appears in the left column of the glossary, which normally represents the business name of a column in a dimension or fact table (although it can be used for other purposes).
Glossary Term	Item that appears under the analysis area heading of the glossary, which normally represents a dimension ot fact table name (although it can be used for other purposes).
Glossary Description	Description of the term being defined.
Action	Either **ADD** or **DELETE** the specified glossary entry.

Output

Output	Description
Result Text	A message to indicate whether the glossary term was successfully added/deleted.

SQL SERVER

SQL Server Parameters: Ws_Api_Glossary

Callable Routine Type: PROCEDURE.

Parameter Name	Datatype	Mode
@p_object_name	VARCHAR(64)	IN
@p_term	VARCHAR(256)	IN
@p_comments	VARCHAR(4000)	IN
@p_option	VARCHAR(12)	IN
@p_result	VARCHAR(256)	OUT

SQL Server Examples: Ws_Api_Glossary

```
DECLARE @v_result_txt           varchar(256)
EXEC Ws_Api_Glossary
  'Data Warehouse'
, 'Overview'
, 'A repository of business information'
, 'ADD'
, @v_result_txt OUTPUT
```

TERADATA

Teradata Parameters: Ws_Api_Glossary

Callable Routine Type: PROCEDURE.

Parameter Name	Datatype	Mode
p_object_name	VARCHAR(64)	IN
p_term	VARCHAR(256)	IN
p_comments	VARCHAR(4000)	IN
p_option	VARCHAR(64)	IN
p_result	VARCHAR(256)	OUT

Teradata Examples: Ws_Api_Glossary

```
DECLARE v_result_txt            varchar(256);
CALL [METABASE].Ws_Api_Glossary
( 'Data Warehouse'
, 'Overview'
, 'A repository of business information'
, 'ADD'
, v_result_txt
);
```

ORACLE

Oracle Parameters: Ws_Api_Glossary

Callable Routine Type: FUNCTION.

Parameter Name	Datatype	Mode
v_object_name	VARCHAR2	IN
v_term	VARCHAR2	IN
v_comments	VARCHAR2	IN
v_option	VARCHAR2	IN
FUNCTION Return Value	VARCHAR2	OUT-Function

Oracle Examples: Ws_Api_Glossary

```
v_result_txt                    varchar2(256);
v_result_txt := Ws_Api_Glossary
( 'Data Warehouse'
, 'Overview'
, 'A repository of business information'
, 'ADD'
);
```

DB2

DB2 Parameters: Ws_Api_Glossary

Callable Routine Type: PROCEDURE.

Parameter Name	Datatype	Mode
p_object_name	VARCHAR(64)	IN
p_term	VARCHAR(256)	IN
p_comments	VARCHAR(4000)	IN
p_option	VARCHAR(64)	IN
p_result	VARCHAR(256)	OUT

DB2 Examples: Ws_Api_Glossary

```
DECLARE v_result_txt            varchar(256);
CALL [METABASE].Ws_Api_Glossary
( 'Data Warehouse'
, 'Overview'
, 'A repository of business information'
, 'ADD'
, v_result_txt
);
```

WS_CONNECT_REPLACE

Synopsis

Replaces the contents of a connection with the details from another connection.

Description

Copies the details of the specified Source connection to the specified Target connection.

Input

Input	Description
Common Input	Includes all 5 inputs of the ***Callable Routines Common Input*** (on page 959).
Action	**REPLACE** the details of the specified Target connection.
Source Connection	The name of the Source connection whose details will be copied.
Target Connection	The name of the Target connection whose details will be changed.

Output

Output	Description
Return Code	Output Return Code: S Success. E Error. F Fatal/Unexpected Error.
Return Message	Output message indicating the action applied or the reason for no action.
Result Number	Output Result Number: 1 Success. -2 Error. -3 Fatal/Unexpected Error.

SQL SERVER

SQL Server Parameters: Ws_Connect_Replace

Callable Routine Type: PROCEDURE.

Parameter Name	Datatype	Mode
@p_sequence	INTEGER	IN
@p_job_name	VARCHAR(64)	IN
@p_task_name	VARCHAR(64)	IN
@p_job_id	INTEGER	IN
@p_task_id	INTEGER	IN
@p_action	VARCHAR(64)	IN
@p_source	VARCHAR(64)	IN
@p_target	VARCHAR(64)	IN
@p_return_code	VARCHAR(1)	OUT
@p_return_msg	VARCHAR(256)	OUT
@p_result	INTEGER	OUT

SQL Server Examples: Ws_Connect_Replace

```
-- The p_ VARIABLES are normally PARAMETERS in a RED-generated Procedure.
DECLARE @p_sequence             integer
DECLARE @p_job_name             varchar(256)
DECLARE @p_task_name            varchar(256)
DECLARE @p_job_id               integer
DECLARE @p_task_id              integer
DECLARE @p_return_msg           varchar(256)
DECLARE @p_status               integer
DECLARE @v_result_num           integer
DECLARE @v_return_code          varchar(1)
DECLARE @v_return_msg           varchar(256)
EXEC Ws_Connect_Replace
  @p_sequence, @p_job_name, @p_task_name, @p_job_id, @p_task_id
, 'REPLACE', 'Connection1', 'Connection2'
, @v_return_code OUTPUT
, @v_return_msg  OUTPUT
, @v_result_num  OUTPUT
```

TERADATA

Teradata Parameters: Ws_Connect_Replace

Callable Routine Type: PROCEDURE.

Parameter Name	Datatype	Mode
p_sequence	INTEGER	IN
p_job_name	VARCHAR(64)	IN
p_task_name	VARCHAR(64)	IN
p_job_id	INTEGER	IN
p_task_id	INTEGER	IN
p_action	VARCHAR(64)	IN
p_source	VARCHAR(64)	IN
p_target	VARCHAR(64)	IN
p_return_code	VARCHAR(1)	OUT
p_return_msg	VARCHAR(256)	OUT
p_result	INTEGER	OUT

Teradata Examples: Ws_Connect_Replace

```
-- The p_ VARIABLES are normally PARAMETERS in a RED-generated Procedure.
DECLARE p_sequence              integer;
DECLARE p_job_name              varchar(256);
DECLARE p_task_name             varchar(256);
DECLARE p_job_id                integer;
DECLARE p_task_id               integer;
DECLARE p_return_msg            varchar(256);
DECLARE p_status                integer;
DECLARE v_result_num            integer;
DECLARE v_return_code           varchar(1);
DECLARE v_return_msg            varchar(256);
CALL [METABASE].Ws_Connect_Replace
( p_sequence, p_job_name, p_task_name, p_job_id, p_task_id
, 'REPLACE', 'Connection1', 'Connection2'
, v_return_code
, v_return_msg
, v_result_num
);
```

ORACLE

Oracle Parameters: Ws_Connect_Replace

Callable Routine Type: FUNCTION.

Parameter Name	Datatype	Mode
p_sequence	NUMBER	IN
p_job_name	VARCHAR2	IN
p_task_name	VARCHAR2	IN
p_job_id	NUMBER	IN
p_task_id	NUMBER	IN
p_action	VARCHAR2	IN
p_source	VARCHAR2	IN
p_target	VARCHAR2	IN
p_return_code	VARCHAR2	OUT
p_return_msg	VARCHAR2	OUT
FUNCTION Return Value	NUMBER	OUT-Function

Oracle Examples: Ws_Connect_Replace

```
-- The p_ VARIABLES are normally PARAMETERS in a RED-generated Procedure.
p_sequence                      number;
p_job_name                      varchar2(256);
p_task_name                     varchar2(256);
p_job_id                        number;
p_task_id                       number;
p_return_msg                    varchar2(256);
p_status                        number;
v_result_num                    number;
v_return_code                   varchar2(1);
v_return_msg                    varchar2(256);
v_result_num := Ws_Connect_Replace
( p_sequence, p_job_name, p_task_name, p_job_id, p_task_id
, 'REPLACE', 'Connection1', 'Connection2'
, v_return_code
, v_return_msg
);
```

DB2

DB2 Parameters: Ws_Connect_Replace

Callable Routine Type: PROCEDURE.

Parameter Name	Datatype	Mode
p_sequence	INTEGER	IN
p_job_name	VARCHAR(64)	IN
p_task_name	VARCHAR(64)	IN
p_job_id	INTEGER	IN
p_task_id	INTEGER	IN
p_action	VARCHAR(64)	IN
p_source	VARCHAR(64)	IN
p_target	VARCHAR(64)	IN
p_return_code	VARCHAR(1)	OUT
p_return_msg	VARCHAR(256)	OUT
p_result	INTEGER	OUT

DB2 Examples: Ws_Connect_Replace

```
-- The p_ VARIABLES are normally PARAMETERS in a RED-generated Procedure.
DECLARE p_sequence              integer;
DECLARE p_job_name              varchar(256);
DECLARE p_task_name             varchar(256);
DECLARE p_job_id                integer;
DECLARE p_task_id               integer;
DECLARE p_return_msg            varchar(256);
DECLARE p_status                integer;
DECLARE v_result_num            integer;
DECLARE v_return_code           varchar(1);
DECLARE v_return_msg            varchar(256);
CALL [METABASE].Ws_Connect_Replace
( p_sequence, p_job_name, p_task_name, p_job_id, p_task_id
, 'REPLACE', 'Connection1', 'Connection2'
, v_return_code
, v_return_msg
, v_result_num
);
```

WS_JOB_ABORT

Synopsis

Aborts a job if it is in a running state.

Description

Aborts the specified job if it is in a running state, which changes it to a failed state, fails all running tasks, and holds all waiting tasks.

Input

Input	Description
Abort Job Name	The name of the job to be aborted. The specified name must exactly match the job name as displayed by the WhereScape RED Scheduler. The specified job must be in a running or failed state in order to be aborted.
Job Instance Identifier	Unique identifier of the running job (that may be in a failed state).
Abort Job Message Text	Custom message text to be recorded in the WhereScape RED Audit Log for the aborted job and the WhereScape RED Task Log for each aborted task.

Output

Output	Description
Result Number **[ORACLE only]**	Output Result Number **[ORACLE only]**: 1 Success. -2 Error. e.g. Due to invalid job name or job not running. -3 Fatal/Unexpected Error.

SQL SERVER

SQL Server Parameters: Ws_Job_Abort

Callable Routine Type: PROCEDURE.

Parameter Name	Datatype	Mode
@p_job_name	VARCHAR(64)	IN
@p_job_sequence	INTEGER	IN
@p_job_msg	VARCHAR(256)	IN

SQL Server Examples: Ws_Job_Abort

```
EXEC Ws_Job_Abort
   'Daily Run'
, 1234
, 'Job aborted via manual execution of Ws_Job_Abort.'
```

TERADATA

Teradata Parameters: Ws_Job_Abort

Callable Routine Type: PROCEDURE.

Parameter Name	Datatype	Mode
p_job_name	VARCHAR(64)	IN
p_job_sequence	INTEGER	IN
p_job_msg	VARCHAR(256)	IN

Teradata Examples: Ws_Job_Abort

```
CALL [METABASE].Ws_Job_Abort
( 'Daily Run'
, 1234
, 'Job aborted via manual execution of Ws_Job_Abort.'
);
```

ORACLE

Oracle Parameters: Ws_Job_Abort

Callable Routine Type: FUNCTION.

Parameter Name	Datatype	Mode
v_job_name	VARCHAR2	IN
v_job_sequence	NUMBER	IN
v_job_msg	VARCHAR2	IN
FUNCTION Return Value	NUMBER	OUT-Function

Oracle Examples: Ws_Job_Abort

```
v_result_num                    number;
v_result_num := Ws_Job_Abort
( 'Daily Run'
, 1234
, 'Job aborted via manual execution of Ws_Job_Abort.'
);
```

DB2

DB2 Parameters: Ws_Job_Abort

Callable Routine Type: PROCEDURE.

Parameter Name	Datatype	Mode
p_job_name	VARCHAR(64)	IN
p_job_sequence	INTEGER	IN
p_job_msg	VARCHAR(256)	IN

DB2 Examples: Ws_Job_Abort

```
CALL [METABASE].Ws_Job_Abort
( 'Daily Run'
, 1234
, 'Job aborted via manual execution of Ws_Job_Abort.'
);
```

WS_JOB_CLEAR_ARCHIVE

Synopsis

Purges archived job logs that are older than the specified age in days.

Description

Deletes job-related logs that were previously archived (into the WX_WRK_AUDIT_ARCHIVE and WX_WRK_ERROR_ARCHIVE tables via a RED Scheduler and/or RED callable routines such as Ws_Job_Clear_Logs and Ws_Job_Clear_Logs_By_Date) depending on their age in days.

When the maximum age of the archived logs to retain is exceeded all the older logs are deleted. For example, if 90 days are retained then all the archived logs that are older than 90 days are deleted. If a maximum age of 0 days is specified, then all the archived logs are deleted. Alternatively, the **TRUNCATE** option can be used to remove all the archived logs, which overrides all other criteria.

Input

Input	Description
Common Input	Includes all 5 inputs of the ***Callable Routines Common Input*** (on page 959).
Maximum Days to Retain	The maximum age (in days) of the archived logs to retain. If 90 days are retained, then all the archived logs that are older than 90 days are **purged/deleted**. If 0 days are retained, then all the archived logs are **purged/deleted**.
Job Name to Purge	The name of the job whose **archived** logs are to be **purged**. Wild cards are supported. Specifying % will match ALL jobs.
Options	The **TRUNCATE** option can be used to remove ALL the archived logs, which overrides all other criteria i.e. irrespective of the days to retain or job name.

Output

Output	Description
Return Code	Output Return Code: S Success. E Error. F Fatal/Unexpected Error.
Return Message	Output message indicating the action applied or the reason for no action.

Output	Description
Result Number	Output Result Number: 1 Success. -2 Error. e.g. Due to invalid job name or job not running. -3 Fatal/Unexpected Error.

SQL SERVER

SQL Server Parameters: Ws_Job_Clear_Archive

Callable Routine Type: PROCEDURE.

Parameter Name	Datatype	Mode
@p_sequence	INTEGER	IN
@p_job_name	VARCHAR(64)	IN
@p_task_name	VARCHAR(64)	IN
@p_job_id	INTEGER	IN
@p_task_id	INTEGER	IN
@p_day_count	INTEGER	IN
@p_job	VARCHAR(64)	IN
@p_options	VARCHAR(256)	IN
@p_return_code	VARCHAR(1)	OUT
@p_return_msg	VARCHAR(256)	OUT
@p_result	INTEGER	OUT

SQL Server Examples: Ws_Job_Clear_Archive

```
-- The p_ VARIABLES are normally PARAMETERS in a RED-generated Procedure.
DECLARE @p_sequence             integer
DECLARE @p_job_name             varchar(256)
DECLARE @p_task_name            varchar(256)
DECLARE @p_job_id               integer
DECLARE @p_task_id              integer
DECLARE @p_return_msg           varchar(256)
DECLARE @p_status               integer
DECLARE @v_result_num           integer
DECLARE @v_return_code          varchar(1)
DECLARE @v_return_msg           varchar(256)
EXEC Ws_Job_Clear_Archive
  @p_sequence, @p_job_name, @p_task_name, @p_job_id, @p_task_id
, 90, 'Daily Run', ''
, @v_return_code OUTPUT
, @v_return_msg  OUTPUT
, @v_result_num  OUTPUT
```

TERADATA

Teradata Parameters: Ws_Job_Clear_Archive

Callable Routine Type: PROCEDURE.

Parameter Name	Datatype	Mode
p_sequence	INTEGER	IN
p_job_name	VARCHAR(64)	IN
p_task_name	VARCHAR(64)	IN
p_job_id	INTEGER	IN
p_task_id	INTEGER	IN
p_day_count	INTEGER	IN
p_job	VARCHAR(64)	IN
p_options	VARCHAR(256)	IN
p_return_code	VARCHAR(1)	OUT
p_return_msg	VARCHAR(256)	OUT
p_result	INTEGER	OUT

Teradata Examples: Ws_Job_Clear_Archive

```
-- The p_ VARIABLES are normally PARAMETERS in a RED-generated Procedure.
DECLARE p_sequence              integer;
DECLARE p_job_name              varchar(256);
DECLARE p_task_name             varchar(256);
DECLARE p_job_id                integer;
DECLARE p_task_id               integer;
DECLARE p_return_msg            varchar(256);
DECLARE p_status                integer;
DECLARE v_result_num            integer;
DECLARE v_return_code           varchar(1);
DECLARE v_return_msg            varchar(256);
CALL [METABASE].Ws_Job_Clear_Archive
( p_sequence, p_job_name, p_task_name, p_job_id, p_task_id
, 90, 'Daily Run', ''
, v_return_code
, v_return_msg
, v_result_num
);
```

ORACLE

Oracle Parameters: Ws_Job_Clear_Archive

Callable Routine Type: FUNCTION.

Parameter Name	Datatype	Mode
p_sequence	NUMBER	IN
p_job_name	VARCHAR2	IN
p_task_name	VARCHAR2	IN
p_job_id	NUMBER	IN
p_task_id	NUMBER	IN
p_day_count	NUMBER	IN
p_job	VARCHAR2	IN
p_options	VARCHAR2	IN
p_return_code	VARCHAR2	OUT
p_return_msg	VARCHAR2	OUT
FUNCTION Return Value	NUMBER	OUT-Function

Oracle Examples: Ws_Job_Clear_Archive

```
-- The p_ VARIABLES are normally PARAMETERS in a RED-generated Procedure.
p_sequence                      number;
p_job_name                      varchar2(256);
p_task_name                     varchar2(256);
p_job_id                        number;
p_task_id                       number;
p_return_msg                    varchar2(256);
p_status                        number;
v_result_num                    number;
v_return_code                   varchar2(1);
v_return_msg                    varchar2(256);
v_result_num := Ws_Job_Clear_Archive
( p_sequence, p_job_name, p_task_name, p_job_id, p_task_id
, 90, 'Daily Run', ''
, v_return_code
, v_return_msg
);
```

DB2

DB2 Parameters: Ws_Job_Clear_Archive

Callable Routine Type: PROCEDURE.

Parameter Name	Datatype	Mode
p_sequence	INTEGER	IN
p_job_name	VARCHAR(64)	IN
p_task_name	VARCHAR(64)	IN
p_job_id	INTEGER	IN
p_task_id	INTEGER	IN
p_day_count	INTEGER	IN
p_job	VARCHAR(64)	IN
p_options	VARCHAR(256)	IN
p_return_code	VARCHAR(1)	OUT
p_return_msg	VARCHAR(256)	OUT
p_result	INTEGER	OUT

DB2 Examples: Ws_Job_Clear_Archive

```
-- The p_ VARIABLES are normally PARAMETERS in a RED-generated Procedure.
DECLARE p_sequence              integer;
DECLARE p_job_name              varchar(256);
DECLARE p_task_name             varchar(256);
DECLARE p_job_id                integer;
DECLARE p_task_id               integer;
DECLARE p_return_msg            varchar(256);
DECLARE p_status                integer;
DECLARE v_result_num            integer;
DECLARE v_return_code           varchar(1);
DECLARE v_return_msg            varchar(256);
CALL [METABASE].Ws_Job_Clear_Archive
( p_sequence, p_job_name, p_task_name, p_job_id, p_task_id
, 90, 'Daily Run', ''
, v_return_code
, v_return_msg
, v_result_num
);
```

WS_JOB_CLEAR_LOGS

Synopsis

Archives job logs when the maximum number of current logs to retain is exceeded.

Description

Moves job-related logs from the current log tables (such as WS_WRK_AUDIT_LOG and WS_WRK_ERROR_LOG) to the corresponding archive log tables (such as WX_WRK_AUDIT_ARCHIVE and WX_WRK_ERROR_ARCHIVE) depending on the number of logs to retain.

When the maximum number of current logs to retain is exceeded the oldest logs are archived for the specified job(s) to reduce the number of current logs to the specified retention limit. For example, if 10 is specified then only the latest 10 logs are retained. If a retained count of 0 is specified, then all the current logs are archived for the specified job(s).

Note: Equivalent functionality is available via a WhereScape RED Scheduler and the "Logs Retained" property of a job.

Input

Input	Description
Common Input	Includes all 5 inputs of the ***Callable Routines Common Input*** (on page 959).
Job Name(s) to Archive	The name of the job(s) whose current logs are to be archived. Wild cards are supported. Specifying % will match ALL jobs.
Maximum Logs to Retain	The maximum number of logs to retain. When the maximum is exceeded the oldest logs are **archived** to reduce the number of current logs to the specified retention limit.

Output

Output	Description
Return Code	Output Return Code: S Success. E Error. F Fatal/Unexpected Error.
Return Message	Output message indicating the action applied or the reason for no action.
Result Number	Output Result Number: 1 Success. -2 Error. e.g. Due to invalid job name or job not running. -3 Fatal/Unexpected Error.

SQL SERVER

SQL Server Parameters: Ws_Job_Clear_Logs

Callable Routine Type: PROCEDURE.

Parameter Name	Datatype	Mode
@p_sequence	INTEGER	IN
@p_job_name	VARCHAR(64)	IN
@p_task_name	VARCHAR(64)	IN
@p_job_id	INTEGER	IN
@p_task_id	INTEGER	IN
@p_job_to_clean	VARCHAR(64)	IN
@p_keep_count	INTEGER	IN
@p_return_code	VARCHAR(1)	OUT
@p_return_msg	VARCHAR(256)	OUT
@p_result	INTEGER	OUT

SQL Server Examples: Ws_Job_Clear_Logs

```
-- The p_ VARIABLES are normally PARAMETERS in a RED-generated Procedure.
DECLARE @p_sequence             integer
DECLARE @p_job_name             varchar(256)
DECLARE @p_task_name            varchar(256)
DECLARE @p_job_id               integer
DECLARE @p_task_id              integer
DECLARE @p_return_msg           varchar(256)
DECLARE @p_status               integer
DECLARE @v_result_num           integer
DECLARE @v_return_code          varchar(1)
DECLARE @v_return_msg           varchar(256)
EXEC Ws_Job_Clear_Logs
  @p_sequence, @p_job_name, @p_task_name, @p_job_id, @p_task_id
, 'Daily Run', 10
, @v_return_code OUTPUT
, @v_return_msg  OUTPUT
, @v_result_num  OUTPUT
```

TERADATA

Teradata Parameters: Ws_Job_Clear_Logs

Callable Routine Type: PROCEDURE.

Parameter Name	Datatype	Mode
p_sequence	INTEGER	IN
p_job_name	VARCHAR(64)	IN
p_task_name	VARCHAR(64)	IN
p_job_id	INTEGER	IN
p_task_id	INTEGER	IN
p_job_to_clean	VARCHAR(64)	IN
p_log_keep	INTEGER	IN
p_return_code	VARCHAR(1)	OUT
p_return_msg	VARCHAR(256)	OUT
p_result	INTEGER	OUT

Teradata Examples: Ws_Job_Clear_Logs

```
-- The p_ VARIABLES are normally PARAMETERS in a RED-generated Procedure.
DECLARE p_sequence              integer;
DECLARE p_job_name              varchar(256);
DECLARE p_task_name             varchar(256);
DECLARE p_job_id                integer;
DECLARE p_task_id               integer;
DECLARE p_return_msg            varchar(256);
DECLARE p_status                integer;
DECLARE v_result_num            integer;
DECLARE v_return_code           varchar(1);
DECLARE v_return_msg            varchar(256);
CALL [METABASE].Ws_Job_Clear_Logs
( p_sequence, p_job_name, p_task_name, p_job_id, p_task_id
, 'Daily Run', 10
, v_return_code
, v_return_msg
, v_result_num
);
```

ORACLE

Oracle Parameters: Ws_Job_Clear_Logs

Callable Routine Type: FUNCTION.

Parameter Name	Datatype	Mode
p_sequence	NUMBER	IN
p_job_name	VARCHAR2	IN
p_task_name	VARCHAR2	IN
p_job_id	NUMBER	IN
p_task_id	NUMBER	IN
p_job_to_clean	VARCHAR2	IN
p_log_keep	NUMBER	IN
p_return_code	VARCHAR2	OUT
p_return_msg	VARCHAR2	OUT
FUNCTION Return Value	NUMBER	OUT-Function

Oracle Examples: Ws_Job_Clear_Logs

```
-- The p_ VARIABLES are normally PARAMETERS in a RED-generated Procedure.
p_sequence                      number;
p_job_name                      varchar2(256);
p_task_name                     varchar2(256);
p_job_id                        number;
p_task_id                       number;
p_return_msg                    varchar2(256);
p_status                        number;
v_result_num                    number;
v_return_code                   varchar2(1);
v_return_msg                    varchar2(256);
v_result_num := Ws_Job_Clear_Logs
( p_sequence, p_job_name, p_task_name, p_job_id, p_task_id
, 'Daily Run', 10
, v_return_code
, v_return_msg
);
```

DB2

DB2 Parameters: Ws_Job_Clear_Logs

Callable Routine Type: PROCEDURE.

Parameter Name	Datatype	Mode
p_sequence	INTEGER	IN
p_job_name	VARCHAR(64)	IN
p_task_name	VARCHAR(64)	IN
p_job_id	INTEGER	IN
p_task_id	INTEGER	IN
p_job_to_clean	VARCHAR(64)	IN
p_keep_count	INTEGER	IN
p_return_code	VARCHAR(1)	OUT
p_return_msg	VARCHAR(256)	OUT
p_result	INTEGER	OUT

DB2 Examples: Ws_Job_Clear_Logs

```
-- The p_ VARIABLES are normally PARAMETERS in a RED-generated Procedure.
DECLARE p_sequence              integer;
DECLARE p_job_name              varchar(256);
DECLARE p_task_name             varchar(256);
DECLARE p_job_id                integer;
DECLARE p_task_id               integer;
DECLARE p_return_msg            varchar(256);
DECLARE p_status                integer;
DECLARE v_result_num            integer;
DECLARE v_return_code           varchar(1);
DECLARE v_return_msg            varchar(256);
CALL [METABASE].Ws_Job_Clear_Logs
( p_sequence, p_job_name, p_task_name, p_job_id, p_task_id
, 'Daily Run', 10
, v_return_code
, v_return_msg
, v_result_num
);
```

WS_JOB_CLEAR_LOGS_BY_DATE

Synopsis

Archives job logs that are older than the specified age in days.

Description

Moves job-related logs from the current log tables (such as WS_WRK_AUDIT_LOG and WS_WRK_ERROR_LOG) to the corresponding archive log tables (such as WX_WRK_AUDIT_ARCHIVE and WX_WRK_ERROR_ARCHIVE) depending on their age in days.

When the maximum age of the current logs to retain is exceeded all the older logs are archived for the specified job(s). For example, if 90 days are retained then all the current logs that are older than 90 days are archived.

Input

Input	Description
Common Input	Includes all 5 inputs of the ***Callable Routines Common Input*** (on page 959).
Job Name(s) to Archive	The name of the job(s) whose current logs are to be **archived**. Wild cards are supported. Specifying % will match ALL jobs.
Maximum Days to Retain	The maximum age (in days) of the current logs to retain. If 90 days are retained then all the current logs that are older than 90 days are archived.

Output

Output	Description
Return Code	Output Return Code: S Success. E Error. F Fatal/Unexpected Error.
Return Message	Output message indicating the action applied or the reason for no action.
Result Number	Output Result Number: 1 Success. -2 Error. e.g. Due to invalid job name or job not running. -3 Fatal/Unexpected Error.

SQL SERVER

SQL Server Parameters: Ws_Job_Clear_Logs_By_Date

Callable Routine Type: PROCEDURE.

Parameter Name	Datatype	Mode
@p_sequence	INTEGER	IN
@p_job_name	VARCHAR(64)	IN
@p_task_name	VARCHAR(64)	IN
@p_job_id	INTEGER	IN
@p_task_id	INTEGER	IN
@p_job_to_clean	VARCHAR(64)	IN
@p_day_count	INTEGER	IN
@p_return_code	VARCHAR(1)	OUT
@p_return_msg	VARCHAR(256)	OUT
@p_result	INTEGER	OUT

SQL Server Examples: Ws_Job_Clear_Logs_By_Date

```
-- The p_ VARIABLES are normally PARAMETERS in a RED-generated Procedure.
DECLARE @p_sequence             integer
DECLARE @p_job_name             varchar(256)
DECLARE @p_task_name            varchar(256)
DECLARE @p_job_id               integer
DECLARE @p_task_id              integer
DECLARE @p_return_msg           varchar(256)
DECLARE @p_status               integer
DECLARE @v_result_num           integer
DECLARE @v_return_code          varchar(1)
DECLARE @v_return_msg           varchar(256)
EXEC Ws_Job_Clear_Logs_By_Date
  @p_sequence, @p_job_name, @p_task_name, @p_job_id, @p_task_id
, 'Daily Run', 90
, @v_return_code OUTPUT
, @v_return_msg  OUTPUT
, @v_result_num  OUTPUT
```

TERADATA

Teradata Parameters: Ws_Job_Clear_Logs_By_Date

Callable Routine Type: PROCEDURE.

Parameter Name	Datatype	Mode
p_sequence	INTEGER	IN
p_job_name	VARCHAR(64)	IN
p_task_name	VARCHAR(64)	IN
p_job_id	INTEGER	IN
p_task_id	INTEGER	IN
p_job_to_clean	VARCHAR(64)	IN
p_day_count	INTEGER	IN
p_return_code	VARCHAR(1)	OUT
p_return_msg	VARCHAR(256)	OUT
p_result	INTEGER	OUT

Teradata Examples: Ws_Job_Clear_Logs_By_Date

```
-- The p_ VARIABLES are normally PARAMETERS in a RED-generated Procedure.
DECLARE p_sequence              integer;
DECLARE p_job_name              varchar(256);
DECLARE p_task_name             varchar(256);
DECLARE p_job_id                integer;
DECLARE p_task_id               integer;
DECLARE p_return_msg            varchar(256);
DECLARE p_status                integer;
DECLARE v_result_num            integer;
DECLARE v_return_code           varchar(1);
DECLARE v_return_msg            varchar(256);
CALL [METABASE].Ws_Job_Clear_Logs_By_Date
( p_sequence, p_job_name, p_task_name, p_job_id, p_task_id
, 'Daily Run', 90
, v_return_code
, v_return_msg
, v_result_num
);
```

ORACLE

Oracle Parameters: Ws_Job_Clear_Logs_By_Date

Callable Routine Type: FUNCTION.

Parameter Name	Datatype	Mode
p_sequence	NUMBER	IN
p_job_name	VARCHAR2	IN
p_task_name	VARCHAR2	IN
p_job_id	NUMBER	IN
p_task_id	NUMBER	IN
p_job_to_clean	VARCHAR2	IN
p_day_count	NUMBER	IN
p_return_code	VARCHAR2	OUT
p_return_msg	VARCHAR2	OUT
p_result	NUMBER	OUT-Function

Oracle Examples: Ws_Job_Clear_Logs_By_Date

```
-- The p_ VARIABLES are normally PARAMETERS in a RED-generated Procedure.
p_sequence                      number;
p_job_name                      varchar2(256);
p_task_name                     varchar2(256);
p_job_id                        number;
p_task_id                       number;
p_return_msg                    varchar2(256);
p_status                        number;
v_result_num                    number;
v_return_code                   varchar2(1);
v_return_msg                    varchar2(256);
v_result_num := Ws_Job_Clear_Logs_By_Date
( p_sequence, p_job_name, p_task_name, p_job_id, p_task_id
, 'Daily Run', 90
, v_return_code
, v_return_msg
);
```

DB2

DB2 Parameters: Ws_Job_Clear_Logs_By_Date

Callable Routine Type: PROCEDURE.

Parameter Name	Datatype	Mode
p_sequence	INTEGER	IN
p_job_name	VARCHAR(64)	IN
p_task_name	VARCHAR(64)	IN
p_job_id	INTEGER	IN
p_task_id	INTEGER	IN
p_job_to_clean	VARCHAR(64)	IN
p_day_count	INTEGER	IN
p_return_code	VARCHAR(1)	OUT
p_return_msg	VARCHAR(256)	OUT
p_result	INTEGER	OUT

DB2 Examples: Ws_Job_Clear_Logs_By_Date

```
-- The p_ VARIABLES are normally PARAMETERS in a RED-generated Procedure.
DECLARE p_sequence              integer;
DECLARE p_job_name              varchar(256);
DECLARE p_task_name             varchar(256);
DECLARE p_job_id                integer;
DECLARE p_task_id               integer;
DECLARE p_return_msg            varchar(256);
DECLARE p_status                integer;
DECLARE v_result_num            integer;
DECLARE v_return_code           varchar(1);
DECLARE v_return_msg            varchar(256);
CALL [METABASE].Ws_Job_Clear_Logs_By_Date
( p_sequence, p_job_name, p_task_name, p_job_id, p_task_id
, 'Daily Run', 90
, v_return_code
, v_return_msg
, v_result_num
);
```

WS_JOB_CREATE

Synopsis

Creates a job based on an existing job and optionally starts it immediately.

Description

Creates a job from the specified existing job if it is in either a holding or waiting state. The new job can be started immediately. Typically, this routine is used to create & start a job from within another job. Only jobs that are in a holding or waiting state can be used as a template for the new job.

Input

Input	Description
Common Input	Includes all 5 inputs of the ***Callable Routines Common Input*** (on page 959).
Template Job Name	The name of the job to be used as a template for the new job. The specified name must exactly match the job name as displayed by the WhereScape RED Scheduler. The specified job must be in a holding or waiting state in order to be used as a template.
New Job Name	The name of the job to be created. The new job name cannot already exist.
Job Description	A description of the new job. When not specified the setting of the Template job is copied.
New Job Status	The initial status/frequency of the new job: **HOLD** - The status of the new job will show as 'On Hold'. The job will not run until it is subsequently released. **ONCE** - The new job will start immediately and upon successful completion the new job will be deleted. **ONCE+HOLD** - The new job will start immediately and upon successful completion the status will show as 'On Hold'.
Thread Count	The number of threads for the new job. When not specified the setting of the Template job is copied.
Scheduler Preference	A scheduler type or a specific scheduler name that is allowed to run the job. When not specified the setting of the Template job is copied. **Note**: Some jobs/tasks can only run in a specific environment such as Windows or UNIX/Linux
Maximum Logs to Retain	The maximum number of logs to retain. When not specified the setting of the Template job is copied.

Input	Description
Success Command	A command-line action to execute upon successful completion of the new job. When not specified the setting of the Template job is copied. The command must be executable within the context of the scheduler that runs the job so it must be a valid Windows/UNIX/Linux command that is appropriate to the scheduler environment.
Failure Command	A command-line action to execute upon failure of the new job. When not specified the setting of the Template job is copied. The command must be executable within the context of the scheduler that runs the job so it must be a valid Windows/UNIX/Linux command that is appropriate to the scheduler environment.

Output

Output	Description
Return Code	Output Return Code: S Success. **N No action because the Template Job is not in a holding or waiting state.** **P No action because the New Job name already exists.** E Error. F Fatal/Unexpected Error.
Return Message	Output message indicating the action applied or the reason for no action.
Result Number	Output Result Number: 1 Success. **-1 Template Job is not in a holding/waiting state or the New Job name already exists.** -2 Error. e.g. Due to invalid job name or job not running. -3 Fatal/Unexpected Error.

SQL SERVER

SQL Server Parameters: Ws_Job_Create

Callable Routine Type: PROCEDURE.

Parameter Name	Datatype	Mode
@p_sequence	INTEGER	IN
@p_job_name	VARCHAR(64)	IN
@p_task_name	VARCHAR(64)	IN
@p_job_id	INTEGER	IN
@p_task_id	INTEGER	IN
@p_template_job	VARCHAR(64)	IN
@p_new_job	VARCHAR(64)	IN
@p_description	VARCHAR(256)	IN
@p_state	VARCHAR(64)	IN
@p_threads	INTEGER	IN
@p_scheduler	VARCHAR(64)	IN
@p_logs	INTEGER	IN
@p_okay	VARCHAR(256)	IN
@p_fail	VARCHAR(256)	IN
@p_att1	VARCHAR(64)	IN
@p_att2	VARCHAR(64)	IN
@p_att3	VARCHAR(64)	IN
@p_att4	VARCHAR(64)	IN
@p_att5	VARCHAR(64)	IN
@p_att6	VARCHAR(64)	IN
@p_att7	VARCHAR(64)	IN
@p_att8	VARCHAR(64)	IN
@p_return_code	VARCHAR(1)	OUT
@p_return_msg	VARCHAR(256)	OUT
@p_result	INTEGER	OUT

SQL Server Examples: Ws_Job_Create

```
-- The p_ VARIABLES are normally PARAMETERS in a RED-generated Procedure.
DECLARE @p_sequence             integer
DECLARE @p_job_name             varchar(256)
DECLARE @p_task_name            varchar(256)
DECLARE @p_job_id               integer
DECLARE @p_task_id              integer
DECLARE @p_return_msg           varchar(256)
DECLARE @p_status               integer
DECLARE @v_result_num           integer
DECLARE @v_return_code          varchar(1)
DECLARE @v_return_msg           varchar(256)
EXEC Ws_Job_Create
  @p_sequence, @p_job_name, @p_task_name, @p_job_id, @p_task_id
, 'Daily Run', 'New Daily Run', 'This is the New Daily Run job.', 'ONCE'
, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL,
NULL
, @v_return_code OUTPUT
, @v_return_msg  OUTPUT
, @v_result_num  OUTPUT
```

TERADATA

Teradata Parameters: Ws_Job_Create

Callable Routine Type: PROCEDURE.

Parameter Name	Datatype	Mode
p_sequence	INTEGER	IN
p_job_name	VARCHAR(64)	IN
p_task_name	VARCHAR(64)	IN
p_job_id	INTEGER	IN
p_task_id	INTEGER	IN
p_template_job	VARCHAR(64)	IN
p_new_job	VARCHAR(64)	IN
p_description	VARCHAR(256)	IN
p_state	VARCHAR(10)	IN
p_threads	INTEGER	IN
p_scheduler	VARCHAR(8)	IN
p_logs	INTEGER	IN

Parameter Name	Datatype	Mode
p_okay	VARCHAR(256)	IN
p_fail	VARCHAR(256)	IN
p_att1	VARCHAR(4000)	IN
p_att2	VARCHAR(4000)	IN
p_att3	VARCHAR(4000)	IN
p_att4	VARCHAR(4000)	IN
p_att5	VARCHAR(4000)	IN
p_att6	VARCHAR(4000)	IN
p_att7	VARCHAR(4000)	IN
p_att8	VARCHAR(4000)	IN
p_return_code	VARCHAR(1)	OUT
p_return_msg	VARCHAR(256)	OUT
p_result	INTEGER	OUT

Teradata Examples: Ws_Job_Create

```
-- The p_ VARIABLES are normally PARAMETERS in a RED-generated Procedure.
DECLARE p_sequence              integer;
DECLARE p_job_name              varchar(256);
DECLARE p_task_name             varchar(256);
DECLARE p_job_id                integer;
DECLARE p_task_id               integer;
DECLARE p_return_msg            varchar(256);
DECLARE p_status                integer;
DECLARE v_result_num            integer;
DECLARE v_return_code           varchar(1);
DECLARE v_return_msg            varchar(256);
CALL [METABASE].Ws_Job_Create
( p_sequence, p_job_name, p_task_name, p_job_id, p_task_id
, 'Daily Run', 'New Daily Run', 'This is the New Daily Run job.', 'ONCE'
, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL,
NULL
, v_return_code
, v_return_msg
, v_result_num
);
```

ORACLE

Oracle Parameters: Ws_Job_Create

Callable Routine Type: FUNCTION.

Parameter Name	Datatype	Mode
p_sequence	NUMBER	IN
p_job_name	VARCHAR2	IN
p_task_name	VARCHAR2	IN
p_job_id	NUMBER	IN
p_task_id	NUMBER	IN
p_template_job	VARCHAR2	IN
p_new_job	VARCHAR2	IN
p_description	VARCHAR2	IN
p_state	VARCHAR2	IN
p_threads	NUMBER	IN
p_scheduler	VARCHAR2	IN
p_logs	NUMBER	IN
p_okay	VARCHAR2	IN
p_fail	VARCHAR2	IN
p_att1	VARCHAR2	IN
p_att2	VARCHAR2	IN
p_att3	VARCHAR2	IN
p_att4	VARCHAR2	IN
p_att5	VARCHAR2	IN
p_att6	VARCHAR2	IN
p_att7	VARCHAR2	IN
p_att8	VARCHAR2	IN
p_return_code	VARCHAR2	OUT
p_return_msg	VARCHAR2	OUT
FUNCTION Return Value	NUMBER	OUT-Function

Oracle Examples: Ws_Job_Create

```
-- The p_ VARIABLES are normally PARAMETERS in a RED-generated Procedure.
p_sequence                      number;
p_job_name                      varchar2(256);
p_task_name                     varchar2(256);
p_job_id                        number;
p_task_id                       number;
p_return_msg                    varchar2(256);
p_status                        number;
v_result_num                    number;
v_return_code                   varchar2(1);
v_return_msg                    varchar2(256);
v_result_num := Ws_Job_Create
( p_sequence, p_job_name, p_task_name, p_job_id, p_task_id
, 'Daily Run', 'New Daily Run', 'This is the New Daily Run job.', 'ONCE'
, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL,
NULL
, v_return_code
, v_return_msg
);
```

DB2

DB2 Parameters: Ws_Job_Create

Callable Routine Type: PROCEDURE.

Parameter Name	Datatype	Mode
p_sequence	INTEGER	IN
p_job_name	VARCHAR(64)	IN
p_task_name	VARCHAR(64)	IN
p_job_id	INTEGER	IN
p_task_id	INTEGER	IN
p_template_job	VARCHAR(64)	IN
p_new_job	VARCHAR(64)	IN
p_description	VARCHAR(256)	IN
p_state	VARCHAR(10)	IN
p_threads	INTEGER	IN
p_scheduler	VARCHAR(8)	IN

Parameter Name	Datatype	Mode
p_logs	INTEGER	IN
p_okay	VARCHAR(256)	IN
p_fail	VARCHAR(256)	IN
p_att1	VARCHAR(4000)	IN
p_att2	VARCHAR(4000)	IN
p_att3	VARCHAR(4000)	IN
p_att4	VARCHAR(4000)	IN
p_att5	VARCHAR(4000)	IN
p_att6	VARCHAR(4000)	IN
p_att7	VARCHAR(4000)	IN
p_att8	VARCHAR(4000)	IN
p_return_code	VARCHAR(1)	OUT
p_return_msg	VARCHAR(256)	OUT
p_result	INTEGER	OUT

DB2 Examples: Ws_Job_Create

```
-- The p_ VARIABLES are normally PARAMETERS in a RED-generated Procedure.
DECLARE p_sequence              integer;
DECLARE p_job_name              varchar(256);
DECLARE p_task_name             varchar(256);
DECLARE p_job_id                integer;
DECLARE p_task_id               integer;
DECLARE p_return_msg            varchar(256);
DECLARE p_status                integer;
DECLARE v_result_num            integer;
DECLARE v_return_code           varchar(1);
DECLARE v_return_msg            varchar(256);
CALL [METABASE].Ws_Job_Create
( p_sequence, p_job_name, p_task_name, p_job_id, p_task_id
, 'Daily Run', 'New Daily Run', 'This is the New Daily Run job.', 'ONCE'
, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL,
NULL
, v_return_code
, v_return_msg
, v_result_num
);
```

WS_JOB_CREATEWAIT

Synopsis

Creates a job based on an existing job and schedules it to start later.

Description

Creates a job from the specified existing job if it is in either a holding or waiting state. The new job is scheduled to start later at the specified release time. Typically, this routine is used to create & schedule a job from within another job. Only jobs that are in a holding or waiting state can be used as a template for the new job.

Input

Input	Description
Common Input	Includes all 5 inputs of the ***Callable Routines Common Input*** (on page 959).
Template Job Name	The name of the job to be used as a template for the new job. The specified name must exactly match the job name as displayed by the WhereScape RED Scheduler. The specified job must be in a holding or waiting state in order to be used as a template.
New Job Name	The name of the job to be created. The new job name cannot already exist.
Job Description	A description of the new job. When not specified the setting of the Template job is copied.
New Job Status	The initial status/frequency of the new job: **HOLD** - The status of the new job will show as 'On Hold'. The job will not run until it is subsequently released. **ONCE** - The new job will start immediately and upon successful completion the new job will be deleted. **ONCE+HOLD** - The new job will start immediately and upon successful completion the status will show as 'On Hold'.
Scheduled Release Date/Time	**The date/time when the new job is scheduled to be run.**
Thread Count	The number of threads for the new job. When not specified the setting of the Template job is copied.
Scheduler Preference	A scheduler type or a specific scheduler name that is allowed to run the job. When not specified the setting of the Template job is copied. **Note**: Some jobs/tasks can only run in a specific environment such as Windows or UNIX/Linux

Input	Description
Maximum Logs to Retain	The maximum number of logs to retain. When not specified the setting of the Template job is copied.
Success Command	A command-line action to execute upon successful completion of the new job. When not specified the setting of the Template job is copied. The command must be executable within the context of the scheduler that runs the job so it must be a valid Windows/UNIX/Linux command that is appropriate to the scheduler environment.
Failure Command	A command-line action to execute upon failure of the new job. When not specified the setting of the Template job is copied. The command must be executable within the context of the scheduler that runs the job so it must be a valid Windows/UNIX/Linux command that is appropriate to the scheduler environment.

Output

Output	Description
Return Code	Output Return Code: S Success. **N No action because the Template Job is not in a holding or waiting state.** **P No action because the New Job name already exists.** E Error. F Fatal/Unexpected Error.
Return Message	Output message indicating the action applied or the reason for no action.
Result Number	Output Result Number: 1 Success. **-1 Template Job is not in a holding/waiting state or the New Job name already exists.** -2 Error. e.g. Due to invalid job name or job not running. -3 Fatal/Unexpected Error.

SQL SERVER

SQL Server Parameters: Ws_Job_CreateWait

Callable Routine Type: PROCEDURE.

Parameter Name	Datatype	Mode
@p_sequence	INTEGER	IN
@p_job_name	VARCHAR(64)	IN
@p_task_name	VARCHAR(64)	IN
@p_job_id	INTEGER	IN
@p_task_id	INTEGER	IN
@p_template_job	VARCHAR(64)	IN
@p_new_job	VARCHAR(64)	IN
@p_description	VARCHAR(256)	IN
@p_state	VARCHAR(64)	IN
@p_release_time	DATETIME	IN
@p_threads	INTEGER	IN
@p_scheduler	VARCHAR(64)	IN
@p_logs	INTEGER	IN
@p_okay	VARCHAR(256)	IN
@p_fail	VARCHAR(256)	IN
@p_att1	VARCHAR(64)	IN
@p_att2	VARCHAR(64)	IN
@p_att3	VARCHAR(64)	IN
@p_att4	VARCHAR(64)	IN
@p_att5	VARCHAR(64)	IN
@p_att6	VARCHAR(64)	IN
@p_att7	VARCHAR(64)	IN
@p_att8	VARCHAR(64)	IN
@p_return_code	VARCHAR(1)	OUT
@p_return_msg	VARCHAR(256)	OUT
@p_result	INTEGER	OUT

SQL Server Examples: Ws_Job_CreateWait

```
-- The p_ VARIABLES are normally PARAMETERS in a RED-generated Procedure.
DECLARE @p_sequence             integer
DECLARE @p_job_name             varchar(256)
DECLARE @p_task_name            varchar(256)
DECLARE @p_job_id               integer
DECLARE @p_task_id              integer
DECLARE @p_return_msg           varchar(256)
DECLARE @p_status               integer
DECLARE @v_result_num           integer
DECLARE @v_return_code          varchar(1)
DECLARE @v_return_msg           varchar(256)
DECLARE @v_run_date             datetime
SET @v_run_date = DATEADD(MONTH, 1, GETDATE())
EXEC Ws_Job_CreateWait
  @p_sequence, @p_job_name, @p_task_name, @p_job_id, @p_task_id
, 'Daily Run', 'New Daily Run2', 'This is the New Daily Run job.', 'ONCE'
, @v_run_date
, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL,
NULL
, @v_return_code OUTPUT
, @v_return_msg  OUTPUT
, @v_result_num  OUTPUT
```

TERADATA

Teradata Parameters: Ws_Job_CreateWait

Callable Routine Type: PROCEDURE.

Parameter Name	Datatype	Mode
p_sequence	INTEGER	IN
p_job_name	VARCHAR(64)	IN
p_task_name	VARCHAR(64)	IN
p_job_id	INTEGER	IN
p_task_id	INTEGER	IN
p_template_job	VARCHAR(64)	IN
p_new_job	VARCHAR(64)	IN
p_description	VARCHAR(256)	IN
p_state	VARCHAR(10)	IN
p_release_time	TIMESTAMP	IN
p_threads	INTEGER	IN
p_scheduler	VARCHAR(8)	IN
p_logs	INTEGER	IN
p_okay	VARCHAR(256)	IN
p_fail	VARCHAR(256)	IN
p_att1	VARCHAR(4000)	IN
p_att2	VARCHAR(4000)	IN
p_att3	VARCHAR(4000)	IN
p_att4	VARCHAR(4000)	IN
p_att5	VARCHAR(4000)	IN
p_att6	VARCHAR(4000)	IN
p_att7	VARCHAR(4000)	IN
p_att8	VARCHAR(4000)	IN
p_return_code	VARCHAR(1)	OUT
p_return_msg	VARCHAR(256)	OUT
p_result	INTEGER	OUT

Teradata Examples: Ws_Job_CreateWait

```
-- The p_ VARIABLES are normally PARAMETERS in a RED-generated Procedure.
DECLARE p_sequence              integer;
DECLARE p_job_name              varchar(256);
DECLARE p_task_name             varchar(256);
DECLARE p_job_id                integer;
DECLARE p_task_id               integer;
DECLARE p_return_msg            varchar(256);
DECLARE p_status                integer;
DECLARE v_result_num            integer;
DECLARE v_return_code           varchar(1);
DECLARE v_return_msg            varchar(256);
CALL [METABASE].Ws_Job_CreateWait
( p_sequence, p_job_name, p_task_name, p_job_id, p_task_id
, 'Daily Run', 'New Daily Run', 'This is the New Daily Run job.', 'ONCE'
, (CURRENT_TIMESTAMP + INTERVAL '1' MONTH)
, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL,
NULL
, v_return_code
, v_return_msg
, v_result_num
);
```

ORACLE

Oracle Parameters: Ws_Job_CreateWait

Callable Routine Type: FUNCTION.

Parameter Name	Datatype	Mode
p_sequence	NUMBER	IN
p_job_name	VARCHAR2	IN
p_task_name	VARCHAR2	IN
p_job_id	NUMBER	IN
p_task_id	NUMBER	IN
p_template_job	VARCHAR2	IN
p_new_job	VARCHAR2	IN
p_description	VARCHAR2	IN
p_state	VARCHAR2	IN
p_release_time	DATE	IN
p_threads	NUMBER	IN
p_scheduler	VARCHAR2	IN
p_logs	NUMBER	IN
p_okay	VARCHAR2	IN
p_fail	VARCHAR2	IN
p_att1	VARCHAR2	IN
p_att2	VARCHAR2	IN
p_att3	VARCHAR2	IN
p_att4	VARCHAR2	IN
p_att5	VARCHAR2	IN
p_att6	VARCHAR2	IN
p_att7	VARCHAR2	IN
p_att8	VARCHAR2	IN
p_return_code	VARCHAR2	OUT
p_return_msg	VARCHAR2	OUT
FUNCTION Return Value	NUMBER	OUT-Function

Oracle Examples: Ws_Job_CreateWait

```
-- The p_ VARIABLES are normally PARAMETERS in a RED-generated Procedure.
p_sequence                      number;
p_job_name                      varchar2(256);
p_task_name                     varchar2(256);
p_job_id                        number;
p_task_id                       number;
p_return_msg                    varchar2(256);
p_status                        number;
v_result_num                    number;
v_return_code                   varchar2(1);
v_return_msg                    varchar2(256);
v_result_num := Ws_Job_CreateWait
( p_sequence, p_job_name, p_task_name, p_job_id, p_task_id
, 'Daily Run', 'New Daily Run', 'This is the New Daily Run job.', 'ONCE'
, ADD_MONTHS(SYSDATE, 1)
, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL,
NULL
, v_return_code
, v_return_msg
);
```

DB2

DB2 Parameters: Ws_Job_CreateWait

Callable Routine Type: PROCEDURE.

Parameter Name	Datatype	Mode
p_sequence	INTEGER	IN
p_job_name	VARCHAR(64)	IN
p_task_name	VARCHAR(64)	IN
p_job_id	INTEGER	IN
p_task_id	INTEGER	IN
p_template_job	VARCHAR(64)	IN
p_new_job	VARCHAR(64)	IN
p_description	VARCHAR(256)	IN
p_state	VARCHAR(10)	IN
p_release_time	TIMESTAMP	IN
p_threads	INTEGER	IN
p_scheduler	VARCHAR(8)	IN

Parameter Name	Datatype	Mode
p_logs	INTEGER	IN
p_okay	VARCHAR(256)	IN
p_fail	VARCHAR(256)	IN
p_att1	VARCHAR(4000)	IN
p_att2	VARCHAR(4000)	IN
p_att3	VARCHAR(4000)	IN
p_att4	VARCHAR(4000)	IN
p_att5	VARCHAR(4000)	IN
p_att6	VARCHAR(4000)	IN
p_att7	VARCHAR(4000)	IN
p_att8	VARCHAR(4000)	IN
p_return_code	VARCHAR(1)	OUT
p_return_msg	VARCHAR(256)	OUT
p_result	INTEGER	OUT

DB2 Examples: Ws_Job_CreateWait

```
-- The p_ VARIABLES are normally PARAMETERS in a RED-generated Procedure.
DECLARE p_sequence              integer;
DECLARE p_job_name              varchar(256);
DECLARE p_task_name             varchar(256);
DECLARE p_job_id                integer;
DECLARE p_task_id               integer;
DECLARE p_return_msg            varchar(256);
DECLARE p_status                integer;
DECLARE v_result_num            integer;
DECLARE v_return_code           varchar(1);
DECLARE v_return_msg            varchar(256);
CALL [METABASE].Ws_Job_CreateWait
( p_sequence, p_job_name, p_task_name, p_job_id, p_task_id
, 'Daily Run', 'New Daily Run', 'This is the New Daily Run job.', 'ONCE'
, (CURRENT TIMESTAMP + 1 MONTH)
, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL,
NULL
, v_return_code
, v_return_msg
, v_result_num
);
```

WS_JOB_DEPENDENCY

Synopsis

Adds or removes a child-to-parent dependency between two jobs to control the child job.

Description

Adds or removes a child-to-parent dependency between two jobs to control the child job. The dependent child job can be defined to fail (if necessary) when the parent job does not complete successfully in the required timeframe. The acceptable timeframe can be defined in terms of the maximum minutes in the past to look back and the maximum minutes in the future to wait for successful completion of the parent job.

Input

Input	Description
Common Input	Includes all 5 inputs of the ***Callable Routines Common Input*** (on page 959).
Action	Either 'ADD' or 'DELETE' the job dependency.
Parent Job Name	The name of Parent Job that the Child Job will depend on.
Child Job Name	The name of the Child Job that will be dependent on the Parent Job.
Parent Success Required	Indicates whether the Child Job will fail when the Parent Job does not complete successfully in the required time frame.
Maximum Look Back Minutes	The Maximum minutes in the past to look back for successful completion of the Parent Job.
Maximum Wait Minutes	The Maximum minutes in the future to wait for successful completion of the Parent Job.

Output

Output	Description
Return Code	Output Return Code: S Success. **W Warning. Dependency already exists ('ADD' action) or does not exist ('DELETE' action).** E Error. F Fatal/Unexpected Error.

Output	Description
Return Message	Output message indicating the action applied or the reason for no action.
Result Number	Output Result Number: 1 Success. **-1 Warning. Dependency already exists ('ADD' action) or does not exist ('DELETE' action).** -2 Error. e.g. Due to invalid job name or job not running. -3 Fatal/Unexpected Error.

SQL SERVER

SQL Server Parameters: Ws_Job_Dependency

Callable Routine Type: PROCEDURE.

Parameter Name	Datatype	Mode
@p_sequence	INTEGER	IN
@p_job_name	VARCHAR(64)	IN
@p_task_name	VARCHAR(64)	IN
@p_job_id	INTEGER	IN
@p_task_id	INTEGER	IN
@p_action	VARCHAR(64)	IN
@p_parent	VARCHAR(64)	IN
@p_child	VARCHAR(64)	IN
@p_required	VARCHAR(1)	IN
@p_look_back	INTEGER	IN
@p_max_wait	INTEGER	IN
@p_return_code	VARCHAR(1)	OUT
@p_return_msg	VARCHAR(256)	OUT
@p_result	INTEGER	OUT

SQL Server Examples: Ws_Job_Dependency

```
-- The p_ VARIABLES are normally PARAMETERS in a RED-generated Procedure.
DECLARE @p_sequence             integer
DECLARE @p_job_name             varchar(256)
DECLARE @p_task_name            varchar(256)
DECLARE @p_job_id               integer
DECLARE @p_task_id              integer
DECLARE @p_return_msg           varchar(256)
DECLARE @p_status               integer
DECLARE @v_result_num           integer
DECLARE @v_return_code          varchar(1)
DECLARE @v_return_msg           varchar(256)
EXEC Ws_Job_Dependency
  @p_sequence, @p_job_name, @p_task_name, @p_job_id, @p_task_id
, 'ADD', 'Daily Run', 'Daily Run Part2', 'Y', 60, 60
, @v_return_code OUTPUT
, @v_return_msg  OUTPUT
, @v_result_num  OUTPUT
```

TERADATA

Teradata Parameters: Ws_Job_Dependency

Callable Routine Type: PROCEDURE.

Parameter Name	Datatype	Mode
p_sequence	INTEGER	IN
p_job_name	VARCHAR(64)	IN
p_task_name	VARCHAR(64)	IN
p_job_id	INTEGER	IN
p_task_id	INTEGER	IN
p_action	VARCHAR(10)	IN
p_parent	VARCHAR(64)	IN
p_child	VARCHAR(64)	IN
p_required	VARCHAR(1)	IN
p_look_back	INTEGER	IN
p_max_wait	INTEGER	IN
p_return_code	VARCHAR(1)	OUT
p_return_msg	VARCHAR(256)	OUT

Parameter Name	Datatype	Mode
p_result	INTEGER	OUT

Teradata Examples: Ws_Job_Dependency

```
-- The p_ VARIABLES are normally PARAMETERS in a RED-generated Procedure.
DECLARE p_sequence              integer;
DECLARE p_job_name              varchar(256);
DECLARE p_task_name             varchar(256);
DECLARE p_job_id                integer;
DECLARE p_task_id               integer;
DECLARE p_return_msg            varchar(256);
DECLARE p_status                integer;
DECLARE v_result_num            integer;
DECLARE v_return_code           varchar(1);
DECLARE v_return_msg            varchar(256);
CALL [METABASE].Ws_Job_Dependency
( p_sequence, p_job_name, p_task_name, p_job_id, p_task_id
, 'ADD', 'Daily Run', 'Daily Run Part2', 'Y', 60, 60
, v_return_code
, v_return_msg
, v_result_num
);
```

ORACLE

Oracle Parameters: Ws_Job_Dependency

Callable Routine Type: FUNCTION.

Parameter Name	Datatype	Mode
p_sequence	NUMBER	IN
p_job_name	VARCHAR2	IN
p_task_name	VARCHAR2	IN
p_job_id	NUMBER	IN
p_task_id	NUMBER	IN
p_action	VARCHAR2	IN
p_parent	VARCHAR2	IN
p_child	VARCHAR2	IN
p_required	VARCHAR2	IN
p_look_back	NUMBER	IN

Parameter Name	Datatype	Mode
p_max_wait	NUMBER	IN
p_return_code	VARCHAR2	OUT
p_return_msg	VARCHAR2	OUT
FUNCTION Return Value	NUMBER	OUT-Function

Oracle Examples: Ws_Job_Dependency

```
-- The p_ VARIABLES are normally PARAMETERS in a RED-generated Procedure.
p_sequence                      number;
p_job_name                      varchar2(256);
p_task_name                     varchar2(256);
p_job_id                        number;
p_task_id                       number;
p_return_msg                    varchar2(256);
p_status                        number;
v_result_num                    number;
v_return_code                   varchar2(1);
v_return_msg                    varchar2(256);
v_result_num := Ws_Job_Dependency
( p_sequence, p_job_name, p_task_name, p_job_id, p_task_id
, 'ADD', 'Daily Run', 'Daily Run Part2', 'Y', 60, 60
, v_return_code
, v_return_msg
);
```

DB2

DB2 Parameters: Ws_Job_Dependency

Callable Routine Type: PROCEDURE.

Parameter Name	Datatype	Mode
p_sequence	INTEGER	IN
p_job_name	VARCHAR(64)	IN
p_task_name	VARCHAR(64)	IN
p_job_id	INTEGER	IN
p_task_id	INTEGER	IN
p_action	VARCHAR(10)	IN
p_parent	VARCHAR(64)	IN
p_child	VARCHAR(64)	IN

Parameter Name	Datatype	Mode
p_required	VARCHAR(1)	IN
p_look_back	INTEGER	IN
p_max_wait	INTEGER	IN
p_return_code	VARCHAR(1)	OUT
p_return_msg	VARCHAR(256)	OUT
p_result	INTEGER	OUT

DB2 Examples: Ws_Job_Dependency

```
-- The p_ VARIABLES are normally PARAMETERS in a RED-generated Procedure.
DECLARE p_sequence              integer;
DECLARE p_job_name              varchar(256);
DECLARE p_task_name             varchar(256);
DECLARE p_job_id                integer;
DECLARE p_task_id               integer;
DECLARE p_return_msg            varchar(256);
DECLARE p_status                integer;
DECLARE v_result_num            integer;
DECLARE v_return_code           varchar(1);
DECLARE v_return_msg            varchar(256);
CALL [METABASE].Ws_Job_Dependency
( p_sequence, p_job_name, p_task_name, p_job_id, p_task_id
, 'ADD', 'Daily Run', 'Daily Run Part2', 'Y', 60, 60
, v_return_code
, v_return_msg
, v_result_num
);
```

WS_JOB_RELEASE

Synopsis

Starts a job if it is in a holding or waiting state.

Description

Releases the specified job if it is in a holding or waiting state, which sets the start time to the current time so that it starts immediately. Typically, this routine is used to start a job from within another job or via a third-party scheduler (rather than a WhereScape RED Scheduler).

Input

Input	Description
Common Input	Includes all 5 inputs of the ***Callable Routines Common Input*** (on page 959).
Release Job Name	The name of the job to be started/released. The specified name must exactly match the job name as displayed by the WhereScape RED Scheduler. The specified job must be in a holding or waiting state in order to be released.

Output

Output	Description
Return Code	Output Return Code: S Success. **N No action because the Template Job is not in a holding or waiting state.** E Error. F Fatal/Unexpected Error.
Return Message	Output message indicating the action applied or the reason for no action.
Result Number	Output Result Number: 1 Success. **-1 No action because the job is not in a holding or waiting state.** -2 Error. -3 Fatal/Unexpected Error.

SQL SERVER

SQL Server Parameters: Ws_Job_Release

Callable Routine Type: PROCEDURE.

Parameter Name	Datatype	Mode
@p_sequence	INTEGER	IN
@p_job_name	VARCHAR(64)	IN
@p_task_name	VARCHAR(64)	IN
@p_job_id	INTEGER	IN
@p_task_id	INTEGER	IN
@p_release_job	VARCHAR(64)	IN
@p_return_code	VARCHAR(1)	OUT
@p_return_msg	VARCHAR(256)	OUT
@p_result	INTEGER	OUT

SQL Server Examples: Ws_Job_Release

```
-- The p_ VARIABLES are normally PARAMETERS in a RED-generated Procedure.
DECLARE @p_sequence             integer
DECLARE @p_job_name             varchar(256)
DECLARE @p_task_name            varchar(256)
DECLARE @p_job_id               integer
DECLARE @p_task_id              integer
DECLARE @p_return_msg           varchar(256)
DECLARE @p_status               integer
DECLARE @v_result_num           integer
DECLARE @v_return_code          varchar(1)
DECLARE @v_return_msg           varchar(256)
EXEC Ws_Job_Release
  @p_sequence, @p_job_name, @p_task_name, @p_job_id, @p_task_id
, 'Daily Run'
, @v_return_code OUTPUT
, @v_return_msg  OUTPUT
, @v_result_num  OUTPUT
```

TERADATA

Teradata Parameters: Ws_Job_Release

Callable Routine Type: PROCEDURE.

Parameter Name	Datatype	Mode
p_sequence	INTEGER	IN
p_job_name	VARCHAR(64)	IN
p_task_name	VARCHAR(64)	IN
p_job_id	INTEGER	IN
p_task_id	INTEGER	IN
p_release_job	VARCHAR(64)	IN
p_return_code	VARCHAR(1)	OUT
p_return_msg	VARCHAR(256)	OUT
p_result	INTEGER	OUT

Teradata Examples: Ws_Job_Release

```
-- The p_ VARIABLES are normally PARAMETERS in a RED-generated Procedure.
DECLARE p_sequence              integer;
DECLARE p_job_name              varchar(256);
DECLARE p_task_name             varchar(256);
DECLARE p_job_id                integer;
DECLARE p_task_id               integer;
DECLARE p_return_msg            varchar(256);
DECLARE p_status                integer;
DECLARE v_result_num            integer;
DECLARE v_return_code           varchar(1);
DECLARE v_return_msg            varchar(256);
CALL [METABASE].Ws_Job_Release
( p_sequence, p_job_name, p_task_name, p_job_id, p_task_id
, 'Daily Run'
, v_return_code
, v_return_msg
, v_result_num
);
```

ORACLE

Oracle Parameters: Ws_Job_Release

Callable Routine Type: FUNCTION.

