



WHERESCAPE RED TERADATA USER GUIDE

6.9.1.0

WhereScape RED Teradata 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

Copyright © 2002-2017 WhereScape Software Limited. All rights reserved. This document may be redistributed in its entirety and in this electronic or printed form only without permission; all other uses of this document and the information it contains require the explicit written permission of WhereScape Software limited.

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 and WhereScape RED are trademarks or registered trademarks of WhereScape Software Limited. Other brands or product names are trademarks or registered trademarks of their respective companies.

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

CONTENTS

| | |
|--|-----------|
| Overview | 1 |
| Overview of WhereScape RED | 2 |
| How to use this Guide | 5 |
| Design | 7 |
| Objects and Windows | 9 |
| Object Types | 10 |
| Working with Objects | 15 |
| Object Check-Outs and Check-Ins | 39 |
| Re-Create Dialog | 44 |
| Organizing Objects | 47 |
| Adding Objects to Projects | 51 |
| Removing Objects from Projects | 52 |
| Using Project/Object Maintenance | 54 |
| Adding Projects to Groups | 55 |
| Removing Projects from Groups | 55 |
| Moving Projects within Groups | 56 |
| List Projects Memberships for an Object(s) | 57 |
| Windows and Panes | 58 |
| Builder Window | 58 |
| Scheduler Window | 63 |
| Diagram Window | 64 |
| Procedure Editor Window | 66 |
| Export Middle Pane Output | 68 |
| Find Function | 70 |
| Tutorials | 71 |
| Default Settings | 72 |
| Settings - Options | 73 |
| Settings - Repository Identification | 74 |
| Settings - Repository Privacy Settings | 75 |
| Settings - Object Types | 78 |
| Object Type Availability | 78 |
| Object Type Names | 79 |
| Object Type Ordering | 80 |
| Object Type End User Setting | 81 |
| Object Type Icon | 82 |

| | |
|---|------------|
| Object Type Color | 85 |
| Object Sub Types | 86 |
| Settings - Global Naming Conventions | 88 |
| Case Conversion | 88 |
| Global Naming of Tables | 89 |
| Global Naming of Indexes | 90 |
| Global Naming of Key Columns | 91 |
| Global Naming of Procedures | 93 |
| Settings - DSS Tables and Columns | 94 |
| DSS Tables | 96 |
| DSS Columns | 97 |
| Settings - Check-Out and Check-In | 99 |
| Code Generation | 100 |
| General | 100 |
| Default Update Procedure Options | 102 |
| Settings - Storage | 104 |
| Target Location | 104 |
| Table Storage | 107 |
| Default Optional CREATE Clause | 108 |
| Index Type | 109 |
| Settings - Versioning | 110 |
| Settings - Documentation | 111 |
| Settings - Other | 112 |
| Settings - User Preferences | 113 |
| Settings - Common | 114 |
| Look and Feel | 114 |
| Local Naming Conventions | 120 |
| Local Paths | 124 |
| Outputs | 125 |
| Other | 127 |
| Settings - Current Repository | 129 |
| Look and Feel | 129 |
| Settings - Language Options | 130 |
| Parameters | 132 |
| <hr/> | |
| Connections | 133 |
| <hr/> | |
| Connection Types | 134 |
| Database - Data Warehouse/Metadata Repository | 135 |
| General | 136 |
| Source System | 136 |
| Big Data Adapter Settings | 136 |
| Database Credentials | 137 |
| Other | 138 |
| When Connection is an OLAP Data Source | 139 |
| Target Table Location [For target enabled licenses] | 139 |
| Database | 141 |

| | |
|---|------------|
| General | 141 |
| Source System | 142 |
| Database Credentials | 142 |
| Other | 143 |
| Target Table Location [For target enabled licenses] | 143 |
| ODBC | 144 |
| General | 144 |
| ODBC | 145 |
| Big Data Adapter Settings | 145 |
| Credentials | 146 |
| Other | 146 |
| Windows | 148 |
| General | 148 |
| Windows Host | 148 |
| Credentials | 149 |
| Other | 149 |
| UNIX | 151 |
| General | 151 |
| UNIX/Linux Host | 152 |
| Credentials | 153 |
| Other | 153 |
| Hadoop | 155 |
| General | 157 |
| Apache Hadoop | 157 |
| Big Data Adapter Settings | 159 |
| Credentials | 159 |
| Other | 160 |
| To test the drag and drop functionality | 160 |
| Closing the Connection | 160 |
| Microsoft Analysis Server 2005+ | 161 |
| Microsoft Analysis Server 2005+ - OLAP Cubes | 162 |
| Microsoft Analysis Server 2005+ - Tabular Mode | 165 |
| Browsing a Connection | 167 |
| Connection Browse Properties | 170 |
| Changing a Connection's Properties | 172 |
| Reset Meta Database Connections | 172 |
| Configuration Settings for BDA | 173 |
| Configuring the BDA Server | 174 |
| Big Data Adapter Settings | 175 |
| Configuring your database for use by BDA | 175 |
| Big Data Adapter Settings | 177 |
| Table Properties | 179 |
| <hr/> | |
| Properties | 180 |
| Rebuilding Update Procedures | 181 |
| Storage | 183 |
| Table Storage Screen - Teradata | 184 |

| | |
|---|------------|
| Location | 184 |
| Table Storage Screen - Tabular | 186 |
| Location | 186 |
| Other | 187 |
| Processing | 187 |
| Bulk Table Storage Change | 187 |
| Override Create DDL | 189 |
| Source | 190 |
| Documentation Fields | 191 |
| Documentation Fields Screen | 192 |
| Notes | 193 |
| | |
| Loading Data | 194 |
| <hr/> | |
| Choosing the Best Load Method | 196 |
| Load Drag and Drop | 197 |
| Data Type Mappings | 199 |
| Database Link Load | 200 |
| Database Link Load - Properties | 200 |
| Database Link Load - Source Screen | 202 |
| ODBC Based Load | 204 |
| Native ODBC Based Load | 205 |
| Native ODBC Based Load - Source Screen | 205 |
| File Actions | 208 |
| Native Loads using UNIX and LINUX | 210 |
| TPT Load | 213 |
| TPT Load - Source Screen | 215 |
| Cleanup after TPT Load Failure | 218 |
| TPT UNIX/Linux Script Load | 220 |
| TPT UNIX Script Load - Properties | 221 |
| SSIS Loader | 224 |
| Loading Data into RED Load Tables using SSIS | 225 |
| Flat File Loads | 235 |
| Loading Data from Flat Files using SSIS | 241 |
| Flat File Load - Source Screen | 251 |
| Source File Details | 252 |
| Trigger File Details | 253 |
| Load Configuration | 253 |
| Archived File Details | 256 |
| SQL Server Integration Services (SSIS) | 257 |
| Script based loads | 258 |
| XML File Load | 260 |
| External Load | 265 |
| Apache Sqoop Load | 265 |
| Handling Missing Source Columns | 272 |
| Load Table Transformations | 275 |

| | |
|---|------------|
| Post-Load Procedures | 275 |
| Changing Load Connection and Schema | 276 |
| Dimensions | 279 |
| <hr/> | |
| Dimensions Overview | 280 |
| Building a Dimension | 281 |
| Drag and Drop | 281 |
| Dimension Properties | 284 |
| Create and Load | 284 |
| Deleting and Changing columns | 285 |
| Adding additional columns | 286 |
| Manually adding previous value columns | 287 |
| Create the table | 288 |
| Generating the Dimension Update Procedure | 289 |
| Generating a Procedure | 289 |
| Processing tab | 290 |
| Source tab | 294 |
| Joining multiple source tables | 295 |
| Using Change Detection - Change Detection Tab | 296 |
| Building and Compiling the Procedure | 299 |
| Indexes | 300 |
| Dimension Artificial Keys | 301 |
| To allow for non identity surrogate keys on Dimensions: | 302 |
| Dimension Column Properties | 303 |
| Changing a Column Name | 308 |
| Dimension Column Transformations | 311 |
| Dimension Hierarchies | 312 |
| Adding a Dimension Hierarchy | 313 |
| Using a Maintained Hierarchy | 315 |
| Snowflake | 315 |
| Creating a Snowflake | 315 |
| Dimension Language Mapping | 317 |
| Staging | 318 |
| <hr/> | |
| Building the Stage Table | 319 |
| Generating the Staging Update Procedure | 322 |
| Generating a Procedure | 322 |
| Procedure type | 322 |
| Locking Request Modifier | 323 |
| Source Table Mapping | 324 |
| Parameter selection | 327 |
| Model/Dimension Joins | 328 |
| Model history information | 329 |
| Building and Compiling the Procedure | 330 |

| | |
|---|------------|
| Stage Table Custom Procedure | 331 |
| Stage Table Column Properties | 331 |
| Stage Table Column Transformations | 337 |
| Permanent Stage Tables | 338 |
| Generating the Permanent Staging Update Procedure | 339 |
| Set Merge Procedure | 345 |
| Data Store Objects | 351 |
| <hr/> | |
| Data Store Objects Overview | 352 |
| Building a Data Store Object | 354 |
| Drag and Drop | 354 |
| Data Store Object Properties | 354 |
| Create and Load | 355 |
| Deleting and Changing columns | 357 |
| Adding additional columns | 357 |
| Create the table | 358 |
| Generating the Data Store Update Procedure | 359 |
| Generating a Procedure | 359 |
| Processing tab | 359 |
| Source tab | 364 |
| Building and Compiling the Procedure | 365 |
| Data Store Artificial Keys | 367 |
| To manually add an extra artificial key column to a Data Store table: | 367 |
| Allowing for non identity surrogate keys on Data Store tables: | 370 |
| Data Store Column Properties | 371 |
| Data Store Column Transformations | 376 |
| EDW 3NF Tables | 377 |
| <hr/> | |
| EDW 3NF Tables Overview | 378 |
| Building EDW 3NF Table | 380 |
| Generating the EDW 3NF Update Procedure | 384 |
| Generating a Procedure | 384 |
| Processing tab | 384 |
| Source tab | 388 |
| Indexes | 389 |
| Converting an existing EDW 3NF Table to a EDW 3NF History Table | 390 |
| EDW 3NF Table Artificial Keys | 392 |
| To manually add an extra artificial key column to an EDW 3NF table: | 392 |
| Artificial keys set via a non identity column: | 393 |
| Allowing for non identity surrogate keys on EDW 3NF tables: | 394 |
| EDW 3NF Table Column Properties | 395 |
| EDW 3NF Table Column Transformations | 400 |
| Data Vaults | 402 |
| <hr/> | |

| | |
|---|------------|
| Data Vault Functions and Features | 403 |
| Load Table Meta Data Columns | 403 |
| Data Vault Stage Table | 404 |
| Hash Key Generation Wizard | 406 |
| Hub, Link and Satellite Creation Wizard | 407 |
| Data Vault Templates | 408 |
| Data Vault Settings | 409 |
| Object Types settings: | 409 |
| Global Naming Conventions settings: | 410 |
| DSS Tables and Columns settings: | 411 |
| Table Column Properties | 412 |
| Maintain Hash Key Columns | 414 |
| Building Data Vault Objects | 415 |
| Creating Load Tables | 415 |
| Creating Data Vault Stage Tables | 417 |
| Generating Update Procedures for the Data Vault Stage Table | 427 |
| Creating the Hub, Link and Satellite Tables | 432 |
| Creating the Hub table | 432 |
| Creating the Link table | 437 |
| Creating the Satellite table | 440 |
| Generating Update Procedures for Hub, Link and Satellite Tables | 444 |
| Hub table | 444 |
| Link and Satellite Tables | 448 |
| | |
| Custom Objects | 449 |
| <hr/> | |
| Model Tables | 450 |
| <hr/> | |
| Model Table Overview | 451 |
| Building a Model Table | 453 |
| Drag and Drop | 453 |
| Model Table Properties | 453 |
| Create and Load | 454 |
| Deleting and Changing columns | 455 |
| Adding additional columns | 455 |
| Create the table | 456 |
| Generating the Model Table Update Procedure | 458 |
| Generating a Procedure | 458 |
| Business Key definition | 458 |
| Locking Request Modifier | 460 |
| Source Table Mapping | 460 |
| Building and Compiling the Procedure | 463 |
| Model Table Artificial Keys | 465 |
| Model Table Custom Procedure | 466 |
| Model History Tables | 466 |
| Generating History Table Update Procedures | 468 |

| | |
|---|------------|
| Model Table Column Properties | 471 |
| Model Table Column Transformations | 476 |
| Fact Tables | 477 |
| <hr/> | |
| Detail Fact Tables | 478 |
| Creating Detail Fact Tables | 478 |
| Generating the Detail Fact Update Procedure | 479 |
| Generating a Procedure | 480 |
| Define Fact Procedure Type and Options | 480 |
| Template | 480 |
| Define Fact Business Key Columns | 481 |
| Source Tab | 484 |
| Fact Table Column Properties | 486 |
| Fact Table Column Transformations | 492 |
| Fact Table Language Mapping | 493 |
| Aggregation | 494 |
| <hr/> | |
| Creating an Aggregate Table | 495 |
| Creating an Aggregate Summary Table | 496 |
| Aggregate Table Column Properties | 496 |
| Aggregate Table Column Transformations | 502 |
| Join Indexes | 503 |
| <hr/> | |
| Creating a Join Index | 504 |
| Views | 508 |
| <hr/> | |
| One to One Views | 509 |
| Model Views for Aliasing | 512 |
| Compound Views, Facts and Dimensions | 515 |
| Dimension View Hierarchies | 519 |
| Adding a Dimension View Hierarchy | 519 |
| Creating a Custom View | 521 |
| View Aliases | 523 |
| Analysis Services OLAP Cubes | 525 |
| <hr/> | |
| OLAP Overview | 526 |
| OLAP Defining the Data Source for the OLAP Cube | 527 |
| OLAP Defining an OLAP Cube | 529 |
| Building a New OLAP Cube | 529 |
| Setting Cube Properties | 532 |
| OLAP Inspecting and Modifying Advanced Cube Properties | 534 |

| | |
|---|------------|
| OLAP Creating an OLAP Cube on the Analysis Services Server | 535 |
| OLAP Cube Objects | 536 |
| OLAP Cube Properties | 536 |
| OLAP Cube Measure Groups | 542 |
| OLAP Cube Measure Group Processing/Partitions | 544 |
| OLAP Cube Measure Group Partitions | 549 |
| OLAP Cube Measures | 552 |
| OLAP Cube Calculations | 555 |
| OLAP Cube Key Performance Indicators | 558 |
| OLAP Cube Actions | 561 |
| OLAP Cube Dimensions | 565 |
| OLAP Cube Measure Group Dimensions | 567 |
| OLAP Dimension Objects | 571 |
| OLAP Dimension Overview | 571 |
| OLAP Dimension Attributes | 576 |
| OLAP Dimension Attribute Relationships | 580 |
| OLAP Dimension Hierarchies | 582 |
| OLAP Dimension User Defined Hierarchy Levels | 584 |
| OLAP Changing OLAP Cubes | 586 |
| OLAP Retrofitting an OLAP Object | 588 |
| | |
| Transformations | 593 |
| <hr/> | |
| Column Transformations | 594 |
| Column Transformation Properties | 595 |
| Load Table Column Transformations | 598 |
| Database Link During Load Transformations | 599 |
| File During Load Transformations | 602 |
| After Load Transformations | 602 |
| Teradata User Defined Functions | 603 |
| Teradata UDF Example | 604 |
| Re-usable Transformations | 606 |
| Creating a New Re-usable Transformation | 606 |
| Specify the Name of the Transformation | 607 |
| Enter Re-usable Transformation Metadata | 608 |
| Define the Transformation Model | 609 |
| Completed Re-usable Transformation | 611 |
| Changing a Re-usable Transformation | 612 |
| Applying Changes to Dependant Transformations | 614 |
| Using Re-usable Transformations | 615 |
| | |
| Exporting Data | 616 |
| <hr/> | |
| Building an Export Object | 617 |
| File Attributes | 621 |
| File Attributes - SSIS Exports | 624 |

| | |
|---|------------|
| Export File Definition | 625 |
| SQL Server Integration Services (SSIS) | 625 |
| Export Column Properties | 626 |
| Script based Exports | 628 |
| Procedures and Scripts | 630 |
| Procedure Generation | 632 |
| Procedure Editing | 638 |
| Procedure Loading and Saving | 641 |
| Procedure Comparisons | 644 |
| Procedure Compilation | 645 |
| Procedure Running | 646 |
| Procedure Syntax | 646 |
| Procedure Properties | 647 |
| Macros | 650 |
| BTEQ Scripts | 650 |
| Script Generation | 651 |
| Script Generation (Windows/Teradata) | 651 |
| 24.11.1.1 Windows PowerShell Scripts | 657 |
| Script Editing | 660 |
| Script Testing | 662 |
| Script Syntax | 662 |
| Script Environment Variables | 665 |
| Calling a Batch File from a Script | 671 |
| Scheduling Scripts | 675 |
| Manually created scripts | 677 |
| Templates | 678 |
| Template Properties | 680 |
| Template Editor | 681 |
| Evaluating an API Outline Template | 682 |
| Template Usage | 686 |
| Windows PowerShell Templates | 687 |
| Scheduler | 688 |
| Scheduler Options | 689 |
| Auto | 692 |
| Tools | 693 |
| Select Job Report Fields | 693 |
| Scheduler States | 695 |
| Scheduling a Job | 698 |
| Working with Jobs | 703 |
| Creating a Job | 706 |

| | |
|--|------------|
| Editing a Job | 718 |
| Editing Tasks in a Job | 722 |
| Editing Task Dependencies | 729 |
| Show Dependencies Diagram | 734 |
| Inserting a Copy of a Job | 736 |
| Deleting a Job | 737 |
| Deleting Job Logs | 739 |
| Starting a Job | 741 |
| Halting a Job | 742 |
| Aborting a Job | 743 |
| Restarting a Job | 744 |
| Creating an Application from a Job | 749 |
| Stand Alone Scheduler Maintenance | 752 |
| SQL to return Scheduler Status | 755 |
| Reset Columns in Job and Task View | 756 |
| Stopping a Linux/UNIX Scheduler from within RED | 757 |
| | |
| Indexes | 759 |
| <hr/> | |
| Index Definition | 760 |
| | |
| Documentation and Diagrams | 765 |
| <hr/> | |
| Creating Documentation | 766 |
| Batch Documentation Creation | 769 |
| Reading the Documentation | 770 |
| Diagrams | 771 |
| Types of Diagrams | 771 |
| Schema Diagram | 773 |
| Source Diagram | 775 |
| Joins Diagram | 779 |
| Links Diagram | 780 |
| Impact Diagram | 781 |
| Dependency Diagram | 783 |
| Working with Diagrams | 786 |
| Creating a Job from a Diagram | 789 |
| Creating an Application from a Diagram | 791 |
| Creating a Project from a Diagram | 794 |
| | |
| Reports | 796 |
| <hr/> | |
| Dimension-Fact Matrix | 797 |
| OLAP Dimension-Cube Matrix | 798 |
| Model Views for a Specified Model | 800 |
| Column Reports | 801 |
| Columns without Comments | 801 |
| All Column Transformations | 802 |

| | |
|--|------------|
| Re-Usable Column Transformations | 804 |
| Column Track-Back | 805 |
| Column Track-Forward | 807 |
| Table Reports | 809 |
| Tables without Comments | 809 |
| Load Tables by Connection | 810 |
| Export Objects by Connection | 812 |
| Records that failed a Dimension Join | 813 |
| External Source Tables/files | 814 |
| Procedure Reports | 816 |
| Modified Procedures | 816 |
| Custom Procedures | 817 |
| Index Reports | 819 |
| Modified Indexes | 819 |
| Object Reports | 820 |
| Objects-Projects Matrix | 820 |
| Modified Objects (excluding indexes) | 822 |
| Objects Checked-out | 824 |
| Loaded or Imported Objects | 825 |
| Job Reports | 827 |
| Object-Job Matrix | 827 |
| Jobs with an Object | 829 |
| Tasks of a Job | 830 |
| Operational Reports | 831 |
| Object Performance History | 831 |
| Job Performance History | 834 |
| Task Performance History | 835 |
| Validate | 837 |
| <hr/> | |
| Validate Meta-data | 838 |
| Validate Workflow Data | 838 |
| Validate Table Create Status | 838 |
| Validate Load Table Status | 839 |
| Validate Procedure Status | 839 |
| List Meta-data Tables not in the Database | 839 |
| List Database Tables not in the Meta-data | 841 |
| List Tables with no related Procedures or Scripts | 843 |
| List Procedures not related to a Table | 844 |
| Compare Meta-data Repository to another | 846 |
| Compare Meta-data Indexes to Database | 849 |
| Teradata: View of Model Validate | 851 |
| List Duplicate Business Key Columns | 853 |
| Query Data Warehouse Objects | 854 |

| | |
|---|------------|
| Promoting Between Environments | 856 |
| Applications | 857 |
| Application Creation | 858 |
| Creating an Application | 858 |
| Define an Application distribution | 858 |
| Output Directory | 859 |
| Objects to Add/Replace | 862 |
| Objects to Delete | 863 |
| Application Loading | 864 |
| Creating and Loading Applications from the Command Line | 864 |
| Importing Object Metadata | 865 |
| Importing Language Files | 867 |
| Data Warehouse Testing | 868 |
| | |
| Backing Up and Restoring Metadata | 872 |
| Backup using DB Routines | 873 |
| Restoring DB Backups | 875 |
| Unloading Metadata | 876 |
| Loading an Unload | 878 |
| | |
| Altering Metadata | 882 |
| Validating Tables | 883 |
| Validating Source (Load) Tables | 885 |
| Validating Procedures | 886 |
| Altering Tables | 887 |
| Validating Indexes | 889 |
| Recompiling Procedures | 889 |
| | |
| Callable Routines | 891 |
| Introduction to Callable Routines | 892 |
| Callable Routines API | 892 |
| Callable Routines per RDBMS | 894 |
| Callable Routines Names Qualifier | 896 |
| Callable Routines Common Input | 896 |
| Callable Routines Invocation | 898 |
| Alternative Invocation Methods | 898 |
| Ws_Api_Glossary | 900 |
| Ws_Connect_Replace | 902 |
| Ws_Job_Abort | 905 |
| Ws_Job_Clear_Archive | 906 |
| Ws_Job_Clear_Logs | 908 |
| Ws_Job_Clear_Logs_By_Date | 911 |

| | |
|--|-------------|
| Ws_Job_Create | 914 |
| Ws_Job_CreateWait | 918 |
| Ws_Job_Dependency | 922 |
| Ws_Job_Release | 925 |
| Ws_Job_Restart | 928 |
| Ws_Job_Schedule | 931 |
| Ws_Job_Status | 934 |
| Ws_Load_Change | 939 |
| Ws_Maintain_Indexes | 943 |
| Ws_Version_Clear | 946 |
| WsParameterRead | 949 |
| WsParameterReadF | 950 |
| WsParameterReadG | 952 |
| WsParameterWrite | 954 |
| WsWrkAudit | 955 |
| WsWrkAuditBulk | 957 |
| WsWrkError | 961 |
| WsWrkErrorBulk | 963 |
| WsWrkTask | 967 |
| Ws_admin_v Views | 970 |
| Ws_admin_v_audit | 971 |
| Ws_admin_v_error | 971 |
| Ws_admin_v_sched | 972 |
| Ws_admin_v_task | 973 |
| Retrofitting | 975 |
| Migrating the Data Warehouse Database Platform | 976 |
| Importing a Data Model | 986 |
| Re-Targeting Source Tables | 993 |
| Retro Column Properties | 995 |
| Retro Column Properties Screen | 999 |
| Retro Column Transformations | 1000 |
| Integrating WhereScape RED into an Existing Warehouse | 1001 |
| Rebuilding | 1002 |
| Integrating | 1003 |
| Integrating, Host Scripts | 1004 |
| Integrating, Selecting a Table Type | 1006 |
| Integrating, Questions | 1006 |
| Integrating, Procedures | 1010 |
| Integrating, Views | 1011 |
| Integrating, WhereScape Tables | 1011 |

| | |
|---|-------------|
| Relationship Maintenance | 1012 |
| Add Relationship | 1012 |
| List Relationships | 1015 |
| Generate Relationships | 1016 |
| Upgrading RED | 1017 |
| Login Checks | 1018 |
| Data Type Mappings | 1021 |
| Using Data Type Mapping Sets | 1022 |
| Maintaining Data Type Mapping Sets | 1024 |
| Creating a New Data Type Mapping Set | 1025 |
| Copying a Data Type Mapping Set | 1029 |
| Editing a Data Type Mapping Set | 1033 |
| Deleting a Data Type Mapping Set | 1043 |
| Loading Data Type Mapping Sets | 1044 |
| Exporting Data Type Mapping Sets | 1046 |
| Data Type Mapping Examples | 1048 |
| Column Context Menu | 1057 |
| Properties | 1059 |
| Change Column(s) | 1062 |
| Add Column | 1064 |
| Duplicate Column | 1065 |
| Delete Column | 1067 |
| Re-space Order Number | 1068 |
| Impact | 1069 |
| Sync Column order with database | 1072 |
| Send Columns to Another Object | 1073 |
| Database Functions | 1076 |
| Using Database Function Sets | 1077 |
| Maintaining Database Function Sets | 1080 |
| Creating a New Database Function Set | 1083 |
| Copying a Database Function Set | 1086 |
| Editing a Database Function Set | 1089 |
| Deleting a Database Function Set | 1101 |
| Loading Database Function Sets | 1103 |
| Exporting Database Function Sets | 1106 |

| | |
|--------------------------|-------------|
| Gather Statistics | 1108 |
| Define Statistics | 1109 |

CHAPTER 1 OVERVIEW

WhereScape RED User Guide for Teradata

The WhereScape RED **User Guide** for Teradata 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.

IN THIS CHAPTER

| | |
|----------------------------------|---|
| Overview of WhereScape RED | 2 |
| How to use this Guide | 5 |

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 an entire enterprise data warehouse or just the user facing views, aggregates and summaries.

In all cases, the core values of WhereScape RED are twofold: its rapid 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, 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.

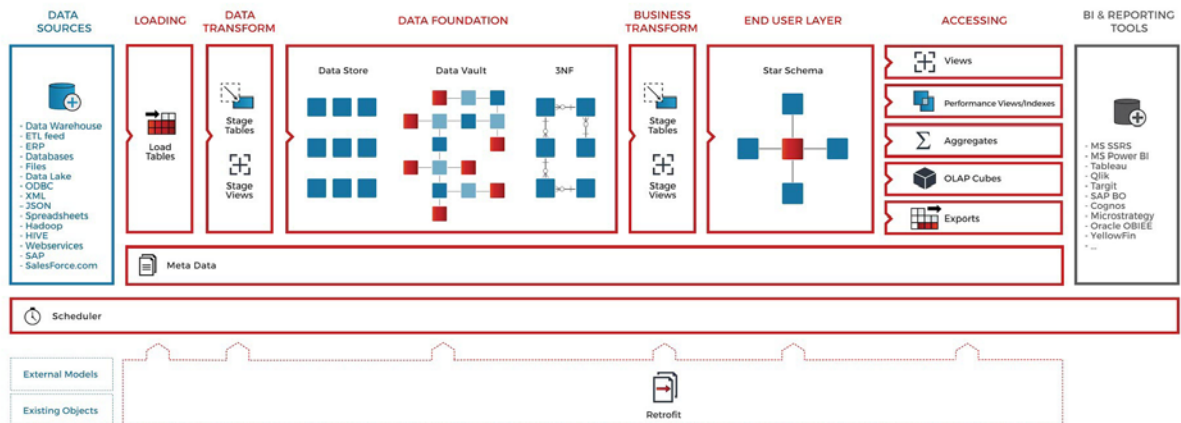
The basic concepts behind WhereScape's Pragmatic Data Warehousing Methodology are:

- 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 views or cubes for end user access
- allow for incremental loads from day one
- use an iterative approach
- minimize cleansing and transformations to ease source system reconciliation
- design the data warehouse independently from the end user tool layer

WhereScape RED supports these concepts to facilitate very rapid building 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.

The diagram below shows the objects and the information flow:



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 tables) are created and updated from stage tables. Second tier tables (model, summary rollup, aggregate, join indexes, etc.) are created and updated from lower level tables.

Feeds for downstream systems (exports) can be created by dragging and dropping from a model, view or aggregate object. Fast export or parallel transporter are used to generate file exports.

Cubes can be created from transactional tables or views.

Procedural code

WhereScape RED for Teradata generates procedural code using Teradata database procedures 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 if a problem occurs, 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.

WhereScape RED and 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 most 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 Load, Stage or Model table from which WhereScape RED builds more advanced data warehouse structures.

HOW TO USE THIS GUIDE

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 for Teradata

The WhereScape RED **User Guide** for Teradata 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 for Teradata

The WhereScape RED **Tutorials** for Teradata 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:

- Populating a predefined EDW 3NF data warehouse
- Building a simple EDW 3NF data warehouse
- Creating permanent staging tables, history tables, aggregate tables
- Building a simple presentation layer
- Scheduling the data warehouse objects
- Fine tuning the data warehouse by adding Analysis Services cubes

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 2

DESIGN

Design Introduction

WhereScape RED for Teradata 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 3

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, Model 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


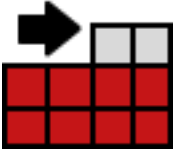
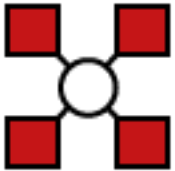

| | |
|--------------------------------|----|
| Object Types..... | 10 |
| Working with Objects..... | 14 |
| Organizing Objects..... | 47 |
| Windows and Panes..... | 58 |
| Export Middle Pane Output..... | 67 |
| Find Function | 70 |


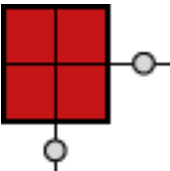

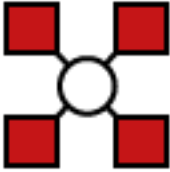
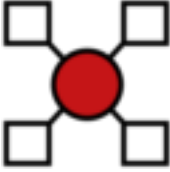

OBJECT TYPES

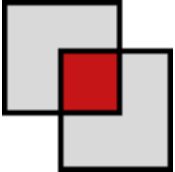




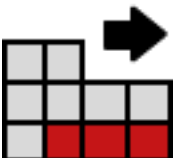
WhereScape RED has a concept of objects which are combined to build the data warehouse. Each WhereScape RED object has properties that allow the data warehouse developer to change how the object is used.







Note: Some objects 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 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 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 |
|--|--|
| <p>Stage Table</p>  | <p>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 speed of the data set. 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.</p> |
| <p>EDW 3NF Table</p>  | <p>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.</p> |
| <p>Data Store Table</p>  | <p>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.</p> |
| <p>Model Table</p>  | <p>Model objects are used to create EDW 3NF models in an enterprise data warehouse. They may contain surrogate keys to other model tables.</p> |
| <p>Fact Table</p>  | <p>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.</p> |
| <p>Aggregate</p>  | <p>The aggregate object type provides a means to speed up access by summarizing data to a higher grain.</p> |

| Object Type | Purpose |
|---|---|
| Join index  | Join index is a Teradata specific object used for performance across multiple tables. |
| 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 cube object type provides a means to develop and manage Microsoft Analysis Services cubes. Cubes would normally be built from fact tables and provide summarized data. |
| 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. |
| Index  | This object type defines database indexes used to improve the access times on any of the table object types (i.e. Load, Stage, Model and Aggregate). |
| Export  | Exports are used to manage exports from the data repository. |

| Object Type | Purpose |
|--|---|
| 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. |
| 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. |
| Template  | <p>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.</p> <p>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.</p> <p>Utility type templates can contain common code for use by other templates.</p> |
| 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 should 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.

It should be noted 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 stage, model 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 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.

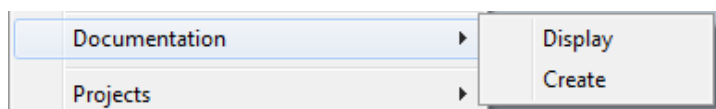
The operations of each of the objects is discussed in the following chapters. A brief overview of some of the more common operations follows:

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 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**:

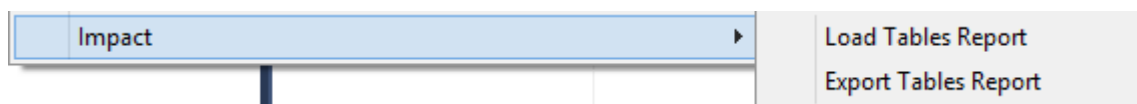


The **Projects** menu option can be used to remove the connection from the current project or to add the connection to the current project. The **List Projects** option displays a list of projects which contain the current object, results are shown in the bottom pane. Multiple objects can be selected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

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

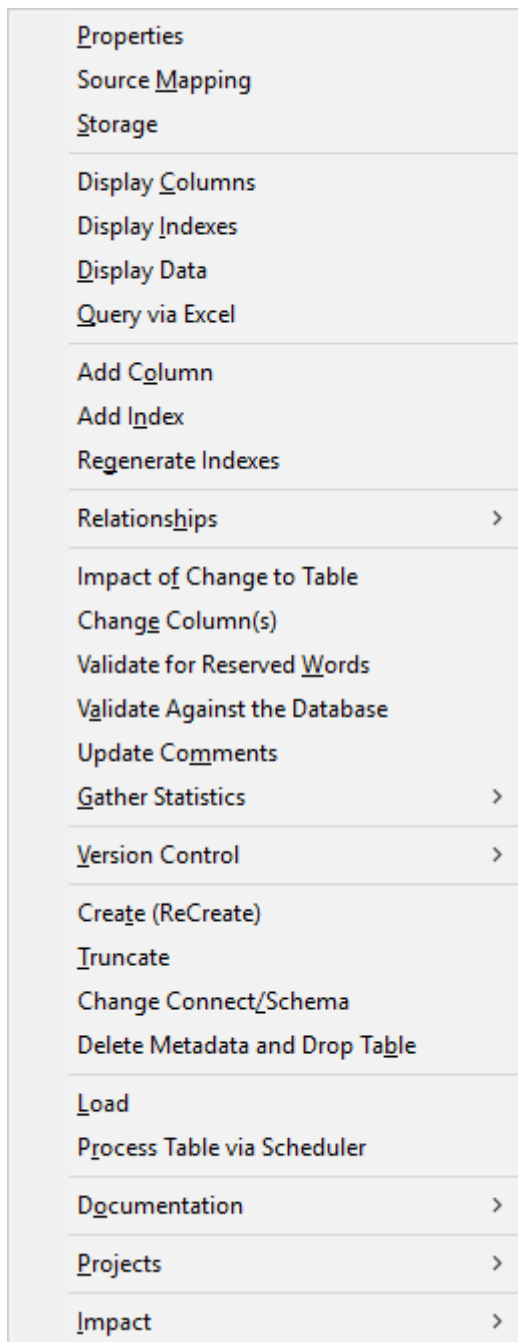


The **Impact** menu enables you to run reports on load and export objects associated with the connection.



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 must be used if more detailed data analysis is required. Query the columns in Excel using **Query via Excel**.

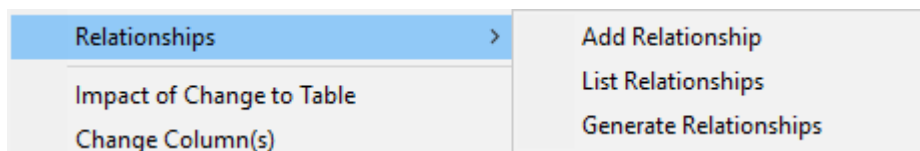


TIP: When a column list has been displayed in the central pane, it is sorted based on the **order** field associated with each column. Clicking the column label **Col name** will sort the columns into alphabetic order. A subsequent click will re-sort based on the **order** field.

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.

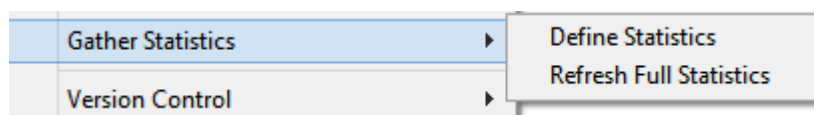
Use **Gather Statistics** to gather statistics on a table. This action will enable the underlying database to optimize each query based on the statistics collected about the data that is being accessed.

Users can either chose to **Define Statistics** or to **Refresh Full 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 see **Editing Tasks** (see "**Editing Tasks in a Job**" on page 722).

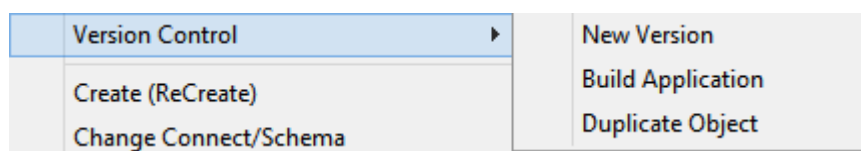
The statistics process from the object context menu for Teradata tables is described below:

| Teradata tables | |
|-------------------------|--|
| Define Statistics | Using the DlgDefineStats dialog, we save the DDL and execute it. This is object specific. |
| Refresh Full Statistics | COLLECT STATISTICS ON database.table (this requires statistics to have been executed on the object). |



WhereScape RED Tip: To Define Statistics on a Table Object see also **Define Statistics** (see "**Gather Statistics**" on page 1108).

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** menu option allows you to build an application file for the load table and the **Duplicate Object** menu 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 en-bulk for a number of load tables. See the **Load changing connections** section under loading data.

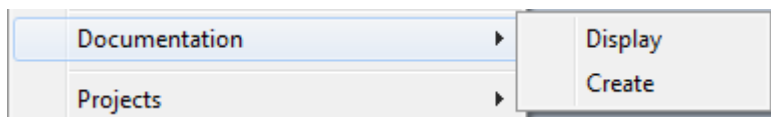
The **Delete metadata and drop table** menu option deletes the metadata definition for the table. It also gives you the option to drop the table in the database (dropping the table in the database is the default option). This is a permanent delete and no recovery is provided, so please use with caution. 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.

Note: 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**:

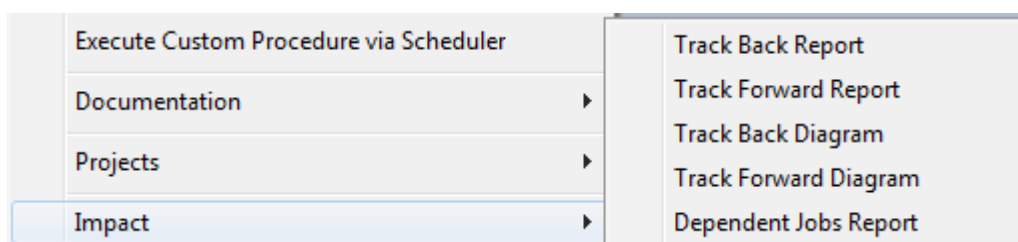


The **Projects** menu option can be used to remove the load table from the current project or to add the load table to the current project. If there aren't any projects in the repository, these two options are unavailable.

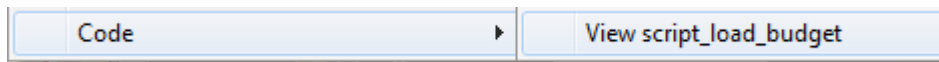


The **Impact** menu option enables you to produce a number of reports and diagrams:

- **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:

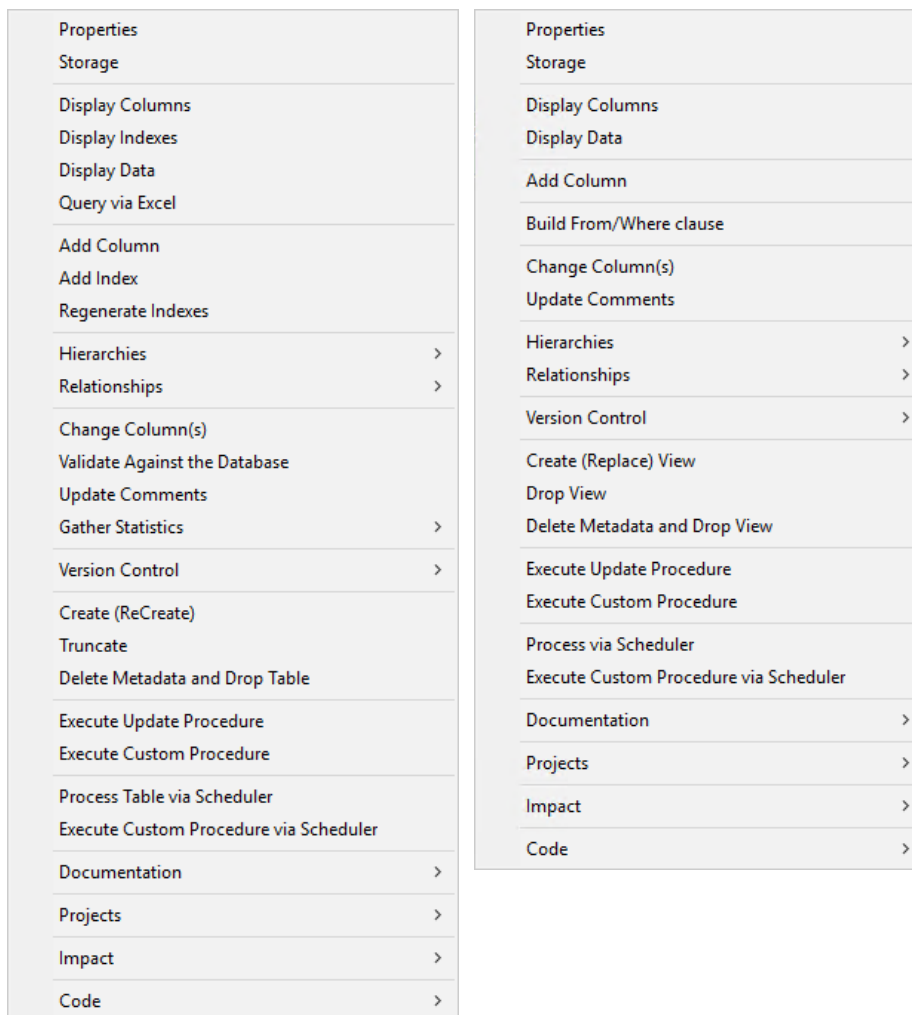


Choose a procedure from the list to open in the procedure editor in view mode.

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

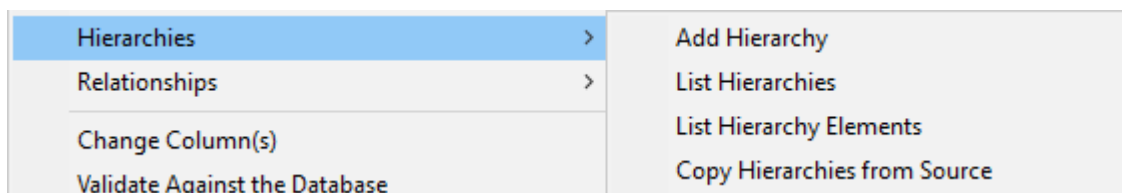
Dimension Tables

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** feature 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 **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 **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:

| | | |
|----------|---|------------------------------|
| Projects | ▶ | View update_dim_customer |
| Impact | ▶ | View get_dim_customer_key |
| Code | ▶ | Rebuild update_dim_customer |
| | | Rebuild get_dim_customer_key |

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:

| | |
|--|---|
| Properties | |
| Storage | |
| Display Columns | |
| Display Indexes | |
| Display Data | |
| Query via Excel | |
| Add Column | |
| Add Index | |
| Regenerate Indexes | |
| Update Comments | |
| Hierarchies | > |
| Relationships | > |
| Change Column(s) | |
| Validate Against the Database | |
| Gather Statistics | > |
| Version Control | > |
| Create (ReCreate) | |
| Truncate | |
| Delete Metadata and Drop Table | |
| Execute Update Procedure | |
| Execute Custom Procedure | |
| Process Table via Scheduler | |
| Execute Custom Procedure via Scheduler | |
| Documentation | > |
| Projects | > |
| Impact | > |
| Code | > |

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:

| | | |
|----------|---|------------------------------|
| Projects | ▶ | View update_ds_product |
| Impact | ▶ | View custom_ds_product |
| Code | ▶ | Rebuild update_ds_product |
| | | Regenerate update_ds_product |

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:

| | |
|--|---|
| Properties | |
| Storage | |
| Display Columns | |
| Display Indexes | |
| Display Data | |
| Query via Excel | |
| Add Column | |
| Add Index | |
| Regenerate Indexes | |
| Update Comments | |
| Hierarchies | > |
| Relationships | > |
| Change Column(s) | |
| Validate Against the Database | |
| Gather Statistics | > |
| Version Control | > |
| Create (ReCreate) | |
| Truncate | |
| Delete Metadata and Drop Table | |
| Execute Update Procedure | |
| Execute Custom Procedure | |
| Process Table via Scheduler | |
| Execute Custom Procedure via Scheduler | |
| Documentation | > |
| Projects | > |
| Impact | > |
| Code | > |

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

Stage, Model and Aggregate Tables

All of these table types have a similar pop-up menu. The standard **stage table** menu is as follows:

| | |
|--|---|
| Properties | |
| Storage | |
| Display Columns | |
| Display Indexes | |
| Display Data | |
| Query via Excel | |
| Report Zero Keys... | |
| Add Column | |
| Add Index | |
| Regenerate Indexes | |
| Relationships | > |
| Change Column(s) | |
| Validate Against the Database | |
| Update Comments | |
| Gather Statistics | > |
| Maintain DV Hash Key Columns... | |
| Version Control | > |
| Create (ReCreate) | |
| Truncate | |
| Delete Metadata and Drop Table | |
| Execute Update Procedure | |
| Execute Custom Procedure | |
| Process Table via Scheduler | |
| Execute Custom Procedure via Scheduler | |
| Documentation | > |
| Projects | > |
| Impact | > |
| Code | > |

The standard **model** menu is as follows:

| | |
|--|---|
| Properties | |
| Storage | |
| Display Columns | |
| Display Indexes | |
| Display Data | |
| Query via Excel | |
| Add Column | |
| Add Index | |
| Regenerate Indexes | |
| Hierarchies | ▶ |
| Change Column(s) | |
| Validate Against the Database | |
| Update Comments | |
| Gather Statistics | ▶ |
| Version Control | ▶ |
| Create (ReCreate) | |
| Truncate | |
| Delete Metadata and Drop Table | |
| Execute Update Procedure | |
| Execute Custom Procedure | |
| Process Table via Scheduler | |
| Execute Custom Procedure via Scheduler | |
| Documentation | ▶ |
| Projects | ▶ |
| Impact | ▶ |
| Code | ▶ |

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:

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 **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

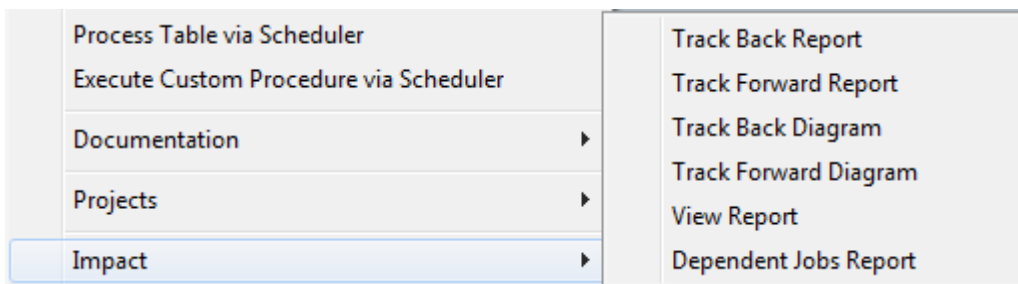
option is only intended for use when working with small/prototype data volumes, and no index handling is performed.

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

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

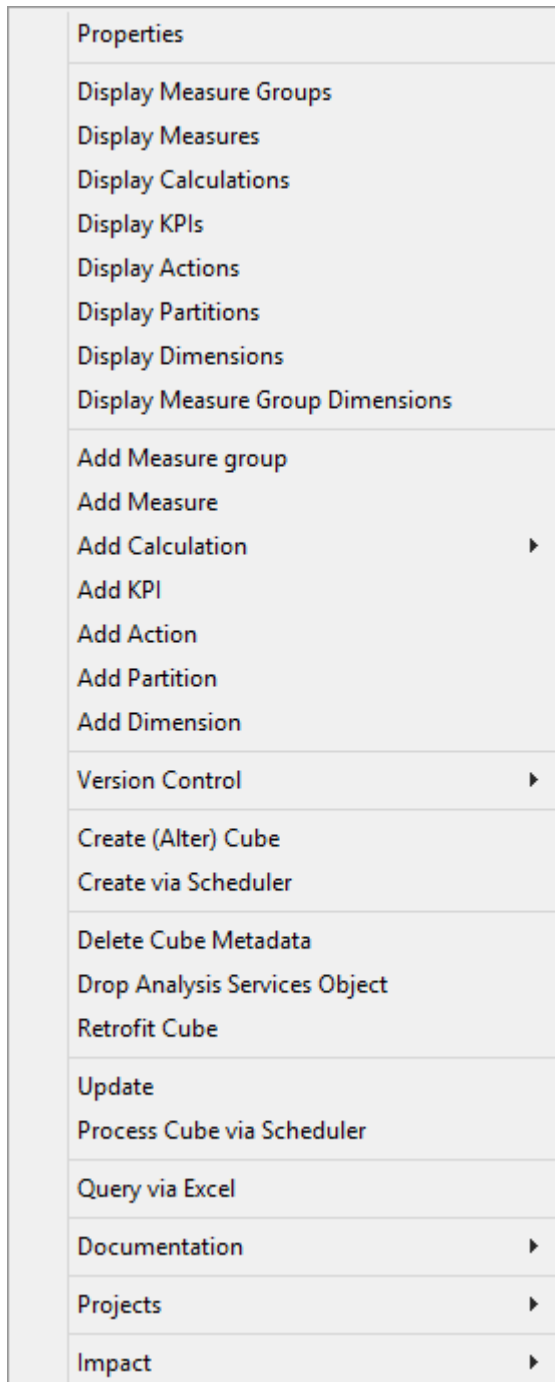
- **Track Back, Track Forward** or **Dependent Jobs** report
- **Track Back** and **Track Forward** diagrams

For **model tables**, the impact menu has a further option of the **View Report**; as seen below:



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 middle pane.

The **Display Measures** menu option lists the measures associated with the measure groups in the cube. This is the default view in the middle 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 middle 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.

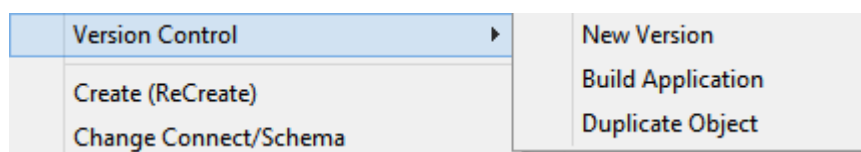
The **Add KPI** menu option adds a new KPI to the cube.

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 to 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.

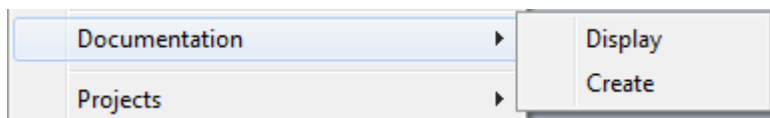
The **Update** menu option processes the cube in Analysis Services interactively from the WhereScape RED interface.

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**:

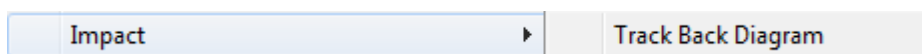


The **Projects** menu option can be used to remove the OLAP cube from the current project or to add the OLAP cube to the current project. The **List Projects** option displays a list of projects which contain the current object, results are shown in the bottom pane. Multiple objects can be selected by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

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

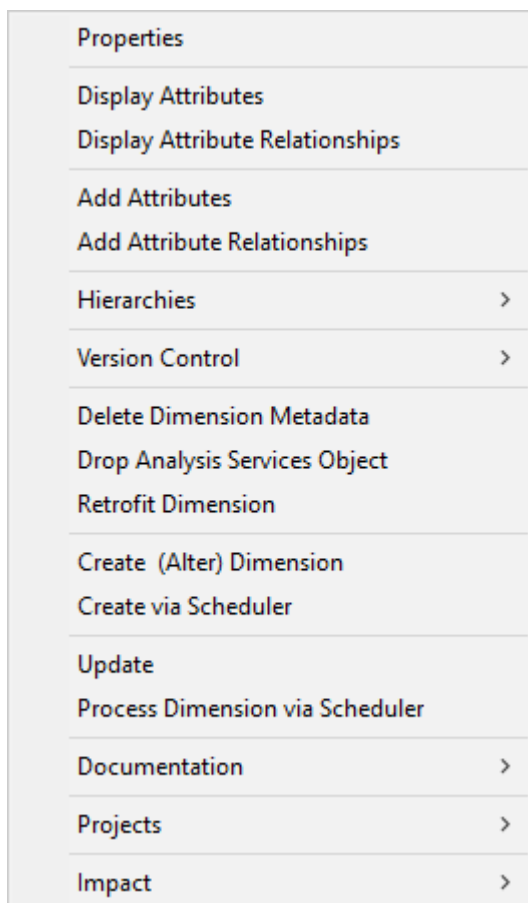


The **Impact** menu option enables you to produce a **Track Back** diagram on the OLAP cube.



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 **Display hierarchies** menu option lists the hierarchies associated with the selected dimension.

The **Display hierarchy levels** menu option lists all the levels for all hierarchies for that dimension.

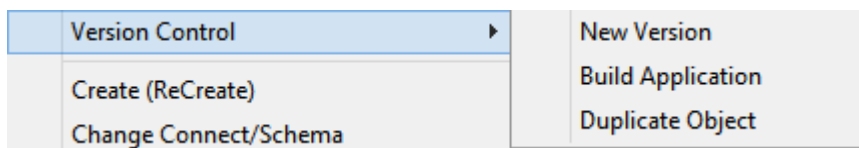
The **Add Attributes** menu option adds a new attribute to the dimension.

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

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

The **Add hierarchy level** menu option adds a level to a hierarchy.

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.

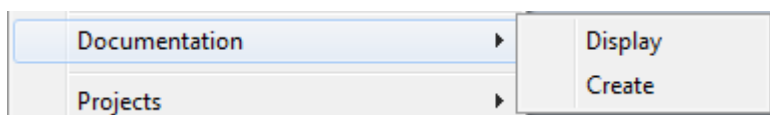
The **Create (Alter) Dimension** menu option creates the OLAP Dimension and supporting objects in Analysis Services (including cube database and dsv) based on the definition in WhereScape RED. This option requires connection and cube database information populated in the OLAP Dimension properties.

The **Create via Scheduler** menu option submits a create of the OLAP Dimension to 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**:

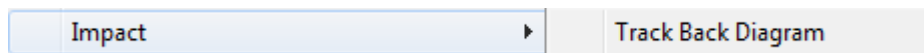


The **Projects** menu option can be used to remove the OLAP dimension from the current project or to add the OLAP dimension to the current project. The **List Projects** option displays a list of projects which contain the current object, results are shown in the bottom pane. Multiple objects can be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

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

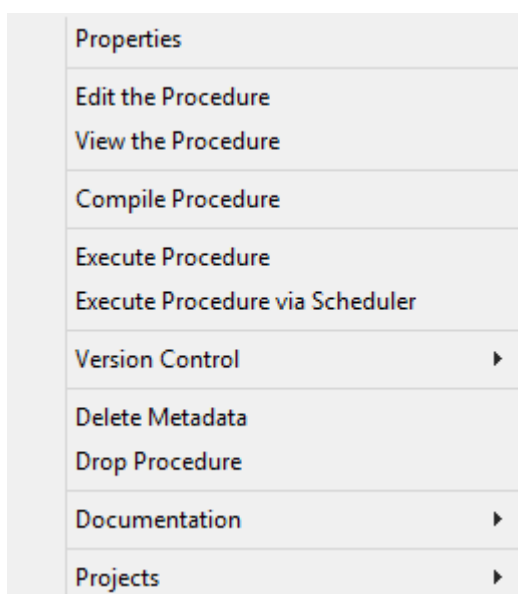


The **Impact** menu option enables you to produce a **Track Back** diagram on the OLAP dimension.



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 menu displayed when a right-click is used on a procedure name is as follows:



If a procedure has been locked as the result 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.

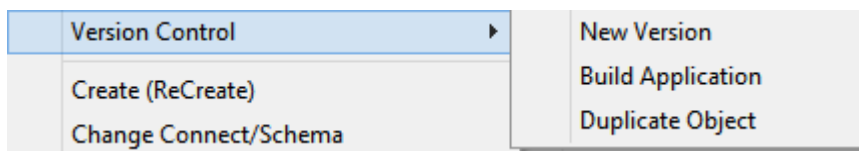
The **View the Procedure** menu option displays a read only copy of the procedure. If the procedure is locked by another user, then viewing the procedure is the only option available.

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 Procedure as a duplicate of this Procedure.

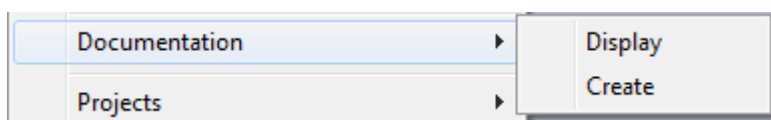


Selecting the **Delete Metadata** menu option deletes the procedure from the meta data. It then asks if the procedure should also be dropped from the database.

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**:



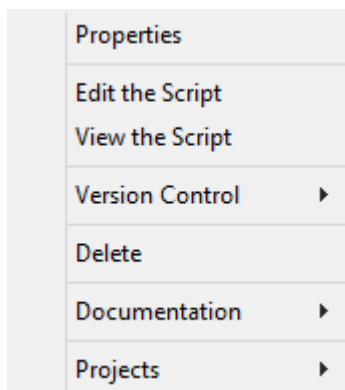
The **Projects** menu option can be used to remove the procedure from the current project or to add the procedure to the current project. The **List Projects** option displays a list of projects which contain the current object, results are shown in the bottom pane. Multiple objects can be selected by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

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



Scripts

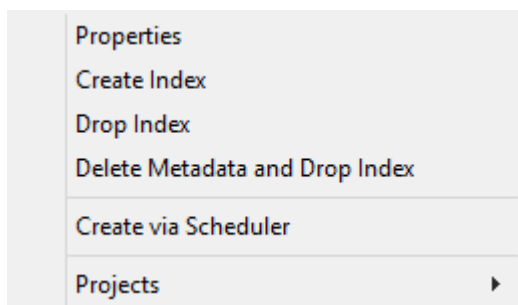
Scripts are very similar in their operations to procedures. The same menu options are available as for procedures and perform the same functionality.



If a script is deleted, it is only removed from the metadata. However, as 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** screen contains the entire definition of the index, including the columns in use, storage parameters and index type, etc. The way 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 **Storage** menu option displays the storage properties screen, containing storage options and parameters.

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.

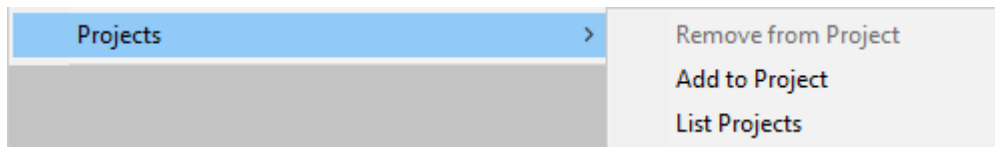
The **Drop Index** menu option drops the index in the database.

The **Delete Metadata and Drop index** 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.

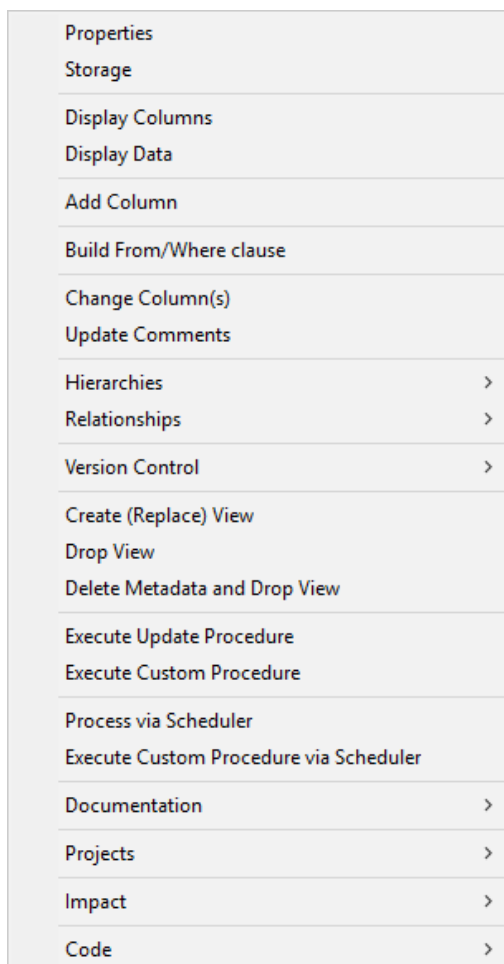
The **Projects** menu option can be used to remove the index from the current project or to add the index to the current project. The **List Projects** option displays a list of projects which contain the current object, results are shown in the bottom pane. Multiple objects can be selected by double-clicking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

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



Views

Views are primarily built to alias column names or add locking clauses to table objects (Load, Stage, Model and Aggregate Tables). The menu below shows the operations that can be performed on views.



The menu options available for views are almost a subset of those for stage tables, described under the stage section above.

The additional menu option is **Build From/Where clause**. This option provides a way to invoke a wizard to define a join between two or more tables.

Display Indexes, Query table via Excel, Add Index, Regenerate Indexes are not available for views.

Join Indexes

Join indexes are an indexing structure containing columns from one or more base tables. They are used to summarize data, pre-perform table joins and re-hash Teradata storage, by providing alternate primary indexing. The standard join index menu is as follows.

| | |
|-------------------------------|---|
| Properties | |
| Storage | |
| Display Columns | |
| Display Indexes | |
| Relationships | > |
| Add Column | |
| Add Index | |
| Regenerate Indexes | |
| Build From/Where Clause | |
| Validate Against the Database | |
| Gather Statistics | > |
| Version Control | > |
| Create (ReCreate) | |
| Drop Join Index | |
| Delete Metadata and Drop Join | |
| Documentation | > |
| Projects | > |
| Impact | > |

The bulk of these menu options are the same as for views and are described under the view section above. The differences are:

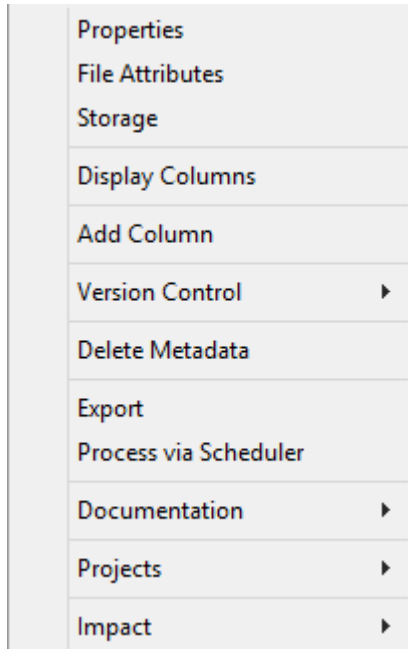
A join index may have an index on it, so the **Add Index** option is provided.

Join indexes are automatically maintained by Teradata, so do not have the **Execute Update Procedure**, **Execute Custom Procedure** or **Process via Scheduler** options.

Exports

Export objects do not exist as an object in their own right in the Teradata database. They are created from a single model, view or aggregate object and are used to generate file exports for a downstream system. Export files are built using Teradata tools, such as FastExport and Teradata Parallel Transporter.

The pop-up menu displayed when a right-click is used 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.

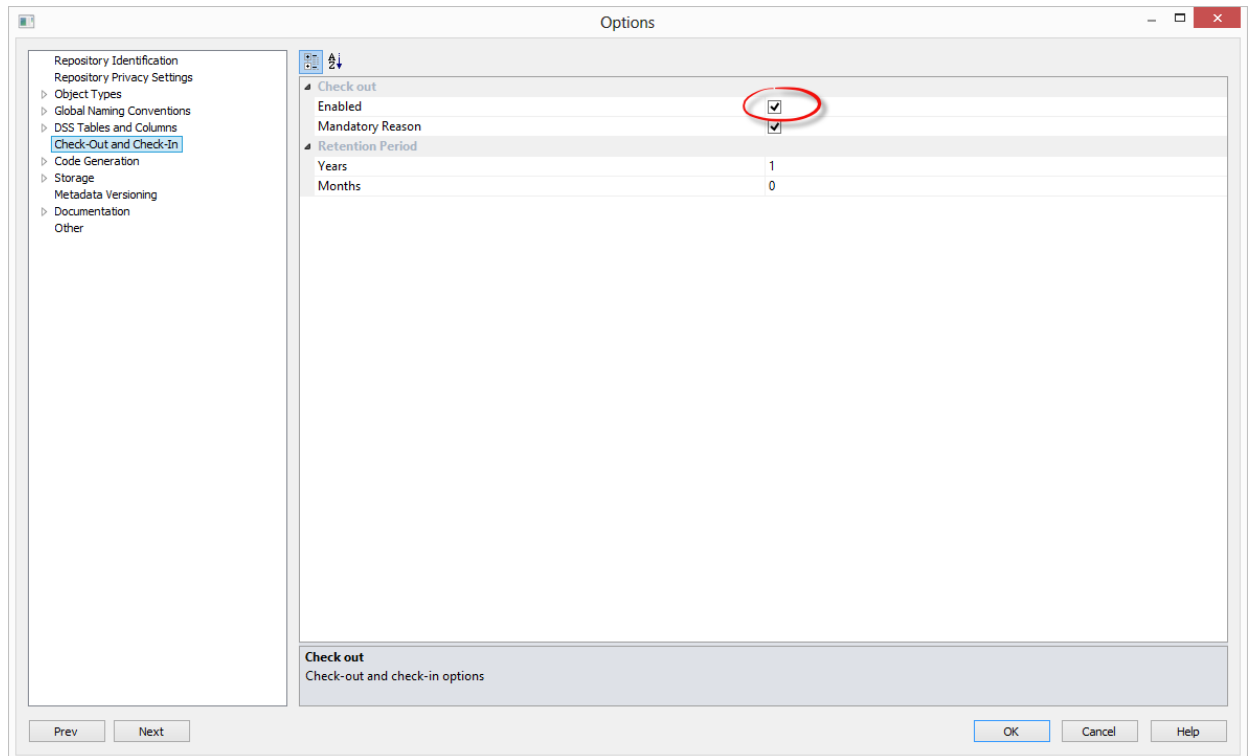
All the other options for export objects are described above under other object types.

OBJECT CHECK-OUTS AND CHECK-INS

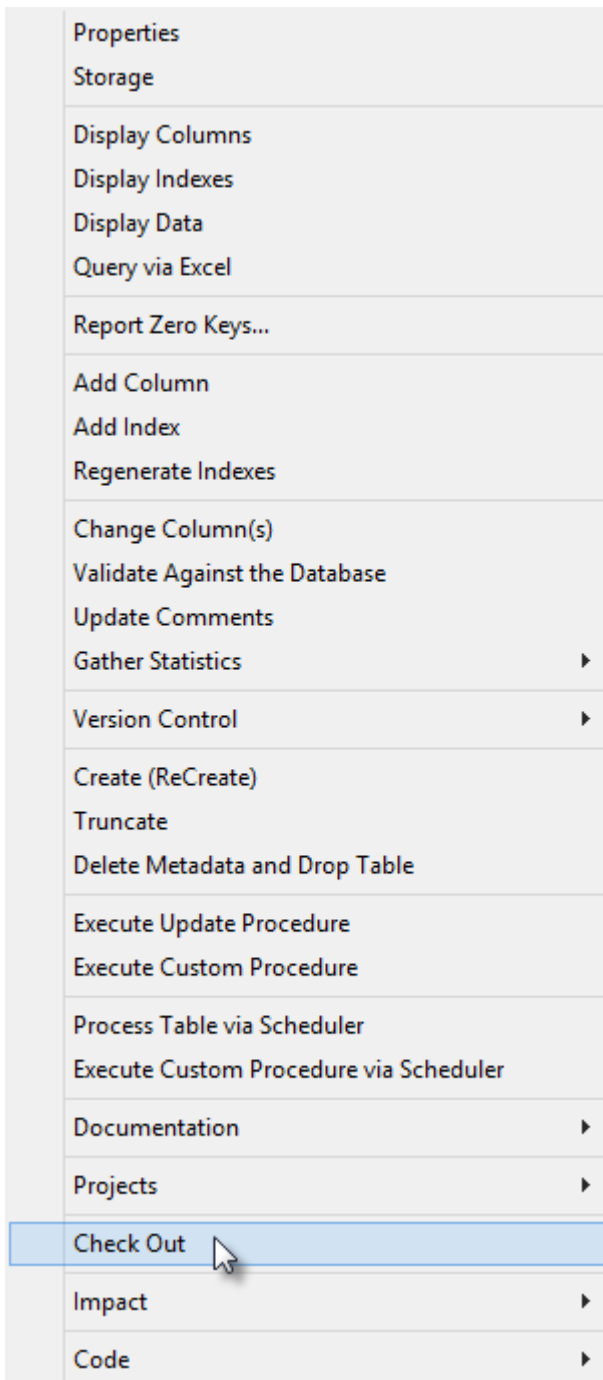
Objects in RED can be checked out for editing to prevent any other users from being able to modify, update or delete any of their associated objects while you are making changes to them.

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.



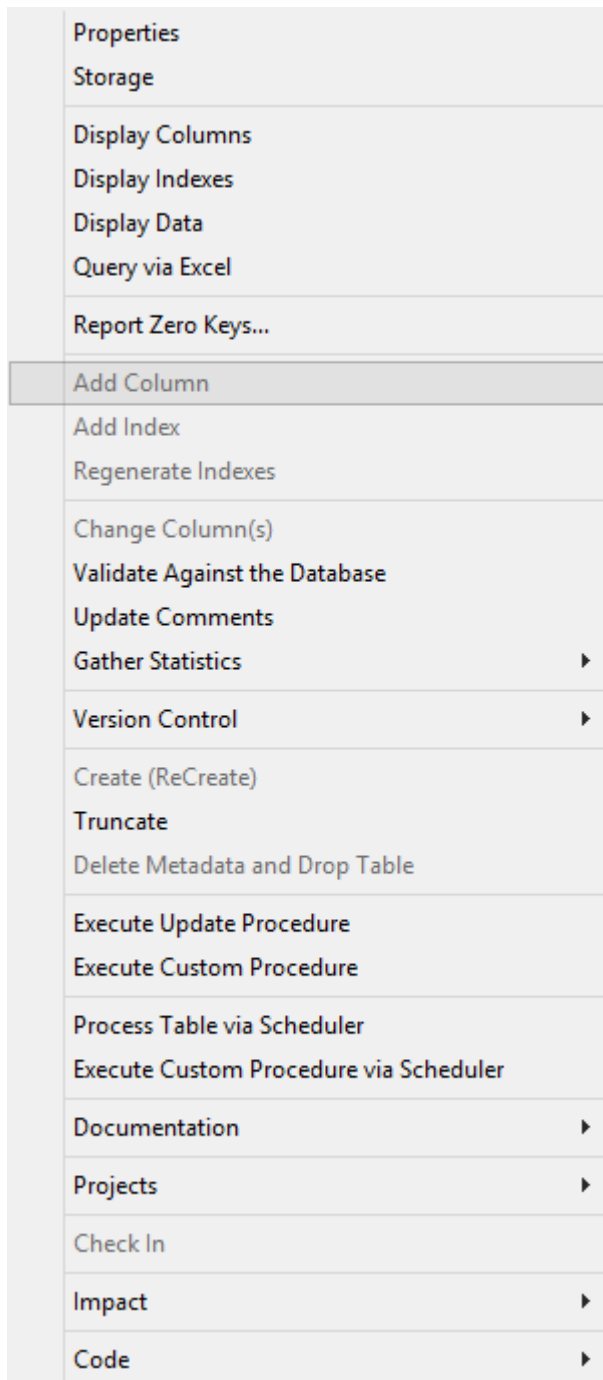
Checked out objects and their associated procedures and scripts cannot be modified, updated or deleted by other users. If a different user tries to access the properties screen or procedure windows for those objects, their fields will be disabled and headed NO UPDATE: Checked Out by (user name).

The screenshot shows a dialog box titled "Stage Table stage_customer- NO UPDATE: Checked Out by WhereScape Documentation". The dialog is divided into several sections:

- Properties:** A sidebar on the left with options: Properties (selected), Storage, Override Create DDL, and Notes.
- Table Name:** A text box containing "stage_customer".
- Table Type:** A dropdown menu set to "Stage".
- Unique Short Name:** A text box containing "stage_customer" with a note "(maximum 22 characters)".
- Description:** A text area containing "This table merges the list of Customers from the CRM and Sales database".
- Update Procedure:** A dropdown menu set to "update_stage_customer", with buttons for "Edit", "Rebuild", "Regenerate", and "Set Based Update".
- Custom Procedure:** A dropdown menu set to "(None)".
- Timestamps:** A section with three text boxes:
 - Metadata Structure Changed: 2014-10-21 11:58:23.813
 - Database Created: 2014-11-17 19:18:12.360
 - Database Altered: 2014-11-17 19:18:12.360

At the bottom right, there are three buttons: "OK", "Cancel", and "Help".

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.

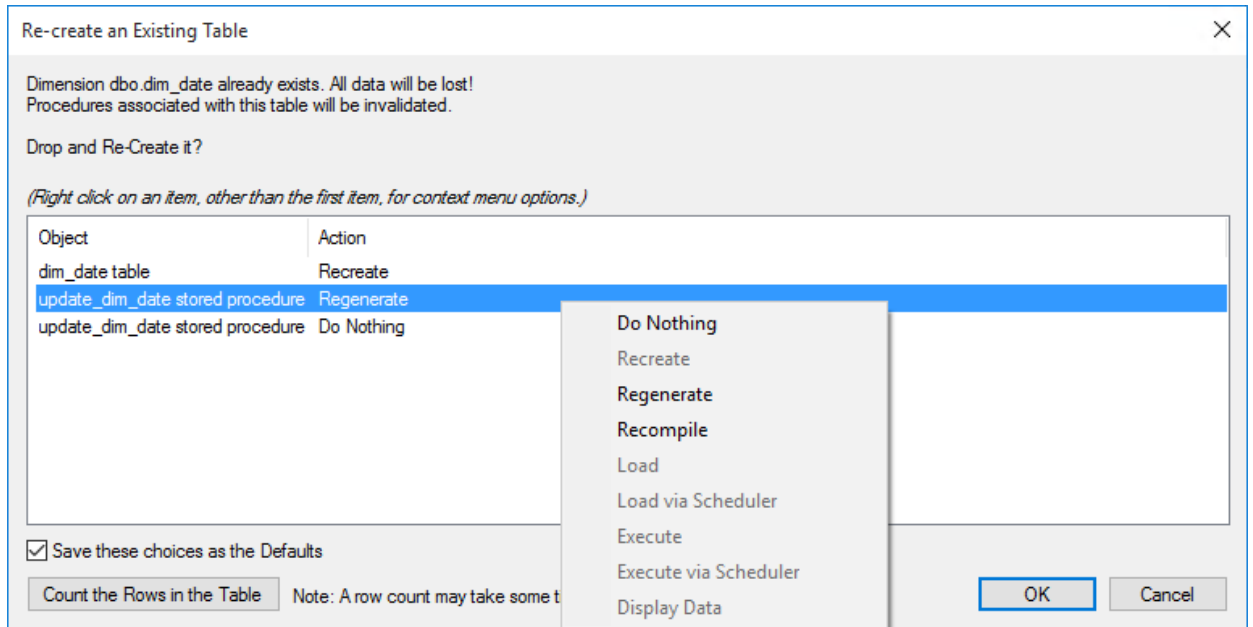


To check-in a checked-out object, simply right-click it and select Check In from the context menu.

| |
|--|
| Properties |
| Storage |
| Display Columns |
| Display Indexes |
| Display Data |
| Query via Excel |
| Report Zero Keys... |
| Add Column |
| Add Index |
| Regenerate Indexes |
| Change Column(s) |
| Validate Against the Database |
| Update Comments |
| Gather Statistics ▶ |
| Version Control ▶ |
| Create (ReCreate) |
| Truncate |
| Delete Metadata and Drop Table |
| Execute Update Procedure |
| Execute Custom Procedure |
| Process Table via Scheduler |
| Execute Custom Procedure via Scheduler |
| Documentation ▶ |
| Projects ▶ |
| Check In |
| Impact ▶ |
| Code ▶ |

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. |

| Object(s) | Action | Description |
|------------------|-----------------------|---|
| | 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. |
| | 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 | <p>Display the table data after recreating the table and update procedure (if applicable).</p> <p>This is the default setting for Views and Dimension Views.</p> <p>This is the default setting for tables when an update procedure is present and has been executed in the previous action.</p> <p>Unavailable when an update procedure is present but has not been executed in the recreate dialog.</p> |

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 types of object 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 the F5 or Ctrl/R key.

Object Groups

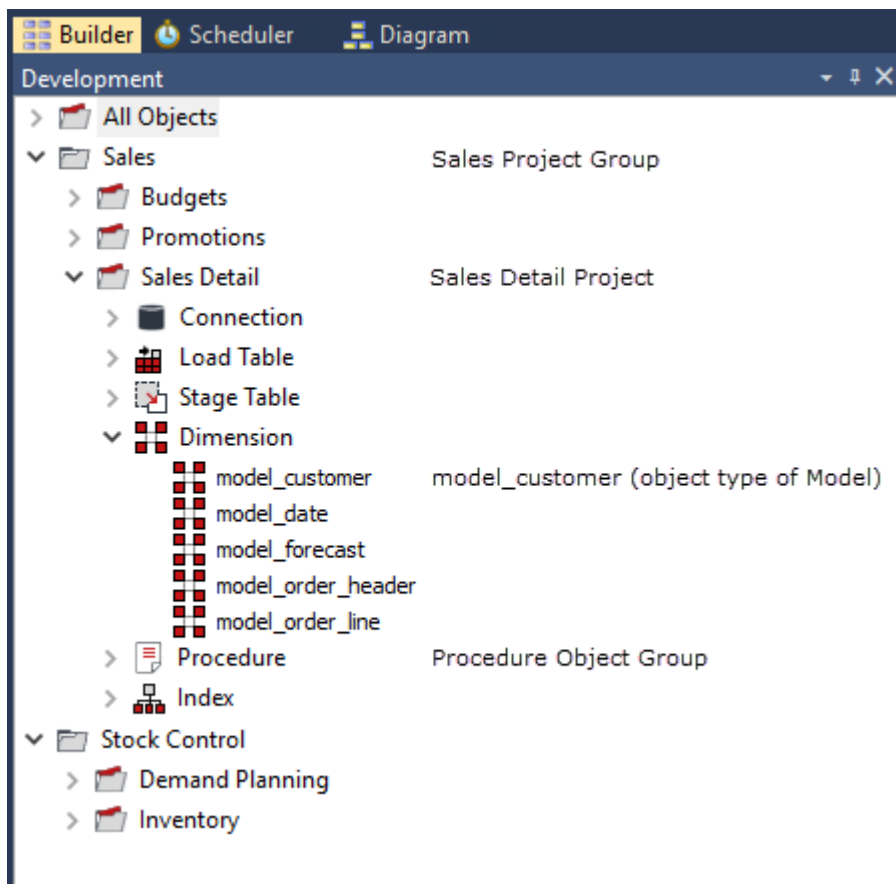
The objects created in WhereScape RED are grouped together into object groups based on object type. For example, we store all the dimension objects in the Dimension **object group**. Optionally, we can choose to display dimension tables as the **Dimension** object group and dimension views as the **Dimension 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 model object called `model_product`, there is and can only be one copy of the object `model_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, 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.

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



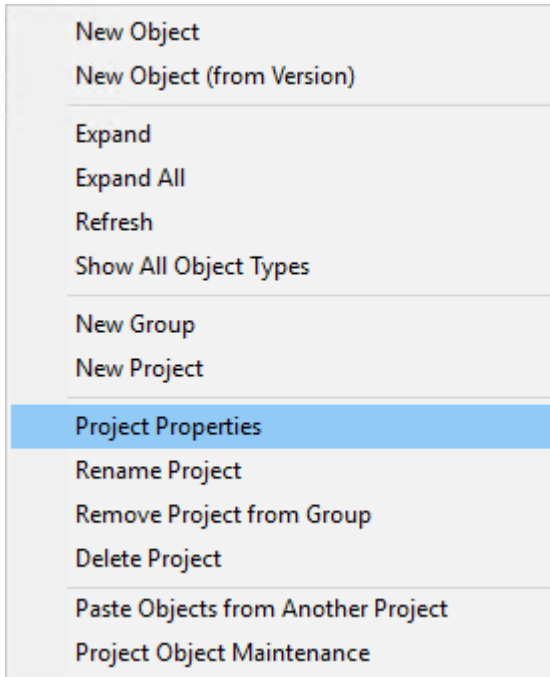
Note: If you delete an object using the right-click menu, then the object will be deleted from the metadata and will be removed from all Projects. Remove the object from the project instead of deleting it.

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.

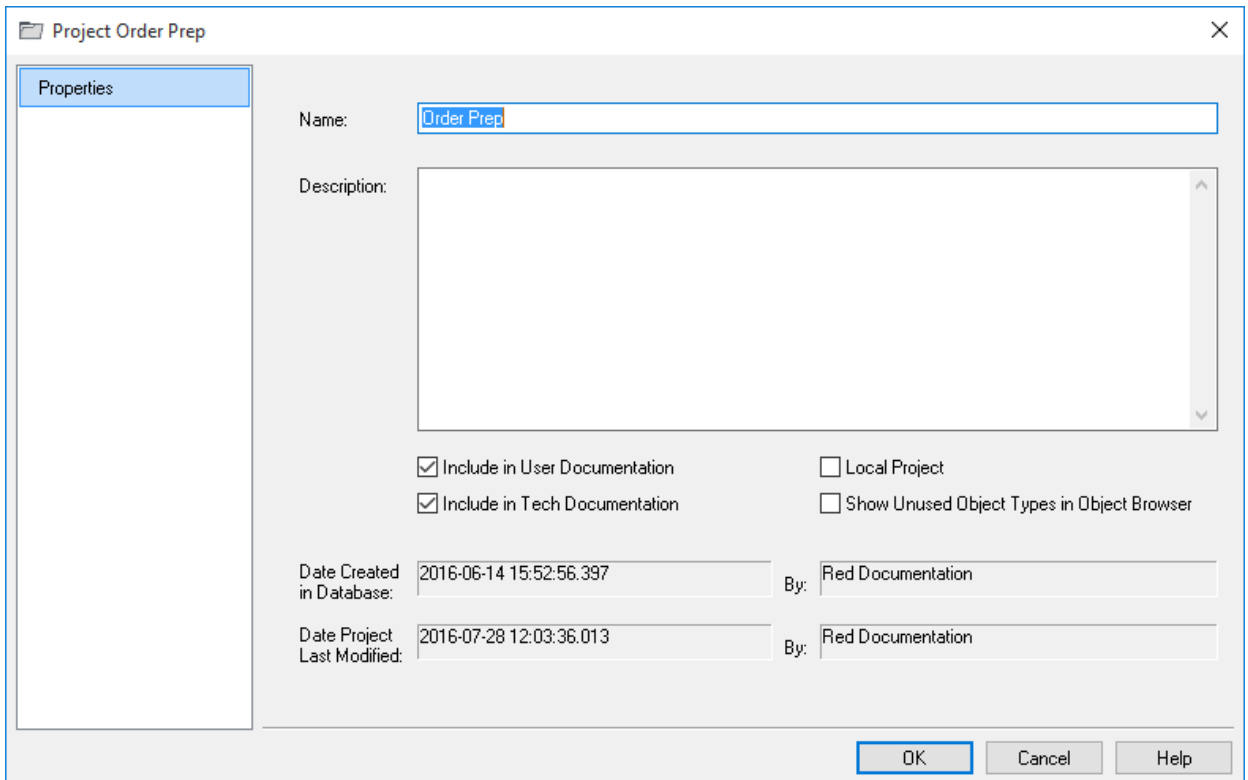
To create a project, right-click in the left pane below the last project and select **New Project**.

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 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

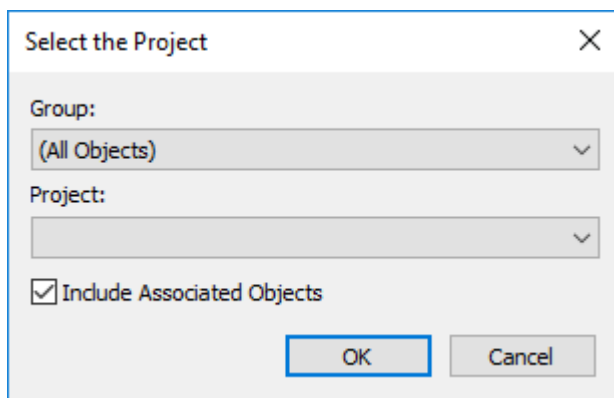
Projects can in turn be grouped together into Project Groups. A project must only appear in one Project Group. To create a Project Group, use the **File/New Group** menu option. Project Groups can be removed by using the right-click menu when positioned on a group name.

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 54)

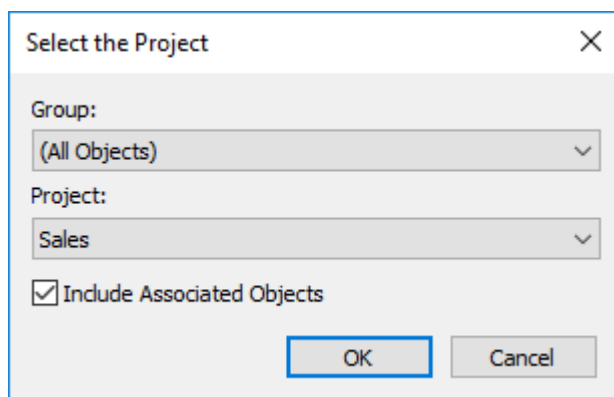
Both options above that use the right-click **Projects/Add to Project** menu result in the following dialog box being displayed:



The dialog box is titled "Select the Project" and has a close button (X) in the top right corner. It contains two dropdown menus: "Group:" with "(All Objects)" selected, and "Project:" which is currently empty. Below the dropdowns is a checked checkbox labeled "Include Associated Objects". At the bottom, there are two buttons: "OK" and "Cancel". The "OK" button is highlighted with a blue border.

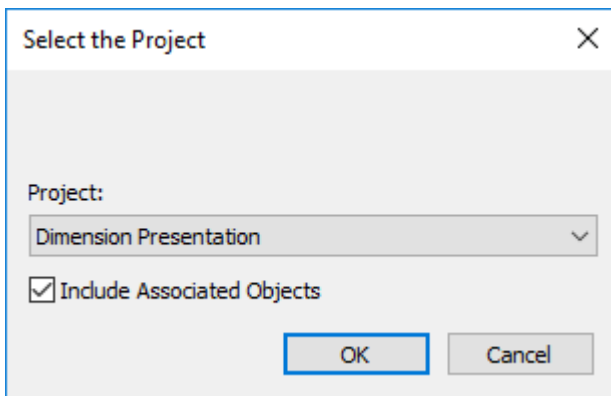
To add an object to an independent project (a project that is not in a group)

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



The dialog box is titled "Select the Project" and has a close button (X) in the top right corner. It contains two dropdown menus: "Group:" with "(All Objects)" selected, and "Project:" with "Sales" selected. Below the dropdowns is a checked checkbox labeled "Include Associated Objects". At the bottom, there are two buttons: "OK" and "Cancel". The "OK" button is highlighted with a blue border.

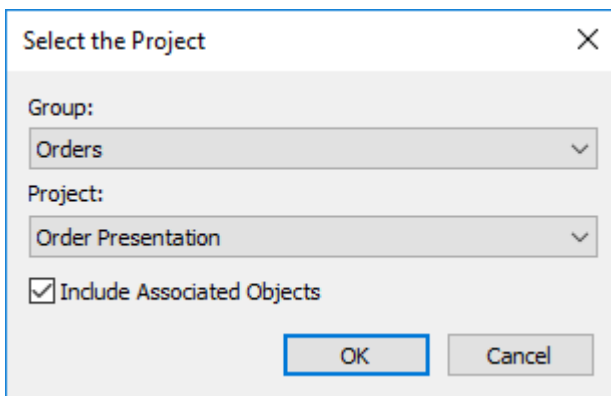
If Groups have been created, both drop-down lists will be visible. If Groups have not been created, only the project drop-down will be visible like this:



The screenshot shows a dialog box titled "Select the Project" with a close button (X) in the top right corner. Inside the dialog, there is a "Project:" label followed by a drop-down menu currently displaying "Dimension Presentation". Below this, there is a checked checkbox labeled "Include Associated Objects". At the bottom of the dialog, there are two buttons: "OK" and "Cancel".

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**.



The screenshot shows a dialog box titled "Select the Project" with a close button (X) in the top right corner. Inside the dialog, there is a "Group:" label followed by a drop-down menu currently displaying "Orders". Below this, there is a "Project:" label followed by a drop-down menu currently displaying "Order Presentation". Below the "Project:" dropdown, there is a checked checkbox labeled "Include Associated Objects". At the bottom of the dialog, there are two buttons: "OK" and "Cancel".

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

REMOVING OBJECTS FROM PROJECTS

To remove objects from a project

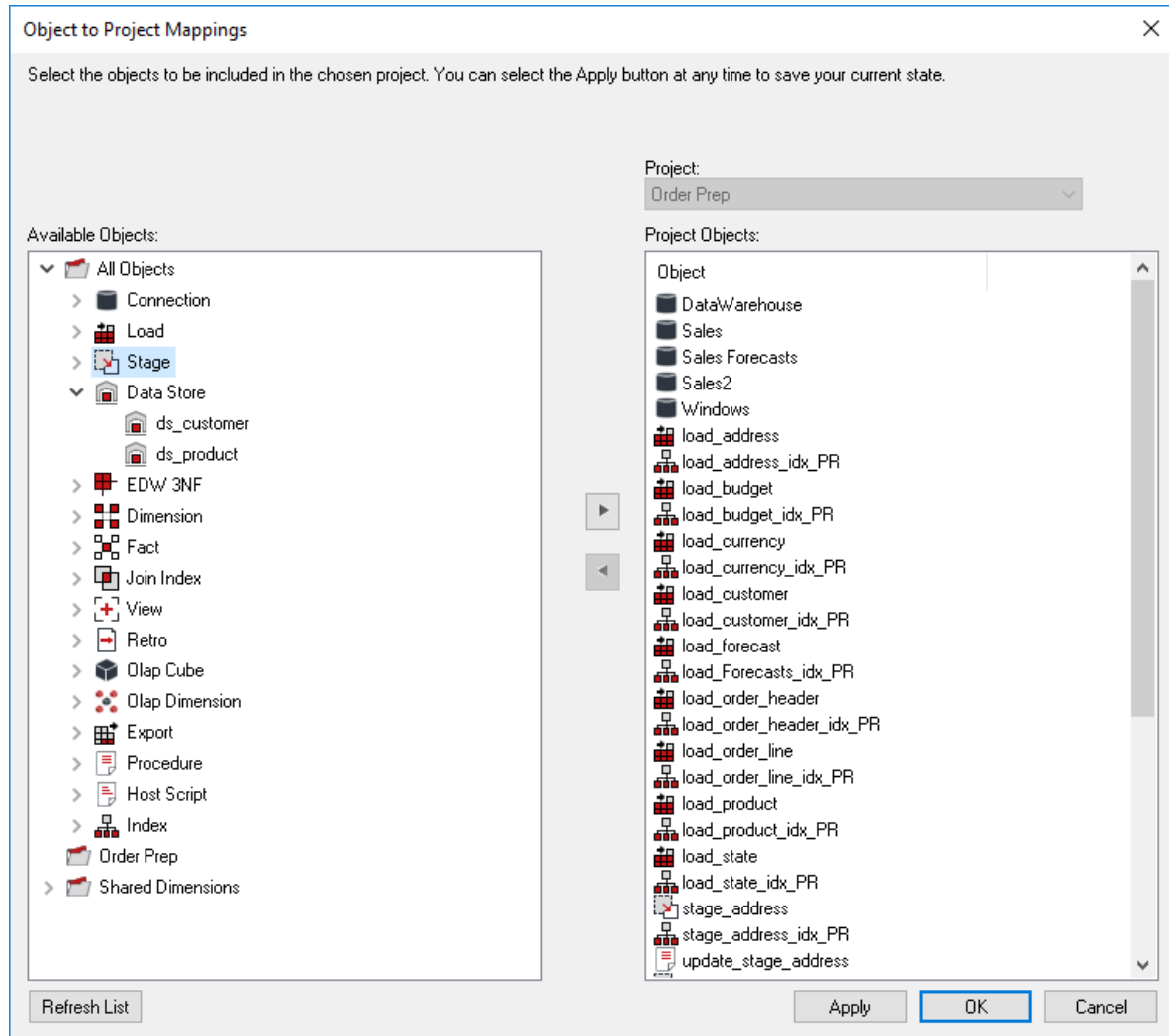
There are different ways to remove objects from a project:

- **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 54)

USING PROJECT/OBJECT MAINTENANCE

The Project/Object Maintenance Facility is invoked by right-clicking 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 left. 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.

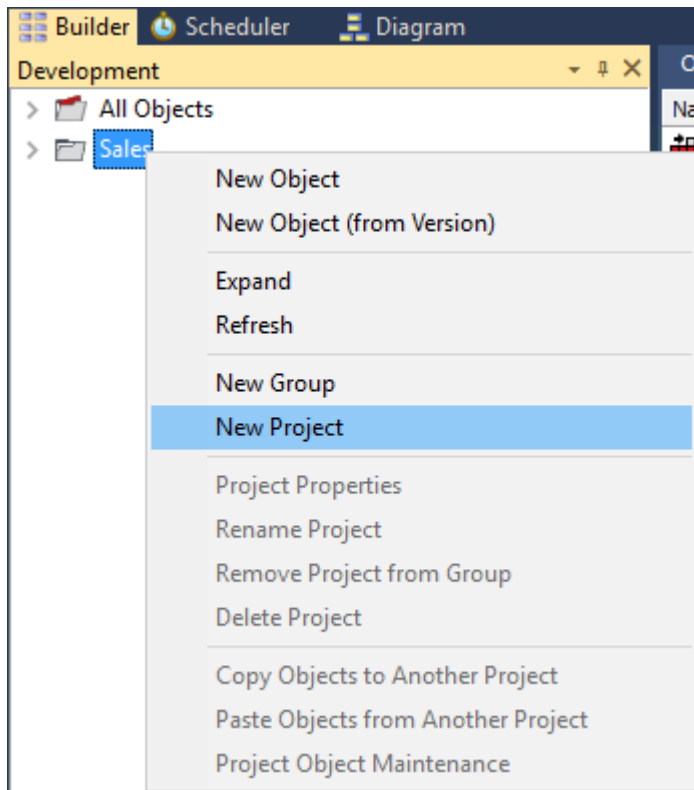
When done, click **OK**.

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 Project**:



The other option is to move an existing project from another group into this group. See *Moving Projects within Groups* (on page 56)

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 PROJECTS MEMBERSHIPS FOR AN OBJECT(S)

To list which projects 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.

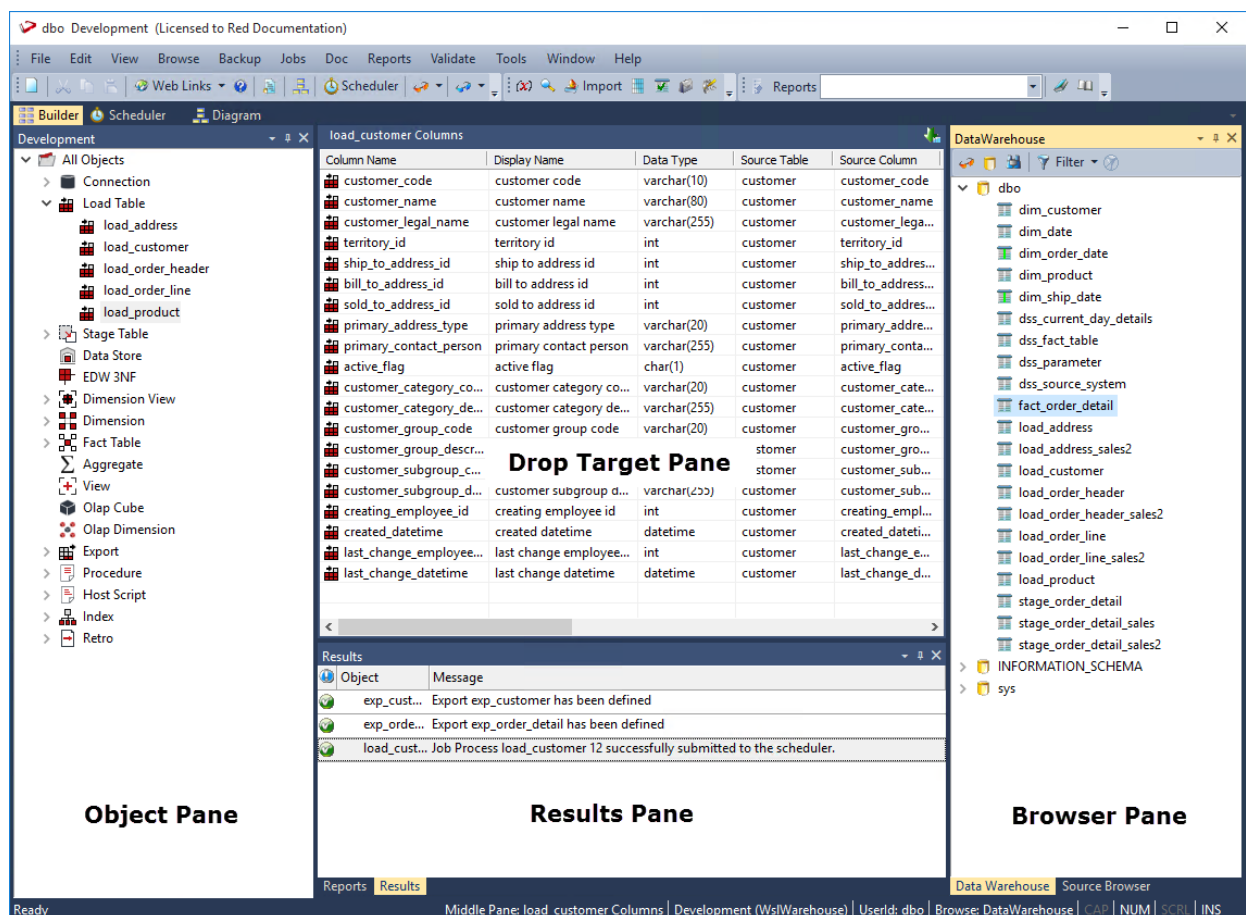
WINDOWS AND PANES

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:

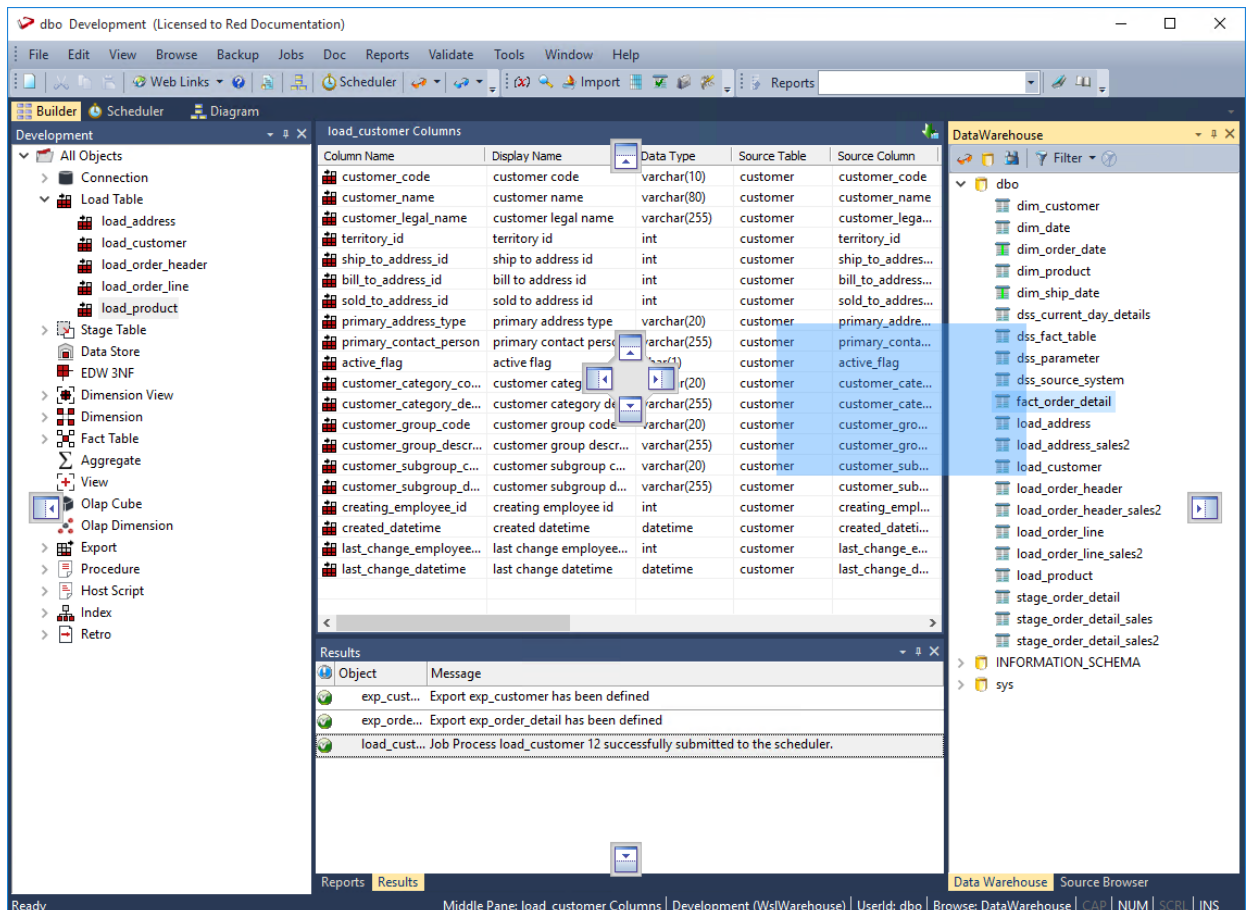
- The **Builder** window
- The **Scheduler** window
- The **Diagram** window
- The **Procedure Editor** window

BUILDER WINDOW

The **Builder Window** has four panes.



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

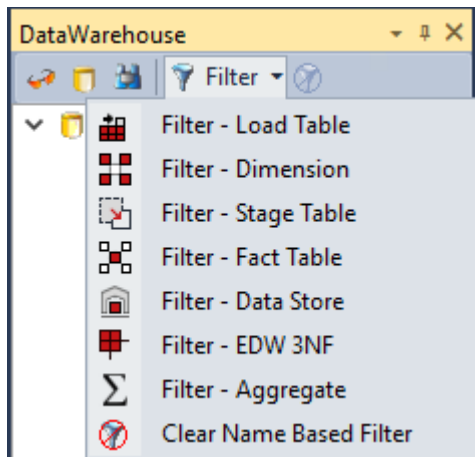


The **left pane** contains all the **objects** within the metadata repository. These objects are stored in object groups (e.g. Model). The object groups are in turn optionally stored in Projects, and the Projects are optionally stored within Project Groups.

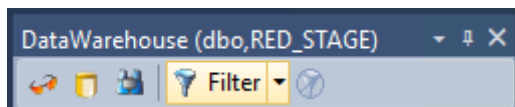
The **middle pane** is used to show the **results** of various **queries** on both the metadata and the underlying source and database tables. The middle pane is also used as the drop target in drag and drop operations. The status line at the bottom of the screen displays the current contents of the middle pane. To send the middle pane output to a file or to clipboard, see **Export Middle Pane Output** (on page 67).

The **right pane** is the **Browser Pane** and it shows both source and data warehouse systems. There are two browser panes available at any one time. This 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.



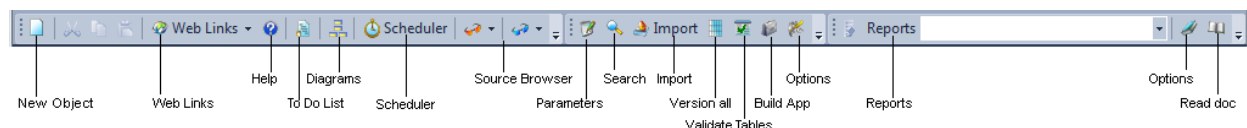
The **bottom** pane is the **results** pane and it shows the results of any command executed on an object. Multiple messages can be displayed. Expand the '+' sign next to an object to see a complete list of messages relating to an object. When a report is run from the main menu, the results are displayed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

Pop-up menus are typically available in all three panes.

F5 or Ctrl+R key 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:

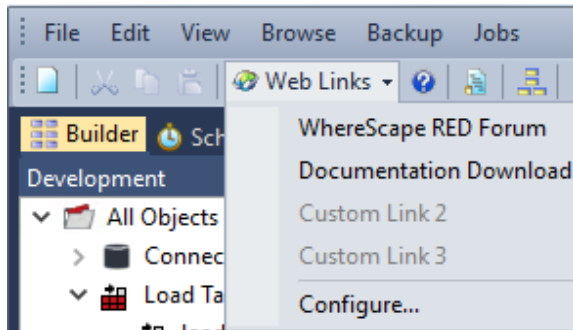


The **diagrammatic view** button (diagram view) switches to the diagrammatic window. This window allows the representation and printing of star schema and track back diagrams.

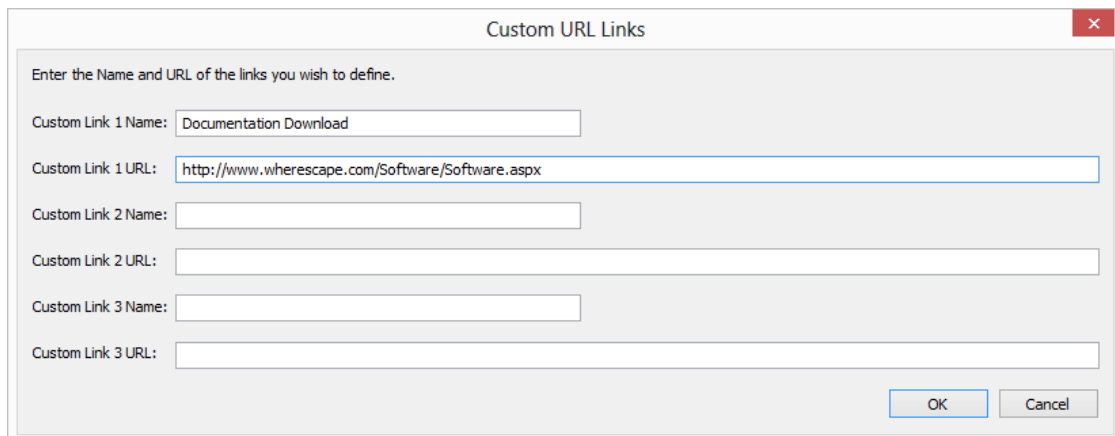
The **scheduler** button switches to the scheduler control window.

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 on the down arrow beside the glasses icon.

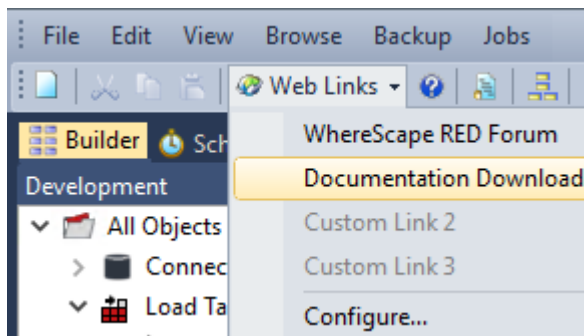
The **Web Links** button brings up the online WhereScape forum in a new tab. To select or enter other web links, click the down arrow beside the Web Links button.



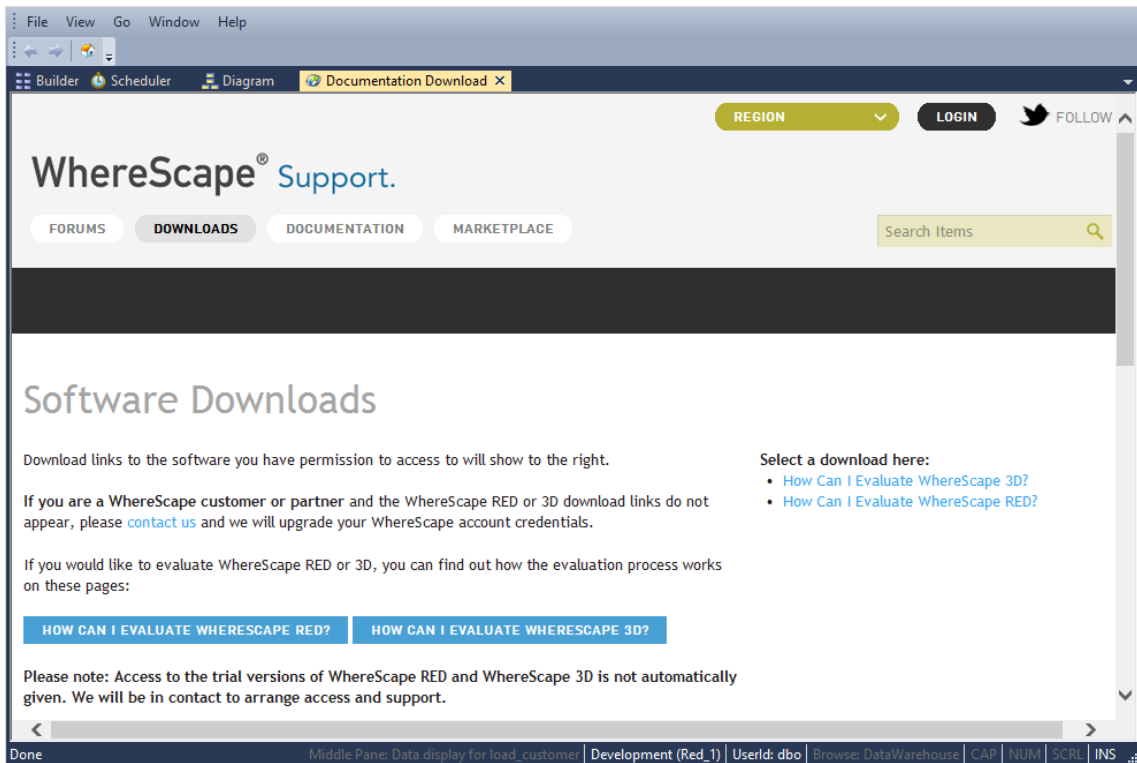
As an example, let us add a Web Link to the Documentation Download on WhereScape's web page. Select **Configure** to open the following dialog box, allowing you to enter three custom URLs. Enter the name and the URL for the first custom link. Click **OK**.



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



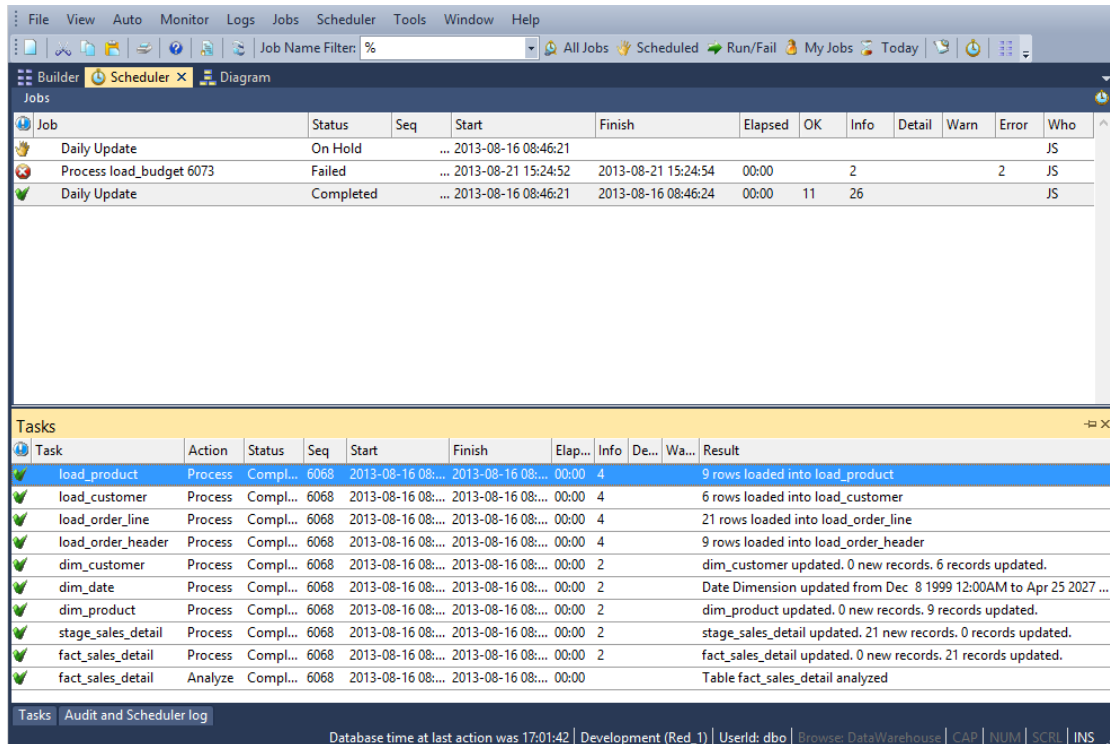
The relevant web page is displayed in a new tab.



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.

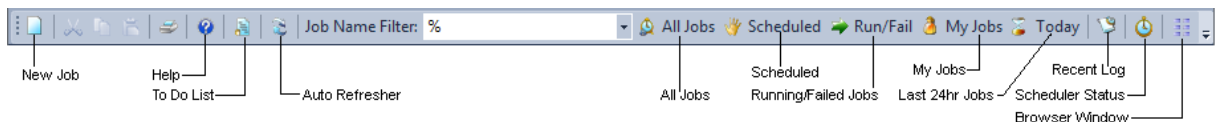


Jobs can be scheduled, monitored, edited and deleted through this window. The window consists of two panes. The toolbar provides a quick way to display various job selections in the **top pane**. Double-click on the job in the top pane to see the tasks of the selected job in the **bottom pane**. Double-click 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 688) 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.

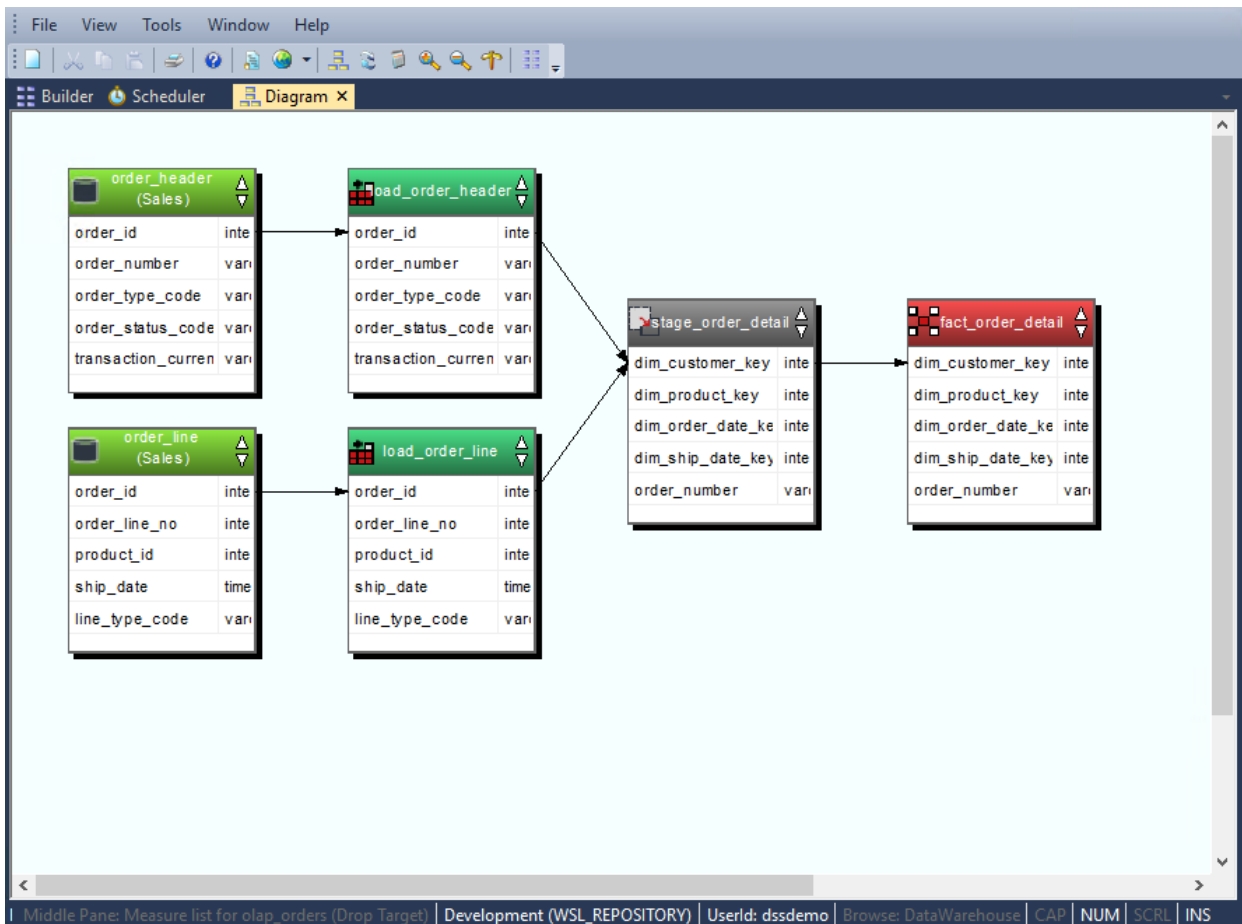
The **Auto Refresh** button, when depressed, will result in a refresh of the current (right pane) display every 10 seconds. Click the button again to stop the auto refresh.

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.



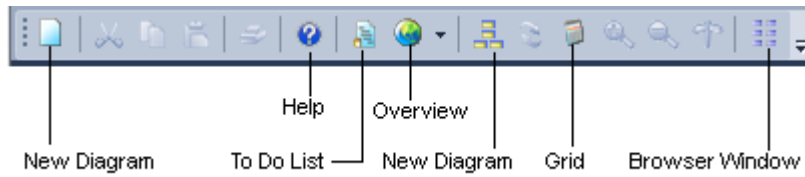
The **Diagram Selection** is as follows:

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

See the **Diagrams** (on page 771) 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.

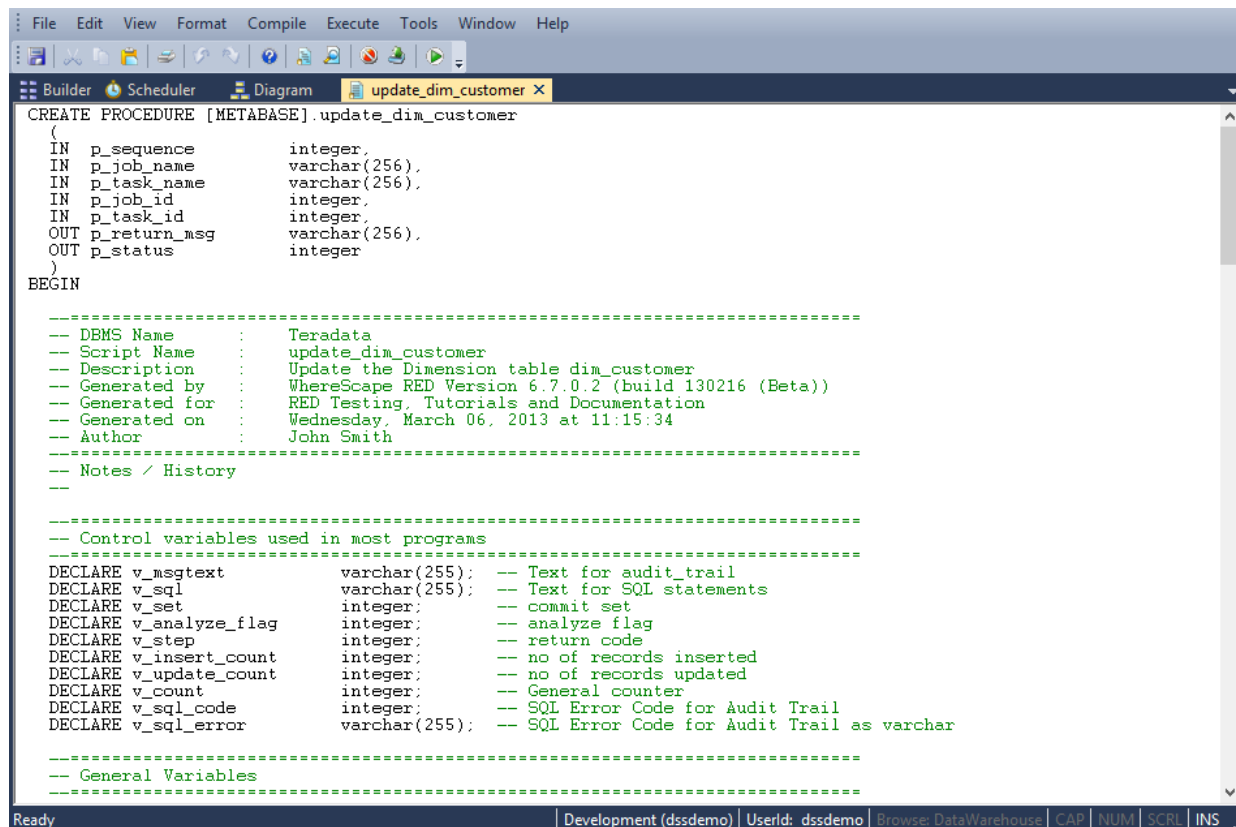
The **Overview** button provides a diagram showing the various objects in the WhereScape metadata and the standard flow of data through these objects. Repeated clicking of the **Overview** button will step through each stage of the data flow.

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.



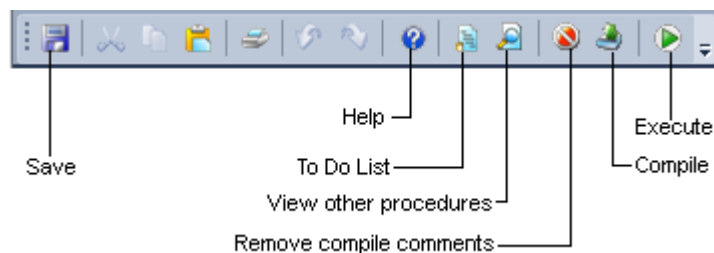
Multiple such windows can be open at any one time, each processing a different procedure.

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 630) 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.

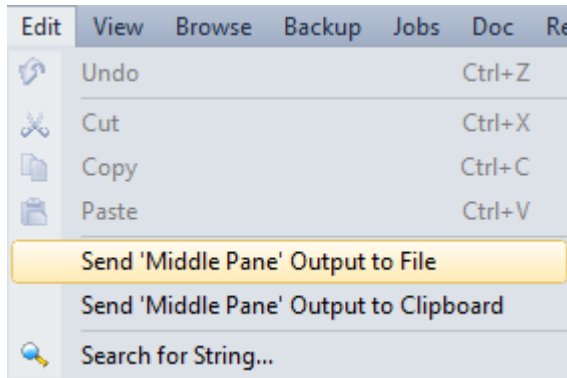
The **View Other Procedures** button allows the concurrent viewing of older versions of the current procedure, other procedures in the metadata, compiled procedures in the database and templates.

The **Compile** button will attempt to compile the procedure. Once compiled the procedure is stored within the database as well as in the metadata.

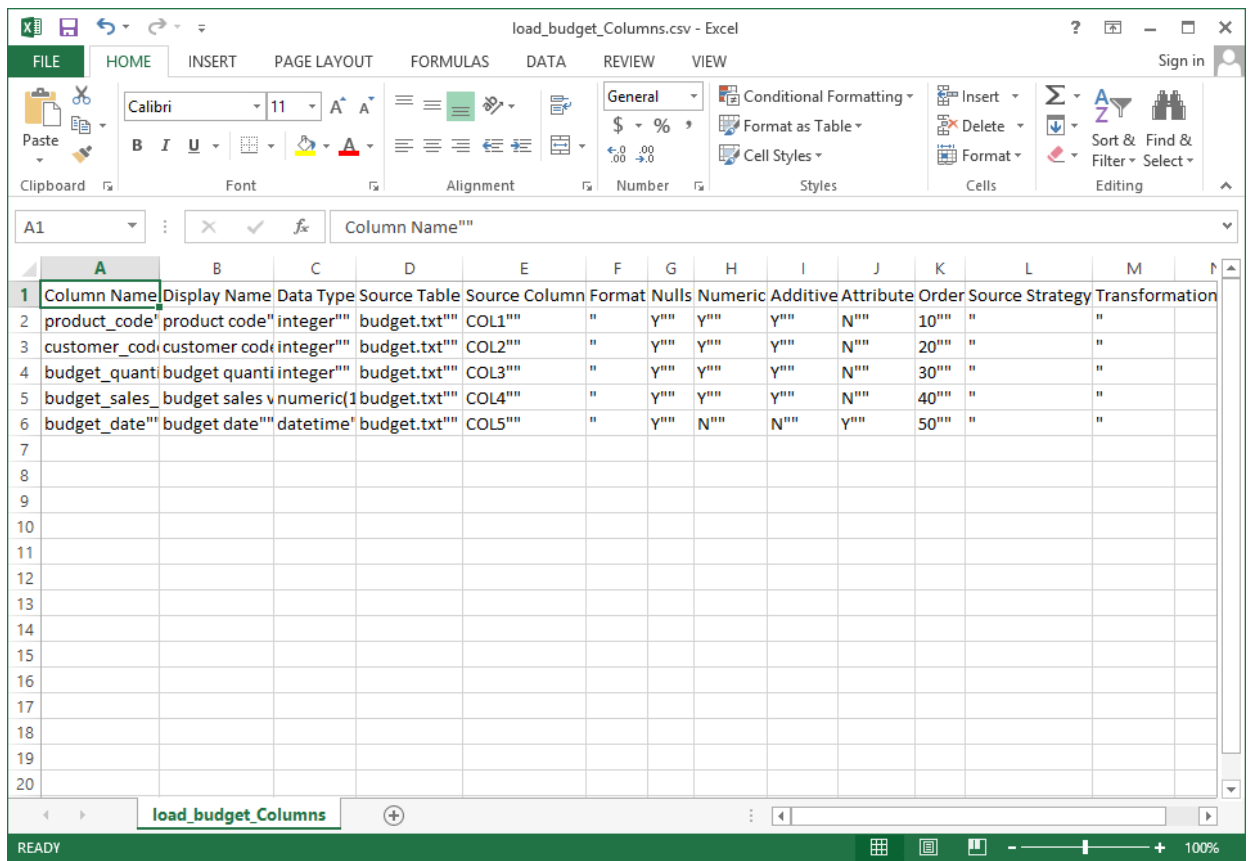
The **Execute** button will run a procedure that conforms to the WhereScape parameter syntax. See the chapter on procedures for more details.

EXPORT MIDDLE PANE OUTPUT

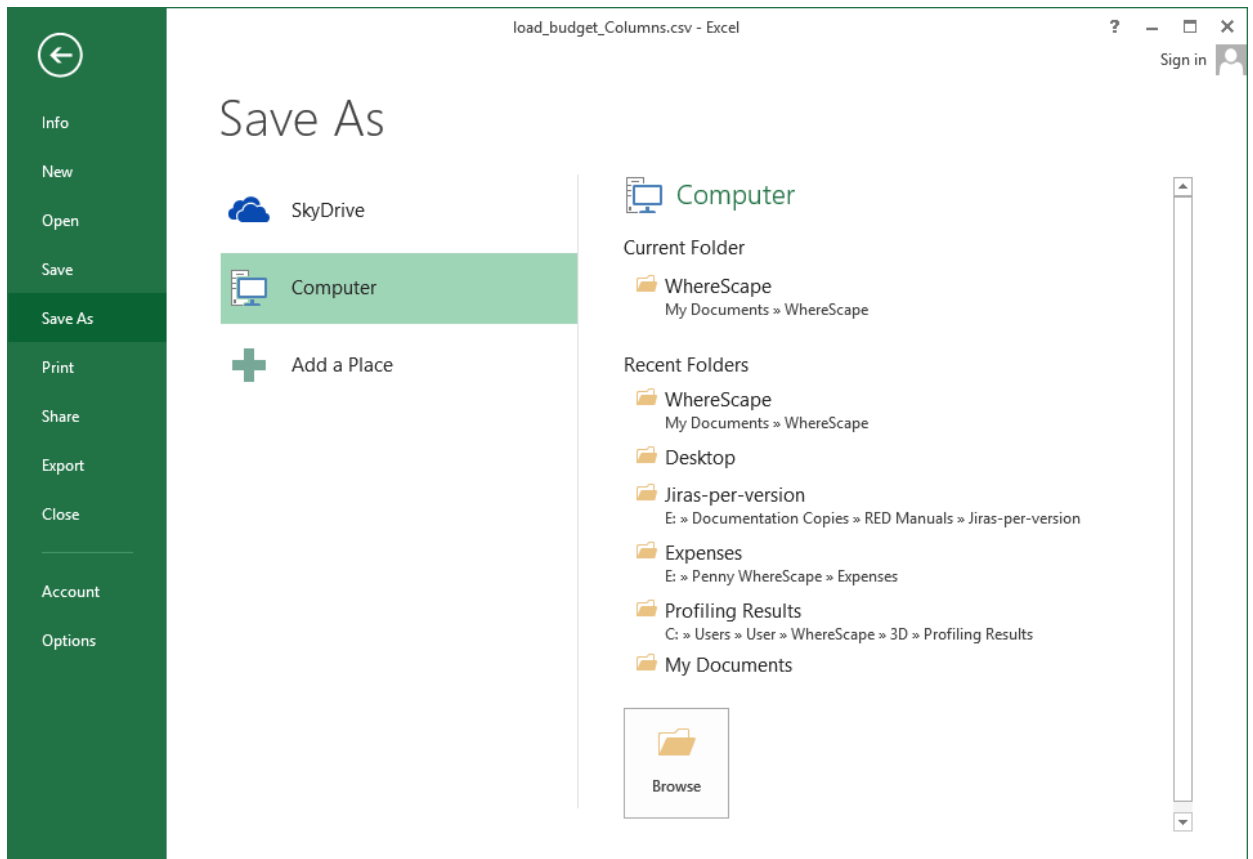
- To send the middle pane output to a file, go to the **Edit** menu and select **Send 'Middle Pane' Output to File**.



The columns will then be displayed in Excel.

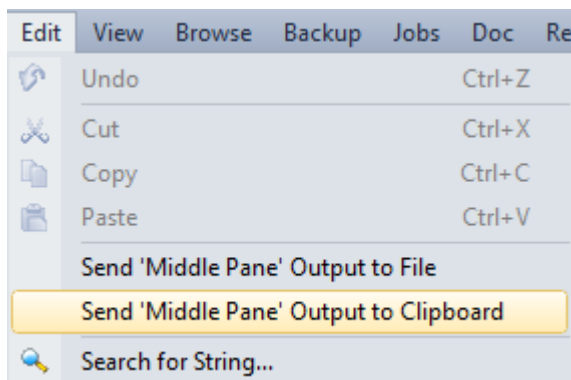


To save to a File, click on the **File tab** and then select either **Save** or **Save As** to save to a file.



To view the settings for **Middle Pane File Output**, see *Export Middle Pane Output Settings* (see "**Outputs**" on page 125).

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

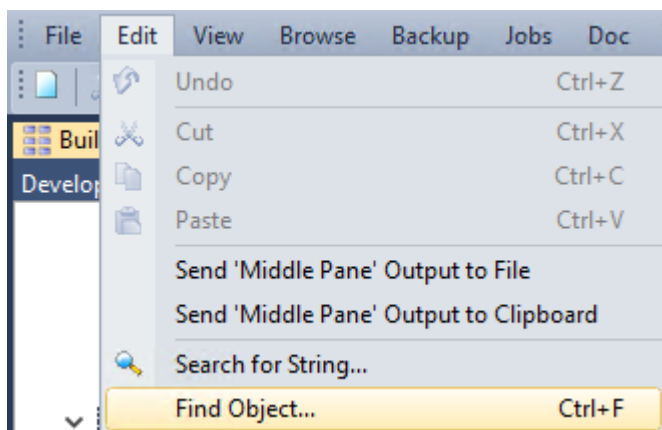


To view the settings for **Middle Pane Clipboard Output**, see *Export Middle Pane Output Settings* (see "**Outputs**" on page 125).

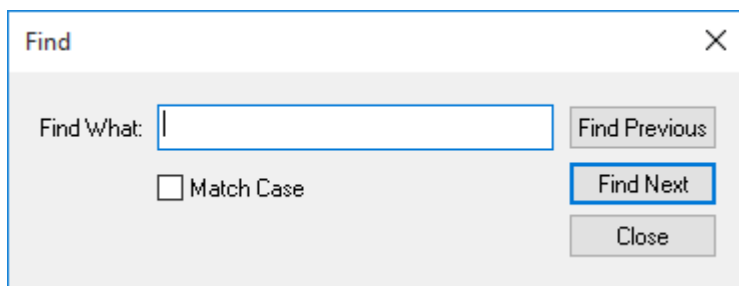
FIND FUNCTION

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:



CHAPTER 4 TUTORIALS

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

CHAPTER 5

DEFAULT SETTINGS

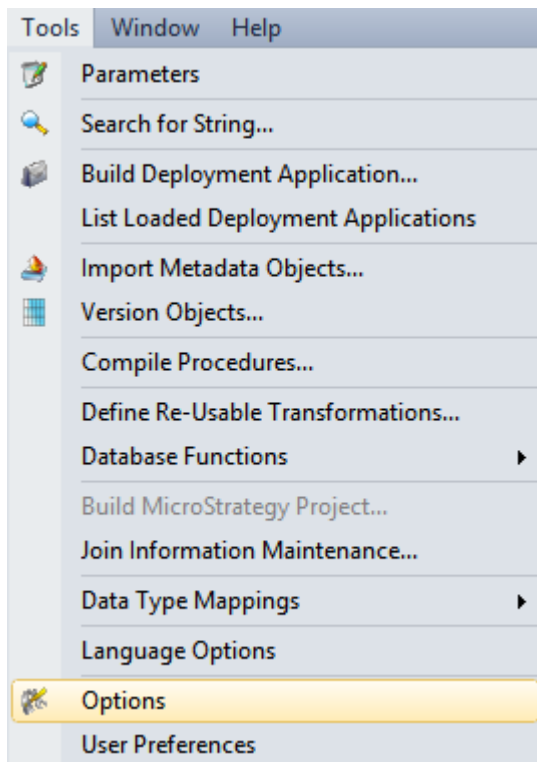
This chapter describes the settings and default values that can be set for a metadata repository. To access these settings, select the **Tools** menu; and then either **Options** or **User Preferences**.

IN THIS CHAPTER

| | |
|-----------------------------------|-----|
| Settings - Options | 73 |
| Settings - User Preferences | 113 |
| Settings - Language Options | 130 |

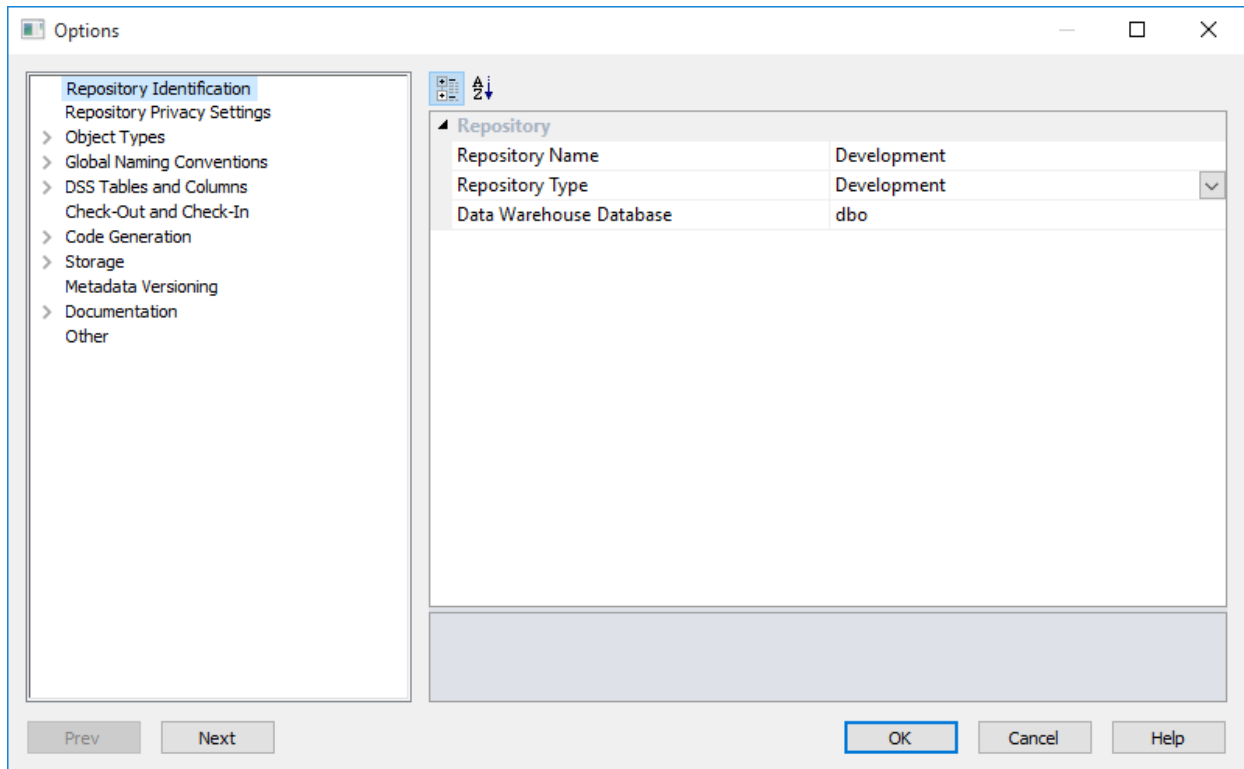
SETTINGS - OPTIONS

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



SETTINGS - 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 top left corner of 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 must reflect the environment. For example, a 'Production' type must be chosen for the production environment.

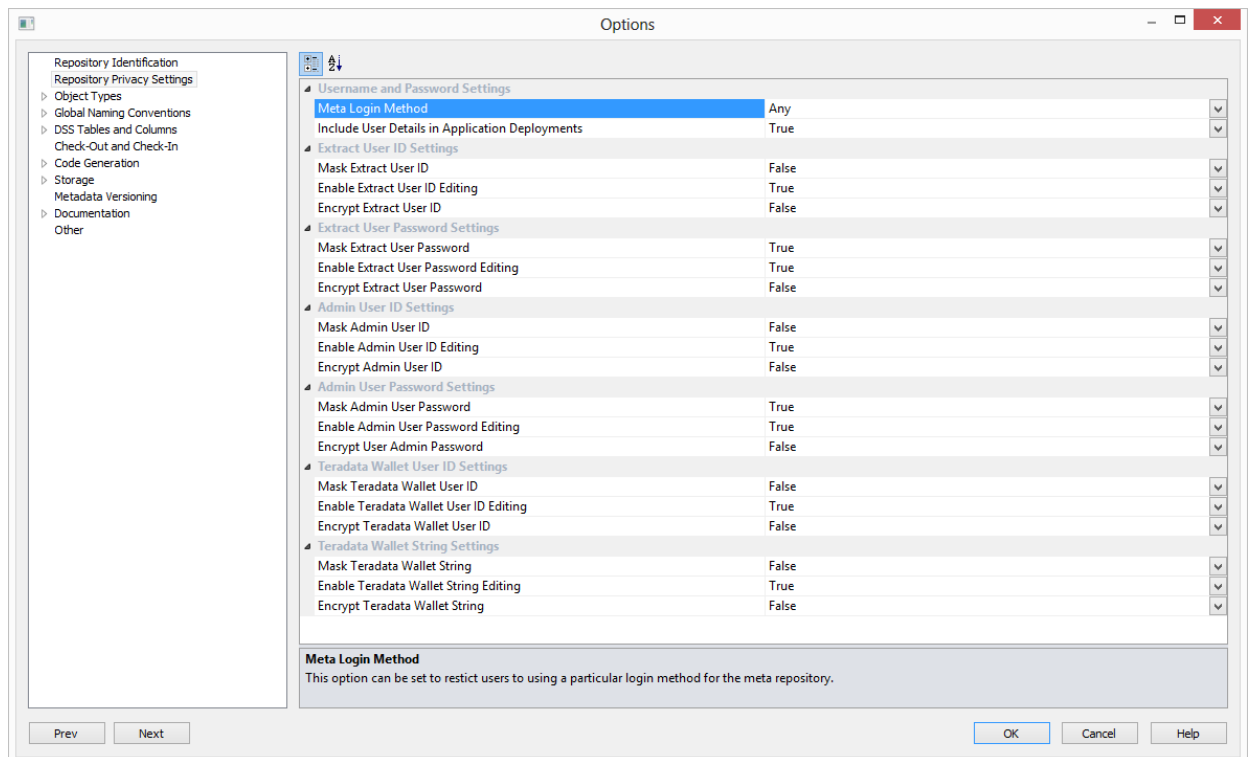
Data Warehouse Database

The database of the data warehouse that is used by the WhereScape RED scheduler. You should not normally need to change this value.

SETTINGS - 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 Password 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 WhereScape 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

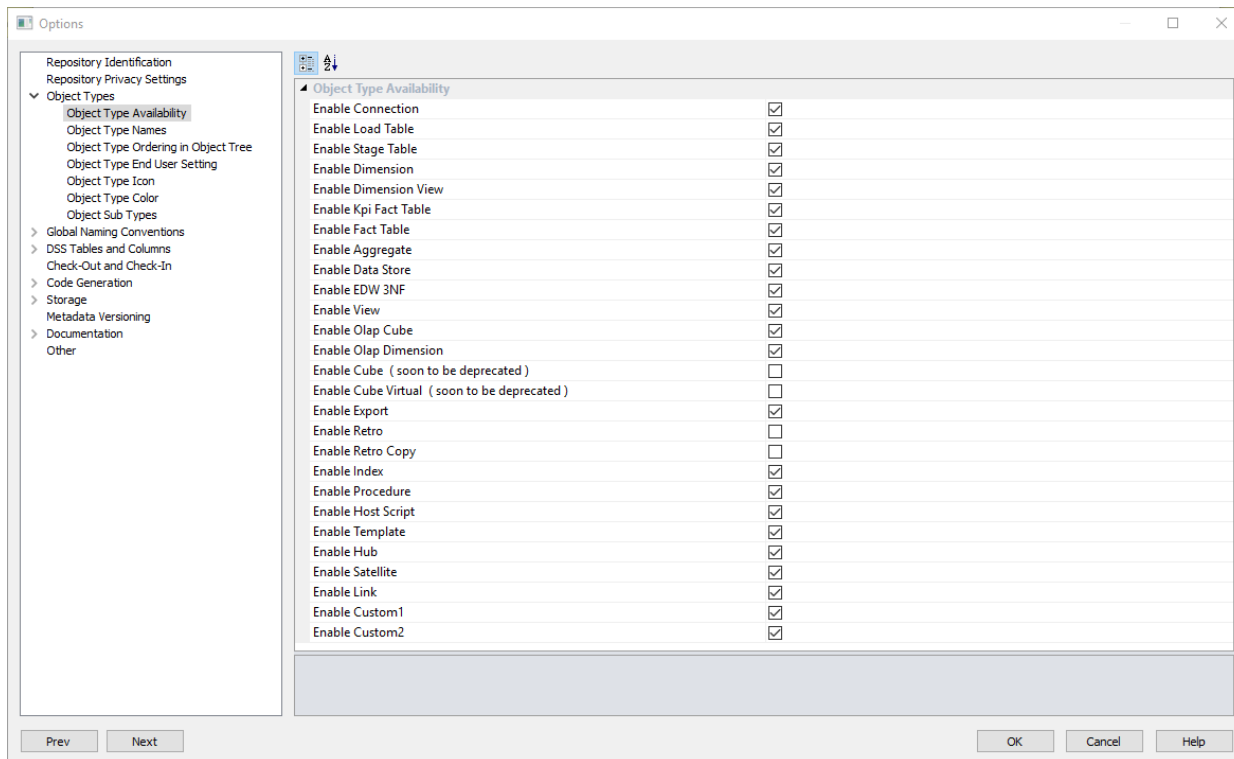
Teradata Wallet String Settings

- **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

SETTINGS - OBJECT TYPES

OBJECT TYPE AVAILABILITY

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

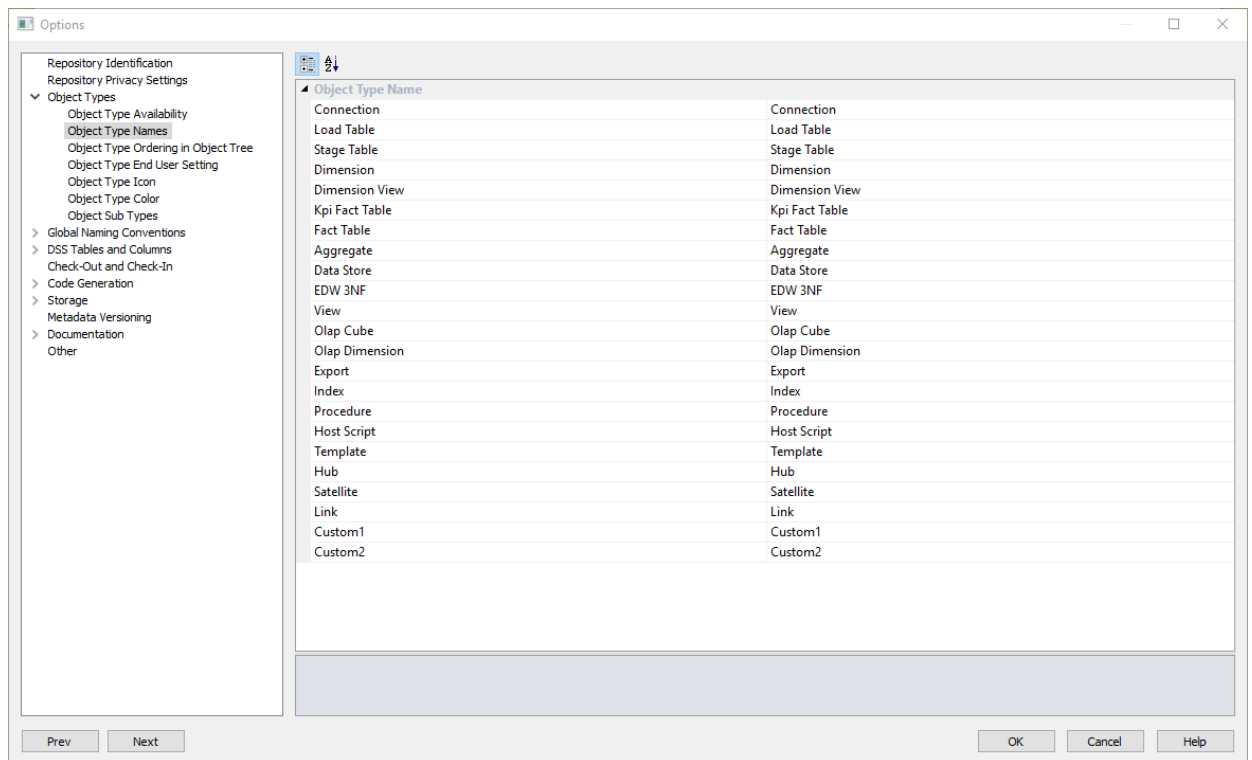


Object Type Availability

Enable/Disable object types in the data warehouse by selecting/clearing the availability check-boxes for each object type.

OBJECT TYPE NAMES

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



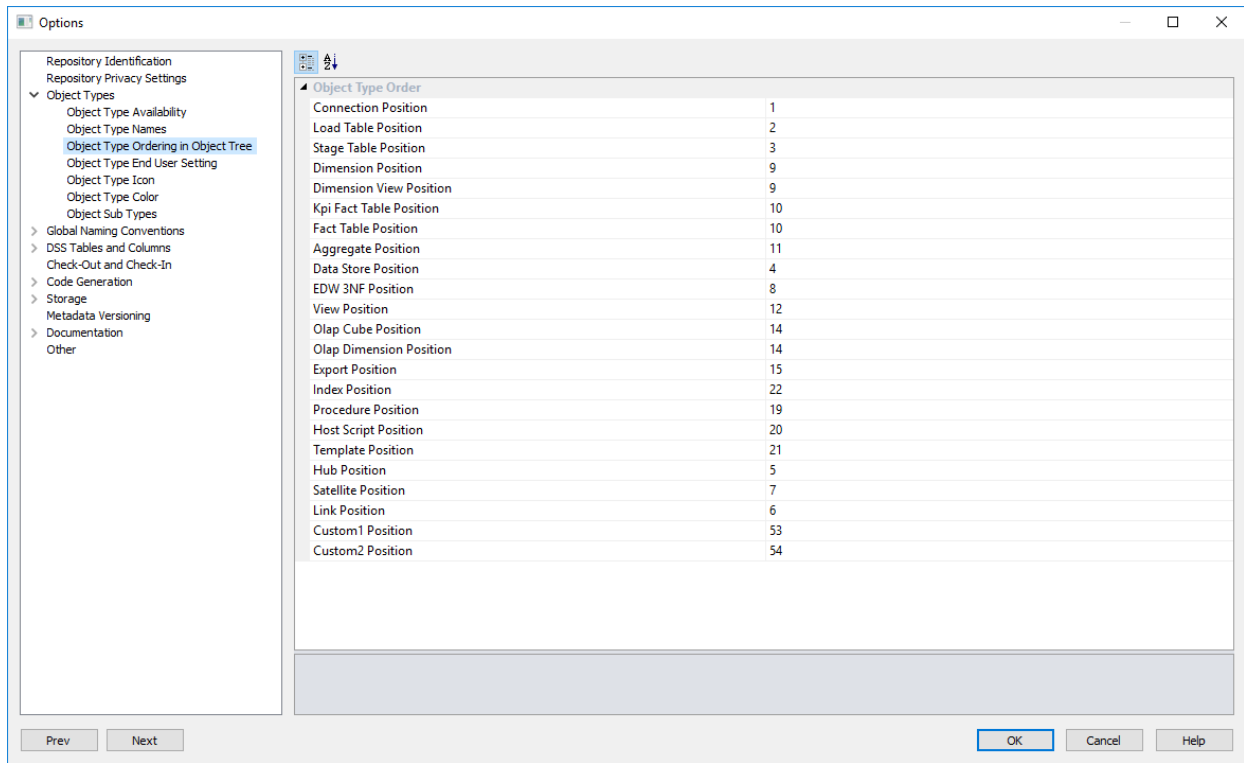
Object Type Name

Set the desired **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 enables users to set the **ordering** in which the object types appear in the object tree.

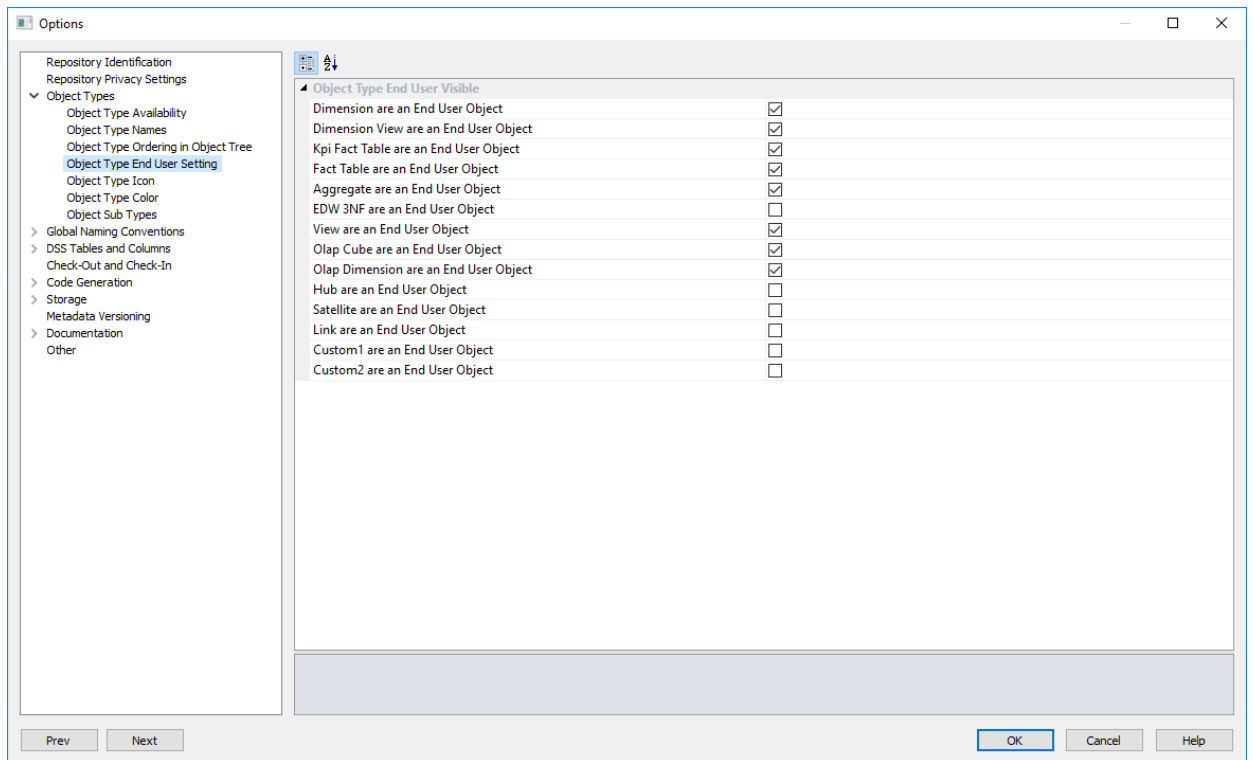


Object Type Order

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

OBJECT TYPE END USER SETTING

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



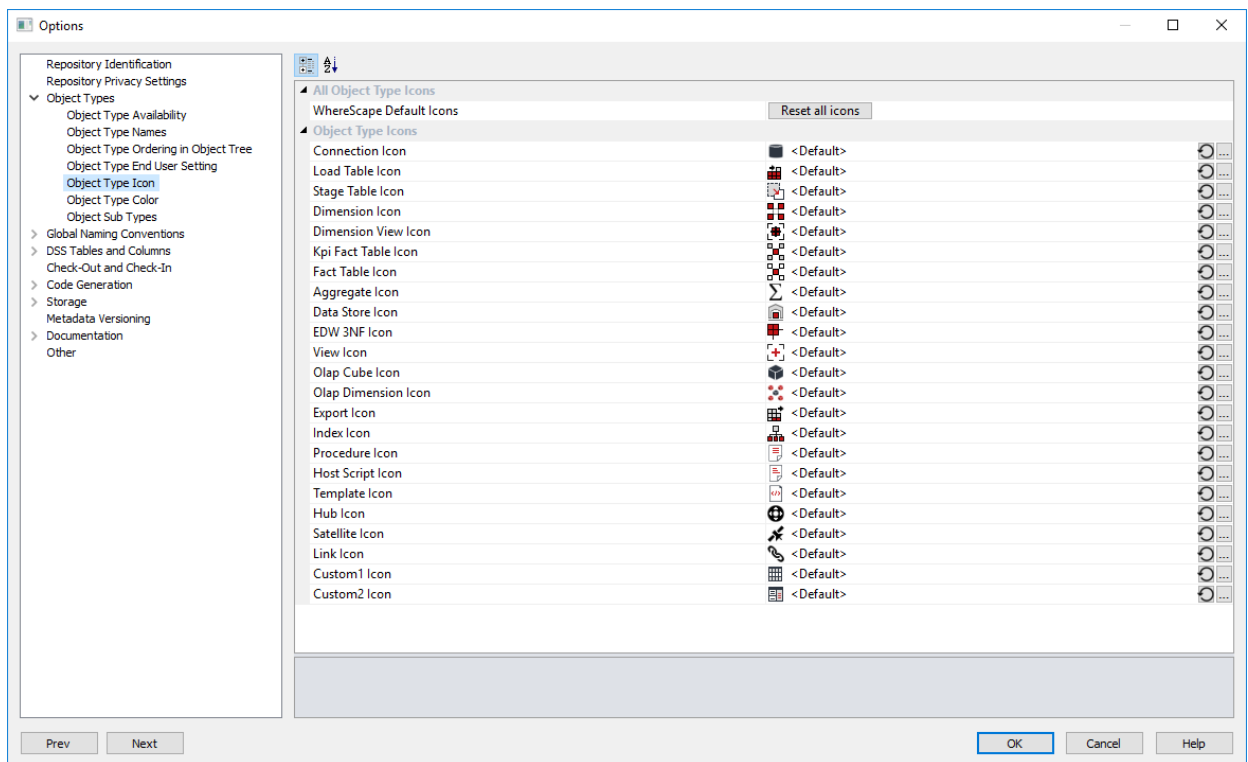
Object Type End User Visible

Set each object to make each object type an end user object.

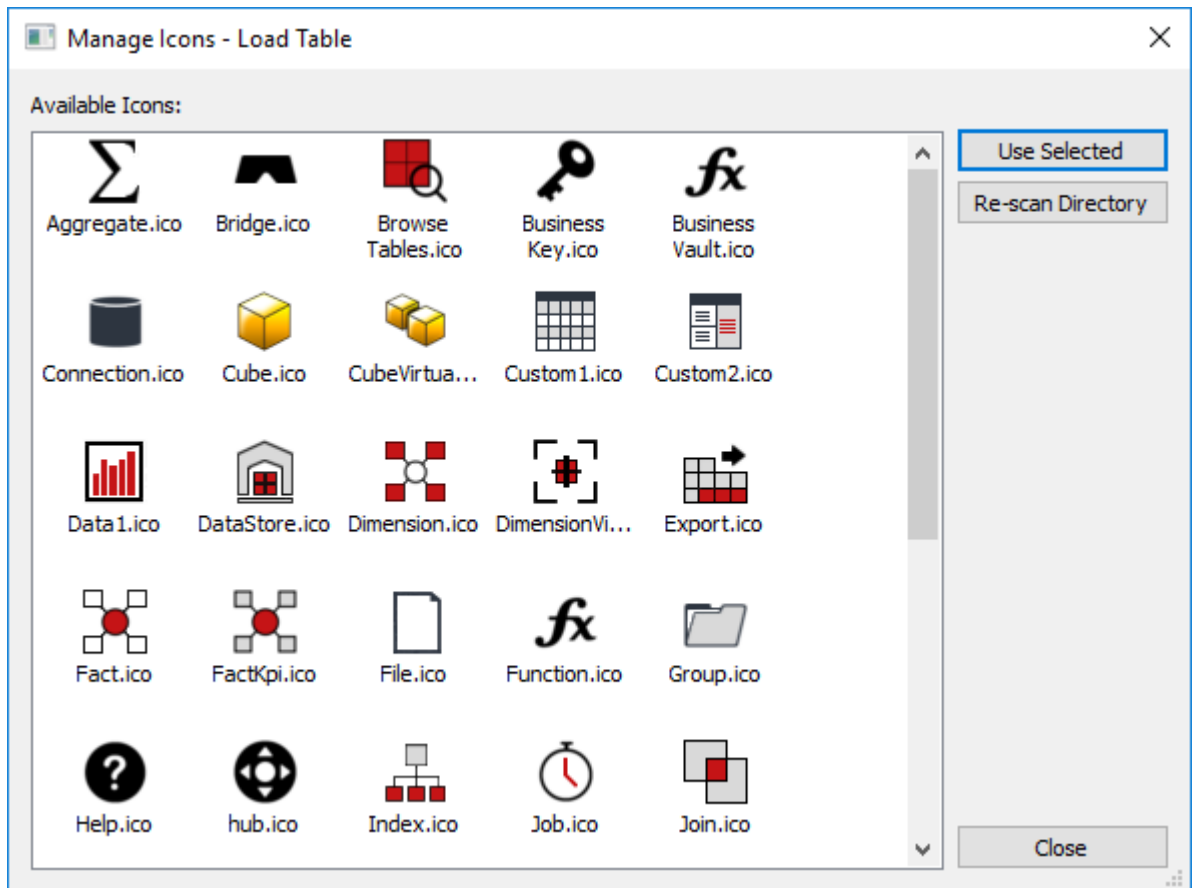
OBJECT TYPE ICON

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

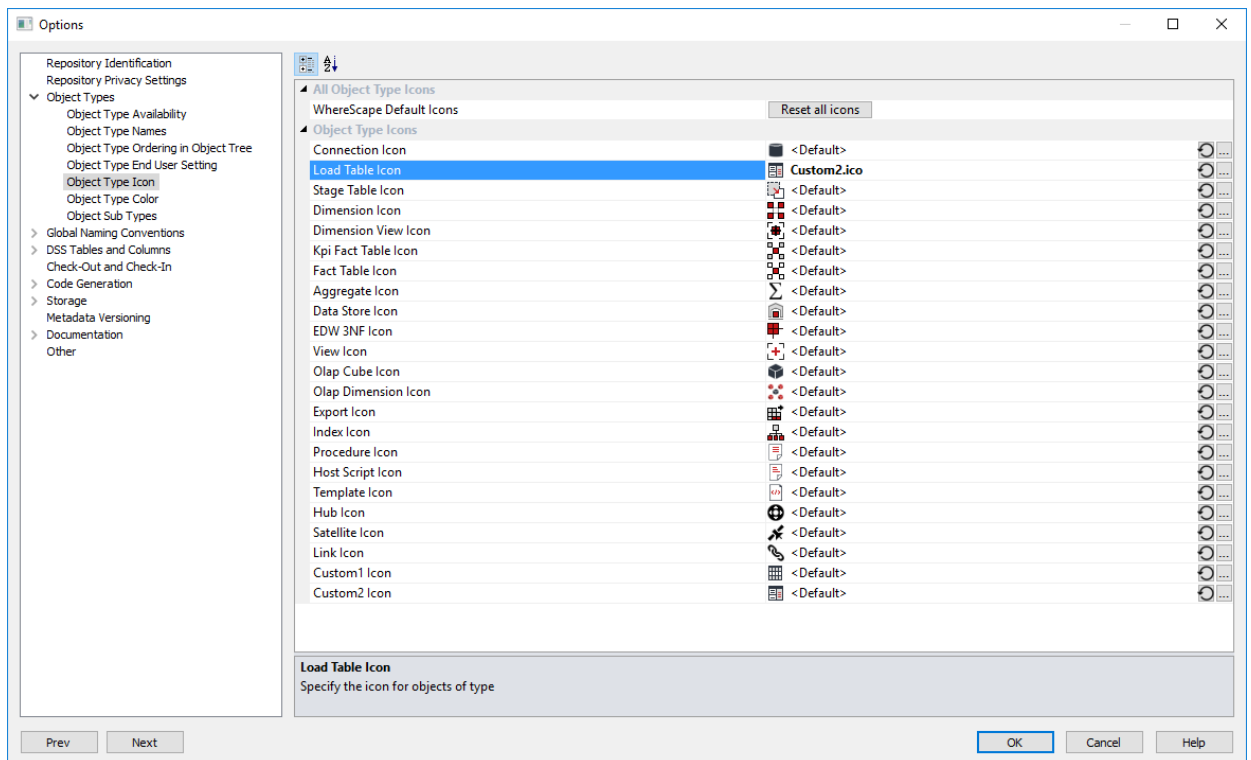
- 1 Create an 'Icons' folder in the WhereScape RED install directory, if it doesn't already exist. For example:
C:\Program Files (x86)\WhereScape\Icons
- 2 Place custom '.ico' files in the Icons folder.
- 3 In RED, select **Options>Object Types>Object Type Icon**.




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



- 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.

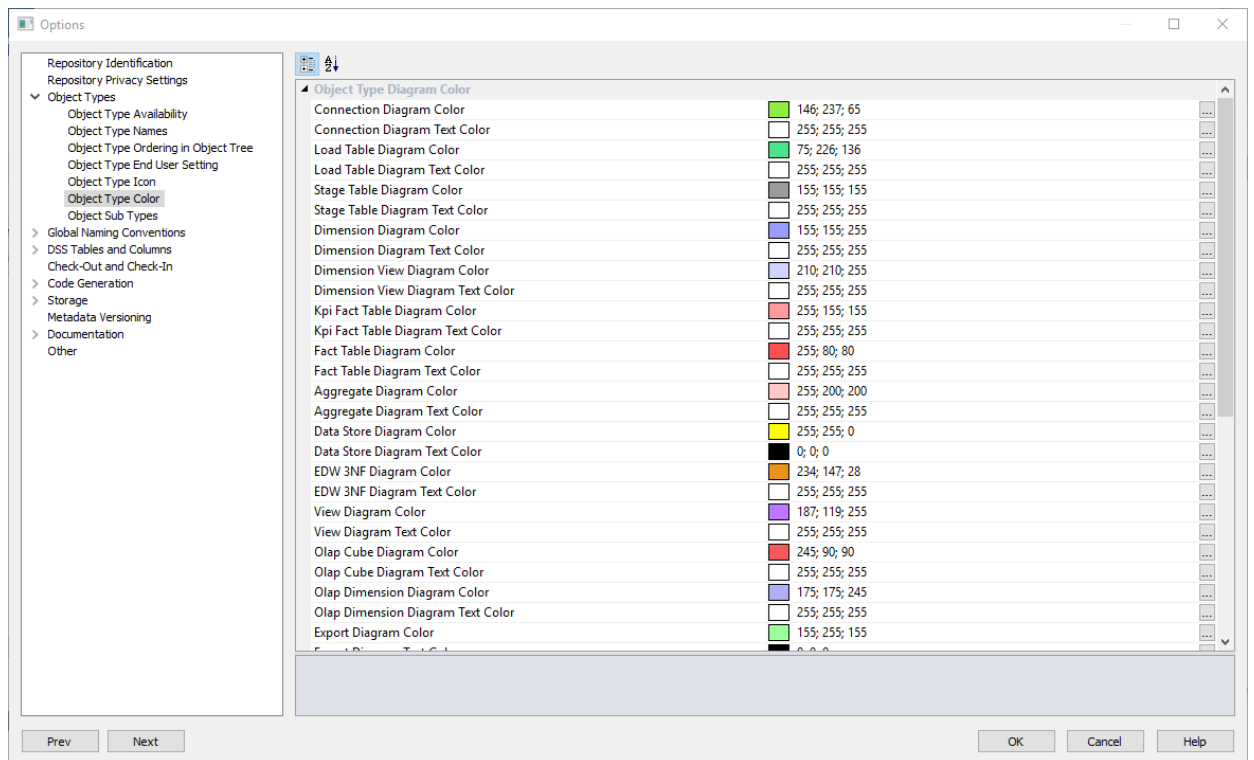
Reset All Icons

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 enables users to set the **diagram colors** for each object type.

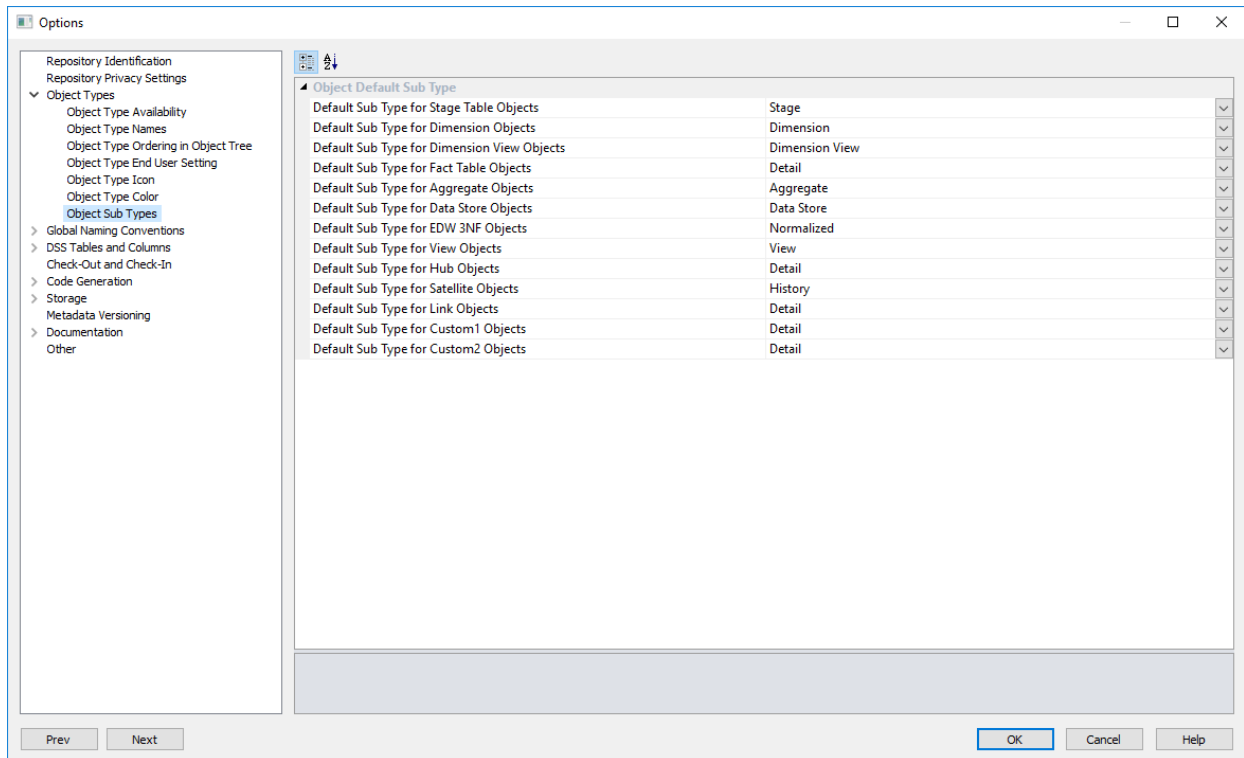


Object Type Diagram Color

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

OBJECT SUB TYPES

This option enables 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 Tools>Options.

Dimension Type ✕

Four methods are provided for managing dimensions. Please select the desired method. ?

1. Normal. The dimension is updated based on a business key, with new records being added if required. All columns except the business key can change. Normal
2. Slowly changing. Changes in the values of selected columns result in new dimensional records being created. In all other respects the same as Type 1. Slowly Changing
3. Previous data retained. The previous values of selected columns are stored in additional columns. In all other respects the same as Type 1. Previous values
4. Date Ranged. The source system provides a date ranged business key. Similar to Type 2 except that we deal with the record as a whole and the dates are provided. Date Ranged

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

☐
Dimension dim_customer_changing
✕

- Properties
- Storage
- Override Create DDL
- Language Mapping
- Purpose
- Concept
- Grain
- Examples
- Usage
- Notes

Table Name: Table Type: Changing Dimension ▼

Unique Short Name:
(maximum 22 characters)

Business Display Name (EUL):

Description:

Update Procedure: (None) ▼ Rebuild

Custom Procedure: (None) ▼

Get Key Function: (None) ▼ Edit Mnemonic (EUL):

Timestamps

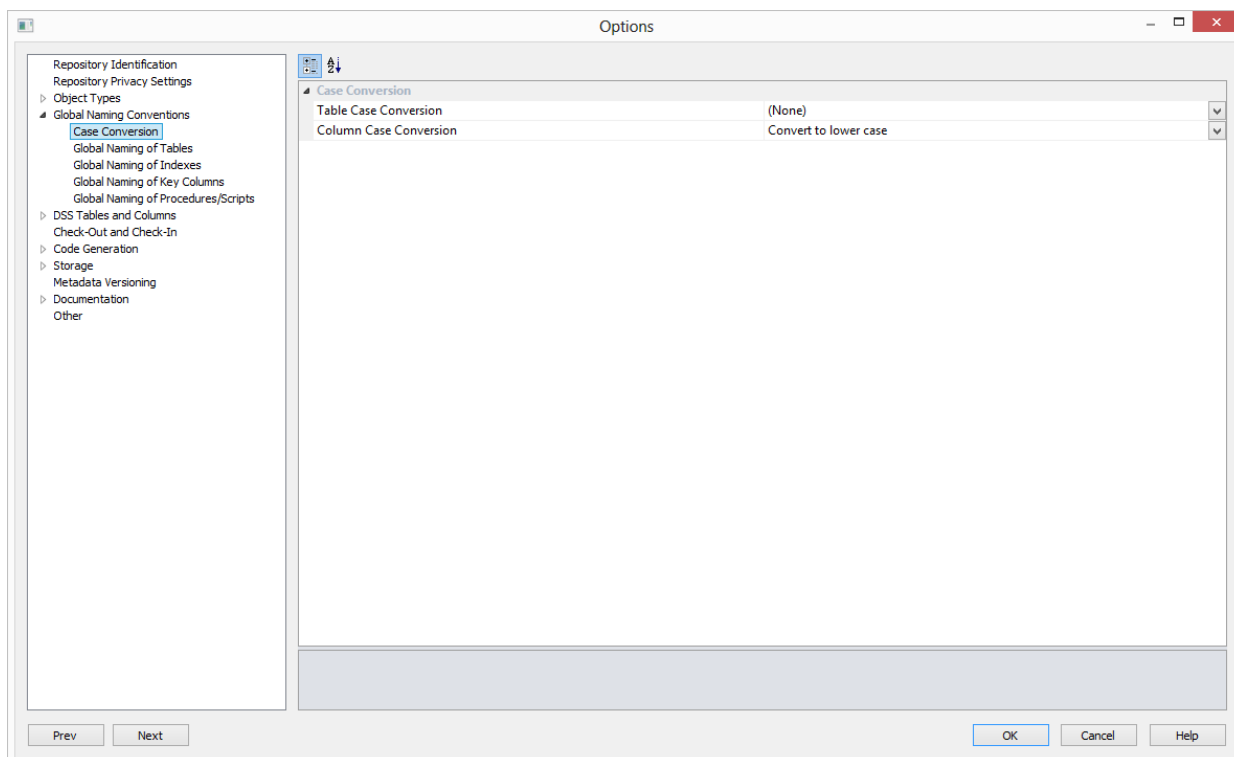
| | | |
|---|---|---|
| Metadata Structure Changed: <input style="width: 95%;" type="text" value="2015-09-30 16:03:46.133"/> | Database Created: <input style="width: 95%;" type="text"/> | Database Altered: <input style="width: 95%;" type="text"/> |
|---|---|---|

OK
Cancel
Help

SETTINGS - GLOBAL NAMING CONVENTIONS

CASE CONVERSION

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

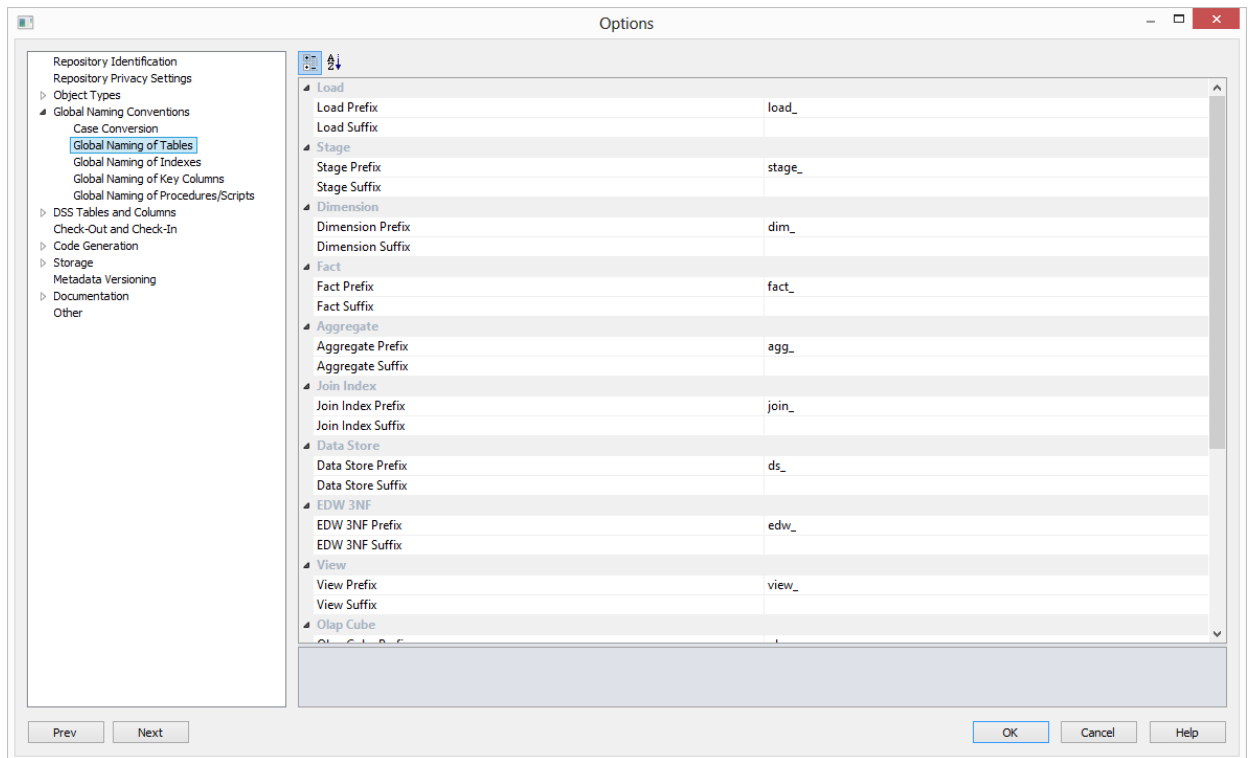


Case Conversion

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.



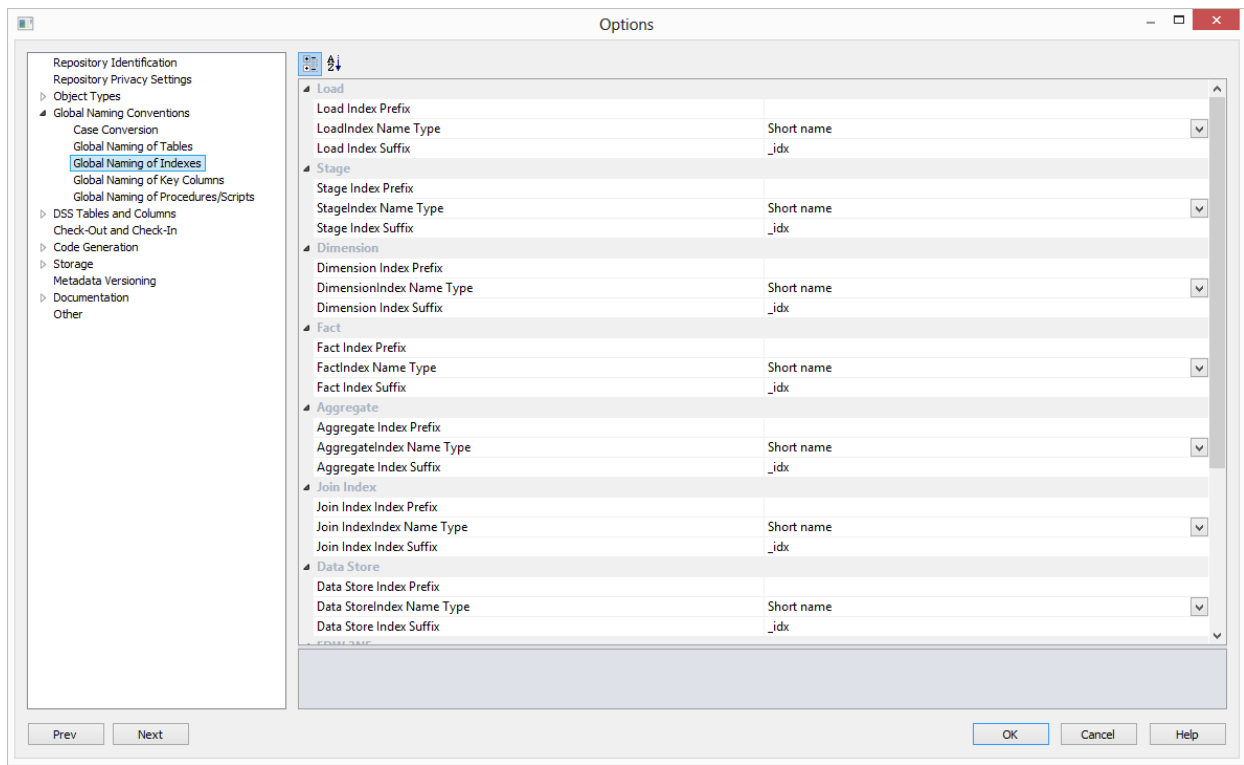
A prefix and/or a suffix string can be applied to an object name. Within Oracle and IBM DB2, a table name may be a maximum of 30 characters long, so these pre and post fix strings should not be more than eight characters long (WhereScape RED short names are a maximum of 22 long in Oracle and SQL Server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

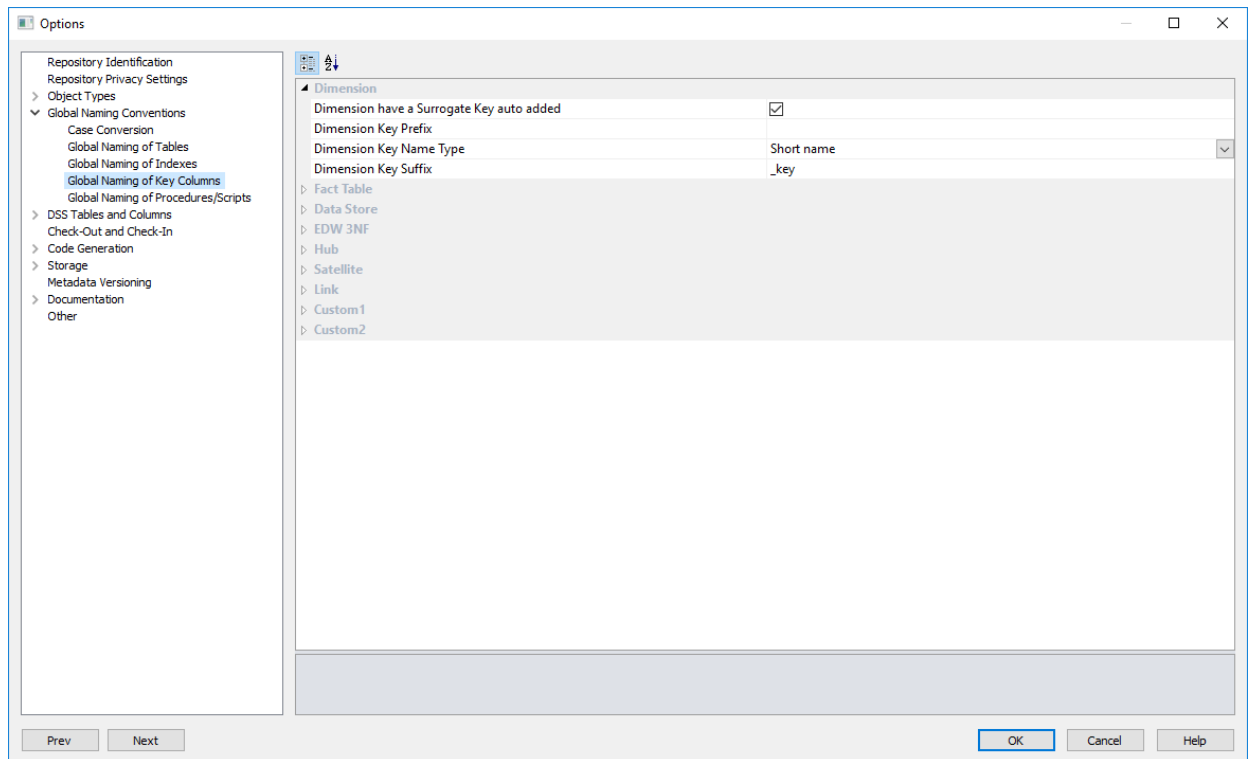
In addition to the naming specified above, WhereScape RED will add up to a further 3 characters to the end of the index name. These additional values will be "_0" through "_99", or "_A" through "_Z", or "_PR". When a new index is manually added, it will have the additional value of "_x" by default. This 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 model table would have indexes generated using the short name and with a suffix of "_idx". Therefore a model_sales model table would have indexes such as model_sales_idx_x.

| Ultimate suffix | meaning |
|-----------------|-------------------------|
| _0 | artificial key |
| _A | primary business key |
| _B through _Z | secondary business keys |
| _PR | primary index |

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 tables, WhereScape RED will build an artificial (surrogate) key for the table if surrogate keys are enabled.

- The naming convention for the surrogate key can be set through the same menu option as above.
- **Prefix** and **suffix** values can also be added.
- There is a choice between the inclusion of the **full table name**, **short name** or **base 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**": for example your load_customer table would generate a key called **dim_customer_key** if it was dragged into a dimension drop target.

To have a table with non identity columns as surrogate keys, you can set the table's **Data Type to integer**. During the procedure generation this will create a logic that associates a sequential number to the artificial key of the dimension when a new row is inserted into the table.

- The example above displays the defaults for **Dimension** options but to set these fields on **Fact**, **Data Store** and **EDW 3NF** tables, expand the fields below Dimension to view and set your required options.
- For more on **Artificial Keys** see *Dimension Artificial Keys*, *Data Store Artificial Keys* and *EDW 3NF Artificial Keys*.

Dimension have a Surrogate Key auto added

Set this field if a Surrogate key column is to be added automatically to a table. Default for Dimension is set. Default for Fact, Data Store and EDW 3NF is not set.

Dimension Key Prefix

Key prefix that can be added to a new Dimension Key.

Dimension Key Name Type

Key name type for new Dimension keys. Select between Short name, Full table name and Base name.

Dimension Key Suffix

Key suffix that can be added to a new Dimension key.

Dimension Data Type

Default data type for the Dimension surrogate column definition. Set to Integer if you want a non identity column to be used as the surrogate key.

Dimension Transformation

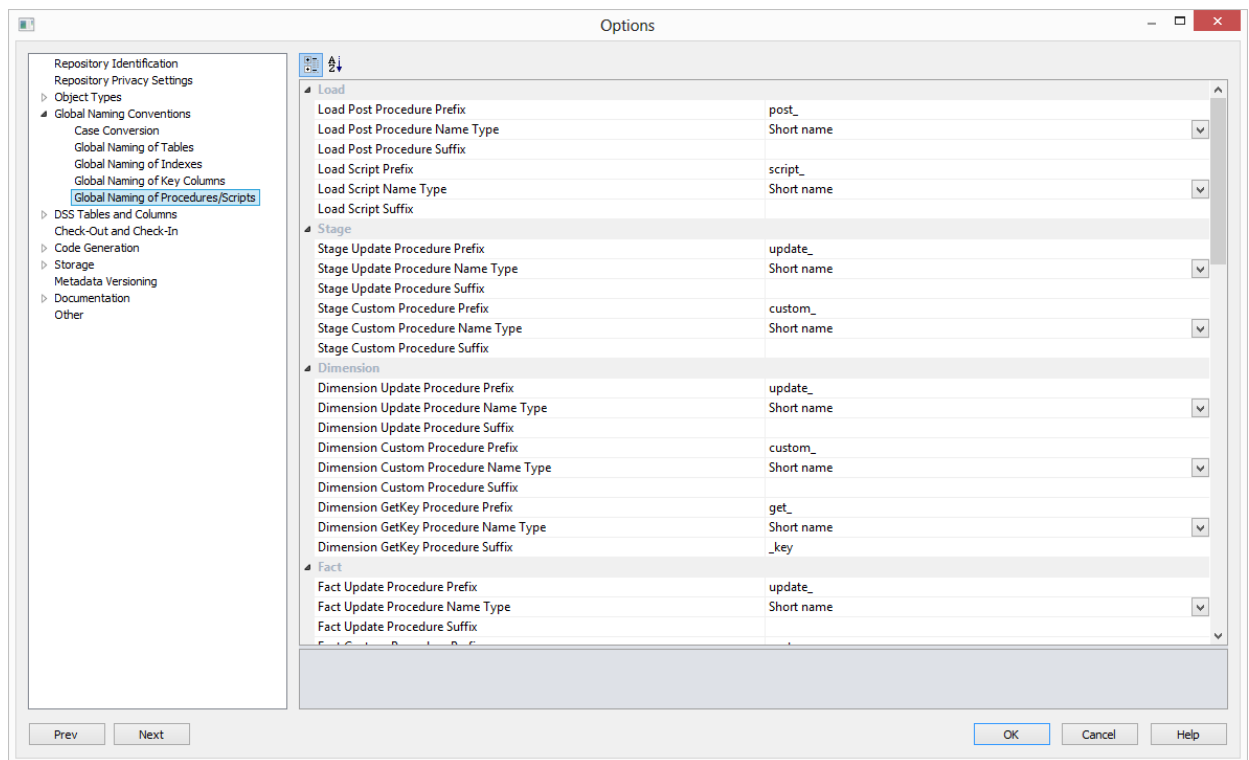
Transformation where a database compliant SQL statement can be used for the surrogate key on new Dimension entries.

GLOBAL NAMING OF PROCEDURES

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 tools/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 stage table called stage_product, then from the example screen above the two possible generated procedures would be called update_stage_product and custom_stage_product.

SETTINGS - 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 |
|------------------------------------|--|
| <code>dss_batch</code> | Not used at this stage. |
| <code>dss_source_system_key</code> | Added to support model tables that cannot be fully conformed, and the inclusion of subsequent source systems. See the section below for more details. |
| <code>dss_fact_table_key</code> | Used in composite rollup fact tables to identify the source fact table that contributed the particular row. |
| <code>dss_create_time</code> | Indicates when a record was created. |
| <code>dss_update_time</code> | Indicates when the record was last updated in the data warehouse. Used in the updating of rollup fact tables and aggregate tables. |
| <code>dss_start_date</code> | Used for model history tables as the start date for a particular version of a row. |
| <code>dss_end_date</code> | Used for model history tables as the end date for a particular version of a row. |
| <code>dss_count</code> | Applied to fact tables. Provides a simple row count variable that can be used by end user tools. |
| <code>dss_current_flag</code> | Used for model history tables. This flag identifies the current record where multiple versions exist. |
| <code>dss_version</code> | Used for model history tables. This column contains the version number of a history 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 model history table. |
| <code>dss_file_name</code> | Identifies a table holding files loaded into load tables. |
| <code>dss_change_hash</code> | Used to identify for a Satellite table. This column identifies the differences in the descriptive columns of a Satellite table which is used for generating the change hash key for creating a Satellite object |

All of these special columns may be renamed through the **Tools>Options>DSS Tables and Columns** menu option. All columns in the screen shot example below (except for `dss_source_system`) can simply be renamed. The two 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`, it is worth considering the fact 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 model table is designed to identify a data source for a model row. Its purpose is to handle 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 of 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 model table to be non conformed. We do however, insist on a standard factory naming 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 model table 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.

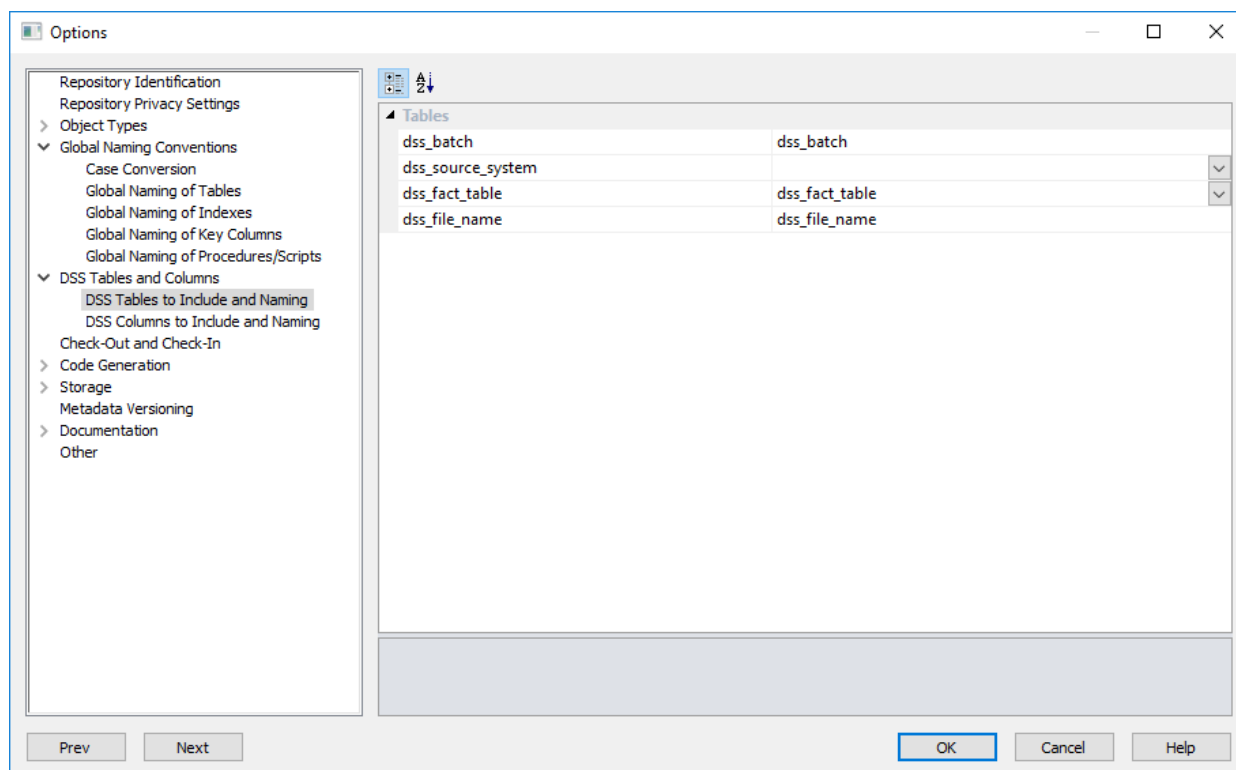
The generated procedure code will always set the key value of this table to 1. Therefore, manual code changes will be required to make use of the functionality that this table offers.

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 model target.
- 2** Rename the `dss_source_system_name` column to match the new table name.
- 3** Delete the last two columns.
- 4** Under the tables properties, change the table type to **Mapping table**. This will prevent the table from being seen as a model table in the documentation.
- 5** Change the `dss_source_system` table name in the screen above, via the tools/options then the ancillary tab.

DSS TABLES

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

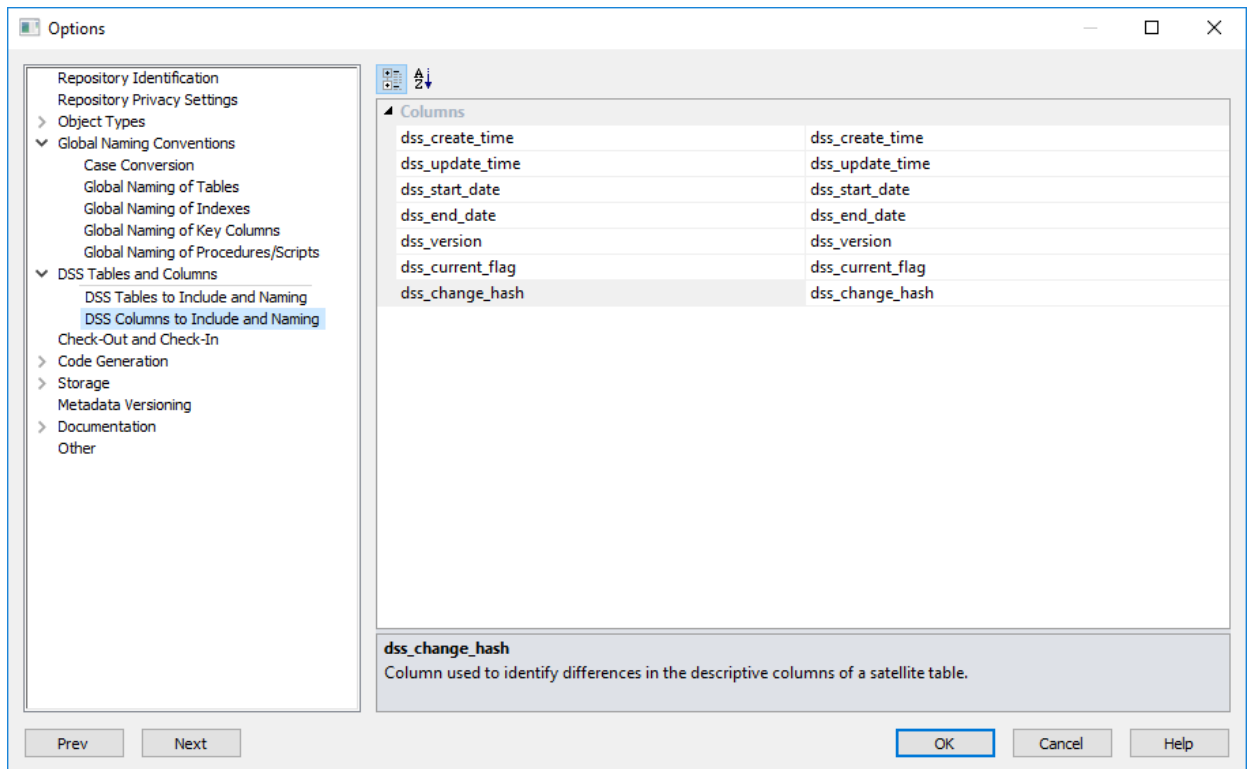


Tables

Set the **DSS Tables**.

DSS COLUMNS

This option enables users to set the **DSS Columns**.



Columns

Set the **DSS Columns**.

dss_create_time

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

dss_update_time

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

dss_start_date

Column used for model history tables. It is used to identify when a model table row was replaced. This is a required field.

dss_end_date

Column used for model history tables. It is used to identify when a model table row was replaced. This is a required field.

dss_version

Column used for model history tables. Is it used to store the version of a model table row. This is required for unique constraints.

dss_current_flag

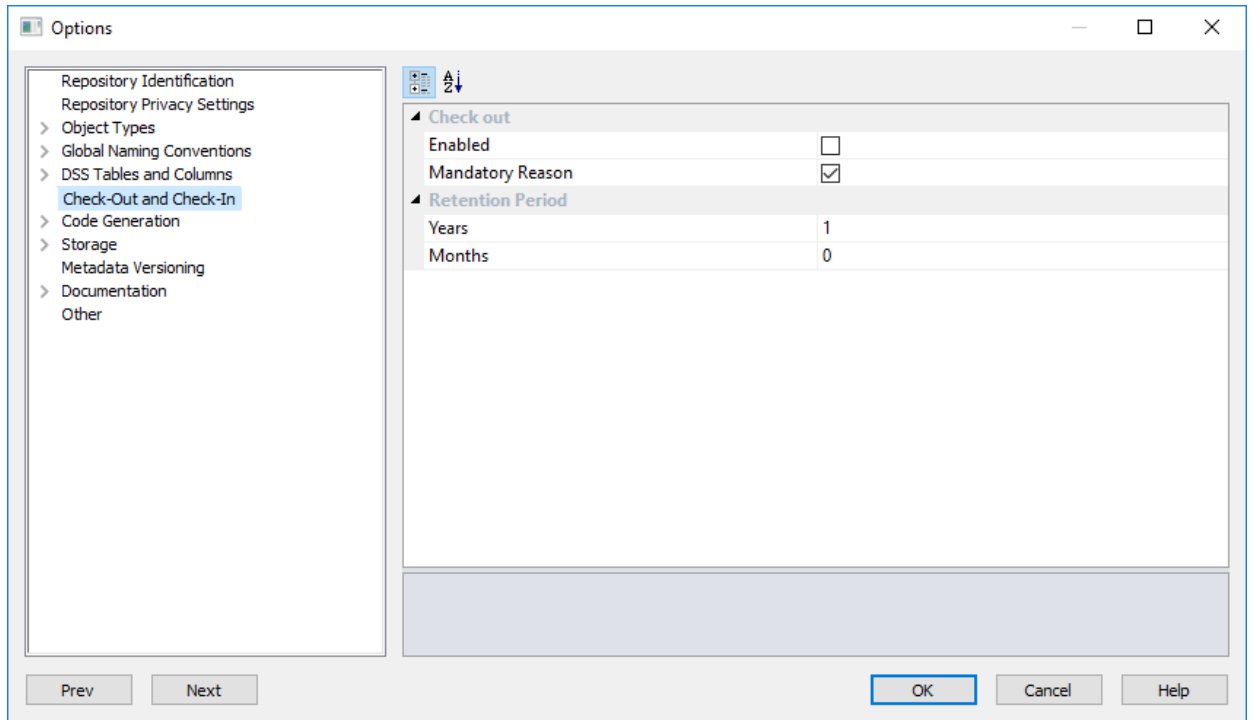
Column used for model history tables. It is used to identify the current model table row. This is a required field.

dss_change_hash

Column used to identify the differences in the descriptive columns of a Satellite table which is used for generating the change hash key for a Satellite object.

SETTINGS - CHECK-OUT AND CHECK-IN

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



Check out

Enabled: Set to True to enable procedures to be checked-in or checked-out.

Mandatory Reason: Set to True if a reason is mandatory.

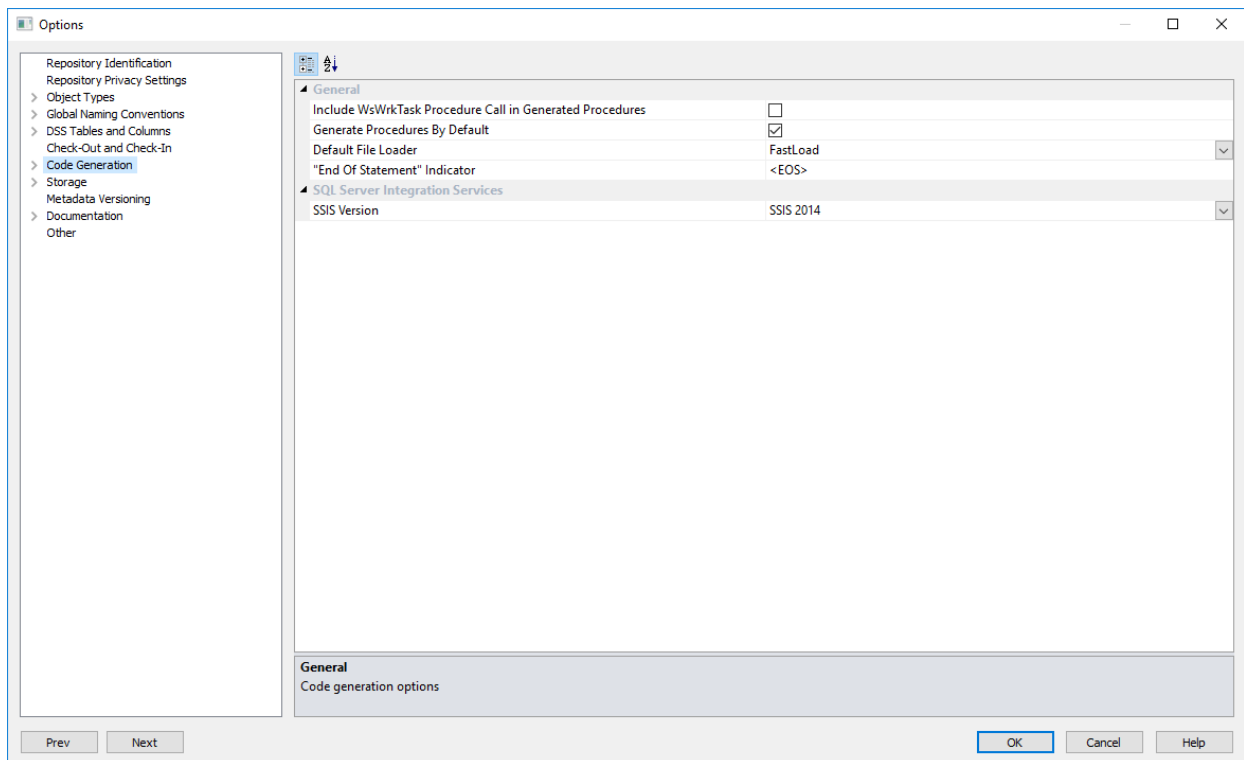
Retention Period

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

CODE GENERATION

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`.

Generate Procedures By Default

Set this option to generate Procedures by default.

Default File Loader

Options for the default file loader are:

- Multiload
- FastLoad
- Load TPT
- Update TPT
- Stream TPT
- No Load

NOTE: When importing a Model from 3D to RED, please select **Load TPT** instead of Fastload as the Default File Loader method. FastLoad is not a valid option for loading Linux files to Teradata.

"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.

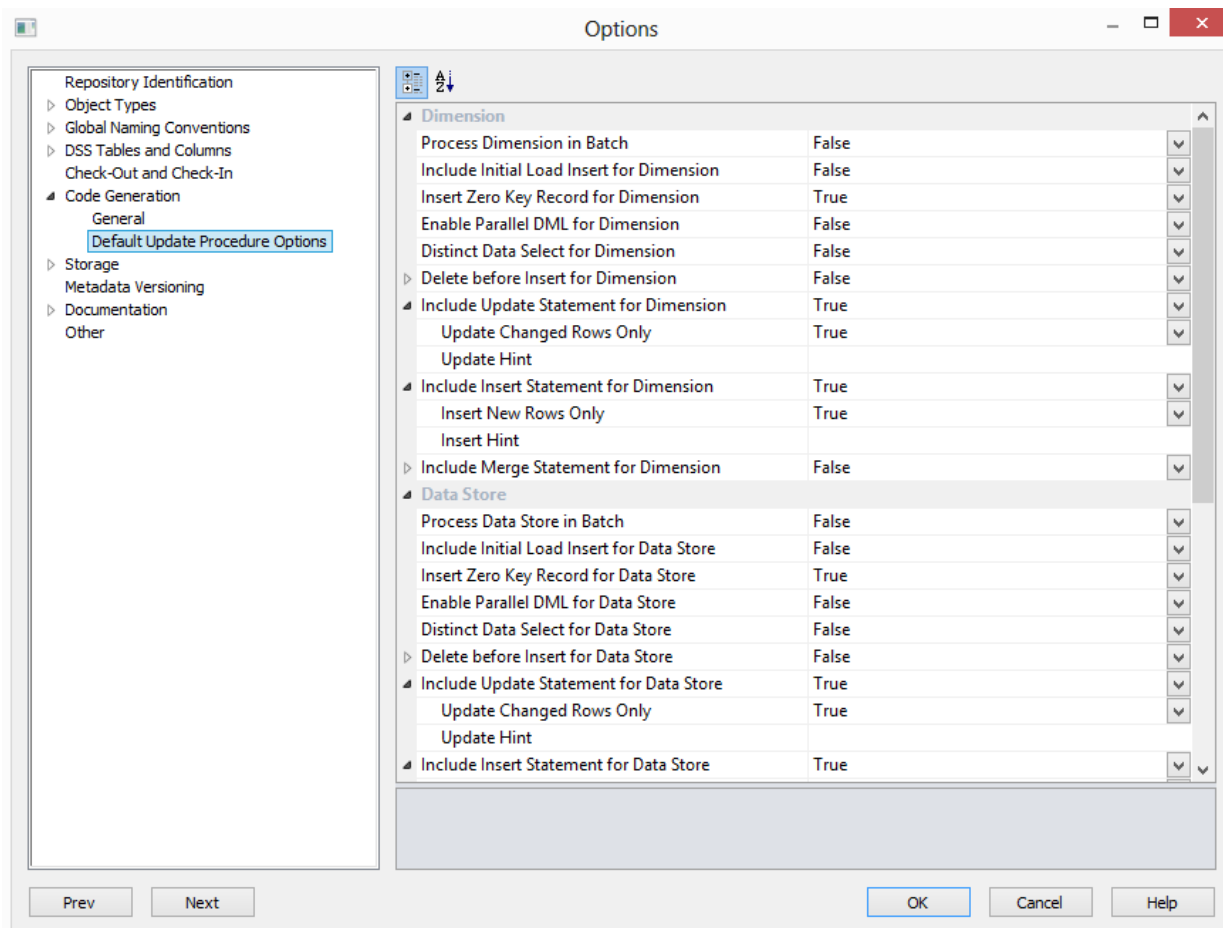
SQL Server Integration Services

SSIS Version

Available version of SQL Server Integration Services. SSIS is not enabled by default for Teradata. 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.



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

The **Include Initial Load Insert** option, when selected, 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.

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

The **Parallel DML** option, when selected, adds all code required to the update procedure for enabling Oracle parallel inserts. **Note:** Oracle only.

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

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

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

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

The **Update Hint** option, when selected, enters a database hint to be used in the UPDATE statement.

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

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

The **Insert Hint** option, when selected, enters a database hint to be used in the INSERT statement.

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

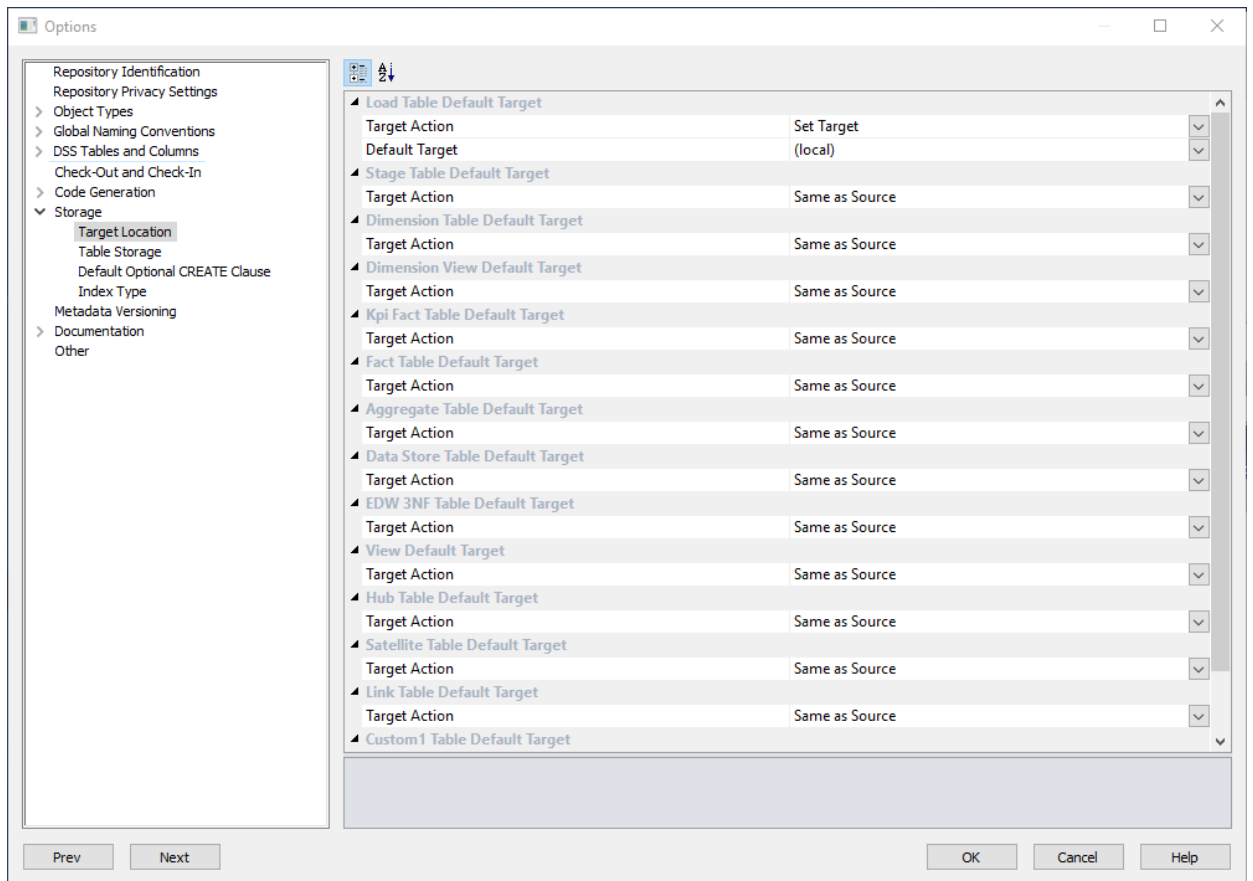
SETTINGS - STORAGE

TARGET LOCATION

Target Location options enable users that are placing objects across multiple databases to set default target locations for new tables.

Default table target locations can be set for the following objects:

- **Load**
- **Stage**
- **Dimension**
- **Kpi Fact**
- **Fact**
- **Aggregate**
- **Join Index**
- **Data Store**
- **EDW 3NF**
- **View**
- **Hub Table**
- **Satellite**
- **Link**
- **Custom**



Target Action

Set Target

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

Default Target

A default target location 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 location, select a **default target** where the new object should be placed as the object is dragged and dropped to the middle work pane.

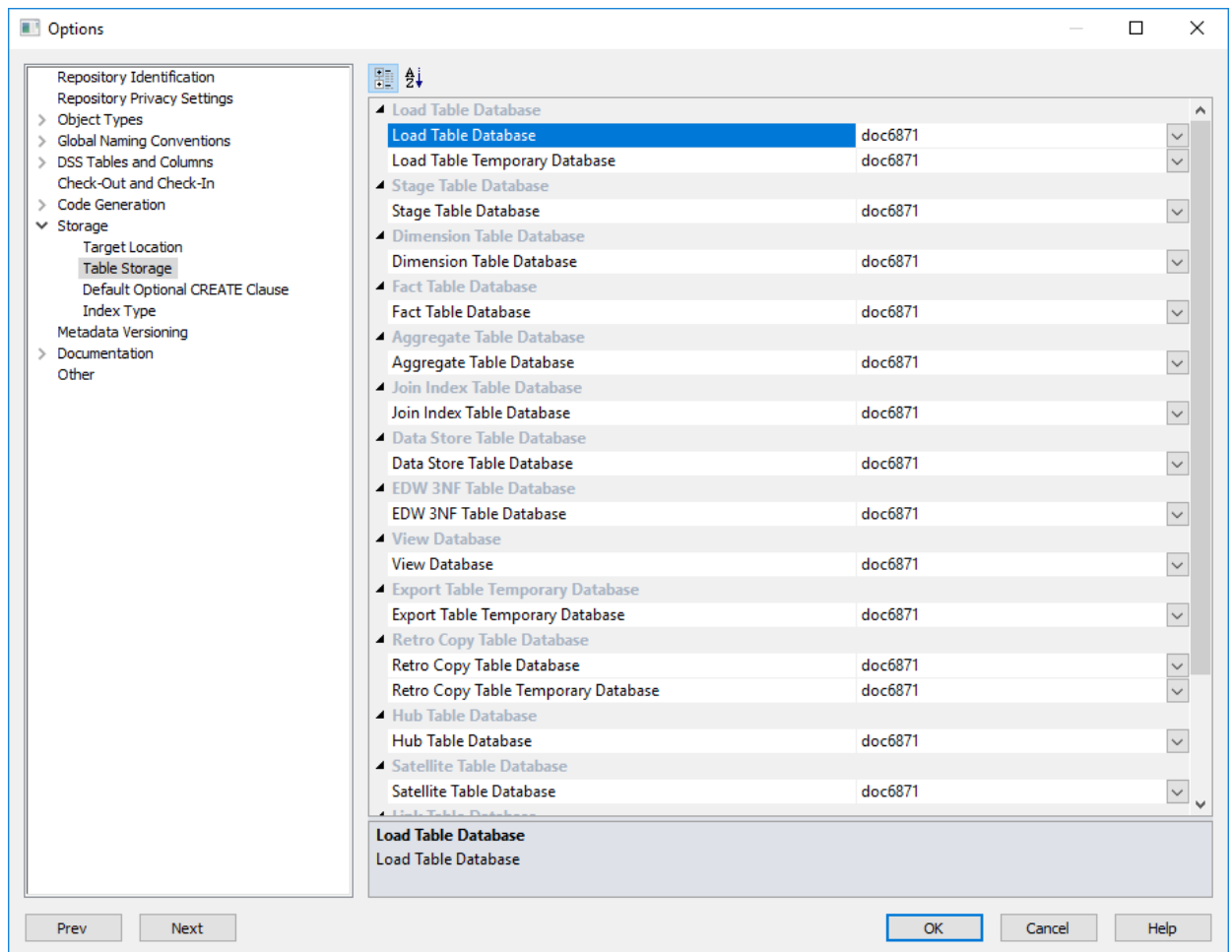
To see more on creating target locations see *Connection to the Data Warehouse* (see "*Database - Data Warehouse/Metadata Repository*" on page 135).

Even though the default target location 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 target location after a table has been created, see *Storage* (on page 182).

TABLE STORAGE

This option enables users to set the **Storage** locations.



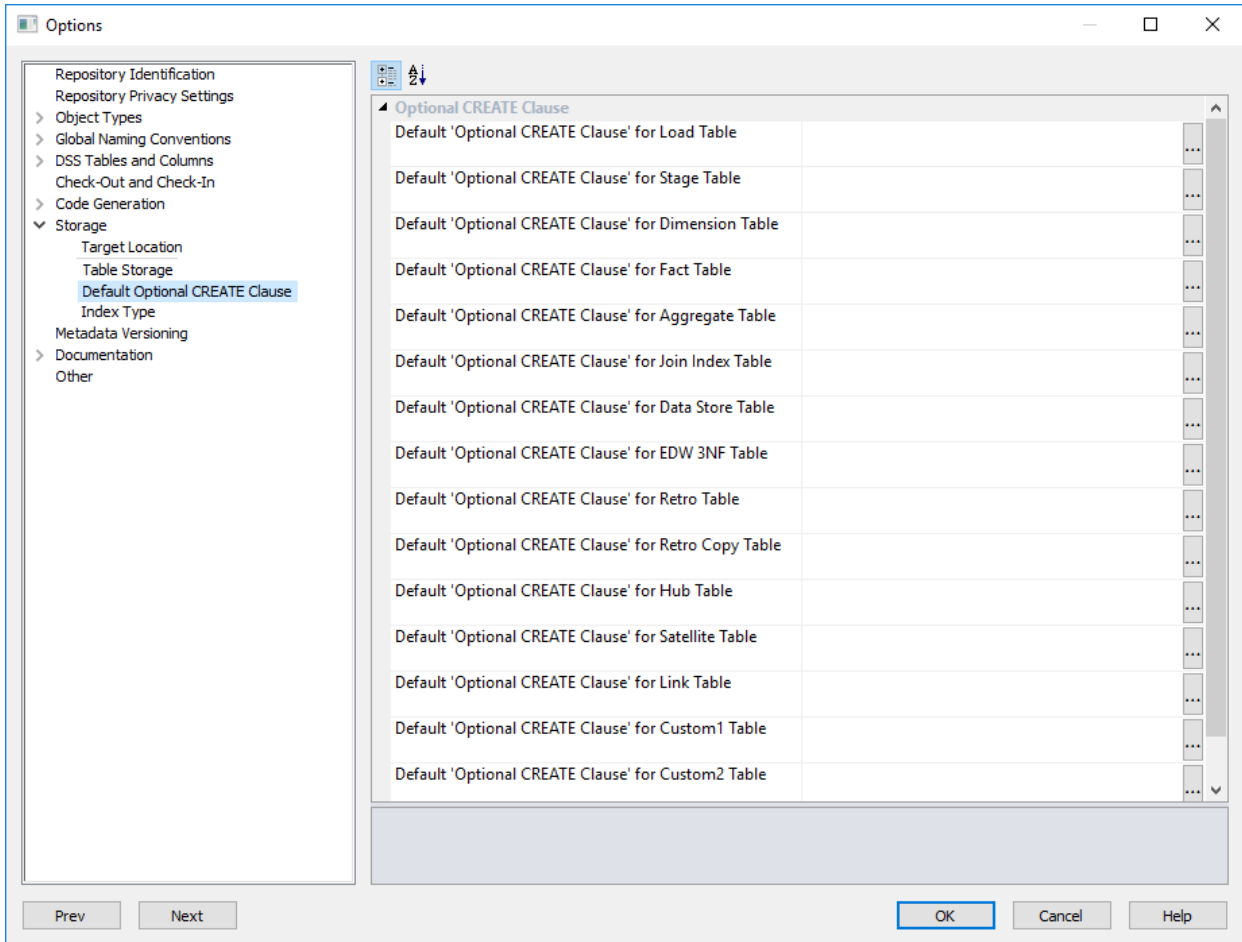
Set the **Storage** locations for each table type.

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 enables you to define a default value for the "Optional CREATE Clause" property of each object type, which is populated when the object is first created.

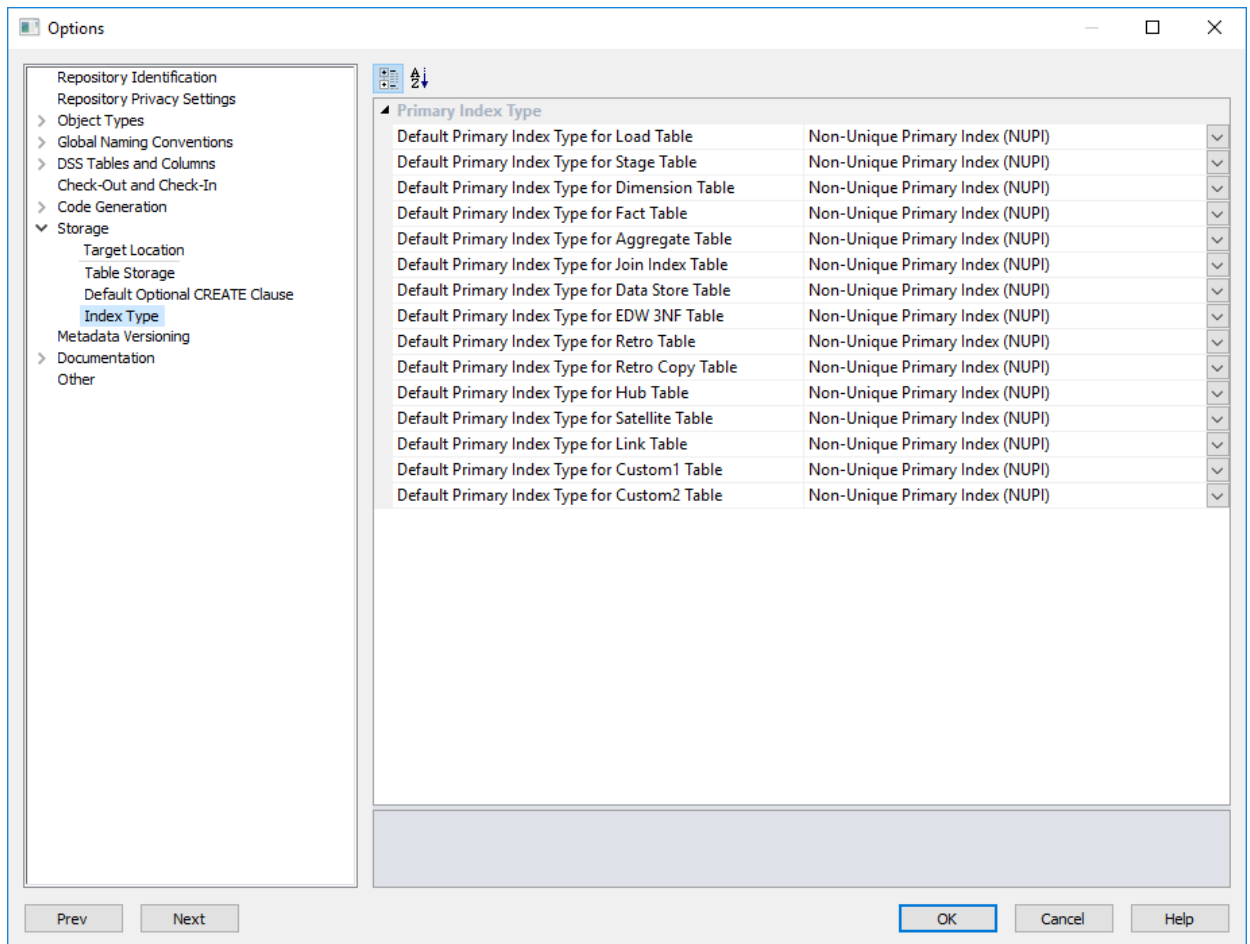
The Optional CREATE Clause text is appended to the DDL CREATE statement when the table is generated.



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.

INDEX TYPE

This option enables you to set the default type of **primary index type** for each table type.



Set the **default primary index type** for each table type.

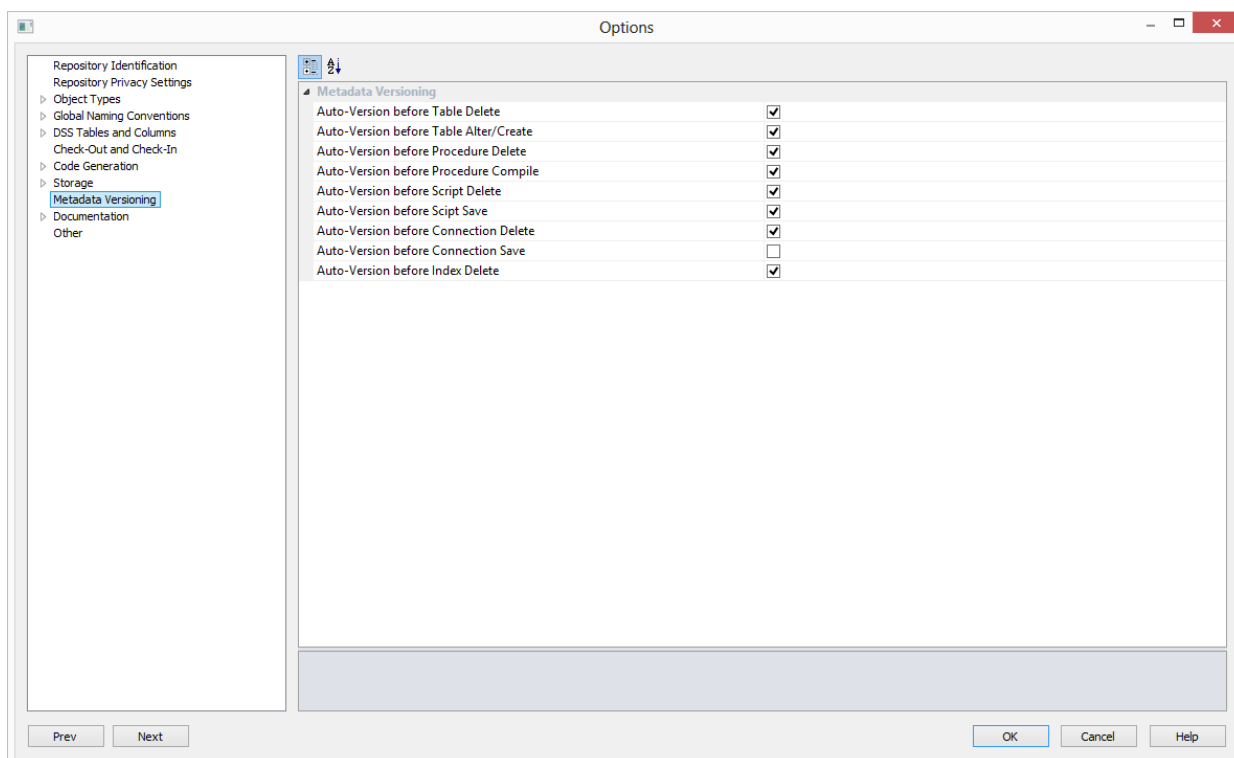
The options are:

- Non-Unique Primary Index (NUPI)
- Unique Primary Index (UPI)
- No Primary Index (NOPI)

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.

SETTINGS - VERSIONING

This option enables users to alter the **Metadata versioning** settings.

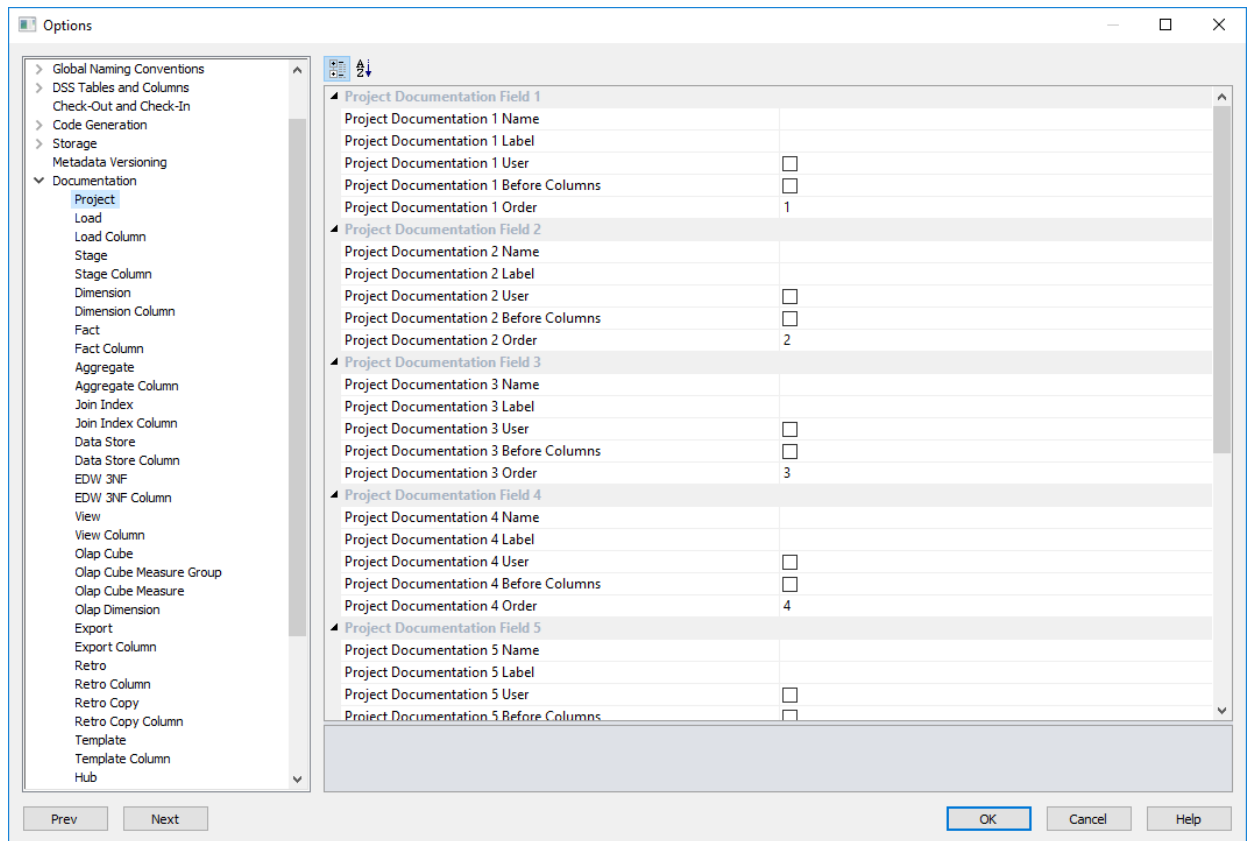


Metadata Versioning

Set each option if you want to auto-version the metadata.

SETTINGS - DOCUMENTATION

This options enables you to alter the **documentation** settings.



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

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

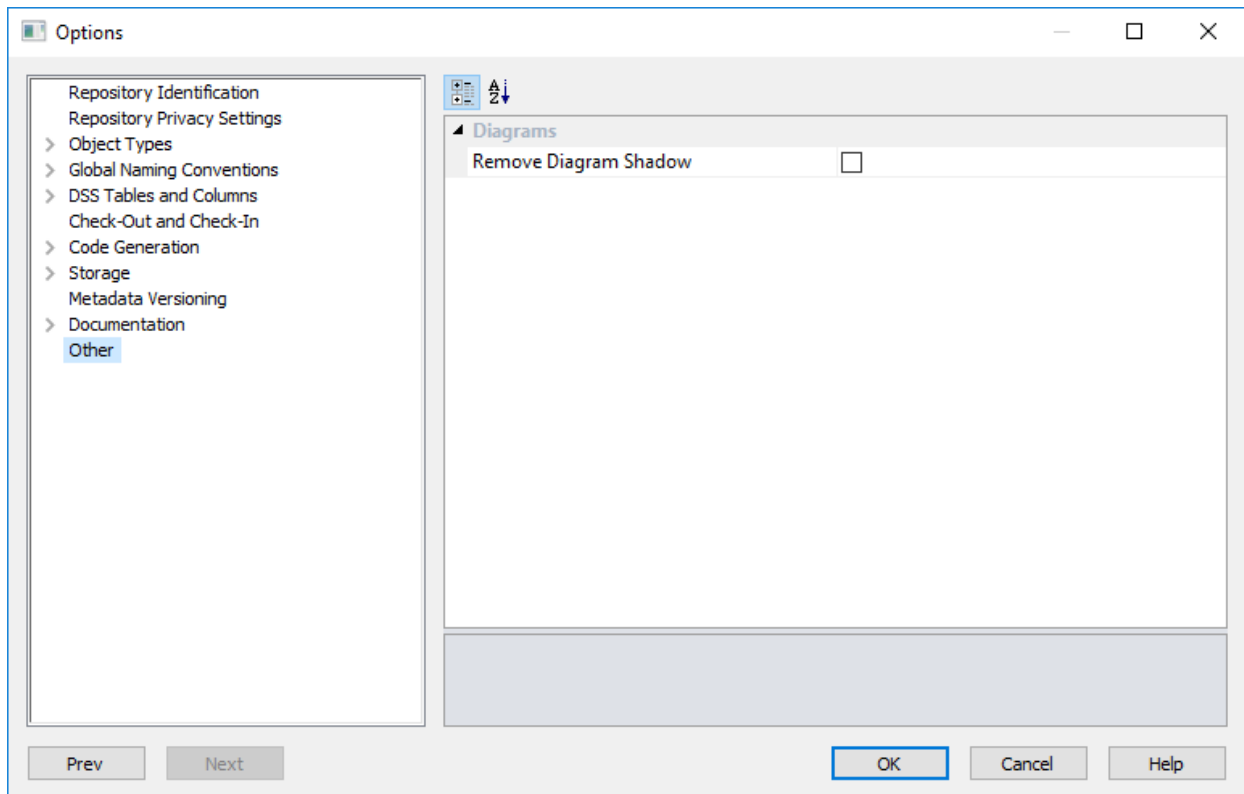
The **Documentation User** defines if the documentation information is visible to end users and included in end user documentation.

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 order that this field appears in the properties dialog tabs.

SETTINGS - OTHER

This option enables you to add or remove shadows in the diagrams.

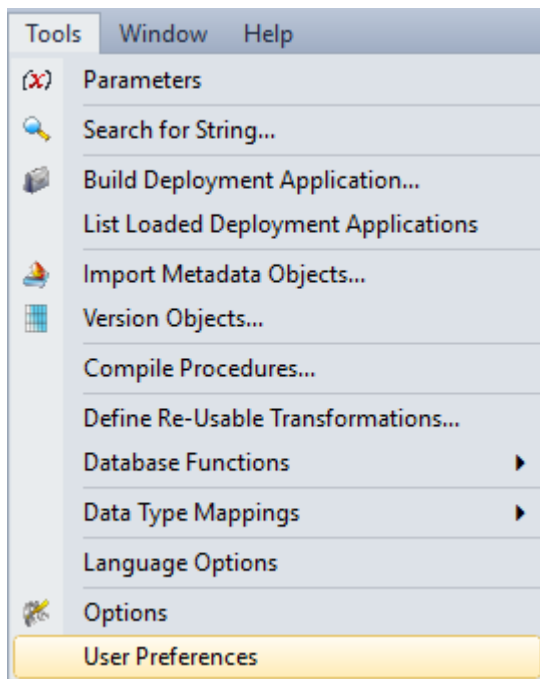


Repository Identification

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

SETTINGS - USER PREFERENCES

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

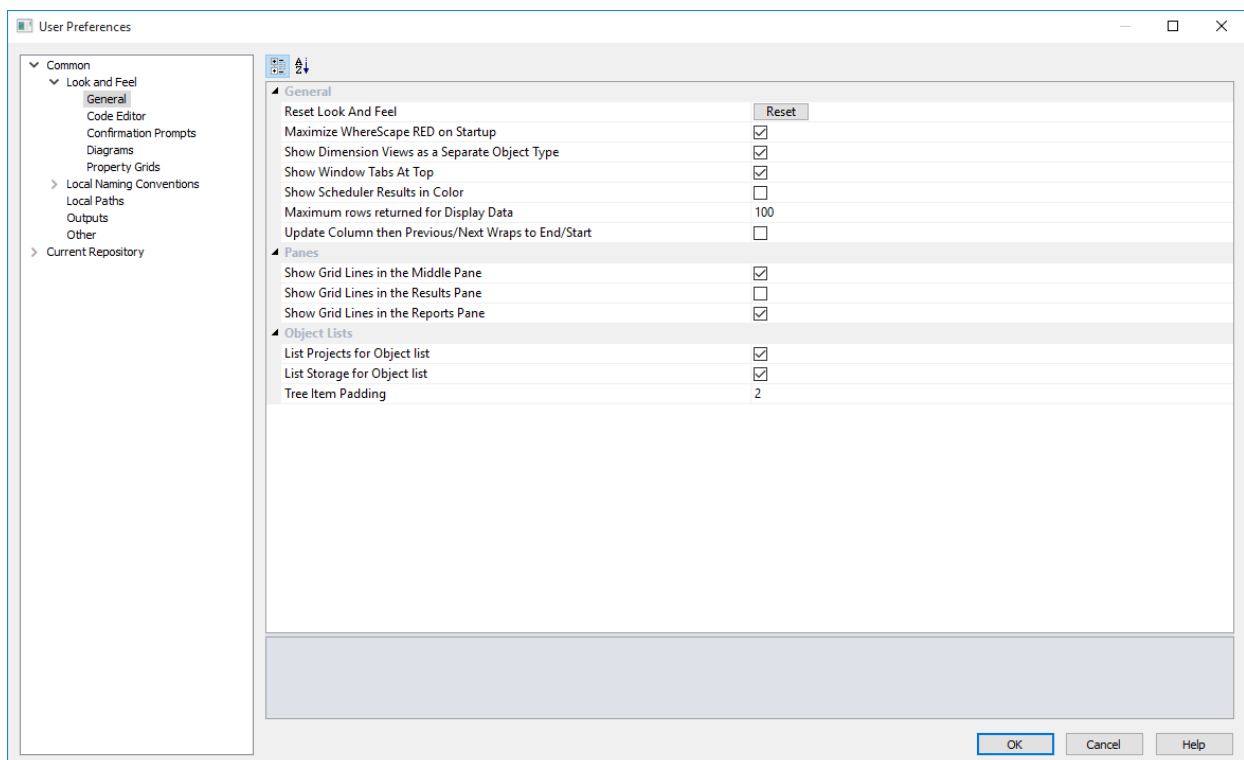


SETTINGS - COMMON

LOOK AND FEEL

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, will start WhereScape RED in full screen mode.

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

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

The **Maximum rows returned for Display Data** option allows you to set 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, will show grid lines in the main work area.

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

The **Show Grid Lines in the Reports Pane** option, when selected, will show 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.

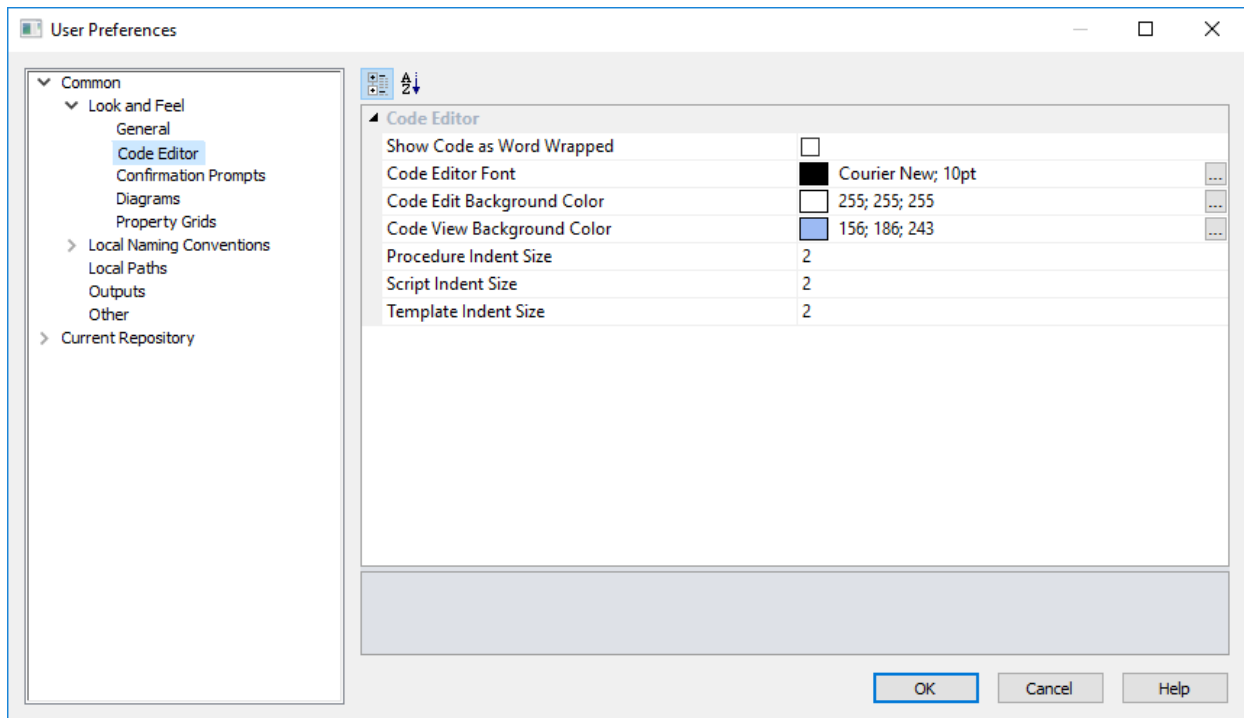
The **List Storage for Object list** option, when selected, shows the storage for each object in the middle pane object list.

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 true, will default to have word wrapping applied to code.

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

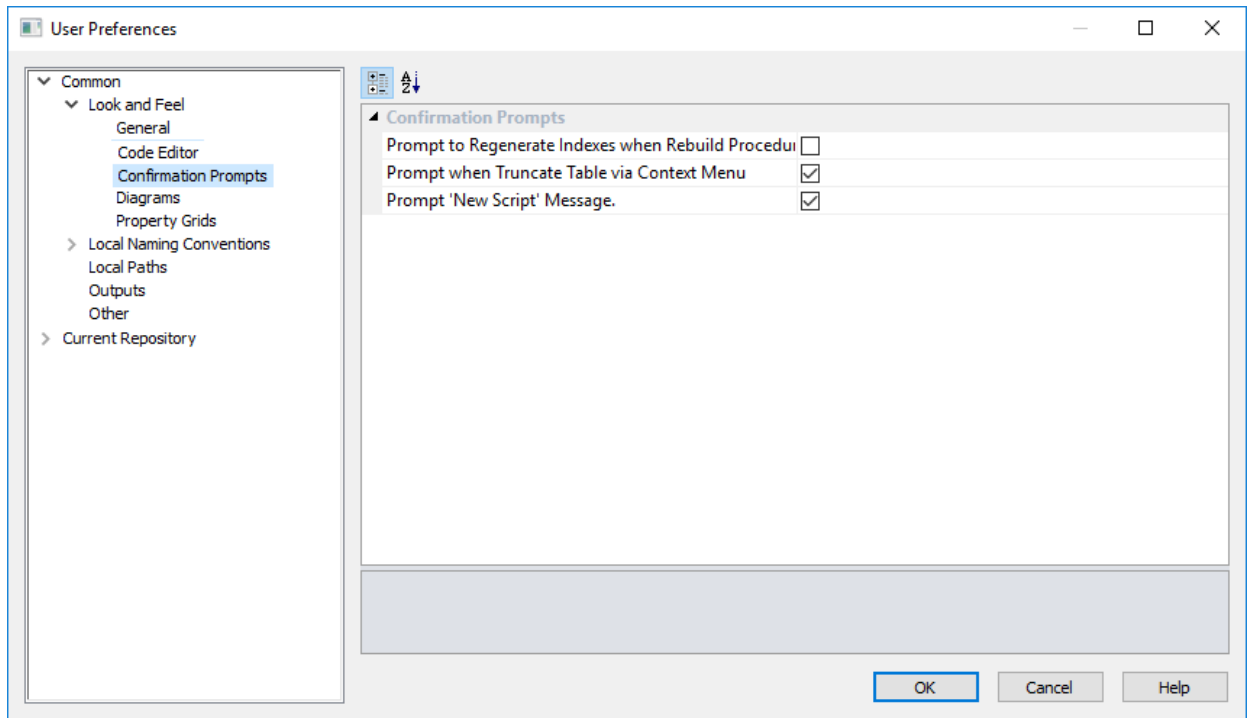
The **Procedure Indent Size** option allows you to specify the number of spaces that are generated when a TAB character is used within the Procedure editor. Permitted range is 2 through 10.

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**.

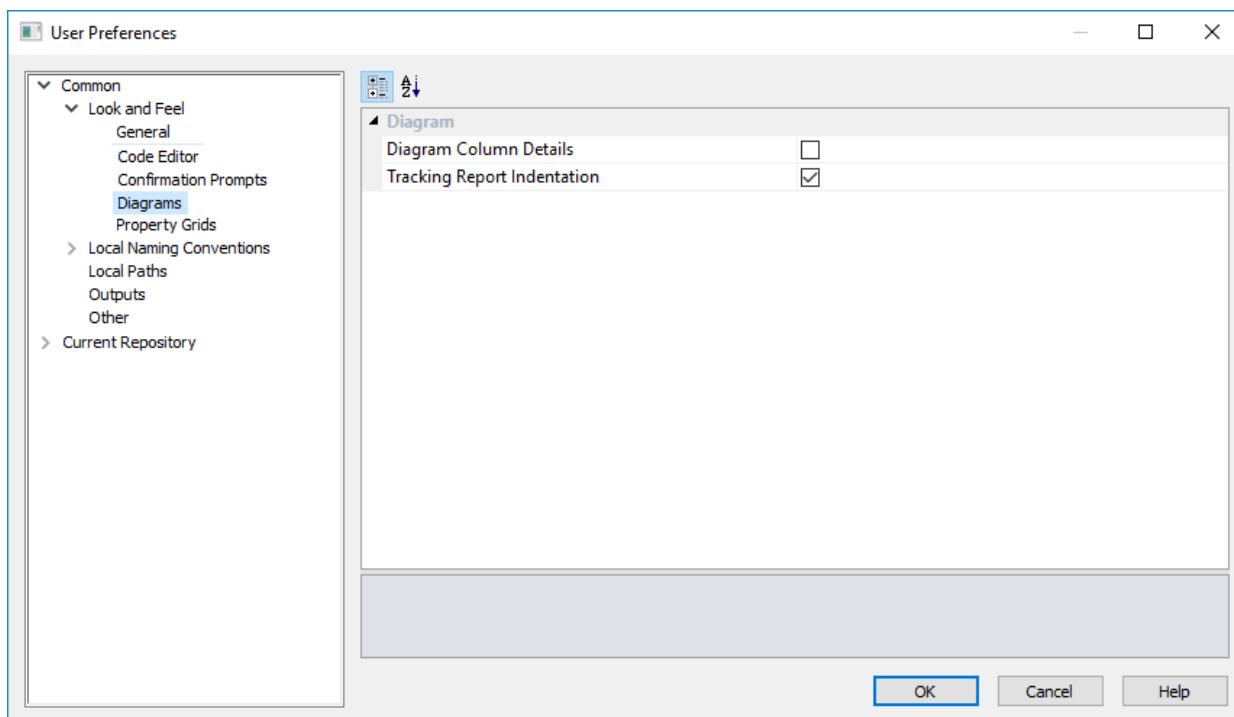


Confirmation Prompts

- **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**.



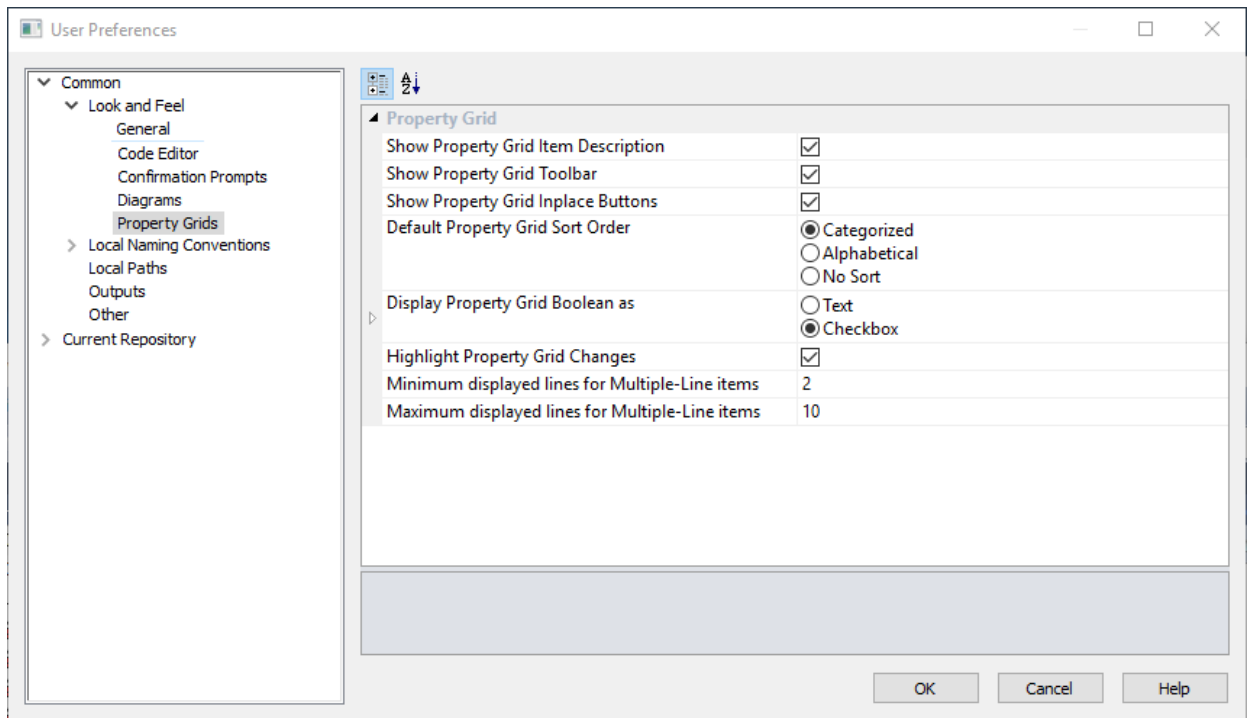
Diagram

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 the **property grids**.



Property Grid

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

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

The **Show Property Grid Inplace Buttons** option, when selected, will show 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.

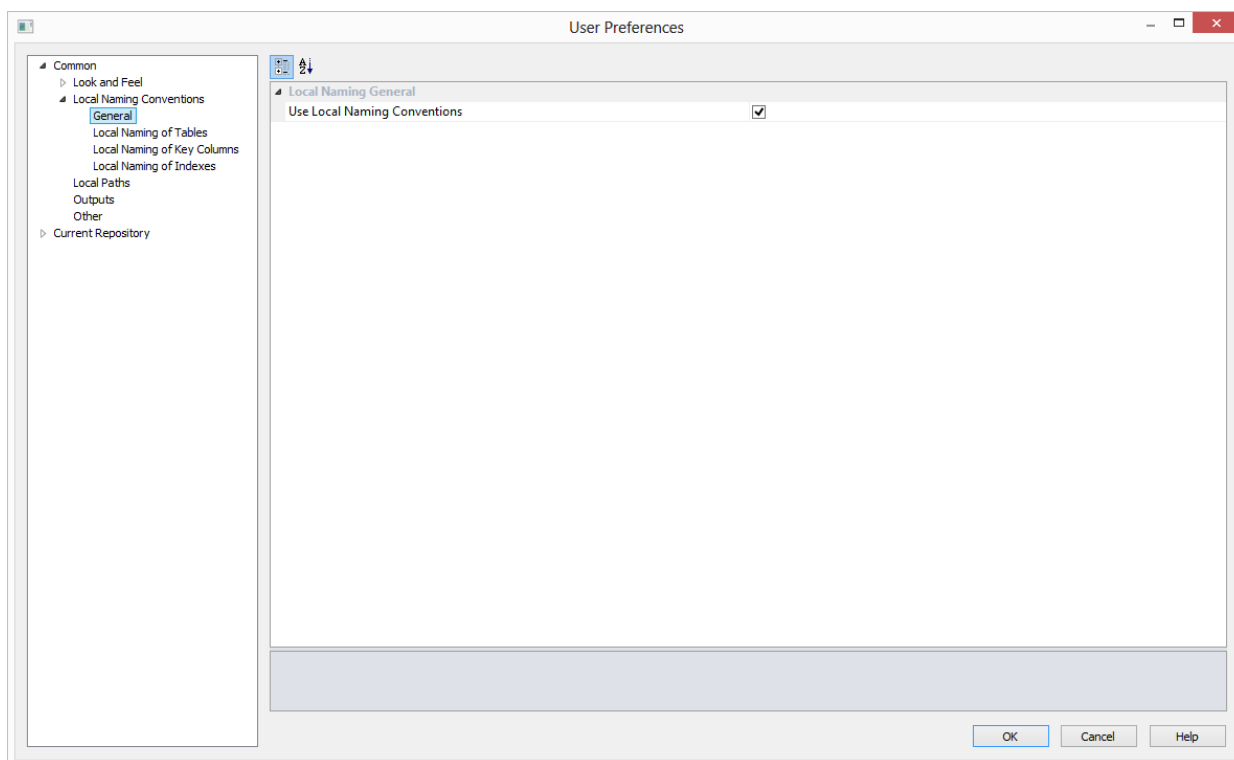
The **Minimum displayed lines for MultiLine items** option, gives the minimum display lines for multi-line inputs.

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

LOCAL NAMING CONVENTIONS

General

This option allows users to set the **local naming conventions**.



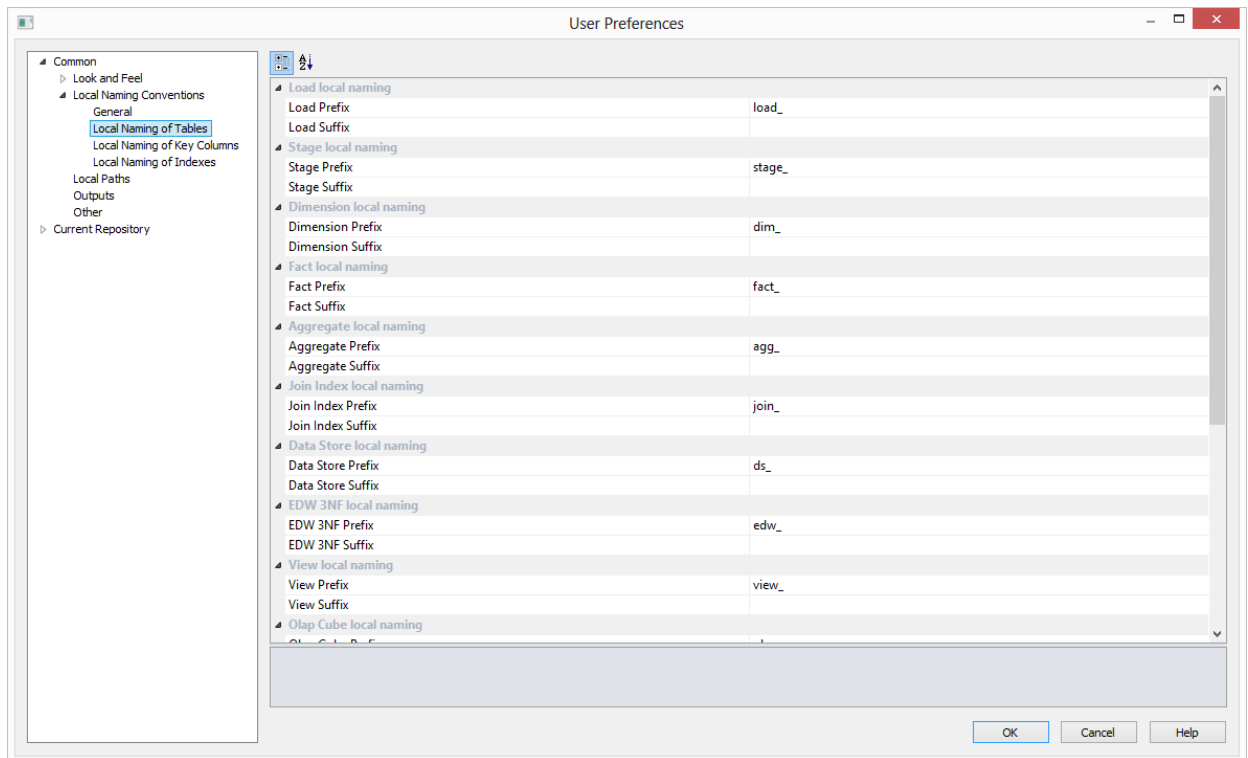
Local Naming General

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

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

Local Naming of Tables

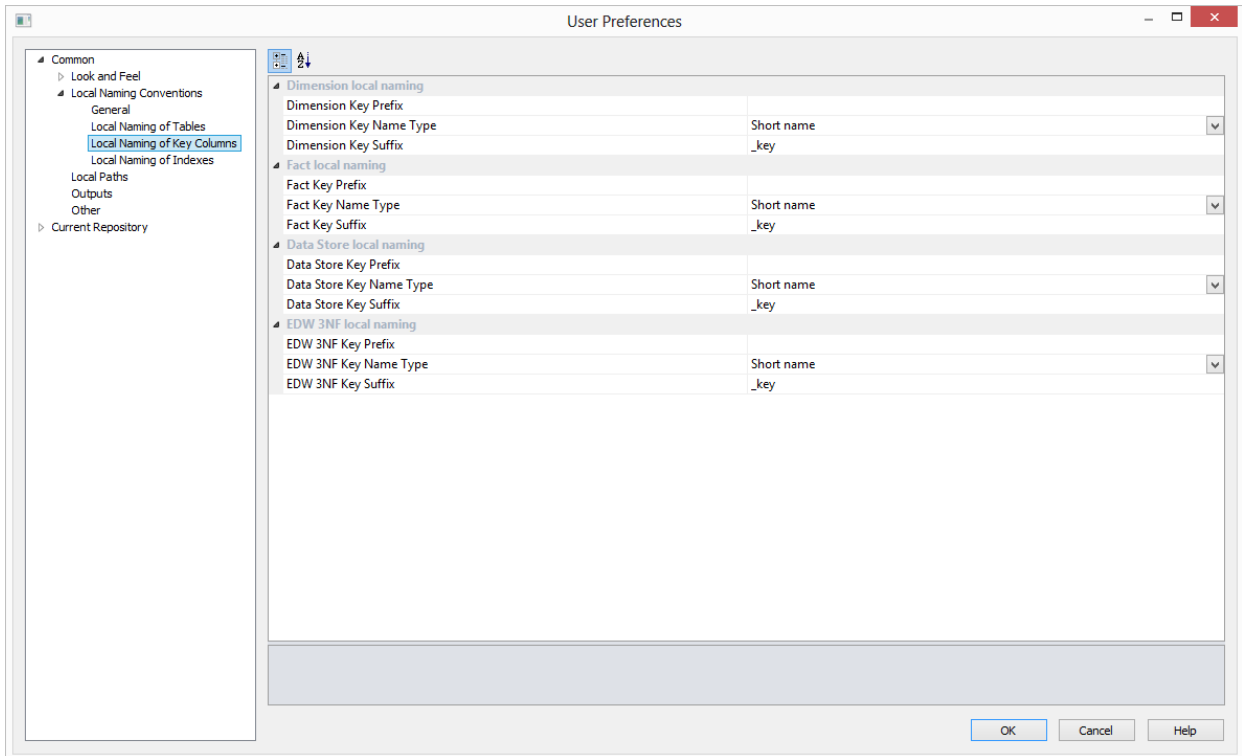
This option allows you to set the **Local Naming of Tables**.



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**.



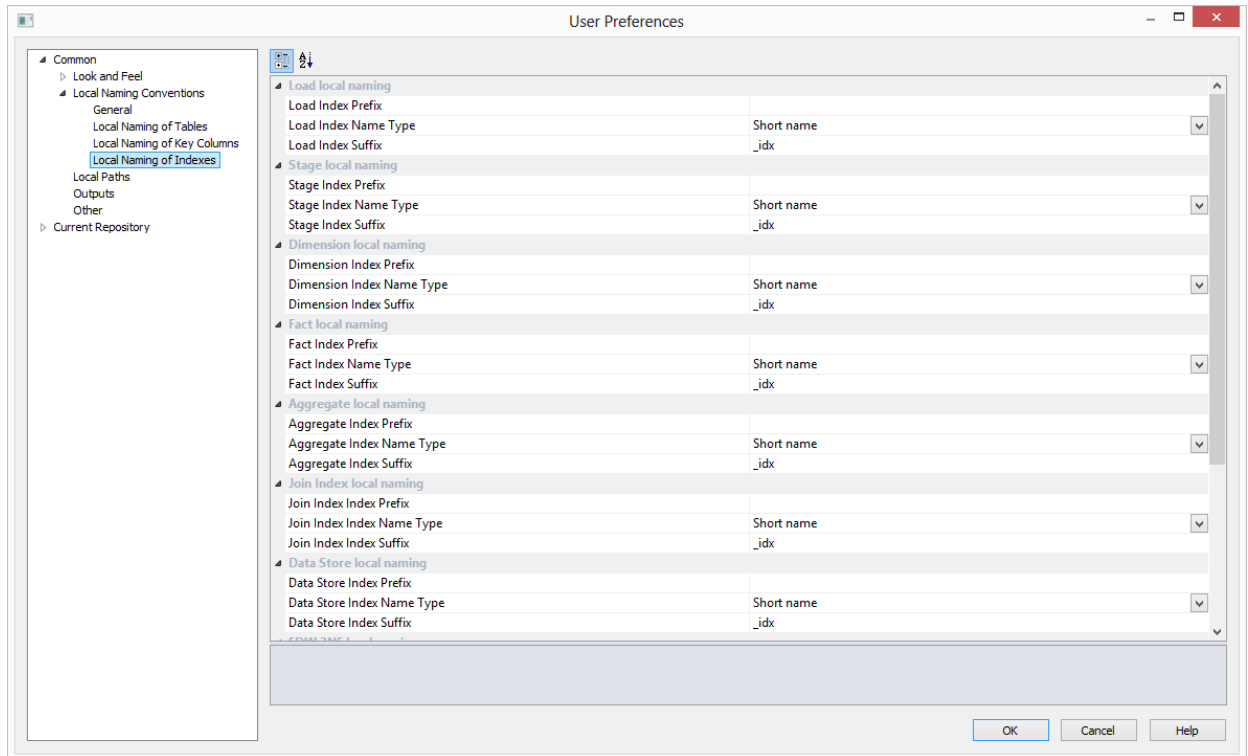
The **Key Prefix** option sets the prefix that will be used in the default key naming convention.

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**.



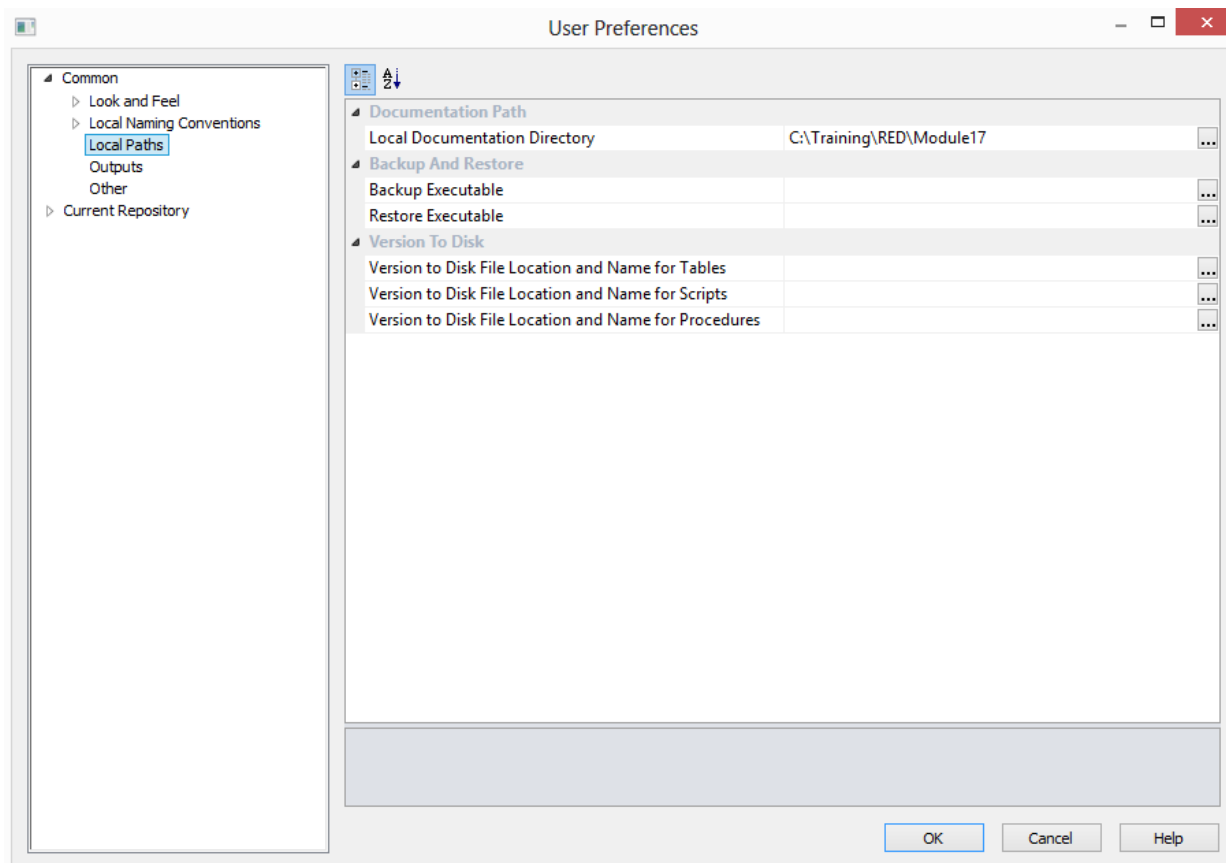
The **Index Prefix** option allows you to set the prefix that will be used in the default index naming convention.

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

Sets 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.

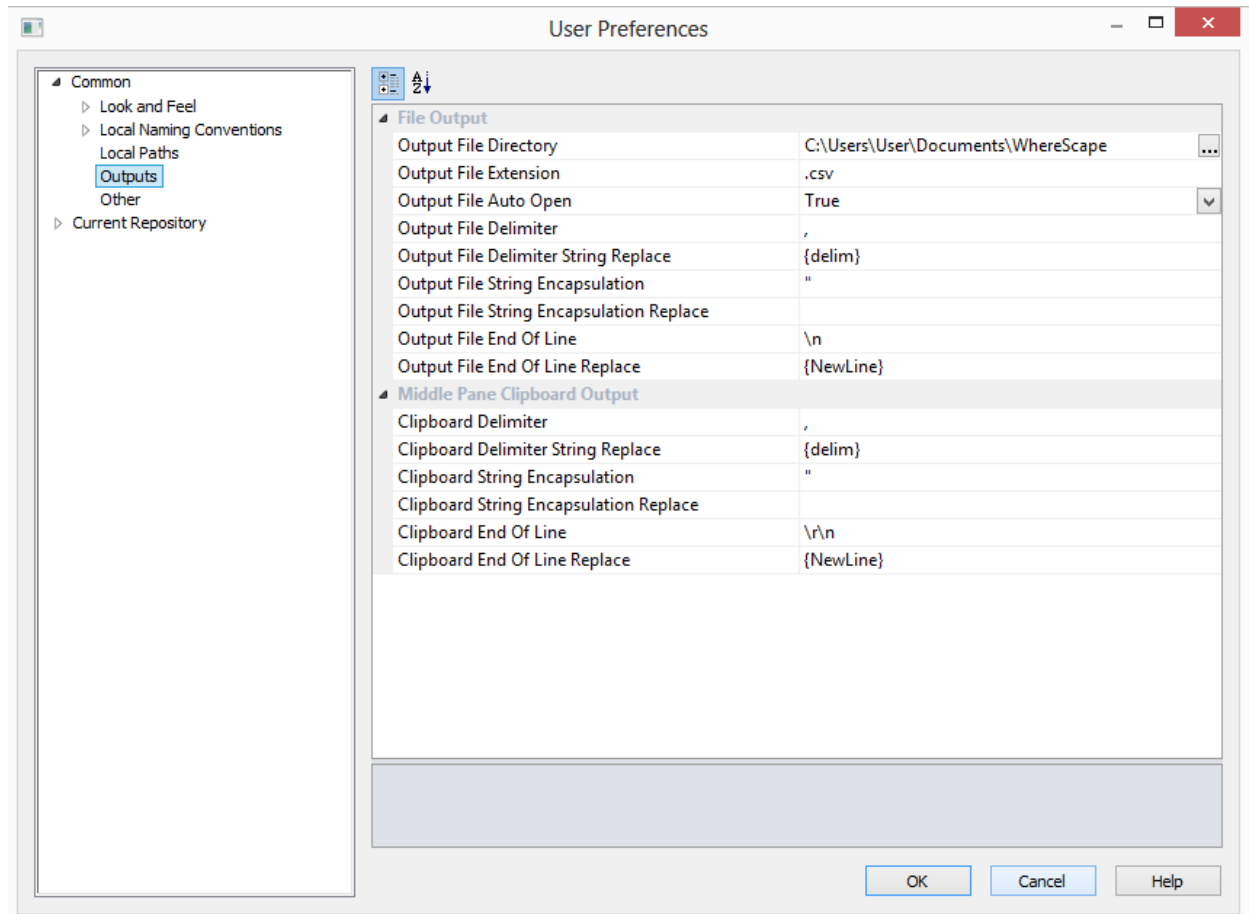
The **Restore Executable** option sets the override for restore executable. By default, WhereScape RED tries to find the path of the restore executable. This is `bcp.exe` for SQL Server, `imp.exe` for Oracle and `db2cmd.exe` for IBM DB2. This edit box provides the ability to specify the exact location 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 version 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 '.

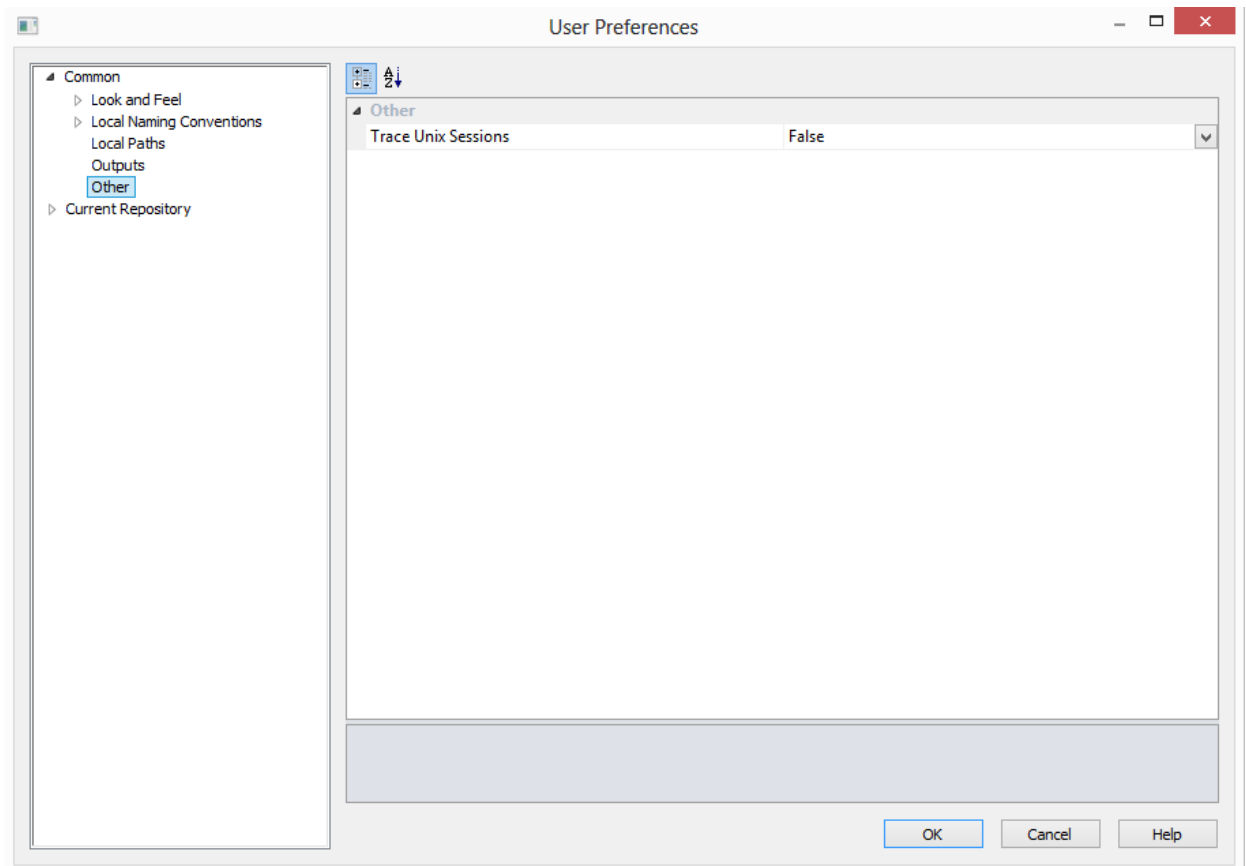
The **Clipboard String Encapsulation Replace** option allows you to set the characters that will replace the encapsulation string if it occurs inside a field.

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

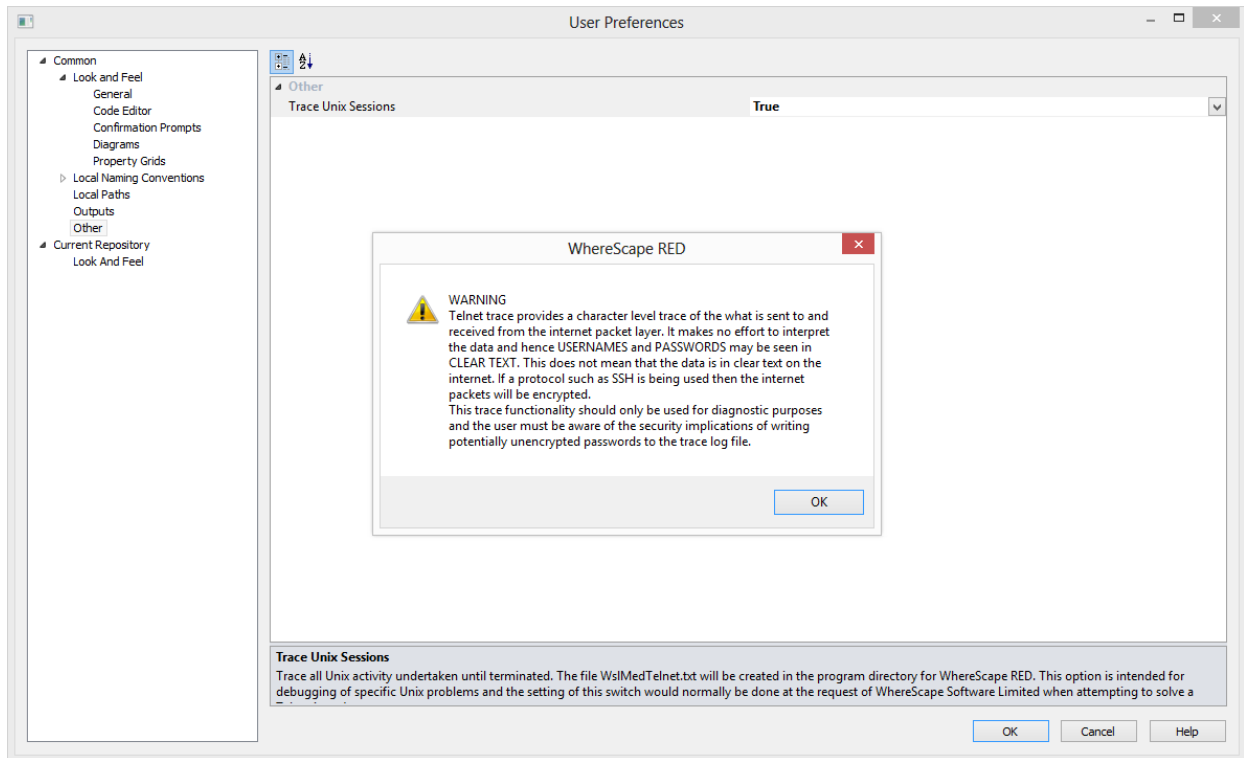
This option allows you to set the **Other** user preferences.



Other

When the **Trace Unix Sessions** option is set, this will trace all Unix activity undertaken by WhereScape RED 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).

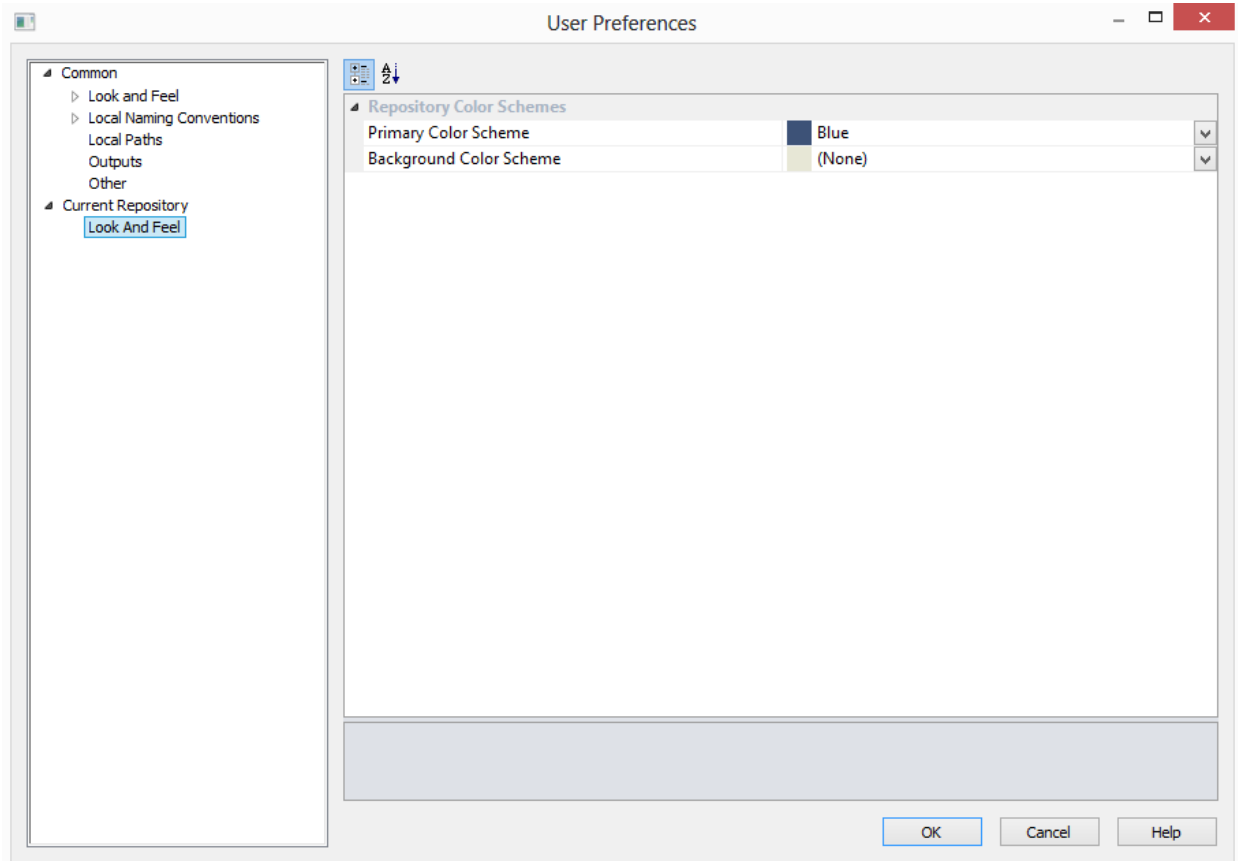
Warning: When set to true, the following warning will appear: "Telnet trace provides a character level trace of the what is sent to and received from the internet packet layer. It makes no effort to interpret the data and hence USERNAMES and PASSWORDS may be seen in CLEAR TEXT. This does not mean that 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."



SETTINGS - CURRENT REPOSITORY

LOOK AND FEEL

This allows you to set the **look and feel** for the **current repository**.

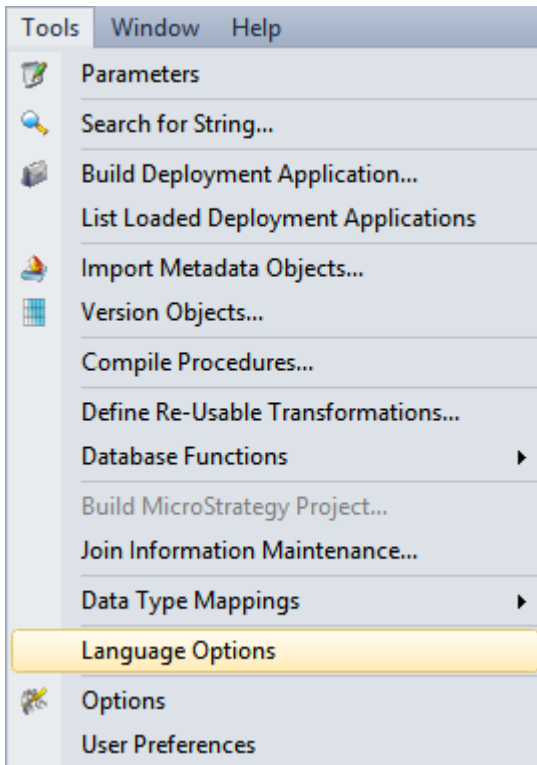


Repository Color Schemes

Set the primary and background **color schemes** for the current repository.

SETTINGS - 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.

The screenshot shows a 'Language Options' dialog box. It features a title bar with a close button (X) in the top right corner. The main content area is divided into three sections:

- Language:** A dropdown menu currently displaying 'French'.
- Language Description:** A large, empty text area with a vertical scrollbar on the right side.
- Analysis Services Language:** A dropdown menu currently displaying 'French (Belgium)'.

At the top right of the dialog, there are two buttons: 'Add Language' (highlighted in blue) and 'Delete Language'. At the bottom right, there are two buttons: 'OK' and 'Cancel'.

Language

The language Reference/ID.

Language Description

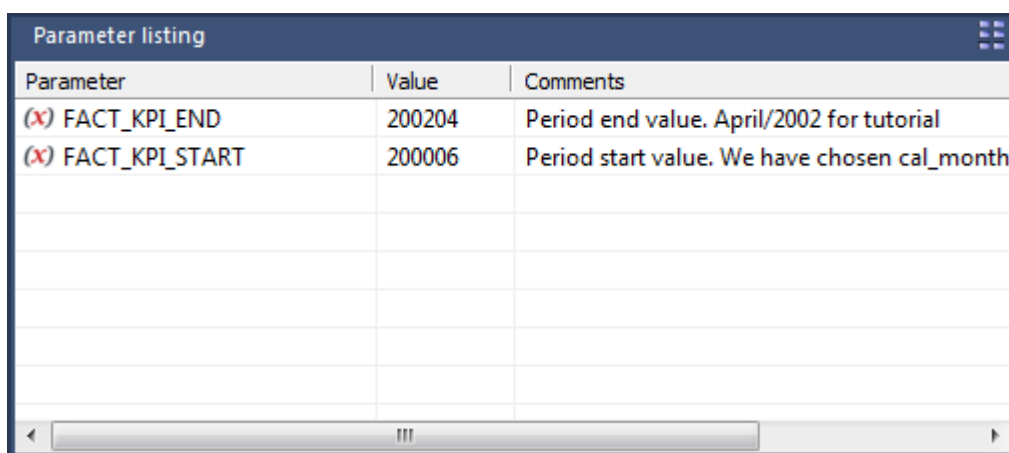
The language Description.

Analysis Services Language

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

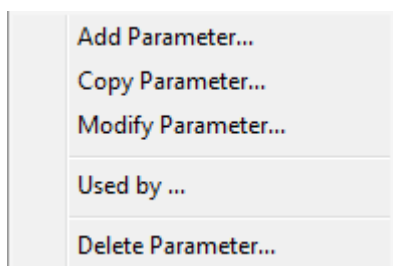
CHAPTER 6 PARAMETERS

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



| Parameter | Value | Comments |
|--------------------|--------|--|
| (X) FACT_KPI_END | 200204 | Period end value. April/2002 for tutorial |
| (X) FACT_KPI_START | 200006 | Period start value. We have chosen cal_month |
| | | |
| | | |
| | | |
| | | |

A parameter can be added, edited, copied or deleted by 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 change 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** (see "**Database Link Load - Source Screen**" on page 202) for more information.

They are also used by **stage table procedures** as variables. See **Generating the Staging Update Procedure** (on page 322) for more information.

Two procedures are provided to allow procedures to read and write parameters. These procedures are **WsParameterRead** (on page 949) and **WsParameterWrite** (on page 954). Using these procedures a procedure can load and use the contents of a parameter, modify an existing parameter, or add a new parameter.

CHAPTER 7

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 or Hadoop to analyze the source file for the new load table and 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 use 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.

NOTE: If a **Teradata TPT compliant ODBC DSN** is defined in the **Connection Properties** menu, the **TPT DSN** is used for TPT ODBC Loads.

- 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.

NOTE: 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 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..... | 134 |
| Browsing a Connection..... | 167 |
| Changing a Connection's Properties | 172 |
| Reset Meta Database Connections | 172 |
| Configuration Settings for BDA..... | 173 |

CONNECTION TYPES

Connections can be set up via the following methods:

- **Connections to the Data Warehouse/Metadata Repository** (see "**Database - Data Warehouse/Metadata Repository**" on page 135)
- **Connections to Another Database** (see "**Database**" on page 140)
- **ODBC Based Connections** (see "**ODBC**" on page 144)
- **Connections to Windows** (see "**Windows**" on page 148)
- **Connections to UNIX/Linux**
- **Connections to Hadoop**
- **Connections to Microsoft Analysis Servers** (see "**Microsoft Analysis Server 2005+**" on page 161)

DATABASE - DATA WAREHOUSE/METADATA REPOSITORY

This section describes the connection to the Data Warehouse. Tutorial 1 gives basic instructions for creating a connection.

This topic describes in greater detail the connection properties as they apply to the Data Warehouse connection.

This connection is used in the drag and drop functionality to create the stage, model and aggregate tables. It is also used to create cubes.



TIP: The Data Warehouse connection must exist if you wish to use drag and drop to create stage tables, model tables, aggregates and cubes.

Data Warehouse connection example

- A User ID and Password must be specified or
- As below a **Teradata Wallet User ID** and **TD Wallet String**.

The screenshot shows the 'Connection DataWarehouse' dialog box with the following configuration:

| Property | Value |
|--|--|
| Connection Name | DataWarehouse |
| Connection Type | Database |
| Database Type | Teradata |
| ODBC Data Source Name (DSN) | RED_REPOSITORY |
| WhereScape RED Metadata Connection Indicator | <input checked="" type="checkbox"/> |
| Source System | |
| Database ID | TD_15_00 |
| Database Link Name | |
| Provider Name | |
| Big Data Adapter Settings | |
| JDBC Connection String (JDBC URL) | jdbc:teradata://192.168.60.226/DATABASE=WSL_REPOSIT... |
| JDBC Driver Class Name | com.teradata.jdbc.TeraDriver |
| Omit Sqoop Driver Option | <input type="checkbox"/> |
| Sqoop Connection Manager Class | |
| Include Database/Schema Name in Sqoop Table Option | <input checked="" type="checkbox"/> |
| Include Sqoop Columns Option | <input checked="" type="checkbox"/> |
| Database Credentials | |
| Extract User ID | dssdemo |
| Extract User Password | ***** |
| Administrator User ID | |
| Administrator User Password | |
| Teradata Wallet User ID | |
| Teradata Wallet String | |
| JDBC User ID | dssdemo |
| JDBC Password | ***** |
| ODBC User Default | Extract User |

Buttons: OK, Cancel, Help

General

Connection Name

Name used to label the connection within WhereScape RED. Typically this is **DataWarehouse**

Connection Type

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

Distinguishes the special connection that identifies the WhereScape RED metadata repository. Set to **True**.

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.

Big Data Adapter Settings

JDBC Connection String (JDBC URL)

Connection string used by the WhereScape Big Data Adapter to access this database. This is required for Apache Sqoop loads involving this connection.

The token \$OBJECT_DATABASE\$ will be replaced by the name of the database containing the object (e.g. load table) being operated on.

Users loading into Teradata from a Hive or Hadoop connection using the Teradata connection manager for Sqoop, who need to load into more than one database, will need to add DATABASE=\$OBJECT_DATABASE\$ into their JDBC URL on the DataWarehouse connection (e.g.

jdbc:teradata://192.168.60.226/DATABASE=\$OBJECT_DATABASE\$).

BDA will replace \$OBJECT_DATABASE\$ with the database containing the load table when loading into this connection, and with the source schema defined on the load table when loading from this connection.

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 if 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 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.

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.

Teradata Wallet User ID

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

Teradata Wallet String

The Teradata Wallet String is the string replacing the username and password for your connection. Teradata TD Wallet is a Teradata product part of the TTU (Tools and Utilities). Refer to Teradata documentation if you don't have a TD Wallet created already.

ODBC User Default

Select either Extract User ID or Teradata Wallet from the drop-down menu as the default log on method.

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.

New Table Default Load Script Template

The default Template used for generating Scripts for new Load Tables created using this connection.

SSIS Connection String

Connection string to be used by Microsoft SQL Server Integration Services (SSIS) to connect to the data source or destination. The SSIS Connection String is a required field for SSIS based loads.

For more details on how to create a SSIS Connection String and load data via an Integration Services Load package, see section **SSIS Loader** or **Loading Data from Flat Files using SSIS**.

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

SSIS Use Column Names

Determines whether to use column names or column titles for SSIS loads. Terada OLE DB driver by default returns titles. Leave this option disabled.

Data Type Mapping Set

Mapping Set to use when converting from a source database data type to a destination database data type.

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 if 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.

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 **TDOLEDB**.

Data Warehouse Server

Data Warehouse Server Name, which is used when the data warehouse is used as the data source for OLAP cubes. Set this to the Teradata TDPID.

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 target database locations for objects in this connection. For this option to be enabled, the **Enable Targets for setting object location field** needs to be enabled in **Settings - Repository identification** (on page 74).

Click the **Add** button to add the required target location for this connection.

- 6 Give the new target location a name and then enter the **Target Database** and/or **Temp Database**.
- 7 Default target database location(s) for **New Tables** can also be set from the **Tools/Options** menu – See *Settings - Storage - Target Location*.
- 8 For more details on setting specific target database locations on a table by table basis see *Storage* (on page 182).

Temp Databases specify the location for temporarily created tables used in Load and Export processes.



WhereScape RED TIP:

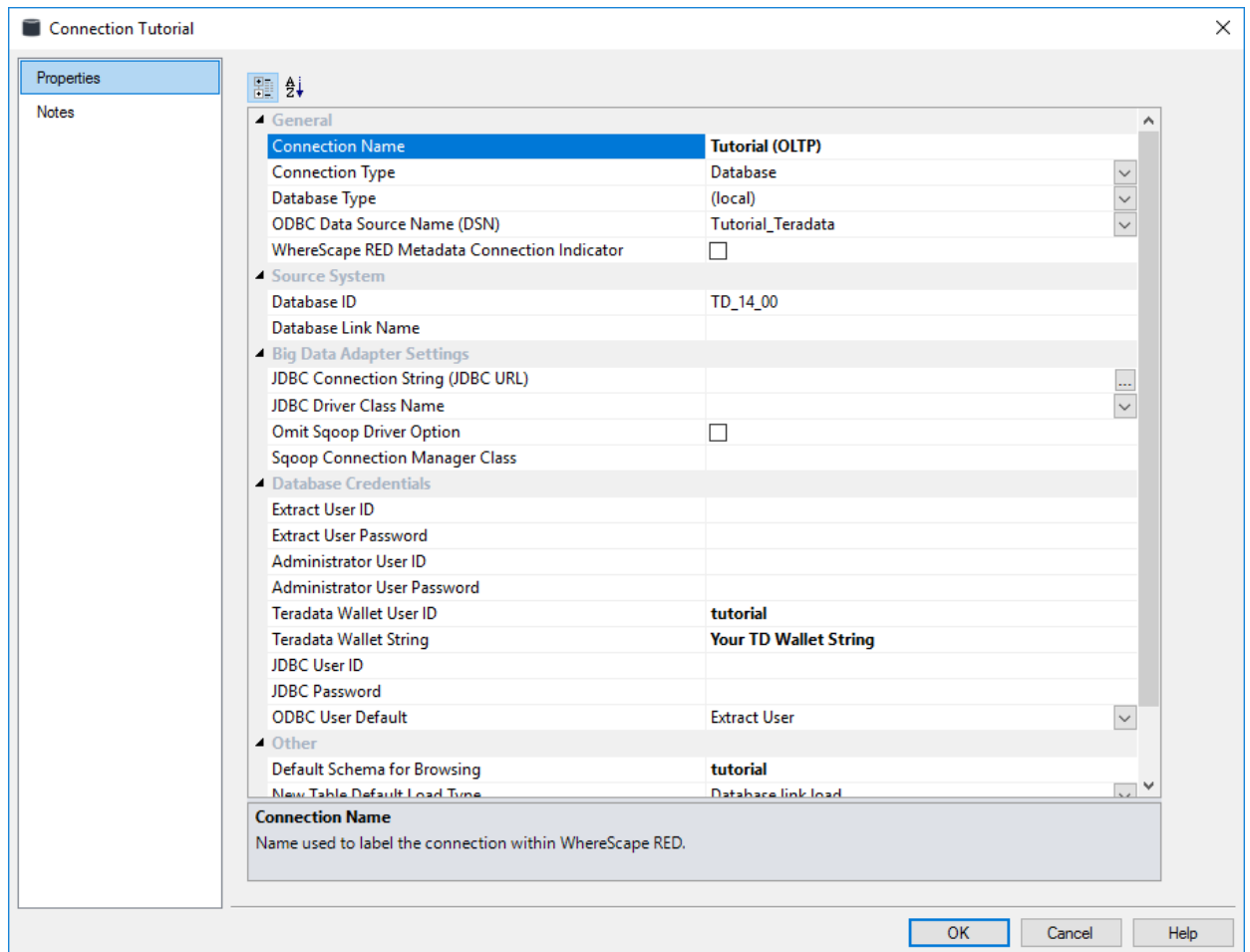
Once the connection has been set up, you can right-click on the connection in the middle pane or double click on the connection name from the left pane to view or edit the connection's Properties.

DATABASE

This section describes connections to another database source inside the same Teradata server, but not in the WhereScape RED meta repository.

Example

Sample database connection object properties screen:



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

Set to **False**.

Source System

Database ID

Database Identifier (Teradata TDPID).

Database Link Name

This field is always blank for Teradata.

Database Credentials

Extract User ID

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

Extract User Password

The password of the data warehouse user.

Administrator User ID

Leave blank.

Administrator User Password

Leave blank.

Teradata Wallet User ID and Teradata Wallet String

Enter the relevant credentials when using the Teradata Wallet log on method instead of Extract User ID and Password for connecting to another database.

ODBC User Default

Select either Extract User ID or Teradata Wallet from the drop-down menu as the default log on method.

Other

Default Schema for Browsing

Optional comma-delimited list of schema(s) 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.

New Table Default Load Script Template

The default Template used for generating Scripts for new Load Tables created using this connection.

SSIS Connection String

Connection string to be used by Microsoft SQL Server Integration Services (SSIS) to connect to the data source or destination. The SSIS Connection String is a required field for SSIS based loads.

For more details on how to create a SSIS Connection String and load data via an Integration Services Load package, see section **SSIS Loader**.

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)** will cause RED to automatically select the relevant mapping set.

Target Table Location [For target enabled licenses]

Add new Target Location

This option allows adding new target database locations for objects in this connection. For this option to be enabled, the **Enable Targets for setting object location field** needs to be enabled in **Settings - Repository identification** (on page 74).

- 1 Click the **Add** button to add the required target location for this connection.
- 2 Give the new target location a name and then enter the **Target Database** and/or **Temp Database**.
- 3 Default target database 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 target database locations on a table by table basis see **Storage** (on page 182).

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.

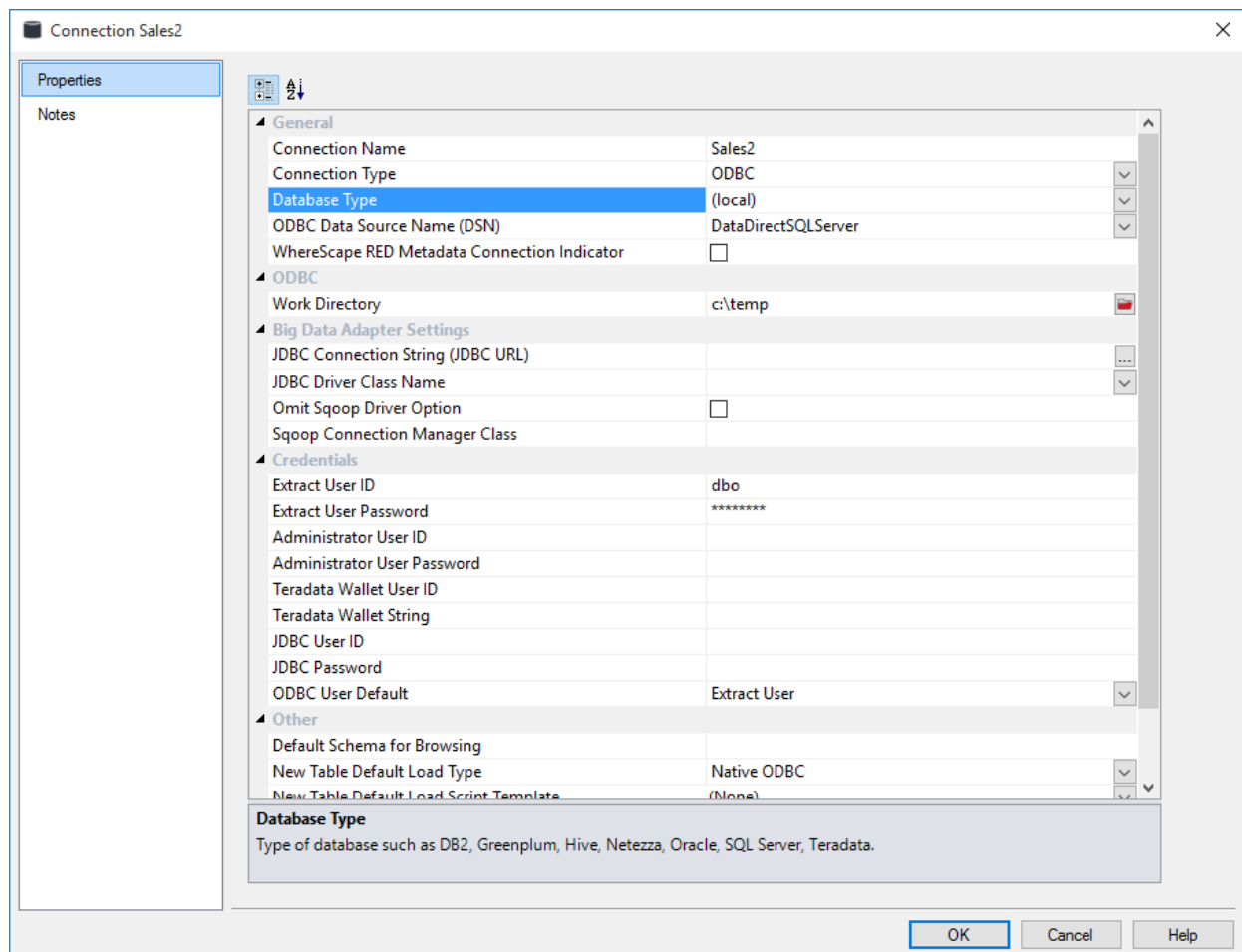
Once the connection has been set up, you can right-click on the connection in the middle pane or double click on the connection name from the left pane to view or edit the connection's Properties.

ODBC

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

Native ODBC Load Example

- A User ID and Password must be specified or
- As below a **Teradata Wallet User ID** and **TD Wallet String**.



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, etc.

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

Set to **False**.

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. See *Native ODBC Based Load* (on page 205).

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 if this is not required.

Credentials

Extract User ID

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

Extract 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 Password

Leave blank.

Teradata Wallet User ID and Teradata Wallet String

Enter the relevant Teradata Wallet credentials instead of the Extract user and password for the ODBC connection.

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 WhereScape Big Data Adapter (optional).

ODBC User Default

Select either Extract User ID or Teradata Wallet from the drop-down menu as the default log on method.

Other

Default Schema for Browsing

Optional comma-delimited list of schema(s) 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. Set to the desired type of **ODBC load** – Native, ODBC, TPT, TPT Script load, Integration Services load or Externally Loaded.

New Table Default Load Script Template

The default Template used for generating Scripts for new Load Tables created using this connection.

SSIS Connection String

Connection string to be used by Microsoft SQL Server Integration Services (SSIS) to connect to the data source or destination. The SSIS Connection String is a required field for SSIS based loads.

For more details on how to create a SSIS Connection String and load data via an Integration Services Load package, see section *SSIS Loader*.

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

TPT ODBC Data Source Name (DSN)

The Teradata TPT compliant ODBC Data Source Name (DSN) that is to be used in the TPT Read Operator for TPT ODBC Loads. If not populated, this defaults to the DSN defined above for this Connection.

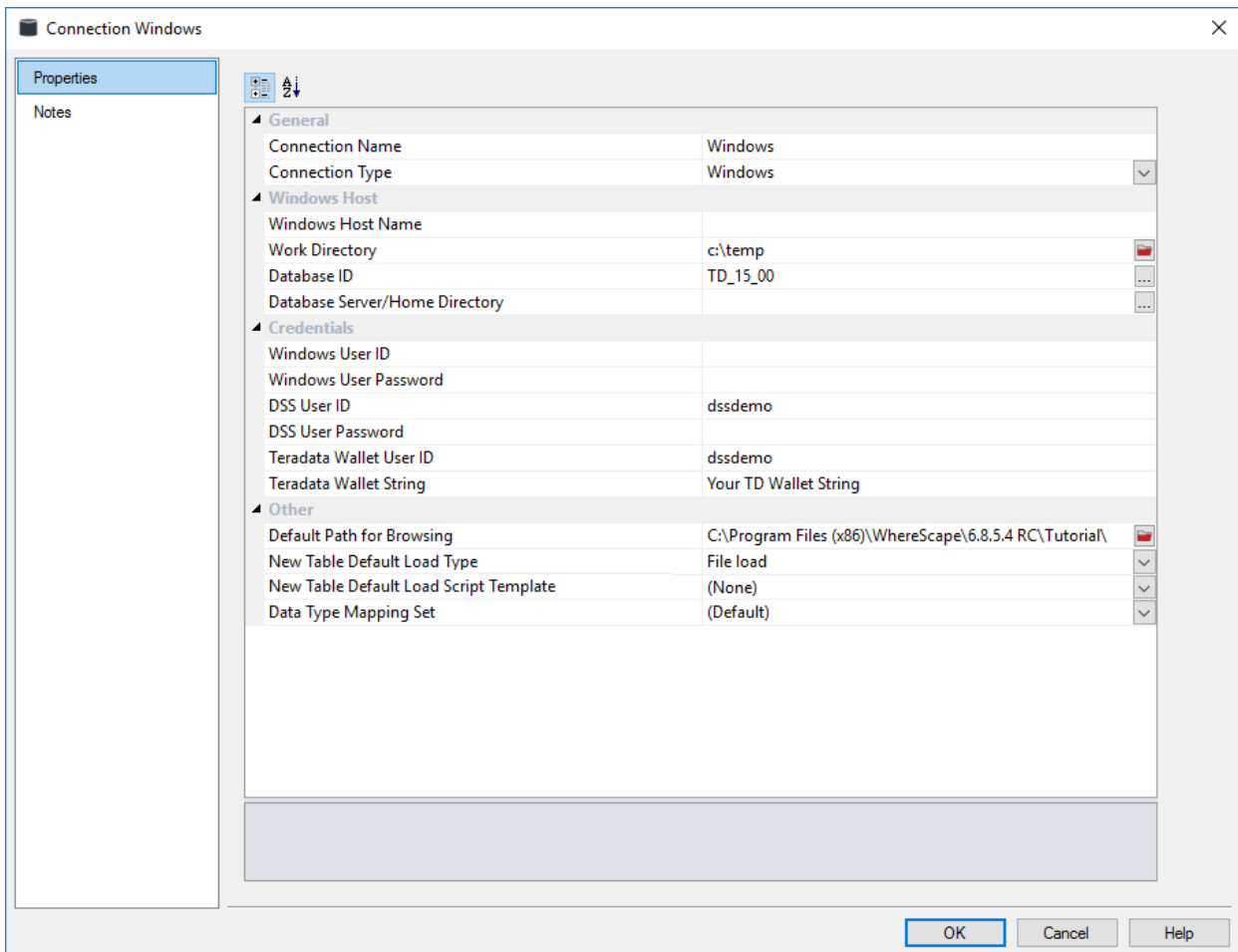
Data Type Mapping Set

Mapping Set to use when converting from a source database data type to a destination database data type. Setting this field to **(Default)** will cause RED to automatically select the relevant mapping 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.

WINDOWS

This connection is back to the PC that you are working on, or to a host Windows PC.



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

Database Identifier (Teradata TDPID).

Database Server/Home Directory

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

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/ Teradata Wallet User ID and Teradata Wallet String

Enter the relevant details for connecting to the **Data Warehouse**. Enter either DSS User ID and Password or Teradata Wallet credentials depending on the log on method selected.

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.

New Table Default Load Script Template

The default Template used for generating Scripts for new Load Tables created using this connection.

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)** will cause RED to automatically select the relevant mapping set.

Once the connection has been set up, you can right-click on the connection in the middle pane and view the Properties for that connection.


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.

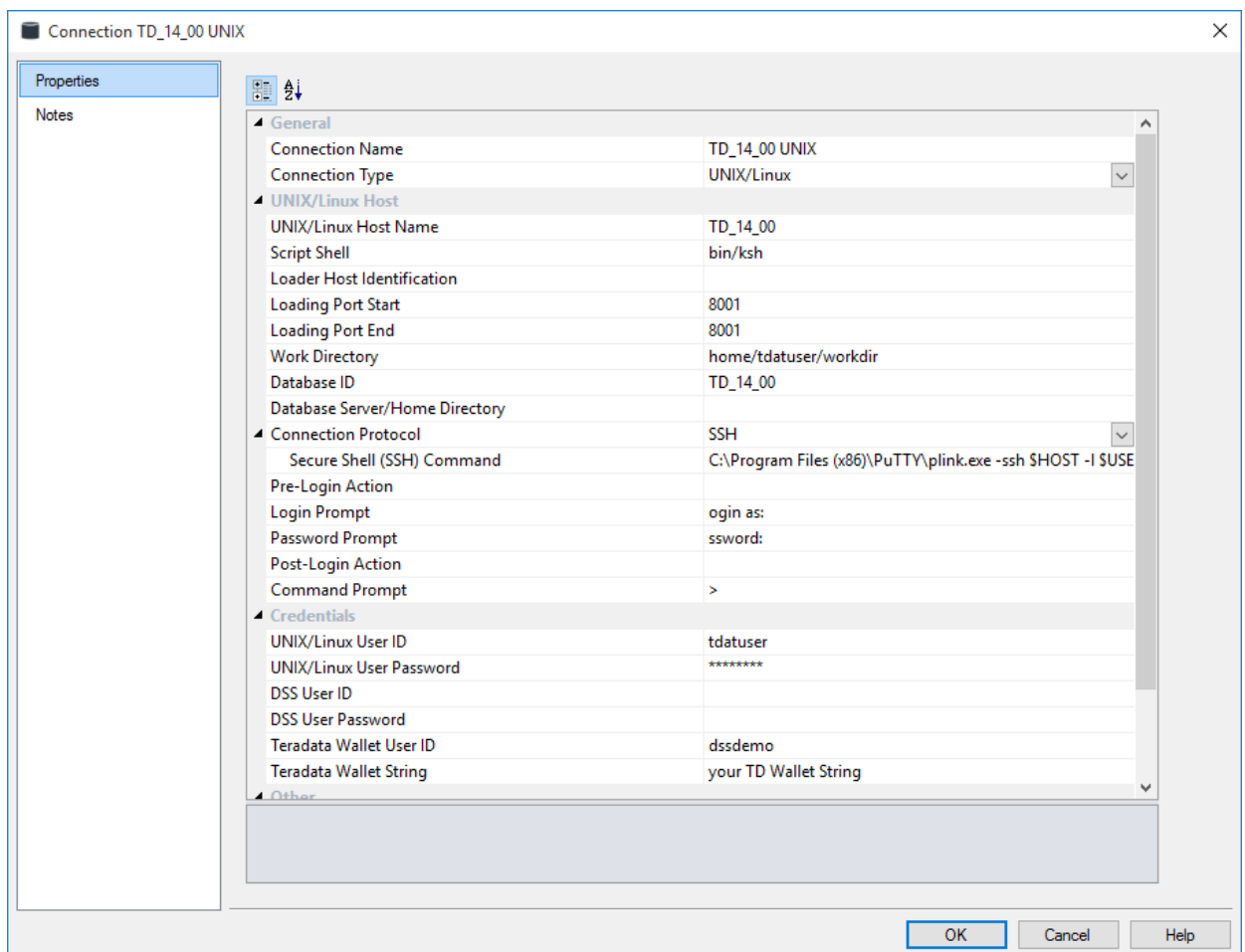
UNIX

This section describes the connection properties as they apply to UNIX connections. From a UNIX connection you can only process flat file loads.

Note: 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 UNIX connection screen:



The screenshot shows a dialog box titled "Connection TD_14_00 UNIX" with a "Properties" tab selected. The "Notes" section is empty. The main area displays a list of connection properties:

| Property | Value |
|--------------------------------|--|
| Connection Name | TD_14_00 UNIX |
| Connection Type | UNIX/Linux |
| UNIX/Linux Host | |
| UNIX/Linux Host Name | TD_14_00 |
| Script Shell | bin/ksh |
| Loader Host Identification | |
| Loading Port Start | 8001 |
| Loading Port End | 8001 |
| Work Directory | home/tdatuser/workdir |
| Database ID | TD_14_00 |
| Database Server/Home Directory | |
| Connection Protocol | |
| Secure Shell (SSH) Command | C:\Program Files (x86)\PuTTY\plink.exe -ssh \$HOST -l \$USER |
| Pre-Login Action | |
| Login Prompt | ogin as: |
| Password Prompt | ssword: |
| Post-Login Action | |
| Command Prompt | > |
| Credentials | |
| UNIX/Linux User ID | tdatuser |
| UNIX/Linux User Password | ***** |
| DSS User ID | |
| DSS User Password | |
| Teradata Wallet User ID | dssdemo |
| Teradata Wallet String | your TD Wallet String |
| Other | |

Buttons at the bottom: OK, Cancel, Help.

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 **UNIX**.

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 (Teradata TDPID).

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: In order to ascertain some of the above fields you will have to 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.

Teradata Wallet User ID and Teradata Wallet String

Enter the relevant Teradata Wallet credentials instead of the DSS User and Password for the Unix connection if using the Teradata log on method.

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.

New Table Default Load Script Template

The default Template used for generating Scripts for new Load Tables created using this connection.

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**

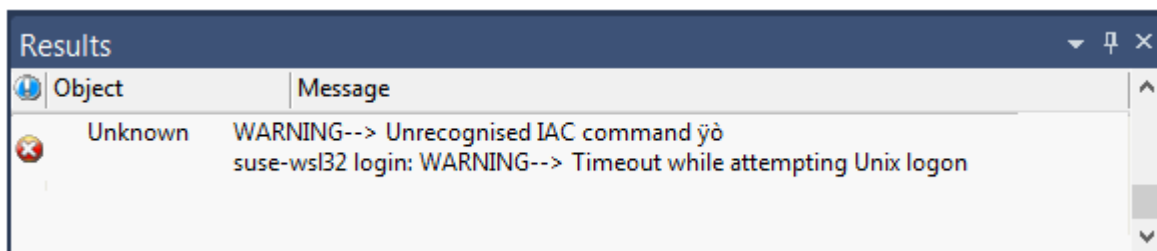
This will provide a telnet window that can be used to log on to the UNIX server.

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

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

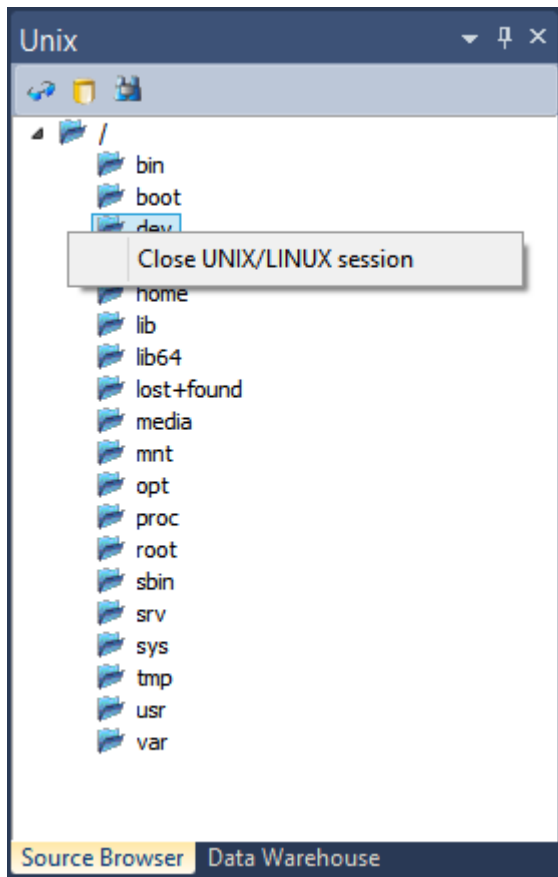


Attempt the connection again, and using the **Window** menu option select the **Telnet** window. This will display the login session, and should provide an insight as to why the connection is not being completed.

If the situation cannot be resolved a telnet trace can be acquired. Select the menu option **Tools/Options** and click on the checkbox **Trace all Unix sessions**. 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 send to <SUPPORT>.

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 enables RED users to connect to the Hadoop System and process two load types from a Hadoop source into a Teradata repository. These connections must always be set via a **Secure Shell (SSH)** protocol.

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

The two load types that can be processed from RED are:

- **Native SSH Load** - connections to Hadoop on UNIX/Linux from which users do flat file loads. To process a Native SSH load, select **UNIX/Linux** as the connection type set the remaining connection properties.
- **TPT Load** - connections to Hadoop from which users can do TPT script-based loads. To process Hadoop TPT loads, select **Hadoop** as the connection type and set the remaining connection properties.

For **Hadoop TPT Loads**, users will need to have the following system prerequisites before setting up a connection within RED:

- Install **Hadoop**
- Include Hadoop Client Jar files in Classpath

Example for an Apache Hadoop environment setup

```
#Hadoop
```

```
export PATH=$PATH:/opt/hadoop-2.4.1/bin:/opt/hadoop-2.4.1/sbin
```

```
#For Teradata TPT Load (Hadoop)
```

```
export CLASSPATH=$(find /opt/hadoop-2.4.1/share/hadoop/hdfs -name *.jar -printf '%p:' | sed 's:$/:')
```

```
export CLASSPATH=$CLASSPATH:$(find /opt/hadoop-2.4.1/share/hadoop/common -name *.jar -printf '%p:' | sed 's:$/:')
```



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).

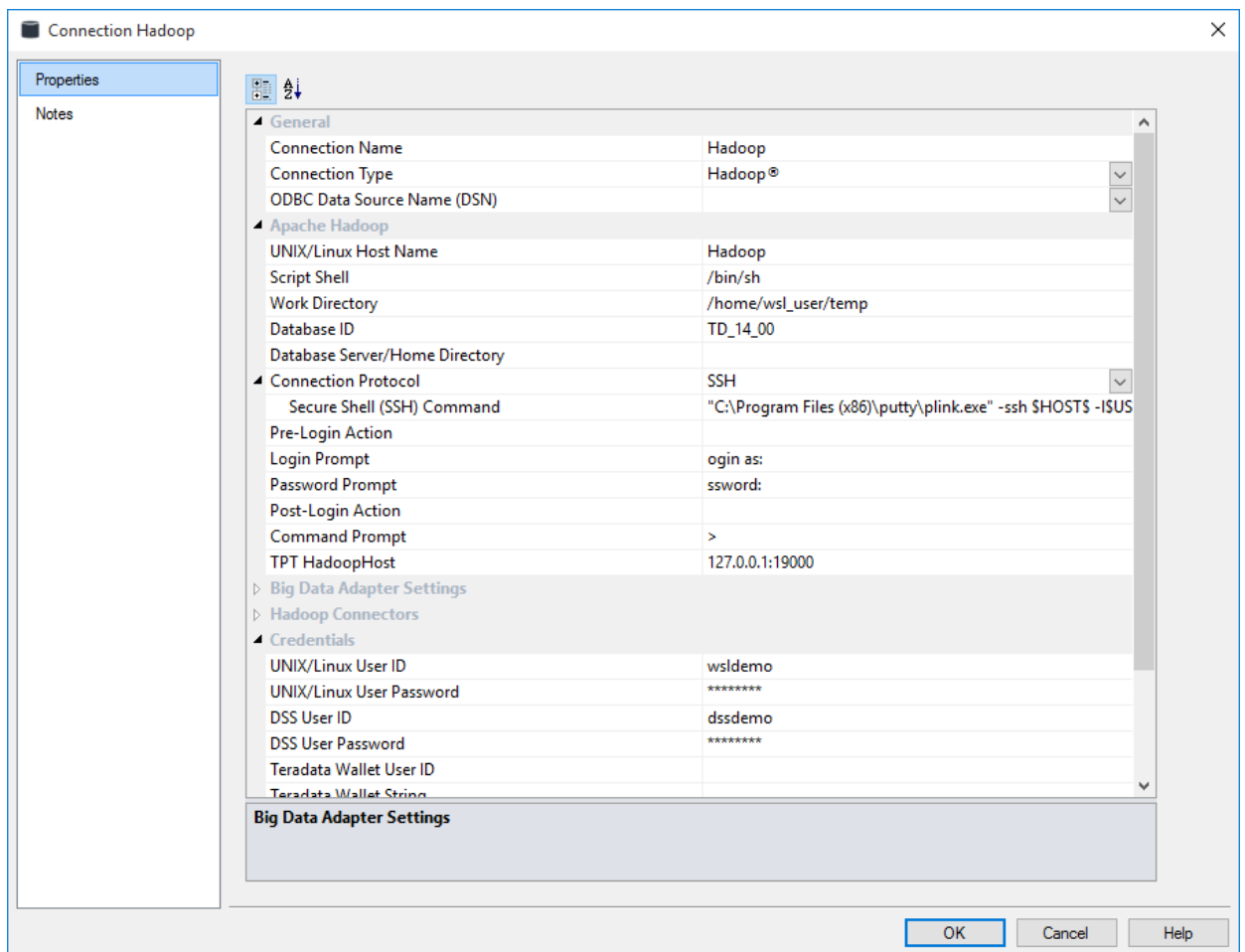
For more information about these settings, see the **Big Data Adapter Settings** fields description below and see also *Connection to the Data Warehouse* (see "*Database - Data Warehouse/Metadata Repository*" on page 135), *Configuring your database for use by BDA* and *Apache Sqoop Load*.

Note: If the **Hadoop** 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 TPT 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 **UNIX**.

Apache Hadoop

UNIX/Linux Host Name

Enter the 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 **C:\Temp** or a sub-directory of C:\Temp is used.

Database ID

Enter the Source Database Identifier (Teradata TDPID).

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\PuTTY\plink.exe -ssh \$HOST\$ -l \$USER\$ -pw \$PASSWORD\$**.

In-built variables **\$HOST\$**, **\$USER\$** and **\$PASSWORD\$** can be used here for the required host, user and password fields.

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: In order to ascertain some of the above fields you will have to log in to the UNIX/Linux system.

TPT HadoopHost

IP address or host name (and optional port number) that identifies the Hadoop Host to a TPT load routine, in order to connect to the Hadoop file system from the machine you run TPT. e.g,

HadoopHost:9000 or **127.0.0.1:9000**.

If this is not specified, then the UNIX/Linux Host Name will be used as the Hadoop Host to the TPT load routine.

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 *Connection to the Data Warehouse* (see "*Database - Data Warehouse/Metadata Repository*" on page 135) 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.

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.

Teradata Wallet User ID and Teradata Wallet String

Enter the relevant Teradata Wallet credentials instead of the DSS User and Password for the Unix connection if using the Teradata log on method.

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**, **Native SSH** or **Externally loaded** options.

New Table Default Load Script Template

The default Template used for generating Scripts for new Load Tables created using this connection.

Data Type Mapping Set

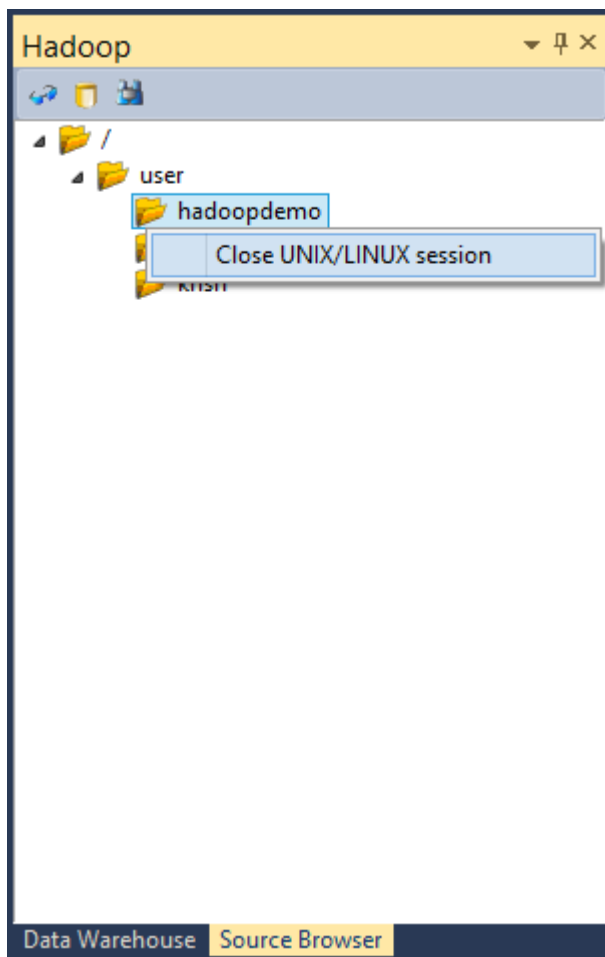
XML files have been created to store mappings from one set of data types to another. Setting this field to **(Default)** will cause RED to automatically select the relevant mapping set; otherwise you can choose one of the standard mapping sets from the drop-down list or create a new one.

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

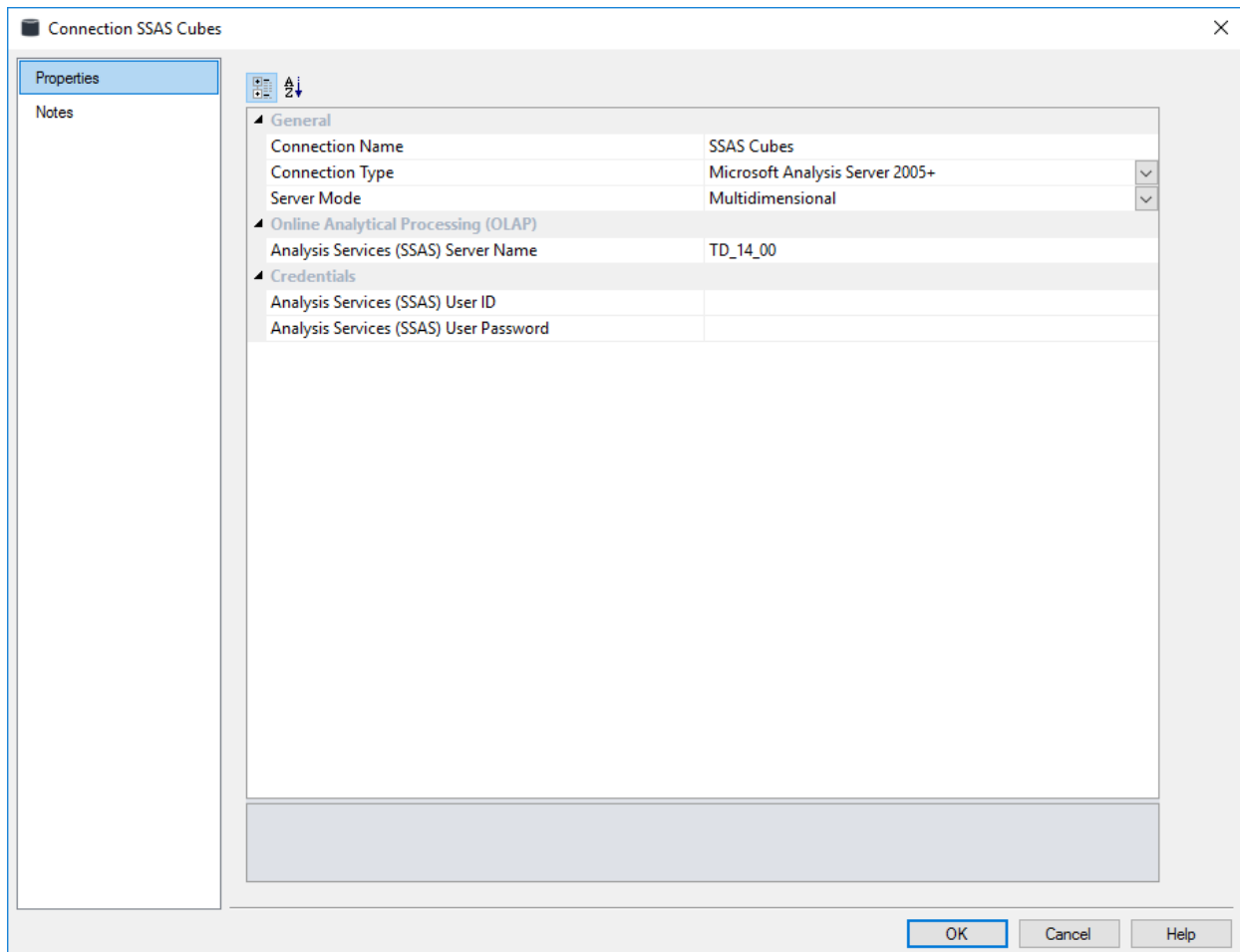
To close the collection, right-click in the browser pane and select **Close UNIX/LINUX session:**



MICROSOFT ANALYSIS SERVER 2005+

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. An example screen shot follows:



Connection Name

Enter a name to identify the connection to the Analysis Services server.

Connection Type

The connection type is chosen from the drop-down list. Select "Microsoft Analysis Server 2005".

Analysis Server

Enter the name of the Analysis Services server you wish to connect to. This must be a valid server name. Contact your system administrator if you do not have a valid server name.

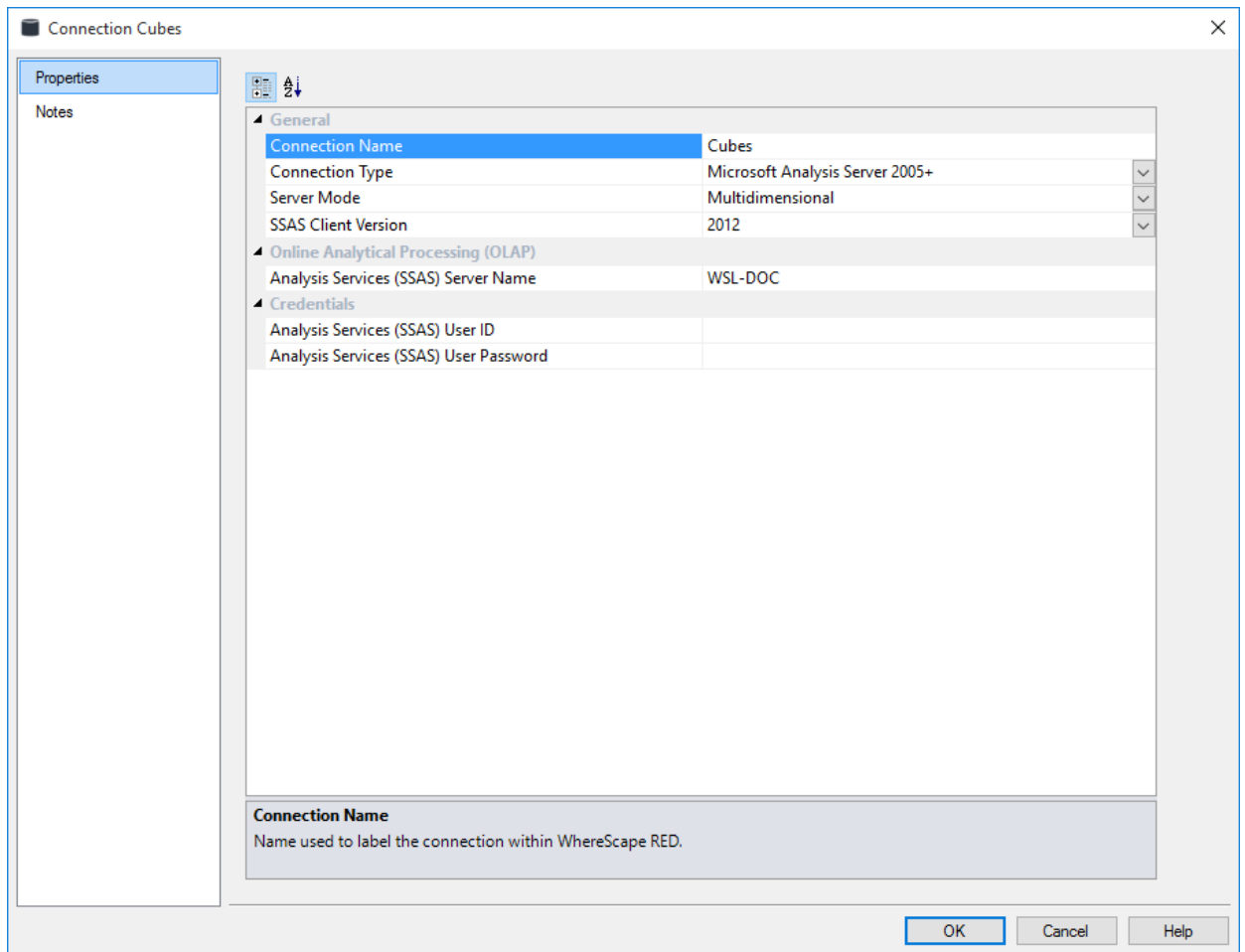
User ID and Password

Not used, leave blank.

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 used to connect to Analysis Services when using SQL Server Authentication. Can be left blank for a trusted connection using Windows Authentication.

Analysis Services (SSAS) User ID and Password

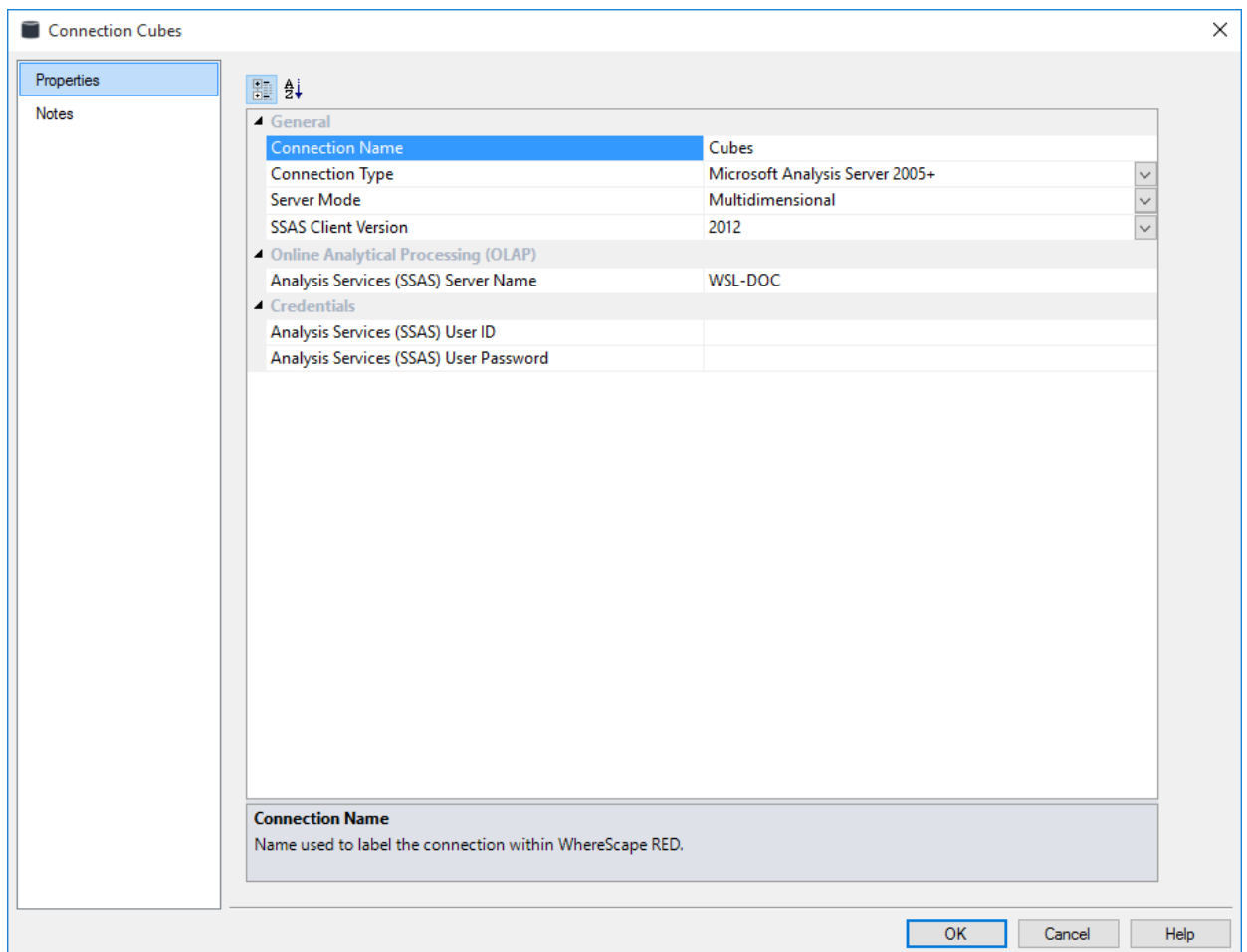
Password used to connect to Analysis Services when using SQL Server Authentication. Can be left blank for a trusted connection using Windows Authentication.

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 1012).

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

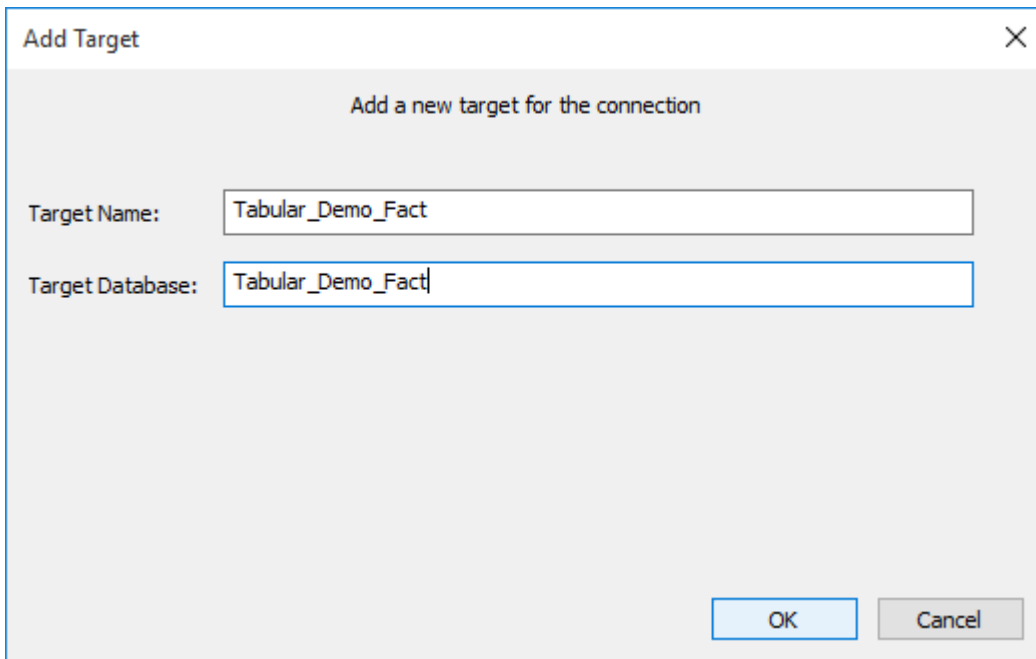
Password to connect to Analysis Services with when using SQL Server Authentication. Can be left blank for a trusted connection using Windows Authentication.

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:

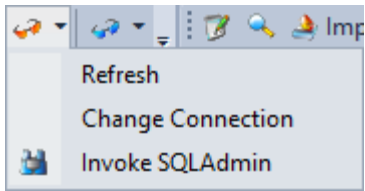


The orange button is used to browse the data warehouse connection and the blue button is used to browse a source system connection.



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.

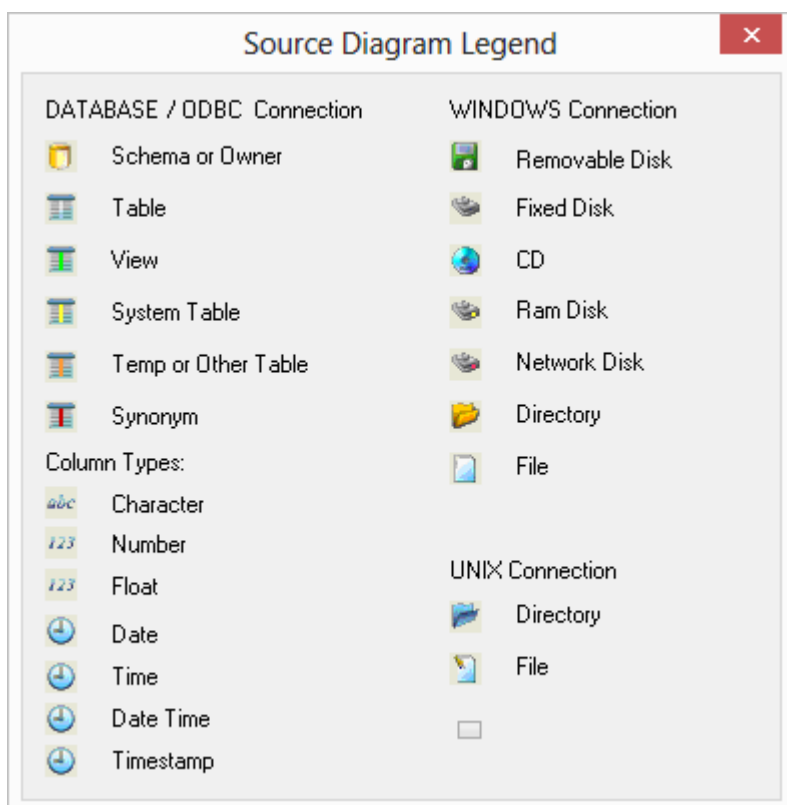


The current connection being browsed is shown in the status bar at the bottom right of the screen.

Browser Icons

When browsing a connection the following legend applies for the source tables and objects.

This legend is displayed via the **Help/Source Legend** menu.



CONNECTION BROWSE PROPERTIES

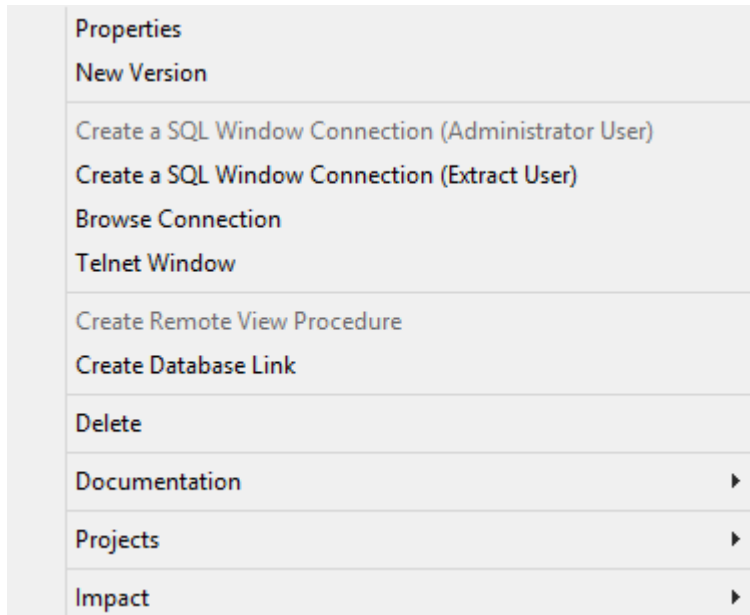


TIP: When browsing to a connection leave the schema field blank in order to see all schemas.

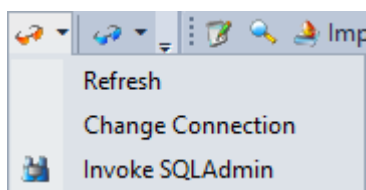
To have RED browsing a specific schema or schemas by default, go to a Connection's Properties screen and enter the schema(s) to browse on the **Default Schema for Browsing** field.

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** is displayed:

The dialog box titled "List Source Tables Connection" contains the following fields and options:

- Connection: DataWarehouse
- User ID: WslWarehouse
- Password: (empty)
- Include Rowcount
- Password is TDWallet string.
- Filter section:
 - Schema: WslWarehouse
 - Name: (None)
 - Object Types:
 - Tables
 - Views
 - System Objects
 - Group: (All)
 - Project: (All)
- Data Type Mapping Set: (Default)
- Buttons: Refresh Current, OK, Cancel

The dialog allows you to change the properties of the connection you are browsing.

The **User ID** and **Password** fields can be changed in order to browse the connection as a different user. A TDWallet string can be supplied as the password by using the **Password is TDWallet string** check-box.

Selecting 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.



Note: Whenever a load connections properties are changed. All load tables that use the connection must be changed. See above. You will be asked if you wish to perform this action when changing the connection.

RESET META DATABASE CONNECTIONS

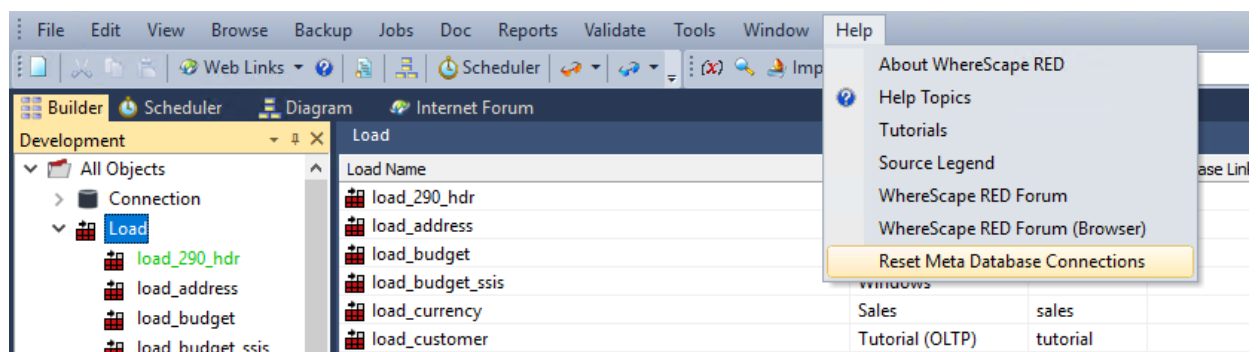
From the **Help** menu, on RED's main top bar, users can select the the **Reset Meta Database Connections** option.

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.

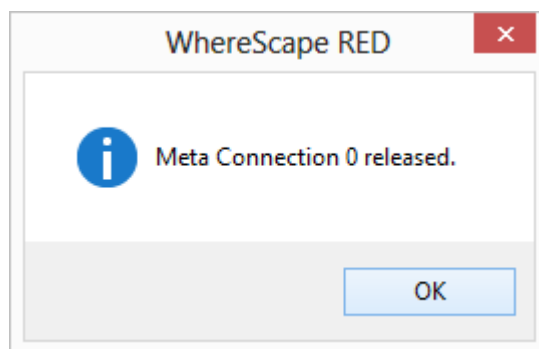
When this option is used, RED attempts to release most existing ODBC connections so the next time an ODBC Connection is used, RED will attempt to reestablish a link.

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.



CONFIGURATION SETTINGS FOR BDA

This topic describes the required settings for connections using the WhereScape Big Data Adapter (BDA).

The WhereScape Big Data Adapter (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.

Important: A Hive connection must exist before BDA settings (**Connection>Properties>Big Data Adapter Settings**) appear in other database connections.

BDA enables RED to use Sqoop as a 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.

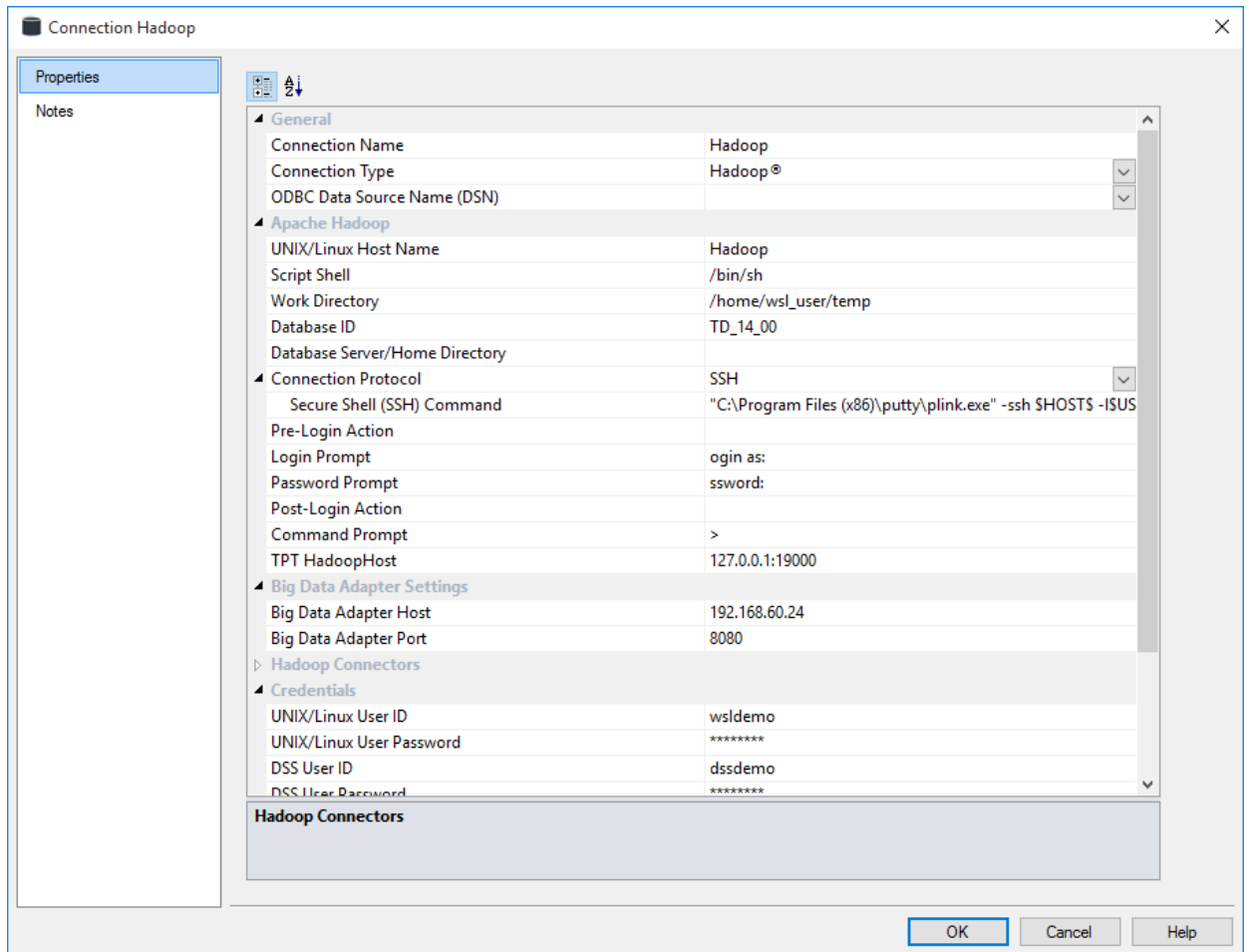
CONFIGURING THE BDA SERVER

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 **Connection to the Data Warehouse** (see "**Database - Data Warehouse/Metadata Repository**" on page 135), **Connections to 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 *Connection to the Data Warehouse* (see "*Database - Data Warehouse/Metadata Repository*" on page 135).