Parameter Name	Datatype	Mode
p_sequence	NUMBER	IN
p_job_name	VARCHAR2	IN
p_task_name	VARCHAR2	IN
p_job_id	NUMBER	IN
p_task_id	NUMBER	IN
p_release_job	VARCHAR2	IN
p_return_code	VARCHAR2	OUT
p_return_msg	VARCHAR2	OUT
FUNCTION Return Value	NUMBER	OUT-Function

Oracle Examples: Ws_Job_Release

```
-- The p_ VARIABLES are normally PARAMETERS in a RED-generated Procedure.
p_sequence                      number;
p_job_name                      varchar2(256);
p_task_name                     varchar2(256);
p_job_id                        number;
p_task_id                       number;
p_return_msg                    varchar2(256);
p_status                        number;
v_result_num                    number;
v_return_code                   varchar2(1);
v_return_msg                    varchar2(256);
v_result_num := Ws_Job_Release
( p_sequence, p_job_name, p_task_name, p_job_id, p_task_id
, 'Daily Run'
, v_return_code
, v_return_msg
);
```

DB2

DB2 Parameters: Ws_Job_Release

Callable Routine Type: PROCEDURE.

Parameter Name	Datatype	Mode
p_sequence	INTEGER	IN
p_job_name	VARCHAR(64)	IN
p_task_name	VARCHAR(64)	IN
p_job_id	INTEGER	IN
p_task_id	INTEGER	IN
p_release_job	VARCHAR(64)	IN
p_return_code	VARCHAR(1)	OUT
p_return_msg	VARCHAR(256)	OUT
p_result	INTEGER	OUT

DB2 Examples: Ws_Job_Release

```
-- The p_ VARIABLES are normally PARAMETERS in a RED-generated Procedure.
DECLARE p_sequence              integer;
DECLARE p_job_name              varchar(256);
DECLARE p_task_name             varchar(256);
DECLARE p_job_id                integer;
DECLARE p_task_id               integer;
DECLARE p_return_msg            varchar(256);
DECLARE p_status                integer;
DECLARE v_result_num            integer;
DECLARE v_return_code           varchar(1);
DECLARE v_return_msg            varchar(256);
CALL [METABASE].Ws_Job_Release
( p_sequence, p_job_name, p_task_name, p_job_id, p_task_id
, 'Daily Run'
, v_return_code
, v_return_msg
, v_result_num
);
```

WS_JOB_RESTART

Synopsis

Starts a job if it is in a failed state.

Description

Releases the specified job if it is in a failed state, which sets the start time to the current time so that it starts immediately. This routine can be executed as part of a database start-up sequence to restart each failed job that may have stopped due to an earlier database shutdown.

Input

Input	Description
Common Input	Includes all 5 inputs of the ***Callable Routines Common Input*** (on page 959).
Restart Job Name	The name of the job to be restarted/released. The specified name must exactly match the job name as displayed by the WhereScape RED Scheduler. The specified job must be in a failed state in order to be restarted.

Output

Output	Description
Return Code	Output Return Code: S Success. **N No action because the job is not in a failed state.** **R No action because the job is currently running.** **U No action because the job is in an unusual state due to an error (result number -2). The job is classified as running but it is NOT actually running or failed so it cannot be restarted. This may occur if the scheduler has failed and the job is in a pending state.** E Error. F Fatal/Unexpected Error.
Return Message	Output message indicating the action applied or the reason for no action.
Result Number	Output Result Number: 1 Success. **-1 No action because the job is not in a failed state or it is currently running.** -2 Error. -3 Fatal/Unexpected Error.

SQL SERVER

SQL Server Parameters: Ws_Job_Restart

Callable Routine Type: PROCEDURE.

Parameter Name	Datatype	Mode
@p_sequence	INTEGER	IN
@p_job_name	VARCHAR(64)	IN
@p_task_name	VARCHAR(64)	IN
@p_job_id	INTEGER	IN
@p_task_id	INTEGER	IN
@p_restart_job	VARCHAR(64)	IN
@p_return_code	VARCHAR(1)	OUT
@p_return_msg	VARCHAR(256)	OUT
@p_result	INTEGER	OUT

SQL Server Examples: Ws_Job_Restart

```
-- The p_ VARIABLES are normally PARAMETERS in a RED-generated Procedure.
DECLARE @p_sequence             integer
DECLARE @p_job_name             varchar(256)
DECLARE @p_task_name            varchar(256)
DECLARE @p_job_id               integer
DECLARE @p_task_id              integer
DECLARE @p_return_msg           varchar(256)
DECLARE @p_status               integer
DECLARE @v_result_num           integer
DECLARE @v_return_code          varchar(1)
DECLARE @v_return_msg           varchar(256)
EXEC Ws_Job_Restart
  @p_sequence, @p_job_name, @p_task_name, @p_job_id, @p_task_id
, 'Daily Run'
, @v_return_code OUTPUT
, @v_return_msg  OUTPUT
, @v_result_num  OUTPUT
```

TERADATA

Teradata Parameters: Ws_Job_Restart

Callable Routine Type: PROCEDURE.

Parameter Name	Datatype	Mode
p_sequence	INTEGER	IN
p_job_name	VARCHAR(64)	IN
p_task_name	VARCHAR(64)	IN
p_job_id	INTEGER	IN
p_task_id	INTEGER	IN
p_restart_job	VARCHAR(64)	IN
p_return_code	VARCHAR(1)	OUT
p_return_msg	VARCHAR(256)	OUT
p_result	INTEGER	OUT

Teradata Examples: Ws_Job_Restart

```
-- The p_ VARIABLES are normally PARAMETERS in a RED-generated Procedure.
DECLARE p_sequence              integer;
DECLARE p_job_name              varchar(256);
DECLARE p_task_name             varchar(256);
DECLARE p_job_id                integer;
DECLARE p_task_id               integer;
DECLARE p_return_msg            varchar(256);
DECLARE p_status                integer;
DECLARE v_result_num            integer;
DECLARE v_return_code           varchar(1);
DECLARE v_return_msg            varchar(256);
CALL [METABASE].Ws_Job_Restart
( p_sequence, p_job_name, p_task_name, p_job_id, p_task_id
, 'Daily Run'
, v_return_code
, v_return_msg
, v_result_num
);
```

ORACLE

Oracle Parameters: Ws_Job_Restart

Callable Routine Type: FUNCTION.

Parameter Name	Datatype	Mode
p_sequence	NUMBER	IN
p_job_name	VARCHAR2	IN
p_task_name	VARCHAR2	IN
p_job_id	NUMBER	IN
p_task_id	NUMBER	IN
p_restart_job	VARCHAR2	IN
p_return_code	VARCHAR2	OUT
p_return_msg	VARCHAR2	OUT
FUNCTION Return Value	NUMBER	OUT-Function

Oracle Examples: Ws_Job_Restart

```
-- The p_ VARIABLES are normally PARAMETERS in a RED-generated Procedure.
p_sequence                      number;
p_job_name                      varchar2(256);
p_task_name                     varchar2(256);
p_job_id                        number;
p_task_id                       number;
p_return_msg                    varchar2(256);
p_status                        number;
v_result_num                    number;
v_return_code                   varchar2(1);
v_return_msg                    varchar2(256);
v_result_num := Ws_Job_Restart
( p_sequence, p_job_name, p_task_name, p_job_id, p_task_id
, 'Daily Run'
, v_return_code
, v_return_msg
);
```

DB2

DB2 Parameters: Ws_Job_Restart

Callable Routine Type: PROCEDURE.

Parameter Name	Datatype	Mode
p_sequence	INTEGER	IN
p_job_name	VARCHAR(64)	IN
p_task_name	VARCHAR(64)	IN
p_job_id	INTEGER	IN
p_task_id	INTEGER	IN
p_restart_job	VARCHAR(64)	IN
p_return_code	VARCHAR(1)	OUT
p_return_msg	VARCHAR(256)	OUT
p_result	INTEGER	OUT

DB2 Examples: Ws_Job_Restart

```
-- The p_ VARIABLES are normally PARAMETERS in a RED-generated Procedure.
DECLARE p_sequence              integer;
DECLARE p_job_name              varchar(256);
DECLARE p_task_name             varchar(256);
DECLARE p_job_id                integer;
DECLARE p_task_id               integer;
DECLARE p_return_msg            varchar(256);
DECLARE p_status                integer;
DECLARE v_result_num            integer;
DECLARE v_return_code           varchar(1);
DECLARE v_return_msg            varchar(256);
CALL [METABASE].Ws_Job_Restart
( p_sequence, p_job_name, p_task_name, p_job_id, p_task_id
, 'Daily Run'
, v_return_code
, v_return_msg
, v_result_num
);
```

WS_JOB_SCHEDULE

Synopsis

Schedules a job if it is in a holding or waiting state.

Description

Schedules the specified job if it is in a holding or waiting state, which will start at the specified time. Typically, this routine is used to schedule a job from within another job.

Input

Input	Description
Common Input	Includes all 5 inputs of the ***Callable Routines Common Input*** (on page 959).
Schedule Job Name	The name of the job to be scheduled. The specified name must exactly match the job name as displayed by the WhereScape RED Scheduler. The specified job must be in a holding or waiting state in order to be scheduled.
Scheduled Release Time	The date/time that the job is to be scheduled to be released/started.

Output

Output	Description
Return Code	Output Return Code: S Success. **N No action because the job is not in a holding or waiting state.** E Error. F Fatal/Unexpected Error.
Return Message	Output message indicating the action applied or the reason for no action.
Result Number	Output Result Number: 1 Success. **-1 No action because the job is not in a holding or waiting state.** -2 Error. -3 Fatal/Unexpected Error.

SQL SERVER

SQL Server Parameters: Ws_Job_Schedule

Callable Routine Type: PROCEDURE.

Parameter Name	Datatype	Mode
@p_sequence	INTEGER	IN
@p_job_name	VARCHAR(64)	IN
@p_task_name	VARCHAR(64)	IN
@p_job_id	INTEGER	IN
@p_task_id	INTEGER	IN
@p_release_job	VARCHAR(64)	IN
@p_release_time	DATETIME	IN
@p_return_code	VARCHAR(1)	OUT
@p_return_msg	VARCHAR(256)	OUT
@p_result	INTEGER	OUT

SQL Server Examples: Ws_Job_Schedule

```
-- The p_ VARIABLES are normally PARAMETERS in a RED-generated Procedure.
DECLARE @p_sequence             integer
DECLARE @p_job_name             varchar(256)
DECLARE @p_task_name            varchar(256)
DECLARE @p_job_id               integer
DECLARE @p_task_id              integer
DECLARE @p_return_msg           varchar(256)
DECLARE @p_status               integer
DECLARE @v_result_num           integer
DECLARE @v_return_code          varchar(1)
DECLARE @v_return_msg           varchar(256)
DECLARE @v_run_date             datetime
SET @v_run_date = GETDATE() + 1
EXEC Ws_Job_Schedule
  @p_sequence, @p_job_name, @p_task_name, @p_job_id, @p_task_id
, 'Daily Run', @v_run_date
, @v_return_code OUTPUT
, @v_return_msg  OUTPUT
, @v_result_num  OUTPUT
```

TERADATA

Teradata Parameters: Ws_Job_Schedule

Callable Routine Type: PROCEDURE.

Parameter Name	Datatype	Mode
p_sequence	INTEGER	IN
p_job_name	VARCHAR(64)	IN
p_task_name	VARCHAR(64)	IN
p_job_id	INTEGER	IN
p_task_id	INTEGER	IN
p_release_job	VARCHAR(64)	IN
p_release_time	TIMESTAMP	IN
p_return_code	VARCHAR(1)	OUT
p_return_msg	VARCHAR(256)	OUT
p_result	INTEGER	OUT

Teradata Examples: Ws_Job_Schedule

```
-- The p_ VARIABLES are normally PARAMETERS in a RED-generated Procedure.
DECLARE p_sequence              integer;
DECLARE p_job_name              varchar(256);
DECLARE p_task_name             varchar(256);
DECLARE p_job_id                integer;
DECLARE p_task_id               integer;
DECLARE p_return_msg            varchar(256);
DECLARE p_status                integer;
DECLARE v_result_num            integer;
DECLARE v_return_code           varchar(1);
DECLARE v_return_msg            varchar(256);
CALL [METABASE].Ws_Job_Schedule
( p_sequence, p_job_name, p_task_name, p_job_id, p_task_id
, 'Daily Run', CURRENT_TIMESTAMP + INTERVAL '1' DAY
, v_return_code
, v_return_msg
, v_result_num
);
```

ORACLE

Oracle Parameters: Ws_Job_Schedule

Callable Routine Type: FUNCTION.

Parameter Name	Datatype	Mode
p_sequence	NUMBER	IN
p_job_name	VARCHAR2	IN
p_task_name	VARCHAR2	IN
p_job_id	NUMBER	IN
p_task_id	NUMBER	IN
p_release_job	VARCHAR2	IN
p_release_time	DATE	IN
p_return_code	VARCHAR2	OUT
p_return_msg	VARCHAR2	OUT
FUNCTION Return Value	NUMBER	OUT-Function

Oracle Examples: Ws_Job_Schedule

```
-- The p_ VARIABLES are normally PARAMETERS in a RED-generated Procedure.
p_sequence                      number;
p_job_name                      varchar2(256);
p_task_name                     varchar2(256);
p_job_id                        number;
p_task_id                       number;
p_return_msg                    varchar2(256);
p_status                        number;
v_result_num                    number;
v_return_code                   varchar2(1);
v_return_msg                    varchar2(256);
v_result_num := Ws_Job_Schedule
( p_sequence, p_job_name, p_task_name, p_job_id, p_task_id
, 'Daily Run', SYSDATE+1
, v_return_code
, v_return_msg
);
```

DB2

DB2 Parameters: Ws_Job_Schedule

Callable Routine Type: PROCEDURE.

Parameter Name	Datatype	Mode
p_sequence	INTEGER	IN
p_job_name	VARCHAR(64)	IN
p_task_name	VARCHAR(64)	IN
p_job_id	INTEGER	IN
p_task_id	INTEGER	IN
p_release_job	VARCHAR(64)	IN
p_release_time	TIMESTAMP	IN
p_return_code	VARCHAR(1)	OUT
p_return_msg	VARCHAR(256)	OUT
p_result	INTEGER	OUT

DB2 Examples: Ws_Job_Schedule

```
-- The p_ VARIABLES are normally PARAMETERS in a RED-generated Procedure.
DECLARE p_sequence              integer;
DECLARE p_job_name              varchar(256);
DECLARE p_task_name             varchar(256);
DECLARE p_job_id                integer;
DECLARE p_task_id               integer;
DECLARE p_return_msg            varchar(256);
DECLARE p_status                integer;
DECLARE v_result_num            integer;
DECLARE v_return_code           varchar(1);
DECLARE v_return_msg            varchar(256);
CALL [METABASE].Ws_Job_Schedule
( p_sequence, p_job_name, p_task_name, p_job_id, p_task_id
, 'Daily Run', CURRENT TIMESTAMP + 1 DAY
, v_return_code
, v_return_msg
, v_result_num
);
```

WS_JOB_STATUS

Synopsis

Returns the current status of a job.

Description

Returns the current status of the specified job as recorded by a WhereScape RED Scheduler. Typically, this routine is used by a third-party scheduler or a user-defined procedure/script to check on a job.

Input

Input	Description
Common Input	Includes all 5 inputs of the ***Callable Routines Common Input*** (on page 959).
Job Sequence	The unique integer identifier of the job to return the status of. This input is optional but when it is specified, the started **within** and started **after** inputs should not be specified.
Job Name	The name of the job to return the status of. The specified name must exactly match the job name as displayed by the WhereScape RED Scheduler.
Started Within Last Minutes	The maximum minutes [0-148599] (up to ~103.1 days) in the past to look back for the job to have started. This input is optional but when it is specified, the job name must be specified and the job sequence and started **after** inputs should not be specified. Note: If multiple instances of the job have started in the specified time frame then the last job to start is returned (i.e. the job with the highest sequence number).
Started After Time	The date/time after which to look for the job to have started. This input is optional but when it is specified, the job name must be specified and the job sequence and started **within** inputs should not be specified. Note: If multiple instances of the job have started in the specified time frame then the last job to start is returned (i.e. the job with the highest sequence number).

Output

Output	Description
Return Code	Output Return Code: S - **Success**. N - **The job exists but it was NOT started within the specified time frame.** E - **Error**. F - **Fatal/Unexpected Error**.
Return Message	Output message indicating the action applied or the reason for no action.
Result Number	Output Result Number: 1 - **Success**. -1 - **The job exists but it was NOT started within the specified time frame.** -2 - **Error**. -3 -**Fatal/Unexpected Error**. 0 - see note below.
Simplified Job Status Code	Simplified Job Status Code: N - **Not Running**. R - **Running**. F - **Failed**. C - **Completed**. 0 - see note below.
Standard Job Status Code	Standard Job Status Code: H - **On Hold**. The job is on hold. A held job can be edited and/or started. W - **Waiting**. The job is waiting to start (it is either waiting for the scheduled time to arrive or is waiting for an available scheduler). B - **Blocked**. The job is blocked because a previous instance of the same job is still running. P - **Pending**. This is the initial interim status of an "about to start running" job. The scheduler has identified that the job is ready to start and is preparing to run it. A job should only be pending for a brief period so if it remains pending for a prolonged period then an unexpected error has occurred. R - **Running**. The job is currently running. F - **Failed**. The job failed due to an error. C - **Completed**. The job completed successfully (but it may have warnings). A completed job cannot be restarted. G - **Failed - Aborted**. The job failed and was subsequently aborted. An aborted job cannot be restarted. E - **Error Completion**. 0 - see note below.

Output	Description
Enhanced Job Status Number	Enhanced Job Status Number that returns an **integer** rather than the standard alphabetic code. The **running** and **completed statuses are enhanced** to distinguish errors or warnings. 1 - **On Hold.** The job is on hold. A held job can be edited and/or started. 2 - **Waiting**. The job is waiting to start (it is either waiting for the scheduled time to arrive or is waiting for an available scheduler). 3 - **Blocked**. The job is blocked because a previous instance of the same job is still running. 4 - **Pending**. This is the initial interim status of an "about to start running" job. The scheduler has identified that the job is ready to start and is preparing to run it. A job should only be pending for a brief period so if it remains pending for a prolonged period then an unexpected error has occurred. 5 - **Running**. The job is currently running and no tasks have failed or produced warnings. 6 - **Running with Errors**. The job is currently running but some tasks have failed. The job will ultimately fail when all the tasks that are NOT dependent on the failed tasks have finished. 7 - **Running with Warnings**. The job is currently running and some tasks have produced warnings. 8 - **Failed**. The job failed due to an error. 9 - **Completed**. The job completed without warnings. A completed job cannot be restarted. 10 - **Completed with Warnings**. The job completed with warnings. A completed job cannot be restarted. 11 - **Failed - Aborted**. The job failed and it was subsequently aborted. An aborted job cannot be restarted. 12 - **Error Completion**. 0 - see note below.

***** NOTE: *****

All three returned status values can also return '**0**' in any of the following situations:

- Illegal combination of parameters specified.
- Unable to locate specified job sequence.
- Unable to locate specified job name.
- Job Not Found having started in the last *SpecifiedMinutes* minutes.
- Job Not Found having started after *SpecifiedDateTime*.

SQL SERVER

SQL Server Parameters: Ws_Job_Status

Callable Routine Type: PROCEDURE.

Parameter Name	Datatype	Mode
@p_sequence	INTEGER	IN
@p_job_name	VARCHAR(64)	IN
@p_task_name	VARCHAR(64)	IN
@p_job_id	INTEGER	IN
@p_task_id	INTEGER	IN
@p_check_sequence	INTEGER	IN
@p_check_job	VARCHAR(64)	IN
@p_started_in_last_mi	INTEGER	IN
@p_started_after_dt	DATETIME	IN
@p_return_code	VARCHAR(1)	OUT
@p_return_msg	VARCHAR(256)	OUT
@p_result	INTEGER	OUT
@p_job_status_simple	VARCHAR(1)	OUT
@p_job_status_standard	VARCHAR(1)	OUT
@p_job_status_enhanced	VARCHAR(2)	OUT

SQL Server Examples: Ws_Job_Status

```
-- The p_ VARIABLES are normally PARAMETERS in a RED-generated Procedure.
DECLARE @p_sequence             integer
DECLARE @p_job_name             varchar(256)
DECLARE @p_task_name            varchar(256)
DECLARE @p_job_id               integer
DECLARE @p_task_id              integer
DECLARE @p_return_msg           varchar(256)
DECLARE @p_status               integer
DECLARE @v_result_num           integer
DECLARE @v_return_code          varchar(1)
DECLARE @v_return_msg           varchar(256)
DECLARE @v_job_status_simple    varchar(1)
DECLARE @v_job_status_standard  varchar(1)
DECLARE @v_job_status_enhanced  varchar(2)
EXEC Ws_Job_Status
  @p_sequence, @p_job_name, @p_task_name, @p_job_id, @p_task_id
, NULL, 'Daily Run', 10, NULL
, @v_return_code OUTPUT
, @v_return_msg  OUTPUT
, @v_result_num  OUTPUT
, @v_job_status_simple   OUTPUT
, @v_job_status_standard OUTPUT
, @v_job_status_enhanced OUTPUT
```

TERADATA

Teradata Parameters: Ws_Job_Status

Callable Routine Type: PROCEDURE.

Parameter Name	Datatype	Mode
p_sequence	INTEGER	IN
p_job_name	VARCHAR(64)	IN
p_task_name	VARCHAR(64)	IN
p_job_id	INTEGER	IN
p_task_id	INTEGER	IN
p_check_sequence	INTEGER	IN
p_check_job	VARCHAR(64)	IN
p_started_in_last_mi	INTEGER	IN
p_started_after_dt	TIMESTAMP	IN
p_return_code	VARCHAR(1)	OUT

Parameter Name	Datatype	Mode
p_return_msg	VARCHAR(256)	OUT
p_result	INTEGER	OUT
p_job_status_simple	VARCHAR(1)	OUT
p_job_status_standard	VARCHAR(1)	OUT
p_job_status_enhanced	VARCHAR(2)	OUT

Teradata Examples: Ws_Job_Status

```
-- The p_ VARIABLES are normally PARAMETERS in a RED-generated Procedure.
DECLARE p_sequence              integer;
DECLARE p_job_name              varchar(256);
DECLARE p_task_name             varchar(256);
DECLARE p_job_id                integer;
DECLARE p_task_id               integer;
DECLARE p_return_msg            varchar(256);
DECLARE p_status                integer;
DECLARE v_result_num            integer;
DECLARE v_return_code           varchar(1);
DECLARE v_return_msg            varchar(256);
DECLARE v_job_status_simple     varchar(1);
DECLARE v_job_status_standard   varchar(1);
DECLARE v_job_status_enhanced   varchar(2);
CALL [METABASE].Ws_Job_Status
( p_sequence, p_job_name, p_task_name, p_job_id, p_task_id
, NULL, 'Daily Run', 10, NULL
, v_return_code
, v_return_msg
, v_result_num
, v_job_status_simple, v_job_status_standard, v_job_status_enhanced
);
```

ORACLE

Oracle Parameters: Ws_Job_Status

Callable Routine Type: FUNCTION.

Parameter Name	Datatype	Mode
p_sequence	NUMBER	IN
p_job_name	VARCHAR2	IN
p_task_name	VARCHAR2	IN
p_job_id	NUMBER	IN
p_task_id	NUMBER	IN
p_check_sequence	NUMBER	IN
p_check_job	VARCHAR2	IN
p_started_in_last_mi	NUMBER	IN
p_started_after_dt	DATE	IN
p_return_code	VARCHAR2	OUT
p_return_msg	VARCHAR2	OUT
p_job_status_simple	VARCHAR2	OUT
p_job_status_standard	VARCHAR2	OUT
p_job_status_enhanced	VARCHAR2	OUT
FUNCTION Return Value	NUMBER	OUT-Function

Oracle Examples: Ws_Job_Status

```
-- The p_ VARIABLES are normally PARAMETERS in a RED-generated Procedure.
p_sequence                      number;
p_job_name                      varchar2(256);
p_task_name                     varchar2(256);
p_job_id                        number;
p_task_id                       number;
p_return_msg                    varchar2(256);
p_status                        number;
v_result_num                    number;
v_return_code                   varchar2(1);
v_return_msg                    varchar2(256);
v_job_status_simple             varchar2(1);
v_job_status_standard           varchar2(1);
v_job_status_enhanced           varchar2(2);
v_result_num := Ws_Job_Status
( p_sequence, p_job_name, p_task_name, p_job_id, p_task_id
, NULL, 'Daily Run', 10, NULL
, v_return_code
, v_return_msg
, v_job_status_simple, v_job_status_standard, v_job_status_enhanced
);
```

DB2

DB2 Parameters: Ws_Job_Status

Callable Routine Type: PROCEDURE.

Parameter Name	Datatype	Mode
p_sequence	INTEGER	IN
p_job_name	VARCHAR(64)	IN
p_task_name	VARCHAR(64)	IN
p_job_id	INTEGER	IN
p_task_id	INTEGER	IN
p_check_sequence	INTEGER	IN
p_check_job	VARCHAR(64)	IN
p_started_in_last_mi	INTEGER	IN
p_started_after_dt	TIMESTAMP	IN
p_return_code	VARCHAR(1)	OUT

Parameter Name	Datatype	Mode
p_return_msg	VARCHAR(256)	OUT
p_result	INTEGER	OUT
p_job_status_simple	VARCHAR(1)	OUT
p_job_status_standard	VARCHAR(1)	OUT
p_job_status_enhanced	VARCHAR(2)	OUT

DB2 Examples: Ws_Job_Status

```
-- The p_ VARIABLES are normally PARAMETERS in a RED-generated Procedure.
DECLARE p_sequence              integer;
DECLARE p_job_name              varchar(256);
DECLARE p_task_name             varchar(256);
DECLARE p_job_id                integer;
DECLARE p_task_id               integer;
DECLARE p_return_msg            varchar(256);
DECLARE p_status                integer;
DECLARE v_result_num            integer;
DECLARE v_return_code           varchar(1);
DECLARE v_return_msg            varchar(256);
DECLARE v_job_status_simple     varchar(1);
DECLARE v_job_status_standard   varchar(1);
DECLARE v_job_status_enhanced   varchar(2);
CALL [METABASE].Ws_Job_Status
( p_sequence, p_job_name, p_task_name, p_job_id, p_task_id
, NULL, 'Daily Run', 10, NULL
, v_return_code
, v_return_msg
, v_result_num
, v_job_status_simple, v_job_status_standard, v_job_status_enhanced
);
```

WS_LOAD_CHANGE

Synopsis

Changes the Connection or Schema of a load table.

Description

Changes either the Connection or the Schema of the specified load table. Only the Connection or Schema can be changed so two calls are required to change both.

Input

Input	Description
Common Input	Includes all 5 inputs of the ***Callable Routines Common Input*** (on page 959).
Change Property	Change either the 'SCHEMA' or the 'CONNECTION' of the specified load table. Separate calls must be made if both the schema and connection need to be changed.
Load Table Name	The name of the load table to be changed.
New Property Value	Either the new schema name or the new connection name.

Output

Output	Description
Return Code	Output Return Code: S Success. E Error. F Fatal/Unexpected Error.
Return Message	Output message indicating the action applied or the reason for no action.
Result Number	Output Result Number: 1 Success. -2 Error. -3 Fatal/Unexpected Error.

SQL SERVER

SQL Server Parameters: Ws_Load_Change

Callable Routine Type: PROCEDURE.

Parameter Name	Datatype	Mode
@p_sequence	INTEGER	IN
@p_job_name	VARCHAR(64)	IN
@p_task_name	VARCHAR(64)	IN
@p_job_id	INTEGER	IN
@p_task_id	INTEGER	IN
@p_action	VARCHAR(64)	IN
@p_table	VARCHAR(64)	IN
@p_new_value	VARCHAR(64)	IN
@p_return_code	VARCHAR(1)	OUT
@p_return_msg	VARCHAR(256)	OUT
@p_result	INTEGER	OUT

SQL Server Examples: Ws_Load_Change

```
-- The p_ VARIABLES are normally PARAMETERS in a RED-generated Procedure.
DECLARE @p_sequence             integer
DECLARE @p_job_name             varchar(256)
DECLARE @p_task_name            varchar(256)
DECLARE @p_job_id               integer
DECLARE @p_task_id              integer
DECLARE @p_return_msg           varchar(256)
DECLARE @p_status               integer
DECLARE @v_result_num           integer
DECLARE @v_return_code          varchar(1)
DECLARE @v_return_msg           varchar(256)
EXEC Ws_Load_Change
  @p_sequence, @p_job_name, @p_task_name, @p_job_id, @p_task_id
, 'CONNECTION', 'load_customer', 'Connection2'
, @v_return_code OUTPUT
, @v_return_msg  OUTPUT
, @v_result_num  OUTPUT
```

TERADATA

Teradata Parameters: Ws_Load_Change

Callable Routine Type: PROCEDURE.

Parameter Name	Datatype	Mode
p_sequence	INTEGER	IN
p_job_name	VARCHAR(64)	IN
p_task_name	VARCHAR(64)	IN
p_job_id	INTEGER	IN
p_task_id	INTEGER	IN
p_action	VARCHAR(64)	IN
p_table	VARCHAR(64)	IN
p_new_value	VARCHAR(255)	IN
p_return_code	VARCHAR(1)	OUT
p_return_msg	VARCHAR(256)	OUT
p_result	INTEGER	OUT

Teradata Examples: Ws_Load_Change

```
-- The p_ VARIABLES are normally PARAMETERS in a RED-generated Procedure.
DECLARE p_sequence              integer;
DECLARE p_job_name              varchar(256);
DECLARE p_task_name             varchar(256);
DECLARE p_job_id                integer;
DECLARE p_task_id               integer;
DECLARE p_return_msg            varchar(256);
DECLARE p_status                integer;
DECLARE v_result_num            integer;
DECLARE v_return_code           varchar(1);
DECLARE v_return_msg            varchar(256);
CALL [METABASE].Ws_Load_Change
( p_sequence, p_job_name, p_task_name, p_job_id, p_task_id
, 'CONNECTION', 'load_customer', 'Connection2'
, v_return_code
, v_return_msg
, v_result_num
);
```

ORACLE

Oracle Parameters: Ws_Load_Change

Callable Routine Type: FUNCTION.

Parameter Name	Datatype	Mode
p_sequence	NUMBER	IN
p_job_name	VARCHAR2	IN
p_task_name	VARCHAR2	IN
p_job_id	NUMBER	IN
p_task_id	NUMBER	IN
p_action	VARCHAR2	IN
p_table	VARCHAR2	IN
p_new_value	VARCHAR2	IN
p_return_code	VARCHAR2	OUT
p_return_msg	VARCHAR2	OUT
FUNCTION Return Value	NUMBER	OUT-Function

Oracle Examples: Ws_Load_Change

```
-- The p_ VARIABLES are normally PARAMETERS in a RED-generated Procedure.
p_sequence                      number;
p_job_name                      varchar2(256);
p_task_name                     varchar2(256);
p_job_id                        number;
p_task_id                       number;
p_return_msg                    varchar2(256);
p_status                        number;
v_result_num                    number;
v_return_code                   varchar2(1);
v_return_msg                    varchar2(256);
v_result_num := Ws_Load_Change
( p_sequence, p_job_name, p_task_name, p_job_id, p_task_id
, 'CONNECTION', 'load_customer', 'Connection2'
, v_return_code
, v_return_msg
);
```

DB2

DB2 Parameters: Ws_Load_Change

Callable Routine Type: PROCEDURE.

Parameter Name	Datatype	Mode
p_sequence	INTEGER	IN
p_job_name	VARCHAR(64)	IN
p_task_name	VARCHAR(64)	IN
p_job_id	INTEGER	IN
p_task_id	INTEGER	IN
p_action	VARCHAR(64)	IN
p_table	VARCHAR(64)	IN
p_new_value	VARCHAR(255)	IN
p_return_code	VARCHAR(1)	OUT
p_return_msg	VARCHAR(256)	OUT
p_result	INTEGER	OUT

DB2 Examples: Ws_Load_Change

```
-- The p_ VARIABLES are normally PARAMETERS in a RED-generated Procedure.
DECLARE p_sequence              integer;
DECLARE p_job_name              varchar(256);
DECLARE p_task_name             varchar(256);
DECLARE p_job_id                integer;
DECLARE p_task_id               integer;
DECLARE p_return_msg            varchar(256);
DECLARE p_status                integer;
DECLARE v_result_num            integer;
DECLARE v_return_code           varchar(1);
DECLARE v_return_msg            varchar(256);
CALL [METABASE].Ws_Load_Change
( p_sequence, p_job_name, p_task_name, p_job_id, p_task_id
, 'CONNECTION', 'load_customer', 'Connection2'
, v_return_code
, v_return_msg
, v_result_num
);
```

WS_MAINTAIN_INDEXES

Synopsis

Drops and/or builds database indexes that are defined in the WhereScape RED metadata.

Description

Drops and/or builds indexes for a specified table or a specified index. Only indexes that are defined in the WhereScape RED metadata are supported. Typically, this routine is used by a WhereScape RED Scheduler and RED-generated procedures to automatically maintain indexes. However, it is also valid for user-defined custom procedures/scripts to execute this routine to control when indexes are dropped and/or created.

Input

Input	Description
Common Input	Includes all 5 inputs of the ***Callable Routines Common Input*** (on page 959).
Table Name	Table Name to process the relevant indexes of. **Note**: The Table Name is ignored when the optional Index Name is specified.
Optional Build Arguments **[ORACLE only]**	Optional arguments related to building indexes **[ORACLE only]**: TABLESPACE=XYZ will override the default tablespace name and use the specified tablespace name. e.g. 'TABLESPACE=non_default_tablespace'. Typically the TABLESPACE argument is used to build indexes in a partition-specific tablespace.
Index Name	Optional Index Name to only process the specified index. When NOT specified all the relevant indexes of the table are processed. **Note**: The Table Name is ignored when the Index Name is specified. **Note**: Must be specified to use the 'DROP' or 'BUILD' index actions.
Index Action	Action that specifies whether indexes are dropped or built and what types of indexes are applicable: **DROP** Drop the specified index (Index Name must be specified). **DROP ALL** Drop ALL the indexes of the table. **PRE DROP** Drop the indexes of the table that are defined as pre-drop. **BUILD** Build the specified index (Index Name must be specified). Otherwise, build all the indexes of the table that were pre-dropped. **BUILD ALL** Build ALL the indexes of the table.

Output

Output	Description
Result Number	Output Result Number: 1 Success. **-1 Warning.** -2 Error. -3 Fatal/Unexpected Error.

Note: Ws_Maintain_Indexes does NOT include a Return Code or Return Message like most of the WhereScape RED Callable routines but it does output a Result Number.

SQL SERVER

SQL Server Parameters: Ws_Maintain_Indexes

Callable Routine Type: PROCEDURE.

Parameter Name	Datatype	Mode
@p_sequence	INTEGER	IN
@p_job_name	VARCHAR(64)	IN
@p_task_name	VARCHAR(64)	IN
@p_job_id	INTEGER	IN
@p_task_id	INTEGER	IN
@p_table_name	VARCHAR(64)	IN
@p_parameter	VARCHAR(64)	IN
@p_index_name	VARCHAR(64)	IN
@p_option	VARCHAR(64)	IN
@p_result	INTEGER	OUT

SQL Server Examples: Ws_Maintain_Indexes

```
-- The p_ VARIABLES are normally PARAMETERS in a RED-generated Procedure.
DECLARE @p_sequence             integer
DECLARE @p_job_name             varchar(256)
DECLARE @p_task_name            varchar(256)
DECLARE @p_job_id               integer
DECLARE @p_task_id              integer
DECLARE @p_return_msg           varchar(256)
DECLARE @p_status               integer
DECLARE @v_result_num           integer
EXEC Ws_Maintain_Indexes
  @p_sequence, @p_job_name, @p_task_name, @p_job_id, @p_task_id
, 'load_customer', NULL, NULL, 'DROP ALL'
, @v_result_num  OUTPUT
```

TERADATA

Teradata Parameters: Ws_Maintain_Indexes

Callable Routine Type: PROCEDURE.

Parameter Name	Datatype	Mode
p_sequence	INTEGER	IN
p_job_name	VARCHAR(64)	IN
p_task_name	VARCHAR(64)	IN
p_job_id	INTEGER	IN
p_task_id	INTEGER	IN
p_table_name	VARCHAR(64)	IN
p_parameter	VARCHAR(4000)	IN
p_index_name	VARCHAR(64)	IN
p_option	VARCHAR(20)	IN
p_result	INTEGER	OUT

Teradata Examples: Ws_Maintain_Indexes

```
-- The p_ VARIABLES are normally PARAMETERS in a RED-generated Procedure.
DECLARE p_sequence              integer;
DECLARE p_job_name              varchar(256);
DECLARE p_task_name             varchar(256);
DECLARE p_job_id                integer;
DECLARE p_task_id               integer;
DECLARE p_return_msg            varchar(256);
DECLARE p_status                integer;
DECLARE v_result_num            integer;
CALL [METABASE].Ws_Maintain_Indexes
( p_sequence, p_job_name, p_task_name, p_job_id, p_task_id
, 'load_customer', NULL, NULL, 'DROP ALL'
, v_result_num
);
```

ORACLE

Oracle Parameters: Ws_Maintain_Indexes

Callable Routine Type: FUNCTION.

Parameter Name	Datatype	Mode
p_sequence	NUMBER	IN
p_job_name	VARCHAR2	IN
p_task_name	VARCHAR2	IN
p_job_id	NUMBER	IN
p_task_id	NUMBER	IN
p_table_name	VARCHAR2	IN
p_parameter	VARCHAR2	IN
p_index_name	VARCHAR2	IN
p_option	VARCHAR2	IN
FUNCTION Return Value	NUMBER	OUT-Function

Oracle Examples: Ws_Maintain_Indexes

```
-- The p_ VARIABLES are normally PARAMETERS in a RED-generated Procedure.
p_sequence                      number;
p_job_name                      varchar2(256);
p_task_name                     varchar2(256);
p_job_id                        number;
p_task_id                       number;
p_return_msg                    varchar2(256);
p_status                        number;
v_result_num                    number;
v_result_num := Ws_Maintain_Indexes
( p_sequence, p_job_name, p_task_name, p_job_id, p_task_id
, 'load_customer', NULL, NULL, 'DROP ALL'
);
```

DB2

DB2 Parameters: Ws_Maintain_Indexes

Callable Routine Type: PROCEDURE.

Parameter Name	Datatype	Mode
p_sequence	INTEGER	IN
p_job_name	VARCHAR(64)	IN
p_task_name	VARCHAR(64)	IN
p_job_id	INTEGER	IN
p_task_id	INTEGER	IN
p_table_name	VARCHAR(64)	IN
p_parameter	VARCHAR(4000)	IN
p_index_name	VARCHAR(64)	IN
p_option	VARCHAR(20)	IN
p_result	INTEGER	OUT

DB2 Examples: Ws_Maintain_Indexes

```
-- The p_ VARIABLES are normally PARAMETERS in a RED-generated Procedure.
DECLARE p_sequence              integer;
DECLARE p_job_name              varchar(256);
DECLARE p_task_name             varchar(256);
DECLARE p_job_id                integer;
DECLARE p_task_id               integer;
DECLARE p_return_msg            varchar(256);
DECLARE p_status                integer;
DECLARE v_result_num            integer;
CALL [METABASE].Ws_Maintain_Indexes
( p_sequence, p_job_name, p_task_name, p_job_id, p_task_id
, 'load_customer', NULL, NULL, 'DROP ALL'
, v_result_num
);
```

WS_VERSION_CLEAR

Synopsis

Purges metadata versions for all objects that do not meet the specified retention criteria.

Description

Deletes metadata versions for all objects that do not meet the specified retention criteria, which can be specified as the minimum number of versions to retain per object and/or the maximum age (in days) of versions to retain. For example, it is possible to specify that a minimum of 5 versions are retained for each object and/or that versions are retained for a maximum of 90 days.

Input

Input	Description
Common Input	Includes all 5 inputs of the ***Callable Routines Common Input*** (on page 959).
Maximum Days to Retain	The maximum age (in days) of the versions to retain. If 90 days are retained then all the versions that are older than 90 days are **purged/deleted** to reduce the number of versions to the specified maximum number of versions per object. If not specified then the retention date of each version determines whether it is deleted.
Minimum Versions per Object to Retain	The minimum number of versions to retain for each object. If 5 is specified then the last 5 versions are retained per object **regardless** of the specified maximum age to retain.
Options	Currently NOT used.

Output

Output	Description
Return Code	Output Return Code: S Success. E Error. F Fatal/Unexpected Error.
Return Message	Output message indicating the action applied or the reason for no action.
Result Number	Output Result Number: 1 Success. -2 Error. -3 Fatal/Unexpected Error.

SQL SERVER

SQL Server Parameters: Ws_Version_Clear

Callable Routine Type: PROCEDURE.

Parameter Name	Datatype	Mode
@p_sequence	INTEGER	IN
@p_job_name	VARCHAR(64)	IN
@p_task_name	VARCHAR(64)	IN
@p_job_id	INTEGER	IN
@p_task_id	INTEGER	IN
@p_day_count	INTEGER	IN
@p_keep_count	INTEGER	IN
@p_options	VARCHAR(256)	IN
@p_return_code	VARCHAR(1)	OUT
@p_return_msg	VARCHAR(256)	OUT
@p_result	INTEGER	OUT

SQL Server Examples: Ws_Version_Clear

```
-- The p_ VARIABLES are normally PARAMETERS in a RED-generated Procedure.
DECLARE @p_sequence             integer
DECLARE @p_job_name             varchar(256)
DECLARE @p_task_name            varchar(256)
DECLARE @p_job_id               integer
DECLARE @p_task_id              integer
DECLARE @p_return_msg           varchar(256)
DECLARE @p_status               integer
DECLARE @v_result_num           integer
DECLARE @v_return_code          varchar(1)
DECLARE @v_return_msg           varchar(256)
EXEC Ws_Version_Clear
  @p_sequence, @p_job_name, @p_task_name, @p_job_id, @p_task_id
, 90, 5, NULL
, @v_return_code OUTPUT
, @v_return_msg  OUTPUT
, @v_result_num  OUTPUT
```

TERADATA

Teradata Parameters: Ws_Version_Clear

Callable Routine Type: PROCEDURE.

Parameter Name	Datatype	Mode
p_sequence	INTEGER	IN
p_job_name	VARCHAR(64)	IN
p_task_name	VARCHAR(64)	IN
p_job_id	INTEGER	IN
p_task_id	INTEGER	IN
p_day_count	INTEGER	IN
p_keep_count	INTEGER	IN
p_options	VARCHAR(256)	IN
p_return_code	VARCHAR(1)	OUT
p_return_msg	VARCHAR(256)	OUT
p_result	INTEGER	OUT

Teradata Examples: Ws_Version_Clear

```
-- The p_ VARIABLES are normally PARAMETERS in a RED-generated Procedure.
DECLARE p_sequence              integer;
DECLARE p_job_name              varchar(256);
DECLARE p_task_name             varchar(256);
DECLARE p_job_id                integer;
DECLARE p_task_id               integer;
DECLARE p_return_msg            varchar(256);
DECLARE p_status                integer;
DECLARE v_result_num            integer;
DECLARE v_return_code           varchar(1);
DECLARE v_return_msg            varchar(256);
CALL [METABASE].Ws_Version_Clear
( p_sequence, p_job_name, p_task_name, p_job_id, p_task_id
, 90, 5, NULL
, v_return_code
, v_return_msg
, v_result_num
);
```

ORACLE

Oracle Parameters: Ws_Version_Clear

Callable Routine Type: FUNCTION.

Parameter Name	Datatype	Mode
p_sequence	NUMBER	IN
p_job_name	VARCHAR2	IN
p_task_name	VARCHAR2	IN
p_job_id	NUMBER	IN
p_task_id	NUMBER	IN
p_day_count	NUMBER	IN
p_keep_count	NUMBER	IN
p_options	VARCHAR2	IN
p_return_code	VARCHAR2	OUT
p_return_msg	VARCHAR2	OUT
FUNCTION Return Value	NUMBER	OUT-Function

Oracle Examples: Ws_Version_Clear

```
-- The p_ VARIABLES are normally PARAMETERS in a RED-generated Procedure.
p_sequence                      number;
p_job_name                      varchar2(256);
p_task_name                     varchar2(256);
p_job_id                        number;
p_task_id                       number;
p_return_msg                    varchar2(256);
p_status                        number;
v_result_num                    number;
v_return_code                   varchar2(1);
v_return_msg                    varchar2(256);
v_result_num := Ws_Version_Clear
( p_sequence, p_job_name, p_task_name, p_job_id, p_task_id
, 90, 5, NULL
, v_return_code
, v_return_msg
);
```

DB2

DB2 Parameters: Ws_Version_Clear

Callable Routine Type: PROCEDURE.

Parameter Name	Datatype	Mode
p_sequence	INTEGER	IN
p_job_name	VARCHAR(64)	IN
p_task_name	VARCHAR(64)	IN
p_job_id	INTEGER	IN
p_task_id	INTEGER	IN
p_day_count	INTEGER	IN
p_keep_count	INTEGER	IN
p_options	VARCHAR(256)	IN
p_return_code	VARCHAR(1)	OUT
p_return_msg	VARCHAR(256)	OUT
p_result	INTEGER	OUT

DB2 Examples: Ws_Version_Clear

```
-- The p_ VARIABLES are normally PARAMETERS in a RED-generated Procedure.
DECLARE p_sequence              integer;
DECLARE p_job_name              varchar(256);
DECLARE p_task_name             varchar(256);
DECLARE p_job_id                integer;
DECLARE p_task_id               integer;
DECLARE p_return_msg            varchar(256);
DECLARE p_status                integer;
DECLARE v_result_num            integer;
DECLARE v_return_code           varchar(1);
DECLARE v_return_msg            varchar(256);
CALL [METABASE].Ws_Version_Clear
( p_sequence, p_job_name, p_task_name, p_job_id, p_task_id
, 90, 5, NULL
, v_return_code
, v_return_msg
, v_result_num
);
```

WSPARAMETERREAD

Synopsis

Returns the value and comment (for most RDBMS) of a WhereScape RED metadata Parameter.

Description

Returns the value and comment (for most RDBMS) of the specified parameter from the DSS_PARAMETER metadata table. For SQL Server, Teradata, and DB2 this routine is a PROCEDURE that returns both the parameter value and comment. However, for Oracle this routine is a FUNCTION that only returns the parameter value. For SQL Server, there is also a WsParameterReadF FUNCTION.

Typically, this routine is used by procedures to read information that is written by another process (automatically or manually via the RED **Tools/Parameters** menu item), which is external to the procedure.

Input

Input	Description
Parameter Name	The case-sensitive name of the WhereScape RED metadata parameter to be retrieved. The name must exactly match an existing parameter, otherwise a NULL value is returned.

Output

Output	Description
Parameter Value	The retrieved value of the parameter. Corresponds to the "Value" property that is visible and maintainable via **Tools/Parameters**.
Parameter Comments	The maximum number of versions to retain for each object. If 5 is specified, then the last 5 versions are retained per object **regardless** of the specified maximum age to retain.

SQL SERVER

SQL Server Parameters: WsParameterRead

Callable Routine Type: PROCEDURE.

Parameter Name	Datatype	Mode
@p_parameter	VARCHAR(64)	IN
@p_value	VARCHAR(**2000**)	OUT
@p_comment	VARCHAR(256)	OUT

Note: There is also a corresponding WsParameterReadF FUNCTION available for SQL Server.

SQL Server Examples: WsParameterRead

```
DECLARE @v_current_date         varchar(4000) -- Same length as
DSS_PARAMETER.dss_parameter_value.
DECLARE @v_comment              varchar(256)
EXEC WsParameterRead 'CURRENT_DATE', @v_current_date OUTPUT, @v_comment
OUTPUT;
```

TERADATA

Teradata Parameters: WsParameterRead

Callable Routine Type: PROCEDURE.

Parameter Name	Datatype	Mode
p_parameter	VARCHAR(64)	IN
p_value	VARCHAR(**2000**)	OUT
p_comment	VARCHAR(256)	OUT

Teradata Examples: WsParameterRead

```
DECLARE v_current_date          varchar(4000); -- Same length as
DSS_PARAMETER.dss_parameter_value.
DECLARE v_comment               varchar(256);
CALL
[METABASE].WsParameterRead('CURRENT_DATE',v_current_date,v_comment);
```

ORACLE

Oracle Parameters: WsParameterRead

Callable Routine Type: FUNCTION.

Parameter Name	Datatype	Mode
v_parameter	VARCHAR2	IN
FUNCTION Return Value	VARCHAR2	OUT-Function

Oracle Examples: WsParameterRead

```
v_current_date                  varchar2(4000);    -- Same length as
DSS_PARAMETER.dss_parameter_value.
v_current_date := WsParameterRead('CURRENT_DATE'); -- NOTE: Does NOT
support comment output.
```

DB2

DB2 Parameters: WsParameterRead

Callable Routine Type: PROCEDURE.

Parameter Name	Datatype	Mode
p_parameter	VARCHAR(64)	IN
p_value	VARCHAR(**2000**)	OUT
p_comment	VARCHAR(256)	OUT

DB2 Examples: WsParameterRead

```
DECLARE v_current_date          varchar(4000); -- Same length as
DSS_PARAMETER.dss_parameter_value.
DECLARE v_comment               varchar(256);
CALL
[METABASE].WsParameterRead('CURRENT_DATE',v_current_date,v_comment);
```

WSPARAMETERREADF

Synopsis

Returns the value of a WhereScape RED metadata Parameter **[SQL Server only]**.

Description

Returns the value of the specified parameter from the DSS_PARAMETER metadata table. This routine is a FUNCTION that is only available for SQL Server. For SQL Server, Teradata, and DB2 there is a WsParameterRead PROCEDURE that returns both the parameter value and comment. For Oracle, the WsParameterRead FUNCTION is equivalent to the SQL Server WsParameterReadF FUNCTION.

Typically, this routine is used by procedures to read information that is written by another process (automatically or manually via the RED **Tools/Parameters** menu item), which is external to the procedure.

Input

Input	Description
Parameter Name	The name of the WhereScape RED metadata parameter to be retrieved. The case-sensitive name must exactly match an existing parameter, otherwise a NULL value is returned.

Output

Output	Description
Parameter Value	The value of the parameter. Corresponds to the "Value" property that is visible and maintainable via **Tools/Parameters**.

SQL SERVER

SQL Server Parameters: WsParameterReadF

Callable Routine Type: FUNCTION.

Parameter Name	Datatype	Mode
@p_parameter	VARCHAR(64)	IN
FUNCTION Return Value	VARCHAR(4000)	OUT-Function

Note: There is also a corresponding WsParameterRead PROCEDURE available for SQL Server.

SQL Server Examples: WsParameterReadF

```
DECLARE @v_current_date          varchar(4000)
SELECT @v_current_date = dbo.WsParameterReadF('CURRENT_DATE')
```

WSPARAMETERREADG

Synopsis

Returns the value of a "global" WhereScape RED metadata Parameter that relates to a load table.

Description

Returns the value of an internal parameter that is defined and populated by WhereScape RED, which is available to a procedure that is currently processing a load table.

The supported parameters are $$TABLE_NAME and $$SOURCE_TABLE.

Input

Input	Description
Global Parameter Name	The supported global parameters are: **$$TABLE_NAME** returns the Load Table Name that the procedure is executing against. Only available for load tables. **$$SOURCE_TABLE** returns the maximum value of the Source Table property from the columns of the Load Table that the procedure is executing against. Only available for load tables. Note: Typically, a Load Table has a single Source Table but if it has multiple sources then the maximum (alphabetically) Source Table Name will be returned.
Job Identifier	Unique identifier of the **held or scheduled job** that the running job is a specific instance of. When invoked from a WhereScape RED Scheduler the routine will be passed the parameter argument. When invoked manually or externally to the WhereScape RED Scheduler, it is recommended to use 0 (zero).
Task or Object Identifier	Unique identifier of the **running task** (of a running job) that executed the routine. When invoked from a WhereScape RED Scheduler the routine will be passed the parameter argument. When invoked manually or externally to the WhereScape RED Scheduler, it should be the **object key**.

Output

Output	Description
Result Table Name	The requested Load Table Name or Source Table Name.

SQL SERVER

SQL Server Parameters: WsParameterReadG

Callable Routine Type: FUNCTION.

Parameter Name	Datatype	Mode
@p_parameter	VARCHAR(64)	IN
@p_job_id	INTEGER	IN
@p_task_id	INTEGER	IN
FUNCTION Return Value	VARCHAR(4000)	OUT-Function

SQL Server Examples: WsParameterReadG

```
-- The p_ VARIABLES are normally PARAMETERS in a RED-generated Procedure.
DECLARE @p_job_id               integer
DECLARE @p_task_id              integer
DECLARE @v_source_name          varchar(64)
SELECT @v_source_name = dbo.WsParameterReadG('$$SOURCE_TABLE', @p_job_id,
@p_task_id) -- Needs dbo.
```

TERADATA

Teradata Parameters: WsParameterReadG

Callable Routine Type: PROCEDURE.

Parameter Name	Datatype	Mode
p_parameter	VARCHAR(64)	IN
p_job_id	INTEGER	IN
p_task_id	INTEGER	IN
p_value	VARCHAR(2000)	OUT

Teradata Examples: WsParameterReadG

```
-- The p_ VARIABLES are normally PARAMETERS in a RED-generated Procedure.
DECLARE p_job_id                integer;
DECLARE p_task_id               integer;
DECLARE v_source_name           varchar(256);
CALL [METABASE].WsParameterReadG('$$SOURCE_TABLE', p_job_id, p_task_id,
v_source_name);
```

ORACLE

Oracle Parameters: WsParameterReadG

Callable Routine Type: FUNCTION.

Parameter Name	Datatype	Mode
p_parameter	VARCHAR2	IN
p_job_id	NUMBER	IN
p_task_id	NUMBER	IN
FUNCTION Return Value	VARCHAR2	OUT-Function

Oracle Examples: WsParameterReadG

```
-- The p_ VARIABLES are normally PARAMETERS in a RED-generated Procedure.
p_job_id                        number;
p_task_id                       number;
v_source_name                   varchar2(64);
v_source_name := WsParameterReadG('$$SOURCE_TABLE', p_job_id, p_task_id);
```

DB2

DB2 Parameters: WsParameterReadG

Callable Routine Type: PROCEDURE.

Parameter Name	Datatype	Mode
p_parameter	VARCHAR(64)	IN
p_job_id	INTEGER	IN
p_task_id	INTEGER	IN
p_value	VARCHAR(2000)	OUT

DB2 Examples: WsParameterReadG

```
-- The p_ VARIABLES are normally PARAMETERS in a RED-generated Procedure.
DECLARE p_job_id                integer;
DECLARE p_task_id               integer;
DECLARE v_source_name           varchar(256);
CALL [METABASE].WsParameterReadG('$$SOURCE_TABLE', p_job_id, p_task_id,
v_source_name);
```

WSPARAMETERWRITE

Synopsis

Updates the value and comment of a WhereScape RED metadata Parameter or creates it.

Description

Updates the value and comment of the specified parameter in the DSS_PARAMETER metadata table. If the specified parameter is not found then it is added.

Typically, this routine is used by procedures to write information that is read by another process (automatically or manually via the RED **Tools/Parameters** menu item), which is external to the procedure.

Input

Input	Description
Parameter Name	The case-sensitive name of the WhereScape RED metadata parameter to be updated or added.
Parameter Value	The new value of the parameter to be assigned. Corresponds to the "Value" property that is visible and maintainable via **Tools/Parameters**.
Parameter Comments	The new comments of the parameter to be assigned. Corresponds to the "Comments" property that is visible and maintainable via **Tools/Parameters**. The parameter comments will not be modified if a NULL value is specified.

Output

Output	Description
Result Number **[SQL SERVER & ORACLE only]**	Output Result Number **[SQL SERVER & ORACLE only]**: **1 Metadata Parameter Updated.** **2 Metadata Parameter Inserted.** -3 Fatal/Unexpected Error.

SQL SERVER

SQL Server Parameters: WsParameterWrite

Callable Routine Type: PROCEDURE.

Parameter Name	Datatype	Mode
@p_parameter	VARCHAR(64)	IN
@p_value	VARCHAR(2000)	IN
@p_comment	VARCHAR(256)	IN

SQL Server Examples: WsParameterWrite

```
-- Example 1: Result Number (procedure return code) NOT captured.
EXEC WsParameterWrite 'LAST_INVOICE_ID', '123456', 'The last invoice id
loaded'

-- Example 2: Result Number (procedure return code) captured.
DECLARE @v_result_num           integer
EXEC @v_result_num = WsParameterWrite 'LAST_INVOICE_ID', '123456', 'The
last invoice id loaded'
```

TERADATA

Teradata Parameters: WsParameterWrite

Callable Routine Type: PROCEDURE.

Parameter Name	Datatype	Mode
p_parameter	VARCHAR(64)	IN
p_value	VARCHAR(2000)	IN
p_comment	VARCHAR(256)	IN

Teradata Examples: WsParameterWrite

CALL **[METABASE]**.WsParameterWrite('LAST_INVOICE_ID', '123456', 'The last invoice id loaded');

ORACLE

Oracle Parameters: WsParameterWrite

Callable Routine Type: FUNCTION.

Parameter Name	Datatype	Mode
v_parameter	VARCHAR2	IN
v_value	VARCHAR2	IN
v_comment	VARCHAR2	IN
FUNCTION Return Value	VARCHAR2	OUT-Function

Oracle Examples: WsParameterWrite

```
v_result_num                    number;
v_result_num := WsParameterWrite('LAST_INVOICE_ID', '123456', 'The last
invoice id loaded');
```

DB2

DB2 Parameters: WsParameterWrite

Callable Routine Type: PROCEDURE.

Parameter Name	Datatype	Mode
p_parameter	VARCHAR(64)	IN
p_value	VARCHAR(2000)	IN
p_comment	VARCHAR(256)	IN

DB2 Examples: WsParameterWrite

```
CALL [METABASE].WsParameterWrite('LAST_INVOICE_ID', '123456', 'The last
invoice id loaded');
```

WSWRKAUDIT

Synopsis

Records a message in the Audit Log.

Description

Adds the specified message to the WS_WRK_AUDIT_LOG workflow metadata table, which is referred to as the Audit Log or Audit Trail. A variety of message types are supported such as Information, Warning, and Error that are included in the corresponding message type counts for the task and job. Audit Log messages are accessible via the "Scheduler" tab/window and/or the WS_ADMIN_V_AUDIT view of the WS_WRK_AUDIT_LOG table.

NOTE: Both the Audit Log and Error/Detail Log support similar information and in user-defined custom procedures either or both logs can be used. However, in RED-generated procedures/scripts the Audit Log is used for higher-level or summary messages while the Error/Detail Log is used for more detailed supporting information.

Input

Input	Description
Common Input	Includes all 5 inputs of the ***Callable Routines Common Input*** (on page 959). **Note**: Refer to the RDBMS-specific parameters for the relative positions (they are NOT declared as the first parameters)
Audit Message Type Code	Audit Message Type Code: B Beginning of a Job or Task. I Information. S Success. W Warning. E Error. F Fatal Error.
Audit Message Text	Custom message text to be recorded in the WhereScape RED Audit Log.
RDBMS Code	RDBMS-specific message code. e.g. The Oracle special variable SQLCODE. It is optional but recommended to populate this when an error occurs.
RDBMS Message	RDBMS-specific message. e.g. The Oracle special variable SQLERRM. It is optional but recommended to populate this when an error occurs.

Output

Output	Description
Result Number	Output Result Number: 1 Success. **-3 Error.**

SQL SERVER

SQL Server Parameters: WsWrkAudit

Callable Routine Type: PROCEDURE.

Parameter Name	Datatype	Mode
@p_status_code	VARCHAR(1)	IN
@p_job_name	VARCHAR(64)	IN
@p_task_name	VARCHAR(64)	IN
@p_sequence	INTEGER	IN
@p_message	VARCHAR(256)	IN
@p_db_code	VARCHAR(10)	IN
@p_db_msg	VARCHAR(256)	IN
@p_task_key	INTEGER	IN
@p_job_key	INTEGER	IN

SQL Server Examples: WsWrkAudit

```
-- The p_ VARIABLES are normally PARAMETERS in a RED-generated Procedure.
DECLARE @p_sequence             integer
DECLARE @p_job_name             varchar(256)
DECLARE @p_task_name            varchar(256)
DECLARE @p_job_id               integer
DECLARE @p_task_id              integer
DECLARE @p_return_msg           varchar(256)
DECLARE @p_status               integer
DECLARE @v_result_num           integer
EXEC @v_result_num = WsWrkAudit
   'I', @p_job_name, @p_task_name, @p_sequence
, 'The task has started.'
, NULL
, NULL
, @p_task_id
, @p_job_id
```

TERADATA

Teradata Parameters: WsWrkAudit

Callable Routine Type: PROCEDURE.

Parameter Name	Datatype	Mode
p_status_code	VARCHAR(1)	IN
p_job_name	VARCHAR(64)	IN
p_task_name	VARCHAR(64)	IN
p_sequence	INTEGER	IN
p_message	VARCHAR(255)	IN
p_db_code	VARCHAR(10)	IN
p_db_msg	VARCHAR(255)	IN
p_task_key	INTEGER	IN
p_job_key	INTEGER	IN

Teradata Examples: WsWrkAudit

```
-- The p_ VARIABLES are normally PARAMETERS in a RED-generated Procedure.
DECLARE p_sequence              integer;
DECLARE p_job_name              varchar(256);
DECLARE p_task_name             varchar(256);
DECLARE p_job_id                integer;
DECLARE p_task_id               integer;
DECLARE p_return_msg            varchar(256);
DECLARE p_status                integer;
CALL [METABASE].WsWrkAudit
( 'I', p_job_name, p_task_name, p_sequence
, 'The task has started.'
, NULL
, NULL
, p_task_id
, p_job_id
);
```

ORACLE

Oracle Parameters: WsWrkAudit

Callable Routine Type: FUNCTION.

Parameter Name	Datatype	Mode
v_status_code	VARCHAR2	IN
v_job_name	VARCHAR2	IN
v_task_name	VARCHAR2	IN
v_sequence	NUMBER	IN
v_message	VARCHAR2	IN
v_db_code	VARCHAR2	IN
v_db_msg	VARCHAR2	IN
v_task_key	NUMBER	IN
v_job_key	NUMBER	IN
FUNCTION Return Value	NUMBER	OUT-Function

Oracle Examples: WsWrkAudit

```
-- The p_ VARIABLES are normally PARAMETERS in a RED-generated Procedure.
p_sequence                      number;
p_job_name                      varchar2(256);
p_task_name                     varchar2(256);
p_job_id                        number;
p_task_id                       number;
p_return_msg                    varchar2(256);
p_status                        number;
v_result_num                    number;
v_result_num := WsWrkAudit
( 'I', p_job_name, p_task_name, p_sequence
, 'The task has started.'
, NULL
, NULL
, p_task_id
, p_job_id
);
```

DB2

DB2 Parameters: WsWrkAudit

Callable Routine Type: PROCEDURE.

Parameter Name	Datatype	Mode
v_status_code	CHARACTER	IN
v_job_name	VARCHAR(64)	IN
v_task_name	VARCHAR(64)	IN
v_sequence	INTEGER	IN
v_message	VARCHAR(256)	IN
v_db_code	VARCHAR(10)	IN
v_db_msg	VARCHAR(256)	IN
v_task_key	INTEGER	IN
v_job_key	INTEGER	IN

DB2 Examples: WsWrkAudit

```
-- The p_ VARIABLES are normally PARAMETERS in a RED-generated Procedure.
DECLARE p_sequence              integer;
DECLARE p_job_name              varchar(256);
DECLARE p_task_name             varchar(256);
DECLARE p_job_id                integer;
DECLARE p_task_id               integer;
DECLARE p_return_msg            varchar(256);
DECLARE p_status                integer;
CALL [METABASE].WsWrkAudit
( 'I', p_job_name, p_task_name, p_sequence
, 'The task has started.'
, NULL
, NULL
, p_task_id
, p_job_id
);
```

WSWRKAUDITBULK

Synopsis

Records multiple messages in the Audit Log.

Description

Adds the specified multiple messages to the WS_WRK_AUDIT_LOG workflow metadata table, which is referred to as the Audit Log or Audit Trail. A variety of message types are supported such as Information, Warning, and Error that are included in the corresponding message type counts for the task and job. Audit Log messages are accessible via the "Scheduler" tab/window and/or the WS_ADMIN_V_AUDIT view of the WS_WRK_AUDIT_LOG table.

NOTE: Both the Audit Log and Error/Detail Log support similar information and in user-defined custom procedures either or both logs can be used. However, in RED-generated procedures/scripts the Audit Log is used for higher-level or summary messages while the Error/Detail Log is used for more detailed supporting information.

Input

Input	Description
Common Input	Includes all 5 inputs of the ***Callable Routines Common Input*** (on page 959). **Note**: Refer to the RDBMS-specific parameters for the relative positions (they are NOT declared as the first parameters)
Audit Message Type Code	Audit Message Type Code: B Beginning of a Job or Task. I Information. S Success. W Warning. E Error. F Fatal Error.
Audit Message(s) Text	Custom message(s) text to be recorded in the WhereScape RED Audit Log. Multiple messages can be specified but each is limited to 256 characters. Each message must be separated by either a new-line (ASCII 10) or tilde (~) character. e.g. Message1~Message2~Message3 will create 3 messages.
RDBMS Code	RDBMS-specific message code. e.g. The Oracle special variable SQLCODE. It is optional but recommended to populate this when an error occurs.
RDBMS Message	RDBMS-specific message. e.g. The Oracle special variable SQLERRM. It is optional but recommended to populate this when an error occurs.