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.

Note: When creating objects in a Hive target database from a Teradata repository, artificial keys must be of type 'int', not 'integer'. To correct this, go to **Tools -> Options -> Global Naming Conventions -> Global Naming of Key Columns** and change the relevant Data Types to 'int'.

Example of a Hive ODBC connection

The screenshot shows the 'Connection Hive' dialog box with the following settings:

| Category | Property | Value |
|---------------------------|--|-----------------------------------|
| General | Connection Name | Hive |
| | Connection Type | ODBC |
| | Database Type | Hive |
| | ODBC Data Source Name (DSN) | hive_odbc |
| | WhereScape RED Metadata Connection Indicator | <input type="checkbox"/> |
| ODBC | Work Directory | c:\temp |
| Big Data Adapter Settings | Big Data Adapter Host | 192.168.60.100 |
| | Big Data Adapter Port | 8080 |
| | Base Target Directory for Sqoop Loads | /tmp |
| | JDBC Connection String (JDBC URL) | jdbc:hive2://192.168.60.100:10000 |
| | JDBC Driver Class Name | org.apache.hive.jdbc.HiveDriver |
| | Omit Sqoop Driver Option | <input type="checkbox"/> |
| | Sqoop Connection Manager Class | |
| Credentials | Extract User ID | wsldemo |
| | Extract User Password | ***** |
| | Administrator User ID | |
| | Administrator User Password | |
| | Teradata Wallet User ID | |
| | Teradata Wallet String | |
| | JDBC User ID | wsldemo |
| JDBC Password | ***** | |
| Other | Default Schema for Browsing | |

Buttons: OK, Cancel, Help

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.

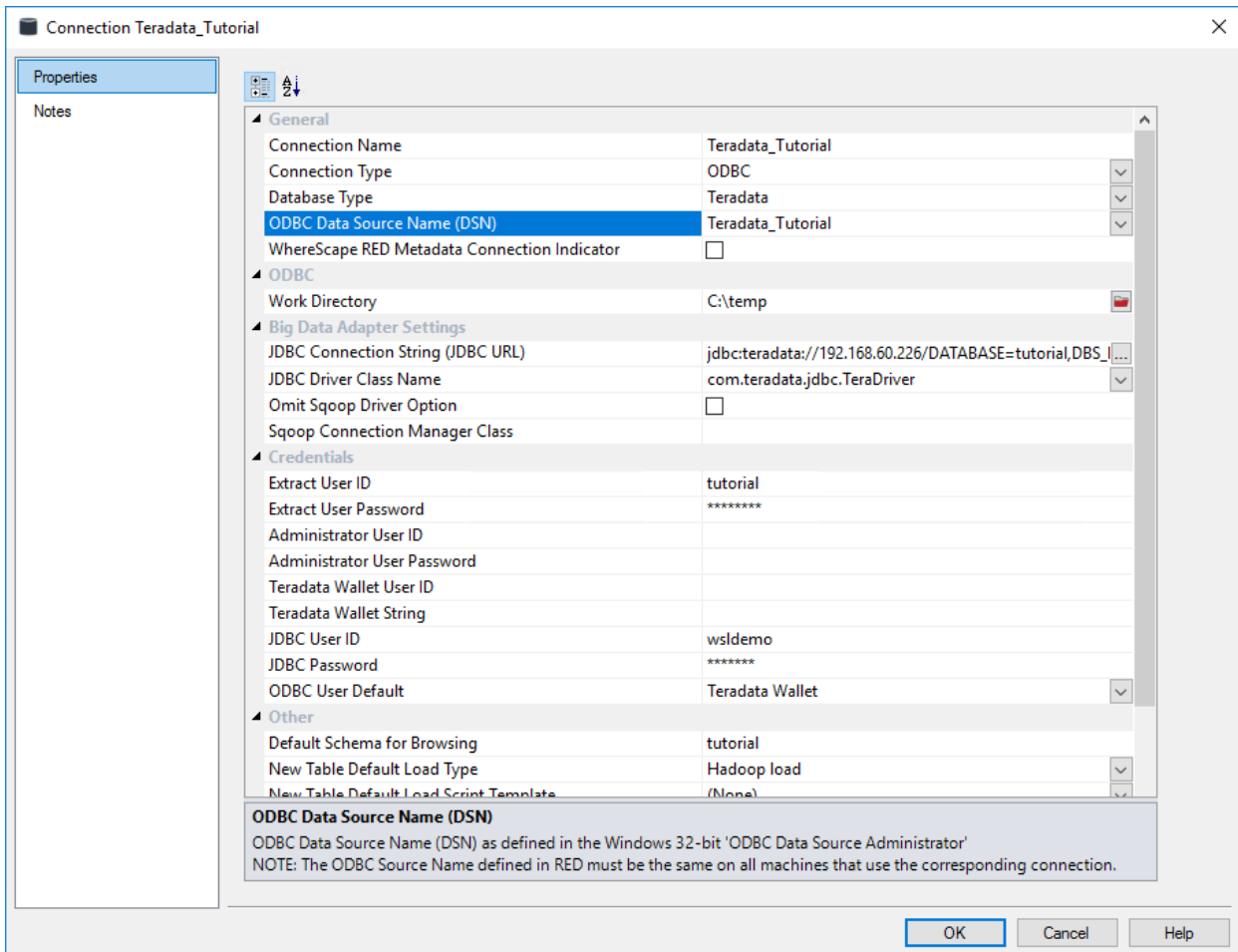
Sqoop Connection Manager Class

Custom Sqoop connection manager class. Corresponds to the --connection-manager command line argument. Leave blank if 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
```



CHAPTER 8

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 179)
- **Storage** (on page 182)
- **Override Create DDL** (on page 189)
- **Source** (on page 190)
- **Documentation Fields** (on page 191)
- **Notes** (on page 192)

IN THIS CHAPTER

| | |
|---------------------------|-----|
| Properties..... | 179 |
| Storage..... | 182 |
| Override Create DDL..... | 189 |
| Source | 190 |
| Documentation Fields..... | 191 |
| Notes..... | 192 |

PROPERTIES

The fields available on the Properties screen depend on the Object Type selected. More specific information is available in the respective chapters describing each Object Type. Rebuilding Tables using Update Procedures and templates is applicable to the objects listed and described in the table below.

The screenshot shows the 'Stage Table stage_budget' Properties dialog box. On the left is a sidebar with 'Properties' selected. The main area contains the following fields and controls:

- Table Name:** Text input field containing 'stage_budget'.
- Table Type:** Dropdown menu set to 'Stage'.
- Unique Short Name:** Text input field containing 'stage_budget' (maximum 22 characters).
- Description:** Large text area.
- Update Procedure:** Dropdown menu set to 'update_stage_budget', with 'Edit', 'Rebuild', and 'Regenerate' buttons.
- Template:** Text input field containing 'wsl_stage_budget_proc_update'.
- Custom Procedure:** Dropdown menu set to '(None)'.
- Rebuild Menu:** A dropdown menu is open under 'Rebuild', showing 'Rebuild without template' and 'Rebuild with template' (checked).
- Timestamps:** A section with three input fields: 'Metadata Structure' (2006-01-05 14:10:30.000), 'Database' (2017-05-30 11:30:05.433), and 'Database Altered:' (2017-05-30 11:30:05.433).

At the bottom right are 'OK', 'Cancel', and 'Help' buttons.

| Fields | Description |
|--------------------------|--|
| Table Name | The user-name of the selected table. |
| Unique Short Name | The short name is derived from the Table Name and is used internally by RED. |
| Table Type | The drop down list provides a list of the available sub types for the selected table type. |
| Description | Optional, free text. |
| Update Procedure | The name of the procedure which will be used when updating the table. |
| Custom Procedure | The name of the procedure which will be used for custom updates of the table. |
| Edit | Edit the content of the displayed update procedure. |

| | |
|-------------------|---|
| Rebuild | Rebuild the Update Procedure. See <i>Rebuilding Update Procedures</i> (on page 181) for more details. |
| Regenerate | Regenerate the selected Update Procedure using the responses to previously provided information. |

REBUILDING UPDATE PROCEDURES

The update procedure for a data warehouse object can be generated by RED using parameters provided by the user, or it can be generated using a prepared template. Templates appropriate to the table type must first be created before they can be selected and used to generate update procedures. The option to select a template is only available if RED detects the presence of an appropriate template.

Note: Update procedures for **Data Vault objects** can only be generated using a template.

Generating an Update Procedure without a template

If RED does not detect a template, the drop-down option is not displayed. RED manually regenerates the update procedure when **Rebuild** is clicked, by prompting the user to respond to a series of input requests. Refer to the **Generating Update Procedure** topic in each of the data warehouse object chapters for further information and the steps to manually generate update procedures.

Generating an Update Procedure Using a Template

If RED detects the presence of a template, the **Rebuild** button provides additional options, in a drop down list, for rebuilding the selected update procedure. Two possible rebuild options are provided:

- **Rebuild** - RED rebuilds the procedure using the last selected option as a default when the Rebuild button is clicked. If a template has not been previously used then RED prompts you for inputs as required for the rebuild.
- **Rebuild without template** - RED rebuilds the update procedure, but ignores any previously used template. RED prompts you for inputs as required for the rebuild.
- **Rebuild with template** - Prompts you to select from a list, a template that is appropriate to the current object type. Depending on the selected template, the user is prompted to provide responses that will be used by the parameters within the template.

Note: If a template has been previously used, RED uses this template by default when **Rebuild** is clicked. The name of this template is displayed below the **Update Procedure field**.

The following table summarizes the conditions that will display the drop-down options in the **Rebuild** button or that require a template. All other conditions will require a manual rebuild.

| Table Object | SQL Server | Oracle | Teradata | DB2 | Greenplum | Netezza | Hive/PDW/Custom |
|--------------|------------|----------|----------|----------|-----------|----------|-----------------|
| Stage | Both | Both | Both | Both | Both | Both | Template Only |
| Data Vault | Template | Template | Template | Template | Template | Template | Template |

| Table Object | SQL Server | Oracle | Teradata | DB2 | Greenplum | Netezza | Hive/PDW/Custom |
|--|------------|--------|---------------------|------|---------------------|---------------------|-----------------|
| Stage | Only | Only | Only | Only | Only | Only | Only |
| Fact | Both | Both | Both | Both | Both | Both | Template Only |
| Fact Rollup | Both | Both | Red Automation Only | Both | Red Automation Only | Red Automation Only | Template Only |
| Fact Kp1 | Both | Both | Red Automation Only | Both | Red Automation Only | Red Automation Only | Template Only |
| Aggregate | Both | Both | Both | Both | Red Automation Only | Red Automation Only | Template Only |
| Dimension | Both | Both | Both | Both | Both | Both | Template Only |
| Data Store | Both | Both | Both | Both | Both | Both | Template Only |
| Normal, Hub, Link, Satellite, Custom1 or 2 | Both | Both | Both | Both | Both | Both | Template Only |

| | |
|----------------------------|--|
| Both | This object type / database combination supports both RED automation for code generation and template based code generation. |
| RED Automation only | This object type / database combination uses RED automation for code generation and template based code generation is not currently available. |
| Template only | This object type / database combination only supports template based code generation. |

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 Teradata example, see *Table Storage Screen - Teradata*.

For a Tabular example see, *Table Storage Screen - Tabular* (on page 186).

For information on changing storage locations for multiple tables at once see, *Bulk Table Storage Change* (on page 187).

TABLE STORAGE SCREEN - TERADATA

Typical Storage screen for a **Teradata** Table:

| Location | |
|-----------------|--------------------|
| Target Location | DataWarehouse:Load |
| Database Type | Teradata |
| Database | WSL_REPOSITORY |
| Temp Database | |

| Storage | |
|-----------------------|-------------------------------------|
| Primary Index | Non-Unique Primary Index (NUPI) |
| Primary Index Name | load_currency_idx_PR |
| Primary Index Columns | currency_code |
| MultiSet | <input checked="" type="checkbox"/> |
| Fallback | <input type="checkbox"/> |
| Data Block Size | |
| Enable Free Space | <input type="checkbox"/> |

Other

| | |
|------------------------|--|
| Optional CREATE Clause | |
|------------------------|--|

Location
The location of the table

Location

Target Location

The target location that defines the path to the location for 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 135). To set default target locations for tables see **Settings - Storage: Target Location** (see "**Target Location**" on page 104).

Database Type

The database type for a connection that is used for target Data Warehouse tables.

Database

The database where the table is located. Leave blank to use the default for the connection or local environment.

Temp Database

The database where temporarily created tables are located. Leave blank to use the default for the connection or local environment. This field is only used in Load and Export processes.

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*.

NOTE: Since Teradata does not support the moving of tables, 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.

Primary Index

Select the Primary Index Type.

Primary Index Name

Name of the Primary Index.

Primary Index Columns

Columns of the Primary Index.

MultiSet

This options makes this table multiset. A Multi Set table allows for duplicate rows.

Fallback

This option enables Fallback. A fallback table is a duplicate copy of a primary table. Each fallback row in a fallback table is stored on a different AMP to the one used for the corresponding row in the primary table. The default is no fallback.

Data Block Size

The block seize for the table. Default is blank.

Enable Free Space

Use Database default Free Space setting. Disabling this will enable setting of the Free Space. Default is blank.

Free Space

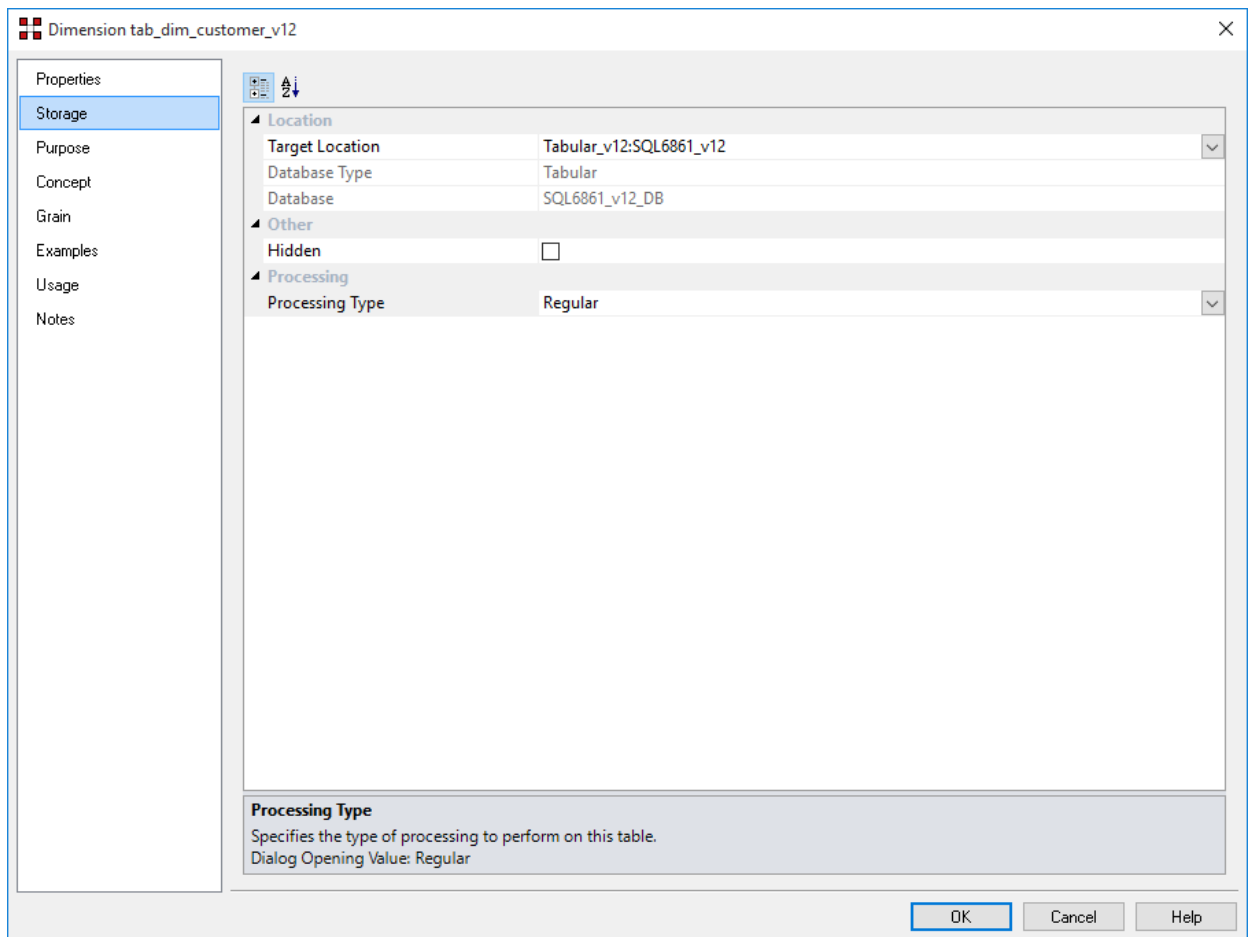
The amount of free space to leave in the table for future update. Enter a value between 0 and 75 for a percentage or to use the Database default set the above option.

Optional CREATE Clause

Database-specific-and-compliant DDL to append to the generated CREATE TABLE statement.

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* (see "*Microsoft Analysis Server 2005+ - OLAP Cubes*" on page 162) . 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.

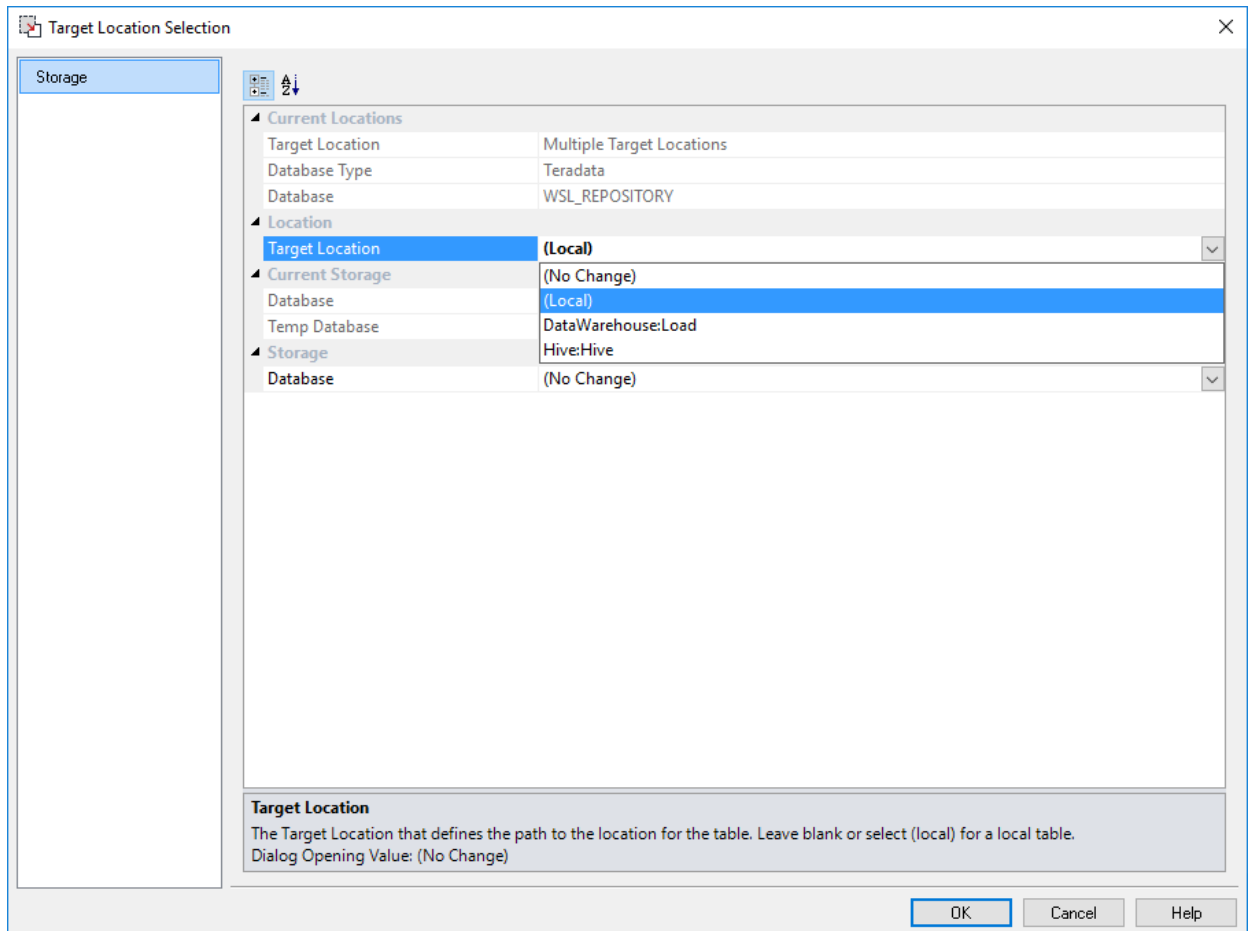
BULK TABLE STORAGE CHANGE

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**.

| Stage Table Name | Short Name | Stage Table Type | Filegroup |
|------------------|----------------|------------------|-----------|
| stage_budget | stage_budget | Stage | |
| stage_customer | stage_customer | Stage | |
| stage | | | |
| stage | | | |
| stage | | | |

- 4 On the Target Location Selection dialog, select the new Target location to change all the selected tables in bulk on the **Target Location** drop-down list.



- 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 Since Teradata does not support the moving of tables, 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.

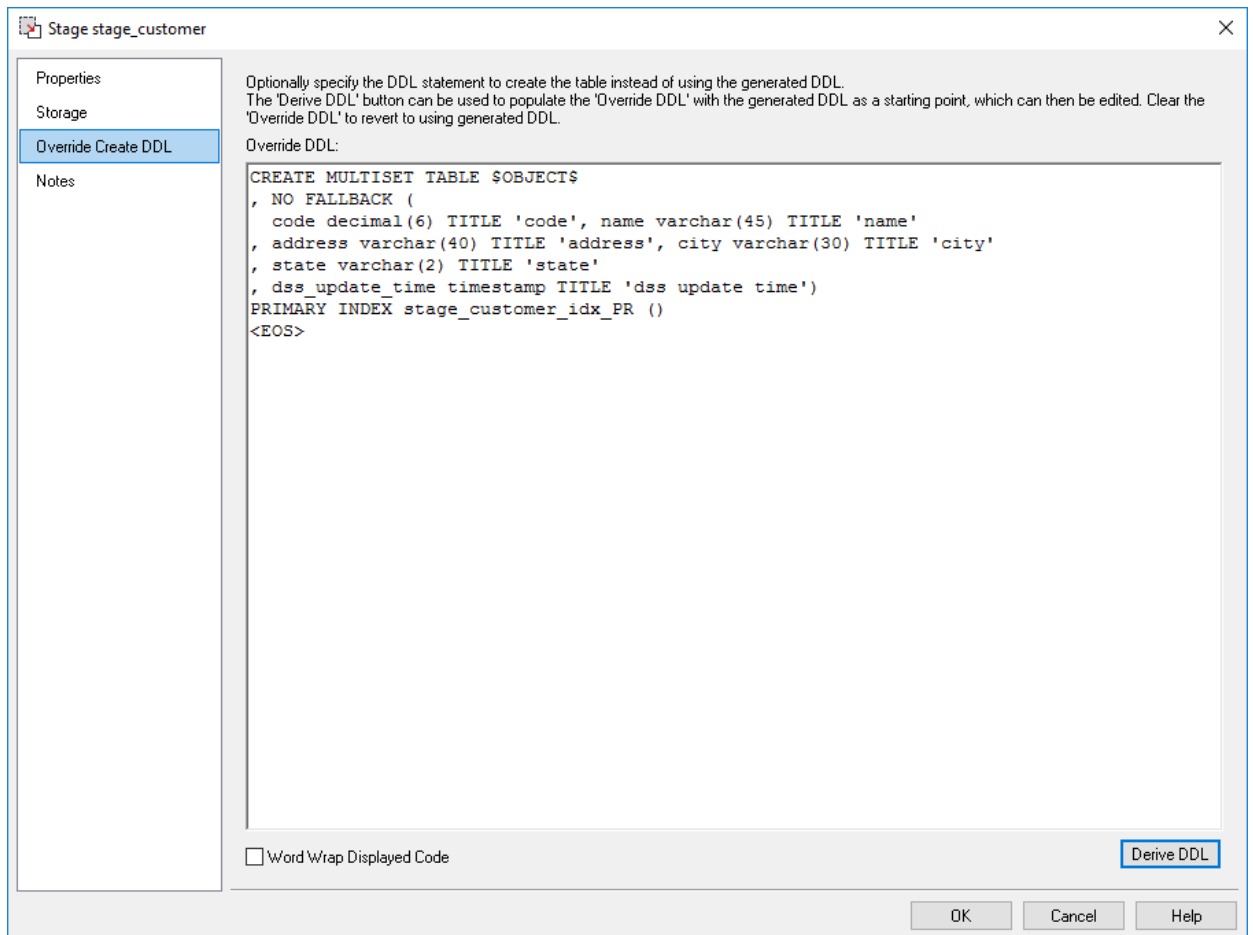
OVERRIDE CREATE DDL

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 "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 at the bottom of the screen 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 194) chapter, or more specifically:

- *Database Link Load - Source Screen* (on page 202)
- *Native ODBC Based Load - Source Screen* (on page 205)
- *TPT Load - Source Screen* (on page 215)
- *Loading Data into RED Load Tables using SSIS* (on page 225)
- *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 appears 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

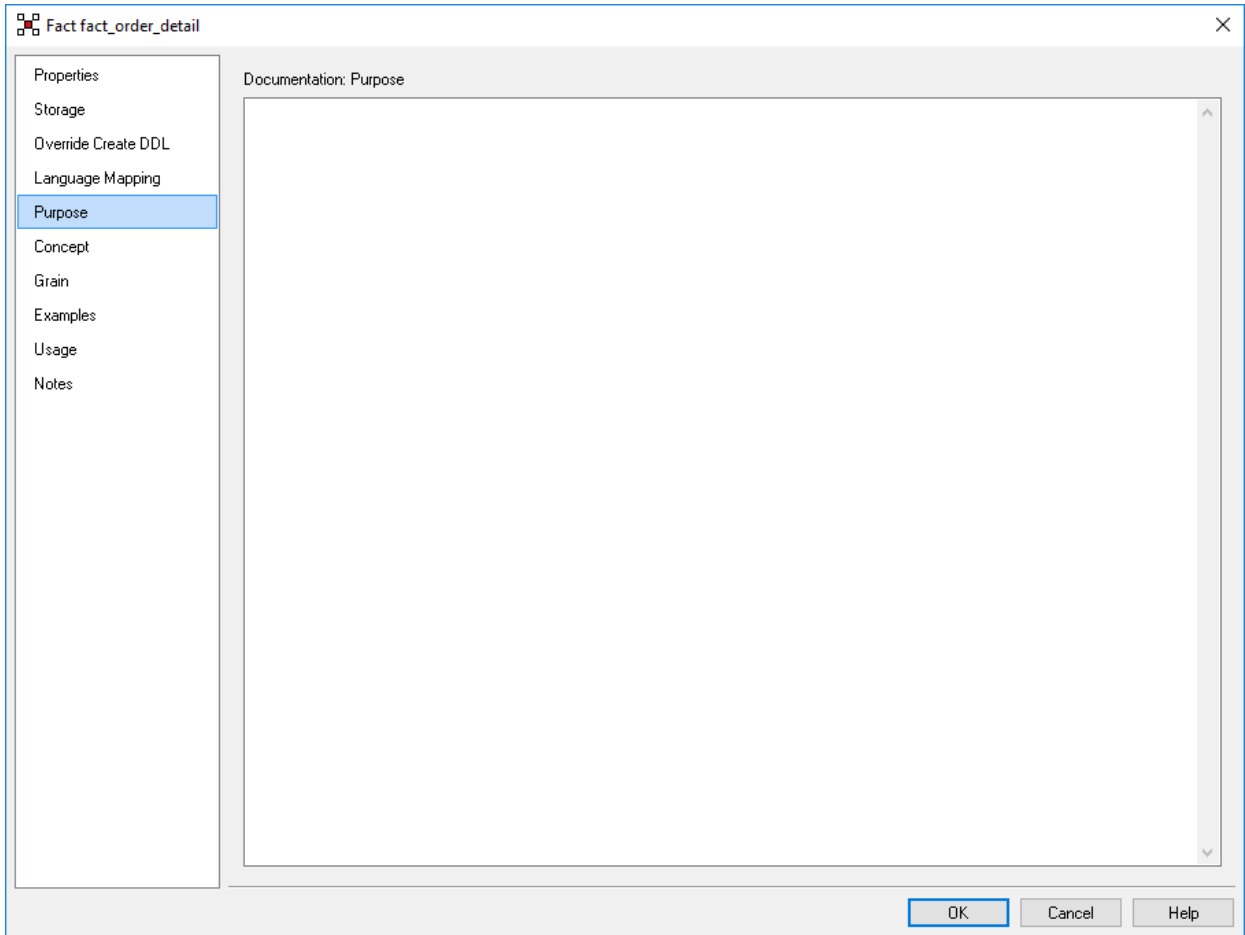
NOTE: The names of these fields are completely arbitrary and can be changed in the generated documentation in the **Tools/Options/Documentation** menu.

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



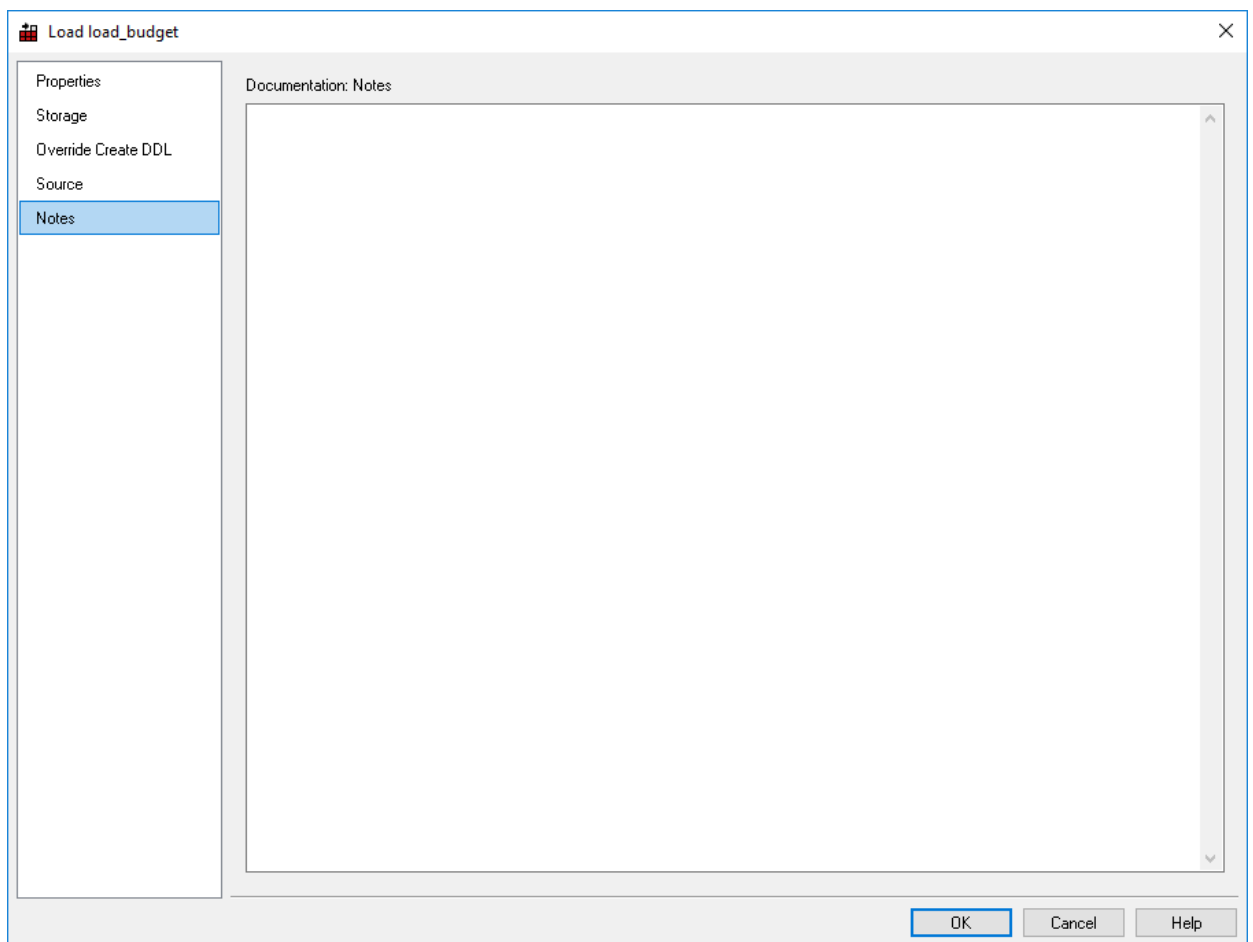
NOTES

The Notes screen is used to enter notes against WhereScape RED objects.

The notes on an object are included in the generated 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.



CHAPTER 9

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. Script-based loads on Windows supports both DOS Batch and **PowerShell scripts** (see "**24.11.1.1 Windows PowerShell Scripts**" on page 657).
- XML file load - loading an XML file from a Windows directory.

IN THIS CHAPTER

| | |
|--|-----|
| Choosing the Best Load Method..... | 196 |
| Load Drag and Drop..... | 197 |
| Database Link Load | 200 |
| ODBC Based Load | 204 |
| Native ODBC Based Load..... | 205 |
| TPT Load | 213 |
| TPT UNIX/Linux Script Load | 220 |
| SSIS Loader | 223 |
| Flat File Loads | 235 |
| XML File Load..... | 260 |
| External Load | 265 |
| Apache Sqoop Load | 265 |
| Handling Missing Source Columns | 272 |
| Load Table Transformations | 275 |
| Changing Load Connection and Schema..... | 275 |

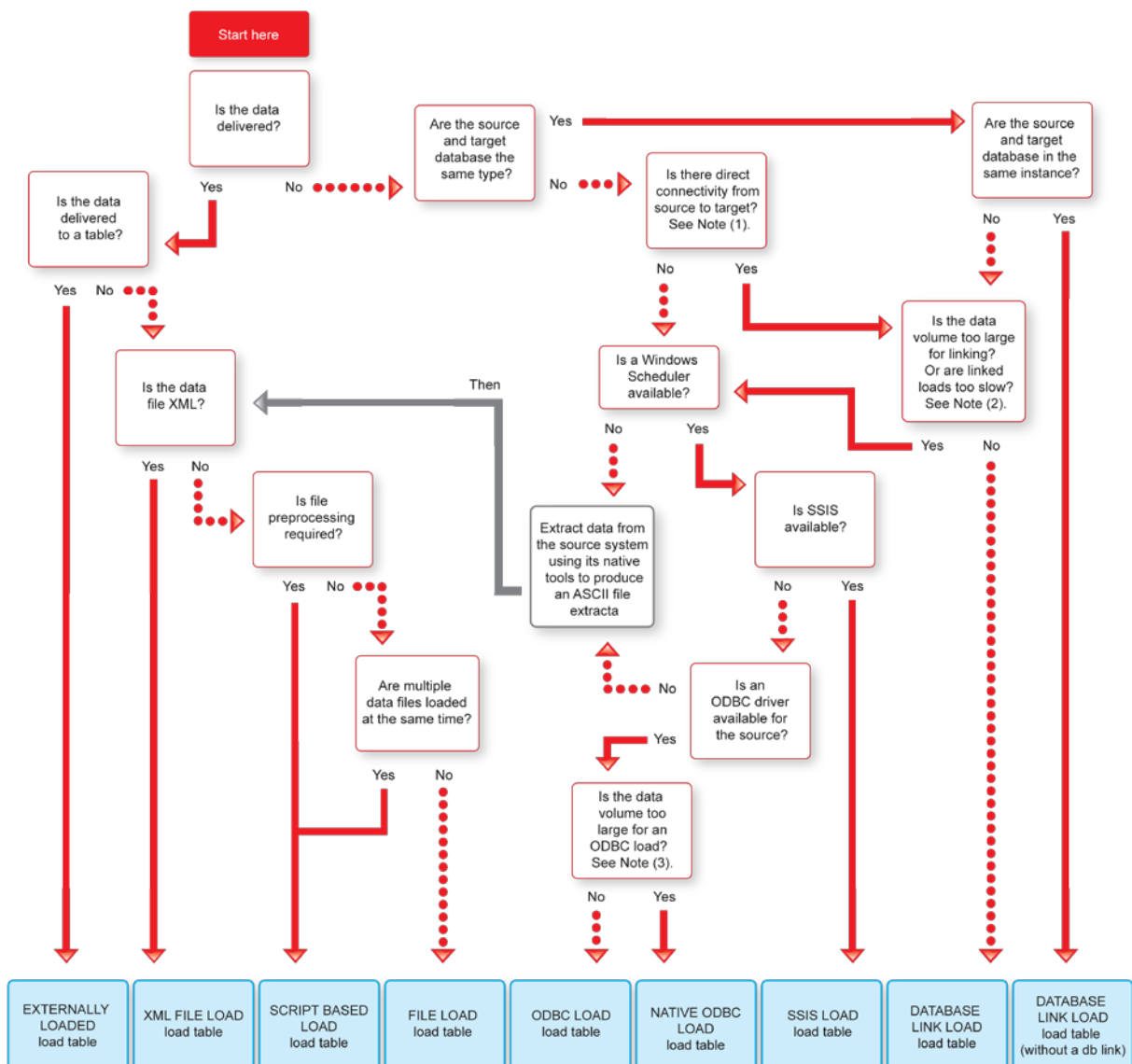
CHOOSING THE BEST LOAD METHOD

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.

Add a New Metadata Object

Define the Type and Name of the New Object.

Specific information for each object type is defined in subsequent screens.

Object Type: Load

Object Name: load_customer

Target Location: (local)

Data Type Mapping: Standard SQL Server to Teradata

Add meta data columns to table

ADD Cancel

NOTE: The option **Add meta data columns to table** is used for creating load tables that are used in creating Data Vault objects. If this option is selected, two DSS columns (**dss_record_source** and **dss_load_date**) are included in the meta data for the table and are populated by transformations. These two DSS columns could equally be applied to other load tables not used in a Data Vault system but are particularly important to comply with the Data Vault standards. Please refer to the **Data Vaults chapter** (see "**Data Vaults**" on page 402) for more details.

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

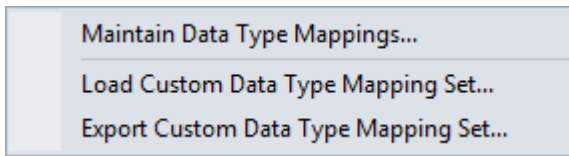
loaded is breached at runtime.

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 Column Transformations** (on page 598)). Incompatible data types that cause load table creation errors are typically caused by:

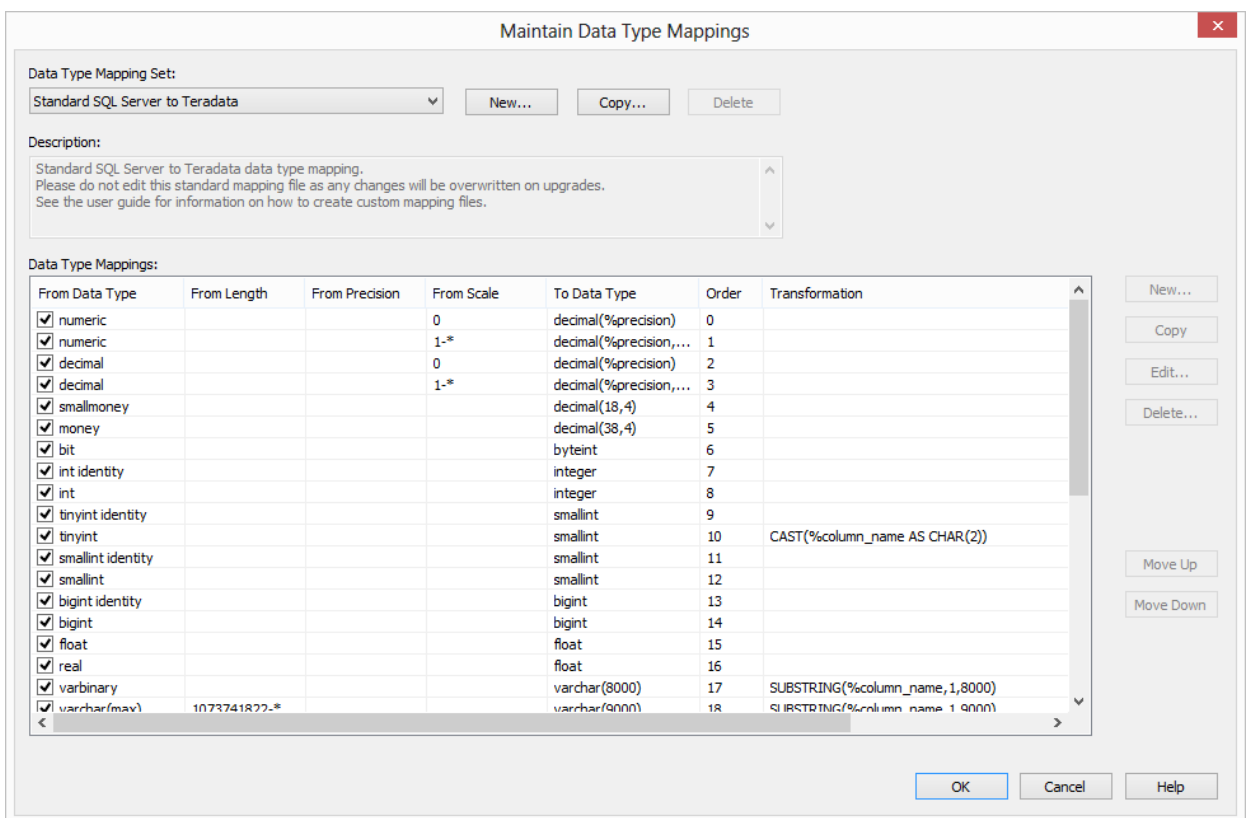
1. User defined data types in the source database.
 2. Incorrect data type mapping during load table definition in WhereScape RED.
 3. Data types that cannot be inserted into (e.g. identity columns on SQL Server)
-

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 1021) for a detailed explanation of Data Type Mappings.

DATABASE LINK LOAD

Database Link Load would have the data loaded from another database located on the same Teradata server into the data warehouse. This load type is used infrequently on Teradata.

DATABASE LINK LOAD - PROPERTIES

The fields of the **Load Table Properties** screen for database link loads are described below:

The screenshot shows the 'Load load_customer' dialog box with the 'Properties' tab selected. The fields are as follows:

- Load Table Name: load_customer
- Unique Short Name: (maximum 22 characters): load_customer
- Description: Customer load table
- Connection: Teradata_Tutorial
- Load Type: Database link load
- Database Link: (empty)
- Script Template: (None)
- Script Name: (None)
- Pre-Load Action: Truncate
- Pre-Load SQL: (empty text area)
- Post Load Procedure: (None)
- Timestamps:
 - Metadata Structure Changed: 2007-11-09 22:29:28.000000
 - Database Created: 2017-03-09 00:57:40.200000
 - Database Altered: 2017-03-09 00:57:40.200000

Buttons: OK, Cancel, Help

Load Table Name

The table name is limited by Teradata to a maximum of 30 characters and must be unique. Table name defaults can be set up (Tools | Options then Naming local or global) to define a prefix or a post fix that can be added in order to identify clearly that this is a load table. Example: load_customer. By default RED uses the prefix load_ for load tables.

Unique Short Name

The table short name is limited in size to 22 characters and must be unique. The short name is used in the naming of indexes, keys and procedures.

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 on bulk see *Changing load Connection and Schema* (on page 275).

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

This field is not active in WhereScape RED for Teradata.

Script Template

The script template used for a script based load.

Script Name

This field is only active for script based loads.

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.

The contents of pre-load sql can be a sql statement or a procedural block. 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 |

```
delete table load_customer all
```

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 Teradata.

Post Load Procedure

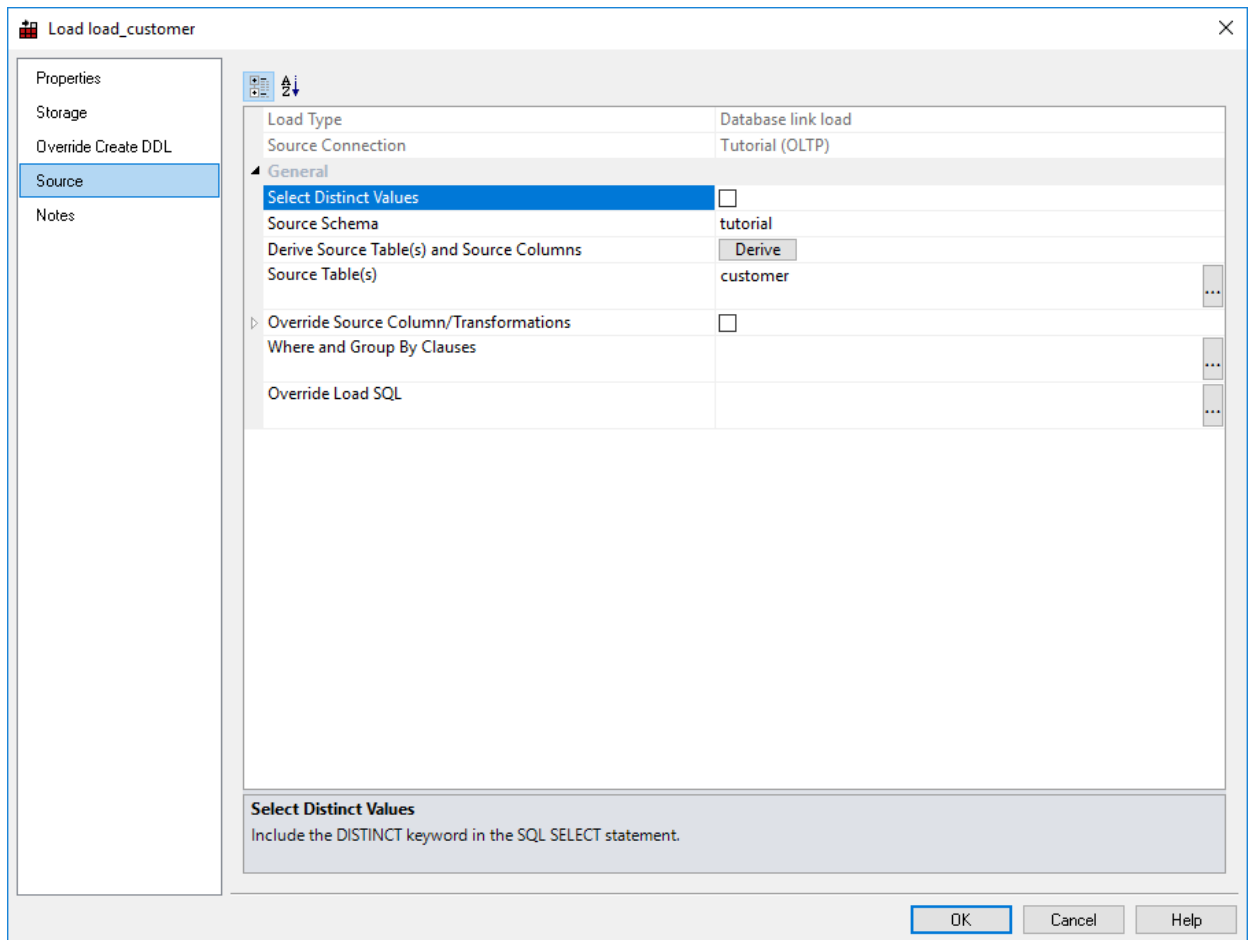
A procedure that is executed immediately following the load. If you execute an externally loaded table, no load occurs, but a post load procedure can still be executed.

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 SCREEN

The fields on the **Load Table Source screen** for database link loads 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 screen.

Source Connection

The connection that identifies the source database. Can be specified via the Properties page.

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 275) below for a fuller explanation.

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.

Parameter values can be in-line replaced and included in the 'Where' clause. Prefix the parameter name with a '\$P' and add a trailing '\$'. For example, if we have a parameter called SALES_LOCATION_CODE we could construct a statement such as WHERE location_code = '\$PSALES_LOCATION_CODE\$' AND region_code = 'NY'. When this statement is executed, the value of the parameter will replace the parameter name. For example if the parameter was set to 'New York' then the statement would execute as: WHERE location_code = 'New York' AND region_code = 'NY'.

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.

ODBC BASED LOAD

An ODBC based load provides an extensive option for acquiring data from many sources. It is slower than most other forms of load, 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, however the data is loaded into the server that is running the scheduler and then pushed to the data warehouse database.

The obvious disadvantage in an ODBC based load is the two network transactions required, and the overhead placed on the Scheduler Server.

The properties screens for an ODBC based load are the same as those of a database link load. Refer to ***Database Link Load - Properties*** and ***Database Link Load - Source Screen*** (on page 202) for more details.

NATIVE ODBC BASED LOAD

A Native ODBC based load is similar to an ODBC load, except it provides a faster method to move data from another database into Teradata.

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 Teradata database via a fastload session.

A scheduler load will perform in the same way, however the data is loaded into the server that is running the scheduler and then pushed to the data warehouse database.

For fastload 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, using the correct casting function for the source database.

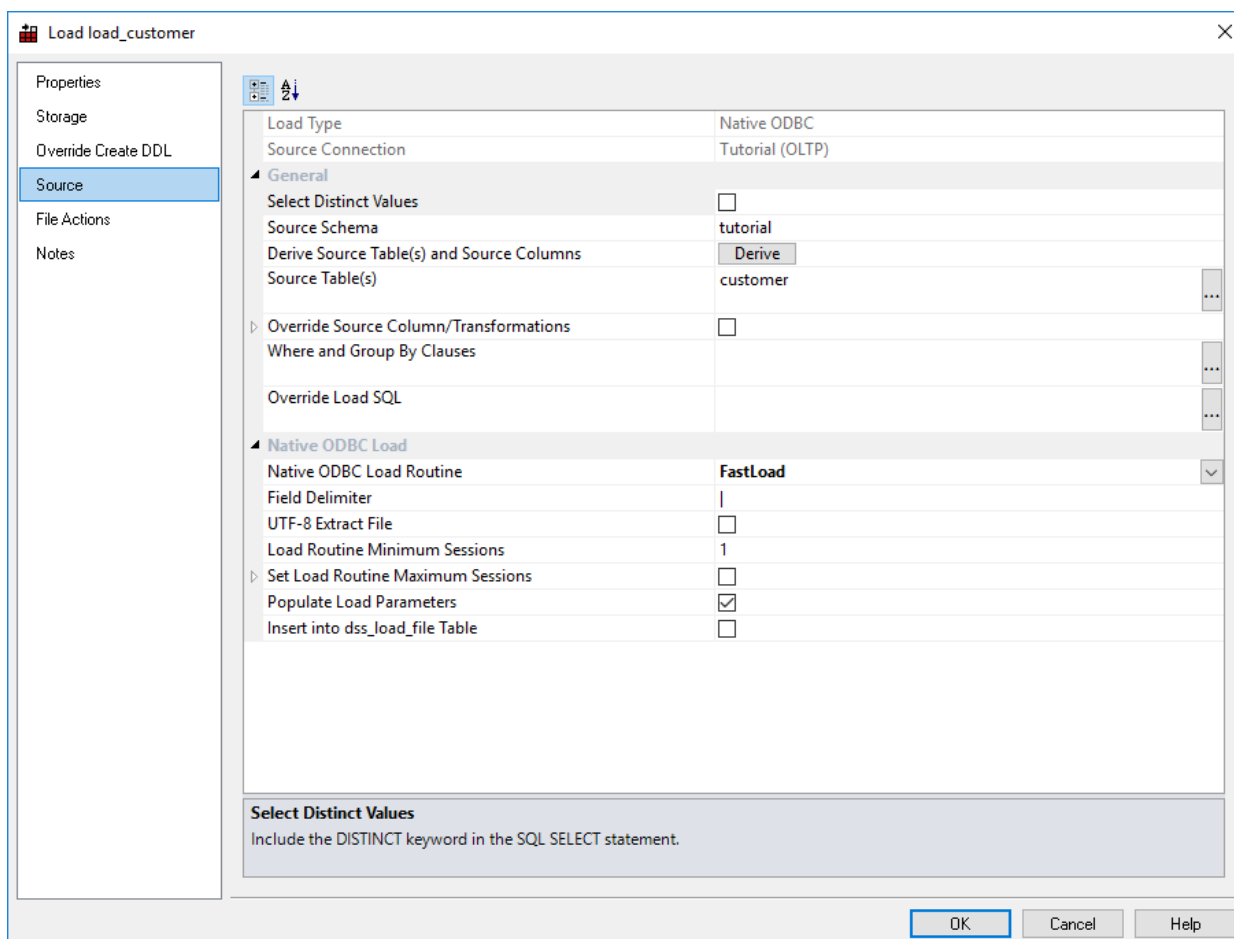
The properties and storage screens for a Native ODBC based load are the same as those of a database link load. Refer to ***Database Link Load - Properties*** for more details.

Details of the Native ODBC Source screen follow.

NOTE: If you are doing **Native Loads using UNIX** and **LINUX**, see section ***Native Loads using UNIX and LINUX*** for more details about this type of load.

NATIVE ODBC BASED LOAD - SOURCE SCREEN

The fields on the **Source** tab of the properties screen for Native ODBC loads 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 screen.

Source Connection

The connection that identifies the source database. Can be specified via the Properties screen.

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.

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.

UTF-8 Extract File

Data will be loaded via a UTF-8 format file. The default is unselected.

Populate Load Parameters

Populate any load-related WhereScape RED parameters, which may be used for validation purposes.

Insert into dss_load_file Table

Populate the metadata table dss_load_file in any generated post_load procedure.

FILE ACTIONS

A file action defines a ftp or copy step that happens to files before or after they are loaded.

Creating a File Action

Before creating a ftp file action, ensure a connection exists for the remote server where the files will be transferred to.

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 the **Action Type** from the drop-down list.
- 4 Enter the **Action Description**.
- 5 Choose the **Action Program** from the drop-down list.
- 6 For a ftp action:
 - Choose the **Action Connection** for the remote server where the file will be transferred to.
 - Enter the ftp commands in **ftp command 1** thru **ftp command 9**.
- 7 For a copy action:
 - To move the data file, enter the copy to location into **File Directory**. If the file is to be renamed at the same time, enter a new name into **File Name**, otherwise enter %FILE_NAME%.
 - To move the trigger file, enter the copy to location into **Trigger Directory**. If the trigger file is to be renamed at the same time, enter a new name into **Trigger Name**, otherwise enter %TRIG_NAME%.
- 8 Click on **Save (Update) Action**.
- 9 Repeat if necessary for additional file actions.
- 10 Click **OK**.

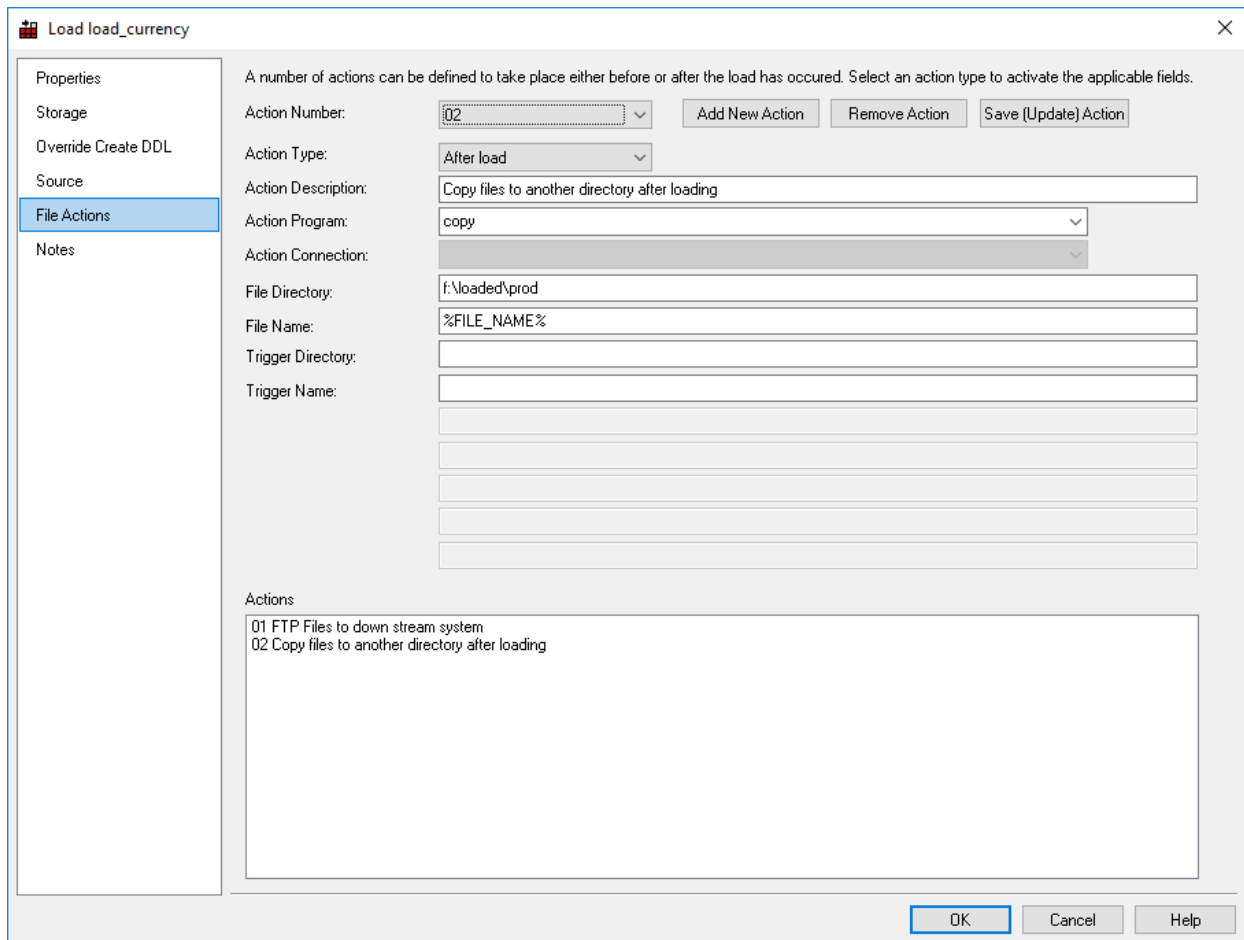
Sample ftp file action:

The screenshot shows a dialog box titled "Load load_currency" with a close button (X) in the top right corner. On the left is a sidebar with a tree view containing the following items: Properties, Storage, Override Create DDL, Source, File Actions (highlighted in blue), and Notes. The main area contains the following fields and controls:

- Instruction: "A number of actions can be defined to take place either before or after the load has occurred. Select an action type to activate the applicable fields."
- Action Number: A dropdown menu with "01" selected. To its right are buttons for "Add New Action", "Remove Action", and "Save (Update) Action".
- Action Type: A dropdown menu with "After load" selected.
- Action Description: A text field containing "FTP Files to down stream system".
- Action Program: A dropdown menu with "ftp" selected.
- Action Connection: A dropdown menu with "UNIX" selected.
- ftp command 1: A text field containing "cd/app/landing/prod".
- ftp command 2: A text field containing "lcd c:\temp".
- ftp command 3: A text field containing "as".
- ftp command 4: A text field containing "put %File_NAME%".
- ftp command 5: An empty text field.
- ftp command 6: An empty text field.
- ftp command 7: An empty text field.
- ftp command 8: An empty text field.
- ftp command 9: An empty text field.
- Actions: A list box containing one entry: "01 FTP Files to down stream system".

At the bottom right of the dialog are three buttons: "OK", "Cancel", and "Help".

Sample copy file action:



NATIVE LOADS USING UNIX AND LINUX

Sometimes the Teradata database server has a LINUX non-tpa node or a MP-RAS UNIX non-tpa node used for loading data from files. This can provide significantly better platform for loading files into Teradata than a network connected Windows Scheduler server.

Native ODBC loads need to extract data from the source system using ODBC on a Windows machine. An additional feature in WhereScape RED can be enabled to transfer the extract file to LINUX or MP-RAS and then load the file on the remote non-tpa node.

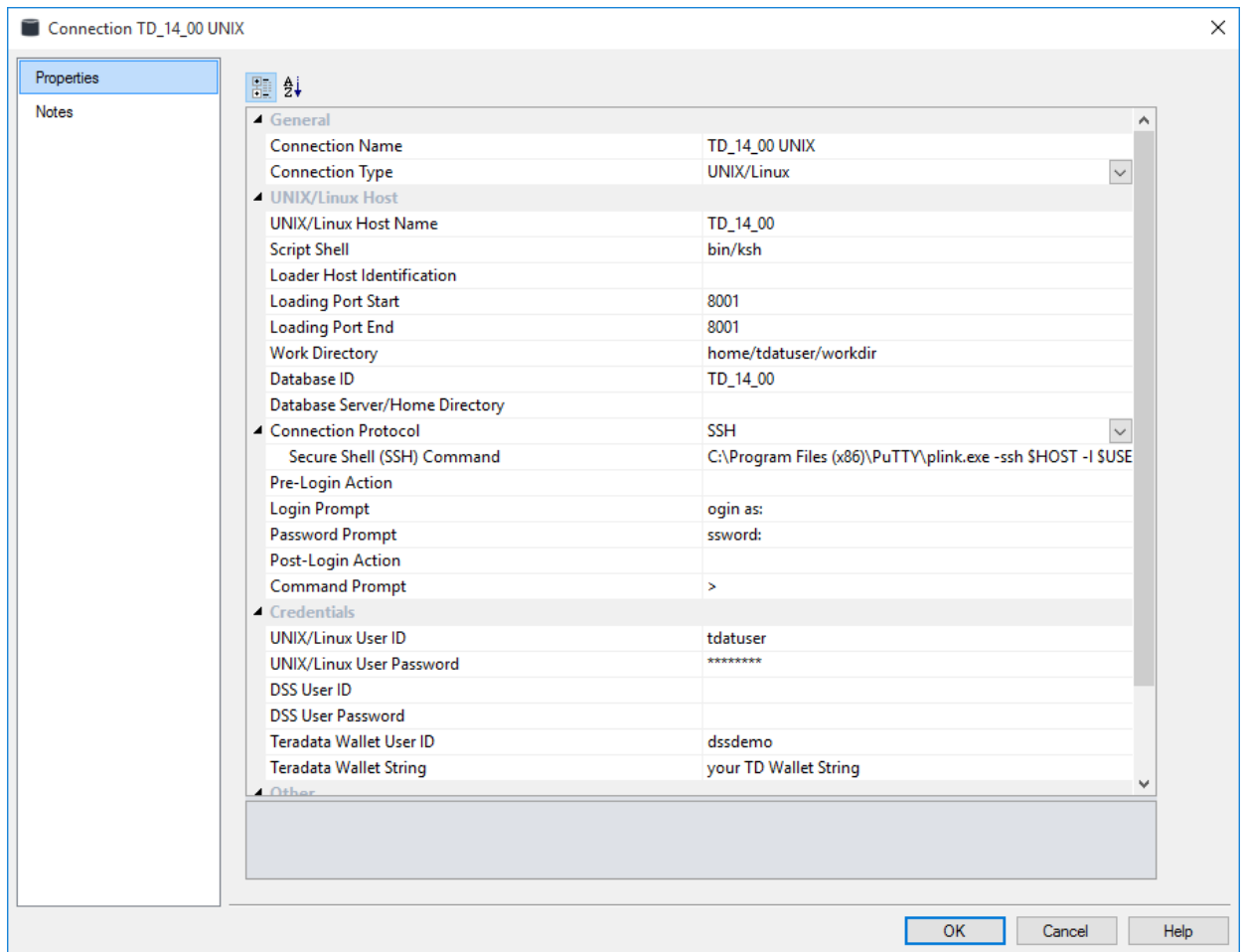
The following steps will enable this functionality when Native ODBC loads are scheduled:

- 1 Create a unix connection for the non-tpa node (see **Connections** (on page 133) for more information on creating connections).

The "Unix user id" and "Unix password" are used to sign in to the remote machine specified by "Host name".

The work directory is the default location extract files will be transferred to.

The following shows the minimum fields that need to be completed:



- 2 Create a new file action on each load table (see **File Actions** (on page 208) for more information on file actions).

The file action should be look something like this:

The screenshot shows a dialog box titled "Load load_customer" with a close button (X) in the top right corner. On the left is a sidebar with a tree view containing "Properties", "Storage", "Override Create DDL", "Source", "File Actions" (which is selected and highlighted in blue), and "Notes". The main area contains the following fields and controls:

- Header text: "A number of actions can be defined to take place either before or after the load has occurred. Select an action type to activate the applicable fields."
- Action Number: A dropdown menu with "02" selected. To its right are three buttons: "Add New Action", "Remove Action", and "Save (Update) Action".
- Action Type: A dropdown menu with "After load" selected.
- Action Description: A text input field containing "Move file from MPRAS to load and initiate load".
- Action Program: A dropdown menu with "unixload" selected.
- Action Connection: A dropdown menu.
- ftp command 1: A text input field containing "lcd \$WORKDIR\$".
- ftp command 2: A text input field containing "cd \$UNIXDIR\$".
- ftp command 3: A text input field containing "as".
- ftp command 4: A text input field containing "prompt".
- ftp command 5: A text input field containing "mput \$FILE\$".
- ftp command 6: An empty text input field.
- ftp command 7: An empty text input field.
- ftp command 8: An empty text input field.
- ftp command 9: An empty text input field.
- Actions: A list box containing two entries: "01 Delete all temporary files" and "02 Move file from MPRAS to load and initiate load".

At the bottom right of the dialog are three buttons: "OK", "Cancel", and "Help".

Note: The variables \$WORKDIR\$ and \$UNIXDIR\$ refer to the work directories of the Native ODBC connection and Unix connection respectively. The variable \$FILE\$ represents the extract file being transferred to LINUX or MP-RAS.

- 3 Ensure ws_sched_tera_550.sh script has been set up on the MP-RAS or LINUX non-tpa node and is scheduled to run on the machine every 5 or 10 minutes using cron or any other scheduling software on the non-tpa node (see the appropriate section in the *WhereScape RED Installation and Administration Guide* for more information on ws_sched_tera_550.sh).

TPT LOAD

A TPT ODBC based load is similar to an ODBC load, except it provides a faster method to move data from another physical database server into your Teradata database. This includes moving data from one Teradata server to another and moving data from any other relational database into Teradata.

NOTE: The **TPT Load** type can only be used with a **Windows** scheduler. If using a **UNIX/Linux** scheduler, refer to section 9.7 and details for using the **TPT Script Load** type for UNIX/Linux.

WhereScape RED runs a TPT ODBC load, by generating a TPT script based on your metadata. The TPT script includes the following steps:

- 1 Extract from the source database using the TPT ODBC operator.
- 2 Import into the Teradata database using either the TPT LOAD or TPT UPDATE operator.

Cross database platform TPT ODBC loads present some challenges due to incorrect assumptions made by TPT from time to time. You may need to add 'During' load transformations to your load tables and/or modify numeric data types from their apparent equivalent data type for a load to complete successfully. This is because TPT assumes a data type with the same name in another database is the same size in Teradata; of course this is not always the case.

The properties and storage screens for a TPT ODBC based load are the same as those of a database link load.

NOTE: TPT Loads from ODBC connections in a Teradata repository must have ODBC connections set up using the appropriate DataDirect ODBC driver.
TPT Loads from ODBC Connections will not work if a non-DataDirect driver is used.

Refer to the previous section for details. Details of the source mapping screen follow in the next section.

Load load_customer

Properties

Storage

Override Create DDL

Source

Notes

Load Table Name: load_customer

Unique Short Name: (maximum 22 characters) load_customer

Description: Customer load table

Connection: Teradata_Tutorial

Load Type: TPT load

Script Connection:

Script Template: (None)

Script Name: (None)

Pre-Load Action: Truncate

Pre-Load SQL:

Post Load Procedure: (None)

Timestamps

| | | |
|-----------------------------|----------------------------|----------------------------|
| Metadata Structure Changed: | Database Created: | Database Altered: |
| 2007-11-09 22:29:28.000000 | 2017-03-09 00:57:40.200000 | 2017-03-09 00:57:40.200000 |

OK Cancel Help

TPT LOAD - SOURCE SCREEN

The fields on the **Source** tab of the Properties screen for TPT loads are described below:

The screenshot shows the 'Load load_customer' Properties dialog box with the 'Source' tab selected. The dialog is divided into several sections:

- General:**
 - Load Type: TPT load
 - Source Connection: Tutorial (OLTP)
 - Select Distinct Values:
 - Source Schema: tutorial
 - Derive Source Table(s) and Source Columns: Derive
 - Source Table(s): customer
 - Override Source Column/Transformations:
 - Allow Missing Source Columns:
 - Fail when Incomplete Load:
 - Where and Group By Clauses: ...
 - Override Load SQL: ...
- Teradata Parallel Transporter (TPT):**
 - TPT Load Type: Load TPT
 - TPT Character Set: ASCII
 - TPT ODBC Operator Attributes: ...
 - TPT Load/Update Operator Attributes: ...
 - TPT Job Name: ...
 - TPT Build Command Options: ...
 - TPT Shared Memory Size: (Default)
 - TPT Minimum Sessions: 1
 - Set TPT Maximum Sessions:
 - Load Read Instances: 1
 - Load Write Instances: 1
- Select Distinct Values:**
 - Include the DISTINCT keyword in the SQL SELECT statement.

Buttons at the bottom: OK, Cancel, Help.

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.

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

This ignores the source and transformation details of this table's columns and uses the override details specified below instead.

Allow Missing Source Columns

Allow the load to occur when one or more of the source columns do not exist. (see the section on *Handling missing source columns* (on page 272)).

Fail when incomplete Load

Controls whether the load reports failure when all the rows extracted are not loaded. The specified exit status impacts any remaining tasks in the currently running job. When fail is specified, the WhereScape RED Scheduler will stop/fail the job and hold any remaining tasks when the load fails. In contrast, when fail is disabled the scheduler will continue to run any dependent tasks in the job that is running.

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.



WhereScape RED 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.

TPT Load Type

Load TPT, Update TPT or Stream TPT.

NOTE: When importing a Model from 3D to RED, select **Load TPT** instead of Fastload as the Default File Loader method. FastLoad is not a valid option for loading Linux files to Teradata. To change the Default File Loader, go to **Tools->Options->Code Generation->General**.

TPT Character Set

Teradata-compliant Character Set Name to use when loading, such as ASCII, UTF8, UTF16.

TPT ODBC Operator Attributes

Optional comma-delimited list of TPT ODBC operator Attributes, e.g. INTEGER DataBlockSize = 2048.

TPT Load/Update Operator Attributes

Optional comma-delimited list of TPT Load Operator or TPT Update Operator Attributes.

TPT Job Name

Job Name for TPT. If not set, will default to Load Name.

TPT Build Command Options

Additional options included as part of the TBuild call.

TPT Shared Memory Size

Shared memory size can be specified in bytes, kilobytes or megabytes. Examples include 2091752,2048K,2M.

TPT Minimum Sessions

Optional minimum number of sessions[1-99]. The default is one session.

Set TPT Maximum Sessions

Enables the 'TPT Maximum Sessions' property. Optional maximum number of sessions[1-99]. The default is one session per available Access Module Processor (AMP). The maximum value cannot exceed the number of available AMP's.

Load Read Instances

Optional number of TPT read instances[1-99]. The default is one instance.

Load Write Instances

Optional number of TPT write instances[1-99]. The default is one instance.

TPT LogView Command Options

Additional options included as part of the TLOGVIEW call. -f*' can be added to this field to get enhanced logging to diagnose issues in the event of a failure.

CLEANUP AFTER TPT LOAD FAILURE

TPT runs all jobs in checkpoint mode by default; and if any one of these jobs fail, it can restart based on the last checkpoint taken for the job.

Automatic Restart

An automatic restart means that a job can restart on its own, without you manually having to resubmit the job. With the Start-of-Data and End-of-Data checkpoints, a job can automatically restart itself when a "retry-able" error occurs (such as a database restart or deadlock) before, during, or after the loading of data. You need to consider the following when dealing with automatic restarts:

Jobs can automatically restart as many times as specified by the value of the RETRY option of the TPT job-launching command. By default, a job can restart up to five times.

If no checkpoint interval is specified for a job, and the job fails during processing, the job restarts either at the Start-of-Data checkpoint or at the End-of-Data checkpoint, depending on which one is the last recorded checkpoint in the checkpoint file.

To avoid reloading data from the beginning, you should specify a checkpoint interval when launching a job so that the restart can be done based on the most recent checkpoint taken.

Manual Restart

If a job fails and terminates, a manual restart can be accomplished by resubmitting the same job with the original job-launching command. By default, all TPT jobs are checkpoint restartable, using one of the two checkpoints at Start-of-Data and End-of-Data.

TPT also provides recovery across job steps within a job, thus if a job has multiple steps, a checkpoint will be created for each successful step. This will allow a job to restart from the failed step by skipping the successful steps. If, for example, you have a step to create or drop tables before a data loading step begins, and the job fails in the data loading step; a restart of the job would resume from the data loading step without recreating or dropping the tables. This can be contrasted with some of the utilities, such as Fastload, where the script might contain the statements DROP TABLE and CREATE TABLE. Such a script could not be used across restarts because those DDL statements would be re-issued.

Removing Checkpoint Files

Job checkpoint files are automatically created by TPT and they are deleted if the job completes without an error. You will however need to remove these checkpoint files before they are automatically deleted if you wish to:

- Rerun an interrupted job from the beginning, rather than restarting it from the last checkpoint taken before the interruption occurred.
- Abandon an interrupted job and run another job, but the new checkpoint files will have the same names as the existing checkpoint files, due to the use of the same job name (or the default checkpoint files created based on the logon user ID).

TPT provides a special command for users to remove checkpoint files, based on either the user ID or the job name.

If the "tbuild" command specifies a job name, the "twbrmcpc <job name>" command can be used. If the "tbuild" command does not specify a job name, the "twbrmcpc <user ID>" can be used. For z/OS, the deletion of checkpoint files can be done through the MVS/ISPF facility.

If you want to delete checkpoint files manually, you can use one of the following commands:

- del %TWB_ROOT%\checkpoint\<job-name>.*
- del %TWB_ROOT%\checkpoint\<user-id>.*

If you want to delete checkpoint files from a user-defined directory, you can use one of the following commands:

- del <user-defined directory>\<job-name>.*
- del <user-defined directory>\<user-id>.*

Note: If a manual restart is necessary and the checkpoint files have been removed, the simplest method to clear the load lock on a table is to recreate the table.

TPT UNIX/LINUX SCRIPT LOAD

Create a **Unix** connection as in the example below:

| Connection Unix | |
|--|-----------------------|
| General | |
| Connection Name | Unix |
| Connection Type | UNIX/Linux |
| UNIX/Linux Host | |
| UNIX/Linux Host Name | 192.XXX.XX.XXX |
| Script Shell | /bin/ksh |
| Loader Host Identification | |
| Loading Port Start | 8001 |
| Loading Port End | 8001 |
| Work Directory | /tmp/ |
| Database ID | TD_14_00 |
| Database Server/Home Directory | |
| Connection Protocol | |
| Secure Shell (SSH) Command | /home/red/work/dir |
| Pre-Login Action | |
| Login Prompt | ogin as: |
| Password Prompt | ssword: |
| Post-Login Action | |
| Command Prompt | ~> |
| Credentials | |
| UNIX/Linux User ID | tdatuser |
| UNIX/Linux User Password | **** |
| DSS User ID | |
| DSS User Password | |
| Teradata Wallet User ID | dssdemo |
| Teradata Wallet String | your TD Sallet String |
| Other | |
| Command Prompt UNIX/Linux command prompt or the tail-end of that prompt. [Typically \$]. | |

TPT UNIX SCRIPT LOAD - PROPERTIES

The fields on the TPT UNIX Script Load Properties screen are the following:



WhereScape RED TIP:

When doing **TPT Script based loads**, it is easy to use the **rebuild** button to the right of the Script-name field to rebuild the scripts.

Load Table Name

The table name is limited by Teradata to a maximum of 30 characters and must be unique. Table name defaults can be set up (**Tools>Options** then Naming local or global) to define a prefix or a post fix that can be added in order to identify clearly that this is a load table. Example: load_customer. By default RED uses the prefix load_ for load tables.

Unique Short Name

The table short name is limited in size to 22 characters and must be unique. The short name is used in the naming of indexes, keys and procedures.

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 section on *Changing load Connection and Schema* (on page 275).

Load Type

The load type is typically defined by the connection, and should not normally be changed. This drop-down lists all valid load types for the connection.

ODBC Script Connection

Choose the **Unix** Connection.

Script Template

The script template used for a script based load.

Script Name

This field is only active if this is a script based load.



TIP: Use the **Rebuild** button after selecting the relevant script to be rebuilt on the the Script 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.

The contents of pre-load sql can be a sql statement or a procedural block. 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 |
| delete table load_customer all |

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.

At the bottom of the Load Table Properties screen are three fields that display information on that load table:

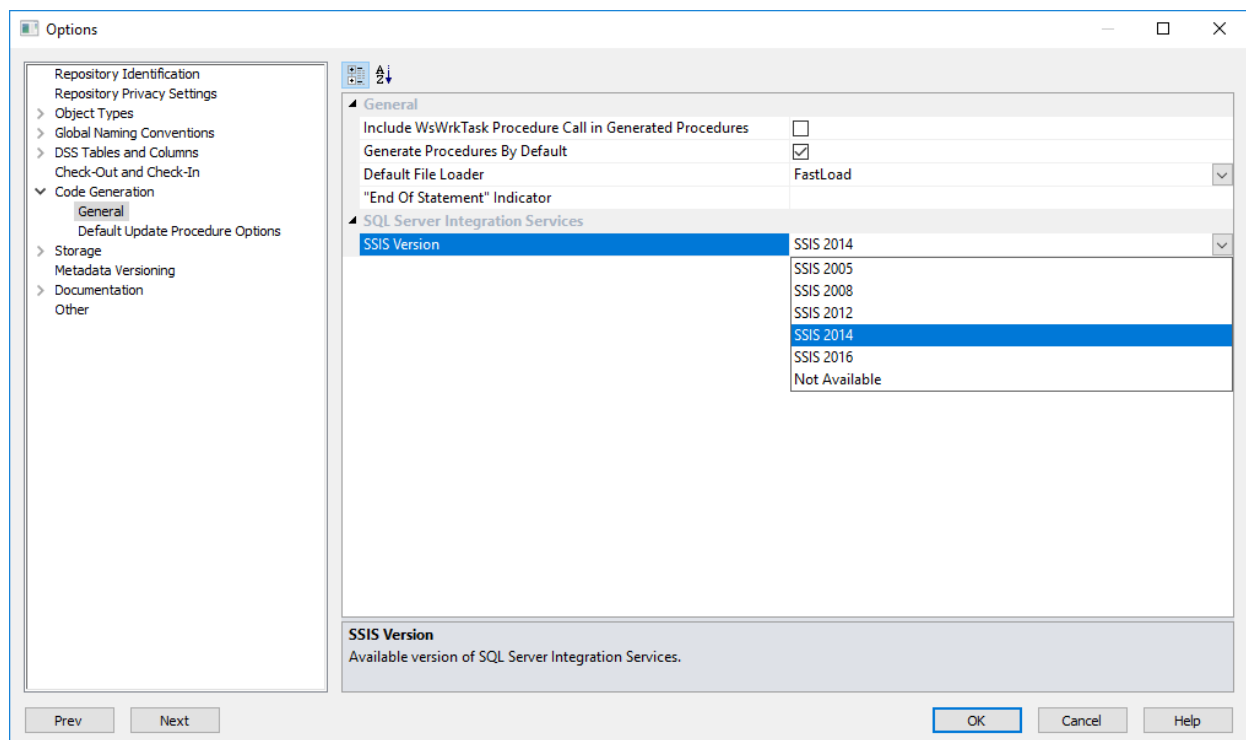
1. Date table structure was last updated
 2. Date created in database
 3. Date last updated in database
-

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 Teradata can be 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.

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 Flat Files 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

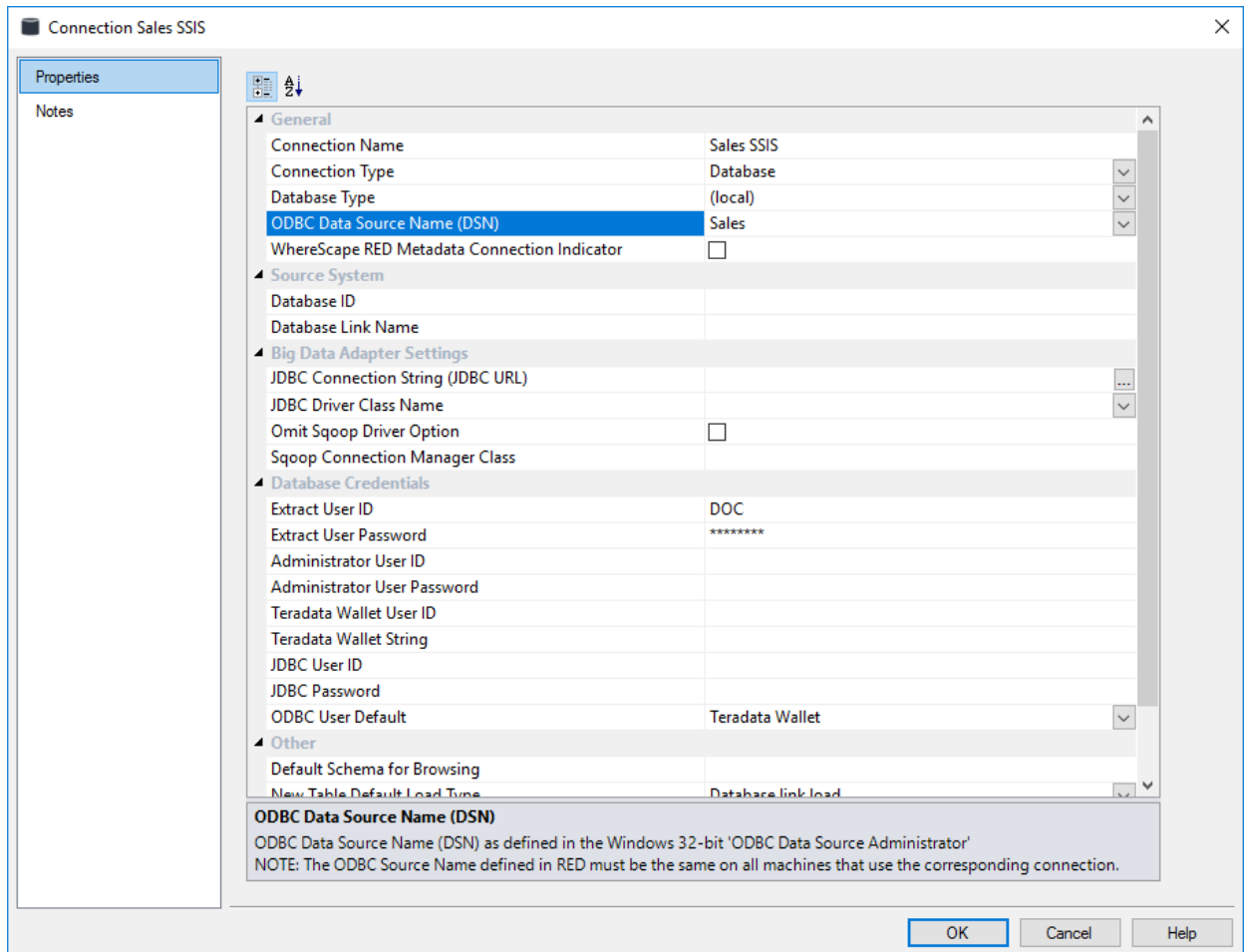
If the connection is a database load then there is additional connection information that should be supplied to use SSIS as a loading option.

This additional information needs to be supplied on both the source connection and the data warehouse connection.

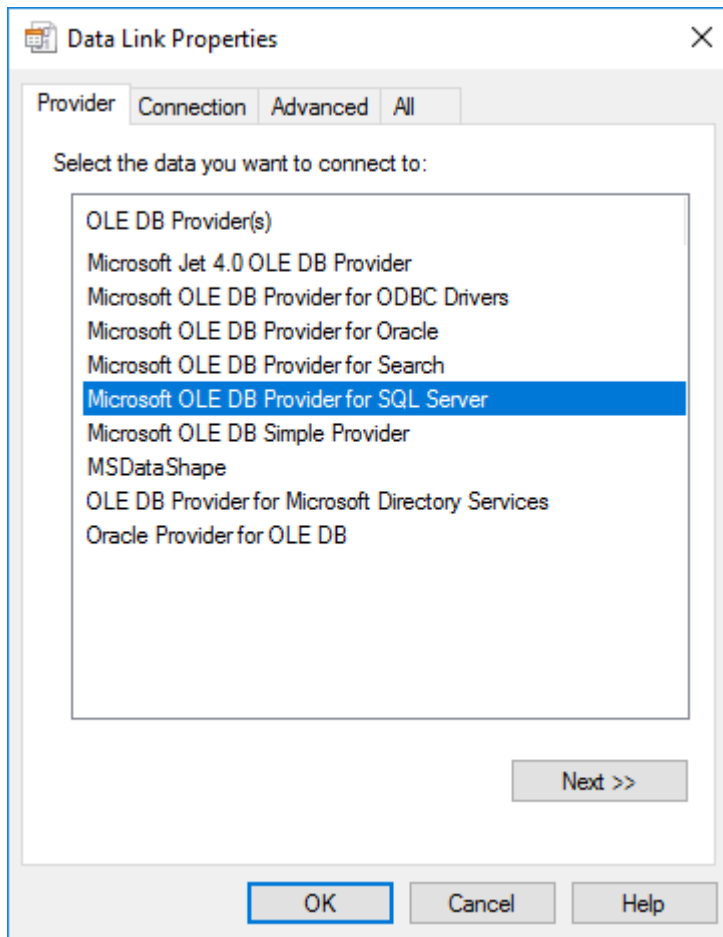
NOTE: SSIS Loads in Teradata can only be processed with a Windows Scheduler.

Creating the SSIS Connection String

- 1 Click on the **ellipsis** button to the right of the SSIS connection string field of the relevant connection:



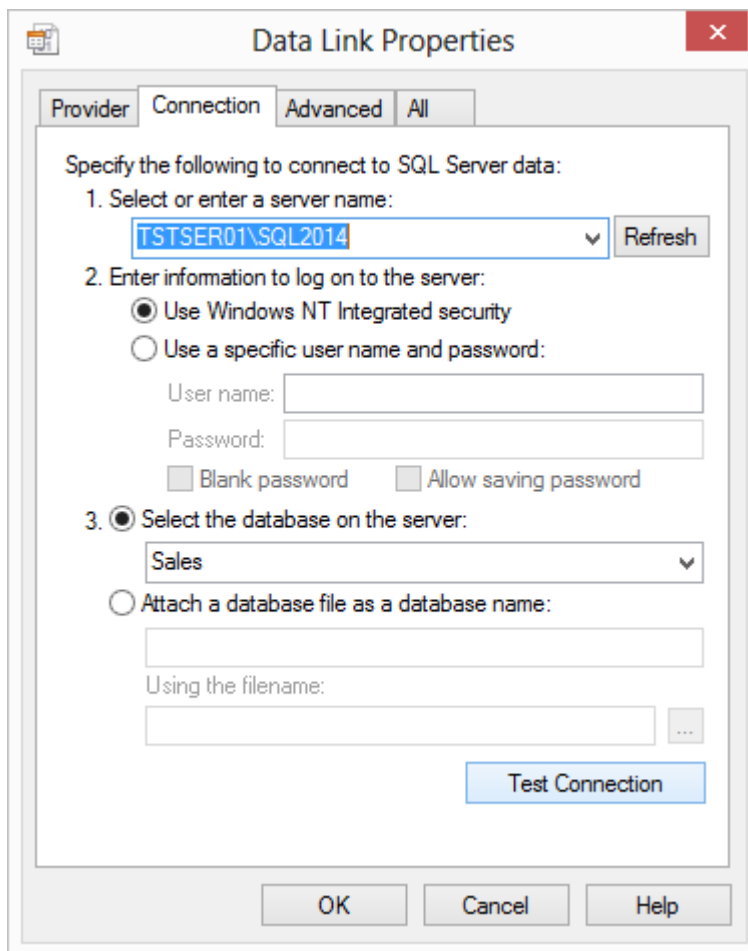
- 2 On the **Provider** tab, select the relevant **OLE DB Provider** and click **Next**.



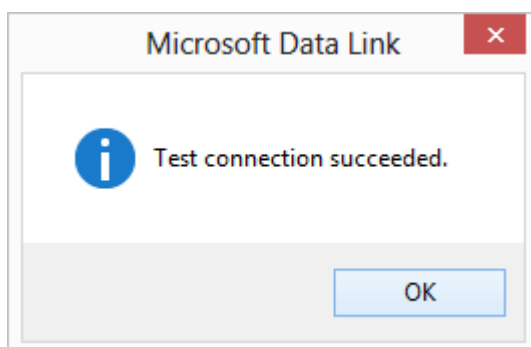
- 3 On the **Connection** tab, select the **server name**, enter the information to log on to the server and select the **database** on the server. Click **Test Connection**.

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 selected.

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.



- 4 Click OK.



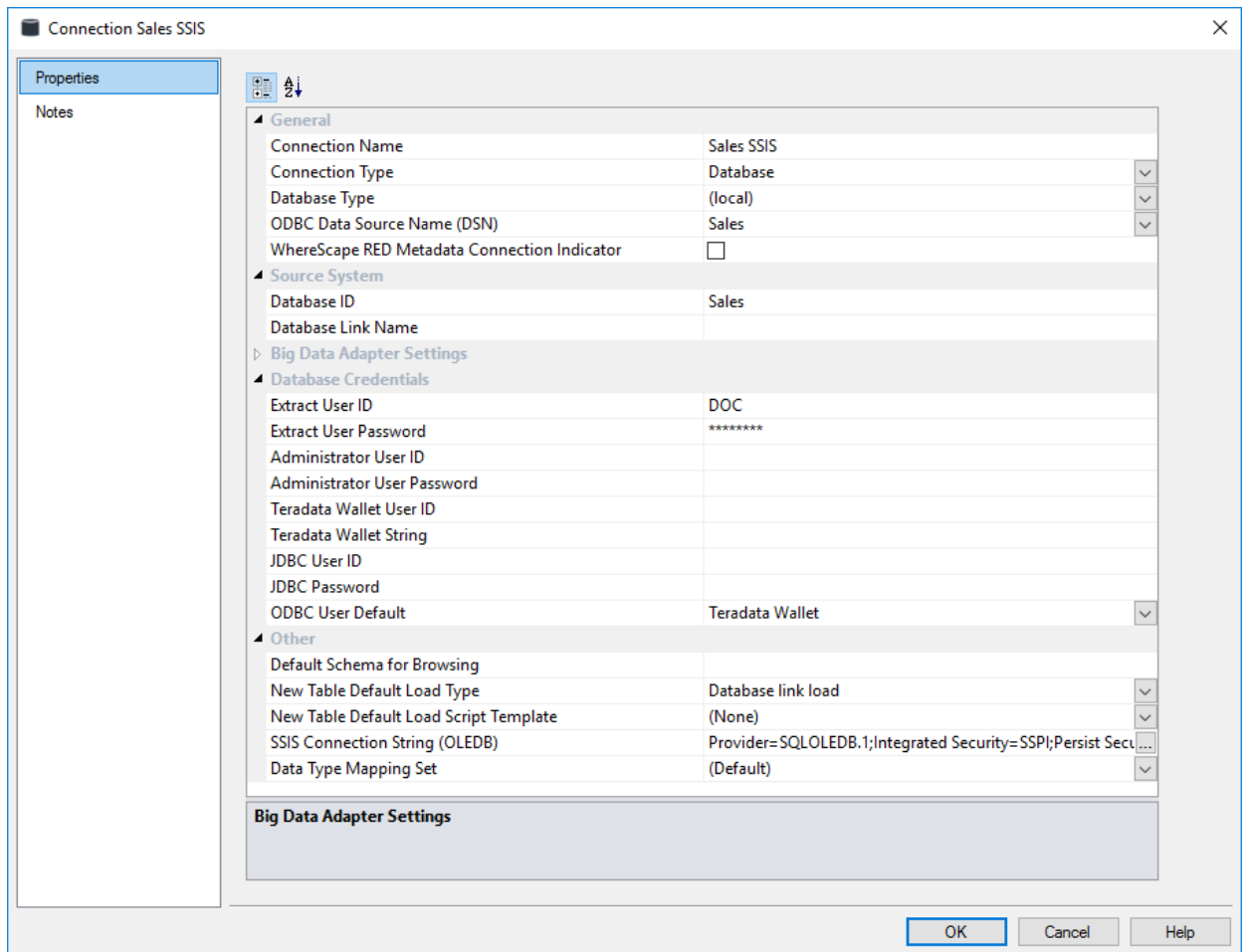
- 5 Click **OK** on the Data Link Properties dialog to save the connection string settings.

The screenshot shows the 'Data Link Properties' dialog box with the 'Connection' tab selected. The dialog is titled 'Data Link Properties' and has a close button (X) in the top right corner. Below the title bar are four tabs: 'Provider', 'Connection', 'Advanced', and 'All'. The 'Connection' tab is active, and the text 'Specify the following to connect to SQL Server data:' is displayed. The settings are as follows:

- 1. Select or enter a server name: A dropdown menu shows 'TSTSER01\SQL2014' with a 'Refresh' button to its right.
- 2. Enter information to log on to the server:
 - Use Windows NT Integrated security
 - Use a specific user name and password:
 - User name: [Empty text box]
 - Password: [Empty text box]
 - Blank password Allow saving password
- 3. Select the database on the server: A dropdown menu shows 'Sales'.
- Attach a database file as a database name:
 - Sales [Empty text box]
 - Using the filename: [Empty text box] ...

At the bottom of the dialog are three buttons: 'OK', 'Cancel', and 'Help'. A 'Test Connection' button is located at the bottom right of the main content area.

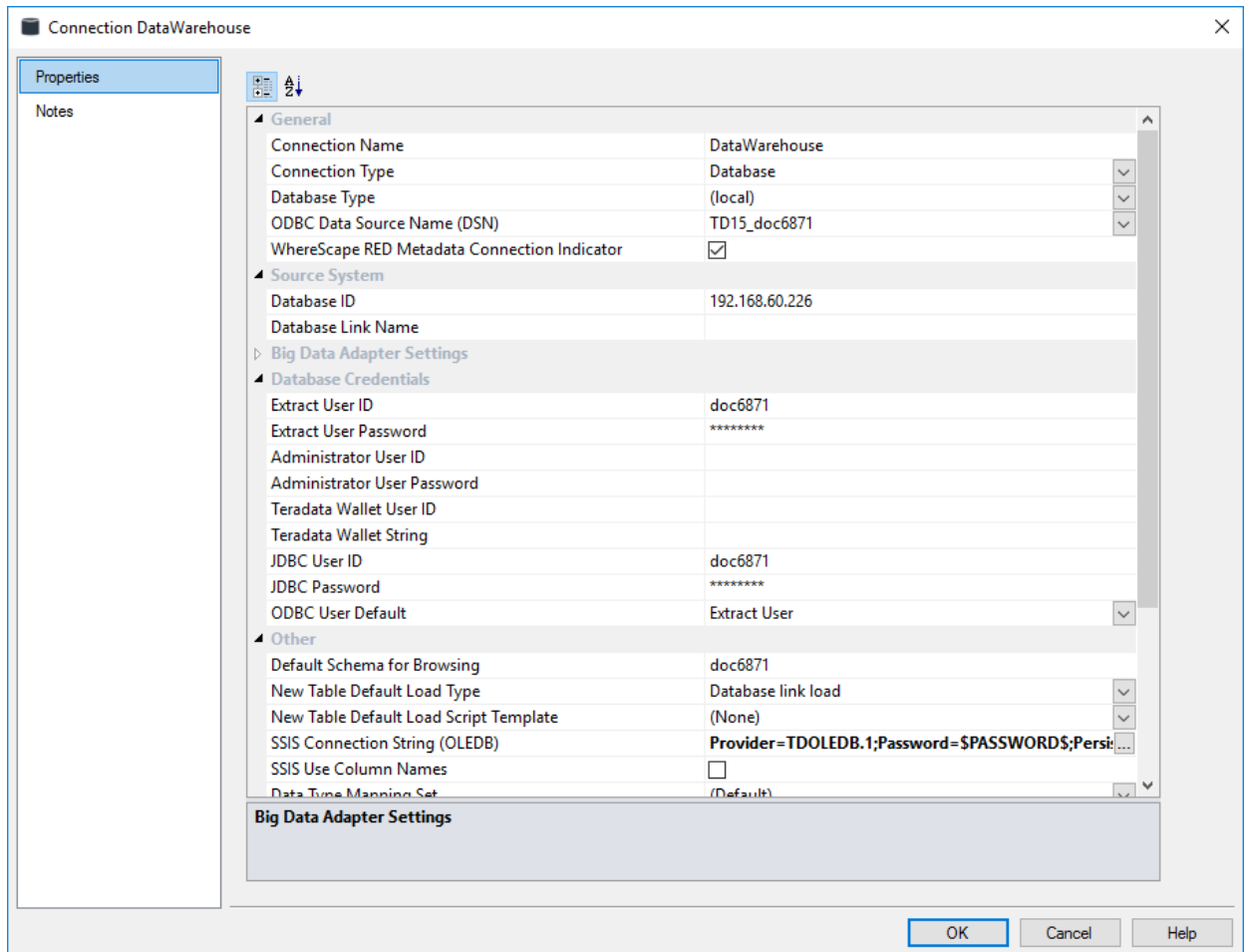
6 The SSIS connection string is displayed.



7 Click **OK** to save the connection.

- Right-click on **Sales SSIS** and select **Browse Connection**.
- Accept the defaults and click **OK**.

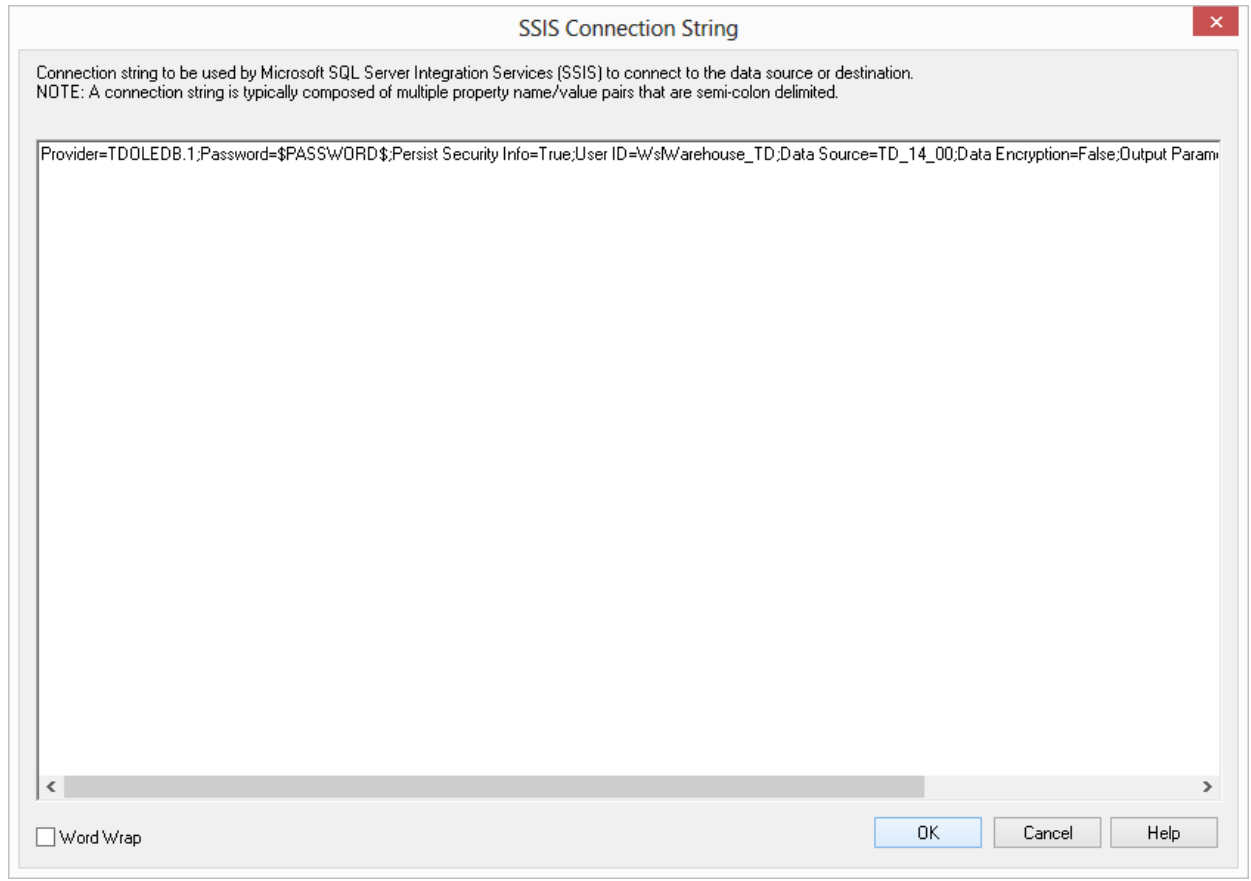
- 8 In SSIS terms, you have now defined your **Source** in **SSIS Connection Manager**. Using the same process, you need to add the SSIS Connection String to the data warehouse connection so you can specify your **Destination** connection:
- Double-click on the **DataWarehouse** connection in the object explorer to open up the Properties dialog.
 - Follow the process above to create the SSIS Connection String, this time selecting the **OLE DB Provider for Teradata**.
 - Click **OK** to save your connection.



Note1: If the connection string is already set, then the ellipsis button will open up 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 194) or **Loading Data from Flat Files using SSIS**).

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.

Load load_customer_ssis

Properties

Storage

Override Create DDL

Source

Notes

Load Table Name: load_customer_ssis

Unique Short Name: (maximum 22 characters) load_customer_ssis

Description:

Connection: Sales SSIS

Load Type: Integration Services load

Database Link:

Script Template: (None)

Script Name: (None)

Pre-Load Action: Truncate

Pre-Load SQL:

Post Load Procedure: (None)

Timestamps

Metadata Structure Changed: 2017-03-13 04:45:29.480000

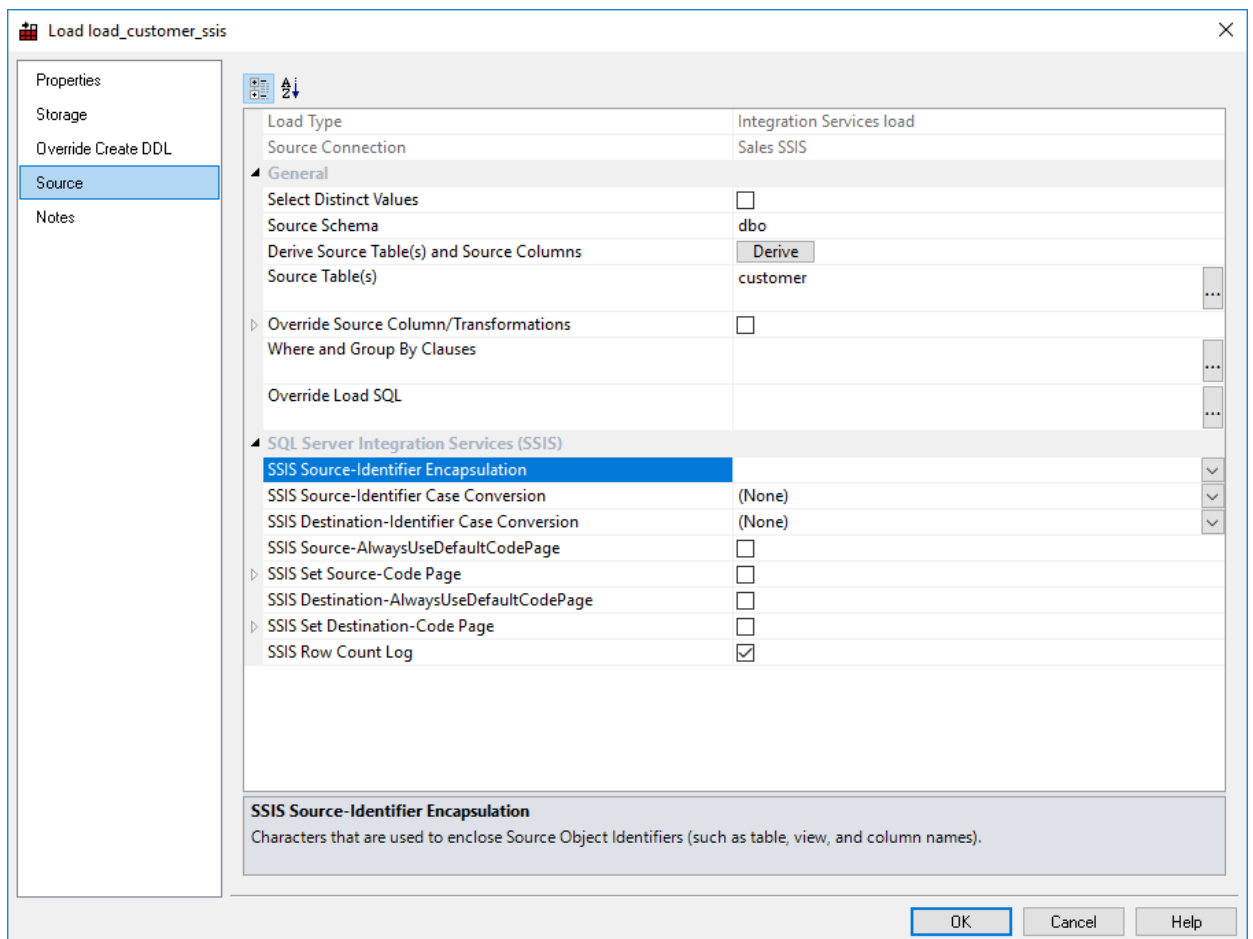
Database Created:

Database Altered:

OK Cancel Help

10 The configuration options available on an SSIS load are available on the **Source** tab of the load table's Properties. These options are:

- **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 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.



FLAT FILE LOADS

Flat file loads can be processed from either a **UNIX/Linux**, **Windows** 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 Flat Files using SSIS*.

The **drag and drop** process for flat files is as follows:

- 1 Browse to the directory and file via the appropriate *UNIX/Linux Connection*, *Windows Connection* (see "*Windows*" on page 148) or *Hadoop Connection*.
- 2 Double click 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 following dialog appears. Rename the file if necessary and click the **ADD** button.

Add a New Metadata Object

Define the Type and Name of the New Object.
Specific information for each object type is defined in subsequent screens.

Object Type: Load

Object Name: load_budget

Target Location: (local)

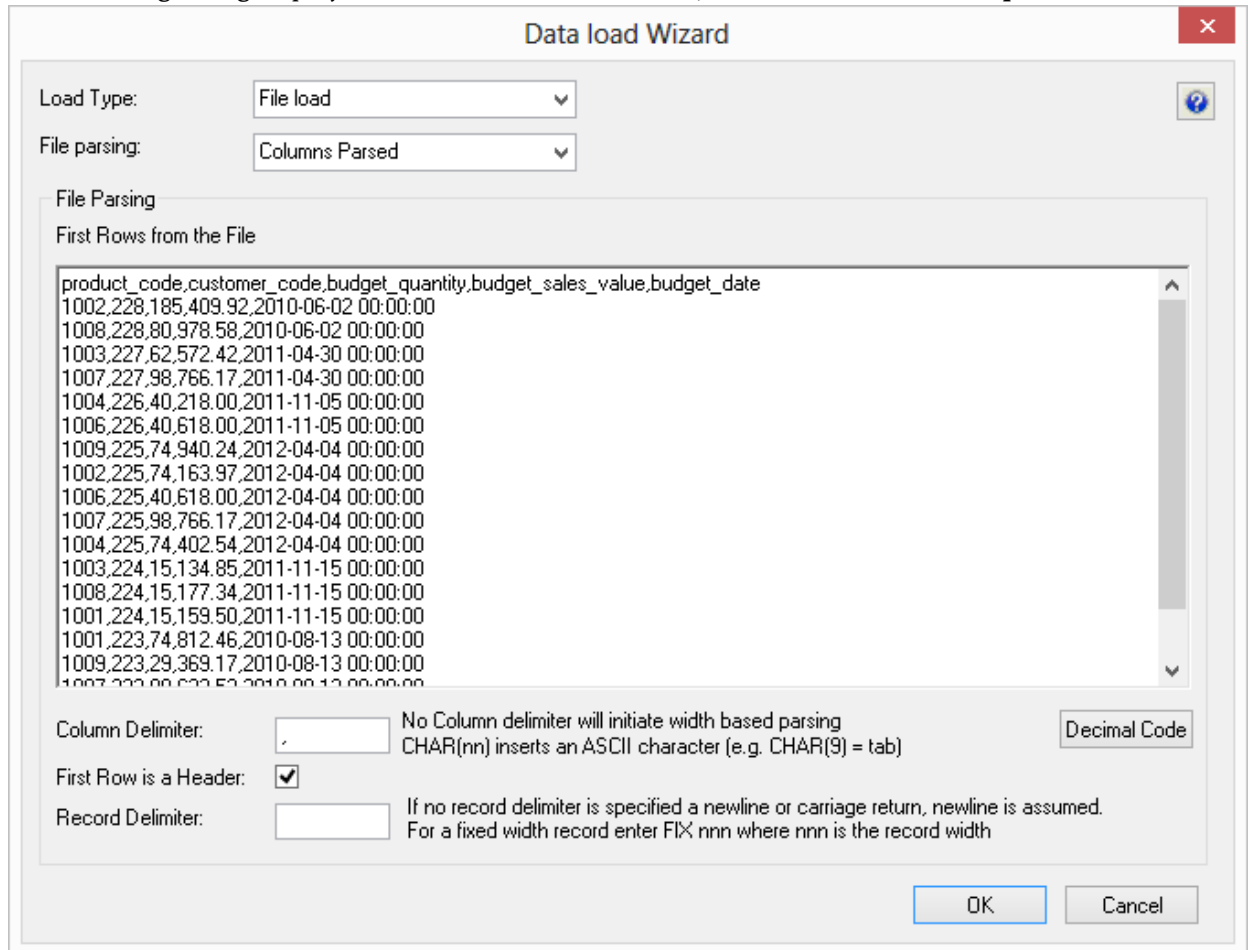
Data Type Mapping: Standard File to Teradata

Add meta data columns to table

ADD Cancel

NOTE: The option **Add meta data columns to table** is used for creating load tables that are used in creating Data Vault objects. If this option is selected, two DSS columns (**dss_record_source** and **dss_load_date**) are included in the meta data for the table and are populated by transformations. These two DSS columns could equally be applied to other load tables not used in a Data Vault system but are particularly important to comply with the Data Vault standards. Please refer to the *Data Vaults chapter* (see "*Data Vaults*" on page 402) for more details.

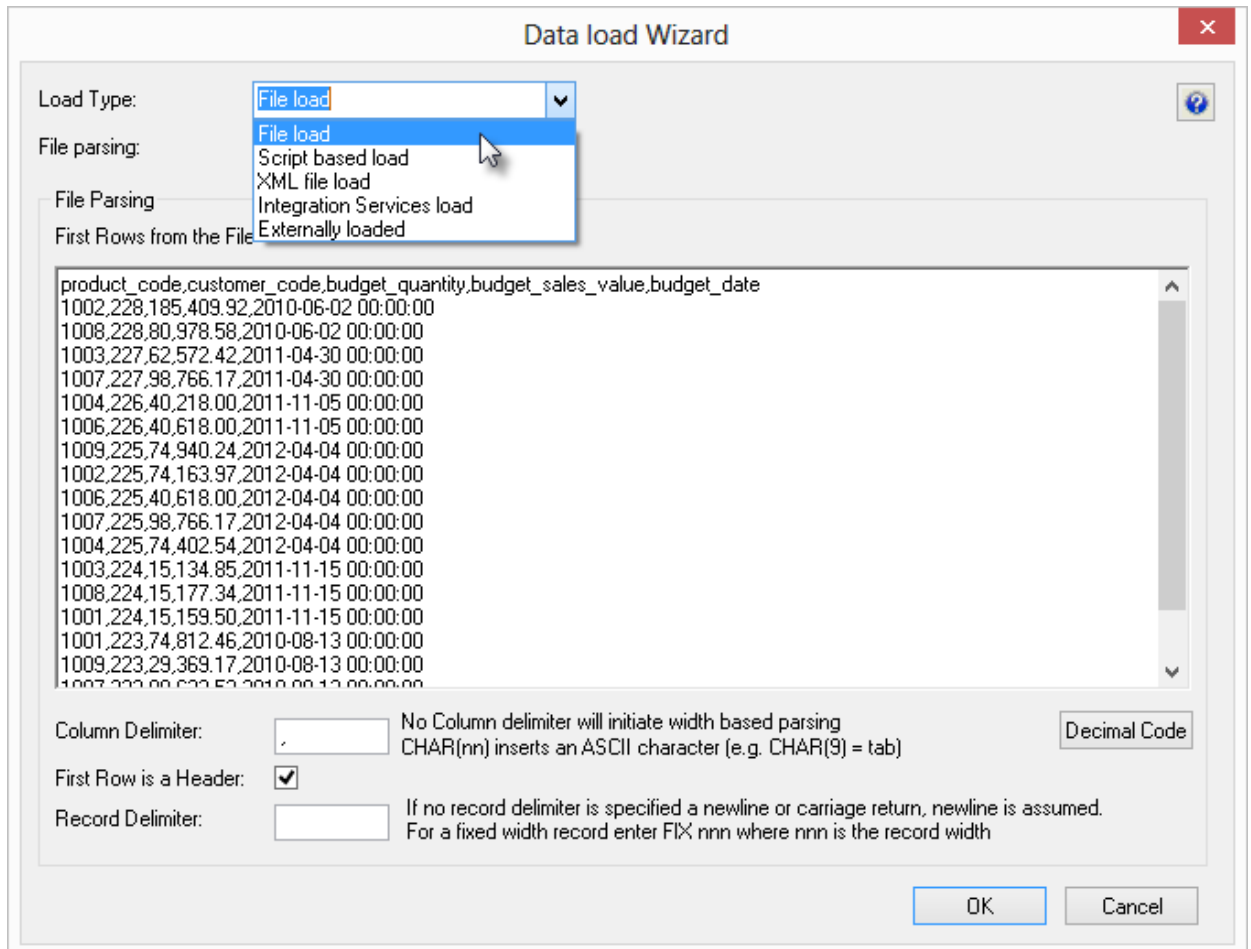
- 5 The following dialog displays for file loads from **Windows, UNIX/Linux** and **Hadoop** connections.



NOTE: First Row is a Header - To make changes in 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: 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.

- 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 perform the load. For further details about Scrip based loads see section **Script based loads**.
- 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 Flat Files 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, 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. 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: 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.

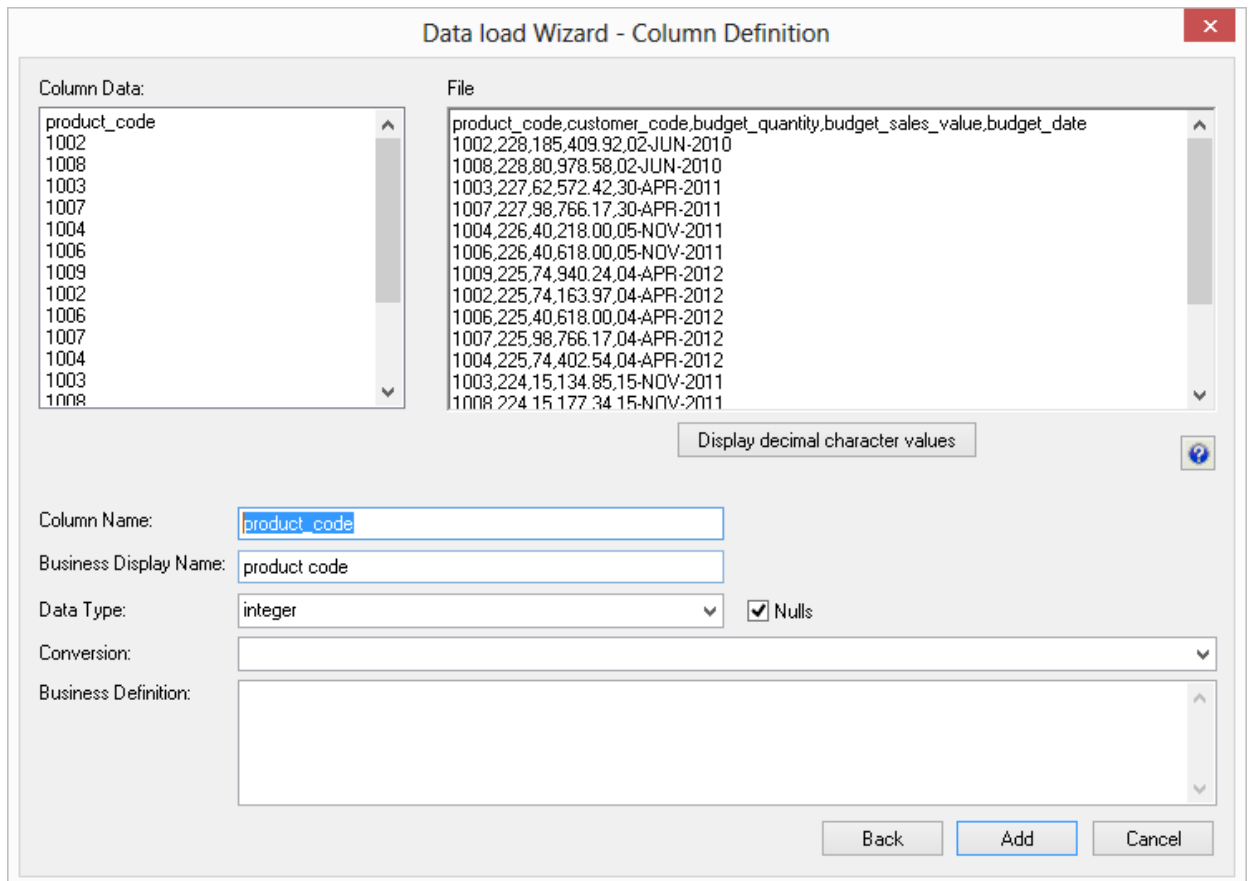
- 7 Once the screen above is completed, another screen will display to allow the breakdown of the source data into columns.

If **no delimiter is entered** then **width based parsing is assumed** and an addition width size field is prompted for.

If this is a **Fixed Width file** loaded via TPT, the source file format can be specified later in the **File Load Source Screen** of the Load table's Properties dialog.

- Use the **Back** button to revert to the previous column if an incorrect width or delimiter is entered.

This following screen is an example of the file parsing technique.



Conversion

During the parsing of the columns a date format conversion string can be used for **date data types only**.

Any other Teradata function can be used with Multi Load or TPT Update for **After Load Transformations only**. Refer to the Multi load and TPT manuals for syntax.

For example, if we are loading a column called 'product_name' we could use the following syntax to bring over only the first 30 characters.

```
substr(product_name, 1, 30)
```

A special variable **%FILE_NAME%** can be used in a File based script load. This will be substituted with the actual source file name.

For example, a transformation such as: '%FILE_NAME%' can be used to store the full file name and path in a database column.

For **After Load** transformations, the conversion string can also 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:

- Go into the **Properties** of the loaded table's relevant column(s) by double-clicking on the column(s) in the middle pane.
- Click the **Transformation** tab.
- Select the **After load** option in the **Transformation Stage** drop-down list.
- **Recreate** the load table.

LOADING DATA FROM FLAT FILES USING SSIS

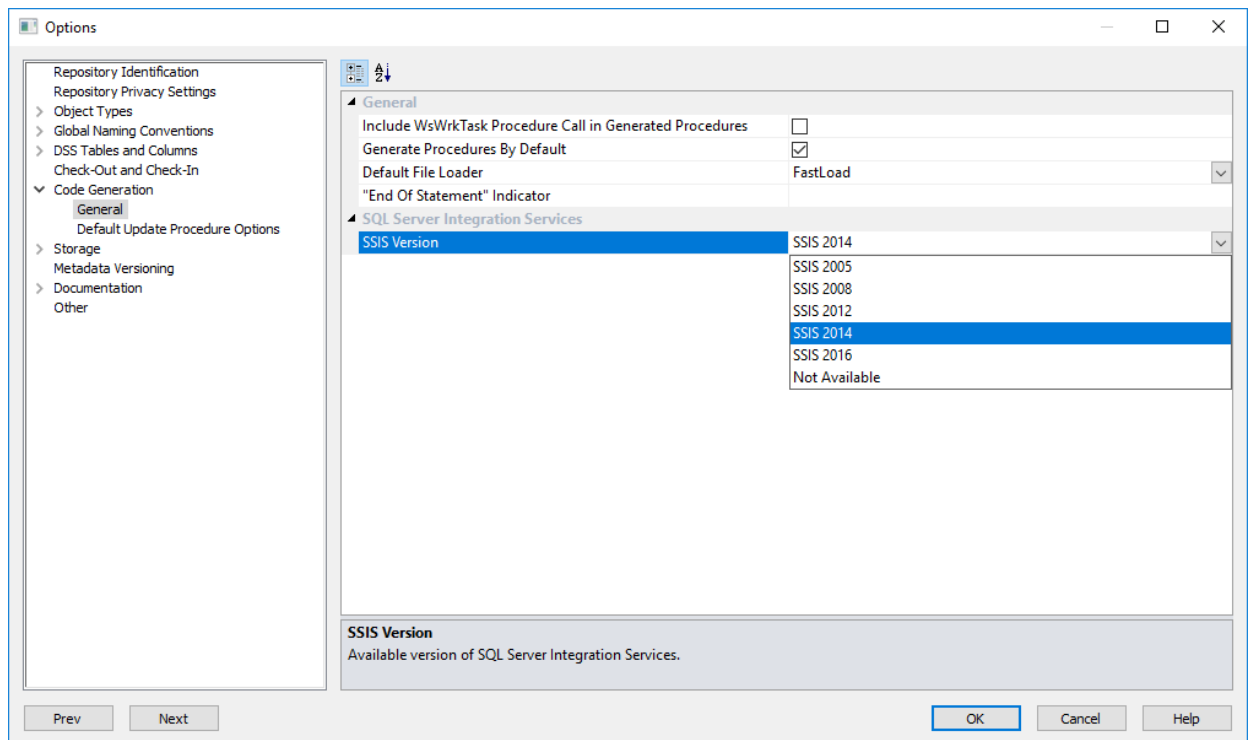
Flat files can be loaded into RED from a **Windows** connection using **SQL Server Integration Services (SSIS)**.

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.

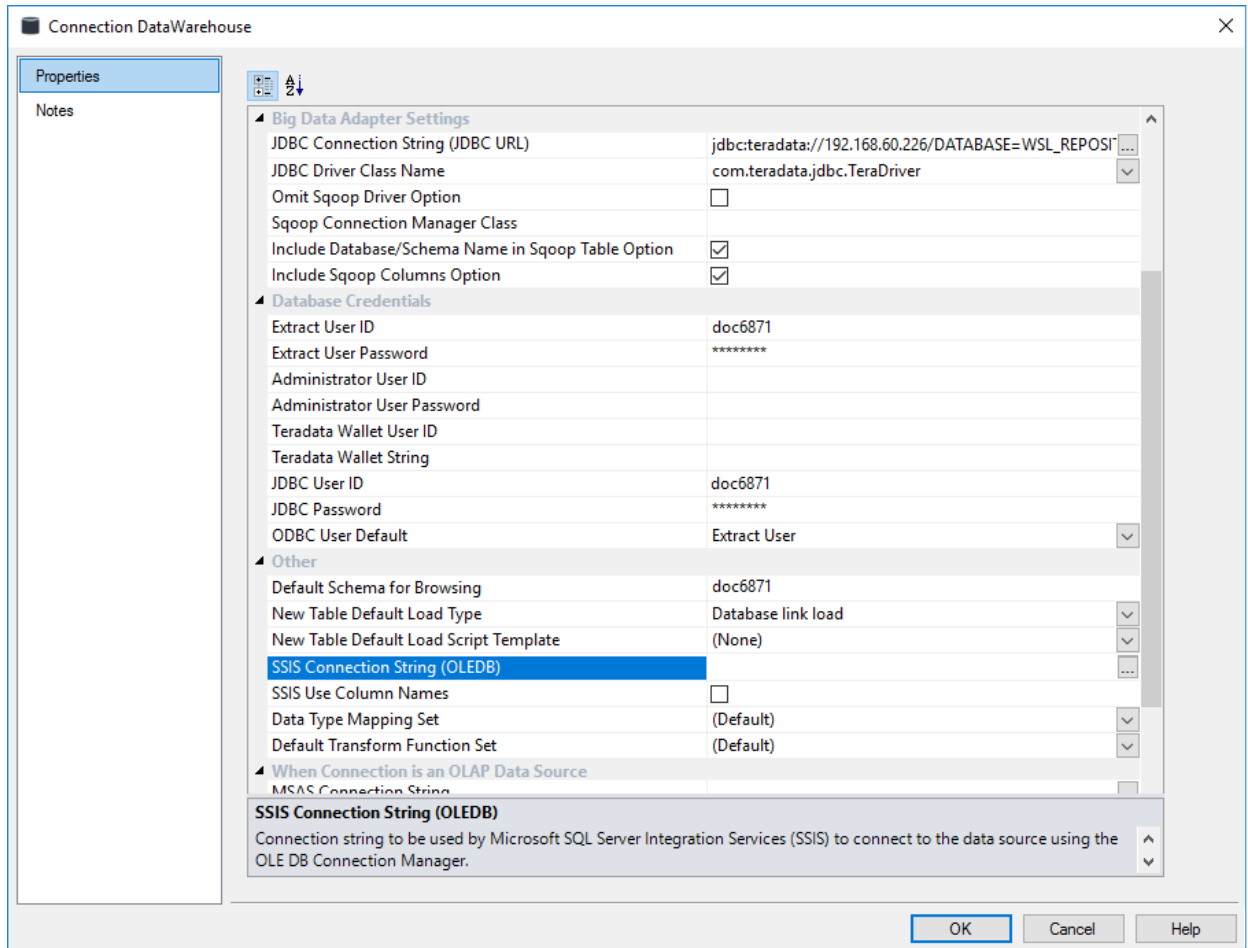
To load files via SSIS, the SSIS connection string must be defined in the DataWarehouse connection.

NOTE: SSIS Loads for Teradata only work with a Windows Scheduler.

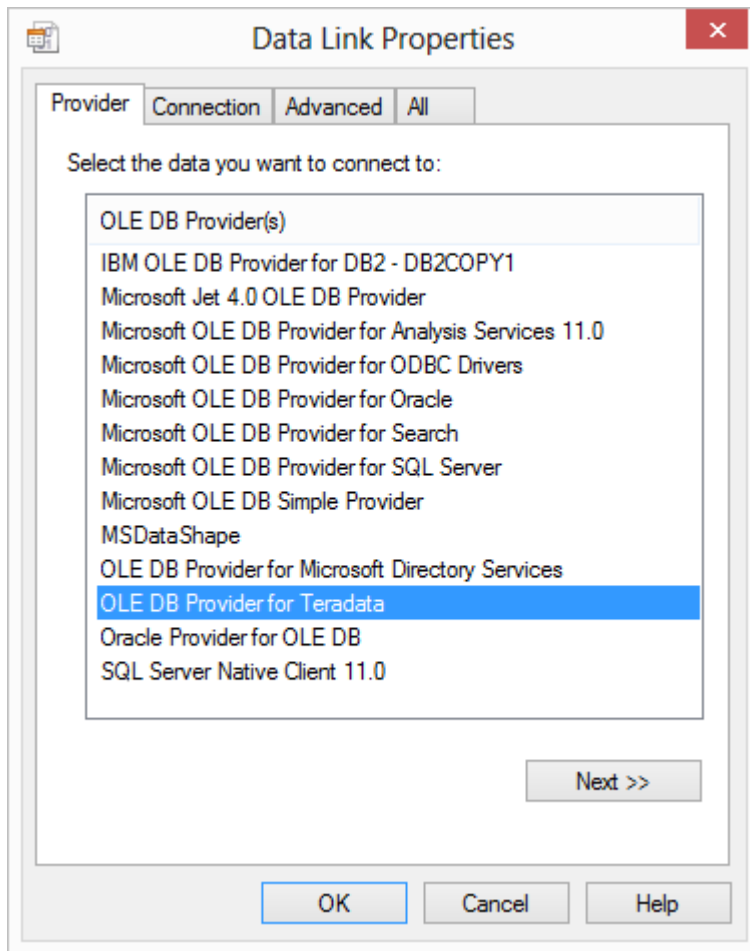
To use SSIS loading, ensure that SSIS loads are enabled and that the SSIS version is set correctly in **Tools>Options>Code Generation>General**.



- 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 up the Properties dialog.
 - Click on the ellipsis button to open the wizard and define the SSIS connection string.

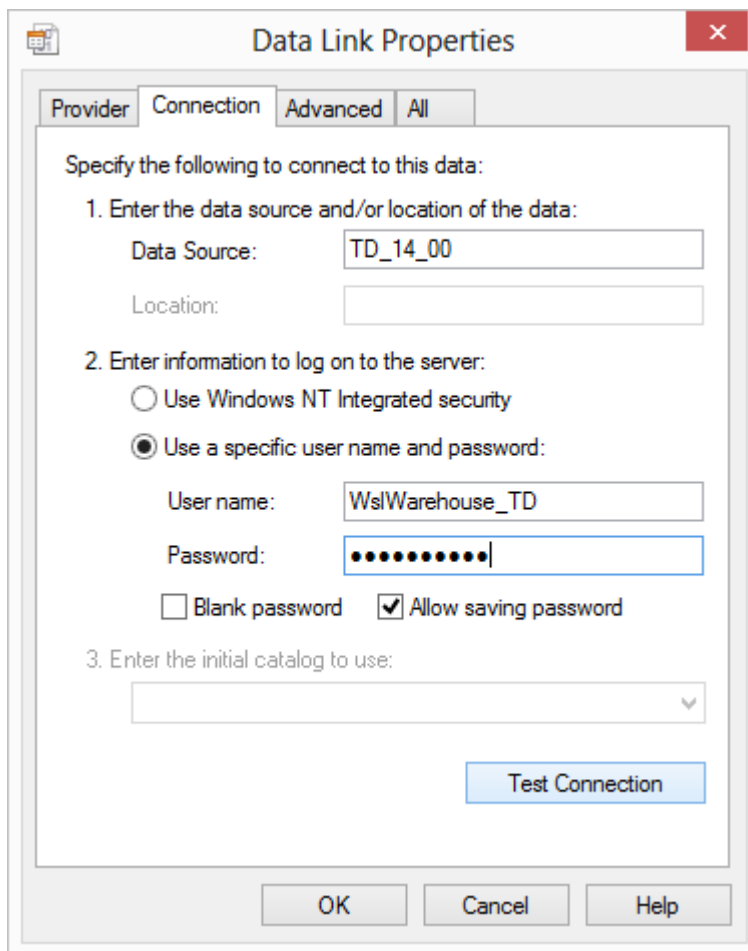


- 2 On the **Provider** tab, select the **OLE DB Provider for Teradata** and click **Next**.

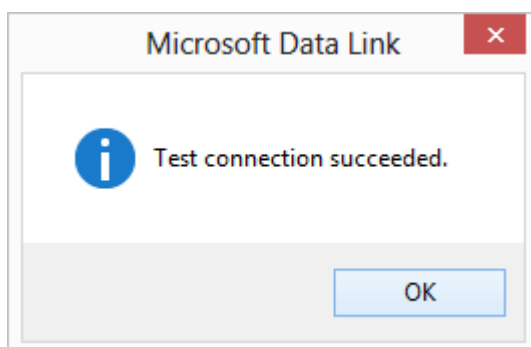


- 3 On the **Connection** tab, enter the **destination data source name**, enter the **information to log on to the server** and select the **Allow saving password** option. Click **Test Connection**.

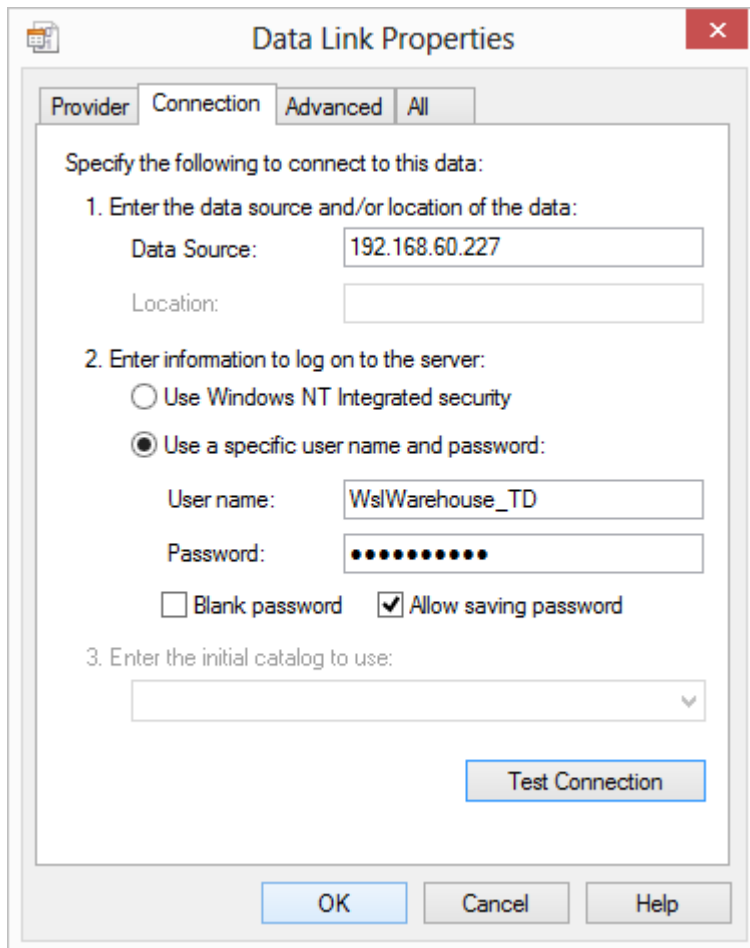
NOTE: When using a specific **user name** and **password** to connect to the server instead of using Windows integrated security, the **Allow saving password** check-box must be selected. It is also recommended that the password on the SSIS connection string field that is displayed in the connection properties is replaced with the **\$PASSWORD\$** token that is substituted at runtime.



4 Click **OK**.



- 5 Click **OK** on the Data Link Properties dialog to save the settings.

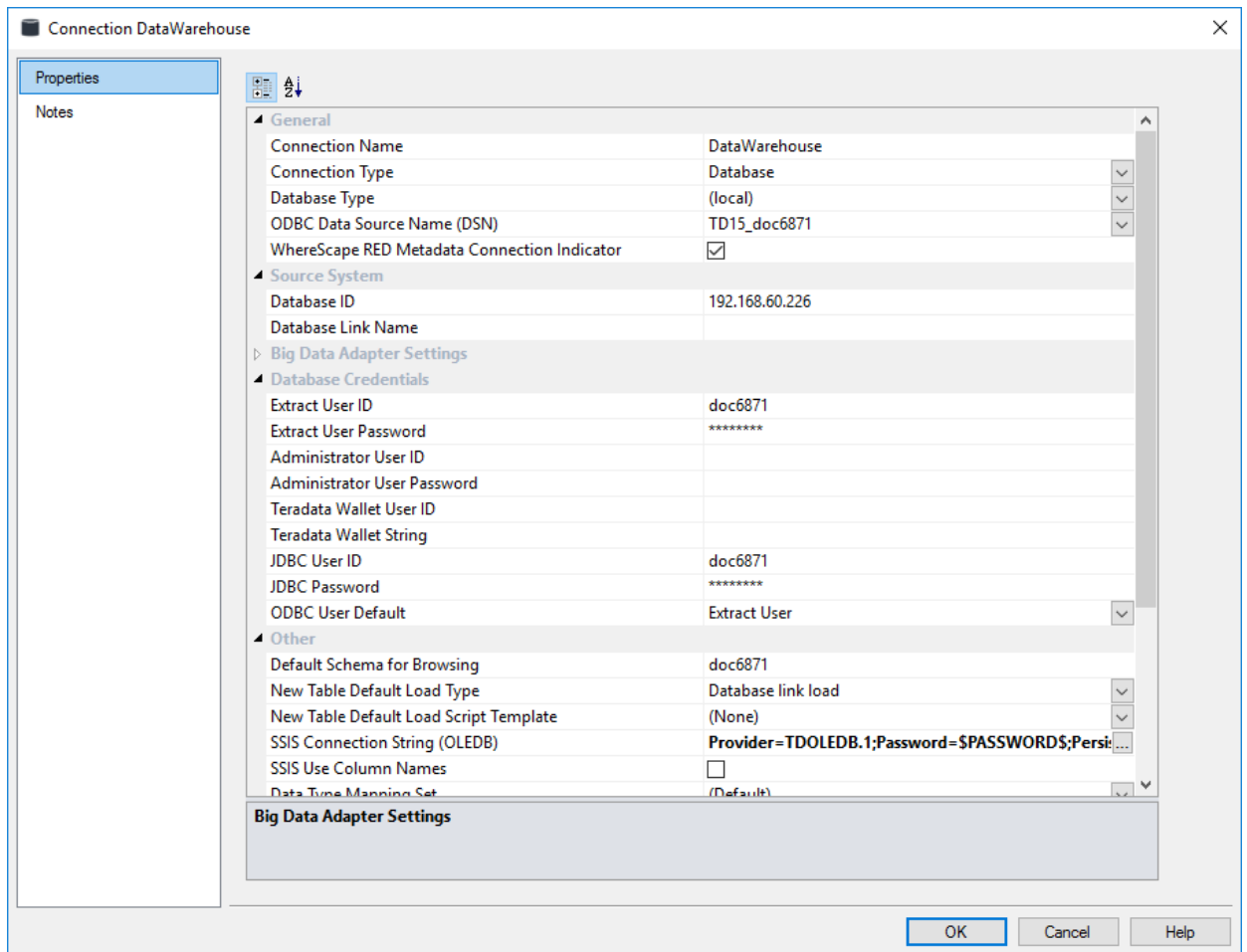


The screenshot shows the 'Data Link Properties' dialog box with the 'Connection' tab selected. The dialog is titled 'Data Link Properties' and has a close button (X) in the top right corner. It contains three tabs: 'Provider', 'Connection', 'Advanced', and 'All'. The 'Connection' tab is active, and the text 'Specify the following to connect to this data:' is displayed. Below this, there are three numbered steps:

1. Enter the data source and/or location of the data:
 - Data Source: 192.168.60.227
 - Location: (empty text box)
2. Enter information to log on to the server:
 - Use Windows NT Integrated security
 - Use a specific user name and password:
 - User name: WslWarehouse_TD
 - Password: (masked with 10 dots)
 - Blank password Allow saving password
3. Enter the initial catalog to use:
 - (empty dropdown menu)

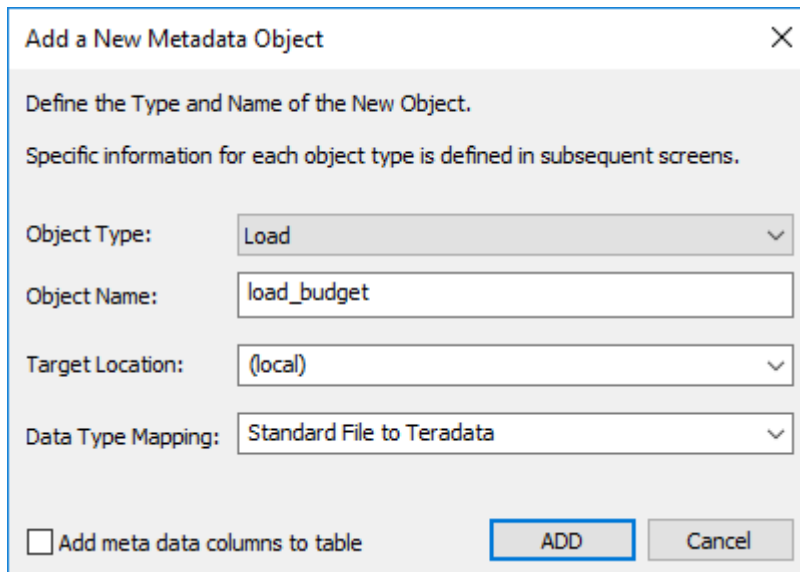
At the bottom of the dialog, there are three buttons: 'Test Connection', 'OK', 'Cancel', and 'Help'.

6 Click **OK** to save your connection.



NOTE: It is recommended that the password on the SSIS connection string field that is displayed in the connection properties is replaced with the **\$PASSWORDS** token that is substituted at runtime.

- 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.



Add a New Metadata Object [X]

Define the Type and Name of the New Object.
Specific information for each object type is defined in subsequent screens.

Object Type:

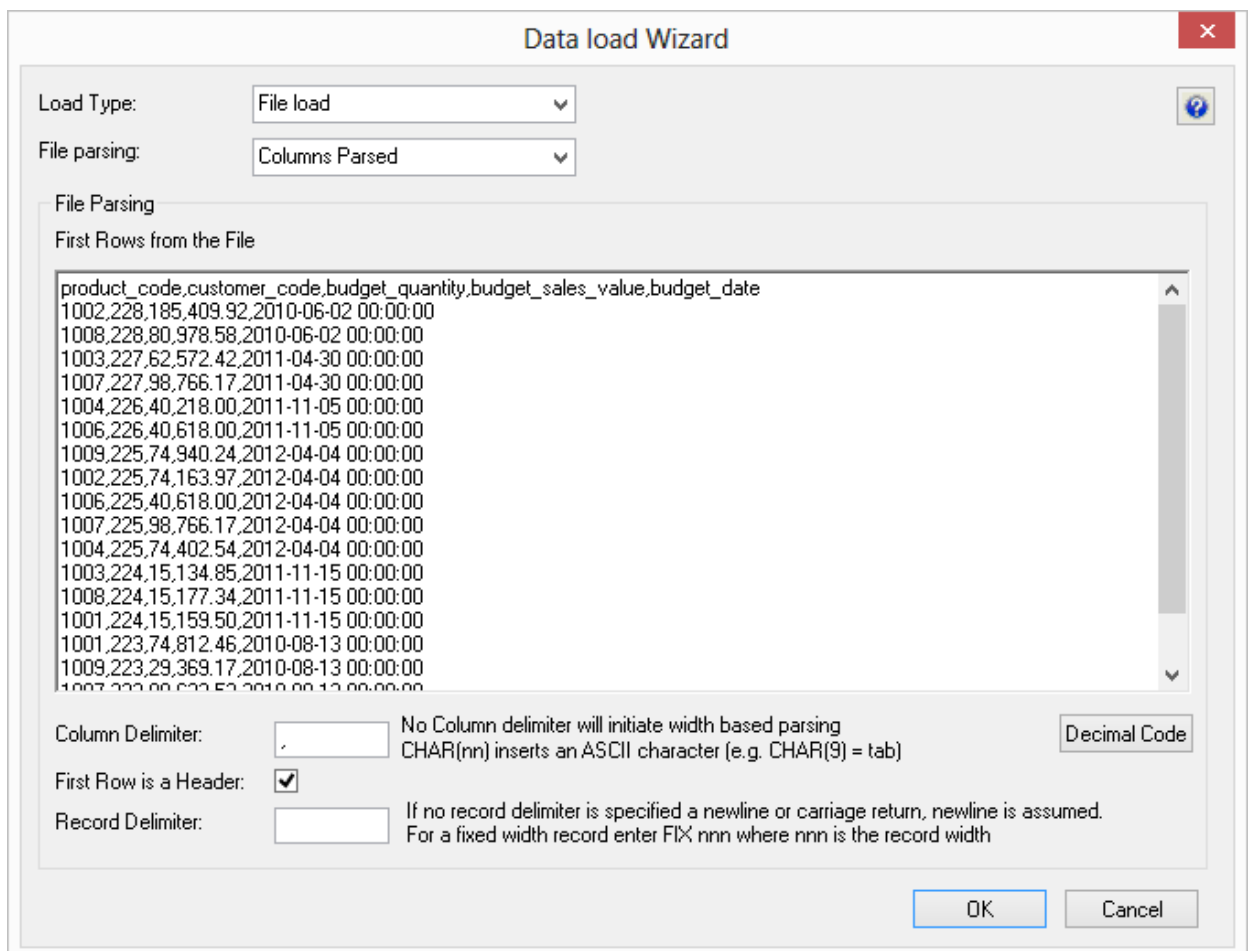
Object Name:

Target Location:

Data Type Mapping:

Add meta data columns to table

- 9 Click the **ADD** button. The following dialog appears.



Data load Wizard [X]

Load Type:

File parsing:

File Parsing

First Rows from the File

```
product_code,customer_code,budget_quantity,budget_sales_value,budget_date
1002,228,185,409.92,2010-06-02 00:00:00
1008,228,80,978.58,2010-06-02 00:00:00
1003,227,62,572.42,2011-04-30 00:00:00
1007,227,98,766.17,2011-04-30 00:00:00
1004,226,40,218.00,2011-11-05 00:00:00
1006,226,40,618.00,2011-11-05 00:00:00
1009,225,74,940.24,2012-04-04 00:00:00
1002,225,74,163.97,2012-04-04 00:00:00
1006,225,40,618.00,2012-04-04 00:00:00
1007,225,98,766.17,2012-04-04 00:00:00
1004,225,74,402.54,2012-04-04 00:00:00
1003,224,15,134.85,2011-11-15 00:00:00
1008,224,15,177.34,2011-11-15 00:00:00
1001,224,15,159.50,2011-11-15 00:00:00
1001,223,74,812.46,2010-08-13 00:00:00
1009,223,29,369.17,2010-08-13 00:00:00
1007,223,00,000.00,2010-08-13 00:00:00
```

Column Delimiter: No Column delimiter will initiate width based parsing
CHAR(nn) inserts an ASCII character (e.g. CHAR(9) = tab)

First Row is a Header:

Record Delimiter: If no record delimiter is specified a newline or carriage return, newline is assumed.
For a fixed width record enter FIX nnn where nnn is the record width

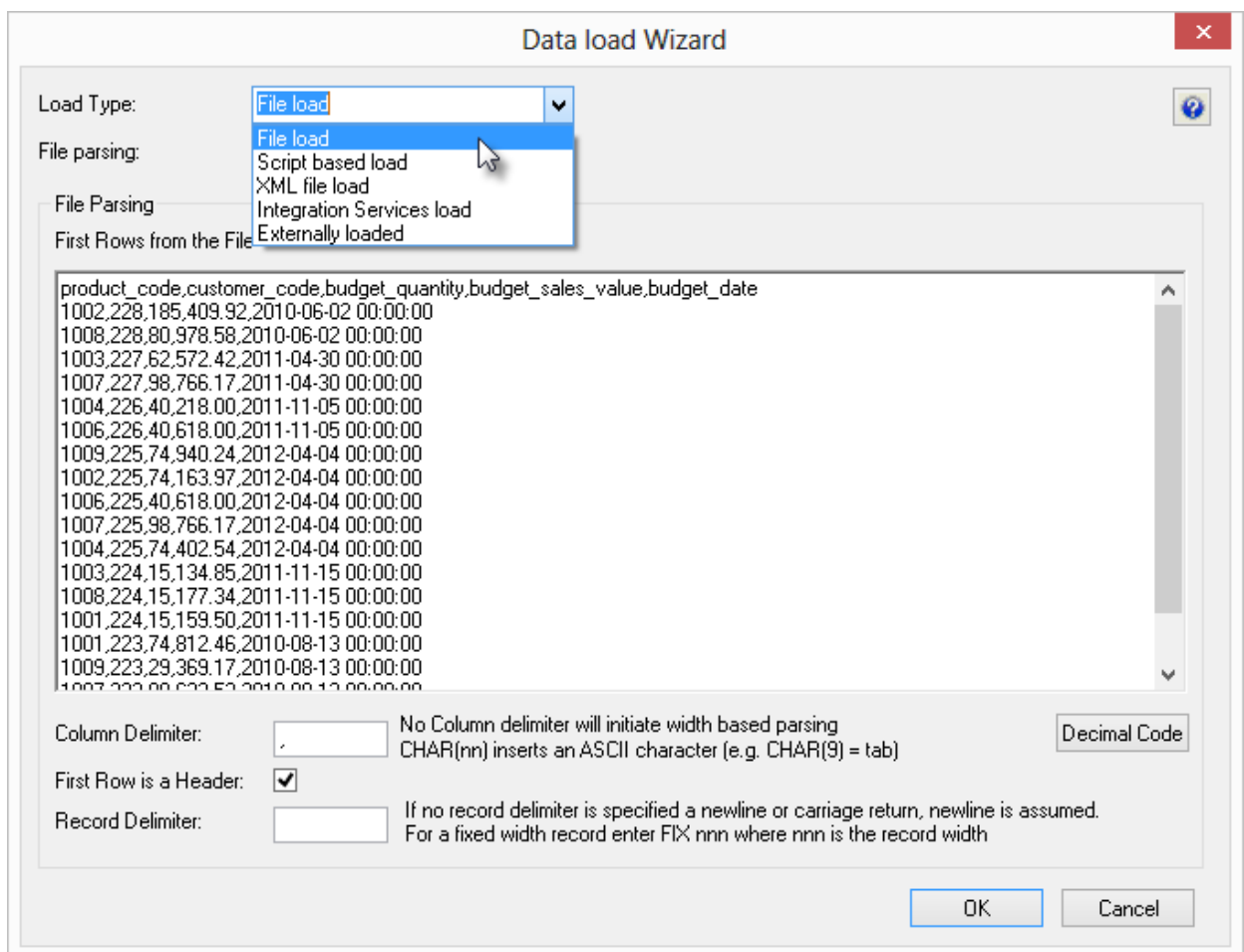
10 There are two options on this screen (buttons at right).

- The first option results in a **File based load table** where the bulk of the load management is handled by the scheduler.
- If you select the second option, RED will generate a windows host script and the load table will be a **Script based load**. This host script is executed by the scheduler to effect the load.

Columns parsed

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.

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 shows the initial file parsed screen.



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.

11 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.

Data load Wizard - Column Definition

| Column Data: | File |
|--------------|---|
| product_code | product_code,customer_code,budget_quantity,budget_sales_value,budget_date |
| 1002 | 1002,228,185,409.92,02-JUN-2010 |
| 1008 | 1008,228,80,978.58,02-JUN-2010 |
| 1003 | 1003,227,62,572.42,30-APR-2011 |
| 1007 | 1007,227,98,766.17,30-APR-2011 |
| 1004 | 1004,226,40,218.00,05-NOV-2011 |
| 1006 | 1006,226,40,618.00,05-NOV-2011 |
| 1009 | 1009,225,74,940.24,04-APR-2012 |
| 1002 | 1002,225,74,163.97,04-APR-2012 |
| 1006 | 1006,225,40,618.00,04-APR-2012 |
| 1007 | 1007,225,98,766.17,04-APR-2012 |
| 1004 | 1004,225,74,402.54,04-APR-2012 |
| 1003 | 1003,224,15,134.85,15-NOV-2011 |
| 1008 | 1008,224,15,177.34,15-NOV-2011 |

Display decimal character values

Column Name:

Business Display Name:

Data Type: Nulls

Conversion:

Business Definition:

Back Add Cancel

- 12** 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.

Load load_budget_ssis

Properties

Storage

Override Create DDL

Source

Notes

Load Table Name: load_budget_ssis

Unique Short Name: (maximum 22 characters) load_budget_ssis

Description:

Connection: Windows

Load Type: Integration Services load

Database Link:

Script Template: (None)

Script Name: (None)

Pre-Load Action: Truncate

Pre-Load SQL:

Post Load Procedure: (None)

Timestamps

Metadata Structure Changed: 2015-09-17 04:11:51.850000

Database Created: 2015-09-17 05:00:25.930000

Database Altered: 2015-09-17 05:00:25.930000

OK Cancel Help

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 6.8.4.0 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 a better load option instead.

- 13** Click **Yes** to Create and Load the table.

FLAT FILE LOAD - SOURCE SCREEN

The fields for the **Flat File Source Screen** are described below:



TIP: If the file has been *dragged and dropped* (see "**Flat File Loads**" on page 235) into the load table (middle pane) then some of the fields on this tab are automatically populated.

| Field | Value |
|--|---|
| Load Type | Script based load |
| Source Connection | Windows |
| Source File Details | |
| Source Directory | C:\Program Files (x86)\WhereScape\6.8.5.3\Tutorial\ |
| Source File Name | budget.txt |
| Source File Field Delimiter | , |
| Source File Record Terminator | |
| Source File has Field Headings/Labels | <input checked="" type="checkbox"/> |
| Trigger File Details | |
| Load Configuration | |
| Check existence of Source File | <input checked="" type="checkbox"/> |
| Wait for Source File or Trigger File | <input type="checkbox"/> |
| File Load Routine | FastLoad |
| MultiLoad/FastLoad Options | ... |
| Character Set | ASCII |
| Script Load supports File Name Wildcards | <input type="checkbox"/> |
| Only Load Latest File | <input type="checkbox"/> |
| Validate using Header/Trailer Data | <input type="checkbox"/> |
| Populate Audit Log with File Actions | <input checked="" type="checkbox"/> |
| Populate Detail Log when Successful | <input checked="" type="checkbox"/> |
| Insert into dss_load_file Table | <input type="checkbox"/> |
| Archived File Details | |

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.

OK Cancel Help

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, see *Templates* (on page 678).

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 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 Fixed Width File Format

Source File format to be set for TPT loads of Fixed width files. This option is only available for TPT Flat File or Script based loads from Windows and UNIX/Linux connections.

The default option for all Fixed width files is **Text**. To change the file format, select from **Text**, **Binary**, **Formatted** or **Unformatted**.

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'.

Load Configuration

Configuration details to control the load processing.

Check existence of Source File

This field controls whether the load process checks if the file exists before performing the load. This check can only be disabled when doing script-based TPT Loads from Windows and UNIX/Linux connections, which can improve the use of built-in TPT functionality to wait for the arrival of the file.

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. This is disabled by default but 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. Expand to set the **Wait Limit** and the **Action when Wait Limit Reached**.

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.

File Load Routine

File Loader utility/mechanism used to load the Source File. Select between **MultiLoad**, **FastLoad**, **Load TPT**, **Update TPT** or **Stream TPT**. If the 'No Load' option is specified the file does not load. This can be useful for a Script Load that has "File Actions".

MultiLoad/FastLoad Options

If **MultiLoad** or **FastLoad** is selected, optional comma-delimited list of TPT Operator Attributes. e.g. INTEGER DataBlockSize =2048.

Character Set

Teradata-compliant Character Set Name to use when loading such as ASCII, UTF8, UTF16.

TPT HadoopHost

Field that identifies the Hadoop host to the TPT load routine. This is set to the **TPT HadoopHost** property of the Hadoop connection if specified there. If this not specified in the connection, the field will display the UNIX/Linux Host name property of the Hadoop Connection.

TPT Job Name

Job name for TPT. If this is not set it will default to Load Name.

TPT Build Command Options

Additional options included as part of the TBuild call.

TPT LogView Command Options

Additional options included as part of the TLogView call. For example, adding "-f '*'" can be added to prove greater diagnostics.

TPT Shared Memory Size

Shared memory size can be specified in bytes, kilobytes or megabytes. Examples include **2091752**, **2048K**, **2M**.

TPT Load Routine Minimum Sessions

Optional minimum number of TPT sessions [1-99]. The default is one instance.

Set Load Routine Maximum Sessions

Enables the "**Load Routine Maximum sessions**" property.

Load Read Instances

Optional number of TPT read instances [1-99]. The default is one instance.

Load Write Instances

Optional number of TPT write instances [1-99]. The default is one instance.

Only Load Latest File

Controls whether only the latest file (or all matching files) is loaded when the "Source File Name" includes a wildcard and multiple matching files are available.

Script Load supports File Name Wildcards

This option is only available for **Update TPT** or **Stream TPT** load routines.

It controls whether the RED generated script supports loading multiple files based on a file name wildcard. When enabled and the "Source File name" contains a wildcard, the RED generated script will loop to load each matching file while preserving the contents of the load table as each file is loaded. In addition the "Archived Source Path" and/or "Archived Source File Name" properties must be specified to allow each successfully loaded file to be archived before loading a subsequent file.

When doing **Hadoop Native TPT loads** where the file load routine is **Update TPT**, users can load multiple files from a Hadoop connection by adding * to the **Source File Name**. e. g.

hadoop_customer.csv*

Fail when incomplete load

Controls whether the load reports failure when ALL the rows in the file are not loaded. The specified exit status impacts any remaining tasks in the currently running job. When fail is specified, The WhereScape RED Scheduler will stop/fail the job and hold any remaining tasks when the load fails. In contrast when fail is disabled, the scheduler will continue to run any dependent tasks in the running job.

Validate using Header/Trailer Data

Controls whether a RED-generated Post-Load procedure validates the integrity of the Source File and the load processing using control totals &/or counts ("check sums") from the FIRST and/of LAST line(s) of the Source File. When enabled, any subsequently RED-generated Post-Load procedure for this table will use the HEADER and/or TRAILER rows as a source of validation information. Alternatively, A Trigger File can be used for the same validation purpose.

Populate Audit Log with File Actions

Controls whether details of any "File Actions" processing are recorded in the WhereScape RED Audit Log.

Populate Detail Log when Successful

Controls whether additional details from the Teradata "File Load Routine" are recorded in the WhereScape RED Error/Detail Log when the load processing is successful.

Insert into dss_load_file Table

Populate the metadata table dss_load_file.

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 together 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 together using one set of enclosing \$ such as \$YYYYMMDD\$.

SQL Server Integration Services (SSIS)

SQL Server Integration Services (SSIS) Attributes.

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 Set Source-Code Page

Enables the SSIS source code page property.

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 Row Count Log

During an SSIS Load include Row Count logging.

SCRIPT BASED LOADS

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 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.

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 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.

The screenshot shows the configuration window for a 'Load Table load_budget_script'. The window has a sidebar on the left with tabs for Properties, Storage, Override Create DDL, Source, and Notes. The main area contains the following fields and controls:

- Load Table Name: load_budget_script
- Unique Short Name (maximum 22 characters): load_budget_script
- Description: (empty text area)
- Connection: Windows (dropdown)
- Load Type: Script based load (dropdown)
- Database Link: (empty text field)
- Script Template: (None) (dropdown)
- Script Name: script_load_budget_script (dropdown) with 'Edit' and 'Rebuild' buttons
- Pre-Load Action: Truncate (dropdown)
- Pre-Load SQL: (empty text area)
- Post Load Procedure: (None) (dropdown)
- Timestamps section with three sub-fields:
 - Metadata Structure Changed: 2016-09-16 16:18:05.890
 - Database Created: (empty text field)
 - Database Altered: (empty text field)

At the bottom right, there are 'OK', 'Cancel', and 'Help' buttons.

Note: Script-based loads on Windows supports both DOS Batch and *PowerShell scripts* (see "24.11.1.1 Windows PowerShell Scripts" on page 657).

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. For more details about XML File load Properties and Source screen fields, please see **Flat File Loads** and **File Load - Source Screen**.

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.

For example:

```
<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 | Teradata |
|----------------------------------|--------------|
| string with length | char() |
| string with maxlength | varchar() |
| integer | integer |
| decimal with precision and scale | numeric(x,y) |
| dateTime (ISO8601) | timestamp |
| i2 | integer |
| i4 | integer |
| r4 | varchar(40) |
| r8 | varchar(40) |
| float | varchar(40) |

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; e.g.:

```
<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 varchar(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 column of numeric(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 column of numeric(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 Teradata as:

```
CAST('20031003100215' AS TIMESTAMP FORMAT 'YYYYMMDDHHMISS')
```

- 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 data type varchar(4000). 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, however, for loading data directly from Hive into Teradata, it is recommended to use the TPT functionality instead.

When processing Teradata loads from Hive/HDFS using Apache Sqoop, WhereScape recommends that you use vendor supplied drivers, such as 'Hortonworks Connector for Teradata'.

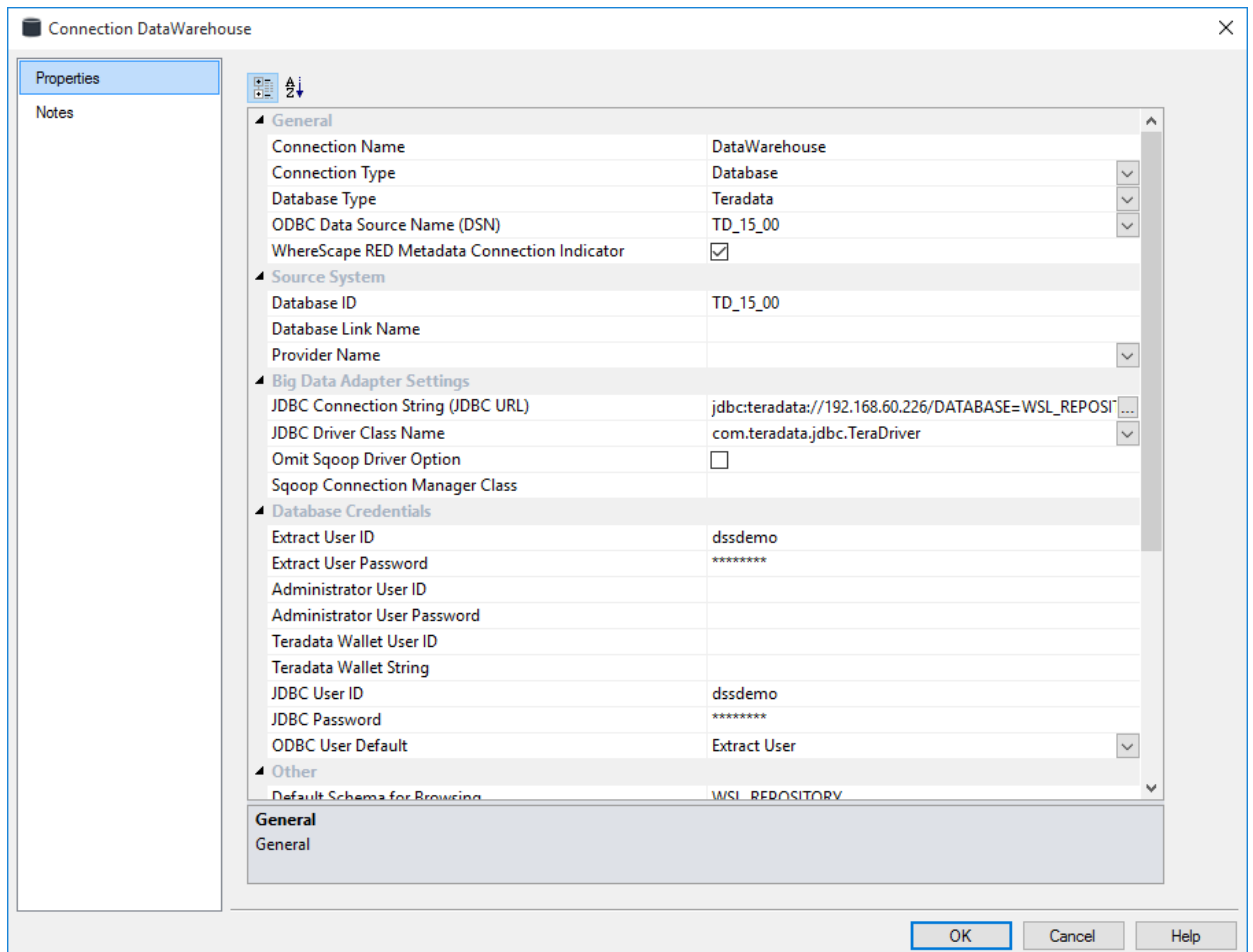
Scheduler loads and Apache Sqoop loads from a Hive connection to a Hive target are not supported.

The following known issues exist when using Sqoop loads with the generic JDBC driver on Teradata:

- 1 Sqoop loads from Hive to Teradata fail if column names and titles are different.
- 2 For Sqoop loads from Hive to Teradata where column names and titles are the same, complete the following to enable the load to work:
In the Load table properties, select the Source tab and on the Generic Hadoop Arguments field enter the following command: `-Dsqoop.export.records.per.statement=1`

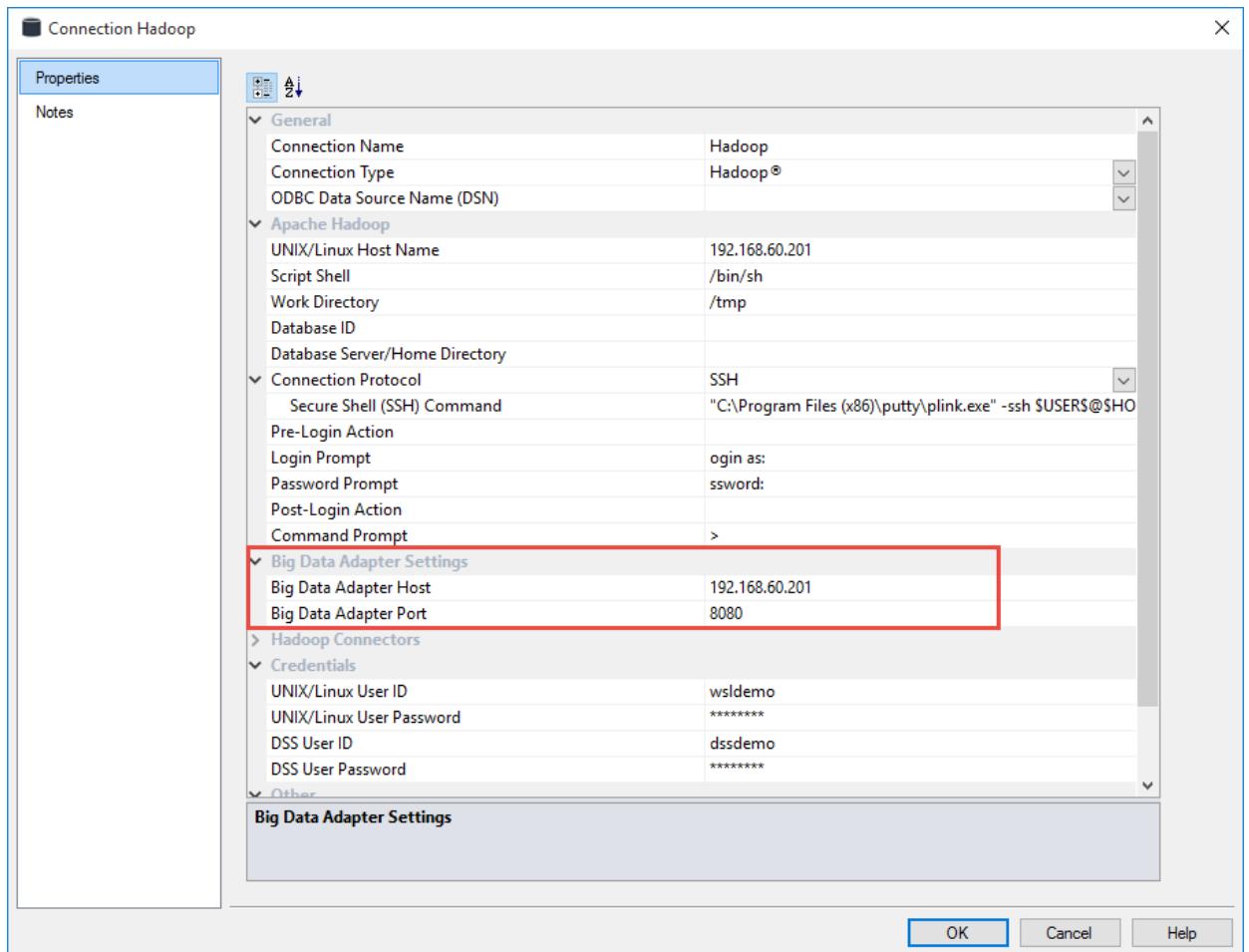
To load tables directly from HDFS/Hive into Teradata target databases using the Apache Sqoop load:

- 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

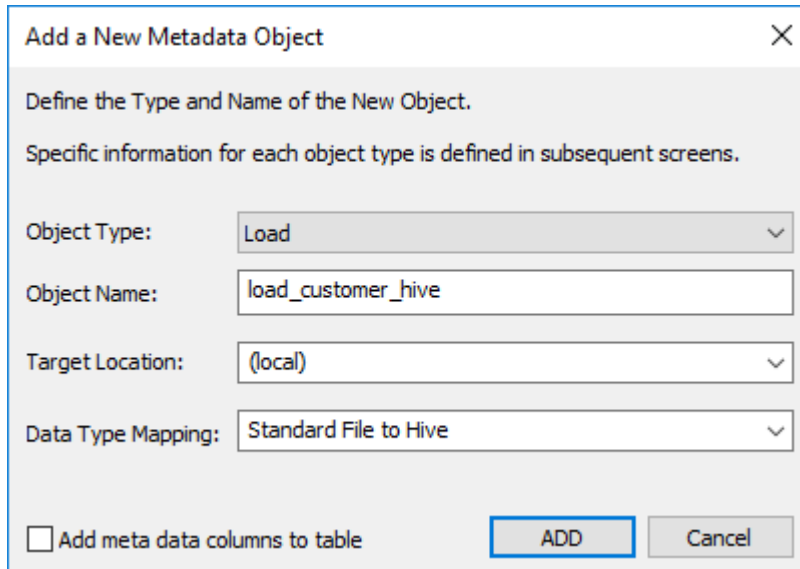


NOTE: Users loading into Teradata from a Hive or Hadoop connection using the Teradata connection manager for Sqoop who need to load into more than one database will need to add DATABASE=\$OBJECT_DATABASE\$ into their JDBC Connection String (JDBC URL) field on their Teradata DataWarehouse connection, e.g.
 jdbc:teradata://192.168.60.226/DATABASE=\$OBJECT_DATABASE\$.
 BDA will replace \$OBJECT_DATABASE\$ with the database containing their load table when doing an Apache Sqoop load.

- 2 When doing loads 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.



Add a New Metadata Object [Close]

Define the Type and Name of the New Object.
Specific information for each object type is defined in subsequent screens.

Object Type: Load [v]

Object Name: load_customer_hive

Target Location: (local) [v]

Data Type Mapping: Standard File to Hive [v]

Add meta data columns to table

ADD Cancel

NOTE: The option **Add meta data columns to table** is used for creating load tables that are used in creating Data Vault objects. If this option is selected, two DSS columns (**dss_record_source** and **dss_load_date**) are included in the meta data for the table and are populated by transformations. These two DSS columns could equally be applied to other load tables not used in a Data Vault system but are particularly important to comply with the Data Vault standards. Please refer to the **Data Vaults chapter** (see "**Data Vaults**" on page 402) for more details.

5 Select **Apache Hadoop Load** from the Load Type drop-down list.

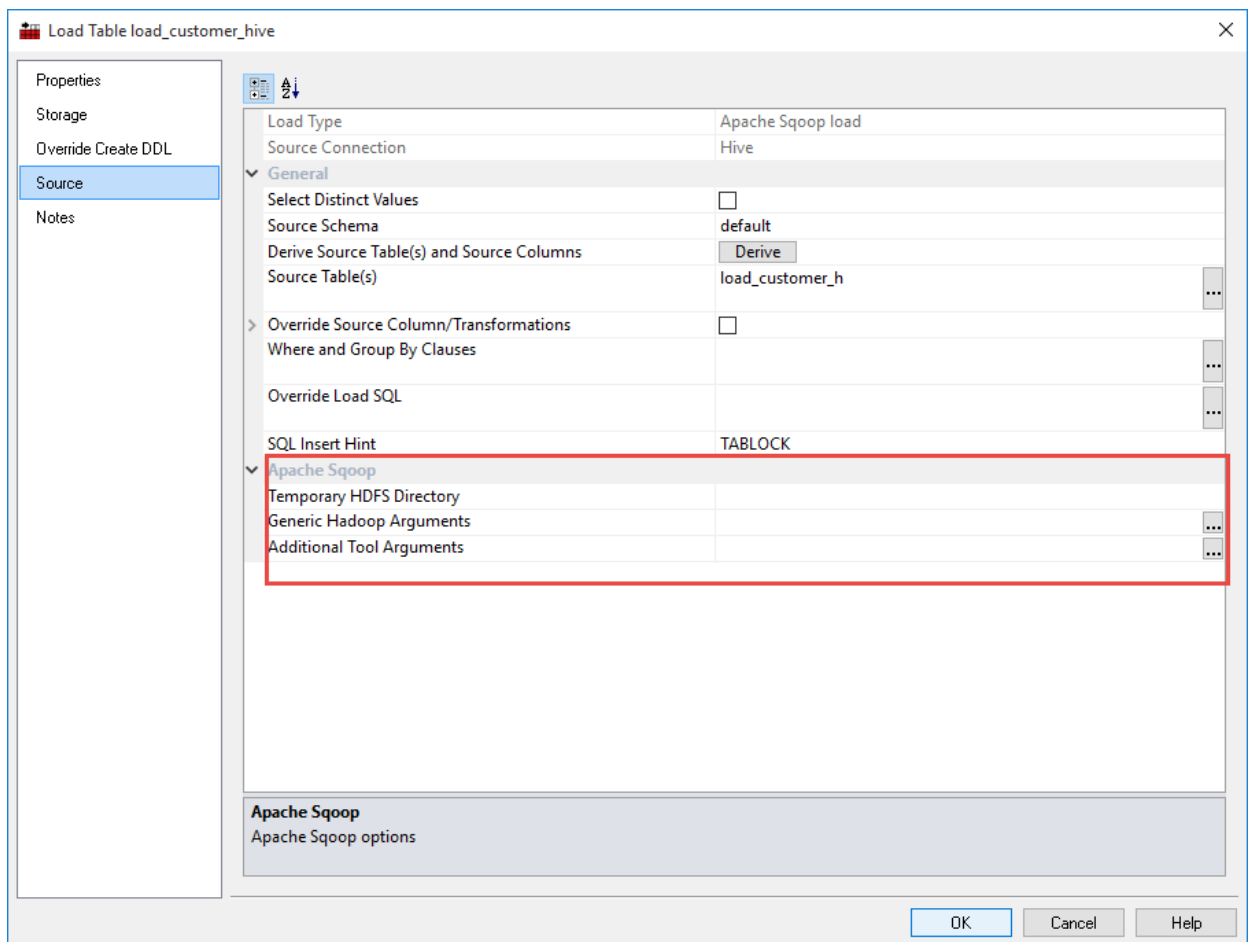
The screenshot shows a dialog box titled "Load Table load_customer_hive" with a close button (X) in the top right corner. On the left is a sidebar with a tree view containing: Properties (selected), Storage, Override Create DDL, Source, File Actions, and Notes. The main area contains the following fields:

- Load Table Name:
- Unique Short Name: (maximum 22 characters)
- Description:
- Connection:
- Load Type: - Database Link:
- Script Template:
- Script Name:
- Pre-Load Action:
- Pre-Load SQL:
- Post Load Procedure:
- Timestamps section with three sub-fields:
 - Metadata Structure Changed:
 - Database Created:
 - Database Altered:

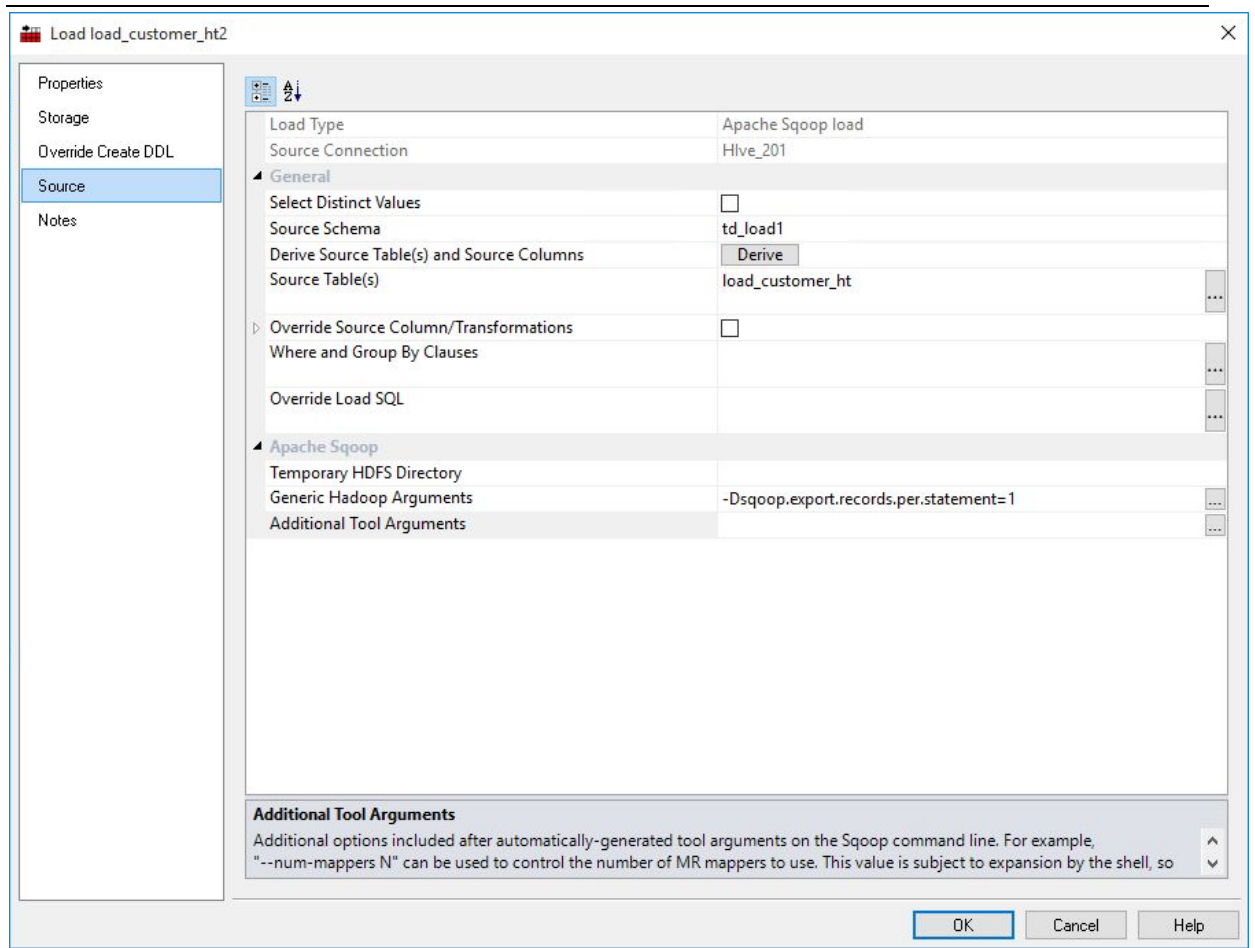
At the bottom right are three buttons: OK, Cancel, and Help.

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.



NOTE: Sqoop loads from Hive to Teradata fail if column names and titles are different. Please ensure column names and titles are the same and then, on Load table properties, go to the source tab and on the Generic Hadoop Arguments field specify the following command=
-Dsqoop.export.records.per.statement=1.



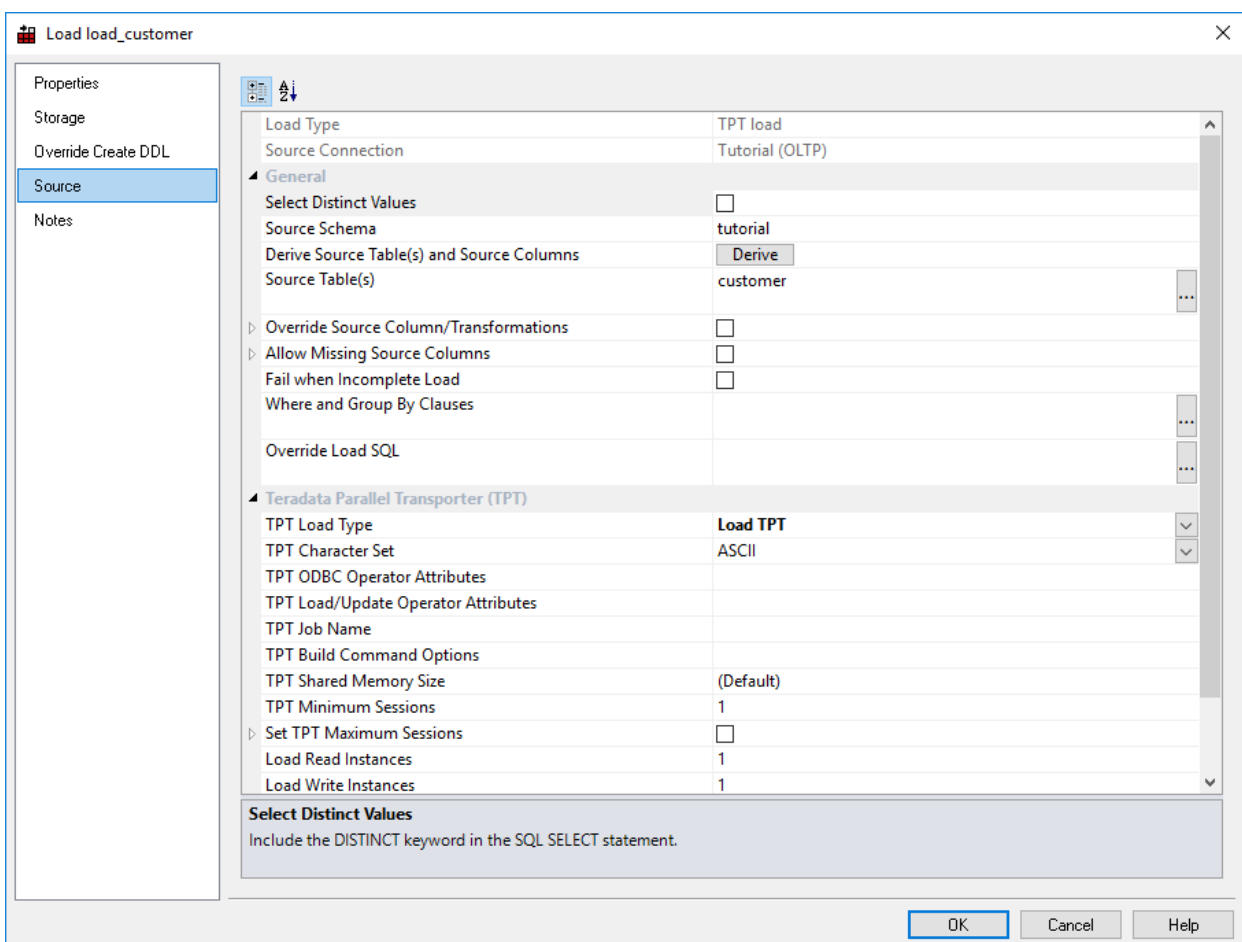
7 Click **Create and Load** to create and load the table.

HANDLING MISSING SOURCE COLUMNS

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 'Non mandatory source column' 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** tab of a load tables properties there are two check-boxes 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 selected 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 list is only available if the **Non Mandatory Source Columns** option is set.

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 593) and *Load Table Column Transformations* (on page 598) 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 598) for more details.

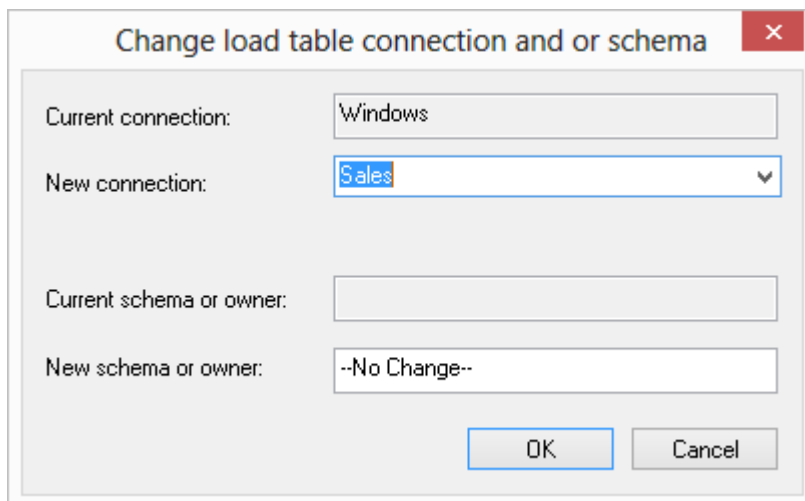
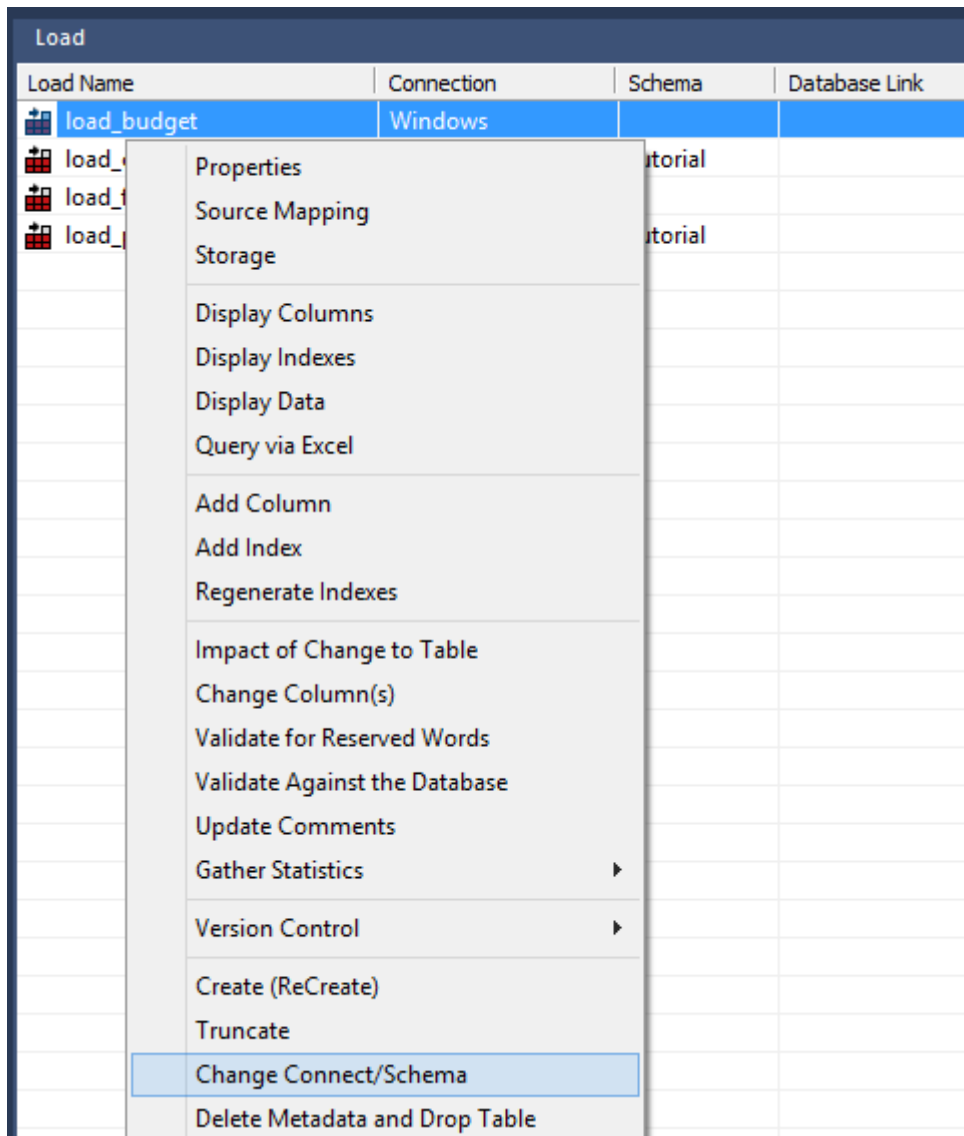
CHANGING LOAD CONNECTION AND SCHEMA

The connection associated with a load table can be changed through the properties of that table.

Connections can also be changed en 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** excepting that it is possible to change from Database to ODBC connections when the following considerations are taken into account.



Switching Connection from ODBC to Database and vice versa

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.

CHAPTER 10

DIMENSIONS

IN THIS CHAPTER

| | |
|---|-----|
| Dimensions Overview..... | 280 |
| Building a Dimension..... | 281 |
| Generating the Dimension Update Procedure | 289 |
| Dimension Artificial Keys | 301 |
| Dimension Column Properties | 303 |
| Dimension Column Transformations..... | 311 |
| Dimension Hierarchies..... | 312 |
| Snowflake | 315 |
| Dimension Language Mapping..... | 317 |

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.
- 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.
- Add columns from other load tables if required. See Building a Dimension.
- Create the dimension table in the database. See Building a Dimension.
- Build the update procedure. See *Generating the Dimension Update Procedure* (on page 289).
- Run the update procedure and analyze the results. See Dimension Initial Build.

Modify the update procedure as required. See Dimension Initial Build.

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 (e.g. 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:

```
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

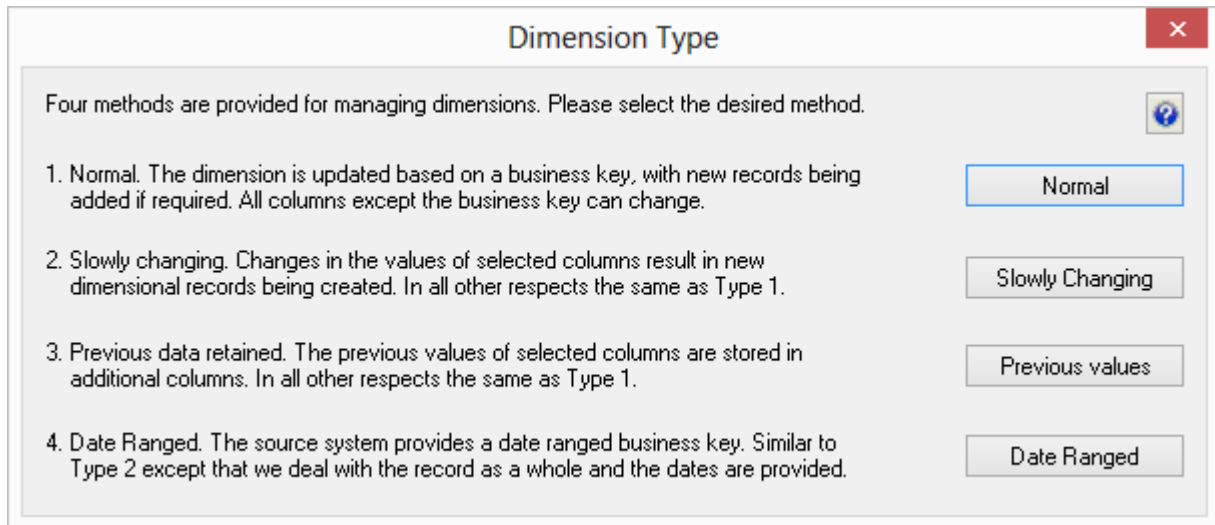
Dimensions are often sourced from one table in the base application. In many cases there are also codes that require description lookups to complete the de-normalization of the dimensional data. The process for building a dimension is the same for most other tables and begins with the drag and drop of the load table that contains the bulk of the dimensional information.