Output

Output	Description
Result Number	**Note**: Not provided for all RDBMS. Output Result Number: 1 Success. -2 Error -3 Fatal/Unexpected Error.

SQL SERVER

SQL Server Parameters: WsWrkAuditBulk

Callable Routine Type: PROCEDURE.

Parameter Name	Datatype	Mode
@p_status_code	VARCHAR(1)	IN
@p_job_name	VARCHAR(64)	IN
@p_task_name	VARCHAR(64)	IN
@p_sequence	INTEGER	IN
@p_message	VARCHAR(4000)	IN
@p_db_code	VARCHAR(10)	IN
@p_db_msg	VARCHAR(256)	IN
@p_task_key	INTEGER	IN
@p_job_key	INTEGER	IN

SQL Server Examples: WsWrkAuditBulk

```
-- The p_ VARIABLES are normally PARAMETERS in a RED-generated Procedure.
DECLARE @p_sequence             integer
DECLARE @p_job_name             varchar(256)
DECLARE @p_task_name            varchar(256)
DECLARE @p_job_id               integer
DECLARE @p_task_id              integer
DECLARE @p_return_msg           varchar(256)
DECLARE @p_status               integer
DECLARE @v_result_num           integer
EXEC @v_result_num = WsWrkAuditBulk
   'I', @p_job_name, @p_task_name, @p_sequence
, 'Message1~Message2~Message3'
, NULL
, NULL
, @p_job_id
, @p_task_id
```

TERADATA

Teradata Parameters: WsWrkAuditBulk

Callable Routine Type: PROCEDURE.

Parameter Name	Datatype	Mode
p_status_code	VARCHAR(64)	IN
p_job_name	VARCHAR(64)	IN
p_task_name	VARCHAR(64)	IN
p_sequence	INTEGER	IN
p_message	VARCHAR(61440)	IN
p_db_code	VARCHAR(10)	IN
p_db_msg	VARCHAR(256)	IN
p_job_id	INTEGER	IN
p_task_id	INTEGER	IN
p_result	INTEGER	OUT

Teradata Examples: WsWrkAuditBulk

```
-- The p_ VARIABLES are normally PARAMETERS in a RED-generated Procedure.
DECLARE p_sequence              integer;
DECLARE p_job_name              varchar(256);
DECLARE p_task_name             varchar(256);
DECLARE p_job_id                integer;
DECLARE p_task_id               integer;
DECLARE p_return_msg            varchar(256);
DECLARE p_status                integer;
DECLARE v_result_num            integer;
CALL [METABASE].WsWrkAuditBulk
( 'I', p_job_name, p_task_name, p_sequence
, 'Message1~Message2~Message3'
, NULL
, NULL
, p_job_id   --### NOTE order.
, p_task_id  --### NOTE order.
, v_result_num
);
```

ORACLE

Oracle Parameters: WsWrkAuditBulk

Callable Routine Type: FUNCTION.

Parameter Name	Datatype	Mode
v_status_code	VARCHAR2	IN
v_job_name	VARCHAR2	IN
v_task_name	VARCHAR2	IN
v_sequence	NUMBER	IN
v_message	VARCHAR2	IN
v_db_code	VARCHAR2	IN
v_db_msg	VARCHAR2	IN
v_task_key	NUMBER	IN
v_job_key	NUMBER	IN
FUNCTION Return Value	NUMBER	OUT-Function

Oracle Examples: WsWrkAuditBulk

```
-- The p_ VARIABLES are normally PARAMETERS in a RED-generated Procedure.
p_sequence                      number;
p_job_name                      varchar2(256);
p_task_name                     varchar2(256);
p_job_id                        number;
p_task_id                       number;
p_return_msg                    varchar2(256);
p_status                        number;
v_result_num                    number;
v_result_num := WsWrkAuditBulk
( 'I', p_job_name, p_task_name, p_sequence
, 'Message1~Message2~Message3'
, NULL
, NULL
, p_task_id
, p_job_id
);
```

DB2

DB2 Parameters: WsWrkAuditBulk

Callable Routine Type: PROCEDURE.

Parameter Name	Datatype	Mode
p_status_code	VARCHAR(64)	IN
p_job_name	VARCHAR(64)	IN
p_task_name	VARCHAR(64)	IN
p_sequence	INTEGER	IN
p_message	VARCHAR(32672)	IN
p_db_code	VARCHAR(10)	IN
p_db_msg	VARCHAR(256)	IN
p_job_id	INTEGER	IN
p_task_id	INTEGER	IN
p_result	INTEGER	OUT

DB2 Examples: WsWrkAuditBulk

```
-- The p_ VARIABLES are normally PARAMETERS in a RED-generated Procedure.
DECLARE p_sequence              integer;
DECLARE p_job_name              varchar(256);
DECLARE p_task_name             varchar(256);
DECLARE p_job_id                integer;
DECLARE p_task_id               integer;
DECLARE p_return_msg            varchar(256);
DECLARE p_status                integer;
DECLARE v_result_num            integer;
CALL [METABASE].WsWrkAuditBulk
( 'I', p_job_name, p_task_name, p_sequence
, 'Message1~Message2~Message3'
, NULL
, NULL
, p_job_id   --### NOTE order.
, p_task_id  --### NOTE order.
, v_result_num
);
```

WSWRKERROR

Synopsis

Records a message in the Error/Detail Log.

Description

Adds the specified message to the WS_WRK_ERROR_LOG workflow metadata table, which is referred to as the Error Log or Detail Log. A variety of message types are supported such as Information, Warning, and Error that are included in the "detail" message counts for the task and job (viewable via the "Scheduler" tab/window). Error/Detail Log messages are accessible via the "Scheduler" tab/window and/or the WS_ADMIN_V_ERROR view of the WS_WRK_ERROR_LOG table.

NOTE: Both the Audit Log and Error/Detail Log support similar information and in user-defined custom procedures either or both logs can be used. However, in RED-generated procedures/scripts the Audit Log is used for higher-level or summary messages while the Error/Detail Log is used for more detailed supporting information.

Input

Input	Description
Common Input	Includes all 5 inputs of the ***Callable Routines Common Input*** (on page 959). **Note**: Refer to the RDBMS-specific parameters for the relative positions (they are NOT declared as the first parameters)
Error/Detail Message Type Code	Error/Detail Message Type Code: E Error. I Information. W Warning.
Error/Detail Message Text	Custom message text to be recorded in the WhereScape RED Error/Detail Log.
RDBMS Code	RDBMS-specific message code. e.g. The Oracle special variable SQLCODE. It is optional but recommended to populate this when an error occurs.
RDBMS Message	RDBMS-specific message. e.g. The Oracle special variable SQLERRM. It is optional but recommended to populate this when an error occurs.
Custom Message Type Code	Custom Message Type Code. For custom usage and has no meaning within the WhereScape RED metadata.

Output

Output	Description
Result Number	Output Result Number: 1 Success. **-3 Error.**

SQL SERVER

SQL Server Parameters: WsWrkError

Callable Routine Type: PROCEDURE.

Parameter Name	Datatype	Mode
@p_status_code	VARCHAR(1)	IN
@p_job_name	VARCHAR(64)	IN
@p_task_name	VARCHAR(64)	IN
@p_sequence	INTEGER	IN
@p_message	VARCHAR(256)	IN
@p_db_code	VARCHAR(10)	IN
@p_db_msg	VARCHAR(256)	IN
@p_task_key	INTEGER	IN
@p_job_key	INTEGER	IN
@p_msg_type	VARCHAR(10)	IN

SQL Server Examples: WsWrkError

```
-- The p_ VARIABLES are normally PARAMETERS in a RED-generated Procedure.
DECLARE @p_sequence             integer
DECLARE @p_job_name             varchar(256)
DECLARE @p_task_name            varchar(256)
DECLARE @p_job_id               integer
DECLARE @p_task_id              integer
DECLARE @p_return_msg           varchar(256)
DECLARE @p_status               integer
DECLARE @v_result_num           integer
EXEC @v_result_num = WsWrkError
   'I', @p_job_name, @p_task_name, @p_sequence
, 'This is an INFO message in the Error/Detail Log.'
, NULL
, NULL
, @p_task_id
, @p_job_id
, NULL
```

TERADATA

Teradata Parameters: WsWrkError

Callable Routine Type: PROCEDURE.

Parameter Name	Datatype	Mode
p_status_code	VARCHAR(1)	IN
p_job_name	VARCHAR(64)	IN
p_task_name	VARCHAR(64)	IN
p_sequence	INTEGER	IN
p_message	VARCHAR(255)	IN
p_db_code	VARCHAR(10)	IN
p_db_msg	VARCHAR(255)	IN
p_task_key	INTEGER	IN
p_job_key	INTEGER	IN
p_msg_type	VARCHAR(10)	IN

Teradata Examples: WsWrkError

```
-- The p_ VARIABLES are normally PARAMETERS in a RED-generated Procedure.
DECLARE p_sequence              integer;
DECLARE p_job_name              varchar(256);
DECLARE p_task_name             varchar(256);
DECLARE p_job_id                integer;
DECLARE p_task_id               integer;
DECLARE p_return_msg            varchar(256);
DECLARE p_status                integer;
CALL [METABASE].WsWrkError
( 'I', p_job_name, p_task_name, p_sequence
, 'This is an INFO message in the Error/Detail Log.'
, NULL
, NULL
, p_task_id
, p_job_id
, NULL
);
```

ORACLE

Oracle Parameters: WsWrkError

Callable Routine Type: FUNCTION.

Parameter Name	Datatype	Mode
v_status_code	VARCHAR2	IN
v_job_name	VARCHAR2	IN
v_task_name	VARCHAR2	IN
v_sequence	NUMBER	IN
v_message	VARCHAR2	IN
v_db_code	VARCHAR2	IN
v_db_msg	VARCHAR2	IN
v_task_key	NUMBER	IN
v_job_key	NUMBER	IN
v_msg_type	VARCHAR2	IN
FUNCTION Return Value	NUMBER	OUT-Function

Oracle Examples: WsWrkError

```
-- The p_ VARIABLES are normally PARAMETERS in a RED-generated Procedure.
p_sequence                      number;
p_job_name                      varchar2(256);
p_task_name                     varchar2(256);
p_job_id                        number;
p_task_id                       number;
p_return_msg                    varchar2(256);
p_status                        number;
v_result_num                    number;
v_result_num := WsWrkError
( 'I', p_job_name, p_task_name, p_sequence
, 'This is an INFO message in the Error/Detail Log.'
, NULL
, NULL
, p_task_id
, p_job_id
, NULL
);
```

DB2

DB2 Parameters: WsWrkError

Callable Routine Type: PROCEDURE.

Parameter Name	Datatype	Mode
v_status_code	CHARACTER	IN
v_job_name	VARCHAR(64)	IN
v_task_name	VARCHAR(64)	IN
v_sequence	INTEGER	IN
v_message	VARCHAR(256)	IN
v_db_code	VARCHAR(10)	IN
v_db_msg	VARCHAR(256)	IN
v_task_key	INTEGER	IN
v_job_key	INTEGER	IN
v_msg_type	VARCHAR(10)	IN

DB2 Examples: WsWrkError

```
-- The p_ VARIABLES are normally PARAMETERS in a RED-generated Procedure.
DECLARE p_sequence              integer;
DECLARE p_job_name              varchar(256);
DECLARE p_task_name             varchar(256);
DECLARE p_job_id                integer;
DECLARE p_task_id               integer;
DECLARE p_return_msg            varchar(256);
DECLARE p_status                integer;
CALL [METABASE].WsWrkError
( 'I', p_job_name, p_task_name, p_sequence
, 'This is an INFO message in the Error/Detail Log.'
, NULL
, NULL
, p_task_id
, p_job_id
, NULL
);
```

WSWRKERRORBULK

Synopsis

Records multiple messages in the Error/Detail Log.

Description

Adds the specified multiple messages to the WS_WRK_ERROR_LOG workflow metadata table, which is referred to as the Error Log or Detail Log. A variety of message types are supported such as Information, Warning, and Error that are included in the "detail" message counts for the task and job (viewable via the "Scheduler" tab/window). Error/Detail Log messages are accessible via the "Scheduler" tab/window and/or the WS_ADMIN_V_ERROR view of the WS_WRK_ERROR_LOG table.

NOTE: Both the Audit Log and Error/Detail Log support similar information and in user-defined custom procedures either or both logs can be used. However, in RED-generated procedures/scripts the Audit Log is used for higher-level or summary messages while the Error/Detail Log is used for more detailed supporting information.

Input

Input	Description
Common Input	Includes all 5 inputs of the ***Callable Routines Common Input*** (on page 959). **Note**: Refer to the RDBMS-specific parameters for the relative positions (they are NOT declared as the first parameters)
Error/Detail Message Type Code	Error/Detail Message Type Code: E Error. I Information. W Warning.
Error/Detail Message(s) Text	Custom message(s) text to be recorded in the WhereScape RED Error/Detail Log. Multiple messages can be specified but each is limited to 256 characters. Each message must be separated by either a new-line (ASCII 10) or tilde (~) character. e.g. Message1~Message2~Message3 will create 3 messages.
RDBMS Code	RDBMS-specific message code. e.g. The Oracle special variable SQLCODE. It is optional but recommended to populate this when an error occurs.
RDBMS Message	RDBMS-specific message. e.g. The Oracle special variable SQLERRM. It is optional but recommended to populate this when an error occurs.

Input	Description
Custom Message Type Code	Custom Message Type Code. For custom usage and has no meaning within the WhereScape RED metadata.

Output

Output	Description
Result Number	**Note**: NOT provided for all RDBMS. Output Result Number: 1 Success. -2 Error. -3 Fatal/Unexpected Error.

SQL SERVER

SQL Server Parameters: WsWrkErrorBulk

Callable Routine Type: PROCEDURE.

Parameter Name	Datatype	Mode
@p_status_code	VARCHAR(1)	IN
@p_job_name	VARCHAR(64)	IN
@p_task_name	VARCHAR(64)	IN
@p_sequence	INTEGER	IN
@p_message	VARCHAR(4000)	IN
@p_db_code	VARCHAR(10)	IN
@p_db_msg	VARCHAR(256)	IN
@p_task_key	INTEGER	IN
@p_job_key	INTEGER	IN
@p_msg_type	VARCHAR(10)	IN

SQL Server Examples: WsWrkErrorBulk

```
-- The p_ VARIABLES are normally PARAMETERS in a RED-generated Procedure.
DECLARE @p_sequence             integer
DECLARE @p_job_name             varchar(256)
DECLARE @p_task_name            varchar(256)
DECLARE @p_job_id               integer
DECLARE @p_task_id              integer
DECLARE @p_return_msg           varchar(256)
DECLARE @p_status               integer
DECLARE @v_result_num           integer
EXEC @v_result_num = WsWrkErrorBulk
   'I', @p_job_name, @p_task_name, @p_sequence
, 'Message1~Message2~Message3'
, NULL
, NULL
, @p_job_id
, @p_task_id
, NULL
```

TERADATA

Teradata Parameters: WsWrkErrorBulk

Callable Routine Type: PROCEDURE.

Parameter Name	Datatype	Mode
p_status_code	VARCHAR(64)	IN
p_job_name	VARCHAR(64)	IN
p_task_name	VARCHAR(64)	IN
p_sequence	INTEGER	IN
p_message	VARCHAR(61440)	IN
p_db_code	VARCHAR(10)	IN
p_db_msg	VARCHAR(256)	IN
p_job_id	INTEGER	IN
p_task_id	INTEGER	IN
p_msg_type	VARCHAR(10)	IN
p_result	INTEGER	OUT

Teradata Examples: WsWrkErrorBulk

```
-- The p_ VARIABLES are normally PARAMETERS in a RED-generated Procedure.
DECLARE p_sequence              integer;
DECLARE p_job_name              varchar(256);
DECLARE p_task_name             varchar(256);
DECLARE p_job_id                integer;
DECLARE p_task_id               integer;
DECLARE p_return_msg            varchar(256);
DECLARE p_status                integer;
DECLARE v_result_num            integer;
CALL [METABASE].WsWrkErrorBulk
( 'I', p_job_name, p_task_name, p_sequence
, 'Message1~Message2~Message3'
, NULL
, NULL
, p_job_id   --### NOTE order.
, p_task_id  --### NOTE order.
, NULL
, v_result_num
);
```

ORACLE

Oracle Parameters: WsWrkErrorBulk

Callable Routine Type: FUNCTION.

Parameter Name	Datatype	Mode
v_status_code	VARCHAR2	IN
v_job_name	VARCHAR2	IN
v_task_name	VARCHAR2	IN
v_sequence	NUMBER	IN
v_message	VARCHAR2	IN
v_db_code	VARCHAR2	IN
v_db_msg	VARCHAR2	IN
v_task_key	NUMBER	IN
v_job_key	NUMBER	IN
v_msg_type	VARCHAR2	IN
FUNCTION Return Value	NUMBER	OUT-Function

Oracle Examples: WsWrkErrorBulk

```
-- The p_ VARIABLES are normally PARAMETERS in a RED-generated Procedure.
p_sequence                      number;
p_job_name                      varchar2(256);
p_task_name                     varchar2(256);
p_job_id                        number;
p_task_id                       number;
p_return_msg                    varchar2(256);
p_status                        number;
v_result_num                    number;
v_result_num := WsWrkErrorBulk
( 'I', p_job_name, p_task_name, p_sequence
, 'Message1~Message2~Message3'
, NULL
, NULL
, p_task_id
, p_job_id
, NULL
);
```

DB2

DB2 Parameters: WsWrkErrorBulk

Callable Routine Type: PROCEDURE.

Parameter Name	Datatype	Mode
p_status_code	VARCHAR(64)	IN
p_job_name	VARCHAR(64)	IN
p_task_name	VARCHAR(64)	IN
p_sequence	INTEGER	IN
p_message	VARCHAR(32672)	IN
p_db_code	VARCHAR(10)	IN
p_db_msg	VARCHAR(256)	IN
p_job_id	INTEGER	IN
p_task_id	INTEGER	IN
p_msg_type	VARCHAR(10)	IN
p_result	INTEGER	OUT

DB2 Examples: WsWrkErrorBulk

```
-- The p_ VARIABLES are normally PARAMETERS in a RED-generated Procedure.
DECLARE p_sequence              integer;
DECLARE p_job_name              varchar(256);
DECLARE p_task_name             varchar(256);
DECLARE p_job_id                integer;
DECLARE p_task_id               integer;
DECLARE p_return_msg            varchar(256);
DECLARE p_status                integer;
DECLARE v_result_num            integer;
CALL [METABASE].WsWrkErrorBulk
( 'I', p_job_name, p_task_name, p_sequence
, 'Message1~Message2~Message3'
, NULL
, NULL
, p_job_id   --### NOTE order.
, p_task_id  --### NOTE order.
, NULL
, v_result_num
);
```

WSWRKTASK

Synopsis

Updates row counts for a task in the Task Log.

Description

Updates row counts for the specified task in the Task Log. Task Log messages (and row counts) are accessible via the "Scheduler" tab/window and/or the WS_ADMIN_V_TASK view of the WS_WRK_TASK_RUN and WS_WRK_TASK_LOG tables.

This routine is intended to be executed by a task of a job since it requires a valid job, task, and job sequence number that are provided by a WhereScape RED Scheduler.

Input

Input	Description
Common Input	Includes **3 inputs** of the ***Callable Routines Common Input*** (on page 959).
Inserted Row Count	The number of rows inserted by the task.
Updated Row Count	The number of rows updated by the task.
Replaced Row Count	The number of rows replaced by the task.
Deleted Row Count	The number of rows deleted by the task.
Discarded Row Count	The number of rows discarded by the task.
Rejected Row Count	The number of rows rejected by the task.
Error Row Count	The number of rows with an error that were failed by the task.

Output

Output	Description
Result Number	Output Result Number: **0** **Success.** **-1** **Warning.** -3 Fatal/Unexpected Error.

SQL SERVER

SQL Server Parameters: WsWrkTask

Callable Routine Type: PROCEDURE.

Parameter Name	Datatype	Mode
@p_job_key	INTEGER	IN
@p_task_key	INTEGER	IN
@p_sequence	INTEGER	IN
@p_inserted	INTEGER	IN
@p_updated	INTEGER	IN
@p_replaced	INTEGER	IN
@p_deleted	INTEGER	IN
@p_discarded	INTEGER	IN
@p_rejected	INTEGER	IN
@p_errored	INTEGER	IN

SQL Server Examples: WsWrkTask

```
-- The p_ VARIABLES are normally PARAMETERS in a RED-generated Procedure.
DECLARE @p_sequence             integer
DECLARE @p_job_name             varchar(256)
DECLARE @p_task_name            varchar(256)
DECLARE @p_job_id               integer
DECLARE @p_task_id              integer
DECLARE @p_return_msg           varchar(256)
DECLARE @p_status               integer
DECLARE @v_result_num           integer
EXEC WsWrkTask
  @p_job_id, @p_task_id, @p_sequence
, 27, 30, 0, 0, 0, 0, 0
```

TERADATA

Teradata Parameters: WsWrkTask

Callable Routine Type: PROCEDURE.

Parameter Name	Datatype	Mode
p_job_key	INTEGER	IN
p_task_key	INTEGER	IN
p_sequence	INTEGER	IN
p_inserted	INTEGER	IN
p_updated	INTEGER	IN
p_replaced	INTEGER	IN
p_deleted	INTEGER	IN
p_discarded	INTEGER	IN
p_rejected	INTEGER	IN
p_errored	INTEGER	IN

Teradata Examples: WsWrkTask

```
-- The p_ VARIABLES are normally PARAMETERS in a RED-generated Procedure.
DECLARE p_sequence              integer;
DECLARE p_job_name              varchar(256);
DECLARE p_task_name             varchar(256);
DECLARE p_job_id                integer;
DECLARE p_task_id               integer;
DECLARE p_return_msg            varchar(256);
DECLARE p_status                integer;
DECLARE v_insert_count          integer;
DECLARE v_update_count          integer;
CALL [METABASE].WsWrkTask
( p_job_id, p_task_id, p_sequence
, v_insert_count, v_update_count, 0, 0, 0, 0, 0
);
```

ORACLE

Oracle Parameters: WsWrkTask

Callable Routine Type: FUNCTION.

Parameter Name	Datatype	Mode
v_job_key	NUMBER	IN
v_task_key	NUMBER	IN
v_sequence	NUMBER	IN
v_inserted	NUMBER	IN
v_updated	NUMBER	IN
v_replaced	NUMBER	IN
v_deleted	NUMBER	IN
v_discarded	NUMBER	IN
v_rejected	NUMBER	IN
v_errored	NUMBER	IN
FUNCTION Return Value	NUMBER	OUT-Function

Oracle Examples: WsWrkTask

```
-- The p_ VARIABLES are normally PARAMETERS in a RED-generated Procedure.
p_sequence                      number;
p_job_name                      varchar2(256);
p_task_name                     varchar2(256);
p_job_id                        number;
p_task_id                       number;
p_return_msg                    varchar2(256);
p_status                        number;
v_result_num                    number;
v_insert_count                  number;
v_update_count                  number;
v_result_num := WsWrkTask
( p_job_id, p_task_id, p_sequence
, v_insert_count, v_update_count, 0, 0, 0, 0, 0
);
```

DB2

DB2 Parameters: WsWrkTask

Callable Routine Type: PROCEDURE.

Parameter Name	Datatype	Mode
p_job_key	INTEGER	IN
p_task_key	INTEGER	IN
p_sequence	INTEGER	IN
p_inserted	INTEGER	IN
p_updated	INTEGER	IN
p_replaced	INTEGER	IN
p_deleted	INTEGER	IN
p_discarded	INTEGER	IN
p_rejected	INTEGER	IN
p_errored	INTEGER	IN

DB2 Examples: WsWrkTask

```
-- The p_ VARIABLES are normally PARAMETERS in a RED-generated Procedure.
DECLARE p_sequence              integer;
DECLARE p_job_name              varchar(256);
DECLARE p_task_name             varchar(256);
DECLARE p_job_id                integer;
DECLARE p_task_id               integer;
DECLARE p_return_msg            varchar(256);
DECLARE p_status                integer;
DECLARE v_insert_count          integer;
DECLARE v_update_count          integer;
CALL [METABASE].WsWrkTask
( p_job_id, p_task_id, p_sequence
, v_insert_count, v_update_count, 0, 0, 0, 0, 0
);
```

WhereScape®

CHAPTER 32
WS_ADMIN_V VIEWS

Admin views provide a means of interacting with the WhereScape RED metadata from within your chosen reporting tools.

The following admin views are available:

Name	Description
ws_admin_v_audit	Created from the ws_wrk_audit_log table
ws_admin_v_dim_col	Created from the ws_dim_tab and ws_dim_col tables
ws_admin_v_error	Created from the ws_wrk_error_log table
ws_admin_v_fact_col	Created from the ws_fact_tab and ws_fact_col tables
ws_admin_v_fact_join	Created from the ws_fact_tab and ws_fact_col tables
ws_admin_v_sched	Created from the ws_wrk_job_log table
ws_admin_v_task	Created from the ws_wrk_task_run and ws_wrk_task_log tables

IN THIS CHAPTER

WS_ADMIN_V_AUDIT

This Audit view is created using columns from the ws_wrk_audit_log table.

Columns

The following columns are created:

Column	Description
wa_time_stamp	the date or time at which this view was created
wa_sequence	See ***Callable Routines Common Input*** (on page 959)
wa_job	See ***Callable Routines Common Input*** (on page 959)
wa_task	See ***Callable Routines Common Input*** (on page 959)
wa_status	See ***Callable Routines Common Input*** (on page 959)
wa_message	the message associated with this audit log
wa_db_msg_desc	the database message associated with this audit log

SQL Script

SQLQuery3.sql - l...TTLE-B\User (53))

```
USE [Warehouse]
GO

/****** Object:  View [dbo].[ws_admin_v_audit]    Script Date: 03/23/2
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

create view [dbo].[ws_admin_v_audit] (
wa_time_stamp, wa_sequence,wa_job, wa_task, wa_status,
wa_message,wa_db_msg_desc )
as
select  wa_time_stamp, wa_sequence,wa_job, wa_task, wa_status,
wa_message,wa_db_msg_desc
from ws_wrk_audit_log
--order by isnull(wa_row_number,0),wa_time_stamp, wa_job, wa_task
GO
```

WS_ADMIN_V_DIM_COL

This Dimension Column view is created from the ws_dim_tab and ws_dim_col tables.

Columns

The following columns are created:

Column	Description
dimension_table	name of the dimension table
dim_display_value	display value of the dimension
dim_description	description of the dimension
dim_type	a character value indicating the dimension type T - time dimension V - dimension view D - dimension C - slowly changing dimension P - previous value dimension else - unknown
column_name	column name
business_name	business name chosen for the column
description	description of the dimension column
data_type	data type of the dimension column
source_table	source table of the dimension table
source_column	source column of the dimension column
nulls_allowed	flag indicating whether or not nulls are allowed
numeric	flag indicating whether or not the column is numeric
additive	flag indicating whether or not the column is additive
attribute	flag indicating whether or not the column is an attribute
format	the dimension column format
column_order	the dimension column order

SQL Script

SQLQuery4.sql - l...TTLE-B\User (53))

```
create view [dbo].[ws_admin_v_dim_col] (
dimension_table,
dim_display_value,
dim_description,
dim_type,
column_name,
business_name,
description,
data_Type,
source_table,
source_column,
nulls_allowed,
numeric,
additive,
attribute,Format,column_order)
AS
select
dt_table_name,
dt_display_name,
dt_description,
CASE dt_type_ind
   WHEN 'T' THEN 'Time dimension'
   WHEN 'V' THEN 'Dimension View'
   WHEN 'D' THEN 'Dimension'
   WHEN 'C' THEN 'Slowly changing dimension'
   WHEN 'P' THEN 'Previous value dimension'
   ELSE 'Unknown'
END,
dc_col_name ,
dc_display_name,
dc_src_strategy,
dc_data_type,
dc_src_table,
dc_src_column,
dc_nulls_flag,
dc_numeric_flag,
dc_additive_flag,
dc_attribute_flag,
dc_format,
dc_order
from ws_dim_tab LEFT OUTER JOIN ws_dim_col
on dt_obj_key = dc_obj_key
where dt_type_ind in ('D','C','T','V','P')
and isnull(dc_eul_flag,'N') = 'Y'
GO
```

WS_ADMIN_V_ERROR

This Error view is created using columns from the ws_wrk_error_log table.

Columns

The following columns are created:

Column	Description
wd_time_stamp	the date or time at which this view was created
wd_sequence	See ***Callable Routines Common Input*** (on page 959)
wd_job	See ***Callable Routines Common Input*** (on page 959)
wd_task	See ***Callable Routines Common Input*** (on page 959)
wd_status	See ***Callable Routines Common Input*** (on page 959)
wd_message	the message associated with this audit log
wd_db_msg_desc	the database message associated with this audit log

SQL Script

SQLQuery5.sql - l...TTLE-B\User (53))

```
USE [Warehouse]
GO

/****** Object:  View [dbo].[ws_admin_v_error]    Script Date: 03/23/2012 11:56:46 ******/
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

create view [dbo].[ws_admin_v_error] ( wd_time_stamp, wd_sequence,wd_job, wd_task, wd_status,
wd_message,wd_db_msg_desc )
as
select  wd_time_stamp, wd_sequence,wd_job, wd_task, wd_status,
wd_message,wd_db_msg_desc
from ws_wrk_error_log
--order by isnull(wd_row_number,0),wd_time_stamp, wd_job, wd_task
GO
```

WS_ADMIN_V_FACT_COL

This Fact Column view is created from the ws_fact_tab and ws_fact_col tables.

Columns

The following columns are created:

Column	Description
fact_table	name of the fact table
fact_display_value	display value of the fact table
fact_description	description of the fact table
fact_type	a character value indicating the fact type D - detail R - rollup / combined S - kpi P - detail partitioned Q - rollup partitioned else unknown
column_name	column name
business_name	business name chosen for the column
description	description of the fact column
data_type	data type of the fact column
source_table	source table of the fact table
source_column	source column of the dimension column
nulls_allowed	flag indicating whether or not nulls are allowed
numeric	flag indicating whether or not the column is numeric
additive	flag indicating whether or not the column is additive
attribute	flag indicating whether or not the column is an attribute
format	the fact column format
column_order	the fact column order

SQL Script

SQLQuery6.sql - I...TTLE-B\User (53))

```
create view [dbo].[ws_admin_v_fact_col] (
"fact_table",
"fact_display_value",
"fact_description",
"fact_type",
"column_name",
"business_name",
"description",
"data_Type",
"source_table",
"source_column",
"nulls_allowed",
"numeric",
"additive",
"attribute",
"format",
"column_order")
as
select
ft_table_name,
ft_display_name,
ft_description,
CASE ft_type_ind
   WHEN 'D' THEN 'Detail'
   WHEN 'R' THEN 'Rollup/Combined'
   WHEN 'S' THEN 'KPI'
   WHEN 'P' THEN 'Detail partitioned'
   WHEN 'Q' THEN 'Rollup partitioned'
   ELSE 'Unknown'
END,
fc_col_name,
fc_display_name,
fc_src_strategy,
fc_data_type,
fc_src_table,
fc_src_column,
fc_nulls_flag,
fc_numeric_flag,
fc_additive_flag,
fc_attribute_flag,
fc_format,
fc_order
from ws_fact_tab LEFT OUTER JOIN ws_fact_col
on ft_obj_key = fc_obj_key
where ft_type_ind in ('D','S','R','P','Q')
and isnull(fc_eul_flag,'N') = 'Y'
GO
```

WS_ADMIN_V_FACT_JOIN

This Fact Column view is created from the ws_fact_tab and ws_fact_col tables.

Columns

The following columns are created:

Column	Description
fact_table	fact table name
fact_column	fact table column name
dimension_table	dimension table name
dimension_column	dimension table column name

SQL Script

SQLQuery7.sql - l...TTLE-B\User (53))

```
USE [Warehouse]
GO

/****** Object:  View [dbo].[ws_admin_v_fact_join]
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

create view [dbo].[ws_admin_v_fact_join] (
fact_table,
fact_column,
dimension_table,
dimension_column)
as
select
ft_table_name ,
fc_col_name,
fc_src_table,
fc_src_column
from ws_fact_tab LEFT OUTER JOIN ws_fact_col
on ft_obj_key = fc_obj_key
where ft_type_ind in ('D','S','R','Q','P')
and isnull(fc_join_flag,'N') = 'Y'
GO
```

WS_ADMIN_V_SCHED

This Scheduled Job view is created from the ws_wrk_job_log table.

Columns

The following columns are created:

Column	Description
type	"Waiting"
job_name	the name of the job
status	a character value indicating the job status H - on hold R - running P - pending W - waiting C - completed B - blocked F - failed G - failed - aborted E - error completion else unknown
sequence	sequence number of the job
started_scheduled	
completed	date completed
hours_elapsed	hours elapsed since job started
minutes_elapsed	minutes elapsed since job started
okay	okay count
info	info count
warn	warning count
detail	detail count
error	error count

SQL Script

SQLQuery8.sql - l...TTLE-B\User (53))

```
USE [Warehouse]
GO

/****** Object:  View [dbo].[ws_admin_v_sched]    Script
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

create view [dbo].[ws_admin_v_sched] (
    "type",
    "job_name",
    "status",
    "sequence",
    "started_scheduled",
    "completed",
    "hours_elapsed" ,
    "minutes_elapsed",
    "okay",
    "info",
    "warn",
    "detail",
    "error")
as
select
    'Waiting',
    wjc_name,
        CASE wjc_status
           WHEN 'H' THEN 'On Hold'
           WHEN 'R' THEN 'Running'
           WHEN 'P' THEN 'Pending'
           WHEN 'W' THEN 'Waiting'
           WHEN 'C' THEN 'Completed'
           WHEN 'B' THEN 'Blocked'
           WHEN 'F' THEN 'Failed'
           WHEN 'G' THEN 'Failed - Aborted'
           WHEN 'E' THEN 'Error Completion'
           ELSE 'Unknown'
        END,
```

```
    wjc_sequence,
    wjc_start_after,
    GETDATE(),
    0 ,
    0,
    0,
    0,
    0,
    0,
    0
      from ws_wrk_job_ctrl
union
     select 'Runnning',wjr_name,
        CASE wjr_status
           WHEN 'H' THEN 'On Hold'
           WHEN 'R' THEN 'Running'
           WHEN 'P' THEN 'Pending'
           WHEN 'W' THEN 'Waiting'
           WHEN 'C' THEN 'Completed'
           WHEN 'B' THEN 'Blocked'
           WHEN 'F' THEN 'Failed'
           WHEN 'G' THEN 'Failed - Aborted'
           WHEN 'E' THEN 'Error Completion'
           ELSE 'Unknown'
        END,
    wjr_sequence, wjr_started, wjr_completed,
    CONVERT(integer,DATEDIFF(mi,wjr_started,GETDATE())/60),
    DATEDIFF(mi,wjr_started,GETDATE()) -
    CONVERT(integer,DATEDIFF(mi,wjr_started,GETDATE())/60)*60,
     wjr_okay_count,
     wjr_info_count, wjr_warning_count, wjr_detail_count,
     wjr_error_count
      from ws_wrk_job_run
union
```

```
      select 'Finished',wjl_name,
         CASE wjl_status
            WHEN 'H' THEN 'On Hold'
            WHEN 'R' THEN 'Running'
            WHEN 'P' THEN 'Pending'
            WHEN 'W' THEN 'Waiting'
            WHEN 'C' THEN 'Completed'
            WHEN 'B' THEN 'Blocked'
            WHEN 'F' THEN 'Failed'
            WHEN 'G' THEN 'Failed - Aborted'
            WHEN 'E' THEN 'Error Completion'
            ELSE 'Unknown'
         END,
      wjl_sequence, wjl_started, wjl_completed,
      wjl_elapsed_hh, wjl_elapsed_mi, wjl_okay_count,
      wjl_info_count, wjl_warning_count, wjl_detail_count,
      wjl_error_count
      from ws_wrk_job_log
GO
```

WS_ADMIN_V_TASK

This Task view is created from the ws_wrk_task_run and ws_wrk_task_log tables.

Columns

The following columns are created:

Column	Description
result	
task_name	the name of the task
status	a character value indicating the task status H - on hold R - running P - pending W - waiting C - completed B - blocked F - failed G - failed - aborted E - error completion else unknown
sequence	sequence number of the task
started	
completed	
hours_elapsed	hours elapsed since the task started
minutes_elapsed	minutes elapsed since the task started
info	info count
warn	warning count
detail	detail count
inserted	record inserted
updated	record updated
replaced	record replaced
deleted	record deleted
discarded	record discarded
rejected	record rejected
errored	record errored

SQL Script

SQLQuery10.sql -...TTLE-B\User (53))

```
USE [Warehouse]
GO

/****** Object:  View [dbo].[ws_admin_v_task]    Script Date
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

create view [dbo].[ws_admin_v_task] (
    "Result",
    "task_name",
    "status",
    "sequence",
    "started",
    "completed",
    "hours_elapsed" ,
    "minutes_elapsed",
    "info",
    "warn",
    "detail",
    "inserted",
    "updated",
    "replaced",
    "deleted",
    "discarded",
    "rejected",
    "errored")
as
    Select
    'Runnning',wtr_name,
        CASE wtr_run_status
           WHEN 'H' THEN 'On Hold'
           WHEN 'R' THEN 'Running'
           WHEN 'P' THEN 'Pending'
           WHEN 'W' THEN 'Waiting'
           WHEN 'C' THEN 'Completed'
           WHEN 'B' THEN 'Blocked'
           WHEN 'F' THEN 'Failed'
           WHEN 'G' THEN 'Failed - Aborted'
           WHEN 'E' THEN 'Error Completion'
           ELSE 'Unknown'
        END,
```

```
    wtr_sequence, wtr_started, wtr_completed,
    CONVERT(integer,DATEDIFF(mi,wtr_started,GETDATE())/60),
    DATEDIFF(mi,wtr_started,GETDATE()) -
    CONVERT(integer,DATEDIFF(mi,wtr_started,GETDATE())/60)*60,
      wtr_info_count, wtr_warning_count, wtr_detail_count,
     wtr_rec_inserted, wtr_rec_updated,wtr_rec_replaced,wtr_rec_deleted,wtr_rec_discarded,
    wtr_rec_rejected,wtr_rec_errored
      from ws_wrk_task_run
union
     select wtl_return_msg,wtl_name,
        CASE wtl_run_status
           WHEN 'H' THEN 'On Hold'
           WHEN 'R' THEN 'Running'
           WHEN 'P' THEN 'Pending'
           WHEN 'W' THEN 'Waiting'
           WHEN 'C' THEN 'Completed'
           WHEN 'B' THEN 'Blocked'
           WHEN 'F' THEN 'Failed'
           WHEN 'G' THEN 'Failed - Aborted'
           WHEN 'E' THEN 'Error Completion'
           ELSE 'Unknown'
        END,
     wtl_sequence, wtl_started, wtl_completed,
     wtl_elapsed_hh, wtl_elapsed_mi,
     wtl_info_count, wtl_warning_count, wtl_detail_count,
     wtl_rec_inserted, wtl_rec_updated,wtl_rec_replaced,wtl_rec_deleted,wtl_rec_discarded,
    wtl_rec_rejected,wtl_rec_errored
     from ws_wrk_task_log
GO
```

WhereScape®

CHAPTER 33
RETROFITTING

WhereScape RED includes an advanced retrofit capability that can be used to:

1. Migrate an existing data warehouse from one relational database to another (known as fork-lifting).
2. Load a data model from a modeling tool.

Retrofitting is achieved using the **Retro** object type in WhereScape RED and the **Retrofit tables wizard**.

For information on migrating an existing data warehouse, see ***Migrating the Data Warehouse Database Platform*** (see "***OLAP Retrofitting an OLAP Object***" on page 617, on page 1117).

For information on importing a data model, see ***Importing a Data Model*** (on page 1126).

IN THIS CHAPTER

MIGRATING THE DATA WAREHOUSE DATABASE PLATFORM

WhereScape RED has an advanced retrofitting wizard for migrating an existing data warehouse from one relational database to another.

The process to migrate an existing data warehouse is:

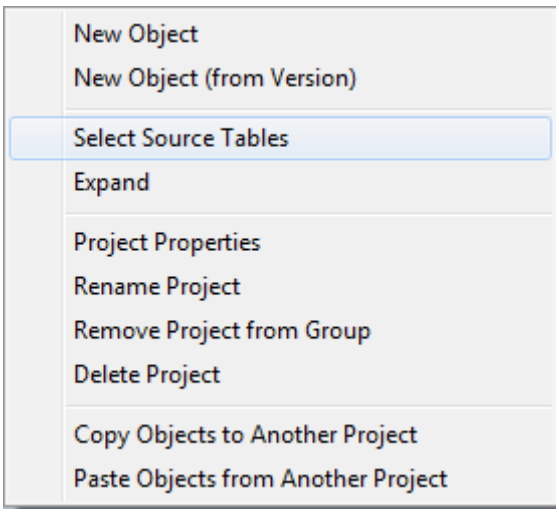
1. Create a connection object to the existing warehouse database.
2. Create Retro objects based on the source tables in the existing warehouse database.
3. Set the Retro objects as Retro Copy type objects.
4. Run a Scheduler task to build the Retro Copy objects from the source tables.
5. Set the Retro objects back to Retro (Retro Definition) type objects.
6. Convert the Retro objects to the Target Object types.

The steps to use this wizard are:

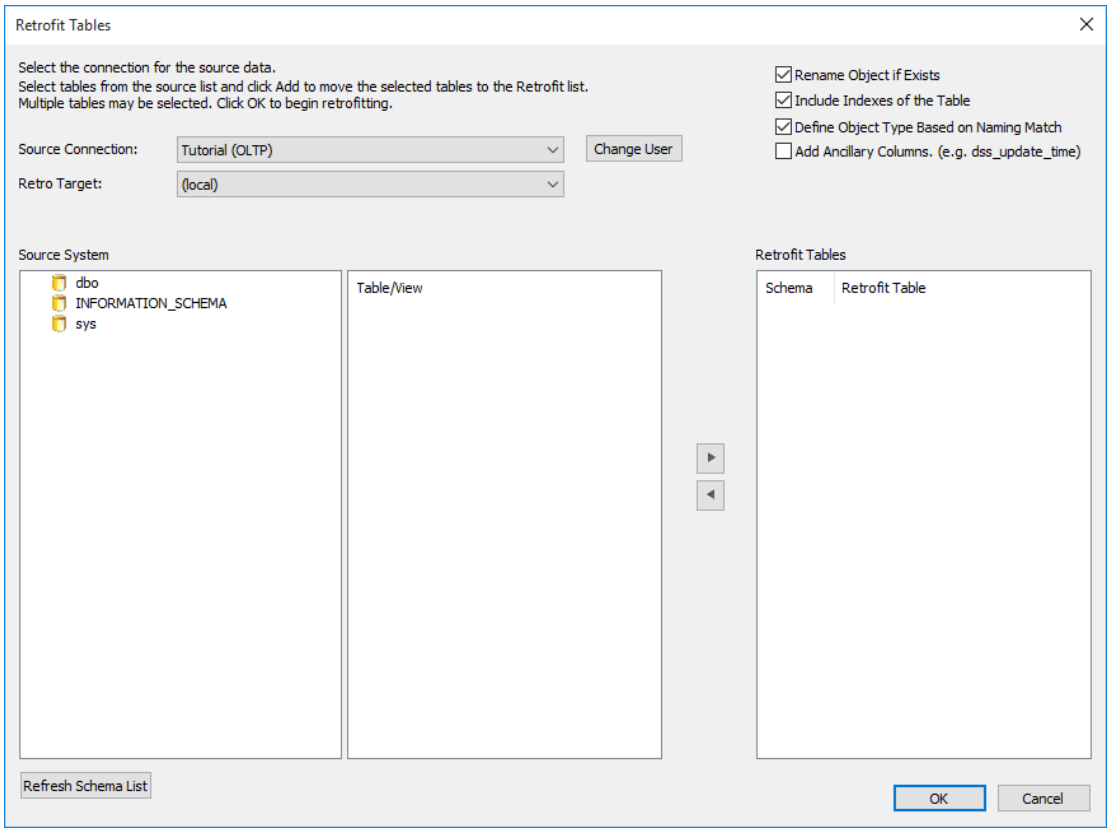
1. Create a connection object for the old data warehouse database, populating the following fields:
   - Connection Name
   - Connection Type => **ODBC**
   - ODBC Source
   - Work Directory
   - Extract user name
   - Extract password

   Note: The extract user must be able to select from all tables to be migrated.

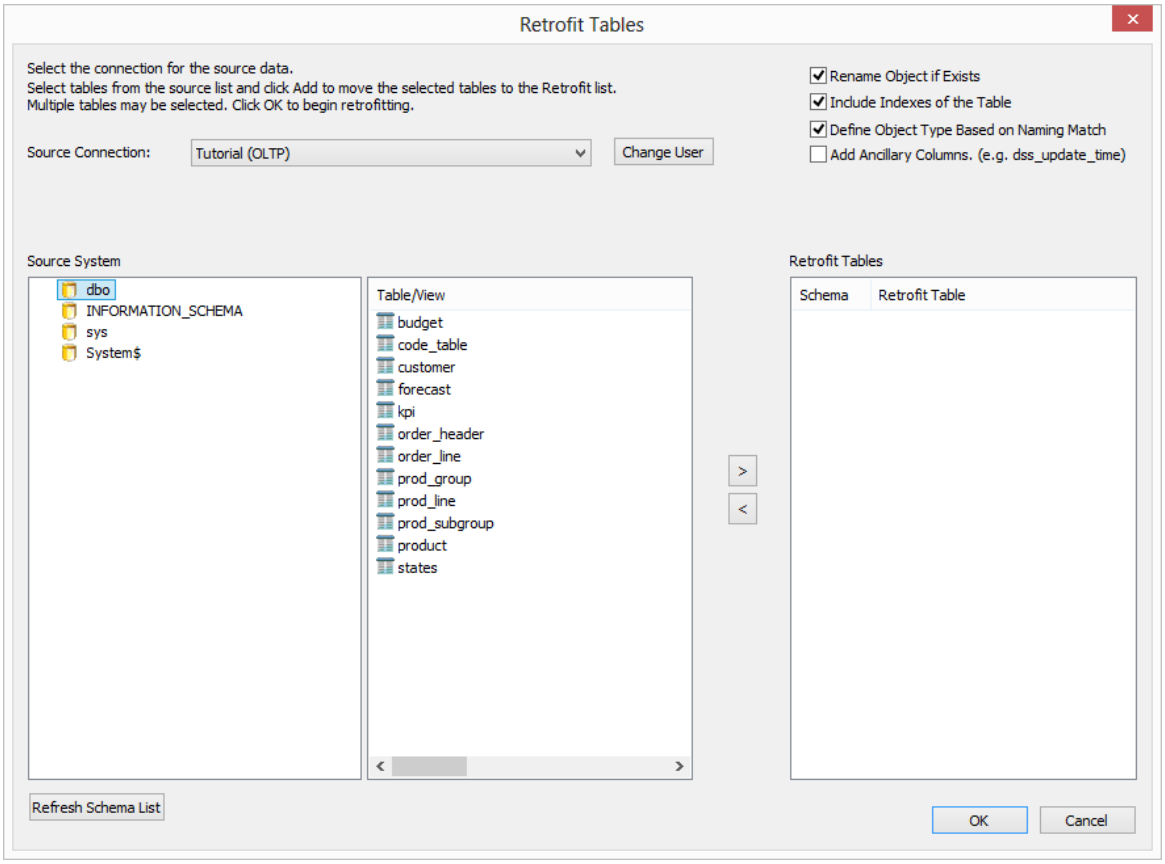
2. Ensure all naming standards in **Tools/Options** are set to match the objects being retrofitted. This saves work later.
3. Ensure **Enable Retro** is selected in the **Tools/Options/Object Types** menu.
4. Right-click on the **Retro object group** in the object tree in the left pane and select **Select Source Tables**.



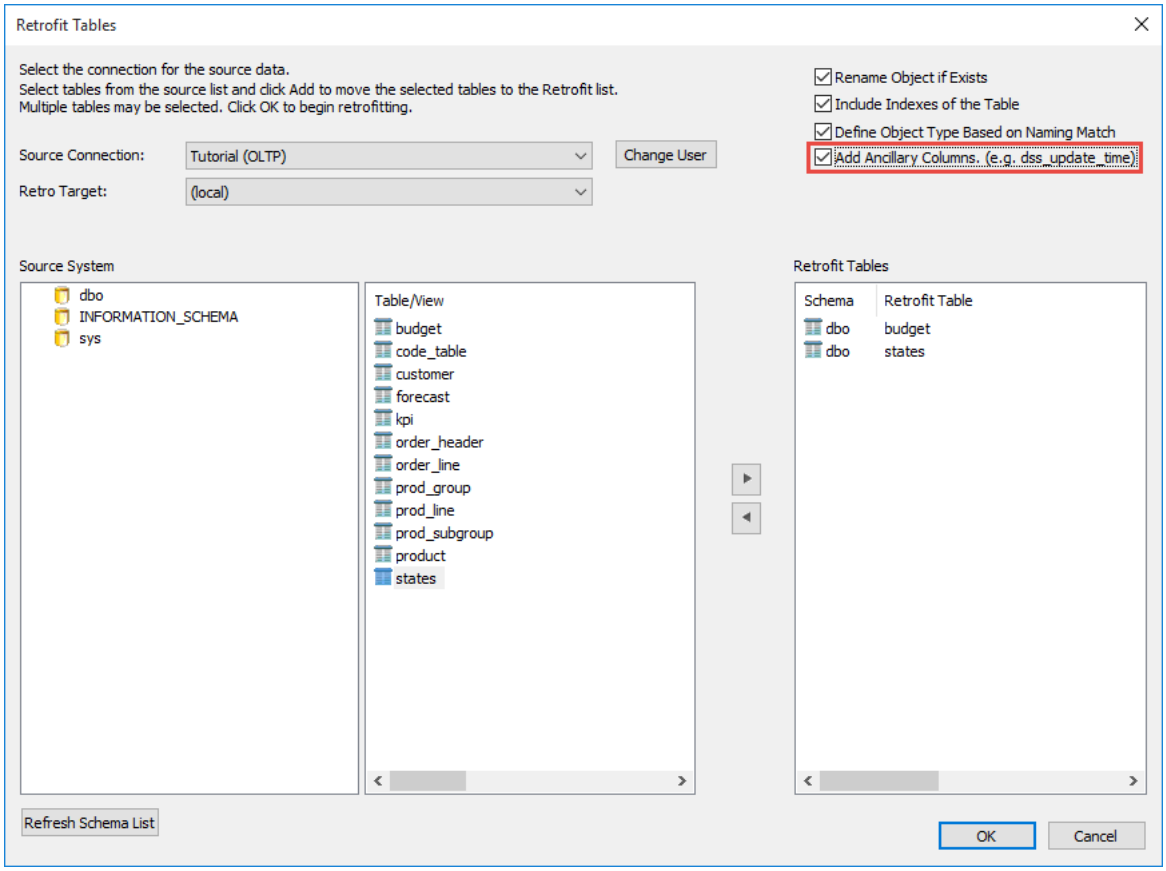
5 The Retrofit Tables dialog appears. In the **Source Connection** drop-down list choose the connection set up in step 1. A list of **databases** appears in the left pane.



6 Double-click on the database/user/schema in the left pane list. A list of **tables** in the database is displayed in the middle pane.

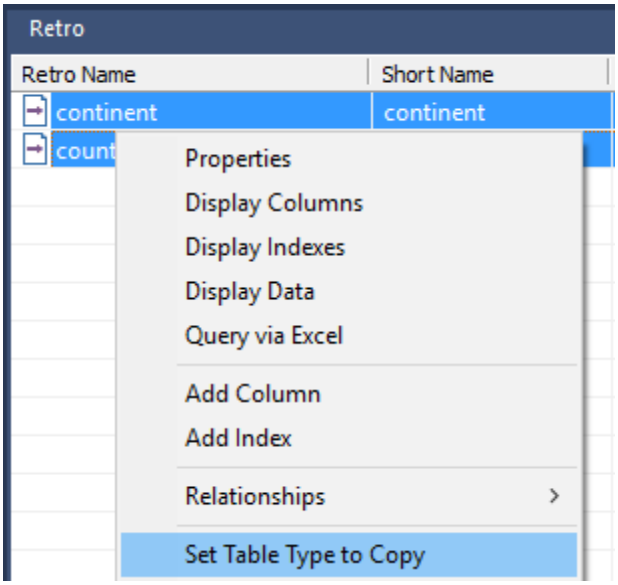


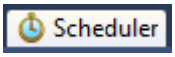
7 Select all the required tables from the middle pane list and click **>** to move them to the right pane. Then click the **Add Ancillary Columns (e.g. dss_update_time)** check-box and click **OK**.



8 WhereScape RED acquires the metadata for the tables being migrated and creates a new WhereScape RED Retro object for each.

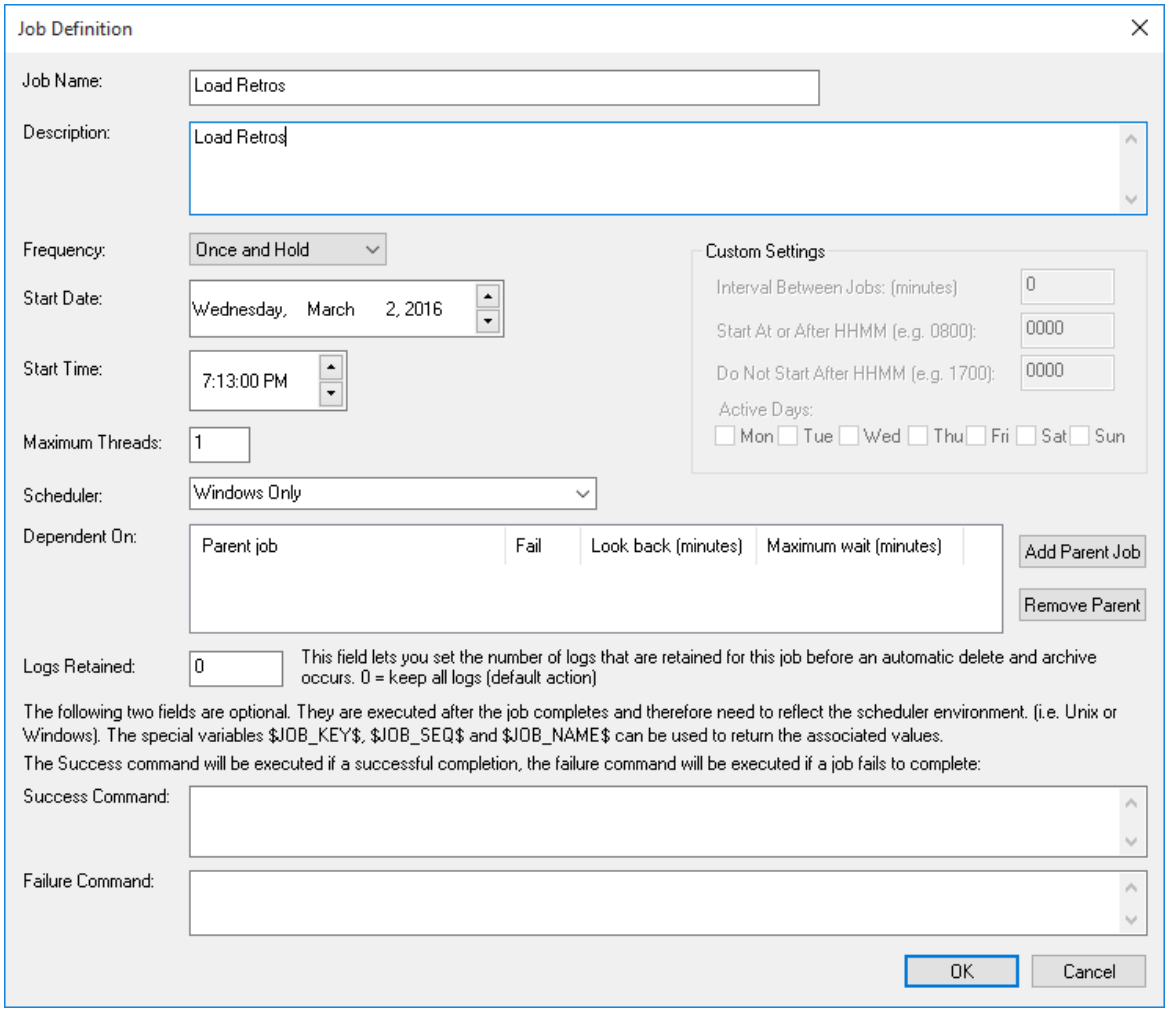
9 Double-click on the **Retro object group** in the left pane. Select all Retros in the middle pane. Right-click and select **Set Table Type to Copy**. This allows the data in the legacy data warehouse to be copied across to the new data warehouse.



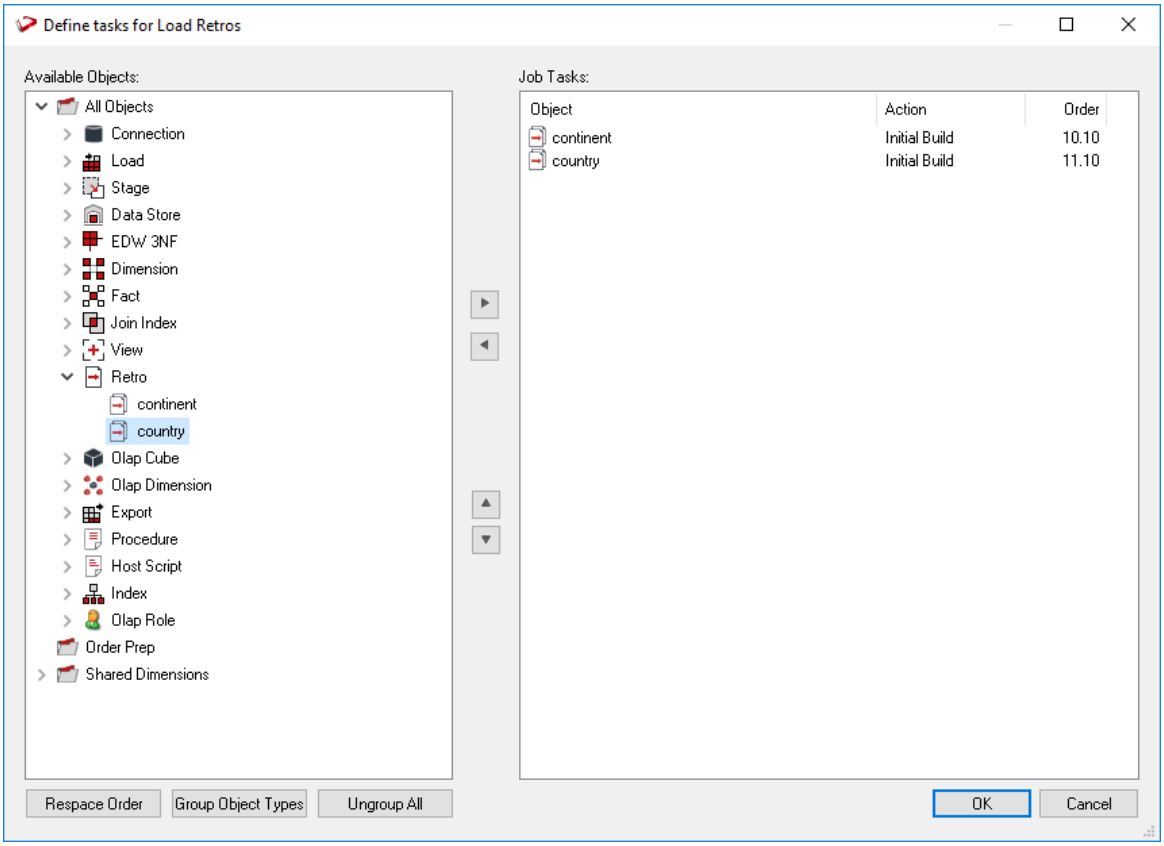
10 Click on the **Scheduler** button on the toolbar. 

11 Create a new job to run straight away and click **OK**.

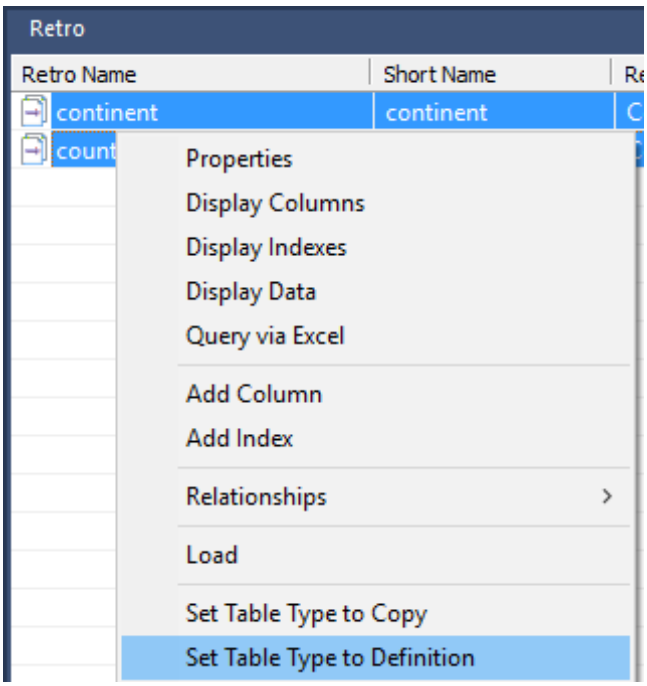
Note: A scheduler must be running on the data warehouse connection for this job to complete, please refer to section **17. Scheduler Installation and Configuration** of the RED Setup Administrator Guide.


Job Definition
Job Name:
Load Retros
Description:
Load Retros
Frequency:
Once and Hold
Start Date:
Wednesday, March 2, 2016
Start Time:
7:13:00 PM
Maximum Threads:
1
Scheduler:
Windows Only
Custom Settings
Interval Between Jobs: (minutes)
0
Start At or After HHMM (e.g. 0800):
0000
Do Not Start After HHMM (e.g. 1700):
0000
Active Days:
Mon Tue Wed Thu Fri Sat Sun
Dependent On:
Parent job
Fail
Look back (minutes)
Maximum wait (minutes)
Add Parent Job
Remove Parent
Logs Retained:
0
This field lets you set the number of logs that are retained for this job before an automatic delete and archive occurs. 0 = keep all logs (default action)
The following two fields are optional. They are executed after the job completes and therefore need to reflect the scheduler environment. (i.e. Unix or Windows). The special variables JOB_KEY, JOB_SEQ and JOB_NAME can be used to return the associated values.
The Success command will be executed if a successful completion, the failure command will be executed if a job fails to complete:
Success Command:
Failure Command:
OK
Cancel

12 Add all Retro objects created in steps (3) to (9) and click on **Group Object Types.**

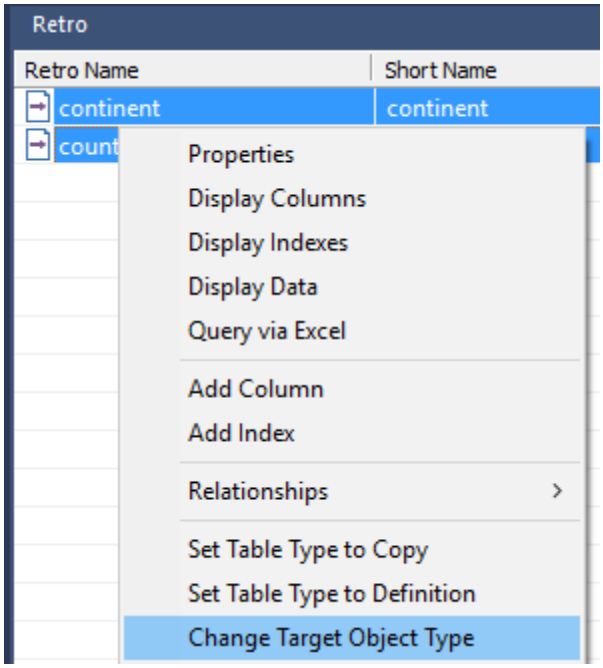


13 Once the job has completed, return to the WhereScape RED builder. Double-click on the **Retro object group**. Select all objects in the middle pane and from the right-click menu select **Set Table Type to Definition**. This indicates the data has been copied into the Retro objects and the Retros can now be converted to the target objects.

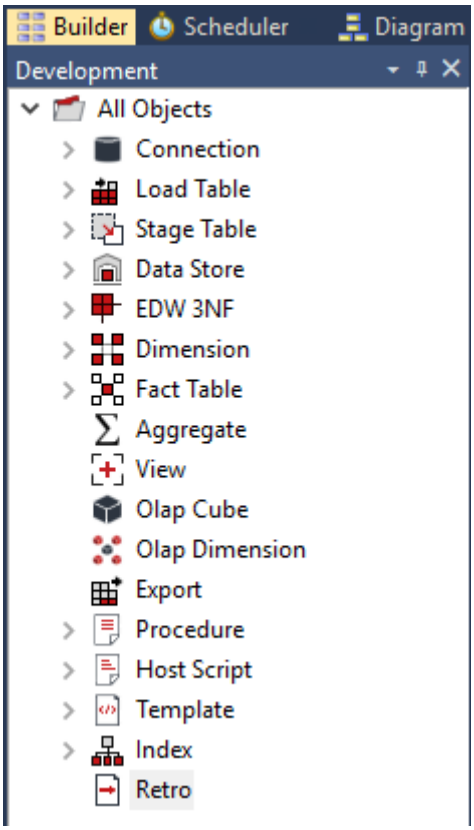


14 In the middle pane, select all objects. Right-click and select **Convert to Target Object**. WhereScape RED now converts the **Retro** objects to the appropriate object types.