Drag and Drop

- 1 Create a dimension target by double-clicking on the **Dimension group** in the left pane.
- 2 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.
- 3 Browse to the Data Warehouse via the Browse/Source Data menu option.
- 4 Drag the load table, that contains the bulk of the dimensional columns, into the middle pane.
- 5 Drop the table anywhere in the pane.
- 6 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.
- 7 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 - (see more details below).
- The second choice is a **slowly changing** dimension where new dimension records are created when certain identified columns in the dimension change. - (see more details below).
- The third choice is a **Previous values** dimension, which allows the storing of the last values 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 among 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 still 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.

Dimension Properties

- Once the dimension type is chosen the properties page 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.

The screenshot shows a dialog box titled "Dimension dim_product". On the left is a sidebar with a list of options: Properties (selected), Storage, Override Create DDL, Language Mapping, Purpose, Concept, Grain, Examples, Usage, and Notes. The main area contains the following fields and controls:

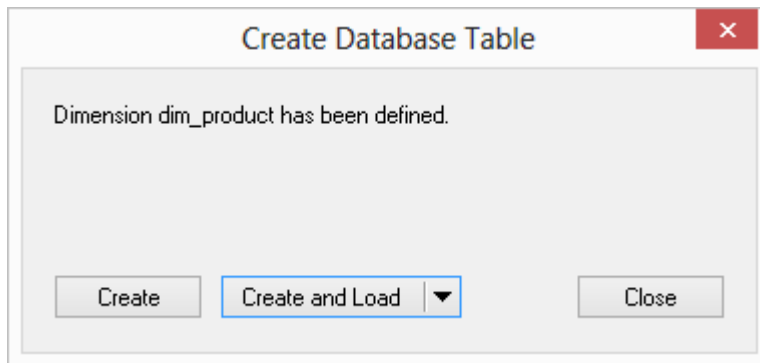
- Table Name:
- Table Type: - Unique Short Name: (maximum 22 characters)
- Business Display Name (EUL):
- Description:
- Update Procedure:
- Custom Procedure:
- Get Key Function:
- Mnemonic (EUL):
- Timestamps section:
 - Metadata Structure Changed:
 - Database Created:
 - Database Altered:

At the bottom right are buttons for , , and .

Create and Load

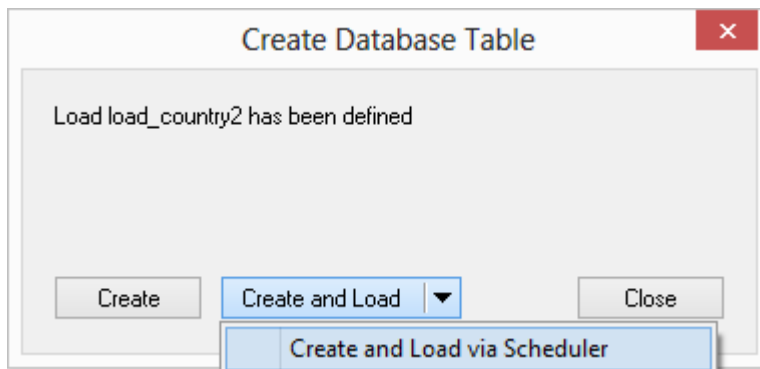
If you chose to build the update procedure the following dialog appears after clicking OK on the Properties page. This dialog asks if you want to create the Dimension table in the database and execute the update procedure.

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 on the Properties dialog you can proceed directly to the *Generating the Dimension Update Procedure* (on page 289) 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 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.
- You can also change the name of a column 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 alphanumeric and the underscore character. See the section on column properties for a fuller description on what the various fields mean.

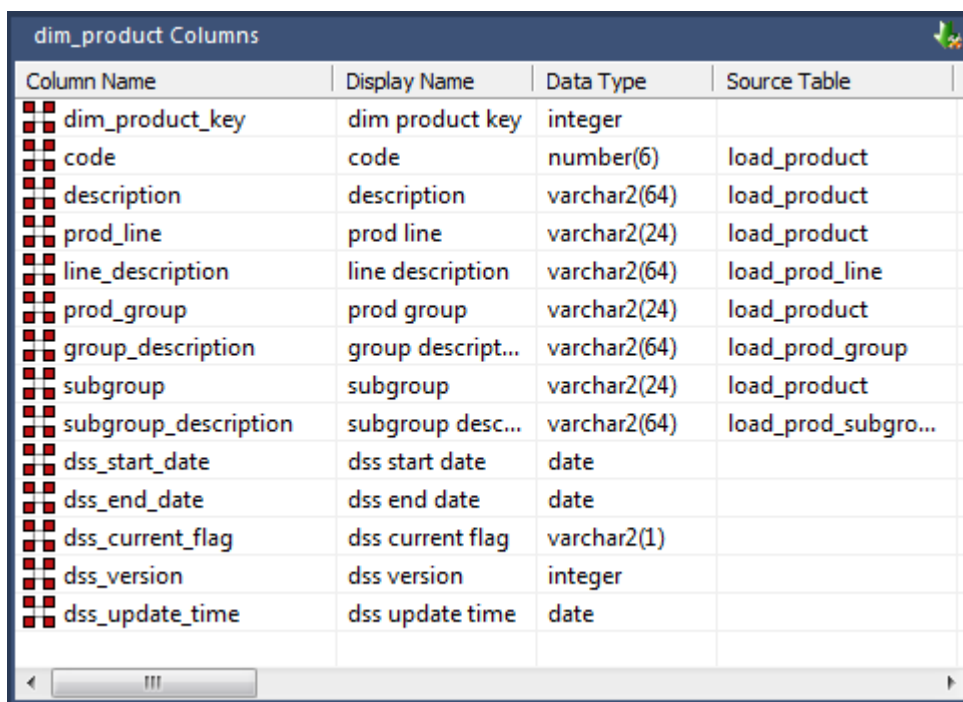


TIP: When prototyping, and in the initial stages of an analysis area build it is best not to remove columns, nor to change their names to any great extent. This type of activity is best left until after end users have used the data and provided feedback.

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 simple to select columns from other load 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.



| Column Name | Display Name | Data Type | Source Table |
|----------------------|-------------------|--------------|---------------------|
| dim_product_key | dim product key | integer | |
| code | code | number(6) | load_product |
| description | description | varchar2(64) | load_product |
| prod_line | prod line | varchar2(24) | load_product |
| line_description | line description | varchar2(64) | load_prod_line |
| prod_group | prod group | varchar2(24) | load_product |
| group_description | group descript... | varchar2(64) | load_prod_group |
| subgroup | subgroup | varchar2(24) | load_product |
| subgroup_description | subgroup desc... | varchar2(64) | load_prod_subgro... |
| dss_start_date | dss start date | date | |
| dss_end_date | dss end date | date | |
| dss_current_flag | dss current flag | varchar2(1) | |
| dss_version | dss version | integer | |
| dss_update_time | dss update time | date | |

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 associated with the table, except for the date dimension where it has the form YYYYMMDD |
| dss_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. |
| 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.

Create the table

Once the dimension has been defined in the metadata you need to physically create the table in the database.

- This is done by right-clicking on the dimension name and selecting **Create (ReCreate)** from the pop-up menu.
- The results dialog box will display 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, 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 of finding this a column is to double click on the list heading 'Col name'. This will sort the column names into alphabetic order. Another double click on the heading will sort the columns back into their create order.

The next section covers the *Generating the Dimension Update Procedure* (on page 289).

GENERATING THE DIMENSION UPDATE PROCEDURE

Once a dimension has been defined in the meta data 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.

Note: You can also generate an update procedure via a template, refer to *Rebuilding Update Procedures* (on page 181) for details.

The zero key row


WhereScape RED always insert a record into the dimension with an artificial key value of zero by default. 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

- 1 Double click on the dimension object to edit the properties for the dimension.
- 2 From the **Update Procedure** drop-down list select (**Build Procedure...**).
- 3 Click **OK** to update the properties and start the process of generating the new procedure.
- 4 The **Update Build Options** screen displays, requesting you to select the relevant build options.
- 5 There are other steps to complete during the procedure generation based on the type of load information. These steps are described below.

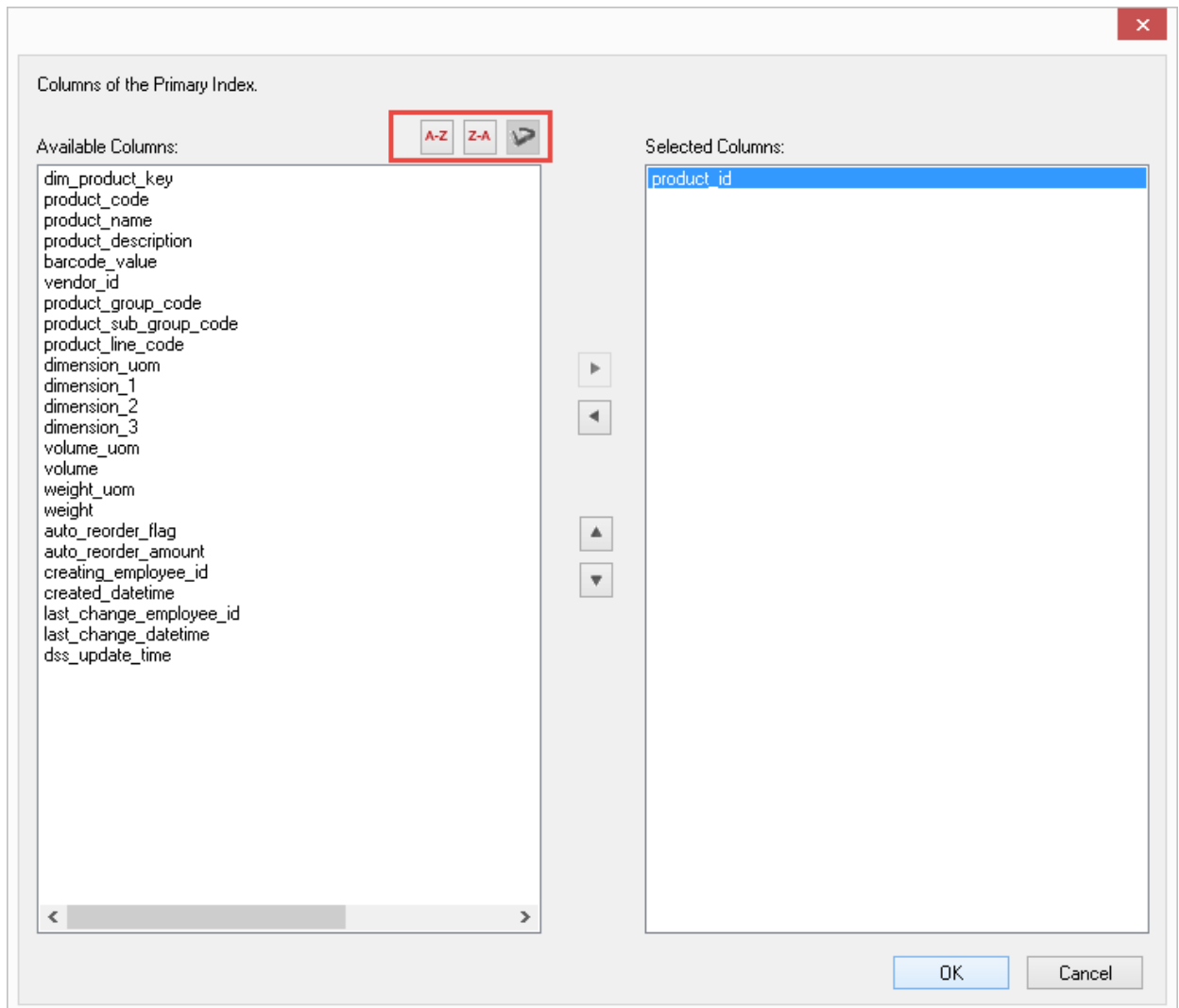
PROCESSING TAB

Business Key Columns: Columns that define the business key for update processing. This is required for include Update options. The source table from which the dimension is derived would normally have some form of unique constraint applied. In most cases this will be the business key. In the example below **product_id** is selected as the business key.

- Clicking on the ellipsis button  on the rightmost of the **Business Key Columns** field will bring up the Business Key selection screen.
- Select the relevant **Business Key**.
- Click **OK** on the Update Build Options dialog.



TIP: 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.

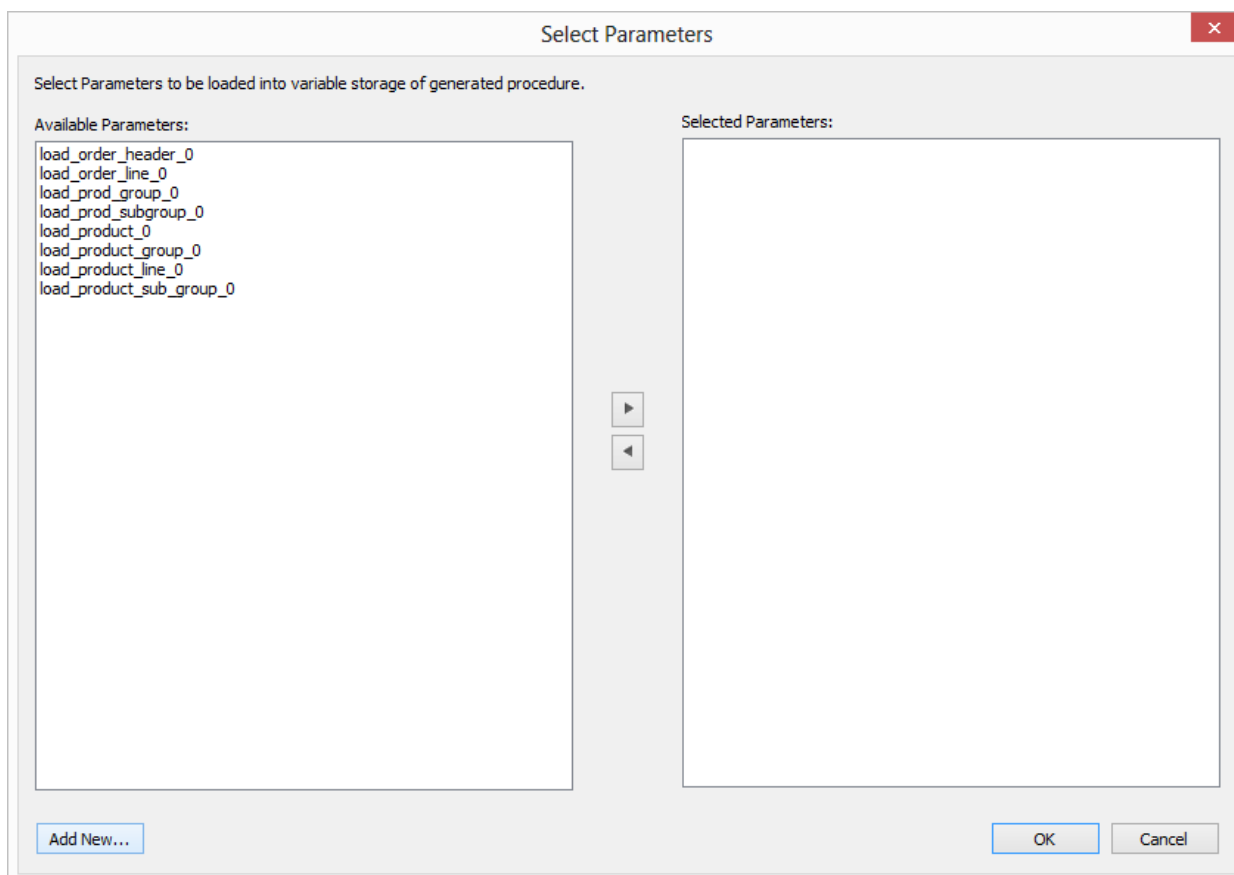


A business key will uniquely identify each dimension record and it can be made up of multiple columns, but it must provide a unique identifier. Where multiple columns uniquely and separately identify the dimension, choose one to act as the primary business key. For example, a source table may have a 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.

NULL Values: None of the columns chosen as the business key should ever contain a NULL value. See the note at the start of this 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 the **Add New** button on the bottom leftmost corner of the Select Parameters dialog.

See *Parameters* (on page 132) for more information on WhereScape RED Parameters.

Include Initial Load Insert: adds an additional insert statement to the update procedure that runs if the target Dimension 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 not set (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 Business Key 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 not set (do not do batch processing).

Delete Before Insert: allows selecting 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. When this option is selected, it enables the **Issue Warning if a Delete occurs** and the **Delete Where Clause** Fields.

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: allows updating the Dimension with either an **Insert/Update** or a **Merge** statement. **Merge** allows you to use one Merge statement instead of two separate Insert and update statements.

Source Table Locking: allows a locking request modifier to be specified for each source table. The specified locking request modifier is applied to each source table during generated update procedures. By default this is set to 'ACCESS' which locks each row being accessed, a blank entry will result in no locking clause in the generated procedure. This option may also be presented in a separate dialog.

Insert Method

Include Insert Statement: set this field to include the insert statement in the procedure. This allows inserting new rows in the Dimension.

Insert New Rows Only: uses change detection to work out which rows will require inserting.

New Row Identification Method: method used to identify that records in source are not currently recorded in the target table. Select **Join** or **Minus**.

Include Update Statement: set this field to include an update statement in the procedure. This allows updating the changing rows in the Dimension. If this is set, the **Update Changed Rows Only** option is available.

Update Changed Rows Only: uses change detection to work out what rows require updating. When set, 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. Select **Join** or **Minus**.

Merge Method

Merge Changed Rows only: uses change detection to work out what rows require merging. When the option is set, it enables the **New Row Identification Method**.

New Row Identification Method: method used to identify which records in the source are not recorded or are recorded differently in the target table. Select between **Join** and **Minus**.

If **non identity columns** are used as artificial keys the only new row identification method is **Join**.



Dimension Update procedures usually perform faster when you use the **Join** method for new row identification.

SOURCE TAB

dim_product Update Build Options.

| | |
|-------------|--|
| Information | |
| Processing | |
| Source | |

| | |
|----------------------|---|
| dim_product | |
| Distinct Data Select | False |
| Source Join | FROM [load_product] load_product JOIN [load_product_line] load_product_line ON load_product.product_line_code = load_product_line.product_line_code JOIN [load_product_group] load_product_group ON load_product.product_group_code = load_product_group.product_group_code LEFT OUTER JOIN [load_product_sub_group] load_product_sub_group ON load_product.product_group_code = load_product_sub_group.product_group_code AND load_product.product_sub_group_code = load_product_sub_group.product_sub_group_code |
| Where Clause | |
| Group By | |

Prev Next OK Cancel Help

Distinct Data Select: ensures duplicate rows are not added to the Dimension. This is achieved by adding the the word DISTINCT 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. See example below for Joining multiple source tables.

Where Clause: The Where clause. Use as a filter to extract only the necessary records that fulfill a specified criteria.

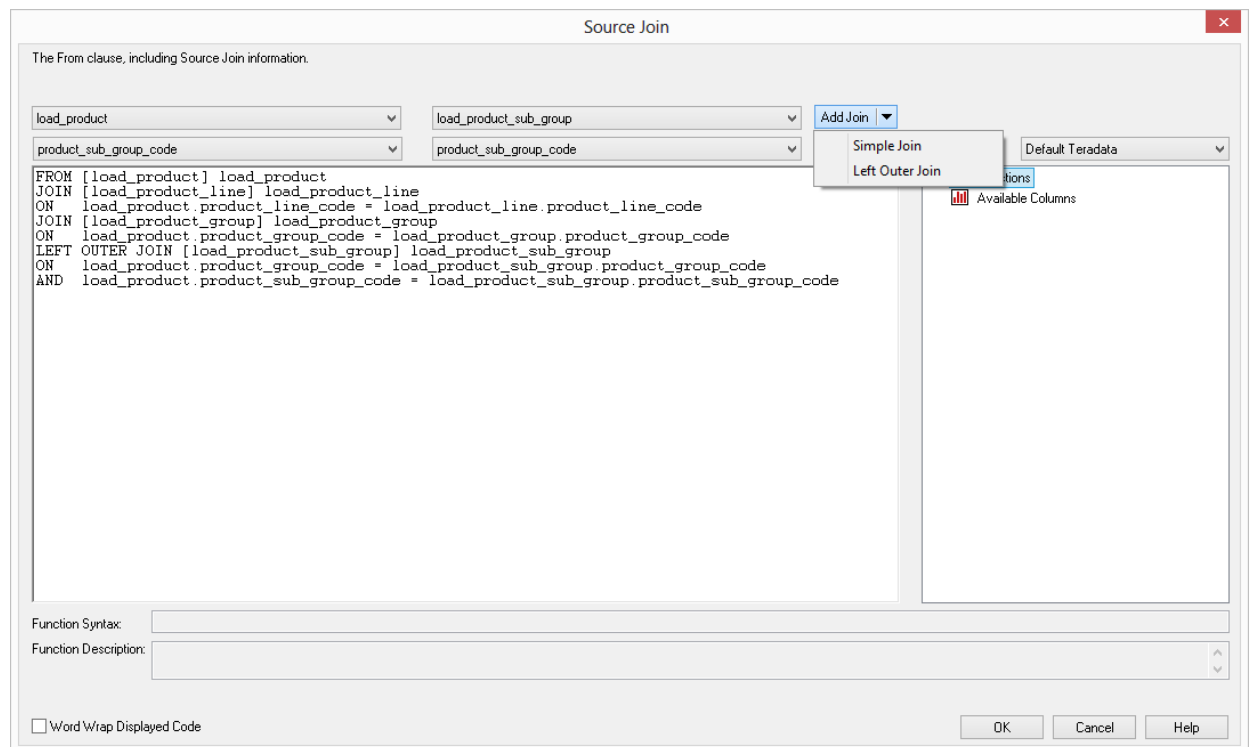
Group By: The Group By clause. Use in collaboration with the SELECT statement to arrange identical data into groups.

Joining multiple source tables

If multiple source tables were used to build the dimension, the tables will need to be joined on the **Source** tab.

- Select the first two tables from the top drop-down list.
- Select one column for each of the two tables from the bottom drop-down lists 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 - `prod_group` and `subgroup`. In this case two joins are actioned for these two tables so both columns can be selected.



Simple Join

Either a 'Where' or 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

Either a 'Where' or from clause join can be generated. The ANSI standard 'from clause' join is recommended. The outer join returns all rows in the master table, regardless of whether or not they are found in the second table. Therefore, 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.

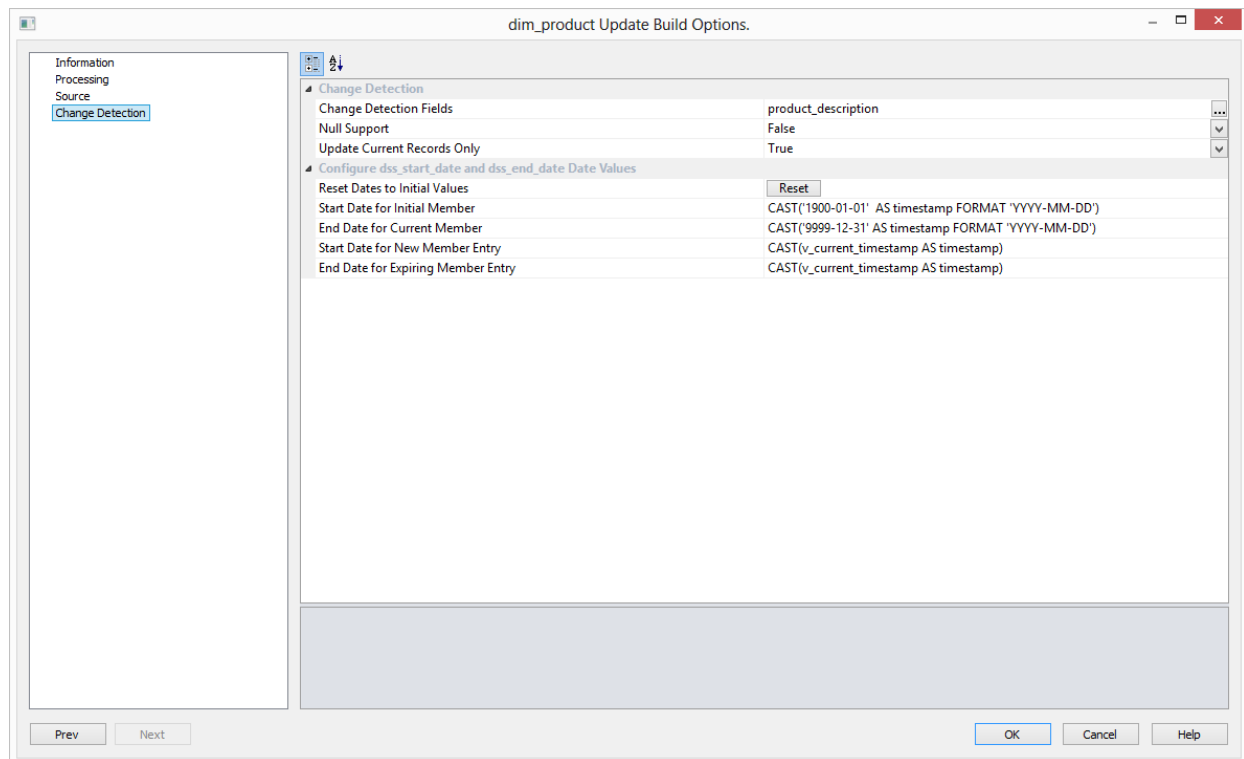
When WhereScape RED builds up a 'Where' clause join, it must place the outer join indicator next to the appropriate column. As this indicator goes on the master, WhereScape RED needs to know which table is master and which subordinate. Select the join column from the master table first.

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 or not a corresponding prod_subgroup entry exists.

As the join columns are selected, the join statement is built up in the large edit window above. Once all joins have been made, the contents of this window can be changed if the join statement is not correct.


- When you are happy with the join clause click the **OK** button to proceed to the next step. This clause will be either the 'Where' clause or a combined from and 'Where' clause depending on the option chosen.
- This clause can be edited in the procedure that is generated if not correct.
- For Teradata, you have the choice between 'Where' statement joins and ANSI standard joins.

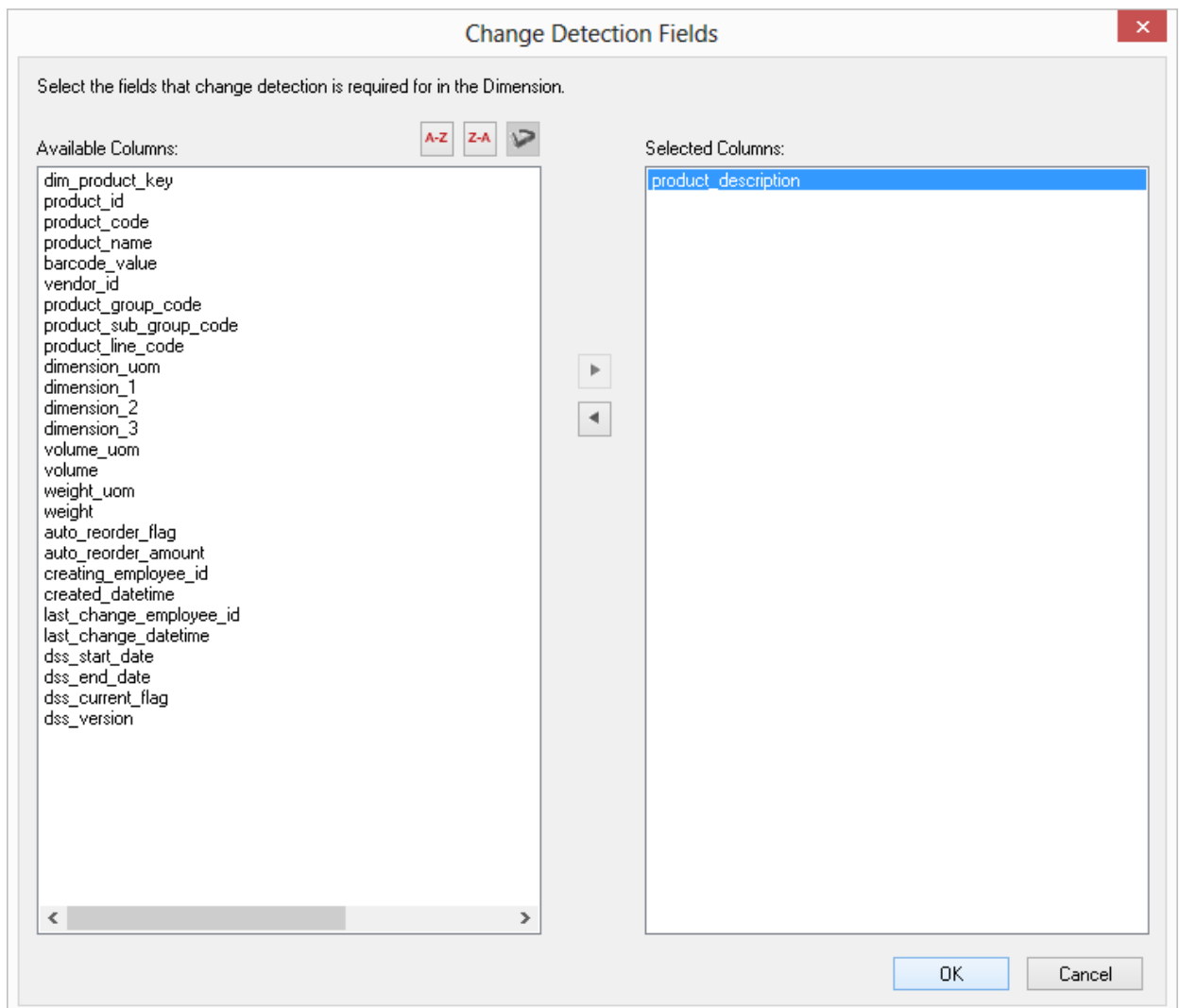
Using Change Detection - Change Detection Tab



Change Detection Fields: if the dimension was defined as a **Changing Dimension** you have to select the change detection fields required for the Dimension on the **Change Detection Tab**. This will allow to select the columns to be managed as slowly changing dimension columns.

The advantages and disadvantages of **changing** dimensions are discussed earlier in this chapter, but as a general rule try to minimize their usage. They invariably complicate the processing and end user queries.

- Click on the **Change Detection** tab in the Update Build Options dialog.
- Click on the ellipsis button  at the rightmost corner of the **Change Detection** field.
- Select the required columns to be managed as slowly changing dimension columns and click **OK** to continue.
- In the example below, the `product_description` is to be managed as a slowly changing column.



Null Support: if this option is set, the change detect column management will cater for Null values in any changing columns. If this is not set and there are Null values in the changing columns there may be errors while running the update procedure. The default for this option is not set (Nulls are not catered for).

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 Coalesce() transformation when loading the column.

Configuring `dss_start_date` and `dss_end_date` for Date Detection Values:

Reset Dates to Initial Values: resets `dss_start_date` and `ds_end_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 start 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.

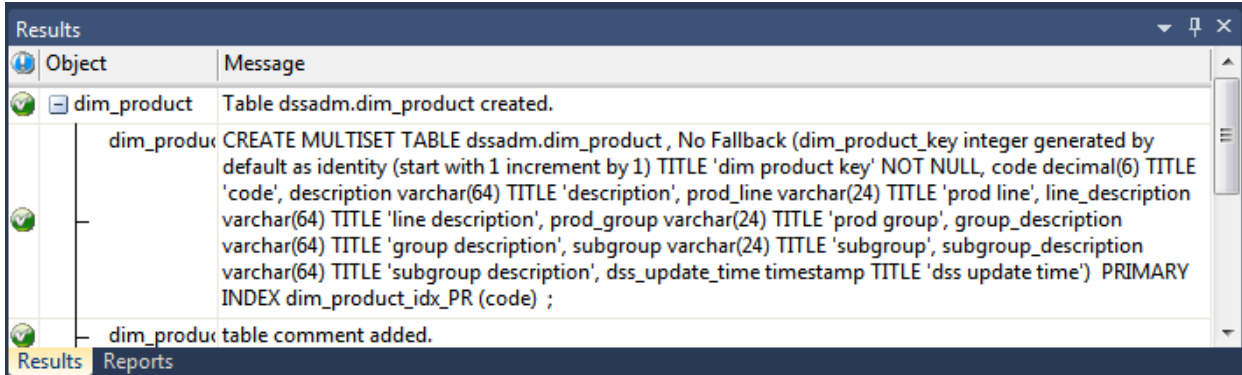
End Date for Expiring Member Entry: the end date for the expiring member entry field contains the end date for any rows updated no longer to no longer be the current row in the history table (i.e. rows that are replaced by a new current row). The value should be specified in an appropriate form, taking 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

- When the steps above are completed the procedure is built and compiled automatically. This will display in the **Results** pane.
- 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 an identity column. 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 have to 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. |

Surrogate keys for a Dimension set via a non identity column:

Normal, Slowly Changing and Date Ranged Dimension Tables can have non identity columns as surrogate keys.

The generation of the update procedure will automatically add logic to the code which will associate a sequential number to the artificial key of the dimension when a new row is inserted into the Dimension table.

The order of these sequential numbers is determined by the business key of the source table. The value of the first newly inserted artificial key will be the value of the highest artificial key in the dimension table plus 1.

This automatically generated logic can be overwritten by defining a user specific logic on the **Dimension Transformation** field on the **Tools/Options** menu or in the transformation column of the

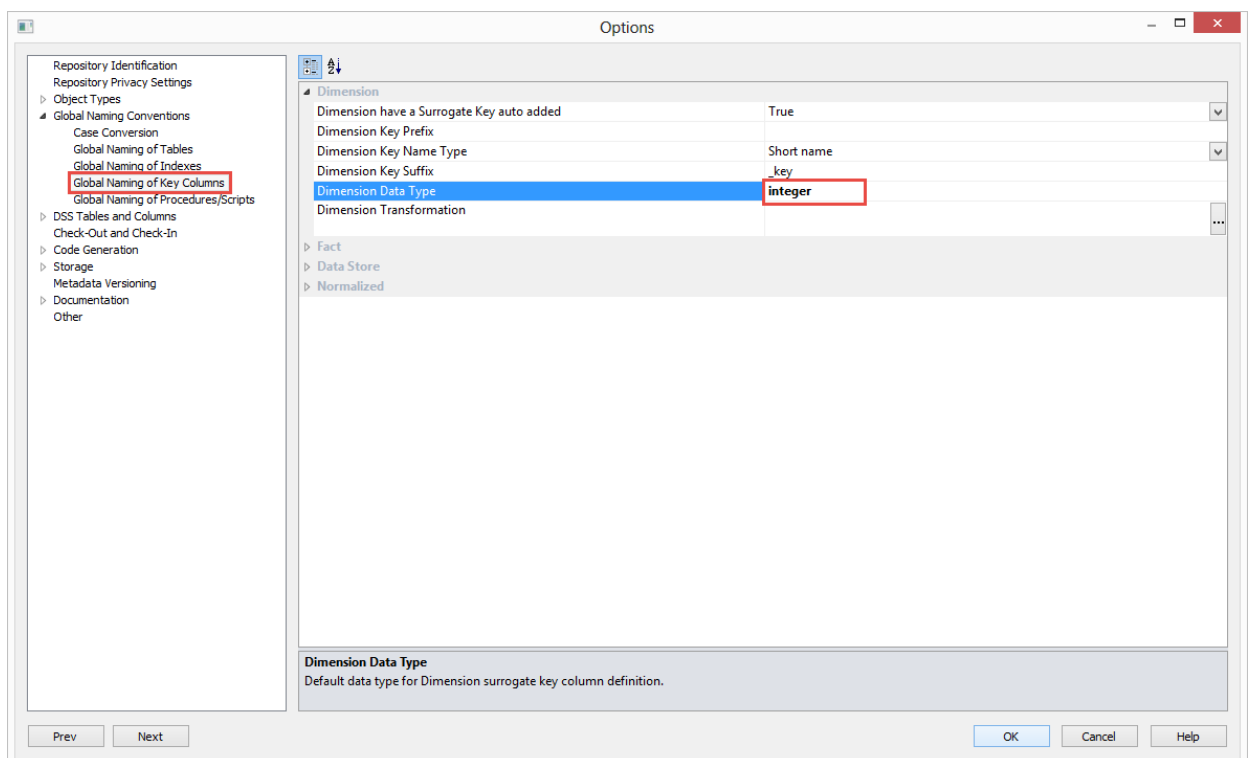
artificial key.

To have a Dimension with a non identity column as a surrogate key, you can **set the Dimension Data Type to integer** in the Tools/Options menu.

The old logic for dimensions can be retained if an identity column is chosen as surrogate key.

To allow for non identity surrogate keys on Dimensions:

- 1 Go to **Tools>Options>Global Naming Conventions>Global Name of Key Columns**.
- 2 Set the **Dimension Data Type** to be **integer** and click **OK**.
- 3 If your tables have been created previously, you will have to **Recreate** the tables after you set this option in the **Tools** menu.



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:

The screenshot shows a dialog box titled "Dimension Column dim_customer.code". On the left, there is a "Properties" sidebar with "Transformation" and "Language Mapping" options. The main area is a table of properties:

| Property | Value |
|------------------------------|-------------------------------------|
| Table Name | dim_customer |
| Column Name | code |
| Column Title | code |
| Column Description | |
| Physical Definition | |
| Column Order | 20 |
| Data Type | decimal(6) |
| Null Values Allowed | <input checked="" type="checkbox"/> |
| Default Value | |
| Character Set | |
| Format | |
| Character Comparison/Sorting | |
| Compress | <input type="checkbox"/> |
| Meta Definition | |
| Numeric | <input checked="" type="checkbox"/> |
| Additive | <input checked="" type="checkbox"/> |
| Attribute | <input type="checkbox"/> |
| End User Layer Display | <input checked="" type="checkbox"/> |
| Business Key | <input checked="" type="checkbox"/> |
| Artificial Key | <input type="checkbox"/> |
| Key Type (0,A,B,C,...) | A |
| Code Generation | |
| Zero Key Value | |
| Source Details | |

At the bottom, there are buttons for "< Update", "Update >", "OK", "Cancel", and "Help". A tooltip for "Column Name" is visible at the bottom of the dialog, stating: "Database-compliant name of the column. Dialog Opening Value: code".

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. 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.

Column Title

Name that the business uses to refer to the column. It does not affect the physical table definition, but rather provides input to the documentation and to the view **ws_admin_v_dim_col** which can be used to assist in the population of a end user tool's end user layer. 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 might 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 a valid for the target database. Typical Teradata databases often have integer, numeric(), varchar(), char(), date and timestamp data types. 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.

Character Set

Database-compliant table column character-set used for storage. Select Latin or Unicode.

Format

Database-compliant table column format. 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. 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.

Character Comparison/Sorting

Determines how the column character values are treated for comparison and sorting operations. Choose from: case specific, not case specific, uppercase case specific or uppercase not case specific.

Compress

Indicates whether the table column values are compressed when stored.

Compress/Compress Value

Optional list of values to be compressed. By default, only NULL is compressed if no list of values is specified.

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. This indicator is set and cleared by WhereScape RED during the dimension update procedure generation process. This checkbox 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. 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. |

| Key type | Meaning |
|----------|---|
| 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.

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].

Join

Indicates whether the table column is used in a table join. Normally this is maintained automatically but can be optionally changed to override the default logic used in the generated update procedure. The default for this option is not set.

CHANGING A COLUMN NAME

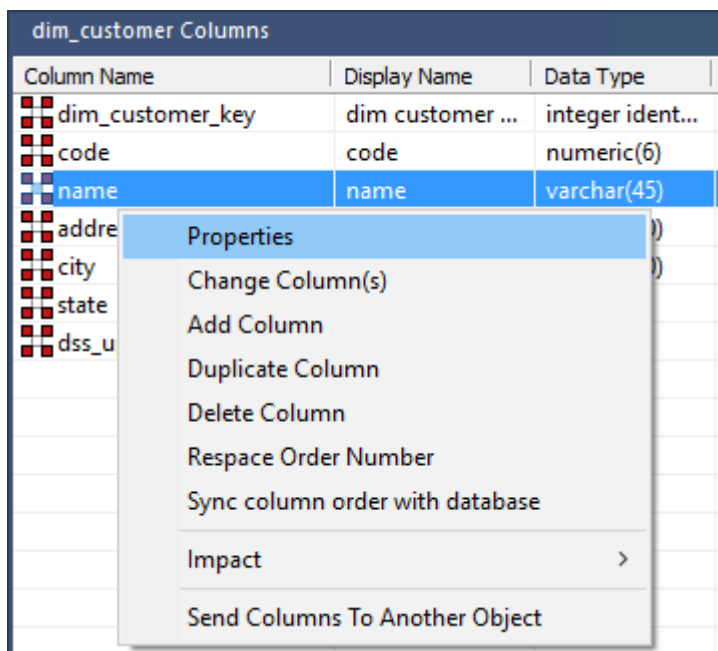
If the **Column name** or **Data type** is changed for a column then the metadata will differ from the table as recorded in the database.

- 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.

Dimension Column dim_customer.name

Properties

- Transformation
- Language Mapping

General

| | |
|--------------------|---|
| Table Name | dim_customer |
| Column Name | cname |
| Column Title | cname |
| Column Description | The full name of the customer. Forms a hierarchy with city and state. |

Physical Definition

| | |
|------------------------------|-------------------------------------|
| Column Order | 30 |
| Data Type | varchar(45) |
| Null Values Allowed | <input checked="" type="checkbox"/> |
| Default Value | |
| Character Set | |
| Format | |
| Character Comparison/Sorting | |
| Compress | <input type="checkbox"/> |

Meta Definition

| | |
|------------------------|-------------------------------------|
| Numeric | <input type="checkbox"/> |
| Additive | <input type="checkbox"/> |
| Attribute | <input checked="" type="checkbox"/> |
| End User Layer Display | <input checked="" type="checkbox"/> |
| Business Key | <input type="checkbox"/> |
| Artificial Key | <input type="checkbox"/> |
| Key Type (0,A,B,C,...) | |

Code Generation

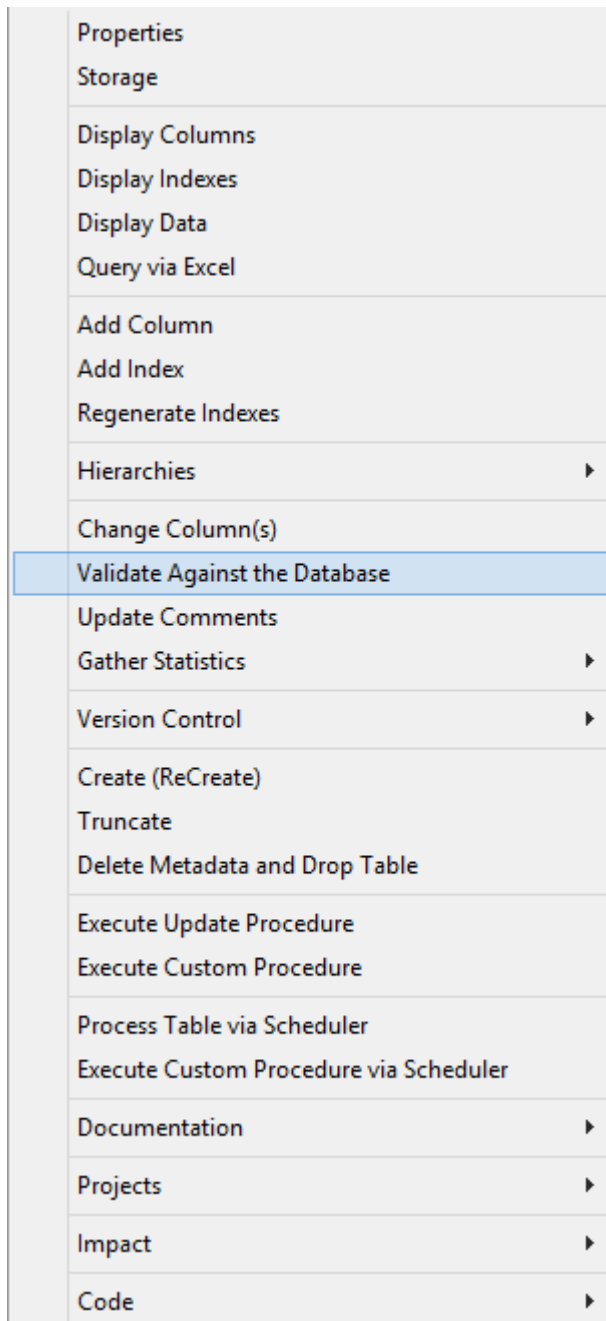
| | |
|----------------|---------|
| Zero Key Value | Unknown |
|----------------|---------|

Source Details

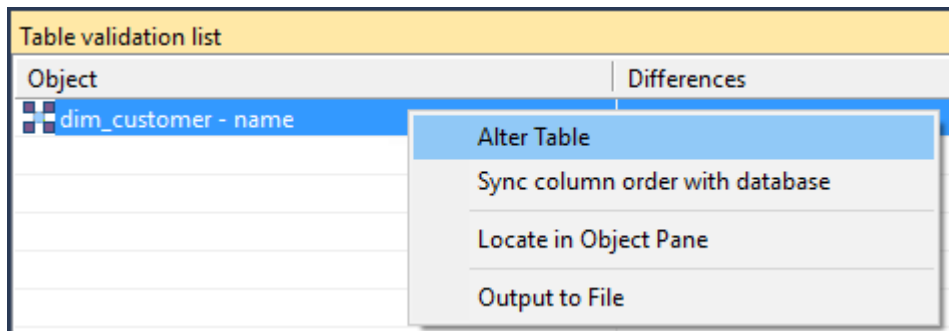
Column Name
Database-compliant name of the column.
Dialog Opening Value: name

< Update Update > **OK** Cancel Help

- 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 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.



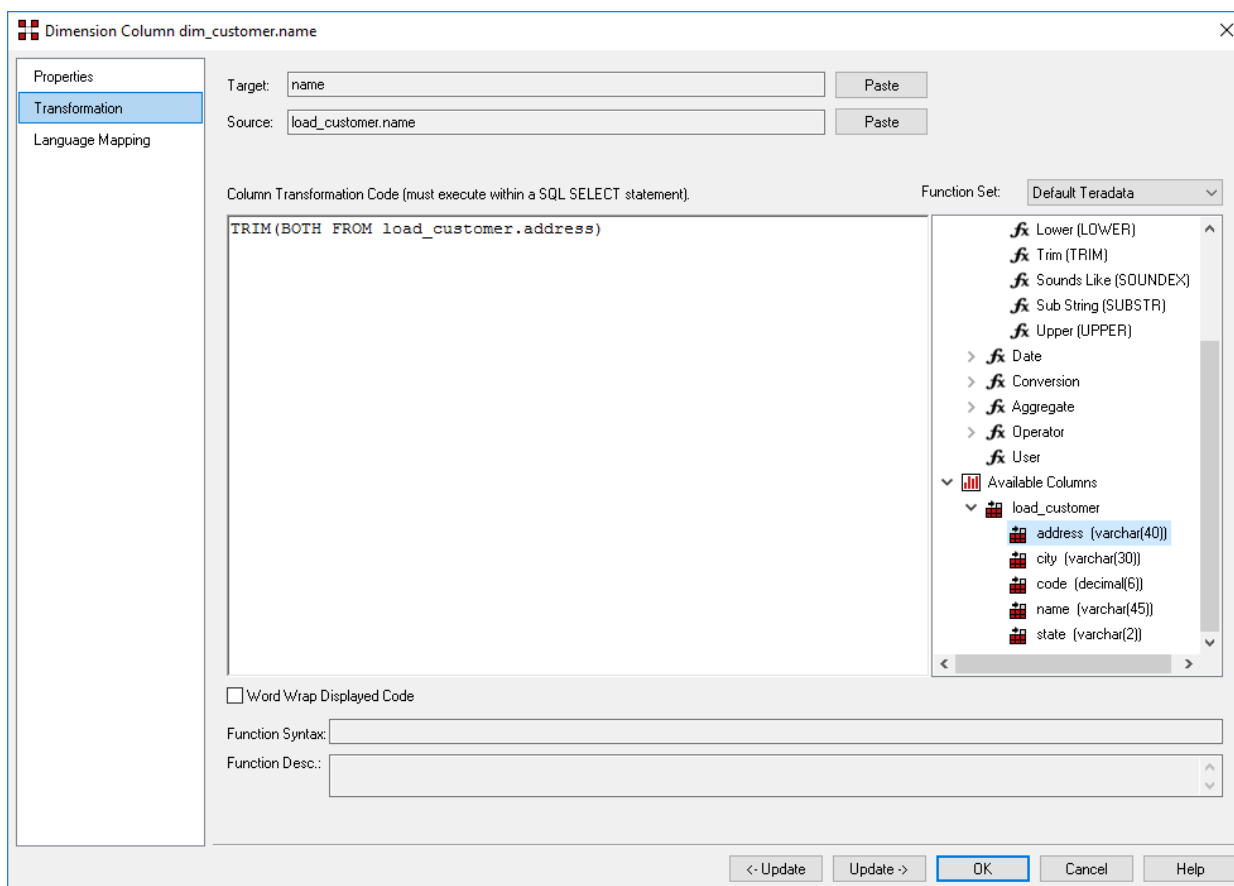
- 7 A warning will appear, displaying the table and column name to be altered. Select **Alter Table**.
- 8 A dialog will appear confirming that `dim_customer` has been altered. Click **OK**.

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 593) 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. They can however be used in conjunction with the WhereScape hierarchy maintenance utility to allow the maintenance of hierarchy elements externally to a production source system.

When used in conjunction with the hierarchy maintenance utility, these dimension hierarchies add a powerful method of enriching the analysis capabilities of the data warehouse. For example, we may have a source system that has a dimension called sales_person. This dimension has no information apart from employee_code, name and title. We could add additional columns of sales_manager, territory, state and region to this dimension. A hierarchy could then be formed from the salesperson name, sales_manager, territory, state and region. The hierarchy maintenance utility allows the maintenance of this hierarchy

externally to the data warehouse. This external hierarchy can then become a source system to enrich the data in the warehouse.

Two areas will be covered. Firstly the creation of hierarchies using WhereScape RED, and secondly the process required to setup and use externally maintained hierarchies as source systems to the data warehouse.

ADDING A DIMENSION HIERARCHY

Any number of hierarchies can be created against a dimension. There is no restriction on the form of the hierarchy.

- 1 To add a new hierarchy, position on the dimension table in the left had pane and using the right-click menu select **Hierarchies/Add Hierarchy**.

The following dialog will appear:

Add Hierarchy [Close]

Hierarchy Name:

Description:

Move columns from the column list into the hierarchy. The hierarchy is a top down list.
For example a date hierarchy may be year, month, day. Year will be the first column shown.

Available Columns:

| Column Name |
|--|
| <input checked="" type="checkbox"/> dim_customer_key |
| <input checked="" type="checkbox"/> code |
| <input checked="" type="checkbox"/> name |
| <input checked="" type="checkbox"/> address |
| <input checked="" type="checkbox"/> city |
| <input checked="" type="checkbox"/> state |
| <input checked="" type="checkbox"/> dss_update_time |

Hierarchy:

Levels

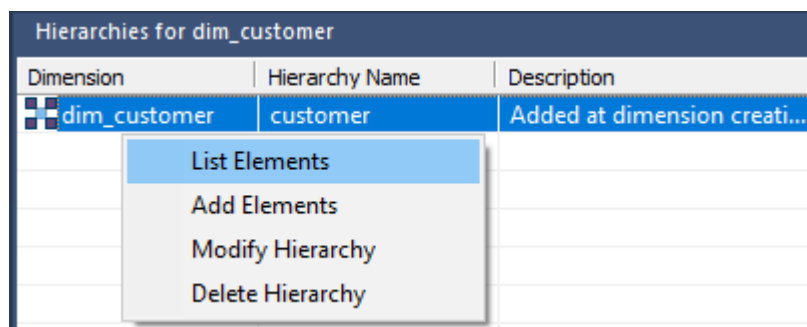
[OK] [Cancel]

- 2 Enter a meaningful name for the hierarchy.
- 3 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 **state** at the highest level, then **city**, then **address** and finally **code** at the lowest level.

- To enter the hierarchy elements, select them in the required order, from the left pane and click the right arrow (>) to add them to the right column.
- 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.



| Dimension | Hierarchy Name | Description |
|--------------|----------------|------------------------------|
| dim_customer | customer | Added at dimension creati... |

A context menu is open over the first row of the table, showing the following options:

- List Elements
- Add Elements
- Modify Hierarchy
- Delete Hierarchy

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.

USING A MAINTAINED HIERARCHY

Once a hierarchy has been added to a dimension, it can be maintained externally by the hierarchy maintenance utility if the **Maintained** attribute is set.

The process for maintaining this hierarchy externally and using it as a source system to the data warehouse is as follows.

- 1 Grant the user that is to undertake the maintenance select access to the following tables: ws_dim_tab, ws_dim_col, ws_hie_header, ws_hie_link, ws_meta_tables.
- 2 Grant the user that is to undertake the maintenance select access to the dimension that has the hierarchy.
- 3 Using the hierarchy maintenance utility (see online documentation) log on to the maintenance area and create and populate/edit the maintenance table.
- 4 In WhereScape RED, create a connection to the hierarchy maintenance schema/database.
- 5 Browse the hierarchy connection.
- 6 Using drag and drop create a load table from the hierarchy maintenance table.
- 7 Edit the columns in the dimension that are to be sourced from the maintenance table and change their source table and column to that of the load table and columns created in step 6.
- 8 Generate a new update procedure for the Dimension and either do a lookup of the maintenance table or a join based on the business key.
- 9 Run the update procedure.

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 normalized 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. It is created 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:

Dimension Business Key Definition

Dimension Table Column List:

- address
- city
- code
- dim_customer_key
- name
- state

Business Keys for snowflake dimension dim_state

Select the business key from the dimension table that matches each business key for this snowflake dimension.

Move them over to the business key list. They must be in the correct order to match the snowflake dimension business keys.

Dimension Business Key List:

- code

Snowflake Business Keys:

- state_code

dim_state

No Warning (if dimension joins do not succeed)

Automatically Add a New Entry (if no valid dimension record)

Write Detail Log Message (when no dimension record match)

Add Text

Enter a text string and press Add Text to add a static Business Key Value

OK

Cancel

- 10 Add the business key column(s) and click **OK**.

DIMENSION LANGUAGE MAPPING

The Dimension Properties screen has a tab called **Language Mapping**.

- 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.

The screenshot shows a dialog box titled "Dimension dim_customer" with a close button (X) in the top right corner. On the left side, there is a vertical menu with the following items: Properties, Storage, Override Create DDL, Language Mapping (highlighted in blue), Purpose, Concept, Grain, Examples, Usage, and Notes. The main area of the dialog is divided into two columns. The left column contains the following fields: "Language :" with a dropdown menu showing "French"; "Business Display Name:" with a text input field containing "dim_customer"; and "Description:" with a text area containing "This is the master Customer List for the Company". The right column contains "Language Settings:" with a text input field containing "Translated Value of Business Display Name" and another text area containing "Translated Value of Dimension Description". At the bottom right of the dialog, there are three buttons: "OK", "Cancel", and "Help".

CHAPTER 11

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 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 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. As the stage table is essentially a subset of the fact table, the design and creation of the stage table is essentially the design and creation of the model table. The steps are:

- 1 Identify the source transactional data that will ultimately constitute the model 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.
- 2 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 319).
- 3 Add columns from other source tables. See *Building the Stage Table* (on page 319).
- 4 Create the stage table in the database. See *Building the Stage Table* (on page 319).
- 5 Build the update procedure. See *Generating the Staging Update Procedure* (on page 322).

NOTE: If you are building a Data Vault system, a **Stage** table with sub type of **Data Vault Stage** can be created to generate hash keys that are used in building Data Vault objects (Hub, Link or Satellite tables). Refer to *Data Vaults* (on page 402) for details.

IN THIS CHAPTER

| | |
|---|-----|
| Building the Stage Table | 319 |
| Generating the Staging Update Procedure | 322 |
| Stage Table Custom Procedure..... | 331 |
| Stage Table Column Properties | 331 |
| Stage Table Column Transformations..... | 337 |
| Permanent Stage Tables | 338 |
| Generating the Permanent Staging Update Procedure | 339 |
| Set Merge Procedure..... | 345 |

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 model table that will be ultimately 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 or not it is practical or even possible to join the data from the different source tables.

We may have to build a model table containing data from invoice headers, invoice lines, order headers and order lines source tables. There are three basic options:

- 1 Join all four tables in one large join in our staging table.
- 2 Update the staging table in two passes. One pass updating the order information and one 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 model 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 (or the use of a custom procedure). 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. Then drag specific columns from the other source tables until we have all the source data that is required.

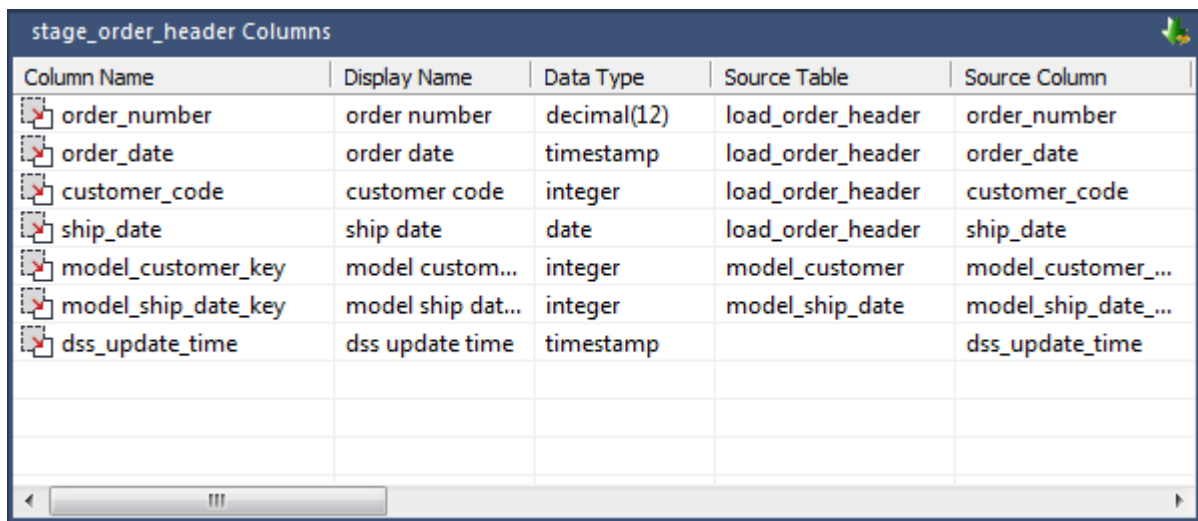
The process for defining the metadata is as follows:

- 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 DataWarehouse connection to display our load tables in the right pane. This is achieved via the **Browse/Source Tables** menu option and then choosing the **DataWarehouse 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 in the middle pane. A dialog will appear to create the new staging object. 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 defined.
- 4 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.

- 5 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 at this stage.
- 6 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.
- 7 If using surrogate keys, then drag in the **model artificial key** from each model table that is to be joined to the stage table. We can only join a model if a business key exists amongst the stage table columns or it is possible to derive that business key in some way from the columns or other model tables.

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 our list of columns for the stage table should look something like the list below. Note the source table for each column.



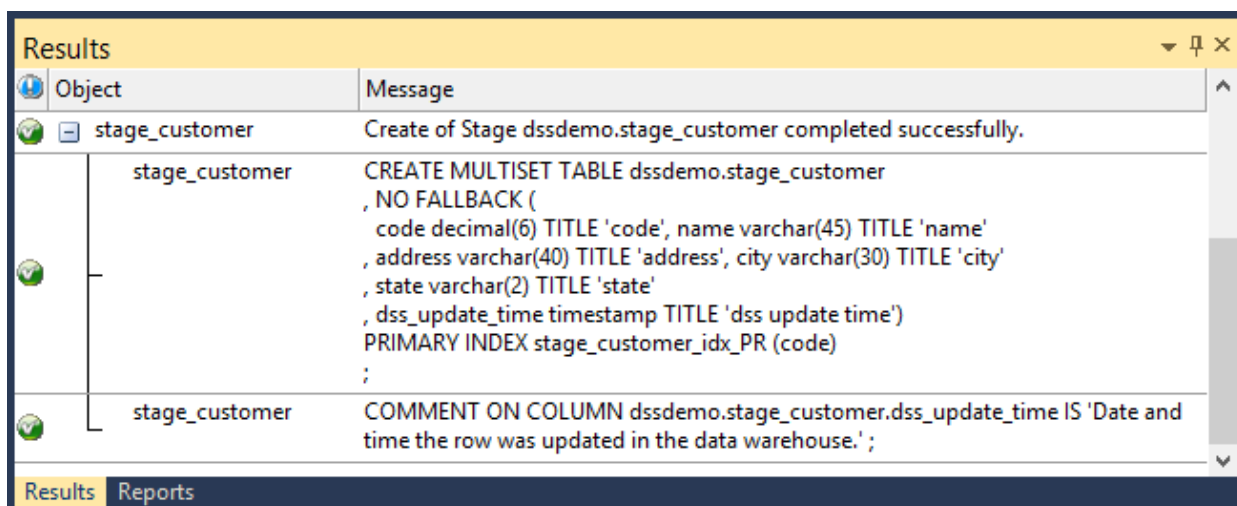
| Column Name | Display Name | Data Type | Source Table | Source Column |
|---------------------|-------------------|-------------|-------------------|---------------------|
| order_number | order number | decimal(12) | load_order_header | order_number |
| order_date | order date | timestamp | load_order_header | order_date |
| customer_code | customer code | integer | load_order_header | customer_code |
| ship_date | ship date | date | load_order_header | ship_date |
| model_customer_key | model custom... | integer | model_customer | model_customer_... |
| model_ship_date_key | model ship dat... | integer | model_ship_date | model_ship_date_... |
| dss_update_time | dss update time | timestamp | | 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_header table. Each model artificial key was dragged from its appropriate table. The final column 'dss_update_time' was generated by RED and has no source.

Create the table

Once the stage table has been defined in the metadata we need to physically create the table in the database.

- 1 Right-click on the stage table name and selecting **Create (ReCreate)** from the pop up menu.
- 2 A results box will appear to show the results of the creation. The following example shows a successful creation.



The screenshot shows a 'Results' window with a table containing the following data:

| Object | Message |
|----------------|--|
| stage_customer | Create of Stage dssdemo.stage_customer completed successfully. |
| stage_customer | CREATE MULTISSET TABLE dssdemo.stage_customer , NO FALLBACK (code decimal(6) TITLE 'code', name varchar(45) TITLE 'name' , address varchar(40) TITLE 'address', city varchar(30) TITLE 'city' , state varchar(2) TITLE 'state' , dss_update_time timestamp TITLE 'dss update time') PRIMARY INDEX stage_customer_idx_PR (code) ; |
| stage_customer | COMMENT ON COLUMN dssdemo.stage_customer.dss_update_time IS 'Date and time the row was updated in the data warehouse.'; |

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 of finding such a column is to double click on the list heading 'Col name'. This will sort the column names into alphabetic 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 322).

GENERATING THE STAGING UPDATE PROCEDURE

Once a stage table has been defined in the metadata and created in the data base an update procedure can be generated to handle the joining of any tables and the lookup of the model table artificial keys.

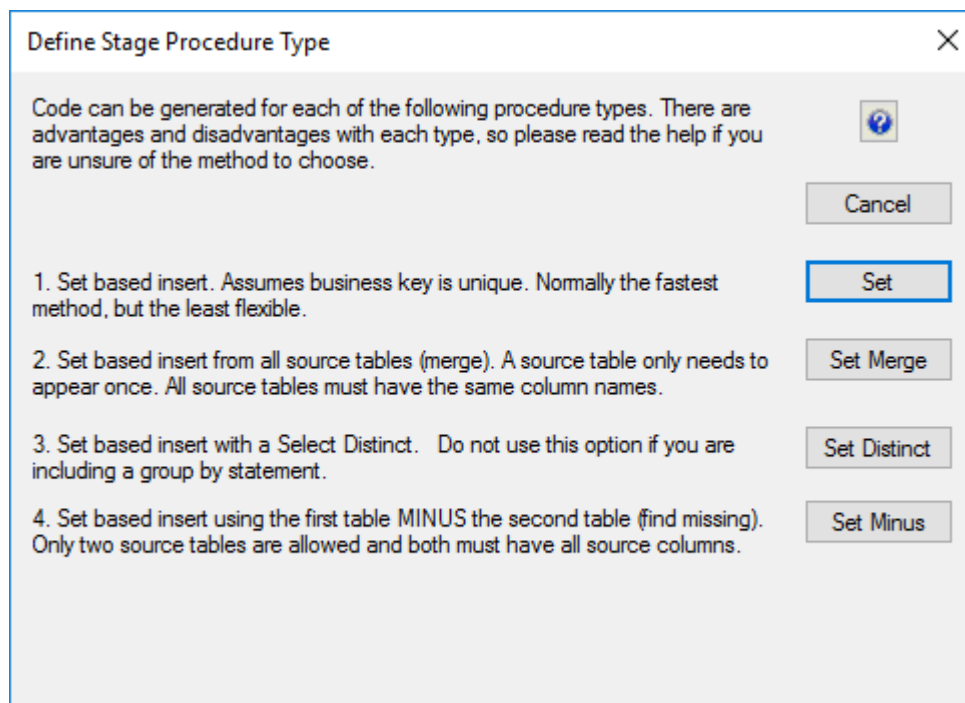
Note: You can also generate an update procedure via a template, refer to *Rebuilding Update Procedures* (on page 181) for details.

Generating a Procedure

- 1 To generate a procedure, right-click on the stage table in the left pane and select **Properties**.
- 2 From the **Update Procedure** drop-down list, select (**Build Procedure...**).
- 3 Click **OK** to update the properties and start the process of generating the new procedure.
- 4 A series of prompts are displayed during the procedure generation to join the tables and link in the model tables.

Procedure type

The first dialog box asks for the type of procedure that is to be generated:



A number of different types of procedure generation are available. Each type is discussed below.

The check box at the bottom of the dialog box is only visible if advanced procedure building features are enabled. The check box enables editing of the 'Where' clause when no table joining is being performed, and hence the 'Where' clause would not be exposed.

Set based procedure

A set based procedure performs one SQL statement to join all the model and source tables together and then insert this data into the stage table. This is normally the fastest method of building the stage table. Caution must be taken in 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 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 345).

Set Distinct

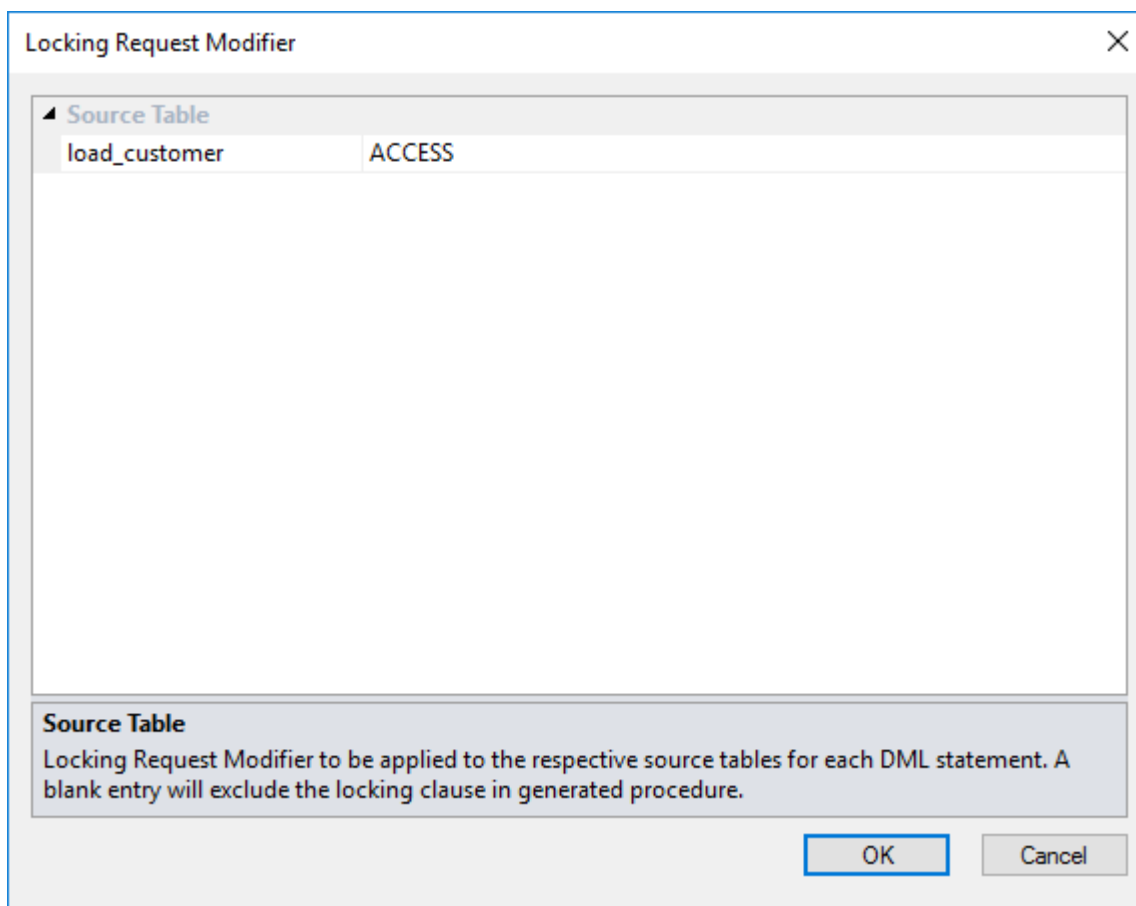
Essentially the same as Set, except for the DISTINCT key word being added to the SELECT statement. This option therefore removes duplicate rows.

Set Minus

The **Set Minus** option can be used to determine change data or for programmatic referential integrity checking. This option works in a similar way to Set Merge. It generates SQL code in this form: *SELECT ... FROM source_table1 {where} MINUS SELECT ... FROM source_table2 {where}*. It requires exactly two source tables to be specified. All source columns must exist in both source tables.

Locking Request Modifier

Source Table: Specify a locking request modifier to be applied to each source table during generated update procedures. By default this is set to 'ACCESS' which locks each row being accessed, a blank entry will result in no locking clause in the generated procedure.

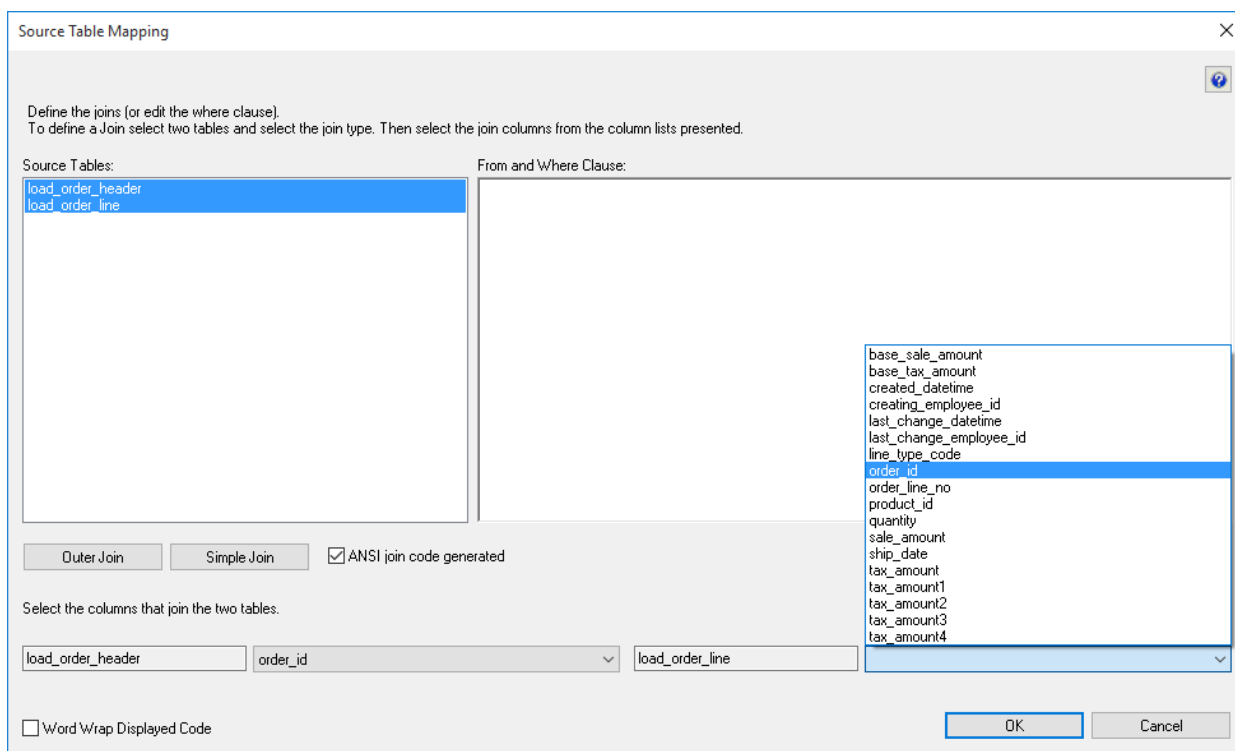


Source Table Mapping

If multiple source tables are present then the definition of the joins between the various tables is required. Note that at this stage we are not joining the model tables. These joins only relate to the source tables.

The joining of the tables will provide part of the construct of the set based update in the update procedure.

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. An additional option is available to allow either an ANSI standard join or a 'Where' clause based join. 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 or not they are found in the second table. Therefore, 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. 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 or not an order_header exists.

As the join columns are selected, the 'Where' statement is built up in the large edit window on the right side. Once all joins have been made, the contents of this window can be changed if the join statement is not correct.

Once satisfied with the 'Where' statement, click the **OK** button 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. It can of course be edited in the procedure that is generated if not correct.

You have the choice between 'Where' statement joins and ANSI standard joins.

Note: 'Where' joins are not available if using outer joins in Teradata.

The example below shows the result of an ANSI standard join which takes place in the 'From' statement.

Source Table Mapping

Define the joins (or edit the where clause).
To define a Join select two tables and select the join type. Then select the join columns from the column lists presented.

Source Tables:

- load_order_header
- load_address
- load_order_line

From and Where Clause:

```
FROM [TABLEOWNER].[load_order_header] load_order_header
JOIN [TABLEOWNER].[load_order_line] load_order_line
ON load_order_header.order_id = load_order_line.order_id
LEFT OUTER JOIN [TABLEOWNER].[load_address] load_address
ON load_order_header.sold_to_address_id = load_address.address_id
```

ANSI join code generated

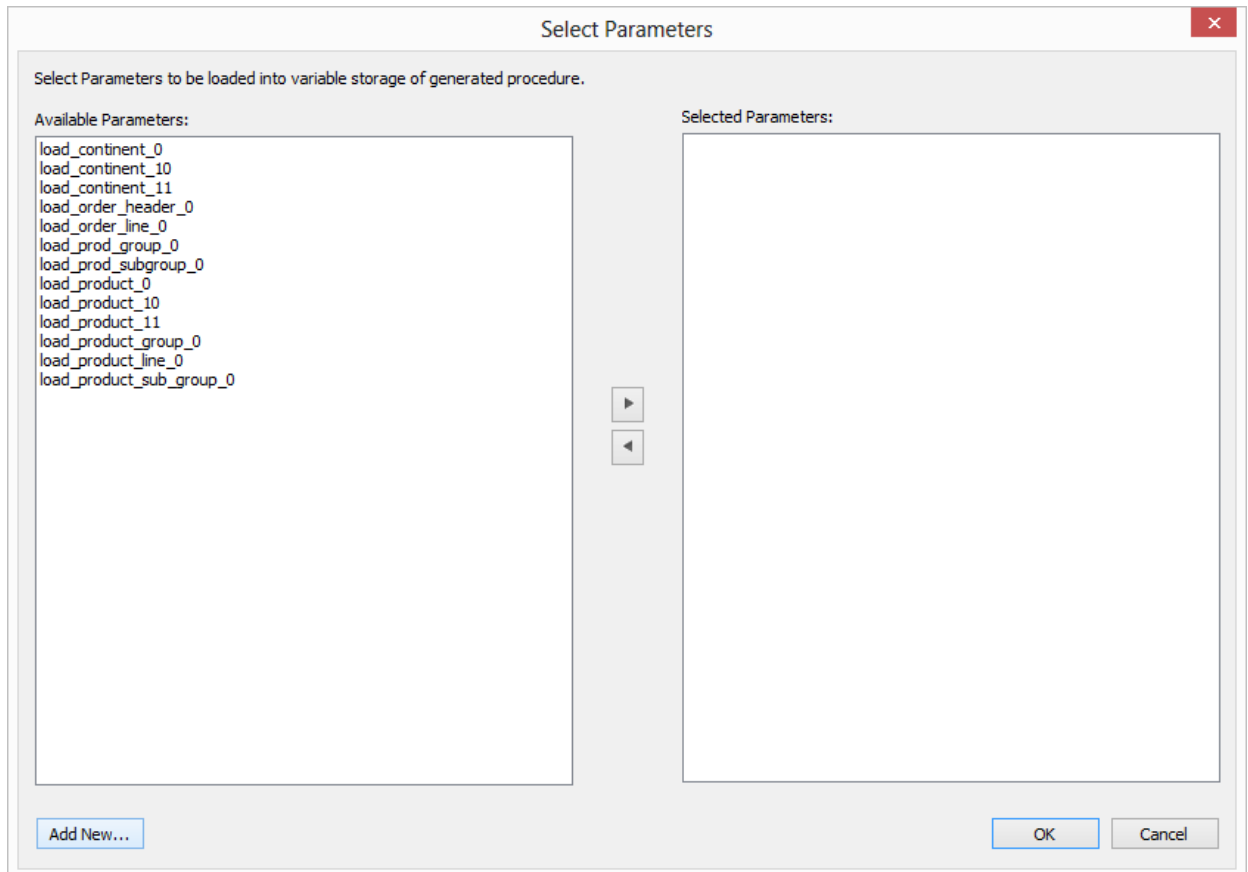
Select the columns that join the two tables. Select the column from the Master Table first.

load_order_header load_address

Word Wrap Displayed Code

Parameter selection

The parameter selection dialog is used for choosing WhereScape RED **parameters** to be included as variables in the stage table procedure. Each parameter chosen is included in the procedure as `v_parameter_name`, limited to the first 30 characters. For example, parameter **THE_DAY_OF_THE_WEEK_FOR_LOADING_MONTH_END_DATA** will be available in the procedure as `v_THE_DAY_OF_THE_WEEK_FOR_LOAD`. RED automatically declares this variable and assigns it the current value of the parameter every time the procedure is run. Parameter variables can be used in column transformations or 'Where' clauses. A sample dialog box follows:



Note: 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.

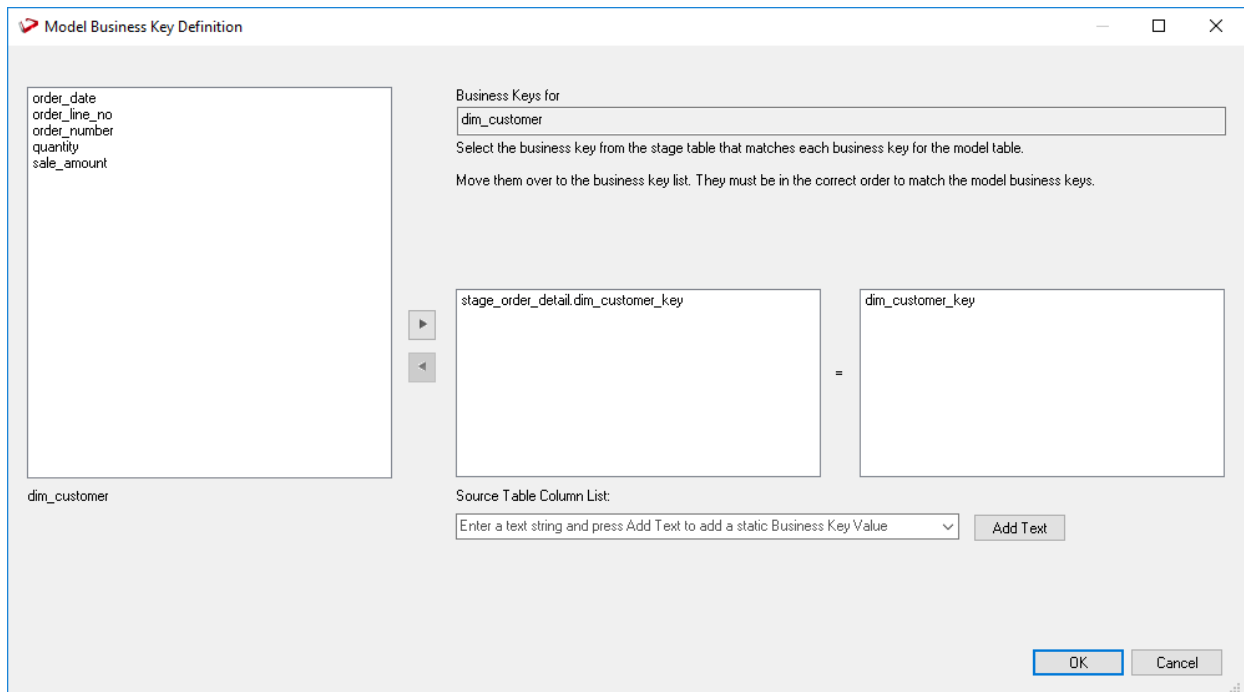
Model/Dimension Joins

For each model/dimension key a dialog will now appear asking for the business key from the stage table that matches the business key for the model/dimension.

In the example below, we are asked for the stage table business key for the customer dimension table. 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 **customer_code**.

We must provide the corresponding business key from the staging table. In the case of our example this is the **customer_code** column.



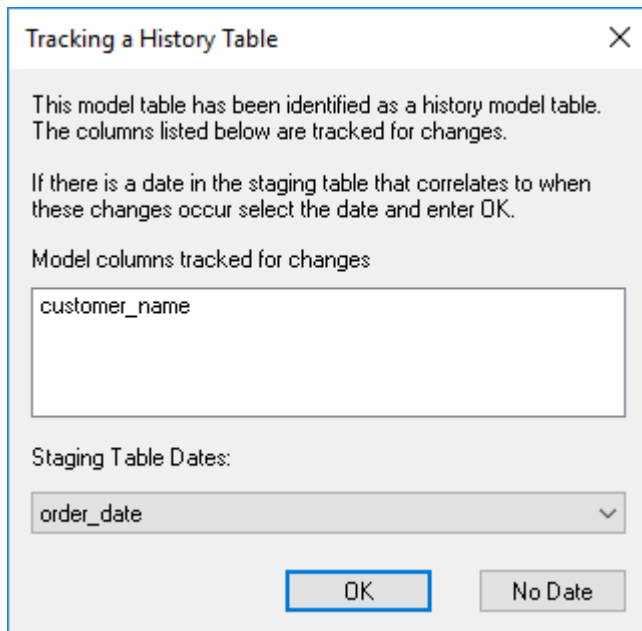
Note: The **Add Text** button and the associated message and edit box are only shown if the user possesses a full license, thus enabling the advanced procedure build options. When the **Add Text** button is clicked any data in the edit box to the right of the button is placed in the stage table column list. In this way a number or string can be assigned as part or all of a model join.

- 1 Click the **OK** button after the correct business key has been entered.
- 2 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.

Model history information

If the model table being joined was defined as a model history table, then an additional dialog will appear asking for a date in the model table that allows the coordination of the model record changes. This dialog asks for a date field in the model table that enables RED to determine which version of the tracked column (the **customer_name** field, below) to use based on the specified date field.

Select the appropriate date field for your business needs and click **OK**. If you wish to take the last (or current) version for the tracked column, select **No Date**.



The screenshot shows a dialog box titled "Tracking a History Table" with a close button (X) in the top right corner. The text inside the dialog reads: "This model table has been identified as a history model table. The columns listed below are tracked for changes." followed by "If there is a date in the staging table that correlates to when these changes occur select the date and enter OK." Below this, there is a section labeled "Model columns tracked for changes" with a text box containing "customer_name". Underneath is a section labeled "Staging Table Dates:" with a dropdown menu currently showing "order_date". At the bottom of the dialog are two buttons: "OK" and "No Date".

For Example:

As seen in the screen above, we have defined the **customer_name** as a column that we expect to have versions for over time i.e. each time the data warehouse processing sees a new **customer_name** value, the model table 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 record in the model table 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.

Building and Compiling the Procedure

- Once the above questions are completed the procedure is built and compiled automatically.
- If the compile fails, an error is 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.

STAGE TABLE CUSTOM PROCEDURE

A second procedure can be created on every stage table. This is called the custom procedure. Rather than modifying the generated procedure, it is often more practical to make additions to the generated code in a separate procedure. This allows for regeneration of the staging table's update procedure without losing changes (and having to reapply them).

The generated procedure for a custom procedure is template code. That is, a procedure that declares and initializes variables, does nothing and returns the correct return code and message for the WhereScape RED scheduler.

STAGE TABLE COLUMN PROPERTIES

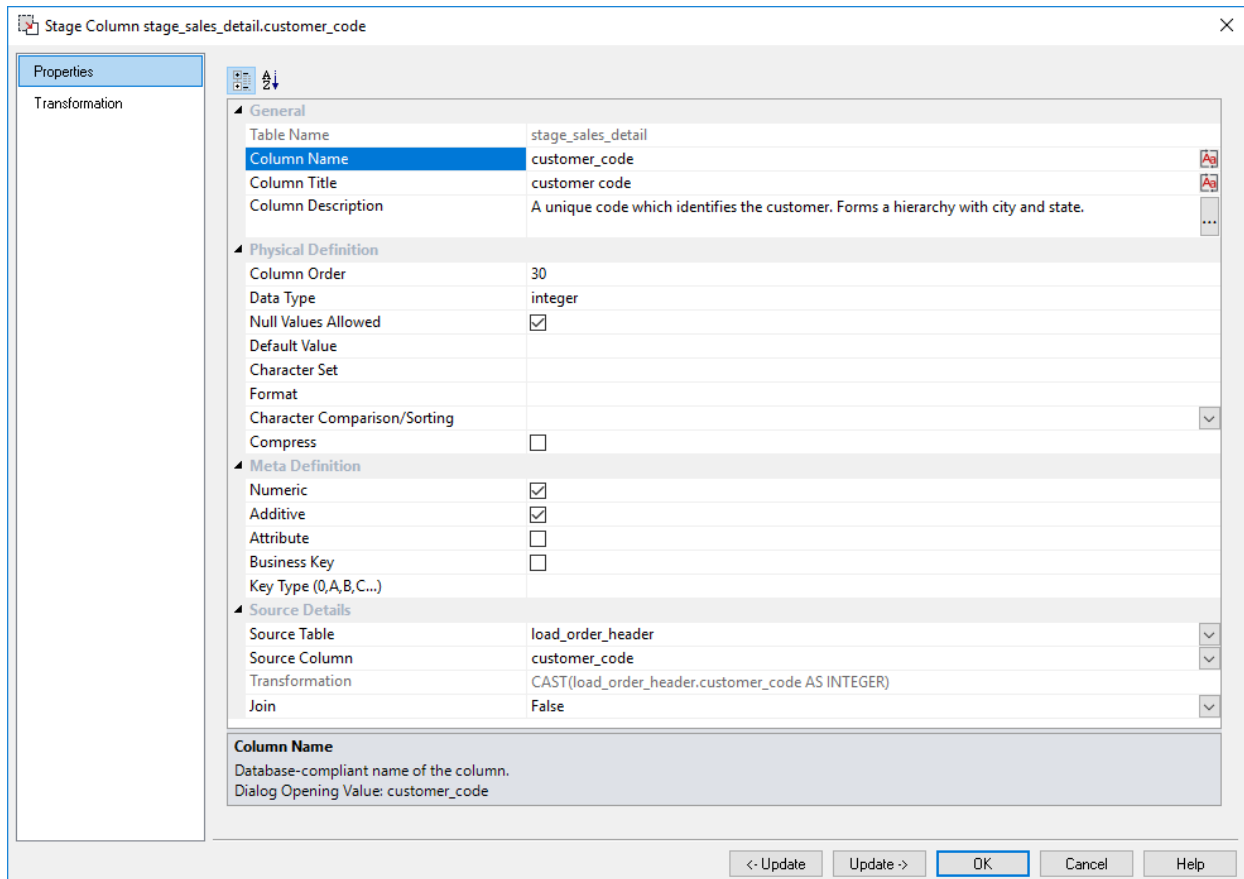
Each stage table column has a set of associated properties. The definition of each property is defined below:

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-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:



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. A good practice is to only use alphanumeric, 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.

Column Title

Name that the business uses to refer to the column. It does not affect the physical table definition, but rather provides input to the documentation and to the view `ws_admin_v_dim_col` which can be used to assist in the population of a end user tool's end user layer. 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 might 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 a valid for the target database. Typical Teradata databases often have integer, numeric(), varchar(), char(), date and timestamp data types. 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.

Character Set

Database-compliant table column character-set used for storage. Select Latin or Unicode.

Format

Database-compliant table column format. 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. 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.

Character Comparison/Sorting

Determines how the column character values are treated for comparison and sorting operations. Choose from: case specific, not case specific, uppercase case specific or uppercase not case specific.

Compress

Indicates whether the table column values are compressed when stored.

Compress/Compress Value

Optional list of values to be compressed. By default, only NULL is compressed if no list of values is specified.

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 an 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. This indicator is set and cleared by WhereScape RED during the dimension update procedure generation process. This checkbox 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. 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 model table join. Used on model tables to indicate the model keys to other model tables. Results in indexes being defined for the columns. Set during the update procedure generation for a model table, based on information from the staging table. |
| 3 | Not used in WhereScape RED for Teradata. |
| 4 | Not used in WhereScape RED for Teradata. |
| 5 | Indicates a column is a start date column. |
| 6 | Indicates a column is a end date column. |
| 7 | History column indicator. Used on model history tables to indicate that the column is being managed as a history column within the context of a model history table. Set when a column is identified during the model history update procedure generation. |
| c | Change Hash Key column indicator. Used in Data Vault tables to indicate the differences in the descriptive columns of a Satellite table. Refer to Data Vaults (on page 402) for details. |
| h | Hub Hash Key column indicator. Used in Data Vault tables to indicate the hash key column of a Hub Table. Refer to Data Vaults (on page 402) for details. |
| l | Link Hash Key column indicator. Used in Data Vault tables to indicate the hash key column of a Link Table. Refer to Data Vaults (on page 402) for details |
| 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. |

Key Sources

This field is only displayed for Hash key types. Displays the hash source columns that are used to generate the selected Hub, Link or Change hash key.

Key Source For

This field is only displayed for Hash key types. Displays the hash keys columns that use the displayed hash key sources.

Source Table

Identifies the source table where the column's data comes from. This source table is normally a load table or a model 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. This may also be a model table key where a model 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.

The transformation screen is as follows:

Note: Transformations are only put into effect when the procedure is re-generated.

See **Transformations** (on page 593) for more details.

PERMANENT STAGE TABLES

Normally stage tables have all data removed from them at the start of their update procedure.

Sometimes it's necessary to hold more than a single set of data in a stage table. Permanent stage tables allow this.

By default, a permanent stage table update does not start with all data being removed. However, options are available to selectively remove data.

For example, a permanent stage table may be used to hold the last three months source data provide functionality to remove all data for a day if new data for that day arrives.

GENERATING THE PERMANENT STAGING UPDATE PROCEDURE

Once a permanent stage table has been defined in the metadata and created in the data base an update procedure can be generated to handle the joining of any tables.

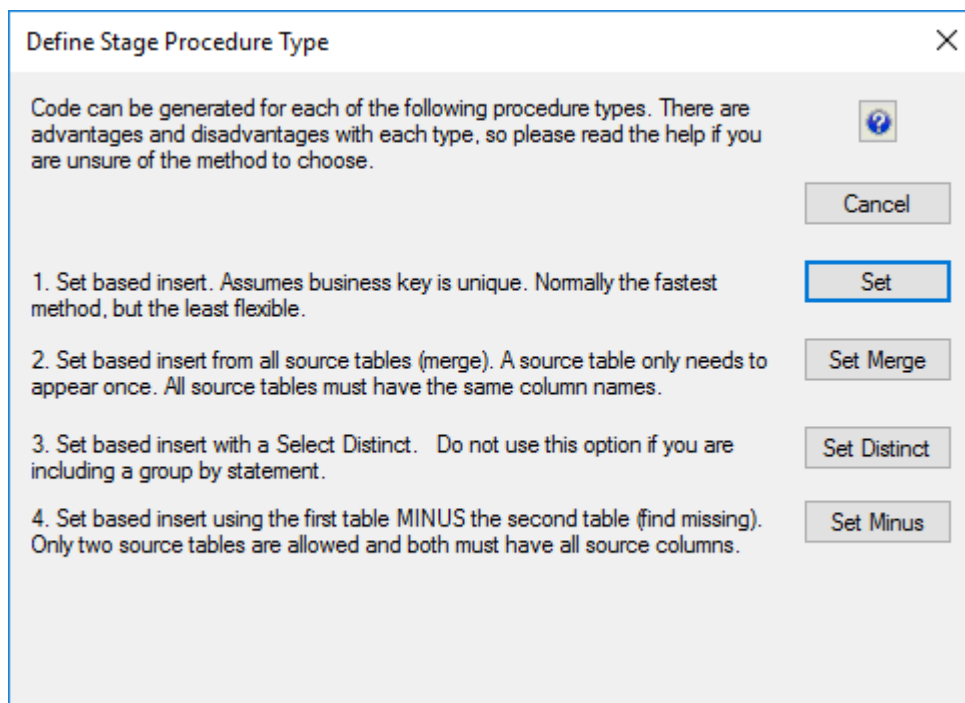
Note: You can also generate an update procedure via a template, refer to *Rebuilding Update Procedures* (on page 181) for details.

Generating a Procedure

- 1 To generate a procedure, right-click on the permanent stage table name in the left pane and select **Properties**.
- 2 From the **Update Procedure** drop-down list select (**Build Procedure...**).
- 3 Click **OK** to update the properties and start the process of generating the new procedure.
- 4 A series of prompts are displayed during the procedure generation to join the tables and link in the model tables.

Procedure type

The first dialog box asks for the type of procedure that is to be generated:



A number of different types of procedure generation are available. Each type is discussed below. A check box appears at the bottom of the dialog if advanced procedure building features are enabled in the **Tools>Options** screen. This check box enables the editing of the 'Where' clause when no table joining is being performed, and hence the 'Where' clause would not be exposed.

Set based procedure

A set based procedure performs one SQL statement to join all the source tables together and then insert this data into the stage table. This is normally the fastest method of building the stage table. Caution must be taken in 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 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.

Set Distinct

Essentially the same as Set, except for the DISTINCT key word being added to the SELECT statement. This options therefore removes duplicate rows.

Set Minus

The **Set Minus** option can be used to determine change data or for programmatic referential integrity checking. This option works in a similar way to Set Merge. It generates SQL code in this form: *SELECT ... FROM source_table1 {where} MINUS SELECT ... FROM source_table2 {where}*. It requires exactly two source tables to be specified. All source columns must exist in both source tables.

Business Key definition

A dialog will appear asking for the business key that will uniquely identify each permanent stage record.

The source table from which the permanent stage is derived would normally have some form of unique constraint applied. In most cases this will be the business key.

In the example below **order_number** and **order_line_no** are selected as the business key.

The screenshot shows a dialog box titled "Define staging Business Key Column(s)". It has a close button (X) in the top right corner. The dialog is divided into two main sections. On the left, there is a "Column List" containing the following items: product_code, quantity, sales_value, tax, and unit_sale_price. On the right, there is a "Business Key List" containing order_line_no and order_number. Between these two lists are two arrow buttons: a right-pointing arrow (to move a column from the left list to the right list) and a left-pointing arrow (to move a column from the right list back to the left list). Above the Business Key List, there is a text instruction: "Select the business keys that uniquely identify each record in the staging table. Move them over to the business key list". Below the Business Key List, there is a checkbox labeled "Include Delete Before Insert" which is currently unchecked. At the bottom right of the dialog, there are two buttons: "OK" and "Cancel".

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 permanent stage, 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.

NULL Values: none of the columns chosen as the business key should ever contain a NULL value. See the note at the start of this chapter.

The **Include delete before insert** check will display an additional dialog wizard for building the 'Where' clause of the delete statement; at the start of the update procedure.

Definition for a Delete Statement Before the Stage Table Insert

Select the permanent stage table column on which the delete will be based. All values in the source (load or stage) table for the selected column will be used to delete the corresponding records in the permanent stage table.

Permanent Stage Table Column: Source (load or stage) Table: Source (load or stage) Table Column:

Issue Warning if a Delete Occurs

Truncate Table Before Insert

Delete Where Clause: The Where Clause can be edited, or directly entered.

```
WHERE stage_order_line.order_number IN ( SELECT load_order_line.order_number FROM [load_order_line]
```

Word Wrap Displayed Code

OK Cancel

Note: If a business key is specified, the generated procedure includes both the update and insert statement; otherwise only an insert statement is supplied.

Source table joins

If multiple source tables are present then the definition of the joins between the various tables is required.

The joining of the tables will provide part of the construct of the set based update in the update procedure.

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. An additional option is available to allow either an ANSI standard join or a 'Where' clause based join. The ANSI standard join should be chosen in most situations. See the example screen in the following section.

Source Table Mapping

Define the joins (or edit the where clause).
To define a Join select two tables and select the join type. Then select the join columns from the column lists presented.

Source Tables:

- load_order_header
- load_order_line

From and Where Clause:

```
FROM [TABLEOWNER].[load_order_header] load_order_header
JOIN [TABLEOWNER].[load_order_line] load_order_line
ON load_order_header.order_id = load_order_line.order_id
```

Outer Join Simple Join ANSI join code generated

Select the columns that join the two tables:

load_order_header load_order_line

Word Wrap Displayed Code

OK Cancel

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 or not they are found in the second table. Therefore, 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. 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 or not 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.

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. It can of course be edited in the procedure that is generated if not correct.

You have the choice between 'Where' statement joins and ANSI standard joins.

Note: 'Where' joins are not available if using outer joins in Teradata.

The example below shows the result of an ANSI standard join which takes place in the 'From' statement.

Source Table Mapping

Define the joins (or edit the where clause).
To define a Join select two tables and select the join type. Then select the join columns from the column lists presented.

Source Tables:

- load_order_header
- load_order_line

From and Where Clause:

```
FROM [TABLEOWNER].[load_order_header] load_order_header  
LEFT OUTER JOIN [TABLEOWNER].[load_order_line] load_order_line  
ON load_order_header.order_id = load_order_line.order_id
```

Outer Join Simple Join ANSI join code generated

Select the columns that join the two tables. Select the column from the Master Table first.

load_order_header load_order_line

Word Wrap Displayed Code

OK Cancel

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.

SET MERGE PROCEDURE

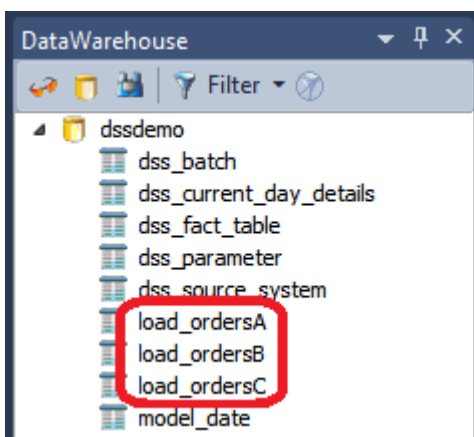
The set merge procedure type allows the merging of two or more identical tables. The tables to be merged must have exactly the same number of columns and column names.

If necessary additional blank columns could be added to one or other of the tables to make them identical.

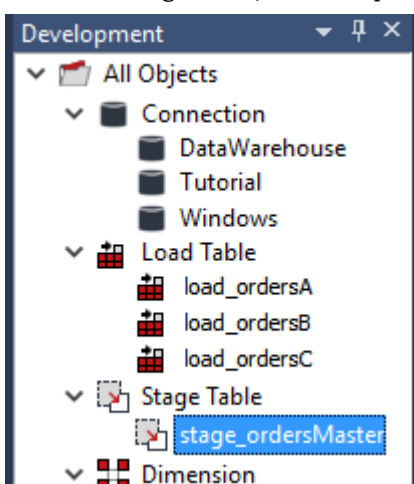
To use this procedure you must simply have the tables to be merged mentioned at least once in the **Source Table** field of a columns Properties.

In this example, we will merge three load tables into a single stage table.

- 1 The browser pane shows the three load tables:








- 1 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.
- 2 Name the stage table, for example **stage_ordersMaster**.

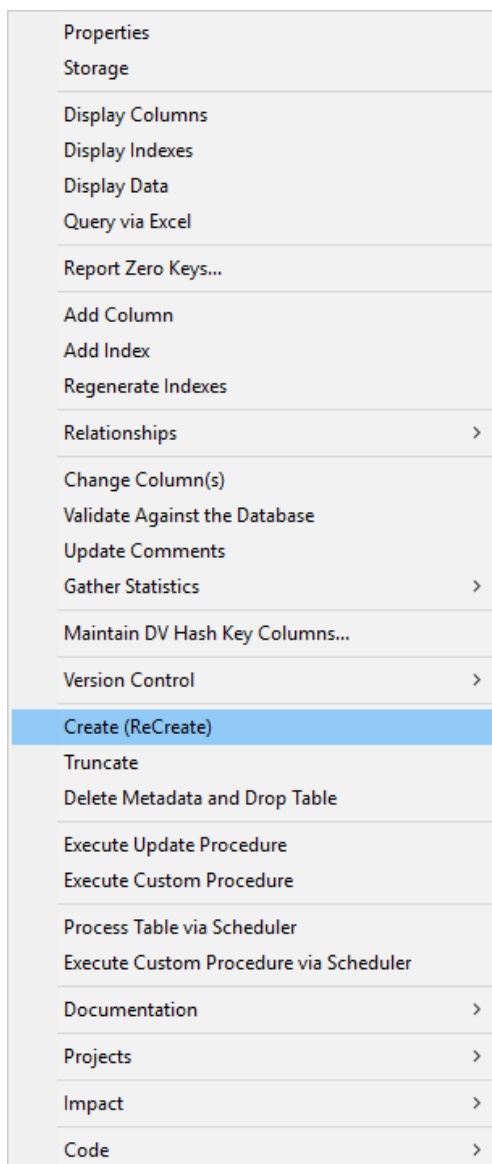


- 3 Next, modify the source table column to include one instance of each of the three load tables; the order does not matter.

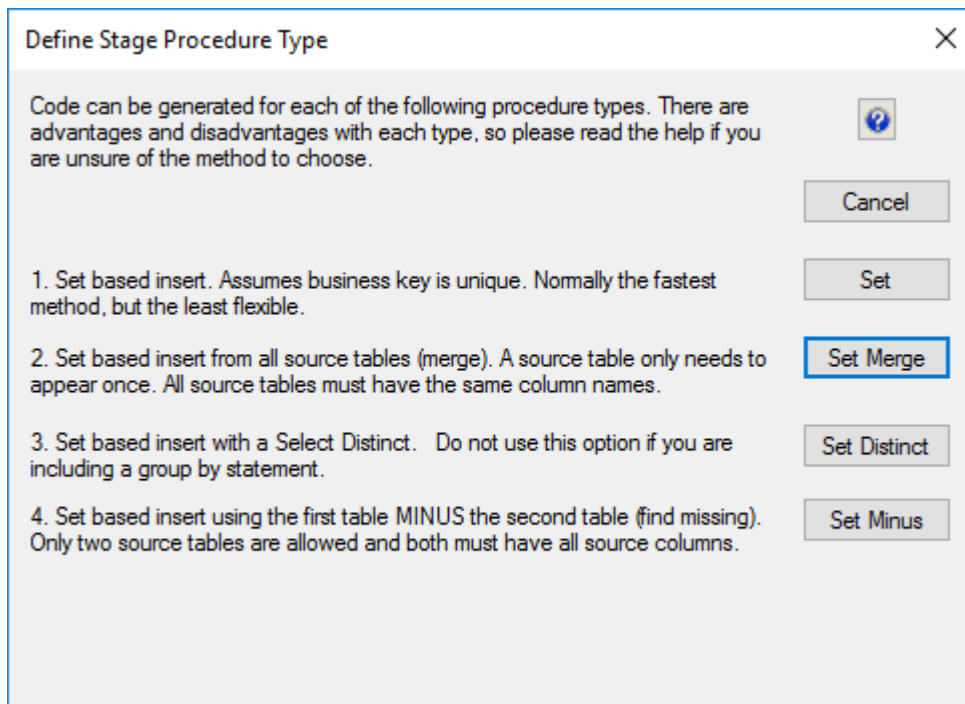
You can do this either by typing in the table directly, or by going to each of the column's properties screen.

| stage_ordersMaster Columns | | | |
|---|-----------------|-------------|--------------|
| Column Name | Display Name | Data Type | Source Table |
|  order_number | order number | numeric(12) | load_ordersA |
|  order_date | order date | datetime | load_ordersB |
|  customer_code | customer code | numeric(6) | load_ordersC |
|  ship_date | ship date | datetime | load_ordersA |
|  dss_update_time | dss update time | datetime | |

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:

```

CREATE PROCEDURE [METABASE].update_stage_ordersMaster
(
  IN   p_sequence      integer,
  IN   p_job_name      varchar(256),
  IN   p_task_name     varchar(256),
  IN   p_job_id        integer,
  IN   p_task_id       integer,
  OUT  p_return_msg    varchar(256),
  OUT  p_status        integer
)
BEGIN
  -----
  -- DBMS Name       : Teradata
  -- Script Name     : update_stage_ordersMaster
  -- Description     : Build the table stage_ordersMaster
  -- Generated by    : Version 6.5.5.2 (build 111221)
  -- Generated for   : Testing, Tutorials and Documentation
  -- Generated on    : Wednesday, February 08, 2012 at 13:52:09
  -- Author         : QuickStart
  -----
  -- Notes / History
  --
  -----
  -- Control variables used in most programs
  -----
  DECLARE v_msgtext      varchar(255); -- Text for audit_trail
  DECLARE v_sql          varchar(255); -- Text for SQL statements
  DECLARE v_set          integer;     -- commit set
  DECLARE v_analyze_flag integer;     -- analyze flag
  DECLARE v_step         integer;     -- return code
  DECLARE v_update_count integer;     -- no of records updated
  DECLARE v_insert_count integer;     -- no of records inserted
  DECLARE v_count        integer;     -- General counter
  DECLARE v_sql_code     integer;     -- SQL Error Code for Audit Trail
  DECLARE v_sql_error    varchar(255); -- SQL Error Code for Audit Trail as varchar
  -----
  -- Variables
  -----
  DECLARE v_dss_update_time timestamp; -- Used for date insert
  -----
  -- Exceptions
  -----
  DECLARE EXIT HANDLER
  FOR SQLEXCEPTION
  BEGIN
    SET v_sql_code = SQLCODE;
    LOCK ROW FOR ACCESS
    SELECT SUBSTR(ErrorText,1,255)
    INTO   :v_sql_error
    FROM   dbc.ErrorMsgs
    WHERE  ErrorCode = :v_sql_code;
    SET v_msgtext = SUBSTR('Unhandled Exception in stage_ordersMaster. ' ||
      'Step ' || CAST(v_step AS VARCHAR(64)) ||
      ' SQL Error Code: ' || CAST(v_sql_code AS VARCHAR(10)) || ' - ' || v_sql_error,1,255);
    SET p_return_msg = v_msgtext;
    CALL [METABASE].WsWrkAudit('F', :p_job_name, :p_task_name, :p_sequence
      , :v_msgtext, :v_sql_code, :v_sql_error, :p_task_id, :p_job_id);
    SET p_status = -3;
  END;

```

```
-----  
-- Main  
-----  
SET v_step = 100;  
SET v_dss_update_time = CURRENT_TIMESTAMP;  
SET v_update_count = 0;  
SET v_insert_count = 0;  
  
-----  
-- Delete all existing data from the table  
-----  
DELETE FROM [stage_ordersMaster] ALL;  
  
SET v_step = 200;  
  
-----  
-- Insert input records into stage_ordersMaster  
-- from load_ordersA  
-----  
INSERT INTO [stage_ordersMaster]  
( order_number  
 , order_date  
 , customer_code  
 , ship_date  
 , dss_update_time  
 )  
SELECT load_ordersA.order_number  
 , load_ordersA.order_date  
 , load_ordersA.customer_code  
 , load_ordersA.ship_date  
 , :v_dss_update_time  
FROM [load_ordersA] load_ordersA  
;  
  
SET v_insert_count = v_insert_count + ACTIVITY_COUNT;  
  
SET v_step = 300;  
  
-----  
-- Insert input records into stage_ordersMaster  
-- from load_ordersB  
-----  
INSERT INTO [stage_ordersMaster]  
( order_number  
 , order_date  
 , customer_code  
 , ship_date  
 , dss_update_time  
 )  
SELECT load_ordersB.order_number  
 , load_ordersB.order_date  
 , load_ordersB.customer_code  
 , load_ordersB.ship_date  
 , :v_dss_update_time  
FROM [load_ordersB] load_ordersB  
;  
  
SET v_insert_count = v_insert_count + ACTIVITY_COUNT;  
  
SET v_step = 400;
```

```
-----  
-- Insert input records into stage_ordersMaster  
-- from load_ordersC  
-----  
INSERT INTO [stage_ordersMaster]  
( order_number  
 , order_date  
 , customer_code  
 , ship_date  
 , dss_update_time  
 )  
SELECT load_ordersC.order_number  
 , load_ordersC.order_date  
 , load_ordersC.customer_code  
 , load_ordersC.ship_date  
 , :v_dss_update_time  
FROM [load_ordersC] load_ordersC  
 ;  
  
SET v_insert_count = v_insert_count + ACTIVITY_COUNT;  
  
SET v_step = 500;  
  
-----  
-- All Done report the results  
-----  
-- Work out the return message  
SET p_status = 1;  
SET v_msgtext = 'stage_ordersMaster updated. '  
 || CAST(v_insert_count AS VARCHAR(64)) || ' new records. '  
 || CAST(v_update_count AS VARCHAR(64)) || ' records updated.';  
SET p_return_msg = v_msgtext;  
  
END;
```

CHAPTER 12

DATA STORE OBJECTS

IN THIS CHAPTER

| | |
|---|-----|
| Data Store Objects Overview..... | 352 |
| Building a Data Store Object..... | 354 |
| Generating the Data Store Update Procedure..... | 359 |
| Data Store Artificial Keys..... | 367 |
| Data Store Column Properties..... | 371 |
| Data Store Column Transformations..... | 376 |

DATA STORE OBJECTS OVERVIEW

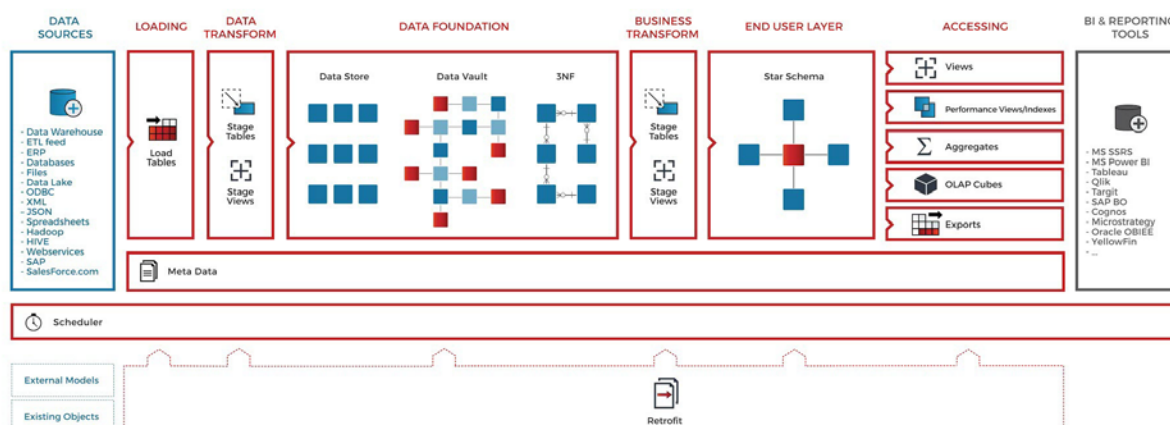
A Data Store Object is a data warehouse object used to store any type of data for later processing. In WhereScape RED, Data Store objects have many of the code generating attributes of stage, dimension and fact tables. Data Store objects can be thought of as a source system for the data warehouse. Alternatively, 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 986) for more details.

The usual steps for creating a Data Store model are defined below and are covered in this chapter. The steps are:

- 1 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.
- 2 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 354).
- 3 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 .
- 4 Add columns from other load and/or stage tables if required. See **Building a Data Store Object** (on page 354).
- 5 Create the Data Store Object in the database. See **Building a Data Store Object** (on page 354).
- 6 Build the update procedure. See **Generating the Data Store Update Procedure** (on page 359).
- 7 Run the update procedure and analyze the results.

If necessary, modify the update procedure or create a custom procedure.



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:

- `COALESCE(business_key,'N/A')`

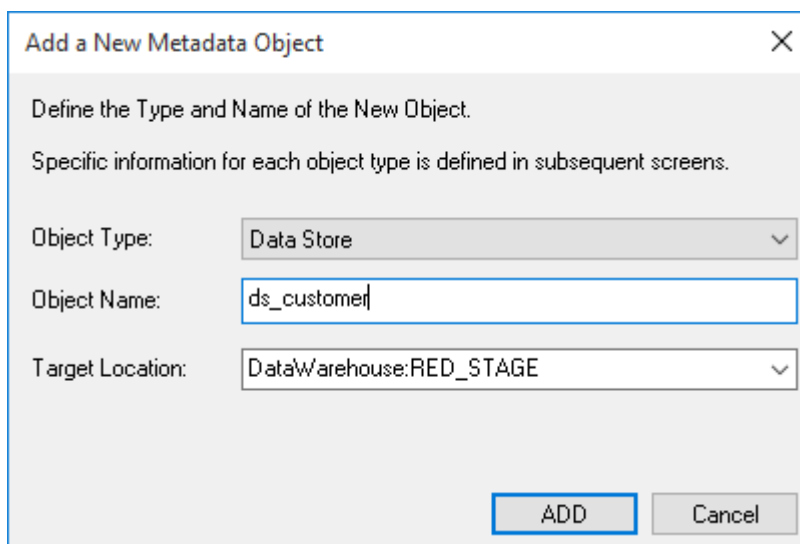
Note: Business keys are assumed to never be Null. If they can be null it is best to transform them to some value prior to the Data Store or stage table update. If this is not done, an un-modified update procedure will probably fail with a duplicate key error on the business key index.

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

- 1 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 this list is displayed in the middle pane, the pane is identified as a target for new Data Store Objects.
- 2 Browse to the Data Warehouse via the **Browse Connection** menu option.
- 3 Drag the load or stage table that contains the bulk of the Data Store Object columns into the middle pane.
- 4 Drop the table anywhere in the pane. The new object dialog box will appear identifying the new object as a Data Store Object and providing a default name based on the load or stage table name.
- 5 Either accept this name or enter a name for Data Store Object and click **ADD** to proceed:



Add a New Metadata Object [X]

Define the Type and Name of the New Object.
Specific information for each object type is defined in subsequent screens.

Object Type: Data Store [v]

Object Name: ds_customer

Target Location: DataWarehouse:RED_STAGE [v]

[ADD] [Cancel]

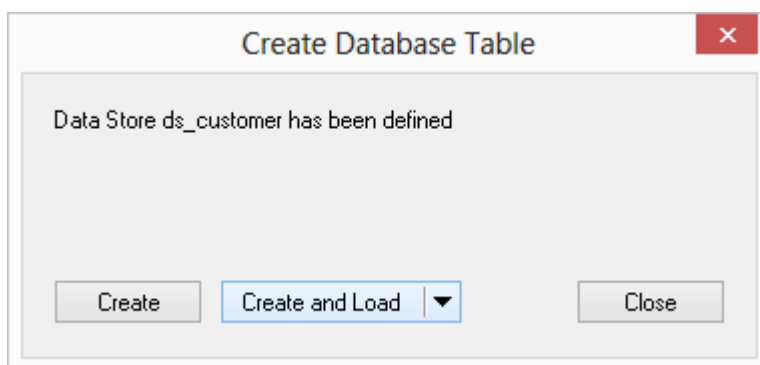
Data Store Object Properties

The table properties dialog for the new table displays.

- 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** 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 just a couple of steps.
- If you want to do this, select the **(Build Procedure...)** option from the **Update Procedure** drop-down, and click **Create and Load** to the next screen.

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.

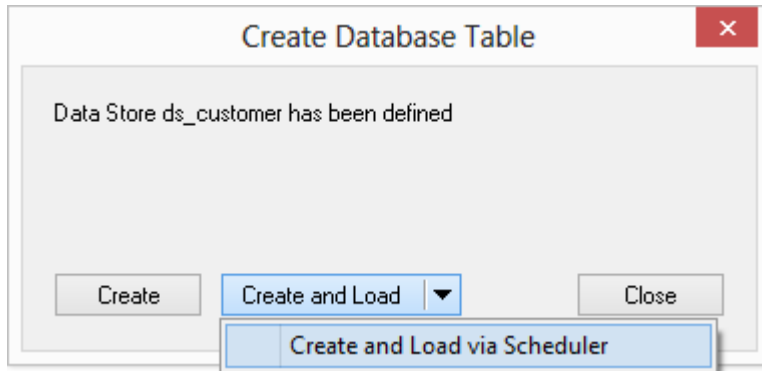


If you are satisfied with the columns that will be used and do not wish to add any columns you can select the **Create and Load** button. Alternatively, the **Create** button creates the table in the repository but does not execute an update, allowing you to change columns before loading data into the table.

If **Create** or **Create and Load** is selected and a new procedure creation was chosen proceed directly to the **Generating the Data Store Update Procedure** (on page 359) section.

If you have additional columns to add or columns to delete, click **Close** and proceed as below (Deleting and Changing Columns).

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 the **Delete** key.
- You can change a column name 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 *Data Store Column Properties* (on page 371) for a full description of fields.



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 Data Store Object columns displayed in the middle pane, this pane is considered a drop target for additional columns.

- It is simple 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 are 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 are available. |

| Column name | Description |
|-----------------------|---|
| 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. |
| 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 |

Create the table

Once the Data Store Object has been defined in the metadata you need to physically create the table in the database.

- 1 To do this, right-click on the Data Store Object name and select **Create (ReCreate)** from the pop-up menu.
- 2 The Results pane will display the results of the creation: 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.
- 3 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 of finding such a column is to double click on the list heading **Col name**. This will sort the column names into alphabetic order.
- 4 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 359).

GENERATING THE DATA STORE UPDATE PROCEDURE

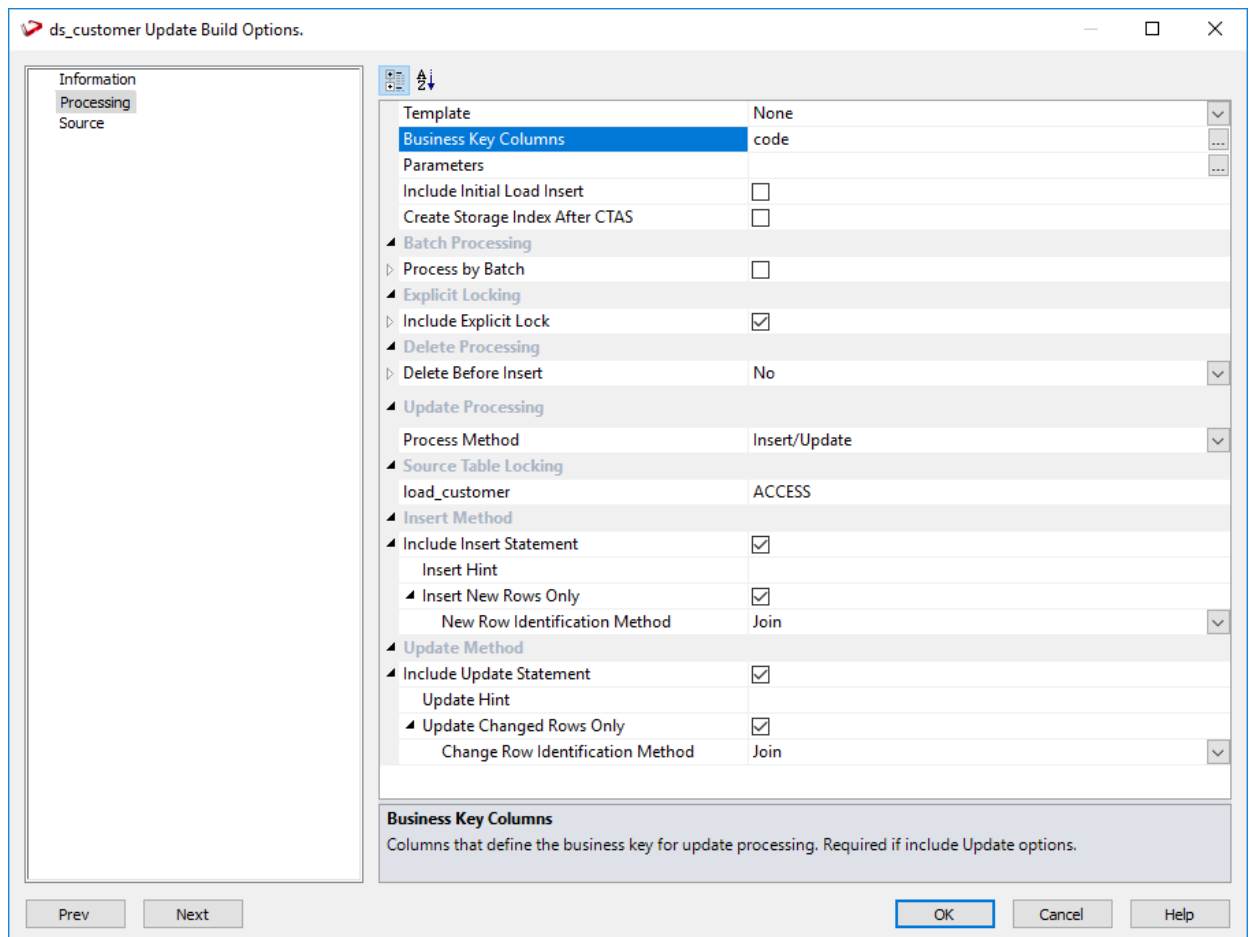
Once a Data Store Object has been defined in the metadata and created in the database, an update procedure can be generated to handle the joining of any tables and the update of the Data Store Object.

Note: You can also generate an update procedure via a template, refer to *Rebuilding Update Procedures* (on page 181) for details.

Generating a Procedure

- 1 To generate a procedure, right-click on the Data Store Object in the left pane and select **Properties**.
- 2 Click on the **Rebuild** button to start the process of generating the new procedure.
- 3 A series of options are available.

PROCESSING TAB



Template: Enables you to generate update procedures via a *template* (see "*Rebuilding Update Procedures*" on page 181).

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.



TIP: Use the column name **ascending/descending** buttons to sort column names. To revert to the metadata column order, click on the **meta column order** button.

Business Key Columns

Columns that define the business key for update processing. Required if include Update options.

Available Columns: A-Z Z-A ▶

- customer_name
- customer_legal_name
- territory_id
- ship_to_address_id
- bill_to_address_id
- sold_to_address_id
- primary_address_type
- primary_contact_person
- active_flag
- customer_category_code
- customer_category_description
- customer_group_code
- customer_group_description
- customer_subgroup_code
- customer_subgroup_description
- creating_employee_id
- created_datetime
- last_change_employee_id
- last_change_datetime

Selected Columns:

customer_code

▶

◀

▲

▼

OK Cancel

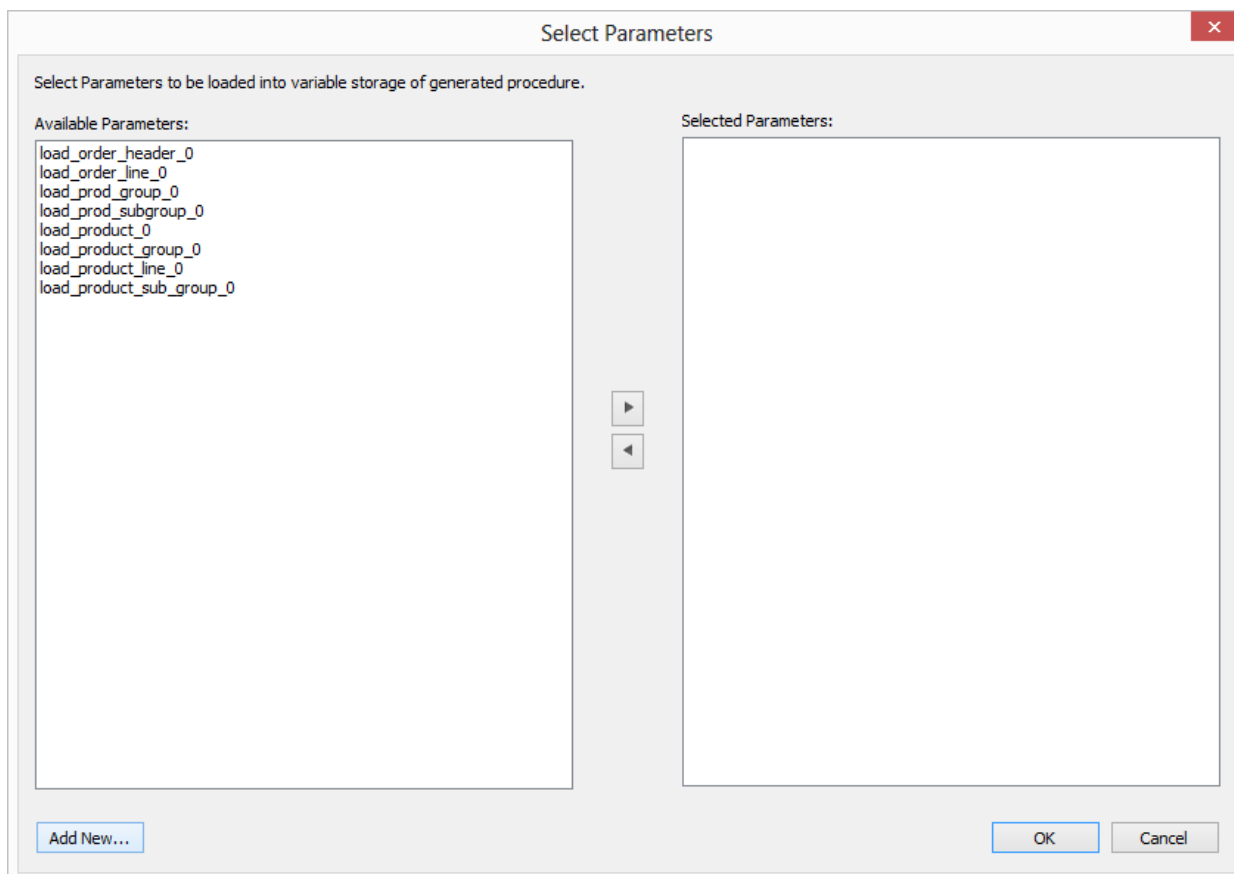
A business key can be made up of multiple columns, but it must provide a unique identifier. Where multiple columns separately uniquely identify rows in the Data Store Object, choose one to act as the primary business key.

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.

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 opens 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 132) for more information on WhereScape RED Parameters

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 not set (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 Business Key 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 not set (do not do batch processing).

Delete Before Insert: allows selecting 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. When this option is selected, it enables the **Issue Warning if a Delete occurs** and the **Delete Where Clause** Fields.

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: allows updating the Dimension with either an **Insert/Update** or a **Merge** statement. **Merge** allows you to use one Merge statement instead of two separate Insert and update statements.

Source Table Locking: allows a locking request modifier to be specified for each source table. The specified locking request modifier is applied to each source table during generated update procedures. By default this is set to 'ACCESS' which locks each row being accessed, a blank entry will result in no locking clause in the generated procedure.

Insert Method

Include Insert Statement: set this field to include the insert statement in the procedure. This allows inserting new rows in the Data Store.

Insert New Rows Only: uses change detection to work out which rows will require inserting.

New Row Identification Method: method used to identify that records in source are not currently recorded in the target table. Select **Join** or **Minus**.

Include Update Statement: set this field to include an update statement in the procedure. This allows updating the changing rows in the Data Store. If this is set, the **Update Changed Rows Only** option is available.

Update Changed Rows Only: uses change detection to work out what rows require updating. When set, 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. Select **Join** or **Minus**.

Merge Method

Merge Changed Rows only: uses change detection to work out what rows require merging. When the option is set, it enables the **New Row Identification Method**.

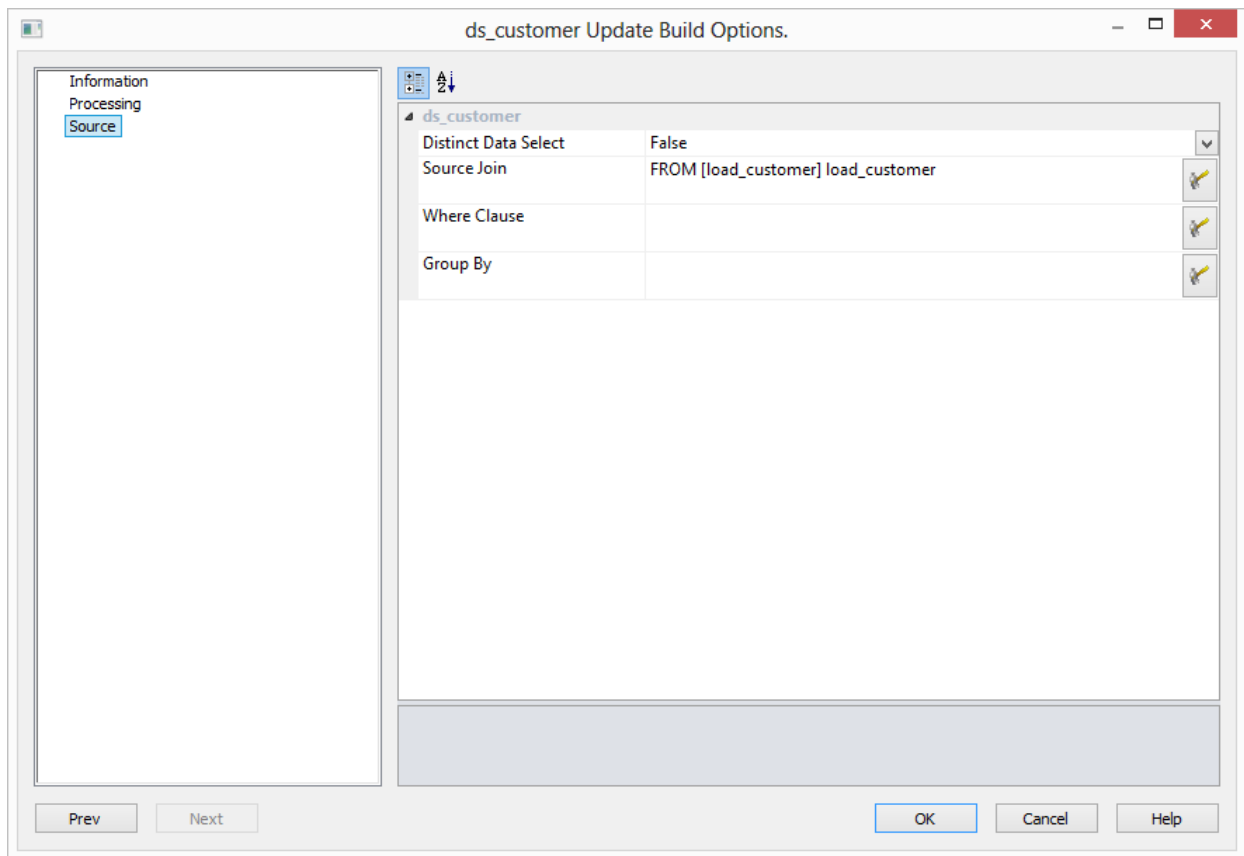
New Row Identification Method: method used to identify which records in the source are not recorded or are recorded differently in the target table. Select between **Join** and **Minus**.

If **non identity columns** are used as artificial keys the only new row identification method is **Join**.



Data Store Update procedures usually perform faster when you use the **Join** method for new row identification.

SOURCE TAB



Distinct Data Select: ensures duplicate rows are not added to the Data Store. This is achieved by adding the word DISTINCT 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.

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 or not they are found in the second table. Therefore, 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 or not a corresponding load_order_line entry exists.

Where Clause: The Where clause. Use as a filter to extract only the necessary records that fulfill a specified criteria.

Group By: The Group By clause. Use in collaboration with the SELECT statement to arrange identical data into groups.

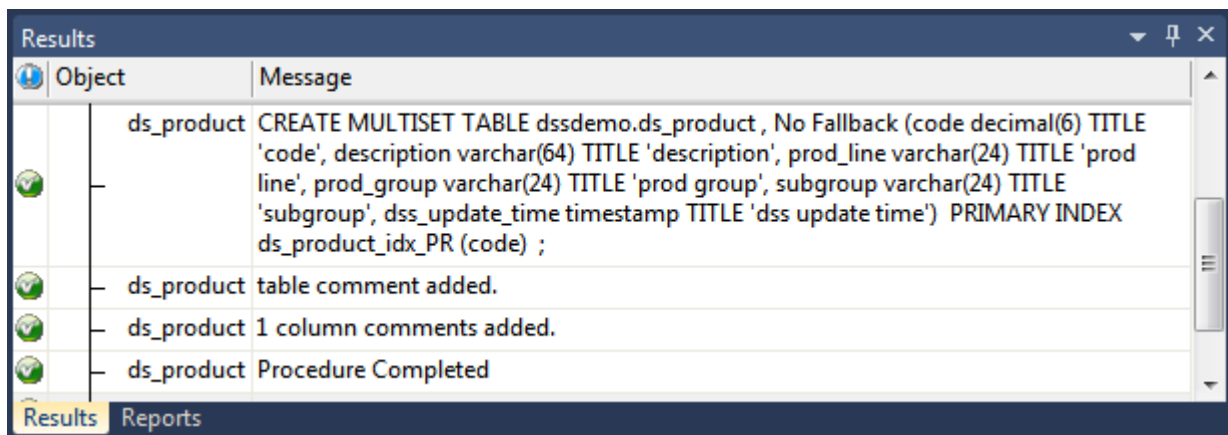
Building and Compiling the Procedure

- 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:



The screenshot shows a 'Results' window with a table containing four rows of execution messages. The first row shows the successful execution of a CREATE MULTiset TABLE statement. The subsequent three rows show messages indicating that a table comment was added, one column comment was added, and the procedure completed successfully.

| Object | Message |
|------------|---|
| ds_product | CREATE MULTiset TABLE dssdemo.ds_product , No Fallback (code decimal(6) TITLE 'code', description varchar(64) TITLE 'description', prod_line varchar(24) TITLE 'prod line', prod_group varchar(24) TITLE 'prod group', subgroup varchar(24) TITLE 'subgroup', dss_update_time timestamp TITLE 'dss update time') PRIMARY INDEX ds_product_idx_PR (code) ; |
| ds_product | table comment added. |
| ds_product | 1 column comments added. |
| ds_product | Procedure Completed |

Additional indexes can be added, or these indexes changed. See the chapter on indexes for further details.

DATA STORE ARTIFICIAL KEYS

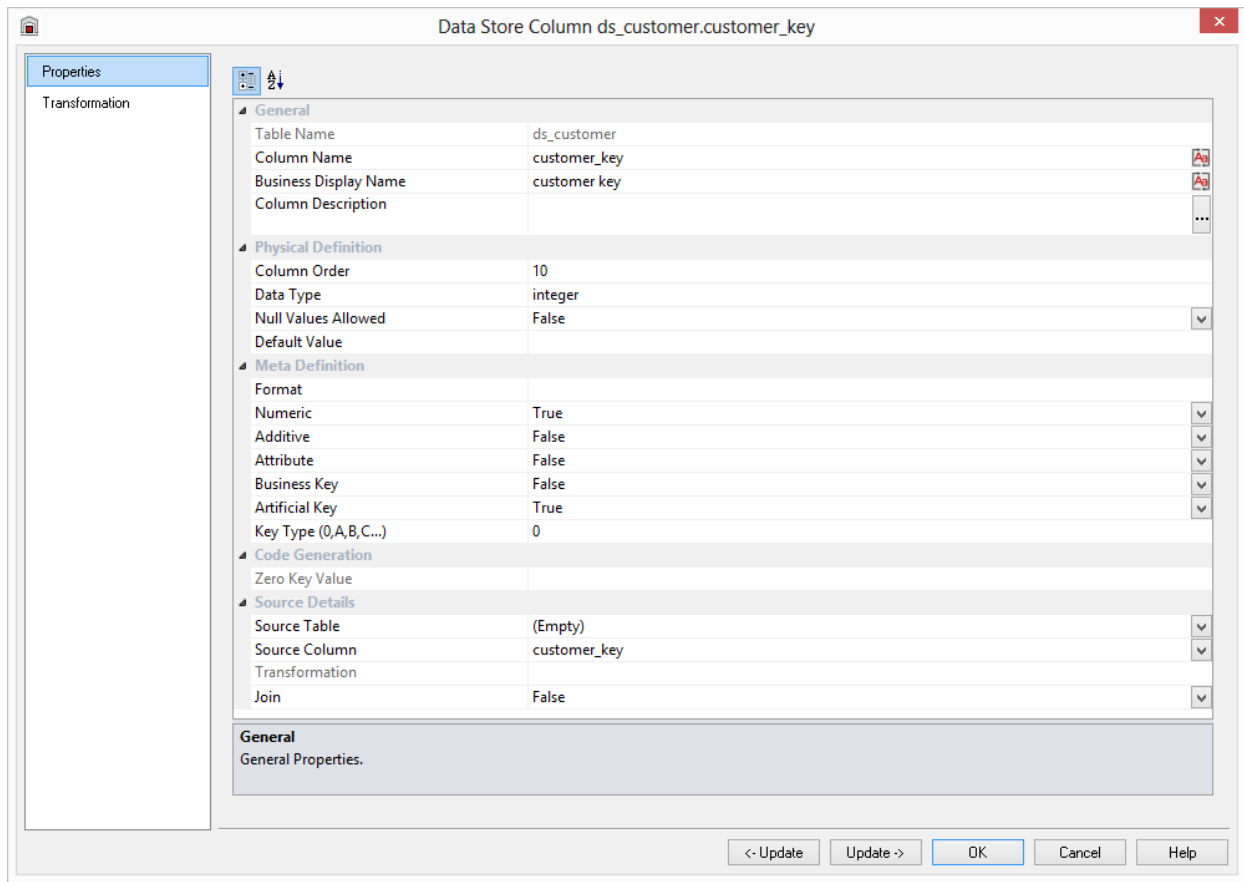
By default, Data Store Objects in WhereScape RED do not have an artificial (surrogate) key. Artificial keys can be added manually, but needing to do so could indicate Data Store Objects are not the correct WhereScape RED object for this table (perhaps an EDW 3NF Table would be more appropriate).

There is also an option for artificial keys to be automatically added to Data Store tables through an option in the **Tools** menu (see below - *Allowing for non identity surrogate keys on Data Store tables*).

To manually add an extra artificial key column to a Data Store table:

- 1 Right click in the middle pane and click either **Add Column** or **Duplicate Column**.
- 2 Edit the properties of the new column to have the correct name and order, source table and column, datatype, key type and flags as below.
- 3 The **Column Name** and **Source Column** should be the same.
- 4 The **Source Table** should be empty.
- 5 The **Data Type** should be integer.
- 6 The **Key Type** should be 0.
- 7 Only the **Numeric** and **Artificial Key** flags should be set.

The following example shows a manually added artificial key column:



The artificial key for a Data Store Object is set via an identity column. 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 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. |

| Key value | Usage |
|--------------|--|
| -10 backward | Pseudo records. In many cases we have to 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. |

Artificial keys set via a non identity column:

Data Store Tables can have non identity columns as surrogate keys.

The generation of the update procedure will automatically add logic to the code which will associate a sequential number to the artificial key when a new row is inserted into the EDW 3NF table.

The order of these sequential numbers is determined by the business key of the source table. The value of the first newly inserted artificial key will be the value of the highest artificial key in the dimension table plus 1.

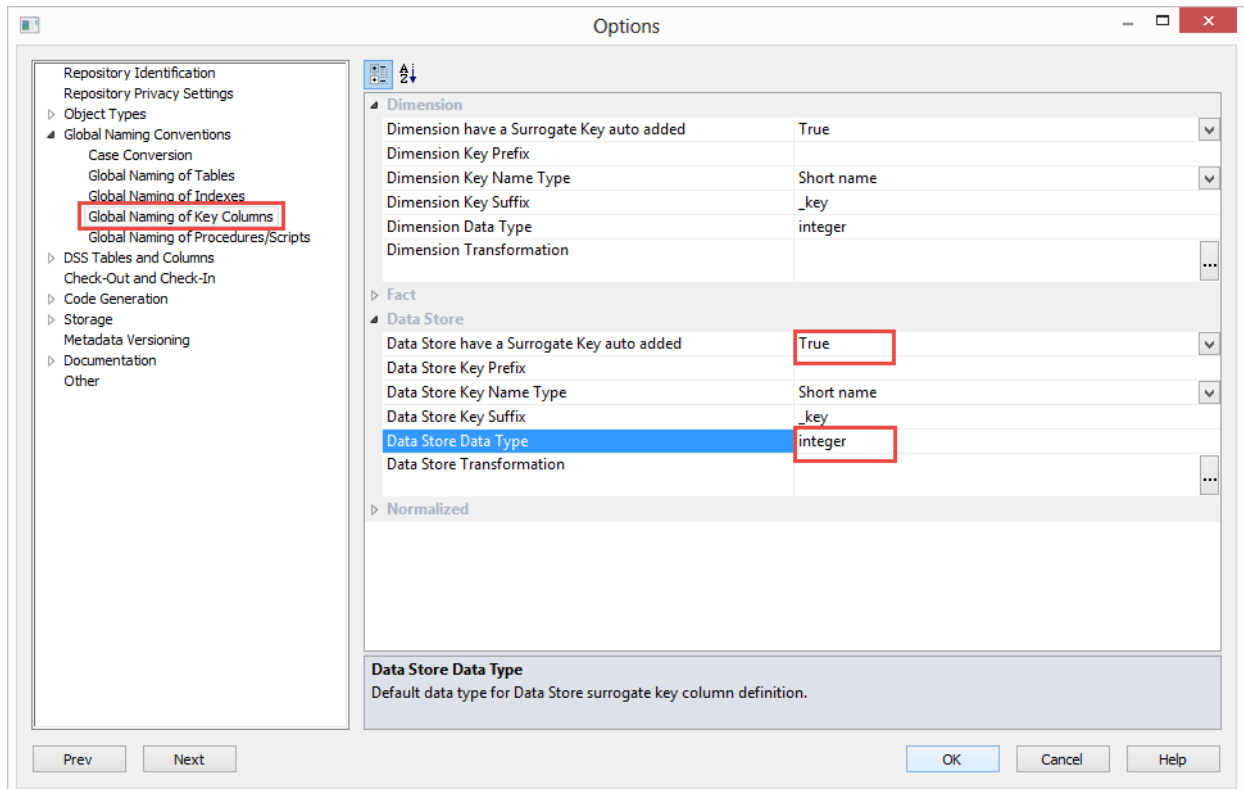
This automatically generated logic can be overwritten by defining a user specific logic on the **Dimension Transformation** field on the **Tools/Options** menu or in the transformation column of the artificial key.

To have an EDW 3NF table with a non identity column as a surrogate key, you can set the table **Data Type** to **integer** in the **Tools/Options** menu.

The old logic for dimensions can be retained if an identity column is chosen as surrogate key.

Allowing for non identity surrogate keys on Data Store tables:

- 1 Go to **Tools -> Options -> Global Naming Conventions -> Global Name of Key Columns**.
- 2 Expand the **Data Store** section.
- 3 Set the **Data Store have a Surrogate Key auto added** field if you want a surrogate key added by default to all Data Store tables.
- 4 Set the **Data Store Data Type** to be **integer** and click **OK**.
- 5 If your tables have been created previously, you will have to **Recreate** the tables after you set this option in the **Tools** menu.



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-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:

The screenshot shows a dialog box titled "Data Store Column ds_customer.customer_code". On the left, there is a "Properties" sidebar with "Transformation" selected. The main area is divided into several sections:

- General:** Table Name: ds_customer; Column Name: customer_code (highlighted); Business Display Name: customer code; Column Description: (empty).
- Physical Definition:** Column Order: 10; Data Type: varchar2(10); Null Values Allowed: True; Default Value: (empty).
- Meta Definition:** Format: (empty); Numeric: False; Additive: False; Attribute: True; Business Key: True; Artificial Key: False; Key Type (0,A,B,C...): A.
- Code Generation:** Zero Key Value: (empty).
- Source Details:** Source Table: load_customer; Source Column: customer_code; Transformation: (empty); Join: False.

At the bottom, there is a "Column Name" section with the text: "Database-compliant name of the column. Dialog Opening Value: customer_code". At the very bottom, there are buttons for "<- Update", "Update ->", "OK", "Cancel", and "Help".

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 a valid for the target database. Typical Teradata databases often have integer, numeric(), varchar(), char(), date and timestamp data types. 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.

Character Set

Database-compliant table column character-set used for storage. Select Latin or Unicode.

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 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:

The screenshot shows a software window titled "Data Store Column ds_customer.customer_code". On the left is a "Properties" sidebar with "Transformation" selected. The main area contains a "Target:" field with "customer_code" and a "Source:" field with "load_customer.customer_code", each with a "Paste" button. Below these is a large text area for "Column Transformation Code (must execute within a SQL SELECT statement)". To the right of this area is a "Function Set:" dropdown menu set to "Default Oracle" and a tree view showing "Functions" and "Available Columns". At the bottom, there is a "Word Wrap Displayed Code" checkbox, "Function Syntax:" and "Function Desc.:" fields, and a row of buttons: "<- Update", "Update ->", "OK", "Cancel", and "Help".

Note: Transformations are only put into effect when the procedure is re-generated.

See *Transformations* (on page 593) for more details.

CHAPTER 13

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 Tables Overview..... | 378 |
| Building EDW 3NF Table..... | 380 |
| Generating the EDW 3NF Update Procedure..... | 384 |
| EDW 3NF Table Artificial Keys..... | 392 |
| EDW 3NF Table Column Properties | 395 |
| EDW 3NF Table Column Transformations | 400 |

EDW 3NF TABLES 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 986) for more details.

The usual steps for creating an EDW 3NF model are defined below and are covered in this chapter. The steps are:

- 1 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.
- 2 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* (see "*Building EDW 3NF Table*" on page 380)
- 3 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* (see "*Building EDW 3NF Table*" on page 380)
- 4 Add columns from other load and/or stage tables if required. See *Building a EDW 3NF Table* (see "*Building EDW 3NF Table*" on page 380)
- 5 Create the EDW 3NF Table in the database. See *Building an EDW 3NF Table* (see "*Building EDW 3NF Table*" on page 380)
- 6 Build the update procedure. See *Generating the EDW 3NF Update Procedure* (on page 384)
- 7 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:

- `COALESCE(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 or set to be added by default through the Tools menu. See *EDW 3NF Table Artificial Keys* 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 or 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.

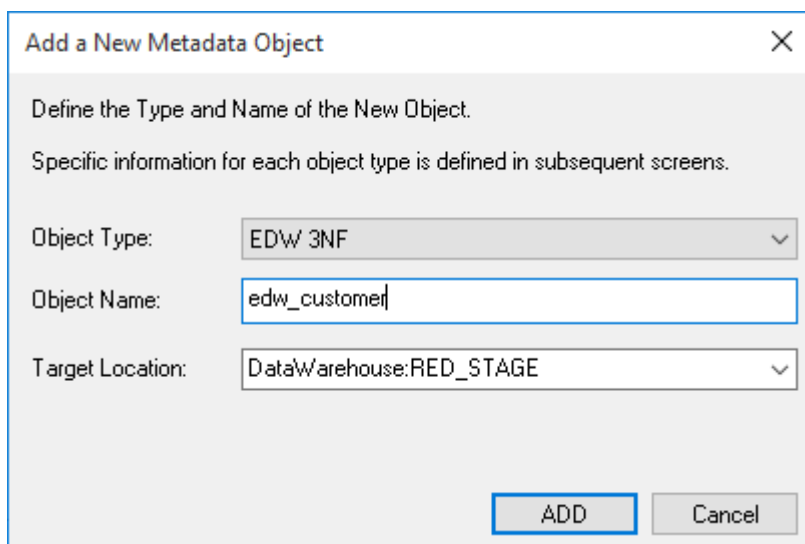
The artificial key is an integer and is built sequentially from 1 upwards. See the section on artificial keys for a more detailed explanation. An artificial key is sometimes referred to as a "surrogate" key.

BUILDING 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

- 1 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 such a list is displayed in the middle pane, the pane is identified as a target for new EDW 3NF tables.
- 2 Browse to the Data Warehouse via the Browse/Source Data menu option.
- 3 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.
- 4 The new object dialog box will appear and will identify the new object as an EDW 3NF table and will provide a default name based on the load or stage table name.
- 5 Either accept this name or enter the name of the EDW 3NF table and click ADD to proceed.



Add a New Metadata Object [X]

Define the Type and Name of the New Object.
Specific information for each object type is defined in subsequent screens.

Object Type: EDW 3NF [v]

Object Name: edw_customer

Target Location: DataWarehouse:RED_STAGE [v]

[ADD] [Cancel]

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.

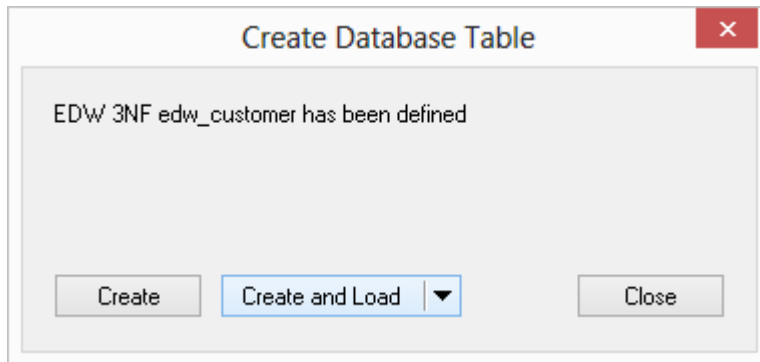
History tables are like slowly changing dimensions in dimensional data warehouses. See **EDW 3NF Table Column Properties** (on page 395) for more information. Change the storage options if desired.

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.

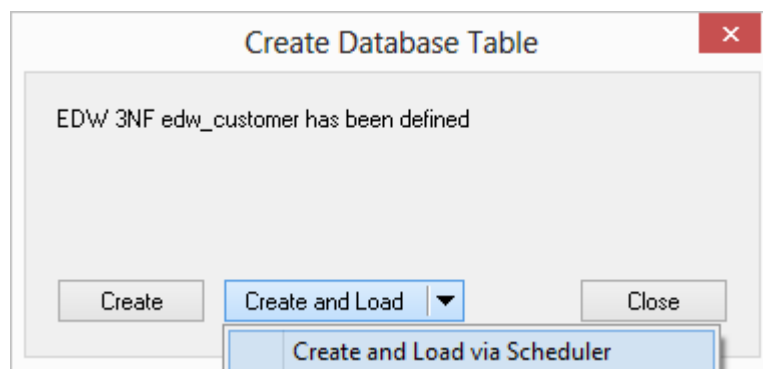
Create and Load

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 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 384) 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 **Finish** and proceed as follows.

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 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 fuller 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. The 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. |
| 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 |

Create 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 achieved by right-clicking on the EDW 3NF table name 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 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 of finding such a column is to double click on the list heading 'Col name'. This will sort the column names into alphabetic order.

Another double click on the heading will sort the columns back into their create order.

The next section covers *Generating the EDW 3NF Update Procedure* (on page 384).

GENERATING THE EDW 3NF UPDATE PROCEDURE

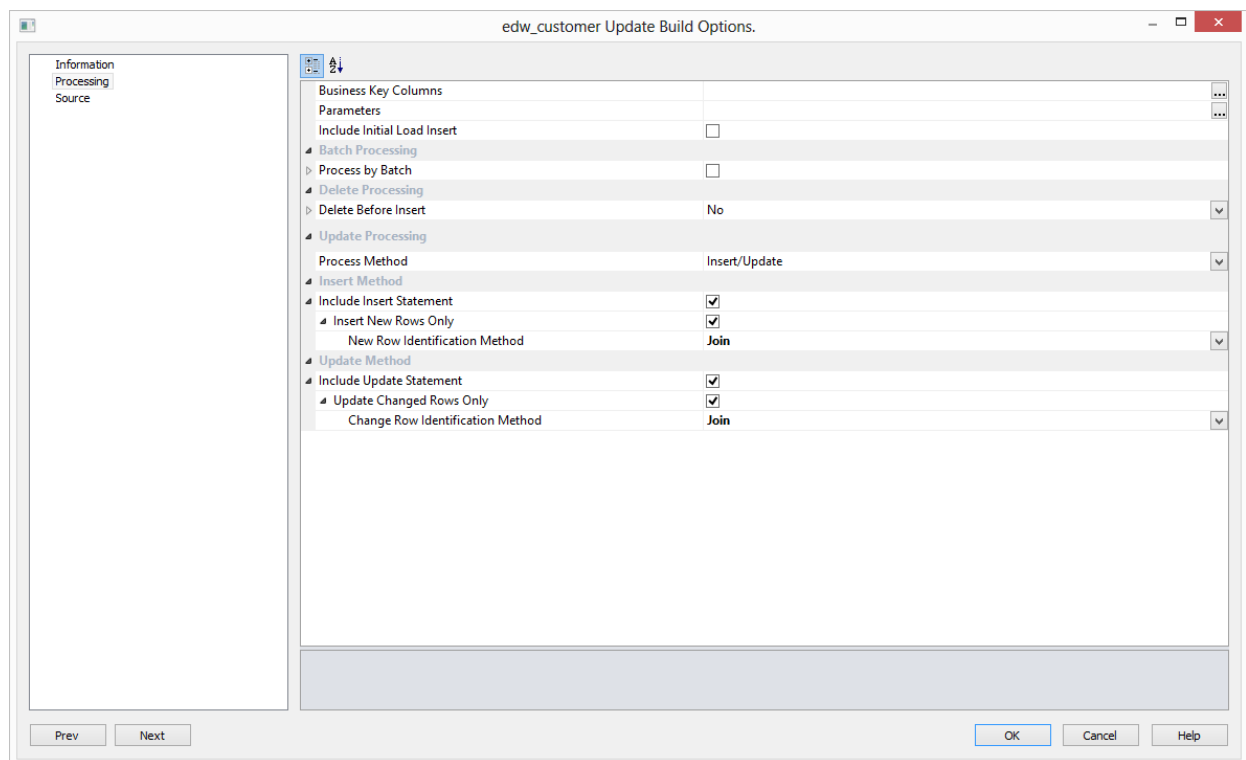
Once an EDW 3NF Object has been defined in the metadata and created in the database, an update procedure can be generated to handle the joining of any tables and the update of the EDW 3NF Object.

Note: You can also generate an update procedure via a template, refer to *Rebuilding Update Procedures* (on page 181) for details.

Generating a Procedure

- 1 To generate a procedure, right-click on the EDW 3NF Object in the left pane and select **Properties**.
- 2 Click on the **Rebuild** button to start the process of generating the new procedure.
- 3 A series of options are available.

PROCESSING TAB



Template: Enables you to generate update procedures via a *template* (see "*Rebuilding Update Procedures*" on page 181).

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.



TIP: Use the column name **ascending/descending** buttons to sort column names. To revert to the metadata column order, click on the **meta column order** button.

Business Key Columns

Columns that define the business key for update processing. Required if include Update options.