Note: If the appropriate Target Object Type has not been set for one or more Retro objects; in the right click menu select **Change Target Object Type** and select the correct Object Type.



15 There are no longer any Retro objects. They have been converted to Load, Stage, Dimension or Fact objects.



16 Change the source table and source column values on all of the retrofitted objects using either the Re-target source table dialog, or by editing column properties. See ***Re-targeting source tables*** (on page 1131) for more information.

17 Convert the old data warehouses code to WhereScape RED procedure in the new data warehouse database. See ***Integrate Procedures*** (see "***Integrating, Procedures***" on page 1148) for more information.

18 If necessary, create new connections to be used with any migrated load tables. Attach a connection to each load table. See ***Loading Data*** (on page 232) for more information.

IMPORTING A DATA MODEL

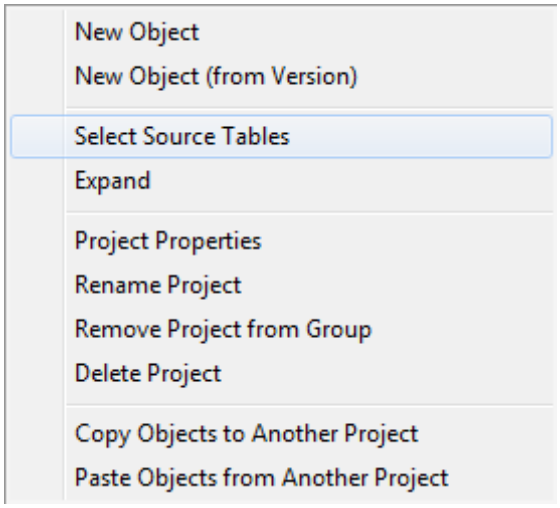
WhereScape RED provides functionality for importing data models from modeling tools.

The process to import a model is:

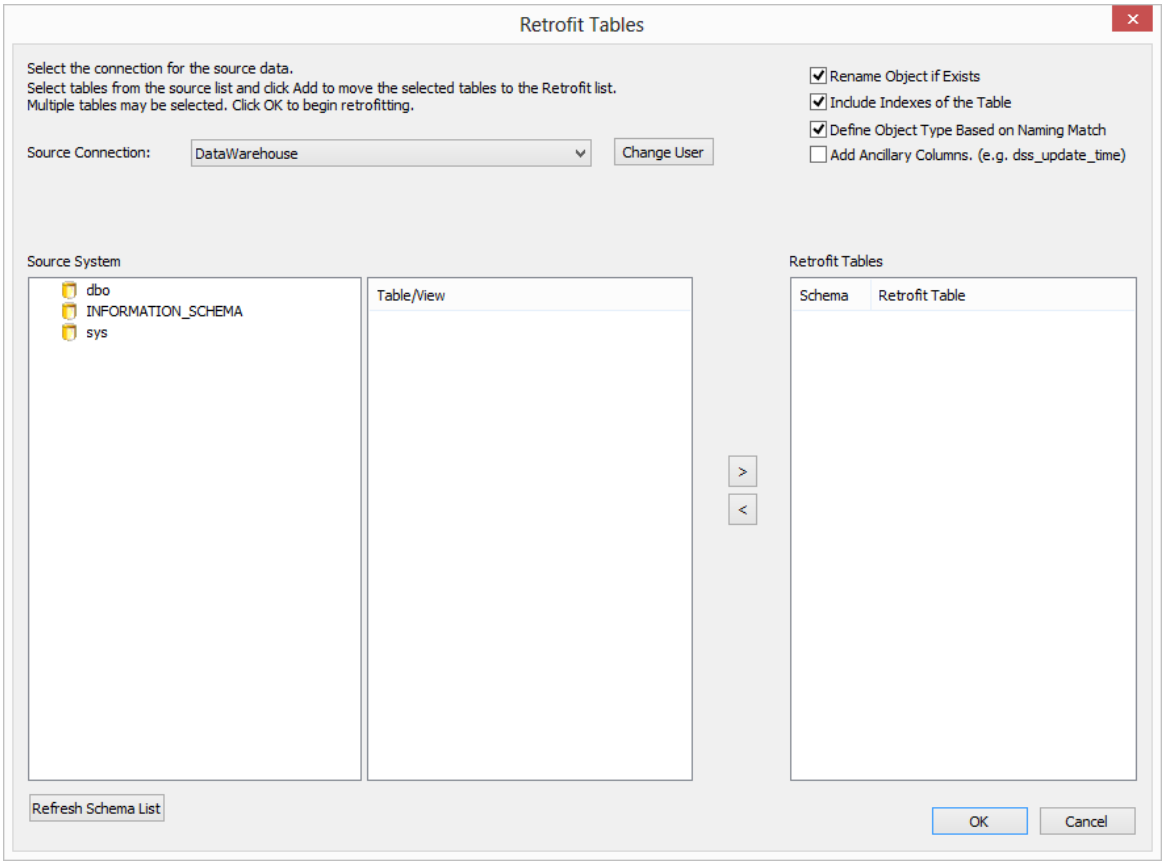
1. Create the physical data model in the modeling tool.
2. Generate DDL for the physical model in the modeling tool.
3. Run the DDL in the data warehouse database to create empty versions of the model tables.
4. Retrofit the tables in the dummy database into the WhereScape RED metadata as Retro objects.
5. Convert the Retro objects to Dimensions and Facts.

The following instructions outline steps 4 and 5 above:

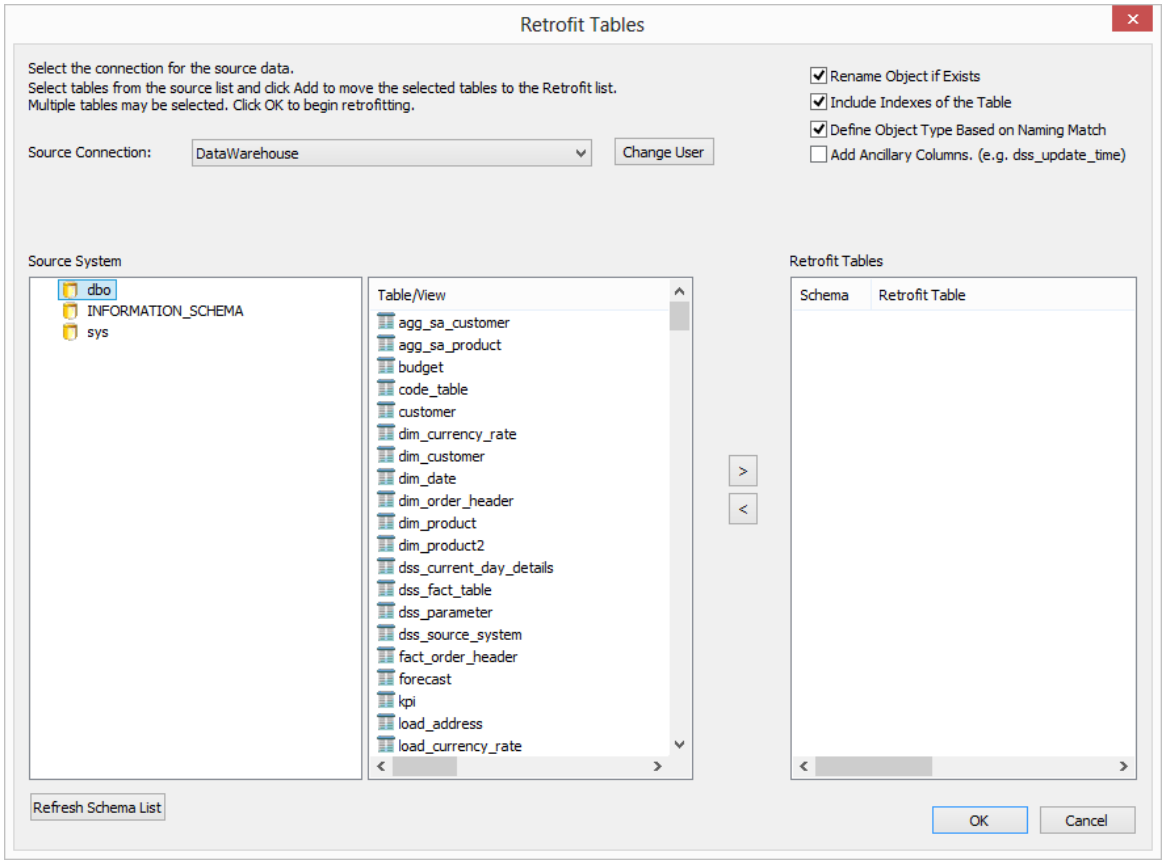
1. Right-click on the **Retro object group** in the object tree in the left pane and select **Select Source Tables**.



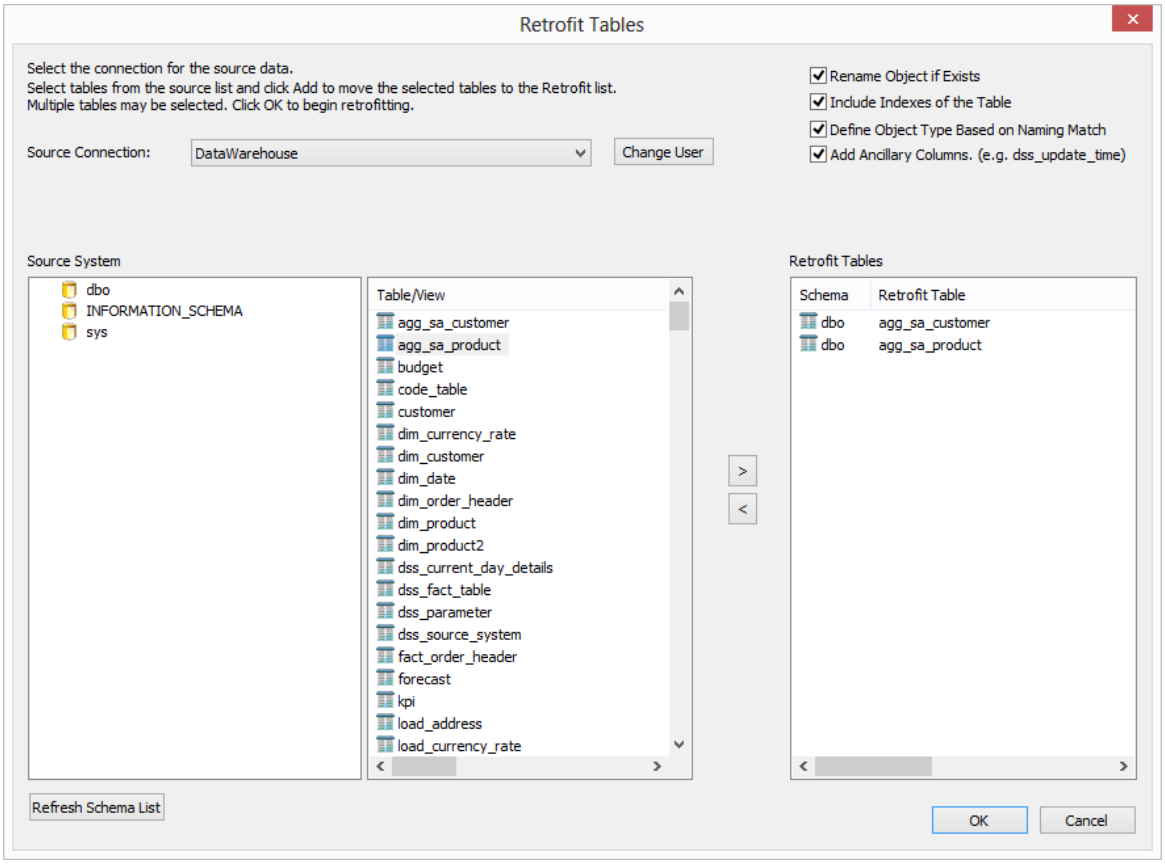
2 The **Retrofit tables** dialog is displayed. In the **Source Connection** drop-down list choose the **DataWarehouse** connection. A list of **databases** appears in the left pane (your list will be different).



3 Double-click on the data warehouse database/schema in the left pane list. A list of **tables** in the database is displayed in the middle pane.

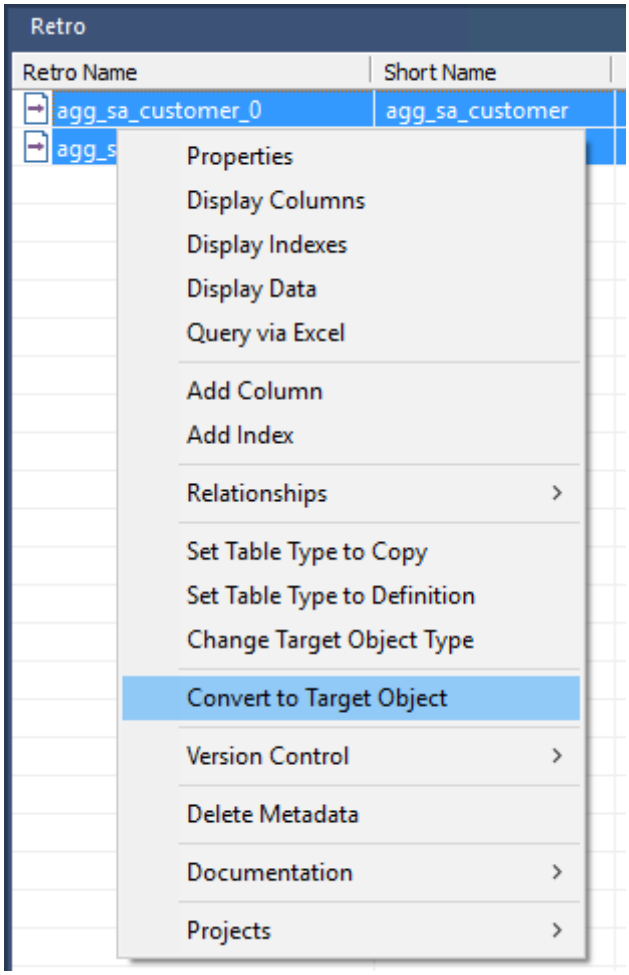


4 Click on the required tables in the middle pane list and click **>** to move them to the right pane. Then click the **Add Ancillary Columns (e.g. dss_update_time)** check-box. Click **OK**.

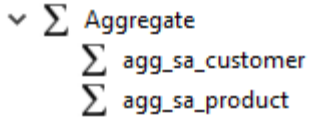


5 Double-click on the **Retro object group** in the object tree in the left pane. Select all objects in the middle pane, right-click and select **Convert to Target Object**.

Note: If the appropriate Target Object Type has not been set for one or more Retro objects; in the right click menu select **Change Target Object Type** and select the correct Object Type.



6 The new aggregate tables have been imported.



7 At this stage, you have created the table metadata only. To create the tables in the data warehouse, double click on the object group in the object tree in the left pane. In the middle pane highlight the tables, then right-click and select **Create (ReCreate)**.

RE-TARGETING SOURCE TABLES

Objects that have been retrofitted into the WhereScape RED metadata have themselves as their source table:

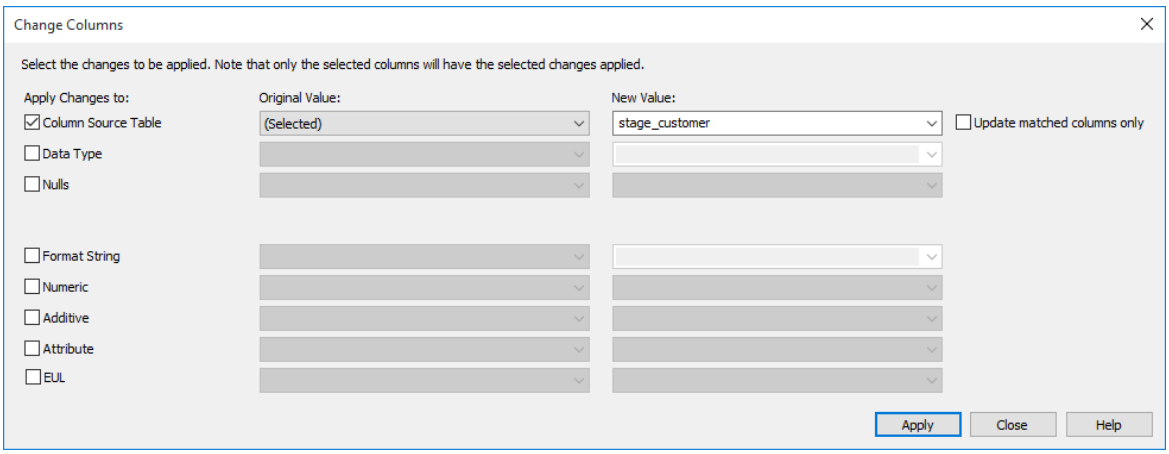
dim_product Columns

Column Name	Display Name	Data Type	Source Table	Source Column
dim_product_key	dim product key	integer ident...		dim_product_key
code	code	numeric(6)	load_product	code
description	description	varchar(64)	load_product	description
prod_line	prod line	varchar(24)	load_product	prod_line
prod_group	prod group	varchar(24)	load_product	prod_group
subgroup	subgroup	varchar(24)	load_product	subgroup
subgroup_description	subgroup desc...	varchar(64)	load_prod_subgro...	subgroup_descript...
line_description	line description	varchar(64)	load_prod_line	line_description
group_description	group descript...	varchar(64)	load_prod_group	group_description
dss_update_time	dss update time	datetime		dss_update_time

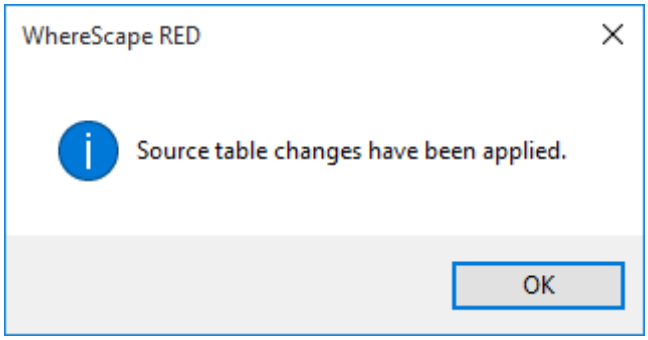
They can be re-targeted to the correct source table(s) using the WhereScape RED re-target wizard as follows.

1 Right-click on a table object in the left pane and select **Change Column(s)**.

2 Select the **Column Source Table** checkbox. Select **dim_product** from the Original Value drop-down list. Select **load_product** from the New Value drop-down list. Click **Apply**.



3 A message will be displayed to show that the source columns have been changed. Click **OK**.



4 Click **Close**.

5 Confirm the **Source Table** column in the middle pane.

dim_product Columns

Column Name	Display Name	Data Type	Source Table
dim_product_key	dim product key	integer ident...	
code	code	numeric(6)	stage_customer
description	description	varchar(64)	stage_customer
prod_line	prod line	varchar(24)	stage_customer
prod_group	prod group	varchar(24)	stage_customer
subgroup	subgroup	varchar(24)	stage_customer
subgroup_description	subgroup desc...	varchar(64)	load_prod_subgro...
line_description	line description	varchar(64)	load_prod_line
group_description	group descript...	varchar(64)	load_prod_group
dss_update_time	dss update time	datetime	

RETRO COLUMN PROPERTIES

Each Retro column has a set of associated properties. The definition of each property is defined below.

If the **Column name** or **Data type** is changed for a column then the metadata will differ from the table as recorded in the database. Use the **Validate/Validate Table Create Status** menu option to compare the metadata to the table in the database. When positioned on the table name after the validate has completed, a right-click menu option **Alter Table** will alter the database table to match the metadata definition.

TIP: If a database table's definition is changed in the metadata then the table will need to be altered in the database. Use the **Validate/Validate Table Create Status** to compare metadata definitions to physical database tables. The option also provides the ability to alter the database table through a pop-up menu option from the validated table name.

Table Name

Database-compliant name of the table that contains the column. [Read-only].

Column Name

Database-compliant name of the column. Typically column-naming standards exclude the use of spaces etc. A good practice is to only use alphanumerics, and the underscore character. Changing this field alters the table's definition.

Note: A case conversion button on the right converts the text between different cases: UPPER CASE, Capitalized Case and lower case. The mode cycles to the next case in the sequence each conversion.

Business Display Name

Name that the business uses to refer to the column, which is included in the RED-generated documentation and can be used in the end user layer of other tools. [Does NOT affect the physical database table]. As such it is a free form entry and any characters are valid.

Note: A case conversion button on the right converts the text between different cases: UPPER CASE, Capitalized Case and lower case. The mode cycles to the next case in the sequence each conversion.

Column Description

This field contains the description for the column. In the case of dimension keys this field is used to show the join between the Retro table and the dimension, once it has been defined as part of the Retro table update procedure generation.

Column Order

Numeric value that controls the relative order of columns in the database create statement. The lowest numbered column will appear first in the table. Although this affects the physical table definition no action will be taken unless the table is re-created in the database. The columns can be re-numbered based on the existing order by choosing the **Respace order number** pop-up menu option when positioned over any column in the table. This action will number the columns in increments of 10 starting at 10. In addition to a simple change of the order field, the column order can be changed by first displaying the columns in the middle pane and then using drag and drop to move the columns around. This drag and drop process will automatically renumber the columns as required.

Data Type

Database-compliant data type that must be valid for the target database. Typical data types for Oracle are integer, number, char(), varchar2() and date. For SQL Server common types are integer, numeric, varchar() and datetime. See the database documentation for a description of the data types available. Changing this field alters the table's definition.

Null Values Allowed

Determines whether the table column can hold NULL values or whether a value is always mandatory.

Default Value

Initial value that is assigned to the column when a row is inserted into the table but no value is specified for the column.

Format

Optional format mask that can be used in end user tools. [Does NOT affect the physical database table]. As such it is a free form entry and any characters are valid. Typically format masks are only used on numeric fields. Example: #,###0.00. It is not worth the effort of populating this field unless it can be utilized by the end user tools in use.

Numeric

Indicates whether the table column holds values that are numeric. This is normally only relevant for fact tables. It does not affect the physical table definition, but rather provides input to the view **ws_admin_v_dim_col** which can be used to assist in the population of an end user tool's end user layer. The use of this field is not relevant unless it can be utilized by the end user tools.

Additive

Indicates whether the table column holds values that are additive. This implies that the column can be summed when performing data grouping in a query. This is normally only relevant for fact tables. It does not affect the physical table definition, but rather provides input to the view **ws_admin_v_dim_col** which can be used to assist in the population of an end user tool's end user layer. The use of this field is not relevant unless it can be utilized by the end user tools.

Attribute

Indicates whether the table column holds values that are descriptive, and/or are used for grouping/summing. An attribute is defined as a column that is non factual. For example, we may have an order number, or a invoice number stored in the fact table. Such columns are considered attributes, rather than facts. This checkbox is therefore normally only relevant for fact tables. This checkbox does not affect the physical table definition, but rather provides input to the view **ws_admin_v_dim_col** which can be used to assist in the population of an end user tools end user layer. The use of this field is not relevant unless it can be utilized by the end user tools.

Business Key

Indicates whether the column is part of the BUSINESS key, which is defined during the update procedure build. [Normally maintained automatically]. Multiple columns can form the primary business key.

Key Type

Key type that is assigned and used when generating the table's update procedure and indexes. [Normally maintained automatically]. It can be altered here, but this should only be done if the consequences are fully understood and tested. The supported values are:

Key type	Meaning
0	The artificial key. Set when the key is added during drag and drop table generation.
1	Component of all business keys. Indicates that this column is used as part of any business key. For example: By default the dss_source_system_key is added to every dimension table. It is considered part of any lookup on that table and has the key type set to 1. Set when the column is added during drag and drop table generation.
2	Indicates that this column is a dimensional join. Used on fact tables to indicate the dimension keys. Results in bitmap indexes being built for the columns. Set during the update procedure generation for a fact table, based on information from the staging table.
3	Slowly changing column indicator. Used on dimension tables to indicate that the column is being managed as a slowly changing column within the context of a slowly changing dimension. Set when a column is identified during the dimension update procedure generation.
4	Previous value column indicator. Used on dimension tables to indicate that the column is being managed as a previous value column. The source column identifies the parent column. Set during the dimension creation.
A	Indicates that the column is part of the primary business key. Set whenever a business key is defined as part of an update procedure generation.
B-Z	Indicates that the column is part of a secondary business key. Only used during index generation and not normally set.

Source Table

Identifies the source table where the column's data comes from.

Source Column

Identifies the source column where the column's data comes from.

Transformation

Transformation. [Read-only].

Join

Indicates whether the table column is used in a table join. [Normally maintained automatically but can be optionally changed to override the default join logic used in the generated update procedure]. The **Source Table** and **Source Column** fields will provide the dimension table's side of the join. The options for this field are: False, True, Manual and Pre Join.

Setting this field to Manual changes the way the dimension table is looked up during the update procedure build. It allows you to join the dimension manually in the Cursor mapping dialog (where the 'Where' clause is built).

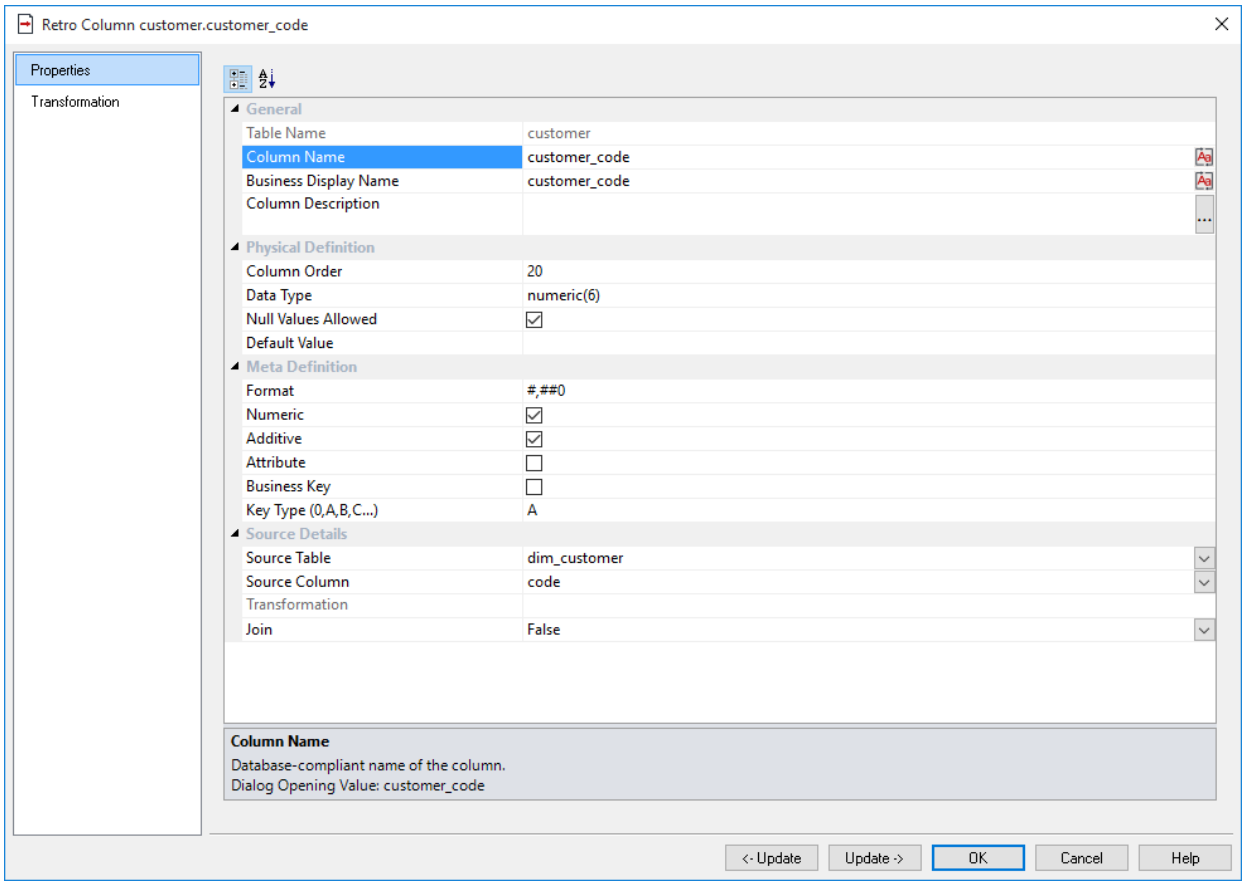
Setting this field to Pre Join activates the **Pre Join Source Table** field and allows you to select a table from the drop-down list.

Pre Join Source Table

Indicates the table from which the pre joined column was sourced. When the Join option is set to False, this field becomes inactive. When the Join option is set to True or Manual, this field is set to the current table name. When the Join option is set to Pre Join, then you can select the required table from the drop-down list.

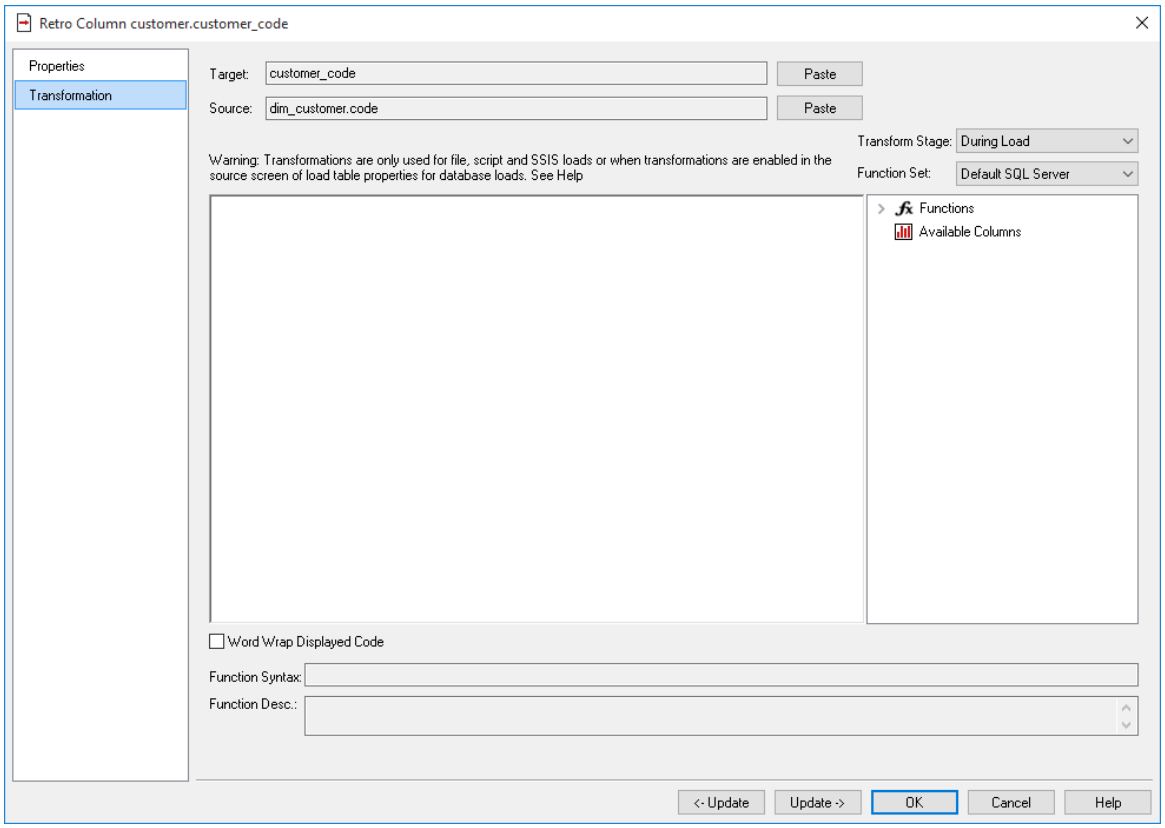
RETRO COLUMN PROPERTIES SCREEN

A sample **Properties** screen is as follows:



RETRO COLUMN TRANSFORMATIONS

It is possible to do transformations on Retro table columns. It is recommended that transformations are not performed on columns that are dimension keys or the business keys for the table. The transformation screen is as follows:



Note: Transformations are only put into effect when the procedure is re-generated.

See ***Transformations*** (on page 650) for more details.

WhereScape®

CHAPTER 34
INTEGRATING WHERESCAPE RED INTO AN EXISTING WAREHOUSE

Two main options exist in terms of bringing WhereScape RED into an existing data warehouse environment:

1 Rebuild tables and procedures with WhereScape RED.
2 Integrating existing tables and procedures into WhereScape RED.

Both options require manual coding changes to stored procedures. The main advantages and disadvantages of these two options are discusses below, and in detail in the following sections.

Rebuilding

The rebuild option is essentially a redevelopment of the existing data warehouse utilizing the knowledge acquired in the initial development and the rapid development capabilities of WhereScape RED. A rebuild will take more time and effort than just integrating existing tables and procedures but will provide a better platform on which to extend the data warehouse. See the ***Rebuild*** (see "***Rebuilding***" on page 1140) section for the approach to achieve this option.

Integrate

Existing data warehouse tables can be identified to WhereScape RED. The tables are seen and can be managed to a degree. The main disadvantage is the increased difficulty in utilizing these tables when trying to extend the data warehouse. This option is however significantly quicker and easier than a rebuild. It is discussed in detail in the ***Integrate*** (see "***Integrating***" on page 1141) section of this chapter.

The decision as to which option to choose will depend on the size and complexity of the existing data warehouse. Another important factor is the degree to which the existing data warehouse is to be extended. If future enhancements revolve around new analysis areas that have little overlap with the existing environment, then a integrate may be the best answer. If the data warehouse is small and relatively simple than a rebuild may be worth considering. In any event the best plan may be to do a test integrate and then re-evaluate the situation.

IN THIS CHAPTER

REBUILDING

The rebuild process essentially is a total re-creation of the data warehouse. One of the major impacts of such an approach is the **end user layer**, or rather the effect on the end user tools and saved queries and reports that are currently in use. The redesign or redeployment of this interface to the end users of the data warehouse may be too large a task to undertake. The problem can be circumvented to some degree though the use of views to make the new data warehouse environment look the same as the previous. But it is this impact and the subsequent re-testing process that must be considered when deciding to undertake a rebuild.

The advantages of a rebuild is the seamless integration of future analysis areas into the data warehouse and the single point of management that is provided. The major steps in the rebuild process will depend very much on the environment being replaced. As a guideline the following steps may be worth considering.

The rebuild process

1. Load a copy of the WhereScape metadata repository into an otherwise empty test environment. See the **Installation and Administration Guide** for instructions on how to load a metadata repository.
2. Ensure there are no public synonyms that point to existing table names, if the rebuild process is to use the same names as some or all of the existing tables.

   Working within the WhereScape RED tool proceed to:
3. Create connections to the new test data warehouse and to all the source systems.
4. Using the source system knowledge from the existing data warehouse, create the appropriate load tables in the data warehouse based on the existing extract or load methodology.
5. Build up the dimension, stage and fact tables using the same column and table names where possible.
6. Examine the existing procedures or update methodology and include this into the generated stored procedures.
7. Test the new environment.
8. Work out a plan to convert the existing data into the new data warehouse. Where possible it is best to keep existing key values and re-assign sequences to match these existing key values where appropriate.
9. Convert and test the old data warehouse data in the new environment.
10. Redeploy the end user tool access.

INTEGRATING

The integrate process

The steps in the integrate process are:

1. Create a test environment (database user) with the existing data warehouse tables loaded.
2. Load a copy of the WhereScape metadata repository into this test environment. See the **Installation and Administration Guide** for instructions on how to load a metadata repository.

   Working within the WhereScape RED tool proceed to:
3. Create any connections to UNIX or Windows servers where host scripts are currently executed. See either ***creating a Windows connection*** (see "***Windows***" on page 164) or ***creating a Unix connection*** (see "***UNIX***" on page 167).
4. Create a Data Warehouse connection mapping back to the test environment. See ***creating a connection to the data warehouse*** (see "***Database - Data Warehouse/Metadata Repository***" on page 143).
5. Incorporate any Host system scripts currently used. See ***incorporating host scripts*** (see "***Integrating, Host Scripts***" on page 1142).
6. Browse the Data Warehouse connection (Browse/Source Tables).
7. Drag and drop each existing data warehouse table into an appropriate object type. See ***Selecting an appropriate table type*** (see "***Integrating, Selecting a Table Type***" on page 1143)
8. Answer the retrofit questions, and build any required procedures. See ***integrate questions*** (see "***Integrating, Questions***" on page 1144).
9. Edit and amend all generated procedures, or create new procedures to handle the existing processing methodology. See ***procedure changes*** (see "***Integrating, Procedures***" on page 1148).
10. Test the new environment.

Removing the metadata for a table

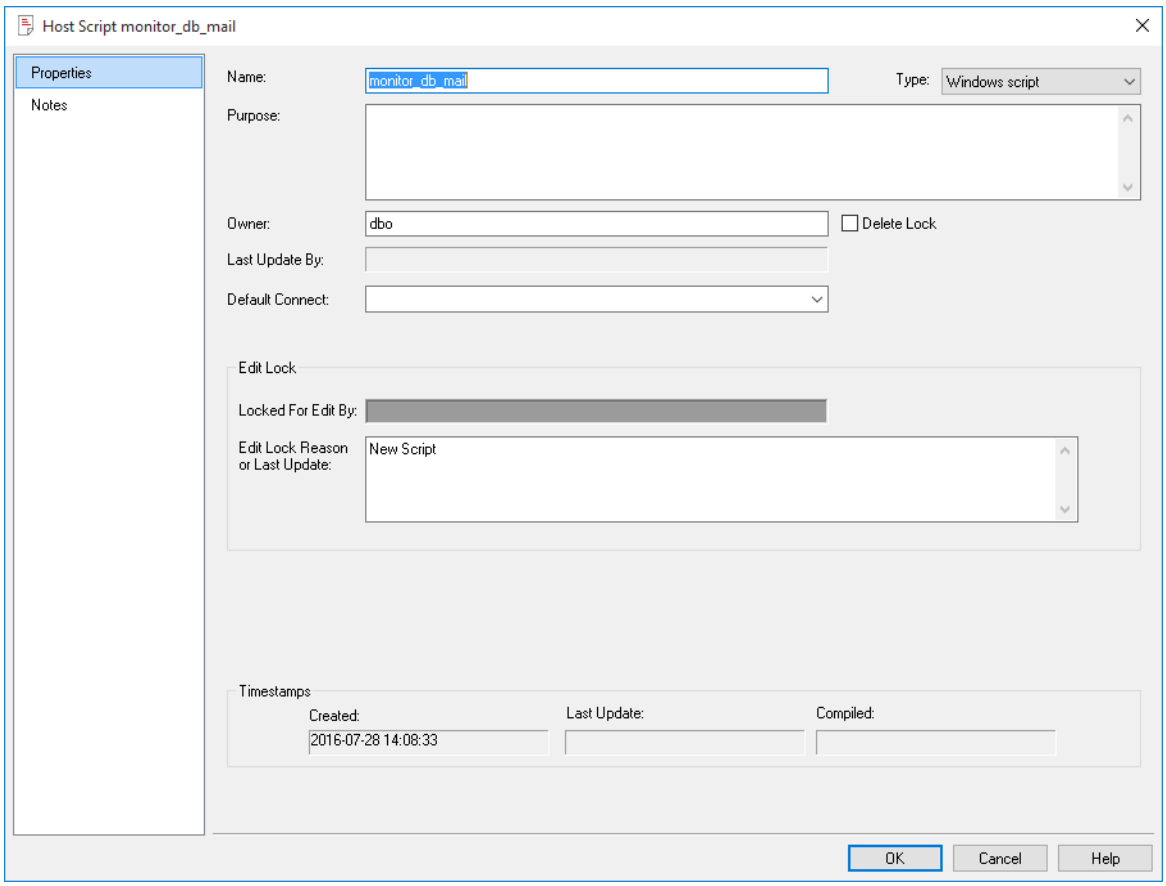
It is possible to delete the metadata for a table without deleting the table itself. For example if the integrate process is incorrectly undertaken, the metadata for the specific table can be removed. To delete the metadata only: First select the table and using the right-click menu select **Delete**. A dialog box will ask if you wish to delete the object and drop the table. Answer **No.** A second dialog box will now appear asking if just the metadata is to be deleted. Answer **Yes** to this question and only the metadata will be removed.

INTEGRATING, HOST SCRIPTS

Existing host scripts (either UNIX or Windows) can be brought into the WhereScape RED metadata. To incorporate an existing script the process is as follows:

1 Create a **Host Script** object using WhereScape RED. In the left pane click on a project or the **All Objects** project and using the right-click menu select **New Object**. The new object dialog box will appear. From the Object Type drop-down select **Host Script** and enter a name for the new script.

2 The following properties dialog will appear. Select the script type; either UNIX or Windows script. Select the appropriate connection from the **Default Connect** drop-down. Fill in the purpose field to define the role of the script and then click Update to store the changes.



3 Double-click on the new script or right-click on the the new script and select **Edit the Script**.

4 Within the script editor, either paste the script or if it is available on the PC, select the **File/Insert from File** option and load the file.

5 The script will need to be modified to handle the standards required by the scheduler. See ***Loading via a host script*** (see "***Script based load***" on page 304) for more details.

INTEGRATING, SELECTING A TABLE TYPE

When integrating existing tables, there may not be a clear decision as to which table type to use. As a guideline, the following groupings can be considered.

Temporary tables:

- Load tables
- Stage tables

Permanent tables:

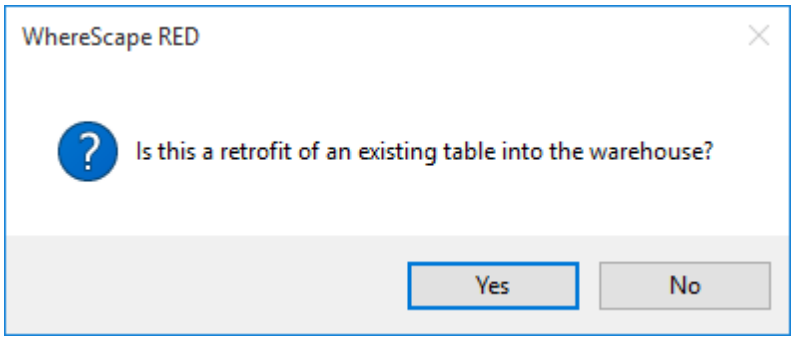
- Dimension tables
- Fact tables
- Aggregate tables

Although these table groups have very distinct names in terms of data warehousing, they do not impose any restrictions on the types of tables they contain. The table groupings are most relevant in the automatic generation of procedures, and in the sequencing for the scheduler.

Typically a mapping table may be stored in the Dimension table group.

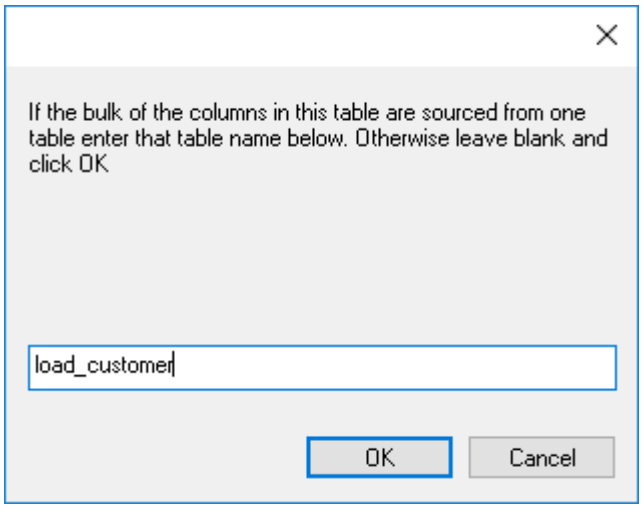
INTEGRATING, QUESTIONS

When a table within the data warehouse schema, that is unknown to WhereScape RED, is dropped onto a table target the following dialog appears.



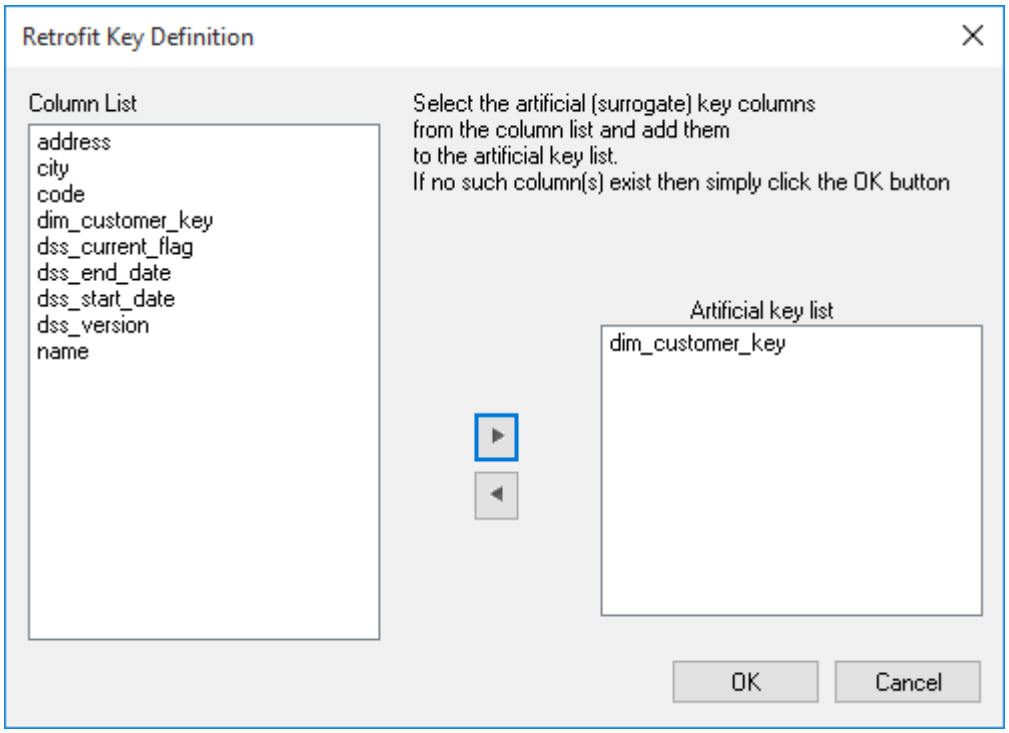
If this is an integrate then click **Yes** to proceed with the integrate process. The standard **New Object** dialog will appear and it would be advisable to leave the name of the object unchanged so that it matches the existing table.

A dialog will ask if the bulk of the columns in this table are derived from another table. If they are enter the table from which these columns derive at this stage. The purpose of this dialog is simply to set the **Source Table** field against each of the columns for the table.



Artificial Key definition

If the table target is a dimension or fact table then the following dialog will appear to enable the definition of any artificial or surrogate key. If no such key exists then simply proceed to the next question.



Get Key Function

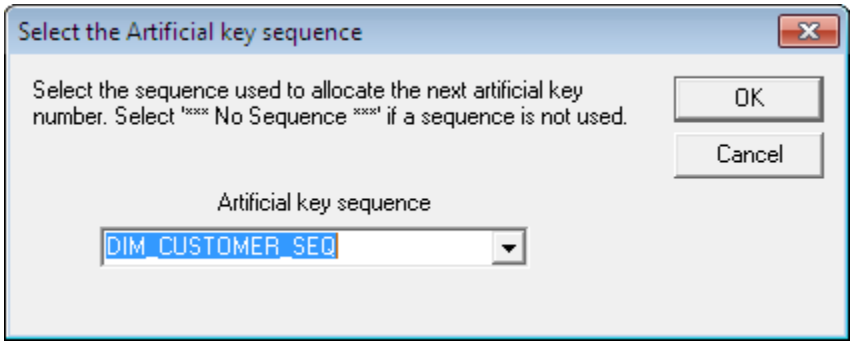
If the target table is a dimension you will be asked if you want to generate a get key function. The get key function is used by WhereScape RED for rapid prototyping. It returns the artificial key when passed the business key. Although this function is possibly not necessary in the existing environment it is useful if the dimension is to be used in any expansion of the data warehouse. If an existing get key type function exists, then this can be used, although changes will probably need to be made to the generated procedures to utilize this existing get key function.

Update Procedure

If the target table is a dimension, stage, fact or aggregate a dialog box will ask if an update procedure or update template is required. If selected a subsequent dialog will define the structure and content of this procedure. If an existing procedure is to be modified to include the scheduler imposed procedure standards it is best to select this option and use the existing procedure name to cut down on the amount of work required. The existing procedure can then be loaded into the generated procedure using the procedure editor.

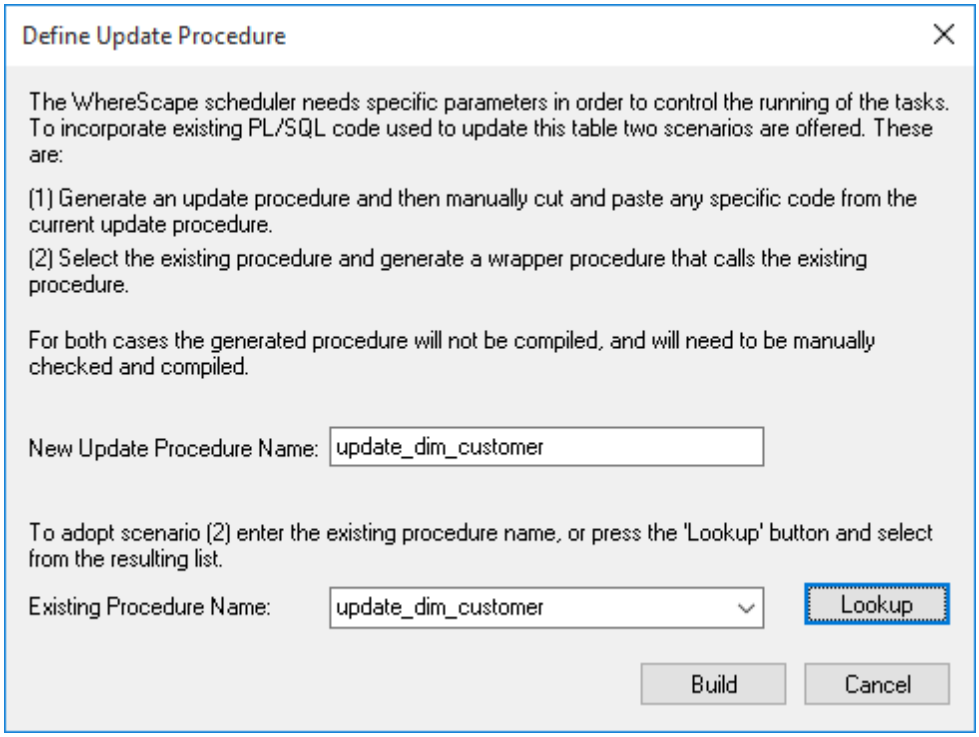
Artificial Key Sequence

If a get key function or update procedure is to be built and an artificial key is in use for the table then a dialog will prompt for the artificial key sequence. A drop-down list provides all the sequences defined for this database user. This sequence is an Oracle sequence that is used in the automatic generation of the artificial key. If no such sequence is used then select the *** no sequence *** option.



Update Procedure Definition

If an update procedure was selected, then the following dialog will display to help define how the update procedure is generated.



As defined in this dialog box the scheduler needs specific parameters in order to control and run a procedure. An existing procedure will not have these parameters. There are basically two forms of generated procedure. The first option is to generate a procedure that looks much like a standard WhereScape RED procedure and then manually insert the changes from the existing procedure. The second option is to generate a wrapper procedure that calls the existing procedure. This second option will still require manual changes to the procedure to return a valid return code and message.

Business Key definition

If a get key or update procedure has been selected a dialog will now prompt for the selection of the business key from the table. Multiple columns may constitute the business key, but they must uniquely identify each record in the table.

Index definition

Any indexes associated with the table will now be automatically defined and loaded into the metadata. Changes may need to be made in terms of when the indexes are dropped and to set the **Drop Before Update** checkbox if appropriate and the scheduler is to be used to manage these indexes.

Procedure creation

If defined the get key and update procedures will now be generated. They will need to be manually edited and compiled. See the section on ***integrating procedures*** (see "***Integrating, Procedures***" on page 1148).

INTEGRATING, PROCEDURES

The procedures managed by the WhereScape scheduler require the following standards.

Parameters

The procedure must have the following parameters in the following order:

Parameter name	Input/Output	Oracle Type	SQL Server/DB2 Type
p_sequence	Input	Number	Integer
p_job_name	Input	Varchar2	Varchar(256)
p_task_name	Input	Varchar2	Varchar(256)
p_job_id	Input	Number	Integer
p_task_id	Input	Number	Integer
p_return_msg	Output	Varchar2	Varchar(256)
p_status	Output	Number	Integer

The input parameters are passed to the procedure by the scheduler. If the procedure is called outside the scheduler then the normal practice is to pass zero (0) in the sequence, job_id and task_id. A description of the run can be passed in the job name and the task name is typically the name of the procedure.

The output parameters must be populated by the procedure on completion. The return_msg can be any string up to 256 characters long that describes the result of the procedures execution. The status must be one of the following values:

Status	Meaning	Description
1	Success	Normal completion
-1	Warning	Completion with warnings
-2	Error	Hold subsequent tasks dependent on this task
-3	Fatal Error	Hold all subsequent tasks

The major task in integrating a procedure will be in adapting it to the WhereScape scheduler standards and work flow.

INTEGRATING, VIEWS

When integrating views an additional step is required if you want WhereScape RED to be able to recreate the view.

The view will be mapped correctly and the **Get Key** function can still be built. This step is only required if the view is to be re-created.

Change the source column on the artificial key to match the artificial key in the table from which the view was created.

INTEGRATING, WHERESCAPE TABLES

When integrating WhereScape generated tables and views a number of additional considerations need to be taken.

The Dimension tables and views make use of secondary business key columns such as dss_source_system_key, dss_current_flag, dss_end_date, dss_version to assist in handling various forms of slowly changing dimensions and non conformance. These secondary business keys need to be flagged as such against the dimension.

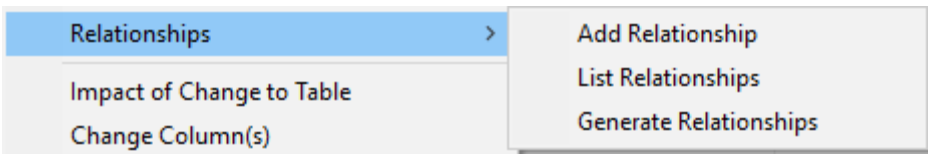
Change the properties of all such columns. The key type should be set to 1, and the primary business key checkbox should be set.

CHAPTER 35
RELATIONSHIP MAINTENANCE

Relationship Maintenance is available for the maintenance of joins between objects; providing a way to record joins between tables when surrogate keys are not being used. This functionality then enables the generation of **Links Diagrams** for these objects.

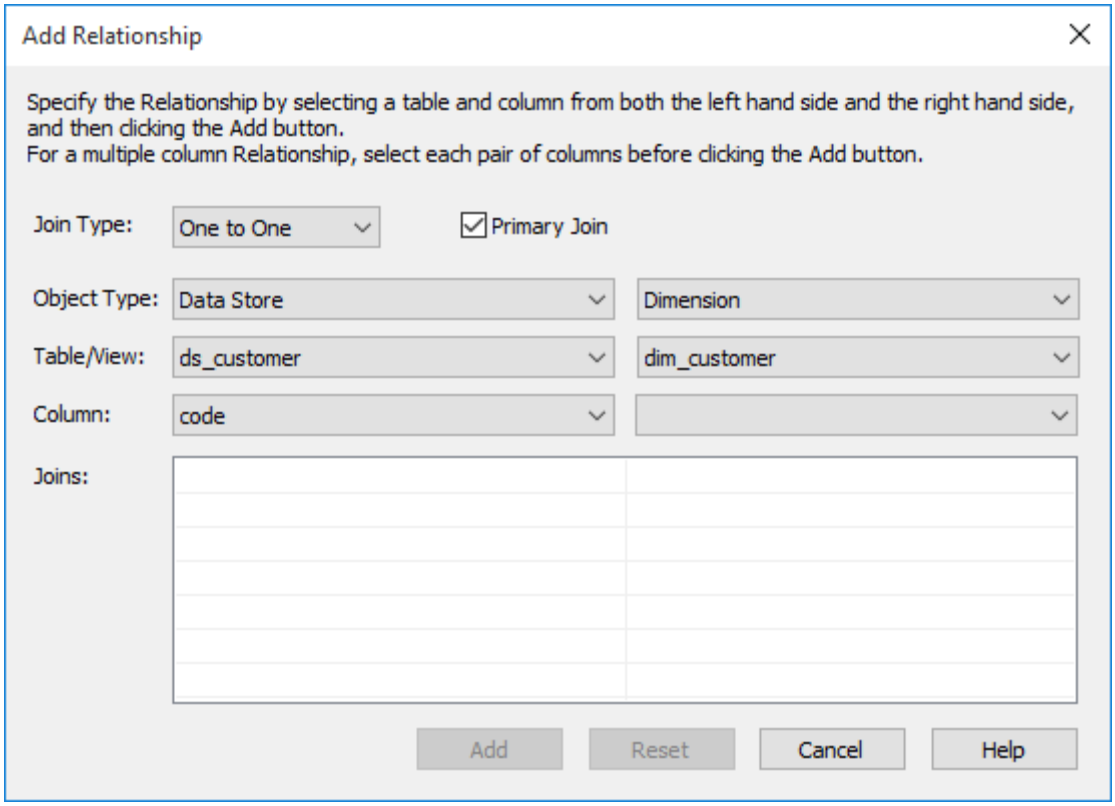
NOTE: It is necessary to explicitly specify relationships for tables on **Tabular** target databases for the relationships to be created in the Tabular database.

Relationship Maintenance options are available in the Relationships sub-menu when right-clicking on an object in the **Object Pane**.



ADD RELATIONSHIP

To add a relationship, right-click on the object in the **Object Pane** and select **Relationships->Add Relationships**. The following dialog appears.



For each object in the relationship, enter in the following details:

Join Type

For the **Join Type**, choose between the following:

- Undefined
- Many to One
- One to One
- One to Many
- Many to Many

Primary Join

If this is a primary join, select the **Primary Join** check-box. For MSAS Tabular tables this defines an active relationship. At most, one relationship between two specific tables can have this enabled.

Object Type

Enter the **Object Type** from the drop-down box (Data Store, Load table, Stage table, etc.).

Table/View

Enter the name of the object in the relationship.

Column

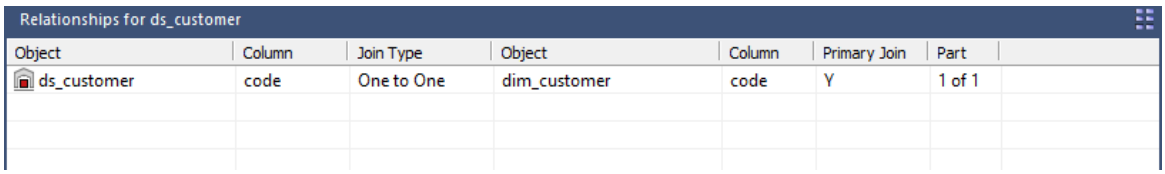
Enter the **Column** to join in each object.

Once you have entered the details for the join, the joined columns are displayed in the list of **Joins** at the bottom of the dialog box. Erroneous joins can be removed by right-clicking on the join and selecting **Remove Join**. All joins can be removed by clicking the **Reset** button.

To add all the relationships shown in the list of **Joins**, click the **Add** button.

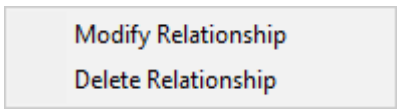
LIST RELATIONSHIPS

To view relationship for an object, right-click on the object in the **Object Pane** and select **Relationships->List Relationships**. The relationships for the selected object are displayed in the **Drop Target Pane** (middle pane).



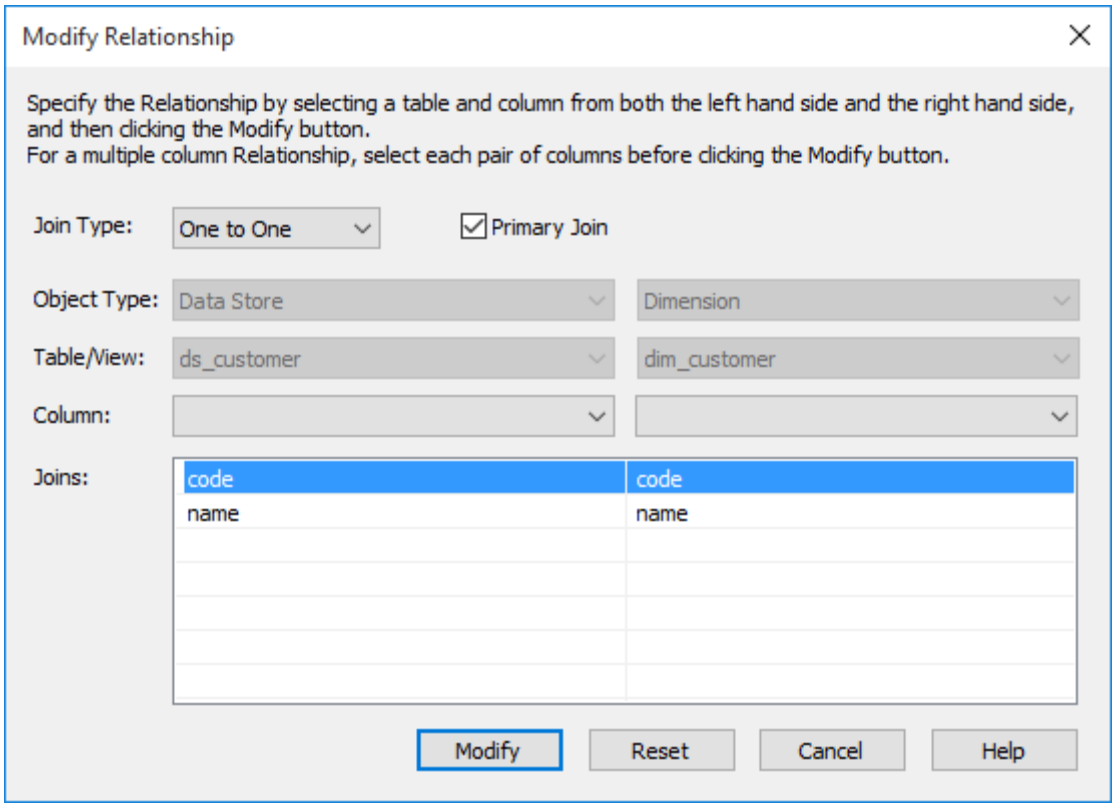
Multi-column joins are shaded when one join is selected.

Right-clicking on a column or join displays the following menu:



Modify Relationship

The Modify Relationship option shows the following dialog, allowing the editing of joins (including multi-column joins) between the two objects in the selected relationship.



Relationships are edited in this dialog in the same way as the **Add Relationship** dialog above. **Object Types** and **Table names** cannot be modified.

Delete Relationship

Deletes the selected relationship.

GENERATE RELATIONSHIPS

To generate relationships in metadata for an object, right-click on the object in the **Object Pane** and select **Relationships->Generate Relationships**. Results are shown in the **Results Pane**.

CHAPTER 36
UPGRADING RED

WhereScape Versions

WhereScape RED has a four part version number normally shown as xx.xx.xx.xx. An example of this may be 6.0.6.0. The first number represents the major release. The second number represents the metadata repository version. The third and fourth numbers relate to application specific releases.
From the example above we see that the current version is version 6 of WhereScape RED. We are on version 6.0 of the metadata repository.

Metadata Changes

A change from a 6.0.. release to a 6.5.. release would indicate a change in the metadata tables. All metadata changes are non destructive. They simply add new columns or new meta tables. In this way they can be applied without harming an existing metadata repository. The impact of a metadata change is that the associated applications (namely the RED executable, the Scheduler, security module etc.) will need to be at the same metadata version. Therefore a 6.0.6.0 version of RED may not successfully run against a version 6.5 metadata repository.

Application changes

The final two numbers in the version represent application releases. Applications are deemed to be all of the executable images supplied with WhereScape RED as well as the UNIX scheduler scripts and the stored procedures. Application changes reflect enhancements and bug fixes. A change in the first number indicates a major enhancement in one of the application components.

Upgrading RED

Upgrading WhereScape RED consists of the following steps:

1. Allow any active jobs to complete. Halting active jobs will allow running tasks to complete with no new tasks starting. Aborting active jobs will kill any running tasks and stop running jobs.
2. Stop all schedulers. Windows schedulers can be stopped with WhereScape's Setup Administrator. To stop a UNIX or Linux scheduler, kill the active scheduler process and comment their crontab entries (to stop the scheduler re-starting itself).
3. Close any WhereScape programs that are running on your machine.
4. Install the new version of RED on your machine.
5. Back up your metadata.
6. In RED Setup Administrator, select the Validate Metadata Repository option. This function may re-compile existing or create new metadata procedures; metadata tables may be altered or new tables created.
   When performing an upgrade of RED:
   - Click **OK** when prompted to validate metadata tables.
   - Click **Yes** when prompted to re-create metadata views.
   - Click **Yes** when prompted to re-compile metadata procedures.

Warning: If the above steps are not all completed during an upgrade the metadata may be left in an inconsistent state.

7 For **Oracle** metadata updates see the note below on **recompiling Oracle invalid procedures** during a RED upgrade.

8 Back up your metadata again (just in case).

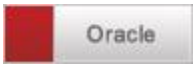
9 If using a UNIX or Linux scheduler and this is a major application enhancement then rename the wsl/bin directory to say wsl/bin_versionxxxxxx. Create a new bin directory and ftp over the files under the unix directory (see the main install instructions). Change the protections on the files (chmod 750 *.sh).