Available Columns: A-Z Z-A ▼

- customer_name
- customer_legal_name
- territory_id
- ship_to_address_id
- bill_to_address_id
- sold_to_address_id
- primary_address_type
- primary_contact_person
- active_flag
- customer_category_code
- customer_category_description
- customer_group_code
- customer_group_description
- customer_subgroup_code
- customer_subgroup_description
- creating_employee_id
- created_datetime
- last_change_employee_id
- last_change_datetime

Selected Columns:

- customer_code

OK Cancel

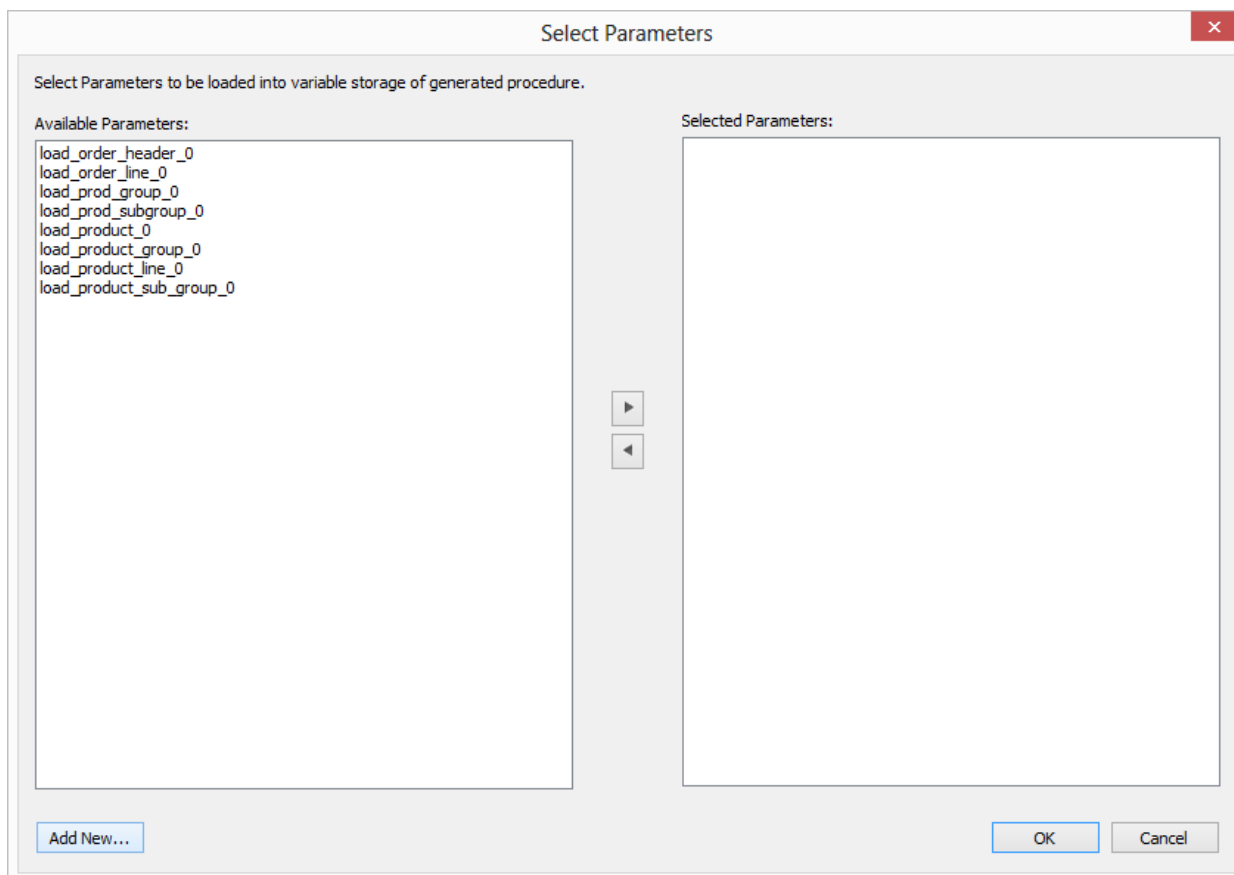
A business key can be made up of multiple columns, but it must provide a unique identifier. Where multiple columns separately uniquely identify rows in the EDW 3NF Object, choose one to act as the primary business key. For example a source table may have a unique constraint on both a product code and a product description. Therefore the description as well as the code must be unique. It is of course possible to combine the two columns, but the normal practice would be to choose the code as the business key.

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.

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.



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`.



TIP: WhereScape RED parameters should be unique within the first 28 characters to avoid conflicting variables names.

See *Parameters* (on page 132) for more information on WhereScape RED Parameters

Include Initial Load Insert: adds an additional insert statement to the update procedure that runs if the target EDW 3NF table 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 not set (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 Business Key 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 not set (do not do batch processing).

Delete Before Insert: allows selecting 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. When this option is selected, it enables the **Issue Warning if a Delete occurs** and the **Delete Where Clause** Fields.

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: allows updating the EDW 3NF table with either an **Insert/Update** or a **Merge** statement. **Merge** allows you to use one Merge statement instead of two separate Insert and update statements.

Source Table Locking: allows a locking request modifier to be specified for each source table. The specified locking request modifier is applied to each source table during generated update procedures. By default, this is set to 'ACCESS' which locks each row being accessed, a blank entry will result in no locking clause in the generated procedure.

Insert Method

Include Insert Statement: set this field to include the insert statement in the procedure. This allows inserting new rows in the EDW 3NF table.

Insert New Rows Only: uses change detection to work out which rows will require inserting.

New Row Identification Method: method used to identify that records in source are not currently recorded in the target table. Select **Join** or **Minus**.

Include Update Statement: set this field to include an update statement in the procedure. This allows updating the changing rows in the EDW 3NF table. If this is set, the **Update Changed Rows Only** option is available.

Update Changed Rows Only: uses change detection to work out which rows require updating. When set, 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. Select **Join** or **Minus**.

Merge Method

Merge Changed Rows only: uses change detection to work out what rows require merging. When the option is set, it enables the **New Row Identification Method**.

New Row Identification Method: method used to identify which records in the source are not recorded or are recorded differently in the target table. Select between **Join** and **Minus**.

If **non identity columns** are used as artificial keys the only new row identification method is **Join**.



EDW 3NF Update procedures usually perform faster when you use the **Join** method for new row identification.

SOURCE TAB

| Distinct Data Select | Source Join | Where Clause | Group By |
|--------------------------|------------------------------------|--------------|----------|
| <input type="checkbox"/> | FROM [load_customer] load_customer | | |

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.

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 or not they are found in the second table. Therefore, 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 or not a corresponding load_order_line entry exists.

Where Clause: The Where clause.

Group By: The Group By clause.

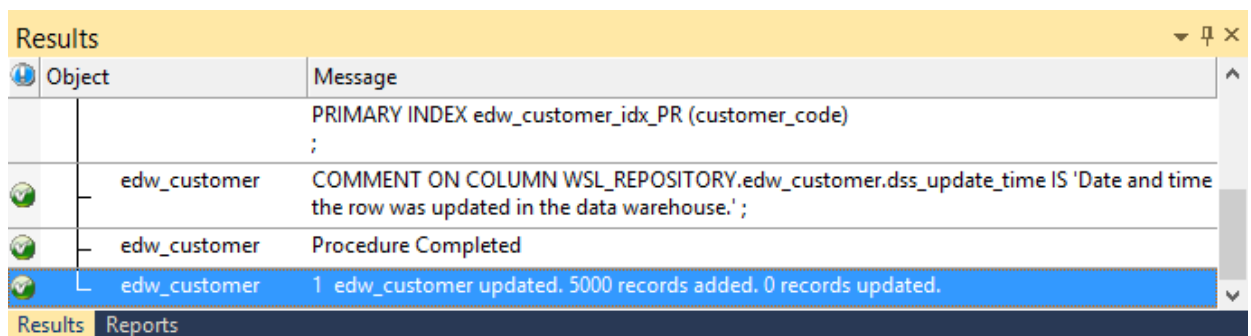
Building and Compiling the Procedure

- 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.

An example of the type of indexes created is as follows:



| Results | | Message |
|---------|--------------|--|
| | | PRIMARY INDEX edw_customer_idx_PR (customer_code) ; |
| ✓ | edw_customer | COMMENT ON COLUMN WSL_REPOSITORY.edw_customer.dss_update_time IS 'Date and time the row was updated in the data warehouse.'; |
| ✓ | edw_customer | Procedure Completed |
| ✓ | edw_customer | 1 edw_customer updated. 5000 records added. 0 records updated. |

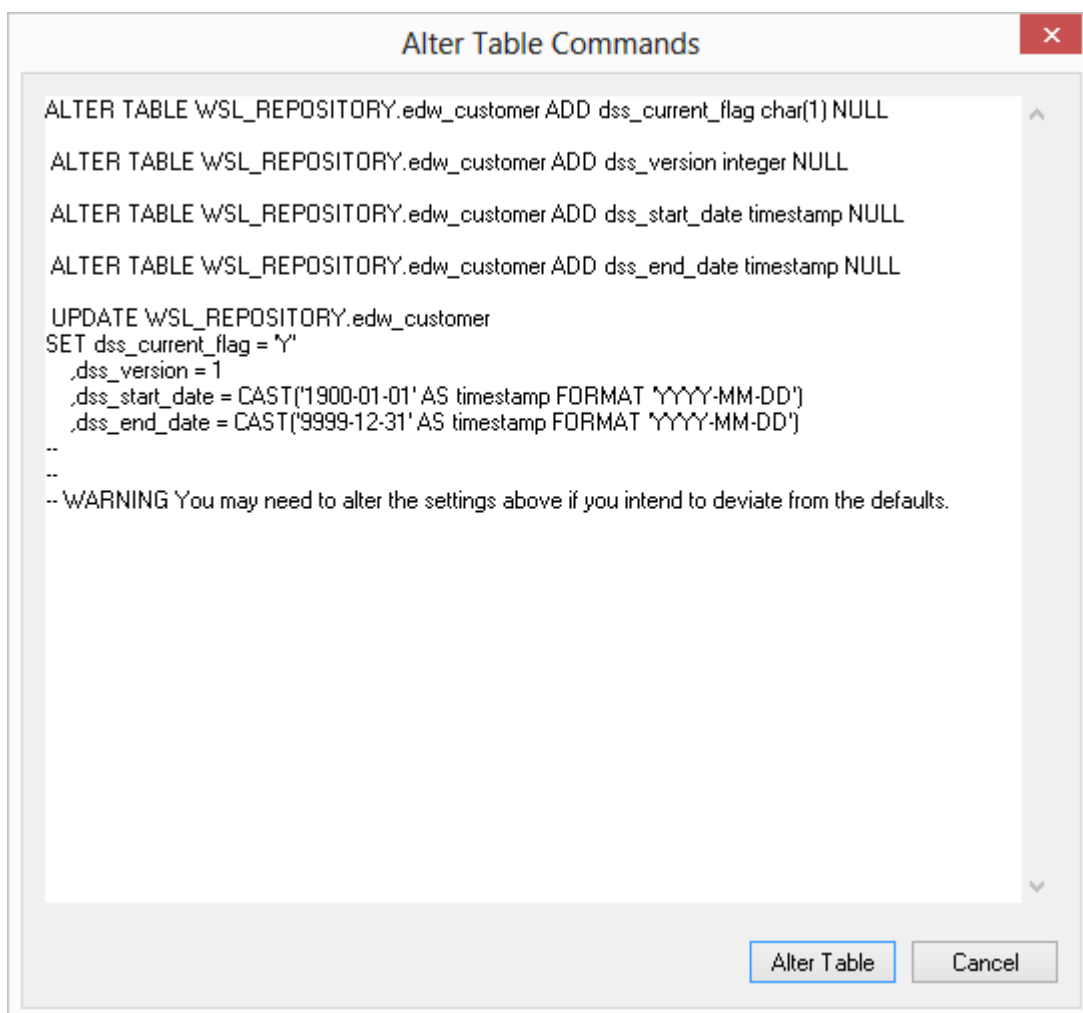
Additional indexes can be added, or these indexes changed. See the chapter on indexes for further details.

Converting an existing EDW 3NF Table to a EDW 3NF History Table

To convert a EDW 3NF table to a EDW 3NF history table, change the table type to **History**, 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 **Alter** on the Adding Additional Columns dialog.
- 2 Click **Alter Table** on the Alter Table Commands dialog:



Note: The SQL in this dialog can be edited if required.

- 3 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 but you can also set an option in the **Tools** menu to have them automatically added to new tables as well (see below - *Allowing for non identity surrogate keys on EDW 3NF tables*).

To manually add an extra artificial key column to an EDW 3NF table:

- 1 Right click in the middle pane and click either **Add Column** or **Duplicate Column**.
- 2 Edit the properties of the new column to have the correct name and order, source table and column, datatype, key type and flags as below.
- 3 The **Column Name** and **Source Column** should be the same.
- 4 The **Source Table** should be empty.
- 5 The **Data Type** should be integer.
- 6 The **Key Type** should be 0.
- 7 Only the **Numeric** and **Artificial Key** flags should be set on.

The following example shows a manually added artificial key column:

The screenshot shows the 'EDW 3NF Column' dialog box with the following properties:

| Category | Property | Value |
|---------------------|------------------------------|-------------------------------------|
| General | Table Name | edw_customer |
| | Column Name | customer_key |
| | Column Title | customer key |
| | Column Description | Manually added artificial key |
| Physical Definition | Column Order | 10 |
| | Data Type | integer |
| | Null Values Allowed | <input type="checkbox"/> |
| | Default Value | |
| | Character Set | |
| | Format | |
| | Character Comparison/Sorting | |
| | Compress | <input type="checkbox"/> |
| Meta Definition | Numeric | <input checked="" type="checkbox"/> |
| | Additive | <input type="checkbox"/> |
| | Attribute | <input type="checkbox"/> |
| | End User Layer Display | <input checked="" type="checkbox"/> |
| | Business Key | <input type="checkbox"/> |
| | Artificial Key | <input checked="" type="checkbox"/> |
| Source Details | Key Type (0,A,B,C,...) | 0 |
| | Source Table | (Empty) |
| | Source Column | customer_key |

Column Name
Database-compliant name of the column.
Dialog Opening Value:

Buttons: OK, Cancel, Help

Artificial Keys set via identity columns:

The artificial key for an EDW 3NF table is set via an identity column. 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. |
| -10 backward | Pseudo records. In many cases we have to 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. |

Artificial keys set via a non identity column:

EDW 3NF Tables can have non identity columns as surrogate keys.

The generation of the update procedure will automatically add logic to the code which will associate a sequential number to the artificial key when a new row is inserted into the EDW 3NF table.

The order of these sequential numbers is determined by the business key of the source table. The value of the first newly inserted artificial key will be the value of the highest artificial key in the dimension table plus 1.

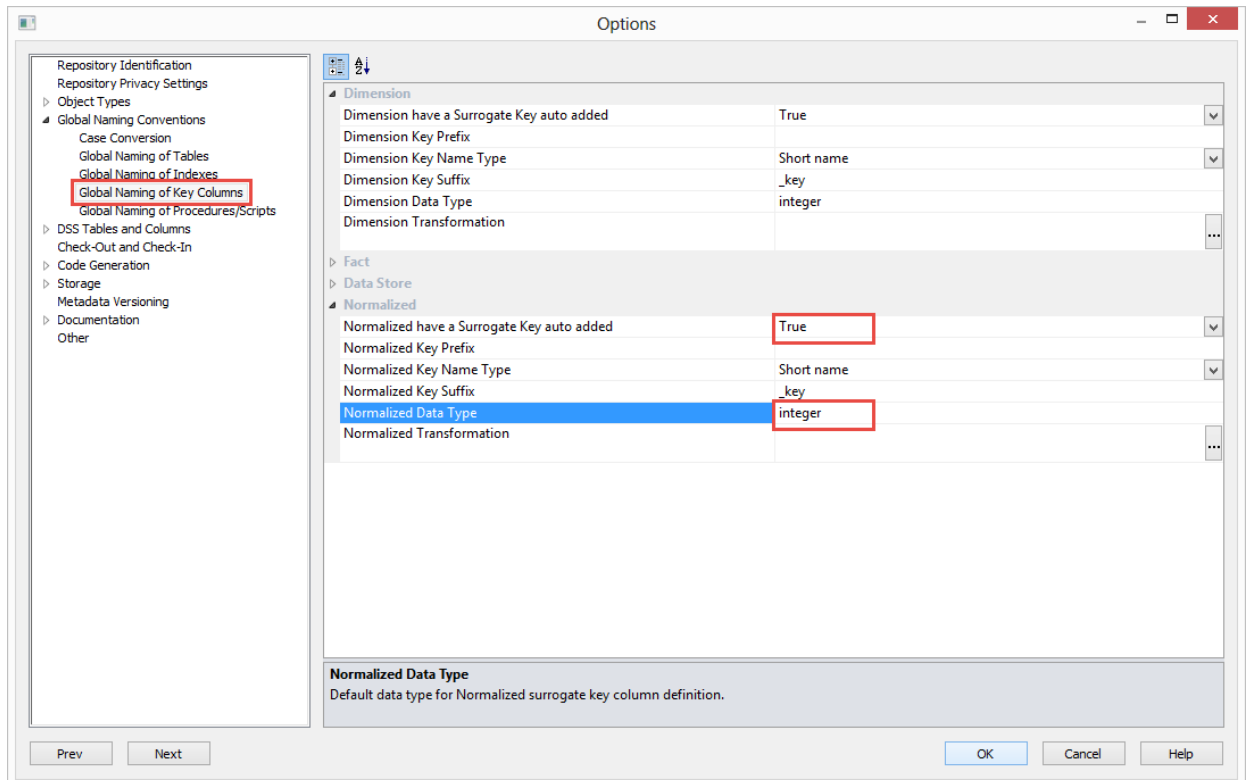
This automatically generated logic can be overwritten by defining a user specific logic on the **Dimension Transformation** field on the **Tools/Options** menu or in the transformation column of the artificial key.

To have an EDW 3NF table with a non identity column as a surrogate key, you can set the table **Data Type** to **integer** in the **Tools/Options** menu.

The old logic for dimensions can be retained, if an identity column is chosen as surrogate key.

Allowing for non identity surrogate keys on EDW 3NF tables:

- Go to Tools>Options>Global Naming Conventions>Global Name of Key Columns.
- Expand the EDW 3NF section.
- Set the EDW 3NF table to have a Surrogate Key auto added if you want a surrogate key added by default to all EDW 3NF tables.
- Set the EDW 3NF Data Type to be **integer** and click **OK**.
- If your tables have been created previously, you will have to **Recreate** the tables after you set this option in the **Tools** menu.



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:

EDW 3NF Column edw_customer.customer_code

Properties

Transformation

General

| | |
|--------------------|---------------|
| Table Name | edw_customer |
| Column Name | customer_code |
| Column Title | customer code |
| Column Description | |

Physical Definition

| | |
|------------------------------|-------------------------------------|
| Column Order | 10 |
| Data Type | varchar(10) |
| Null Values Allowed | <input checked="" type="checkbox"/> |
| Default Value | |
| Character Set | |
| Format | |
| Character Comparison/Sorting | |
| Compress | <input type="checkbox"/> |

Meta Definition

| | |
|------------------------|-------------------------------------|
| Numeric | <input type="checkbox"/> |
| Additive | <input type="checkbox"/> |
| Attribute | <input checked="" type="checkbox"/> |
| End User Layer Display | <input checked="" type="checkbox"/> |
| Business Key | <input checked="" type="checkbox"/> |
| Artificial Key | <input type="checkbox"/> |
| Key Type (0,A,B,C,...) | A |

Source Details

| | |
|------------------|---------------|
| Source Table | load_customer |
| Source Column(*) | customer_code |

Column Name

Database-compliant name of the column.
Dialog Opening Value: customer_code

<- Update Update -> OK Cancel Help

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. A good practice is to only use alphanumeric, 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.

Column Title

Name that the business uses to refer to the column. It does not affect the physical table definition, but rather provides input to the documentation and to the view **ws_admin_v_dim_col** which can be used to assist in the population of a end user tool's end user layer. 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 might 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 a valid for the target database. Typical Teradata databases often have integer, numeric(), varchar(), char(), date and timestamp data types. 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.

Character Set

Database-compliant table column character-set used for storage. Select Latin or Unicode.

Format

Database-compliant table column format. 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. 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.

Character Comparison/Sorting

Determines how the column character values are treated for comparison and sorting operations. Choose from: case specific, not case specific, uppercase case specific or uppercase not case specific.

Compress

Indicates whether the table column values are compressed when stored.

Compress/Compress Value

Optional list of values to be compressed. By default, only NULL is compressed if no list of values is specified.

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. This indicator is set and cleared by WhereScape RED during the dimension update procedure generation process. This checkbox should not normally be altered.

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 model table key where a model 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 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 other EDW 3NF table is looked up during the update procedure build. It allows you to join the other EDW 3NF table manually in the Cursor mapping dialog (where the 'Where' clause is built). The usual dialog for matching the other EDW 3NF table's business key to a column or columns in the table is not displayed if this option is enabled.

Setting this field to Pre Join activates the **Join Source** 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:

The screenshot shows a dialog box titled "EDW 3NF Column edw_customer.customer_code". On the left, there is a "Properties" sidebar with "Transformation" selected. The main area contains the following fields and controls:

- Target:** customer_code (with a "Paste" button)
- Source:** load_customer.customer_code (with a "Paste" button)
- Column Transformation Code:** A large text area with the instruction "(must execute within a SQL SELECT statement)".
- Function Set:** A dropdown menu currently set to "Default Teradata".
- Functions:** A tree view showing "Functions" and "Available Columns".
- Word Wrap Displayed Code
- Function Syntax:** An empty text input field.
- Function Desc.:** An empty text input field with a scroll bar.

At the bottom, there are buttons for "<- Update", "Update ->", "OK", "Cancel", and "Help".

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 593) for more details.

CHAPTER 14

DATA VAULTS

The Data Vault system is an alternative approach to modelling an enterprise data warehouse that has been gaining popularity among organizations.

The Data Vault data warehouse architecture was invented by Dan Linstedt to provide an alternative to the traditional data warehouse modelling approach that includes developing 3rd Normal Form (3NF) type models or dimensional star schema models. The data vault methodology seeks to improve the efficiency of data ingestion and the flexibility of structure changes. For more information about Data Vaults, please refer to Dan Linstedt's website (<http://danlinstedt.com/solutions-2/data-vault-basics/>).

WhereScape RED has been enhanced to expand its current Data Vault functionality and provide improved automation for creating and managing Data Vault objects in WhereScape RED managed Data Warehouses. The enhancement includes the following:

- New DSS columns for Load tables
- New Wizard for Hash key generation
- New Wizard for building Hub, Link and Satellite tables
- New Templates for Procedure generation

All these enhancements are designed to be compliant with Data Vault 2.0 standards and are described in the succeeding sections below.

Note that previous releases of WhereScape RED used a workflow for Data Vault objects that is similar to the workflow for creating Data Store Objects.

The Hub, Link and Satellite tables are based on standard Load or Stage tables (that do not include the hash key column type flags) then WhereScape RED reverts to this behaviour and the resulting procedures are generated by internal WhereScape RED automation and not via templates.

If the above legacy method of creating Data Vault Objects is required, please refer to the Data Store Objects chapter of this User Guide. You can also refer to the previous version of this User Guide, for details on the previous process used to import Data Vault objects into WhereScape 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

| | |
|---|-----|
| Data Vault Functions and Features | 403 |
| Building Data Vault Objects | 414 |

DATA VAULT FUNCTIONS AND FEATURES

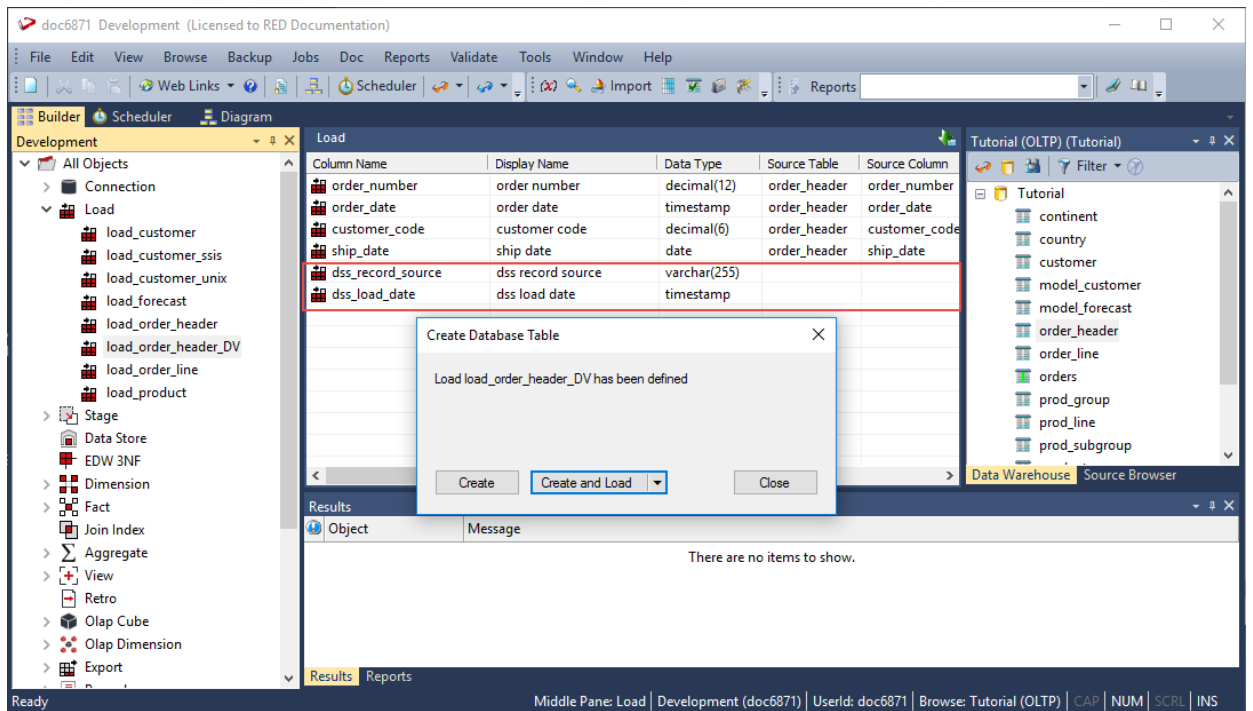
The following describe the WhereScape RED functions and features that are used for building Data Vault objects (Hub, Link and Satellite) to create a Data Vault model.

LOAD TABLE META DATA COLUMNS

The option to add default meta data columns to a new Load table object in WhereScape RED.

The screenshot shows a dialog box titled "Add a New Metadata Object" with a close button (X) in the top right corner. Below the title bar, there is a grey header area with the text "Define the Type and Name of the New Object." and "Specific information for each object type is defined in subsequent screens." The main area contains four rows of form fields: "Object Type:" with a dropdown menu showing "Load"; "Object Name:" with a text input field containing "load_product_DV"; "Target Location:" with a dropdown menu showing "(local)"; and "Data Type Mapping:" with a dropdown menu showing "Standard Teradata to Teradata". At the bottom left, there is a checkbox labeled "Add meta data columns to table" which is checked and highlighted with a red rectangular box. To the right of the checkbox are two buttons: "ADD" and "Cancel".

If the option **Add meta data columns to table** is selected, two DSS columns (**dss_record_source** and **dss_load_date**) are included in the meta data for the table and are populated by transformations. Note that these two DSS columns could equally be applied to other load tables not used in a Data Vault model but are particularly important to comply with the Data Vault standards.



DATA VAULT STAGE TABLE

WhereScape RED object type called **Data Vault Stage** table has been introduced:

The screenshot shows a dialog box titled "Stage stage_product_DV". On the left is a sidebar with "Properties" selected. The main area contains the following fields:

- Table Name: stage_product_DV
- Unique Short Name: stage_product_DV (maximum 22 characters)
- Description: (empty text area)
- Update Procedure: (None) with Rebuild and Regenerate buttons
- Template: To Be Selected
- Custom Procedure: (None)
- Timestamps section with Metadata Structure (2017-06-07 02:13:33.530000), Database, and Database Altered fields.

The "Table Type" dropdown is highlighted with a red box and is open, showing the following options: Data Vault Stage, Stage, Data Vault Stage, Permanent Stage, and Work Table. The "Data Vault Stage" option is currently selected.

The **Data Vault Stage** option can now be selected from **Table Type** drop-down of the **Stage Table Properties** screen. This object type is created via a Wizard which is described in the section below.

HASH KEY GENERATION WIZARD

This Wizard is launched when building a **Stage** table with **Table Type** of **Data Vault Stage**. It enables you to specify the source columns to be used in defining the **Hub**, **Link** and **Change** hash key columns.

Maintain Hash Key Columns

Hub Hash Keys
Link Hash Keys
Change Hash Key

Please select columns to use in the hash by moving columns from the Available list to the Selected list. Enter a name for the Hash then use the Add button to add the Hash to the list of Hub Hash keys below.

Available Columns:

- code
- description
- prod_line
- prod_group
- subgroup

Selected Columns:

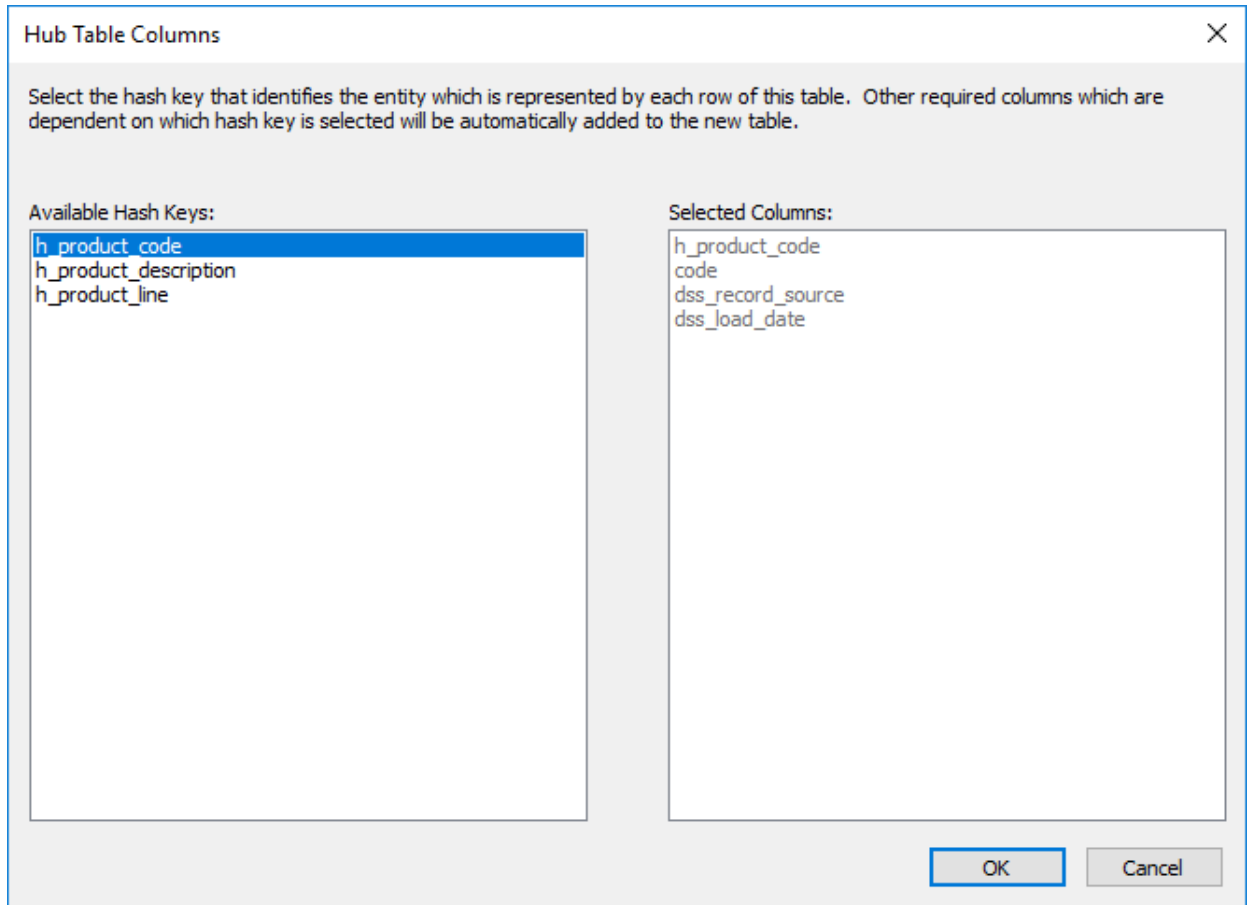
Hash Key Name:

| Hash Key Column | Source Columns |
|-----------------|----------------|
|-----------------|----------------|

The generated Hash Keys are used to build the **Hub**, **Link** and **Satellite** objects in WhereScapeRED. The detailed steps for using this Wizard is described in the succeeding section, **Creating Data Vault Stage tables** (on page 417).

HUB, LINK AND SATELLITE CREATION WIZARD

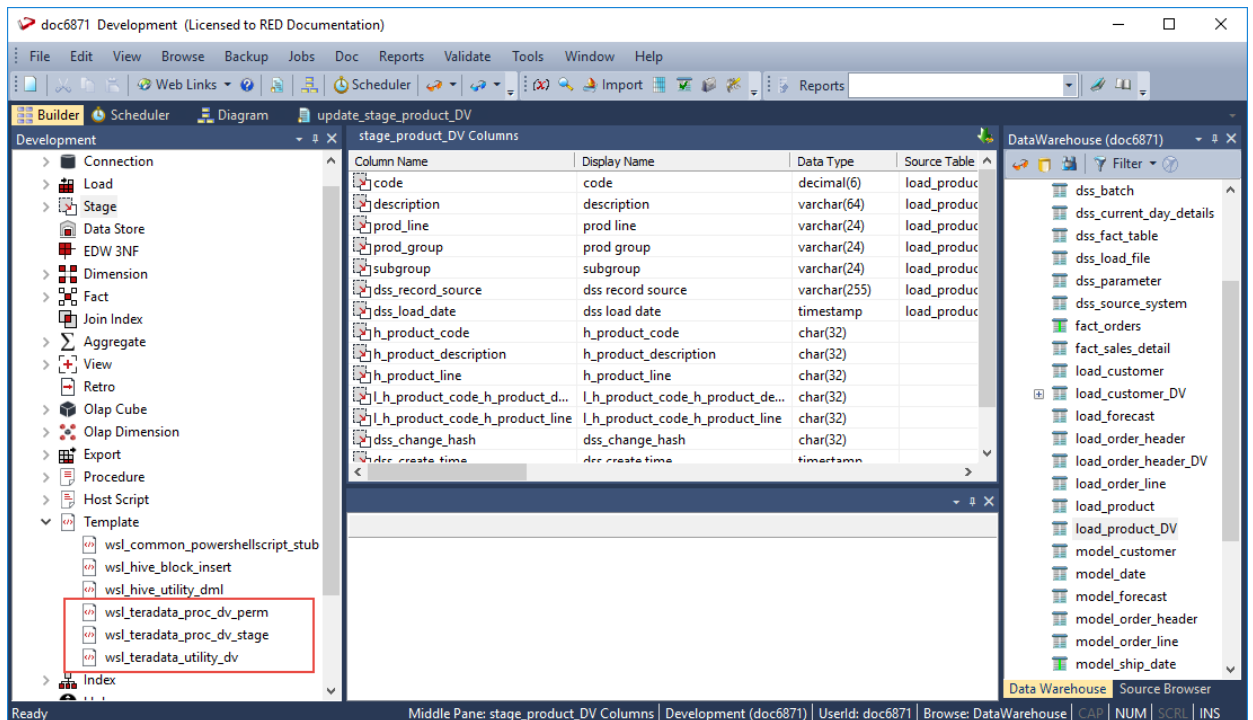
This Wizard is launched when building a Data Vault object (**Hub**, **Link** or **Satellite** table) by dragging and dropping the source **Data Vault Stage** table from the right pane to the middle pane.



The detailed steps for using this Wizard is described in the succeeding section, ***Creating the Hub, Link and Satellite Tables*** (on page 431).

DATA VAULT TEMPLATES

Templates are used to generate update procedures for **Data Vault** objects. Users must select a template to use when generating the update procedure for **Data Vault** objects created in WhereScape RED.



The **Teradata Data Vault** templates are described below:

- **wsl_teradata_proc_dv_stage** – this template creates a Teradata procedure for updating WhereScapeRED **Data Vault Stage** tables.
- **wsl_teradata_proc_dv_perm** – this template creates a Teradata procedure for updating WhereScapeRED **Data Vault** objects (**Hub**, **Link** and **Satellite** tables).
- **wsl_teradata_utility_dv** – this utility template contains generic Teradata macros that are used by the other two templates above.

The above templates are available in WhereScape RED version 6.9.1.0 and above. If the templates are not visible in **Template** objects list after installing/upgrading WhereScape RED, use the WhereScape RED Setup Administrator to Validate the MetaData Repository.

Note: Templates for SQL Server and Oracle are also supplied by WhereScape. For other database types, users need to create/provide their own templates.

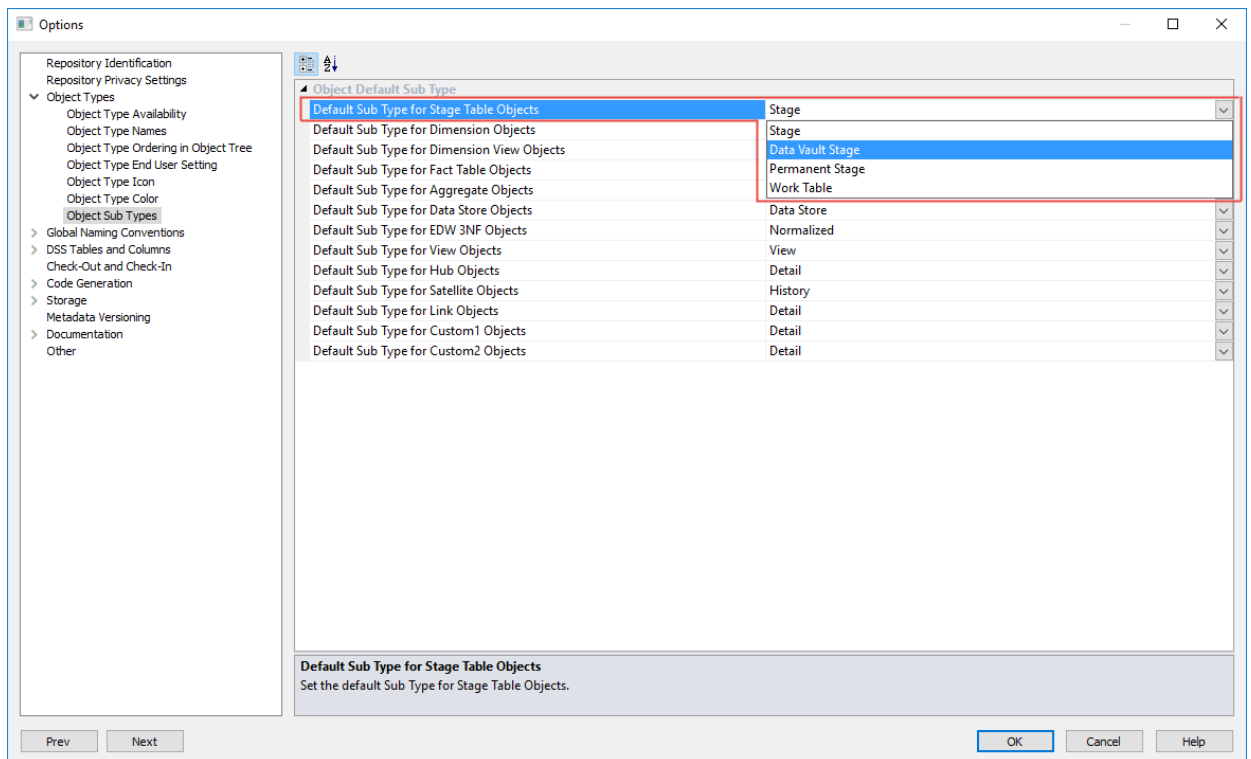
A Wizard to generate update procedures via templates is used to populate **Data Vault** tables. The detailed steps for using this Wizard is described in the succeeding section, **Building Data Vault Objects** (on page 414).

DATA VAULT SETTINGS

Settings for Data Vault objects can be configured from the **Tools > Options** screen.

OBJECT TYPES SETTINGS:

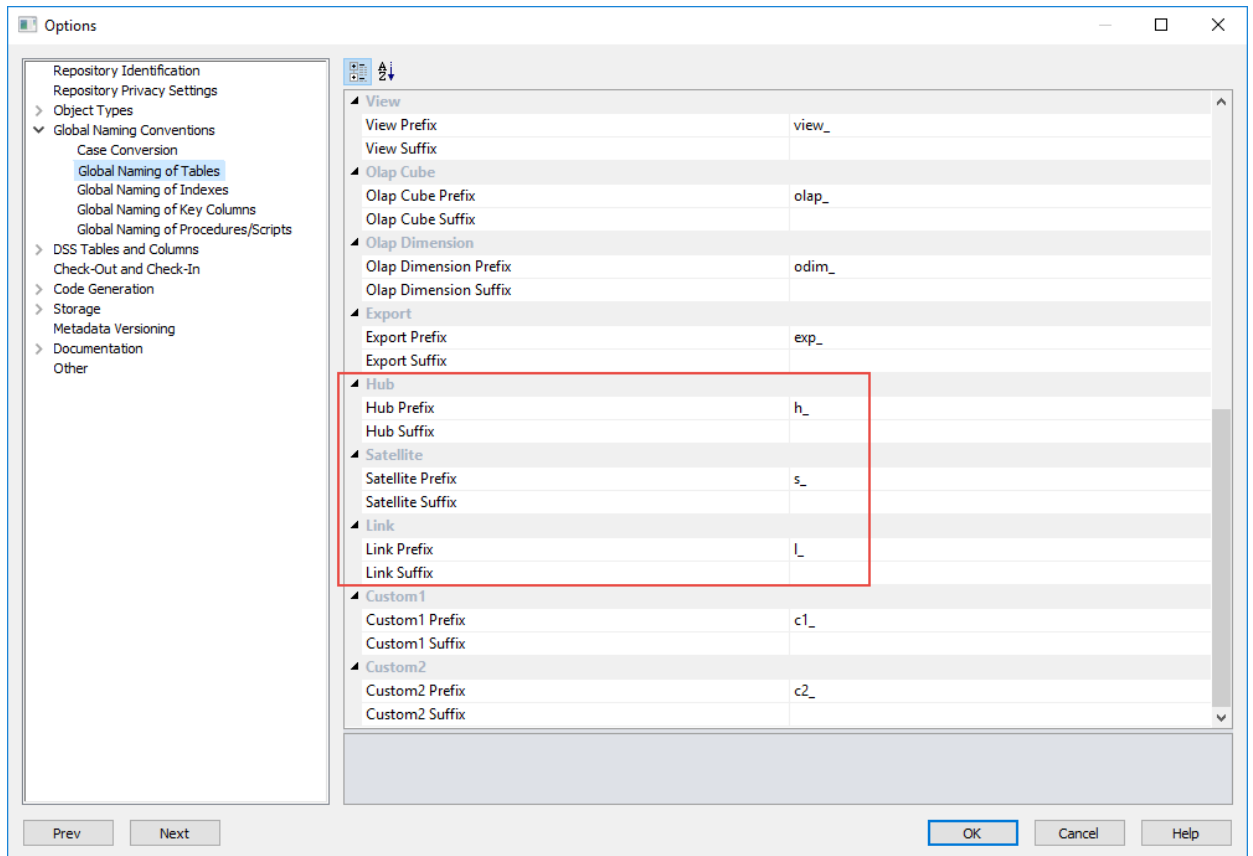
The **Default Sub Type for Stage Table Objects** drop-down includes the **Data Vault Stage** option.



Configure this setting, if you want to set **Data Vault Stage** to be the default **Table Type** in the **Stage Table Properties** screen.

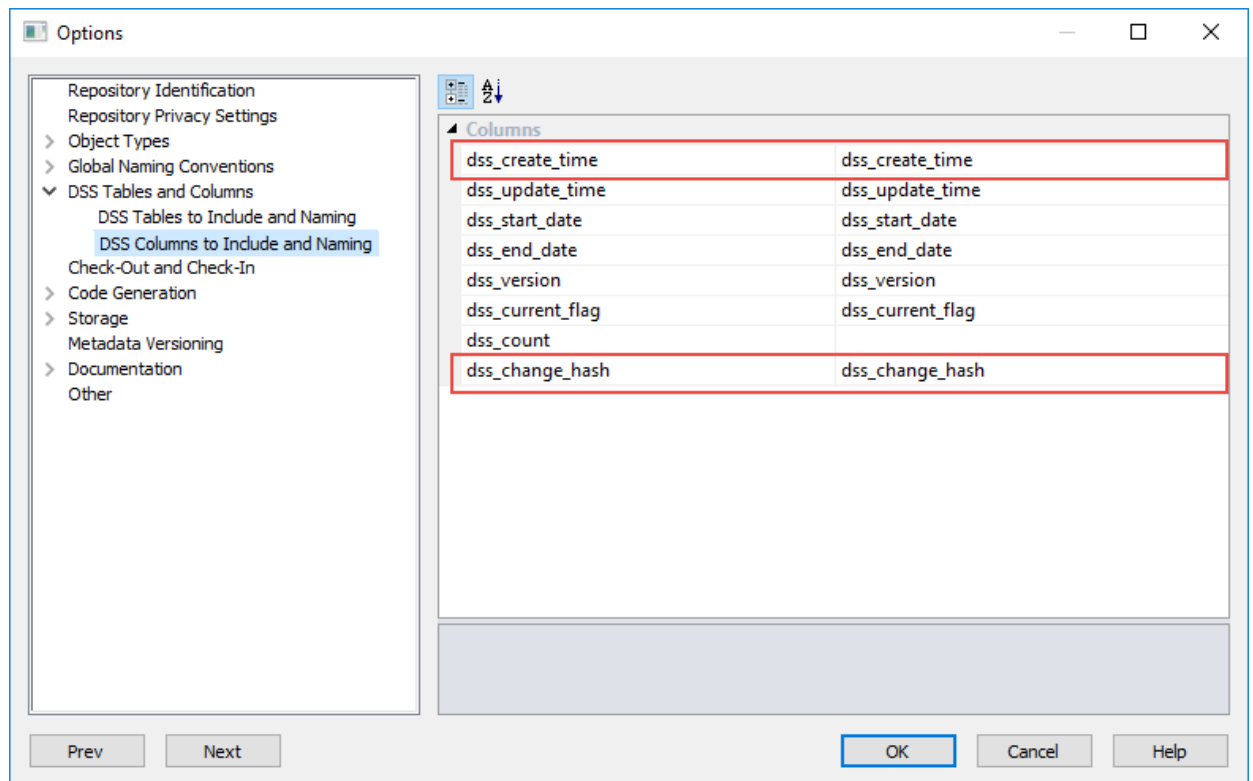
GLOBAL NAMING CONVENTIONS SETTINGS:

The **Global Naming of Tables** setting for **Hub**, **Satellite** and **Link** tables have been set to comply with the recommended standard naming convention for **Data Vault** tables. You can edit this setting to suit your requirements.



DSS TABLES AND COLUMNS SETTINGS:

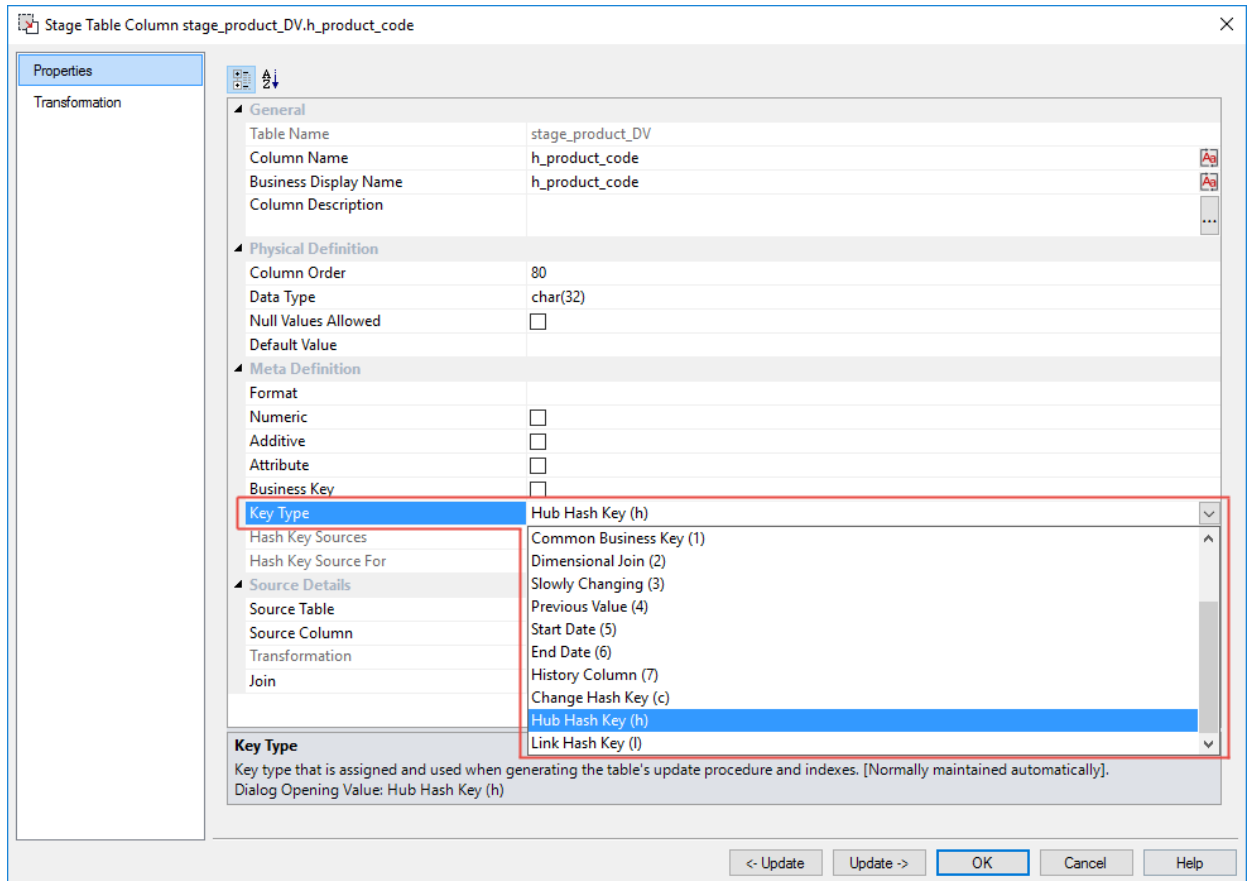
The **DSS Columns to Include and Naming** setting includes two additional columns which is described below:



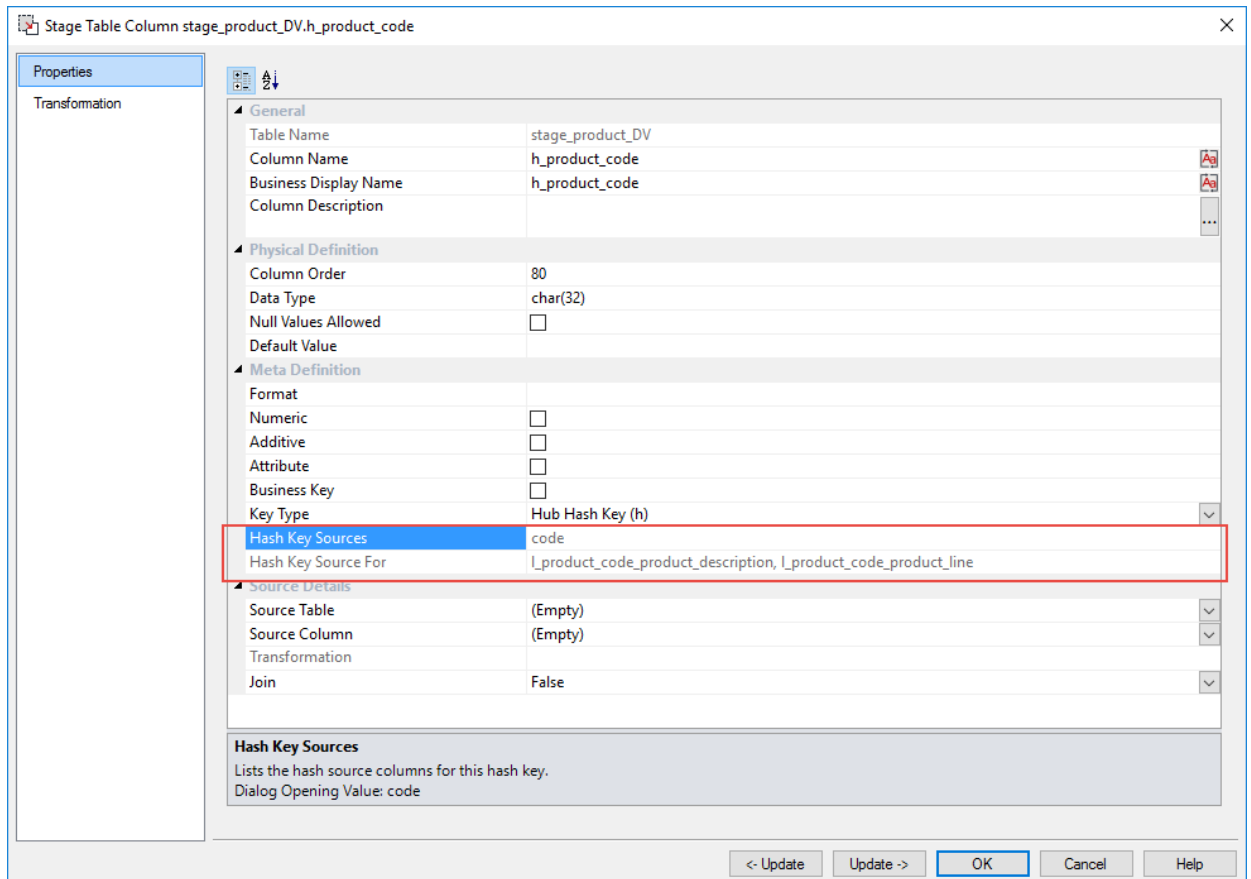
- **dss_create_time** – this column is added to all Stage, ODS, Normalized, Dimension, Fact and Aggregate tables for information purposes only. Leave blank to deactivate.
- **dss_change_hash** – this column is used to identify the differences in the descriptive columns of a Satellite table which is required for generating the change hash key for a Satellite table.

TABLE COLUMN PROPERTIES

The **Key Type** field drop-down include options for **Data Vault** hash keys, e.g. **Change Hash Key (c)**, **Hub Hash Key (h)** and **Link Hash Key (l)**.



Hash Key source information are also displayed for these key types.



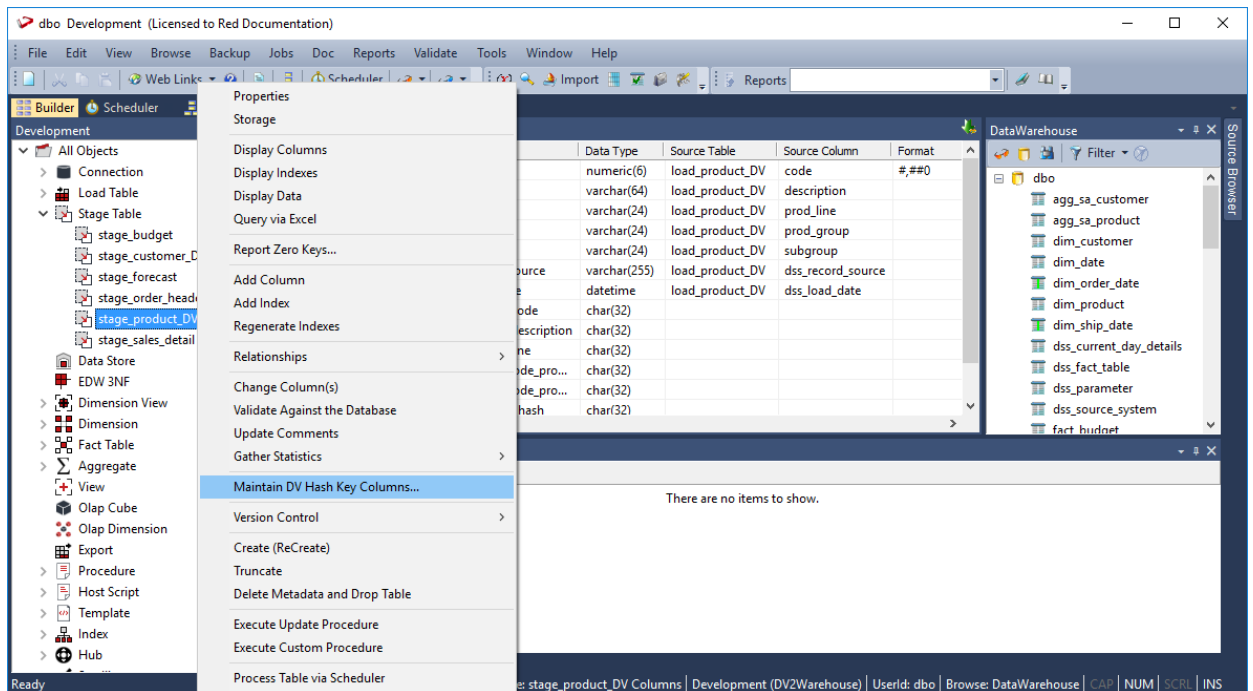
- **Hash Key Sources** – displays the source columns that are used to generate the selected **Hub**, **Link** or **Change** hash key.
- **Hash Key Source For** – displays the hash keys columns that use the displayed hash key sources.

The hash key generation Wizard enables you to specify the source columns to be used in defining the **Hub**, **Link** and **Change** hash key columns. The detailed steps for using this Wizard is described in the succeeding section, *Creating Data Vault Stage Tables* (on page 417).

Once the hash key columns have been defined, another Wizard is used to generate the procedure to populate the columns. The detailed steps for using this Wizard is described in the next section, *Generating Update Procedures for the Data Vault Stage Table* (on page 427).

MAINTAIN HASH KEY COLUMNS

The context menu for **Stage Table** objects, listed in the left pane provides an option for maintaining **Data Vault** hash key columns.



You can review the composition of existing hash keys for a **Data Vault Stage** table (Hub, Link and Satellite) and create additional hash keys by selecting the **Maintain DV Hash Key Columns** option from the selected **Stage Table**'s context menu. This launches the hash key generation Wizard which enables you to maintain the source columns defined for the hash keys.

Note: To remove or change a hash key column, you need to delete it first, e.g. right click the column listed in the middle pane and then select **Delete Column** from the context menu.

BUILDING DATA VAULT OBJECTS

To build Data Vault objects (Hub, Link and Satellite) in WhereScape RED, involves the following procedures.

- 1 Creating Load Tables with the required DSS columns.
- 2 Creating Data Vault Stage tables.
- 3 Generating update procedures for the Stage table via templates.
- 4 Creating the Hub, Link and Satellite tables.
- 5 Generating update procedures for the Hub, Link and Satellite tables via templates.

The detailed steps for each procedure are outlined in the following sections.

CREATING LOAD TABLES

The following describe the steps for creating a **Load** table:

- 1 Browse to the source system connection required (Browse>Source Tables).
- 2 Double-click the **Load Table** object group in the left pane, the middle pane displays a list of existing Load tables.
- 3 Click the source table from the right pane and drag it to the middle pane. You need to create the **Load** table with the required DSS columns—the option to add default meta data columns to the load table must be selected:

Add a New Metadata Object

Define the Type and Name of the New Object.
Specific information for each object type is defined in subsequent screens.

Object Type: Load

Object Name: load_product_DV

Target Location: (local)

Data Type Mapping: Standard Teradata to Teradata

Add meta data columns to table

ADD Cancel

When the build table is performed, the load table created has the two additional columns, which are populated by transformations:

- **dss_record_source** – the connection or source for the load table.
- **dss_load_date** – the date when the data was loaded to the table. This is updated every time a load operation is performed.

These DSS columns added, include column description and transformation information.

The screenshot shows the WhereScape Developer interface. The main window displays a table titled 'load_product_DV Columns' with the following data:

| Ad... | Att... | Order | Column Description | Transformation |
|-------|--------|--------|--------------------|------------------------------------|
| Y | N | 10 | | |
| N | Y | 20 | | |
| N | Y | 30 | | |
| N | Y | 40 | | |
| N | Y | 50 | | |
| N | Y | 999... | Record source. | 'Tutorial (OLTP),Tutorial,product' |
| N | Y | 999... | Load date. | current_timestamp |

Below the table, a 'Results' window shows the following messages:

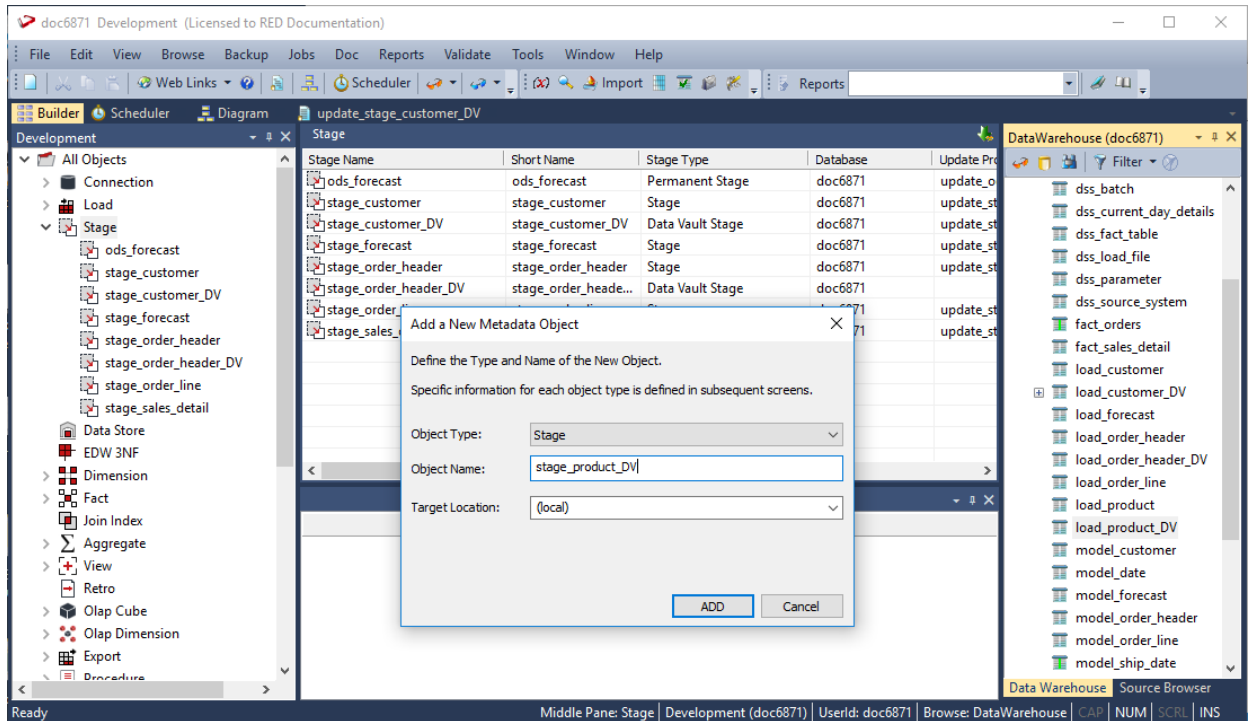
| Object | Message |
|----------|---|
| load_... | ; |
| load_... | 9 rows inserted. |
| load_... | UPDATE doc6871.load_product_DV SET dss_record_source = 'Tutorial (OLTP),Tutorial,product', dss_load_date = current_timestamp WHERE dss_load_date IS NULL; |
| load_... | 9 rows updated |
| load_... | Load Completed Normally. 9 rows inserted. |

The interface also shows a left-hand navigation pane with 'All Objects' expanded to 'Load' and 'load_product_DV' selected. A right-hand pane shows a tree view of the 'Tutorial (OLTP) (Tutorial)' data warehouse with various tables listed.

CREATING DATA VAULT STAGE TABLES

The following describe the steps for creating a **Data Vault Stage** table:

- 1 Browse to the Data Warehouse connection to create the **Stage** table.
- 2 Double-click the **Stage Table** object group in the left pane, the middle pane displays a list of existing **Stage** tables.
- 3 Click the source **Load** table from the right pane and drag it to the middle pane. The selected **Load** table must have the required DSS columns (**dss_record_source** and **dss_load_date**).



- 4 The **Add a New Metadata Object** screen appears and classifies the new object as a **Stage** table. It provides a default name based on the **Load** table name. Accept the name or enter a new name for the **Stage** table and click **ADD** to continue.
- 5 On the **Table Properties** screen, select the **Data Vault Stage** option from the **Table Type** drop-down.

The screenshot shows the 'Stage stage_product_DV' configuration window. The 'Table Type' dropdown menu is open, showing the following options: Data Vault Stage, Stage, Data Vault Stage, Permanent Stage, and Work Table. The 'Data Vault Stage' option is highlighted. The window also shows fields for Table Name, Unique Short Name, Description, Update Procedure, Custom Procedure, and Timestamps.

Notes:

You can set this table type to be the default by configuring the **Default Sub Type for Stage Table Objects** setting in the **Tools > Options > Object Types > Object Sub Types** screen.

RED displays the name of the previously used update procedure template by default, below the **Update Procedure** drop-down.

- Click **OK** on the **Table Properties** screen to launch the Wizard that enables you to define the source columns for the **Hub**, **Link** and **Change** hash key columns.

Maintain Hash Key Columns

Hub Hash Keys
Link Hash Keys
Change Hash Key

Please select columns to use in the hash by moving columns from the Available list to the Selected list. Enter a name for the Hash then use the Add button to add the Hash to the list of Hub Hash keys below.

Available Columns:

- code
- description
- prod_line
- prod_group
- subgroup

Selected Columns:

Hash Key Name:

| Hash Key Column | Source Columns |
|-----------------|----------------|
|-----------------|----------------|

- 7 Select the source column(s) to use in defining the first Hub hash key. The **Hash Key Name** is formed based on the prefix (defined in the **Tools > Options > Global Naming Conventions** settings) and the source column(s) name. You can manually amend the name if required.

Maintain Hash Key Columns

Hub Hash Keys
Link Hash Keys
Change Hash Key

Please select columns to use in the hash by moving columns from the Available list to the Selected list. Enter a name for the Hash then use the Add button to add the Hash to the list of Hub Hash keys below.

Available Columns:
description
prod_line
prod_group
subgroup

Selected Columns:
code

Hash Key Name:

| Hash Key Column | Source Columns |
|-----------------|----------------|
|-----------------|----------------|

- Click **Add** to create the first Hub hash key. Repeat the same steps as required. Once all the required **Hub Hash Keys** are created, click **Next** to progress to the **Link** hash key generation screen.

Maintain Hash Key Columns

Hub Hash Keys
Link Hash Keys
Change Hash Key

Please select columns to use in the hash by moving columns from the Available list to the Selected list. Enter a name for the Hash then use the Add button to add the Hash to the list of Hub Hash keys below.

Available Columns:
code
description
prod_line
prod_group
subgroup

Selected Columns:

Hash Key Name:

| Hash Key Column | Source Columns |
|-----------------------|----------------|
| h_product_code | code |
| h_product_description | description |
| h_product_line | prod_line |

- 9 Creating the **Link Hash Keys** involves the same process, follow the previous steps (6 and 7) to select *multiple* source columns to combine to create the **Link** hash key. Click **Add** to create the first Link hash key. Repeat the same steps for any subsequent keys. Once all the required **Link Hash Keys** are created, click **Next** to progress to the **Change** hash key generation screen.

Maintain Hash Key Columns

Please select columns to use in the hash by moving columns from the Available list to the Selected list. Enter a name for the Hash then use the Add button to add the Hash to the list of Link Hash keys below.

Available Columns:

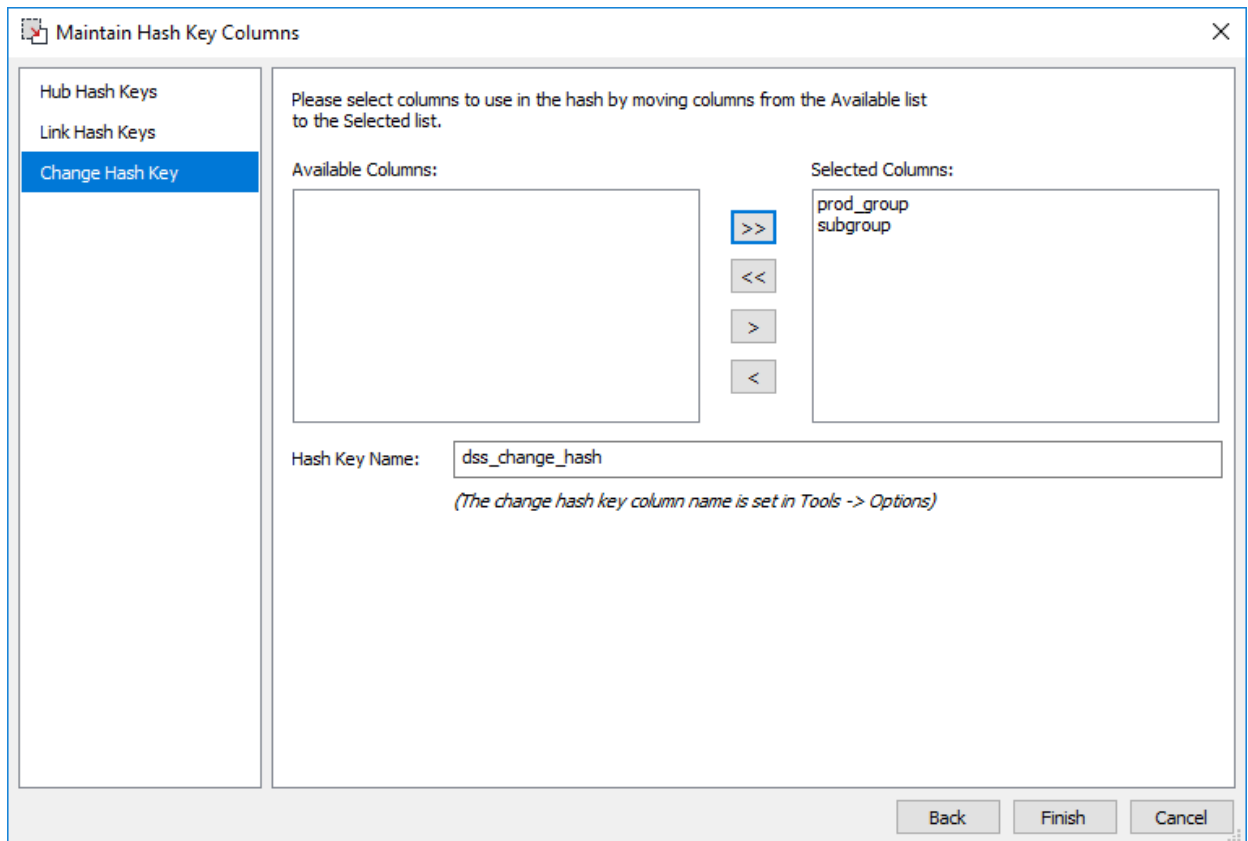
- l_h_product_code_h_product_line
- l_h_product_code_h_product_description
- h_product_line
- h_product_description
- h_product_code
- prod_group
- subgroup

Selected Columns:

Hash Key Name:

| Hash Key Column | Source Columns |
|--|-------------------|
| l_h_product_code_h_product_description | code, description |
| l_h_product_code_h_product_line | code, prod_line |

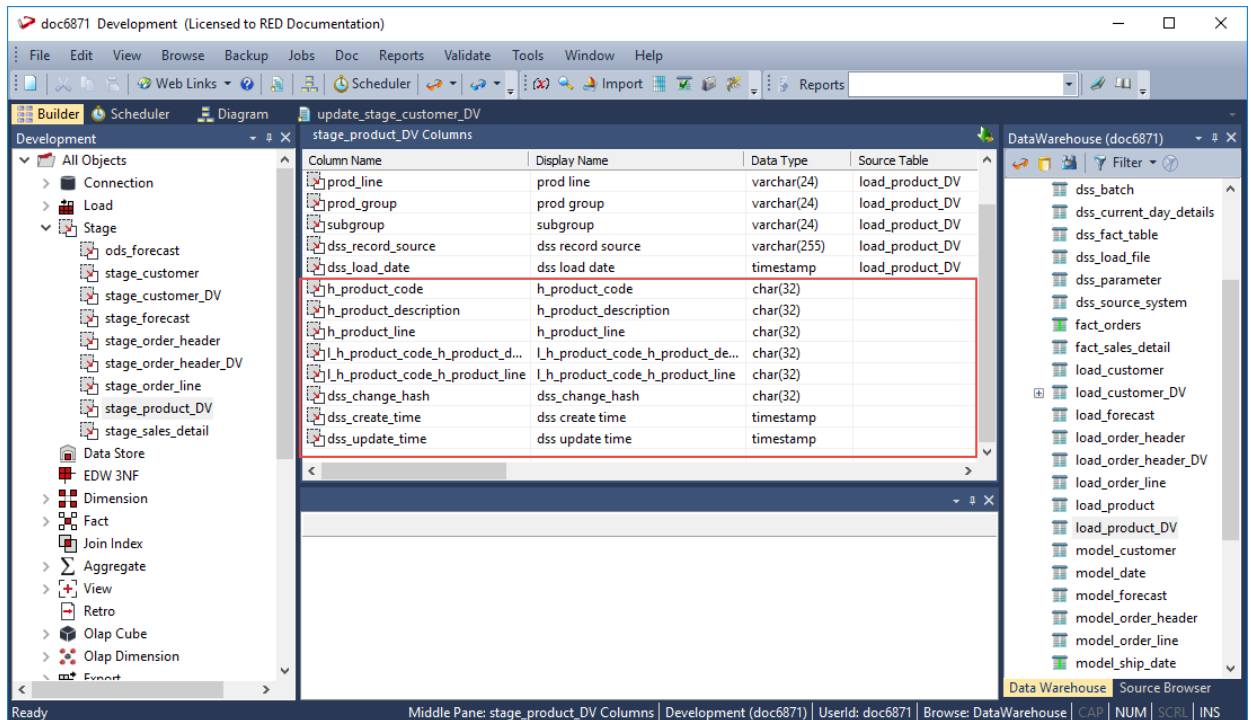
10 Follow the same steps to select the columns to use for the **Change Hash Key**.



Notes: Creating a **Change Hash Key** via the new Wizard in WhereScape RED is limited to one key per **Data Vault Stage** table. This limitation does not apply for Data Vault models imported from WhereScape 3D.

The name of the Change hash key column is fixed and cannot be changed on this screen. It is defined in the **Tools > Options > Global Naming Conventions** settings.

- Click **Finish**, once you have defined the required list of descriptive columns for the Change hash key. The new **Stage** table is added to the **Stage Table** objects list in the left pane and the columns included in the table are listed in the middle pane.



In addition to the columns defined from the Load table, the following columns and their metadata have been added to the Data Vault Stage table:

- The Hub Hash Keys
- The Link Hash Keys
- The Change Hash Key
- The DSS_CREATE_TIME column
- The DSS_UPDATE_TIME column

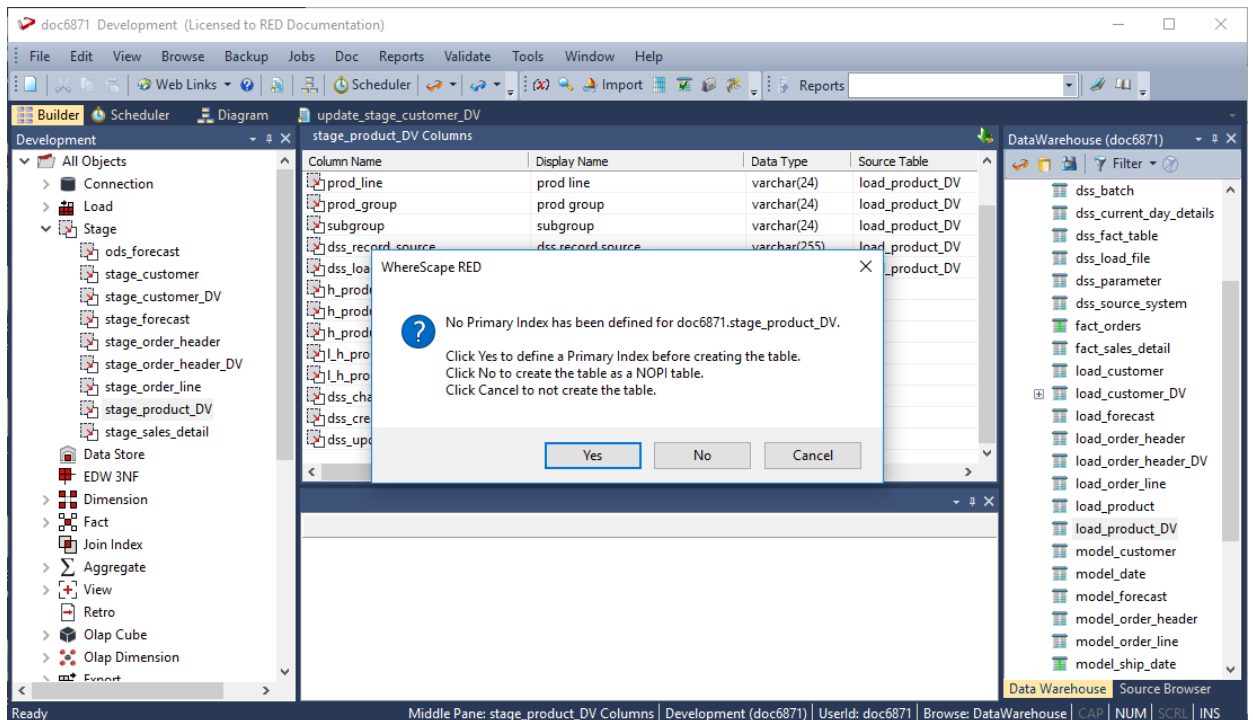
The metadata for the hash columns include the source columns that were used to create them (used to generate the hash keys).

The hash keys created are used in the subsequent Data Vault object (Hub, Links and Satellites) creation Wizards.

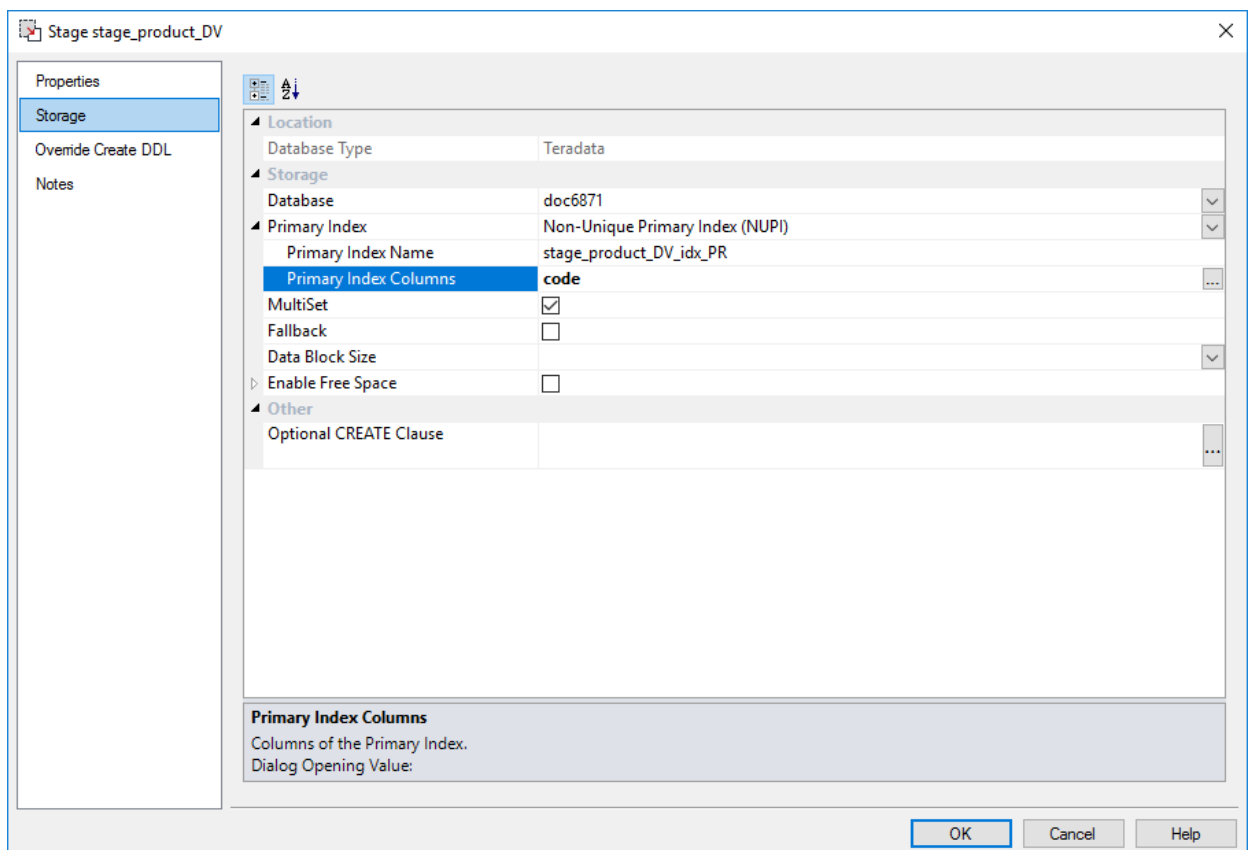
Tip: You can review the composition of existing hash keys for a Data Vault Stage table (Hub, Link and Satellite) and create additional hash keys by selecting the **Maintain DV Hash Key Columns** option from the selected **Stage Table**'s context menu. This launches the hash key generation Wizard which enables you to maintain the source columns defined for the hash keys.

Note: To remove or change a hash key column, you need to delete it first, e.g. right click the column listed in the middle pane and then select **Delete Column** from the context menu.

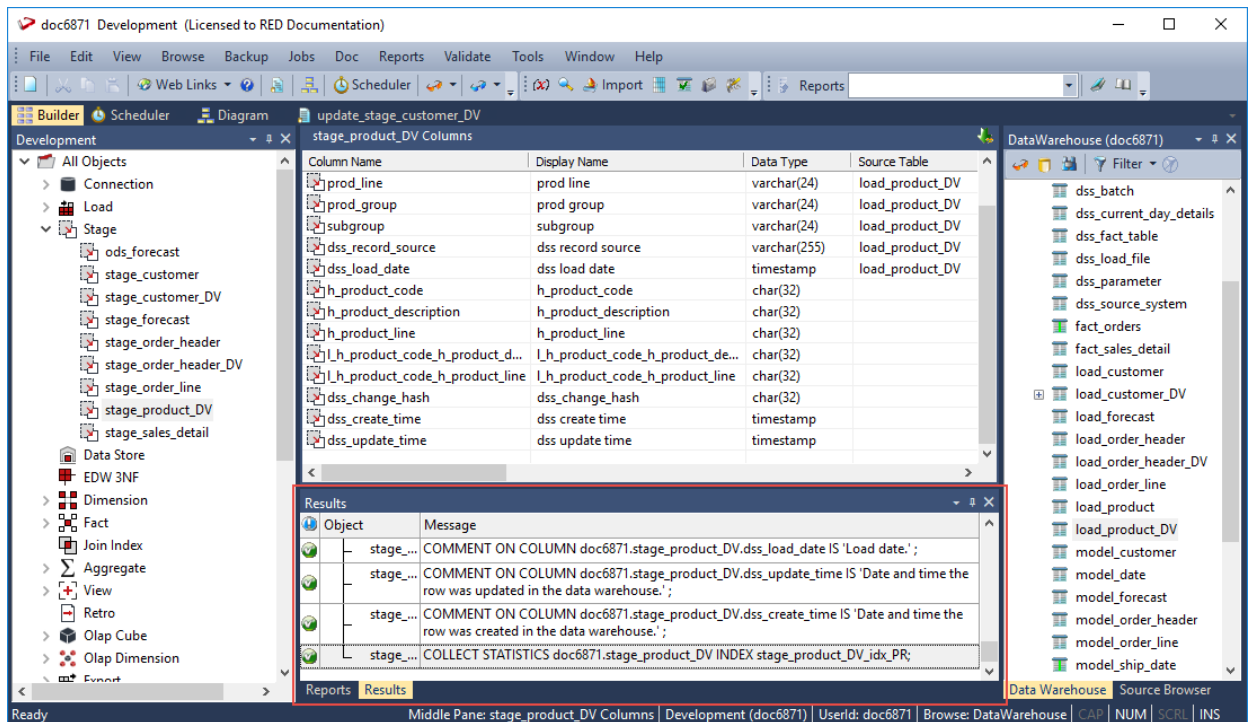
- 12 Right-click the new **Stage** table you defined from the left pane, under the **Stage Table** objects list and select **Create (ReCreate)** from the context menu to create the table.
- 13 Click Yes on the Primary Index prompt.



- 14 Specify **code** on the **Primary Index Columns** field and then click **OK**.



15 The **Results** pane displays confirmation that the **Stage** table was successfully created.



Once the new Data Vault Stage table is defined and created, clicking the **Rebuild** button on the **Table Properties** screen launches the Wizard to generate the procedure to populate the table. This Wizard utilizes a template to create the procedure.

The detailed steps for using this Wizard is described in the next section, **Generating Update Procedures for the Data Vault Stage Table**.

GENERATING UPDATE PROCEDURES FOR THE DATA VAULT STAGE TABLE

After successfully defining and creating the Stage table, you can generate the update procedure via a template to populate the table.

Note: Please ensure that you have installed the WhereScape supplied *templates* (see "**Data Vault Templates**" on page 407) or created your own Data Vault templates, before performing the steps below.

- 1 Click the **Rebuild** button on the **Table Properties** screen to launch the procedure generation Wizard to populate the table.

Stage stage_product_DV

Properties

Storage

Override Create DDL

Notes

Table Name: stage_product_DV

Table Type: Data Vault Stage

Unique Short Name: (maximum 22 characters) stage_product_DV

Description:

Update Procedure: (None) Rebuild Regenerate

Template: wsl_teradata_proc_dv_stage

Custom Procedure: (None)

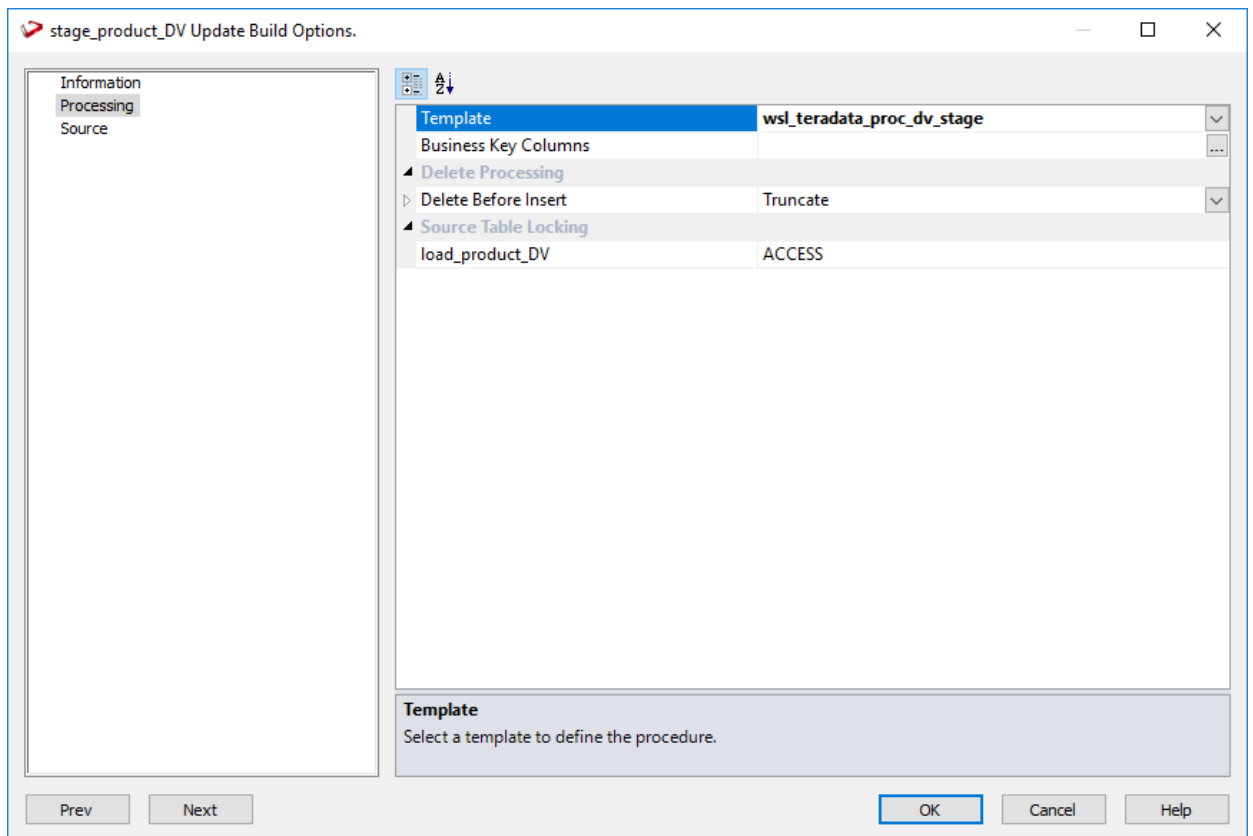
Timestamps

| | | |
|----------------------------|----------|-------------------|
| Metadata Structure | Database | Database Altered: |
| 2017-05-28 23:09:50.730000 | | |

OK Cancel Help

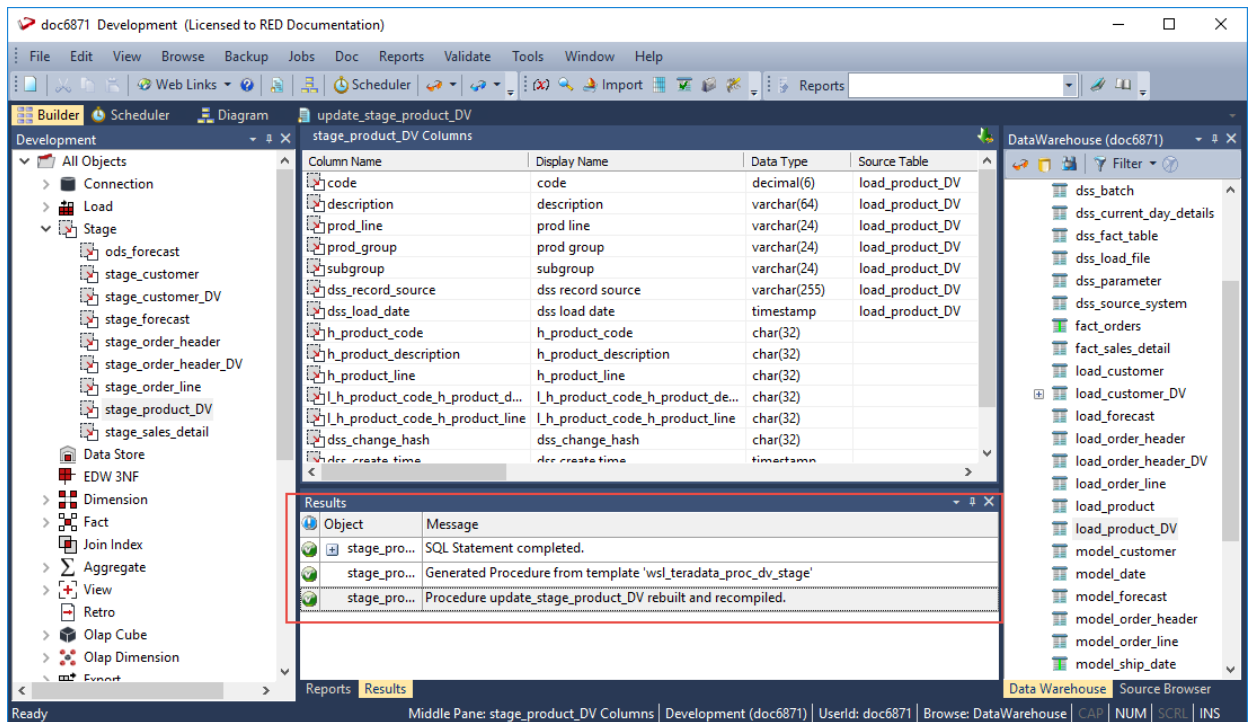
Note: RED displays the name of the previously used update procedure template by default, below the **Update Procedure** drop-down.

- 2 On the **Processing** tab of **Table Update Build Options** screen, select the template to use from the **Template** drop-down or use the previous update procedure template.

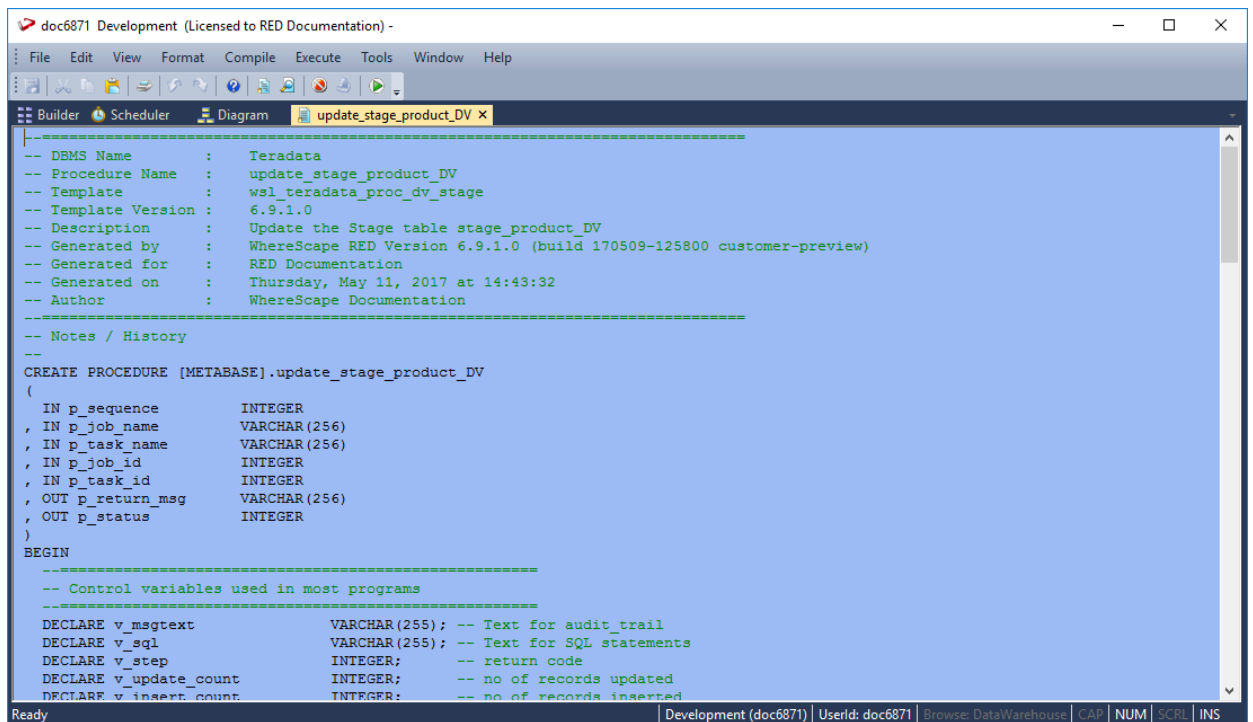


Note that you do not need to define the **Business Key Columns** because the procedure generation is based on the defined hash keys.

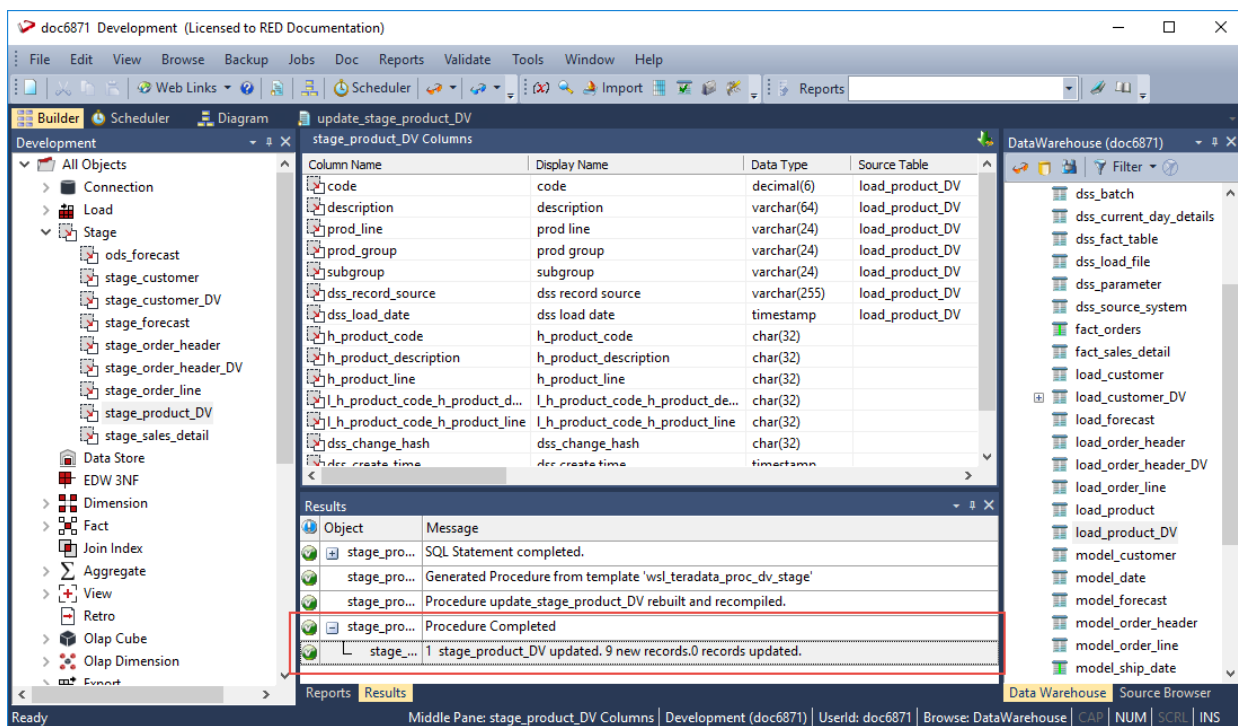
- Click **OK** to proceed with the procedure generation. The **Results** pane displays confirmation that the procedure was generated.



- Right-click the **Stage** table in the left pane, under the **Stage Table** objects list and select **Code>View update** from the context menu to view the contents of the update procedure generated.



- Right-click the **Stage** table in the left pane and select **Execute Update Procedure** from the context menu to run the procedure. The **Results** pane displays the number of records created.



CREATING THE HUB, LINK AND SATELLITE TABLES

After successfully creating and populating the **Stage** table you can now create the **Hub**, **Link** and **Satellite** tables. The **Hub**, **Link** and **Change** hash keys information stored in the **Stage** table is used by the Wizard for building these **Data Vault** objects.

CREATING THE HUB TABLE

The following describe the steps for creating a Hub table:

- 1 Browse to the Data Warehouse connection to create the Hub table.
- 2 Double-click the **Hub** object group in the left pane, the middle pane displays a list of existing Hub tables.
- 3 Click the source **Data Vault Stage** table from the right pane and drag it to the middle pane.
- 4 The Hub table creation Wizard appears and prompts you to select the Hash Key to use from the **Available Hash Keys** pane. The columns that comprise the selected Hash Key are displayed under the **Selected Columns** pane—these are the columns that will be populated by the Wizard on the new **Hub** table.

Hub Table Columns

Select the hash key that identifies the entity which is represented by each row of this table. Other required columns which are dependent on which hash key is selected will be automatically added to the new table.

Available Hash Keys:

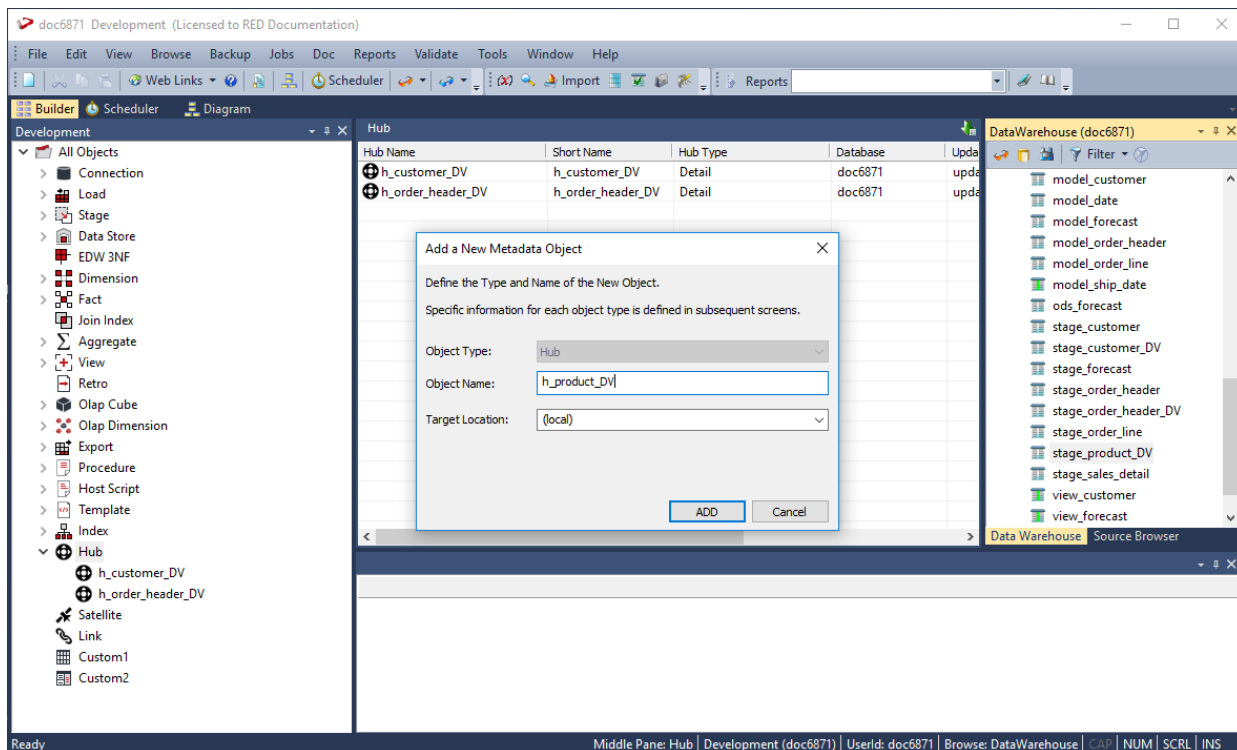
- h_product_code
- h_product_description
- h_product_line

Selected Columns:

- h_product_code
- code
- dss_record_source
- dss_load_date

OK Cancel

- 5 Select the Hash Key you want to use from the **Available Hash Keys** pane to see the columns that will be included in your Hub table under the **Selected Columns** pane. Click **OK** to continue.
- 6 The **Add a New Metadata Object** screen appears and classifies the new object as a Hub table. It provides a default name based on the source **Data Vault Stage** table name. Accept the name or enter a new name for the Hub table and click **ADD** to continue.



- Click the **OK** button on the **Table Properties** screen to finish defining the meta data for the Hub table. The new Hub table is added to the **Hub** objects list in the left pane and the columns included in the table are listed in the middle pane.

The screenshot displays the WhereScape Development tool interface. The left pane shows a tree view of objects under 'Hub', with 'h_product_DV' selected. The middle pane shows the 'h_product_DV Columns' table with the following data:

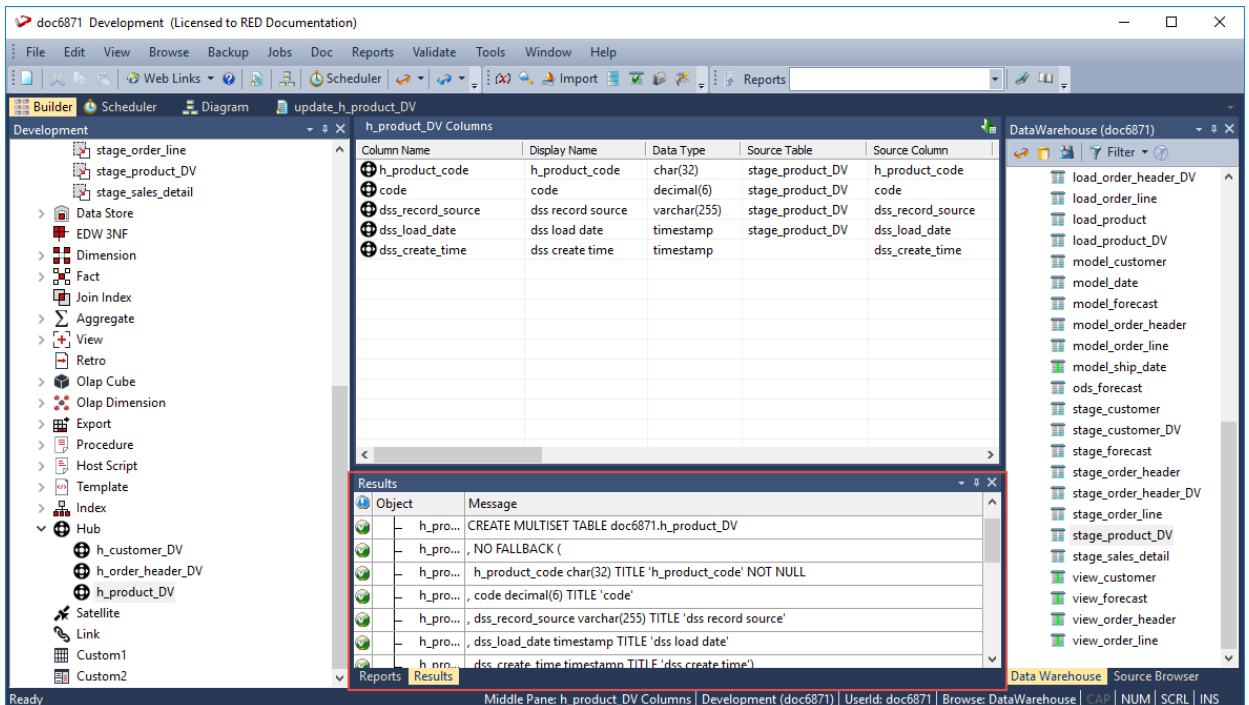
| Column Name | Display Name | Data Type | Source Table | Source Column |
|-------------------|-------------------|--------------|------------------|-------------------|
| h_product_code | h_product_code | char(32) | stage_product_DV | h_product_code |
| code | code | decimal(6) | stage_product_DV | code |
| dss_record_source | dss record sou... | varchar(255) | stage_product_DV | dss_record_source |
| dss_load_date | dss load date | timestamp | stage_product_DV | dss_load_date |
| dss_create_time | dss create time | timestamp | | dss_create_time |

The right pane shows a list of objects in the 'DataWarehouse (doc6871)' database, including 'model_customer', 'model_date', 'model_forecast', 'model_order_header', 'model_order_line', 'model_ship_date', 'ods_forecast', 'stage_customer', 'stage_customer_DV', 'stage_forecast', 'stage_order_header', 'stage_order_header_DV', 'stage_order_line', 'stage_order_line_DV', 'stage_product_DV', 'stage_sales_detail', 'view_customer', and 'view_forecast'. The bottom pane shows the 'Results' window with the following data:

| Object | Message |
|--------------|---------------------------|
| h_product_DV | Generating relationships. |
| h_product_DV | No relationships found. |

The status bar at the bottom indicates: 'Ready | Middle Pane: h_product_DV Columns | Development (doc6871) | Userid: doc6871 | Browse: DataWarehouse | CAP | NUM | SCRL | INS'

- 8 Right-click the new Hub table you defined in the left pane and select **Create (ReCreate)** from the context menu to create the table.
- 9 Click Yes on the Primary Index prompt and specify **code** on the **Primary Index Columns** field and then click **OK**. The **Results** pane displays confirmation that the Hub table was successfully created.



After the new Hub table is defined and created, clicking the **Rebuild** button on the **Table Properties** screen launches the Wizard to generate the update procedure to populate the table. This Wizard utilizes a template to create the procedure.

The detailed steps for using this Wizard is described in the section, **Generating Update Procedures for the Hub, Link and Satellite Tables** (see "**Generating Update Procedures for Hub, Link and Satellite Tables**" on page **Error! Bookmark not defined.**).

Once you have run the generated update procedures, you can view the generated Hub Hash keys, by right-clicking the new **Hub** table you created in the left pane and then selecting **Display Data** from the context menu:

The screenshot shows the WhereScape Development tool interface. The main window is titled "doc6871 Development (Licensed to RED Documentation)". The left pane shows a tree view of objects, with the "Hub" folder expanded to show "h_product_DV". The middle pane displays a table titled "Data display for doc6871.h_product_DV" with the following data:

| h_product_code | code | dss_record_source | dss_load_date |
|----------------------------------|------|----------------------------------|-----------------------|
| 10467B374D4D93EAD2A9ACABAEF45AC2 | 1009 | Tutorial (OLTP).Tutorial.product | 2017-05-11 00:23:4... |
| D5878BEDB482491A80B838FBC70E910D | 1005 | Tutorial (OLTP).Tutorial.product | 2017-05-11 00:23:4... |
| FC9CECDD0C0DFFFE501D1174C2134B4 | 1004 | Tutorial (OLTP).Tutorial.product | 2017-05-11 00:23:4... |
| 22D44976DA37FF85D9617B0F80EBBF6E | 1002 | Tutorial (OLTP).Tutorial.product | 2017-05-11 00:23:4... |
| F758B6D720517D591C7898E0F4314C01 | 1001 | Tutorial (OLTP).Tutorial.product | 2017-05-11 00:23:4... |
| 86EC9581E85B1A83BA19A29FBDD7C74 | 1003 | Tutorial (OLTP).Tutorial.product | 2017-05-11 00:23:4... |
| 5B3960E14BB2DF8F5338740ED13EB41 | 1007 | Tutorial (OLTP).Tutorial.product | 2017-05-11 00:23:4... |
| 200A04373E7DF68B15D99C79C68DBFC3 | 1006 | Tutorial (OLTP).Tutorial.product | 2017-05-11 00:23:4... |
| 3FE50916FC279E9291B59F26F5D226BD | 1008 | Tutorial (OLTP).Tutorial.product | 2017-05-11 00:23:4... |

Below the table, the "Results" pane shows the following messages:

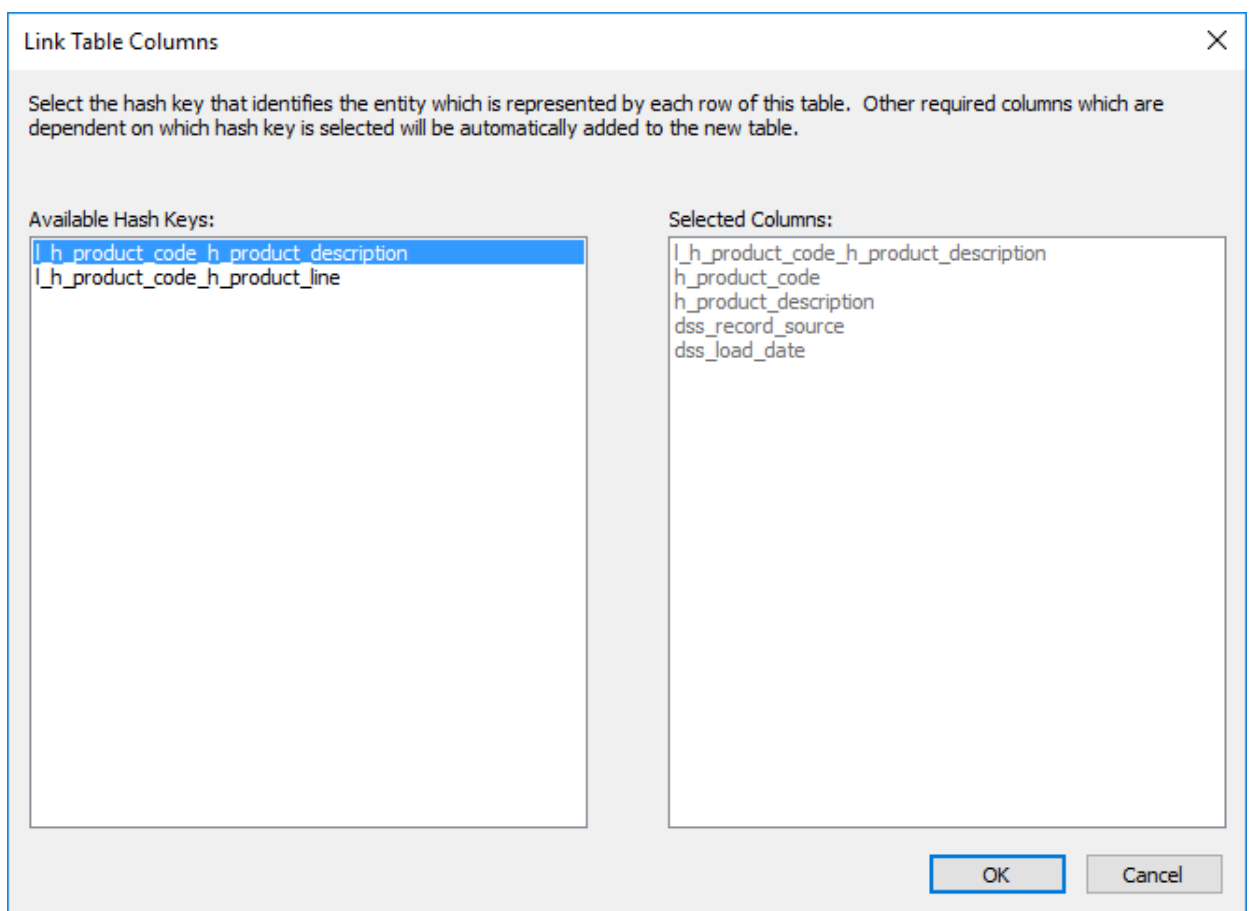
- Object
- Message
- h_product... SQL Statement completed.
- h_product... Generated Procedure from template 'wsl_teradata_proc_dv_perm'
- h_product... Procedure Completed
- h_pro... 1 h_product_DV updated. 9 new records.

The right pane shows a list of objects in the Data Warehouse, including "load_order_header_DV", "load_order_line", "load_product", "load_product_DV", "model_customer", "model_date", "model_forecast", "model_order_header", "model_order_line", "model_ship_date", "ods_forecast", "stage_customer", "stage_customer_DV", "stage_forecast", "stage_order_header", "stage_order_header_DV", "stage_order_line", "stage_product_DV", "stage_sales_detail", "view_customer", "view_forecast", "view_order_header", and "view_order_line".

CREATING THE LINK TABLE

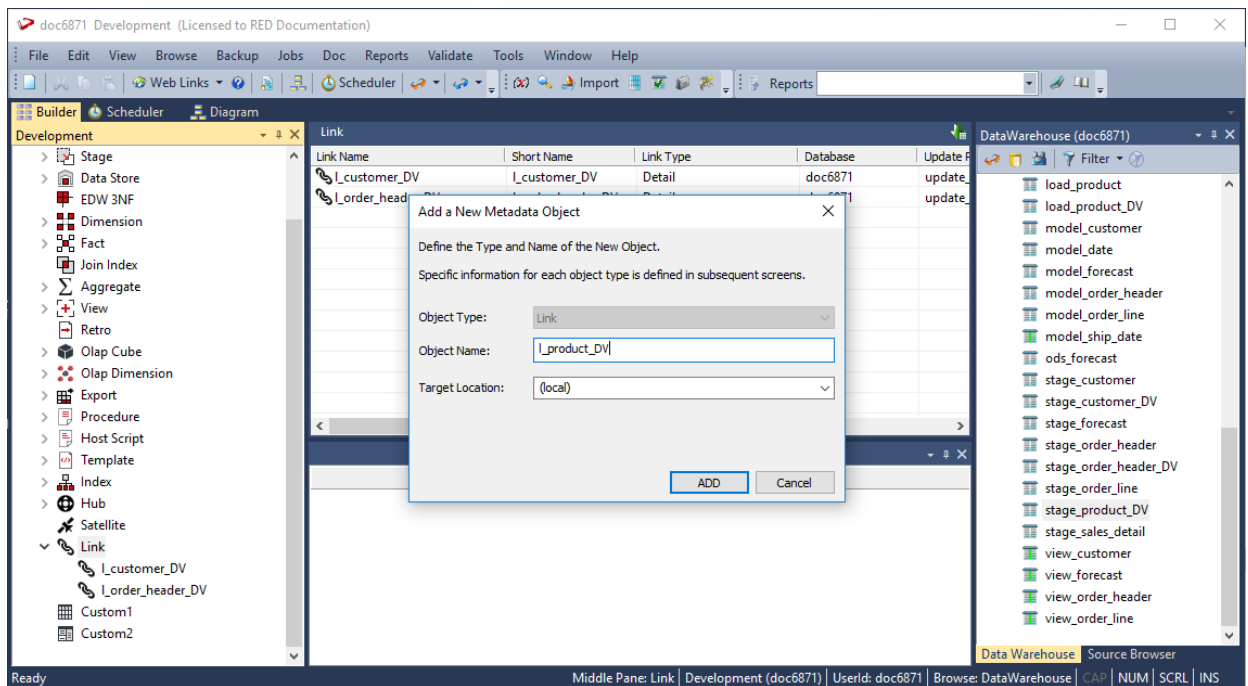
The steps for creating the Link table is similar to the steps used in creating the Hub table:

- 1 Browse to the Data Warehouse connection to create the Hub table.
- 2 Double-click the **Link** object group in the left pane, the middle pane displays a list of existing Link tables.
- 3 Click the source **Data Vault Stage** table from the right pane and drag it to the middle pane.
- 4 The Link table creation Wizard appears and prompts you to select the Hash Key to use from the **Available Hash Keys** pane. The columns that comprise the selected Hash Key are displayed under the **Selected Columns** pane—these are the columns that will be populated by the Wizard on the new **Link** table.

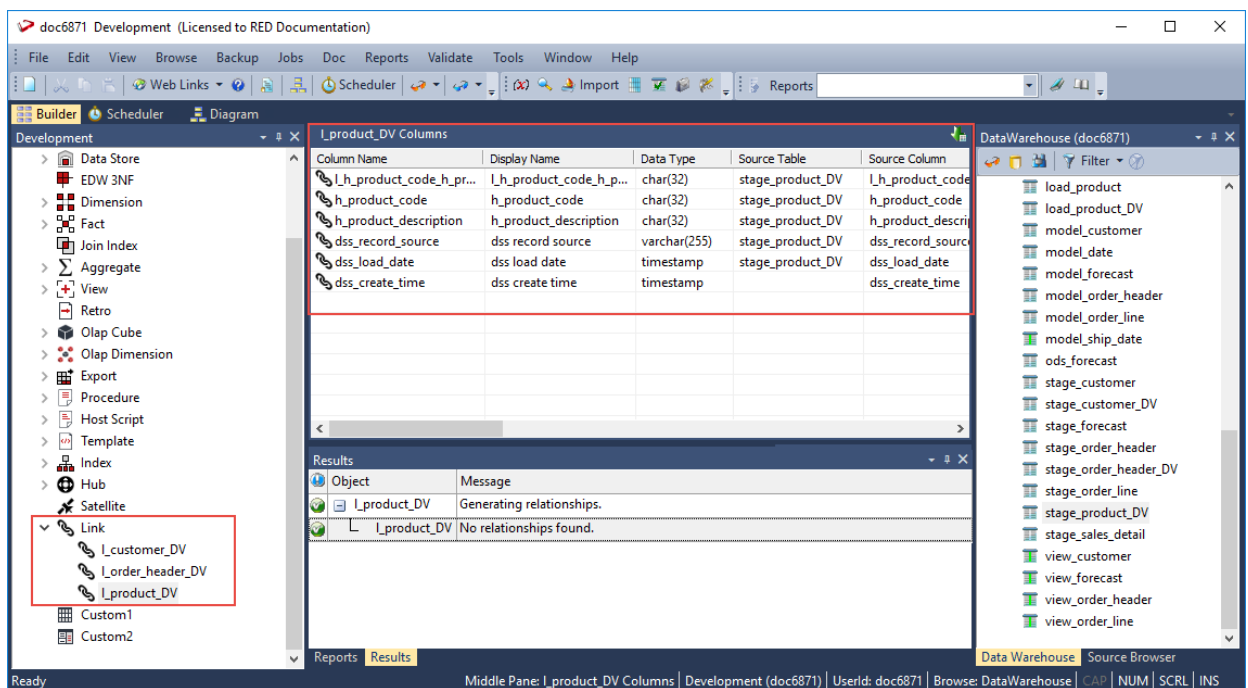


- 5 Select the Hash Key you want to use from the **Available Hash Keys** pane to see the columns that will be included in your Link table under the **Selected Columns** pane. Click **OK** to continue.

- The **Add a New Metadata Object** screen appears and classifies the new object as a Link table. It provides a default name based on the source **Data Vault Stage** table name. Accept the name or enter a new name for the Link table and click **ADD** to continue.

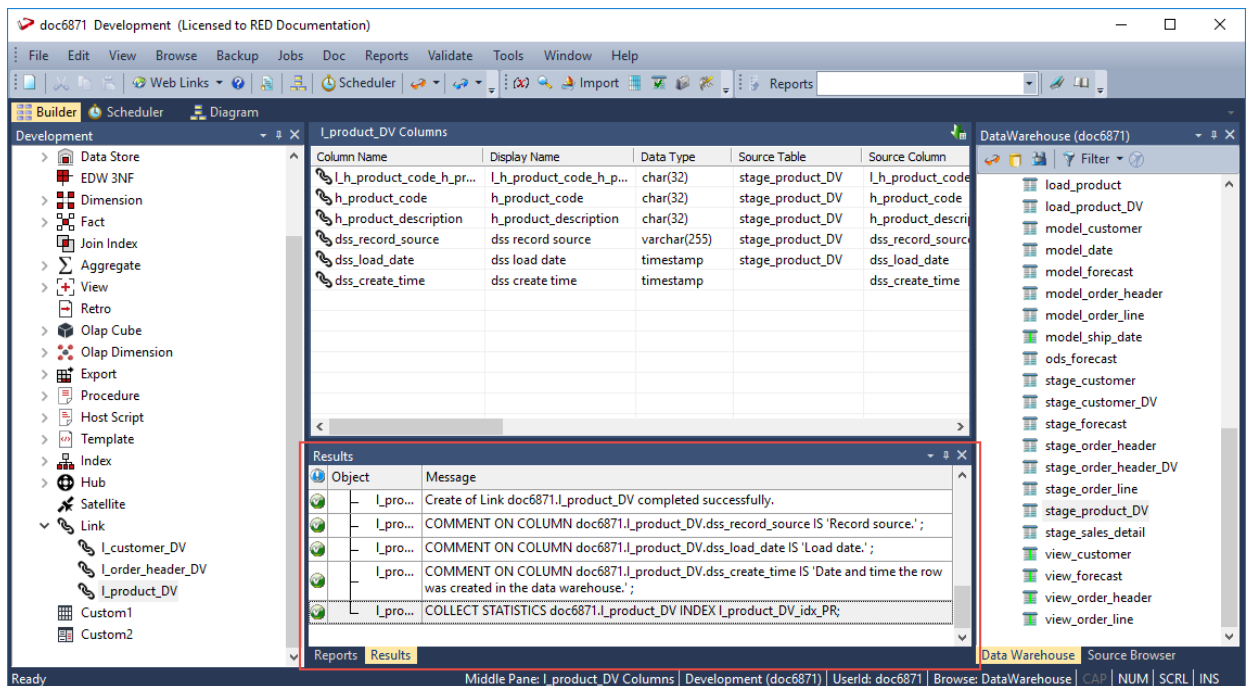


- Click the **OK** button on the **Table Properties** screen to finish defining the meta data for the Link table. The new Link table is added to the **Link** objects list in the left pane and the columns included in the table are listed in the middle pane.



- Right-click the new Link table you defined in the left pane and select **Create (ReCreate)** from the context menu to create the table.

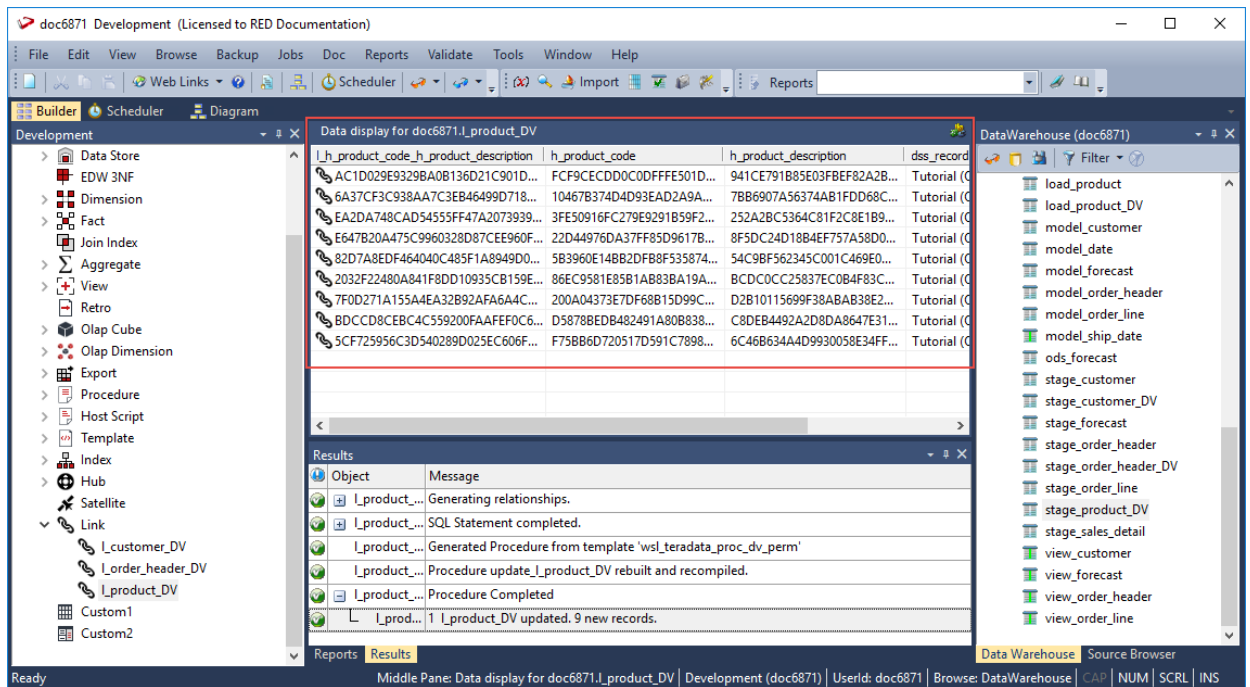
- Click Yes on the Primary Index prompt and specify **code** on the **Primary Index Columns** field and then click **OK**. The **Results** pane displays confirmation that the Link table was successfully created.



After the new Link table is defined and created, clicking the **Rebuild** button on the **Table Properties** screen launches the Wizard to generate the update procedure to populate the table. This Wizard utilizes a template to create the procedure.

The detailed steps for using this Wizard is described in the succeeding section, **Generating Update Procedures for the Hub, Link and Satellite Tables** (see "Generating Update Procedures for Hub, Link and Satellite Tables" on page **Error! Bookmark not defined.**).

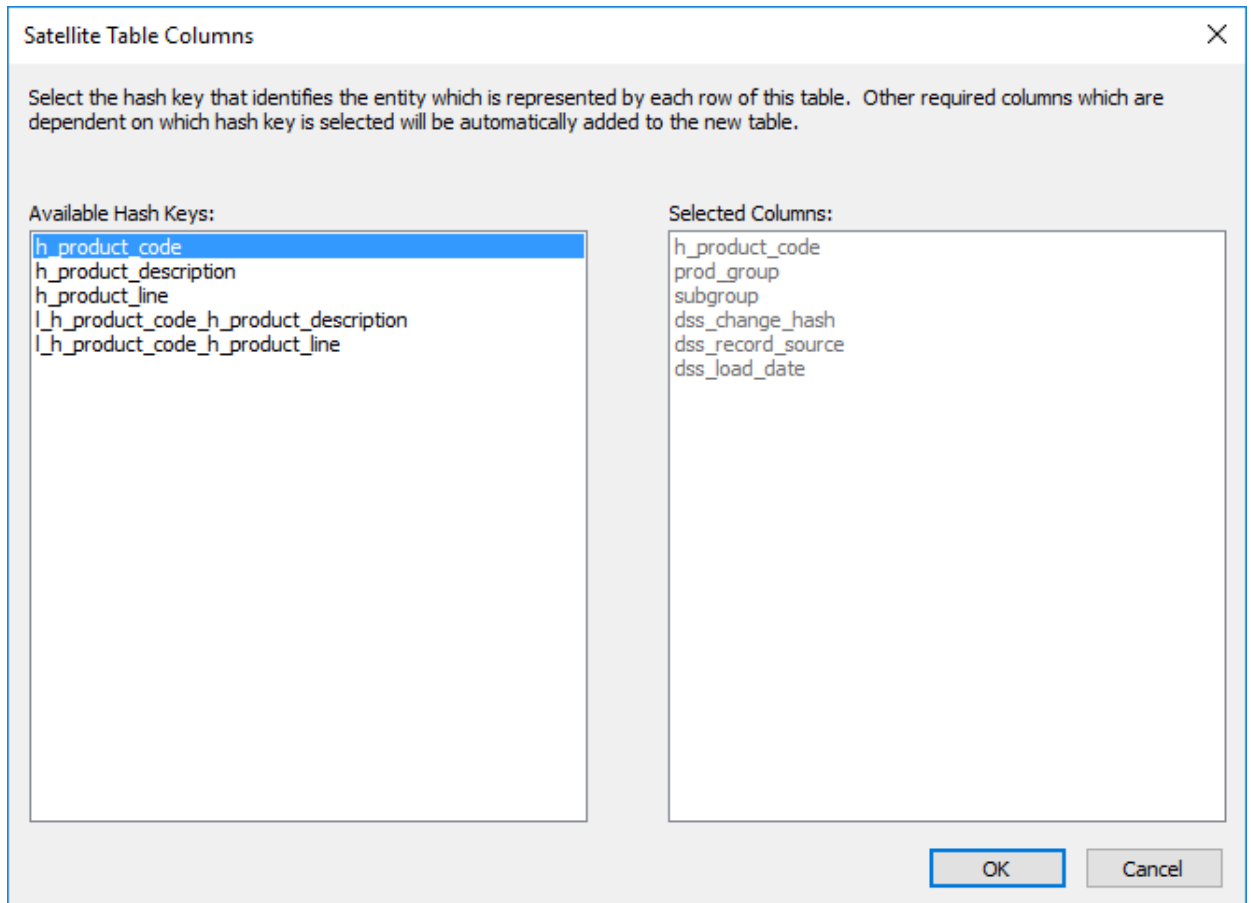
Once you have run the generated update procedures, you can view the generated Link Hash keys, by right-clicking the new Link table you created in the left pane and then selecting **Display Data** from the context menu:



CREATING THE SATELLITE TABLE

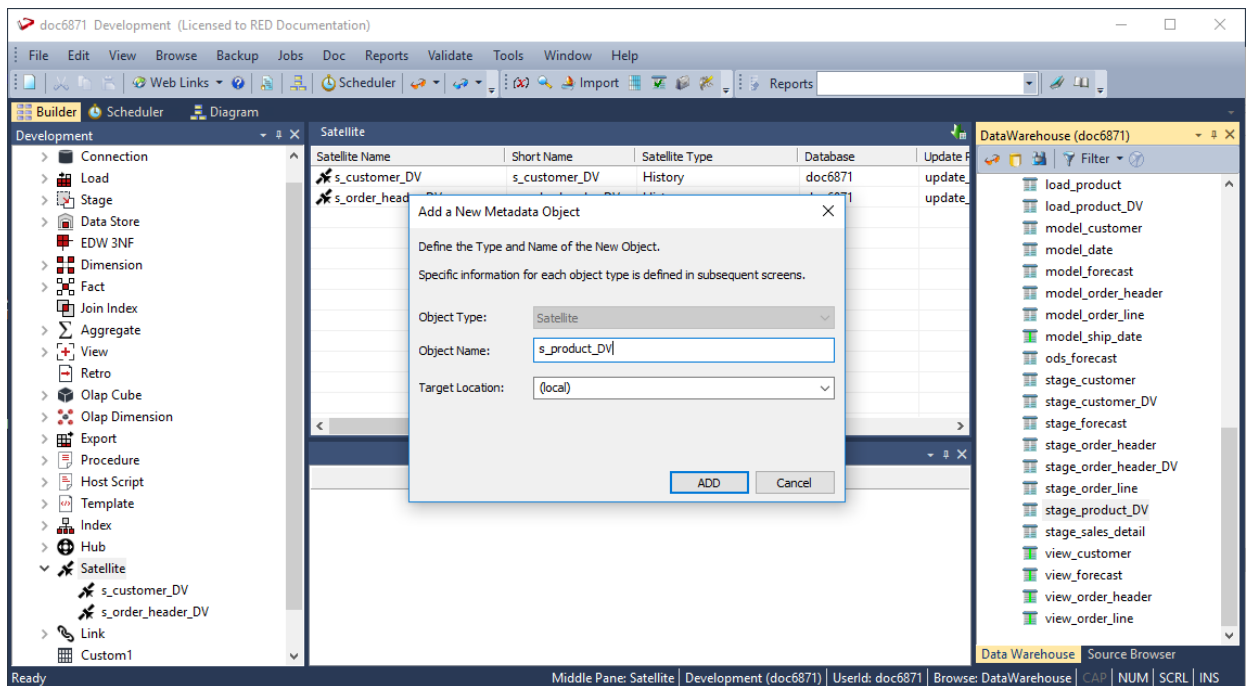
The following describe the steps for creating a Satellite table:

- 1 Browse to the Data Warehouse connection to create the Satellite tables.
- 2 Double-click the **Satellite** object group in the left pane, the middle pane displays a list of existing Satellite tables.
- 3 Click the source **Data Vault Stage** table from the right pane and drag it to the middle pane.
- 4 The Satellite table creation Wizard appears and prompts you to select the Hash Key to use from the **Available Hash Keys** pane. The columns that comprise the selected Hash Key are displayed under the **Selected Columns** pane—these are the columns that will be populated by the Wizard on the new Satellite table.

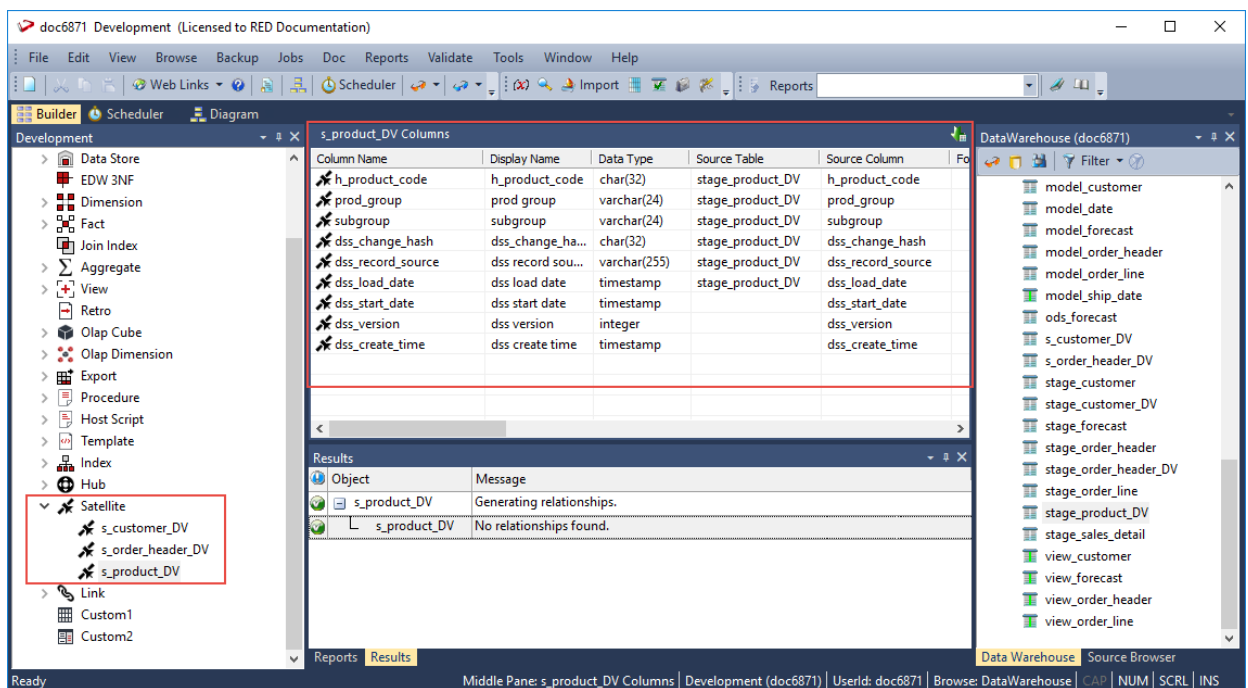


- 5 Select the Hash Key you want to use from the **Available Hash Keys** pane to see the columns that will be included in your Satellite table under the **Selected Columns** pane. Click **OK** to continue.

- The **Add a New Metadata Object** screen appears and classifies the new object as a Satellite table. It provides a default name based on the source **Data Vault Stage** table name. Accept the name or enter a new name for the Satellite table and click **ADD** to continue.

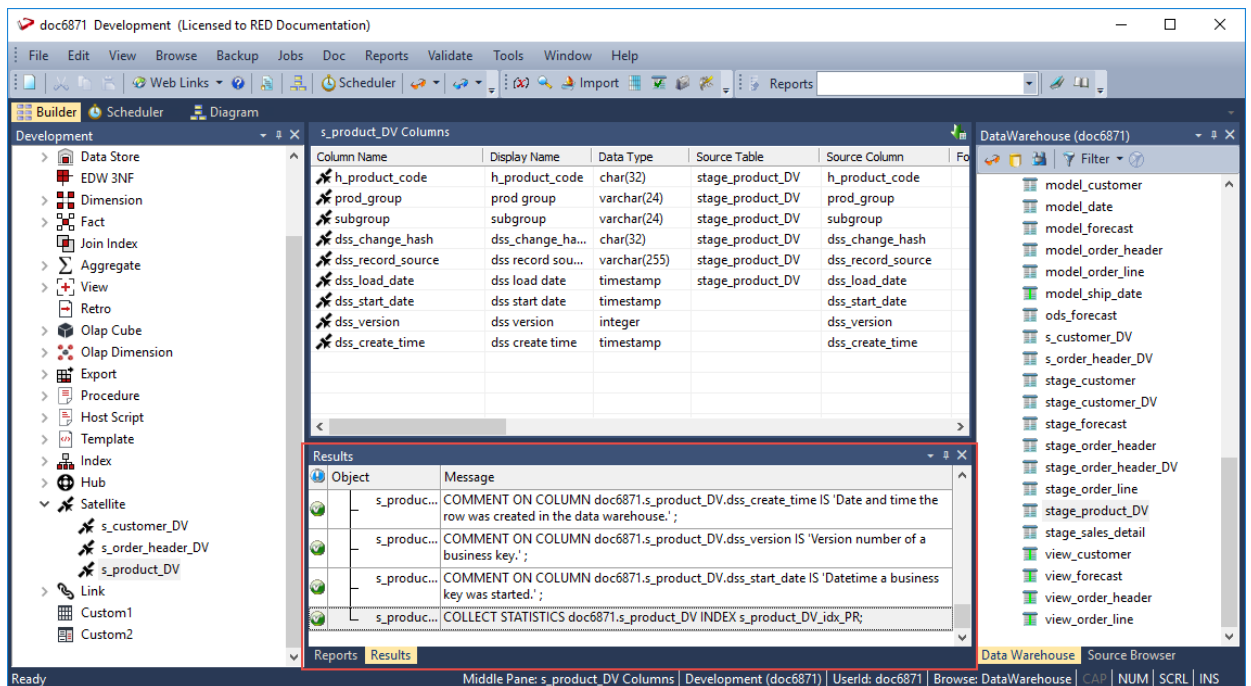


- Click the **OK** button on the **Table Properties** screen to finish defining the meta data for the Satellite table. The new Satellite table is added to the **Satellite** objects list in the left pane and the columns included in the table are listed in the middle pane.



- Right-click the new Satellite table you defined in the left pane and select **Create (ReCreate)** from the context menu to create the table.

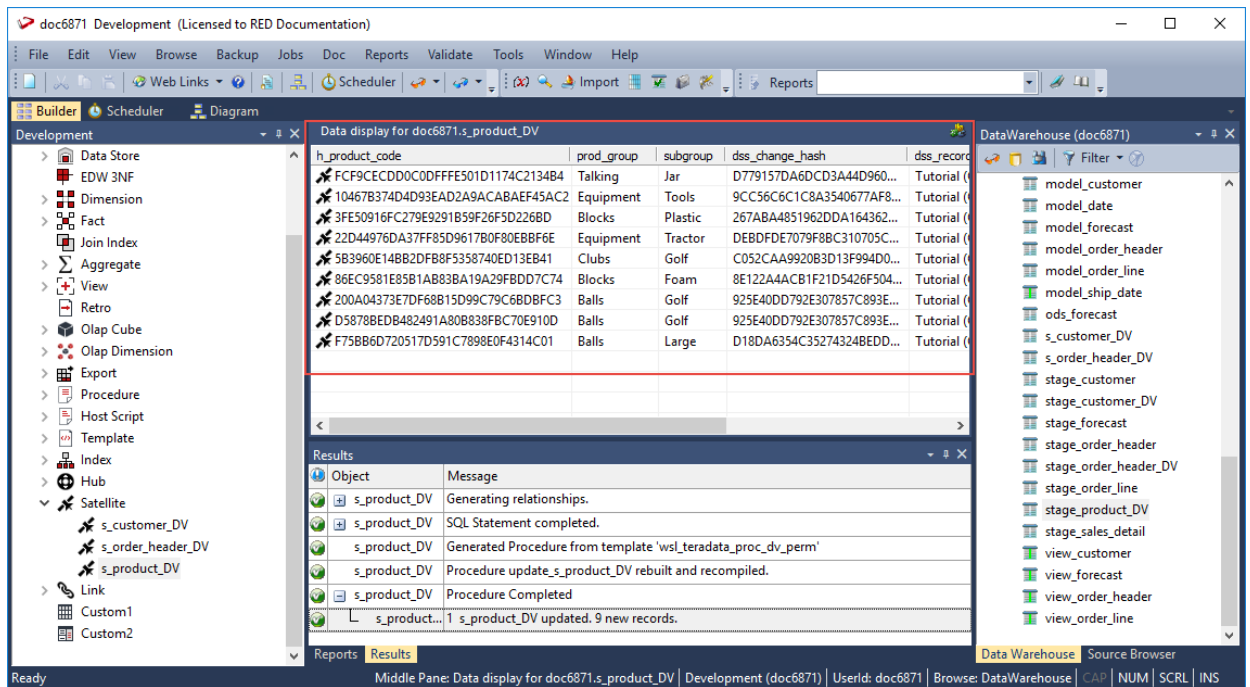
- Click Yes on the Primary Index prompt and specify **code** on the **Primary Index Columns** field and then click **OK**. The **Results** pane displays confirmation that the Satellite table was successfully created.



After the new Satellite table is defined and created, clicking the **Rebuild** button on the **Table Properties** screen launches the Wizard to generate the update procedure to populate the table. This Wizard utilizes a template to create the procedure.

The detailed steps for using this Wizard is described in the succeeding section, **Generating Update Procedures for the Hub, Link and Satellite Tables** (see "**Generating Update Procedures for Hub, Link and Satellite Tables**" on page **Error! Bookmark not defined.**).

Once you have run the generated update procedures, you can view the generated Satellite Hash keys, by right-clicking the new Satellite table you created in the left pane and then selecting **Display Data** from the context menu:



GENERATING UPDATE PROCEDURES FOR HUB, LINK AND SATELLITE TABLES

The following describe the steps for generating update procedures via a template.

HUB TABLE

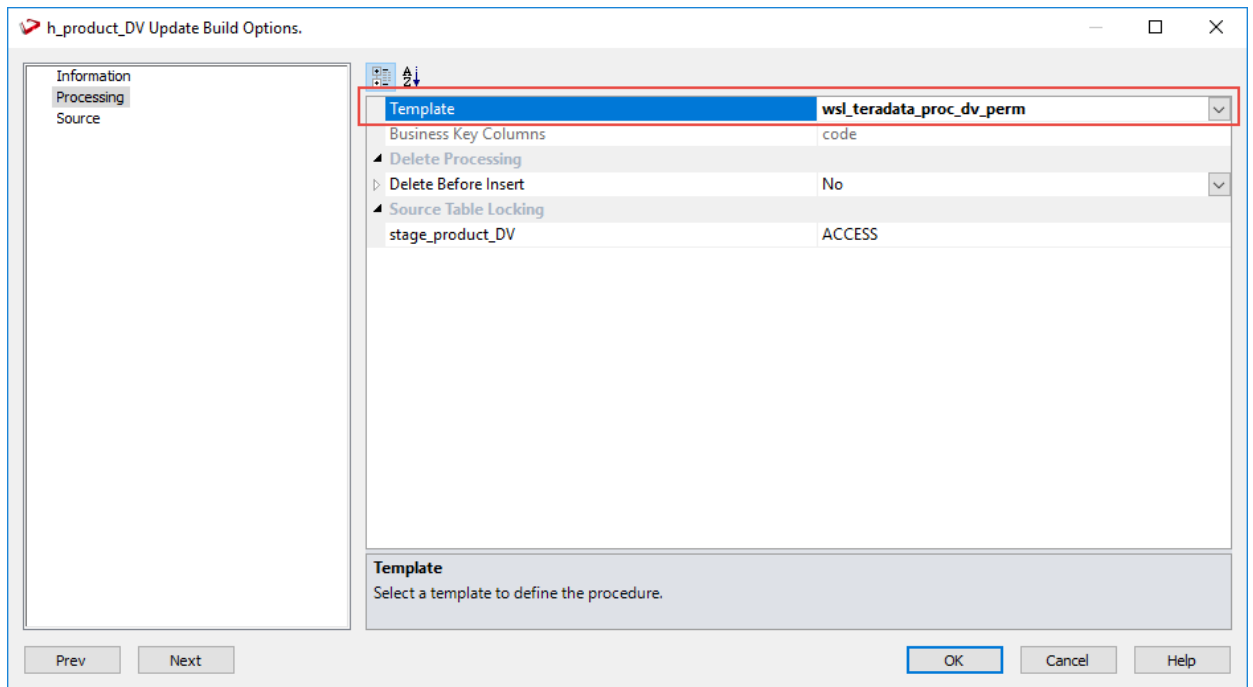
After successfully defining and creating the Hub table, you can generate the update procedure via a template to populate the table.

- 1 Click the **Rebuild** button on the **Table Properties** screen to launch the procedure generation Wizard to populate the table.

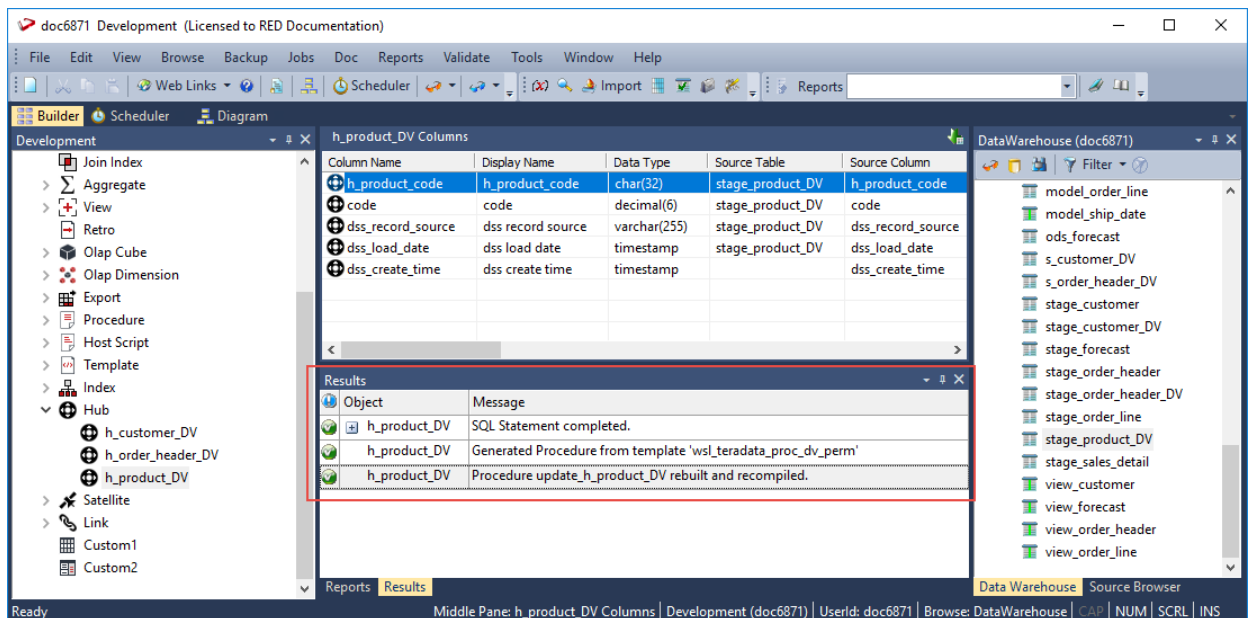
The screenshot shows the 'Table Properties' dialog for 'Hub h_product_DV'. The 'Table Name' is 'h_product_DV', 'Table Type' is 'Detail', 'Unique Short Name' is 'h_product_DV', and 'Business Display Name (EUL)' is 'h_product_DV'. The 'Update Procedure' dropdown is set to '(None)', and the 'Template' field contains 'wsl_teradata_proc_dv_perm'. The 'Rebuild' button is highlighted. The 'Custom Procedure' dropdown is also set to '(None)'. The 'Get Key Function' dropdown is set to '(None)', and the 'Mnemonic (EUL)' field is empty. The 'Timestamps' section shows 'Metadata Structure' as '2017-05-29 02:25:33.080000', 'Database' as an empty field, and 'Database Altered' as an empty field. The 'OK', 'Cancel', and 'Help' buttons are at the bottom right.

Note: RED displays the name of the previously used update procedure template by default, below the **Update Procedure** drop-down.

- On the **Processing** tab of **Table Update Build Options** screen, select the template to use from the **Template** drop-down or use the previous update procedure template.



- Click **OK** to proceed with the procedure generation. The **Results** pane displays confirmation that the procedure was generated.



- Right-click the Hub table in the left pane, under the **Hub** objects list and select **Code>View update** from the context menu to view the contents of the update procedure generated.

```

doc6871 Development (Licensed to RED Documentation)
File Edit View Format Compile Execute Tools Window Help
Builder Scheduler Diagram update_h_product_DV x
-----
-- DEMS Name      : Teradata
-- Procedure Name : update_h_product_DV
-- Template       : wsl_teradata_proc_dv_perm
-- Template Version : 6.9.1.0
-- Description    : Update the Hub table h_product_DV
-- Generated by   : WhereScape RED Version 6.9.1.0 (build 170512-133036 customer-preview)
-- Generated for  : RED Documentation
-- Generated on   : Friday, May 26, 2017 at 13:55:37
-- Author        : WhereScape Documentation
-----
-- Notes / History
--
CREATE PROCEDURE [METABASE].update_h_product_DV
(
  IN p_sequence      INTEGER
  , IN p_job_name    VARCHAR(256)
  , IN p_task_name   VARCHAR(256)
  , IN p_job_id      INTEGER
  , IN p_task_id     INTEGER
  , OUT p_return_msg VARCHAR(256)
  , OUT p_status     INTEGER
)
BEGIN
-----
-- Control variables used in most programs
-----
DECLARE v_msgstext  VARCHAR(255); -- Text for audit_trail
DECLARE v_sql       VARCHAR(255); -- Text for SQL statements
DECLARE v_step      INTEGER;      -- return code
DECLARE v_update_count INTEGER;   -- no of records updated
DECLARE v_insert_count INTEGER;   -- no of records inserted
DECLARE v_count     INTEGER;      -- General Counter
DECLARE v_sql_code  INTEGER;      -- SQL Error Code for Audit Trail
-----

```

- Right-click the Hub table in the left pane and select **Execute Update Procedure** from the context menu to run the procedure. The **Results** pane displays the number of records created.

| Column Name | Display Name | Data Type | Source Table | Source Column |
|-------------------|-------------------|--------------|------------------|-------------------|
| h_product_code | h_product_code | char(32) | stage_product_DV | h_product_code |
| code | code | decimal(6) | stage_product_DV | code |
| dss_record_source | dss record source | varchar(255) | stage_product_DV | dss_record_source |
| dss_load_date | dss load date | timestamp | stage_product_DV | dss_load_date |
| dss_create_time | dss create time | timestamp | stage_product_DV | dss_create_time |

| Object | Message |
|--------------|---|
| h_product_DV | SQL Statement completed. |
| h_product_DV | Generated Procedure from template 'wsl_teradata_proc_dv_perm' |
| h_product_DV | Procedure update_h_product_DV rebuilt and recompiled. |
| h_product_DV | Procedure Completed |
| h_product... | 1 h_product_DV updated: 9 new records. |

LINK AND SATELLITE TABLES

Follow the same steps described above to create and execute the update procedures for the Link and Satellite tables, via a template to populate the tables.

CHAPTER 15

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 377).

CHAPTER 16

MODEL TABLES

IN THIS CHAPTER

| | |
|---|-----|
| Model Table Overview | 451 |
| Building a Model Table..... | 453 |
| Generating the Model Table Update Procedure | 458 |
| Model Table Artificial Keys | 465 |
| Model Table Custom Procedure | 466 |
| Model History Tables..... | 466 |
| Generating History Table Update Procedures | 468 |
| Model Table Column Properties..... | 471 |
| Model Table Column Transformations | 476 |

MODEL TABLE OVERVIEW

Model objects are used to create EDW 3NF models in an enterprise data warehouse. They can contain surrogate keys to other model tables.

A model is built from the Data Warehouse connection. Unless you are doing a retro-fit of an existing system, model tables are typically built from one or more stage tables.

The normal steps for creating a model table are defined below and are covered in this chapter. The steps are:

- Identify the source transactional or reference data that will constitute the model table. If the data is sourced from multiple tables ascertain if a join between the source tables is possible.
- Using the 'drag and drop' functionality drag the load table that is the primary source of information for the model table into a model target. See ***Building a Model Table*** (on page 453).
- 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 model table and skip the next 4 steps. See ***Building a Model Table*** (on page 453).
- Add columns from other load tables if required. See ***Building a Model Table*** (on page 453).
- Create the model table in the database. See ***Building a Model Table*** (on page 453).
- Build the update procedure. See ***Generating the Model Table Update Procedure*** (on page 458).
- Run the update procedure and analyze the results. See ***Generating the Model Table Update Procedure*** (on page 458).

Modify the update procedure as required.

Model Keys

Model tables have up to 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 model table. Where the model 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 model table
- The customer code in a customer model table
- The calendar date in a date model table
- The 24 hour time in a time model table (i.e.HHMM) (e.g.1710)
- The airport short code in an airport model 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. For example the 'Where' clause in a model update may become:

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 model 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 model table record to another model table. When joining model tables to other model tables it would be possible to perform the join using the business key. For model 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 often to use some form of artificial key. A price is paid in the additional processing required to build the model table rows (particularly high volume transaction model tables rows), but this is offset by the reduced query times, storage and index sizes.

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.

Note: The default behavior of RED can be changed to not automatically add surrogate keys. See **Settings - Repository Attributes and Options** (see "**Settings - Repository Identification**" on page 74).

BUILDING A MODEL TABLE

Model Tables are often sourced from one table in the base application. The process for building a model table is the same for most other tables and begins with the drag and drop of the stage table that contains the bulk of the model information.

Drag and Drop

Create a model target by double clicking on the **Dimension object group** in the left pane. The middle pane will display a list of all existing Dimension tables.

Browse to the Data Warehouse via the Browse/Source Data menu option.

Drag the table that contains the bulk of the model 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 Dimension table and will provide a default name based on the load table name.

Either accept this name or **change the name to reflect the new model table** and click OK to proceed.

Model Table Properties

At this stage change the Table type to **Model Table** and change any other storage options if desired.

If prototyping, and the model table is simple (i.e. one source table) then it is possible to create, load and update the model table in a couple of steps.

If you want to do this select the **(Build Procedure...)** option from the 'Update Procedure' drop-down, and click **Create and Load** on the next dialog.

Create and Load

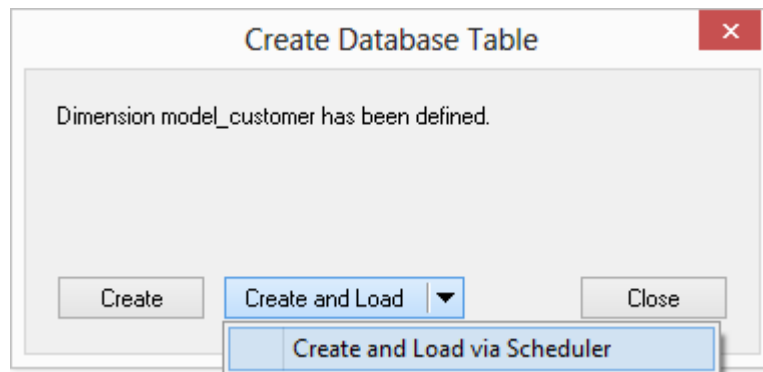
If you chose to build the update procedure the following dialog appears after clicking OK on the Properties page. This dialog asks if you want to create the Model table in the database and execute the update procedure.

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 Model Table Update Procedure* (on page 458) section.

If you have additional columns to add or columns to delete then select **Close** and proceed as follows.

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 model table will be displayed in the middle pane. At this stage it is possible to delete any unwanted columns by highlighting a column name or a group of names and clicking the **Delete** key.

The name of a column can also be changed at this stage 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 fuller description on what the various fields mean.

Note: 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 model 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 load tables and drag these columns into the middle pane.

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 model table can 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 |
|---------------------------------|---|
| <code>model_customer_key</code> | The unique identifier (artificial key) for the model table. This key is used in the joins to the fact table. It is generated via an identity associated with the table, except for the date model table where it has the form YYYYMMDD. If you have changed the default behavior of RED not automatically add surrogate keys, this column will not have been added. See <i>Settings - Repository Attributes and Options</i> (see " <i>Settings - Repository Identification</i> " on page 74). |
| <code>dss_start_date</code> | Used for model history tables. This column provides a date time stamp when the model table record came into existence. It is used to ascertain which model table record should be used when multiple are available. |
| <code>dss_end_date</code> | Used for model history tables. This column provides a date time stamp when the model table record ceased to be the current record. It is used to ascertain which model table record should be used when multiple are available. |
| <code>dss_current_flag</code> | Used for model history tables. This flag identifies the current record where multiple versions exist. |
| <code>dss_version</code> | Used for model history tables. This column contains the version number of a model table 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 model history table. |
| <code>dss_update_time</code> | Indicates when the record was last updated in the data warehouse. |
| <code>dss_create_time</code> | Indicates when the record was first created in the data warehouse |

Create the table

Once the model table has been defined in the metadata we need to physically create the table in the database. This is achieved by right-clicking on the model name 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 model 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 of finding such a column is to double click on the list heading 'Col name'. This will sort the column names into alphabetic order. Another double click on the heading will sort the columns back into their create order.

The next section covers the *Generating the Model Table Update Procedure* (on page 458).

GENERATING THE MODEL TABLE UPDATE PROCEDURE

Once a model table has been defined in the metadata and created in the data base an update procedure can be generated to handle the joining of any tables and the update of the model table records.

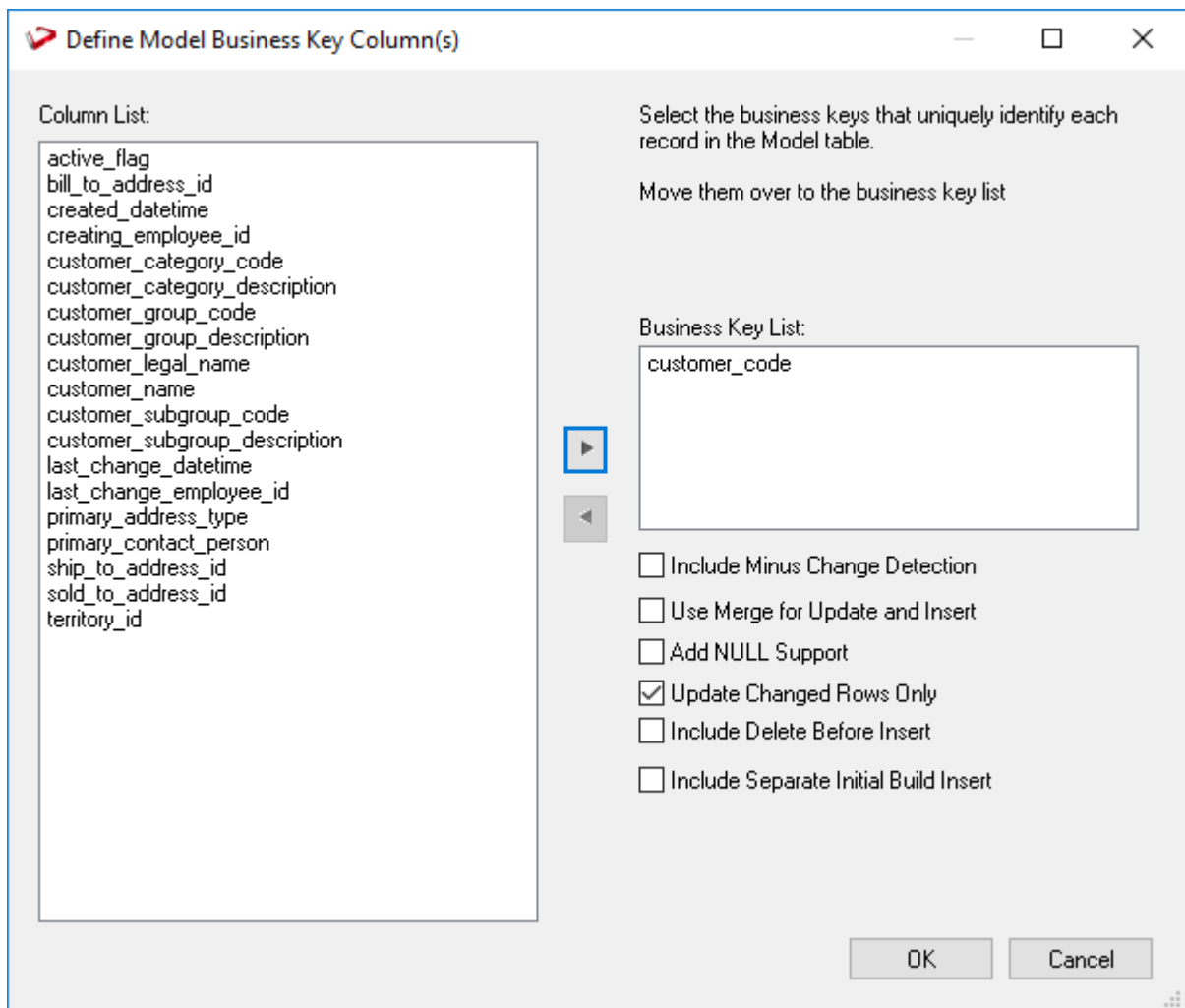
Note: You can also generate an update procedure via a template, refer to *Rebuilding Update Procedures* (on page 181) for details.

Generating a Procedure

To generate a procedure, right-click on the model 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.

Business Key definition

A dialog will appear asking for the business key that will uniquely identify each model table record. The source table from which the model 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 customer code is selected as the 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 model 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. It is of course possible to combine the two columns, but the normal practice would be to choose the code as the business key.

None of the columns chosen as the business key should ever contain a NULL value. See the note at the start of this chapter.

The **Include Minus Change Detection** checkbox will detect new rows using a minus sub-query rather than the default where not exists query. Enabling this option can significantly improve performance.

The **Use Merge for Update and Insert** checkbox will generate merge syntax. This option is only available for non-history model tables.

The **Include Separate Initial Build Insert** adds a second insert to the procedure to separately insert all data if the target model table is empty. This significantly improves performance with a large model table being loaded the first time.

Locking Request Modifier

Source Table: Specify a locking request modifier to be applied to each source table during generated update procedures. By default, this is set to 'ACCESS' which locks each row being accessed, a blank entry will result in no locking clause in the generated procedure.

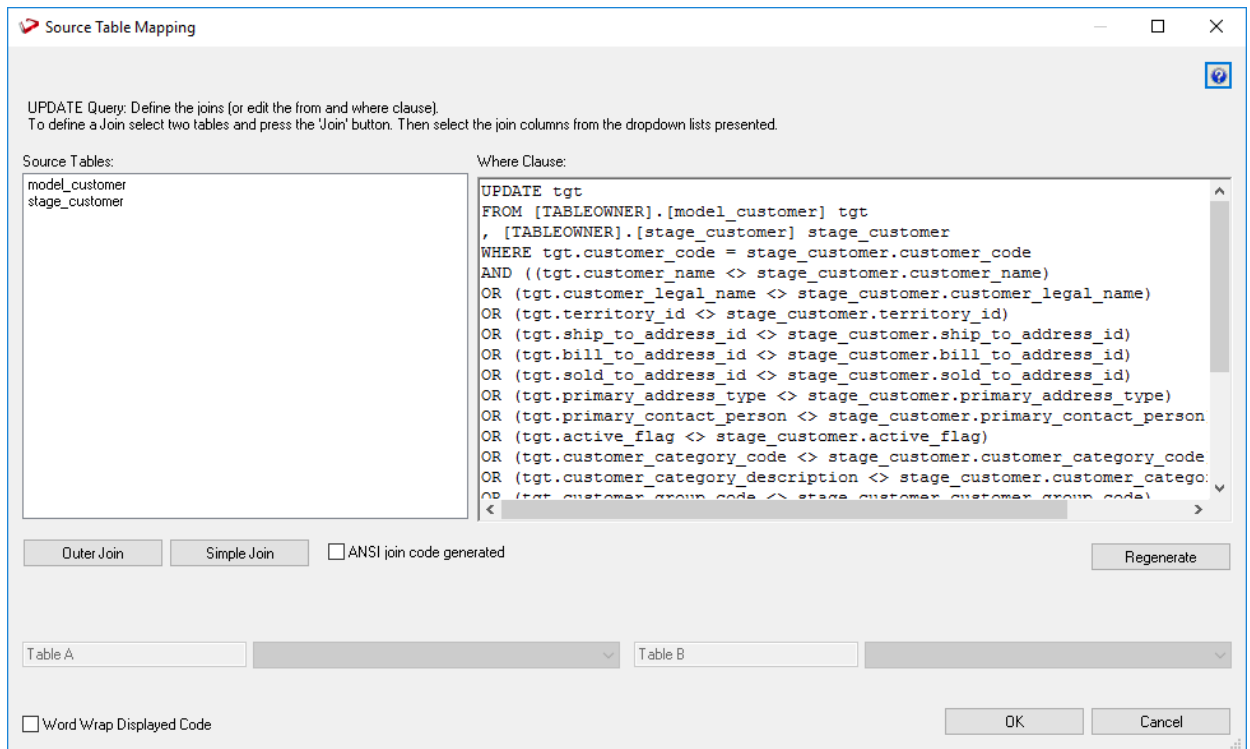
| Source Table | |
|----------------|--------|
| stage_customer | ACCESS |

Source Table
Locking Request Modifier to be applied to the respective source tables for each DML statement. A blank entry will exclude the locking clause in generated procedure.

OK Cancel

Source Table Mapping

WhereScape RED generates a default update statement with a 'Where' clause to join source and target tables together. The default update statement can be edited via the following dialog.



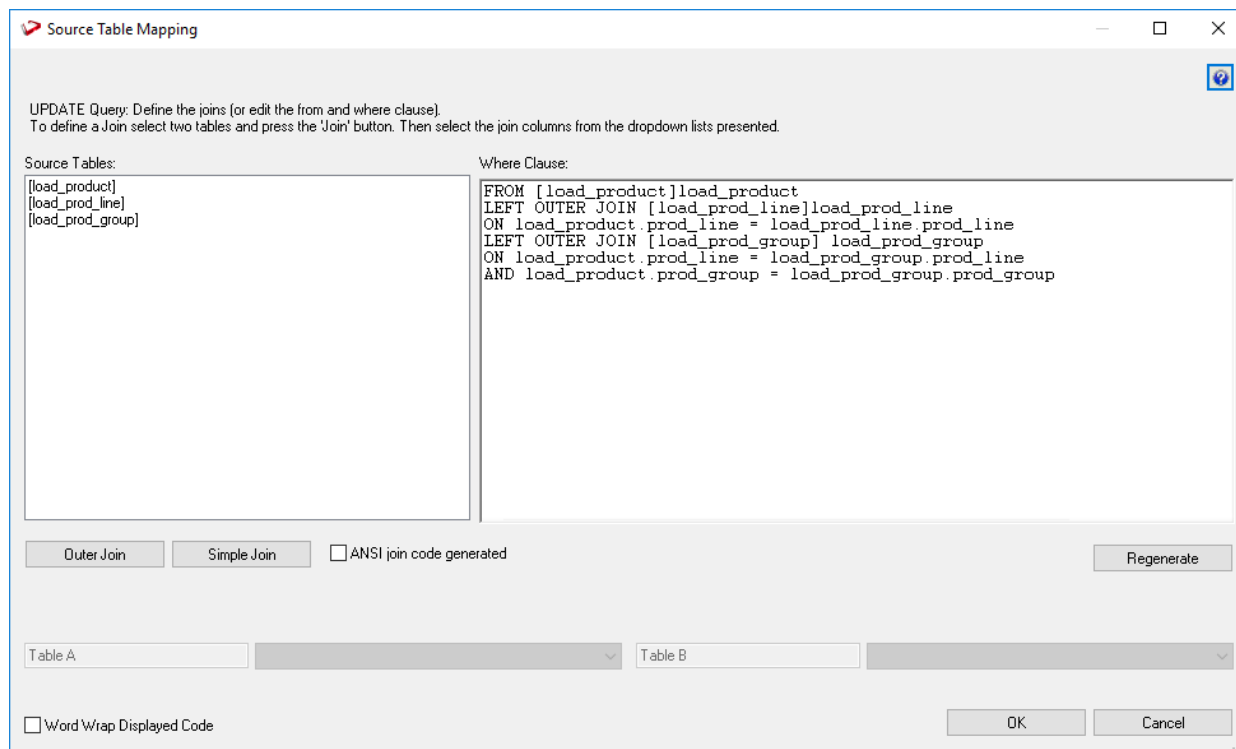
If there is more than one source table, additional joins will also have to be created. See [Joining multiple source tables](#) below.

Insert query where clause

If multiple source tables were used to build the model table, then a dialog box will appear prompting for the joins. This applies to the insert statement only in the generated procedure. See [Joining multiple source tables](#) for more information.

Joining multiple source tables

The example below shows the joining of the product, prod_line and prod_group tables as supplied with the tutorial data set.



Select two tables in the left box and then click one of the join buttons. The columns for the two tables then appear at the bottom of the dialog and one column is selected from each drop-down list to effect the join between the selected tables. In the example above, the load_product and load_prod_group tables are joined by two columns namely prod_line and group. 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 either a 'Where' clause or from clause join (ANSI). 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, and returns all rows in the master table, regardless of whether or not they are found in the second table. Therefore, 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 RED builds up a 'Where' clause join, it must place the outer join indicator next to the appropriate column. 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 or not 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.

Once satisfied with the join clause click the **OK** button to proceed to the next step. This clause will be a combined from and 'Where' clause. This clause can of course be edited in the procedure that is generated if not correct.

Only ANSI Outer Joins are available in Teradata.

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 in the RED meta repository to support the model table. The primary index is the only Active index. Secondary indexes can add significant performance cost during updates in Teradata, so these are defined in the RED meta repository but are not active (so are not created on the table). An example of the type of indexes defined is as follows:

| Index Name | Table | Columns | Type | Key Type | Rebuild Frequ... | Unique | Foreign | Pre Drop | Active | Partition | Parallel | Columns |
|----------------------|---------------|---------|---------|------------|------------------|--------|---------|----------|--------|-----------|----------|-------------------|
| model_product_idx_0 | model_product | 1 | Index | Artificial | Never | Y | N | N | N | | 0 | model_product_key |
| model_product_idx_A | model_product | 1 | Index | Business | Never | Y | N | N | N | | 0 | code |
| model_product_idx_PR | model_product | 1 | Primary | Primary | Never | N | N | N | Y | | 0 | code |

This example shows three indexes being created. They are:

- 1 A primary key constraint placed on the artificial key for the model table.
- 2 A unique index placed on the business key for the model table.
- 3 The primary index of the model table.

Only the third kind of index is active. To activate one of the other indexes as a secondary index, click the active checkbox in the index properties dialog.

Additional indexes can be added, or these indexes changed. See the chapter on indexes for further details.

MODEL TABLE ARTIFICIAL KEYS

The artificial (surrogate) key for a model table is set via an identity column. 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 model table is as follows:

| Key value | Usage |
|---------------|---|
| 1 upwards | The normal model table artificial keys are numbered from 1 upwards, with a new number assigned for each distinct model table record. |
| 0 | Used as a join to the model table when no valid join existed. It is the normal convention in the WhereScape generated code that any model 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 a model table is not appropriate for the record. For example, we may have a sales system that has a promotion model table. Not all sales have promotions. In this situation it is best to create a specific record in the model table 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 have to 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 model table only contains SKU based information. To be able to map the budget records to the model table 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 model table. |

MODEL TABLE CUSTOM PROCEDURE

A second procedure can be created on every model table. This is called the custom procedure. Rather than modifying the generated procedure, it is often more practical to make additions to the generated code in a separate procedure. This allows for regeneration of the model table's update procedure without losing changes (and having to reapply them).

The generated procedure for a custom procedure is template code. That is, a procedure that declares and initializes a variable, does nothing and returns the correct return code and message for the WhereScape RED scheduler.

MODEL HISTORY TABLES

Model history tables are a special type of model table where new records are created when certain identified columns in the model table change.

With any model table we identify a business key that uniquely identifies the model table records. For example in the case of the product model table from the tutorial the product code is deemed to be the business key. The code uniquely identifies each product within the model table. 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 model tables is what do we do when the name or description changes. Do we want to track transactional records in other model table 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 is 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 (using a model history table), or should the description of the product simply change to reflect its current name (a standard model table).

The decision is not a simple one and the advantages and disadvantages of each of the two choices is discussed below.

Model History Table

- Allows the most comprehensive analysis capabilities when just using the product model table.
- 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 any other model tables joined to this model table.

Model Table

- 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' model table.
- 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. Model history tables do have a place, but there is nearly always an alternate method that provides equal or better results. In the example above a promotion model history table coupled with the product model table could provide the same analysis results whilst still keeping product only analysis simple and easy to understand.

GENERATING HISTORY TABLE UPDATE PROCEDURES

Business Key definition

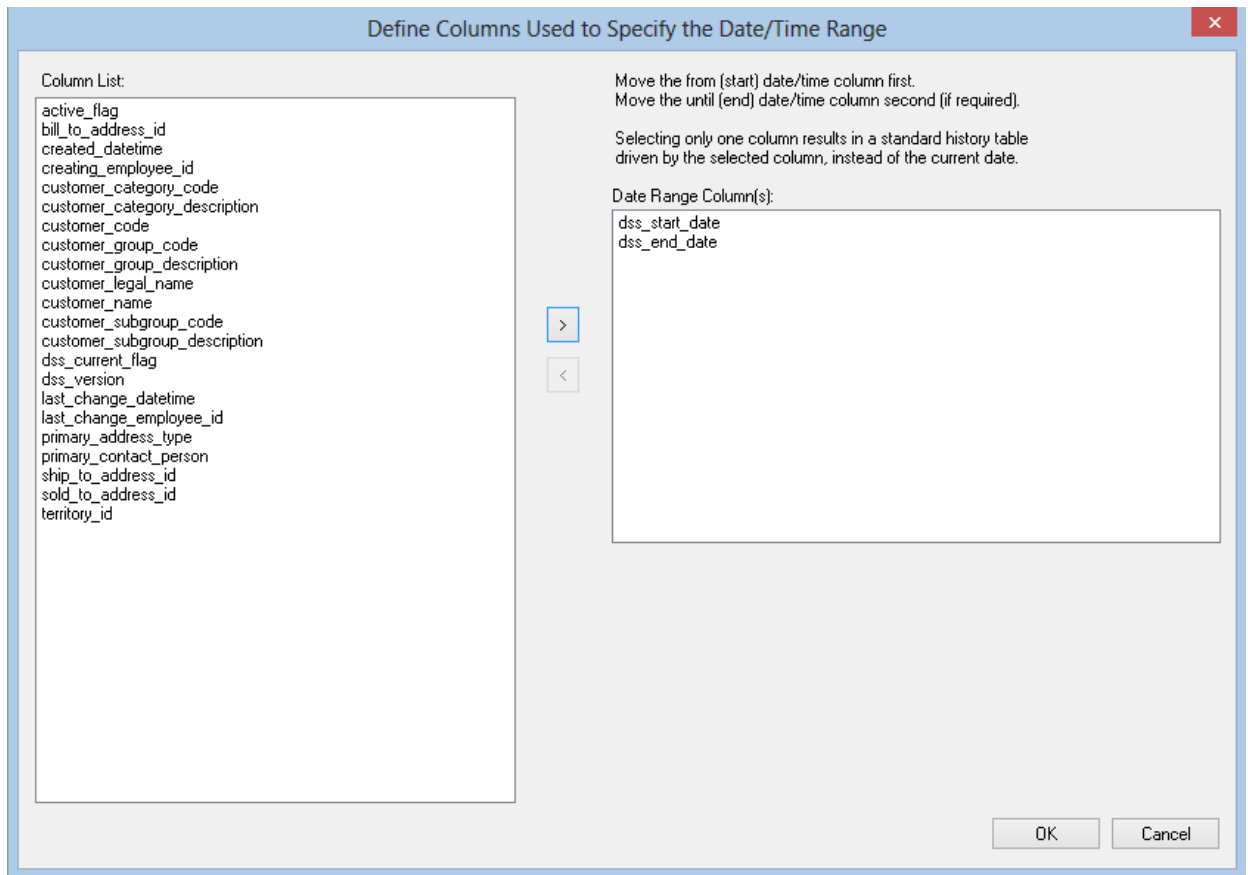
This is almost identical to standard model tables.

The screenshot shows a dialog box titled "Define Model Business Key Column(s)". On the left, under "Column List:", there is a list of columns including active_flag, bill_to_address_id, created_datetime, creating_employee_id, customer_category_code, customer_category_description, customer_group_code, customer_group_description, customer_legal_name, customer_name, customer_subgroup_code, customer_subgroup_description, last_change_datetime, last_change_employee_id, primary_address_type, primary_contact_person, ship_to_address_id, sold_to_address_id, and territory_id. On the right, under "Business Key List:", the column customer_code is selected. Below the lists are navigation arrows (> and <). At the bottom, there are three checkboxes: "Source System Supplied Start and End Dates" (checked), "Include Delete Before Insert" (unchecked), and "Include Separate Initial Build Insert" (unchecked). "OK" and "Cancel" buttons are at the bottom right.

An additional check-box, **Source System supplied Start and End dates** is supplied. This allows for start and end dates to be specified in the source data.

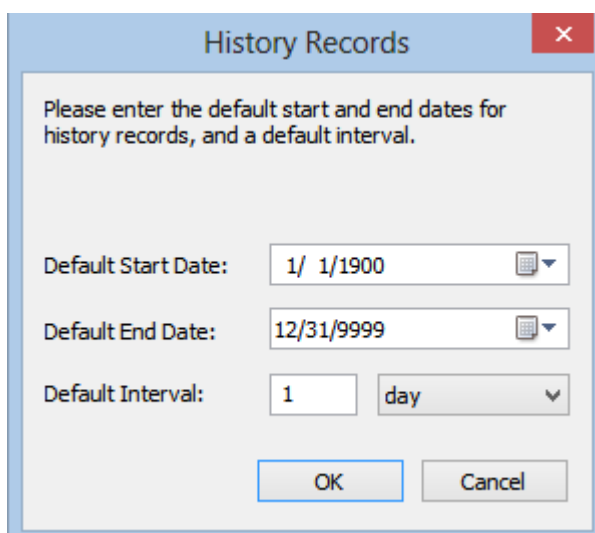
Source System supplied Start and End dates

If Source System supplied Start and End dates are enabled, the define start and end date dialog is displayed. Choose the **start date** first, then the **end date** and click **OK**.



Default Start and End Dates

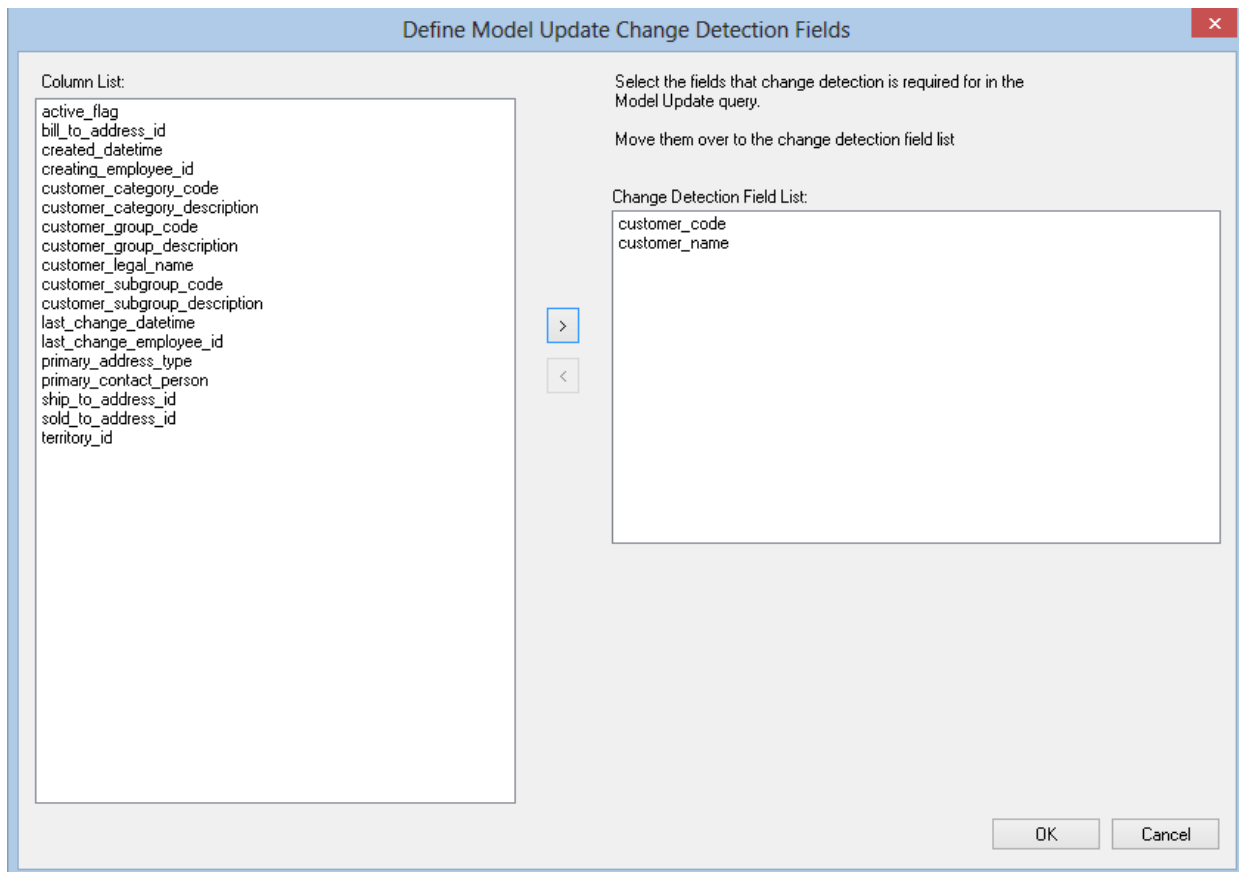
The `dss_start_date` and `dss_end_date` columns are used to track the start and end dates of history records. Defaults must be entered to ensure the null values are not placed in these columns. Use the history records dialog to enter these.



The default interval and interval type should also be chosen. These provide the difference between the end of the previous record and the start of the new record.

History columns

The change detection fields dialog requests the selection of the columns to be managed as history columns. Select the required columns and click **OK** to proceed. In the example below; columns `customer_code` and `customer_name` are to be managed as history columns.



Update query statement

This is the same as standard model tables.

Insert query where clause

This is the same as standard model tables.

Joining multiple source tables

This is the same as standard model tables.

MODEL TABLE COLUMN PROPERTIES

Each model 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.



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:

Dimension Column model_date.calendar_date

Properties

Transformation
Language Mapping

General

| | |
|--------------------|---------------------------------|
| Table Name | model_date |
| Column Name | calendar_date |
| Column Title | calendar date |
| Column Description | The calendar date for this row. |

Physical Definition

| | |
|------------------------------|-------------------------------------|
| Column Order | 1949 |
| Data Type | date |
| Null Values Allowed | <input checked="" type="checkbox"/> |
| Default Value | |
| Character Set | |
| Format | |
| Character Comparison/Sorting | |
| Compress | <input type="checkbox"/> |

Meta Definition

| | |
|------------------------|-------------------------------------|
| Numeric | <input type="checkbox"/> |
| Additive | <input type="checkbox"/> |
| Attribute | <input checked="" type="checkbox"/> |
| End User Layer Display | <input checked="" type="checkbox"/> |
| Business Key | <input checked="" type="checkbox"/> |
| Artificial Key | <input type="checkbox"/> |
| Key Type (0,A,B,C,...) | A |

Source Details

| | |
|---------------|---------------|
| Source Table | model_date |
| Source Column | calendar_date |

Column Name

Database-compliant name of the column.
Dialog Opening Value: calendar_date

<- Update Update -> **OK** Cancel Help

The two special update keys allow you to update the column and step either forward or backward to the next columns properties.

Table Name

Database-compliant name of the table that contains the column. [Read-only].

Column Name

Database-compliant name of the column. A good practice is to only use alphanumeric, 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.

Column Title

Name that the business uses to refer to the column. It does not affect the physical table definition, but rather provides input to the documentation and to the view **ws_admin_v_dim_col** which can be used to assist in the population of a end user tool's end user layer. 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 might 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 a valid for the target database. Typical Teradata databases often have integer, numeric(), varchar(), char(), date and timestamp data types. 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.

Character Set

Database-compliant table column character-set used for storage. Select Latin or Unicode.

Format

Database-compliant table column format. 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. 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.

Character Comparison/Sorting

Determines how the column character values are treated for comparison and sorting operations. Choose from: case specific, not case specific, uppercase case specific or uppercase not case specific.

Compress

Indicates whether the table column values are compressed when stored.

Compress/Compress Value

Optional list of values to be compressed. By default, only NULL is compressed if no list of values is specified.

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. This indicator is set and cleared by WhereScape RED during the dimension update procedure generation process. This checkbox 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. 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 model table join. Used on model tables to indicate the model keys to other model tables. Results in indexes being defined for the columns. Set during the update procedure generation for a model table, based on information from the staging table. |
| 3 | Not used in WhereScape RED for Teradata. |
| 4 | Not used in WhereScape RED for Teradata. |
| 5 | Indicates a column is a start date column. |
| 6 | Indicates a column is a end date column. |
| 7 | History column indicator. Used on model history tables to indicate that the column is being managed as a history column within the context of a model history table. Set when a column is identified during the model history update procedure generation. |

| Key type | Meaning |
|----------|--|
| 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, stage table or another model 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 model 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 model table key where a model table is being joined.

Transformation

Transformation. [Read-only].

MODEL TABLE COLUMN TRANSFORMATIONS

Each model 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 possible to do transformations on model table columns. It is recommended that transformations are not performed on columns that are model keys or the business keys for the table.

The transformation screen is as follows:

The screenshot shows a dialog box titled "Dimension Column model_date.calendar_date". On the left, there is a sidebar with three tabs: "Properties", "Transformation" (which is selected and highlighted in blue), and "Language Mapping". The main area of the dialog contains the following elements:

- Target:** A text box containing "calendar_date" and a "Paste" button to its right.
- Source:** A text box containing "model_date.calendar_date" and a "Paste" button to its right.
- Column Transformation Code (must execute within a SQL SELECT statement):** A large, empty text area for entering the transformation code.
- Function Set:** A dropdown menu currently set to "Default Teradata".
- Functions and Available Columns:** A panel on the right side of the code area, containing a "Functions" section with a plus icon and an "Available Columns" section with a plus icon.
- Word Wrap Displayed Code:** A checkbox that is currently unchecked.
- Function Syntax:** A text box for entering the function syntax.
- Function Desc.:** A text box for entering the function description.

At the bottom of the dialog, there are five buttons: "<- Update", "Update ->", "OK" (highlighted in blue), "Cancel", and "Help".

Note: Transformations are only put into effect when the procedure is re-generated.

See *Transformations* (on page 593) for more details.

CHAPTER 17

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.

IN THIS CHAPTER

| | |
|--|-----|
| Detail Fact Tables | 478 |
| Fact Table Column Properties | 485 |
| Fact Table Column Transformations..... | 491 |
| Fact Table Language Mapping..... | 493 |

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. Such 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.

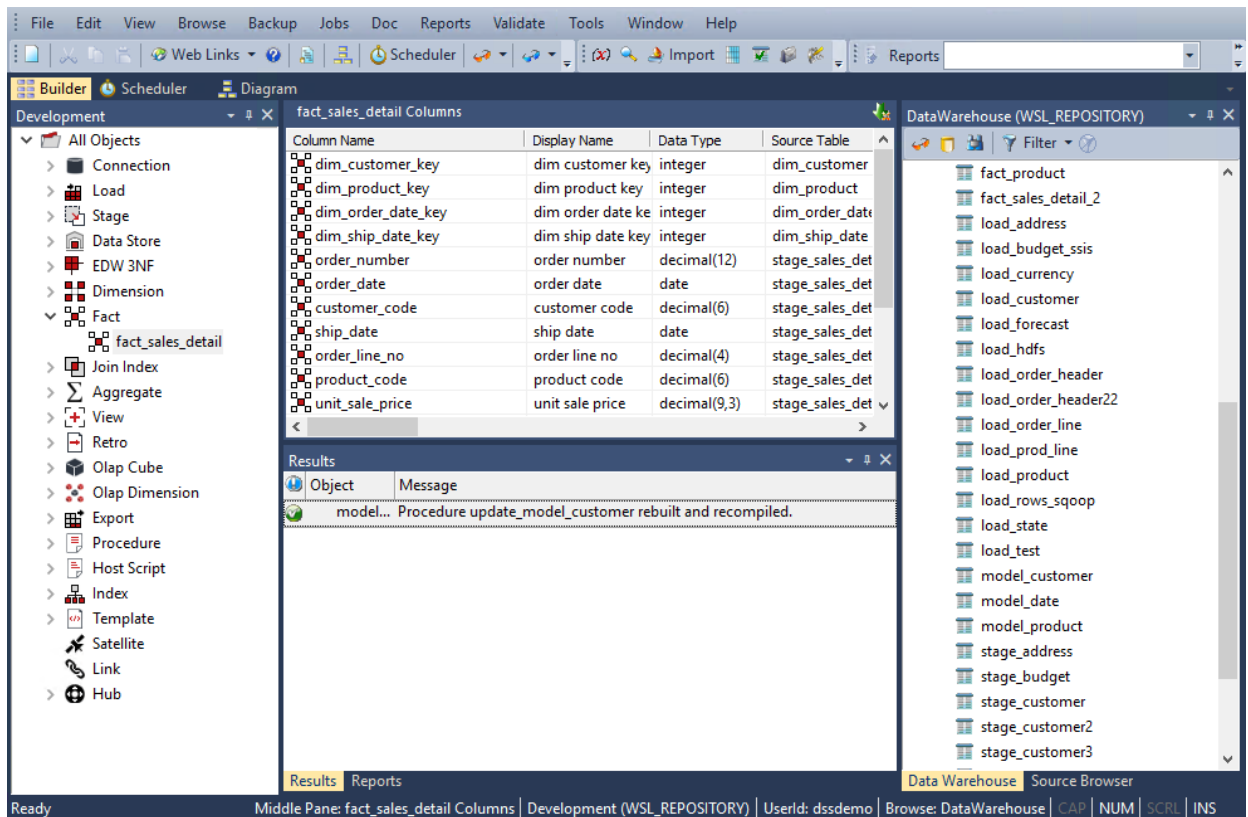
CREATING DETAIL FACT TABLES

A detail fact table is typically created by dragging a staging table onto a fact table list.

In the following example screen the fact table list has been produced by double clicking on the Fact Table object group under the Sales project. A list is produced showing the existing fact tables.

A **fact_sales_detail** detail fact table can be created by selecting the stage_sales_detail name in the right pane of the builder window, holding down the left mouse button and dragging the table into the middle (fact table list) pane.

Once released, the dialog to create the fact table will start.



When a staging table is dragged into the list window (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.

Note: You can also generate an update procedure via a template, refer to *Rebuilding Update Procedures* (on page 181) for details.

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.

Define Fact Procedure Type and Options

- The first dialog displayed when generating a detail fact table update procedure is the define Fact Procedure Type and Options dialog.
- Several other fields need to be set or adjusted on this dialog to ensure the required type of update procedure is generated.
- Once the required options have been selected, click **OK** to proceed to the next dialog.

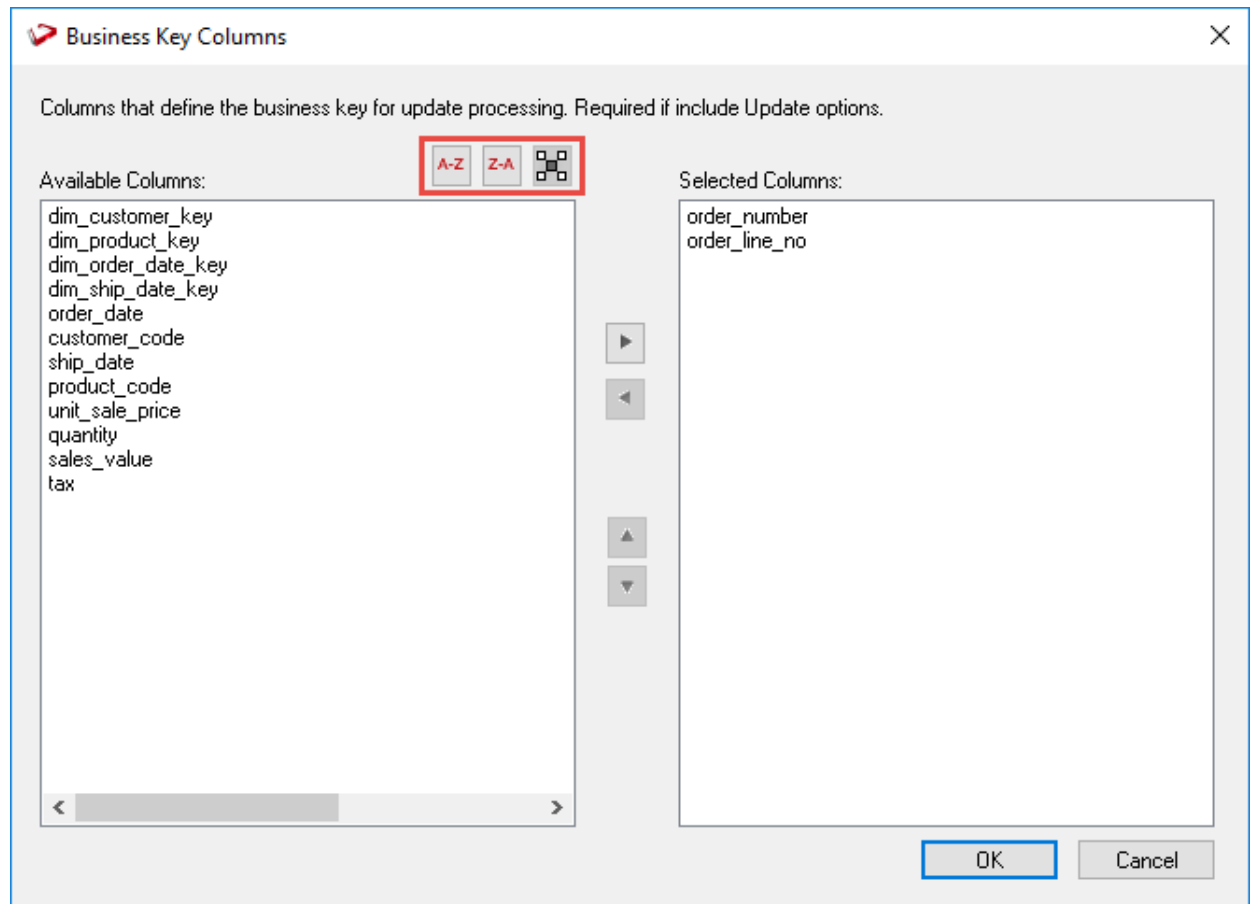
| Option | Value |
|---------------------------------|-------------------------------------|
| Template | None |
| Business Key Columns | order_number, order_line_no |
| Parameters | |
| Include Initial Load Insert | <input type="checkbox"/> |
| Create Storage Index After CTAS | <input type="checkbox"/> |
| Batch Processing | |
| Process by Batch | <input type="checkbox"/> |
| Explicit Locking | |
| Include Explicit Lock | <input checked="" type="checkbox"/> |
| Delete Processing | |
| Delete Before Insert | No |
| Update Processing | |
| Process Method | Insert/Update |
| Source Table Locking | |
| stage_sales_detail | ACCESS |
| Insert Method | |
| Include Insert Statement | <input type="checkbox"/> |
| Update Method | |
| Include Update Statement | <input type="checkbox"/> |

Template

Enables you to generate update procedures via a *template* (see "*Rebuilding Update Procedures*" on page 181).

Define Fact Business Key Columns

The next dialog displayed 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_number` and `order_line_no` are selected for the business key list.



TIP: 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.

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.

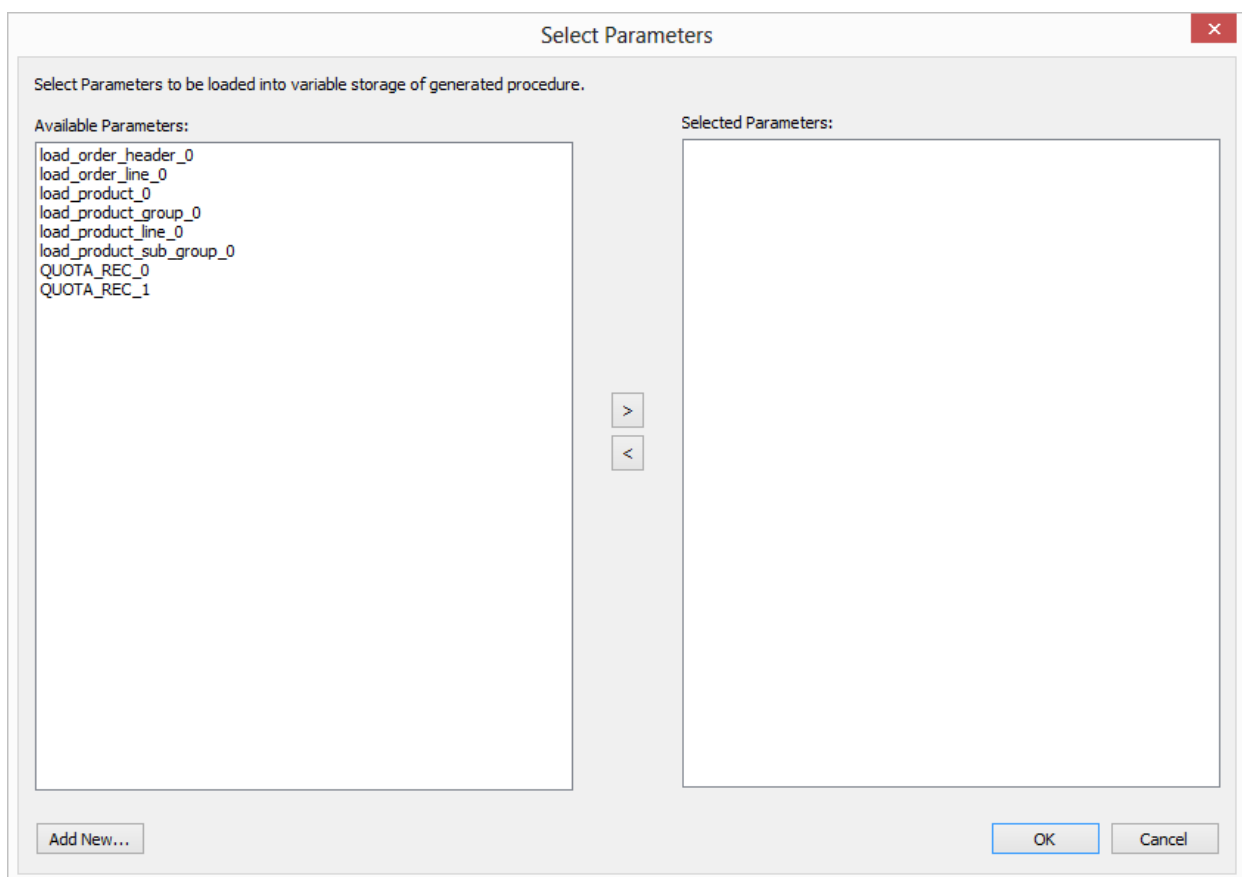
NULL Values: none of the columns chosen as the business key should ever contain a NULL value.

Select Parameters

The next dialog displayed is the Select Parameters dialog. If WhereScape RED parameters exist in the metadata, the following dialog is displayed. Any parameters selected in this dialog (by moving them to right side), 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.

Select a parameter by clicking on the parameter and then the > arrow to move it to the right column; then click **OK**.

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.



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.



TIP: WhereScape RED parameters should be unique within the first 28 characters to avoid conflicting variables names.

See *Parameters* (on page 132) for more information on WhereScape RED Parameters.

Include initial load insert

The **include initial load insert** option adds an additional insert statement to the update procedure that runs if the target Data Store 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

The **process by batch** field allows the user to select a column to use to break up the data being processed in a loop based on the distinct values in the column. 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).

Batch Processing Field - allows selecting a field to batch process on. If you select a date field you will have the ability to 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

The **delete before insert** 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. If this option is set Issue a Warning if a Delete Occurs and Delete Where Clause fields are enabled. The default for this field is off (i.e. a delete statement is not added to the procedure).

Truncate - if this option is chosen, the delete is ignored and the Fact Object is truncated.

Issue warning if a delete occurs: sets the procedure to a warning state if any deletes occur.

Delete 'Where' clause: the delete where clause that is appended to the generated delete statement to constrain the rows deleted.

Update processing

The update **process method** option allows you to use the Merge statement instead of two separate Insert and Update statements. The default value for this option is Insert/Update.

Source table locking

This section allows a locking request modifier to be specified for each source table. The specified locking request modifier is applied to each source table during generated update procedures. By default this is

set to 'ACCESS' which locks each row being accessed, a blank entry will result in no locking clause in the generated procedure.

Insert method

The **include insert statement** option includes an insert statement in the procedure to insert new rows in the Fact Object. If this option is chosen, then the **New rows only** option is available. Choosing this option, displays the **code type** drop-down, enabling the generated update statement to use either a sub-select with *except* change detection or *checksum* change detection to work out what rows in the source table(s) are new. If this option is turned off, the update procedure will not contain an insert statement. The default for this field is on (i.e. an insert statement is included).

Update method

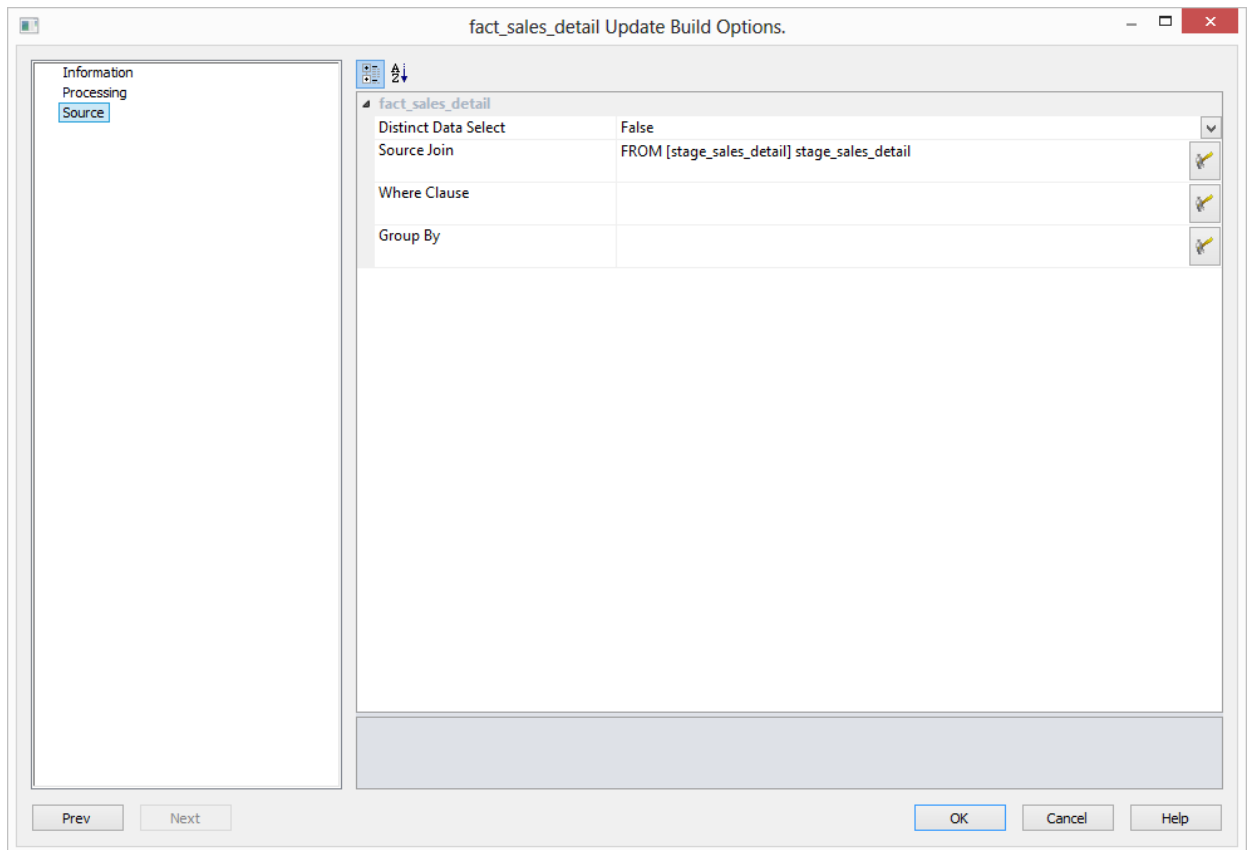
The **include update statement** option includes an update statement in the procedure to update changing rows in the Fact Object. If this option is chosen, then the **Changed rows only** option is available. Choosing this option, displays the **code type** drop-down, enabling the generated update statement to use either a sub-select with *except* change detection or *checksum* change detection to work out what rows in the Fact Object require updating. If this option is turned off, the update procedure will not contain an update statement. The default for this field is on (i.e. an update statement is included).

Changed and New Rows Only

Choosing this option, displays the **code type** drop-down, enabling the generated merge statement to use either a sub-select with **Join** change detection or **Minus** change detection to work out what rows in the Fact Object are new or which require updating.

SOURCE TAB

This dialog is used to join source tables, add 'Where' clauses and specify group by clauses.



Distinct data select

The **distinct data select** option ensures duplicate rows are not added to the Data Store Object. 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 (i.e. duplicates are not removed).

As source table joins should have been performed in the stage table, see *Generating the Staging Update Procedure* (on page 322) for more details.

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.

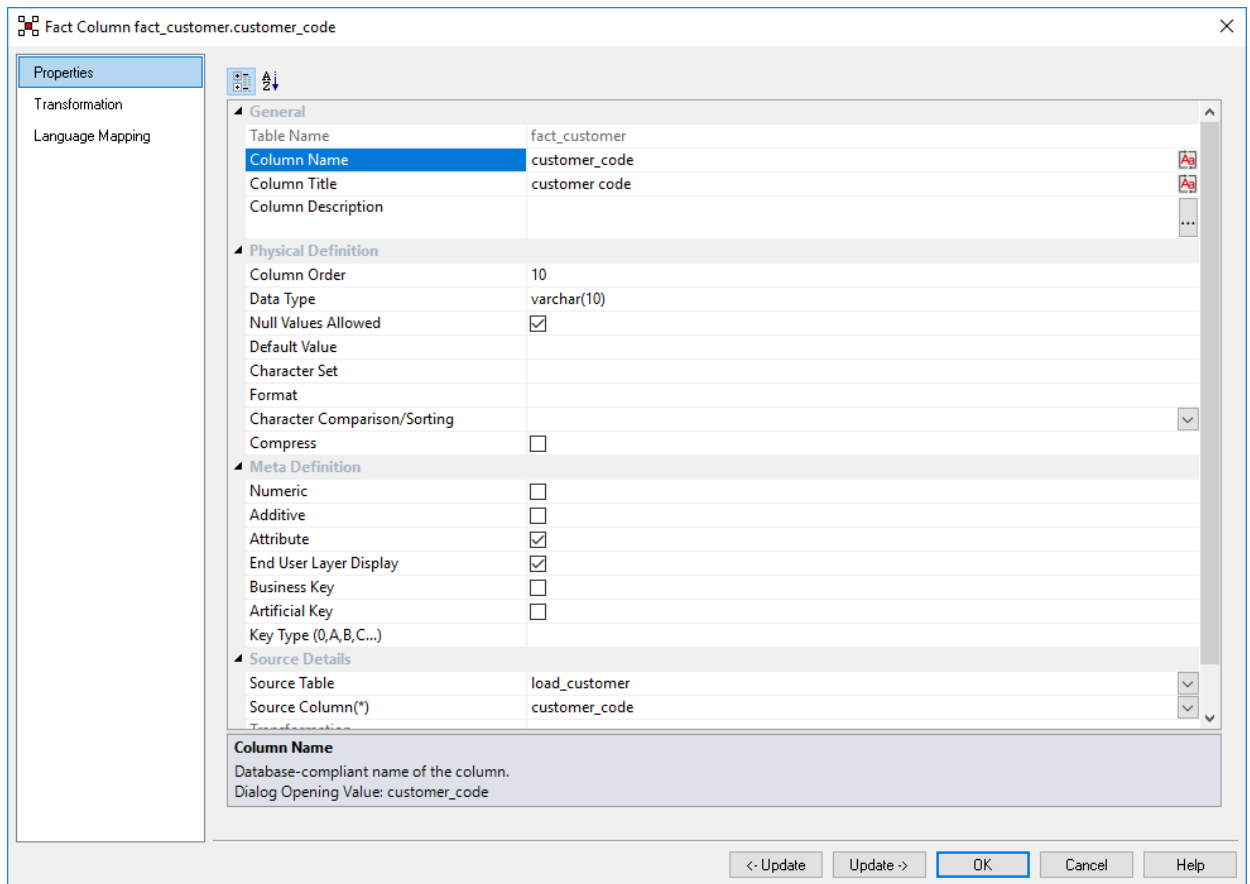
A right-click menu option of **Alter table** is available when positioned on the table name after the validate has completed. This option will alter the database table to match the metadata definition.



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:



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 alphanumeric, 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 a valid for the target database. Typical Teradata databases often have integer, numeric(), varchar(), char(), date and timestamp data types. 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. |

| Key type | Meaning |
|----------|--|
| 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.

The screenshot shows a dialog box titled "Fact Column fact_customer.customer_code". On the left, there is a sidebar with three tabs: "Properties", "Transformation" (which is selected and highlighted in blue), and "Language Mapping". The main area of the dialog contains the following elements:

- Target:** A text box containing "customer_code" with a "Paste" button to its right.
- Source:** A text box containing "load_customer.customer_code" with a "Paste" button to its right.
- Column Transformation Code (must execute within a SQL SELECT statement):** A large empty text area for entering the transformation logic.
- Function Set:** A dropdown menu currently set to "Default Teradata".
- Available Columns:** A list box showing "Functions" and "Available Columns" (with a red icon).
- Word Wrap Displayed Code:** An unchecked checkbox.
- Function Syntax:** An empty text box.
- Function Desc.:** An empty text box with a scroll bar.

At the bottom of the dialog, there are five buttons: "<- Update", "Update ->", "OK" (which is highlighted with a blue border), "Cancel", and "Help".

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 593) 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.

The screenshot shows a dialog box titled "Fact Table Column fact_sales_detail.quantity". On the left, a sidebar contains three tabs: "Properties", "Transformation", and "Language Mapping", with "Language Mapping" selected. The main area of the dialog is divided into several sections:

- Language :** A dropdown menu currently set to "French".
- Language Settings:** A section containing two text input fields, both containing the word "quantity".
- Business Display Name:** A section containing two text input fields, both containing the word "quantity".
- Business Definition:** A section containing two text input fields, both containing the text "Quantity of product sold (i.e. number of product units)". Each field has small up and down arrow icons on its right side.

At the bottom of the dialog, there are five buttons: "<- Update", "Update ->", "OK", "Cancel", and "Help". The "OK" button is highlighted with a blue border.

CHAPTER 18

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.

IN THIS CHAPTER

| | |
|--|-----|
| Creating an Aggregate Table | 495 |
| Creating an Aggregate Summary Table | 496 |
| Aggregate Table Column Properties | 496 |
| Aggregate Table Column Transformations | 501 |

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. 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 Create the aggregate table in the database by right-clicking on the aggregate and selecting **Create(ReCreate)**.
- 4 Create a procedure to update the aggregate by right-clicking on the aggregate, selecting **Properties** and selecting **(Build Procedure...)** in the Update Procedure field. You will be asked for the date in the fact table that is to be used as the basis for rebuilding changes in the fact table. The aggregate update process looks at any records that have been updated in the fact table in the last 7 days (by default). It then rebuilds all the information for the dates that have been altered.
You will also be asked to specify a locking request modifier to be applied to each source table during generated update procedures. By default this is set to 'ACCESS' which locks each row being accessed, a blank entry will result in no locking clause in the generated procedure.

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.
- 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.

AGGREGATE TABLE COLUMN PROPERTIES

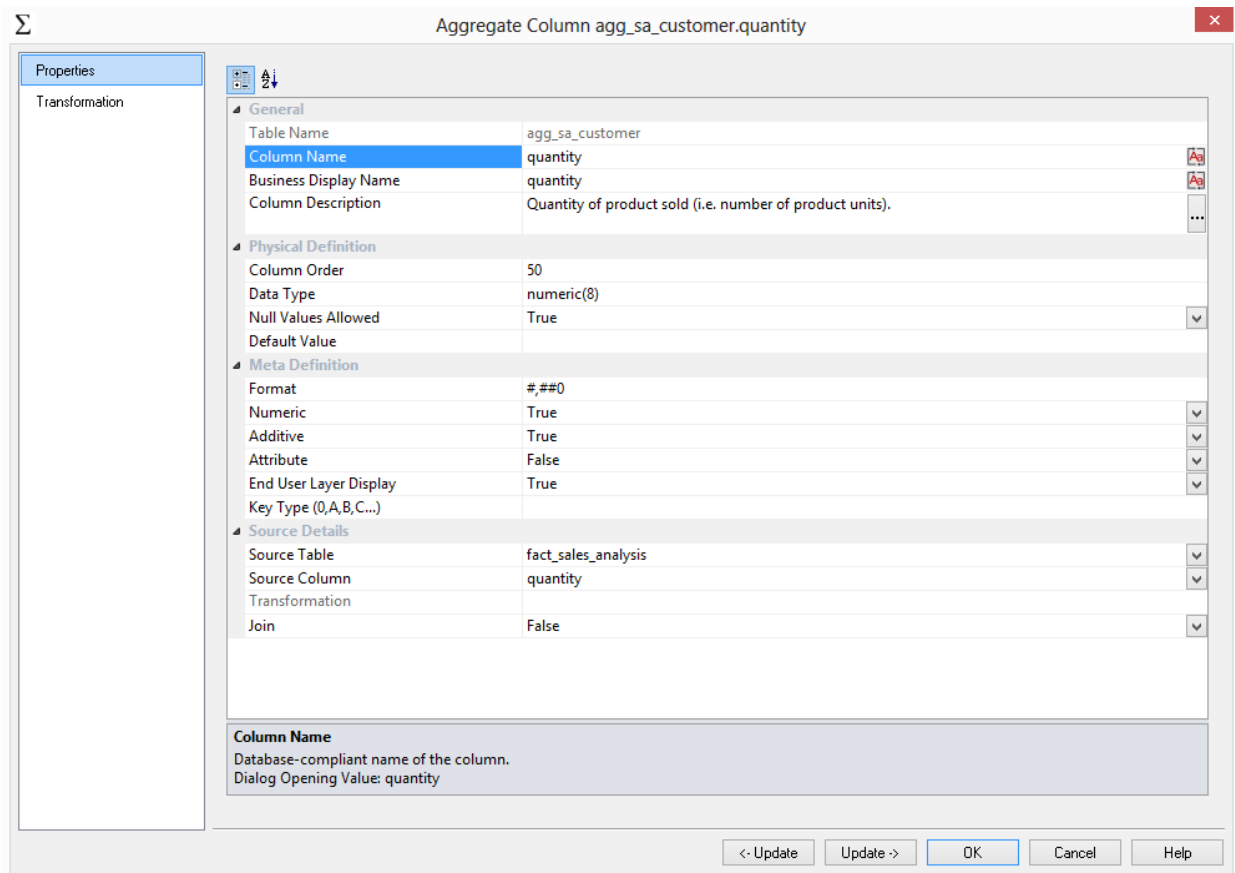
Each aggregate table column has a set of associated properties. The definition of each property is defined below:

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-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:



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 alphanumeric, 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 a valid for the target database. Typical Teradata databases often have integer, numeric(), varchar(), char(), date and timestamp data types. 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. |

| Key type | Meaning |
|----------|---|
| 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.

The screenshot shows a dialog box titled "Aggregate Column agg_product.budget_quantity". On the left, there is a sidebar with "Properties" and "Transformation" tabs, with "Transformation" selected. The main area contains the following fields and controls:

- Target:** budget_quantity (with a Paste button)
- Source:** fact_sales_analysis.budget_quantity (with a Paste button)
- Column Transformation Code (must execute within a SQL SELECT statement):** A large empty text area.
- Function Set:** Default SQL Server (dropdown menu)
- Functions:** A tree view showing "Functions" and "Available Columns".
- Word Wrap Displayed Code
- Function Syntax:** An empty text field.
- Function Desc.:** An empty text field with a scroll bar.

At the bottom right, there are buttons for "<- Update", "Update ->", "OK", "Cancel", and "Help".

Note: Transformations are only put into effect when the procedure is re-generated.

See *Transformations* (on page 593) for more details.

CHAPTER 19

JOIN INDEXES

Join indexes are used to perform one or more of the following tasks in Teradata:

- 1 Replicate all or part of a single table using a new primary index
- 2 Join multiple tables in a pre-join table
- 3 Aggregates one or more columns of one or more tables

WhereScape RED supports all of these uses.

IN THIS CHAPTER

| | |
|----------------------------|-----|
| Creating a Join Index..... | 504 |
|----------------------------|-----|

CREATING A JOIN INDEX

Drag and Drop

- 1 In the left pane double click on the **join index** group to list the join indexes in the middle pane and set join indexes as the drop target.
- 2 From the Data Warehouse browse (right) pane drag a table into the middle pane.
- 3 The new object dialog box will appear and will identify the new object as a Join index and will provide a default name based on the join index name.
- 4 Either accept this name or enter the name of the join index and click **OK** to proceed.

Join Index Properties

At this stage change the storage options if desired and click on **OK**.

If prototyping, and the join index is simple (i.e. one source table) then you can create the join index automatically by answering **Create and Load** to the next question and specifying the primary index. Otherwise proceed to the next section.

Pre-joined Join Indexes

If joining multiple tables in a pre-join table, add the columns from the other tables.

Define the join between the source tables using the 'Where' clause builder (right-click on the join index and select **Build From/Where clause**).

Define source table joins for Join Index

Define the Joins (or edit the where clause).
To define a Join select two tables and select the join type. Then select the join columns from the column lists presented.

Source Tables:

- [load_order_header]
- [load_order_line]

From and Where Clause:

```
FROM [load_order_header] load_order_header
JOIN [load_order_line] load_order_line
ON load_order_header.order_id = load_order_line.order_id
```

Outer Join Simple Join ANSI join code generated

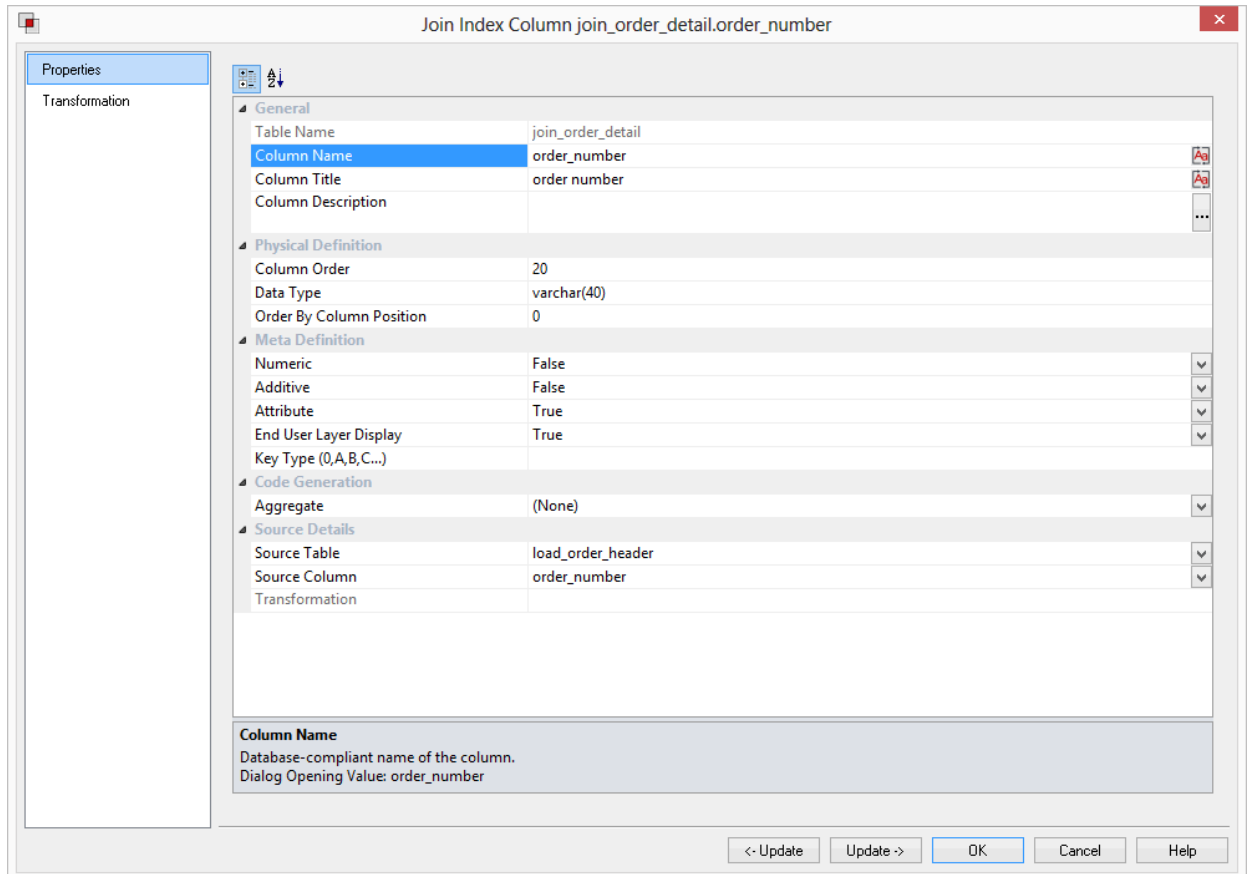
Select the columns that join the two tables.

[load_order_header] [load_order_line]

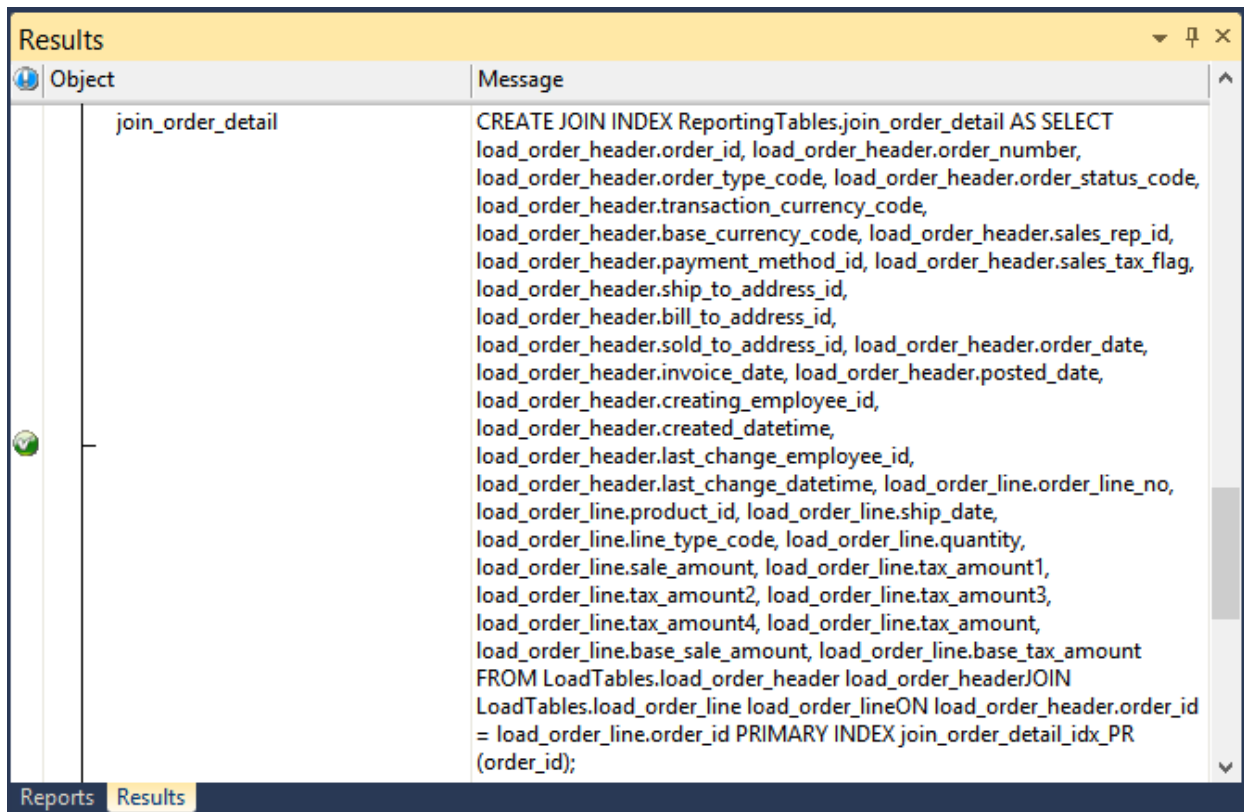
Word Wrap Displayed Code

OK Cancel

If aggregation is required, proceed to the next section, otherwise remove any checks from the **Sum**, **Count** and **Group by** checkboxes on each column of the join index.



Create the join index in the database by right-clicking on the join index and selecting **Create(ReCreate)**.



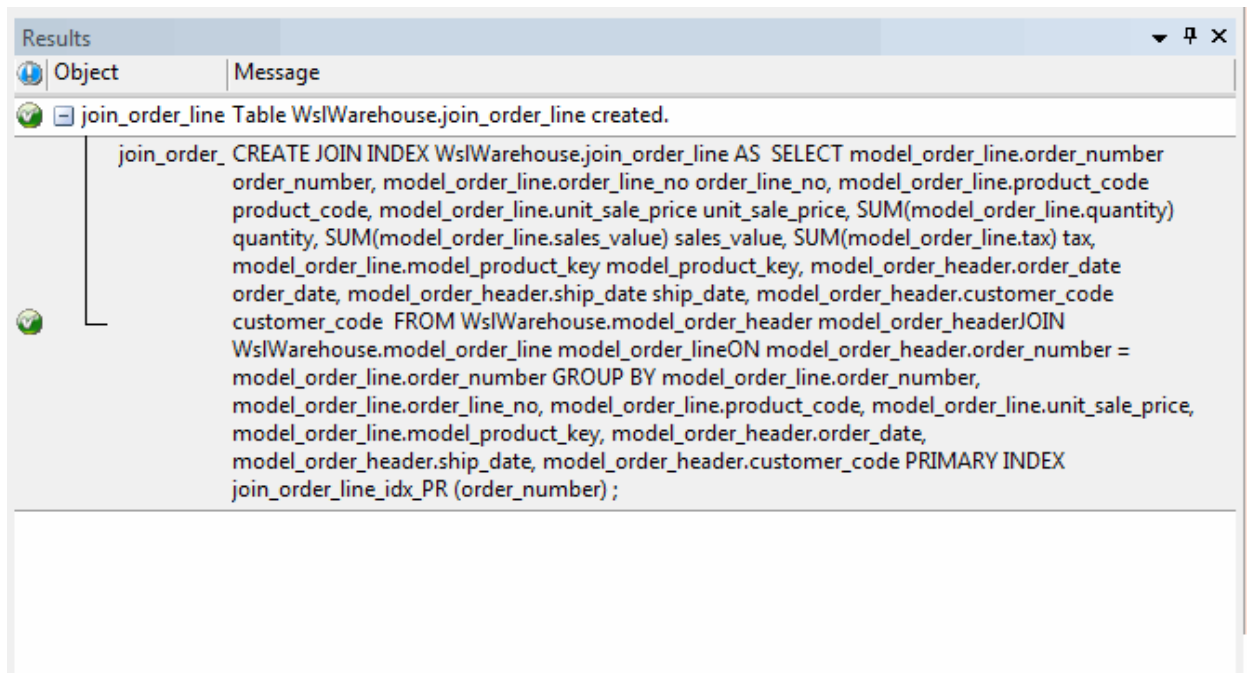
The screenshot shows a 'Results' window with a table containing one row. The 'Object' column contains 'join_order_detail' and the 'Message' column contains a long SQL command. The command is a 'CREATE JOIN INDEX' statement for 'ReportingTables.join_order_detail'. It lists numerous columns from 'LoadTables.load_order_header' and 'LoadTables.load_order_line' tables. The columns include order_id, order_number, order_type_code, order_status_code, transaction_currency_code, base_currency_code, sales_rep_id, payment_method_id, sales_tax_flag, ship_to_address_id, bill_to_address_id, sold_to_address_id, order_date, invoice_date, posted_date, creating_employee_id, last_change_datetime, last_change_employee_id, product_id, ship_date, line_type_code, quantity, and various tax amounts. The command concludes with a 'PRIMARY INDEX' definition for 'join_order_detail_idx_PR' on the 'order_id' column.

| Object | Message |
|-------------------|---|
| join_order_detail | CREATE JOIN INDEX ReportingTables.join_order_detail AS SELECT load_order_header.order_id, load_order_header.order_number, load_order_header.order_type_code, load_order_header.order_status_code, load_order_header.transaction_currency_code, load_order_header.base_currency_code, load_order_header.sales_rep_id, load_order_header.payment_method_id, load_order_header.sales_tax_flag, load_order_header.ship_to_address_id, load_order_header.bill_to_address_id, load_order_header.sold_to_address_id, load_order_header.order_date, load_order_header.invoice_date, load_order_header.posted_date, load_order_header.creating_employee_id, load_order_header.created_datetime, load_order_header.last_change_employee_id, load_order_header.last_change_datetime, load_order_line.order_line_no, load_order_line.product_id, load_order_line.ship_date, load_order_line.line_type_code, load_order_line.quantity, load_order_line.sale_amount, load_order_line.tax_amount1, load_order_line.tax_amount2, load_order_line.tax_amount3, load_order_line.tax_amount4, load_order_line.tax_amount, load_order_line.base_sale_amount, load_order_line.base_tax_amount FROM LoadTables.load_order_header load_order_header JOIN LoadTables.load_order_line load_order_line ON load_order_header.order_id = load_order_line.order_id PRIMARY INDEX join_order_detail_idx_PR (order_id); |

Aggregated Join Indexes

To build an aggregated join index, edit each column and set the Sum, Count and Group by check-boxes as appropriate.

Create the aggregate table in the database by right-clicking on the aggregate and selecting **Create(ReCreate)**.



```
Results
Object | Message
-----|-----
join_order_line Table WslWarehouse.join_order_line created.

join_order_ CREATE JOIN INDEX WslWarehouse.join_order_line AS SELECT model_order_line.order_number
order_number, model_order_line.order_line_no order_line_no, model_order_line.product_code
product_code, model_order_line.unit_sale_price unit_sale_price, SUM(model_order_line.quantity)
quantity, SUM(model_order_line.sales_value) sales_value, SUM(model_order_line.tax) tax,
model_order_line.model_product_key model_product_key, model_order_header.order_date
order_date, model_order_header.ship_date ship_date, model_order_header.customer_code
customer_code FROM WslWarehouse.model_order_header model_order_header JOIN
WslWarehouse.model_order_line model_order_line ON model_order_header.order_number =
model_order_line.order_number GROUP BY model_order_line.order_number,
model_order_line.order_line_no, model_order_line.product_code, model_order_line.unit_sale_price,
model_order_line.model_product_key, model_order_header.order_date,
model_order_header.ship_date, model_order_header.customer_code PRIMARY INDEX
join_order_line_idx_PR (order_number);
```

CHAPTER 20

VIEWS

Views are normally created to manage locking in Teradata, to join tables together for presentation to users or to provide additional methods for updating underlying tables. In our tutorials we create a view on each model table to provide an access path to the model tables with a built in locking clause.

Views can be created from any table type.

IN THIS CHAPTER

| | |
|--|-----|
| One to One Views | 509 |
| Model Views for Aliasing..... | 512 |
| Compound Views, Facts and Dimensions | 515 |
| Creating a Custom View | 521 |
| View Aliases | 522 |

ONE TO ONE VIEWS

A one to one view is a database view of a model table. It may be a full or partial view. It is typically used to provide an access path to the model tables with a built in locking clause.

In many data warehouses 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.

The process for creating a view is as follows:

- 1 Double click on **View** in the left pane.
- 2 Browse to the data warehouse in the right pane.
- 3 Drag a table from the right pane into the center pane.
 - The dialog box that displays defaults the object type to a view.
 - Change the view name as required, and click **ADD**.
- 4 The View properties dialog displays:

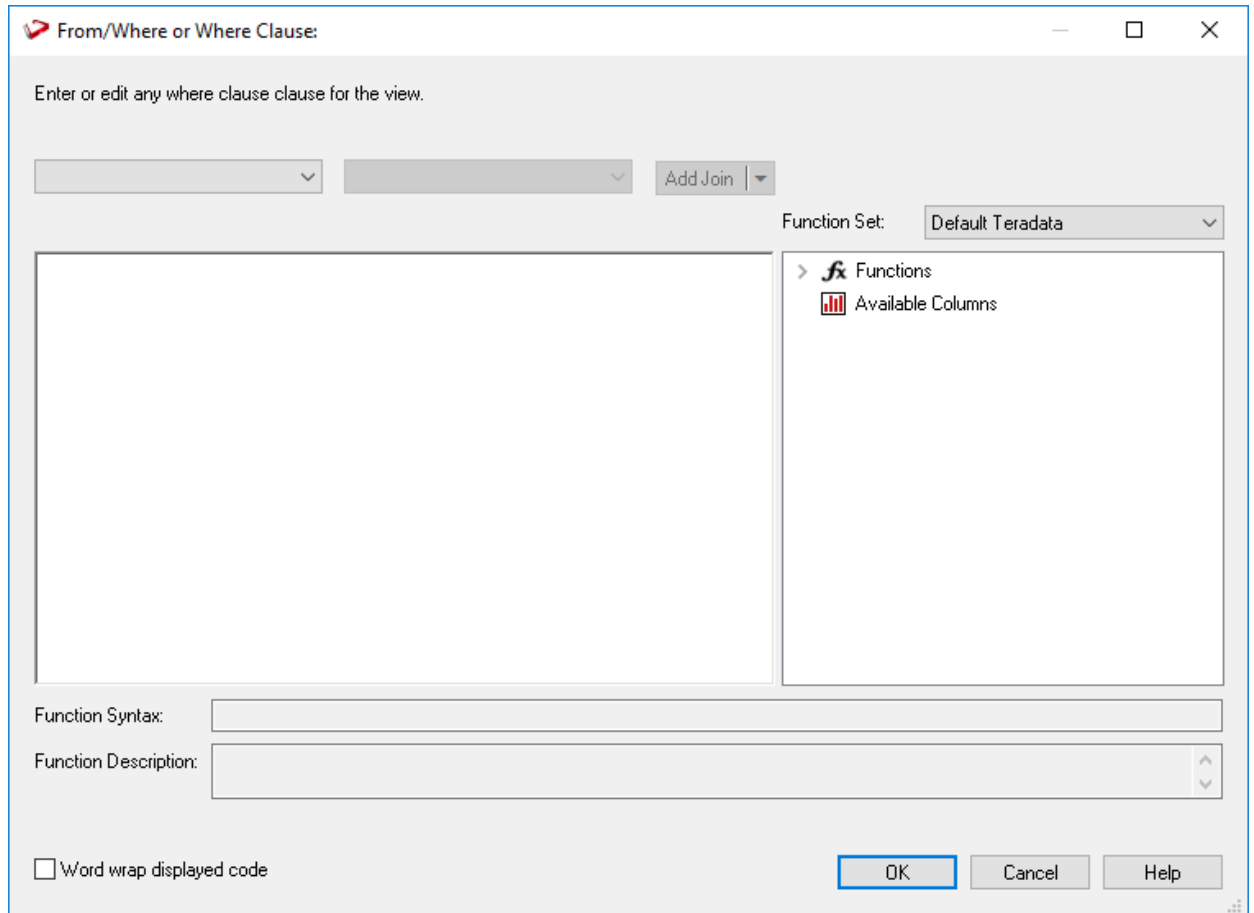
The screenshot shows the 'View view_customer' dialog box. The left sidebar has 'Properties' selected. The main area contains the following fields and controls:

- View Name: view_customer
- View Type: View
- Unique Short Name (maximum 22 characters): view_customer
- Business Display Name (EUL): view_customer
- Description: (empty text area)
- Update Procedure: (None)
- Custom Procedure: (None)
- Distinct Data Select:
- From/Where or Where Clause: (empty text area)
- Table Locking Mode: LOCK ROW FOR ACCESS
- Mnemonic (EUL): (empty text field)
- Timestamps: Metadata Structure Changed: 2016-10-24 22:48:01.580000

Red boxes highlight the Distinct Data Select checkbox, the From/Where or Where Clause field, and the Table Locking Mode dropdown. A red box also highlights a small button in the bottom right corner of the main area.

Change the following properties, if desired:

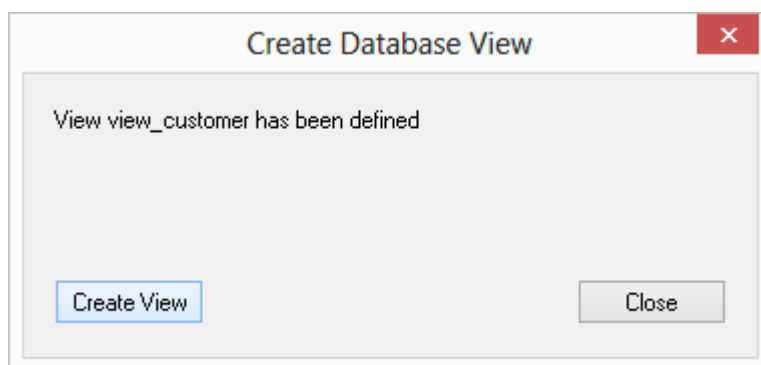
- Tick the **Distinct Data Select** check-box if you want the view to return only distinct values.
- You can enter a **From/Where** clause, but this can be done later. The ellipses button can be used to open the From/Where Clause editor dialog:



- The **Table Locking Mode** can be changed at this point. The default is LOCK ROW FOR ACCESS, all other Teradata locking modes are available for use.

5 Click **OK**.

6 A dialog box displays indicating that the view has been defined. Click **Create view** to create the view in Teradata.



MODEL VIEWS FOR ALIASING

A model view is a database view of a model table used to create an alternative logical view of a model table. It may have some or all columns renamed to match the aliased name of the view. Model Views may be used to look up surrogate keys when building staging tables.

The process for creating a model view is as follows:

- 1 Double click on **View** in the left pane.
- 2 Browse to the data warehouse in the right pane.
- 3 Drag a model table from the right pane into the middle pane.
 - The dialog box that displays defaults the object type to a view.
 - Change the view name as required, and click **Add**.
- 4 The View property defaults dialog displays - change the Table Type to **Model/Dimension View**.
- 5 Tick the **Distinct Data Select** check-box if you want the Model/Dimension View to return only different values.
- 6 Click **OK**.
- 7 The view column definition dialog box is displayed - set the rename values as appropriate.
- 8 A dialog box displays indicating that the view has been defined - click **Create view** to create the view in Teradata.

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:

The column names for the view being created can be modified by filling in the following form. If the Default button is pressed nothing will be changed.

Remove Column Prefix: ---> Add Column Prefix:

Remove Business Display Prefix: ---> Add Business Display Prefix:

Change Column Names for Specific Columns

| Old Column Name: | New Column Name: |
|---|--|
| <input type="text" value="calendar_date"/> | <input type="text" value="ship_date"/> |
| <input type="text" value="model_date_key"/> | <input type="text" value="model_ship_date_key"/> |
| <input type="text"/> | <input type="text"/> |
| <input type="text"/> | <input type="text"/> |
| <input type="text"/> | <input type="text"/> |

OK Default

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 a quire 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 in order 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.

COMPOUND VIEWS, FACTS AND DIMENSIONS

A compound view is used to join tables and views together for presentation to users. They simplify user access, particularly if multiple model history tables need to be accessed in a single query.

The process for creating a compound view is as follows:

- 1 Double click on **View** in the left pane.
- 2 Browse to the data warehouse in the right pane.
- 3 Drag an existing one-to-one view from the right pane into the center pane. Change the view name as required, click **ADD**.
- 4 The view property defaults dialog will appear. A 'Where' clause could be entered, but this can be done later using the 'Where' clause builder. The table locking mode can also be changed at this point. The default is lock for access, all other Teradata locking modes are available for use. Change the table type to be a **dimension view** or **fact view** (as appropriate) if a star schema presentation layer is being built.

Note: You will need to create **fact** and **dimension** views to use **Analysis Services Cubes**.

- 5 Tick the **Distinct Data Select** check-box if you want the view to return only distinct values.
- 6 Click **OK** and then **Finish**, not creating the view for now.
- 7 Add additional columns from other existing one-to-one views from the right pane into the center pane.

Join the source objects

If columns from more than one table or view have been added, then we have to define the joins between the source objects. This is achieved by right-clicking on the view name and selecting **Build From/Where** clause from the pop up menu.

To join tables, select the 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. An additional option is available to allow either an ANSI standard join or a 'Where clause' based join. The ANSI standard join should be chosen in most situations. See the example screen in the following section.

The screenshot shows a dialog box titled "Source Table Mapping" with a close button (X) in the top right corner. Below the title bar, there is a help icon. The main area contains the following text: "Define the joins (or edit the where clause). To define a Join select two tables and select the join type. Then select the join columns from the column lists presented." Below this text, there are two main sections: "Source Tables:" and "From and Where Clause:". The "Source Tables:" section contains a list box with two items: "load_order_header" and "load_order_line". The "From and Where Clause:" section contains a text area with the following SQL code:

```
FROM [TABLEOWNER].[load_order_header] load_order_header
JOIN [TABLEOWNER].[load_order_line] load_order_line
ON load_order_header.order_id = load_order_line.order_id
```

 Below the text area, there are three buttons: "Outer Join", "Simple Join", and a checked checkbox labeled "ANSI join code generated". Below these buttons, there is a section titled "Select the columns that join the two tables." which contains two dropdown menus. The first dropdown menu is set to "load_order_header" and the second is set to "load_order_line". At the bottom left, there is a checkbox labeled "Word Wrap Displayed Code". At the bottom right, there are "OK" and "Cancel" buttons.

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 or not 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. 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 or not 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.

Once satisfied with the 'Where' statement click the **OK** button 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. It can of course be edited in the procedure that is generated if not correct.

You have the choice between 'Where' statement joins and ANSI standard joins.

Note: 'Where' joins are not available if using outer joins in Teradata.

The example below shows the result of an ANSI standard join which takes place in the 'From' statement.

Define source table joins for View

Define the Joins (or edit the where clause).
To define a Join select two tables and select the join type. Then select the join columns from the column lists presented.

Source Tables:

- load_order_header
- load_order_line

From and Where Clause:

```
FROM [TABLEOWNER].[view_order_header] view_order_header  
LEFT OUTER JOIN [TABLEOWNER].[view_order_line] view_order_line  
ON view_order_header.order_number = view_order_line.order_number
```

Outer Join Simple Join ANSI join code generated

Select the columns that join the two tables. Select the column from the Master Table first.

load_order_header load_order_line

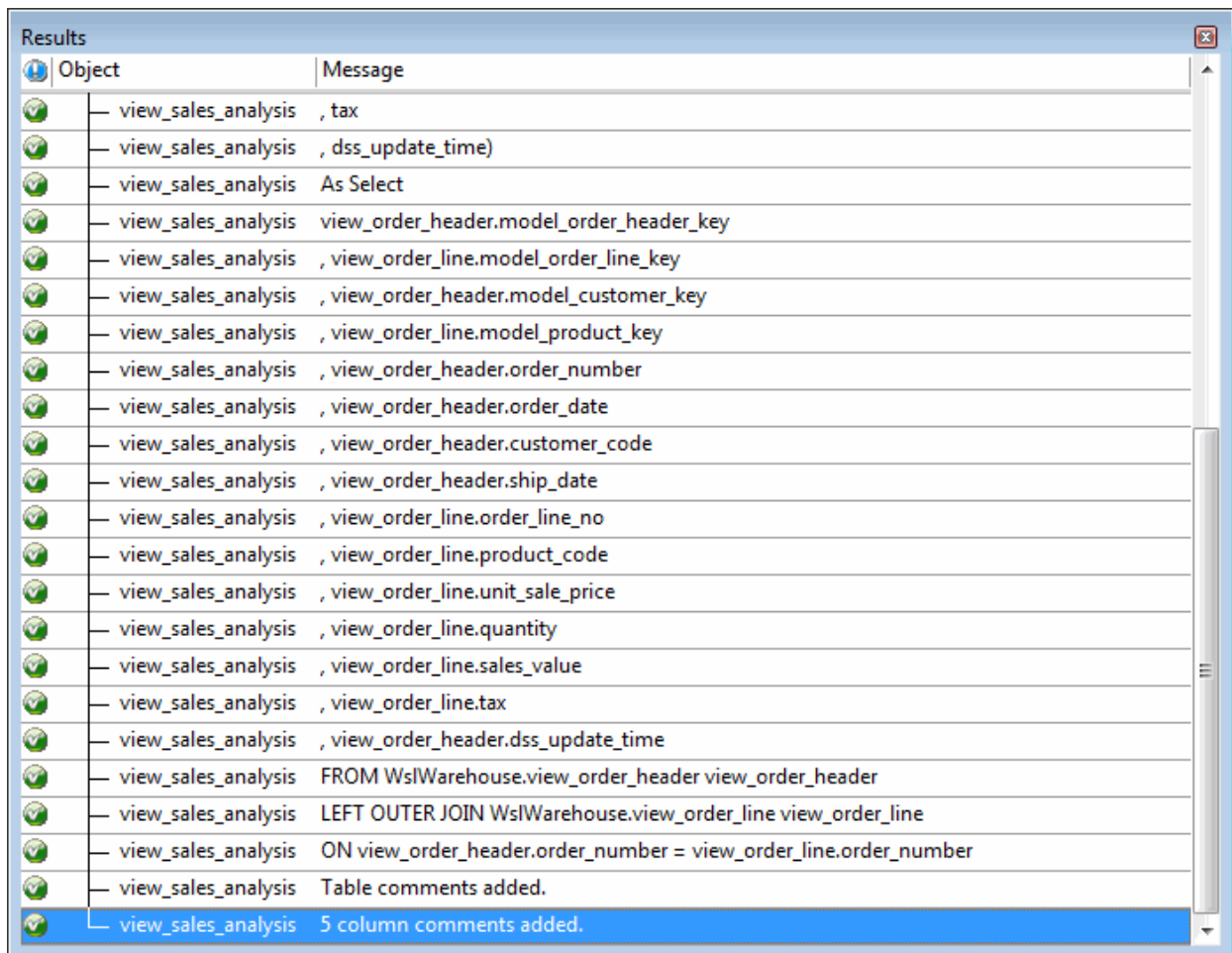
Word Wrap Displayed Code

OK Cancel

Create the view

Once the view has been defined in the metadata we need to physically create the view in the database. This is achieved by using the right-clicking on the view name and selecting **Create (ReCreate)** from the pop up menu.

The output from the creation are visible in the output windows. The following example shows a successful creation.



| Object | Message |
|---------------------|--|
| view_sales_analysis | , tax |
| view_sales_analysis | , dss_update_time) |
| view_sales_analysis | As Select |
| view_sales_analysis | view_order_header.model_order_header_key |
| view_sales_analysis | , view_order_line.model_order_line_key |
| view_sales_analysis | , view_order_header.model_customer_key |
| view_sales_analysis | , view_order_line.model_product_key |
| view_sales_analysis | , view_order_header.order_number |
| view_sales_analysis | , view_order_header.order_date |
| view_sales_analysis | , view_order_header.customer_code |
| view_sales_analysis | , view_order_header.ship_date |
| view_sales_analysis | , view_order_line.order_line_no |
| view_sales_analysis | , view_order_line.product_code |
| view_sales_analysis | , view_order_line.unit_sale_price |
| view_sales_analysis | , view_order_line.quantity |
| view_sales_analysis | , view_order_line.sales_value |
| view_sales_analysis | , view_order_line.tax |
| view_sales_analysis | , view_order_header.dss_update_time |
| view_sales_analysis | FROM WslWarehouse.view_order_header view_order_header |
| view_sales_analysis | LEFT OUTER JOIN WslWarehouse.view_order_line view_order_line |
| view_sales_analysis | ON view_order_header.order_number = view_order_line.order_number |
| view_sales_analysis | Table comments added. |
| view_sales_analysis | 5 column comments added. |

The contents of this window are a message to the effect that the view was created followed by a copy of the actual database create statement.

If the view 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 of finding such a column is to double click on the list heading **Col name**. This will sort the column names into alphabetic 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.

DIMENSION VIEW HIERARCHIES

The various hierarchies associated with a dimension view can be recorded in the WhereScape RED metadata. These hierarchies are often not used in any form, except to provide documentary completeness and for creating **Analysis Services OLAP Cubes**.

ADDING A DIMENSION VIEW HIERARCHY

Any number of hierarchies can be created against a dimension view. There is no restriction on the form of the hierarchy. To add a new hierarchy, position on the dimension view in the left pane and using the right-click menu, select **Hierarchies/Add Hierarchy**. The following dialog will appear.

Add Hierarchy [X]

Hierarchy Name:

Description:

Move columns from the column list into the hierarchy. The hierarchy is a top down list. For example a date hierarchy may be year, month, day. Year will be the first column shown.

Available Columns:

| Column Name |
|---|
| <input type="checkbox"/> dim_customer_key |
| <input type="checkbox"/> code |
| <input type="checkbox"/> name |
| <input type="checkbox"/> address |
| <input type="checkbox"/> city |
| <input type="checkbox"/> state |
| <input type="checkbox"/> dss_update_time |

Hierarchy: Levels

[OK] [Cancel]

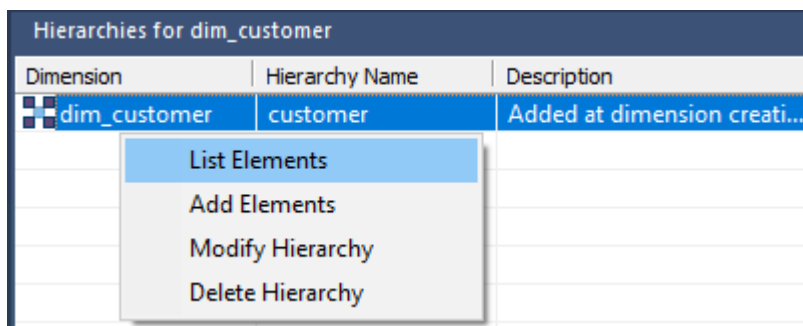
- 1 Enter a meaningful name for the hierarchy.
- 2 Enter a meaningful description for the hierarchy. This description is carried through into the Hierarchy Description field of any OLAP Dimensions that are built from the original Dimension object.

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 view may have **state** at the highest level, then **city**, then **address** and finally **code** 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 column. 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.



| Dimension | Hierarchy Name | Description |
|--------------|----------------|------------------------------|
| dim_customer | customer | Added at dimension creati... |

- List Elements
- Add Elements
- Modify Hierarchy
- Delete Hierarchy

CREATING A CUSTOM VIEW

A custom view can be created within 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 RED and the 'Select' component of the view is customized. The second option is where the view is totally custom and no columns need to be defined in RED, although it is good practice to still define the columns for documentation purposes.

To create a Custom or 'User Defined' view proceed as follows:

- 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 properties of the view.

View view_customer

Properties

View Create Statement

Storage

View Aliases

Purpose

Concept

Grain

Examples

Usage

Notes

View Name: view_customer

View Type: User Defined View

Unique Short Name: (maximum 22 characters) view_customer

Business Display Name (EUL): view_customer

Description:

Update Procedure: (None) Rebuild Regenerate

Custom Procedure: (None)

Distinct Data Select:

From/Where or Where Clause:

Table Locking Mode: LOCK ROW FOR ACCESS Mnemonic (EUL):

Timestamps

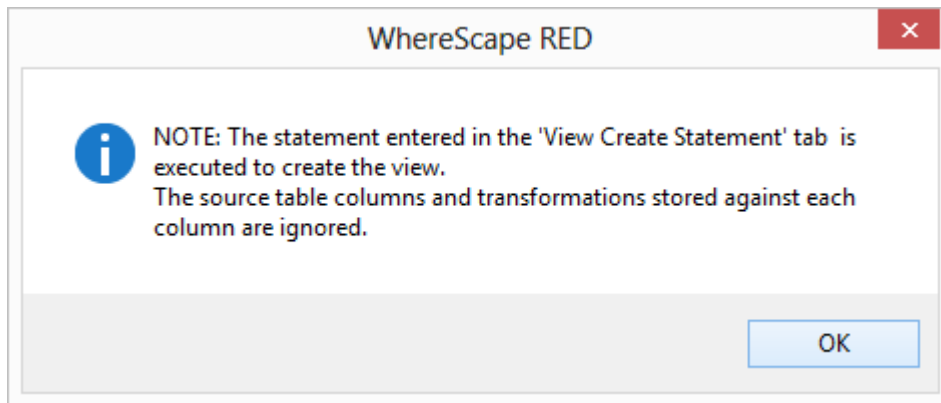
Metadata Structure Changed: 2016-11-09 01:39:56.280000

Database Created: 2016-11-09 01:40:17.840000

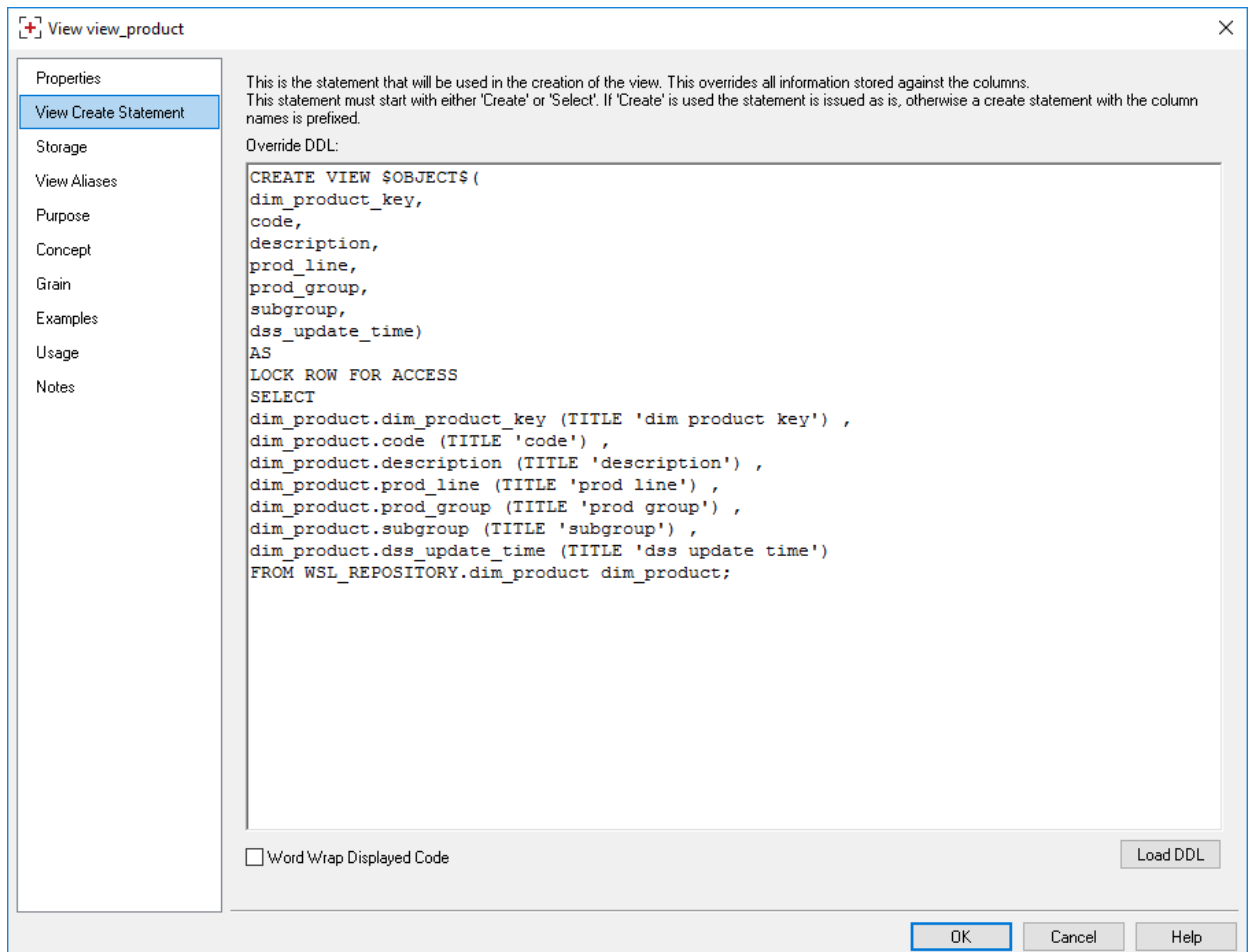
Database Altered: 2016-11-09 01:40:17.840000

OK Cancel Help

- The following message is displayed. Click **OK**.

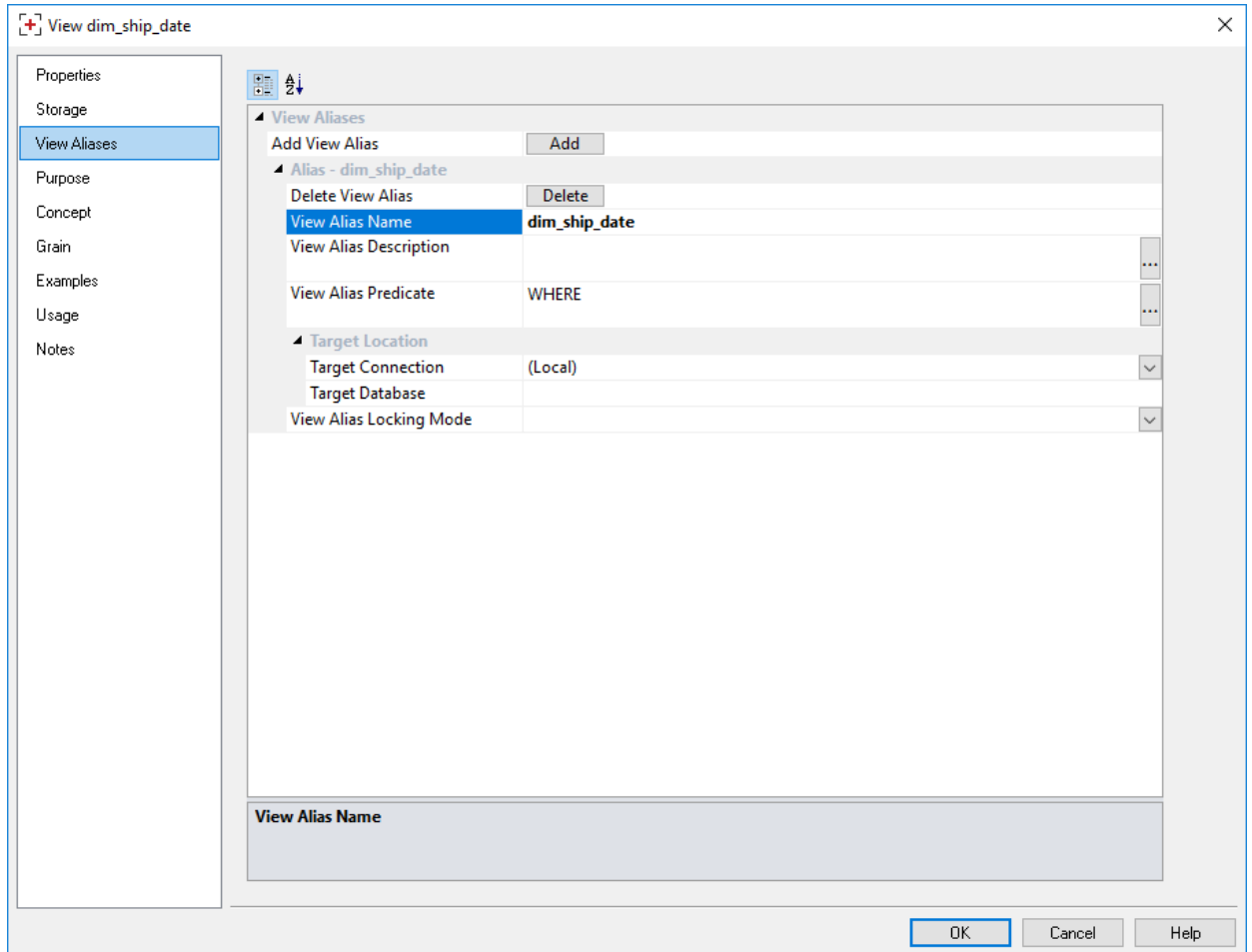


- Edit the new tab **View Create Statement** and insert the SQL Statement that will be used to create the view. This SQL Statement must start with either 'Create' or 'Select'. If 'Create' is used then the columns in the view are ignored and the statement will be issued to create the view. If the statement starts with 'Select' then RED will build up a view create statement form the column names and the supplied **Select** clause. A button 'Load from Columns' is available to get a sample **Select** statement based on the columns in the view and any transformations.



VIEW ALIASES

View Aliases provide multiple deployments of the same view into different Teradata presentation layers. The **View Aliases** tab enables you to define additional/replica views.



Add button

Enables you to add a View Alias.

Delete button

Enables 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.

CHAPTER 21

ANALYSIS SERVICES OLAP CUBES

IN THIS CHAPTER

| | |
|---|-----|
| OLAP Overview | 526 |
| OLAP Defining the Data Source for the OLAP Cube | 526 |
| OLAP Defining an OLAP Cube | 529 |
| OLAP Inspecting and Modifying Advanced Cube Properties | 534 |
| OLAP Creating an OLAP Cube on the Analysis Services Server..... | 535 |
| OLAP Cube Objects | 536 |
| OLAP Dimension Objects | 571 |
| OLAP Changing OLAP Cubes | 586 |
| OLAP Retrofitting an OLAP Object | 588 |

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.

WhereScape RED supports cubes in Microsoft Sql Server Analysis Services versions 2005 and 2008.

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.

The screenshot shows the 'Connection DataWarehouse' dialog box with the 'When Connection is an OLAP Data Source' section expanded. The fields in this section are:

| Property | Value |
|----------------------------|--------------|
| MSAS Connection String | |
| Connection Provider/Driver | TDOLEDB |
| Data Warehouse Server | TD15_doc6871 |
| Data Warehouse Database ID | doc6871 |

Other visible fields in the dialog include:

- Source System:** Database ID (TD15_doc6871), Database Link Name
- Database Credentials:** Extract User ID (doc6871), Extract User Password (*****), Administrator User ID, Administrator User Password, Teradata Wallet User ID, Teradata Wallet String, ODBC User Default (Extract User)
- Other:** Default Schema for Browsing (doc6871), New Table Default Load Type (Database link load), New Table Default Load Script Template ((None)), SSIS Connection String (OLEDB), SSIS Use Column Names (checkbox), Data Type Mapping Set ((Default)), Default Transform Function Set ((Default))

At the bottom of the dialog, there are 'OK', 'Cancel', and 'Help' buttons. A note at the bottom states: '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.'

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.

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 **TDOLEDB**.

Data Warehouse Server

Data Warehouse Server Name, which is used when the data warehouse is used as the data source for OLAP cubes. Set this to the Teradata TDPID.

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.

OLAP DEFINING AN OLAP CUBE

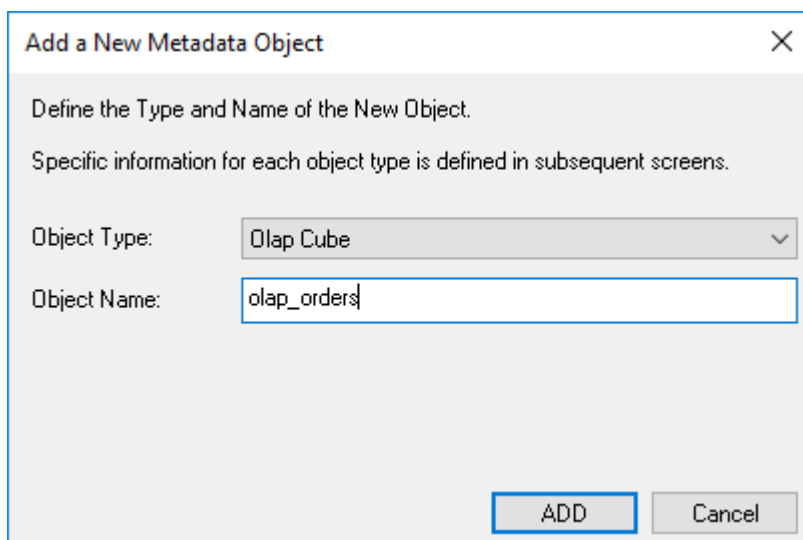
OLAP Cubes can be created from fact views. A single cube can contain data from multiple source star schemas, each defined with a measure group. An OLAP Cube consists of many parts namely, measure groups, measures, calculations, actions, dimensions, dimension hierarchies, dimension attributes and dimension 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.

Note: 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** for example.

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 view from the source pane into the target pane.
- 4 Set the cube name in the **Create new metadata Object** dialog box and click **ADD**.



Add a New Metadata Object

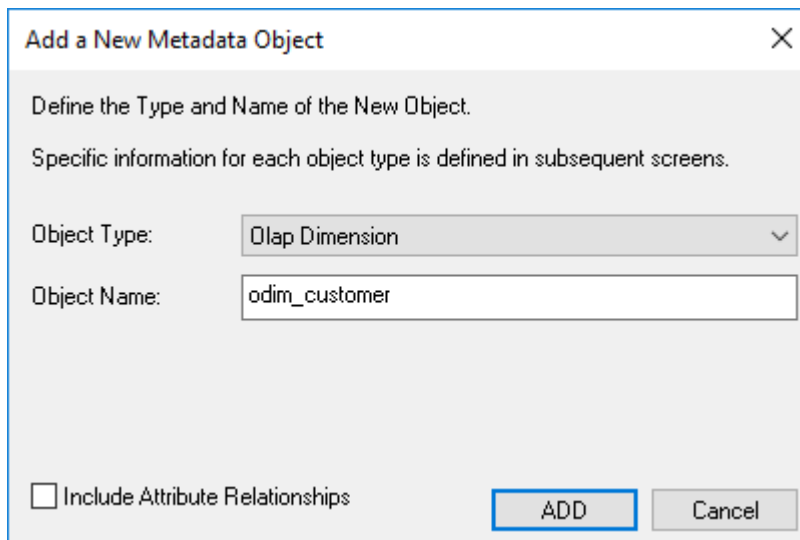
Define the Type and Name of the New Object.
Specific information for each object type is defined in subsequent screens.

Object Type: Olap Cube

Object Name: olap_orders

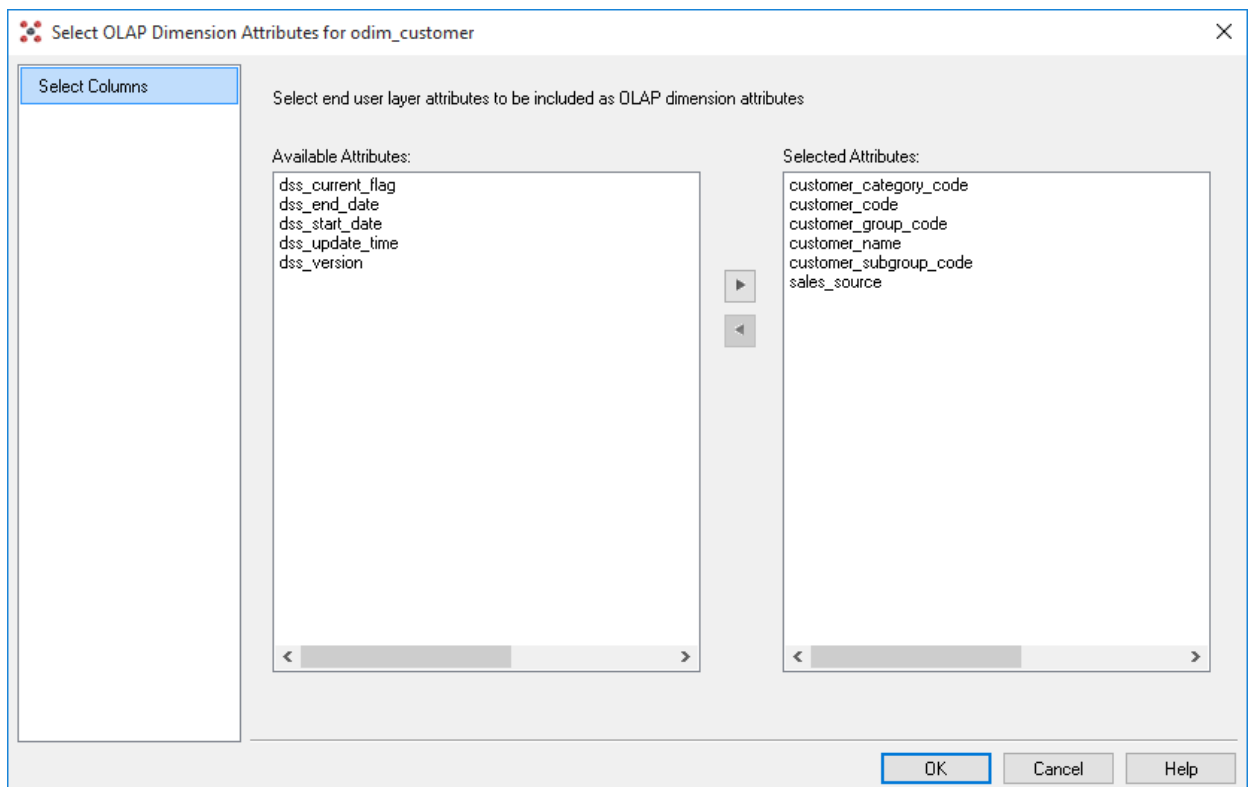
ADD Cancel

- 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 view metadata). Set the dimension name in the **Add a new metadata Object** dialog box and click **ADD**. Repeat this for each dimension as required.



Note: If you wish to include Attribute Relationships in Analysis Services for this dimension, click on the **Include Attribute Relationships** checkbox.

- 6 A 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.

Manage Cube Measures

The following measures have been added to the cube. Select any measures that are not appropriate and click the Remove button if required.

Pre-built calculated members will be generated for each measure if the check boxes to the right are set. These calculated members can be deleted later if not required.

| Measure | Calculated Members: |
|---|--|
| <input checked="" type="checkbox"/> quantity | <input checked="" type="checkbox"/> Month To Date |
| <input checked="" type="checkbox"/> gross sale amount | <input checked="" type="checkbox"/> Year To Date |
| <input checked="" type="checkbox"/> net sale amount | <input type="checkbox"/> Moving Quarter |
| <input checked="" type="checkbox"/> tax amount | <input checked="" type="checkbox"/> Moving Year |
| | <input checked="" type="checkbox"/> Same Month Previous Year |
| | <input checked="" type="checkbox"/> Previous Year To date |

Use OLAP Functions

Dimension and level information for calculated members:

Date Dimension:

Date Hierarchy:

Month Level:

Year Level:

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 view 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 measures 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.

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 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 and select **Properties**.

Olap Cube olap_orders

Properties

Language Mapping

Purpose

Concept

Grain

Internal Cube Name: olap_orders

Cube Publish Name: olap_orders

Cube Description:

Cube Database Connection (Analysis Services): SSAS Cubes

Cube Database Name: OrderAnalysis

Data Source Connection (Data Warehouse): Data Warehouse

Data Source Provider Type: TDOLLEDB

Data Source Server: TD_14_00

Post Create XML/A Script: (None) Edit

Post Update XML/A Script: (None) Edit

Processing Mode: Regular

Processing Priority: 0

Partition Processing Mode: All Partitions - Sequential

Process Cube Dimensions: Enabled Dimensions (Use setting on Cube Dimension) Process Selected Cube Dimensions in Parallel

Storage Mode: MOLAP

Default Measure:

Estimated Rows: 0

Visible: True

OK Cancel Help

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 by the limitations imposed 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. 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.

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.

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.

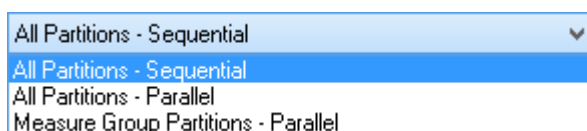
Partition Processing Mode

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.

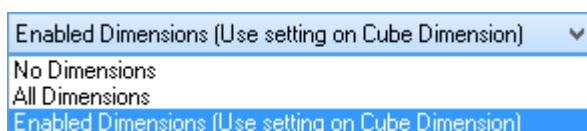
Measure Group Partitions - Parallel will process each measure group sequentially with all partitions of that same measure group being updated in parallel.



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** checkbox of each Dimension's Properties screen. See *OLAP Cube Dimensions*.



Process Selected Cube Dimensions in Parallel

Selecting 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 views. Enter the size of the fact view if known, otherwise leave as zero.

Visible

Indicates whether the cube is visible to client applications.

ROLAP stands for Relational Online Analytical Processing.

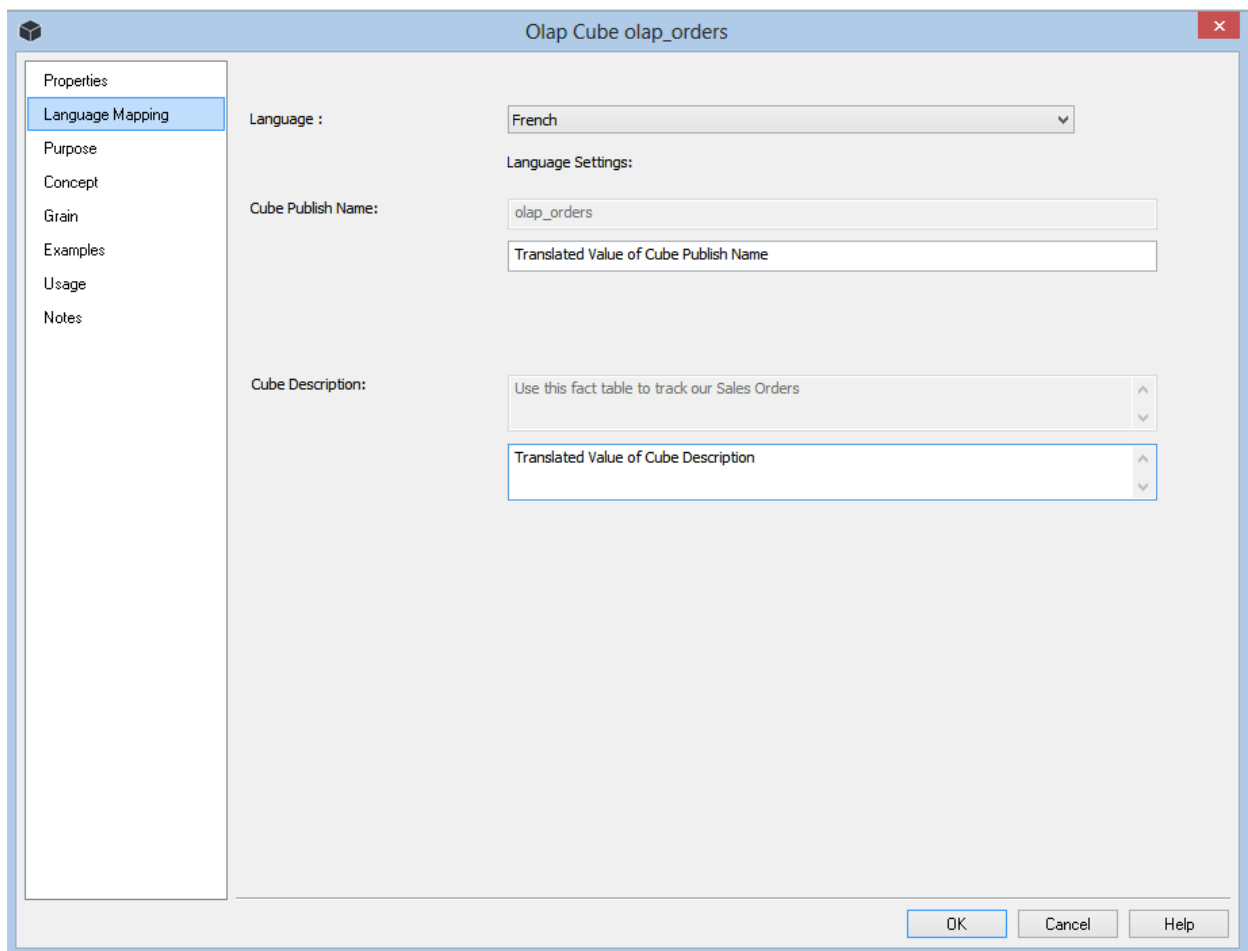
ROLAP is an alternative to the MOLAP (Multidimensional OLAP) technology. While both ROLAP and MOLAP analytic tools are designed to allow analysis of data through the use of a multidimensional data model, ROLAP differs significantly in that it does not require the pre-computation and storage of information. 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 "**Settings - Language Options**" on page 130) 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 Processing a tab of the **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 order_detail Definition

Measure Group Name:

Measure Group Description:

Source Table Type:

Source Table:

Estimated Rows:

Storage Mode:

Ignore Unrelated Dimension:

Measure Group Type:

Processing Mode:

Processing Priority:

<- Update

Update ->

OK Cancel Help

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.

The screenshot shows the 'Olap Cube Measure Group olap_orders.sales_detail' dialog box. The 'Processing / Partitions' tab is active. The 'Process Method' is set to 'Full process'. The 'Increment Filter' is empty. The 'Partitioning Method' is 'Auto create partitions as required (one numeric level)'. The 'Partition By Dimension' is 'dim_order_date'. The 'Partition By Attribute' is 'order cal month'. The 'Partition Value Type' is 'YYYYMM'. The 'Fact Partition Lookup Clause' is 'dss_update_date < GETDATED-14'. The 'Max # Auto Create Partitions' is 12. The '# Historic Partitions Updated' is 3. The dialog has 'OK', 'Cancel', and 'Help' buttons at the bottom right.

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. In order 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 transactional data undergoes changes 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 transactional 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. In order to know when to create a new partition WhereScape RED executes a query against the fact view and the date dimension to acquire each unique period. When dealing with a large fact view, 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 view 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 view 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 view 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.

The screenshot shows the 'Measure Group Partition Definition' dialog box. The 'Partition Properties' tab is active on the left. The main area contains the following fields and values:

- Cube Name: olap_orders
- Data Source: order_detail
- Fact Table: fact_order_detail
- Partition Name: fact_order_detail_200701
- Partition Description: Auto added partition filtering on dimension dim_order_date where column order_cal_month equals 200701
- Data Slice Formulas: [dim_order_date].[order cal month].&[200701]
- Aggregation Prefix: (empty)
- Filter Statement: (empty)
- Storage Mode: MOLAP
- Partition Type: Local
- Remote Server: (empty)
- Processing Method: Default processing

Buttons for 'Update <', 'Update >', 'OK', 'Cancel', and 'Help' are visible.

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, then a server must be specified where the partition will be located.

Remote Server

If a Remote partition is chosen, 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:

The screenshot shows the 'Cube Measure Definition' dialog box. The 'Measure Properties' tab is active. The fields are as follows:

| Property | Value |
|--------------------|---|
| Cube Name | olap_orders |
| Measure Name | quantity |
| Measure Group | order_detail |
| Source Table | fact_order_detail |
| Source Column | quantity |
| Data Type | int |
| Aggregation Method | Sum |
| Display Format | ## |
| Null Processing | Automatic |
| Order Number | 160 |
| Visible | True |
| Display Folder | |
| Description | Quantity of product sold (i.e. number of product units) |

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 view 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.

Fact Table

A read only field that indicates the fact view that is related to the Measure Group.

Source Column

Specifies from which numeric attribute of the underlying fact view the Measure is built. This is a drop-down list populated from REDs metadata definition of the fact view 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 view 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:

| | |
|-----------------|------------------|
| Display Folder: | financials:sales |
|-----------------|------------------|

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:

The screenshot shows a dialog box titled "Olap Cube Calculated Member olap_orders.mtd quantity". On the left, there is a sidebar with "Properties" selected and "Language Mapping" below it. The main area contains the following fields and controls:

- Cube Name: olap_orders
- Calculated Member Name: mtd quantity
- Description: Calculated member
- Expression: AGGREGATE(MTD([dim_order_date].[calendar]),[Measures].[quantity])
- Parent Hierarchy: Measures
- Parent Member: (empty)
- Associated Measure Group: sales_detail
- Display Folder: (empty)
- Display Format: (empty)
- Font: Microsoft Sans Serif, 8, Bold
- Fore Color: (dropdown)
- Back Color: (dropdown)
- Visible: True
- Non Empty Behavior: (empty)
- Order Number: 130

Buttons: "<- Update", "Update ->", "OK", "Cancel", "Help".

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

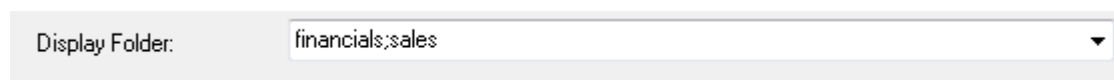
Specifies against which Measure Group the Measure is defined. This is related to the fact view of which the Measure is an attribute.

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:



The image shows a user interface element for setting a display folder. It consists of a label 'Display Folder:' followed by a text input field containing the text 'financials;sales'. To the right of the input field is a small downward-pointing arrow, indicating a dropdown menu.

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

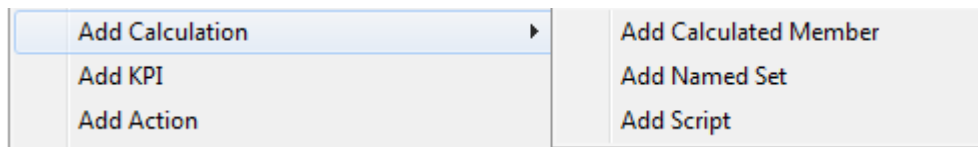
The create order of the member in the dimension hierarchy.

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:

The screenshot shows the 'Cube KPI Definition' dialog box. The 'KPI Definition' tab is selected in the left-hand pane. The main area contains the following fields and values:

- Cube Name: olap_orders
- KPI Name: sales_growth
- Description: The KPI shows growth against the same period last year. The goal is to attain 10% growth year on year.
- Display Folder: (empty)
- Associated Measure Group: order_detail
- Value Expression: [Measures].[order_count]
- Goal Expression: [Measures].[prev year month order count]*1.1
- Status Indicator: Smiley Face
- Status Expression: KpiValue("Sales Growth")/KpiGoal("Sales Growth") >= 0.85
- Trend Indicator: Standard Arrow
- Trend Expression: KpiValue("Sales Growth")/KpiGoal("Sales Growth")*1.1
- Parent KPI: (empty)
- Weight: (empty)

Buttons at the bottom: OK, Cancel, Help.

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:

| | |
|-----------------|---|
| Display Folder: | <input type="text" value="financials;sales"/> |
|-----------------|---|

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 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.

Status Indicator

A visual element that provides a quick indication of the status for a KPI. The display of the element is determined by the value of the MDX expression that evaluates status.

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.

Display Folder

The folder in which the KPI will appear when a user is browsing the cube.

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.

Current Time Member

An MDX expression that returns the member that identifies the temporal context of the KPI.

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 will be shown:

The screenshot shows the 'Cube Action Definition' dialog box with the following configuration:

- Cube Name: olap_orders
- Action Type: Drillthrough Action
- Action Name: order_detail Drillthrough Action
- Target Type: Cells
- Measure Group Members: order_detail
- Condition: (Empty text area)
- Server: (Empty text area)
- Report Path: (Empty text area)
- Format / Content Type: (Empty dropdown)
- Expression: (Empty text area)
- Default: True
- Invocation: Interactive
- Application: (Empty text area)
- Description: Drill through action
- Caption: (Empty text area)
- Caption is MDX: False

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 list is enabled for Drill through actions. This gets or sets the current DrillThroughAction as the default action when multiple drill through actions are defined. This is important for users of Excell 2007 to browse the Olap Cube because Excell 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** from the right-click menu. The Properties are shown below:

The screenshot shows the 'Cube Dimension dim_customer Definition' dialog box. It has a left sidebar with 'Cube Dimension Propertie' and 'Language Mapping'. The main area contains the following fields:

- Internal Dimension Name: odim_customer (dropdown)
- OLAP Dimension Name: dim_customer (text)
- Cube Dimension Name: dim_customer (text)
- Dimension Description: This is the master Customer List for the Company (text area)
- Order Number: 10 (text)
- Process Cube Dimension With Cube: (checkbox)
- Visible: True (dropdown)
- All Member Aggregation Usage: Default (dropdown)
- Hierarchy Unique Name Style: IncludeDimensionName (dropdown)
- Member Unique Name Style: Native (dropdown)
- Source Table Type: Dimension (dropdown)
- Source Table: dim_customer (dropdown)
- Source Table Key: dim_customer_key (dropdown)
- Processing Mode: Regular (dropdown)
- All Caption: All customer (text)
- OLAP Dimension Type: Regular (dropdown)
- Unknown Member Action: None (dropdown)
- Unknown Member Name: Unknown (text)

Buttons: <- Update, Update ->, OK, Cancel, Help.

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 checkbox you can enable the dimension to be processed with the OLAP Cube.

Visible

Determines whether or not 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 views and dimensions in the underlying star schema.

The **Properties** are shown below:

Measure Group Dimension dim_customer Definition

Cube Dimension Properties

Internal Dimension Name: odim_customer

OLAP Dimension Name: dim_customer

Cube Dimension Name: dim_customer

Dimension Description: This is the master Customer List for the Company

Order Number: 10 Process Cube Dimension With Cube

Visible: True

All Member Aggregation Usage: Default

Hierarchy Unique Name Style: IncludeDimensionName

Member Unique Name Style: Native

Measure Group: order_detail

Measure Group Column: dim_customer_key

Relationship Type: Regular

Cardinality: Many

Source Table Type: Dimension

Source Table: dim_customer

Source Table Key: dim_customer_key

Processing Mode: Regular

All Caption: All customer

OLAP Dimension Type: Regular

Unknown Member Action: None

Unknown Member Name: Unknown

<- Update Update ->

OK Cancel Help

Internal Dimension Name

A read only field displaying the name of the OLAP Dimension in WhereScape RED.

OLAP Dimension Name

A ready 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 view 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 view 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:

Olap Dimension odim_customer

Internal Dimension Name: odim_customer

Dimension Publish Name: dim_customer

Dimension Description:

Default Database Connection (Analysis Services): SSAS Cubes

OLAP Database Name: OrderAnalysis

Data Source Connection (Data Warehouse): DataWarehouse

Data Source Provider Type: TDOLEDB

Data Source Server: TD_14_00

Data Source Database: dssdemo

Post Create XML/A Script: (None) Edit

Source Table Type: Dimension

Source Table: dim_customer

Source Table Key: dim_customer_key

Processing Group: ByAttribute

Processing Mode: Regular

Processing Method: Default process

Storage Mode: MDLAP

All Caption: All customer

OLAP Dimension Type: Regular

Unknown Member Action: None

Unknown Member Name: Unknown

OK Cancel Help

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

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 dimension 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**.

The screenshot shows a dialog box titled "Dimension odim_customer Definition" with a close button in the top right corner. On the left is a vertical navigation pane with the following items: "Dimension Properties", "Language Mapping" (highlighted), "Purpose", "Concept", "Grain", "Examples", "Usage", and "Notes". The main area is divided into several sections:

- Language :** A dropdown menu set to "French".
- Language Settings:** A section header.
- Dimension Publish Name:** A text box containing "dim_customer", followed by a text box for "Translated Value of Dimension Publish Name".
- All Caption:** A text box containing "All customer", followed by a text box for "Translated Value of All Caption".
- Dimension Description:** A text box containing "This is the master Customer List for the Company", followed by a text box for "Translated Value of Customer Dimension".
- Unknown Member Name:** A text box containing "Unknown", followed by a text box for "Translated Value of Unknown Member Name".

At the bottom right, there are three buttons: "OK", "Cancel", and "Help".

See **Language Settings** (see "**Settings - Language Options**" on page 130) 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:

Olap Dimension Attribute odim_order_date.dim_order_date_key

Properties

Language Mapping

Dimension Name: odim_order_date

Internal Attribute Name: dim_order_date_key

Published Name: order_date

Description: Key for dim_order_date

Estimated Count: 1

Member Names Unique: False

Hierarchy Visible: True

Hierarchy Enabled: True

Hierarchy Optimized State: FullyOptimized

Hierarchy Display Folder:

Order By: Key

Order By Attribute:

Type: Regular

Usage: Key

Key Column: dim_order_date

Name Column: dim_order_date

Value Column: dim_order_date

<- Update

Update ->

OK

Cancel

Help

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.

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.

| | |
|---------------------|--|
| Order By: | Key |
| Order By Attribute: | |
| Type : | order_current_cal_year order_current_cal_ytd order_moving_cal_year |
| Usage: | Regular |
| Key Column: | dim_order_date |
| Name Column: | dim_order_date |
| Value Column: | dim_order_date |

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

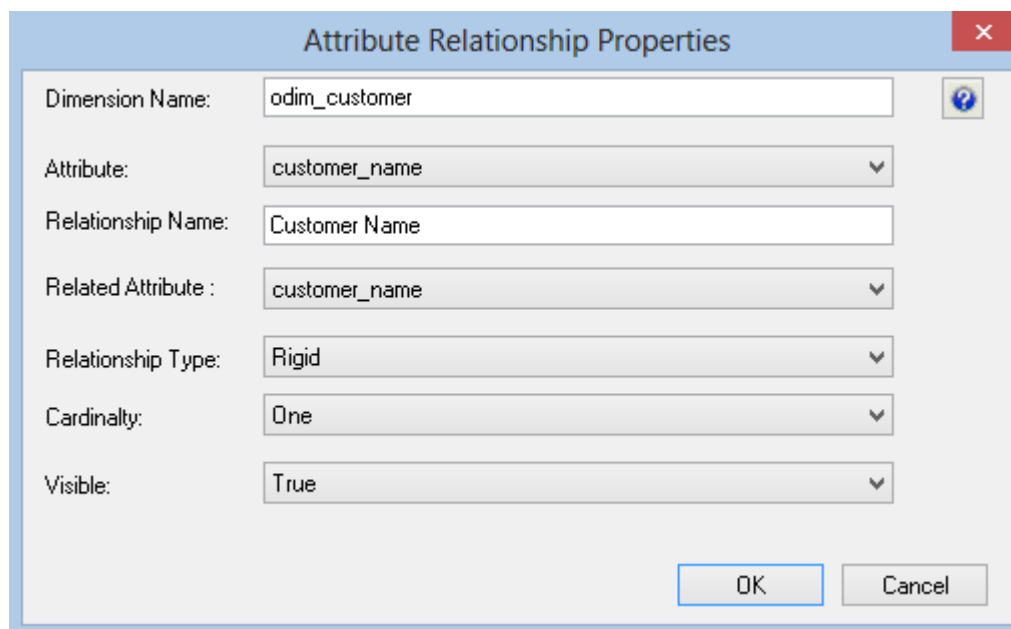
Display the Attributes in the middle pane, select the Attribute to delete, right-click and select **Delete**.

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 in order 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:



The screenshot shows a dialog box titled "Attribute Relationship Properties". It contains the following fields and values:

- Dimension Name: odim_customer
- Attribute: customer_name
- Relationship Name: Customer Name
- Related Attribute: customer_name
- Relationship Type: Rigid
- Cardinality: One
- Visible: True

Buttons: OK, Cancel

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 odim_customer Hierarchy Definition

Hierarchy Properties

Language Mapping

Dimension Name: odim_customer

Internal Hierarchy Name: customer

Hierarchy Publish Name: customer

Hierarchy Description: Added at dimension creation for cube support

All Member Name:

Allow Duplicate Names: True

Member Keys Unique: NotUnique

Member Names Unique: False

Display Folder:

OK Cancel Help

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.

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 an Attribute Relationship, 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**.

The screenshot shows a dialog box titled "Dimension odim_customer Hierarchy Level Definition". The dialog is divided into two main sections. On the left is a sidebar with two items: "Hierarchy Level Properties" (which is selected) and "Language Mapping". The main area of the dialog contains several fields and dropdown menus:

- Dimension Name:** A text box containing "odim_customer".
- Hierarchy:** A dropdown menu with "customer" selected.
- Level Number:** A text box containing "2".
- Internal Level Name:** A text box containing "Group Code".
- Level Publish Name:** A text box containing "Group Code".
- Level Description:** A text area containing "The group code for the customer. Forms a hierarchy with customer_code".
- Source Attribute:** A dropdown menu with "customer_group_code" selected.
- Hide If:** A dropdown menu with "Never" selected.

At the bottom right of the dialog are three buttons: "OK", "Cancel", and "Help".

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 contain 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**.

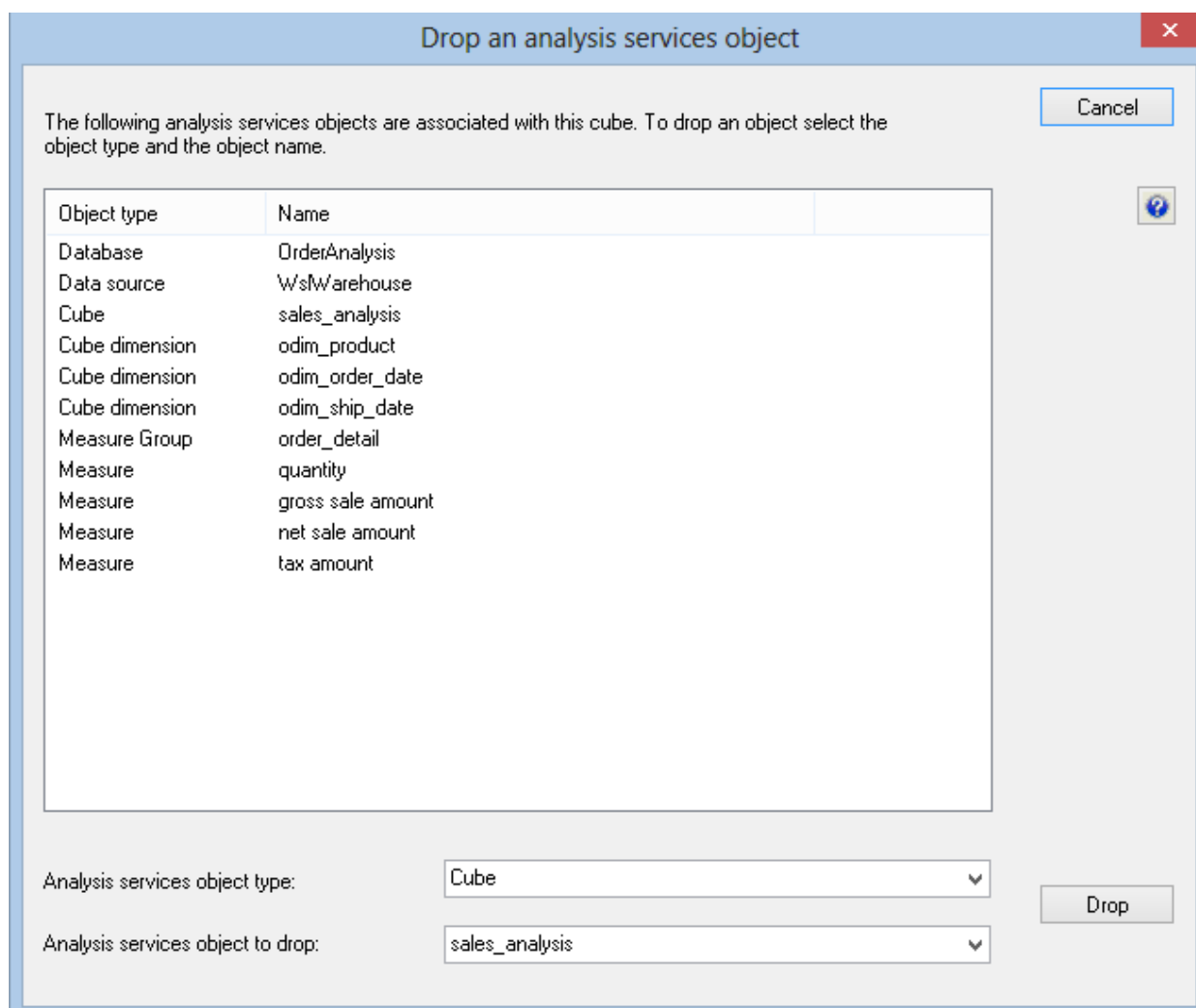
OLAP CHANGING OLAP CUBES

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 this is 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 using the drop-down boxes in the Drop Analysis Services Object dialog 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 or not 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 in order to be changed (eg dimension structures), and some only need to be recreated (eg 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 brief 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. Therefore, 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). |

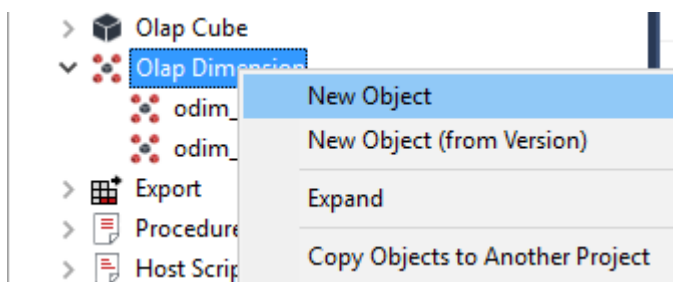
OLAP RETROFITTING AN OLAP OBJECT

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**.



- 2 Enter any name for the object name and click **ADD**.

Add a New Metadata Object ✕

Define the Type and Name of the New Object.
Specific information for each object type is defined in subsequent screens.

Object Type:

Object Name:

Include Attribute Relationships

3 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.

Olap Dimension odim_customer

Properties

Language Mapping

Purpose

Concept

Grain

Examples

Usage

Notes

Internal Dimension Name: odim_customer

Dimension Publish Name: dim_customer

Dimension Description: Customer Dimension

Default Database Connection: (Analysis Services) Cubes

OLAP Database Name: Tutorial5

Data Source Connection: (Data Warehouse) DataWarehouse

Data Source Provider Type:

Data Source Server:

Data Source Database:

Post Create XML/A Script: (None) Edit

Source Table Type:

Source Table:

Source Table Key:

Processing Group: ByAttribute

Processing Mode: Regular

Processing Method: Default process

Storage Mode: MDLAP

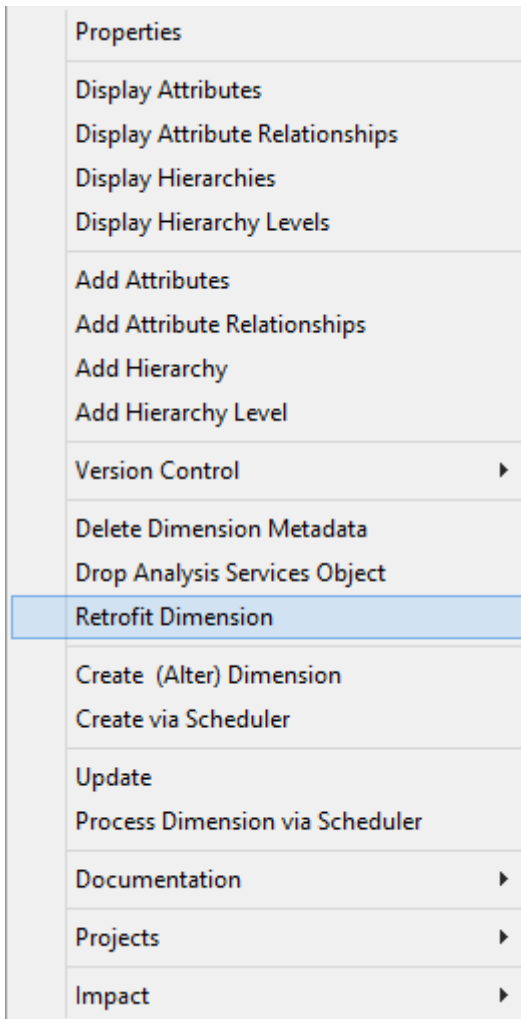
All Caption: All odim_customer

OLAP Dimension Type:

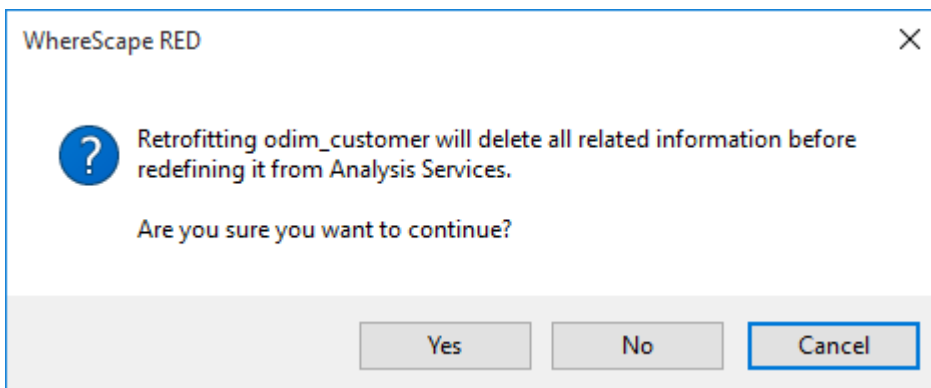
Unknown Member Action: None Unknown Member Name: Unknown

OK Cancel Help

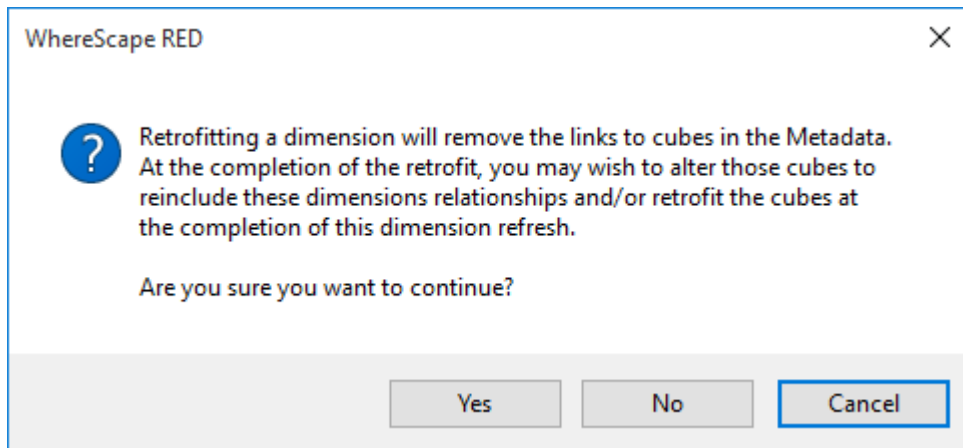
- The OLAP Dimension can now be retrofitted by right-clicking the object in the left pane and choosing **Retrofit Dimension**.



- Two warning dialogs now appear. The first dialog warns that the existing information will be deleted before being redefined from Analysis Services. Select **Yes**.



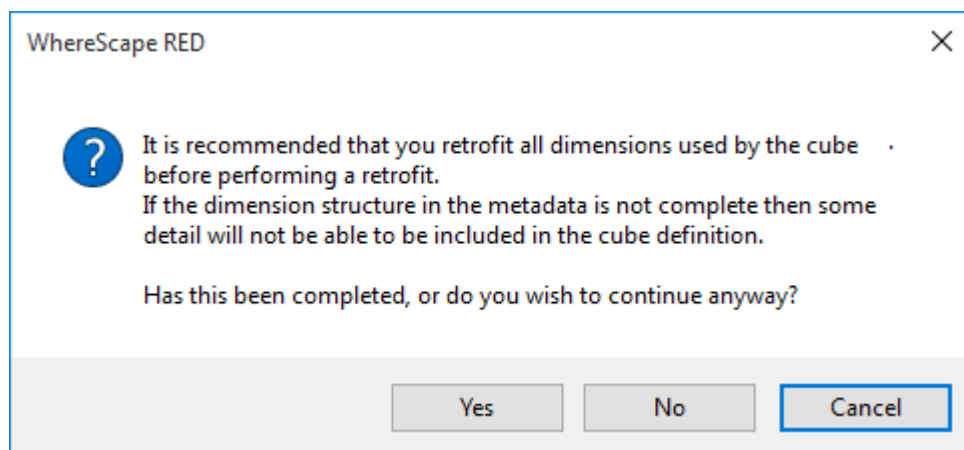
- The second dialog warns that retrofitting a dimension will remove the links to cubes in the Metadata. Select **Yes**.



- 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.

CHAPTER 22

TRANSFORMATIONS

Standard *column transformations* (on page 594) 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 two ways in WhereScape RED:

- Teradata User Defined Functions (UDFs)
- WhereScape RED User Defined Transformations

IN THIS CHAPTER

| | |
|--------------------------------------|-----|
| Column Transformations | 594 |
| Teradata User Defined Functions..... | 603 |
| Re-usable Transformations | 606 |

COLUMN TRANSFORMATIONS

Each table, view, join index or export object column can have a transformation associated with it. For all table types, except for load tables, views and join indexes, 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.

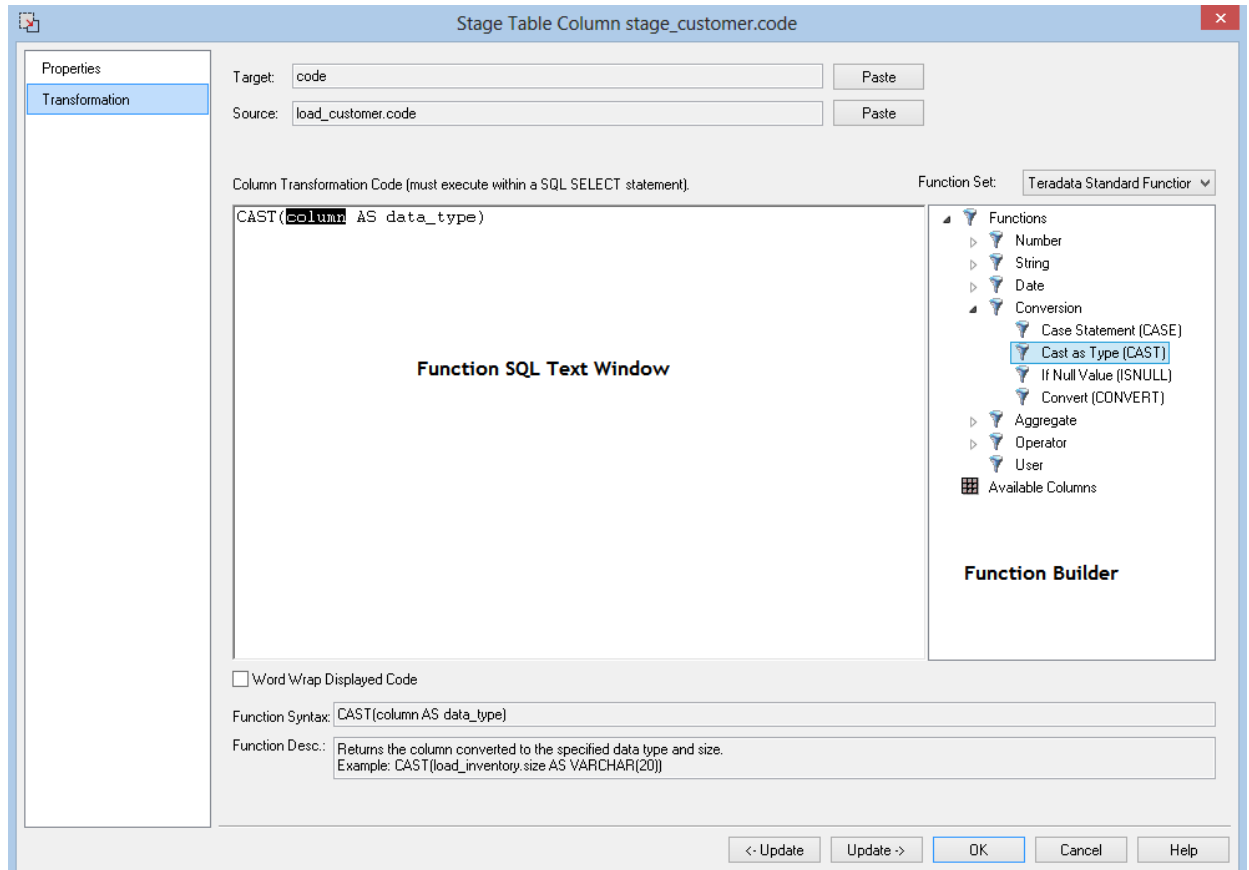
Column transformations on load tables are more complex, due to the unique nature of load tables. See **Load Table Column Transformations** (on page 598) for more details.

View and Join Index transformations are included in the Database Definition Language (DDL) that creates the object in the Teradata database. Any changes to transformations require these object types to be dropped and recreated.

Export object column transformations are dynamically applied for file loads. 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 **User Defined 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 598) 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 the function is clicked. Essentially the same as the function model loaded into the **Function SQL Text Window** when the function is double-clicked. 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 **User Defined Transformations**. This is only visible for User Defined Transformations. Read only.

Localize Transformation

The **Localize transformation** button breaks the link between a **column transformation** and the **User Defined Transformation** it's based on. If this button is clicked, changes to the underlying user defined transformation cannot be automatically propagated to the column transformation. This is only visible for User Defined Transformations.

LOAD TABLE COLUMN TRANSFORMATIONS

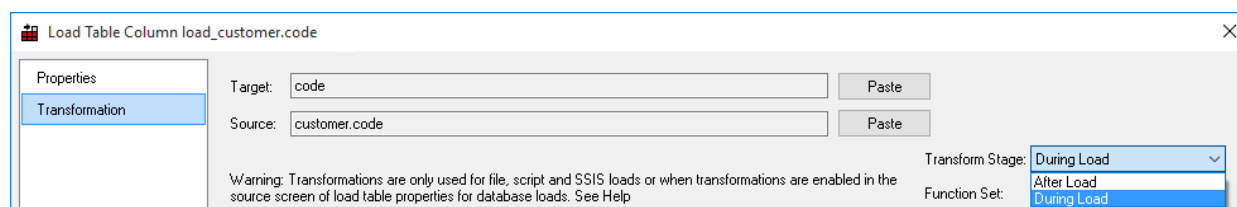
Overview

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 | <ul style="list-style-type: none"> • During Load transformations • After Load transformations • Post Load procedure |
| ODBC Based Load | <ul style="list-style-type: none"> • During Load transformations • After Load transformations • Post load procedure |
| File Based Load | <ul style="list-style-type: none"> • During Load transformations • After Load transformations • Post load procedure |
| Integration Services Load | <ul style="list-style-type: none"> • During Load transformations • After Load transformations |
| Script Based Load | <ul style="list-style-type: none"> • During Load transformations • After Load transformations • Post load procedure |
| Externally Loaded | <ul style="list-style-type: none"> • 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.

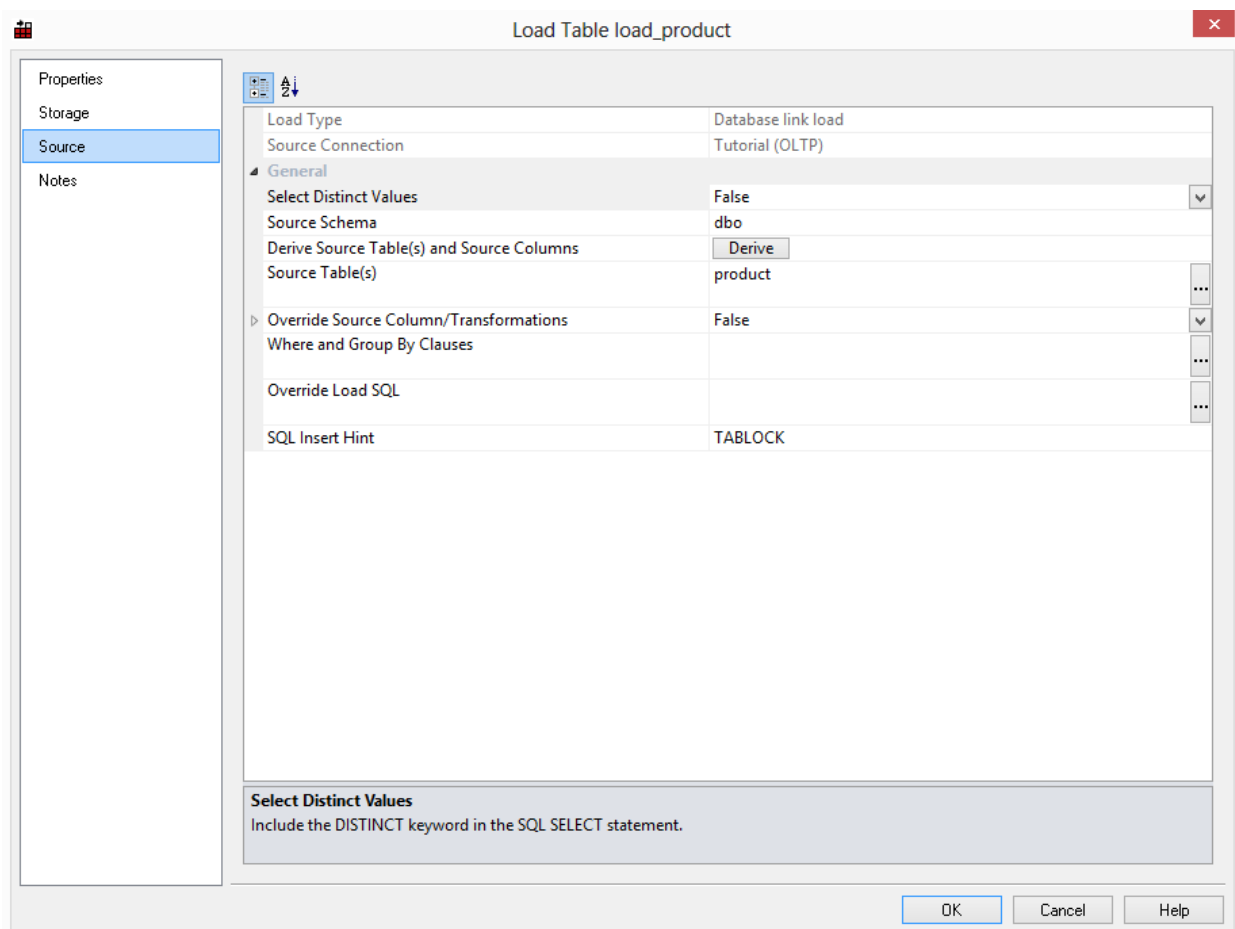
All **After** transformations take place in the native Teradata SQL language. The **During** transformations differ in terms of which language is used. This is particularly true for file based loads. Normally the **During** transformation will occur in the native SQL language of the source database.

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](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 |
| | | |  |  |

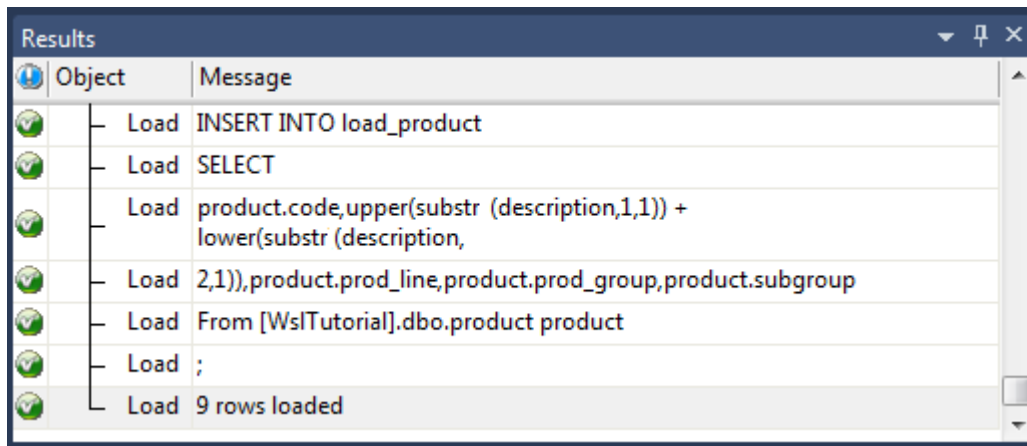
The SQL code from the load results is:

| Results | |
|--|--|
| Object | Message |
|  load_proc | Table load_product truncated. |
|  Load | INSERT INTO load_product |
|  Load | SELECT |
|  Load | product.code,product.description,product.prod_line,product.prod_group,pr |
|  Load | From [WsITutorial].dbo.product product |
|  Load | ; |
|  Load | 9 rows loaded |

If the column 'description' has a transformation defined as follows:

upper(substr(description,1,1))||lower(substr(description,2,1))

then the following SQL statement will be executed.



The screenshot shows a 'Results' window with a table of execution messages. The table has two columns: 'Object' and 'Message'. The messages represent the steps of a SQL query execution, from the initial insert to the final row count.

| Object | Message |
|--------|--|
| Load | INSERT INTO load_product |
| Load | SELECT |
| Load | product.code,upper(substr (description,1,1)) + lower(substr (description, |
| Load | 2,1)),product.prod_line,product.prod_group,product.subgroup |
| Load | From [WsITutorial].dbo.product product |
| Load | ; |
| Load | 9 rows loaded |

FILE DURING LOAD TRANSFORMATIONS

The loading of flat files is performed using Fastload, Multi-load or TPT. The contents of the **Transformation** tab in a column's Properties are the functions and conversions supported by the database loader.

Example

Multiload performs transformations such as:

- `FORMAT 'DD-MMM-YYYY'` converts from a value such as 23-Mar-1999 to a Teradata date.
- `CAST(:COL AS NUMERIC(18,4))` converts the data to a numeric(18,4)
- `COALESCE(LTRIM(RTRIM(: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

substr(code,1,5)

would result in the following SQL statement being executed after the load:

```
update load_product set code = substr(code,1,5);
```

TERADATA USER DEFINED FUNCTIONS

Teradata User Defined Functions (UDFs) can be built using the C programming language and compile directly onto the Teradata server and registered via a SQL command.

This can be daunting for many data warehouse developers, who may not know C well, and may not have access to compile UDFs on the Teradata server.

TERADATA UDF EXAMPLE

A standard example given for Teradata UDFs is building a REPLACE function similar to the Oracle database's REPLACE function.

Here is a simplified example:

Step 1: Write c code for UDF as replace.c:

```
void replace ( VARCHAR_LATIN *inputStr,
              VARCHAR_LATIN *inputFrom,
              VARCHAR_LATIN *inputTo,
              VARCHAR_LATIN *result,
              int *inputStrIsNull,
              int *inputFromIsNull,
              int *inputToIsNull,
              int *resultIsNull,
              char sqlstate[6],
              SQL_TEXT extname[129],
              SQL_TEXT specific_name[129],
              SQL_TEXT error_message[257])
{
    int linputFrom;
    int linputTo;
    char *pinputStr;
    strcpy(sqlstate, NoSqlError);
    strcpy((char *) error_message, " ");
    *resultIsNull = IsNotNull;
    *result = '\0';
    if (*inputStrIsNull == IsNull)
    {
        strcpy(sqlstate, "22004");
        strcpy((char *) error_message, "First argument cannot be null.");
        *resultIsNull = IsNull;
        return;
    }
    if (*inputFromIsNull == IsNull || (linputFrom=strlen((const char *)inputFrom)) == 0)
    {
        strcpy((char *)result, (const char *)inputStr);
        return;
    }
    linputTo = (*inputToIsNull == IsNull ? 0 : strlen((const char *)inputTo));
    for (pinputStr = (char *)inputStr; *pinputStr != '\0'; )
    {
        if (strncmp((const char *)pinputStr, (const char *)inputFrom, linputFrom) == 0)
        {
            strncat((char *)result, (const char *)inputStr, (int)(pinputStr-(char *)inputStr));
            if (linputTo != 0) strcat((char *)result, (const char *)inputTo);
            pinputStr = (char *)inputStr = pinputStr + linputFrom;
        }
        else
            ++pinputStr;
    }
    strcat((char *)result, (const char *)inputStr);
}
```

Step 2: Put c file onto the Server

Copy the c program file onto the data base server into the following location:

C:\program files\ncr\tdat\tdconfig\tdbs_udf\replace.c

Step 3: Register and Compile UDF via SQL:

```
CREATE FUNCTION replace
( Str VARCHAR(512)
, aFrom VARCHAR(512)
, aTo VARCHAR(512)
)
RETURNS VARCHAR(512)
LANGUAGE C
NO SQL
SPECIFIC replace2
EXTERNAL NAME 'SS!replace!C:\program files\ncr\tdat\tdconfig\tdbs_udf\replace.c'
PARAMETER STYLE SQL;
```

Step 4: Use the Function

Simply use the UDF in a RED column transformation or 'Where' clause like any other SQL function.

RE-USABLE TRANSFORMATIONS

WhereScape RED Re-usable Transformations allow a complex SQL transformation using standard Teradata functions (including UDFs) to be built once and reused in multiple column transformations. Functionality is included to manage and propagate changes to user defined transformations.

CREATING A NEW RE-USABLE TRANSFORMATION

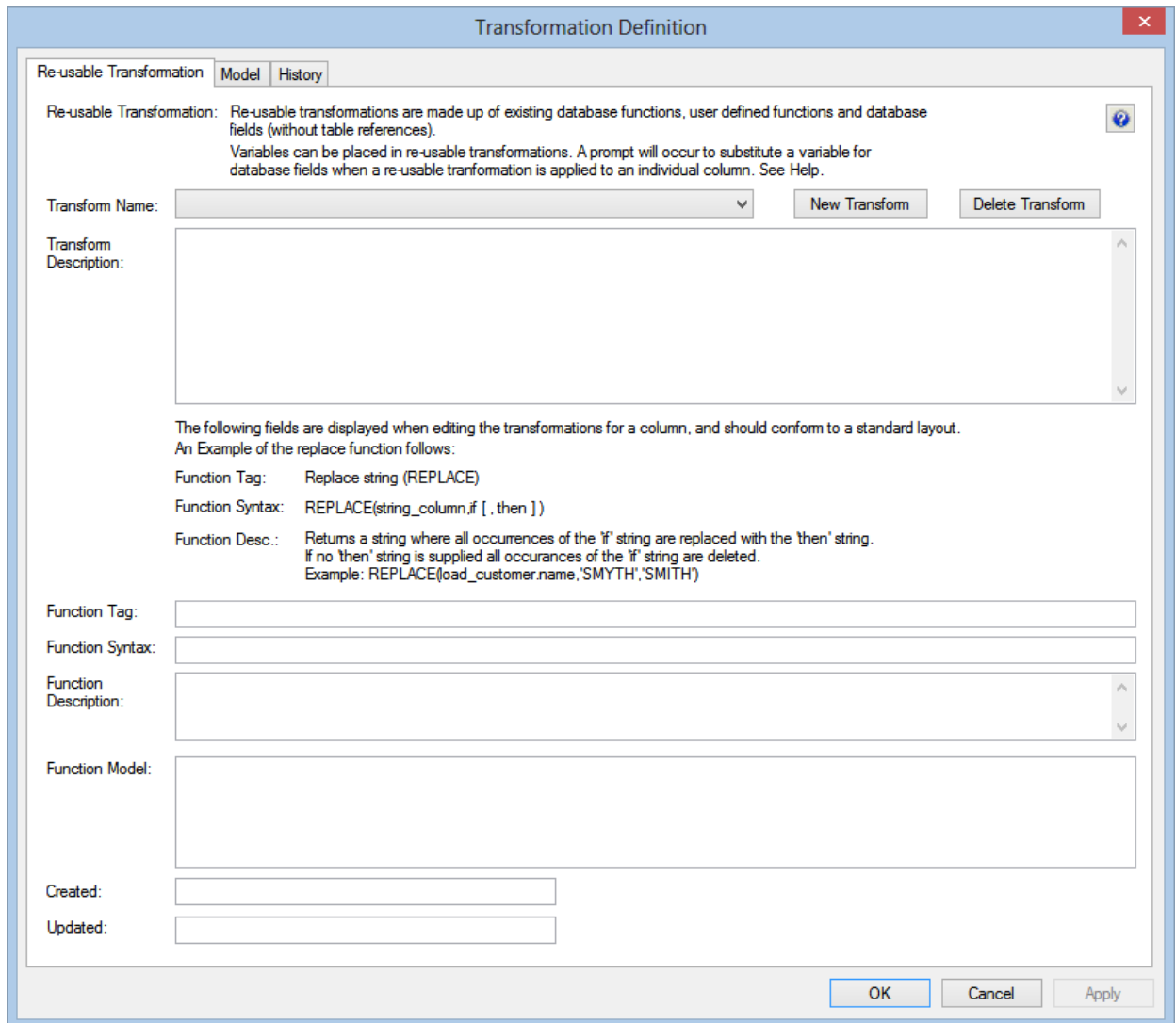
New **re-usable transformations** are created from the **Tools/Define Re-Usable Transformations** menu.

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:



The image shows a dialog box titled "Transformation Definition" with a close button (X) in the top right corner. It has two tabs: "Re-usable Transformation" (selected) and "Model History".

Under the "Re-usable Transformation" tab, there is a text area with the following text: "Re-usable Transformation: Re-usable transformations are made up of existing database functions, user defined functions and database fields (without table references). Variables can be placed in re-usable transformations. A prompt will occur to substitute a variable for database fields when a re-usable transformation is applied to an individual column. See Help." There is a help icon (question mark) to the right of this text.

Below this text is a "Transform Name:" label followed by a dropdown menu. To the right of the dropdown are two buttons: "New Transform" and "Delete Transform".

Below the dropdown is a "Transform Description:" label followed by a large text area.

Below the text area is a paragraph: "The following fields are displayed when editing the transformations for a column, and should conform to a standard layout. An Example of the replace function follows:"

Below this paragraph are three lines of text:

- Function Tag: Replace string (REPLACE)
- Function Syntax: REPLACE(string_column,if [, then])
- Function Desc.: Returns a string where all occurrences of the 'if' string are replaced with the 'then' string. If no 'then' string is supplied all occurrences of the 'if' string are deleted. Example: REPLACE(load_customer.name,'SMYTH','SMITH')

Below these examples are four input fields:

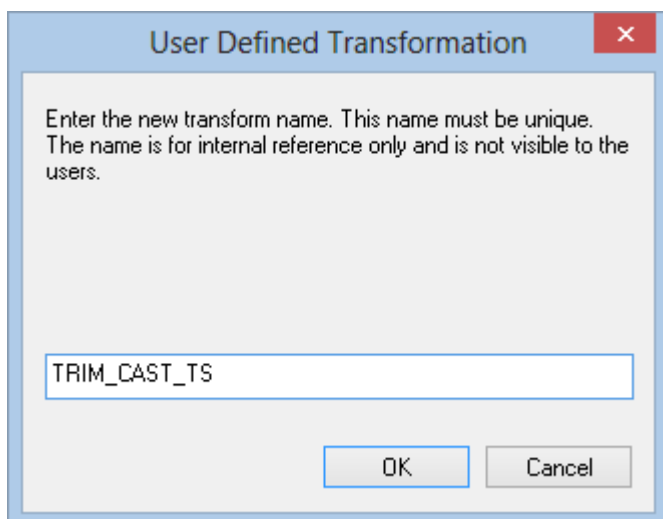
- Function Tag: [text input]
- Function Syntax: [text input]
- Function Description: [text area with scrollbars]
- Function Model: [text area]

At the bottom left are two more input fields:

- Created: [text input]
- Updated: [text input]

At the bottom right are three buttons: "OK", "Cancel", and "Apply".

Click **New Transform** and enter a name for the User defined 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 name the function will appear as for users to select 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

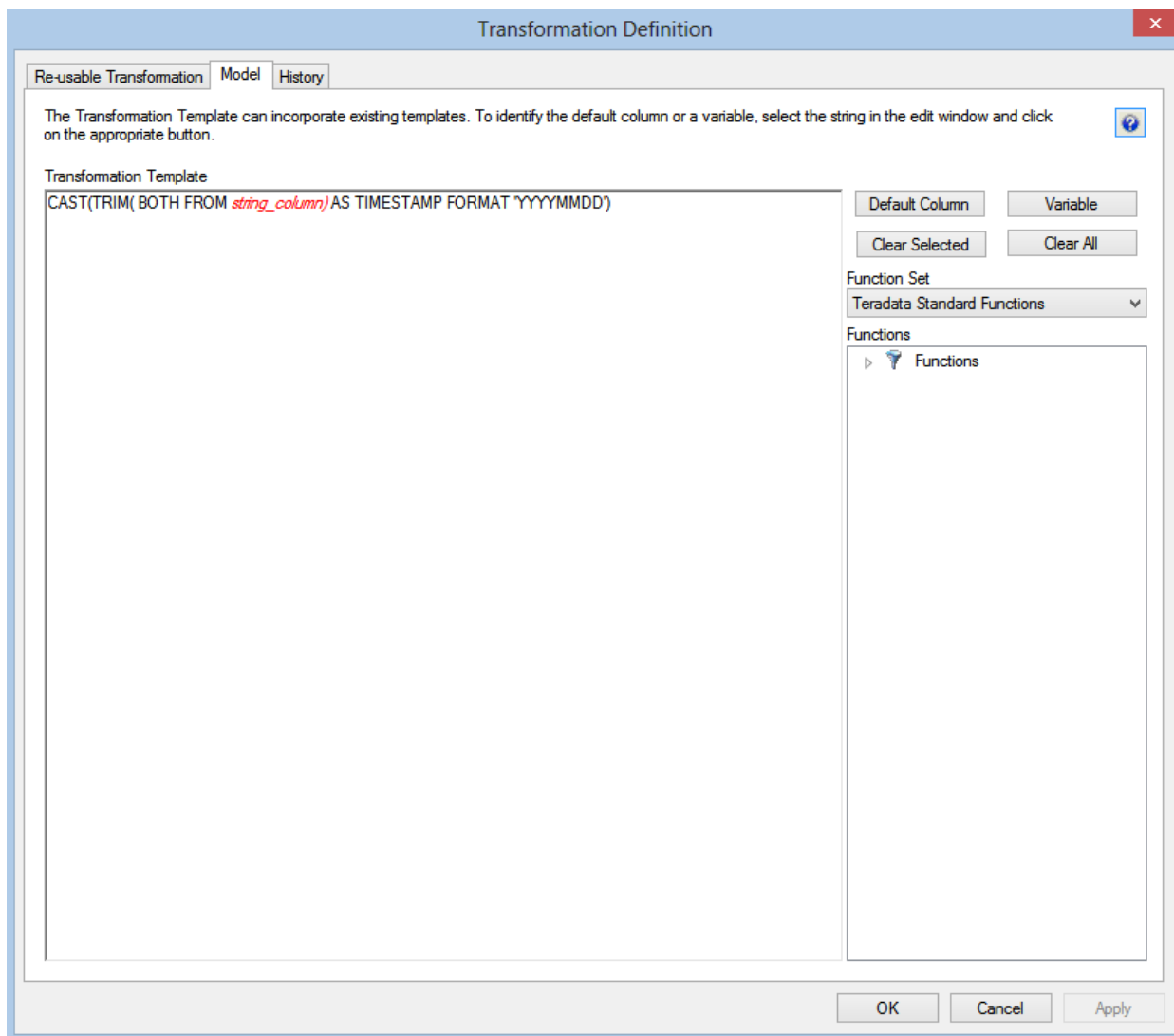
Once the transformation has been created and the metadata entered, the actual SQL code used by the transformation can be defined. The SQL code can be entered directly or via the **Function Builder** on the right side. To use the function builder, expand the tree to find the required function.

Example of Building a Transformation Model

To build a model to CAST a trimmed string in YYYYMMDD format to a timestamp, do the following:

- Click on the **Model** tab
- Expand **Functions** heading
- Expand the **Conversion** heading
- Double click on **Cast as Type (CAST)**
- Expand the **String** heading
- Double click on **Trim (TRIM)**
- Highlight **data_type** and type **TIMESTAMP FORMAT 'YYYYMMDD'**

You should see the following:



Now Click **OK**.

COMPLETED RE-USABLE TRANSFORMATION

✕
Transformation Definition

Re-usable Transformation Model History

Re-usable Transformation: Re-usable transformations are made up of existing database functions, user defined functions and database fields (without table references). ?

Variables can be placed in re-usable transformations. A prompt will occur to substitute a variable for database fields when a re-usable transformation is applied to an individual column. See Help.

Transform Name: New Transform Delete Transform

Transform Description:

The following fields are displayed when editing the transformations for a column, and should conform to a standard layout.
An Example of the replace function follows:

Function Tag:

Function Syntax:

Function Desc.:

Function Tag:

Function Syntax:

Function Description:

Function Model:

Created:

Updated:

OK Cancel Apply

CHANGING A RE-USABLE TRANSFORMATION

To change a re-usable transformation:

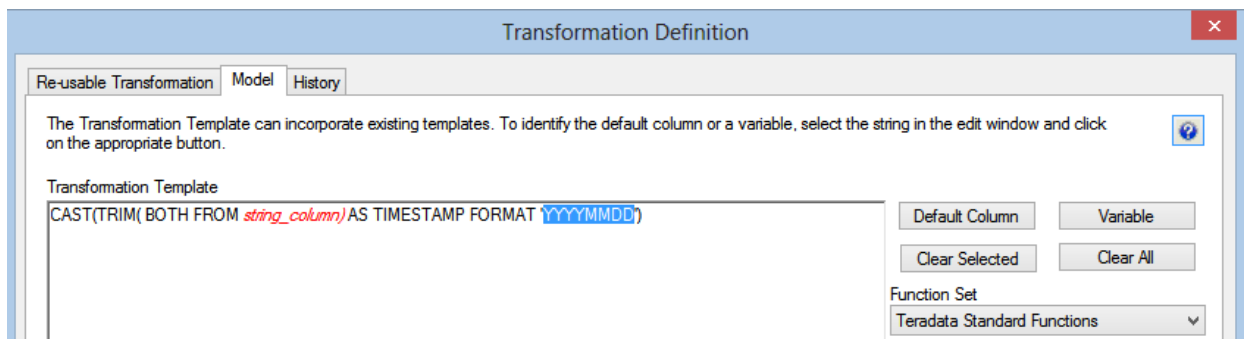
- Select **Re-usable transformations** from the Tools menu.
- Choose the transformation from the **Transform Name** drop-down list.
- Click on the **Model** Tab.
- Change the SQL as required.
- Click **OK**.

Example of a change to the Model SQL

In the example used in *Creating a New User Defined Transformation* (see "*Creating a New Re-usable Transformation*" on page 606), the SQL was:

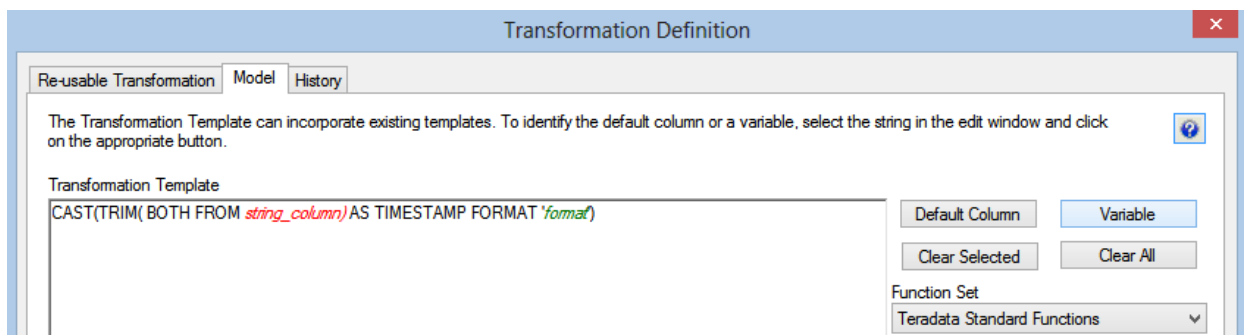
```
CAST(TRIM( BOTH FROM string_column) AS TIMESTAMP FORMAT 'YYYYMMDD')
```

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 that can be substituted when the User Defined Transformation is used.

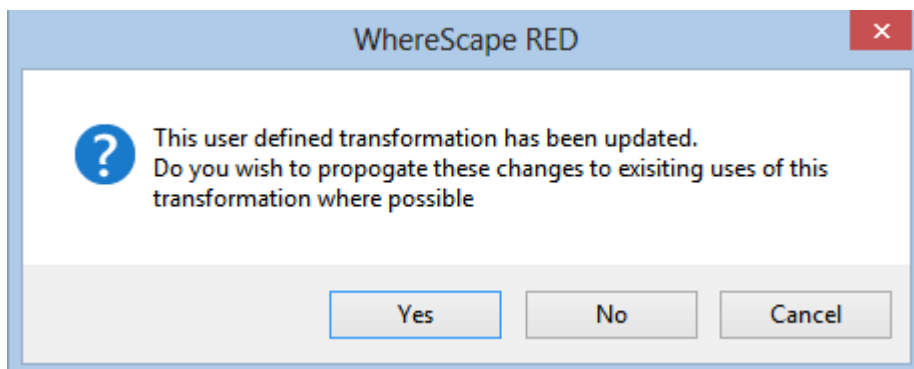
Now **format** is green and in italics:



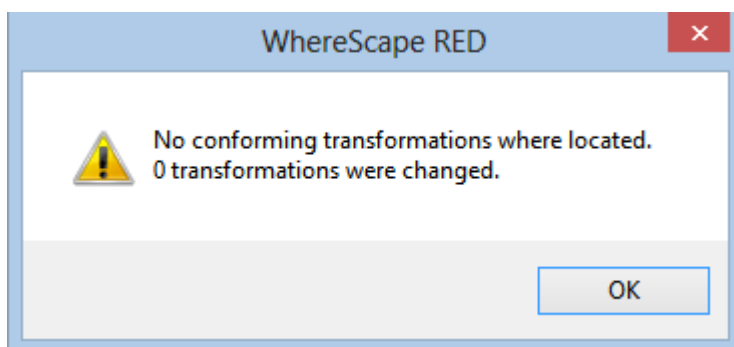
Click **OK**.

APPLYING CHANGES TO DEPENDANT TRANSFORMATIONS

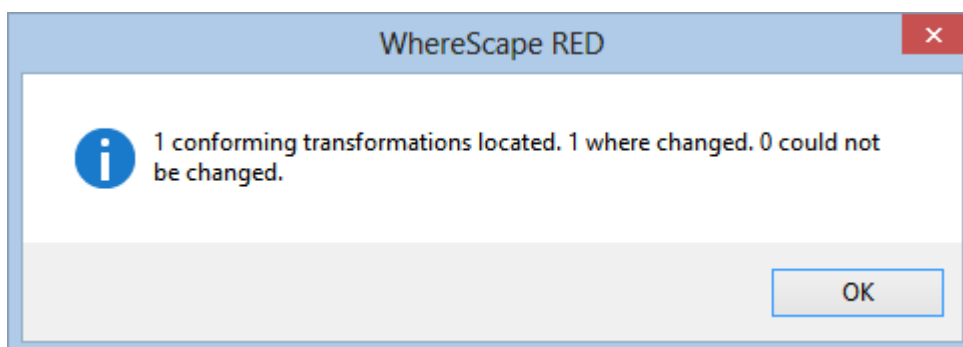
After changing a **Re-usable Transformation**, a dialog appears asking to confirm that changes should be applied to individual columns using the transformation, where possible:



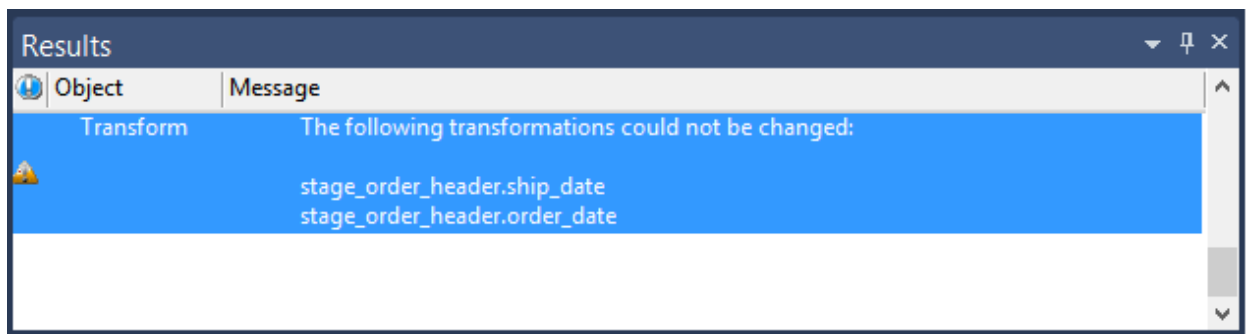
If the **Re-usable Transformation** doesn't have any dependant columns, then the following message is displayed:



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 User Defined 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

User defined 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 595)

CHAPTER 23

EXPORTING DATA

Export objects are used in 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.

- **File export** - an export where most of the processing is managed and controlled by the scheduler.
- **Script-based export** - an export where a Windows or a Unix/Linux script file is executed to perform the export. Script-based exports on Windows supports both DOS Batch and PowerShell scripts *for more information* (see "**24.11.1.1 Windows PowerShell Scripts**" on page 657).
- **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

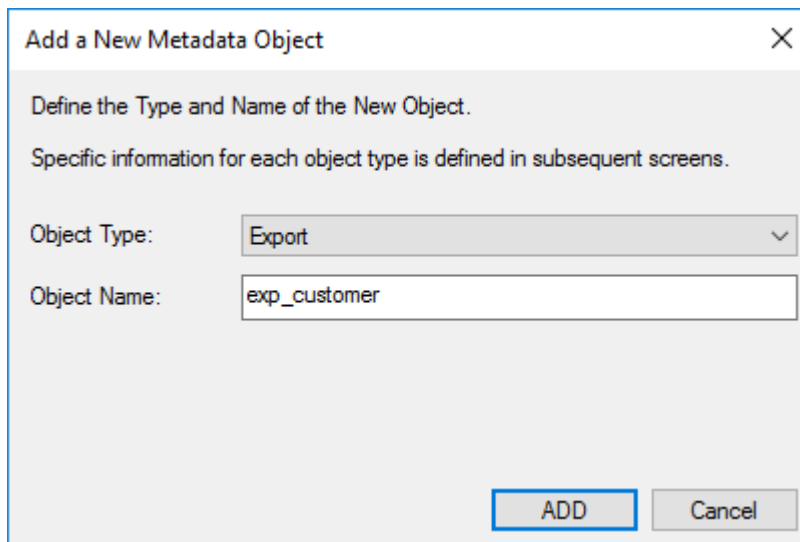
| | |
|---------------------------------|-----|
| Building an Export Object | 617 |
| File Attributes..... | 621 |
| Export Column Properties | 626 |
| Script based Exports | 628 |

BUILDING AN EXPORT OBJECT

Creating an Export:

- 1 Browse the Data Warehouse connection.
- 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:



Add a New Metadata Object

Define the Type and Name of the New Object.
Specific information for each object type is defined in subsequent screens.

Object Type: Export

Object Name: exp_customer

ADD Cancel

- 4 Rename the export object if it needs to be renamed and click **ADD**.

The Properties dialog displays:

Export exp_test

Properties

File Attributes

Storage

Purpose

Concept

Grain

Examples

Usage

Notes

Export Object Name: exp_customer

Unique Short Name: (maximum 22 characters) exp_customer

Description:

Connection:

Export Type: File export

Database Link:

Script Template: (None)

Script Name: (None)

Pre-Export Action: No action

Pre-Export Sql:

Where Clause: Allows filtering of the export data.

Post Export Procedure: (None)

Timestamps

Metadata Structure Changed: 2016-08-07 22:53:58.530000

Last Exported:

OK Cancel Help

5 If exporting the object via **Windows**, change the Connection to **Windows**.

From a Windows connection, **File**, **Script based** and **Integration Services** export methods are supported.

Export exp_test

Properties

File Attributes

Storage

Purpose

Concept

Grain

Examples

Usage

Notes

Export Object Name: exp_customer

Unique Short Name: (maximum 22 characters) exp_customer

Description:

Connection: Windows

Export Type: File export

Database Link:

Script Template: (None)

Script Name: (None)

Pre-Export Action: No action

Pre-Export Sql:

Where Clause: Allows filtering of the export data.

Post Export Procedure: (None)

Timestamps

Metadata Structure Changed: 2016-08-07 22:53:58.530000

Last Exported:

OK Cancel Help

- 6 If exporting the object via **Unix/Linux**, change the connection to **Unix**.

From a **Unix/Linux** connection **File export** and **Script based** exports methods are supported.

Export exp_test

Properties

File Attributes

Storage

Purpose

Concept

Grain

Examples

Usage

Notes

Export Object Name: exp_customer

Unique Short Name: (maximum 22 characters) exp_customer

Description:

Connection: UNIX

Export Type: File export

Database Link:

Script Template: (None)

Script Name: (None)

Pre-Export Action: No action

Pre-Export Sql:

Where Clause: Allows filtering of the export data.

Post Export Procedure: (None)

Timestamps

Metadata Structure Changed: 2016-08-07 22:53:58.530000

Last Exported:

OK Cancel Help



TIP: When doing a **Script based export** on Windows or Unix/Linux, use the **Rebuild** button after selecting the relevant script to be rebuilt on the the Script Name drop-down menu.

- Click on the **File Attributes** tab to fill in the fields described in the next section to set the relevant location, name and contents of the exported data file.
- When the File Attributes fields are filled in, go back to the **Properties** tab, select (**Build Script**) from the **Script Name** drop-down menu and click **OK**.
- Export the object after filling in the Files Attributes fields by right-clicking on the export object on the left pane and selecting **Export**.

FILE ATTRIBUTES

The following fields are available to define the location, name and contents of the exported data file:

- **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 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 Routine** - database specific routine to use to export the data. FastExport or TPT Data Connector for Windows exports, or TPT Data Connector for Unix/Linux exports.
- **Export Format** - routine-specific format to use to export the data. Select one of the default Teradata formats: Text, Unformatted, Delimited, Binary, Formatted, FastLoad or a special WhereScape RED derived format: Delimited or Width Fixed Text.
- **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.
- **Export Options** - allows the entry of export utility options. If more than one option is required a semi-colon should be used between options.
- **Header Row in Export** - if a header line is required, choose business names or column names from this drop-down list.
- **Trigger Path** - the trigger file indicates that the export to the main file has completed and 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** - this 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 customer_YYYYMMDD\$.txt which would result in a file name like customer_20150520.txt.
- **Trigger Delimiter** - multiple fields in the trigger file need to be separated by the trigger delimiter.

- **Trigger Parameter 1,2,3** - 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** - tick this check-box if you want to compress the export file after it has been created.
- **Compress Utility Path** - directory in which the compress utility exists.
- **Compress Utility Name** - name of the compression utility executable.
- **Compress Parameters** - name of the file to be compressed (using the RED variable \$EXPPFILE\$) and any commands or switches required to make the compression utility work. These parameters will depend on the compression utility used.

For Windows **File** or **Script based** Exports, you can select the between the **FastExport** or **TPT Data Connector** export routines.