10 Restart all schedulers. Windows schedulers can be restarted with WhereScape's Setup Administrator. For any UNIX or Linux schedulers, uncomment any commented out crontab entries - this is enough to restart the schedulers.

11 Restart any halted or aborted jobs.

NOTES:

1. There are different versions of the scripts for each database and for UNIX versus Linux. You should also be looking in the sub directory with the highest version number.



When using a UNIX/Linux scheduler for Oracle, It is recommended that users copy the new meta_backup_680.sh file to use Datapump'sexpdb/impd instead the deprecated exp/imp tools. This version uses the data pump export executables expdp/impdp. It assumes that the scheduler and the Oracle database reside on the same server.

2. Metadata tables do not change between minor releases, but metadata procedure often and usually do change.

3. RED will not let you sign into an old repository version using a newer version of RED

4. RED will let you sign into a new repository version using an older version of RED, but it will warn you that this may potentially cause issues

5. It is very important when using the windows scheduler to have the installed RED version on the scheduler server EXACTLY matching the stored procedures.

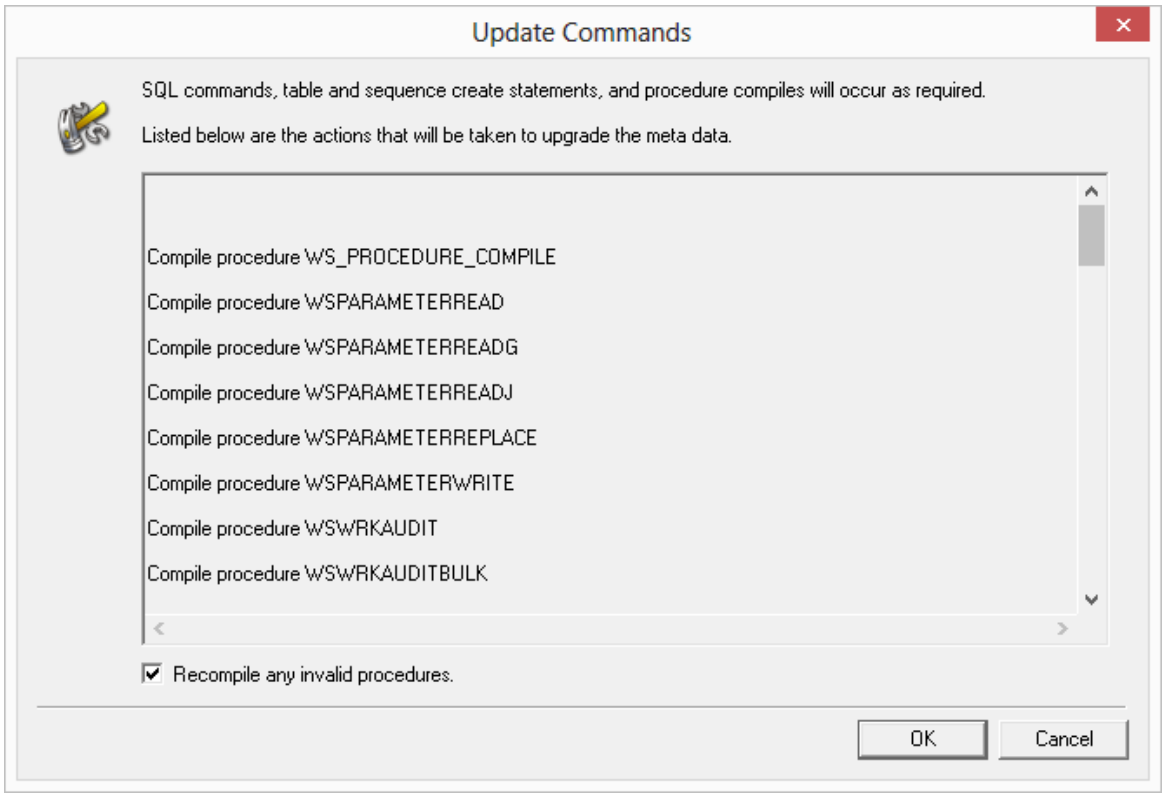
6. Side by side installations are possible (two versions in two directories on the same machine), but be careful with schedulers. If you install the new version of RED in a new directory, you will have to remove and reinstall all windows schedulers in order for the scheduler(s) to use the new version of RED.

7. As a general rule, the UNIX and Linux scheduler scripts, metadata tables/procedures and RED front end should all be in sync.

8. If the scripts for the UNIX or Linux scheduler have changed, you should replace files in your old and new shell scripts in the WhereScape Program Files directory and see if any have changed.

Oracle invalid procedures:

When validating an Oracle Metadata Repository, tick the **Recompile any invalid procedures** check-box to have any invalid user procedures recompiled during a metadata validate.



WhereScape®

CHAPTER 37
LOGIN CHECKS

The following checks are performed during login; and if necessary, warning messages are displayed, see more about each specific warning by following the links below:

Login Data Source does not match the data warehouse connection's ODBC DSN

More than one connection has the data warehouse option enabled

Oracle Individual User

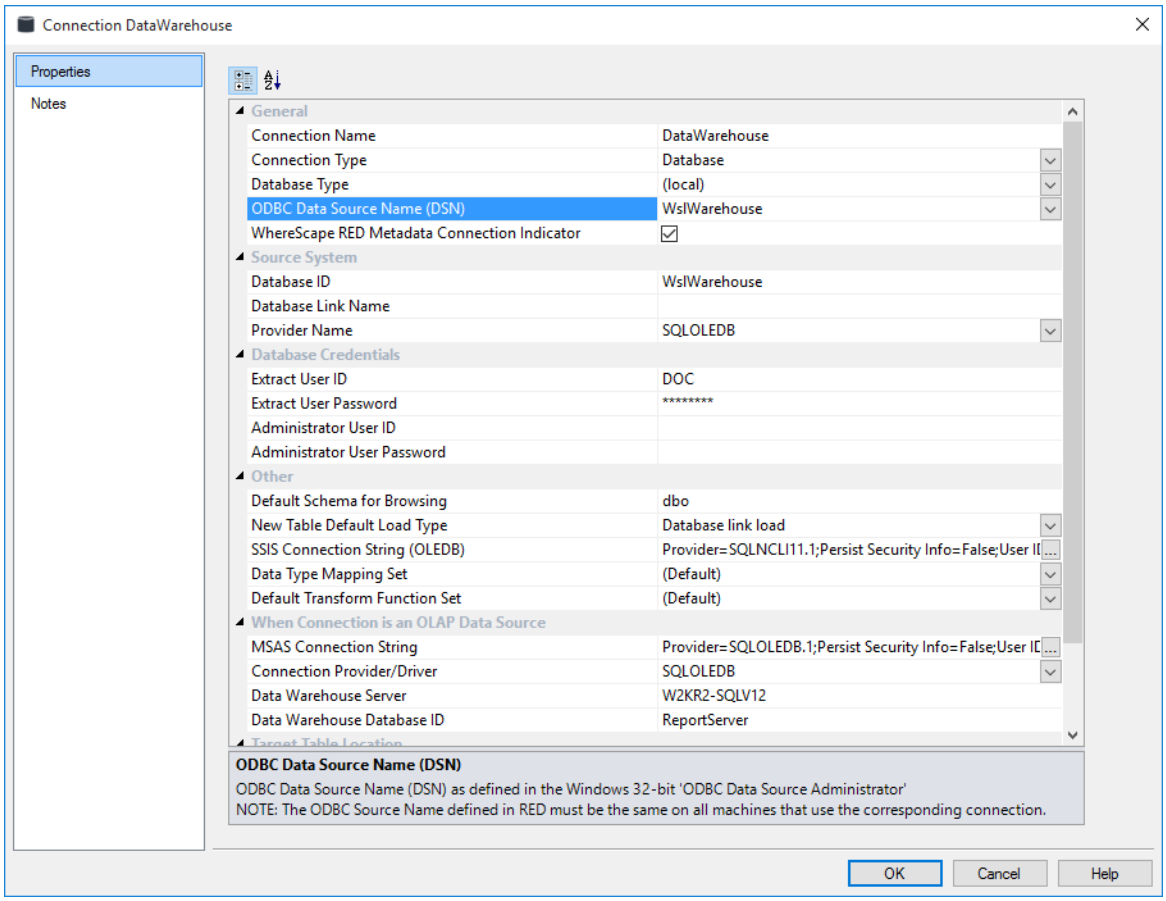
IN THIS CHAPTER

DATA SOURCE DOES NOT MATCH THE DATA WAREHOUSE CONNECTION'S ODBC DSN

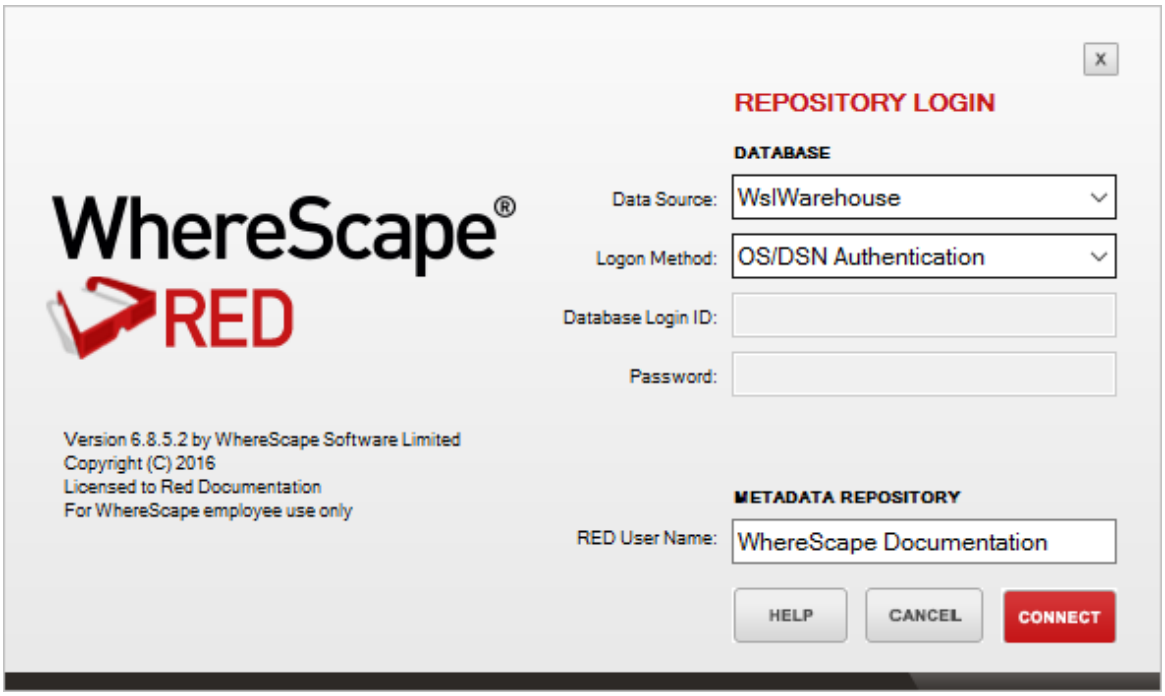
Warning about the login Data Source not matching the data warehouse connection's **ODBC DSN:**

You can correct this issue by performing one of the following actions:

- Alter the Data Warehouse Connection **ODBC Data Source Name (DSN)** to match the login **Data Source**.



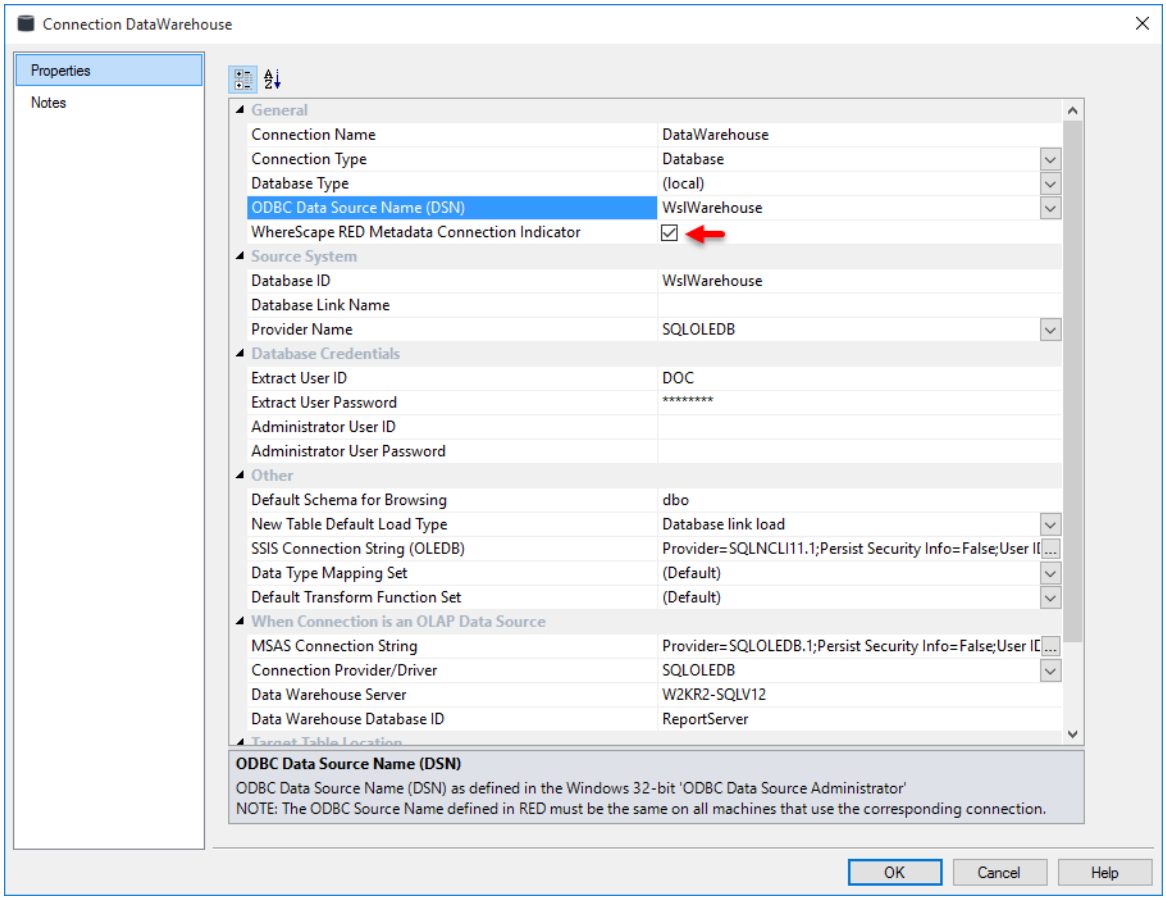
- Logoff and log back in using the **Data Source** with the same name as the Data Warehouse Connection **ODBC Source**.



MORE THAN ONE CONNECTION HAS THE DATA WAREHOUSE INDICATOR ENABLED

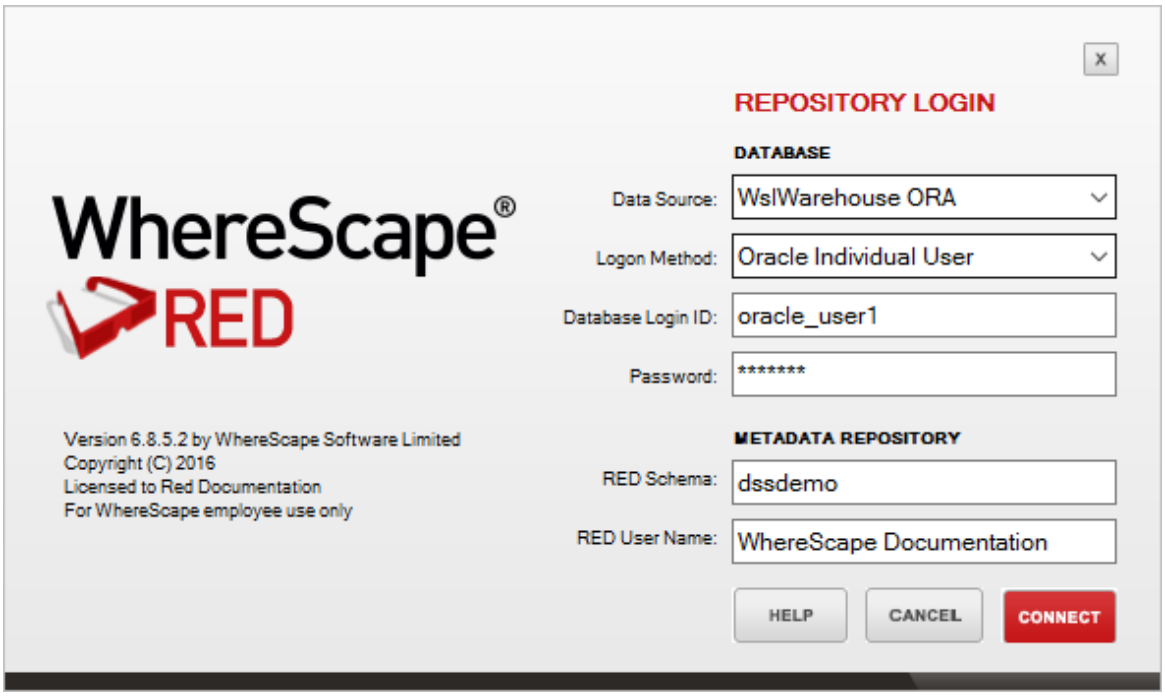
Warning about more than one connection having the data warehouse option enabled:

You can correct this issue by editing all the connections and making sure that only one is set to Data Warehouse:



ORACLE INDIVIDUAL USER

For any errors or for more information about logging in, or creating an **Oracle Individual User**, see section **9.3.1 Creating an Oracle Individual User** of the Installation Guide.



WhereScape®

CHAPTER 38
DATA TYPE MAPPINGS

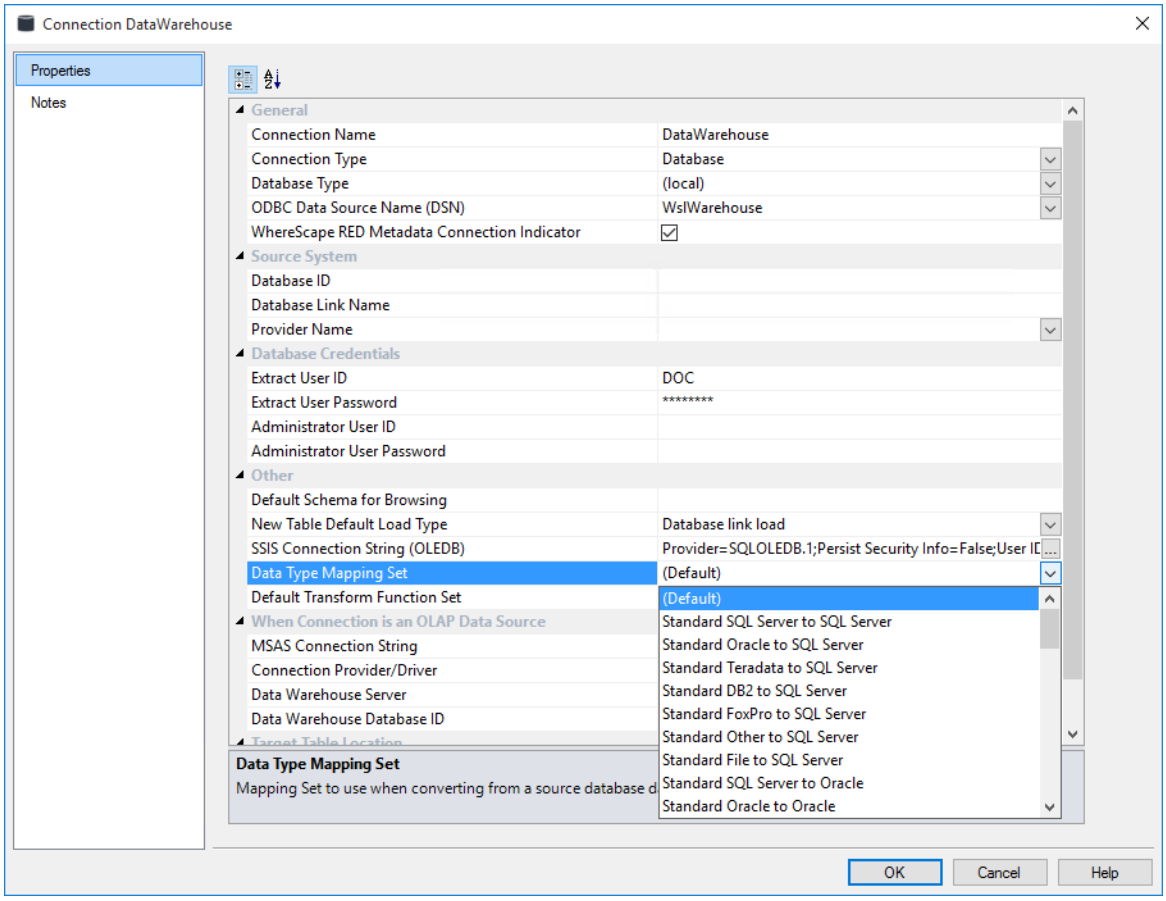
IN THIS CHAPTER

USING DATA TYPE MAPPING SETS

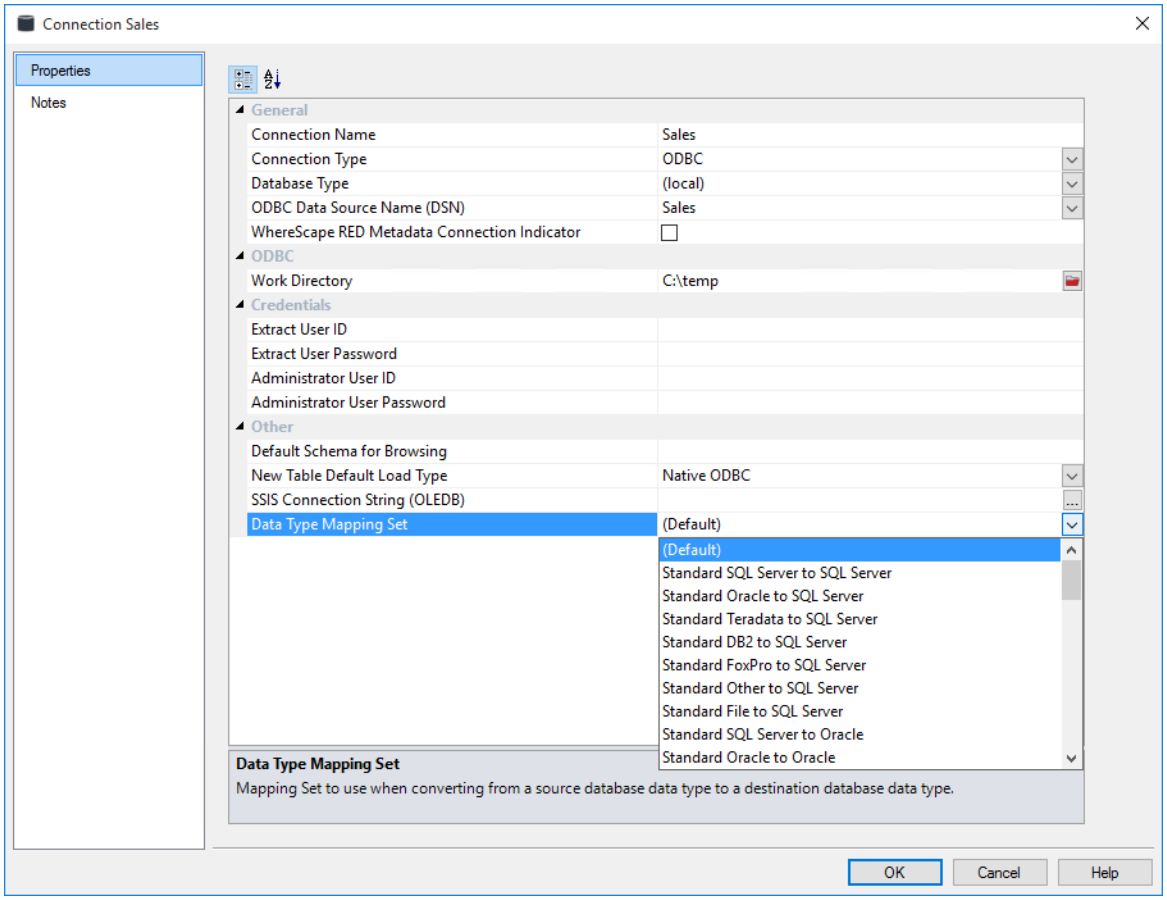
Data type mapping sets contain a list of mappings that are used when loading tables into the data warehouse.

Custom data type mapping sets give you the ability to automatically change the data type of any column or to add column transformations when dragging and dropping new load tables. These mapping sets may be created, edited, deleted, imported and exported using the **Data Type Mappings** options on the **Tools** menu.

Data Warehouse Connection Properties Dialog



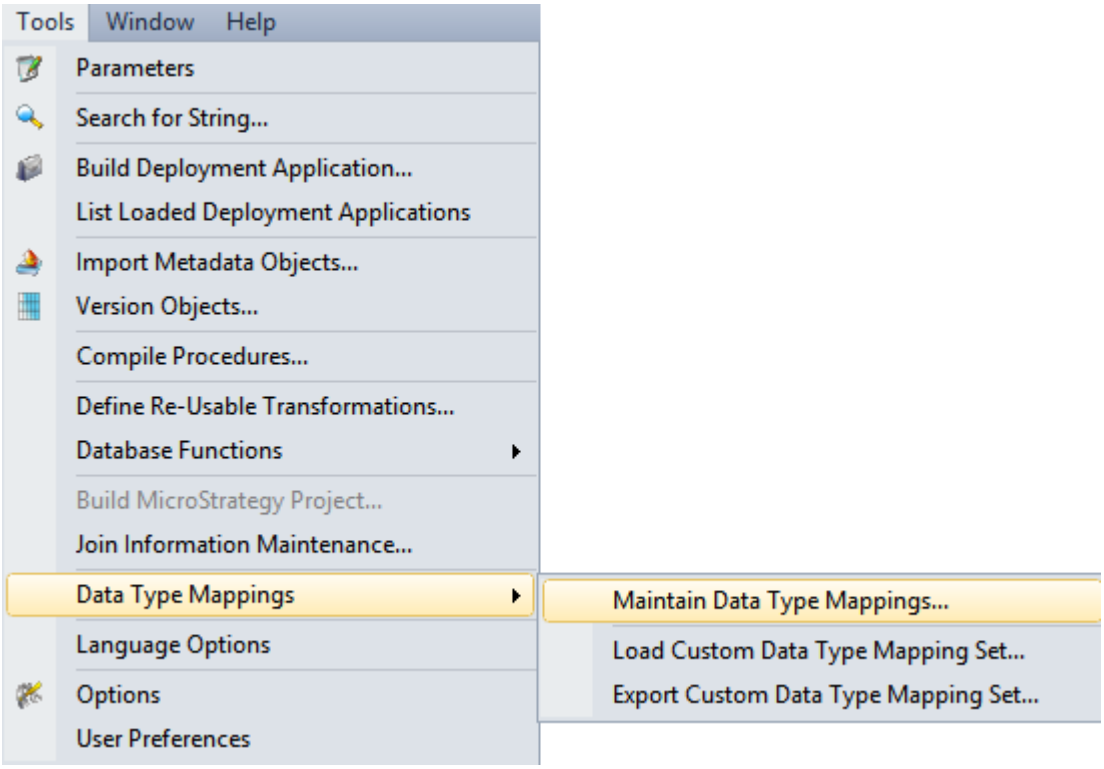
Non-Data Warehouse Connection Properties Dialog



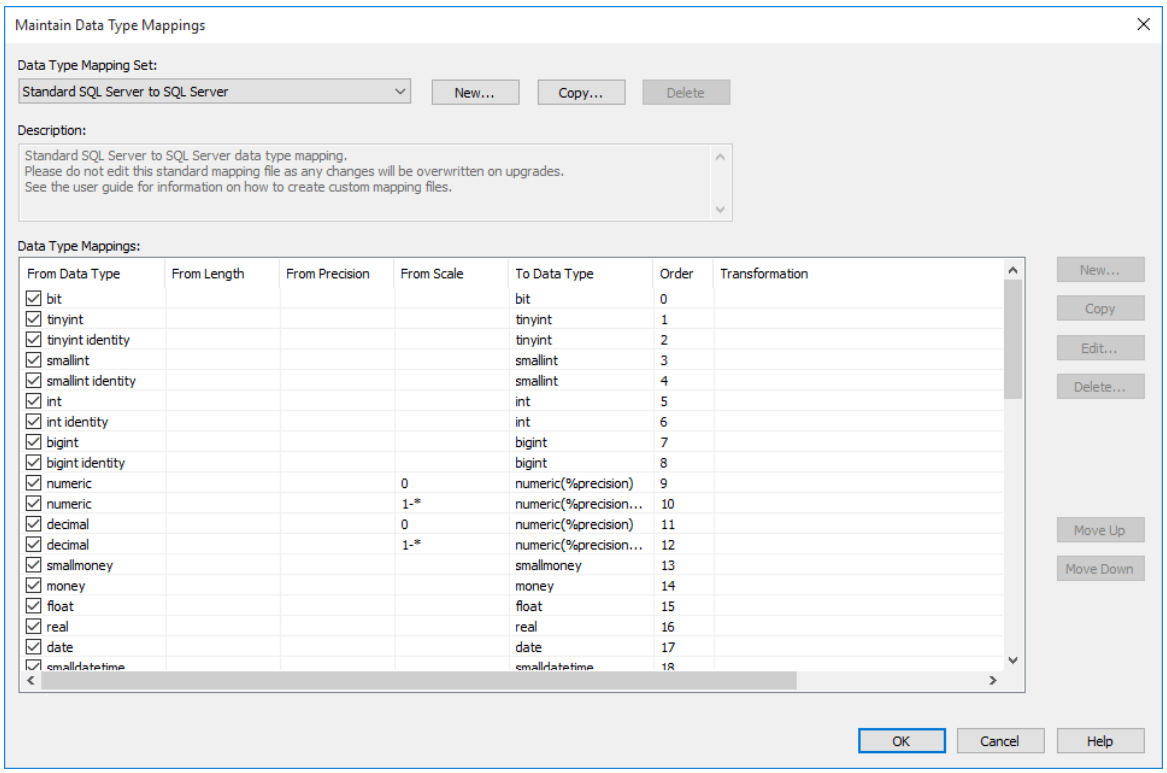
In the Connection Properties dialogs, a drop-down list **Data Type Mapping Set** is displayed. This is the default set that will be used when loading tables into the data warehouse.

MAINTAINING DATA TYPE MAPPING SETS

To maintain the data type mapping sets, select **Tools/Data Type Mappings/Maintain Data Type Mappings...**



The **Maintain Data Type Mappings** dialog is displayed.



Select a data type mapping set from the **Data Type Mapping Set** drop-down list. Of the standard files, only the files relevant to the database that you are on will be displayed in the list.

To create a data type mapping set, see ***Creating a New Data Type Mapping Set*** (on page 1169)

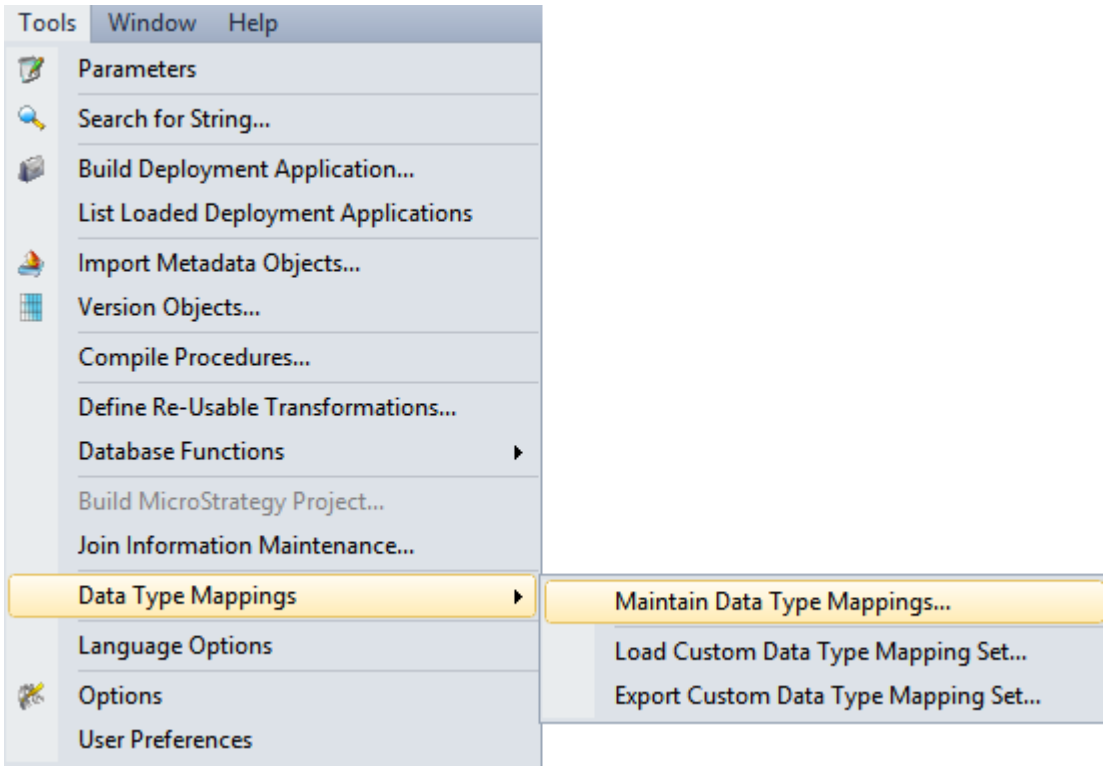
To copy a data type mapping set, see ***Copying a Data Type Mapping Set*** (on page 1173)

To edit a data type mapping set, see ***Editing a Data Type Mapping Set*** (on page 1177)

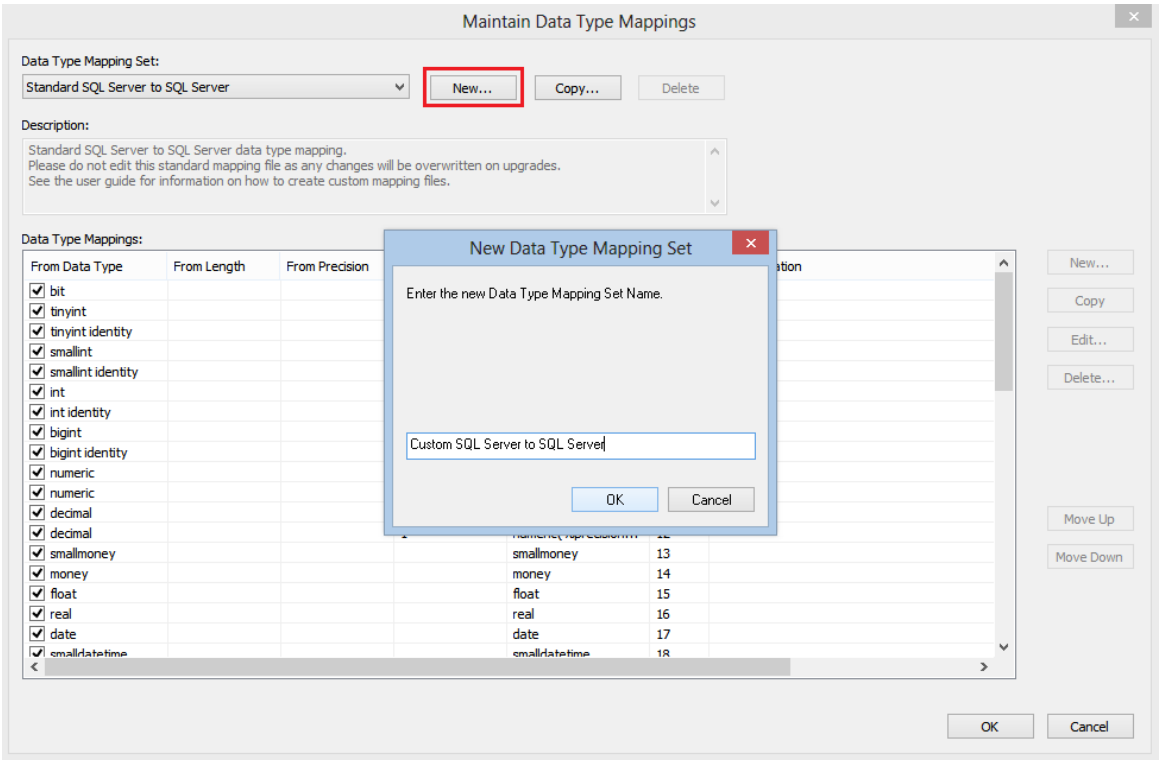
To delete a data type mapping set, see ***Deleting a Data Type Mapping Set*** (on page 1186)

CREATING A NEW DATA TYPE MAPPING SET

To create a new data type mapping set, select **Tools/Data Type Mappings/Maintain Data Type Mappings...**



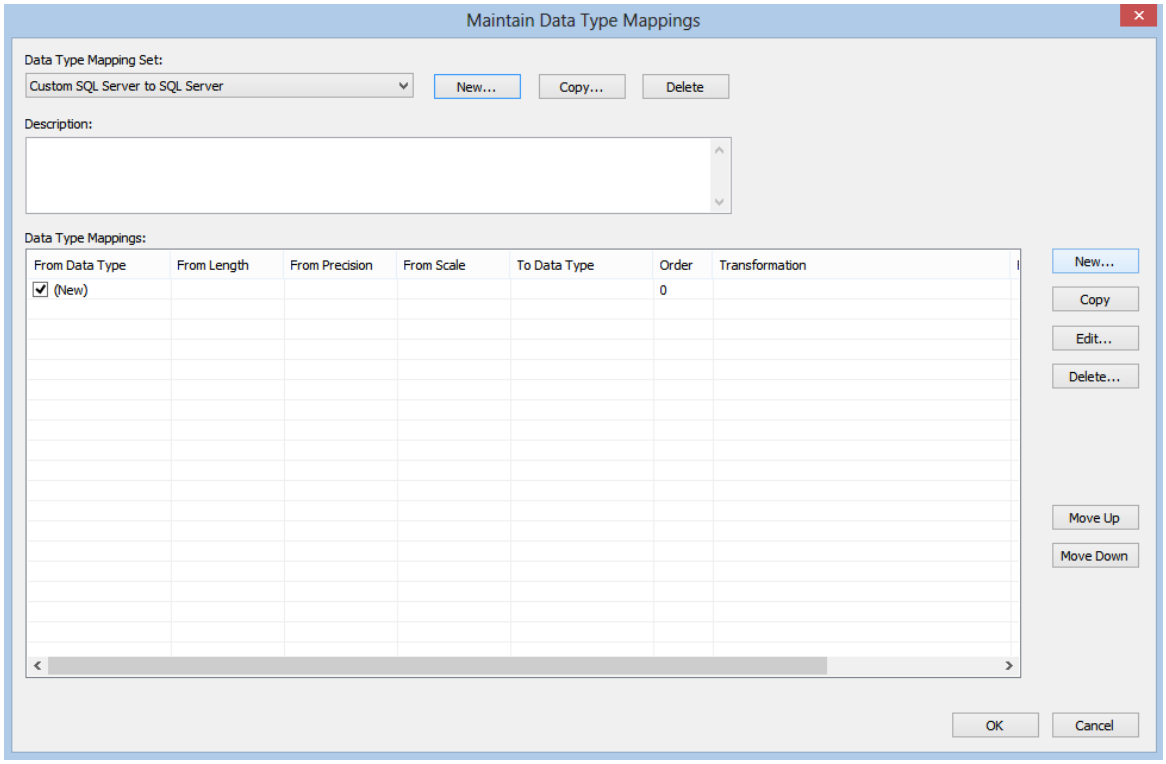
Click on the **New** button, enter the name of the new Mapping Set and click **OK**.



Description

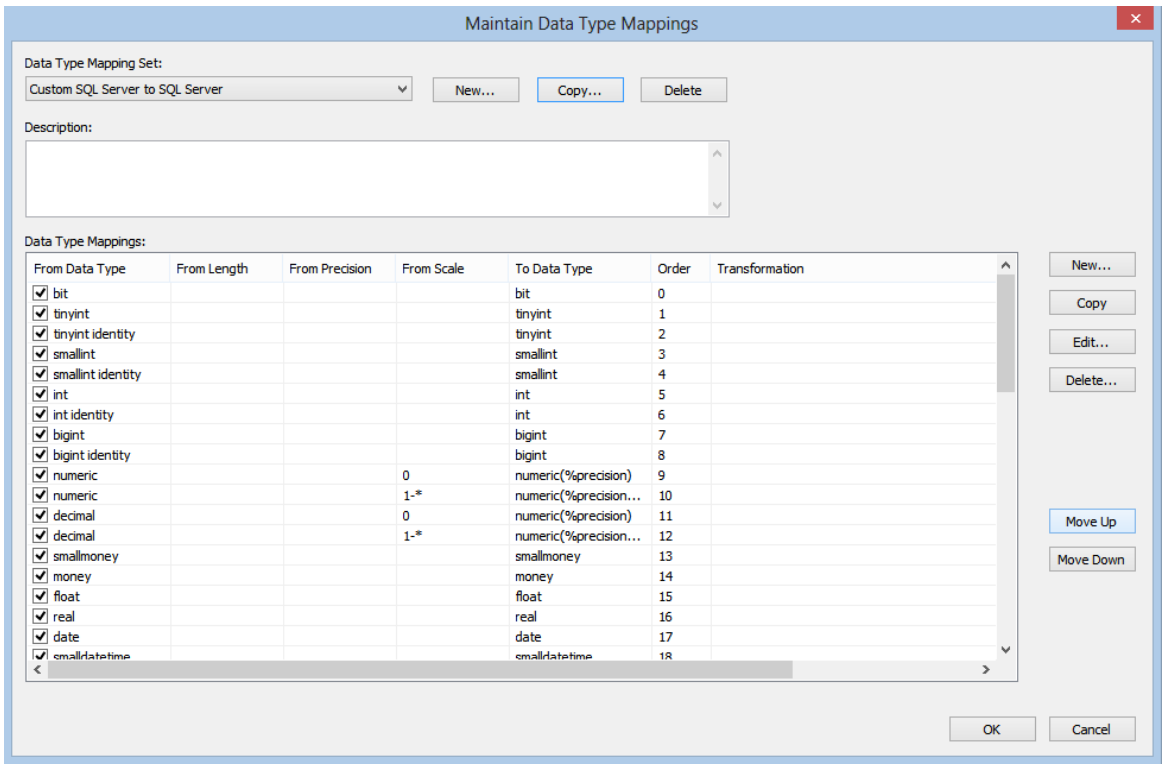
Enter the description for the data type mapping set.

You can then enter the individual mappings by clicking the **New** button on the right side of the dialog.



Similarly, you can delete an individual mapping, using the **Delete** button on the right side of the dialog; or you can edit a mapping, using the **Edit** button on the right side of the dialog.

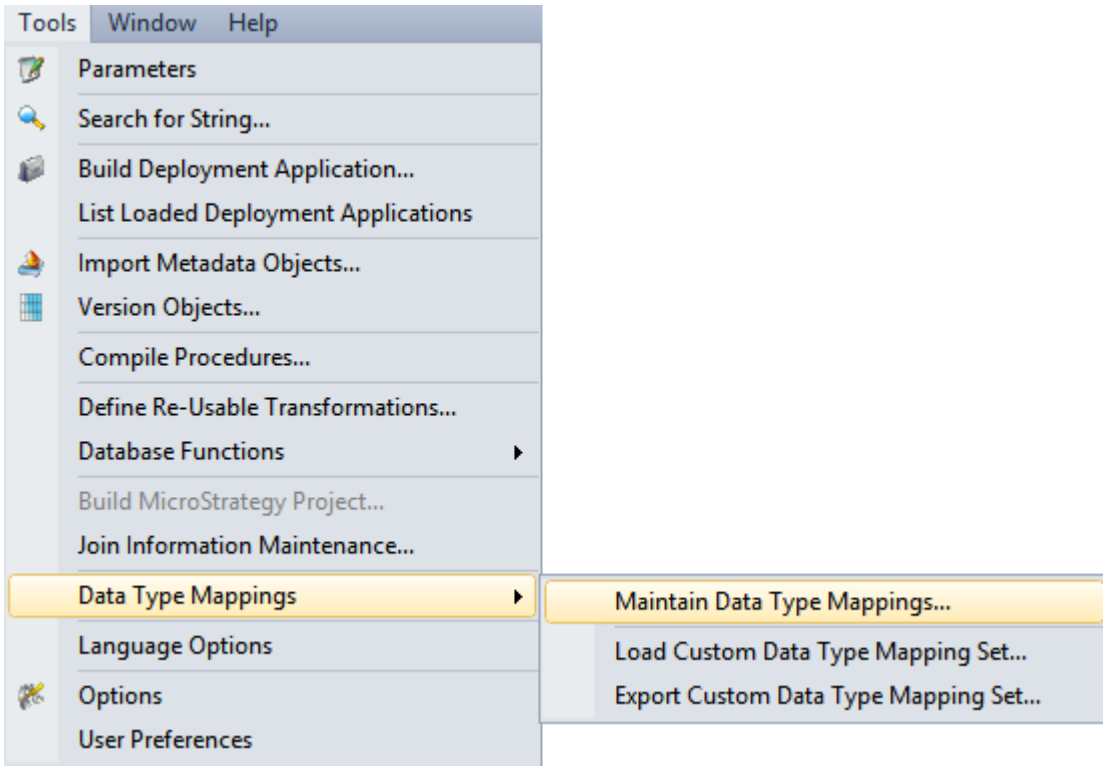
To move a mapping up or down in the list, select a mapping and then click on the **Move Up** or **Move Down** button on the right side of the dialog.



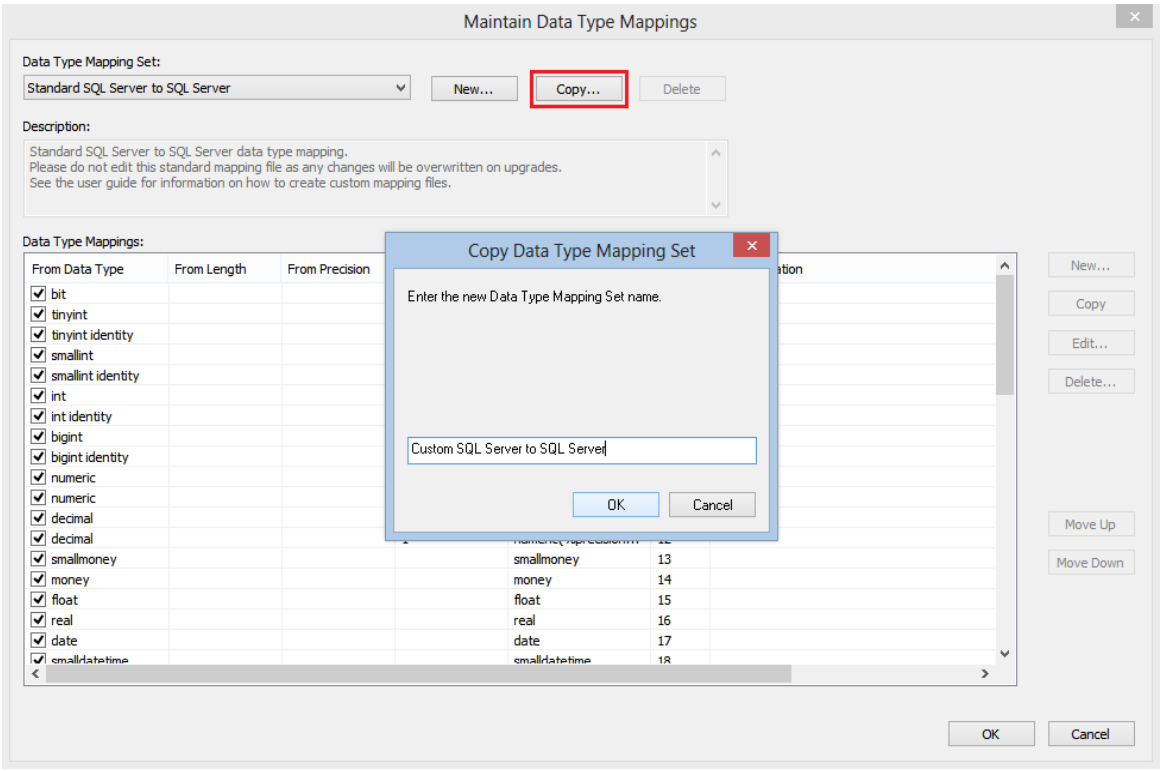
Note: The order of the data type mappings is significant as when loading a table, the procedure checks the data type mappings from top to bottom and stops when a data type and its parameters are correctly matched. A blank parameter means that it will match to anything.

COPYING A DATA TYPE MAPPING SET

To copy an existing data type mapping set, select **Tools/Data Type Mappings/Maintain Data Type Mappings...**



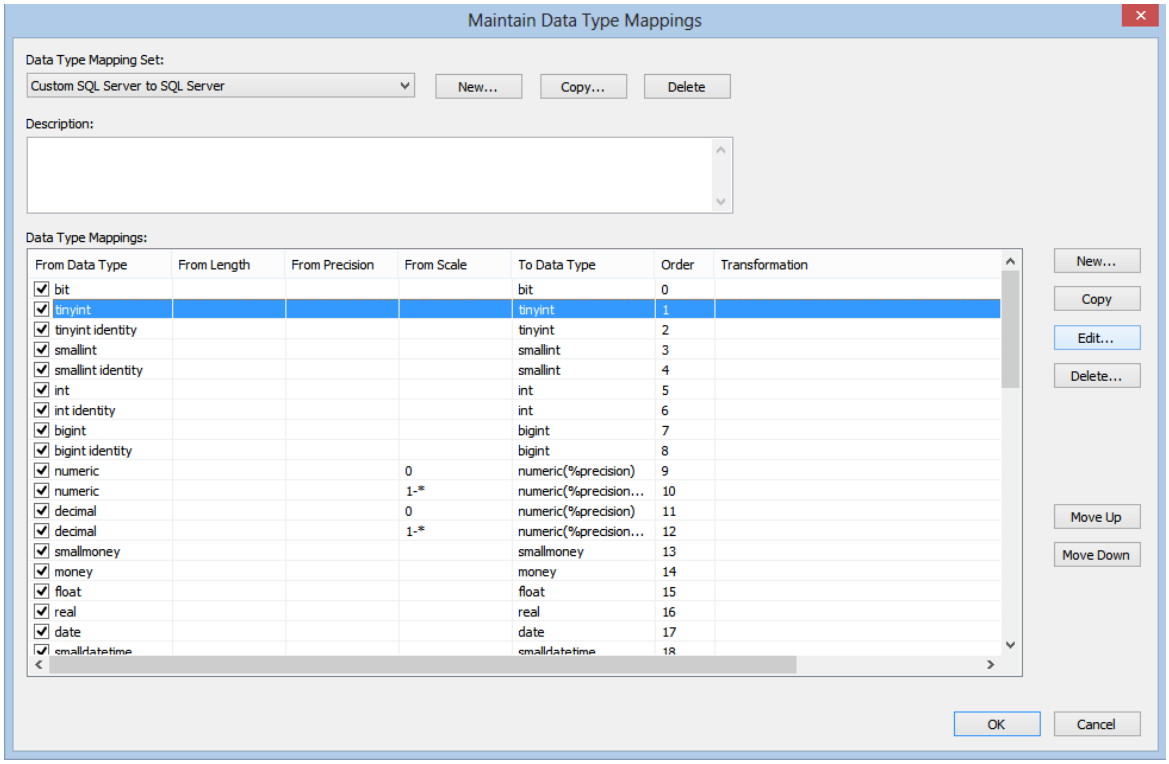
Select the Mapping Set to be copied from the drop-down list and then click on the **Copy** button. Enter the name of the new Mapping Set and click **OK**.



Description

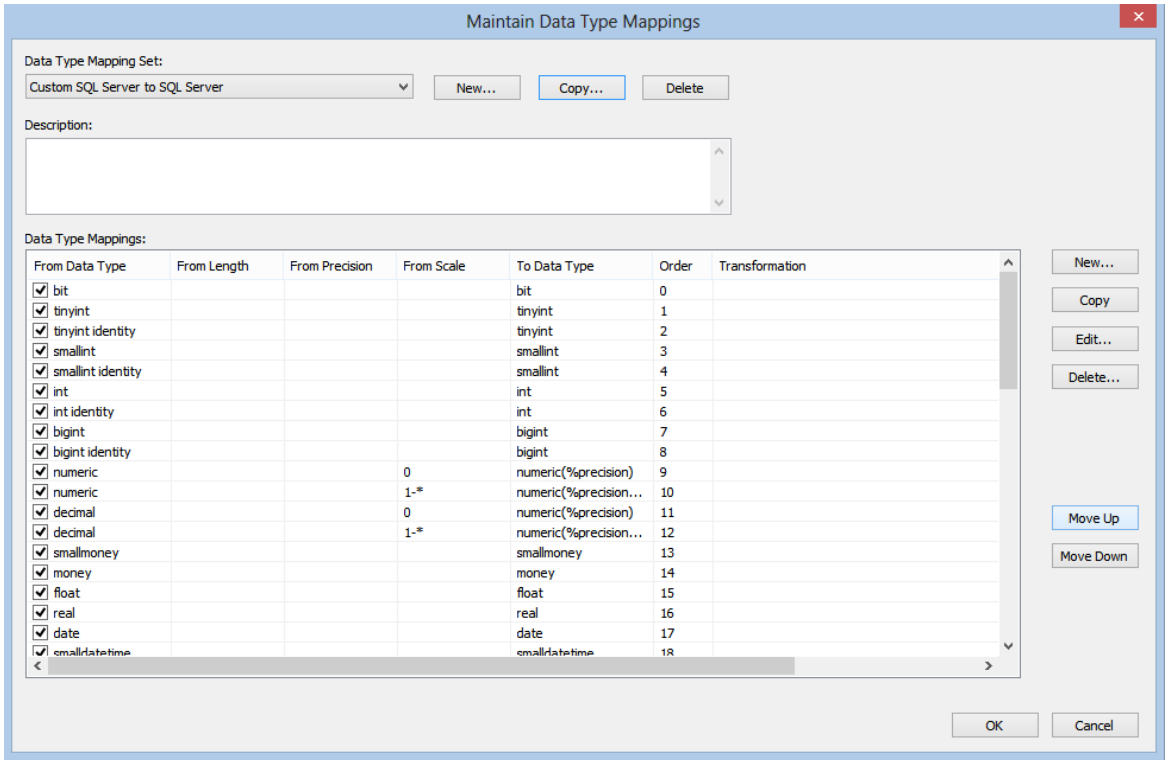
Enter the description for the data type mapping set.

To edit a data type mapping, select the mapping and then click on the **Edit** button on the right side of the dialog.



Similarly, you can delete a mapping, using the **Delete** button on the right side of the dialog; or you can add a new mapping, using the **New** button on the right side of the dialog.

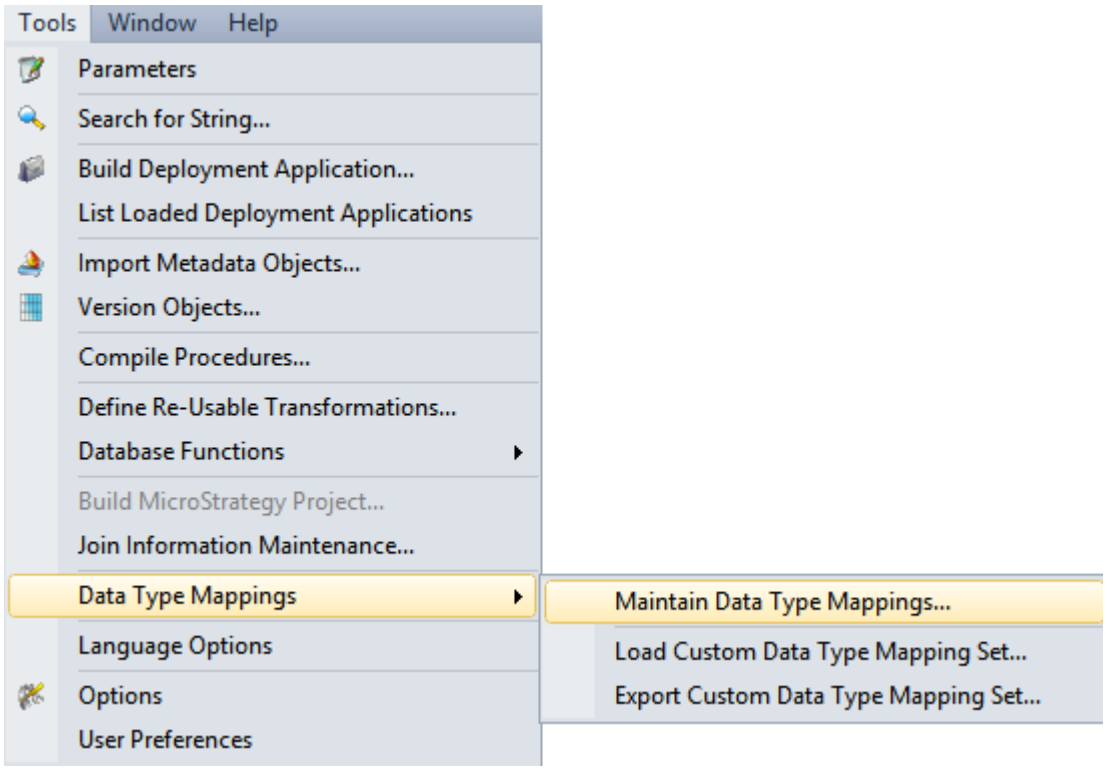
To move a mapping up or down in the list, select the mapping and then use the **Move Up** or **Move Down** button on the right side of the dialog.



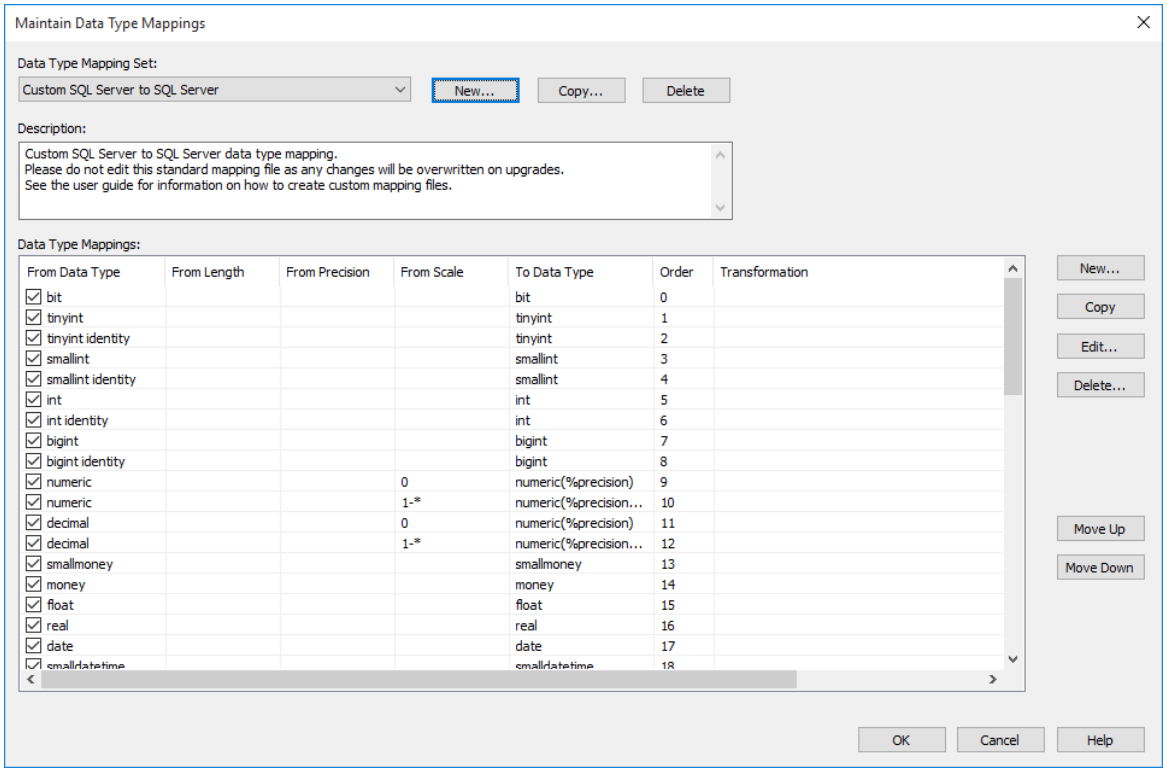
Note: The order of the data type mappings is significant as when loading a table, the procedure checks the data type mappings from top to bottom and stops when a data type and its parameters are correctly matched. A blank parameter means that it will match to anything.

EDITING A DATA TYPE MAPPING SET

To edit a data type mapping set, select **Tools/Data Type Mappings/Maintain Data Type Mappings...**



Select a data type mapping set from the **Data Type Mapping Set** drop-down list.

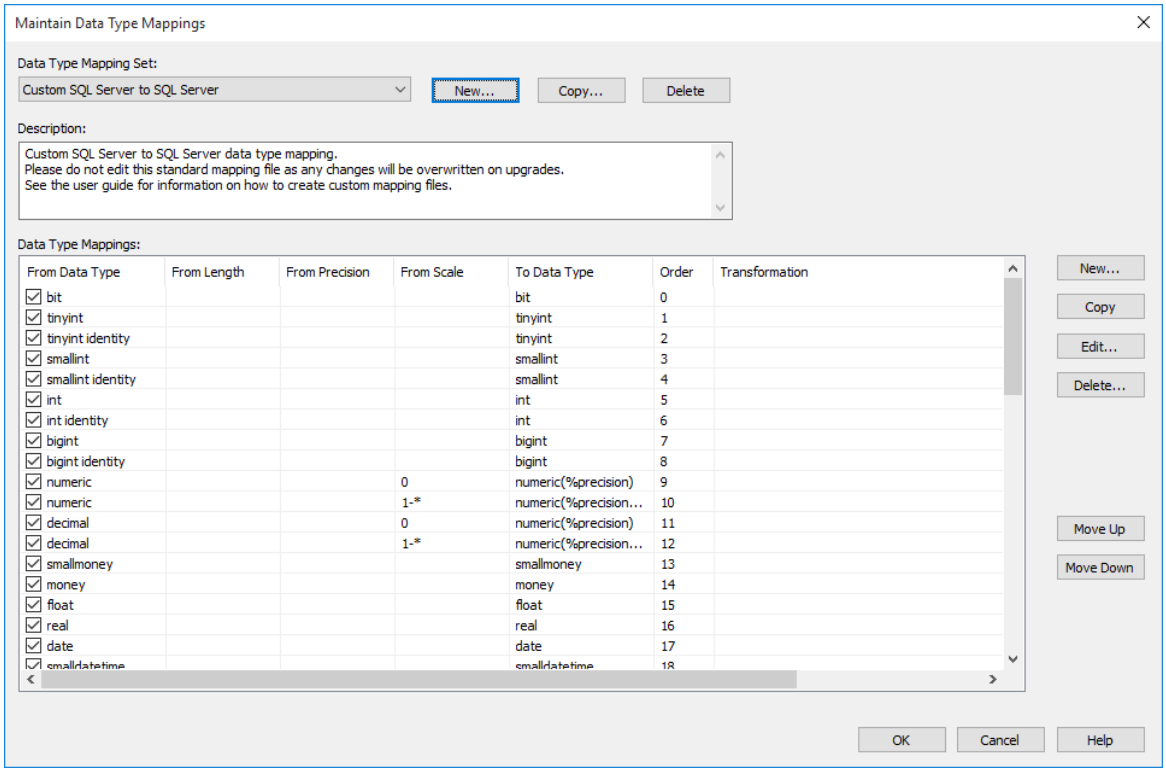


On the right is a group of buttons used to maintain the list of mappings in a data type mapping set. These buttons are not available for standard mapping sets. Only **user defined** mapping sets are editable.

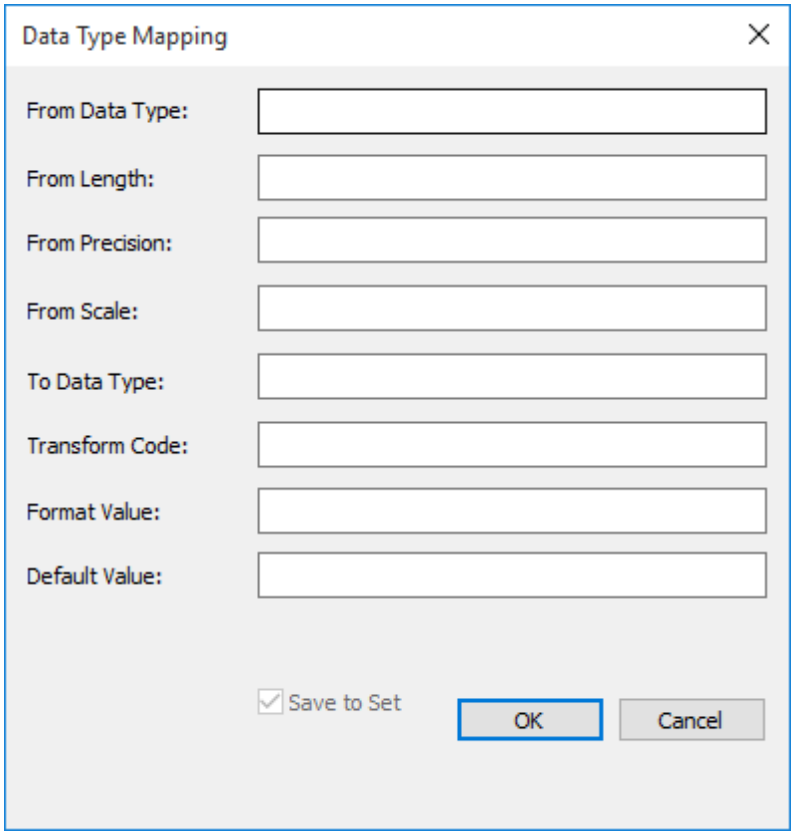
Note: The order of the mappings within a set is significant as when loading a table, the procedure checks the data type mappings from top to bottom and stops when a data type and its parameters are correctly matched. A blank parameter means that it will match to anything.

To add a new mapping to the data type mapping set:

Click on the **New** button on the right.



Enter the details and click **OK**.



From Data Type

The source Data Type to map to the Target Data Type.

From Length

The length parameter to match. (the number of characters or the number of bytes used to store the number)

From Precision

The precision parameter to match. (the number of digits in a number)

From Scale

The scale parameter to match. (the number of digits to the right of the decimal point in a number)

To Data Type

The data type, including parameters, which is the output of this mapping. Available tokens are %data_type, %length, %precision and %scale. See ***Data Type Mapping Examples*** (on page 1192)

Transform Code

The code to transform values into the mapped Data Type. Available tokens are %table_name, %column_name, %format, %data_type, %length, %precision and %scale. See ***Data Type Mapping Examples*** (on page 1192)

Format Value

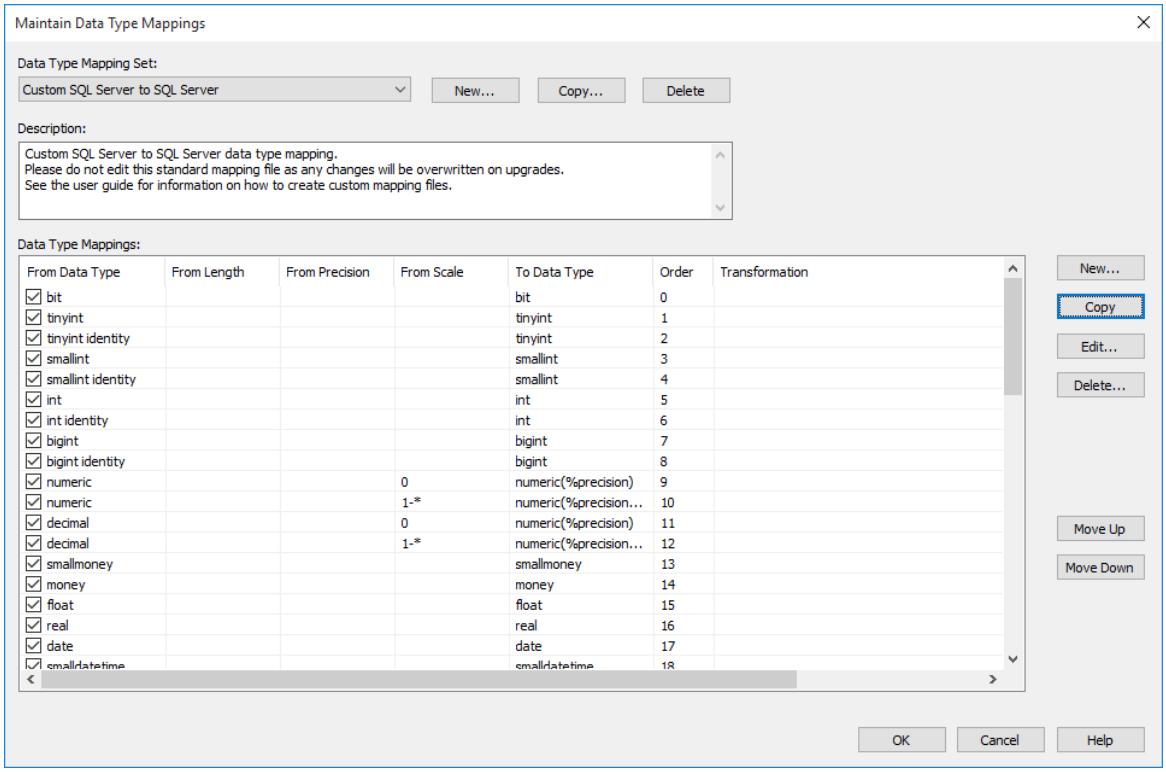
The format of this Conversion's Data Type output.

Default Value

The default value of this Conversion's Data Type output.

To copy an existing mapping in the data type mapping set:

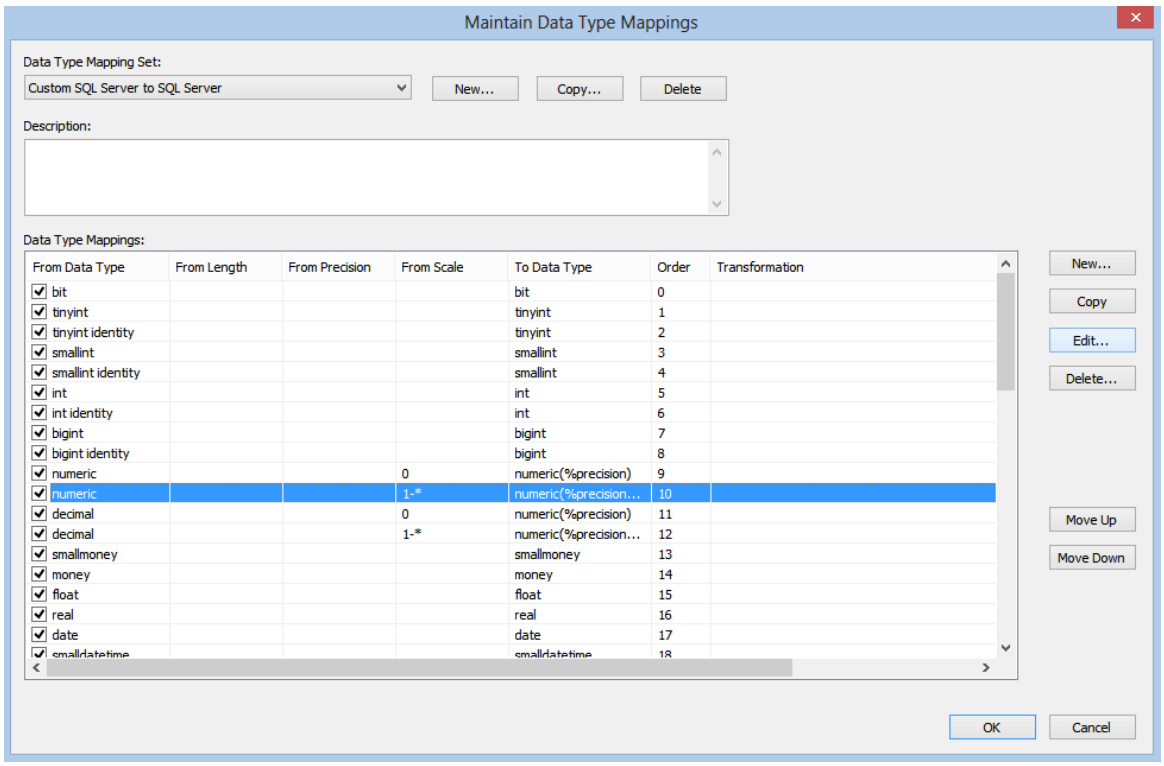
Select the mapping and then click on the **Copy** button on the right.


Maintain Data Type Mappings
Data Type Mapping Set:
Custom SQL Server to SQL Server
New...
Copy...
Delete
Description:
Custom SQL Server to SQL Server data type mapping.
Please do not edit this standard mapping file as any changes will be overwritten on upgrades.
See the user guide for information on how to create custom mapping files.
Data Type Mappings:
From Data Type | From Length | From Precision | From Scale | To Data Type | Order | Transformation
bit | bit | 0
tinyint | tinyint | 1
tinyint identity | tinyint | 2
smallint | smallint | 3
smallint identity | smallint | 4
int | int | 5
int identity | int | 6
bigint | bigint | 7
bigint identity | bigint | 8
numeric | 0 | numeric(%precision) | 9
numeric | 1-* | numeric(%precision... | 10
decimal | 0 | numeric(%precision) | 11
decimal | 1-* | numeric(%precision... | 12
smallmoney | smallmoney | 13
money | money | 14
float | float | 15
real | real | 16
date | date | 17
New...
Copy
Edit...
Delete...
Move Up
Move Down
OK
Cancel
Help

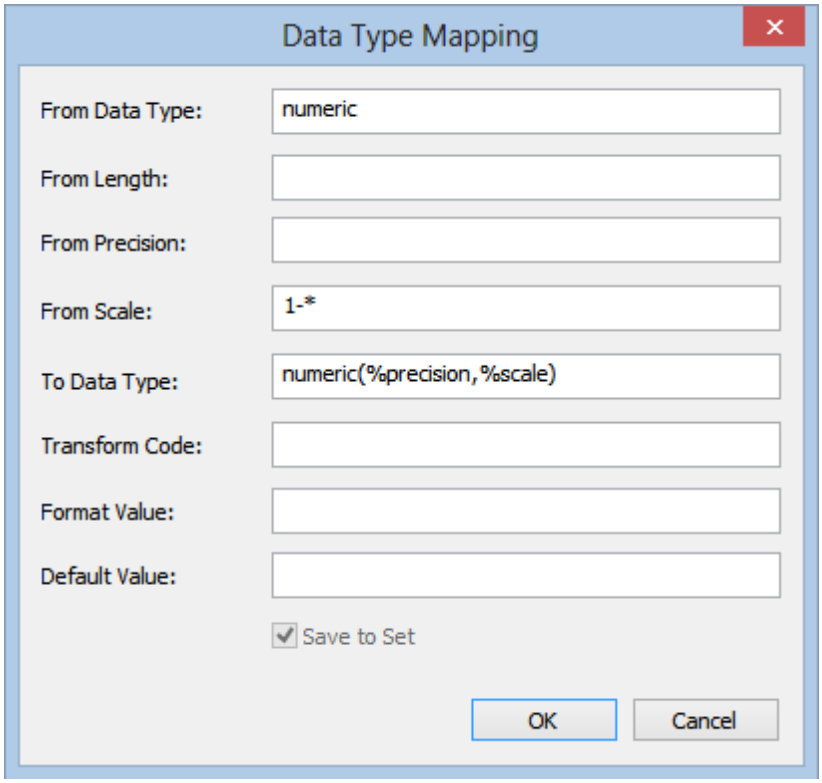
A copy of the mapping will be added to the bottom of the list. Select the mapping and click on the **Edit** button to change the details.

To edit an existing mapping in the data type mapping set:

Select the mapping you wish to edit and click the **Edit** button.

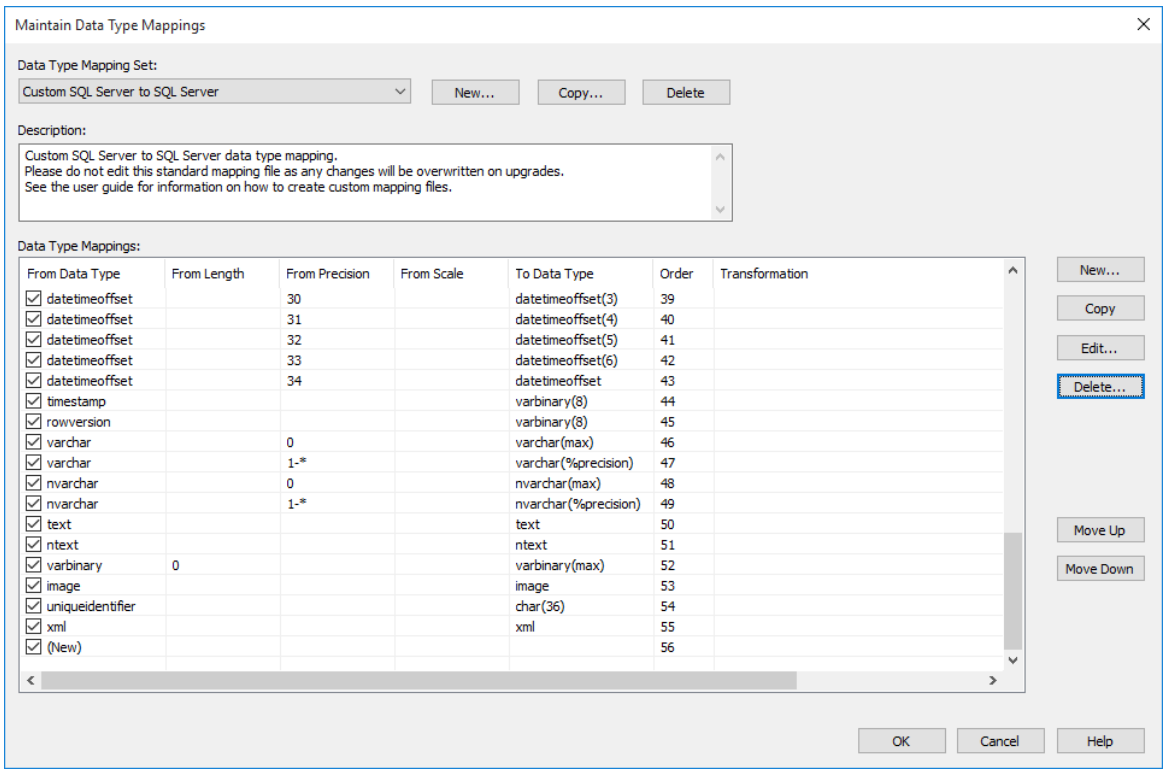


Change any fields as required and click **OK**.

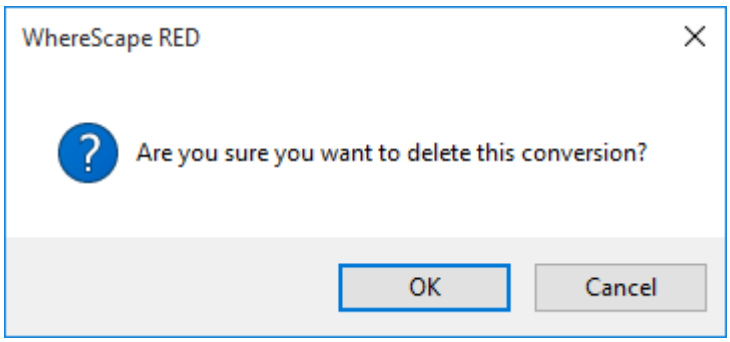


To delete an existing mapping in the data type mapping set:

Select the mapping and then click on the **Delete** button.

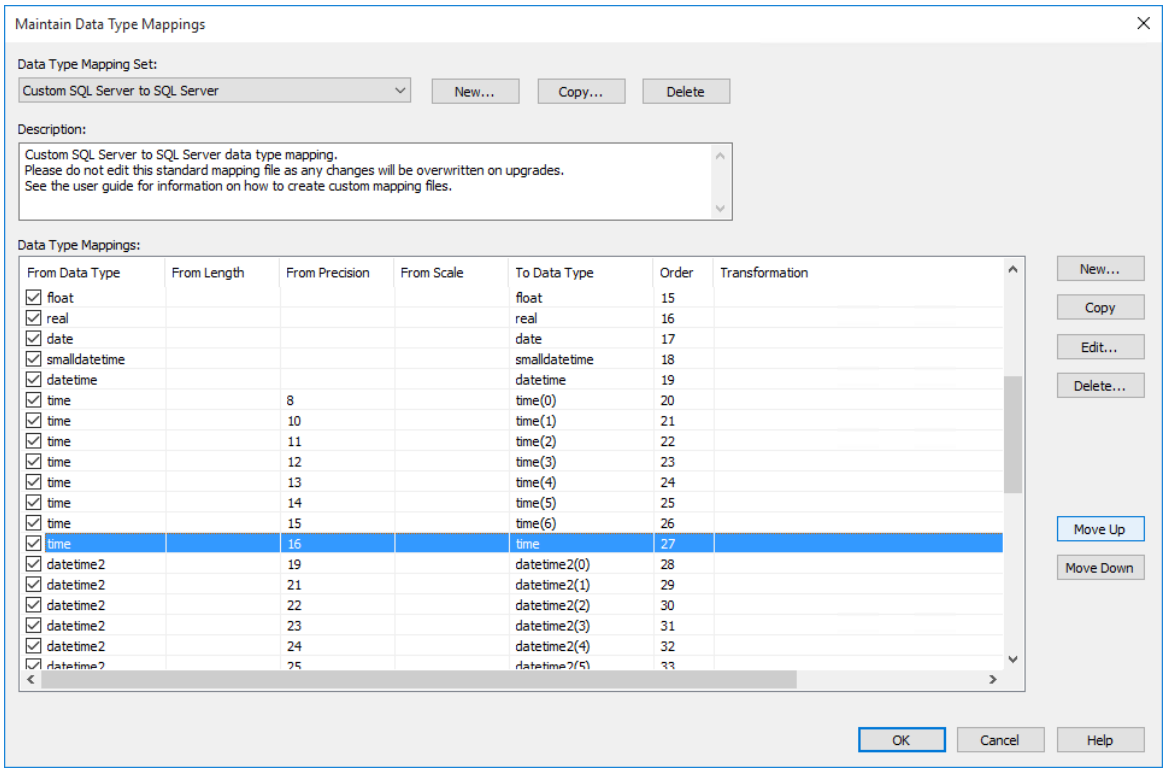


Click **OK** to delete.



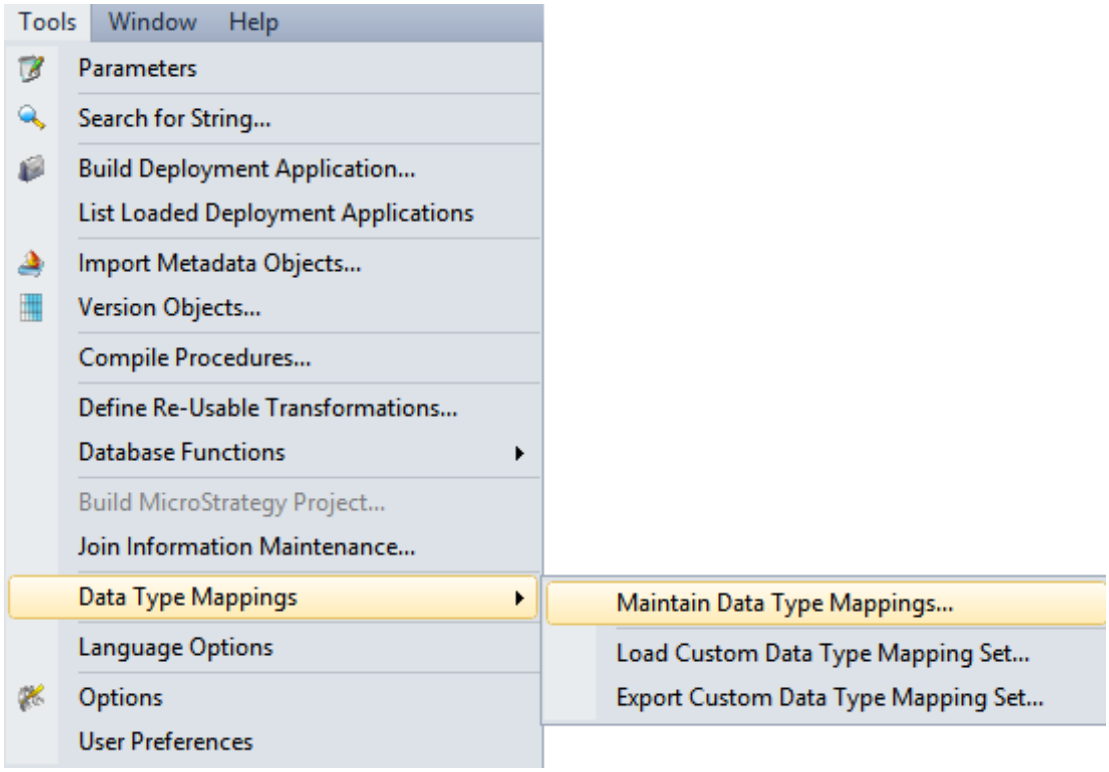
To move a mapping in the data type mapping set up or down in the list:

Select a mapping and then click on the **Move Up** button on the right to move the mapping up in the list; else click on the **Move Down** button on the right to move the mapping down in the list.

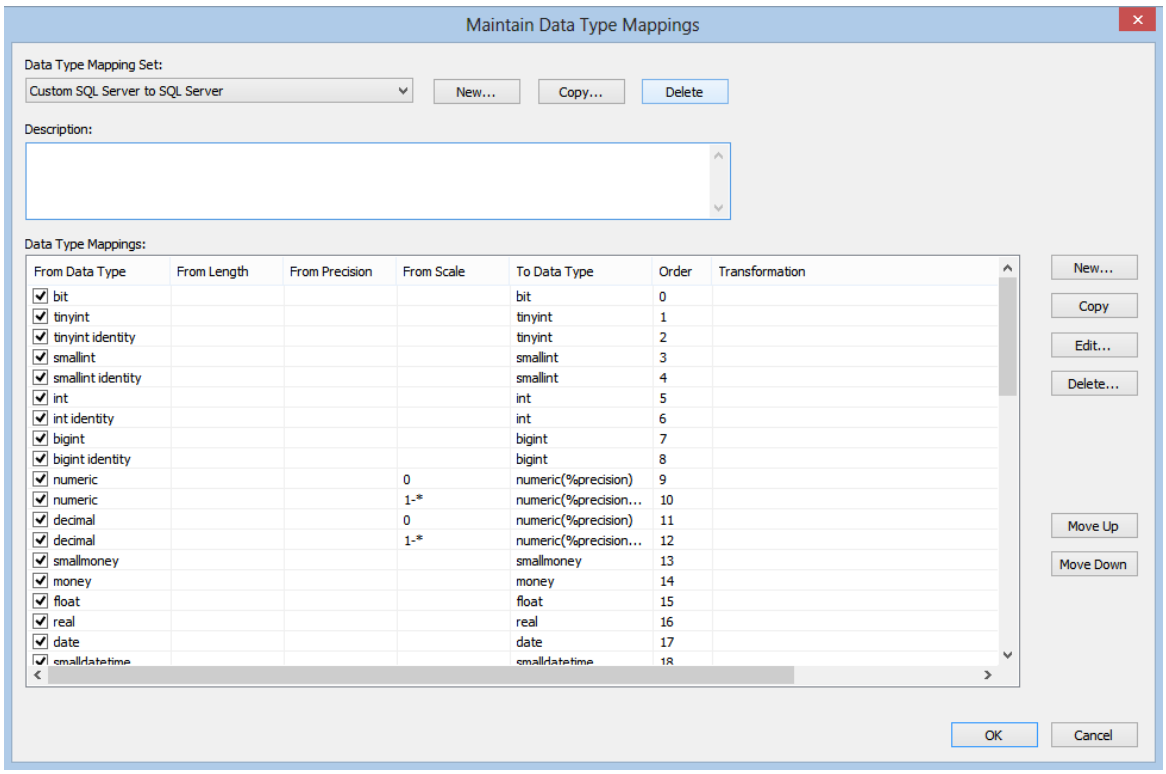


DELETING A DATA TYPE MAPPING SET

To delete a data type mapping set, select **Tools/Data Type Mappings/Maintain Data Type Mappings...**



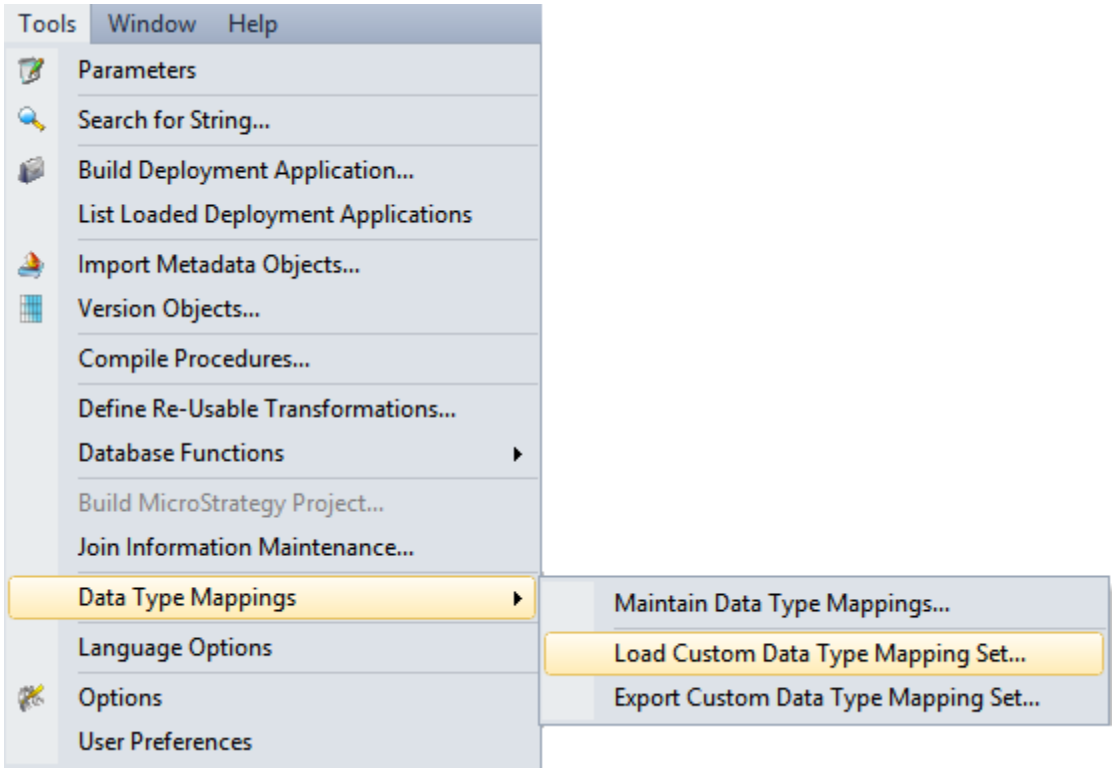
Select the Mapping Set to be deleted and click the **Delete** button.



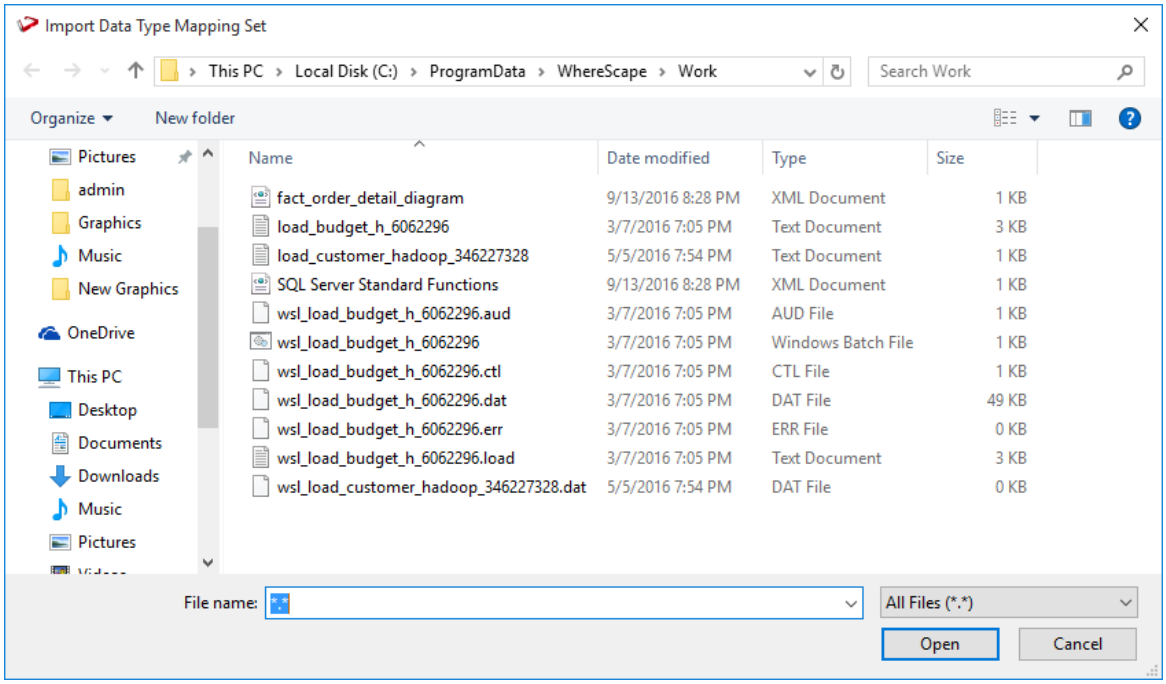
Note: The **Delete** button is disabled for standard data type mapping sets.

LOADING DATA TYPE MAPPING SETS

The **Load Custom Data Type Mapping Set** menu option allows you to load a custom data type mapping set from an XML file into the metadata repository.
To load a data type mapping set, select **Tools/Data Type Mappings/Load Custom Data Type Mapping Set...**

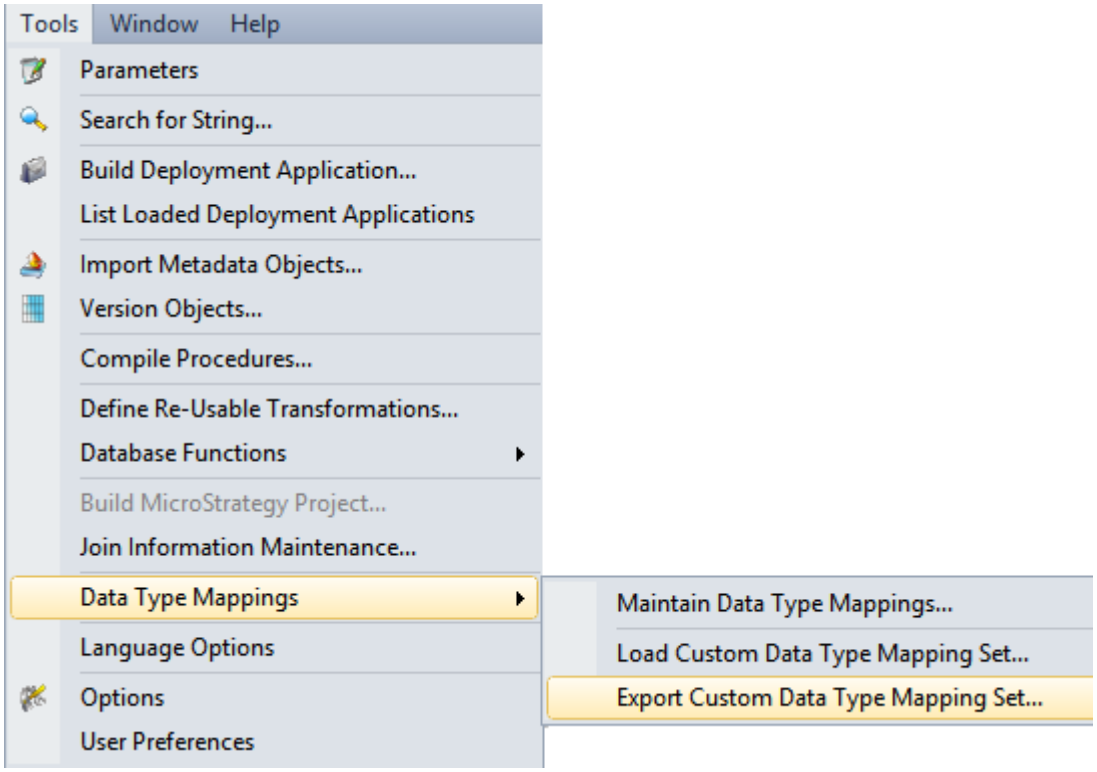


The following dialog is displayed. Select the xml file to load the data type mappings. By default RED expects the xml files to be in **ProgramData\WhereScape\Work.**

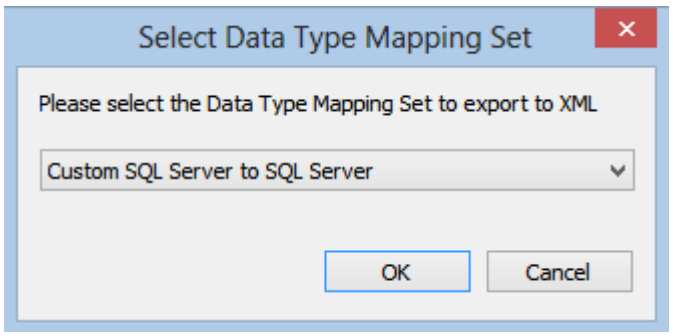


EXPORTING DATA TYPE MAPPING SETS

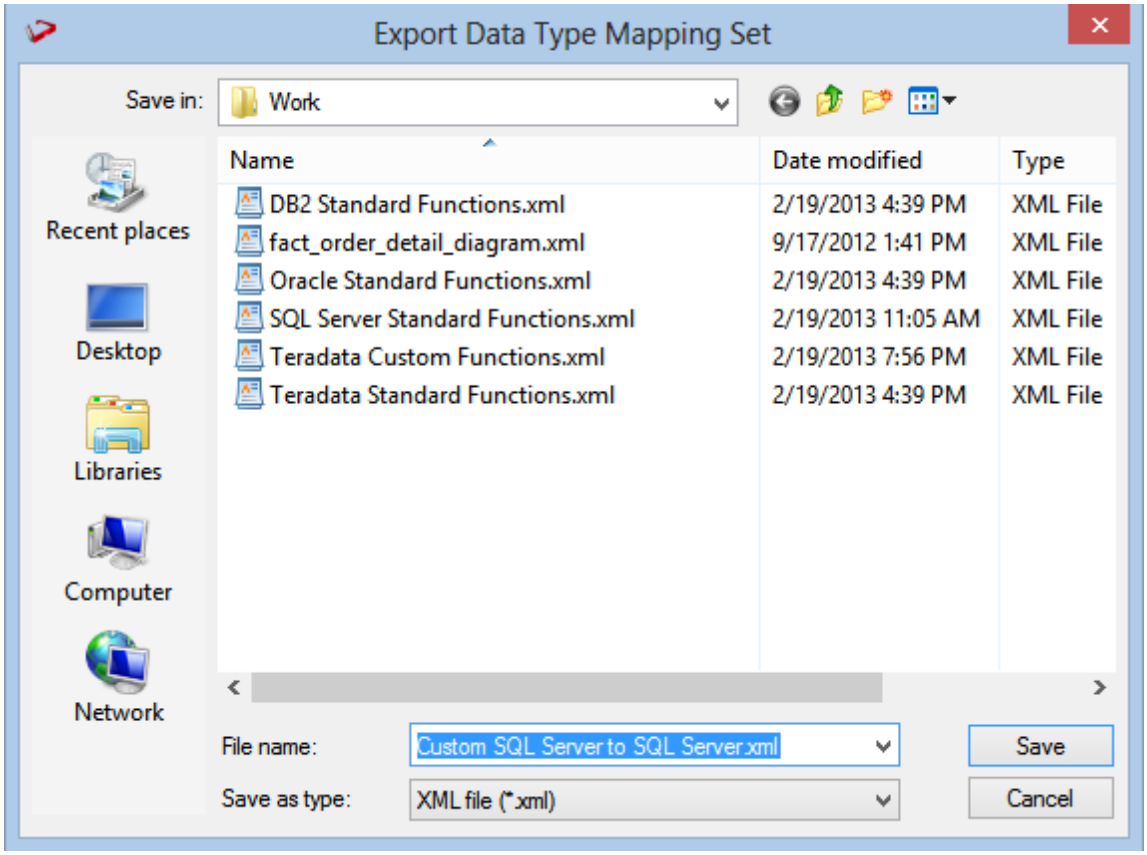
The **Export Custom Data Type Mapping Set** menu option allows you to export a custom data type mapping set from the metadata repository to an XML file.
To export a data type mapping set, select **Tools/Data Type Mappings/Export Custom Data Type Mapping Set...**



Select the data type mapping set to export from the drop-down list. Click **OK**.



By default, RED exports the xml file to **ProgramData\WhereScape\Work**, but this can be changed. Change the File name if necessary and click **Save**.



DATA TYPE MAPPING EXAMPLES

WhereScape RED allows you to create **Custom** Data Type Mapping Sets. These give you the ability to automatically change the data type of any column or to add column transformations when dragging and dropping new load tables.

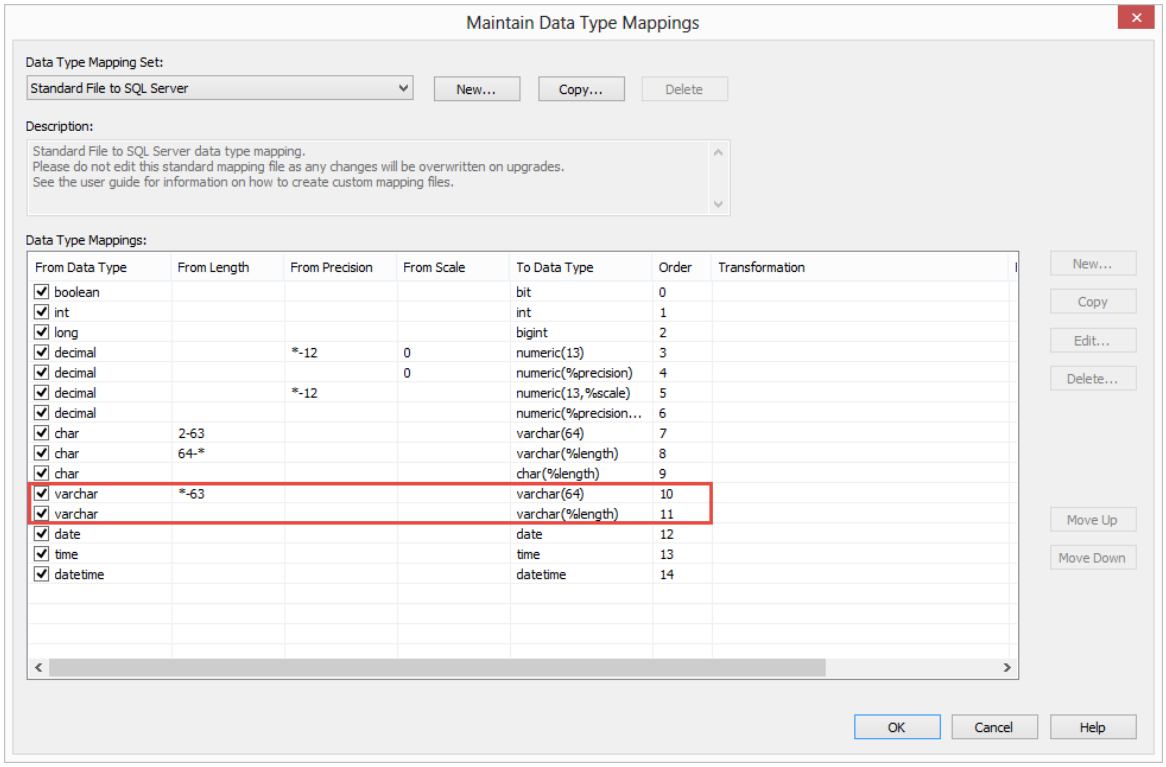
The examples in this topic demonstrate how **Custom** Data Type Mapping Sets can be configured using the following variables:

- %length
- %scale
- %precision
- %table_name
- %column_name
- %format
- %prompt

%length

In the example below, when converting a varchar in a standard file to SQL Server format, we follow the following steps in the given order:

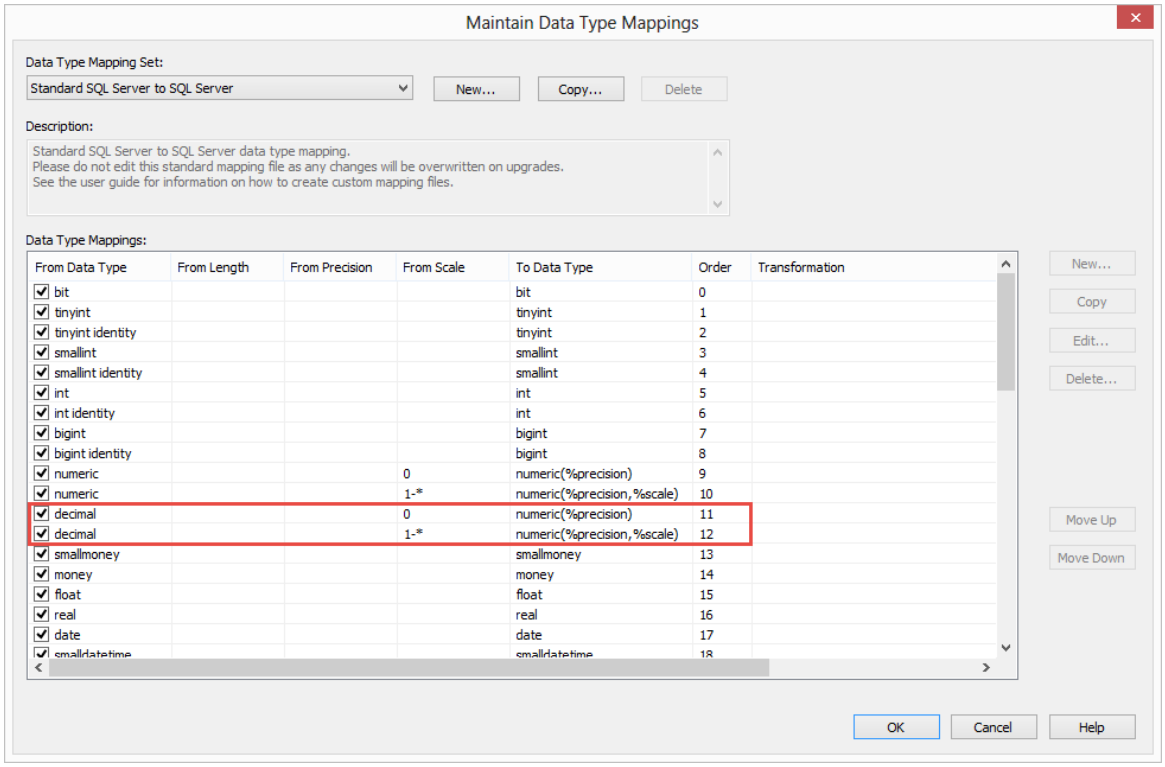
- If the varchar is of a length less than or equal to 63, the **data type** will become varchar(64).
- If the first step was NOT applied, i.e. the varchar is of a length greater than 63, then the **data type** will become varchar(%length); where we substitute the length for the variable '%length'. Thus if the varchar is of length 64 then the resulting data type will be varchar(64), but if the varchar is of length 123 then the resulting data type will be varchar(123).



%scale

In the example below, when converting a decimal in SQL Server to SQL Server format, we follow the following steps in the given order:

- If the decimal has a scale of zero, the **data type** will become numeric(%precision); where we substitute the number of digits in the number for the variable '%precision'. Thus if the decimal has 8 digits then the resulting data type will be numeric(8).
- If the first step was NOT applied, i.e. the decimal has a scale of 1 or greater, then the **data type** will become numeric(%precision,%scale); where we not only substitute the number of digits in the number for the variable '%precision', but we also substitute the scale for the variable '%scale'. Thus if the decimal is made up of 8 digits and has 3 digits after the decimal point (example 12345,678), the resulting data type will be numeric(8,3).

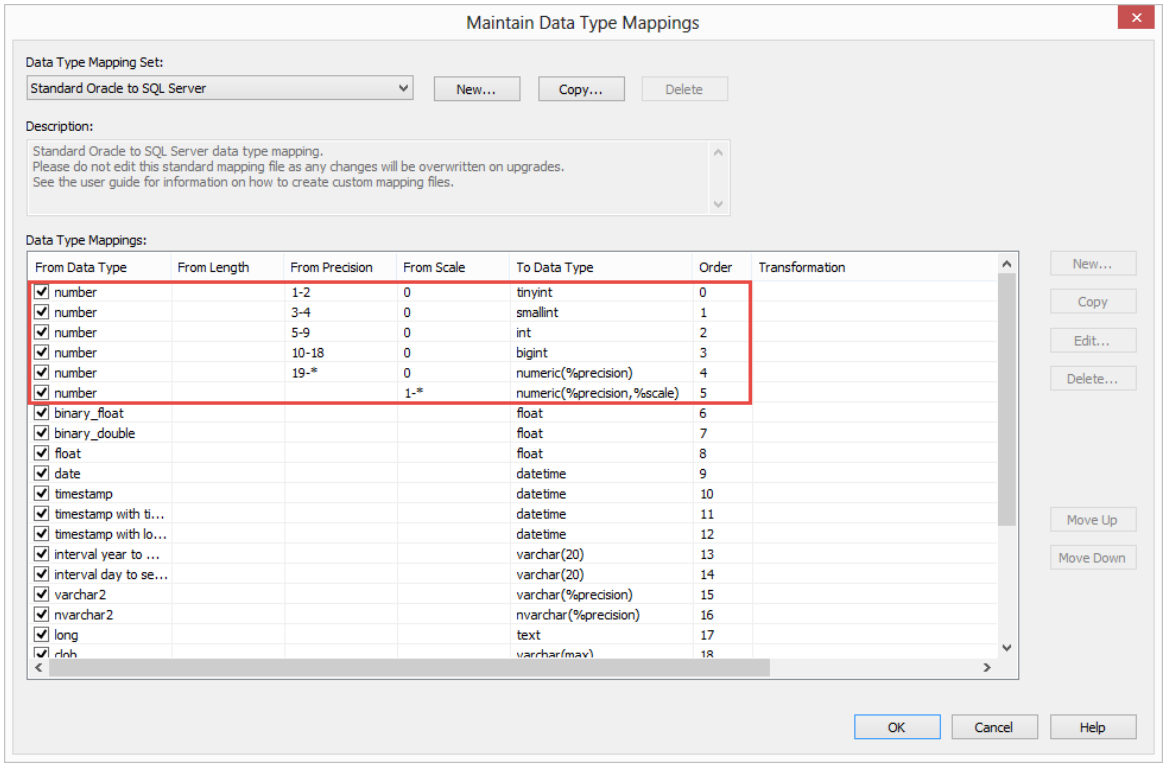


Note: The **Scale** is the number of digits to the right of the decimal point in a number.

%precision

In the example below, when converting a number in Oracle to SQL Server format, we follow the following steps in the given order:

- If the number is 1 or 2 digits long with a scale of zero, the **data type** will become tinyint.
- If the number is 3 or 4 digits long with a scale of zero, the **data type** will become smallint.
- If the number is 5-9 digits long with a scale of zero, the **data type** will become int.
- If the number is 10-18 digits long with a scale of zero, the **data type** will become bigint.
- If the number is 19 or greater digits long with a scale of zero, the **data type** will become numeric(%precision); where we substitute the number of digits in the number for the variable '%precision'. Thus if the number is made up of 22 digits, the resulting data type will be numeric(22).
- If however, the scale is not zero and there are digits to the right of the decimal point in the number, then the **data type** will become numeric(%precision,%scale); where we substitute both the number of digits in the number for the variable '%precision' and the number of digits to the right of the decimal point for the variable '%scale'. Thus if the number is made up of 8 digits and has 3 digits after the decimal point (example 12345,678), the resulting data type will be numeric(8,3).

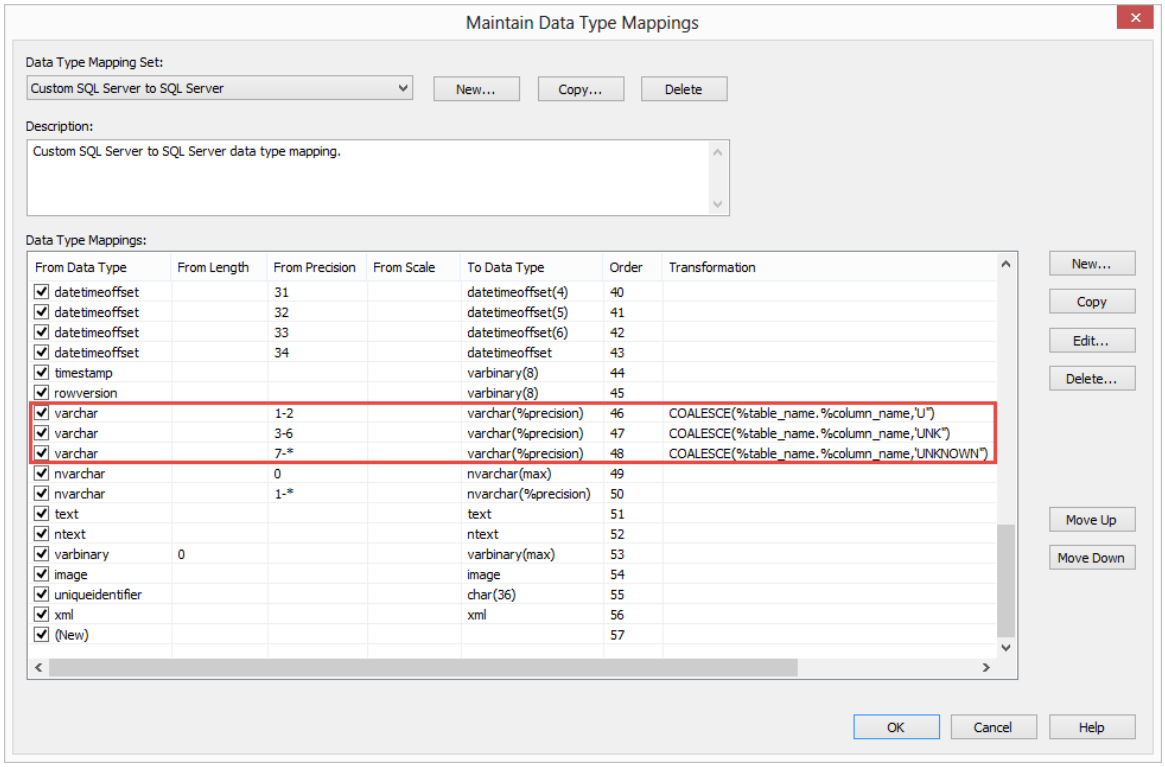


Note: The **Precision** is the total number of digits in a number.

%table_name and %column_name

In the example below, we use the following transformations to handle NULL for different lengths of varchars:

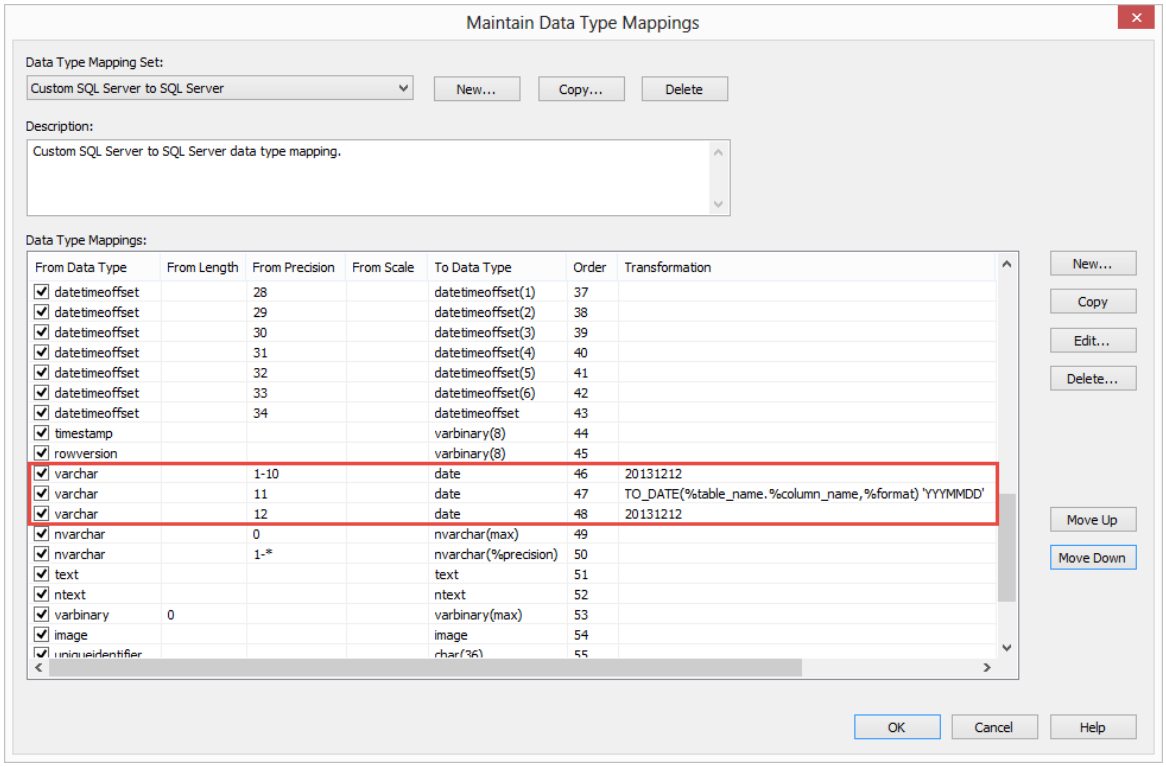
- If the varchar is 1 or 2 digits/chars long, the **data type** will become varchar(%precision); where we substitute the number of digits/chars in the varchar for the variable '%precision'. Secondly, the **value** of the column will become the column value (if it is not null), else it will become 'U'.
- If the varchar is 3-6 digits/chars long, the **data type** will become varchar(%precision); where we substitute the number of digits/chars in the varchar for the variable '%precision'. Secondly, the **value** of the column will become the column value (if it is not null), else it will become 'UNK'.
- If the varchar is 7 or more digits/chars long, the **data type** will become varchar(%precision); where we substitute the number of digits/chars in the varchar for the variable '%precision'. Secondly, the **value** of the column will become the column value (if it is not null), else it will become 'UNKNOWN'.



%format

In the example below, we use the following transformations to convert a certain character field to a date:

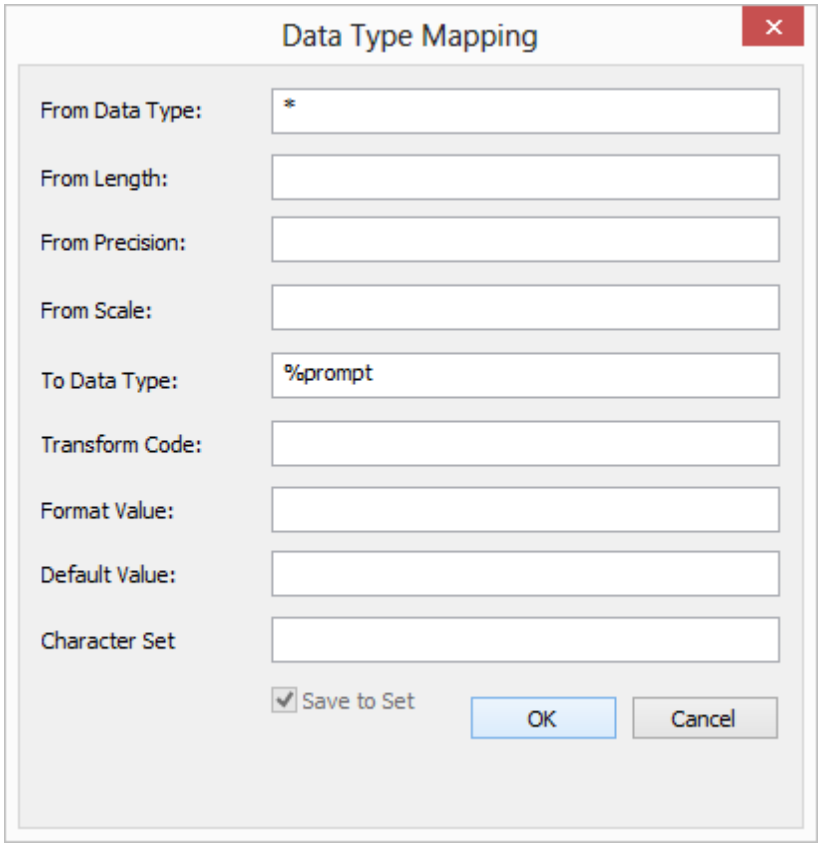
- If the varchar has a length of 1-10, the **data type** will become date and the **value** of the column will become the date 20131212 (a chosen date in the future).
- If the varchar has a length of 11, the **data type** will become date and the **value** of the column will use the transformation TO_DATE(%table_name.%column_name,%format); where we substitute 'YYYYMMDD' for the variable '%format'. Thus, the value of the column will be converted to a date of format 'YYYYMMDD'.
- If the varchar has a length of 12 or greater, the **data type** will become date and the **value** of the column will become the date 20131212 (a chosen date in the future).



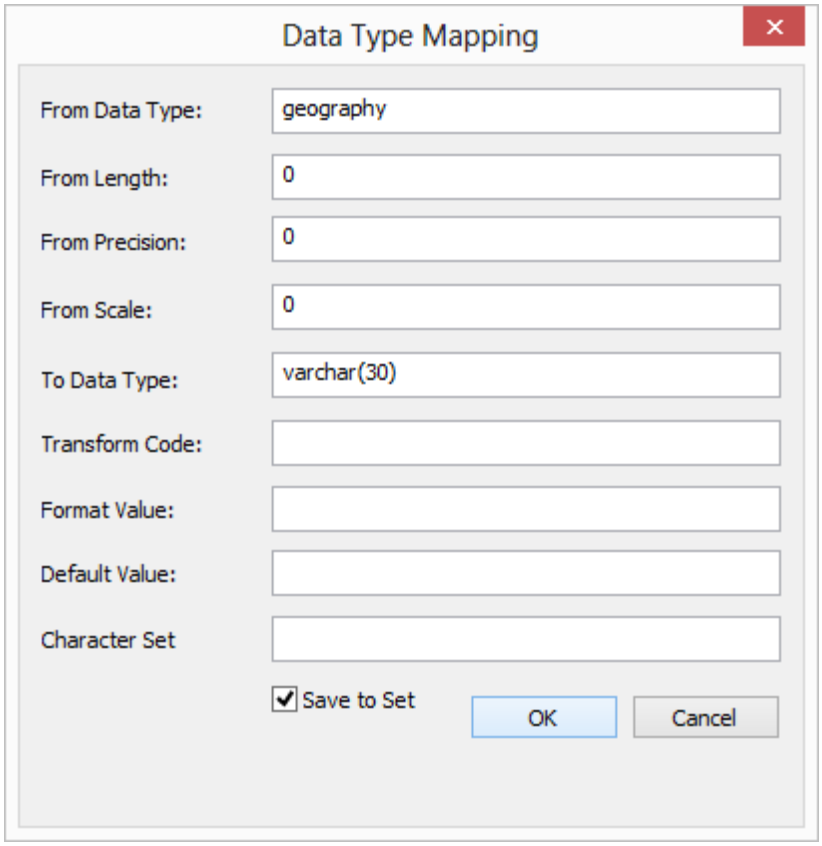
%prompt

In the example below we use %prompt to help the user to define a mapping for an unknown datatype that is not already mapped in the previous mapping rules.
This variable must be used with a **custom** Data Type mapping set, as described in the following steps:

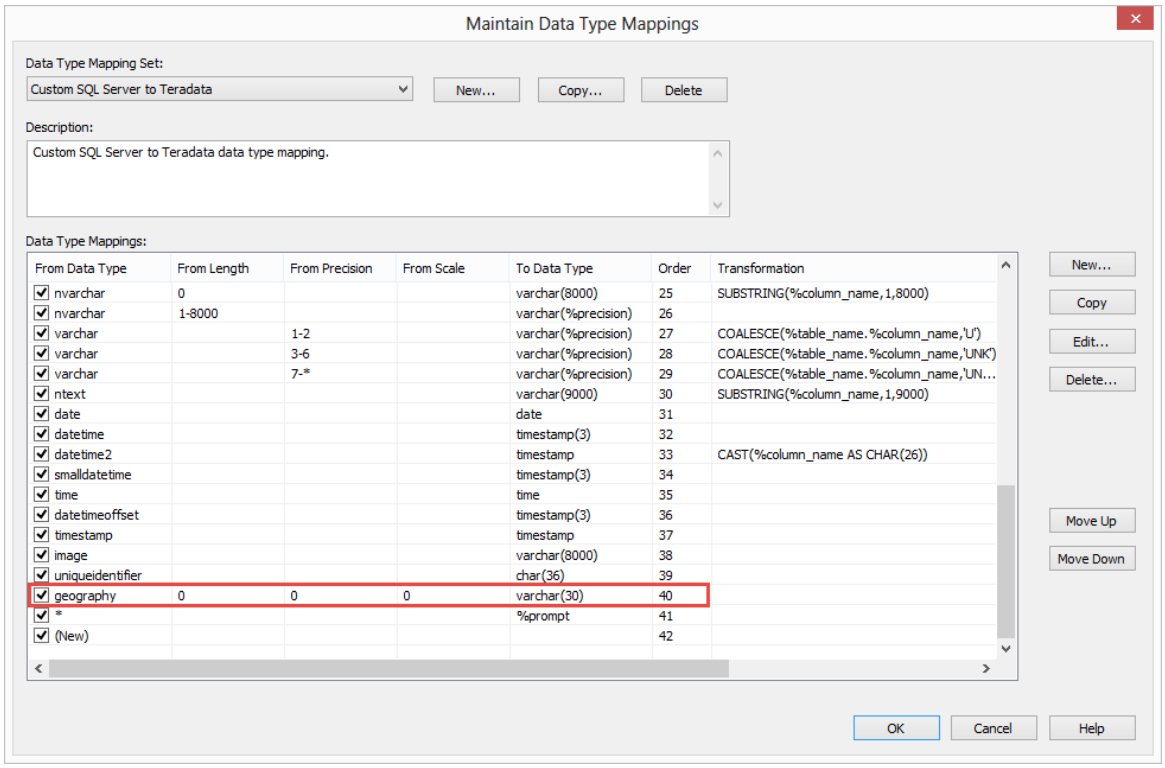
- Create a new custom set or copy from an existing set.
- Create a new Data Type mapping with a **From Data Type** of star (*) and a **To Data Type** of **%prompt**. Click OK to save the New Data Type Mapping to the Custom set.


Data Type Mapping
From Data Type:
*
From Length:
From Precision:
From Scale:
To Data Type:
%prompt
Transform Code:
Format Value:
Default Value:
Character Set
Save to Set
OK
Cancel

- When browsing a connection to load a table, set the **Data Type Mapping Set** to the new **Custom SQL Server to Teradata set**. This can be set on the List Sources Tables Connection Dialog or on the Connection Screen.
- As the table is dragged and dropped to the middle pane, RED will prompt to have the new datatype mapping defined.
- In the example below, just before loading the table, users can map the unknown **geography** SQL Server datatype mapping to a **varchar(30)** in Teradata.



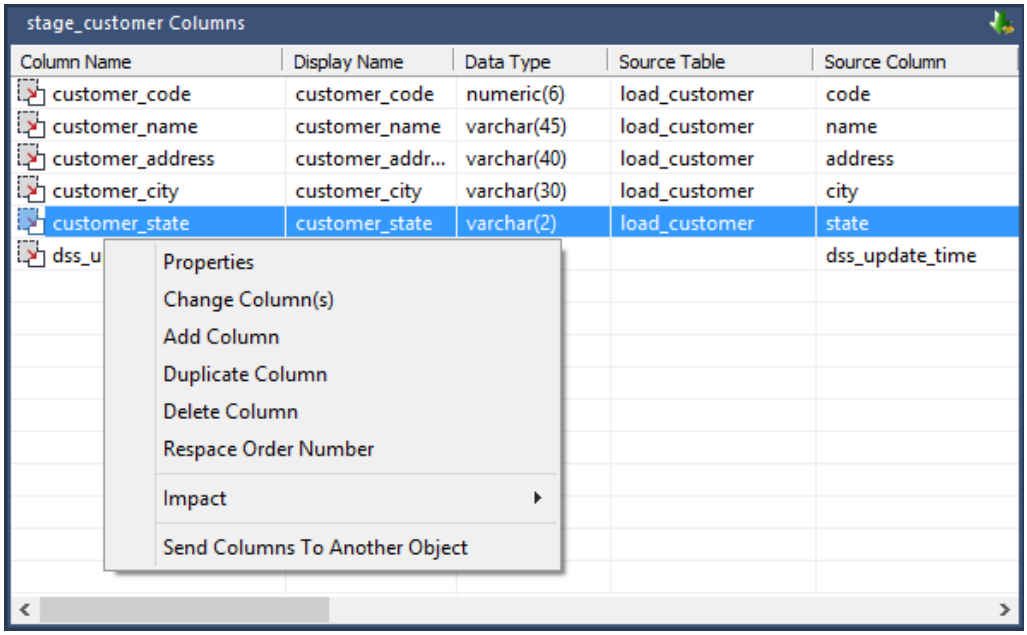
- Clicking the **Save to Set** check-box in the Data Type mapping screen above will save the mapping to the custom set that was used for loading the table.



WhereScape®

CHAPTER 39
COLUMN CONTEXT MENU

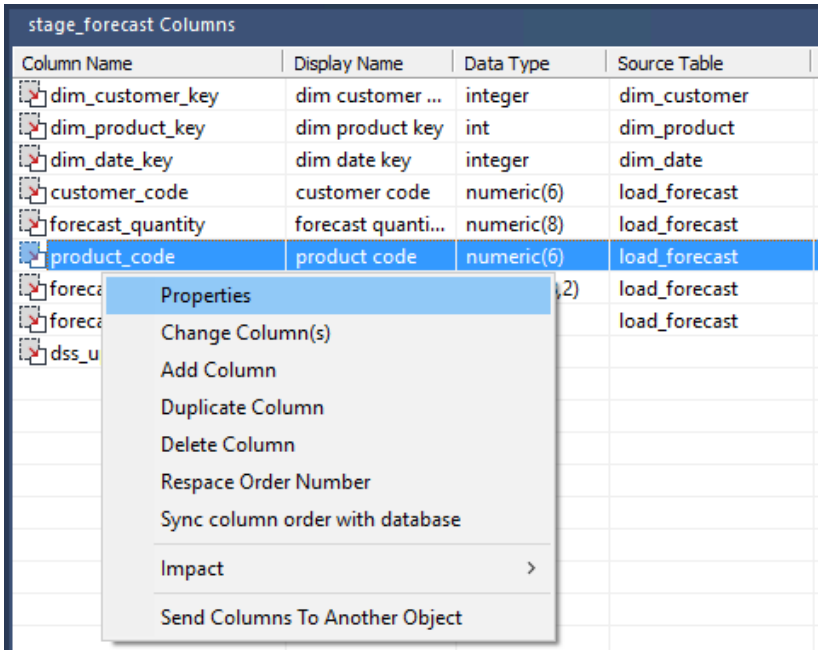
To view the column context menu, click on an object in the left pane to display the columns in the middle pane. When positioned on a column in the middle pane, right-click on the column to bring up the menu.



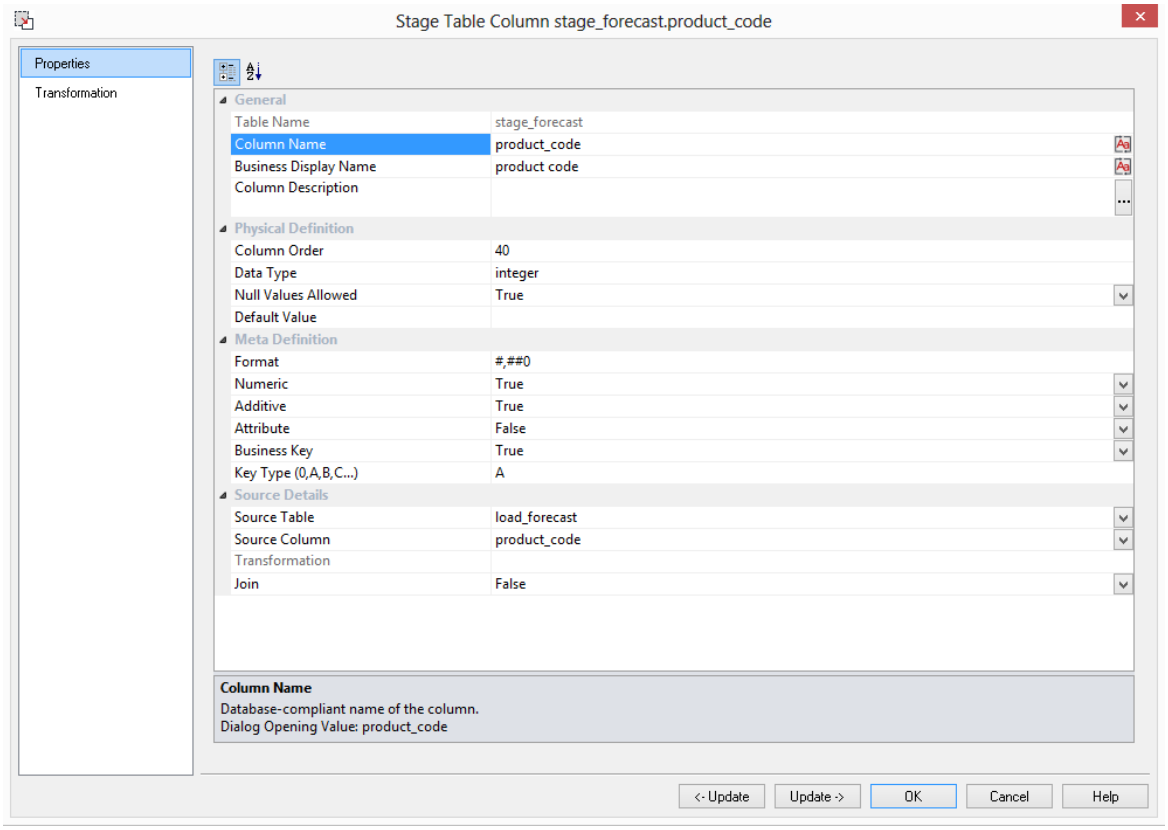
IN THIS CHAPTER

PROPERTIES

To display the column Properties, right-click on a column in the middle pane and select **Properties**.



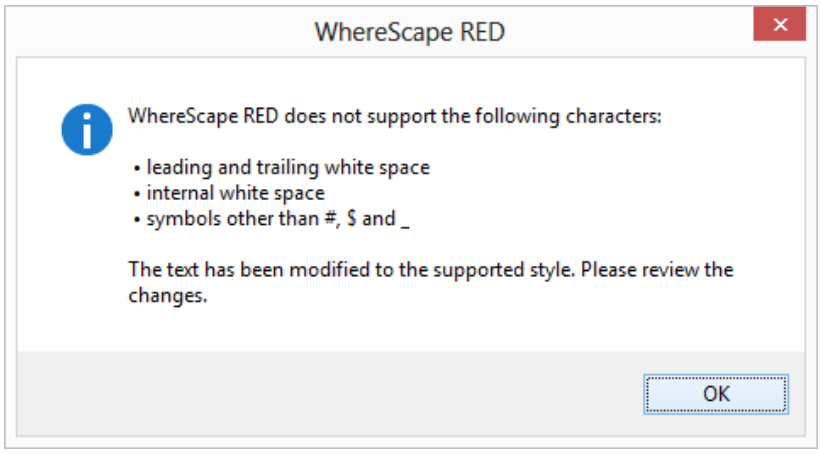
Edit any field as required and then click **OK** to close.



Warning: WhereScape RED does not support the following characters in **Column Names**:

- leading and trailing white spaces
- internal white spaces
- symbols other than #, $ and _

If users attempt to enter any of the above characters in Column Names, the following dialog will be displayed, advising users to review changes made by RED to correct any unsupported column name characters:



Note: There is a variation in column Properties, depending on the object type.

For Dimension tables, see ***Dimension Column Properties*** (on page 353).

For Stage tables, see ***Stage Table Column Properties*** (on page 388).

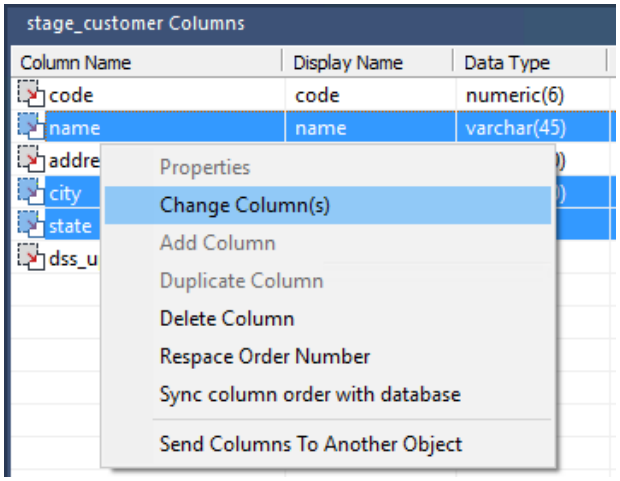
For Data Store tables, see ***Data Store Column Properties*** (on page 420).

For EDW 3NF tables, see ***EDW 3NF Table Column Properties*** (on page 444).

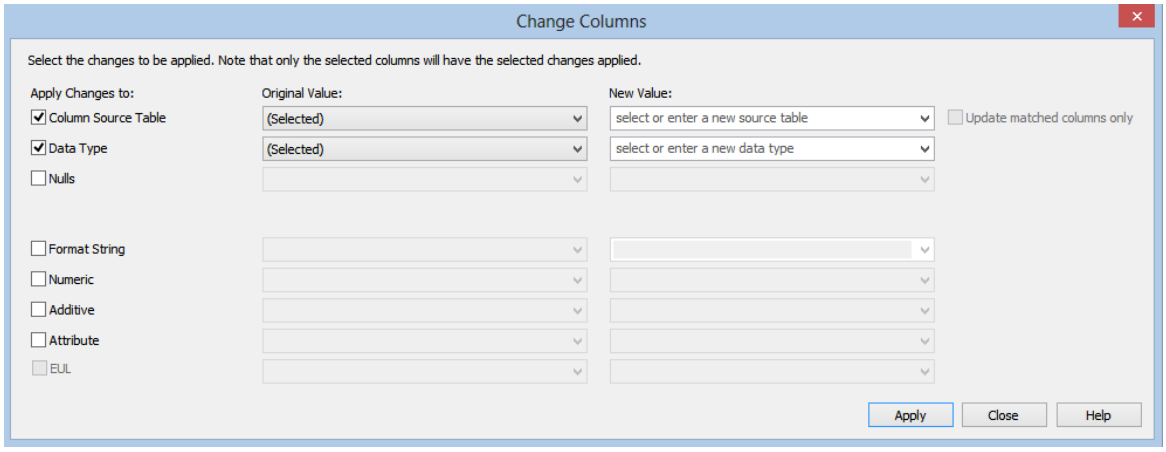
For Fact tables, see ***Fact Table Column Properties*** (on page 516).

CHANGE COLUMN(S)

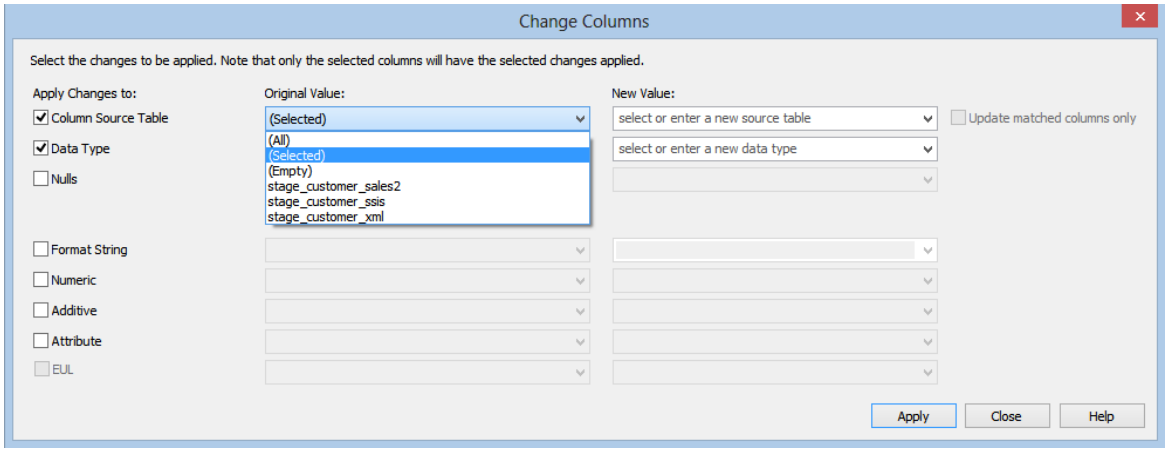
To change the properties for multiple columns, right-click on a column in the middle pane and select **Change Column(s)**.



To change a column property, you first need to select the relevant **check-boxes** on the left. Each check-box, when selected, allows you to change the value for that field in the column properties.



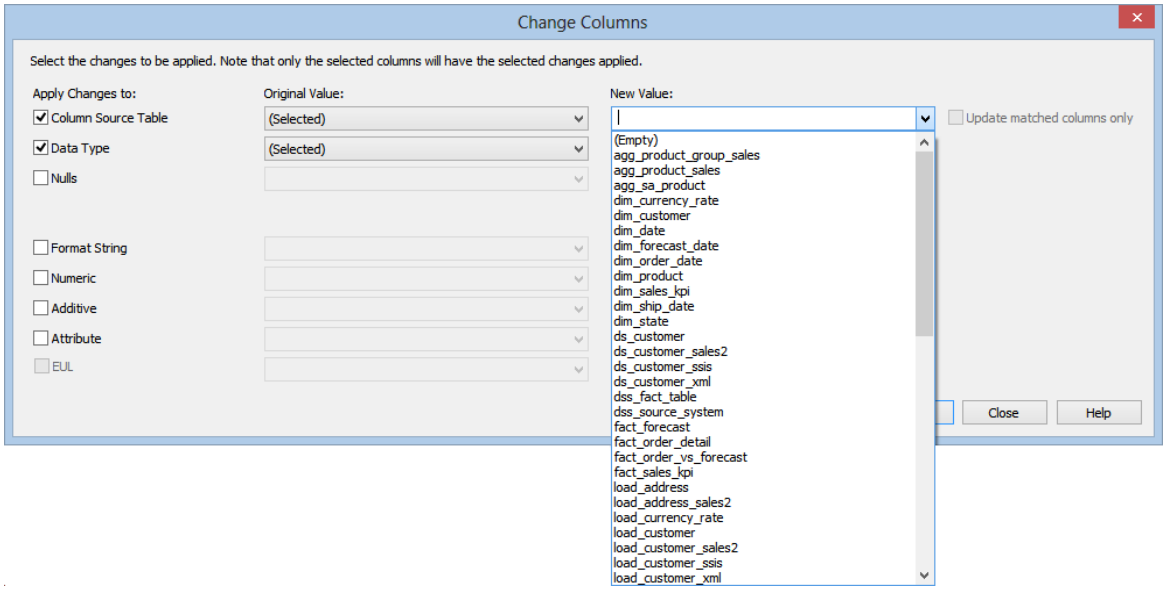
In the **Original Value** column, select the value/s to be changed.



- Choosing **(All)** will change the selected property for all of the columns in the table.
- Choosing **(Selected)** will change the selected property for the selected column in the table.
- Choosing **(Empty)** will change the selected property for all of the columns where that property field is empty. This option is only available if there is a column where this property is empty.
- Choosing one of the **other** options will change the selected property for all the columns in the table having that value.

Note: **(Selected)** is the default for the **Original Value** column.

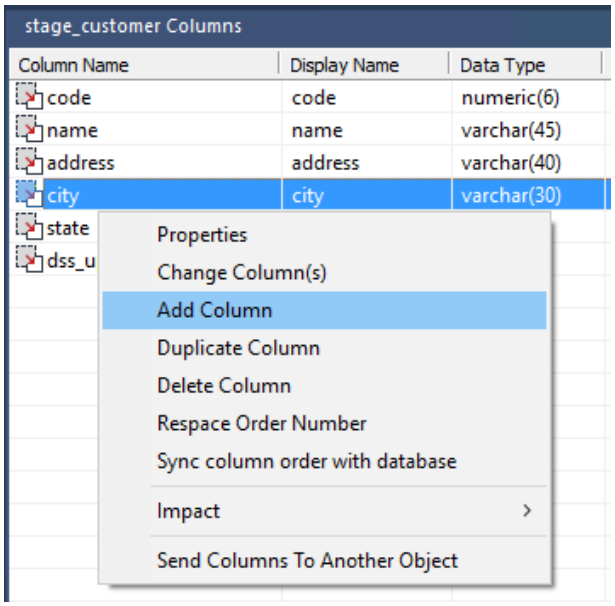
In the **New Value** column, select the new value to be assigned; or key in the new value.



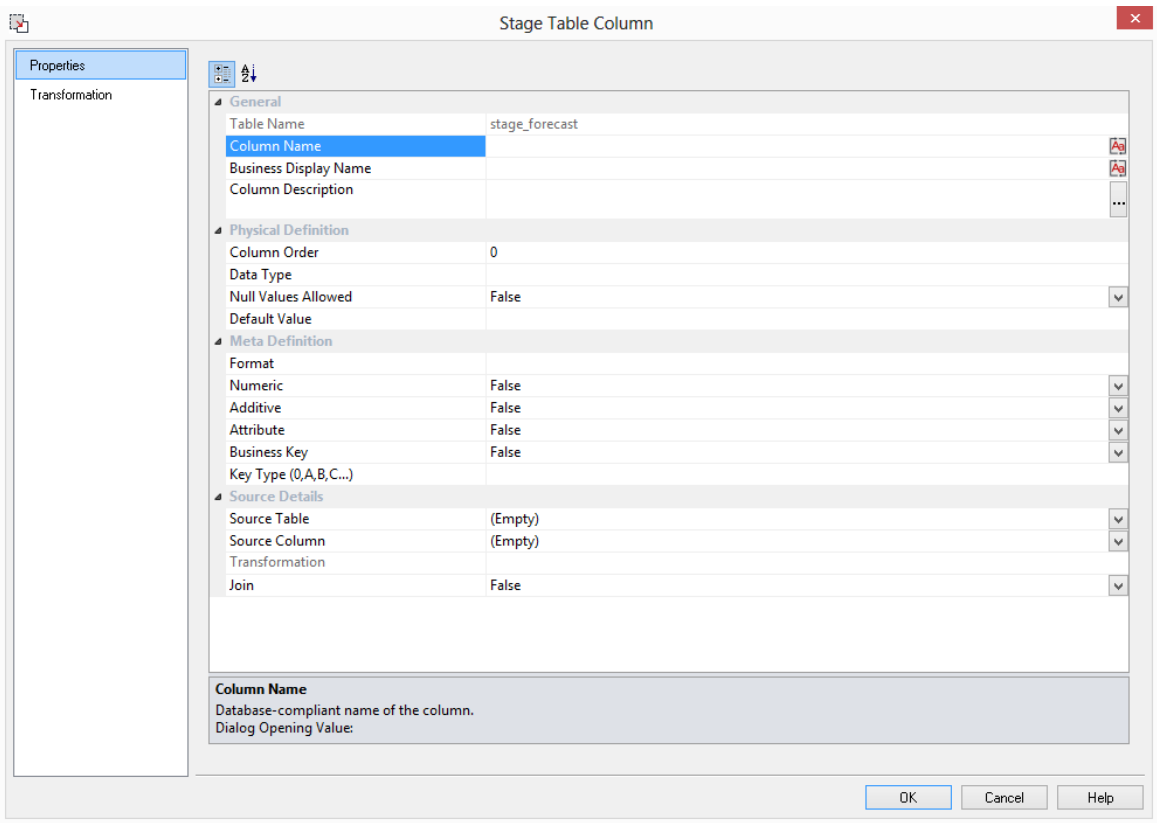
Note: When editing the property **Column Source Table**, selecting the **Update matched columns only** check-box will validate that the selected column name exists in the new source table. If it does not exist, then the update will not take place.

ADD COLUMN

To add a column, right-click on one of the columns in the middle pane and select **Add Column**.

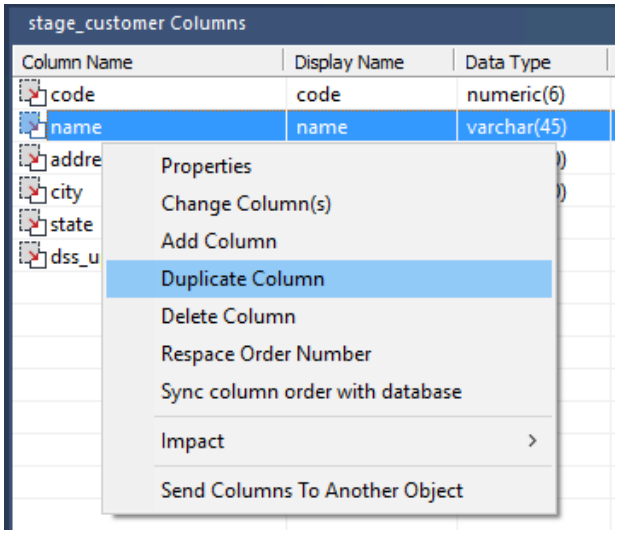


Enter the details to define a new column and click **OK**.

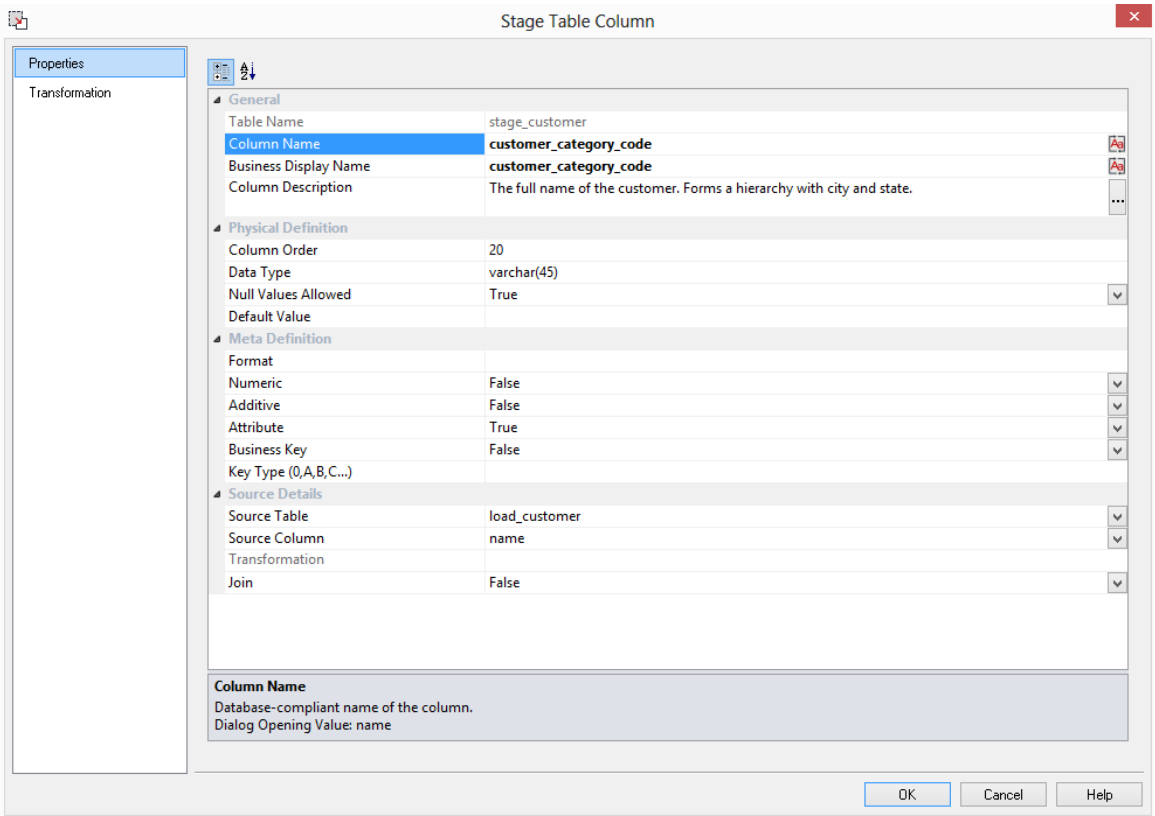


DUPLICATE COLUMN

To copy a column, right-click on one of the columns in the middle pane and select **Duplicate Column**.

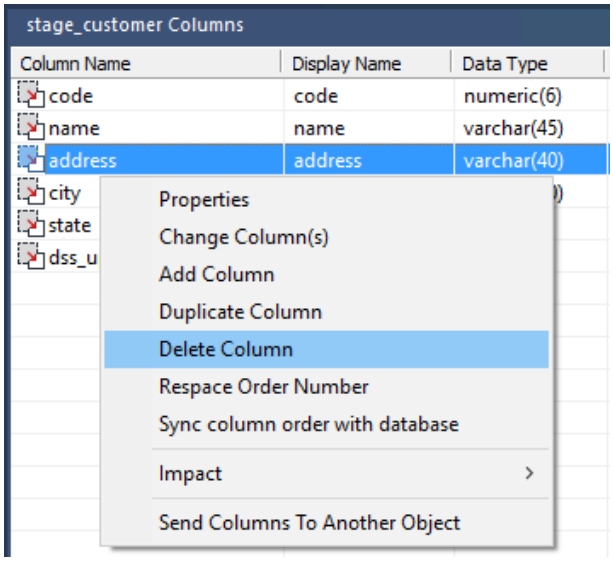


Change the **Column Name** and the **Business Display Name** and any other properties to define a new column and click **OK**.

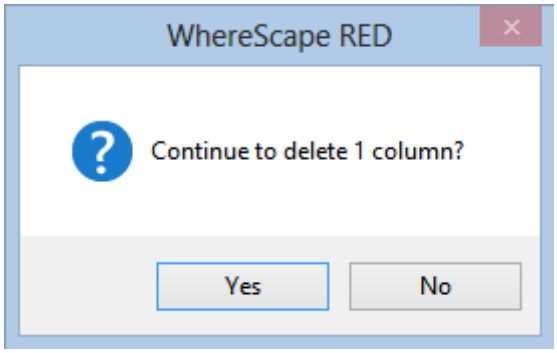


DELETE COLUMN

To delete a column, right-click on one of the columns in the middle pane and select **Delete Column**.

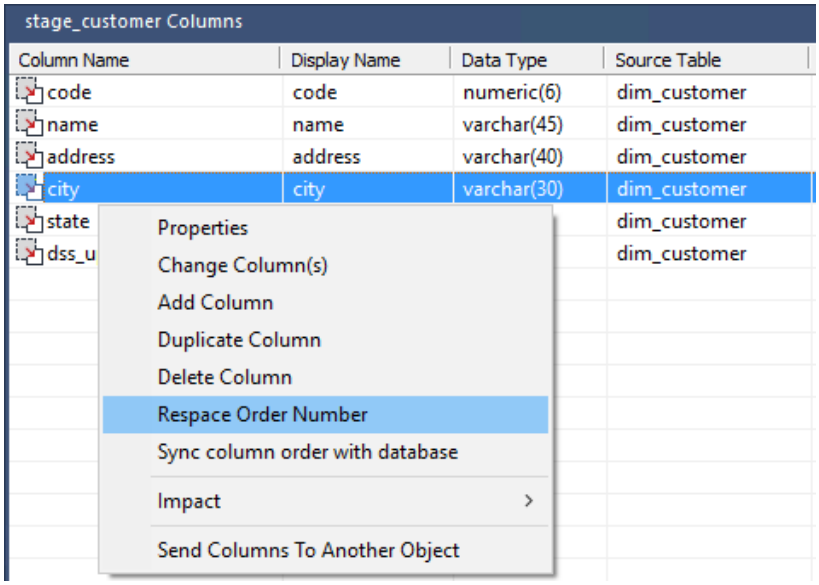


Click **Yes** to continue with the delete.

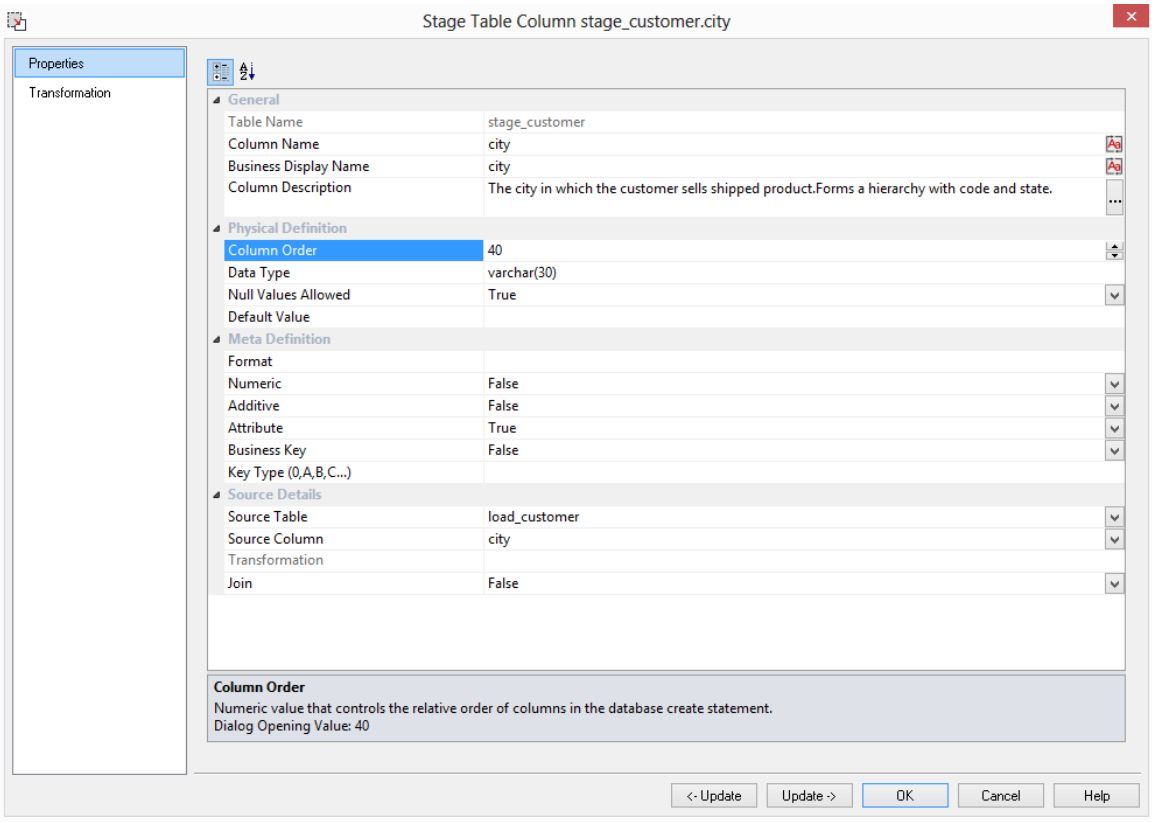


RE-SPACE ORDER NUMBER

To re-space the column order, right-click on any column in the middle pane and select **Respace Order Number.**



The **Column Order** number for each column will be adjusted so that the column order numbers are evenly spaced.

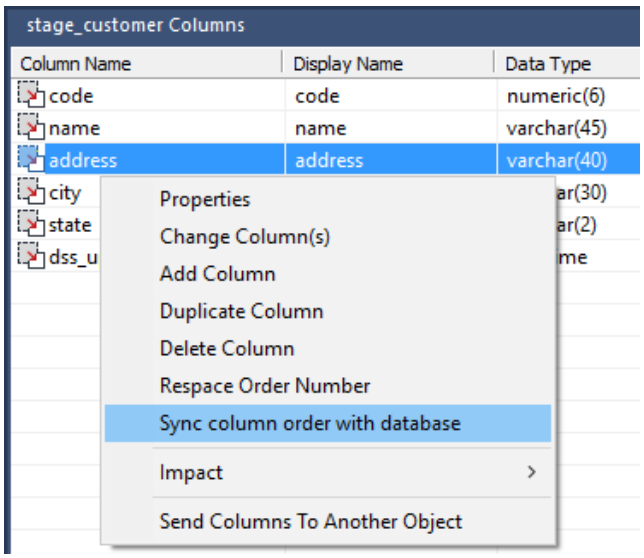


SYNC COLUMN ORDER WITH DATABASE

To synchronize the metadata's column order to match the same order in the physical table in the database:

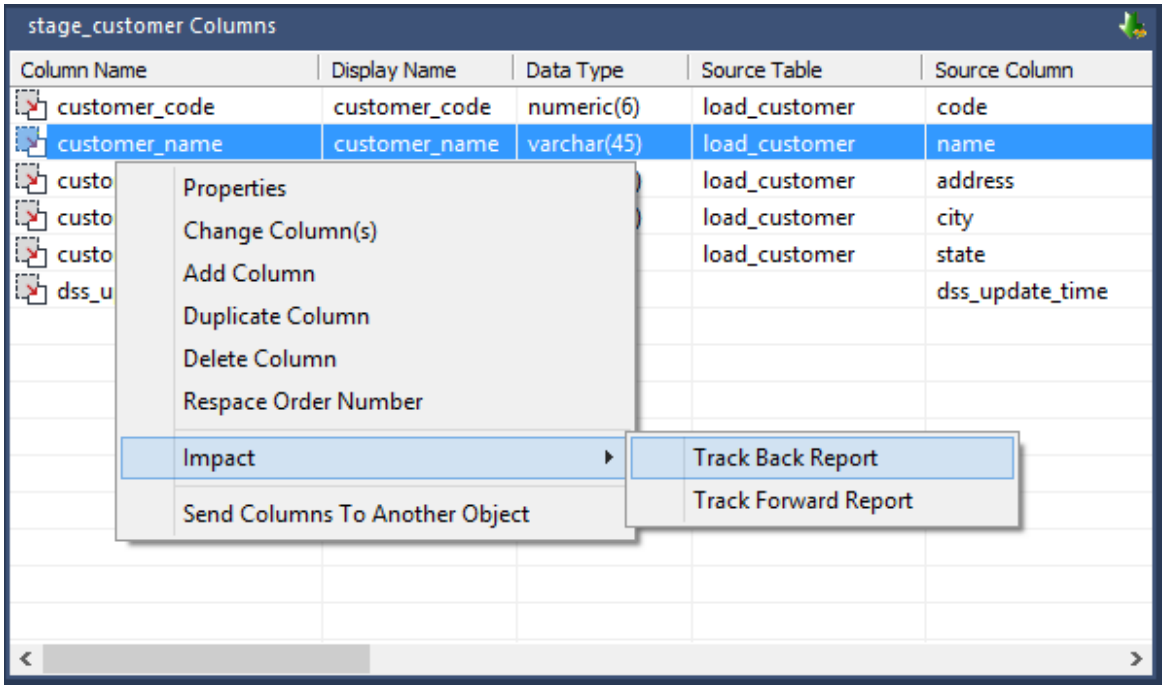
- Right-click on one of the columns in the middle pane and select **Sync the column order with the database**.

This will reorder the metadata columns to match the column order in the database table.

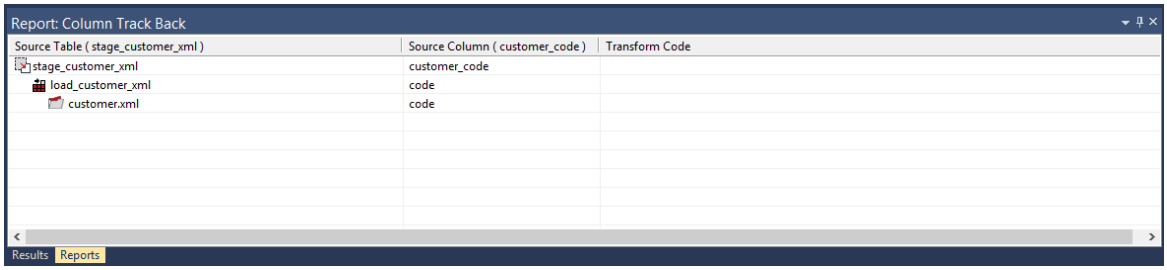


IMPACT

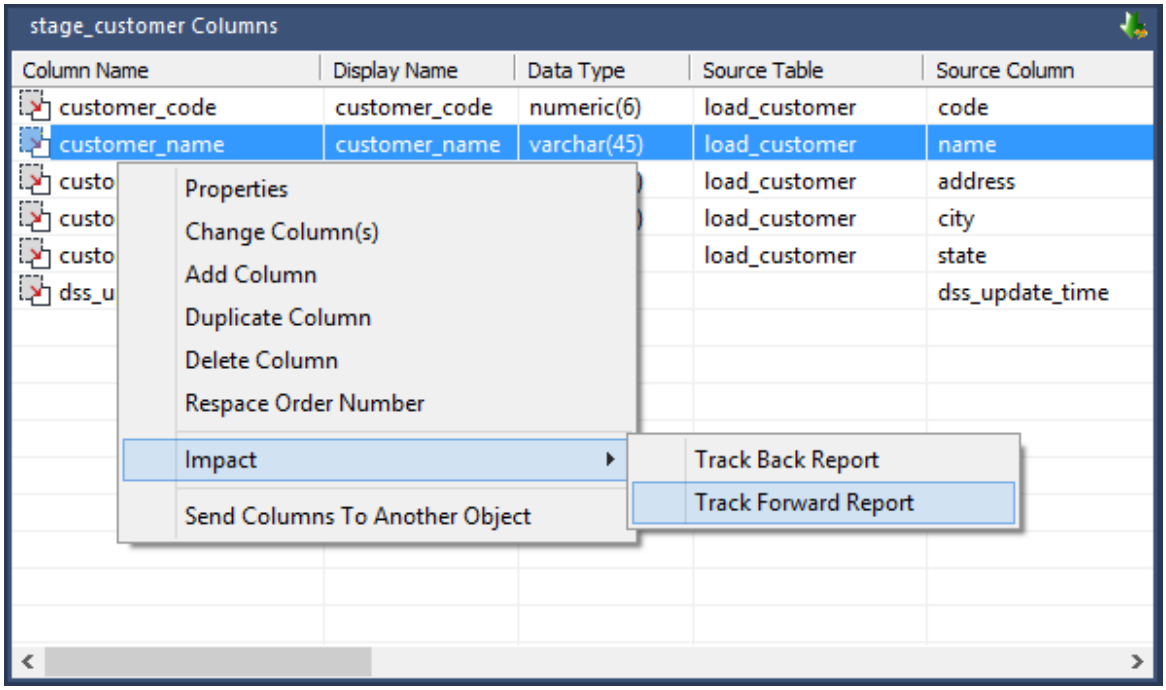
To display a Track Back Report, right-click on a column in the middle pane and select **Impact > Track Back Report**.



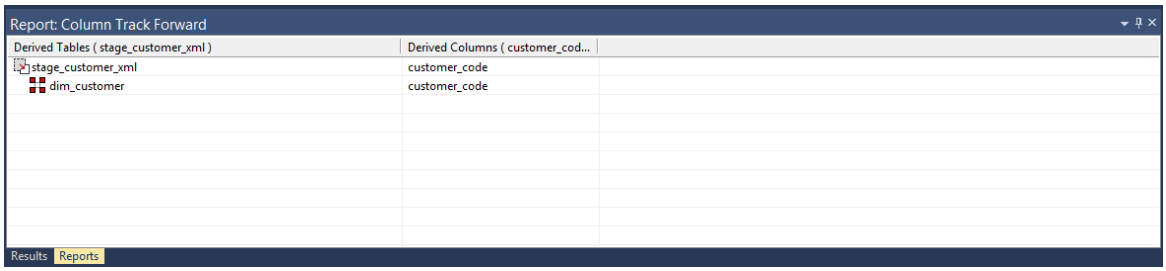
The report will be displayed in the bottom middle pane. This report lists the origins of the selected column. See ***Track Back Report*** (see "***Column Track-Back***" on page 872).



To display a Track Forward Report, right-click on a column in the middle pane and select **Impact > Track Forward Report**.

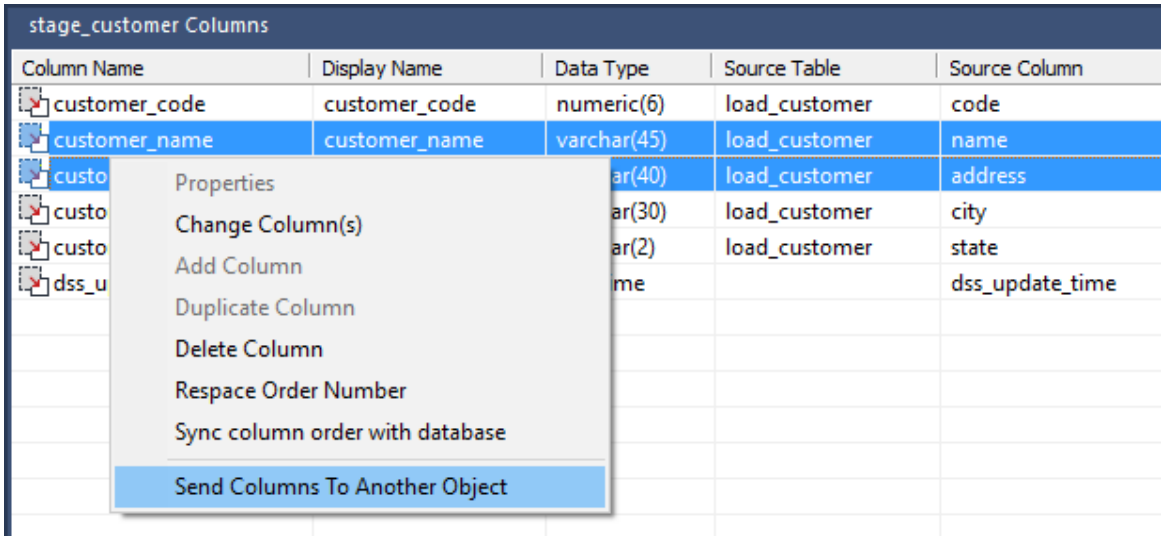


The report will be displayed in the bottom middle pane. This report lists the columns derived from the selected column. See ***Track Forward Report*** (see "***Column Track-Forward***" on page 874)

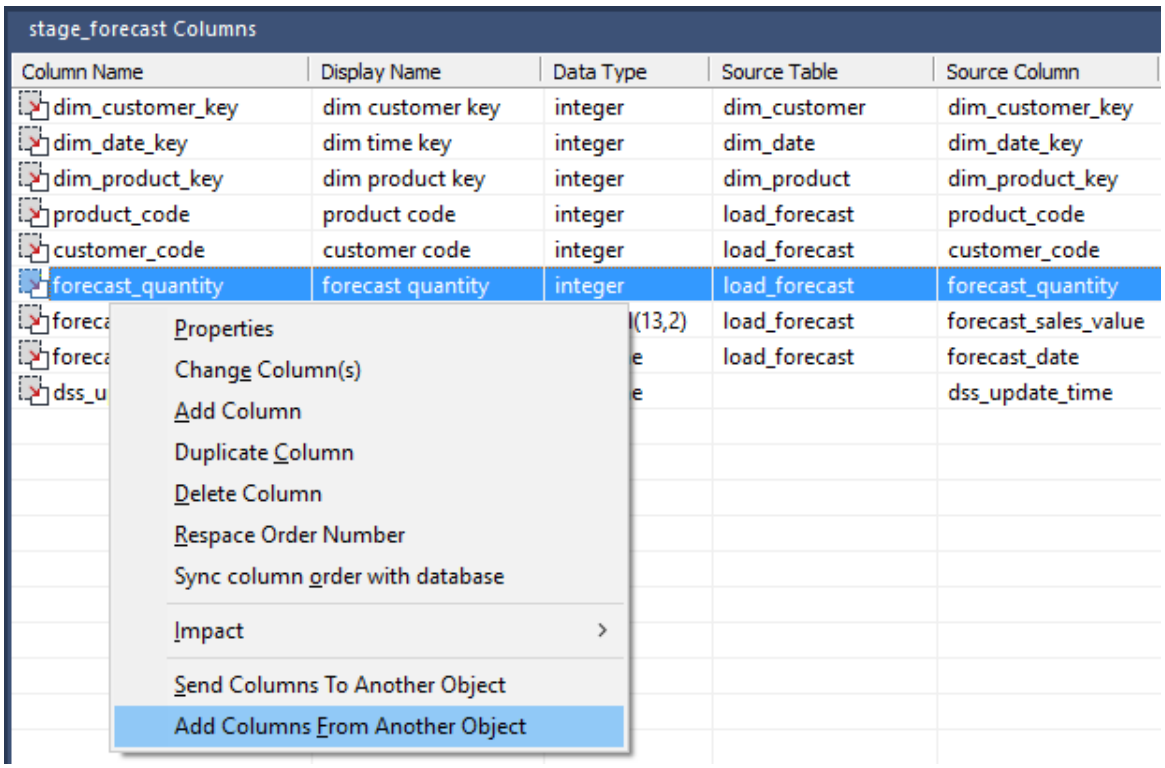


SEND COLUMNS TO ANOTHER OBJECT

To send/copy columns to another object, right-click on a column in the middle pane and select **Send Columns To Another Object**.



Click on the destination table in the left pane, then right-click in the middle pane and select **Add Columns From Another Object**.



The columns will be added to the destination table using the same functionality and settings as drag and drop.

stage_forecast Columns

Column Name	Display Name	Data Type	Source Table	Source Column
dim_customer_key	dim customer key	integer	dim_customer	dim_customer_key
dim_date_key	dim time key	integer	dim_date	dim_date_key
dim_product_key	dim product key	integer	dim_product	dim_product_key
product_code	product code	integer	load_forecast	product_code
customer_code	customer code	integer	load_forecast	customer_code
forecast_quantity	forecast quantity	integer	load_forecast	forecast_quantity
customer_name	customer_name	varchar(45)	stage_customer	customer_name
customer_address	customer_address	varchar(40)	stage_customer	customer_address
forecast_sales_value	forecast sales value	decimal(13,2)	load_forecast	forecast_sales_value
forecast_date	forecast date	datetime	load_forecast	forecast_date
dss_update_time	dss update time	datetime		dss_update_time

WhereScape®

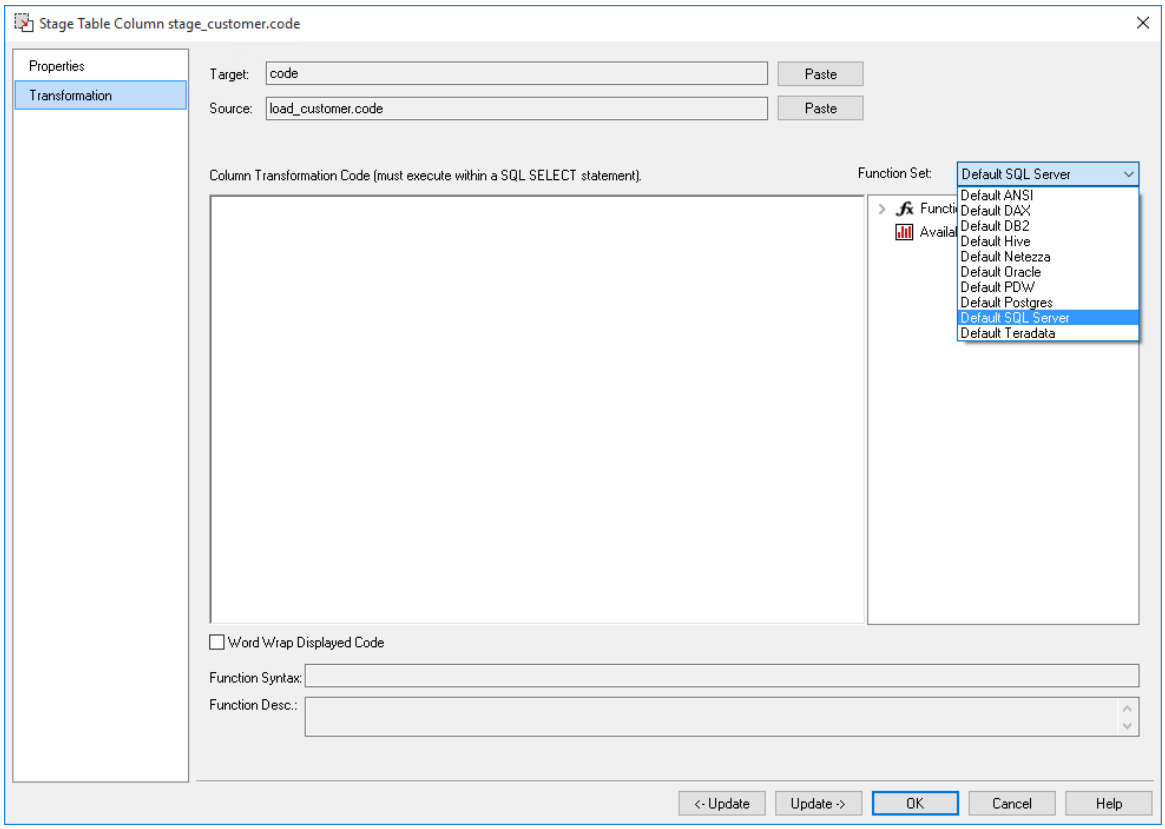
CHAPTER 40
DATABASE FUNCTIONS

IN THIS CHAPTER

USING DATABASE FUNCTION SETS

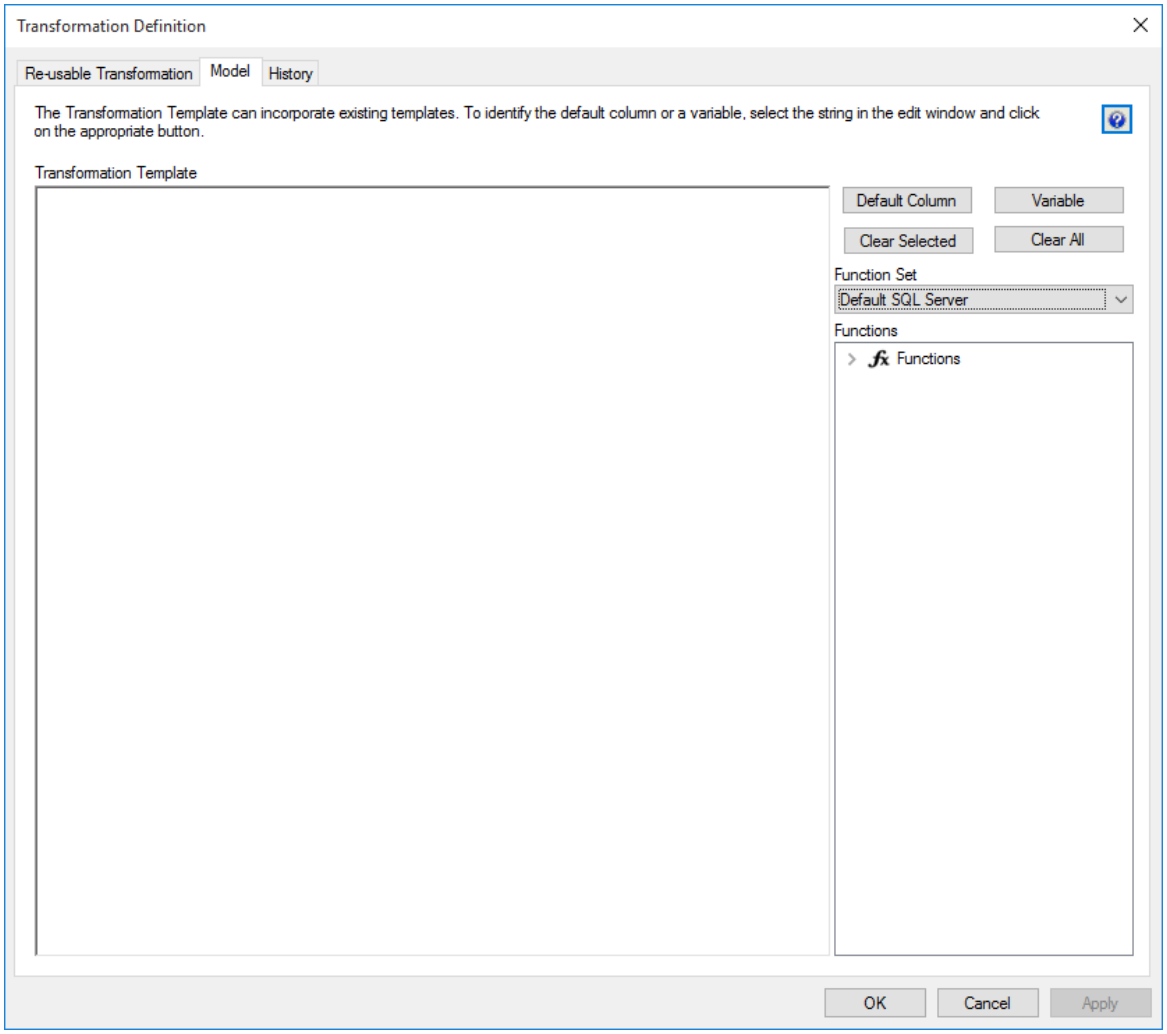
Database function sets contain a list of functions and operators that can be used for building transformations. These function sets may be created, edited, deleted, imported and exported using the **Database Functions** options on the **Tools** menu.

Column Transformation Properties Dialog



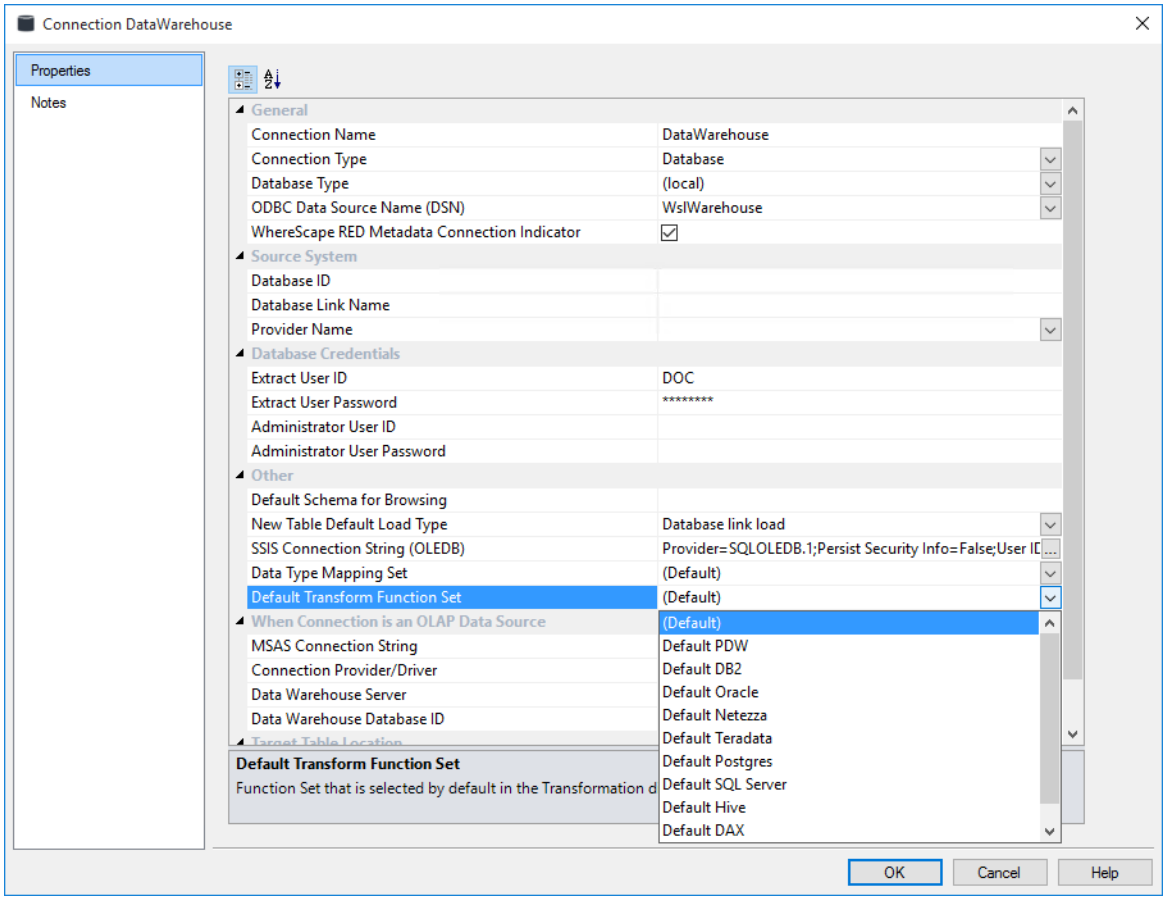
A drop-down list allows the user to select which set of functions are to be displayed in the tree view when creating a transformation on a column of a table.

Transformation Definition Dialog



A drop-down list allows the user to select which set of functions are to be displayed in the tree view when creating a re-usable transformation in **Tools/Define Re-Usable Transformations...**

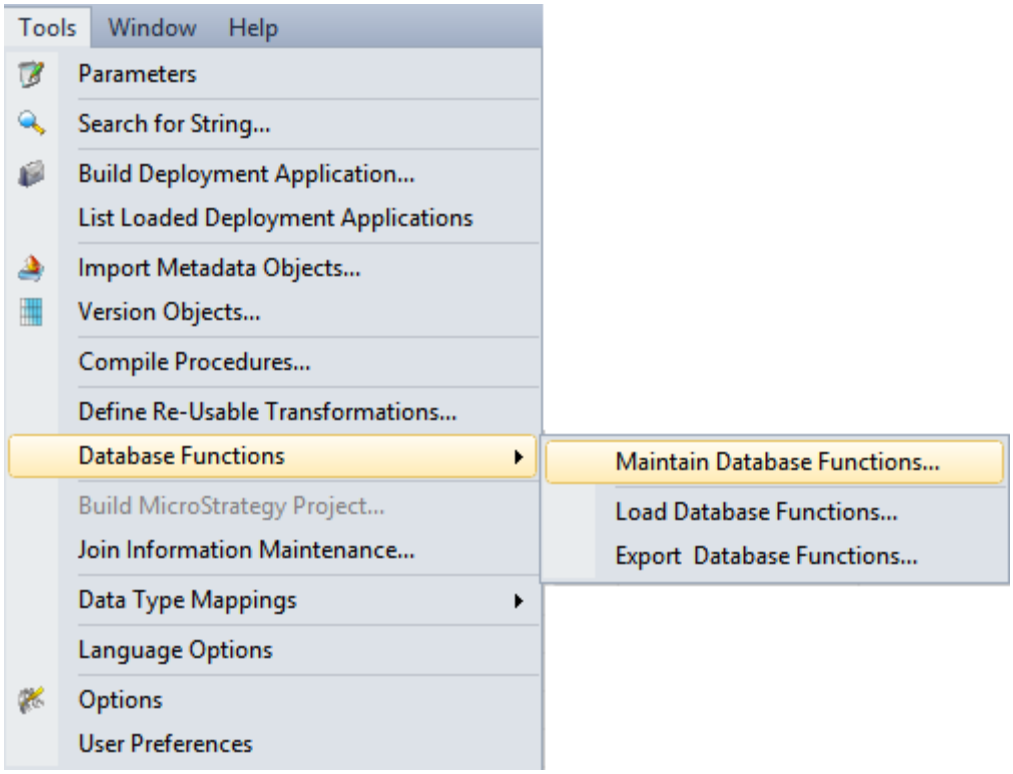
Connection Properties Dialog



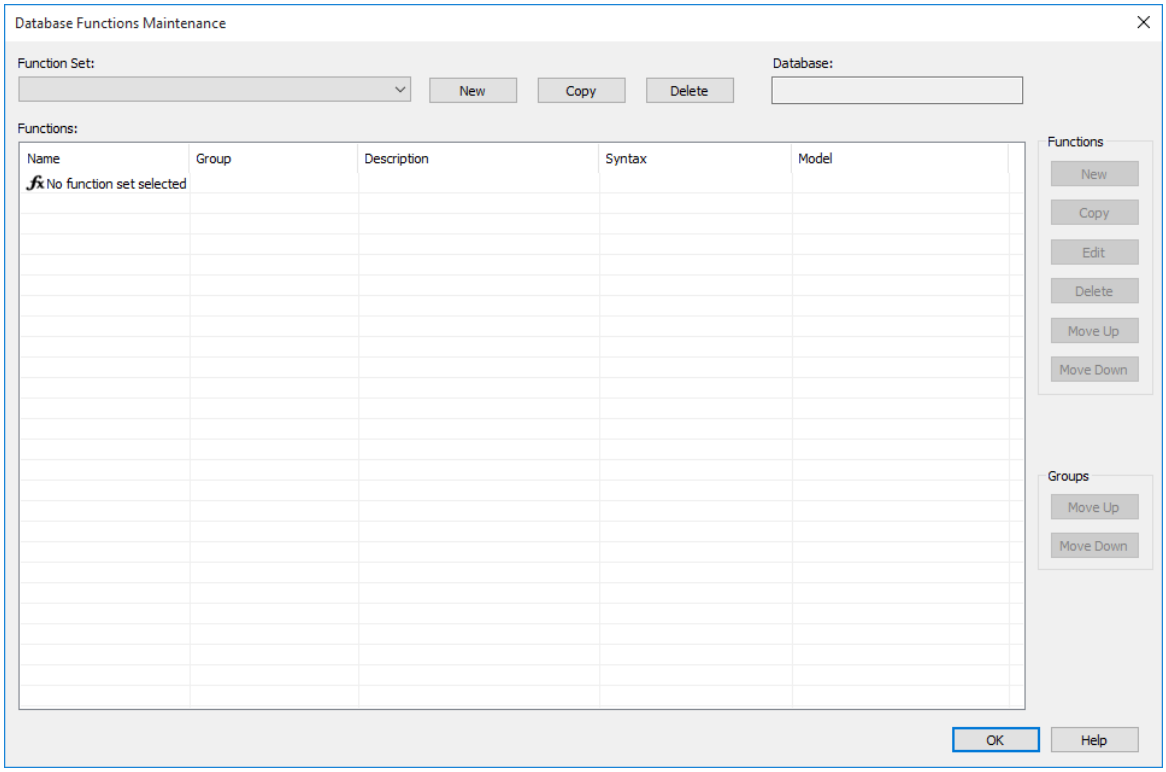
In the Connection Properties dialog, when the **Data Warehouse** check-box is selected, a drop-down list **Default Transform Function Set** is displayed. This is the default set that will be selected in the transformation dialogs above.

MAINTAINING DATABASE FUNCTION SETS

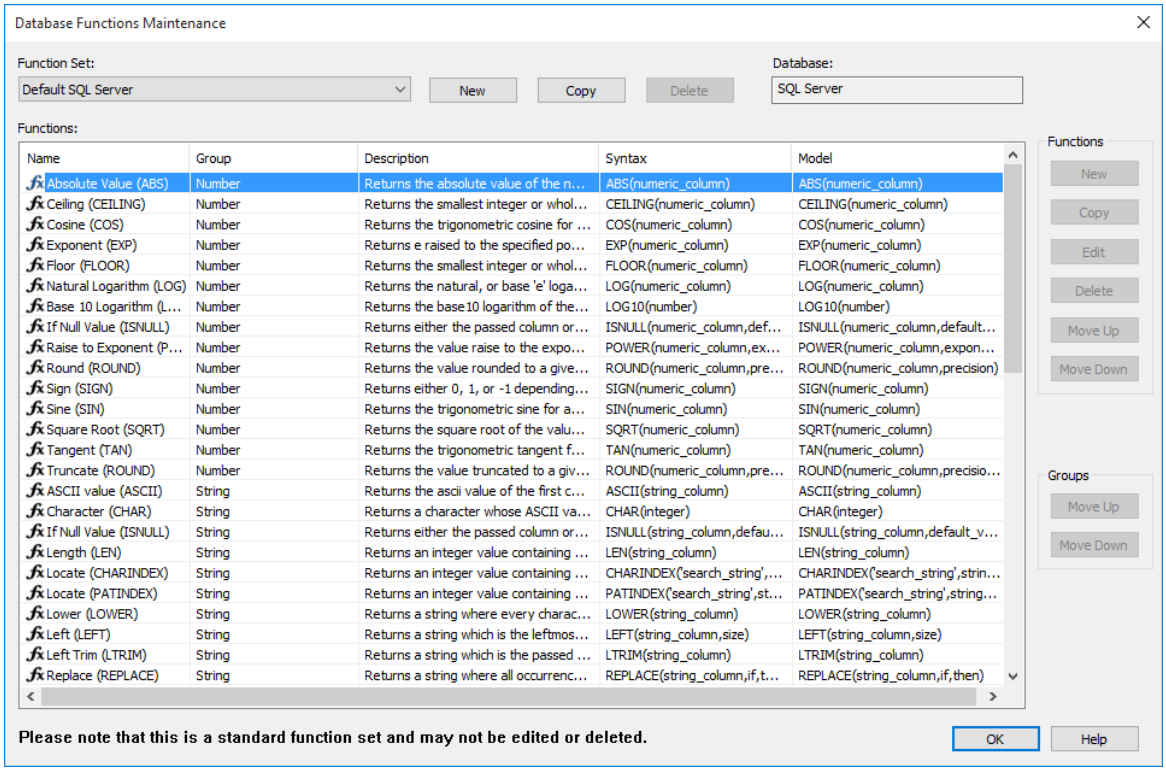
To maintain the database function sets, select **Tools/Database Functions/Maintain Database Functions...**



The **Maintain Database Functions** dialog is displayed.



Select a database function set from the **Function Set** drop-down list.



To create a database function set, see ***Creating a New Database Function Set*** (on page 1223)

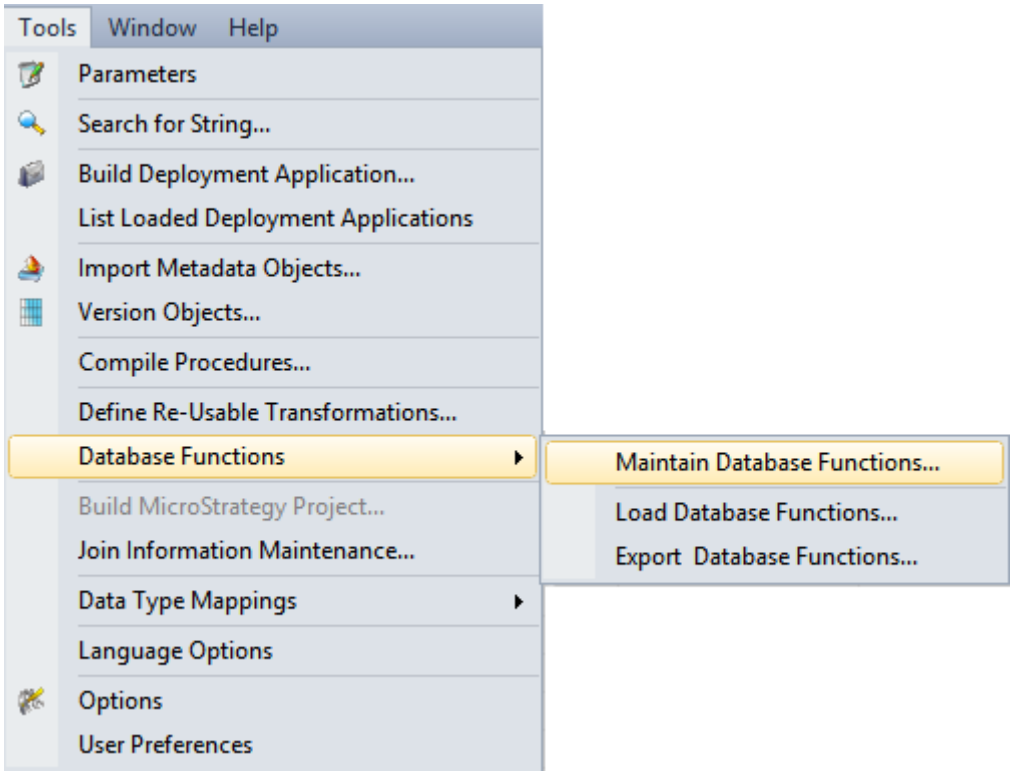
To copy a database function set, see ***Copying a Database Function Set*** (on page 1225)

To edit a database function set, see ***Editing a Database Function Set*** (on page 1228)

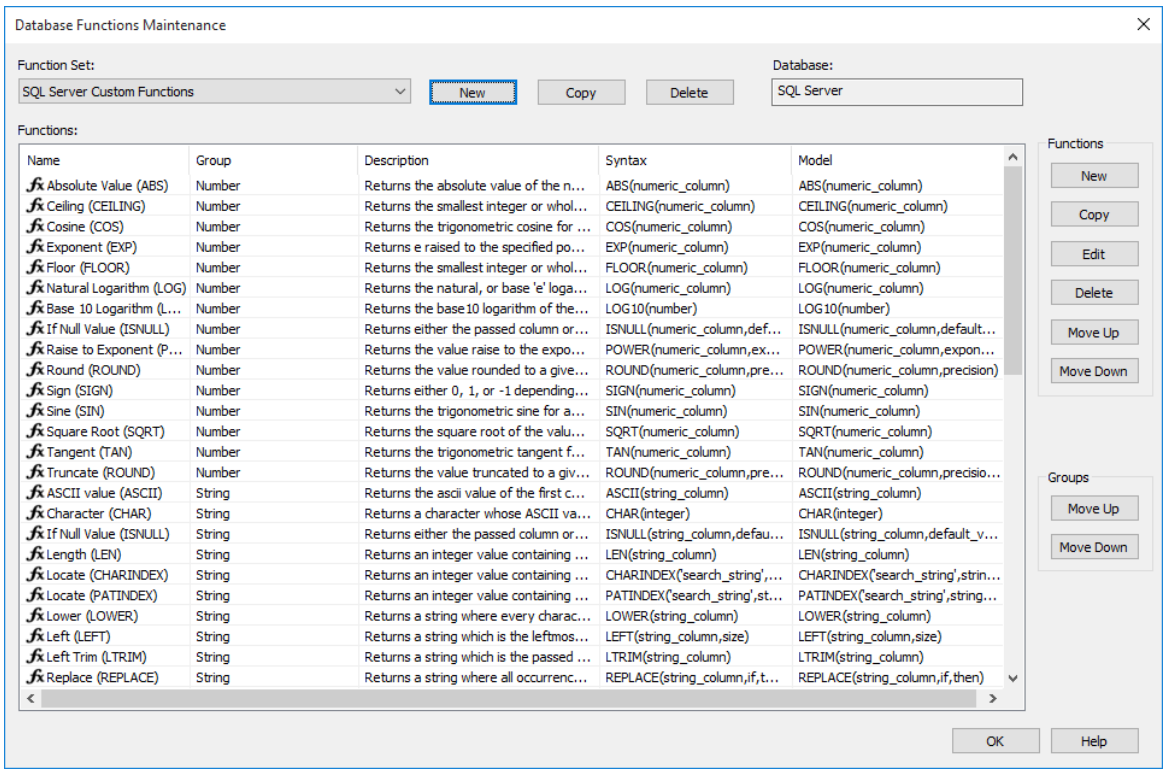
To delete a database function set, see ***Deleting a Database Function Set*** (on page 1237)

CREATING A NEW DATABASE FUNCTION SET

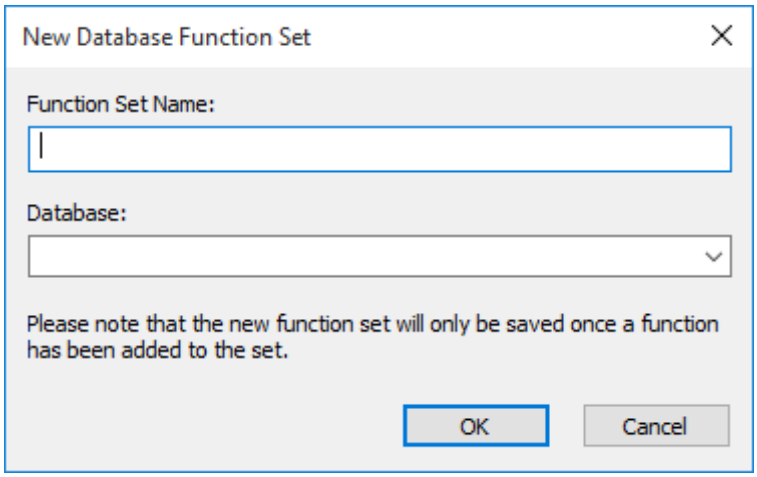
To create a new database function set, select **Tools/Database Functions/Maintain Database Functions...**



Click on the **New** button.



Enter a **Function Set Name** and select a **Database** from the drop-down list. Click **OK**.



Function Set Name

Enter a unique function set name.

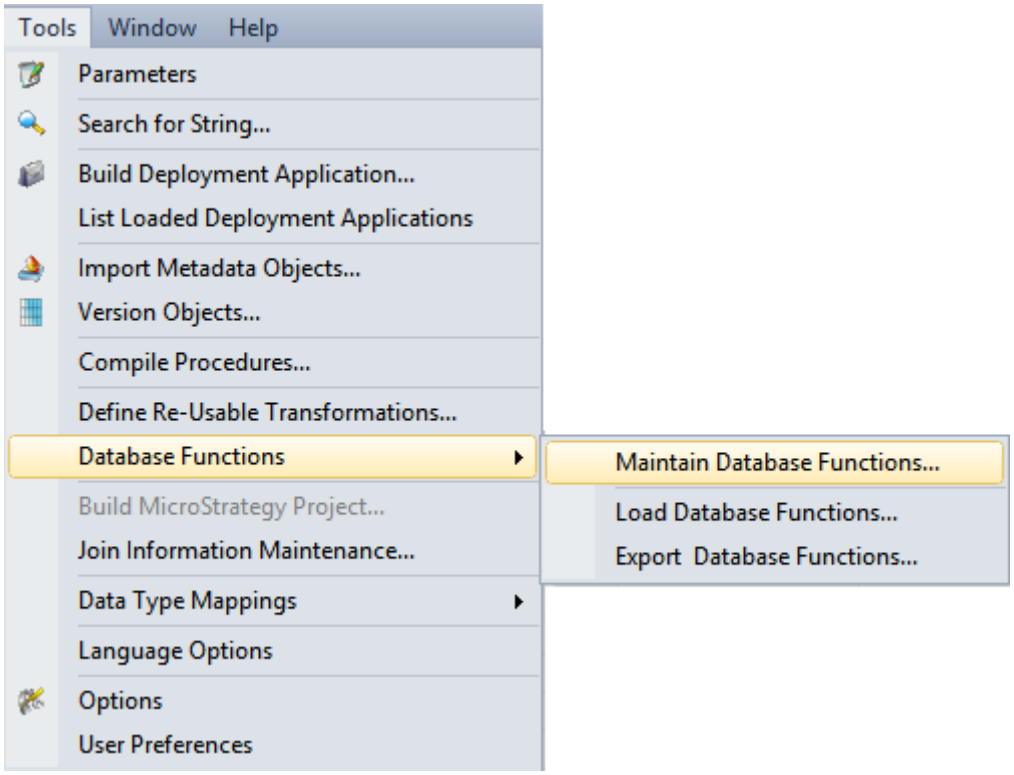
Note: The function set name may not contain the phrase "Standard Functions" as this is reserved for WhereScape supplied function sets.

Database

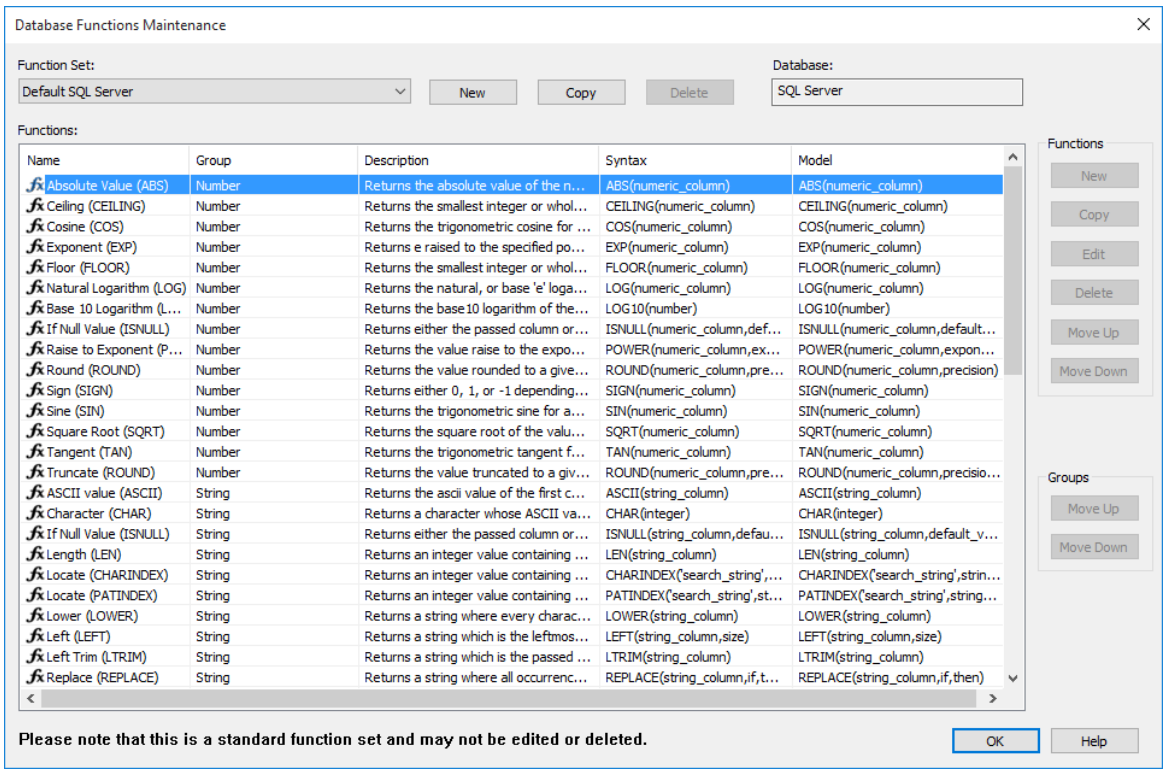
Select the database from the drop-down list or type in the name of the database if not already in the list. This field is mandatory.

COPYING A DATABASE FUNCTION SET

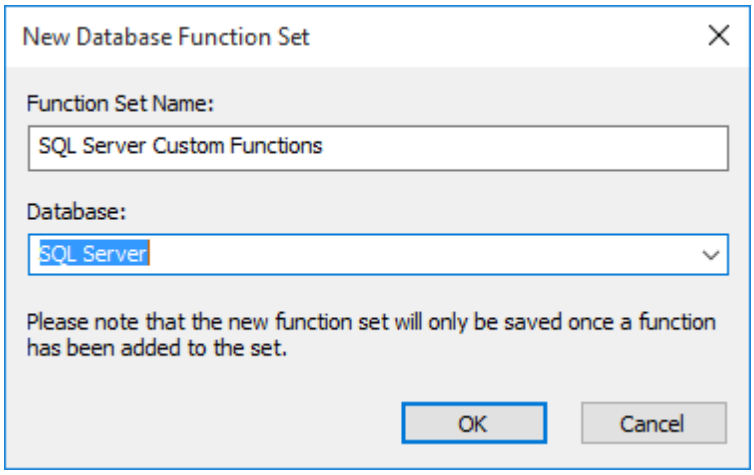
To copy an existing database function set, select **Tools/Database Functions/Maintain Database Functions...**



Select a **Function Set** from the drop-down list and click on the **Copy** button.



Enter the new **Function Set Name** and select a **Database** from the drop-down list. Click **OK**.



Function Set Name

Enter a unique function set name.

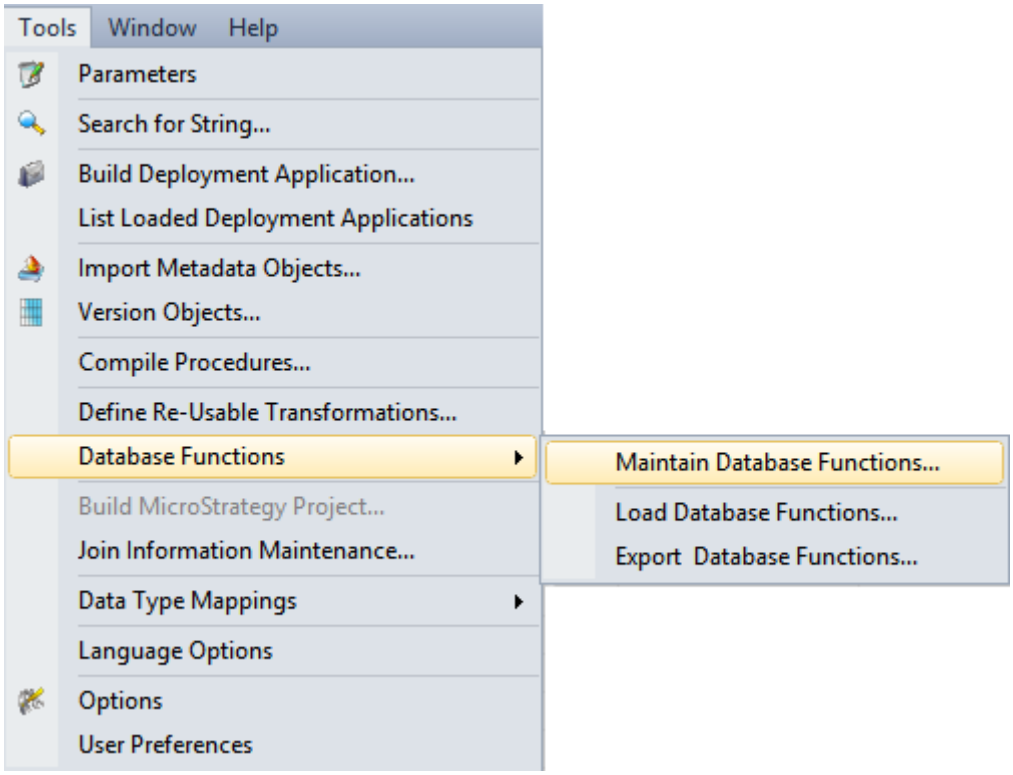
Note: The function set name may not contain the phrase "Standard Functions" as this is reserved for WhereScape supplied function sets.

Database

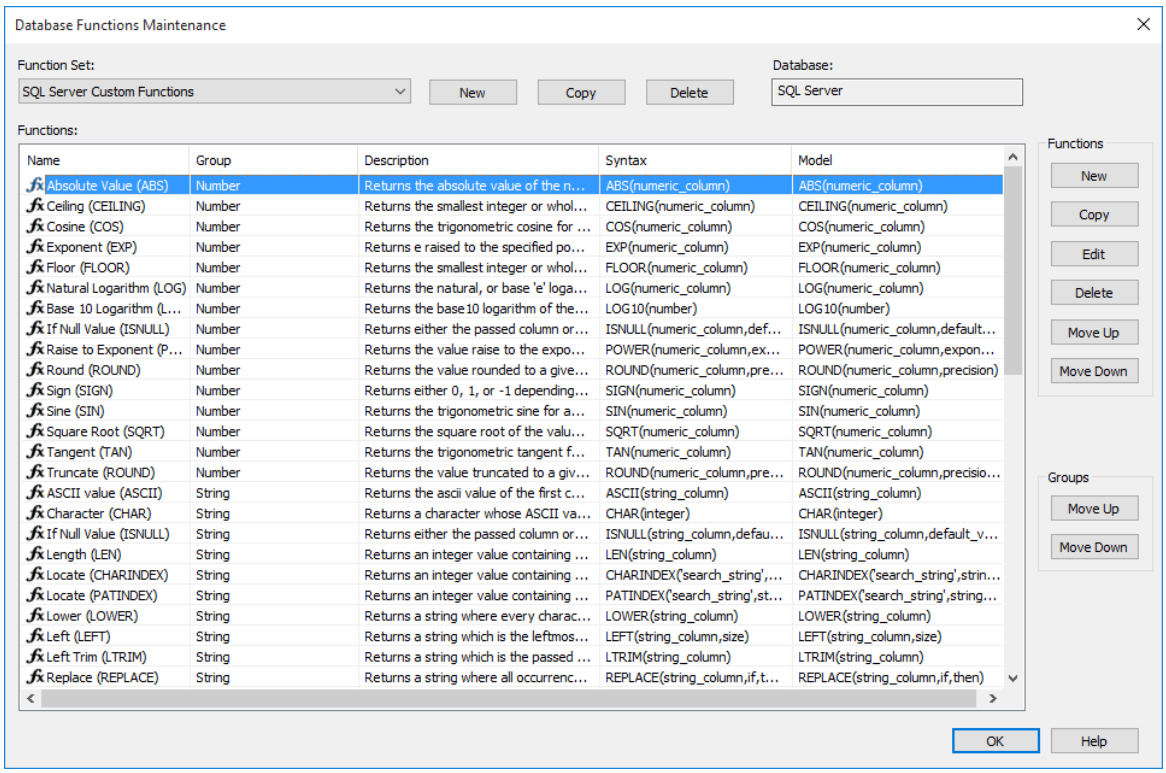
Select the database from the drop-down list or type in the name of the database if not already in the list. This field is mandatory.

EDITING A DATABASE FUNCTION SET

To edit a database function set, select **Tools/Database Functions/Maintain Database Functions...**



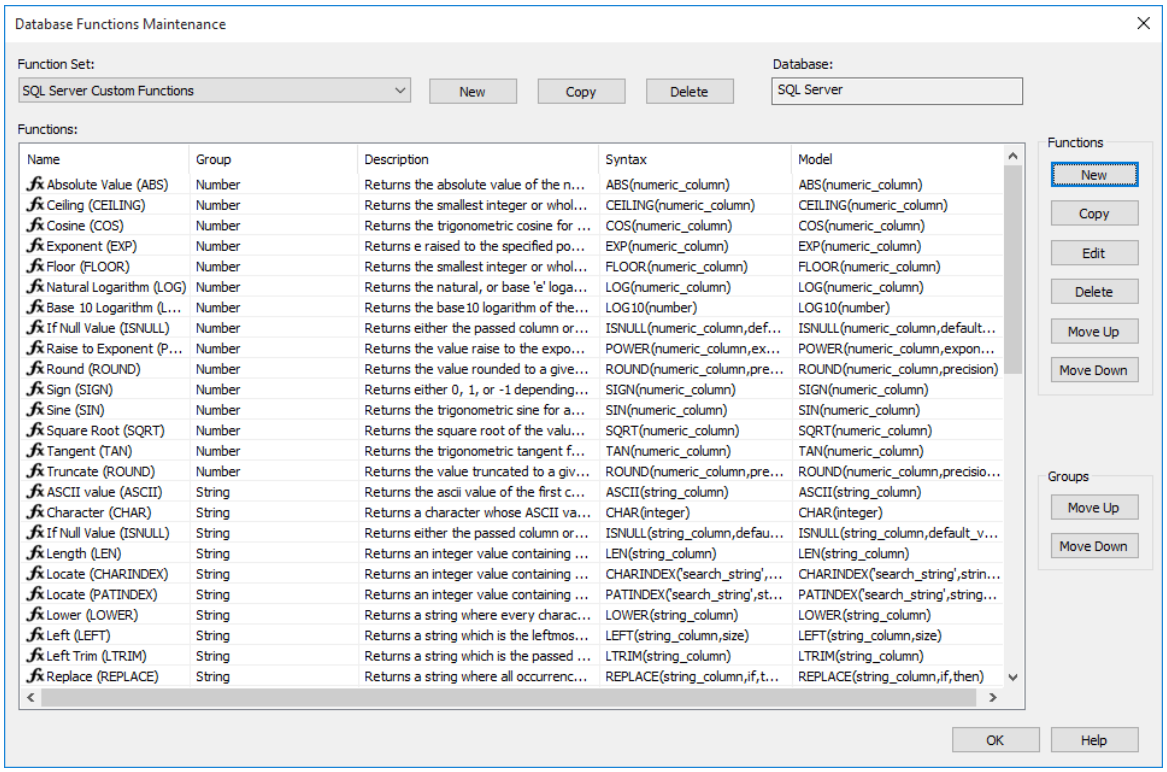
Select a database function set from the **Function Set** drop-down list.



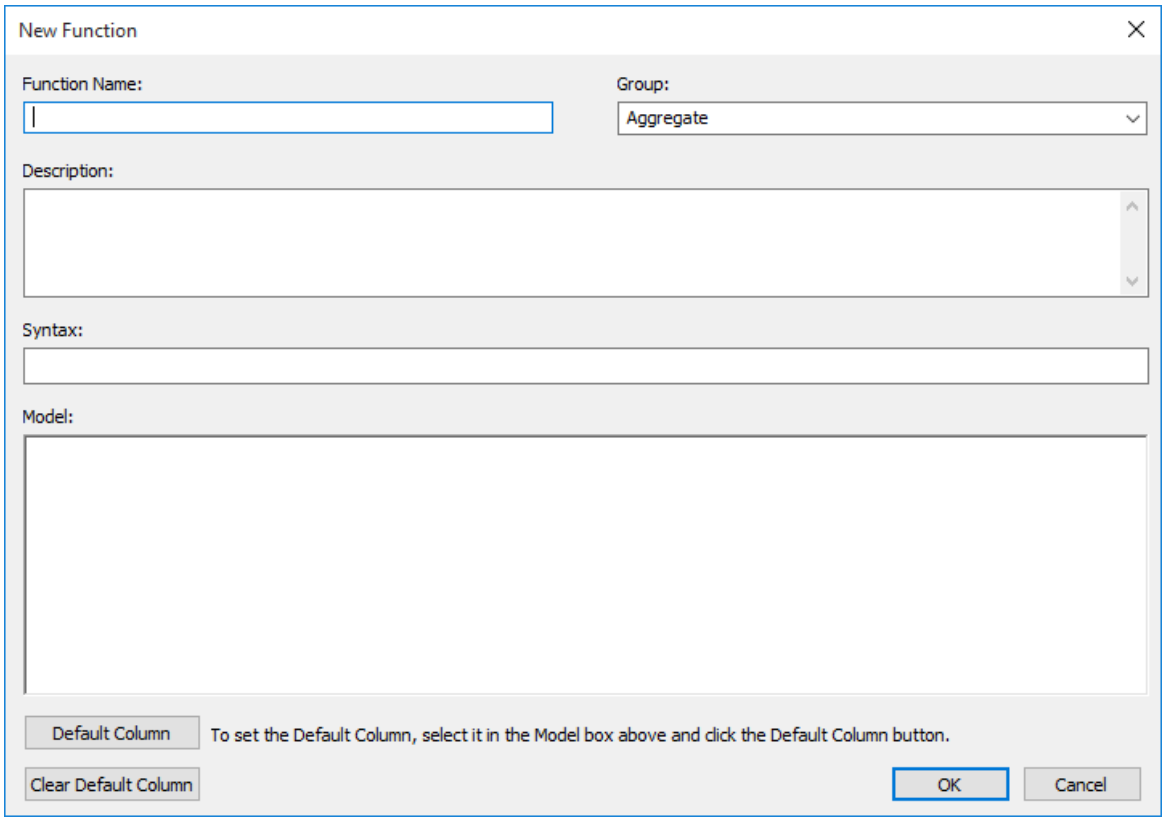
On the right is a group of buttons used to maintain the list of functions in a function set. These buttons are not available for standard function sets.

To add a new function to the database function set:

Click on the **New** button on the right.



Enter the details for the new function and click **OK**.



Function Name

This field is mandatory and must be unique within the group.

Group

This field is mandatory. Select a group from the drop-down list or add a new group name.

Description

Enter a description for the function.

Syntax

Enter the syntax for the function.

Model

This field is mandatory. This is the text that will be pasted into the transformation/ model fields when the function is selected.

Default Column

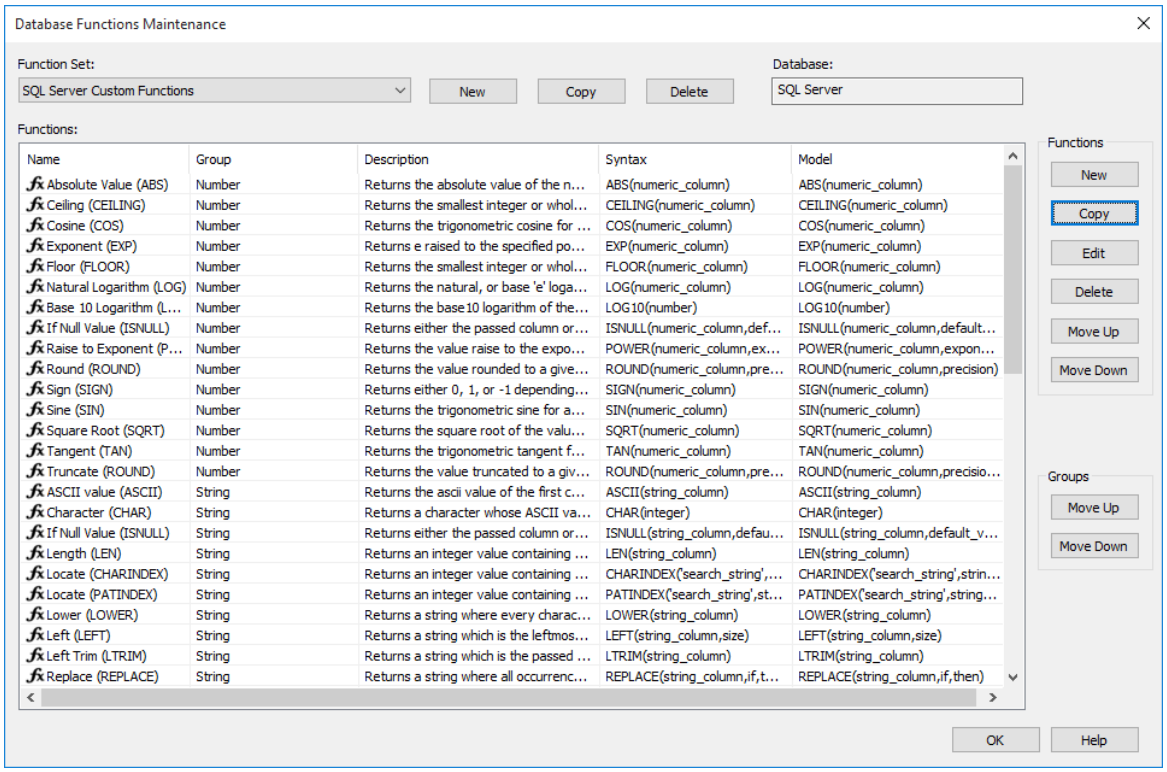
This is the text that will automatically be highlighted when the function is used in the transformation/model dialogs. To set the default column, highlight it in the model field and click the **Default Column** button. The default column will now show in red in the Model field.

Clear Default Column

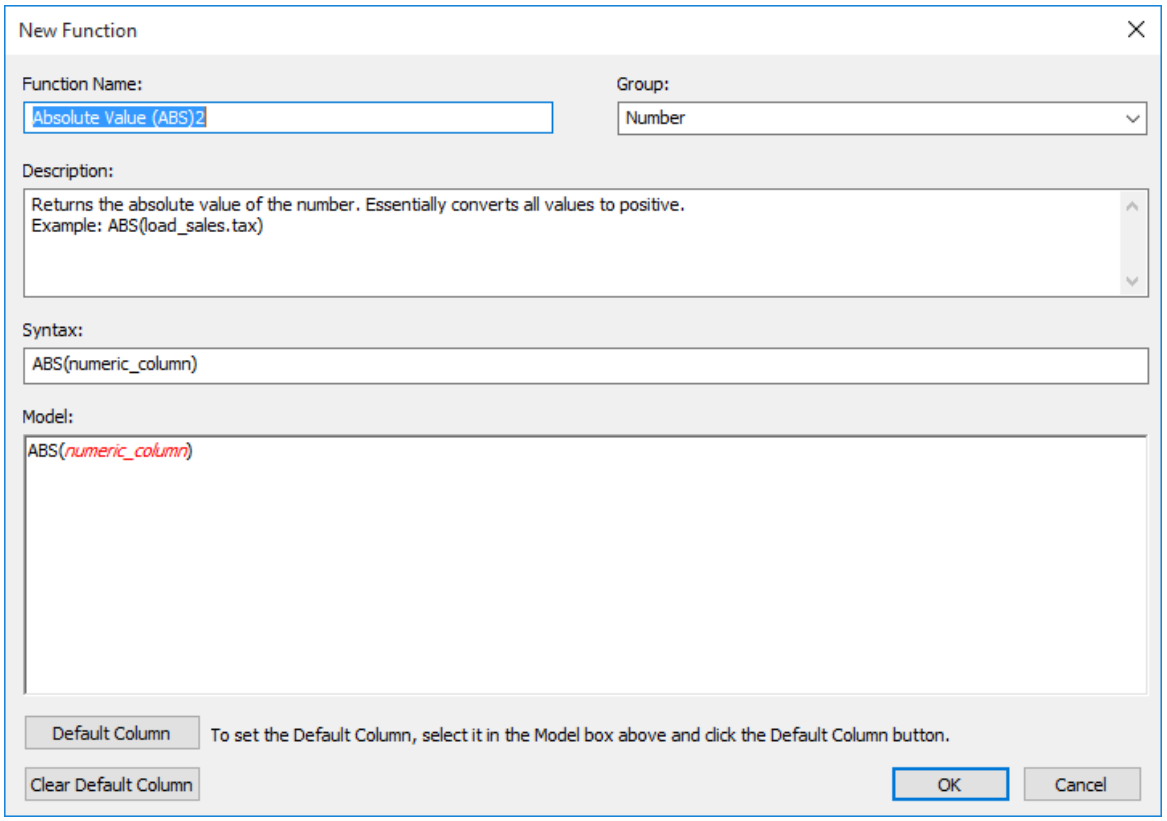
Click this button to clear the default column.

To copy an existing function in the database function set:

Select the function and then click on the **Copy** button on the right.

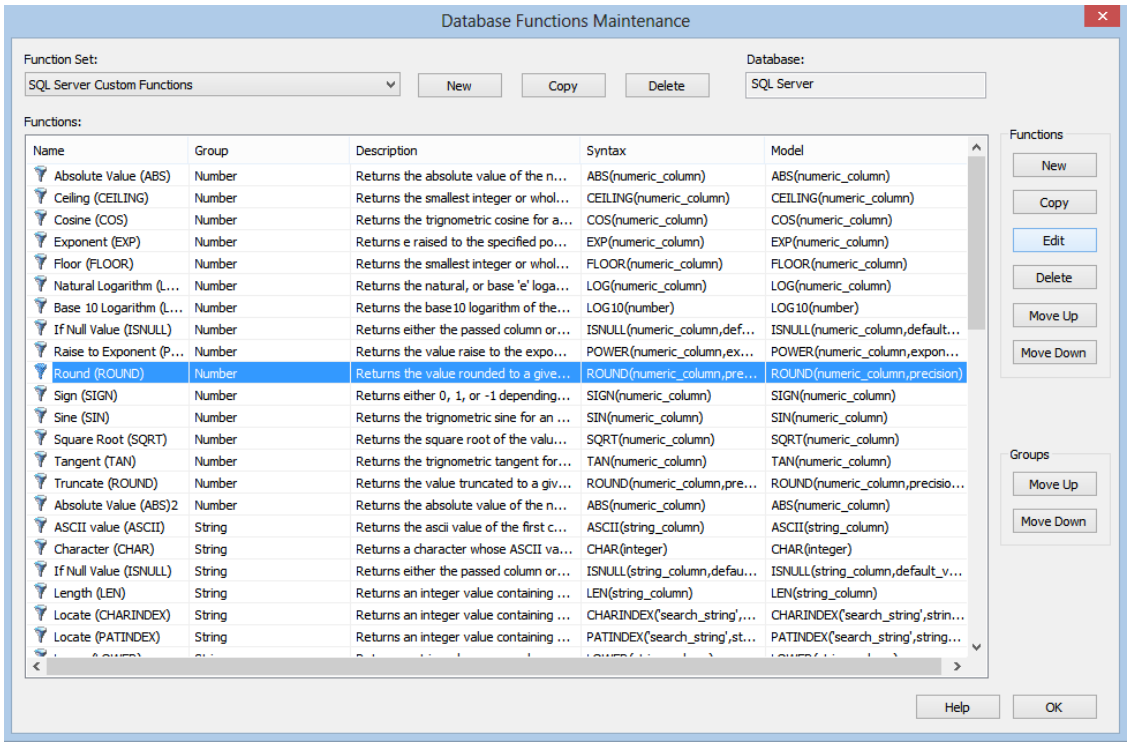


Enter a new **Function Name** and change any other details; then click **OK**.

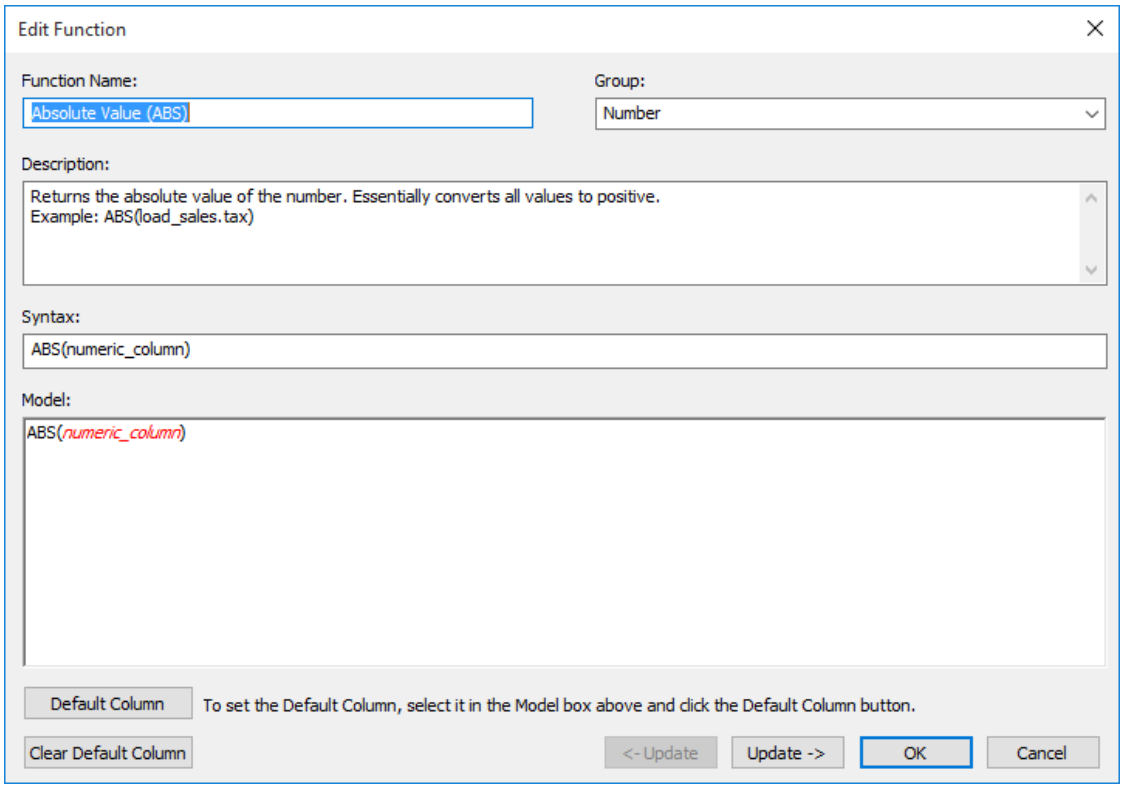


To edit an existing function in the database function set:

Select the function and then click on the **Edit** button on the right.



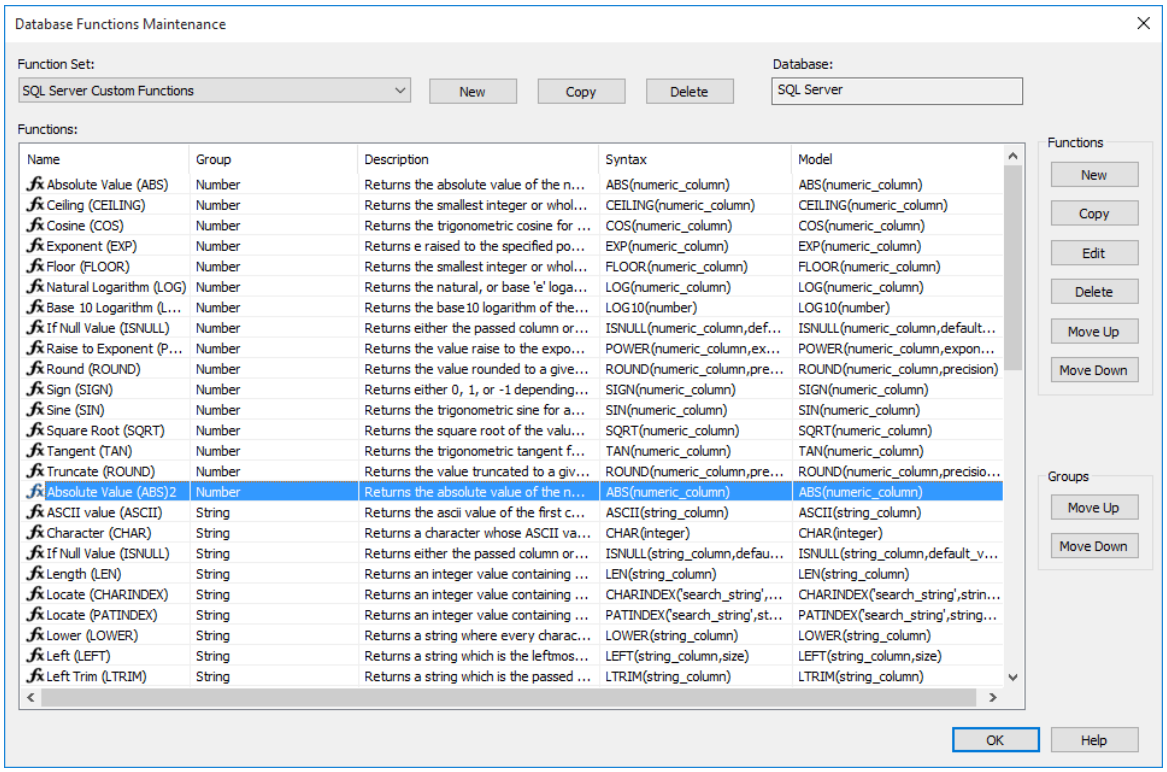
Change any fields as required and then click **OK**.



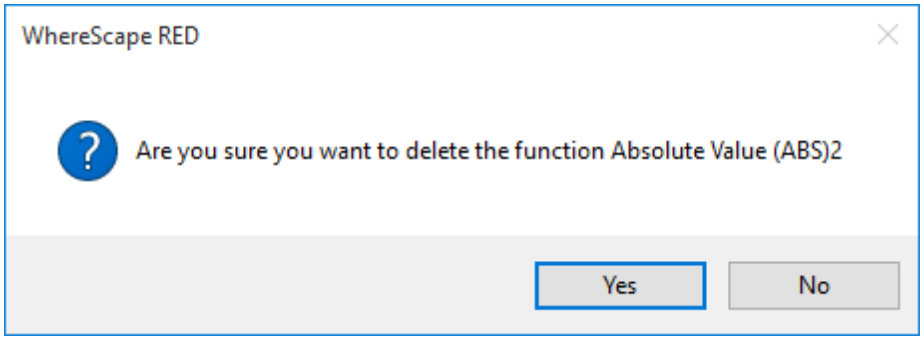
Note: A function can also be edited by double-clicking on the function.

To delete an existing function in the database function set:

Select the function and then click on the **Delete** button.

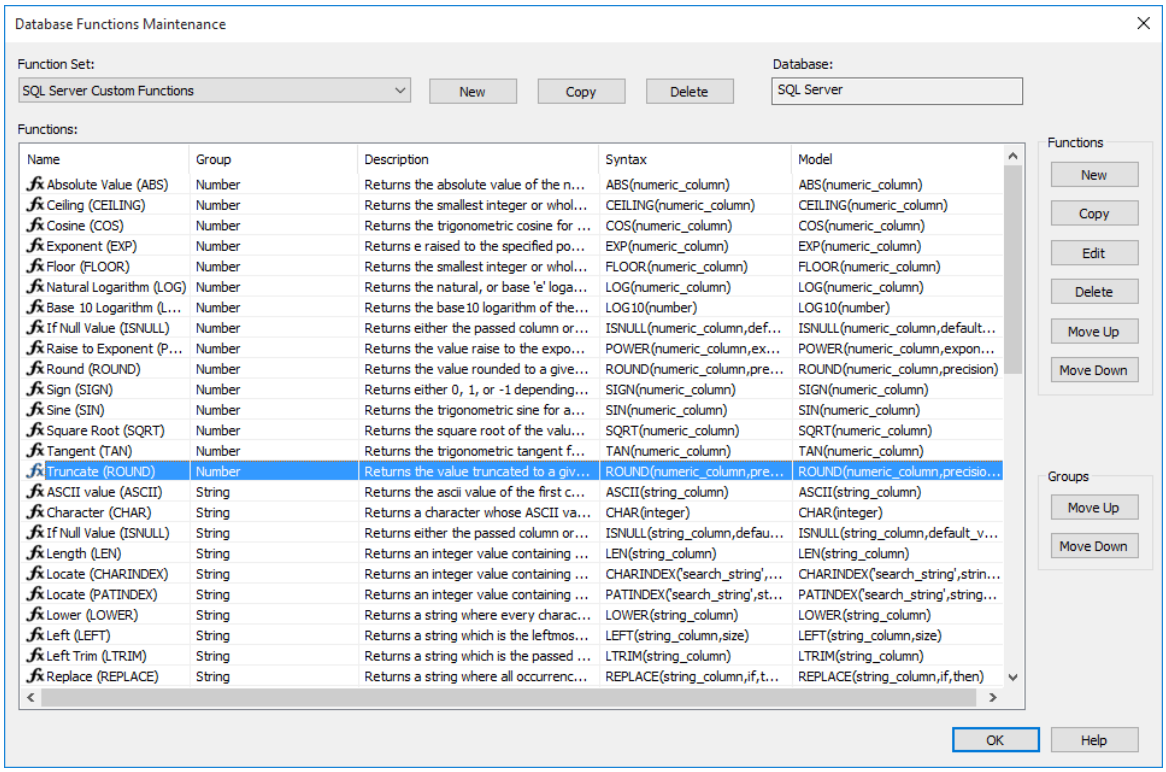


Click **Yes** to confirm the delete.



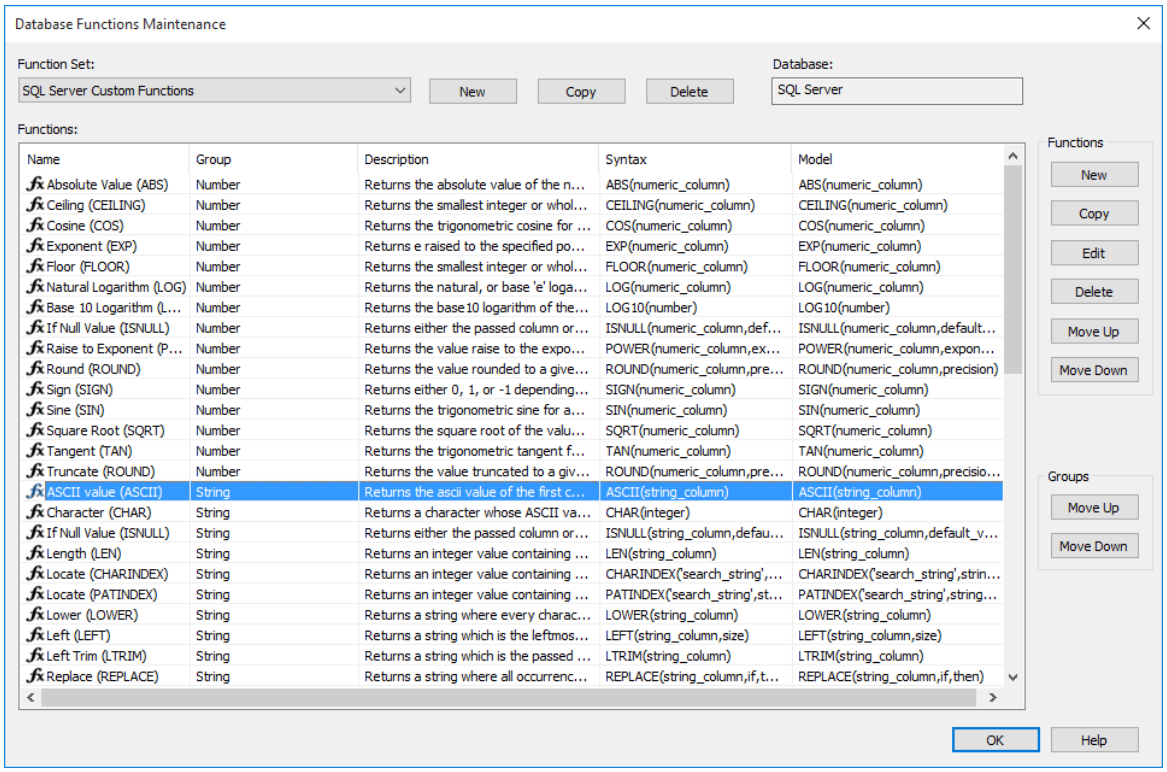
To move a function in the database function set up or down in the list:

Select a function and then click on the **Move Up** button on the right to move the function up in the function list, within its group; else click on the **Move Down** button on the right to move the function down in the function list, within its group.



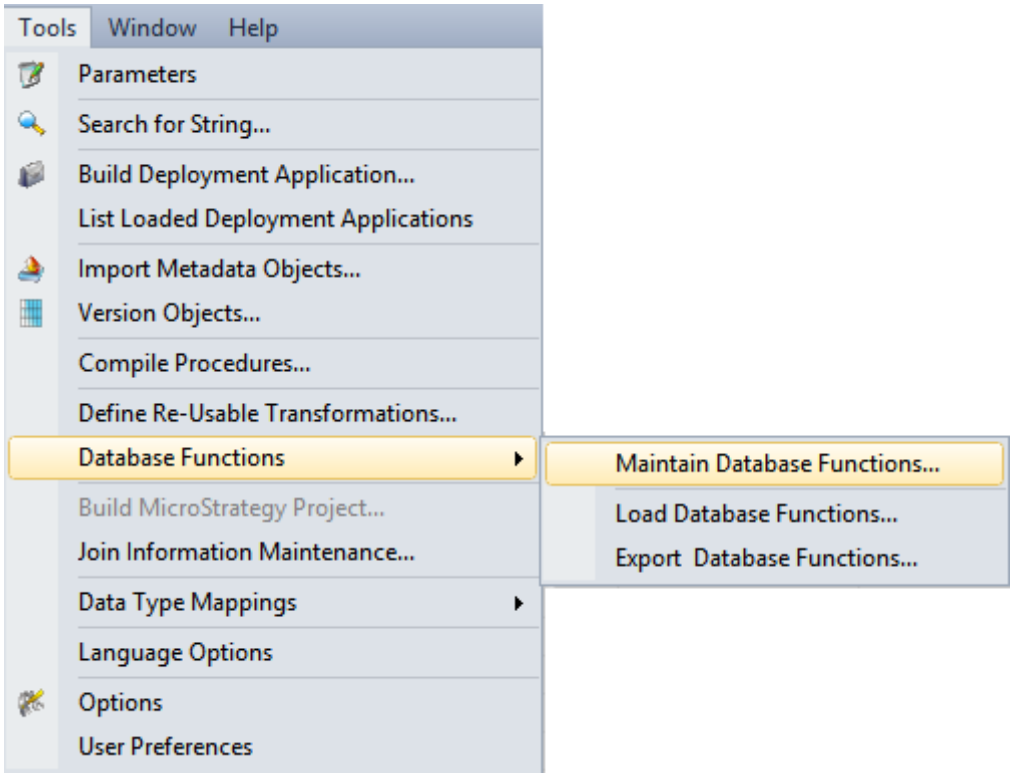
To move a group of functions in the database function set up or down in the list:

Using the Group column, select any function in a particular group and then click on the **Move Up** button under the **Groups** heading on the right to move the function group up in the function list. Similarly, use the **Move Down** button under the Groups heading to move a function group down in the function list.

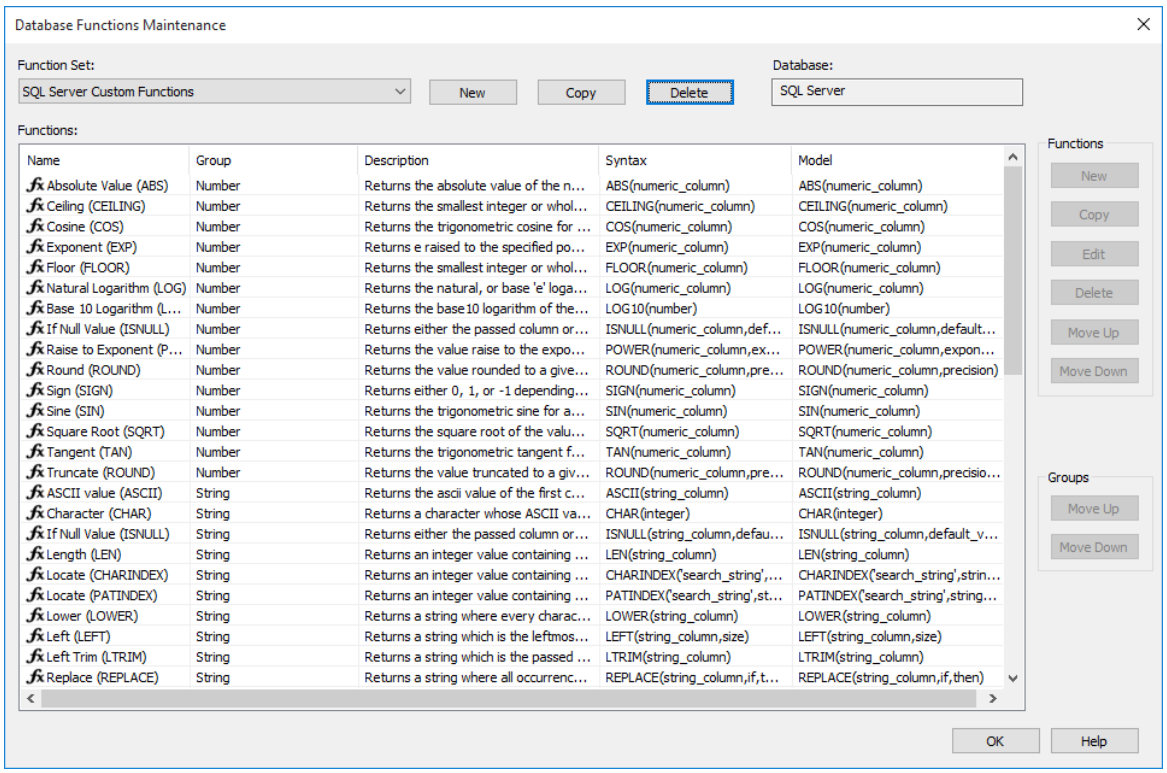


DELETING A DATABASE FUNCTION SET

To delete a database function set, select **Tools/Database Functions/Maintain Database Functions...**

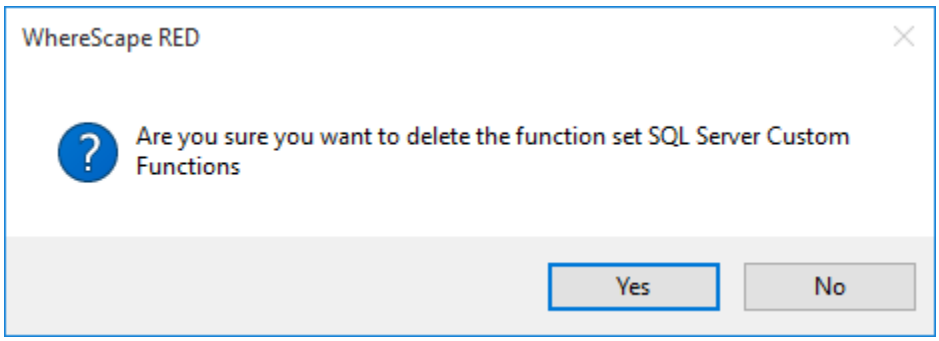


Select a **Function Set** from the drop-down list and click on the **Delete** button.



Note: The **Delete** button is disabled for standard function sets.

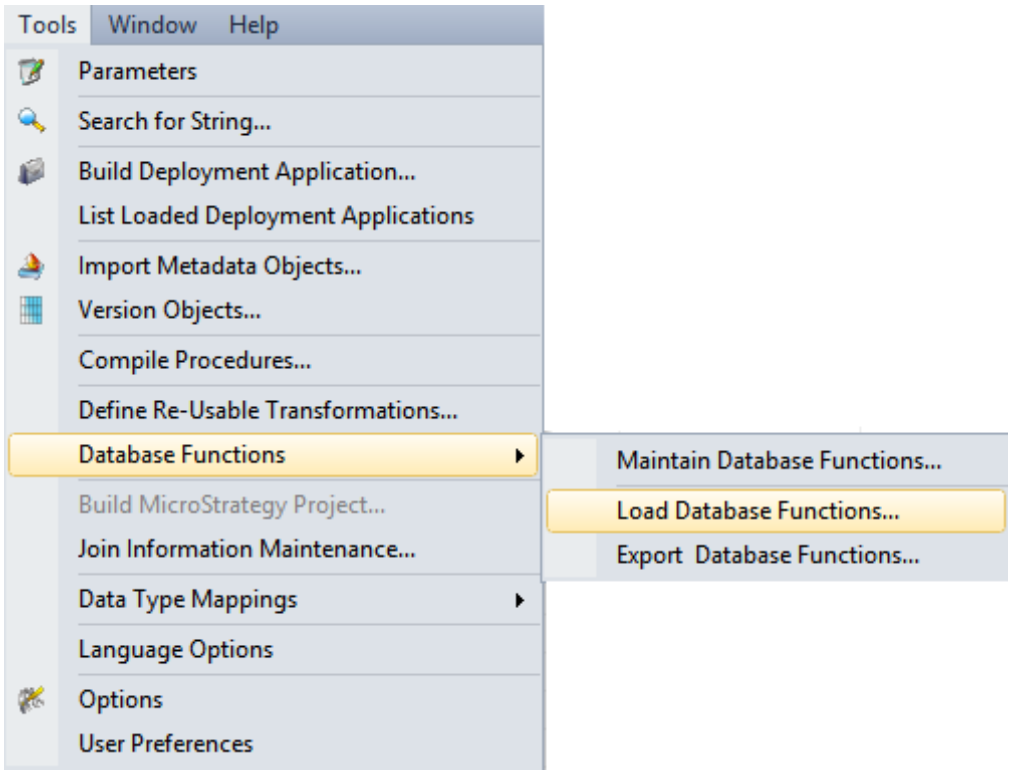
Click **Yes** to confirm the delete.



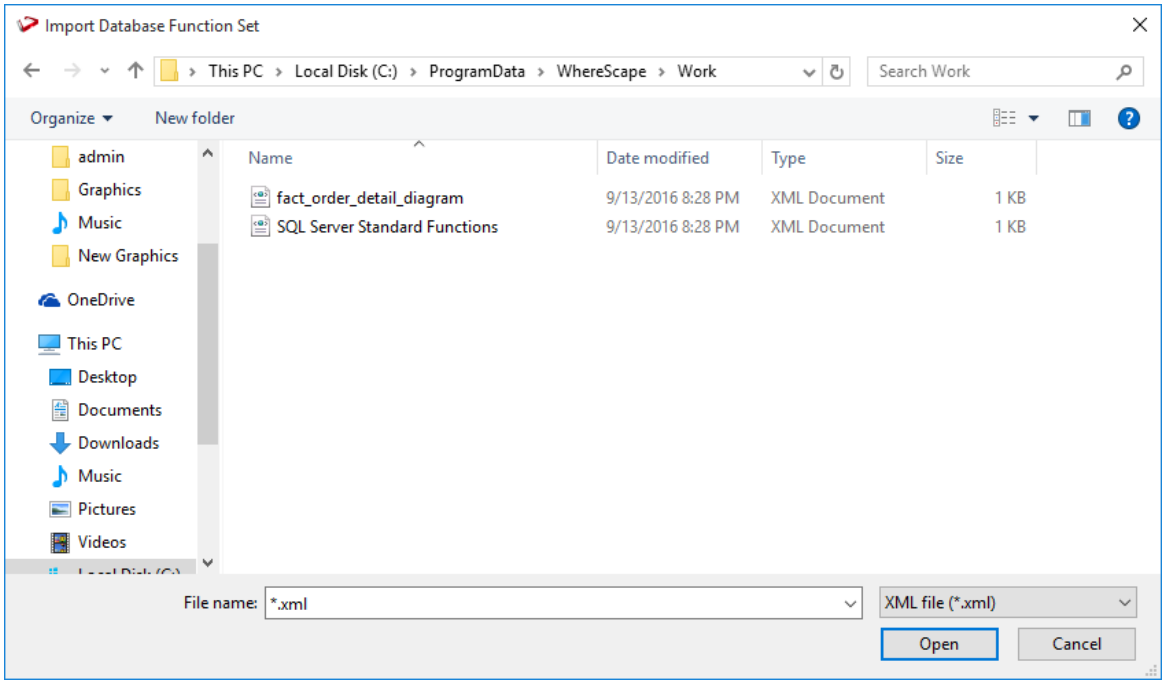
When all functions are deleted, the function set ceases to exist.

LOADING DATABASE FUNCTION SETS

To load a database function set, select **Tools/Database Functions/Load Database Functions...**

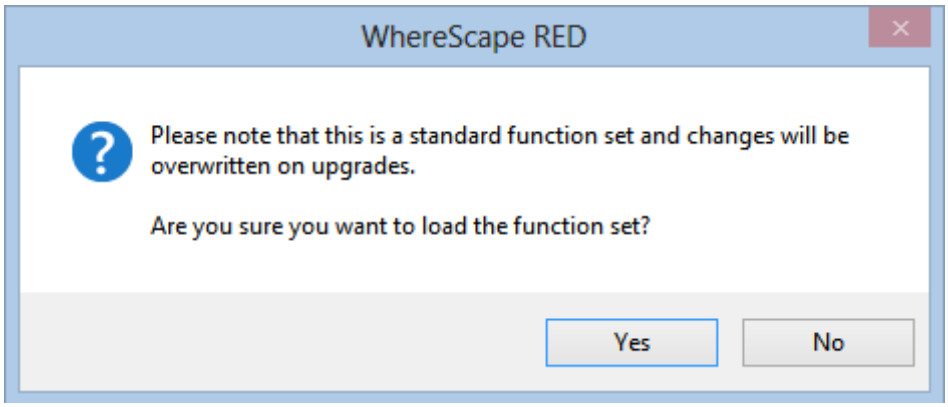


The following dialog is displayed. Select the xml file to load the database functions. By default RED expects the xml files to be in **ProgramData\WhereScape\Work.**

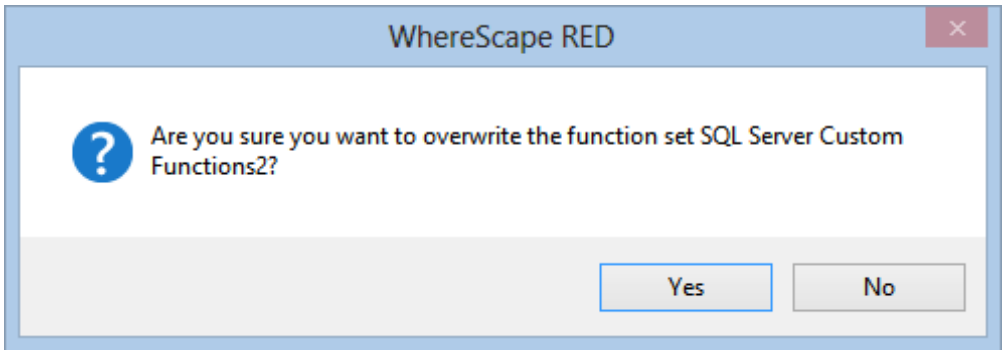


The xml file is validated using the schema definition file at <install directory>\Administrator\Function Sets\Database Function Set.xsd

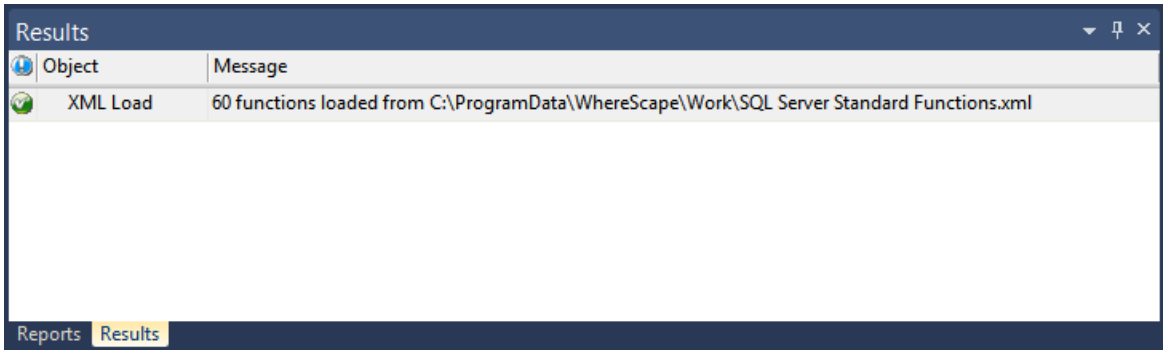
If a function set containing the phrase **Standard Function** is loaded, a warning is displayed:



If an existing function set (not a standard set) is about to be overwritten, a warning is displayed:

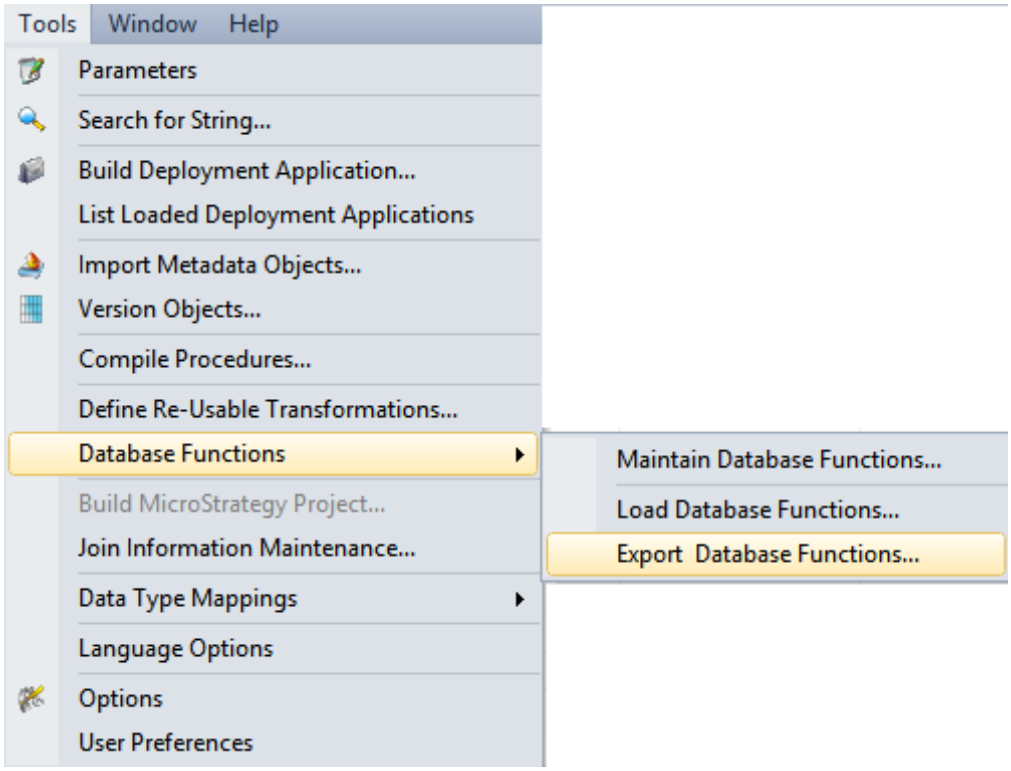


A message is displayed in the results pane.

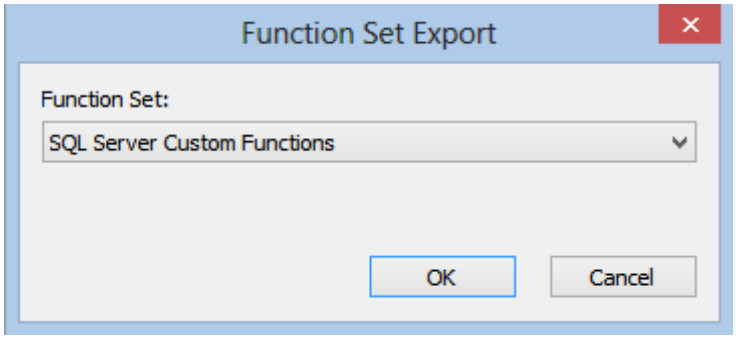

Results
Object
Message
XML Load
60 functions loaded from C:\ProgramData\WhereScape\Work\SQL Server Standard Functions.xml
Reports
Results

EXPORTING DATABASE FUNCTION SETS

To export Database Function Sets, select **Tools/Database Functions/Export Database Functions...**



Select the **Function Set** to export from the drop-down list. Click **OK**.



*-/By default, RED exports the xml file to **ProgramData\WhereScape\Work**, but this can be changed. Change the File name if necessary and click **Save**.