1 Example screen for a Windows Script based Export using the **TPT Data Connector**.

The screenshot shows the 'Export exp_customer' dialog box with the following configuration:

| Property | Value |
|-------------------------------|-------------------------------------|
| Export Type | File export |
| Destination Connection | Windows |
| Export File Definition | |
| Export File Path | c:\data\ |
| Export File Name | customer_YYYYMMDD\$.txt |
| Export Routine | TPT Data Connector |
| Export Format | Delimited |
| Export File Delimiter | |
| Optionally Enclosed By | |
| Export Options | |
| Header Row | Column names |
| Trigger File | |
| Trigger Path | c:\data |
| Trigger Name | customer_YYYYMMDD\$.txt |
| Trigger Delimiter | |
| Trigger Parameter 1 | Row Count |
| Trigger Parameter 2 | |
| Trigger Parameter 3 | |
| Compression | |
| Compress After Export | <input checked="" type="checkbox"/> |
| Compress Utility Path | c:\Program Files\7-Zip\ |
| Compress Utility Name | 7z.exe |
| Compress Parameters | a -y\$EXPPFILES.zip \$EXPPFILES |

Export Routine
Database-specific routine to use to export the data.

Buttons: OK, Cancel, Help

2 Example screen for a **Unix/Linux** File/Script based Export using the **TPT Data Connector**.

| | |
|-------------------------------|-------------------------------------|
| Export Type | Script based export |
| Destination Connection | UNIX |
| Export File Definition | |
| Export File Path | /home/red/wsl/export |
| Export File Name | customer_SYYYYMMDDS.txt |
| Export Routine | TPT Data Connector |
| Export Format | Delimited |
| Export File Delimiter | |
| Optionally Enclosed By | |
| Export Options | |
| Header Row | Column names |
| Trigger File | |
| Trigger Path | home/red |
| Trigger Name | customer_SYYYYMMDDS.trg |
| Trigger Delimiter | |
| Trigger Parameter 1 | Row Count |
| Trigger Parameter 2 | |
| Trigger Parameter 3 | |
| Compression | |
| Compress After Export | <input checked="" type="checkbox"/> |
| Compress Utility Path | /bin |
| Compress Utility Name | gzip |
| Compress Parameters | -f9 \$EXPFILES |

Export File Definition

OK Cancel Help



TIP: With all the relevant fields filled in, remember to go back the Properties tab and select **(Build Script)** from the Script Name drop-down menu.

Right-click on the export object on the left pane and select **Export**.

FILE ATTRIBUTES - SSIS EXPORTS

The following fields below are available to define the location, name and definitions of the exported data file:

Integration Services File Attributes screen:

| | |
|--|-------------------------------------|
| Export Type | Integration Services export |
| Destination Connection | Windows |
| Export File Definition | |
| Export File Path | c:\data\ |
| Export File Name | customer_YYYYMMDD\$.txt |
| Export Format | Delimited |
| Export File Delimiter | |
| Optionally Enclosed By | |
| Header Row | Column names |
| SQL Server Integration Services (SSIS) | |
| SSIS Row Count Log | <input checked="" type="checkbox"/> |

Export File Path
Full path (absolute path) of the folder/directory containing the Export File on the Windows or UNIX/Linux system.

OK Cancel Help

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

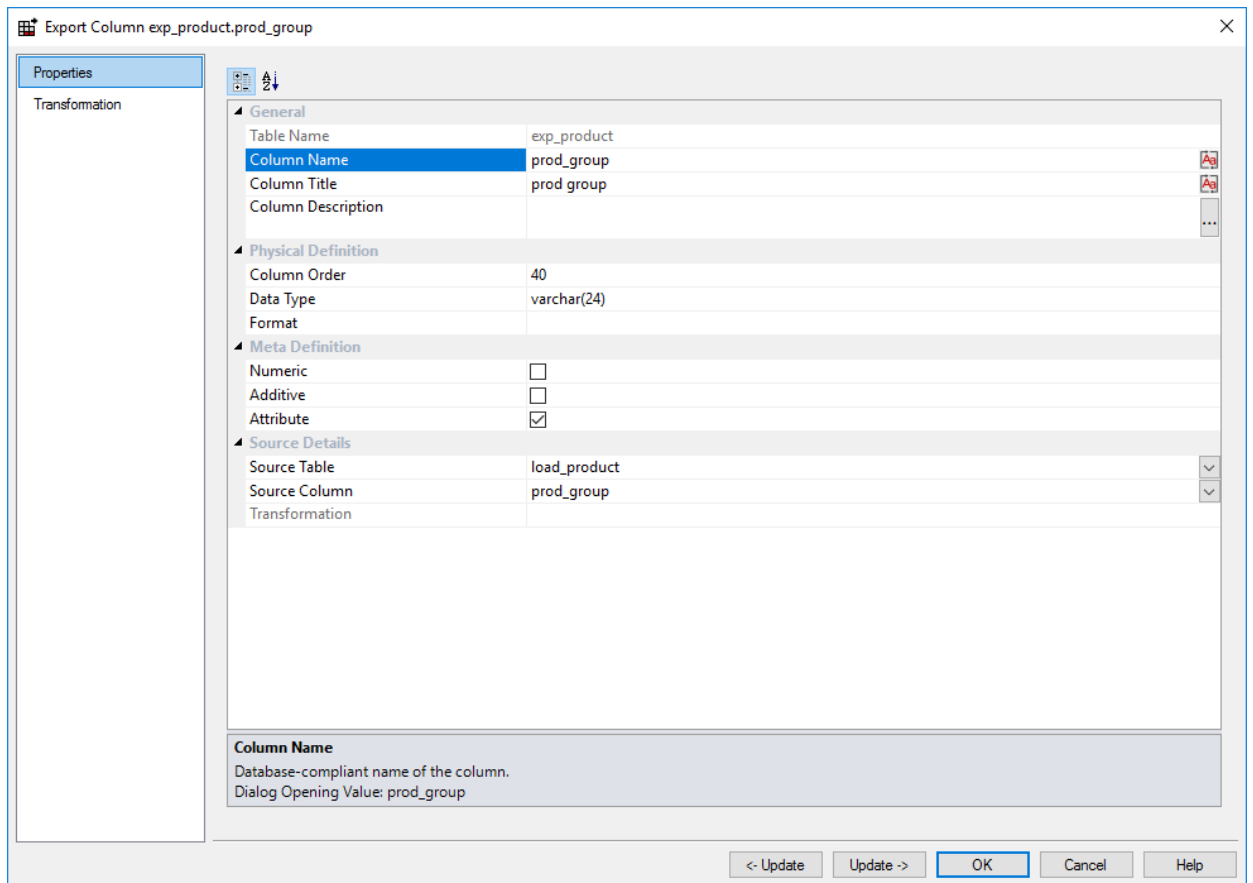
If a header line is required, choose business names or column names from this drop-down list. This option is not available in DB2.

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 field].
- **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 alphanumeric, 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.

- **Column Title** - 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`. It 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 a valid for the target database. Typical Teradata databases often have integer, numeric(), varchar(), char(), date and timestamp data types. 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 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** - this is a Read-only field.

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 an 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 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.

Export exp_customer

Properties

File Attributes

Storage

Purpose

Concept

Grain

Examples

Usage

Notes

Export Object Name: exp_customer

Unique Short Name: (maximum 22 characters) exp_customer

Description:

Connection: Windows

Export Type: Script based export

Database Link:

Script Template: (None)

Script Name: exp_customer_script Edit Rebuild

Pre-Export Action: No action

Pre-Export Sql:

Where Clause: Allows filtering of the export data.

Post Export Procedure: (None)

Timestamps

Metadata Structure Changed: 2017-03-15 00:30:05.100000

Last Exported:

OK Cancel Help

Note: Script-based exports on Windows supports both DOS Batch and PowerShell scripts *for more information* (see "24.11.1.1 Windows PowerShell Scripts" on page 657).

CHAPTER 24

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 Windows batch files.

Procedures

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. In this chapter the generation, editing and compilation of procedures is covered.

Scripts

Host scripts are generated when a script based file load is chosen during a file drag and drop from a Windows 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 testing of host scripts as well as explaining the components required to allow them to work in the scheduler environment.

IN THIS CHAPTER

| | |
|---|-----|
| Procedure Generation | 632 |
| Procedure Editing | 638 |
| Procedure Loading and Saving | 641 |
| Procedure Comparisons | 644 |
| Procedure Compilation | 645 |
| Procedure Running..... | 646 |
| Procedure Syntax..... | 646 |
| Procedure Properties | 647 |
| Macros | 650 |
| BTEQ Scripts..... | 650 |
| Script Generation | 651 |
| Script Editing..... | 660 |
| Script Testing | 662 |
| Script Syntax | 662 |
| Script Environment Variables | 665 |
| Calling a Batch File from a Script..... | 671 |
| Scheduling Scripts..... | 674 |
| Manually created scripts | 677 |

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 **Stage, Model** and **Aggregate** tables an Update or Initial Build procedure can be generated by selecting the option above from the appropriate drop-down box.

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 model table that is updated from multiple stage tables. This wrapper procedure calls two child procedures, one for each stage table that is to update the model table. A status is reported back to the audit trail for each stage and an overall status ascertained for the model table update.

The wrapper procedure may look as follows:

```

CREATE PROCEDURE [METABASE].update_model_order_header
(
  IN p_sequence          integer,
  IN p_job_name         varchar(256),
  IN p_task_name        varchar(256),
  IN p_job_id           integer,
  IN p_task_id          integer,
  OUT p_return_msg      varchar(256),
  OUT p_status          integer
)
BEGIN

  -- =====
  -- DBMS Name      :      Teradata
  -- Script Name    :      update_model_order_header
  -- Description    :      Build the table model_order_header
  -- Generated by   :      WhereScape RED
  -- Generated for  :      WhereScape Limited
  -- Author        :      WhereScape Developer
  -- =====
  -- Notes / History
  --

  -- =====
  -- Control variables used in most programs
  -- =====
  DECLARE v_msgtext      varchar(255); -- Text for audit_trail
  DECLARE v_sql          varchar(255); -- Text for sql statements
  DECLARE v_set          integer;     -- commit set
  DECLARE v_analyze_flag integer;     -- analyze flag
  DECLARE v_step         integer;     -- return code
  DECLARE v_update_count integer;     -- no of records updated
  DECLARE v_insert_count integer;     -- no of records inserted
  DECLARE v_count       integer;     -- General counter
  DECLARE v_sql_code    integer;     -- SQL Error Code for Audit Trail
  DECLARE v_sql_error   varchar(255); -- SQL Error Code for Audit Trail as varchar

  -- =====
  -- Variables
  -- =====
  DECLARE v_dss_update_time timestamp; -- Used for date insert
  DECLARE v_order_msg      varchar(256);
  DECLARE v_order_status   integer;
  DECLARE v_retcode        varchar(256);
  DECLARE v_invoice_msg    varchar(256);
  DECLARE v_invoice_status integer;
  DECLARE v_status         integer;

  -- =====
  -- Exceptions
  -- =====
  DECLARE EXIT HANDLER
  FOR SQLEXCEPTION
  BEGIN
    SET v_sql_code = SQLCODE;
    SELECT ErrorText
    INTO   :v_sql_error
    FROM   dbc.ErrorMsgs
    WHERE  ErrorCode = :v_sql_code;
    SET v_msgtext = 'Unhandled Exception in update_model_order_header. ' ||
      ' Step ' || CAST(v_step AS VARCHAR(64)) ||
      ' SQL Error Code: ' || CAST(v_sql_code AS VARCHAR(10)) || ' - ' || v_sql_error;
    SET p_return_msg = v_msgtext;
    CALL [METABASE].WsWrkAudit('F', :p_job_name, :p_task_name, :p_sequence
      :v_msgtext, :v_sql_code, :v_sql_error, :p_task_id, :p_job_id);
    SET p_status = -3;
  END;

```

```
-----
-- Main
-----
SET v_step = 100;
SET v_dss_update_time = CURRENT_TIMESTAMP;
SET v_update_count = 0;
SET v_insert_count = 0;

-----
-- Update model order header table with info from the orders
-- stage table. Report the results to the audit trail.
-----
CALL [METABASE].update_model_order_header_o(:p_sequence, :p_job_name, :p_task_name
, :p_job_id, :p_task_id, :v_order_msg, :v_order_status);

SET v_step = 200;

IF ( v_order_status = 1 ) THEN
    SET v_retcode = 'S';
ELSEIF ( v_order_status = -1 ) THEN
    SET v_retcode = 'W';
ELSEIF ( v_order_status = -2 ) THEN
    SET v_retcode = 'E';
ELSEIF ( v_order_status = -3 ) THEN
    SET v_retcode = 'F';
END IF;

SET v_step = 300;

CALL [METABASE].WsWrkAudit(:v_retcode, :p_job_name, :p_task_name, :p_sequence
, :v_order_msg, NULL, NULL, :p_task_id, :p_job_id);

SET v_step = 400;

-----
-- If the previous update (orders) was successful, then
-- update model order header table with info from the invoices
-- stage table. Report the results to the audit trail.
-----
IF ( v_order_status > -2 ) THEN

    CALL [METABASE].update_model_order_header_o(:p_sequence, :p_job_name, :p_task_name
, :p_job_id, :p_task_id, :v_invoice_msg, :v_invoice_status);

    SET v_step = 500;

    IF ( v_invoice_status = 1 ) THEN
        SET v_retcode = 'S';
    ELSEIF ( v_invoice_status = -1 ) THEN
        SET v_retcode = 'W';
    ELSEIF ( v_invoice_status = -2 ) THEN
        SET v_retcode = 'E';
    ELSEIF ( v_invoice_status = -3 ) THEN
        SET v_retcode = 'F';
    END IF;

    SET v_step = 600;

    CALL [METABASE].WsWrkAudit(:v_retcode, :p_job_name, :p_task_name, :p_sequence
, :v_invoice_msg, NULL, NULL, :p_task_id, :p_job_id);

    SET v_step = 700;
ELSE
    SET v_invoice_status = 0;
```

```

END IF;

-----
-- All Done report the results
-----
SET v_step = 800;

IF ( v_invoice_status < v_order_status ) THEN
    SET v_status = v_invoice_status;
ELSE
    SET v_status = v_order_status;
END IF;

SET v_msgtext = 'model_order_header update ';

IF ( v_status = 1 ) THEN
    SET v_msgtext = v_msgtext || 'completed.';
ELSEIF ( v_status = -1 ) THEN
    SET v_msgtext = v_msgtext || 'completed with warnings.';
ELSE
    SET v_msgtext = v_msgtext || 'failed.';
END IF;

SET p_status = v_status;
SET p_return_msg = v_msgtext;

END;

```

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 *Connection to the Data Warehouse* (see "*Database - Data Warehouse/Metadata Repository*" on page 135) and *Storage* (on page 182).

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.


```
-----  
-- Main  
-----  
SET v_step = 100;  
SET v_dss_update_time = CURRENT_TIMESTAMP;  
SET v_update_count = 0;  
SET v_insert_count = 0;  
  
-----  
-- Delete all existing data from the table  
-----  
DELETE FROM [TABLEOWNER].[stage_customer] ALL;  
  
SET v_step = 200;  
  
-----  
-- Insert input records into stage_customer  
-----  
INSERT INTO [TABLEOWNER].[stage_customer]  
(  
    customer_code,  
    customer_name,  
    customer_legal_name,  
    territory_id,  

```



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.

[METABASE] is used as a placeholder for the Teradata database metadata repository to enable the deployment between environments without regenerating the procedures.

```
CREATE PROCEDURE [METABASE].update_edw_customer
(
  IN p_sequence      integer,
  IN p_job_name      varchar(256),
  IN p_task_name     varchar(256),
  IN p_job_id        integer,
  IN p_task_id       integer,
  OUT p_return_msg   varchar(256),
  OUT p_status       integer
)
BEGIN
-----
-- DBMS Name       : Teradata
-- Script Name     : update_edw_customer
-- Description     : Update the EDW 3NF table edw_customer
-- Generated by    : WhereScape RED Version 6.8.4.3 (build 150908-203700 pre-release)
-- Generated for   : RED Documentation
-- Generated on    : Thursday, September 17, 2015 at 12:37:07
-- Author         : WhereScape Documentation
-----
-- Notes / History
--
-----
-- Control variables used in most programs
-----
```

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 two spaces into the text. A **shift/tab** removes two 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-click 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 two spaces. A **shift/tab** will remove two 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 using the **Comment** option in the right-click pop-up menu.

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 un commented by choosing the **Uncomment** option.

Note: Only lines that start with two dashes in the left most column can be uncommented.

Inserting Steps

The right-click pop-up menu has an option to **insert step**. This option will insert a code line of the format 'SET 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 and Detail Messages

The Alt/A key can be used to insert an extra audit message to be written to the audit log while the procedure is running. The default message can be changed as appropriate. The inserted code is:

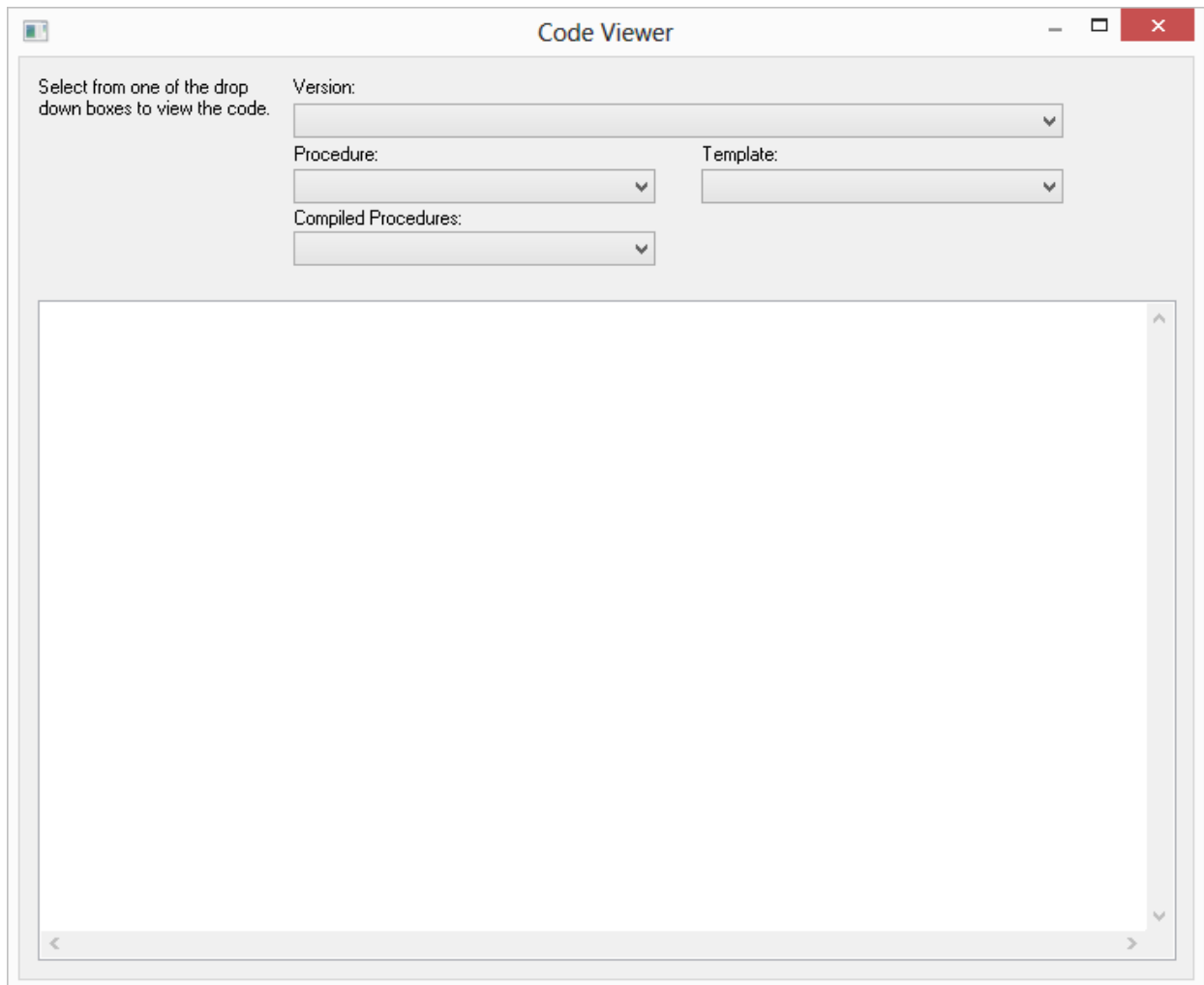
```
SET v_msgtext = ' Step ' || CAST(v_step AS VARCHAR(64))
CALL [METABASE].WsWrkAudit('I', :p_job_name, :p_task_name, :p_sequence
, :v_msgtext, '', '', :p_task_id, :p_job_id);
```

The Alt/D key can be used to insert an extra detail (or error) message to be written to the detail log while the procedure is running. The default message can be changed as appropriate. The inserted code is:

```
SET v_msgtext = ' Step ' || CAST(v_step AS VARCHAR(64))
CALL [METABASE].WsWrkError('I', :p_job_name, :p_task_name, :p_sequence
, :v_msgtext, '', '', :p_task_id, :p_job_id, NULL);
```

Viewing other procedural code

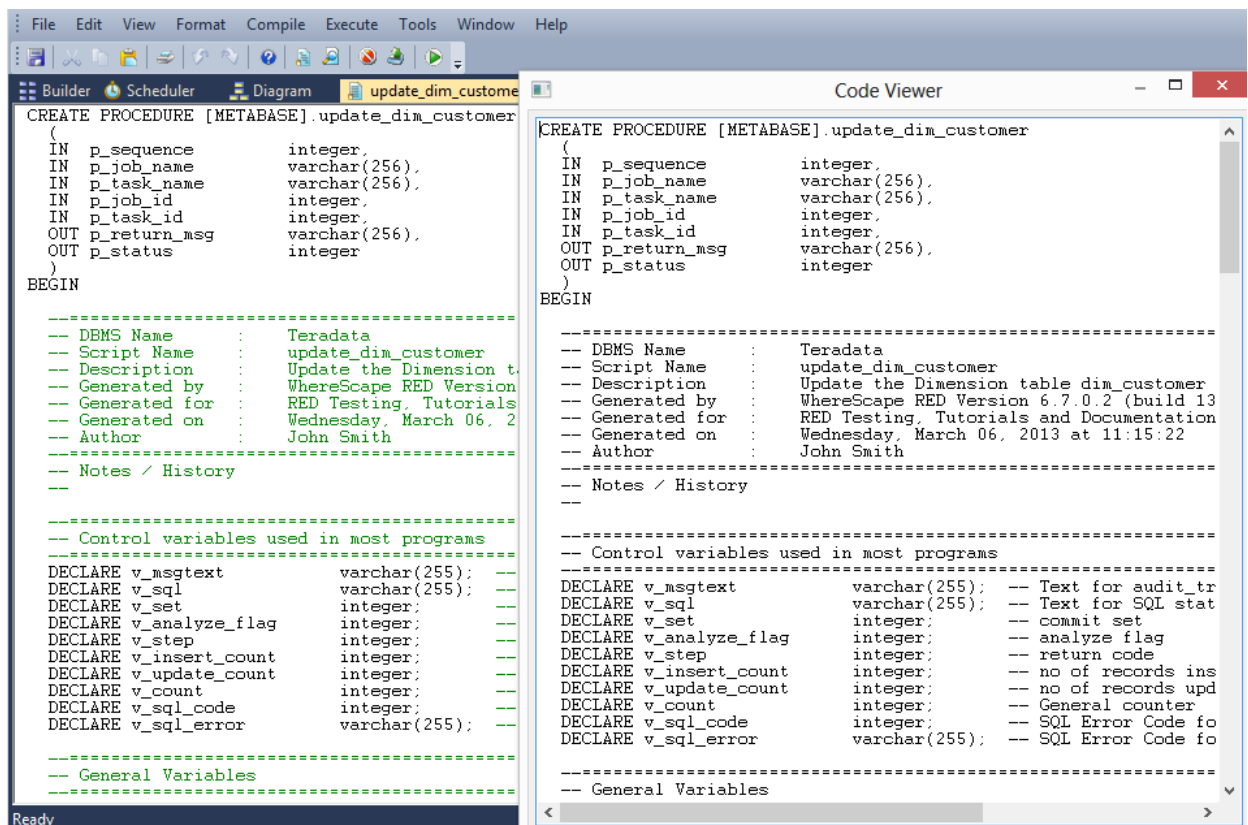
During the editing process it is possible to pop up a window containing other procedural code. This window allows cut and paste operations. In such a way it can be used as a work area or as a source of code. Select the **Tools/View Procedure or Template** menu option to bring the viewer window up. A dialog as follows will appear:



A number of drop-down lists 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 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 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 Teradata, a compiled procedure can be viewed via the Teradata Administrator by doing a Show Definition of the procedure located in the meta repository database.

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 to create a version of a procedure. These are:

- 1 By setting **Auto-Version before Procedure Compile** under **Tools/Options/Versioning**. If set 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 when 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)

Version Definition

Create version for Procedure update_dim_product

Version Name or Short Description:

Detailed Description:

Retain Until: Tuesday , November 18, 2025

OK Cancel

- 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 at this stage.

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:

```
-->-- set v_step = 200;
| SELECT COUNT(*) --<<--
  INTO   :v_count --<<--
  FROM   [model_customer] model_customer --<<--
  WHERE  dss_update_time >= CURRENT_TIMESTAMP - INTERVAL '7' DAY; --<<--
```

In this example the line SET v_step = 100; has been removed from the current code in the edit window and the remaining three lines have been inserted.

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 'SET v_step = 100;' would be added and the following three lines deleted.

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--.

```

-----
-- Exceptions
-----
DECLARE EXIT HANDLER
FOR SQLEXCEPTION
BEGIN
    SET v_sql_code = SQLCODE;
    SELECT SUBSTR(ErrorText,1,255)
    INTO   v_sql_error
    FROM   dbc_ErrorMsgs
    WHERE  ErrorCode = v_sql_code;
    SET v_msgtext = SUBSTR('Unhandled Exception in model_customer. ' ||
        ' Step ' || CAST(v_step AS VARCHAR(64)) ||
        ' SQL Error Code: ' || CAST(v_sql_code AS VARCHAR(10)) || ' - ' || v_sql_error,1,255);
    SET p_return_msg = v_msgtext;
    CALL [METABASE].WsWrkAudit('F', :p_job_name, :p_task_name, :p_sequence
        , :v_msgtext, :v_sql_code, :v_sql_error, :p_task_id, :p_job_id);
    SET p_status = -3;
END;

-----
-- Main
-----
SET v_step = 100;
SET v_update_count = 0;
SET v_insert_count = 0;

SELECT COUNT(*)
INTO   v_count
FROM   [model_customer] model_customer
|--W--line 72-- SPL5000:W(L72), E(5628):Column dss_update_date not found in Ws1Warehouse.model_customer.
WHERE  dss_update_date >= CURRENT_TIMESTAMP - INTERVAL '7' DAY;

```

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.

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 or Output | Data Type |
|----------------|-----------------|--------------|
| p_sequence | Input | Integer |
| p_job_name | Input | Varchar(256) |
| p_task_name | Input | Varchar(256) |
| p_job_id | Input | Integer |
| p_task_id | Input | Integer |
| p_return_msg | Output | Varchar(256) |
| p_status | Output | 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.

The screenshot shows a dialog box titled "Procedure update_dim_product". On the left is a "Properties" tab and a "Notes" section. The main area contains the following fields and controls:

- Name:** update_dim_product
- Type:** Procedure (dropdown)
- Purpose:** update_dim_product (text area)
- Owner:** auto created
- Delete Lock:** (unchecked)
- Last Update By:** (empty text field)
- Default Connect:** DataWarehouse (dropdown)
- Edit Lock:**
 - Locked For Edit By:** WhereScape Documentation (highlighted in red)
 - Release Edit Lock:** (button)
 - Edit Lock Reason or Last Update:** new procedure (text area)
- Timestamps:**
 - Created:** 2017-03-27 02:53:52
 - Last Update:** (empty text field)
 - Compiled:** 2017-03-27 02:53:57

At the bottom right are buttons for "OK", "Cancel", and "Help".

In the example above, the **Delete Lock** check box is not selected. Selecting this check box prevents the procedure from being deleted through the **Delete** menu option. It also prevents the procedure from being overwritten, if a new procedure generation is requested.

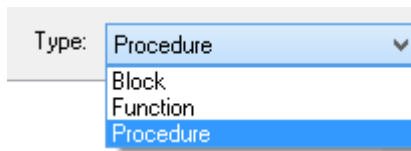
Also in the example above, the procedure is currently being edited and is shown as being Locked for Edit by "WhereScape Documentation". If procedures or scripts have already been opened for editing, they can only subsequently be opened for viewing.

Edit locks 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.

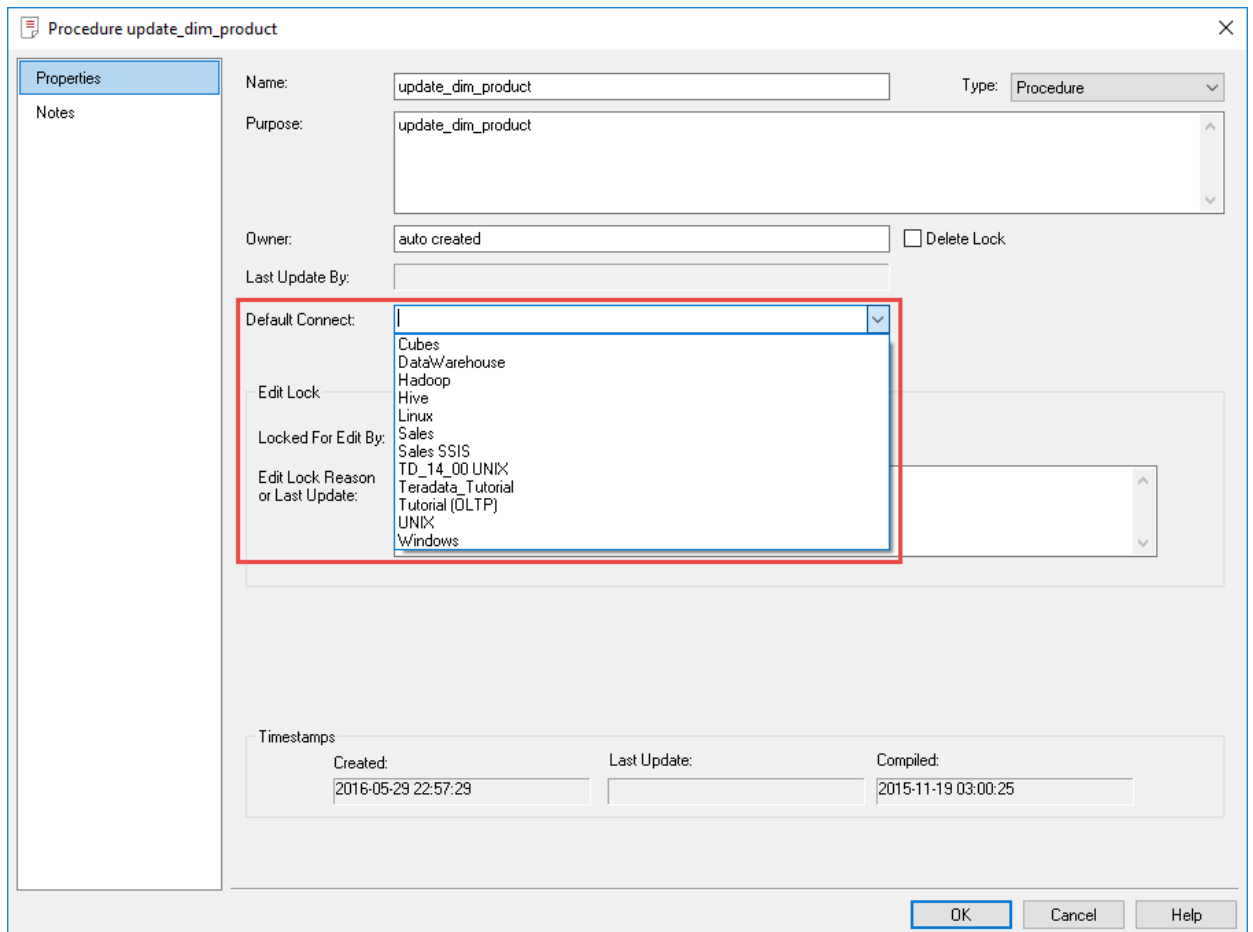
The **Edit Lock Reason** is for information only, and can be used as another comment field if desired.

The **Type** drop 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.



MACROS

WhereScape RED can also retrofit, run, schedule and generate Teradata macros.

Template **macros** can be generated for moving data into simple staging tables without surrogate keys. A macro is generated by selecting the ***** Create New Macro ***** option from a drop-down box that is used to display the update procedure in a staging table's properties.

A utility is available from WhereScape to mass retrofit large numbers of existing Teradata macros into the WhereScape RED metadata.

BTEQ SCRIPTS

WhereScape RED can also retrofit, run, schedule and generate Teradata BTEQ scripts.

Template **bteq scripts** can be generated for moving data into simple staging tables without surrogate keys. A BTEQ script is generated by selecting the ***** Create New Bteq ***** option from a drop-down box that is used to display the update procedure in a staging table's properties.

SCRIPT GENERATION

WhereScape RED generates template scripts to assist in the loading of textual files from Windows. These scripts are generated when 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: Windows script supports both DOS Batch and PowerShell scripts *for more information* (see "24.11.1.1 Windows PowerShell Scripts" on page 657).

SCRIPT GENERATION (WINDOWS/TERADATA)

A sample Windows script for Teradata is as follows. The key components of the script are described below:

```
@echo off
setlocal enabledelayedexpansion
setlocal enableextensions
REM *****
REM ***** DBMS Name      : Teradata
REM ***** Description   : File Load/Movement for C:\temp\forecast.txt into load_forecast
REM ***** Generated by  : WhereScape RED
REM ***** Generated for  : WhereScape Limited
REM ***** Author       : WhereScape Developer
REM *****
REM ***** Notes / History
REM *****
REM *****
REM ***** NOTE: The following environment variables will be set
REM ***** WSL_LOAD_TABLE = the load table name
REM ***** WSL_LOAD_DB     = the load table names database
REM ***** WSL_TEMP_DB     = the load table names database for temporary tables
REM ***** WSL_SEQUENCE   = a unique sequence number for the scheduler
REM ***** WSL_WORKDIR    = the work directory defined in the connection
REM ***** WSL_SCHEDILOG  = the directory and file name of the scheduler
REM ***** log file which is used to defined the parameter batch file
REM ***** when this is scheduled
REM ***** WSL_SERVER     = the server defined in the connection
REM ***** WSL_DATABASE   = the database defined in the connection
REM ***** WSL_METABASE   = the database storing the meta data
REM ***** WSL_USER       = the dss user defined in the connection
REM ***** WSL_PWD        = the dss password defined in the connection
REM ***** WSL_REMOTEHOSTn = the nth remote server for sending files to
REM ***** WSL_REMOTEUSERn = the username for the nth remote server
REM ***** WSL_PARAMnmn   = 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 *****
if defined WSL_LOAD_TABLE (SET LOAD_TABLE=%WSL_LOAD_TABLE%) else SET LOAD_TABLE=load_forecast
if defined WSL_LOAD_DB (SET LOAD_DB=%WSL_LOAD_DB%) else SET LOAD_DB=WslWarehouse
if defined WSL_TEMP_DB (SET TEMP_DB=%WSL_TEMP_DB%) else SET TEMP_DB=WslWarehouse
if defined WSL_SERVER (SET SERVER=%WSL_SERVER%) else SET SERVER=
if defined WSL_DATABASE (SET DATABASE=%WSL_DATABASE%) else SET DATABASE=
if defined WSL_METABASE (SET METABASE=%WSL_METABASE%) else SET METABASE=WslWarehouse.
if defined WSL_USER (SET USER=%WSL_USER%) else SET USER=WslWarehouse
if defined WSL_PWD (SET PWD=%WSL_PWD%) else SET PWD=wsl
if defined WSL_WORKDIR (SET WORKDIR=%WSL_WORKDIR%) else goto LABEL_FILE_DEFAULT
if not defined WSL_SCHEDILOG (SET PARAMBAT="%WORKDIR%\parameters.bat") else (SET PARAMBAT="%WSL_SCHEDILOG:log=bat%")
if defined WSL_SEQUENCE (SET SEQUENCE=%WSL_SEQUENCE%) else goto LABEL_FILE_DEFAULT
SET FILECTL=%WORKDIR%\wsl_%LOAD_TABLE%%SEQUENCE%.ctl
SET FILEAUD=%WORKDIR%\wsl_%LOAD_TABLE%%SEQUENCE%.aud
goto LABEL_WAIT
:LABEL_FILE_DEFAULT
SET FILECTL=wsl_load_forecast100002.ctl
SET FILEAUD=wsl_load_forecast100002.aud
SET SEQUENCE=100002
SET PARAMBAT=parameters.bat
:LABEL_WAIT
ECHO Starting > %FILEAUD%
IF EXIST %PARAMBAT% (CALL %PARAMBAT%) ELSE (ECHO Parameters batch file does not exist >> %FILEAUD%)
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.
REM
SET LOAD_FILE=C:\temp\forecast.txt
REM
REM Remove Existing Parameters
echo DELETE FROM %METABASE%\dss_parameter WHERE dss_parameter_name like '____' | bteq .Logon %DATABASE%\%USER%.%PWD% >> %FILEAUD%
```



```

REM
REM *****
REM ***** W A I T   T I M E R *****
REM *****
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 "%LOAD_FILE%" goto LABEL_FILECHECK
REM Call %$!sleep to wait for 30 seconds. It will return remaining seconds
"C:\Program Files\WhereScape\%$!sleep" %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 "%LOAD_FILE%" goto LABEL_FOUND
echo -1
echo File "%LOAD_FILE%" was not found.
exit
REM
REM *****
REM ***** L O A D   T H E   D A T A *****
REM *****
:LABEL_FOUND
REM
echo Load of "%LOAD_FILE%" >> %FILEAUD%
SET /A RESULT_CODE=1
SET RESULT_MSG=Load Completed Normally.
:LABEL_LOAD
REM
REM *****
REM ***** L O A D I N G *****
REM *****
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
SET FIRST_CHAR=%FILE_NAME:~0,1%
if ^%FIRST_CHAR%^=="^" SET FILE_NAME=%FILE_NAME:~1,-1%
FOR /F "usebackq delims==" %%i IN ("%FILE_NAME%") do SET FILE_SHORT=%%~nxi
echo Loading %FILE_NAME% >> %FILEAUD%
echo LOGON %DATABASE%\%USER%,%PWD% >> %FILECTL%
echo DROP TABLE %TEMP_DB%.load_forecast_e1 >> %FILECTL%
echo DROP TABLE %TEMP_DB%.load_forecast_e2 >> %FILECTL%
echo SET RECORD VARTEXT " " >> %FILECTL%
echo RECORD 2 >> %FILECTL%
echo DEFINE >> %FILECTL%
echo   product_code (varchar(50)) >> %FILECTL%
echo   customer_code (varchar(50)) >> %FILECTL%
echo   forecast_quantity (varchar(50)) >> %FILECTL%
echo   forecast_sales_value (varchar(50)) >> %FILECTL%
echo   forecast_date (varchar(50)) >> %FILECTL%
echo file=%FILE_NAME% >> %FILECTL%
echo SHOW >> %FILECTL%
echo BEGIN LOADING %LOAD_DB%.load_forecast ERRORFILES %TEMP_DB%.load_forecast_e1. %TEMP_DB%.load_forecast_e2 >> %FILECTL%
echo INSERT INTO %LOAD_DB%.load_forecast >> %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 END LOADING; >> %FILECTL%
echo LOGOFF; >> %FILECTL%
fastload < %FILECTL% >> %FILEAUD%
SET ERRLEV=%errorlevel%
IF %ERRLEV% EQU 0 GOTO LABEL_OKAY
IF %ERRLEV% LEQ 4 GOTO LABEL_WARNING
:LABEL_FAIL
echo -3
IF %ERRLEV% EQU 8 echo Load Failed. A user error occurred in the loader.
IF %ERRLEV% EQU 12 echo Load Failed. A fatal error occurred in the loader.
IF %ERRLEV% EQU 16 echo Load Failed. No message destination available from the loader.
type %FILEAUD% >&2
:ERR_EXIT
exit
:LABEL_WARNING
echo -1
echo Load Completed Normally, with warnings from the loader.
:LABEL_OKAY
REM
REM *****
REM ***** R E N A M E *****
REM *****
:LABEL_RENAME
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 *****
REM ***** E X I T *****
REM *****
:LABEL_EXIT
echo %RESULT_CODE%
echo %RESULT_MSG%
exit

```



WhereScape RED TIP: Parameters

Parameters can also be added to scripts to facilitate deployment processes or environment changes without the need to regenerate scripts. They 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 a Parameter to the script, create the desired parameter(s) in **Tools->Parameters-> Add Parameter**.

X

Parameter Maintenance

Parameter:

Value:

Comments:

```

SET LOAD_FILE="$PSourcePath$\forecast.txt"
SET TRIG_FILE="$PSourcePath$\forecast.trg"

```

The script makes use of a number of environmental variables. These variable are acquired from both the table and connection properties. These variables are established in the environment by either RED or the scheduler. If the script is to be executed outside of 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 *****
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_FOUND
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 ***** T R I G G E R *****
REM
REM First clear any existing parameters
REM
:LABEL_TRIGGER
SET TRIG_NAME=NO_MORE_FILES
for %%A in ("%TRIG_FILE%") do if "!TRIG_NAME!"=="NO_MORE_FILES" SET TRIG_NAME=%%A
if "!TRIG_NAME!"=="NO_MORE_FILES" goto LABEL_TRIG_EXIT
FOR /F "usebackq delims==" %%i IN ("%TRIG_NAME%") do SET TRIG_SHORT=%%~nxi
REM
REM Get the first four and all remaining parameters
REM
for /F "tokens=1,2,3,4,5,6,7,8,9* usebackq delims=^|" %%a in (%TRIG_NAME%) do SET A=%%a&SET B=%%b&SET C=%%c&SET D=%%d&SET E=%%e&SET F=%%f&SET G=%%g&SET H=
REM
REM For each parameter write it to the parameter table
REM
if defined A echo call %METABASE%\ParameterWrite('FORECAST_0', '%A%', 'script load'); | bteq .Logon %DATABASE%\%USER%, %PWD% >> %FILEAUD%
if defined B echo call %METABASE%\ParameterWrite('FORECAST_1', '%B%', 'script load'); | bteq .Logon %DATABASE%\%USER%, %PWD% >> %FILEAUD%
if defined C echo call %METABASE%\ParameterWrite('FORECAST_2', '%C%', 'script load'); | bteq .Logon %DATABASE%\%USER%, %PWD% >> %FILEAUD%
if defined D echo call %METABASE%\ParameterWrite('FORECAST_3', '%D%', 'script load'); | bteq .Logon %DATABASE%\%USER%, %PWD% >> %FILEAUD%
if defined E echo call %METABASE%\ParameterWrite('FORECAST_4', '%E%', 'script load'); | bteq .Logon %DATABASE%\%USER%, %PWD% >> %FILEAUD%
if defined F echo call %METABASE%\ParameterWrite('FORECAST_5', '%F%', 'script load'); | bteq .Logon %DATABASE%\%USER%, %PWD% >> %FILEAUD%
if defined G echo call %METABASE%\ParameterWrite('FORECAST_6', '%G%', 'script load'); | bteq .Logon %DATABASE%\%USER%, %PWD% >> %FILEAUD%
if defined H echo call %METABASE%\ParameterWrite('FORECAST_7', '%H%', 'script load'); | bteq .Logon %DATABASE%\%USER%, %PWD% >> %FILEAUD%
if defined I echo call %METABASE%\ParameterWrite('FORECAST_8', '%I%', 'script load'); | bteq .Logon %DATABASE%\%USER%, %PWD% >> %FILEAUD%
if defined J echo call %METABASE%\ParameterWrite('FORECAST_9', '%J%', 'script load'); | bteq .Logon %DATABASE%\%USER%, %PWD% >> %FILEAUD%
:LABEL_TRIG_EXIT
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 okay 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 I N G *****
REM
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
SET FIRST_CHAR=%FILE_NAME:~0,1%
if ^"%FIRST_CHAR%"==" SET FILE_NAME=%FILE_NAME:~1,-1%
FOR /F "usebackq delims==" %%i IN ("%FILE_NAME%") do SET FILE_SHORT=%%~nxi
echo Loading %FILE_NAME% >> %FILEAUD%
echo LOGON %DATABASE%\%USER%, %PWD% >> %FILECTL%
echo DROP TABLE %TEMP_DB%.load_forecast_e1; >> %FILECTL%
echo DROP TABLE %TEMP_DB%.load_forecast_e2; >> %FILECTL%
echo SET RECORD VARTEXT " "; >> %FILECTL%
echo RECORD 2; >> %FILECTL%
echo DEFINE >> %FILECTL%
echo product_code (varchar(50)) >> %FILECTL%
echo customer_code (varchar(50)) >> %FILECTL%
echo forecast_quantity (varchar(50)) >> %FILECTL%
echo forecast_sales_value (varchar(50)) >> %FILECTL%
echo forecast_date (varchar(50)) >> %FILECTL%
echo file=%FILE_NAME%; >> %FILECTL%
echo SHOW; >> %FILECTL%
echo BEGIN LOADING %LOAD_DB%.load_forecast ERRORFILES %TEMP_DB%.load_forecast_e1, %TEMP_DB%.load_forecast_e2; >> %FILECTL%
echo INSERT INTO %LOAD_DB%.load_forecast >> %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 VALUES >> %FILECTL%
echo ( :product_code >> %FILECTL%
echo :customer_code >> %FILECTL%
echo :forecast_quantity >> %FILECTL%
echo :forecast_sales_value >> %FILECTL%
echo :forecast_date ( FORMAT 'DD-MM-YYYY' ) >> %FILECTL%
echo ) >> %FILECTL%
echo END LOADING; >> %FILECTL%
echo LOGOFF; >> %FILECTL%
fastload < %FILECTL% >> %FILEAUD%
SET ERRLEV=%errorlevel%

```

This section calls Fastload, Multi-Load or TPT to load the file. It makes use of a temporary file to build a control file and then calls Fastload, Multi-Load or TPT 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.



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 DELETE FROM %LOAD_DB%.%LOAD_TABLE% ALL; | bteq .Logon
%DATABASE%/%USER%,%PWD% >> %FILEAUD%
```

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 *****
REM ***** R E N A M E *****
REM *****
:LABEL_RENAME
SET YYYY=%DATE:~4%
SET MM=%DATE:~10,2%
SET DD=%DATE:~7,2%
SET HOUR1=%TIME:~0,1%X
IF %HOUR1% EQU %X X% GOTO LABEL_LEAD_ZERO
SET HH=%TIME:~0,2%
GOTO LABEL_LEAD_END
:LABEL_LEAD_ZERO
SET HH=0%TIME:~1,1%
:LABEL_LEAD_END
SET MI=%TIME:~3,2%
SET SS=%TIME:~6,2%
REM rename the file
move "%FILE_NAME%" "C:\temp\loaded\forecast_%YYYY%%MM%%DD%%HH%%MI%%SS%.txt" >> %FILEAUD%
SET ERRLEV=%errorlevel%
IF %ERRLEV% EQU 0 GOTO LABEL_MOVE3
echo -3
echo %FILE_NAME% NOT MOVED >> %FILEAUD%
SET MSG=The file %FILE_NAME% has NOT been moved to C:\temp\loaded\forecast_%YYYY%%MM%%DD%%HH%%MI%%SS%.txt
echo call %METABASE%$WrkAuditL('I', '%LOAD_TABLE%', '%SEQUENCE%', '%MSG%', NULL, NULL); | bteq .Logon %DATABASE%/%USER%,%PWD% >> %FILEAUD%
exit
GOTO LABEL_AFTER_MOVE3
:LABEL_MOVE3
echo %FILE_NAME% moved to C:\temp\loaded\forecast_%YYYY%%MM%%DD%%HH%%MI%%SS%.txt >> %FILEAUD%
SET MSG=The file %FILE_NAME% has been moved to C:\temp\loaded\forecast_%YYYY%%MM%%DD%%HH%%MI%%SS%.txt
echo call %METABASE%$WrkAuditL('I', '%LOAD_TABLE%', '%SEQUENCE%', '%MSG%', NULL, NULL); | bteq .Logon %DATABASE%/%USER%,%PWD% >> %FILEAUD%
:LABEL_AFTER_MOVE3
REM rename the trigger file
move %TRIG_NAME% "C:\temp\loaded\forecast_%YYYY%%MM%%DD%%HH%%MI%%SS%.trg" >> %FILEAUD%
SET ERRLEV=%errorlevel%
IF %ERRLEV% EQU 0 GOTO LABEL_MOVE4
echo -3
echo %TRIG_NAME% NOT MOVED >> %FILEAUD%
SET MSG=The file %TRIG_NAME% has NOT been moved to C:\temp\loaded\forecast_%YYYY%%MM%%DD%%HH%%MI%%SS%.trg
echo call %METABASE%$WrkAuditL('I', '%LOAD_TABLE%', '%SEQUENCE%', '%MSG%', NULL, NULL); | bteq .Logon %DATABASE%/%USER%,%PWD% >> %FILEAUD%
exit
GOTO LABEL_AFTER_MOVE4
:LABEL_MOVE4
echo %TRIG_NAME% moved to C:\temp\loaded\forecast_%YYYY%%MM%%DD%%HH%%MI%%SS%.trg >> %FILEAUD%
SET MSG=The file %TRIG_NAME% has been moved to C:\temp\loaded\forecast_%YYYY%%MM%%DD%%HH%%MI%%SS%.trg
echo call %METABASE%$WrkAuditL('I', '%LOAD_TABLE%', '%SEQUENCE%', '%MSG%', NULL, NULL); | bteq .Logon %DATABASE%/%USER%,%PWD% >> %FILEAUD%
:LABEL_AFTER_MOVE4
REM
REM
IF %ERRLEV% EQU 0 GOTO LABEL_OKAY
IF %ERRLEV% LEQ 4 GOTO LABEL_WARNING
:LABEL_FAIL
echo -3
IF %ERRLEV% EQU 8 echo Load Failed. A user error occurred in the loader.
IF %ERRLEV% EQU 12 echo Load Failed. A fatal error occurred in the loader.
IF %ERRLEV% EQU 16 echo Load Failed. No message destination available from the loader.
type %FILEAUD% >&2
:ERR_EXIT
exit
:LABEL_WARNING
echo -1
echo Load Completed Normally, with warnings from the loader.
:LABEL_OKAY
REM
```

24.11.1.1 WINDOWS POWERSHELL SCRIPTS

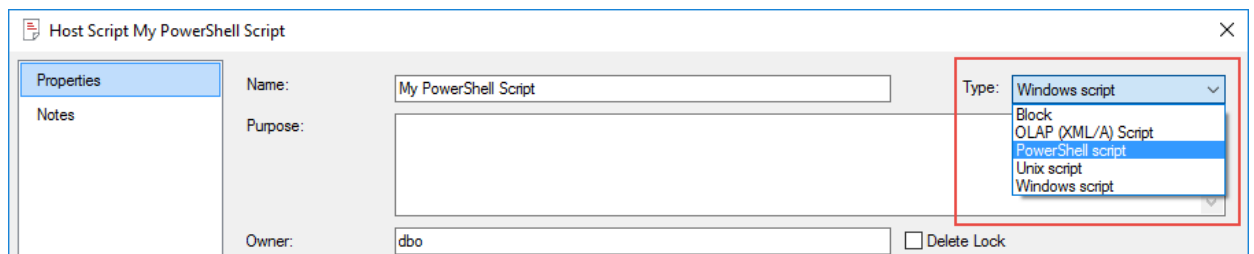
In addition to the conventional Windows scripting and other tools, WhereScape RED also supports Windows PowerShell scripts for loading data into a WhereScape RED managed Data Warehouse, as well as for exporting data from a WhereScape RED managed Data Warehouse.

The Windows PowerShell command line and scripting environment was introduced by Microsoft in Windows 7. For more information about PowerShell, please refer to the Microsoft TechNet website (<https://technet.microsoft.com/en-us/library/dd742419.aspx>).

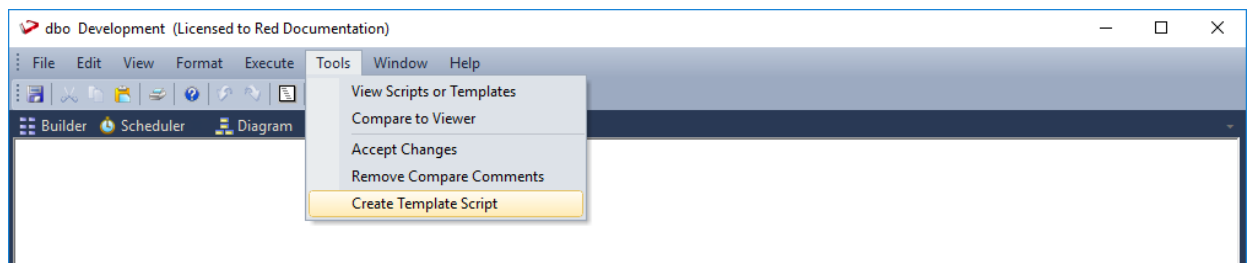
Note that WhereScape RED does not automatically create the PowerShell scripts, you must write them either directly via the WhereScape RED script editor or via a template that generates a PowerShell script. Each method is described below:

Via the WhereScape RED Script Editor

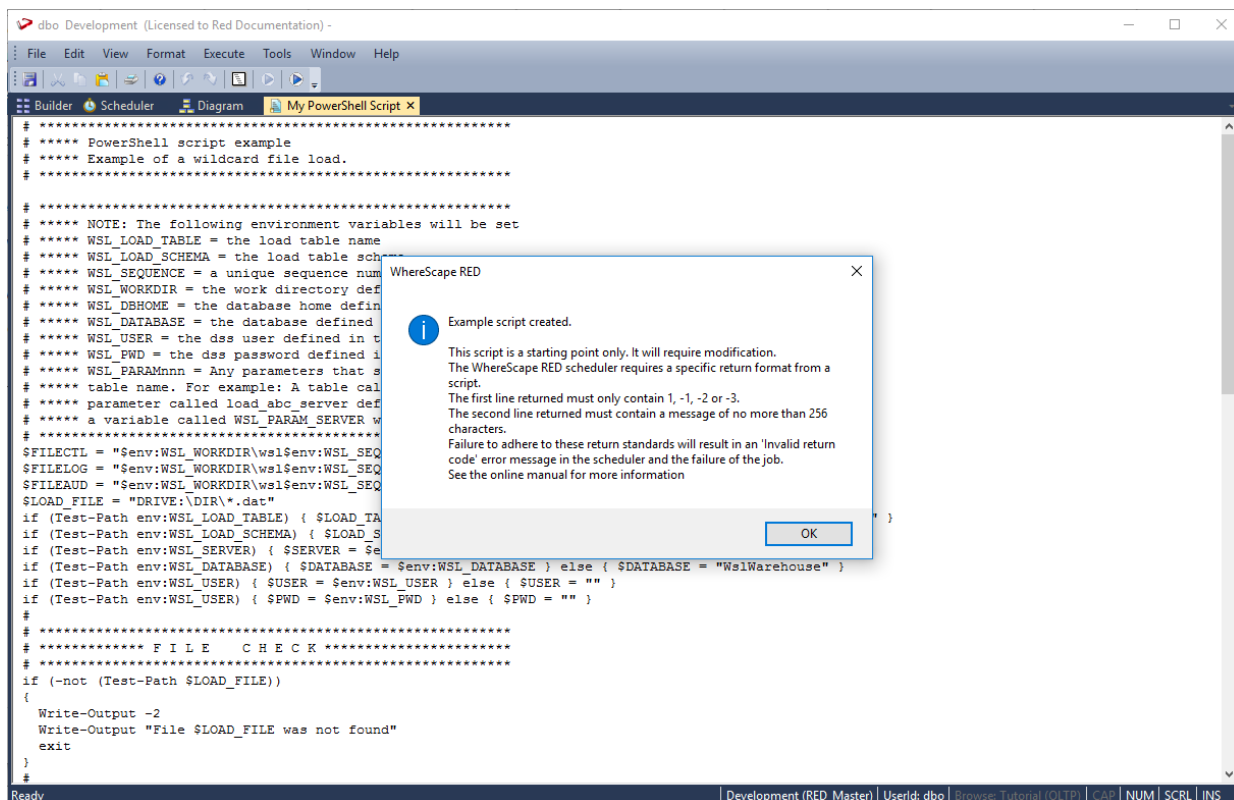
You can manually write the PowerShell script from scratch, using the Script Editor. You need to create a Host Script object and set the script type to PowerShell script from the **Host Script Properties** screen.



You can then open the blank Host Script created of type PowerShell Script to write your own script, or you can use the sample PowerShell scripts in WhereScape RED, which can be generated by selecting **Tools > Create Template Script** from the Script Editor screen.



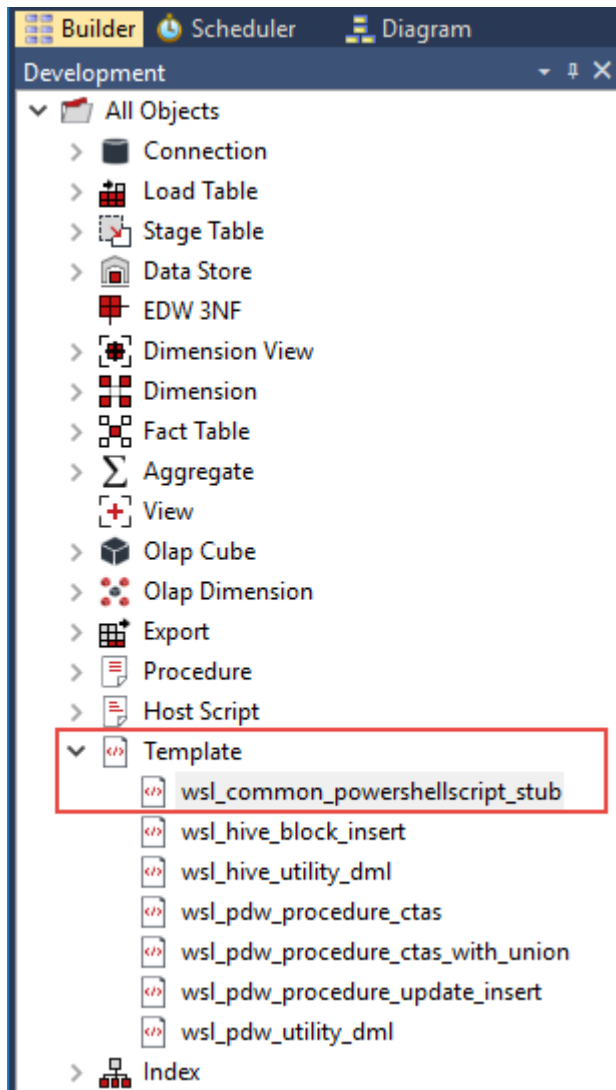
The contents generated is an actual PowerShell script syntax that you can use to load data into a WhereScape RED managed Data Warehouse. You can also modify it and use it as a starting point to write your own PowerShell script for Load or Export objects.



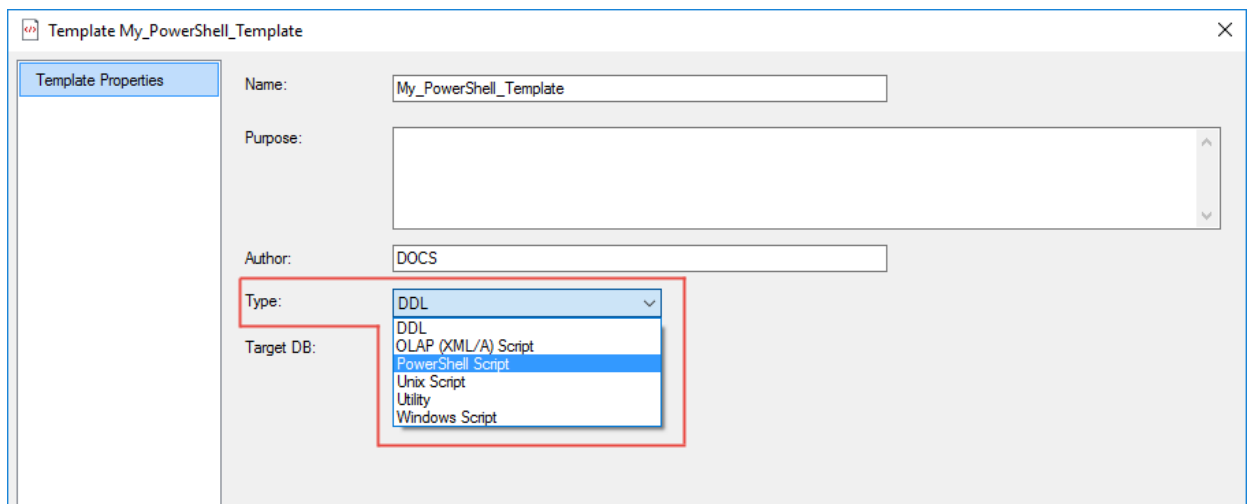
Via the Stub Template

WhereScape RED also provides the Template object `wsl_common_powershellscript_stub` which is a basic PowerShell script type stub template that serves as guide on the use of a template to generate a PowerShell script.

Additional PowerShell Templates can be downloaded from the WhereScape website (<https://www.wherescape.com/support/software-downloads-documentation/templates/>).



You can also create your own Template object and select PowerShell Script under the **Type** drop-down of the **Template Properties** screen.



Similar to the WhereScape RED Script Editor method described above, you can edit the template created to write your own script or open an existing PowerShell script to copy its contents to your template.

Note:

WhereScape RED uses the 32-bit version of PowerShell by default. To use the 64-bit version, the following code must be added as the first line of the script. The script is relaunched and will be run in 64-bit version.

```
if ($psHOME -like "*syswow64*") {  
    & (join-path ($psHOME -replace "syswow64", "sysnative") powershell.exe) -  
file (join-path $psScriptroot $myInvocation.MyCommand) @args  
    exit  
}
```

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-click 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.

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 lists 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 can not 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.

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.

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.

Example:

In the following example the first line '@echo off' prevents unwanted information from being reported to standard out. A Multi-Load script file is built up (echo statements). The mload command is then executed to run the load.

```
@echo off
echo Loading c:\temp\budget.txt >> wsl_load_budget100001.aud
echo .Logtable WslWarehouse.load_budget_e1; > wsl_load_budget100001.ctf
echo .Logon DemoTDAT/WslWarehouse,wsl; >> wsl_load_budget100001.ctf
echo DROP TABLE WslWarehouse.ET_load_budget; >> wsl_load_budget100001.ctf
echo DROP TABLE WslWarehouse.UV_load_budget; >> wsl_load_budget100001.ctf
echo DROP TABLE WslWarehouse.WT_load_budget; >> wsl_load_budget100001.ctf
echo .Begin Import Mload tables WslWarehouse.load_budget >> wsl_load_budget100001.ctf
echo   Worktables WslWarehouse.WT_load_budget >> wsl_load_budget100001.ctf
echo   Errortables WslWarehouse.ET_load_budget >> wsl_load_budget100001.ctf
echo           WslWarehouse.UV_load_budget; >> wsl_load_budget100001.ctf
echo .Layout Transaction; >> wsl_load_budget100001.ctf
echo   .Field product_code * VARCHAR(50); >> wsl_load_budget100001.ctf
echo   .Field customer_code * VARCHAR(33); >> wsl_load_budget100001.ctf
echo   .Field budget_quantity * VARCHAR(50); >> wsl_load_budget100001.ctf
echo   .Field budget_sales_value * VARCHAR(50); >> wsl_load_budget100001.ctf
echo   .Field budget_date * VARCHAR(50); >> wsl_load_budget100001.ctf
echo .DML Label Inserts; >> wsl_load_budget100001.ctf
echo   INSERT INTO WslWarehouse.load_budget >> wsl_load_budget100001.ctf
echo   ( product_code>> wsl_load_budget100001.ctf
echo   , customer_code>> wsl_load_budget100001.ctf
echo   , budget_quantity>> wsl_load_budget100001.ctf
echo   , budget_sales_value>> wsl_load_budget100001.ctf
echo   , budget_date>> wsl_load_budget100001.ctf
echo   ) >> wsl_load_budget100001.ctf
echo   VALUES >> wsl_load_budget100001.ctf
echo   (:product_code>> wsl_load_budget100001.ctf
echo   ,:customer_code>> wsl_load_budget100001.ctf
echo   ,:budget_quantity>> wsl_load_budget100001.ctf
```

```
echo , :budget_sales_value>> wsl_load_budget100001.ctf
echo , :budget_date ( FORMAT 'DD-MMM-YYYY') >> wsl_load_budget100001.ctf
echo ); >> wsl_load_budget100001.ctf
echo .Import Infile 'c:\temp\budget.txt' >> wsl_load_budget100001.ctf
echo From 2 >> wsl_load_budget100001.ctf
echo Format Vartext '^,' >> wsl_load_budget100001.ctf
echo Layout Transaction >> wsl_load_budget100001.ctf
echo Apply Inserts; >> wsl_load_budget100001.ctf
echo .End Mload; >> wsl_load_budget100001.ctf
echo .Logoff; >> wsl_load_budget100001.ctf
mload -b < wsl_load_budget100001.ctf >> wsl_load_budget100001.aud
SET ERRLEV=%errorlevel%
IF %ERRLEV% EQU 0 GOTO LABEL_OKAY
IF %ERRLEV% LEQ 4 GOTO LABEL_WARNING
:LABEL_FAIL
echo -3
IF %ERRLEV% EQU 8 echo Load Failed. A user error occurred in the loader.
IF %ERRLEV% EQU 12 echo Load Failed. A fatal error occurred in the loader.
IF %ERRLEV% EQU 16 echo Load Failed. No message destination available from the loader.
type wsl_load_budget100001.aud >&2
:ERR_EXIT
exit
:LABEL_WARNING
echo -1
echo Load Completed Normally, with warnings from the loader.
:LABEL_OKAY
echo 1
echo Load Completed Normally.
type wsl_load_budget100001.aud
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:

| Windows variable | UNIX/Linux variable | Description | | | | | | | | | |
|-------------------|------------------------------|--|----|------------------------------|-----------------------------|---------|--------------------------|----------------------------|------------|---------------------|-----------------------|
| WSL_LOAD_FULLNAME | LOAD_FULLNAME | The fully-qualified load table name. | | | | | | | | | |
| WSL_LOAD_TABLE | LOAD_TABLE | The unqualified load table name. | | | | | | | | | |
| WSL_LOAD_SCHEMA | LOAD_SCHEMA | <p>The schema for the load table.</p> <p>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.</p> <p>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:</p> <table border="1" data-bbox="715 1198 1433 1482"> <thead> <tr> <th>OS</th> <th>If no trailing dot is wanted</th> <th>If a trailing dot is wanted</th> </tr> </thead> <tbody> <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> </tbody> </table> | OS | If no trailing dot is wanted | If a trailing dot is wanted | Windows | !WSL_LOAD_SCHEMA : . = ! | !WSL_LOAD_SCHEMA : . = ! . | UNIX/Linux | \${LOAD_SCHEMA% . } | \${LOAD_SCHEMA% . } . |
| OS | If no trailing dot is wanted | If a trailing dot is wanted | | | | | | | | | |
| Windows | !WSL_LOAD_SCHEMA : . = ! | !WSL_LOAD_SCHEMA : . = ! . | | | | | | | | | |
| UNIX/Linux | \${LOAD_SCHEMA% . } | \${LOAD_SCHEMA% . } . | | | | | | | | | |
| WSL_LOAD_DB | LOAD_DB | The name of the database for the load table. | | | | | | | | | |
| WSL_TEMP_DB | TEMP_DB | <p>Teradata: The name of the database for load temporary tables.</p> <p>PDW: The name of the staging database for the load.</p> <p>Others: Not Used.</p> | | | | | | | | | |
| WSL_TGT_DSN | TGT_DSN | The ODBC data source name (DSN) for the load table's storage connection. | | | | | | | | | |
| WSL_TGT_SERVER | TGT_SERVER | The server for the load table's storage connection. | | | | | | | | | |
| WSL_TGT_DBPORT | TGT_DBPORT | The database port for the load table's storage connection. | | | | | | | | | |
| WSL_TGT_DBID | TGT_DBID | The <i>Database ID</i> property of the load table's storage | | | | | | | | | |

| Windows variable | UNIX/Linux variable | Description |
|------------------|---------------------|---|
| | | connection. 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 <i>Database ID</i> 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 | <p>The Source Schema property of the load.</p> <p>Note: The property is fetched without modification, so there may or may not be a trailing dot depending on how it is configured.</p> <p>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:</p> <table border="1" data-bbox="719 600 1433 891"> <thead> <tr> <th data-bbox="719 600 858 703">OS</th> <th data-bbox="865 600 1145 703">If no trailing dot is wanted</th> <th data-bbox="1152 600 1433 703">If a trailing dot is wanted</th> </tr> </thead> <tbody> <tr> <td data-bbox="719 712 858 792">Windows</td> <td data-bbox="865 712 1145 792">!WSL_SRC_SCHEMA:. =!</td> <td data-bbox="1152 712 1433 792">!WSL_SRC_SCHEMA:. =!.</td> </tr> <tr> <td data-bbox="719 801 858 891">UNIX/ Linux</td> <td data-bbox="865 801 1145 891">\${SRC_SCHEMA%.}</td> <td data-bbox="1152 801 1433 891">\${SRC_SCHEMA%.}.</td> </tr> </tbody> </table> | 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%.}. |
|------------------------|------------------------------|---|----|------------------------------|-----------------------------|----------------|-------------------------|--------------------------|------------------------|------------------|-------------------|
| 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%.}. | | | | | | | | | |
| WSL_SRC_USER | SRC_USER | The user id for the source connection. | | | | | | | | | |
| WSL_SRC_PWD | SRC_PWD | The password for the source connection. | | | | | | | | | |

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 | <p>The unqualified export table name.</p> <p>Note: For Windows script exports from PDW, this variable is initialized with the fully-qualified export table name, due to historical usage.</p> <p>To enable this variable to be uniformly described and used as the unqualified export table name, an additional variable <code>WSL_EXP_SIMPLENAME</code> 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:</p> <pre>if defined WSL_EXP_SIMPLENAME (SET WSL_EXP_TABLE=!WSL_EXP_SIMPLENAME!) else SET WSL_EXP_TABLE=</pre> <p>After such a command is executed, the variable <code>WSL_EXP_TABLE</code> will contain the unqualified export table name.</p> | | | | | | | | | |
| WSL_EXP_SCHEMA | EXP_SCHEMA | <p>The schema for the export table.</p> <p>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.</p> <p>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:</p> <table border="1" data-bbox="671 1570 1386 1854"> <thead> <tr> <th>OS</th> <th>If no trailing dot is wanted</th> <th>If a trailing dot is wanted</th> </tr> </thead> <tbody> <tr> <td>Windows</td> <td>!WSL_EXP_SCHEMA: . = !</td> <td>!WSL_EXP_SCHEMA: . = ! .</td> </tr> <tr> <td>UNIX/Linux</td> <td>`\${EXP_SCHEMA%.}`</td> <td>`\${EXP_SCHEMA%.}`.</td> </tr> </tbody> </table> | 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%.}`. |
| 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 | | | | | | | | | |

| Windows variable | UNIX/Linux variable | Description |
|------------------|---------------------|---|
| | | tables. Others: Not used. |
| 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 <i>Database ID</i> 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 <i>Database ID</i> 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. |

| Windows variable | UNIX/Linux variable | Description | | | | | | | | | |
|------------------|------------------------------|---|----|------------------------------|-----------------------------|---------|-----------------------------|-------------------------------|------------|---------------------|--------------------------|
| WSL_META_SCHEMA | META_SCHEMA | <p>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.</p> <p>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:</p> <table border="1"> <thead> <tr> <th>OS</th> <th>If no trailing dot is wanted</th> <th>If a trailing dot is wanted</th> </tr> </thead> <tbody> <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> </tbody> </table> | OS | If no trailing dot is wanted | If a trailing dot is wanted | Windows | !WSL_META_SCHEMA : . = ! | !WSL_META_SCHEMA : . = ! . | UNIX/Linux | \${META_SCHEMA% . } | \${META_SCHEMA% . } . |
| OS | If no trailing dot is wanted | If a trailing dot is wanted | | | | | | | | | |
| Windows | !WSL_META_SCHEMA : . = ! | !WSL_META_SCHEMA : . = ! . | | | | | | | | | |
| UNIX/Linux | \${META_SCHEMA% . } | \${META_SCHEMA% . } . | | | | | | | | | |
| WSL_META_USER | META_USER | The user id for the meta-repository connection. | | | | | | | | | |
| WSL_META_PWD | META_PWD | The password for the meta-repository connection. | | | | | | | | | |
| WSL_WORKDIR | WORKDIR | <p>Windows: The work directory defined in the Windows connection.</p> <p>UNIX/Linux: The work directory defined in the UNIX/Linux or Hadoop connection.</p> | | | | | | | | | |
| WSL_SEQUENCE | SEQ | A unique sequence number for the load or export task. | | | | | | | | | |
| WSL_PARAMnnn | PARAMnnn | <p>Any parameters that start with the load table or export object name.</p> <p>Example:</p> <p>A table called <i>load_abc</i> has a parameter called <i>load_abc_server</i> defined. In this case, a variable called <i>WSL_PARAM_SERVER</i> (Windows) or <i>PARAM_SERVER</i> (UNIX/Linux) will be created.</p> | | | | | | | | | |

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
```

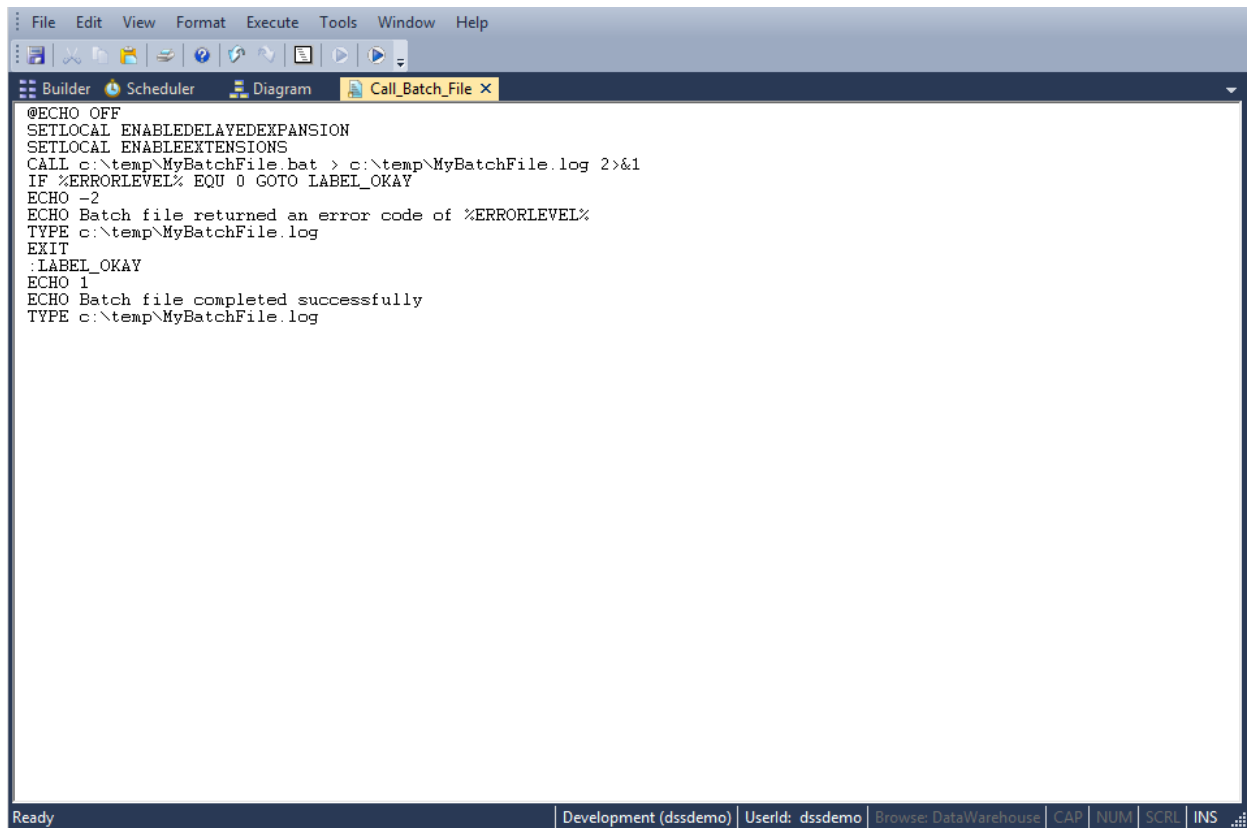
Create the Host Script in RED:

The screenshot shows a dialog box titled "Host Script Call_Batch_File" with a close button (X) in the top right corner. On the left side, there is a sidebar with "Properties" selected and "Notes" below it. The main area contains the following fields:

- Name:** Call_Batch_File
- Type:** Windows script (dropdown menu)
- Purpose:** Calls a batch file (text area)
- Owner:** dssdemo (text field) with a Delete Lock checkbox to its right.
- Last Update By:** (empty text field)
- Default Connect:** Windows (dropdown menu)
- Edit Lock:**
 - Locked For Edit By:** (greyed out text field)
 - Edit Lock Reason or Last Update:** New Script (text area)
- Timestamps:**
 - Created:** 2015-11-19 03:12:07
 - Last Update:** (empty text field)
 - Compiled:** (empty text field)

At the bottom right, there are three buttons: OK, Cancel, and Help.

Edit the Script and enter the following:

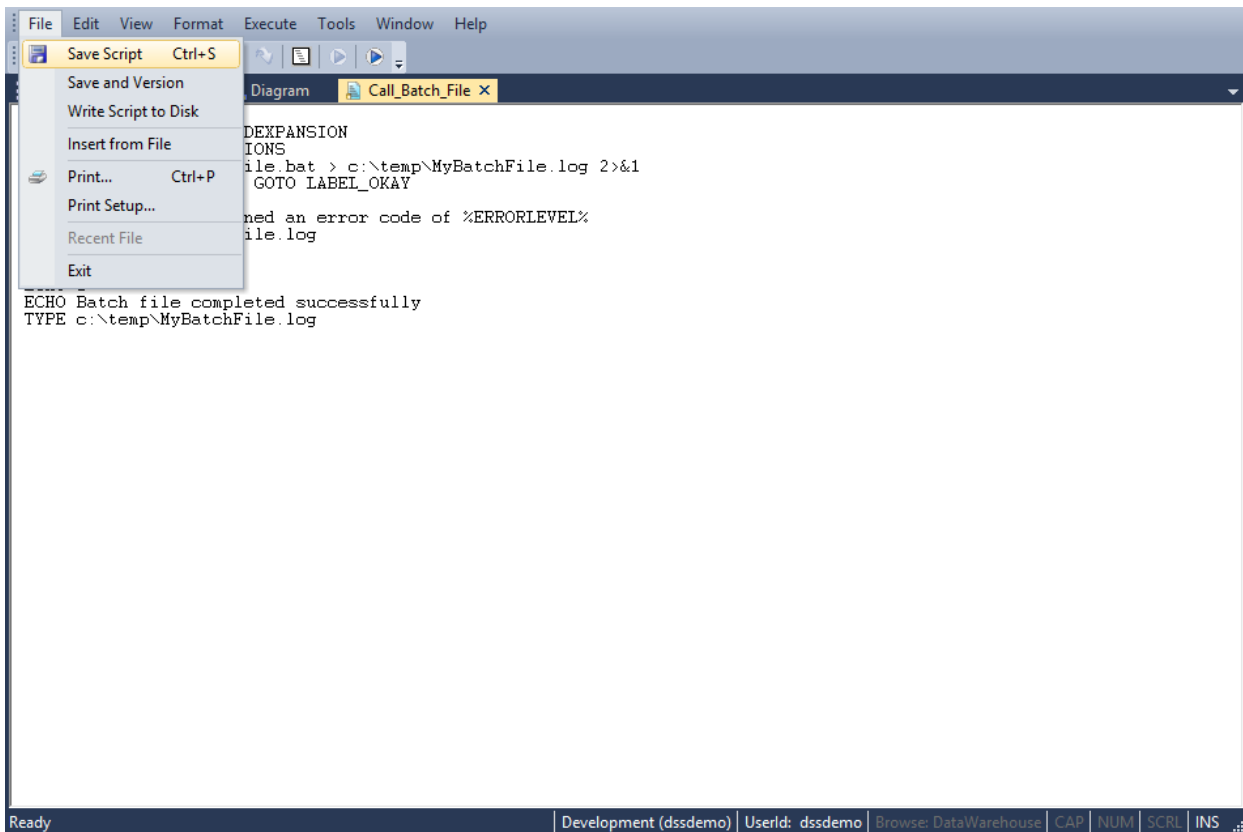


The screenshot shows a software window titled 'Call_Batch_File' with a menu bar (File, Edit, View, Format, Execute, Tools, Window, Help) and a toolbar. The main area contains a batch script with the following content:

```
@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
```

The status bar at the bottom of the window displays: Ready | Development (dssdemo) | Userid: dssdemo | Browse: DataWarehouse | CAP | NUM | SCRL | INS

Save the Script:



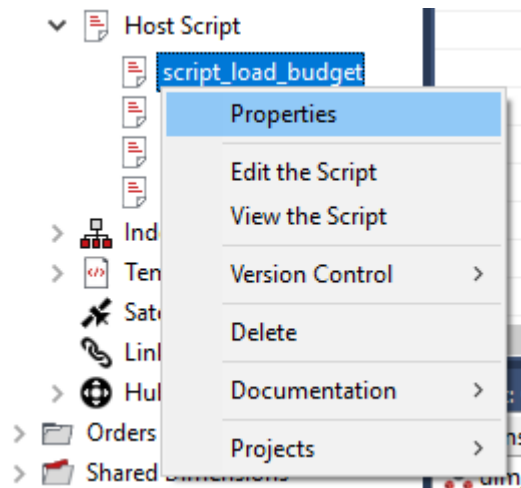
When the script is executed, you will see the following results:

The screenshot shows a 'Results' window with a table containing the following data:

| Object | Message |
|--------|-------------------------------------|
| | # Hello |
| | # 1 |
| | # Batch file completed successfully |
| | # Hello |

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**.

Host Script script_load_budget

Properties

Name: Type: Windows script

Purpose: Auto generated file load

Owner: Delete Lock

Last Update By:

Default Connect: Windows

Edit Lock

Locked For Edit By:

Edit Lock Reason or Last Update:

Timestamps

Created: Last Update: Compiled:

OK Cancel Help

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%\ws1%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. |

CHAPTER 25

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 |
| | | PowerShell Script |
| | | Unix Script |
| | | Olap/XMLA Script |
| Load Script | All database types | Windows Script |
| | | PowerShell Script |
| | | Unix Script |
| | | Olap/XMLA Script |
| Update Procedure | Custom | Block |
| | Hive | Block |
| | SQL, Teradata, Oracle and PDW | Block |
| | | Procedure |

Note: Script-based loads and exports on Windows supports both DOS Batch and PowerShell scripts *for more information* (see "24.11.1.1 Windows PowerShell Scripts" on page 657).

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..... | 679 |
| Template Editor..... | 681 |
| Template Usage..... | 685 |

TEMPLATE PROPERTIES

The properties screen for a template is shown below.

The screenshot shows a 'Template Properties' dialog box with the following fields and values:

- Name: wsl_string_utility
- Purpose: Utility template for string functions
- Author: dssdemo
- Created: 2016-09-02 02:23:28.27
- Last Update: 2016-09-02 02:25:11.54
- Type: Utility
- Target DB: Teradata

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

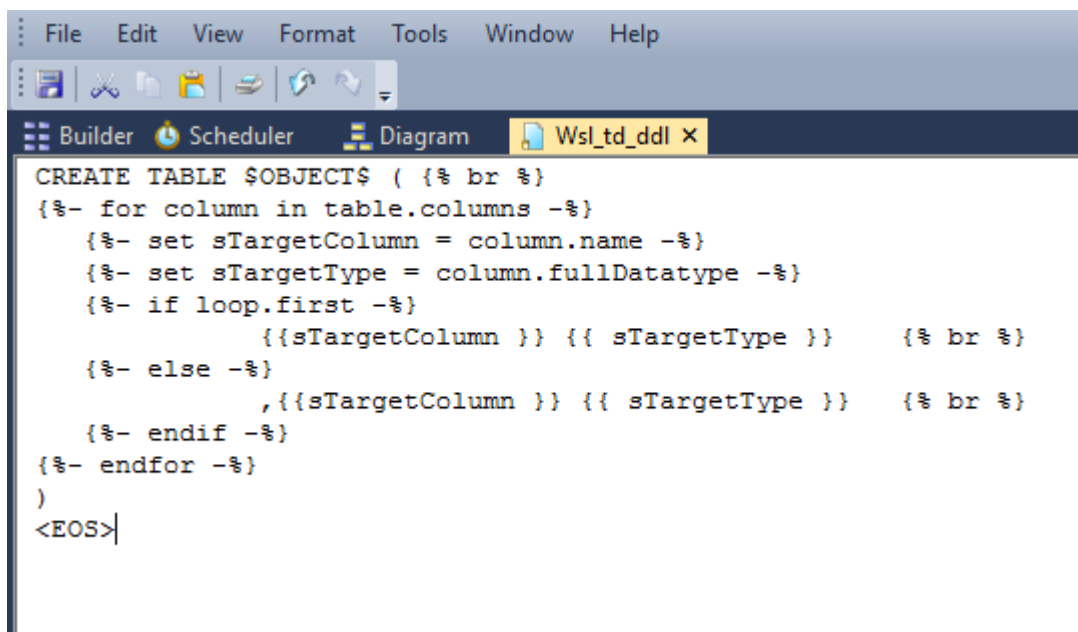
The **Target DB** sets the type of database connections for this template. The template will only be selectable for an operation when the **Target DB** field matches that of the object. Target DB is restricted based on your license.

- 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.

TEMPLATE EDITOR

Right-click on a template and select **'Edit Template'** or **'View Template'** to open the Template Editor.



```
CREATE TABLE $OBJECT$ ( {% br %}
{%- for column in table.columns -%}
    {%- set sTargetColumn = column.name -%}
    {%- set sTargetType = column.fullDatatype -%}
    {%- if loop.first -%}
        {{sTargetColumn }} {{ sTargetType }}      {% br %}
    {%- else -%}
        ,{{sTargetColumn }} {{ sTargetType }}      {% br %}
    {%- endif -%}
{%- endfor -%}
)
<EOS>
```

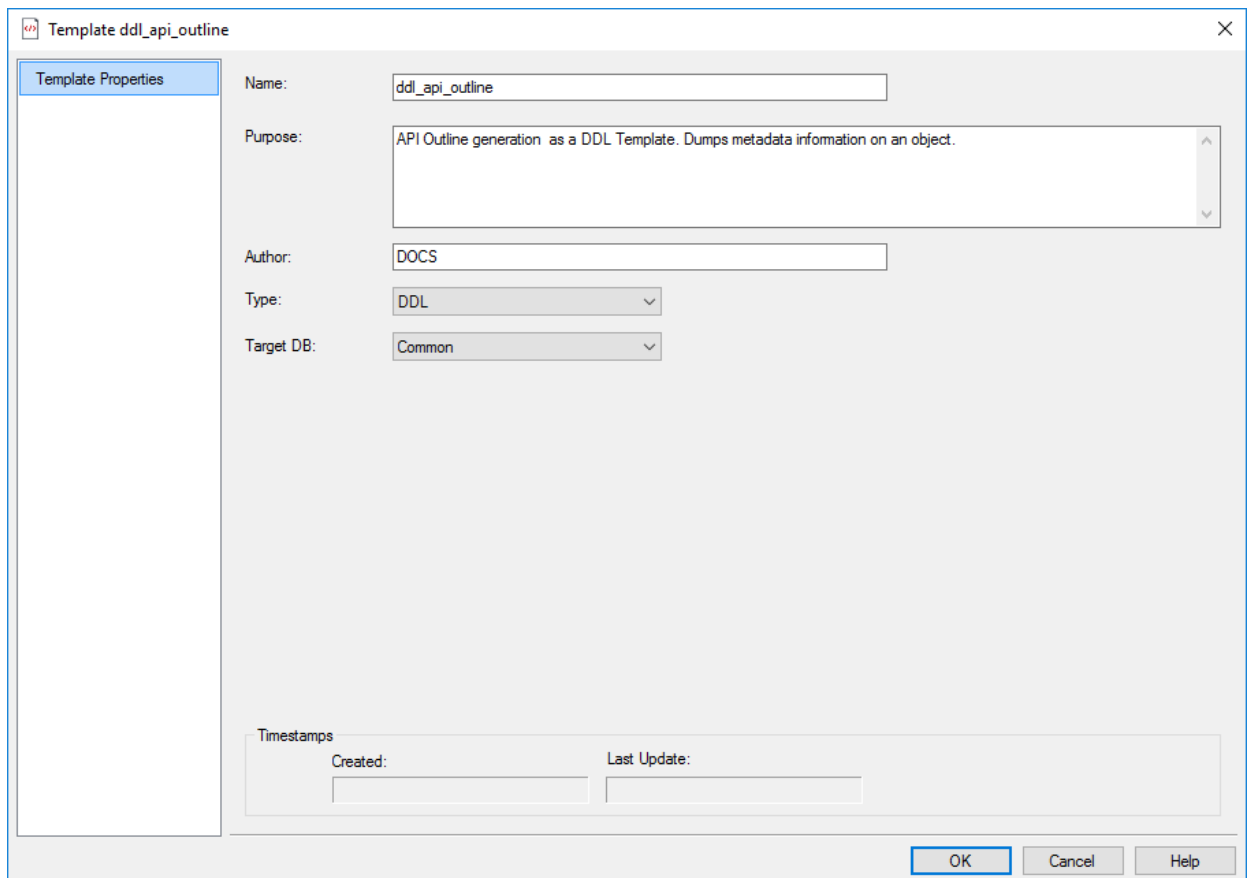
EVALUATING AN API OUTLINE TEMPLATE

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 will use the DDL template type because 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.

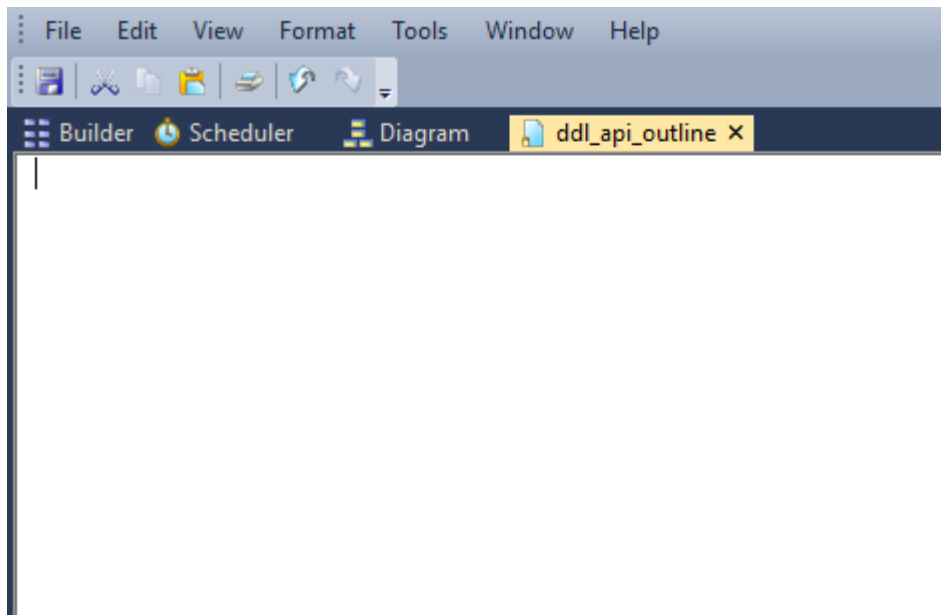


The screenshot shows a dialog box titled "Template ddl_api_outline" with a close button (X) in the top right corner. The dialog is divided into two main sections. On the left is a "Template Properties" sidebar. The main area contains the following fields:

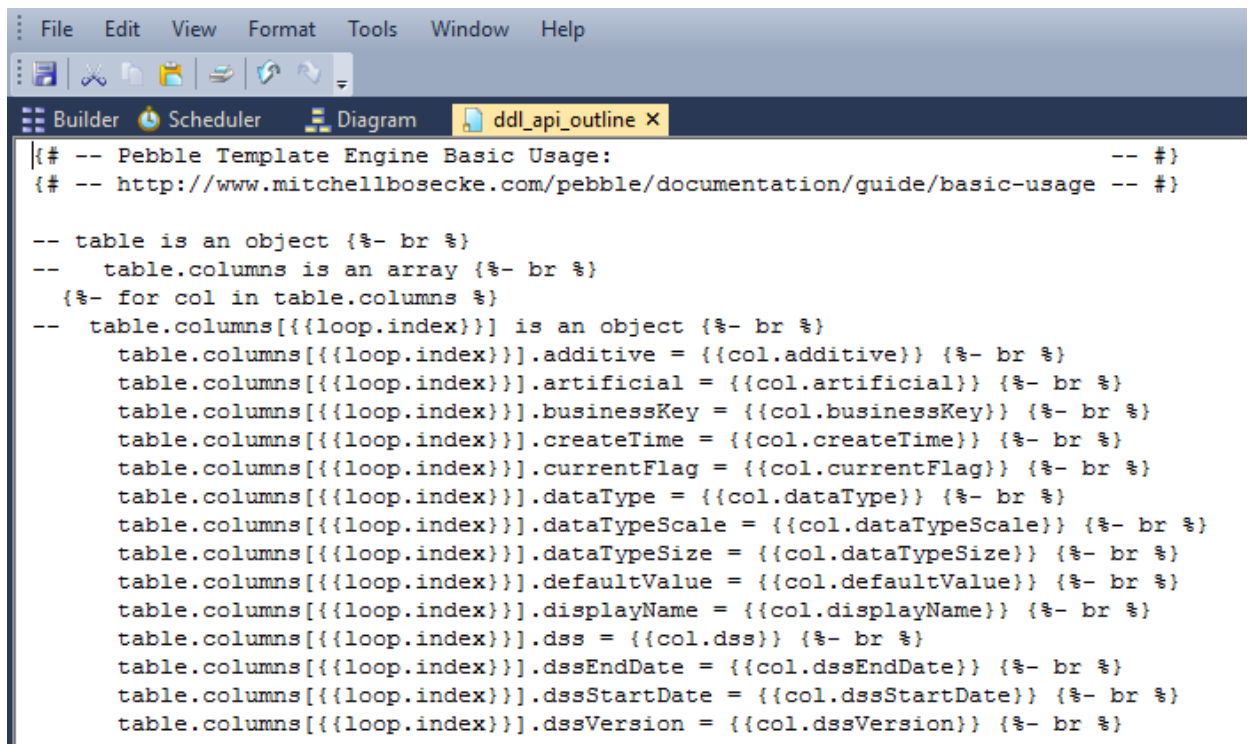
- Name:** A text box containing "ddl_api_outline".
- Purpose:** A text area containing "API Outline generation as a DDL Template. Dumps metadata information on an object."
- Author:** A text box containing "DOCS".
- Type:** A dropdown menu with "DDL" selected.
- Target DB:** A dropdown menu with "Common" selected.

At the bottom of the dialog, there is a "Timestamps" section with two text boxes: "Created:" and "Last Update:". At the very bottom right, there are three buttons: "OK", "Cancel", and "Help".

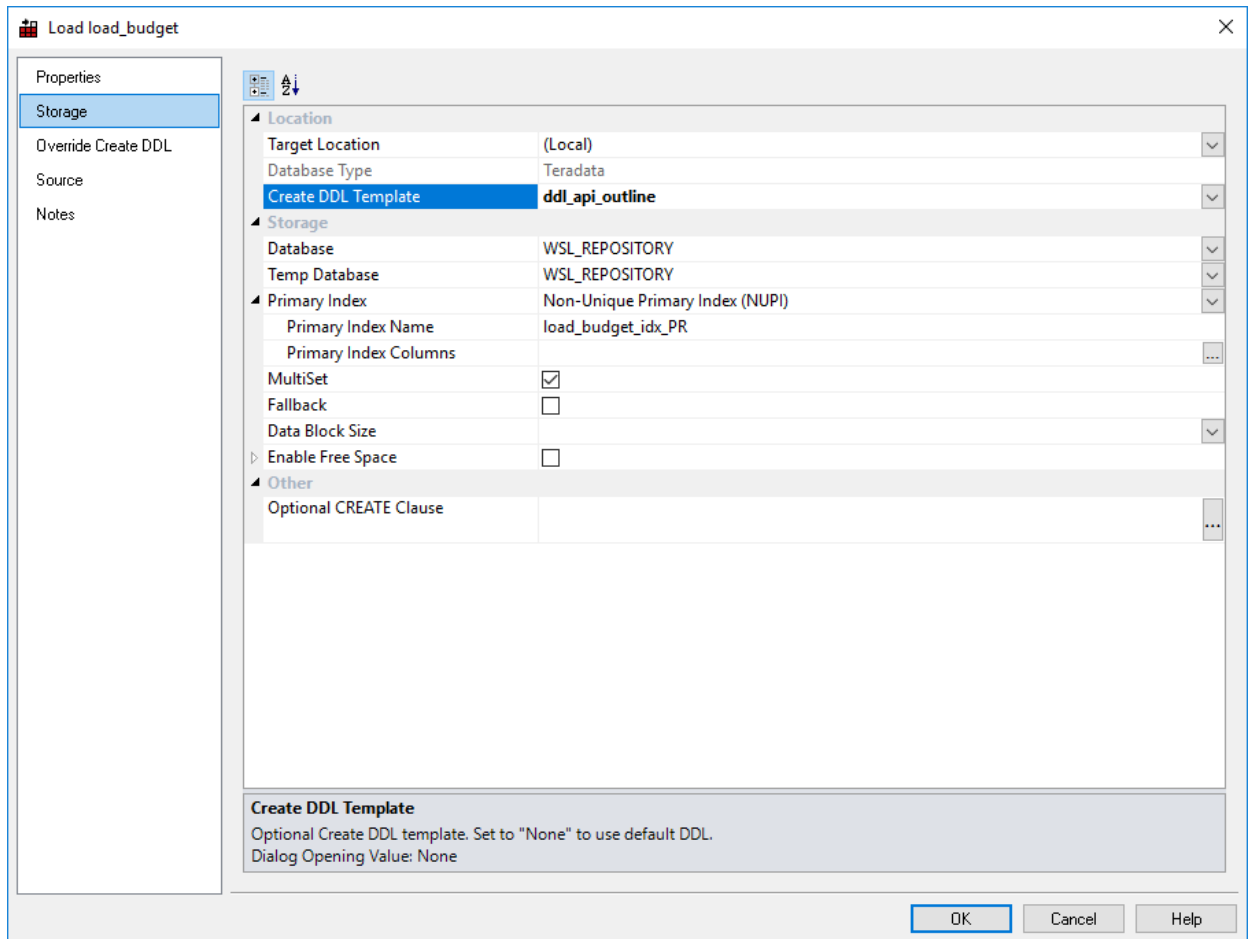
- 2 Open the template in the Template Editor.



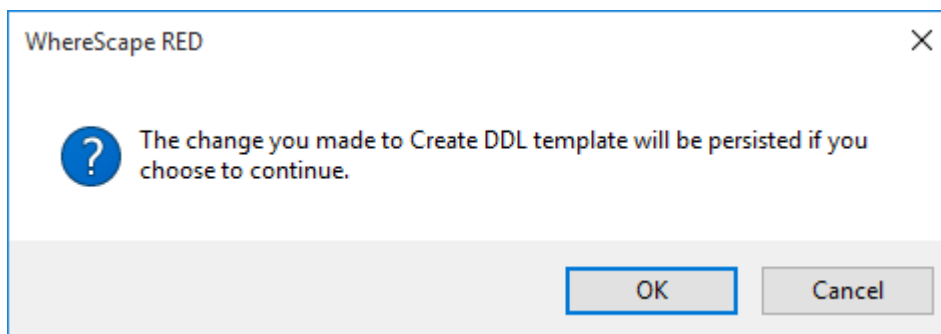
- 3 Click **Tools > Create API Example Outline**. The API Example Outline text is added to the blank template.



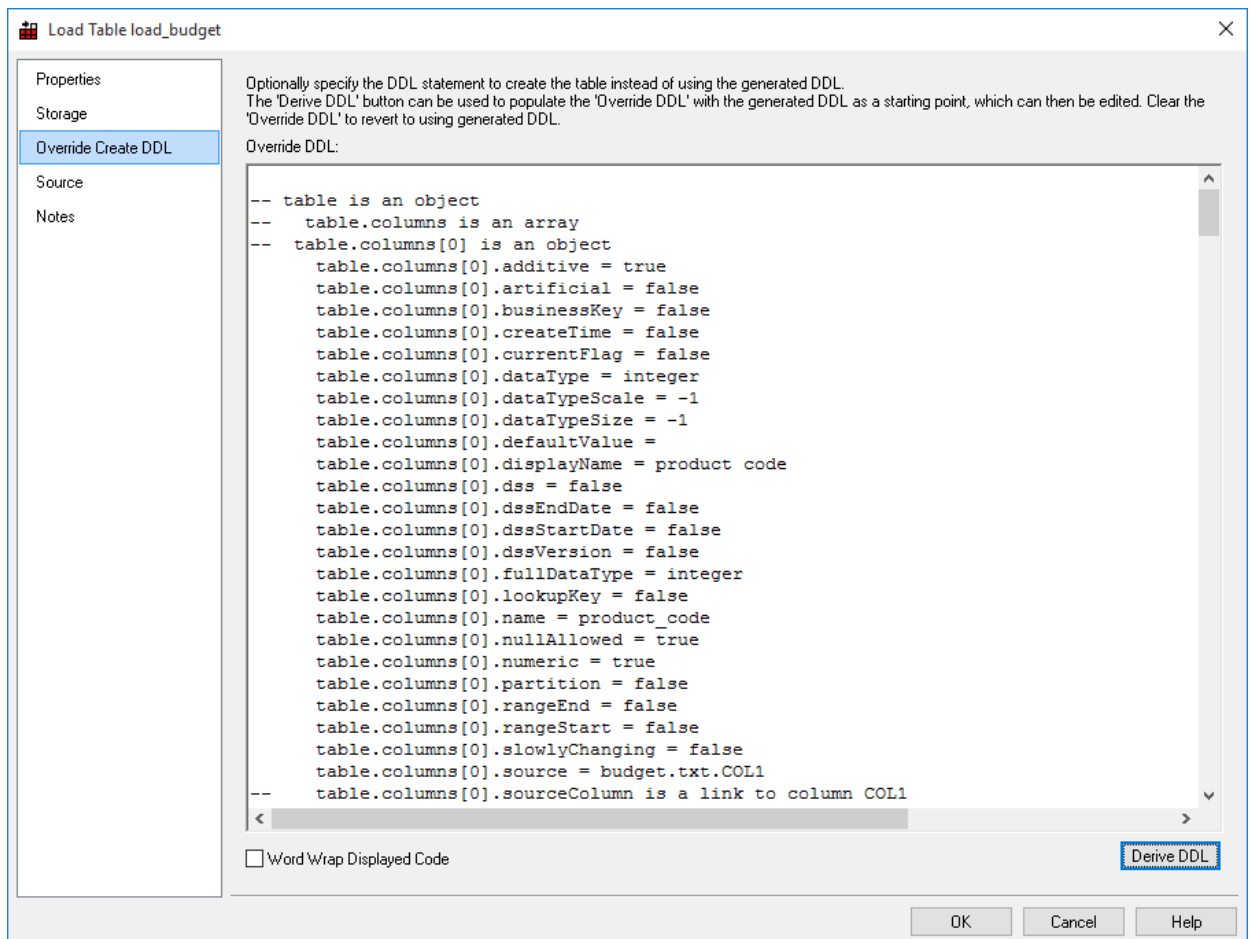
- 4 **Save** and **close** the template.
- 5 Open the Properties dialog for the **Load Table** you wish to evaluate.
- 6 In the **Storage** tab, select the template you created in the **Create DDL Template** drop-down box.



- 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**.



- 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.



- 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 **Override 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>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>File Attributes>Script Template | |
| Load Script | Load Table Properties>Source>Script Template | |
| Update Procedure | When building the Update Procedure, specify the Template field on the Update Build Options>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. |

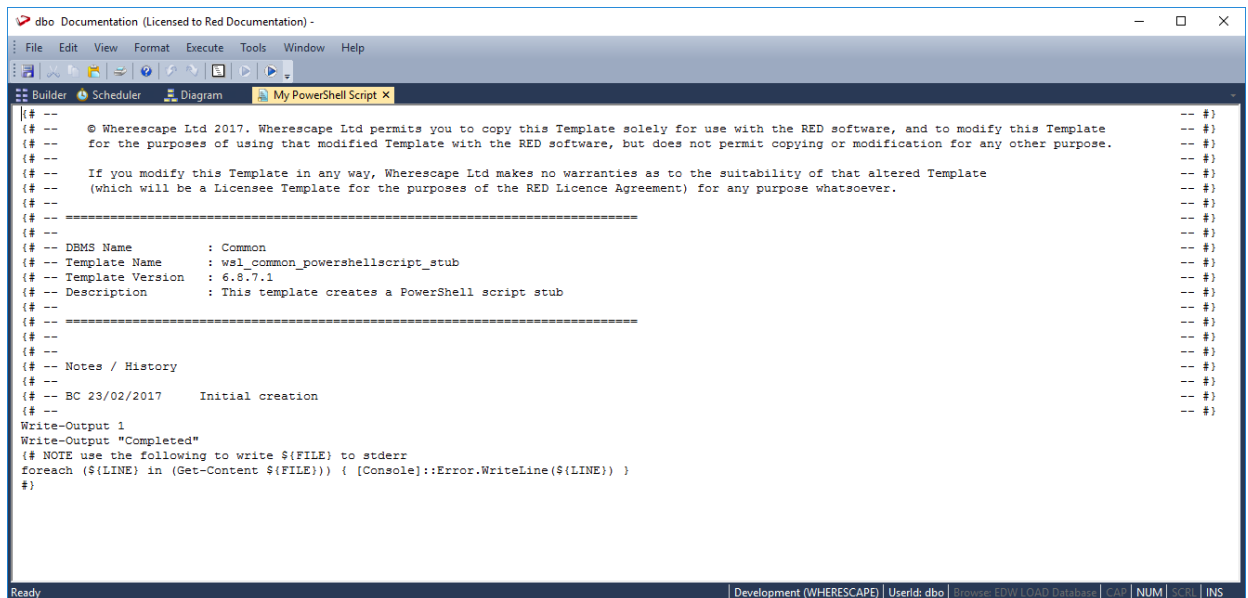
To check which Operations are supported by Templates on your Target DB, see **Templates** (on page 678).

WINDOWS POWERSHELL TEMPLATES

PowerShell Template (wsl_common_powershellscript_stub):

You can use the basic PowerShell stub template available in WhereScape RED that serves as guide on the use of a template to generate a PowerShell script.

Additional PowerShell Templates can be downloaded from the WhereScape website (<https://www.wherescape.com/support/software-downloads-documentation/templates/>).



The screenshot shows a text editor window titled "dbo Documentation (Licensed to Red Documentation)". The editor contains a PowerShell script template with the following content:

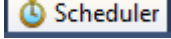
```
{# --
{# -- © Wherescape Ltd 2017. Wherescape Ltd permits you to copy this Template solely for use with the RED software, and to modify this Template
{# -- for the purposes of using that modified Template with the RED software, but does not permit copying or modification for any other purpose.
{# --
{# -- If you modify this Template in any way, Wherescape Ltd makes no warranties as to the suitability of that altered Template
{# -- (which will be a Licensee Template for the purposes of the RED Licence Agreement) for any purpose whatsoever.
{# --
{# -- =====
{# --
{# -- DBMS Name          : Common
{# -- Template Name     : wsl_common_powershellscript_stub
{# -- Template Version  : 6.8.7.1
{# -- Description      : This template creates a PowerShell script stub
{# --
{# -- =====
{# --
{# -- Notes / History
{# --
{# -- BC 23/02/2017    Initial creation
{# --
{# --
Write-Output 1
Write-Output "Completed"
{# NOTE use the following to write ${FILE} to stderr
foreach ($LINE in (Get-Content ${FILE})) { [Console]::Error.WriteLine(${LINE}) }
#}
```

Notes:

The PowerShell stub template (wsl_common_powershellscript_stub) is listed under the Template objects pane, along with the other templates available in WhereScape RED. If this stub template is not visible after installing/upgrading WhereScape RED, use the WhereScape RED Setup Administrator to Validate the Metadata Repository. For more information, please refer to the WhereScape RED Installation Guide.

CHAPTER 26

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 under Windows (as a system service). It processes pre-defined 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

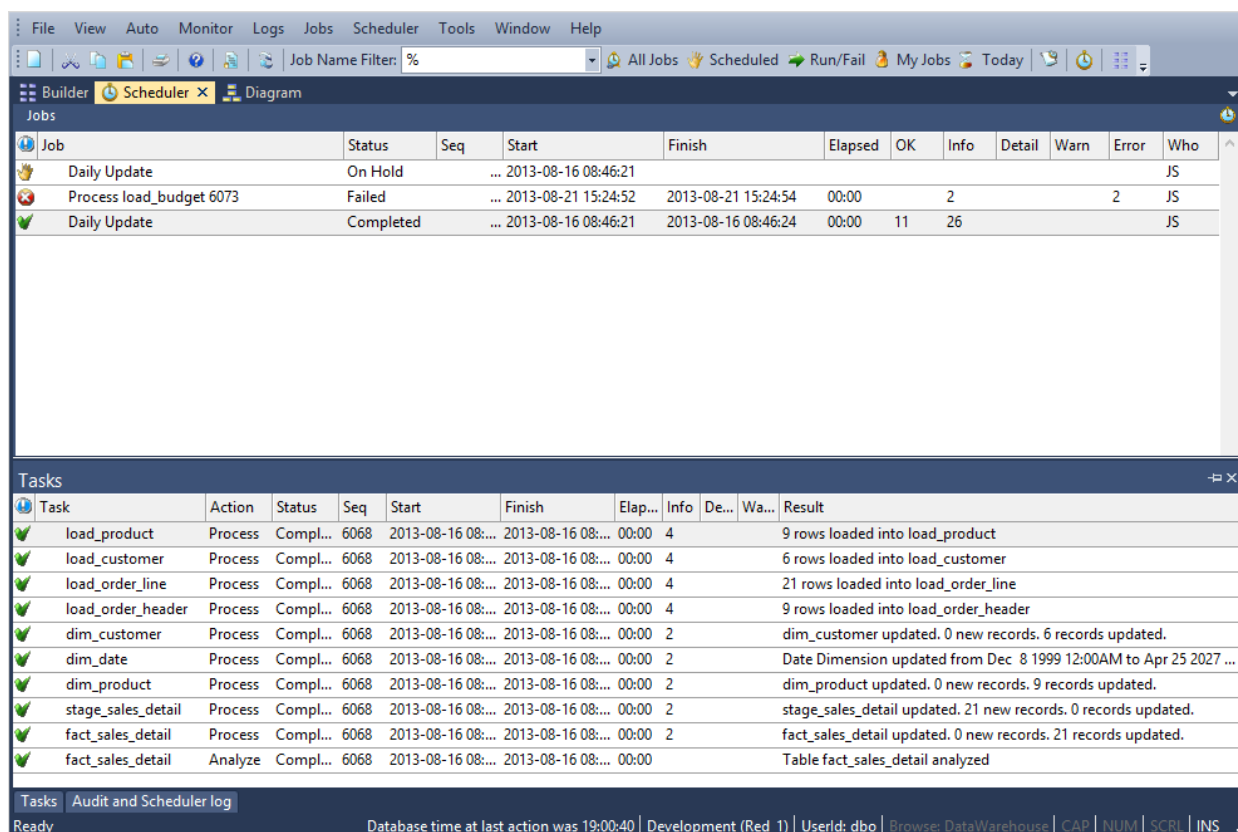
It is possible to view the status of a job without using the WhereScape RED product. Three views are provided to assist in this undertaking. They are `ws_admin_v_audit`, `ws_admin_v_error` and `ws_admin_v_sched`. Queries can be issued using these views to see the results or status of a job.

IN THIS CHAPTER

| | |
|---|-----|
| Scheduler Options | 689 |
| Scheduler States | 695 |
| Scheduling a Job | 698 |
| Working with Jobs | 703 |
| Stand Alone Scheduler Maintenance | 752 |
| SQL to return Scheduler Status | 755 |
| Reset Columns in Job and Task View | 756 |
| Stopping a Linux/UNIX Scheduler from within RED | 756 |

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. |
| Inf | Informational messages written to the audit trail about the the running of the job. |
| Det | Lines written to the detail or error logs |
| Warn | The number of warnings written to the audit trail. |
| Err | 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:

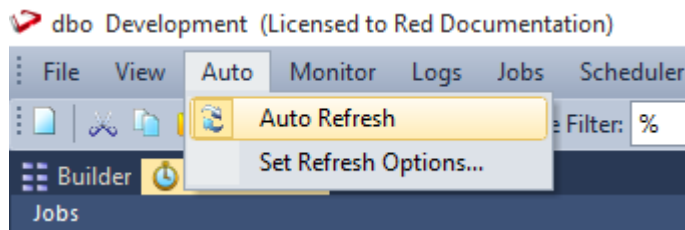
| Column | Description |
|------------|--|
| Task | The object name |
| Status | The status of the message |
| Sequence | This is a unique number assigned to each job iteration and job. If you enter a new job it will acquire a new sequence number. In normal daily processing when no new jobs have been created sequence numbers will be sequential. |
| Timestamp | Time of message output |
| Message | The message |
| DB Message | Message from database |

| Column | Description |
|--------|------------------------------|
| Job | Job relating to this message |

AUTO

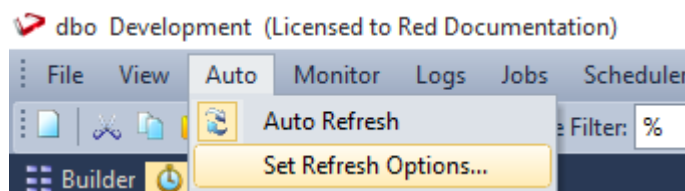
Auto Refresh

Use auto refresh to automatically refresh all jobs.



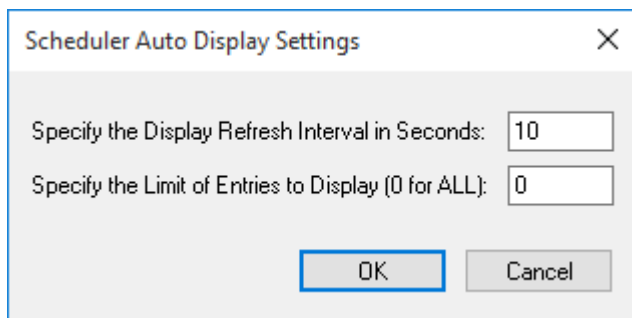
Set Refresh Options

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 display of jobs when on auto refresh will stop at the selected number of rows, so when set to 100, the refresh will stop after 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.

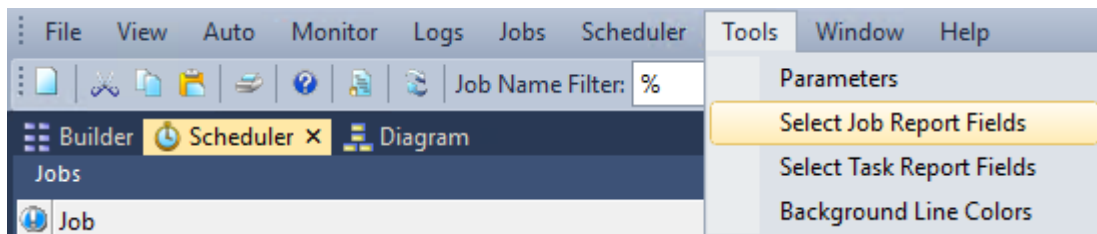


TOOLS

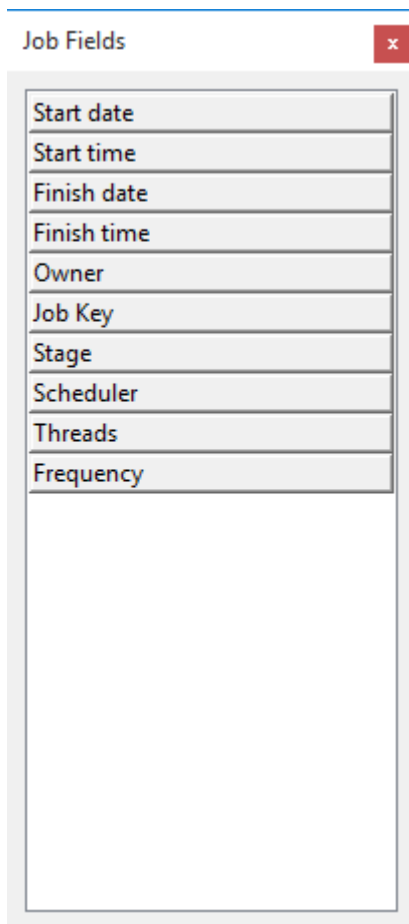
SELECT JOB REPORT FIELDS

The **Select Job Report Fields** menu option in the scheduler pane enables users to select some 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 in order 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. |
| 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 tasks 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 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

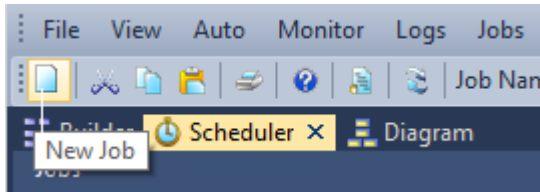
| State | Description |
|-------------------|--|
| Held | The task has been held due to a prior dependency failure. The problem must be rectified and the job restarted. |
| Waiting (Blank) | Tasks that are waiting to run either due to a shortage of threads, or prior dependencies normally have a blank 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

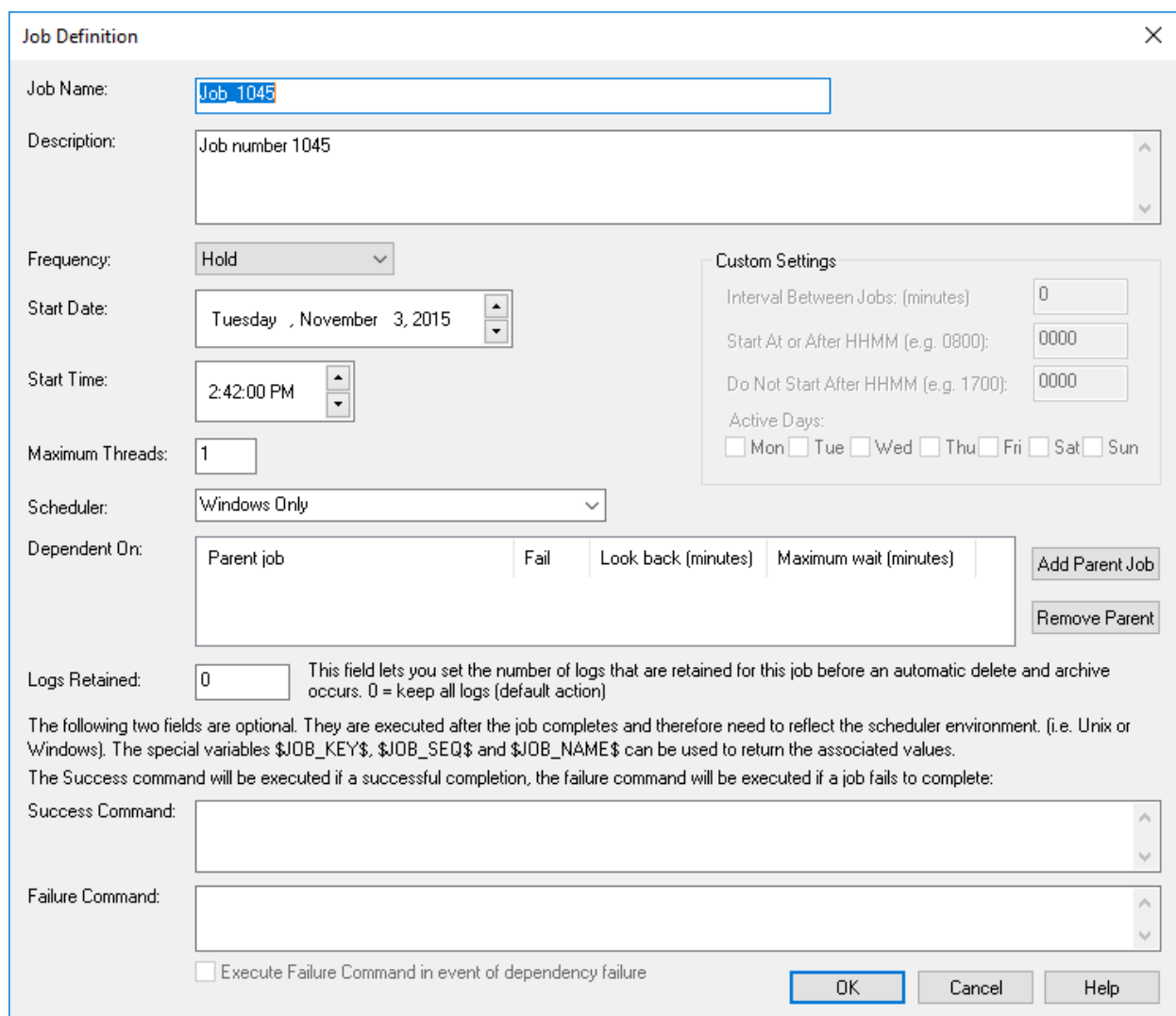
To schedule a job

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.

A screenshot of the 'Job Definition' dialog box. The 'Job Name' field contains 'Job_1045'. The 'Description' field contains 'Job number 1045'. The 'Frequency' is set to 'Hold'. The 'Start Date' is 'Tuesday, November 3, 2015' and the 'Start Time' is '2:42:00 PM'. The 'Maximum Threads' is '1'. The 'Scheduler' is set to 'Windows Only'. The 'Dependent On' section shows 'Parent job' with options for 'Fail', 'Look back (minutes)', and 'Maximum wait (minutes)'. There are 'Add Parent Job' and 'Remove Parent' buttons. The 'Logs Retained' is '0'. There are fields for 'Success Command' and 'Failure Command'. At the bottom, there is a checkbox for 'Execute Failure Command in event of dependency failure' and 'OK', 'Cancel', and 'Help' buttons.

Job Definition

Job Name: Job_1045

Description: Job number 1045

Frequency: Hold

Start Date: Tuesday, November 3, 2015

Start Time: 2:42:00 PM

Maximum Threads: 1

Scheduler: Windows Only

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:

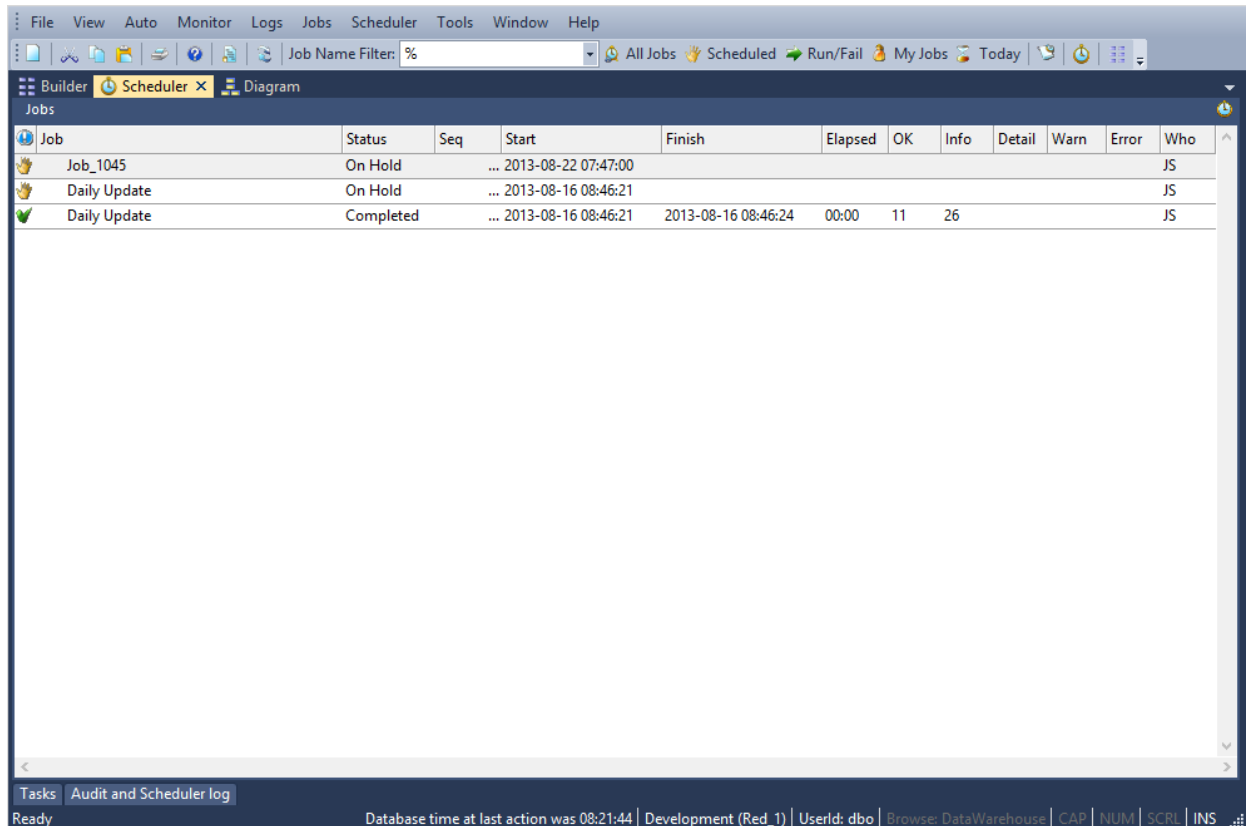
Failure Command:

Execute Failure Command in event of dependency failure

OK Cancel Help

See **Creating a Job** (on page 706) for more details on how to create a 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 the **Dependent On** field in the **Job Definition** dialog.

Job Definition [X]

Job Name: Enterprise Reporting Daily Refresh

Description: Full Daily Refresh of all Presentation Tables for the Enterprise Reporting Layer

Frequency: Custom

Start Date: Thursday, March 3, 2016

Start Time: 3:00:00 AM

Maximum Threads: 4 Inactive Wait Interval (seconds): 30

Scheduler: Windows Only

Dependent On: Parent job | Fail | Look back (minutes) | Maximum wait (minutes) | Add Parent Job | Remove Parent

Custom Settings

Interval Between Jobs: (minutes) 1440

Start At or After HHMM (e.g. 0800): 0300

Do Not Start After HHMM (e.g. 1700): 0500

Active Days: Mon Tue Wed Thu Fri Sat Sun

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

Click on the **Add Parent Job** button.

Job Definition [X]

Job Name: Enterprise Reporting Daily Refresh

Description: Full Daily Refresh of all Presentation Tables for the Enterprise Reporting Layer

Frequency: Custom

Start Date: Thursday, March 3, 2016

Start Time: 3:00:00 AM

Maximum Threads: 4 Inactive Wait Interval (seconds): 30

Scheduler: Windows Only

Custom Settings

- Interval Between Jobs: (minutes) 1440
- Start At or After HHMM (e.g. 0800): 0300
- Do Not Start After HHMM (e.g. 1700): 0500
- 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

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**.

Add a Parent Job [X]

Select a Job on which this Job is dependent. Specify the amount of time to look back for the Parent Job completion. Normally this would be in the 8 to 10 hour range to prevent seeing a previous days run.

Next specify a maximum time to wait for this Parent Job to complete. A failure will occur if the Parent Job does not complete in the specified time.

Parent Job:
Shared Dimensions Daily Refresh [v]

Maximum Time to Look Back for the Parent Job Completion (minutes):
60

Maximum Time to Wait for the Parent Job to Complete (minutes):
20

Fail if Parent Job Does Not Complete Successfully in the Required Time:

[Add] [Cancel]

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 checkbox checked to ensure that this job does not run.

Click **Add**.

NOTE: Clearing the checkbox to fail if the parent fails will simply ensure that this job waits for the completion of the dimension refresh and, irrespective of the dimensions refresh's 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.

Job Definition
✕

Job Name:

Description:

Frequency: Custom ▾

Start Date:

Start Time:

Maximum Threads: Inactive Wait Interval (seconds):

Scheduler: Windows Only ▾

| Parent job | Fail | Look back (minutes) | Maximum wait (minutes) | |
|---------------------------------|------|---------------------|------------------------|---|
| Shared Dimensions Daily Refresh | Y | 60 | 20 | <input type="button" value="Add Parent Job"/> <input type="button" value="Remove Parent"/> |

Logs Retained: 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)

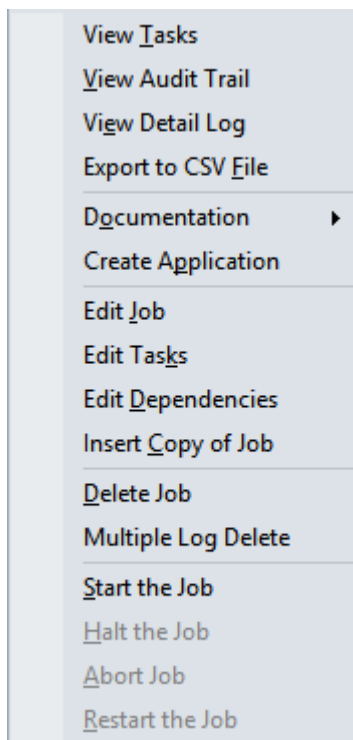
The following two fields are optional. They are executed after the job completes and therefore need to reflect the scheduler environment. (i.e. Unix or Windows). The special variables \$JOB_KEY\$, \$JOB_SEQ\$ and \$JOB_NAME\$ can be used to return the associated values.
The Success command will be executed if a successful completion, the failure command will be executed if a job fails to complete:

Success Command:

Failure Command:

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 718)

The **Edit Tasks** option enables you to edit the tasks of a job. See *Editing Tasks* (see "*Editing Tasks in a Job*" on page 722)

The **Edit Dependencies** option enables you to edit the task dependencies of a job. See *Editing Dependencies* (see "*Editing Task Dependencies*" on page 729)

The **Insert Copy of Job** option enables you to insert a copy of a job. See *Inserting a Copy of a Job* (on page 736)

The **Delete Job** option enables you to delete a job. See *Deleting a Job* (on page 737)

The **Multiple Log Delete** option enables you to delete multiple logs of a job. See *Deleting Job Logs* (on page 739)

The **Start the Job** option enables you to start a job. See *Starting a Job* (on page 741)

The **Halt the Job** option enables you to halt a job. See *Halting a Job* (on page 742)

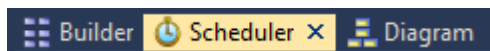
The **Abort the Job** option enables you to abort a job. See *Aborting a Job* (on page 743)

The **Restart the Job** option enables you to restart a job. See *Restarting a Job* (on page 744)

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.

Job Definition
✕

Job Name:

Description:

Frequency:

Start Date:

Start Time:

Maximum Threads:

Scheduler:

Dependent On:
Fail Look back (minutes) Maximum wait (minutes)
Add Parent Job
Remove Parent

Logs Retained: 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:

Execute Failure Command in event of dependency failure

Complete the fields and click **OK**. The main fields are described in the following table:

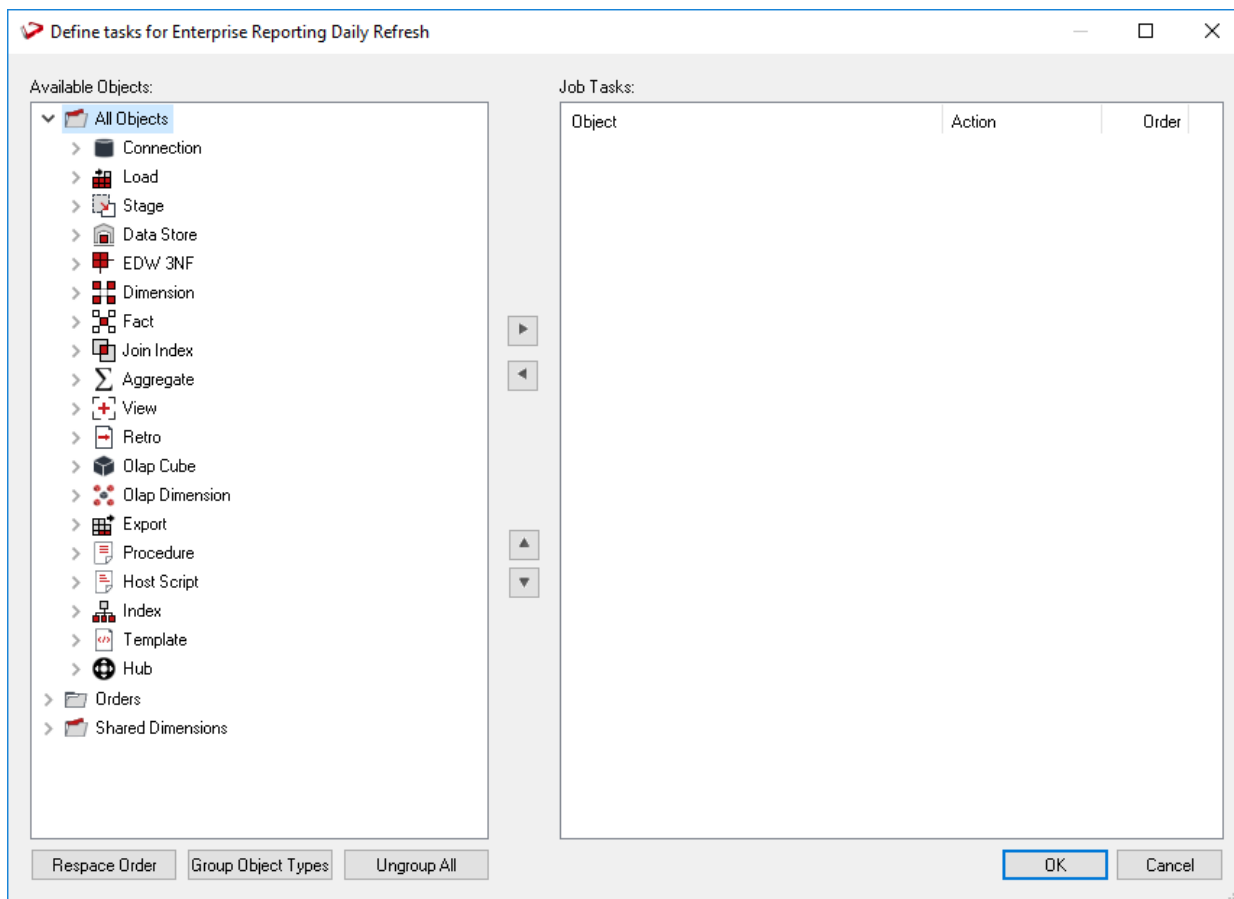
| Field | Description |
|-------------|---|
| Job Name | The Scheduler defaults to the next job number in the sequence. You can alter this to any alphanumeric. Note: Only alphanumerics, spaces and the underscore are supported in the name. |
| Description | A description of the job |

| Field | Description |
|----------------------------------|---|
| Frequency | <p>When the job runs. The options available in the drop-down list are:</p> <ul style="list-style-type: none"> • 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 | <ul style="list-style-type: none"> • The date and time for the job to start. |
| Max Threads | <p>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.</p> |
| Scheduler | <p>It is possible to have multiple schedulers running. Select the desired scheduler from this drop-down. The valid options are:</p> <p>Windows Preferred, Windows Only, or the name of a specific scheduler can be entered (e.g. WIN0002)</p> |
| Dependent On | <p>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.</p> <p>See the Dependency example in <i>Scheduling a Job</i> (on page 698)</p> |
| Logs Retained | <p>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.</p> |
| Okay command and Failure command | <p>These are 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.</p> <p>Notes:</p> <ol style="list-style-type: none"> 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: <pre>>> c:\scheduler\success_failure_prod.log 2>&1</pre> |

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 to add it to the left pane. Normally objects such as load or fact tables are scheduled rather than procedures.

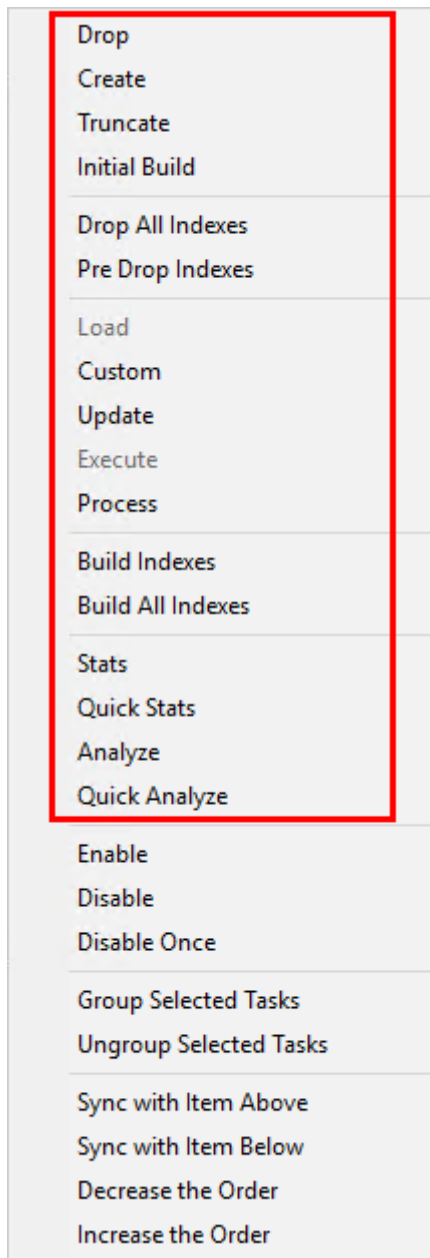
To set the action on a task

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:

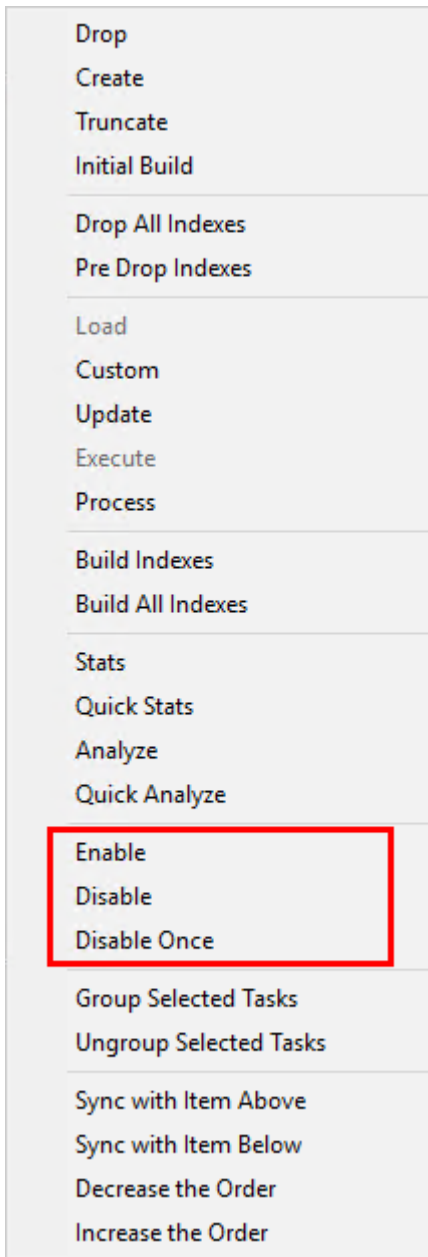
| Action | Description |
|----------|--|
| Drop | Drop table, view, join index or index. |
| Create | Create table, view, join index or index. |
| Truncate | Delete all rows from the table. |

| Action | Description |
|------------------------|--|
| 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. |
| Stats | Refreshes predefined statistics on a table or index: COLLECT STATISTICS ON DatabaseName.TableName; COLLECT STATISTICS ON DatabaseName.TableName INDEX IndexName; |
| Quick Stats | Refreshes predefined statistics on an index using sampling: COLLECT STATISTICS USING SAMPLE ON DatabaseName.TableName INDEX IndexName; |
| Analyze | Refreshes predefined statistics on a table or index: COLLECT STATISTICS ON DatabaseName.TableName; COLLECT STATISTICS ON DatabaseName.TableName INDEX IndexName; |
| Quick Analyze | Refreshes predefined statistics on an index using sampling: COLLECT STATISTICS USING SAMPLE ON DatabaseName.TableName INDEX IndexName; |

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:



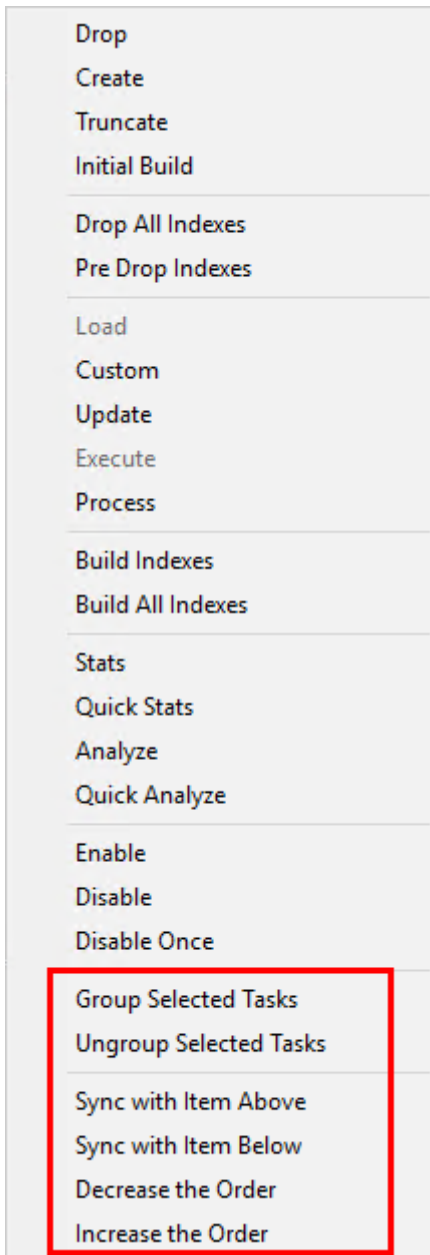
The following states are available:

| 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:

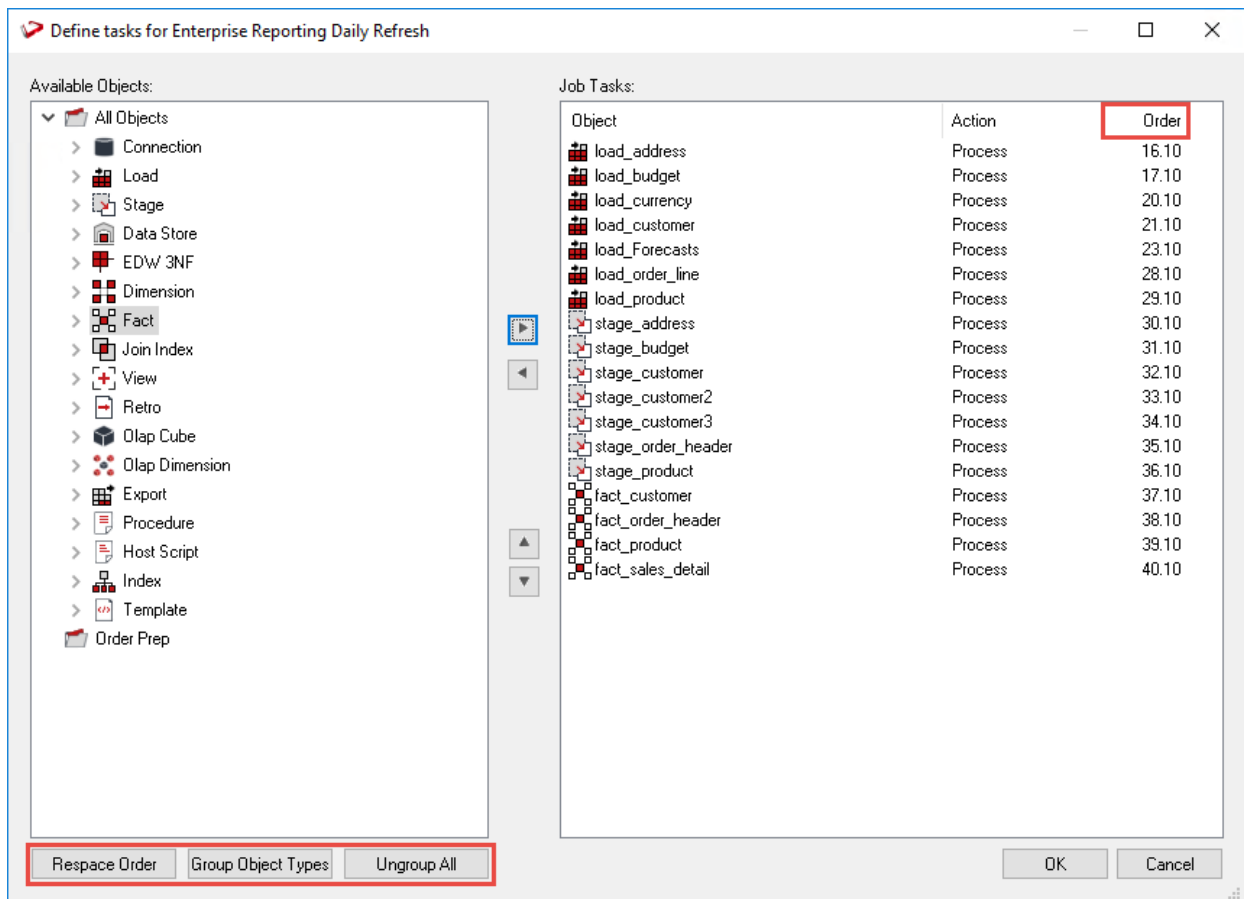
| Task Option | Description |
|------------------------|--|
| Group Selected Tasks | Groups two or more selected tasks to have the same order value, allowing them to run in parallel if the maximum threads setting allows. |
| Ungroup Selected Tasks | Un-group selected tasks. |
| Sync with Item Above | 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 adding tasks, click **OK**.

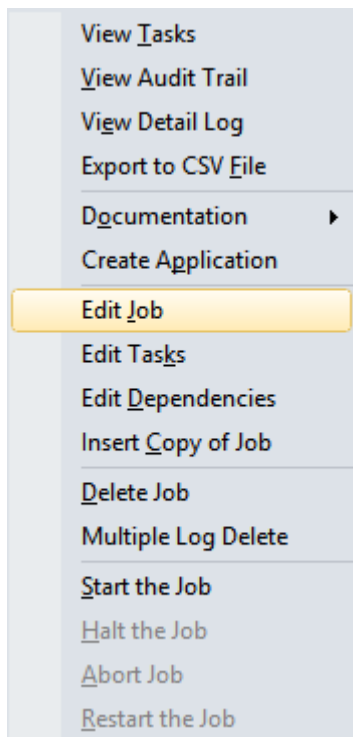
EDITING A JOB

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.

Job Definition [X]

Job Name:

Description:

Frequency:

Start Date:

Start Time:

Maximum Threads: Inactive Wait Interval (seconds):

Scheduler:

Dependent On: Fail

Logs Retained: 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:

Execute Failure Command in event of dependency failure

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. |
| Description | A description of the job |

| Field | Description |
|----------------------------------|---|
| Frequency | <p>When the job runs. The options available in the drop-down list are:</p> <ul style="list-style-type: none"> • 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 | <ul style="list-style-type: none"> • The date and time for the job to start. |
| Max Threads | <p>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.</p> |
| Scheduler | <p>It is possible to have multiple schedulers running. Select the desired scheduler from this drop-down. The valid options are:</p> <p>Windows Preferred, Windows Only, or the name of a specific scheduler can be entered (e.g. WIN0002)</p> |
| Dependent On | <p>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.</p> <p>See the Job Dependency example in <i>Scheduling a Job</i> (on page 698)</p> |
| Logs Retained | <p>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.</p> |
| Okay command and Failure command | <p>These are 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.</p> <p>Notes:</p> <ol style="list-style-type: none"> 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: <pre>>> c:\scheduler\success_failure_prod.log 2>&1</pre> |

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

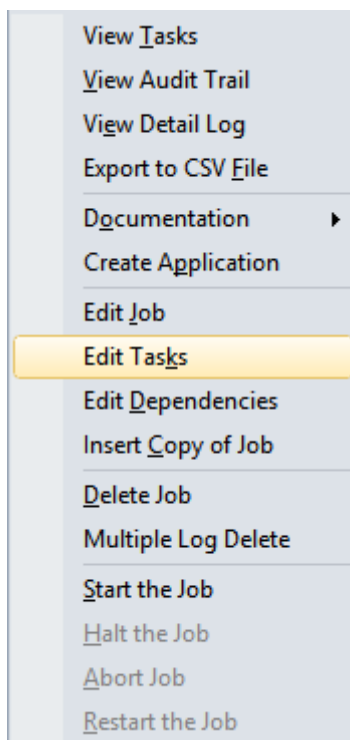
Once jobs have been created, you can edit their tasks.

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.

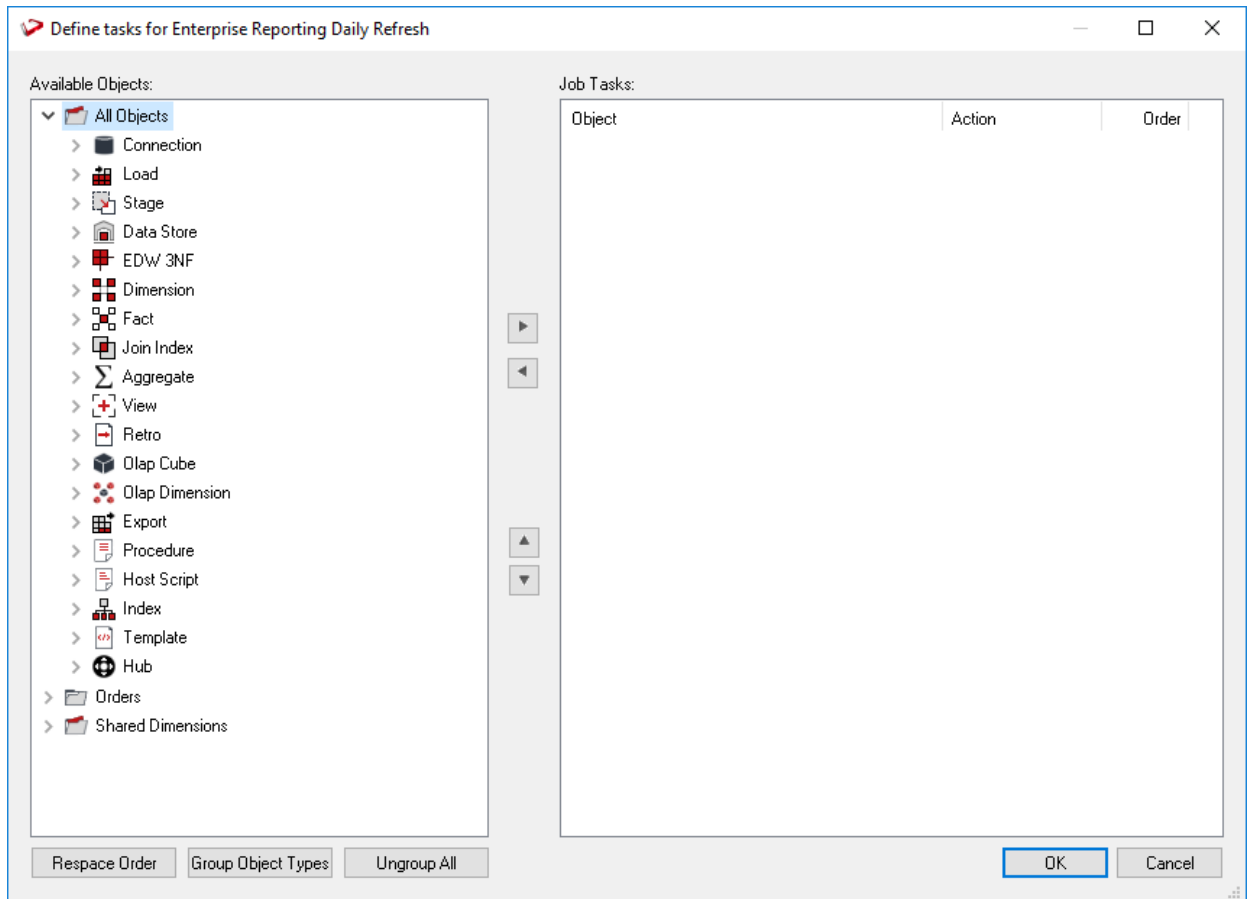
JOB TASK LIMIT - There is maximum number of **999 tasks** that can be added to a job.

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 to add it to the left pane. Normally objects such as load or fact tables are scheduled rather than procedures.

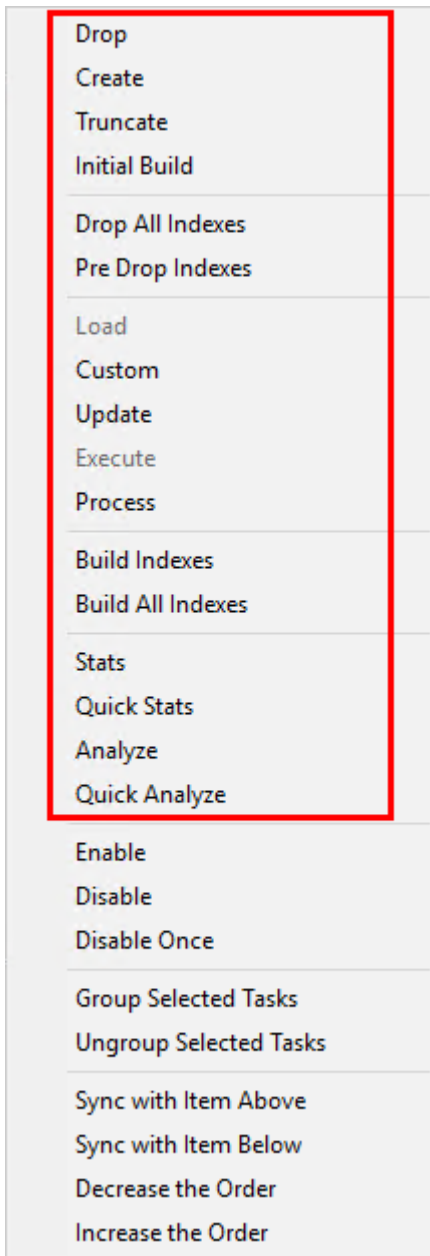
To set the action on a task

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:

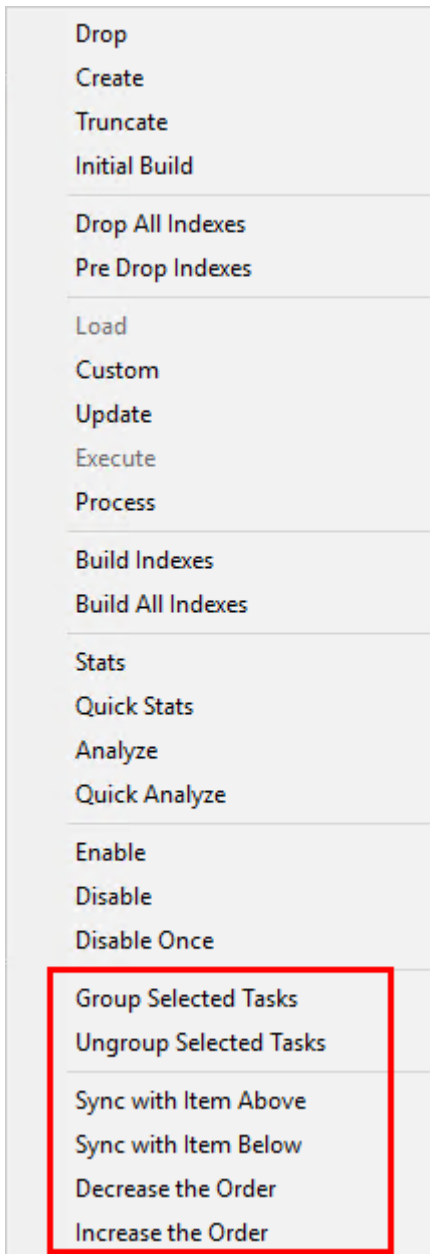
| Action | Description |
|----------|--|
| Drop | Drop table, view, join index or index. |
| Create | Create table, view, join index or index. |
| Truncate | Delete all rows from the table. |

| Action | Description |
|------------------------|--|
| 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. |
| Stats | Refreshes predefined statistics on a table or index: COLLECT STATISTICS ON DatabaseName.TableName; COLLECT STATISTICS ON DatabaseName.TableName INDEX IndexName; |
| Quick Stats | Refreshes predefined statistics on an index using sampling: COLLECT STATISTICS USING SAMPLE ON DatabaseName.TableName INDEX IndexName; |
| Analyze | Refreshes predefined statistics on a table or index: COLLECT STATISTICS ON DatabaseName.TableName; COLLECT STATISTICS ON DatabaseName.TableName INDEX IndexName; |
| Quick Analyze | Refreshes predefined statistics on an index using sampling: COLLECT STATISTICS USING SAMPLE ON DatabaseName.TableName INDEX IndexName; |

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:

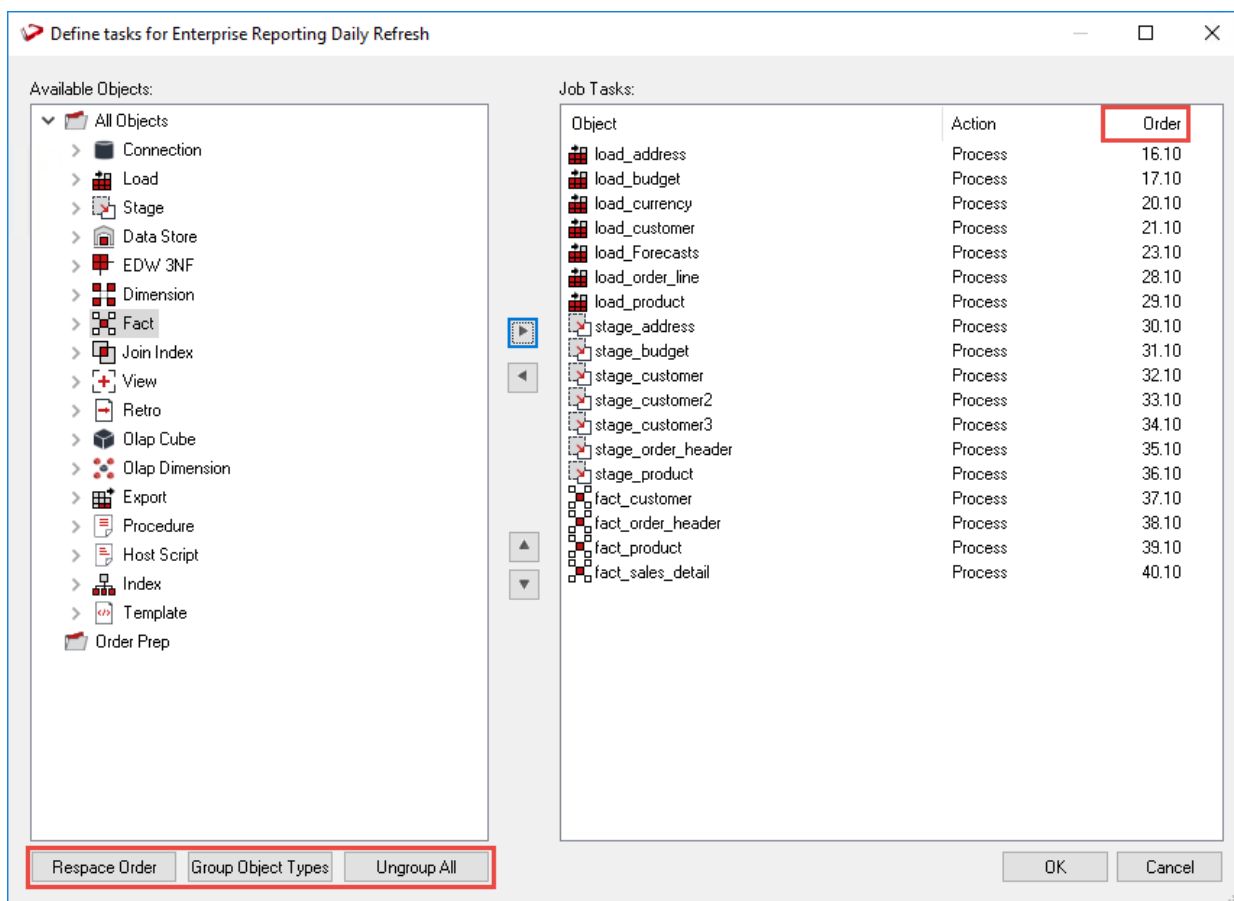
| Task Option | Description |
|----------------------|---|
| Group Selected Tasks | Groups two or more selected tasks to have the same order value, allowing them to run in parallel if the maximum threads setting allows. |

| Task Option | Description |
|------------------------|--|
| 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 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 editing tasks, click **OK**.

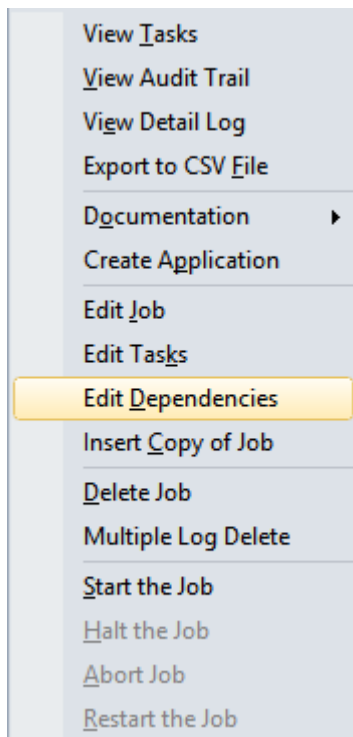
EDITING TASK DEPENDENCIES

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_address | Process | --> | stage_order_detail_sales | Process |
| load_address | Process | --> | stage_order_detail_sales2 | Process |
| load_address_sales2 | Process | --> | stage_order_detail_sales | Process |
| load_address_sales2 | Process | --> | stage_order_detail_sales2 | Process |
| load_currency_rate | Process | --> | stage_order_detail_sales | Process |
| load_currency_rate | Process | --> | stage_order_detail_sales2 | Process |
| load_forecast | Process | --> | stage_order_detail_sales | Process |
| load_forecast | Process | --> | stage_order_detail_sales2 | Process |
| load_order_header | Process | --> | stage_order_detail_sales | Process |
| load_order_header | Process | --> | stage_order_detail_sales2 | Process |
| load_order_header_partner | Process | --> | stage_order_detail_sales | Process |
| load_order_header_partner | Process | --> | stage_order_detail_sales2 | Process |
| load_order_header_sales2 | Process | --> | stage_order_detail_sales | Process |
| load_order_header_sales2 | Process | --> | stage_order_detail_sales2 | Process |
| load_order_line | Process | --> | stage_order_detail_sales | Process |
| load_order_line | Process | --> | stage_order_detail_sales2 | Process |
| load_order_line_partner | Process | --> | stage_order_detail_sales | Process |
| load_order_line_partner | Process | --> | stage_order_detail_sales2 | Process |
| load_order_line_sales2 | Process | --> | stage_order_detail_sales | Process |
| load_order_line_sales2 | Process | --> | stage_order_detail_sales2 | Process |
| load_sales_quota | Process | --> | stage_order_detail_sales | Process |
| load_sales_quota | Process | --> | stage_order_detail_sales2 | Process |
| stage_order_detail_sales | Process | --> | stage_order_detail_1 | Process |
| stage_order_detail_sales2 | Process | --> | stage_order_detail_1 | Process |
| stage_order_detail_1 | Process | --> | stage_currency_rate | Process |
| stage_order_detail_1 | Process | --> | stage_forecast | Process |
| stage_order_detail_1 | Process | --> | stage_order_detail | Process |
| stage_currency_rate | Process | --> | fact_order_detail | Process |
| stage_forecast | Process | --> | fact_order_detail | 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**.

| Parent Task | Parent Action | Map | Child Task | Child Action |
|---------------------------|---------------|-----|---------------------------|--------------|
| load_address | Process | --> | stage_order_detail_sales | Process |
| load_address | Process | --> | stage_order_detail_sales2 | Process |
| load_address_sales2 | Process | --> | stage_order_detail_sales | Process |
| load_address_sales2 | Process | --> | stage_order_detail_sales2 | Process |
| load_currency_rate | Process | --> | stage_order_detail_sales | Process |
| load_currency_rate | Process | --> | stage_order_detail_sales2 | Process |
| load_forecast | Process | --> | stage_order_detail_sales | Process |
| load_forecast | Process | --> | stage_order_detail_sales2 | Process |
| load_order_header | Process | --> | stage_order_detail_sales | Process |
| load_order_header | Process | --> | stage_order_detail_sales2 | Process |
| load_order_header_partner | Process | --> | stage_order_detail_sales | Process |
| load_order_header_partner | Process | --> | stage_order_detail_sales2 | Process |
| load_order_header_sales | Process | --> | stage_order_detail_sales | Process |
| load_order_header_sales | Process | --> | stage_order_detail_sales2 | Process |
| load_order_line | Process | --> | stage_order_detail_sales | Process |
| load_order_line | Process | --> | stage_order_detail_sales2 | Process |
| load_order_line_partner | Process | --> | stage_order_detail_sales | Process |
| load_order_line_partner | Process | --> | stage_order_detail_sales2 | Process |
| load_order_line_sales2 | Process | --> | stage_order_detail_sales | Process |
| load_order_line_sales2 | Process | --> | stage_order_detail_sales2 | Process |
| load_sales_quota | Process | --> | stage_order_detail_sales | Process |
| load_sales_quota | Process | --> | stage_order_detail_sales2 | Process |
| stage_order_detail_sales | Process | --> | stage_order_detail_1 | Process |
| stage_order_detail_sales2 | Process | --> | stage_order_detail_1 | Process |
| stage_order_detail_1 | Process | --> | stage_currency_rate | Process |
| stage_order_detail_1 | Process | --> | stage_forecast | Process |
| stage_order_detail_1 | Process | --> | stage_order_detail | Process |
| stage_currency_rate | Process | --> | fact_order_detail | Process |
| stage_forecast | Process | --> | fact_order_detail | Process |

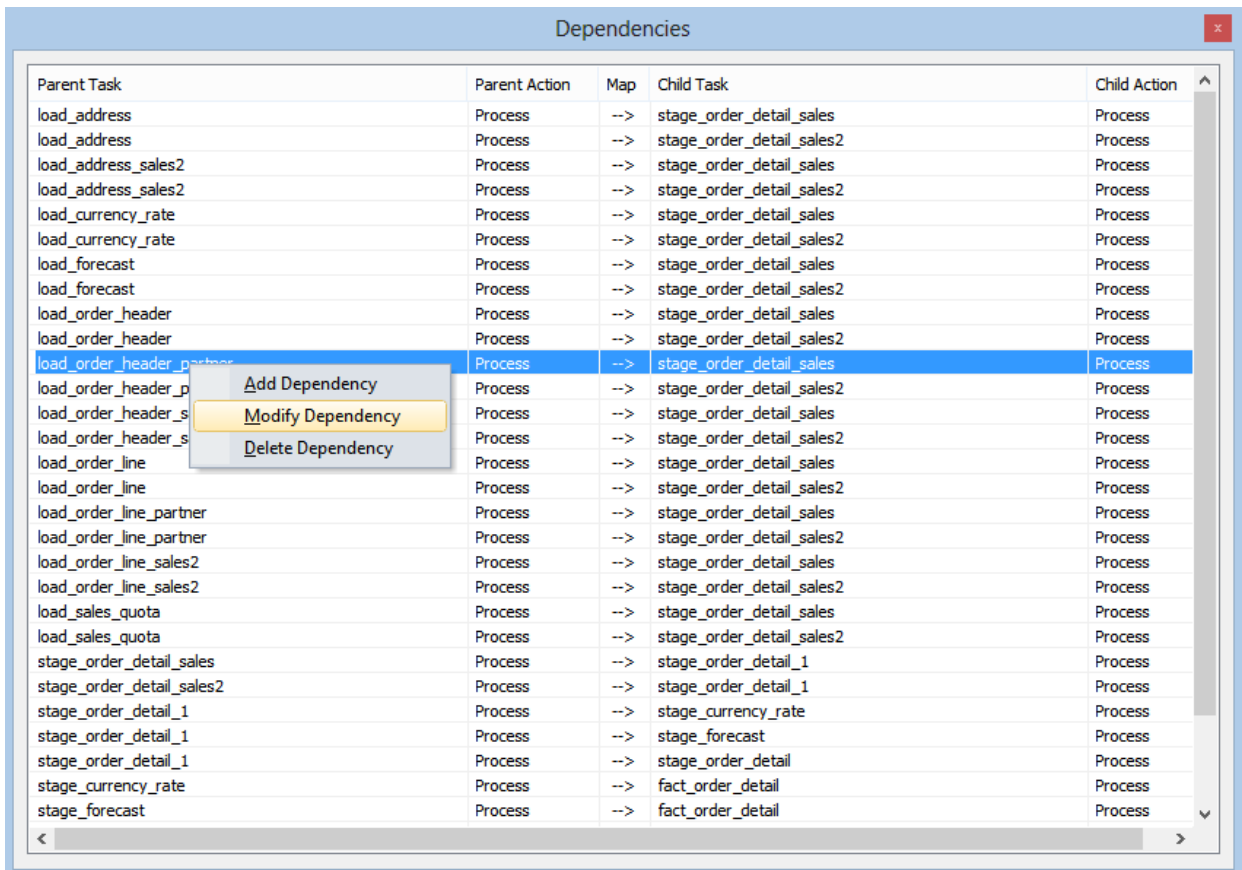
Select the **Parent** and **Child** tasks from the drop-down lists to create the dependency and click **OK**.

Enter the Parent and Child Dependency. The child task will not be actioned until the parent has successfully completed.

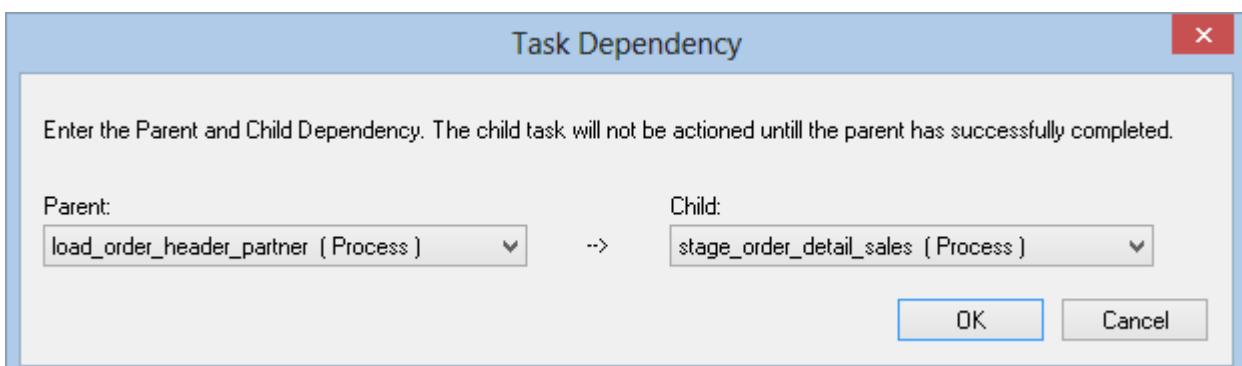
Parent: --> Child:

To modify a task dependency

To modify a task dependency, right-click on the dependency in the Dependencies pane and select **Modify Dependency**.

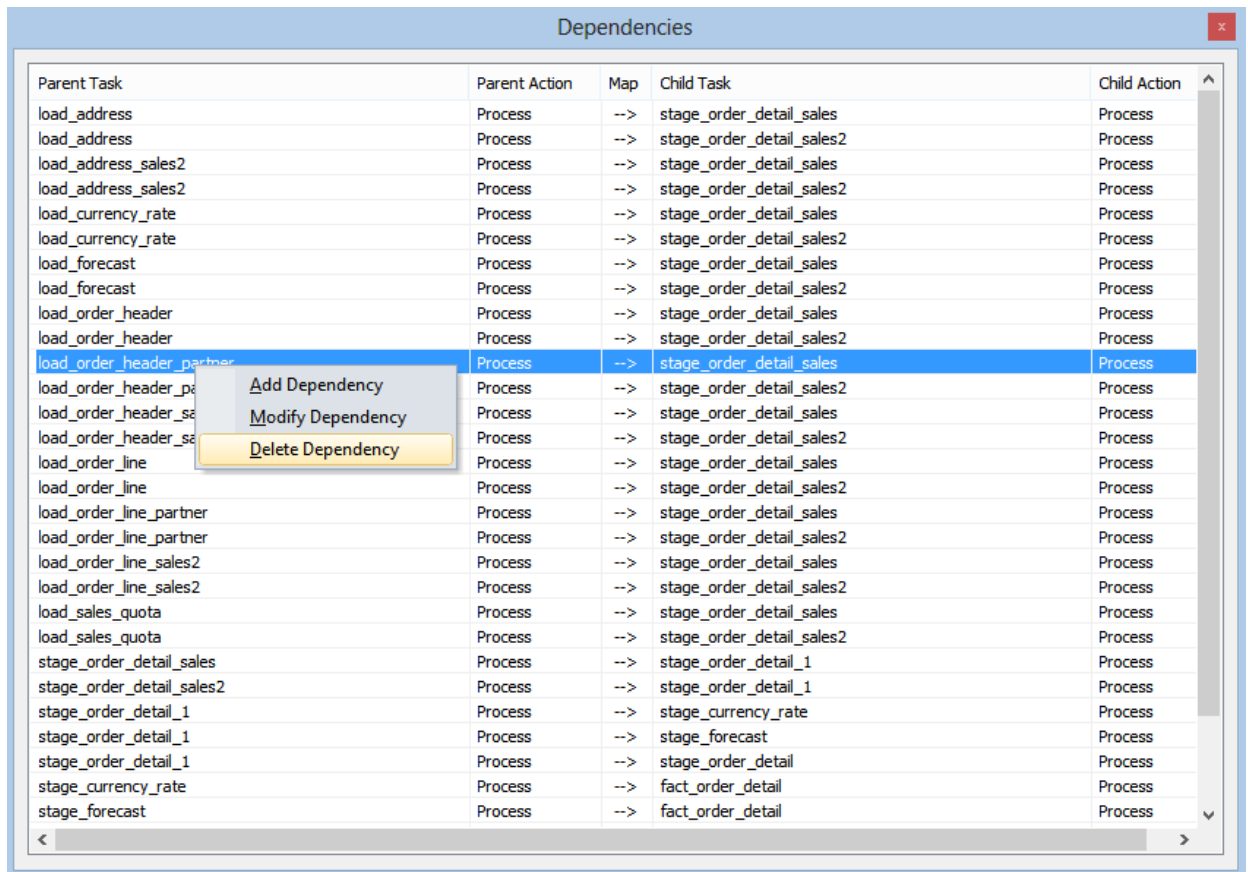


Change the **Parent** and **Child** tasks to modify the dependency and click **OK**.



To delete a task dependency

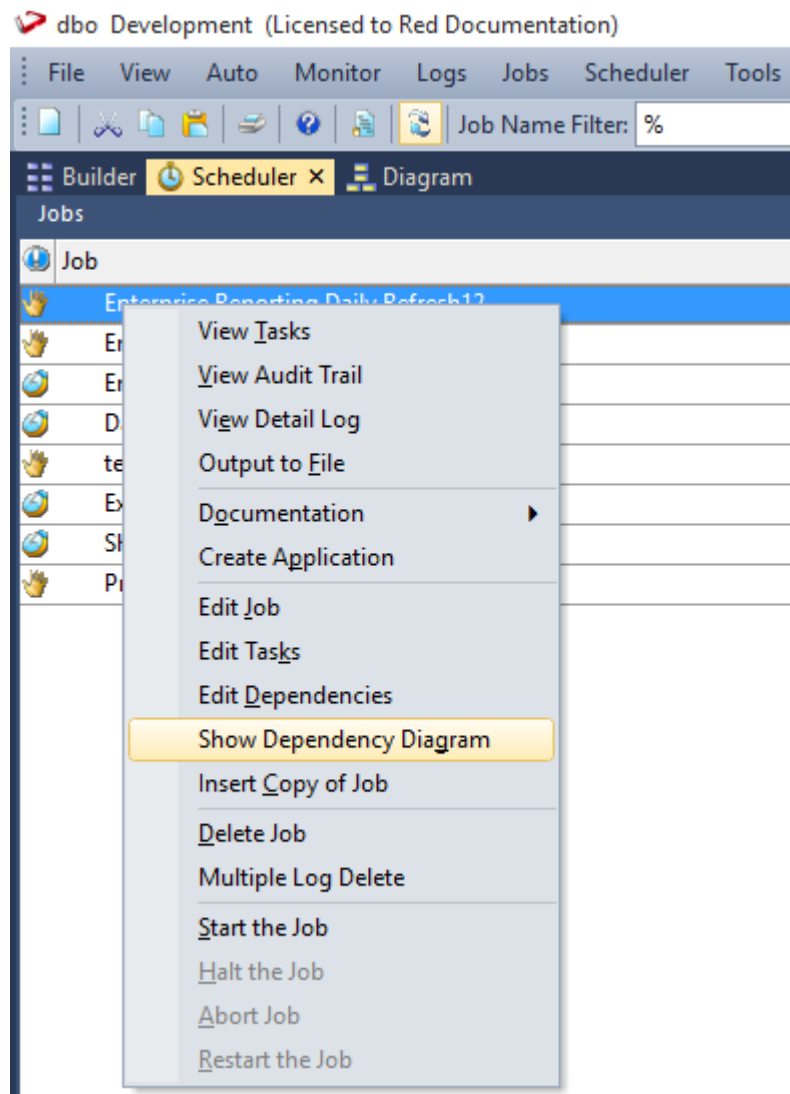
To delete a task dependency, right-click on the dependency in the Dependencies pane and select **Delete Dependency**.



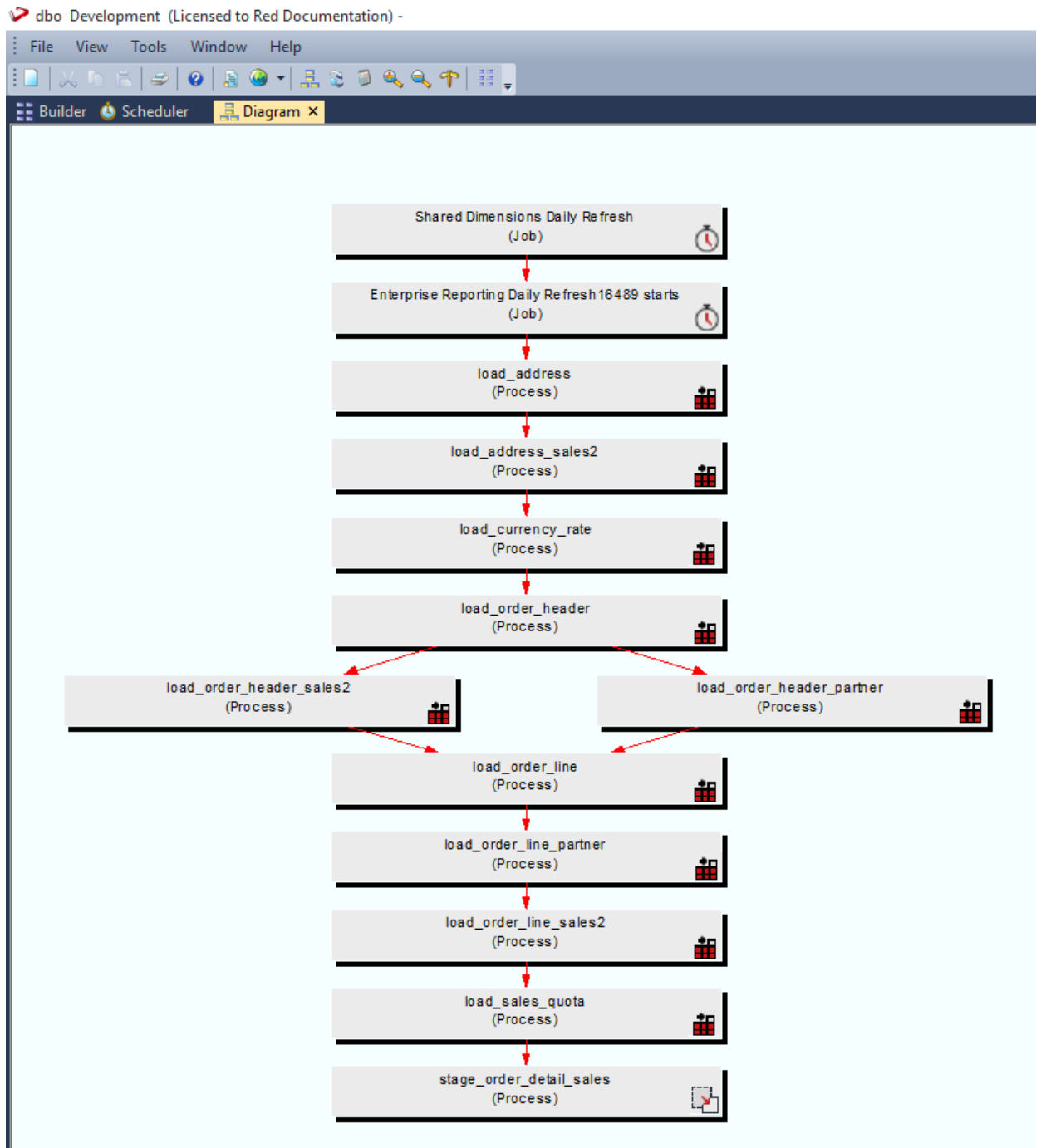
The dependency will be deleted without warning.

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.



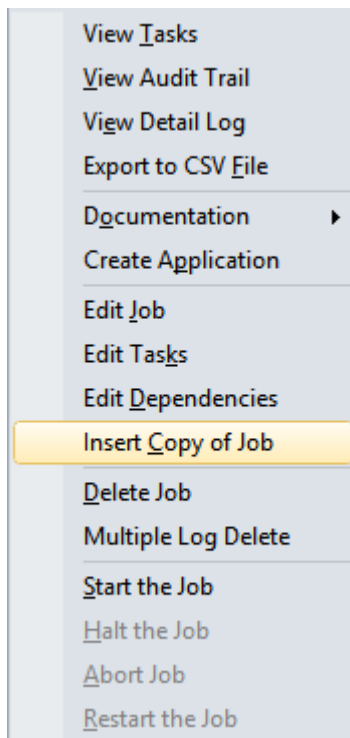
Job Dependency Diagram view



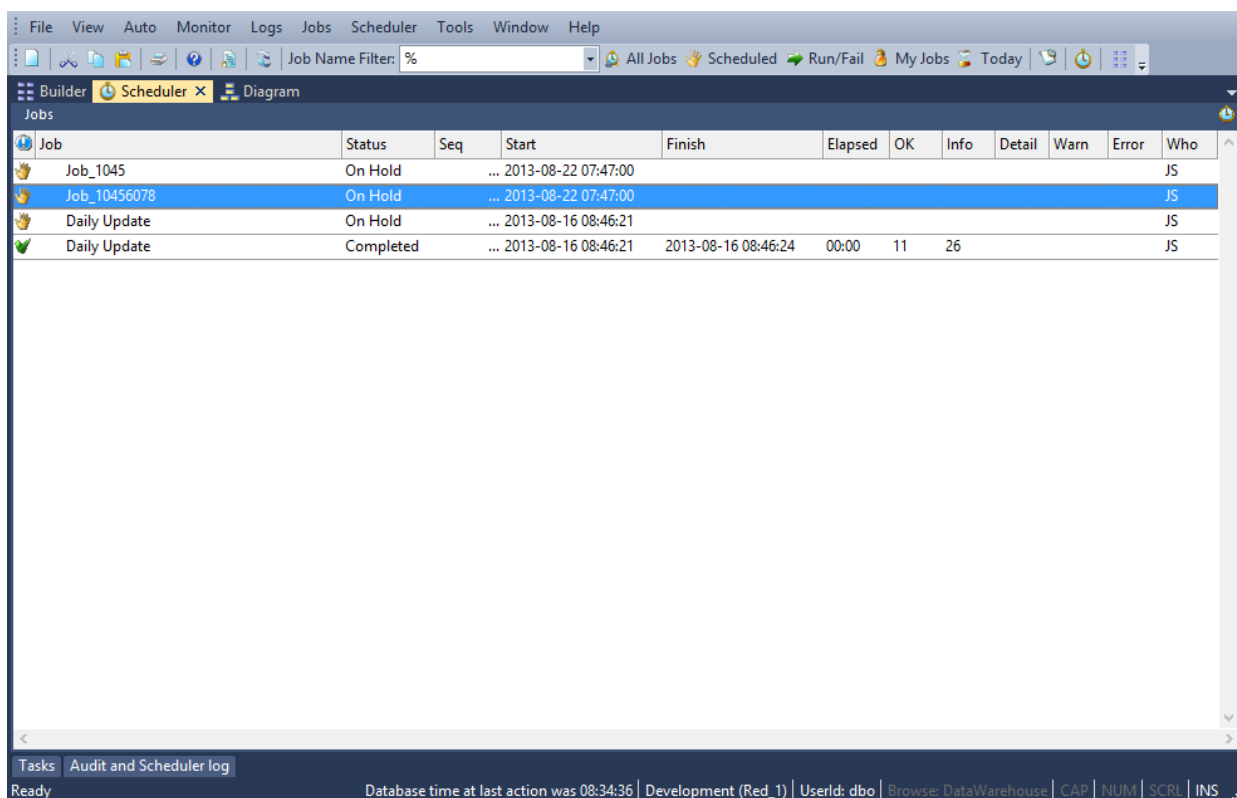
INSERTING A COPY OF A JOB

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**.



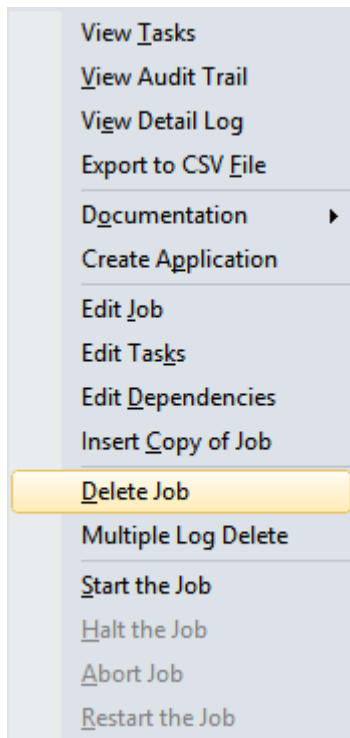
The new job will immediately be visible and the Status will be **On Hold**.



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**.



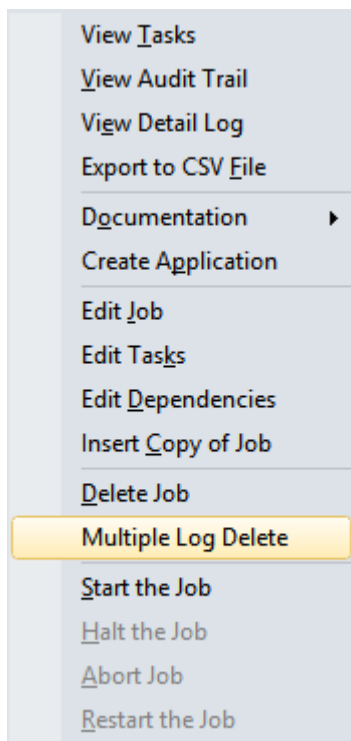
A warning message will be displayed; click **Yes** to delete.



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.

Delete Multiple Job Logs

Multiple Job Logs can be deleted by specifying one or more of the limitations below.

Job Name:
 If no name is specified then all logs are deleted. SQL Wildcards (%) may be used.

Job State:
 Select a specific job state, or All to delete all logs


Job Owner:
 Select a specific job owner, or All to delete all logs.

Days Prior:
 Number of days prior to the current date to start deleting from. (e.g. 14 will delete all logs more than 14 days old).

Archive Audit Trail Logs During Delete

A warning message will be displayed. Click **Yes** to delete.

WhereScape RED

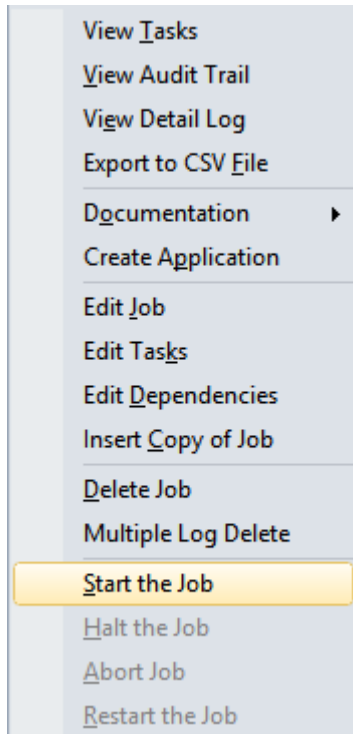
 You are about to delete and archive all job logs for jobs named Process fact_forecast 6718 more than 14 days old

Are you sure you wish to delete these logs?

STARTING A JOB

To start a job

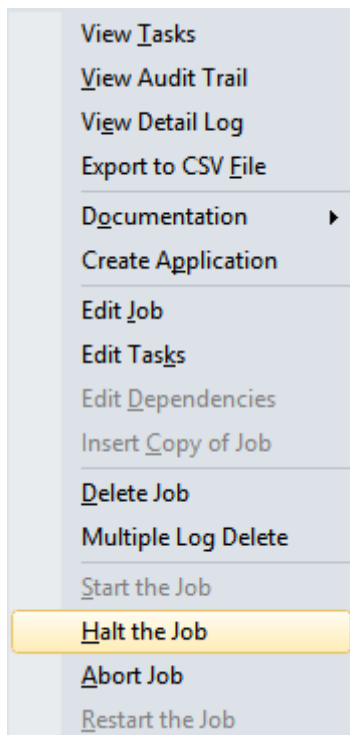
Multiple job logs can be deleted by right-clicking on a job in the scheduler window and choosing **Multiple Log Delete**.



HALTING A JOB

To halt a job

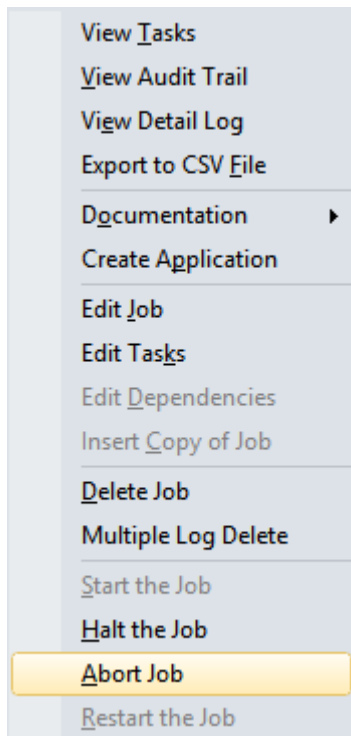
A job can be halted by right-clicking on a job in the scheduler window and choosing **Halt the Job**.



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.

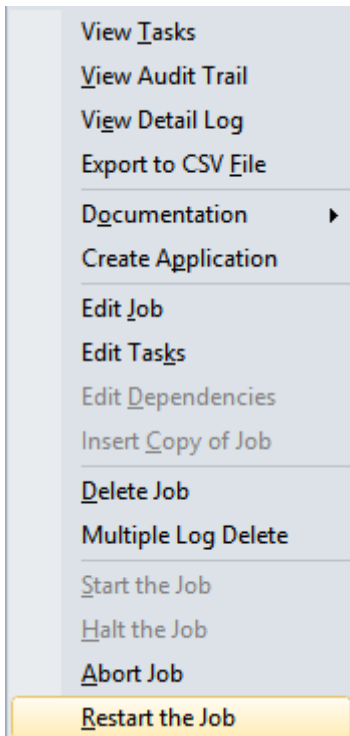
Effects of aborting a job

Load and update processes are not stopped for all objects in Teradata repositories.

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.

| Job | Status | Seq | Start | Fini... | Elapsed | OK | Info | Detail | Warn | Error | Who |
|------------------------------------|-----------|------|---------------------|---------|---------|----|------|--------|------|-------|-----|
| Process fact_forecast 6718 | On Hold | 6718 | 2013-02-14 15:36:00 | | | | | | | | JS |
| Refresh Order Details | On Hold | 7248 | 2013-02-14 12:39:36 | | | | | | | | JS |
| Shared Dimensions Daily Refresh | On Hold | 1150 | 2013-02-14 03:00:00 | | | | | | | | WQ |
| Enterprise Reporting Daily Refresh | On Hold | 6280 | 2013-01-31 17:18:32 | | | | | | | | JS |
| Process_to_fact_order_detail | On Hold | 6730 | 2013-01-31 16:20:00 | | | | | | | | JS |
| Refresh Order Details | Failed | 7258 | 2013-02-14 12:39:36 | 201... | 00:00 | 3 | 11 | | | 3 | JS |
| Refresh Order Details | Completed | 7256 | 2013-02-14 12:29:05 | 201... | 00:00 | 5 | 12 | | | | JS |

| Task | Action | Status | Seq | Start | Finish | Ela... | l... | D... | W... | Result |
|--------------------|---------|-----------|------|----------------|----------------|--------|------|------|------|---|
| load_order_head... | Process | Completed | 7258 | 2013-02-14 ... | 2013-02-14 ... | 00:... | 4 | | | 9 rows loaded into load_order_header |
| load_order_line | Process | Completed | 7258 | 2013-02-14 ... | 2013-02-14 ... | 00:... | 4 | | | 21 rows loaded into load_order_line |
| stage_order_detail | Process | Completed | 7258 | 2013-02-14 ... | 2013-02-14 ... | 00:... | 2 | | | stage_order_detail updated. 21 new records. 0 records up... |
| fact_order_detail | Process | Failed | 7258 | 2013-02-14 ... | 2013-02-14 ... | 00:... | 1 | | | Ws_Act_Update(6.5.5.1) step 1600: update_fact_order_det... |

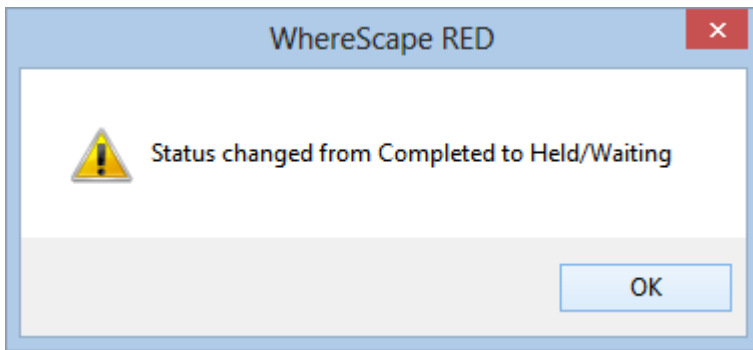
Tasks Audit and Scheduler log
Ready Database time at last action was 13:03:10 | Development (WslWarehouse) | UserId: dbo | Browse: DataWarehouse | CAP | NUM | SCRL | INS

To rerun a task, right-click on the completed task and select **Change to On Hold**.

| Task | Action | Status | Seq | Start | Finish | Ela... | l... | D... | W... | Result |
|--------------------|---------|-----------|------|----------------|----------------|--------|------|------|------|---|
| load_order_head... | Process | Completed | 7258 | 2013-02-14 ... | 2013-02-14 ... | 00:... | 4 | | | 9 rows loaded into load_order_header |
| load_order_line | Process | Completed | 7258 | 2013-02-14 ... | 2013-02-14 ... | 00:... | 4 | | | 21 rows loaded into load_order_line |
| stage_order_detail | Process | Completed | 7258 | 2013-02-14 ... | 2013-02-14 ... | 00:... | 2 | | | stage_order_detail updated. 21 new records. 0 records up... |
| fact_order_detail | Process | Failed | 7258 | 2013-02-14 ... | 2013-02-14 ... | 00:... | 1 | | | Ws_Act_Update(6.5.5.1) step 1600: update_fact_order_det... |

Tasks Audit and Scheduler log
Ready Database time at last action was 13:03:10 | Development (WslWarehouse) | UserId: dbo | Browse: DataWarehouse | CAP | NUM | SCRL | INS

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.

| Task | Action | Status | Seq | Start | Finish | Ela... | I... | D... | W... | Result |
|--------------------|---------|-----------|------|----------------|----------------|--------|------|------|------|---|
| load_order_head... | Process | Completed | 7258 | 2013-02-14 ... | 2013-02-14 ... | 00:... | 4 | | | 9 rows loaded into load_order_header |
| load_order_line | Process | Completed | 7258 | 2013-02-14 ... | 2013-02-14 ... | 00:... | 4 | | | 21 rows loaded into load_order_line |
| fact_order_detail | Process | Failed | 7258 | 2013-02-14 ... | 2013-02-14 ... | 00:... | 1 | | | Ws_Act_Update(6.5.5.1) step 1600: update_fact_order_det... |
| stage_order_detail | Process | Held | 7258 | 2013-02-14 ... | 2013-02-14 ... | 00:... | 2 | | 1 | stage_order_detail updated. 21 new records. 0 records up... |

Tasks | Audit and Scheduler log
Ready Database time at last action was 13:15:07 | Development (WslWarehouse) | UserId: dbo | Browse: DataWarehouse | CAP | NUM | SCRL | INS

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.

| Task | Action | Status | Seq | Start | Finish | Ela... | I... | D... | W... | Result |
|--------------------|---------|-----------|------|----------------|----------------|--------|------|------|------|---|
| load_order_head... | Process | Completed | 7258 | 2013-02-14 ... | 2013-02-14 ... | 00:... | 4 | | | 9 rows loaded into load_order_header |
| load_order_line | Process | Completed | 7258 | 2013-02-14 ... | 2013-02-14 ... | 00:... | 4 | | | 21 rows loaded into load_order_line |
| fact_order_detail | Process | Failed | 7258 | 2013-02-14 ... | 2013-02-14 ... | 00:... | 1 | | | Ws_Act_Update(6.5.5.1) step 1600: update_fact_order_det... |
| stage_order_detail | Process | Held | 7258 | 2013-02-14 ... | 2013-02-14 ... | 00:... | 2 | | 1 | stage_order_detail updated. 21 new records. 0 records up... |

Tasks | Audit and Scheduler log
Ready Database time at last action was 13:15:07 | Development (WslWarehouse) | UserId: dbo | Browse: DataWarehouse | CAP | NUM | SCRL | INS

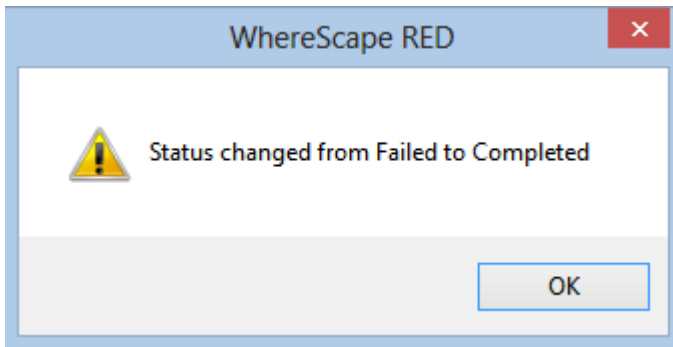
To skip over a task, right-click on the task and select **Change to Completed**.

| Task | Action | Status | Seq | Start | Finish | Elas... | I... | D... | W... | Result |
|----------------------|---------|-----------|------|----------------|----------------|---------|------|------|------|---|
| load_order_head... | Process | Completed | 7258 | 2013-02-14 ... | 2013-02-14 ... | 00:... | 4 | | | 9 rows loaded into load_order_header |
| load_order_line | Process | Completed | 7258 | 2013-02-14 ... | 2013-02-14 ... | 00:... | 4 | | | 21 rows loaded into load_order_line |
| fact_order_detail | Process | Failed | 7258 | 2013-02-14 ... | 2013-02-14 ... | 00:... | 1 | | | Ws_Act_Update(6.5.5.1) step 1600: update_fact_order_det... |
| stage_order_detail P | | | | 4 ... | 2013-02-14 ... | 00:... | 2 | 1 | | stage_order_detail updated. 21 new records. 0 records up... |

Tasks | Audit and Scheduler log

Ready | Database time at last action was 13:16:06 | Development (WslWarehouse) | UserId: dbo | Browse: DataWarehouse | CAP | NUM | SCRL | INS

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.

| Jobs | | | | | | | | | | | | |
|------------------------------------|-----------|------|---------------------|--------|---------|----|------|--------|------|-------|-----|--|
| Job | Status | Seq | Start | Fin... | Elapsed | OK | Info | Detail | Warn | Error | Who | |
| Process fact_forecast 6718 | On Hold | 6718 | 2013-02-14 15:36:00 | | | | | | | | JS | |
| Refresh Order Details | On Hold | 7248 | 2013-02-14 12:39:36 | | | | | | | | JS | |
| Shared Dimensions Daily Refresh | On Hold | 1150 | 2013-02-14 03:00:00 | | | | | | | | WQ | |
| Enterprise Reporting Daily Refresh | On Hold | 6280 | 2013-01-31 17:18:32 | | | | | | | | JS | |
| Process_to_fact_order_detail | On Hold | 6730 | 2013-01-31 16:20:00 | | | | | | | | JS | |
| Refresh Order Details | Failed | 7258 | 2013-02-14 12:39:36 | 201... | 00:00 | 3 | 11 | | | 3 | JS | |
| Refresh Order Details | Completed | 7256 | 2013-02-14 12:29:05 | 201... | 00:00 | 5 | 12 | | | | JS | |

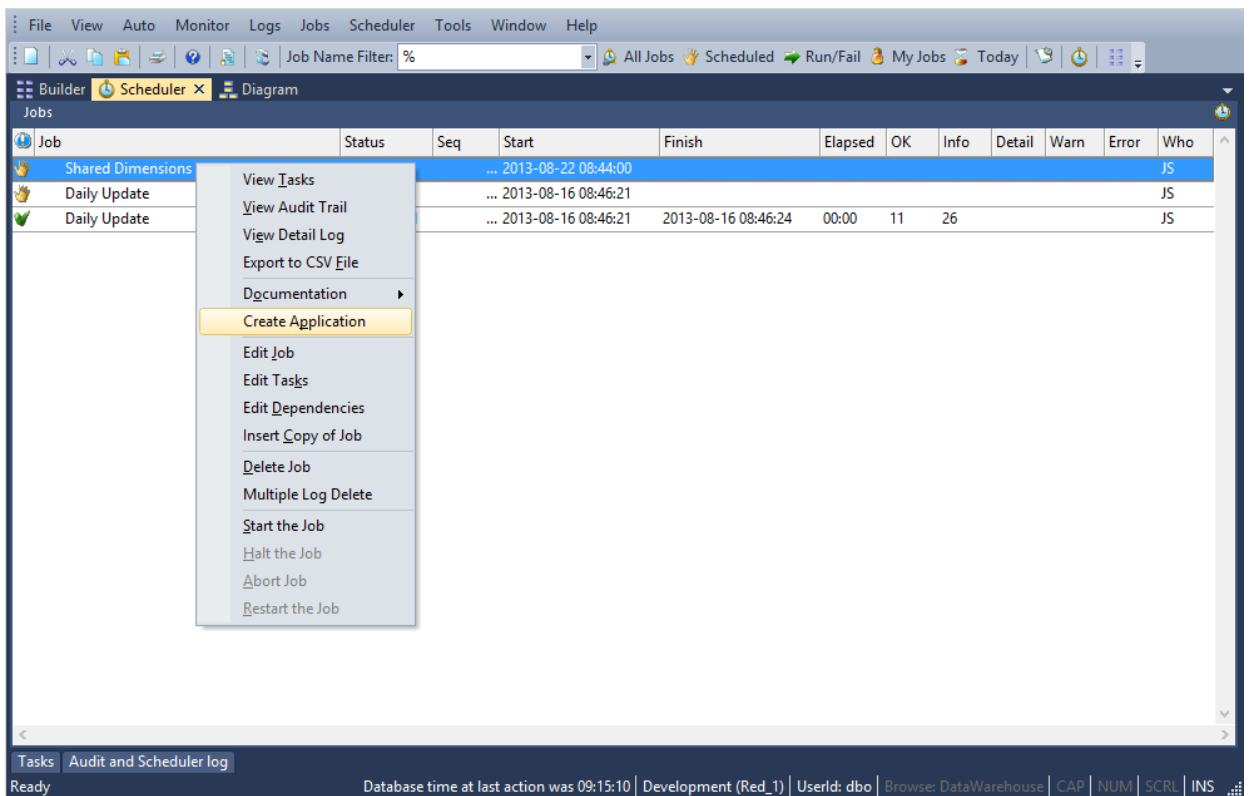
| Tasks | | | | | | | | | | | | |
|--------------------|---------|-----------|------|----------------|----------------|--------|------|------|------|---|--|--|
| Task | Action | Status | Seq | Start | Finish | Ela... | I... | D... | W... | Result | | |
| load_order_head... | Process | Completed | 7258 | 2013-02-14 ... | 2013-02-14 ... | 00:... | 4 | | | 9 rows loaded into load_order_header | | |
| load_order_line | Process | Completed | 7258 | 2013-02-14 ... | 2013-02-14 ... | 00:... | 4 | | | 21 rows loaded into load_order_line | | |
| fact_order_detail | Process | Completed | 7258 | 2013-02-14 ... | 2013-02-14 ... | 00:... | 1 | 1 | | Ws_Act_Update(6.5.5.1) step 1600: update_fact_order_det... | | |
| stage_order_detail | Process | Held | 7258 | 2013-02-14 ... | 2013-02-14 ... | 00:... | 2 | 1 | | stage_order_detail updated. 21 new records. 0 records up... | | |

Ready Database time at last action was 13:18:12 | Development (WsIWarehouse) | UserId: dbo | Browse: DataWarehouse | CAP | NUM | SCRL | INS

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.

Build Deployment Application

Application

Objects to Add/Replace

Objects to Delete

This process builds the files necessary to allow the deployment of a data warehouse solution. These files are read and processed by the Setup Administrator utility. Specify the application identification details and then select the objects to be deployed to and/or deleted from another repository.

Output Directory: C:\Training\RED\Module17\ Browse...

Application Identifier: Shared_Dimen Application Version: 160303140037

Application Name: Application files created from Job Shared Dimensions Daily Refresh

Description: These application files contain the objects included in, and the job Shared Dimensions Daily Refresh

You can select or enter a previous application file as a starting point for this application.

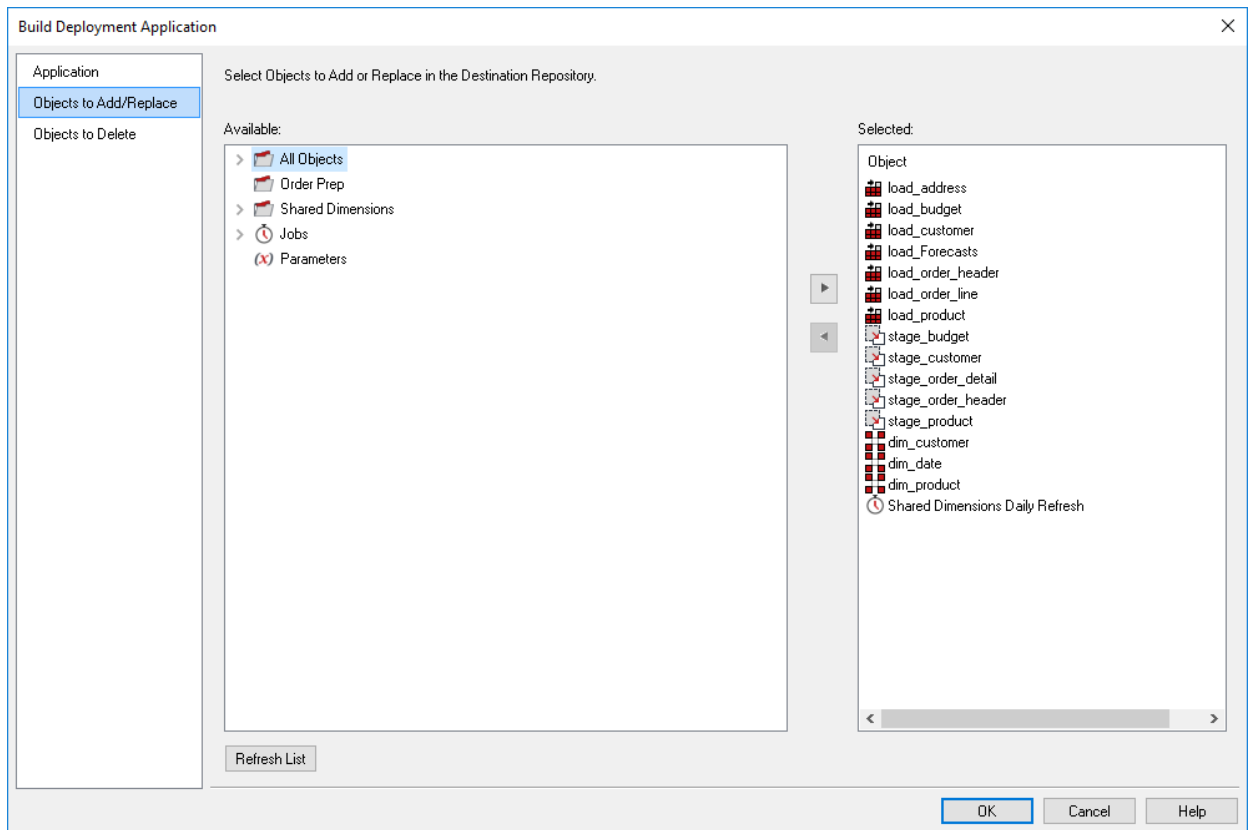
Previous Application: Browse...

Pre Application Load SQL. (the following optional SQL statement will be issued before the application load commences):

Post Application Load SQL. (the following optional SQL statement will be issued after the application load completes):

OK Cancel Help

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**.

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 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.

WhereScape®
RED

Version 6.8.5.2 by WhereScape Software Limited
Copyright (C) 2016
Licensed to RED Documentation
For WhereScape employee use only

REPOSITORY LOGIN

DATABASE

Data Source: TD_15_00

Logon Method: Teradata Wallet

Database Login ID: dssdemo

TD Wallet String: Your TD Wallet string

METADATA REPOSITORY

RED Database: dssdemo

RED User Name: John Smith

HELP CANCEL CONNECT

Scheduler maintenance grants

| Statement | Reason |
|---|----------------------------|
| grant select on ws_dbc_connect to dsssched; | Repository access |
| grant select on ws_meta to dsssched; | Repository access |
| grant select on ws_meta_tables to dsssched; | Repository access |
| grant select on ws_meta_names to dsssched; | Repository access |
| grant select on ws_obj_type to dsssched; | Repository access |
| grant select on ws_obj_object to dsssched; | Object access (job create) |
| grant select on ws_obj_pro_map to dsssched; | Object access (job create) |
| grant select on ws_obj_project to dsssched; | Object access (job create) |

| Statement | Reason |
|--|------------------------------------|
| grant select on ws_obj_group to dsssched; | Object access (job create) |
| grant select on ws_pro_gro_map to dsssched; | Object access (job create) |
| grant select on ws_wrk_audit_log to dsssched; | Scheduler status |
| | |
| grant select,insert,update on ws_user_adm to dsssched; | Repository access |
| grant select,delete on ws_wrk_error_log to dsssched; | Scheduler status, and job deletion |
| grant select,update on ws_wrk_scheduler to dsssched; | Scheduler status, poll |
| | |
| grant select,insert,update,delete on ws_wrk_dependency to dsssched; | Job creation, maintenance |
| grant select,insert,update,delete on ws_wrk_job_ctrl to dsssched; | Job creation, maintenance |
| grant select,insert,delete on ws_wrk_job_log to dssched; | Job maintenance |
| grant select,update,delete on ws_wrk_job_run to dssched; | Job maintenance |
| grant select,insert,update,delete on ws_wrk_dependency to dssched; | Job maintenance |
| grant select,insert,update,delete on ws_wrk_job_dependency to dssched; | Job maintenance |
| grant select,delete on ws_wrk_job_thread to dssched; | Job maintenance |
| grant select,insert on ws_wrk_sequence to dssched; | Job creation |
| grant select,insert,update,delete on ws_wrk_task_ctrl to dssched; | Task maintenance |
| grant select,update,delete on ws_wrk_task_run to dssched; | Task maintenance |
| grant select,insert,delete on ws_wrk_task_log to dssched; | Task maintenance |
| grant select,insert,update on dss_parameter to dssched; | Task maintenance |

| Statement | Reason |
|--|---|
| grant select on ws_pro_header to dsssched; | Right-click used by option in parameters listing |
| grant select on ws_pro_line to dsssched; | Right-click used by option in parameters listing |
| grant select on ws_scr_header to dsssched; | Right-click used by option in parameters listing |
| grant select on ws_scr_line to dsssched; | Right-click used by option in parameters listing |
| grant select on ws_load_tab to dsssched; | Right-click used by option in parameters listing |
| grant select on ws_load_col to dsssched; | Right-click used by option in parameters listing |
| grant select on ws_stage_tab to dsssched; | Right-click used by option in parameters listing |
| grant select on ws_stage_col to dsssched; | Right-click used by option in parameters listing |
| grant select on ws_dim_tab to dsssched; | Right-click used by option in parameters listing |
| grant select on ws_dim_col to dsssched; | Right-click used by option in parameters listing |
| grant select on ws_agg_tab to dsssched; | Right-click used by option in parameters listing |
| grant select on ws_agg_col to dsssched; | Right-click used by option in parameters listing |

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

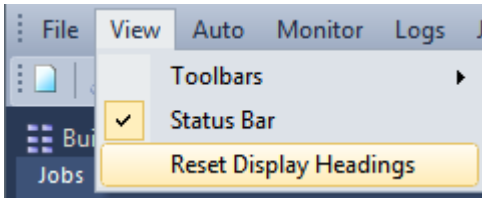
This SQL returns the scheduler status:

```
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_interval/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'
```

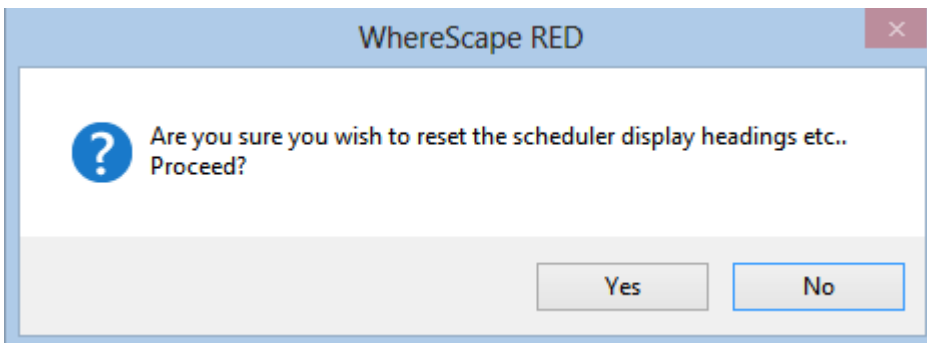
The procedure sets the status in the metadata.

RESET COLUMNS IN JOB AND TASK VIEW

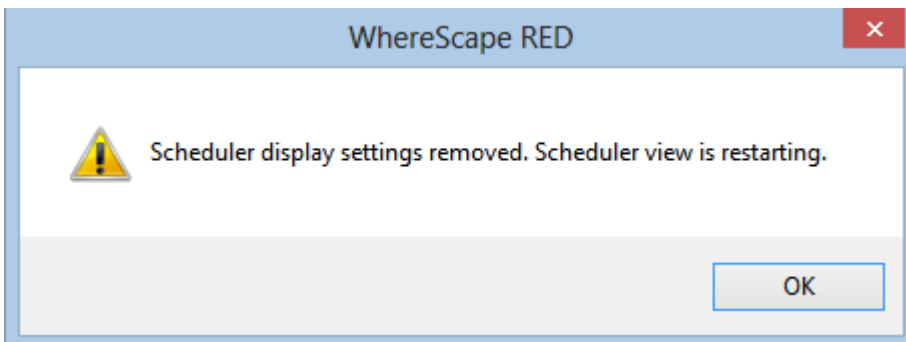
Job and Task Report headings can be reset by selecting the **View/Reset Display Headings** menu option from the scheduler window. The short-cut keys are Alt+V-R.



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.



STOPPING A LINUX/UNIX SCHEDULER FROM WITHIN RED

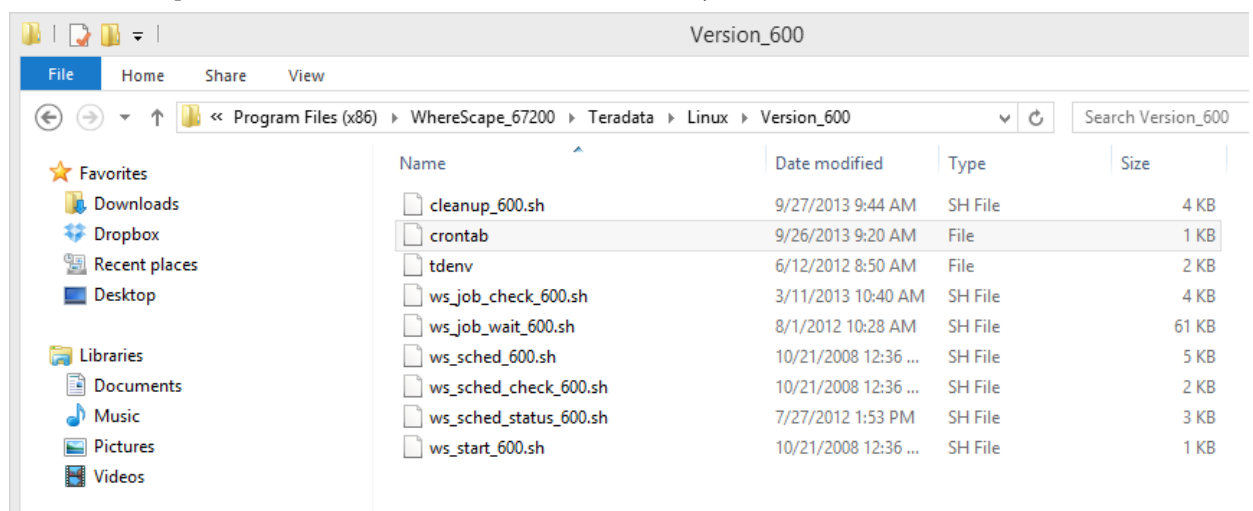
To stop a Linux/UNIX Scheduler from within RED, follow the steps below:

- 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.

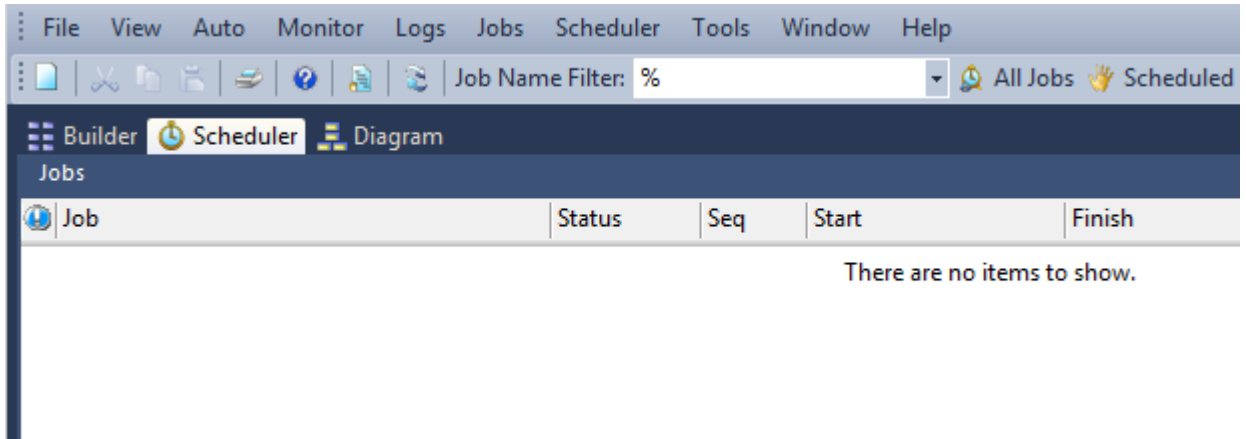


```
File Edit Format View Help
#
## each day cleanup log files etc.
##
#####00 18 * * * _HOME_/wsl/bin/cleanup_600.sh tdenv >/dev/null 2>&1
#
## check and if required start the scheduler
##
#####0,20,40 * * * * _HOME_/wsl/bin/ws_sched_check_600.sh tdenv >>_HOME_/wsl/sched/log/sched_tdenv.log 2>&1
#
## check and if required start the monitor
##
#####0,30 * * * * _HOME_/wsl/bin/ws_mon_check_600.sh tdenv >>_HOME_/wsl/sched/log/mon_tdenv.log 2>&1
```

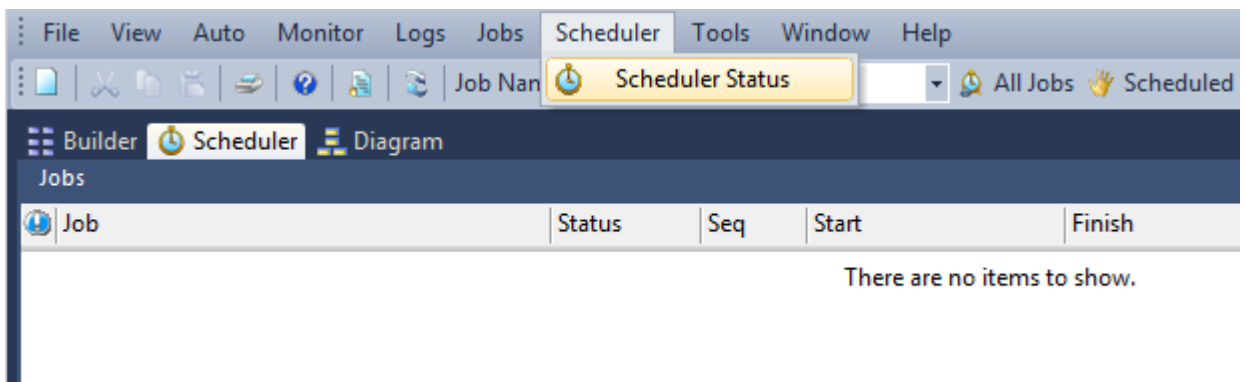
Note: There may be several different versions of the scheduler files for a given database and platform (UNIX or Linux). For example, there may be different folders in `...\\WhereScape\\Teradata\\Linux\\: Version_550` and `Version_600`. The highest version number script less than or equal to the version of RED in use should always be used.



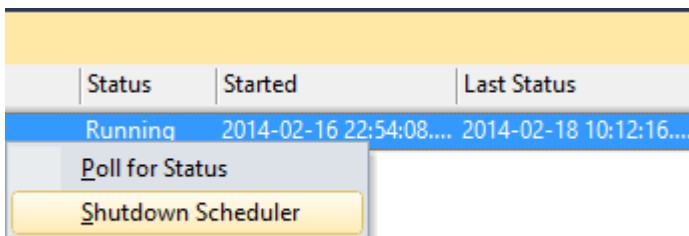
- 2 Start RED and click on the **Scheduler** tab.



- 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 27

INDEXES

Indexes may exist on any table defined in RED. By default, 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

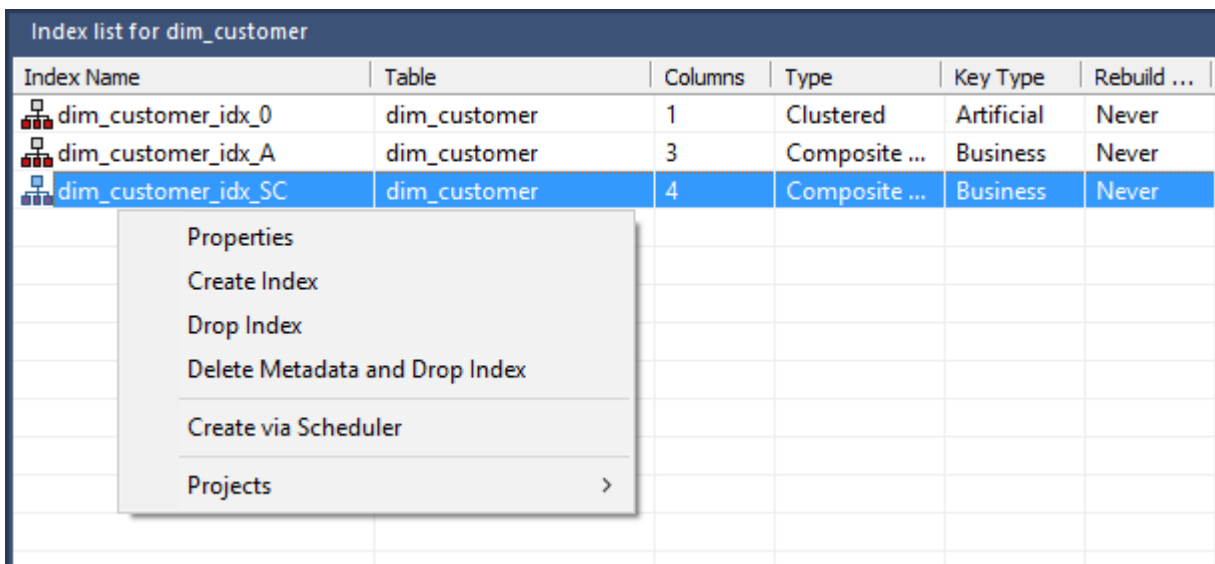
| | |
|-----------------------|-----|
| Index Definition..... | 760 |
|-----------------------|-----|

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



The screenshot shows a table titled "Index list for dim_customer" with the following data:

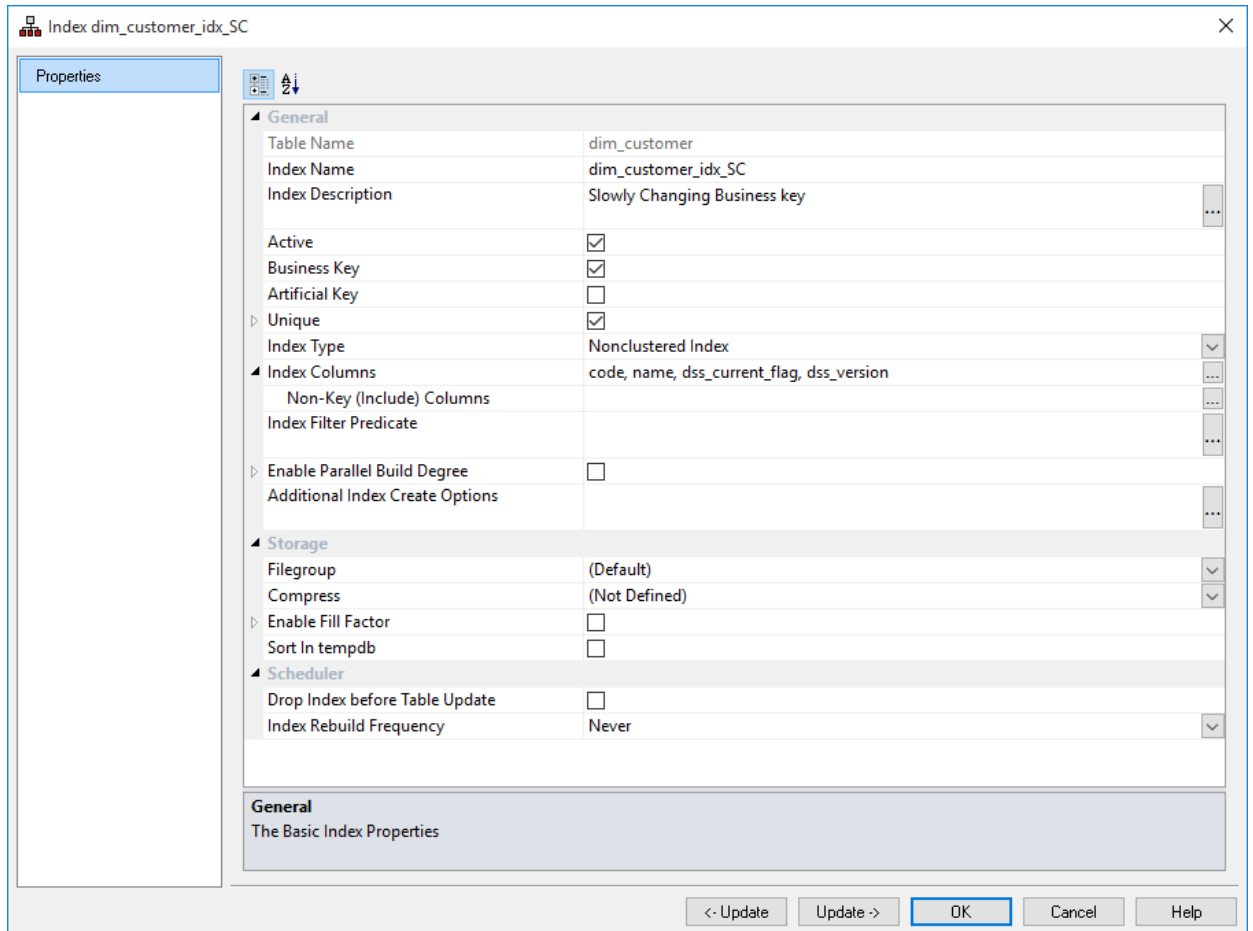
| Index Name | Table | Columns | Type | Key Type | Rebuild ... |
|---------------------|--------------|---------|---------------|------------|-------------|
| dim_customer_idx_0 | dim_customer | 1 | Clustered | Artificial | Never |
| dim_customer_idx_A | dim_customer | 3 | Composite ... | Business | Never |
| dim_customer_idx_SC | dim_customer | 4 | Composite ... | Business | Never |

A context menu is open over the row for "dim_customer_idx_SC", showing the following options:

- 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.



The fields are described below:

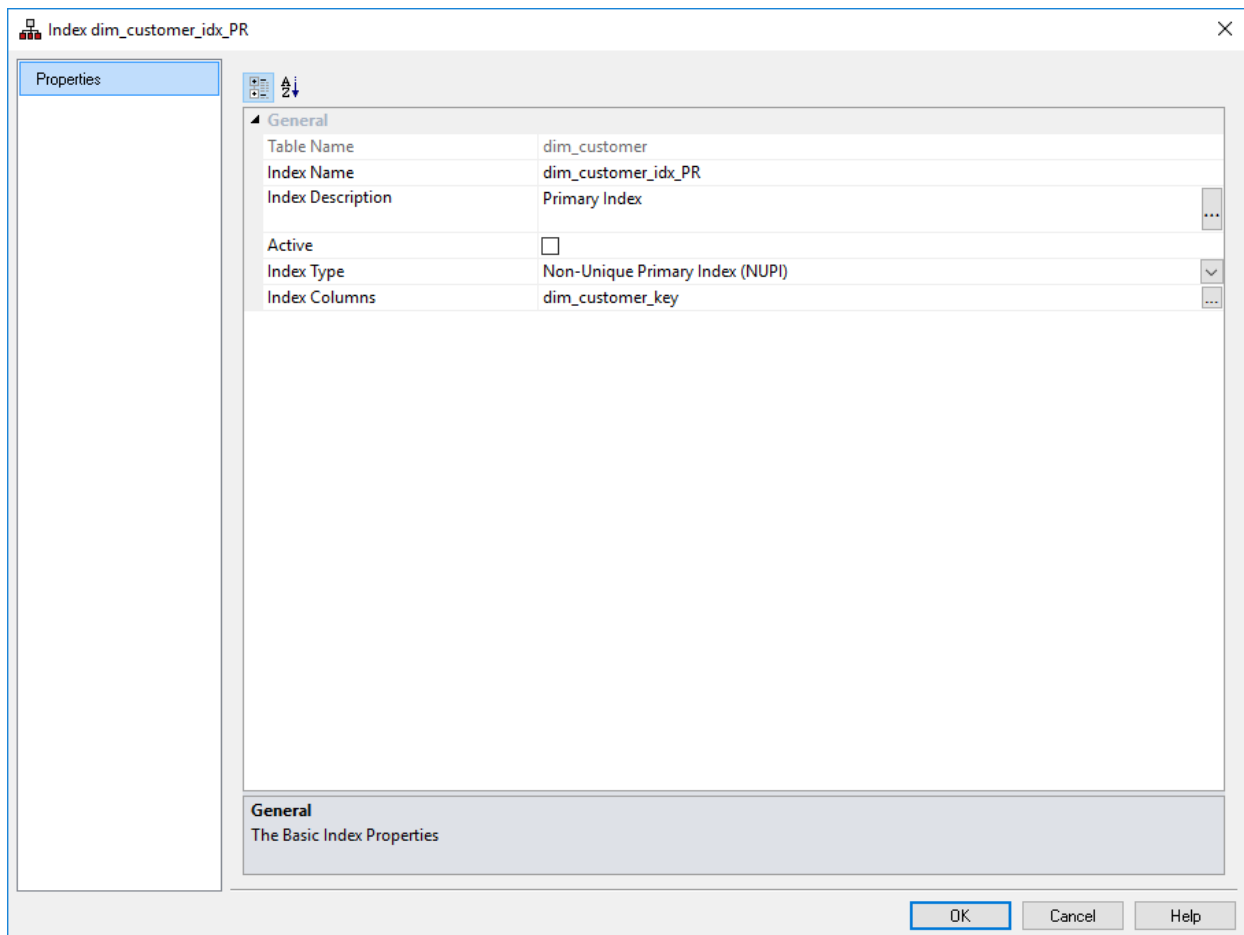
| Field | Description |
|-------------------|--|
| Index name | Typically, the table short name followed by: <ul style="list-style-type: none"> • _idx_0 indicating primary key • _idx_x where x = any letter a to z indicating business keys • _idx_PR indicating primary index |
| Index description | Free flow description of the index |

| Field | Description |
|--------------------|---|
| Rebuild frequency | <p>When the index is rebuilt by the scheduler. Select an option from the drop-down list:</p> <ul style="list-style-type: none"> • Never (default) • Daily • End Month • Start Month • Saturday • Sunday • Monday • Tuesday • Wednesday • Thursday • Friday |
| Active checkbox | <ul style="list-style-type: none"> • When selected means the index is in use. • When not selected means the index is not managed by the scheduler. • The active checkbox on a primary index may not be turned off as all tables in Teradata require a primary index. |
| Artificial Key | When checked indicates that this is the surrogate (artificial) key generated by the system. |
| Business Key | Denotes a business key. |
| Unique | <p>Specifies that the index is a unique index</p> <p>Note: If both unique and artificial are set it is assumed to be a primary key constraint and it is added as such.</p> |
| Primary Index | Specifies that the index is a Primary Index |
| Drop before Update | <p>The index is dropped before the update procedure runs on the table and is reinstated after the update is completed.</p> <p>The Drop before update checkbox on a primary index may not be turned on as all tables in Teradata require a primary index.</p> |
| Hash Index | Defines the index as a Teradata hash index. This limits the hash-ordering to one column, rather than all columns of the index (the default). |
| Index columns | Shows the columns in the order that will be applied to the index. The order can be changed using the up/down buttons on the left. For a primary index without any indexed columns, the table is created as NOT PRIMARY INDEX table. See example below. |

| Field | Description |
|---------------|---|
| Table columns | Shows all columns in the table that can be indexed. These table columns can be added or removed by highlighting the column and checking the appropriate button. |

Indexes are normally managed by the scheduler as part of the normal processing of a table.

Below is an example of a **NO PRIMARY INDEX** index definition:



CHAPTER 28

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 meta data. 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 766) and **read** (see "**Reading the Documentation**" on page 770) the documentation.

IN THIS CHAPTER

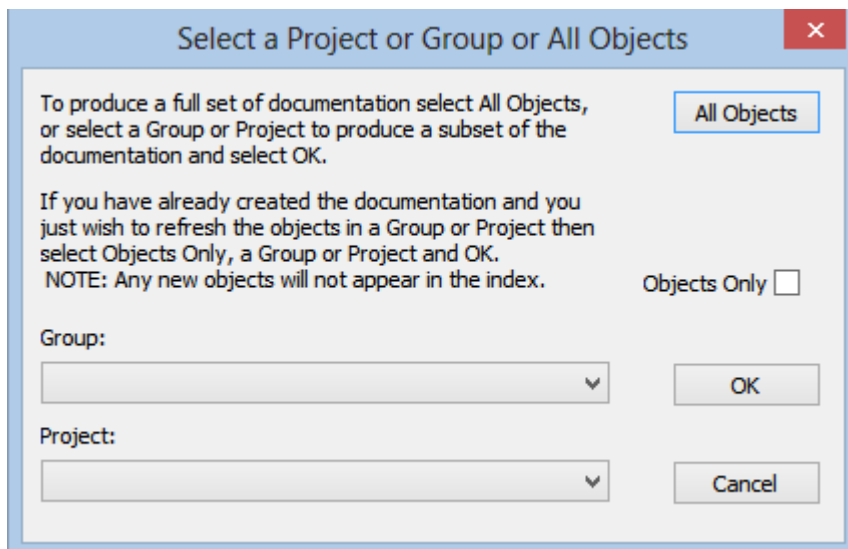
| | |
|-----------------------------------|-----|
| Creating Documentation..... | 766 |
| Batch Documentation Creation..... | 769 |
| Reading the Documentation | 770 |
| Diagrams | 771 |

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**.

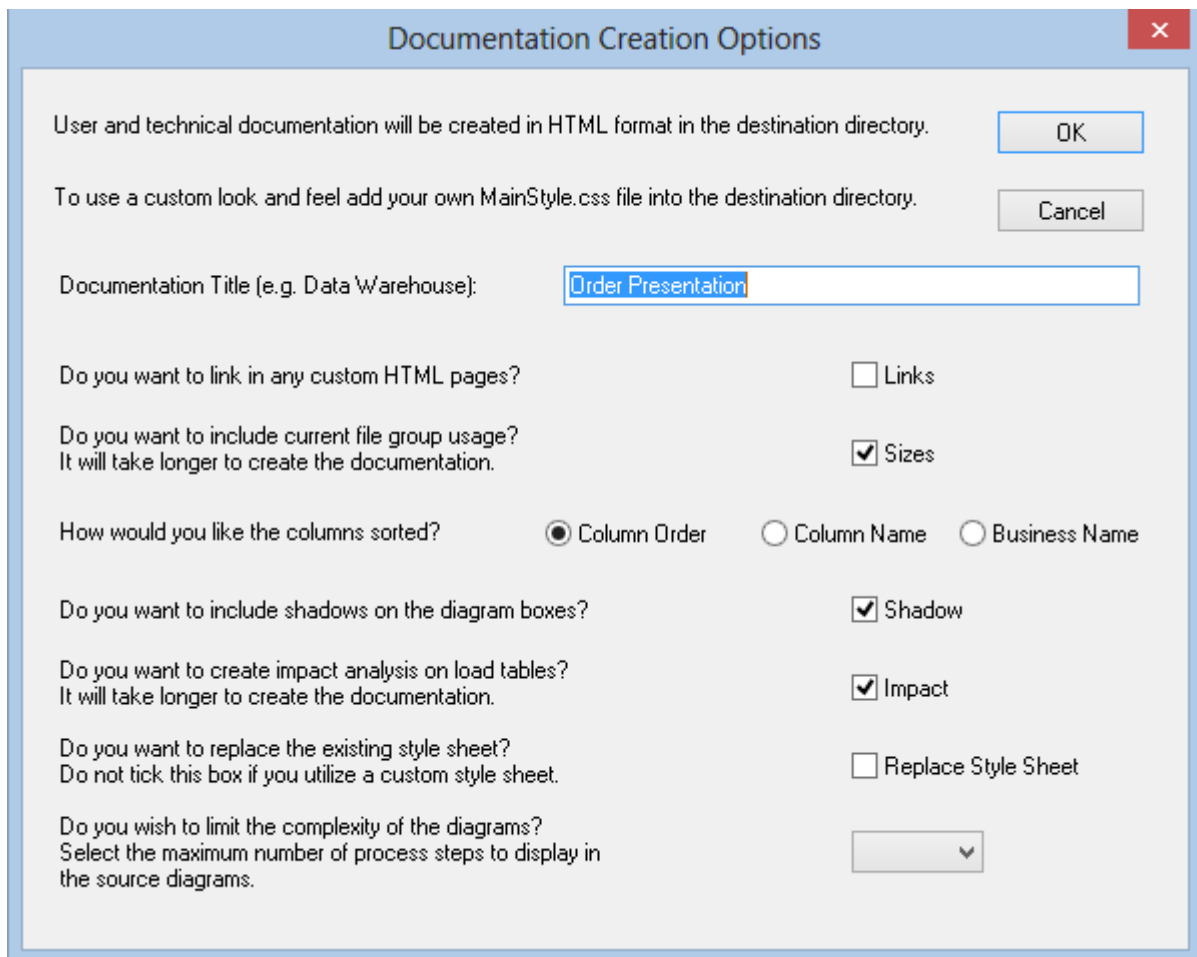
If the repository has projects or groups in use, then the following dialog will appear to allow the selection of a specific group or project. The default is all objects in the repository.



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 Sizes 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, so should normally only be used for final documentation.

The sorted checkbox sorts the columns within the tables into alphabetical order. By default, the columns are in the order that 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 select 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

The screenshot shows a dialog box titled "Document Links" with a close button (X) in the top right corner. The dialog contains the following elements:

- Instruction: "To include any personalized links from the index page complete the details below."
- Text input field: "Please enter the text which will form the html link" with a dropdown arrow.
- Text input field: "Please enter the html filename to link to (Either a filename which is relative to the documentation directory or use the browse button for an absolute filename.)" with a dropdown arrow.
- Buttons: "Finished", "More", "Cancel", "Browse", and "Delete".

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 off the columns in the model 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 MetaDatabase { /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 | Yes | Metadata database 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. |

Note: User Name and Password are not required when using a trusted connection or operating system authentication.

Example

The following example connects to a Teradata repository using the WslWarehouse ODBC data source, a username of Jack, a password of fly1nG, a metadata database of ProdMeta and generates documentation into the C:\Temp\my_doco directory with 4 hops in diagrams:

```
med.exe /BD /UJack /P fly1nG /C "WslWarehouse" /M "ProdMeta" /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 **Read 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.

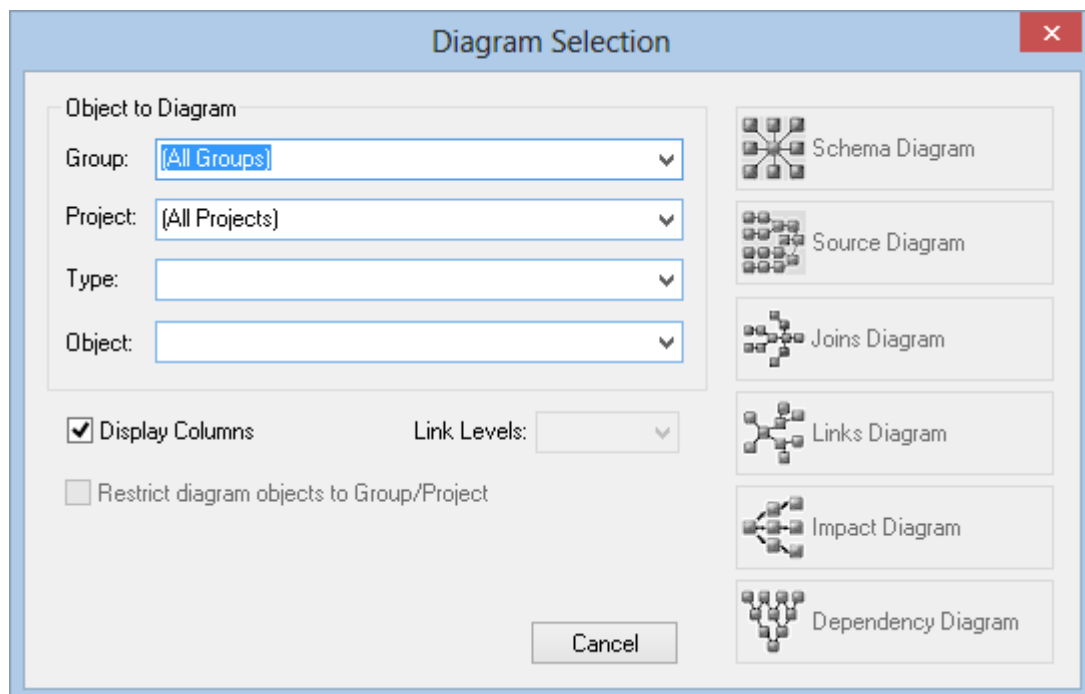
DIAGRAMS

TYPES OF DIAGRAMS

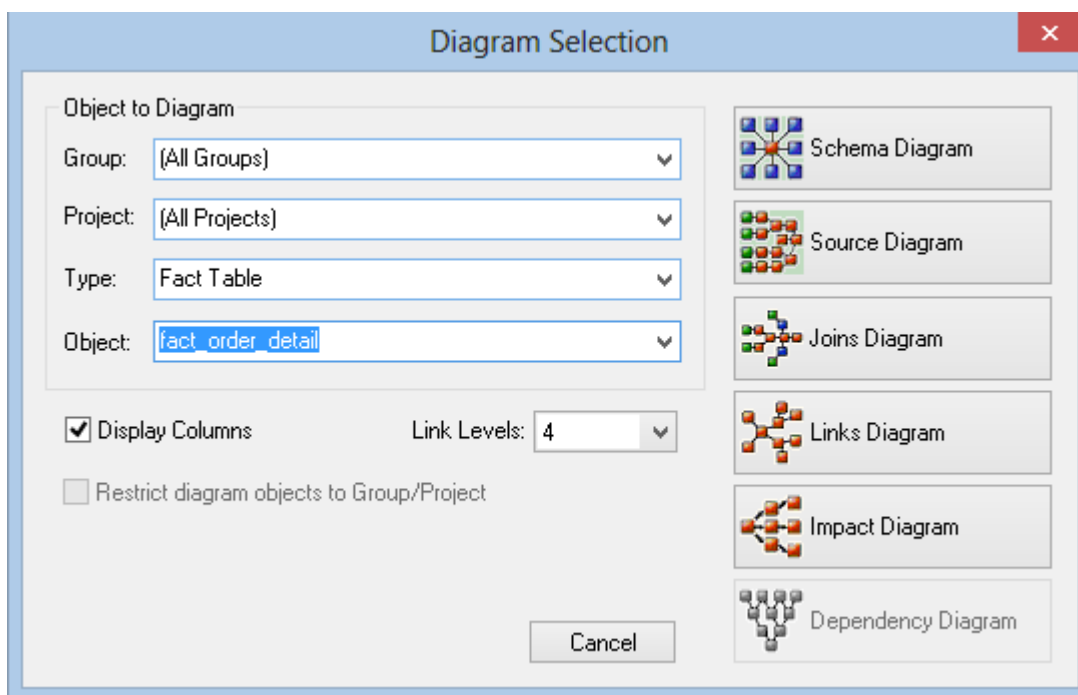
Six types of diagrams are provided to give visual representation of what has been created. These are:

- The **Schema Diagram** (on page 773)
- The **Source Diagram** (on page 775)
- The **Joins Diagram** (on page 779)
- The **Links Diagram** (on page 780)
- The **Impact Diagram** (on page 781)
- The **Dependency Diagram** (on page 783)

- 1 To display the **Diagram Selection** dialog, click on the diagrammatic view button



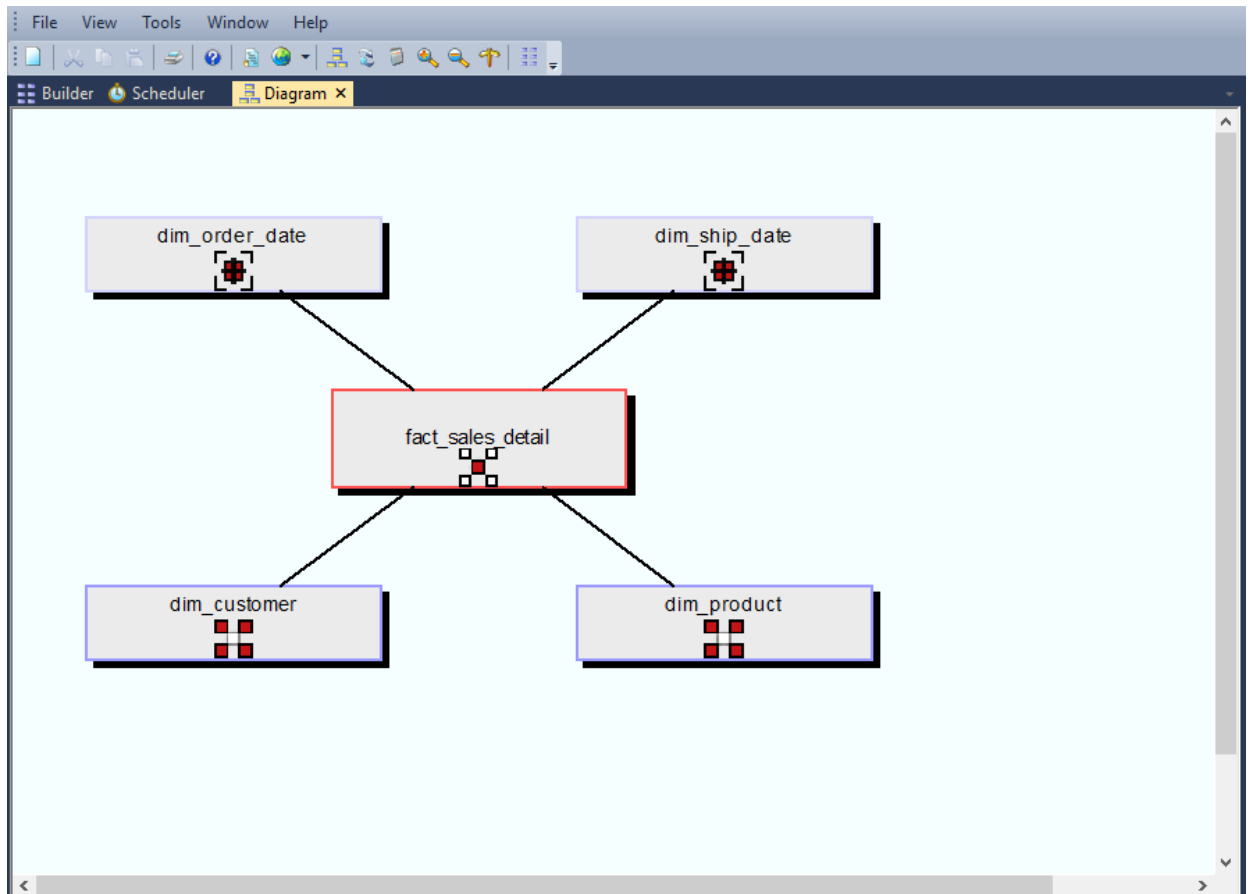
- 2 Choose the object to diagram by optionally choosing the **Type** to limit the selection list; and then selecting the **Object**. The diagram type buttons on the right will then become active and you can choose the type of diagram to display.



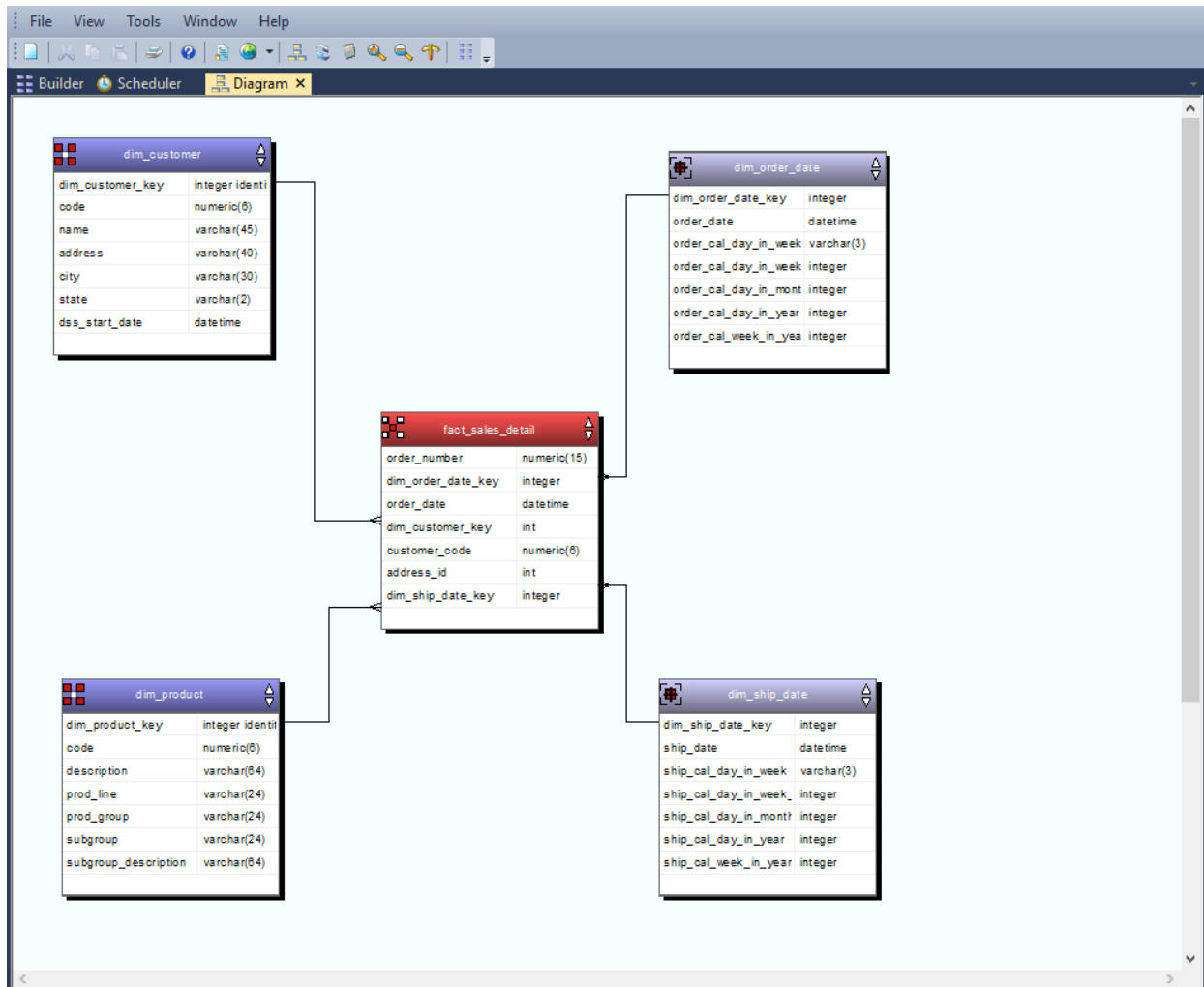
SCHEMA DIAGRAM

A star schema diagram can be displayed for a fact_table, aggregate table, fact view or 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.



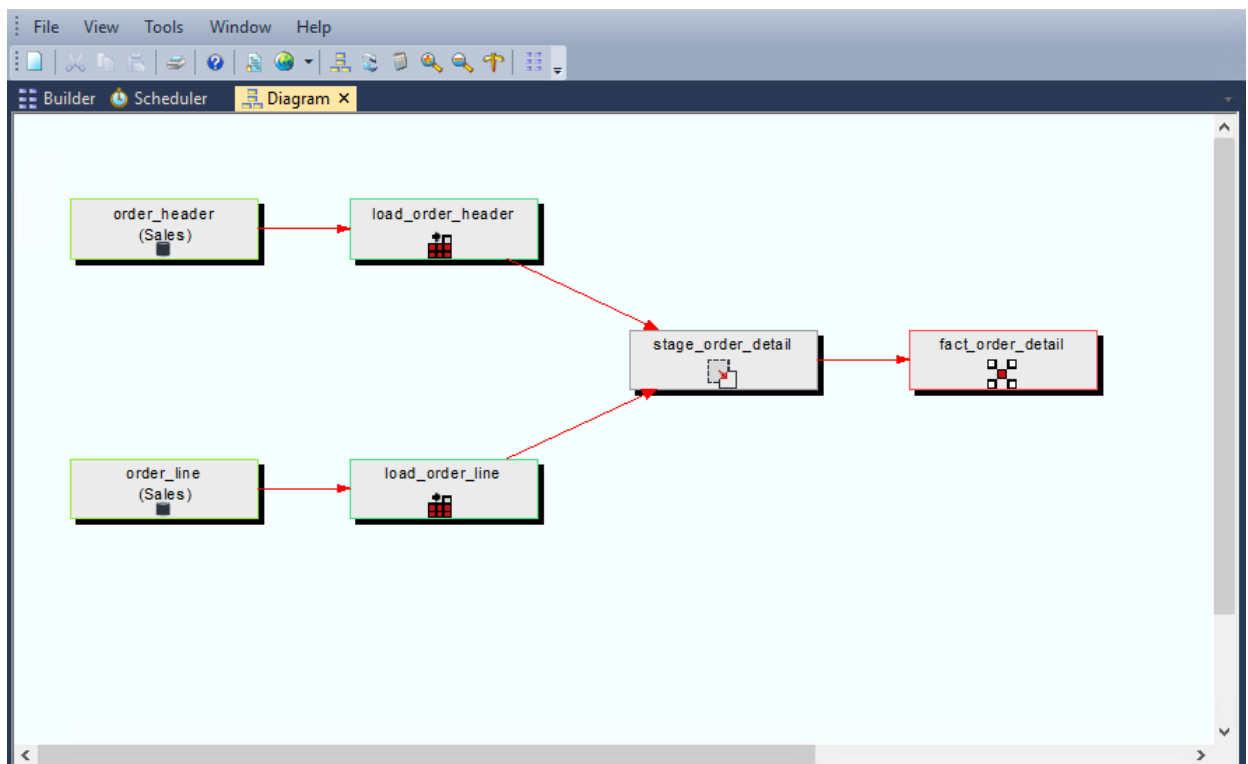
Note: A star schema diagram for a fact view will display only the selected fact view and related dimension views.

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.

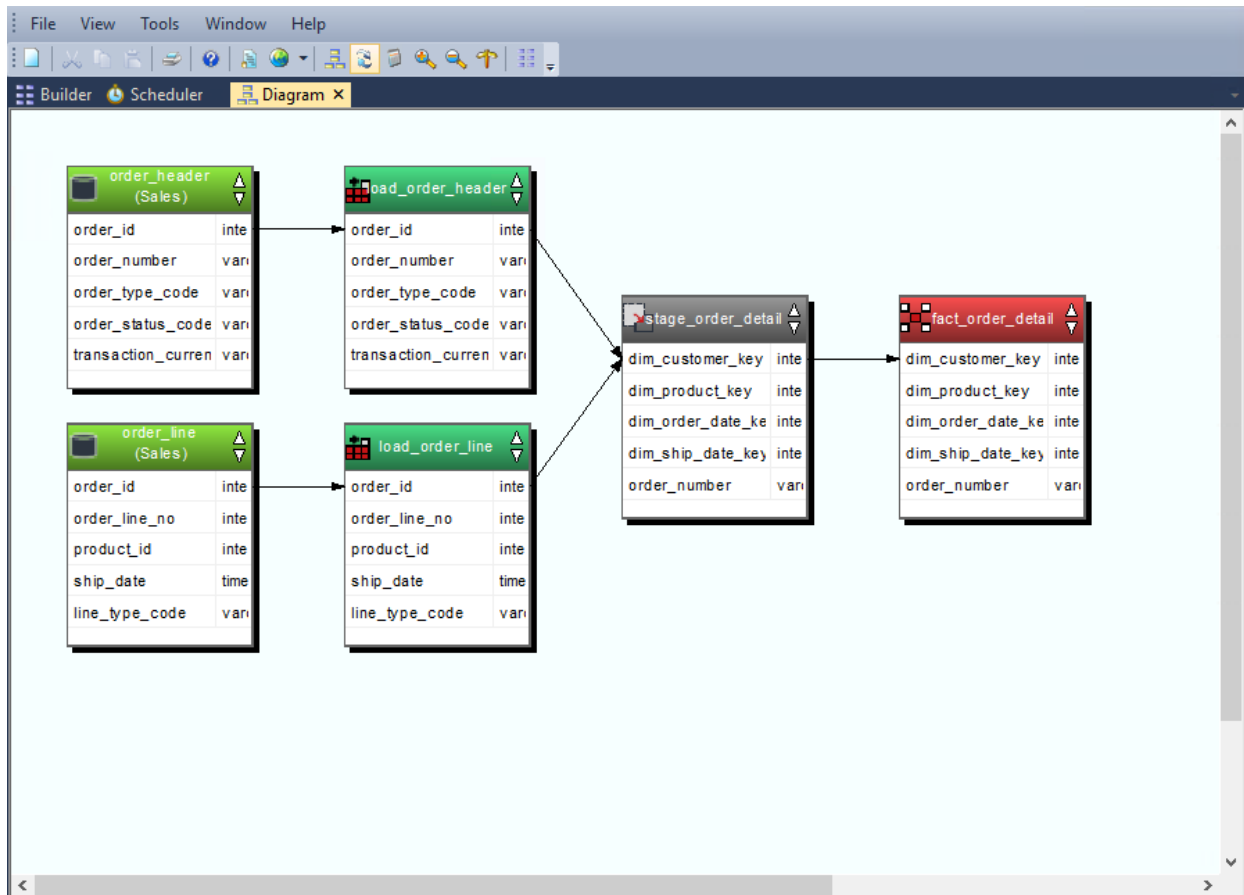
SOURCE DIAGRAM

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.

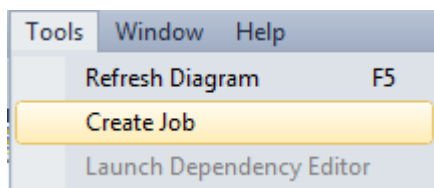


The Source diagrams are produced in **Standard Diagram** format as part of the technical documentation when you select **Doc > Create Documentation** from the main builder window.

Creating a Job from a Source Tracking 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:

Job Definition

Job Name:

Description:

Frequency:

Start Date:

Start Time:

Maximum Threads:

Scheduler:

Dependent On:

| Parent job | Fail | Look back (minutes) | Maximum wait (minutes) |
|------------|------|---------------------|------------------------|
| | | | |

Logs Retained: 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:

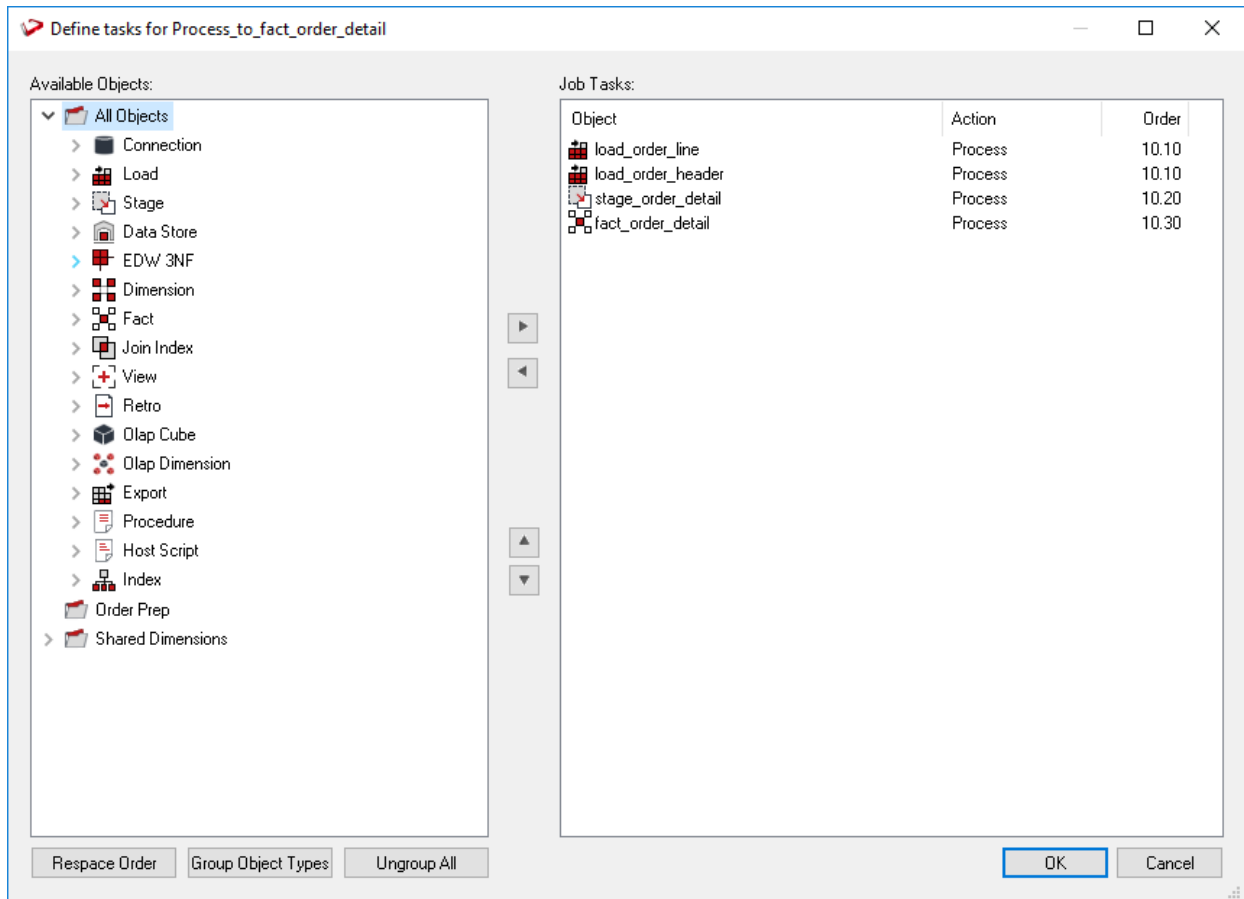
Success Command:

Failure Command:

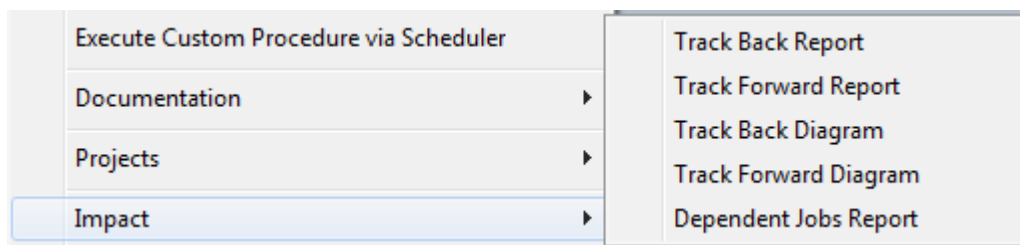
Execute Failure Command in event of dependency failure

Make any changes here that are required and click **OK**.

For the diagram above, a job is created with the following **tasks**:



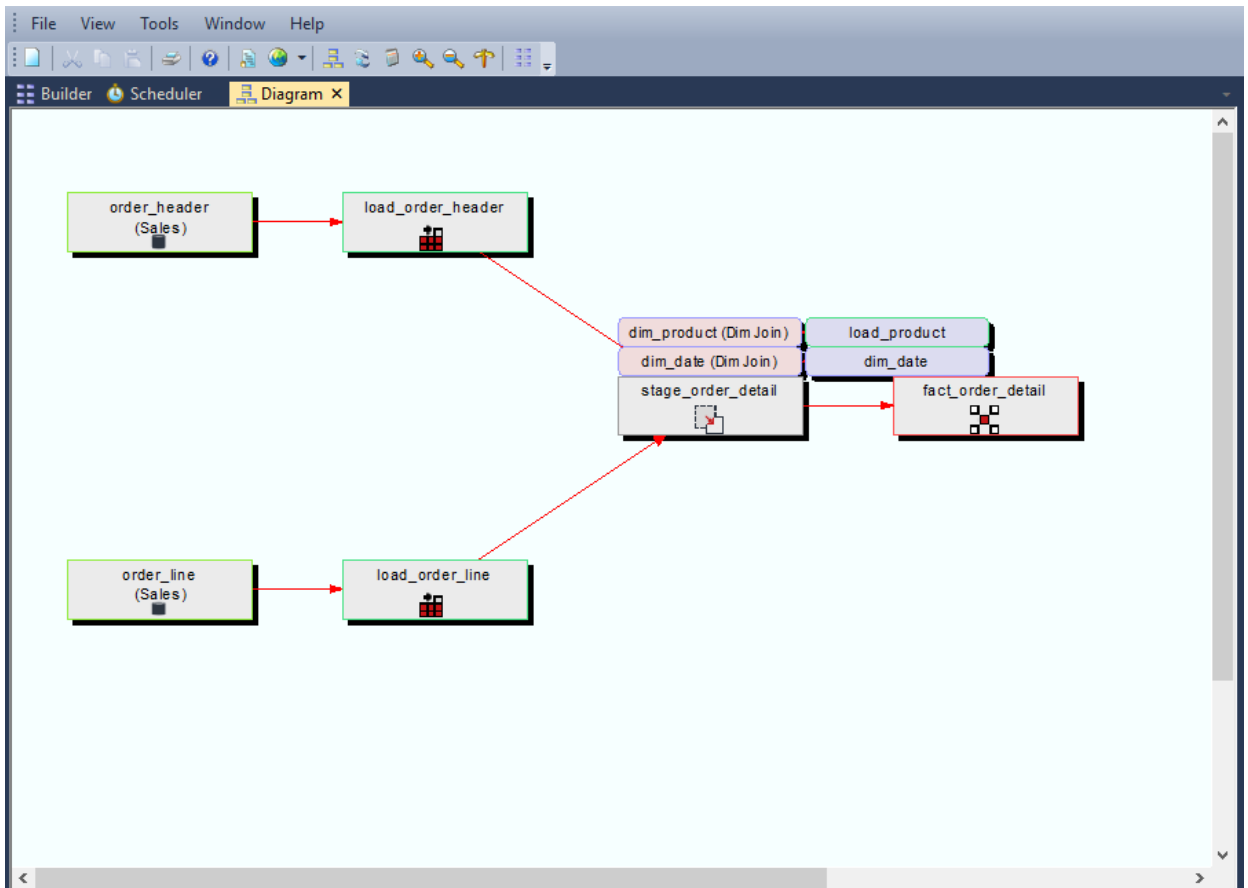
Note: It is also possible to display the source tracking diagram by right-clicking on a table and choosing **Impact/Track Back Diagram**:



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.

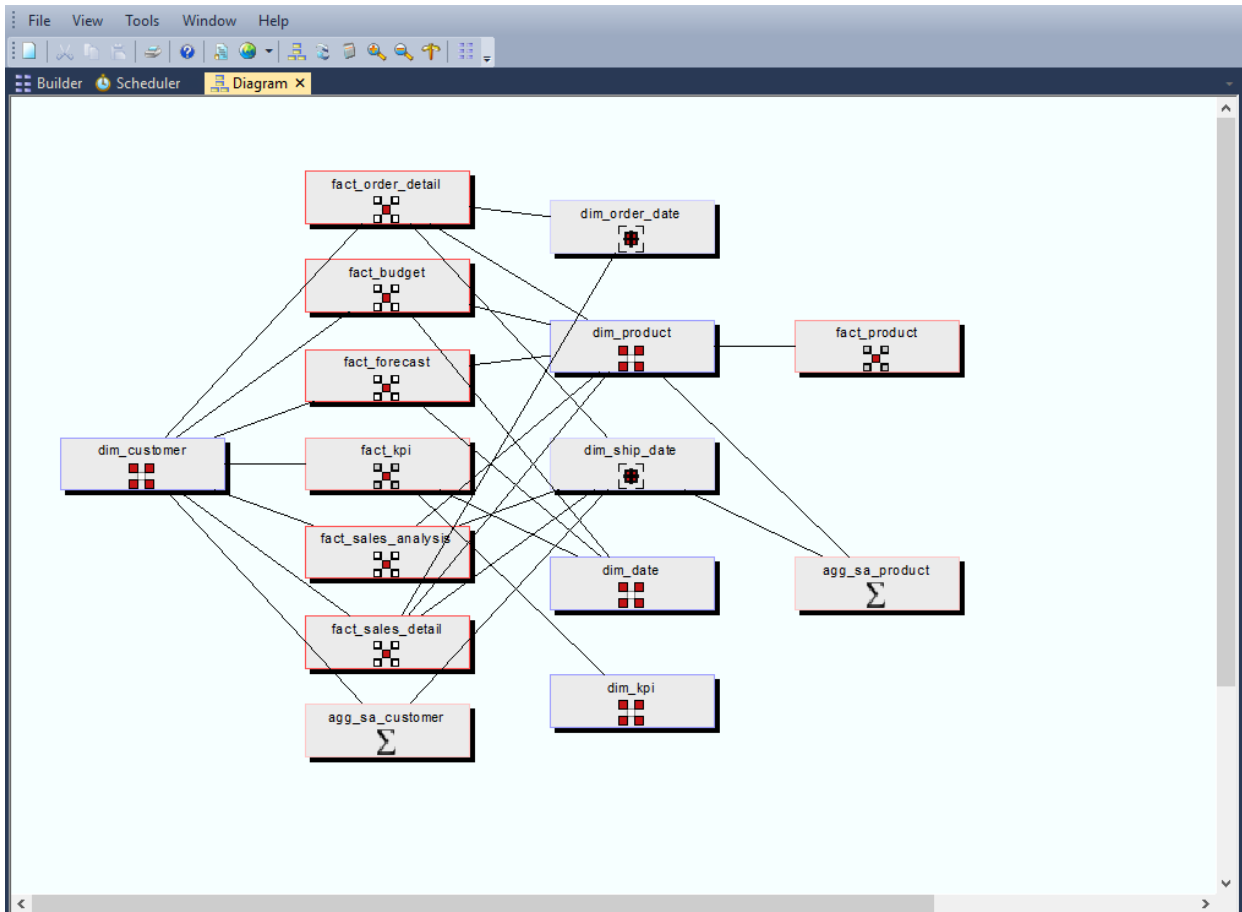
An example of a **Joins** diagram in **Standard Diagram** format is displayed below.



LINKS DIAGRAM

A linked tables diagram can be displayed 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.



Notes:

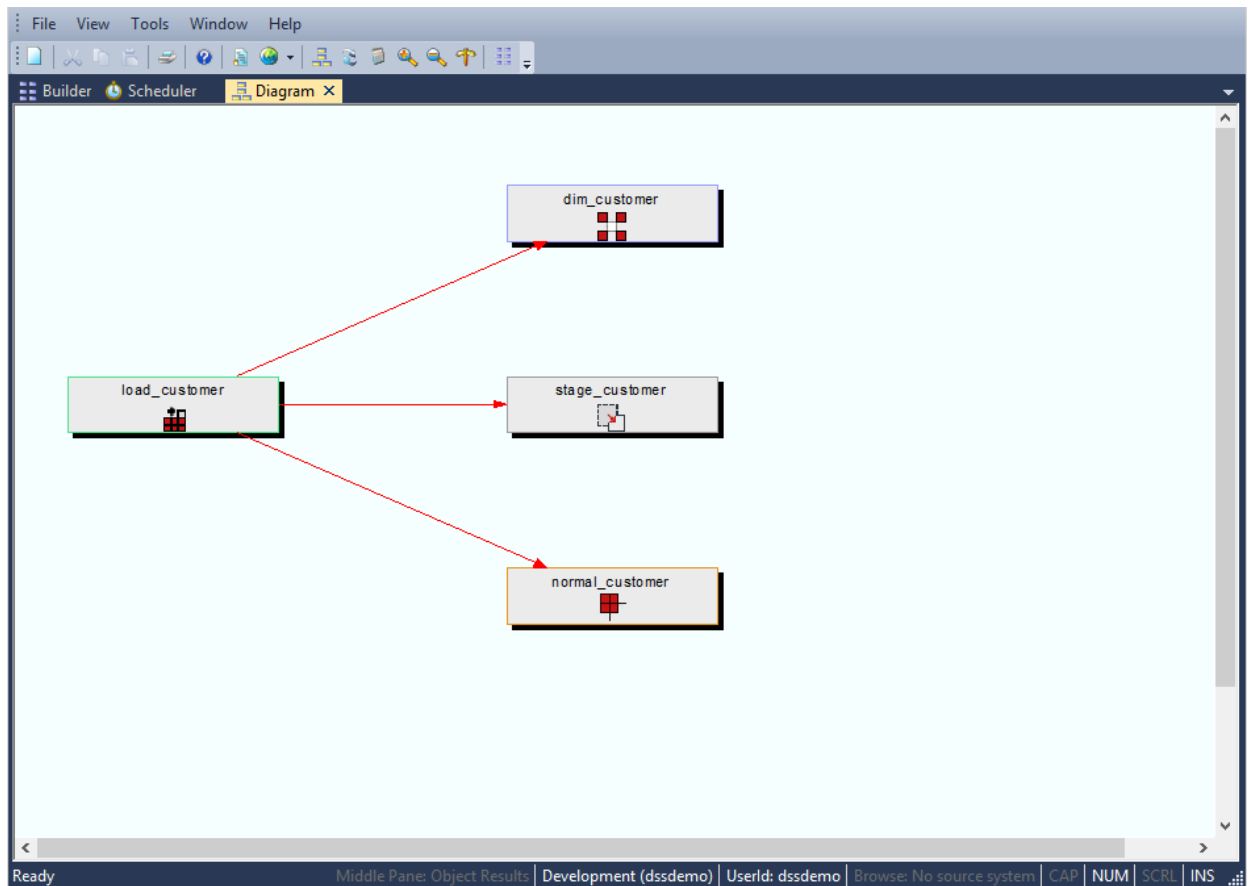
1. A linked table diagram for a model object will display only related model objects, aggregate objects and views of type "Model View".
2. A linked table diagram for an aggregate object will display only related model objects, aggregate objects and views of type "Model View".
3. A linked table diagram for a view of type "Model View" will display only related model objects, aggregate objects and views of type "Model View".
4. A linked table diagram for a view of type "View" will display only related views of type "View".

IMPACT DIAGRAM

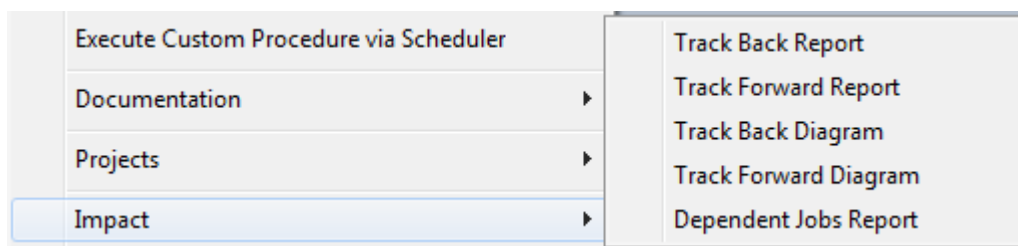
A track forward impact diagram can be displayed for any table. It shows connections forward from the chosen table to the subsequent tables built with columns from this table.

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.



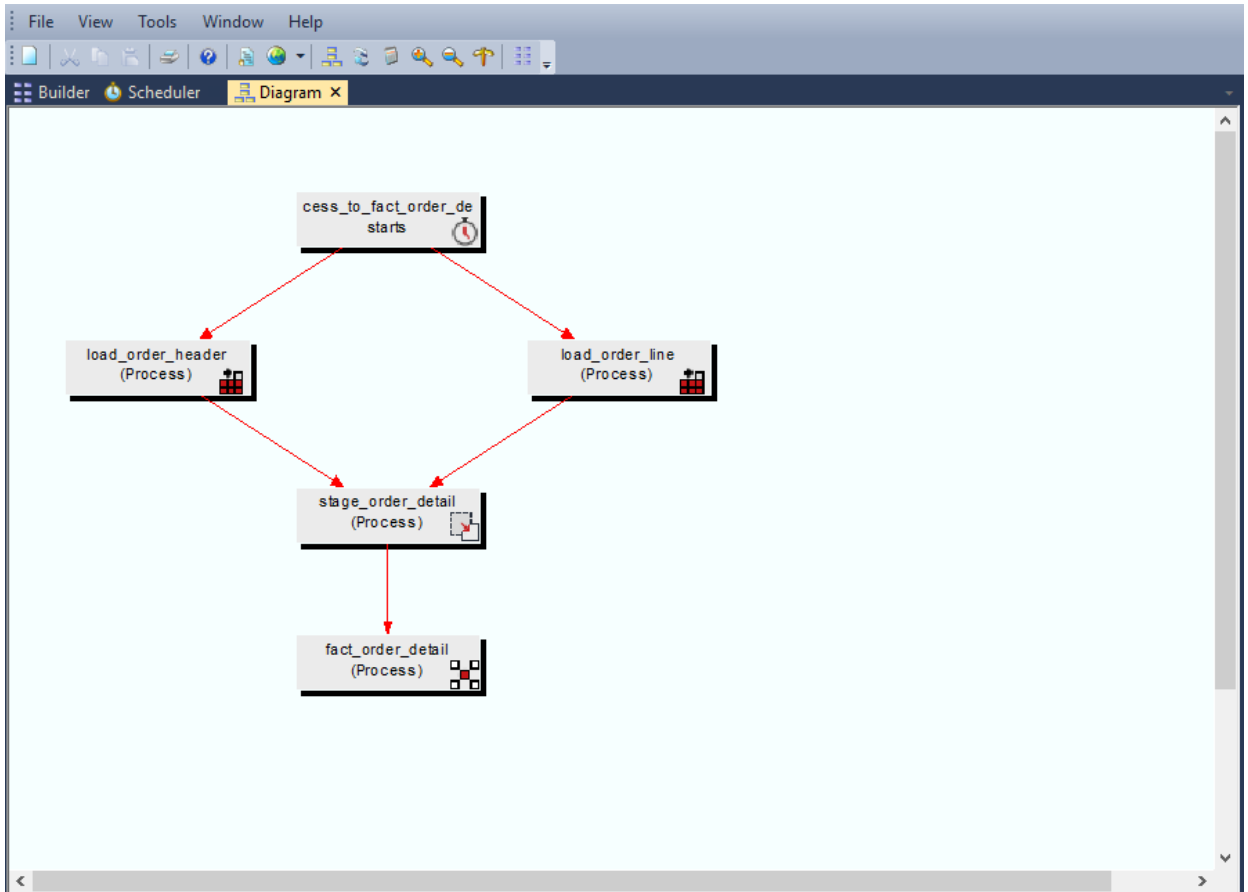
Note: It is also possible to display the track back / forward diagram by right-clicking on a table and choosing **Impact > Track Back Diagram** or **Impact > Track Forward Diagram**:



DEPENDENCY DIAGRAM

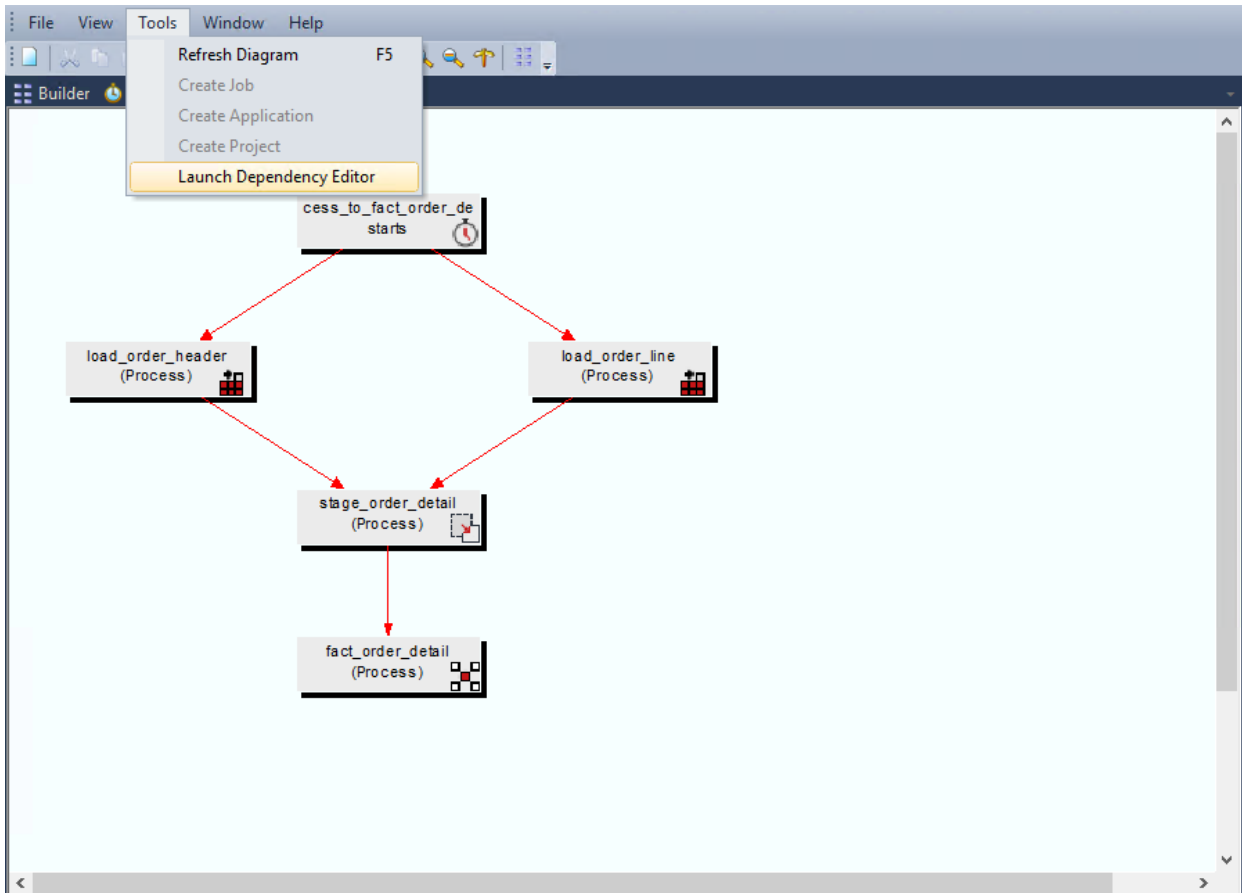
A job dependency diagram can be displayed for any job defined in the WhereScape RED scheduler. It shows the parent and child relationships between tasks within a job.

An example of a **Dependency** diagram in **Standard Diagram** format is displayed below.



Editing a Job's Dependencies from a Job Dependency diagram

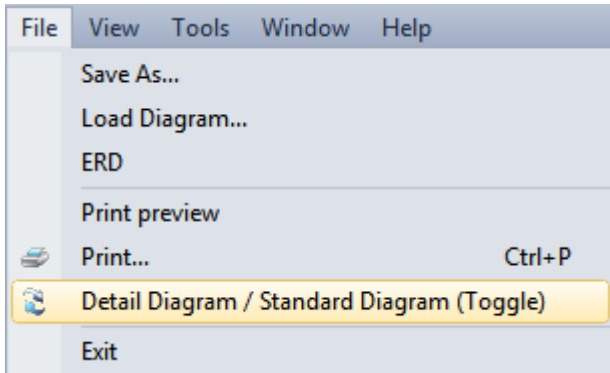
Once a Job Dependency diagram has been created for a job, its dependencies can be edited from the diagram. To do this, select **Launch Dependency Editor** from the Tools menu:



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)**



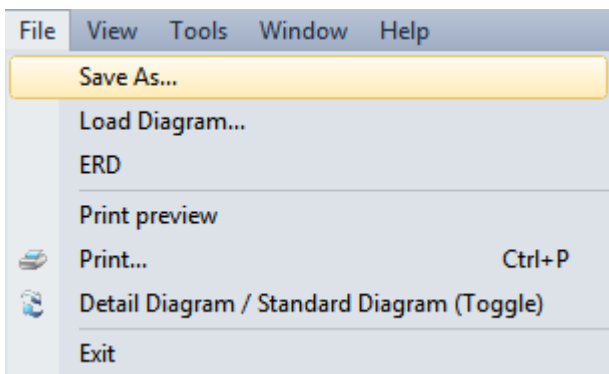
or use the **toggle** button.



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 in WhereScape RED, then the diagram will be loaded.

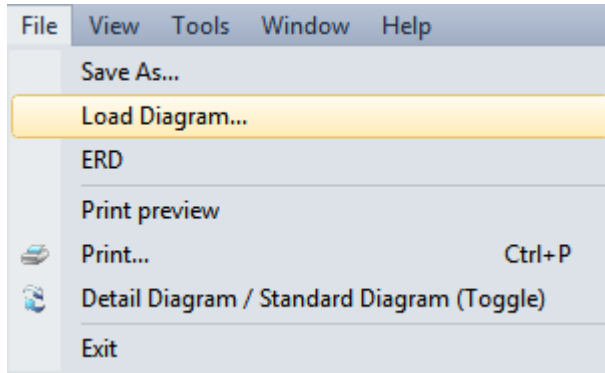


Diagram Print

A diagram can be printed by selecting **File > Print ...**

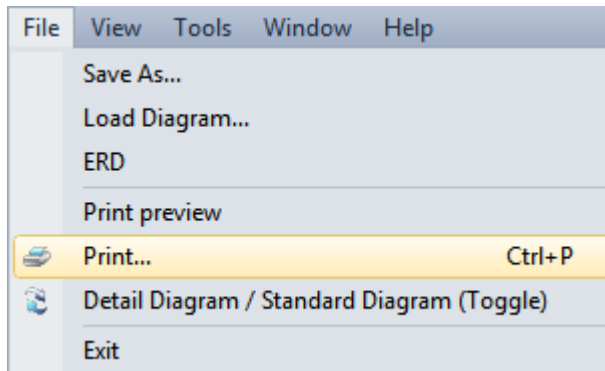
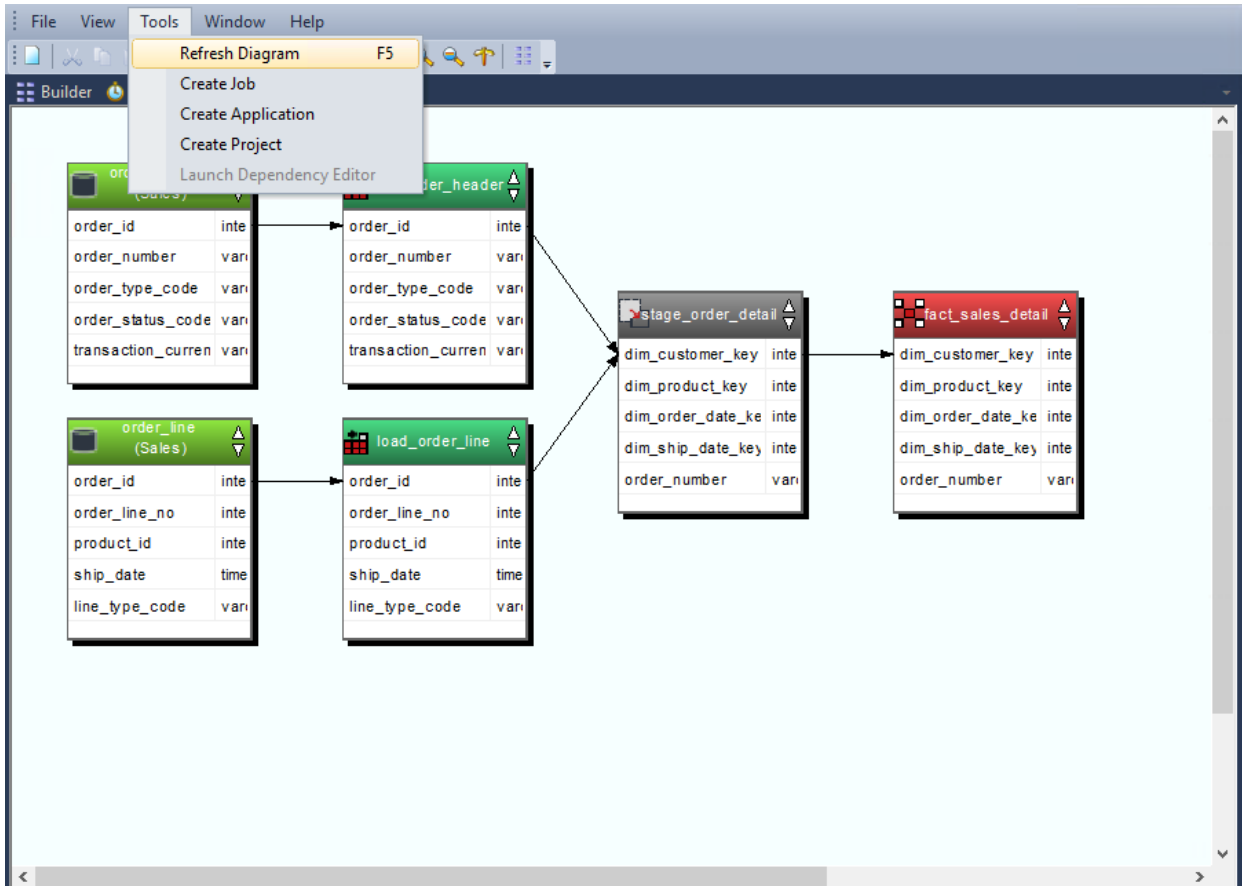


Diagram Refresh

Once a diagram has been displayed, it can be refreshed by choosing **Tools > Refresh Diagram**, or by pressing **F5**.

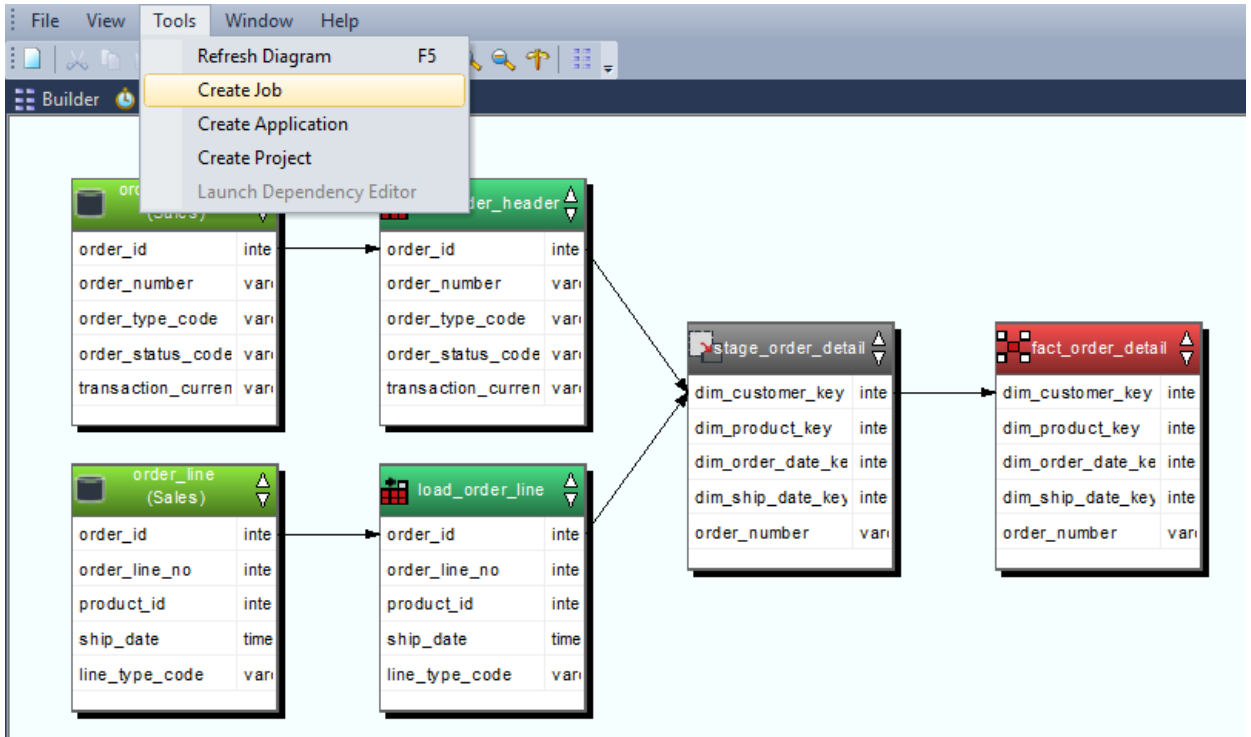


CREATING A JOB FROM A DIAGRAM

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**.

Job Definition ✕

Job Name:

Description:

Frequency:

Start Date:

Start Time:

Maximum Threads:

Scheduler:

Dependent On:

Logs Retained: 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:

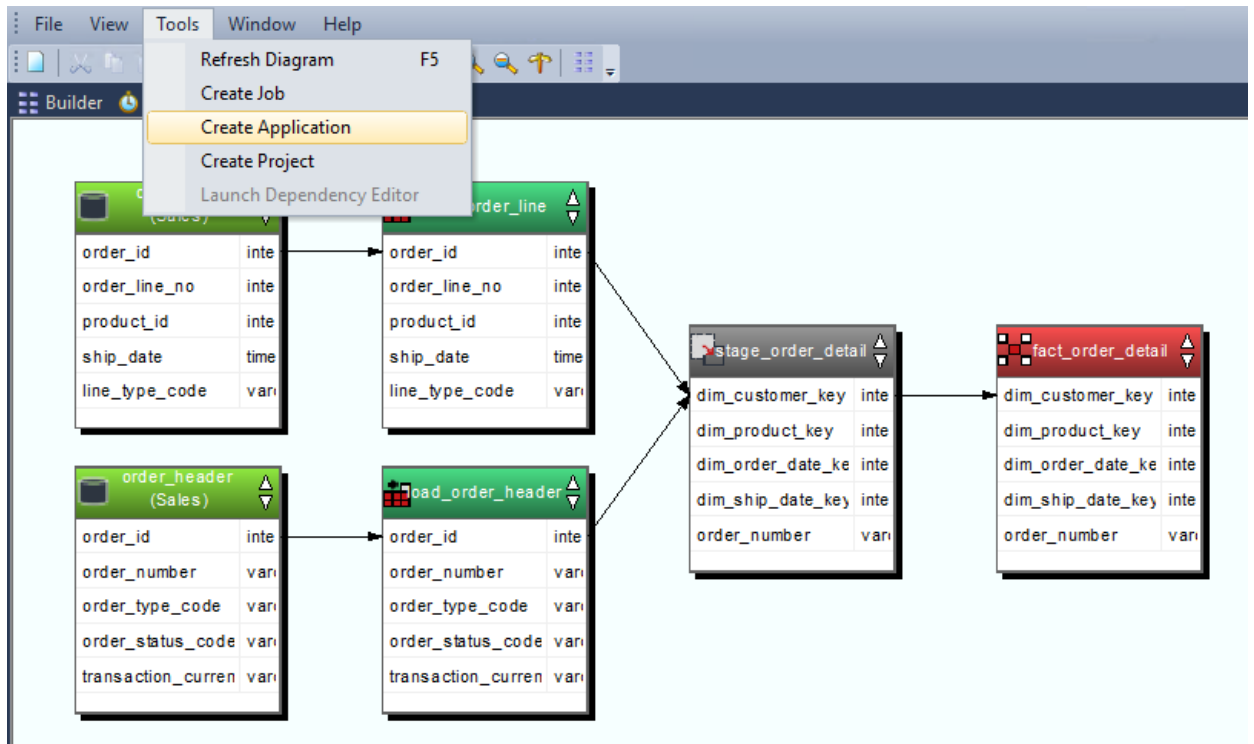
Execute Failure Command in event of dependency failure

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 then click **OK**.

Build Deployment Application

This process builds the files necessary to allow the deployment of a data warehouse solution. These files are read and processed by the Setup Administrator utility. Specify the application identification details and then select the objects to be deployed to and/or deleted from another repository.

Output Directory:

Application Identifier: Application Version:

Application Name:

Description:

You can select or enter a previous application file as a starting point for this application.

Previous Application:

Pre Application Load SQL. (the following optional SQL statement will be issued before the application load commences):

Post Application Load SQL. (the following optional SQL statement will be issued after the application load completes):

3 A dialog will display, confirming the creation of the application files. Click **OK**.

WhereScape RED

i The following files have been created in:
C:\ProgramData\WhereScape\Application\
app_obj_fact_order_d_130822101354.wst
app_data_fact_order_d_130822101354.wst
app_id_fact_order_d_130822101354.wst
app_con_fact_order_d_130822101354.wst
app_map_fact_order_d_130822101354.wst
app_del_fact_order_d_130822101354.wst
app_ext_fact_order_d_130822101354.wst

These files can be moved to another location or system if required.
All files must be moved for the application to load.

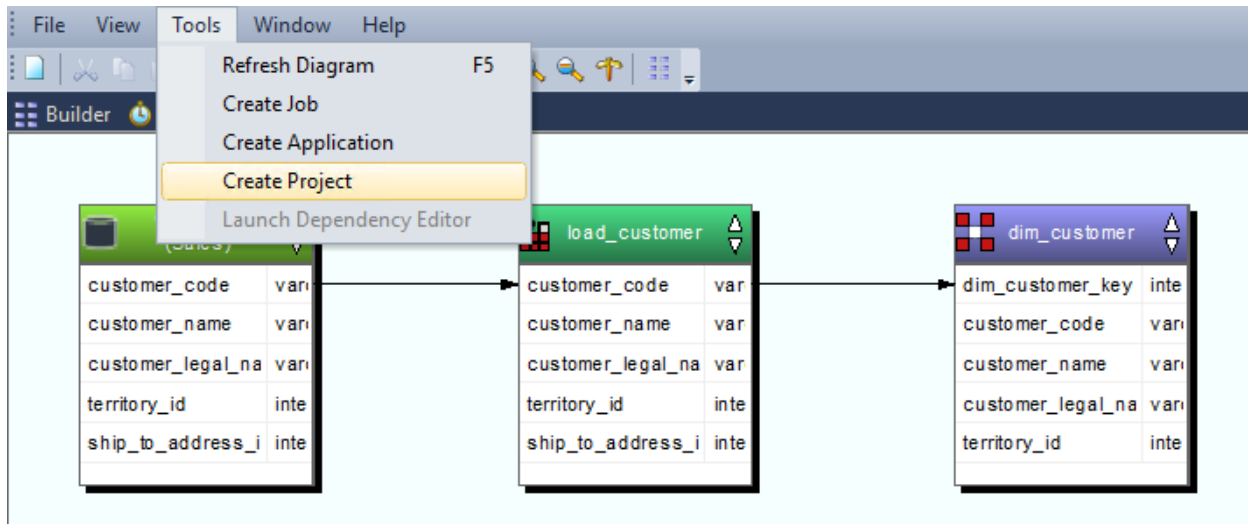
Note: Creating an application from a diagram will save the objects in the diagram and the associated objects, including indexes.

CREATING A PROJECT FROM A DIAGRAM

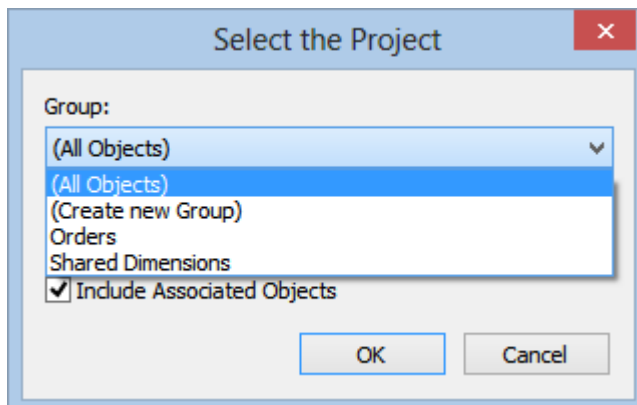
A project can be created from a **Source** diagram or a **Joins** diagram.

To Create a Project from a Diagram

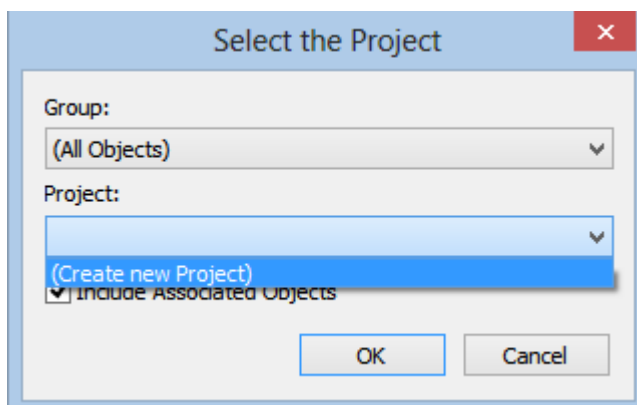
- 1 Once the diagram is displayed, select **Tools/Create Project**.



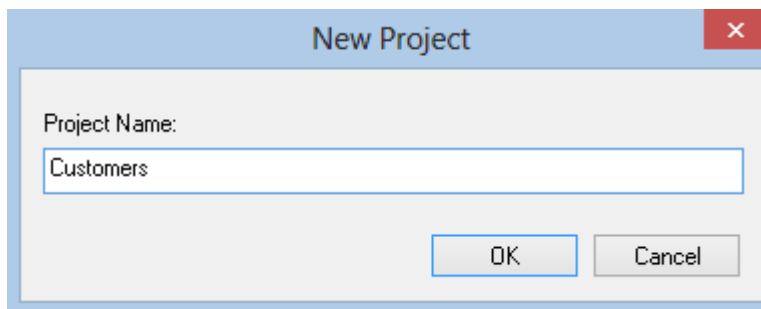
- 2 Select an existing group or create a new group.



- 3 Select to **Create new Project**.

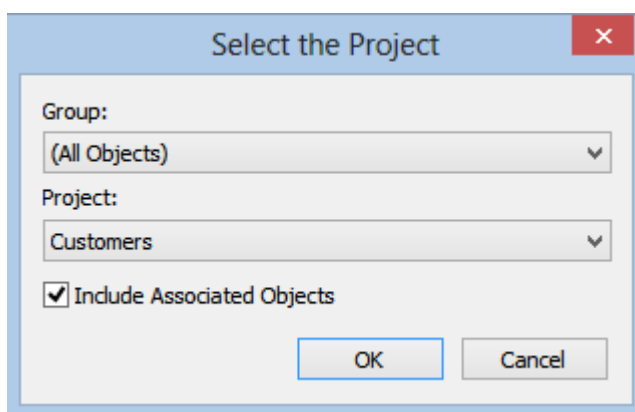


- 4 Enter the name of the new project and click **OK**.



The screenshot shows a dialog box titled "New Project". It has a close button (X) in the top right corner. The main area contains a label "Project Name:" followed by a text input field containing the text "Customers". At the bottom right, there are two buttons: "OK" and "Cancel".

- 5 Click **OK**.



The screenshot shows a dialog box titled "Select the Project". It has a close button (X) in the top right corner. The main area contains two dropdown menus: "Group:" with "(All Objects)" selected, and "Project:" with "Customers" selected. Below the dropdowns is a checked checkbox labeled "Include Associated Objects". At the bottom right, there are two buttons: "OK" and "Cancel".

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 checkbox is selected by default.

CHAPTER 29

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 | 797 |
| OLAP Dimension-Cube Matrix..... | 798 |
| Model Views for a Specified Model | 800 |
| Column Reports..... | 801 |
| Table Reports | 809 |
| Procedure Reports | 816 |
| Index Reports | 819 |
| Object Reports | 820 |
| Job Reports | 827 |
| Operational Reports | 831 |

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.

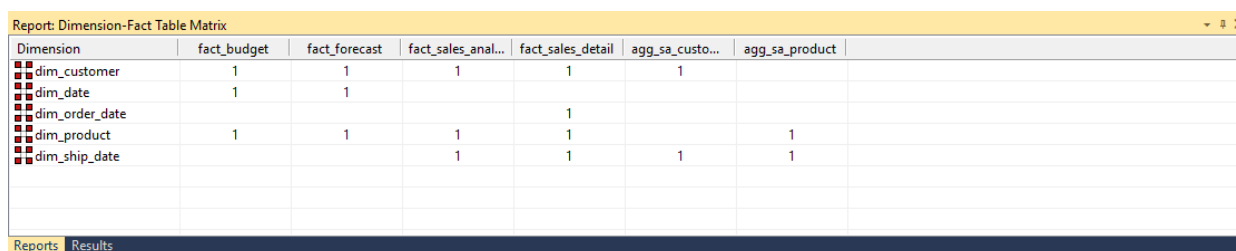
Results

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



| Dimension | fact_budget | fact_forecast | fact_sales_anal... | fact_sales_detail | agg_sa_custo... | agg_sa_product |
|----------------|-------------|---------------|--------------------|-------------------|-----------------|----------------|
| dim_customer | 1 | 1 | 1 | 1 | 1 | |
| dim_date | 1 | 1 | | | | |
| dim_order_date | | | | 1 | | |
| dim_product | 1 | 1 | 1 | 1 | | 1 |
| dim_ship_date | | | 1 | 1 | 1 | 1 |

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.

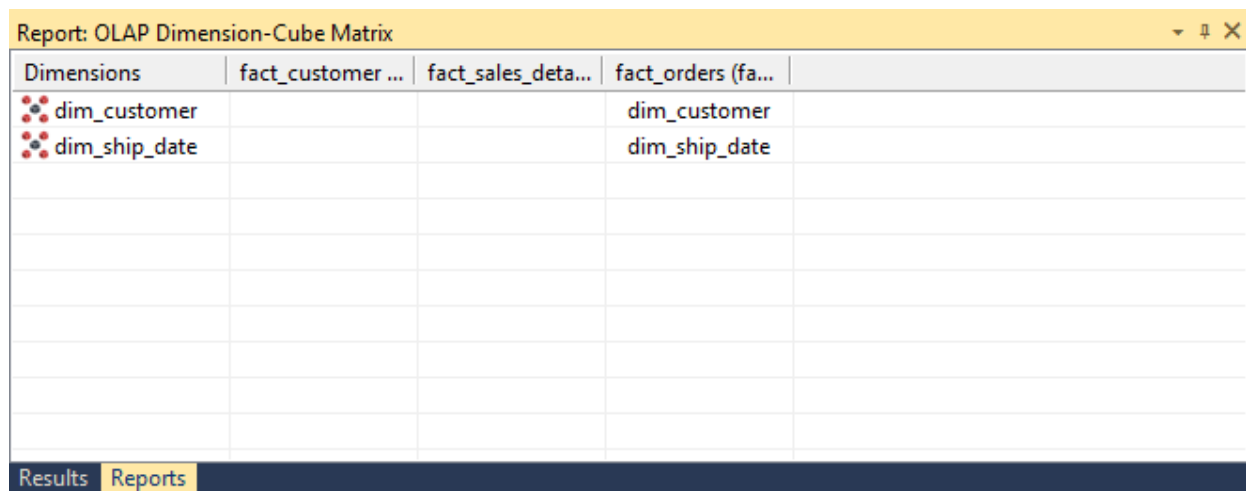
Results

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



The screenshot shows a report window with the title "Report: OLAP Dimension-Cube Matrix". The report displays a table with the following structure:

| Dimensions | fact_customer ... | fact_sales_deta... | fact_orders (fa... |
|---------------|-------------------|--------------------|--------------------|
| dim_customer | | | dim_customer |
| dim_ship_date | | | dim_ship_date |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |

At the bottom of the window, there are tabs for "Results" and "Reports", with "Reports" currently selected.

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.

MODEL VIEWS FOR A SPECIFIED MODEL

This report shows model views built on a specified model table.

Objects Included

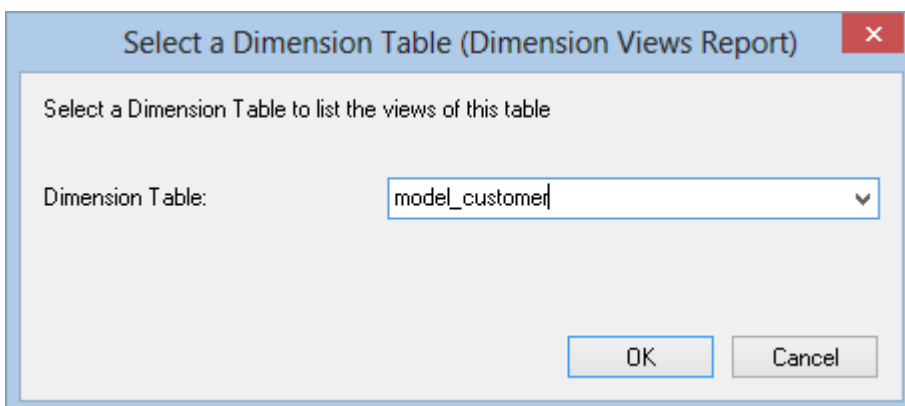
The following WhereScape RED object types are included in this report:

- Model Views

Parameters

This report has one parameter:

- Model table name



Select a Dimension Table (Dimension Views Report)

Select a Dimension Table to list the views of this table

Dimension Table:

OK Cancel

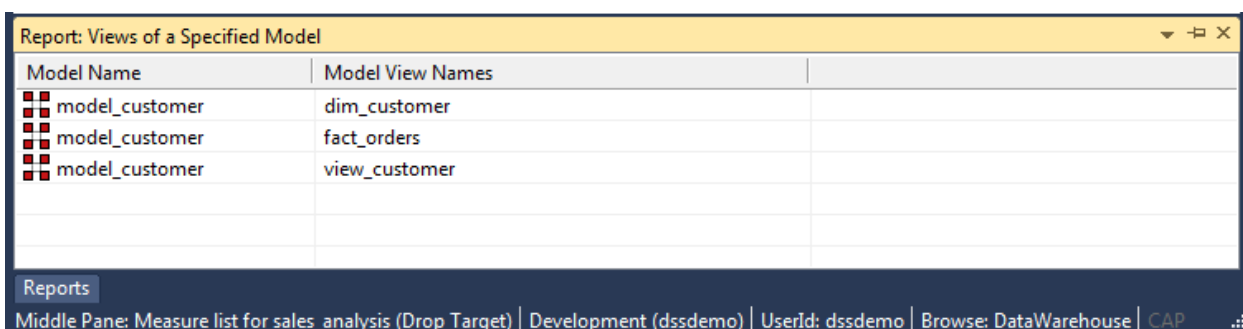
Results

The results of this report are displayed as a list of objects in the metadata repository with the following columns:

- Model Name (*the name of the Model table*)
- Model View Names

The result set is sortable by clicking on the appropriate column heading.

Report Example



| Model Name | Model View Names |
|----------------|------------------|
| model_customer | dim_customer |
| model_customer | fact_orders |
| model_customer | view_customer |

Reports

Middle Pane: Measure list for sales_analysis (Drop Target) | Development (dssdemo) | UserId: dssdemo | Browse: DataWarehouse | CAP

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:

- Model Tables
- Views
- Aggregate Tables
- Join Indexes
- Hubs, Satellites and Links

Parameters

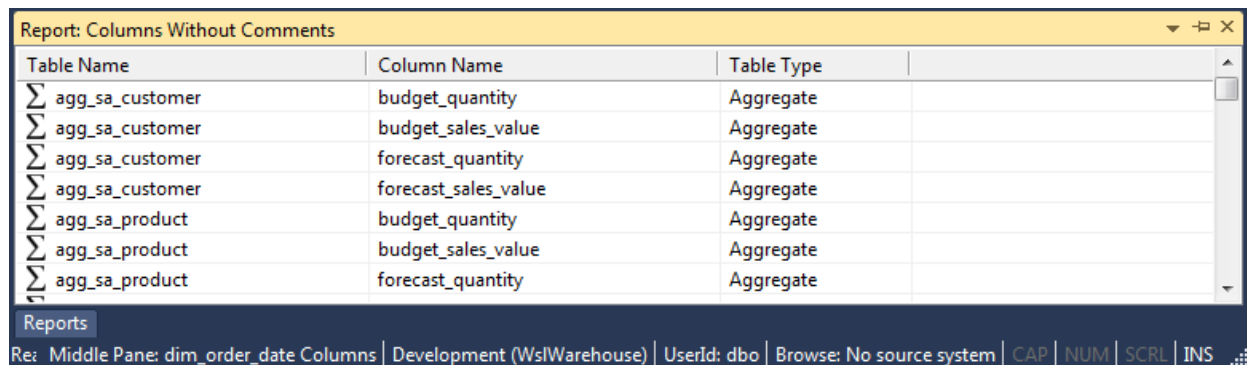
There are no parameters for this report.

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 model table*)
- Column Name
- Table type

Report Example



| Table Name | Column Name | Table Type |
|-----------------|----------------------|------------|
| agg_sa_customer | budget_quantity | Aggregate |
| agg_sa_customer | budget_sales_value | Aggregate |
| agg_sa_customer | forecast_quantity | Aggregate |
| agg_sa_customer | forecast_sales_value | Aggregate |
| agg_sa_product | budget_quantity | Aggregate |
| agg_sa_product | budget_sales_value | Aggregate |
| agg_sa_product | forecast_quantity | Aggregate |

Reports

Re: Middle Pane: dim_order_date Columns | Development (WslWarehouse) | UserId: dbo | Browse: No source system | CAP | NUM | SCRL | INS

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
- Model Tables
- Aggregate Tables
- Join Indexes
- Views
- Exports
- Hubs, Satellites and Links

Parameters

There are no parameters for this report.

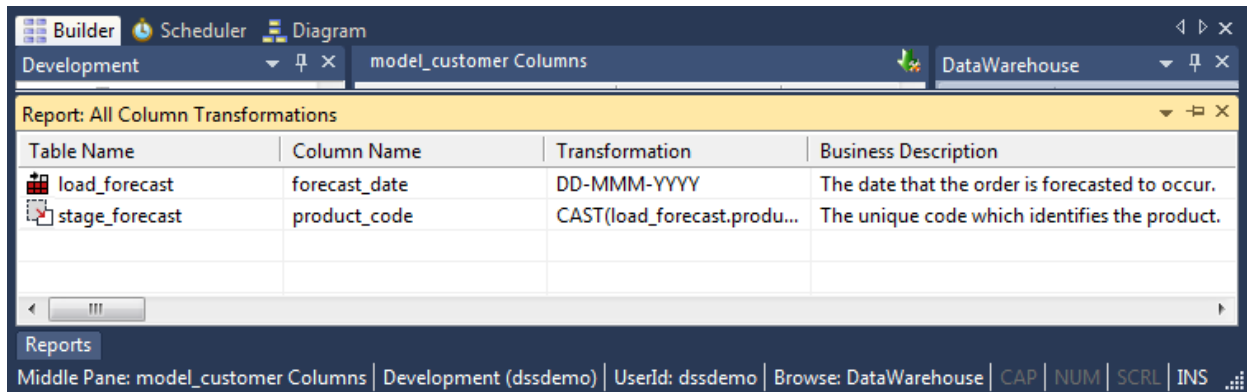
Results

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



| Table Name | Column Name | Transformation | Business Description |
|----------------|---------------|-----------------------------|---|
| load_forecast | forecast_date | DD-MMM-YYYY | The date that the order is forecasted to occur. |
| stage_forecast | product_code | CAST(load_forecast.produ... | The unique code which identifies the product. |

Reports
Middle Pane: model_customer Columns | Development (dssdemo) | UserId: dssdemo | Browse: DataWarehouse | CAP | NUM | SCRL | INS

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.

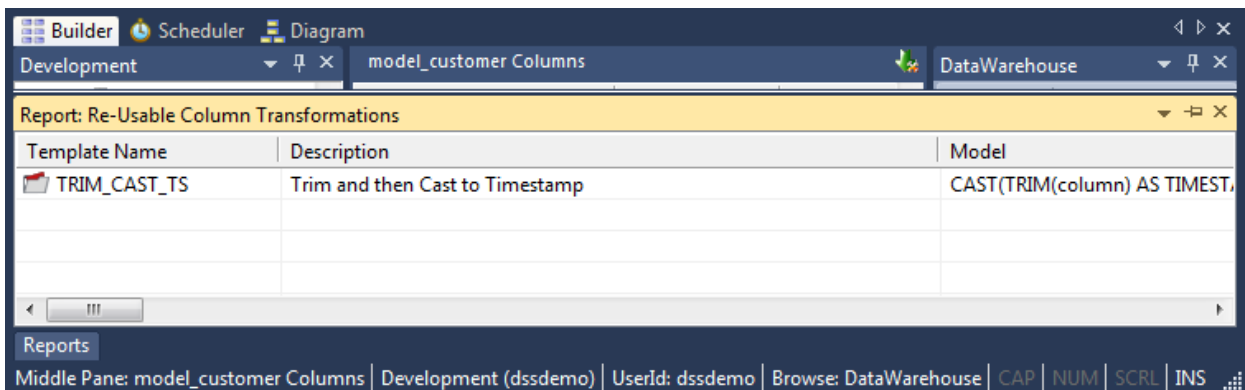
Results

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



| Template Name | Description | Model |
|---------------|---------------------------------|-----------------------------|
| TRIM_CAST_TS | Trim and then Cast to Timestamp | CAST(TRIM(column) AS TIMEST |
| | | |
| | | |

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 shows the lineage of a specified column in a specified table, including any transformations.

Objects Included

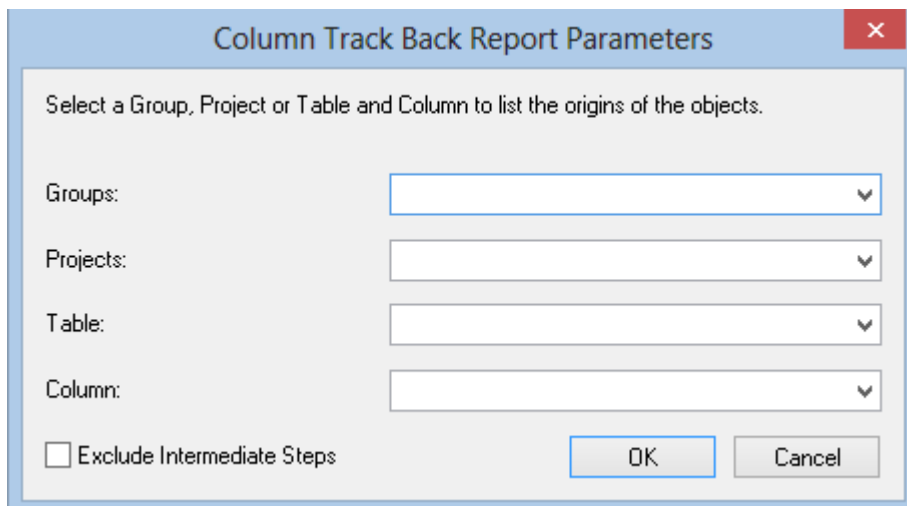
The following WhereScape RED object types are included in this report:

- Load Tables
- Stage Tables
- Model Tables
- Aggregate Tables
- Join Indexes
- Views
- Exports
- Hubs, Satellites and Links

Parameters

This report has two parameters:

- Groups
- Projects
- Table
- Column



Column Track Back Report Parameters

Select a Group, Project or Table and Column to list the origins of the objects.

Groups:

Projects:

Table:

Column:

Exclude Intermediate Steps

OK Cancel

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:

Report: Column Track Back

| Tables (fact_sales_analysis) | Columns | Source Tables | Source Columns |
|--------------------------------|-------------------|--------------------|-------------------|
| fact_sales_analysis | dim_customer_key | dim_customer | dim_customer_key |
| fact_sales_analysis | dim_product_key | dim_product | dim_product_key |
| fact_sales_analysis | dim_ship_date_key | dim_ship_date | dim_ship_date_key |
| fact_sales_analysis | dim_ship_date_key | dim_date | dim_date_key |
| fact_sales_analysis | quantity | fact_sales_detail | quantity |
| fact_sales_analysis | quantity | stage_sales_detail | quantity |
| fact_sales_analysis | quantity | load_order_line | quantity |
| fact_sales_analysis | quantity | order_line | quantity |
| fact_sales_analysis | sales_value | fact_sales_detail | sales_value |
| fact_sales_analysis | sales_value | stage_sales_detail | sales_value |
| fact_sales_analysis | sales_value | load_order_line | sales_value |
| fact_sales_analysis | sales_value | order_line | sales_value |
| fact_sales_analysis | tax | fact_sales_detail | tax |

Reports Results

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:

Report: Column Track Back (Excluding intermediate steps)

| Tables (fact_sales_analysis) | Columns | Source Tables | Source Columns |
|--------------------------------|----------------------|---------------|----------------|
| fact_sales_analysis | quantity | order_line | quantity |
| fact_sales_analysis | sales_value | order_line | sales_value |
| fact_sales_analysis | tax | order_line | tax |
| fact_sales_analysis | budget_quantity | budget.txt | COL3 |
| fact_sales_analysis | budget_sales_value | budget.txt | COL4 |
| fact_sales_analysis | forecast_quantity | forecast.txt | COL3 |
| fact_sales_analysis | forecast_sales_value | forecast.txt | COL4 |
| | | | |
| | | | |
| | | | |
| | | | |

Reports Results

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

Column Track Forward Report Parameters

Select a Group, Project or Table and Column to list the columns derived from the objects.

Groups:

Projects:

Table:

Column:

Exclude Intermediate Steps

OK Cancel

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:

| Tables (load_customer) | Columns | Impact Tables | Impact Columns |
|------------------------|---------|----------------|------------------|
| load_customer | code | dim_customer | code |
| load_customer | code | stage_customer | customer_code |
| load_customer | code | customer | customer_code |
| load_customer | name | dim_customer | name |
| load_customer | name | stage_customer | customer_name |
| load_customer | name | customer | name |
| load_customer | name | odim_customer | name |
| load_customer | address | dim_customer | address |
| load_customer | address | stage_customer | customer_address |
| load_customer | address | customer | address |
| load_customer | city | dim_customer | city |
| load_customer | city | stage_customer | customer_city |
| load_customer | city | customer | city |
| load_customer | city | odim_customer | city |

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:

| Tables (load_customer) | Columns | Impact Tables | Impact Columns |
|------------------------|---------|---------------|----------------|
| load_customer | code | customer | customer_code |
| load_customer | name | odim_customer | name |
| load_customer | address | customer | address |
| load_customer | city | odim_customer | city |
| load_customer | state | odim_customer | state |

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 four reports for analyzing Tables:

- Tables Without Comments
- Load Tables by Connection
- Export Objects by Connection
- 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:

- Model Tables
- Views
- Aggregate Tables
- Join Indexes

Parameters

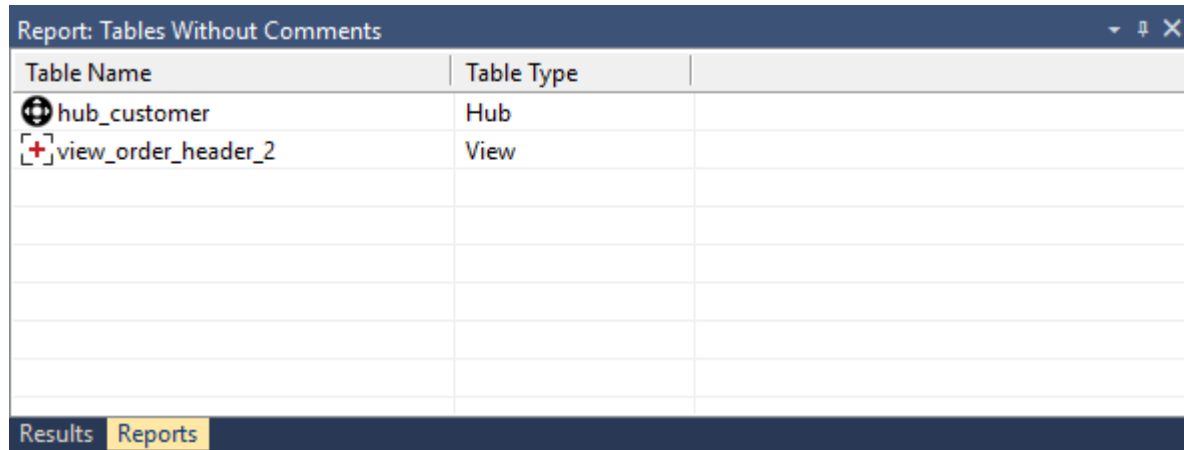
There are no parameters for this report.

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 model table*)
- Table type

Report Example



| Table Name | Table Type |
|---------------------|------------|
| hub_customer | Hub |
| view_order_header_2 | View |
| | |
| | |
| | |
| | |
| | |
| | |

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.

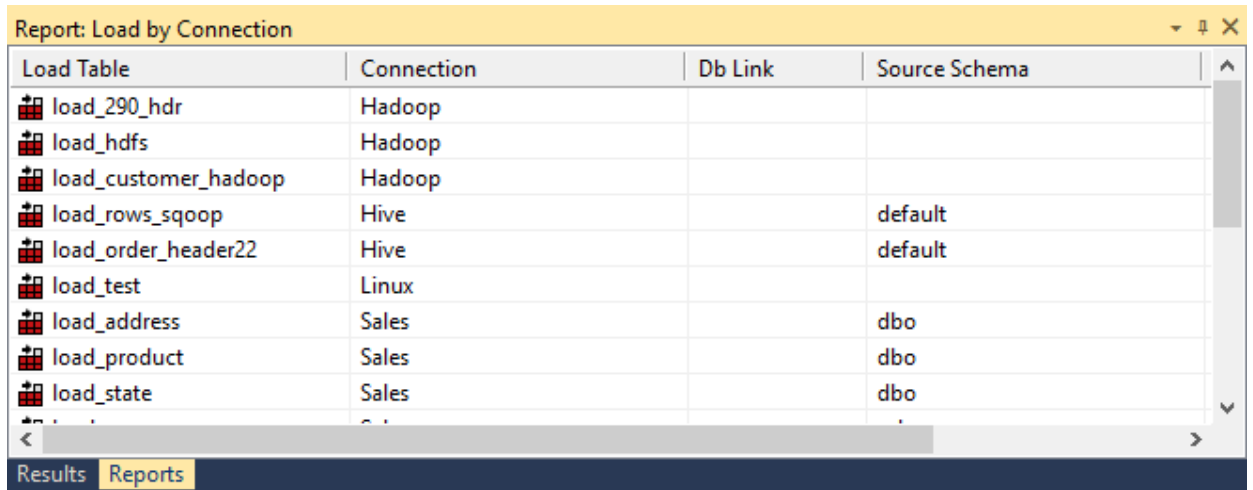
Results

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*)
- Connection
- 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



| Load Table | Connection | Db Link | Source Schema |
|----------------------|------------|---------|---------------|
| load_290_hdr | Hadoop | | |
| load_hdfs | Hadoop | | |
| load_customer_hadoop | Hadoop | | |
| load_rows_sqoop | Hive | | default |
| load_order_header22 | Hive | | default |
| load_test | Linux | | |
| load_address | Sales | | dbo |
| load_product | Sales | | dbo |
| load_state | Sales | | dbo |

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.

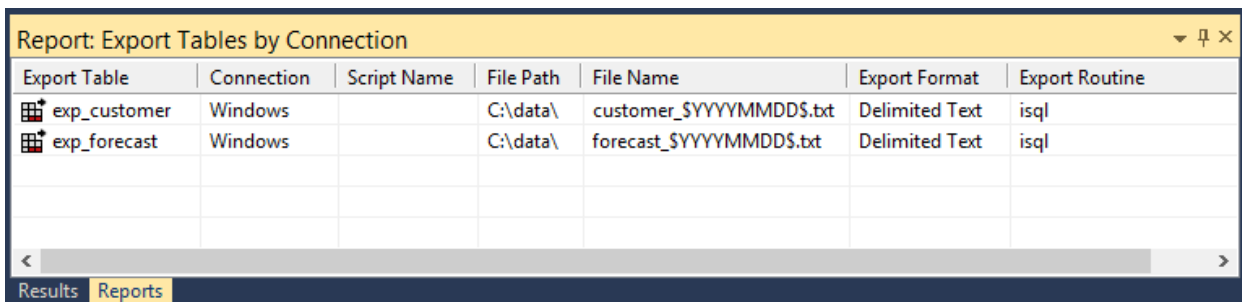
Results

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



| Export Table | Connection | Script Name | File Path | File Name | Export Format | Export Routine |
|--------------|------------|-------------|-----------|-------------------------|----------------|----------------|
| exp_customer | Windows | | C:\data\ | customer_YYYYMMDD\$.txt | Delimited Text | isql |
| exp_forecast | Windows | | C:\data\ | forecast_YYYYMMDD\$.txt | Delimited Text | isql |

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

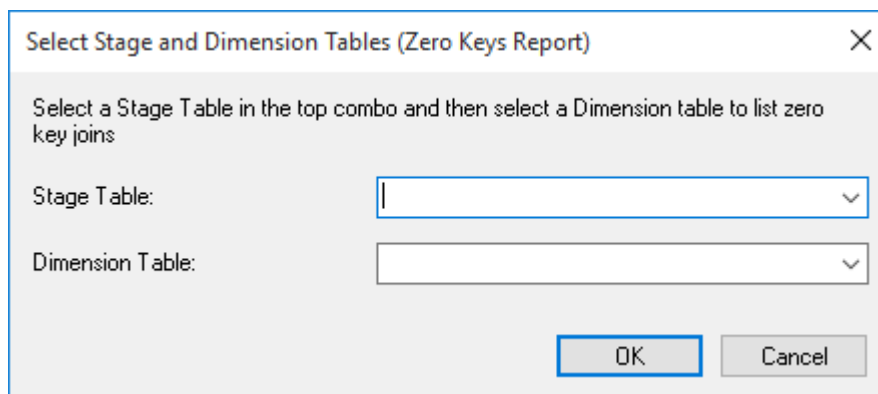
The following WhereScape RED object types are included in this report:

- 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*)



Select Stage and Dimension Tables (Zero Keys Report) X

Select a Stage Table in the top combo and then select a Dimension table to list zero key joins

Stage Table:

Dimension Table:

OK Cancel

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



| customer_co... | count |
|----------------|-------|
| <NULL> | 21 |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |

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 TABLES/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.

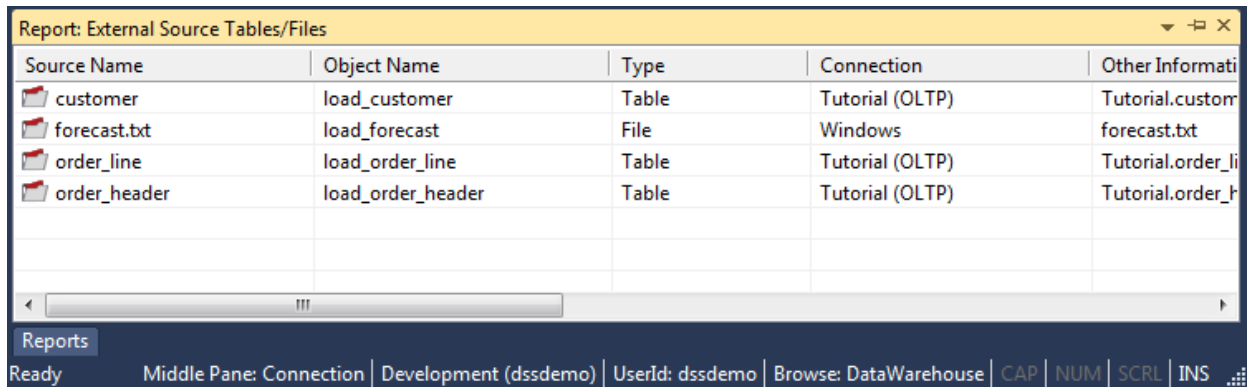
Results

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



| Source Name | Object Name | Type | Connection | Other Information |
|--------------|-------------------|-------|-----------------|-------------------|
| customer | load_customer | Table | Tutorial (OLTP) | Tutorial.custom |
| forecast.txt | load_forecast | File | Windows | forecast.txt |
| order_line | load_order_line | Table | Tutorial (OLTP) | Tutorial.order_li |
| order_header | load_order_header | Table | Tutorial (OLTP) | Tutorial.order_r |

Reports
Ready Middle Pane: Connection | Development (dssdemo) | UserId: dssdemo | Browse: DataWarehouse | CAP | NUM | SCRL | INS

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.

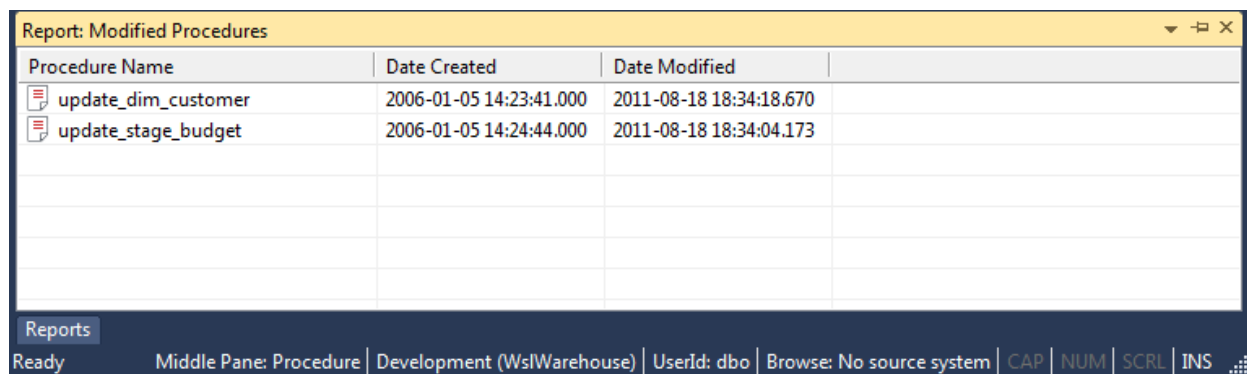
Results

The results of this report are displayed as a list of modified procedures in the metadata repository with the following columns:

- 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



| Procedure Name | Date Created | Date Modified |
|---------------------|-------------------------|-------------------------|
| update_dim_customer | 2006-01-05 14:23:41.000 | 2011-08-18 18:34:18.670 |
| update_stage_budget | 2006-01-05 14:24:44.000 | 2011-08-18 18:34:04.173 |
| | | |
| | | |
| | | |

Reports
Ready Middle Pane: Procedure | Development (WslWarehouse) | UserId: dbo | Browse: No source system | CAP | NUM | SCRL | INS

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.

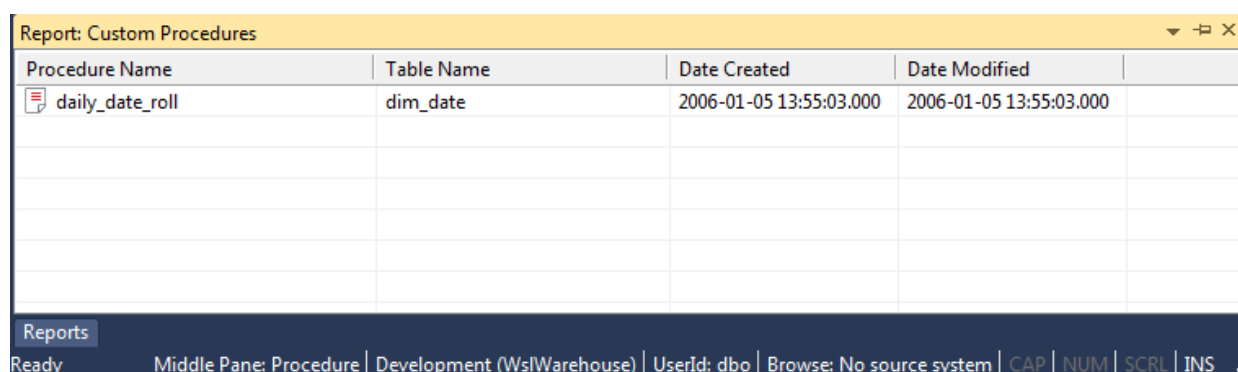
Results

The results of this report are displayed as a list of custom procedures in the metadata repository with the following columns:

- 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



| Procedure Name | Table Name | Date Created | Date Modified |
|-----------------|------------|-------------------------|-------------------------|
| daily_date_roll | dim_date | 2006-01-05 13:55:03.000 | 2006-01-05 13:55:03.000 |
| | | | |
| | | | |
| | | | |
| | | | |

Reports
Ready Middle Pane: Procedure | Development (WslWarehouse) | UserId: dbo | Browse: No source system | CAP | NUM | SCRL | INS

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.

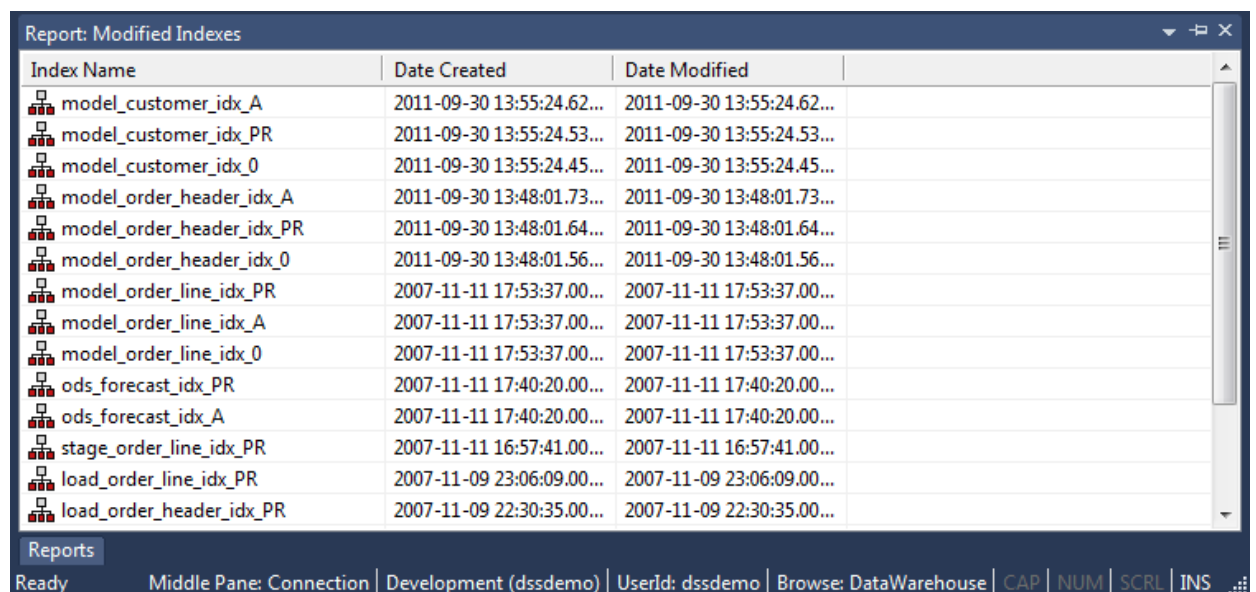
Results

The results of this report are displayed as a list of indexes in the metadata repository with the following columns:

- 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



The screenshot shows a window titled "Report: Modified Indexes" with a table containing the following data:

| Index Name | Date Created | Date Modified |
|---------------------------|---------------------------|---------------------------|
| model_customer_idx_A | 2011-09-30 13:55:24.62... | 2011-09-30 13:55:24.62... |
| model_customer_idx_PR | 2011-09-30 13:55:24.53... | 2011-09-30 13:55:24.53... |
| model_customer_idx_0 | 2011-09-30 13:55:24.45... | 2011-09-30 13:55:24.45... |
| model_order_header_idx_A | 2011-09-30 13:48:01.73... | 2011-09-30 13:48:01.73... |
| model_order_header_idx_PR | 2011-09-30 13:48:01.64... | 2011-09-30 13:48:01.64... |
| model_order_header_idx_0 | 2011-09-30 13:48:01.56... | 2011-09-30 13:48:01.56... |
| model_order_line_idx_PR | 2007-11-11 17:53:37.00... | 2007-11-11 17:53:37.00... |
| model_order_line_idx_A | 2007-11-11 17:53:37.00... | 2007-11-11 17:53:37.00... |
| model_order_line_idx_0 | 2007-11-11 17:53:37.00... | 2007-11-11 17:53:37.00... |
| ods_forecast_idx_PR | 2007-11-11 17:40:20.00... | 2007-11-11 17:40:20.00... |
| ods_forecast_idx_A | 2007-11-11 17:40:20.00... | 2007-11-11 17:40:20.00... |
| stage_order_line_idx_PR | 2007-11-11 16:57:41.00... | 2007-11-11 16:57:41.00... |
| load_order_line_idx_PR | 2007-11-09 23:06:09.00... | 2007-11-09 23:06:09.00... |
| load_order_header_idx_PR | 2007-11-09 22:30:35.00... | 2007-11-09 22:30:35.00... |

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 is four reports for analyzing Objects:

- Objects-Project Matrix
- Modified Objects (excluding indexes)
- Objects Checked-out
- Loaded or Imported Objects

OBJECTS-PROJECTS MATRIX

This report shows all objects that exist in one or more projects (other than **All Objects**) and the project(s) they exist in.

Objects Included

All object types are included in this report.

Parameters

There are no parameters for this report.

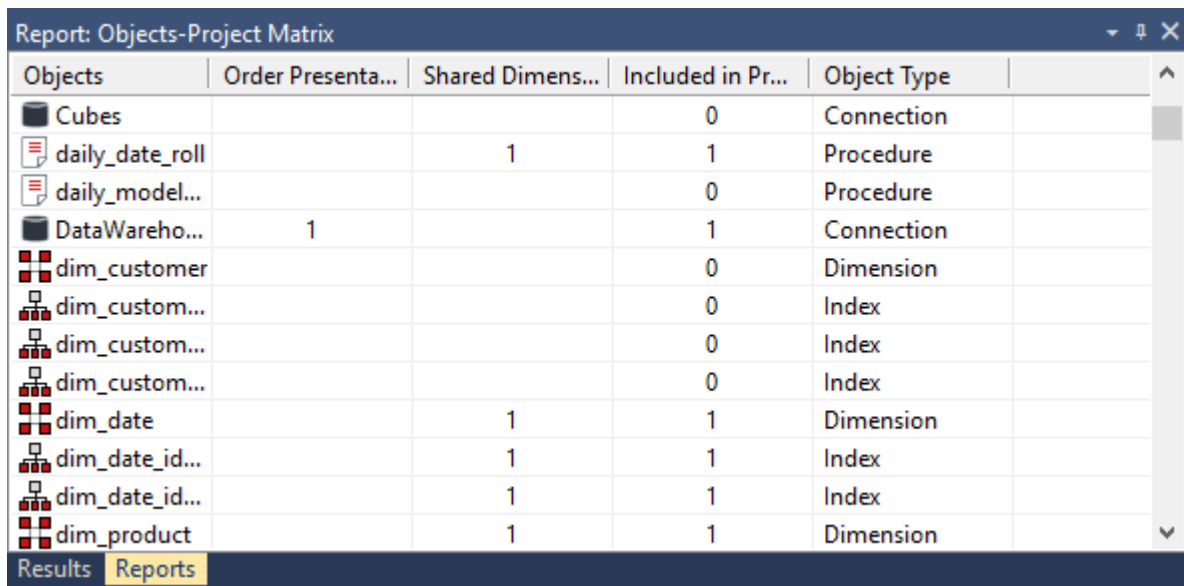
Results

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



| Objects | Order Presenta... | Shared Dimens... | Included in Pr... | Object Type |
|-----------------|-------------------|------------------|-------------------|-------------|
| Cubes | | | 0 | Connection |
| daily_date_roll | | 1 | 1 | Procedure |
| daily_model... | | | 0 | Procedure |
| DataWareho... | 1 | | 1 | Connection |
| dim_customer | | | 0 | Dimension |
| dim_custom... | | | 0 | Index |
| dim_custom... | | | 0 | Index |
| dim_custom... | | | 0 | Index |
| dim_date | | 1 | 1 | Dimension |
| dim_date_id... | | 1 | 1 | Index |
| dim_date_id... | | 1 | 1 | Index |
| dim_product | | 1 | 1 | Dimension |

Results Reports

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
- Model Tables
- Aggregate Tables
- Join Indexes
- Views
- 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

| Name | Object Type | Date Created | Date Modified | M ^ |
|--------------------------|-------------|---------------------------|---------------------------|-----|
| view_order_line | View | 2007-11-11 16:58:45.00... | 2007-11-11 16:58:45.00... | |
| update_stage_order_li... | Procedure | 2007-11-11 16:57:40.00... | 2007-11-11 16:57:40.00... | |
| model_order_line | Dimension | 2007-11-11 16:55:50.00... | 2007-11-11 16:55:50.00... | |
| stage_order_line | Stage | 2007-11-11 16:54:12.00... | 2007-11-11 16:55:18.00... | |
| load_order_line | Load | 2007-11-09 23:05:42.00... | 2007-11-09 23:05:42.00... | |
| daily_model_date_roll | Procedure | 2007-11-09 22:55:56.00... | 2007-11-09 22:55:56.00... | |
| update_model_date | Procedure | 2007-11-09 22:55:31.00... | 2007-11-09 22:55:31.00... | |
| model_date | Dimension | 2007-11-09 22:55:05.00... | 2007-11-09 22:55:05.00... | |
| view_order_header | View | 2007-11-09 22:46:45.00... | 2007-11-09 22:46:45.00... | |
| update_model_order_... | Procedure | 2007-11-09 22:45:39.00... | 2007-11-09 22:45:39.00... | |
| model_order_header | Dimension | 2007-11-09 22:44:16.00... | 2007-11-09 22:44:16.00... | |

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.

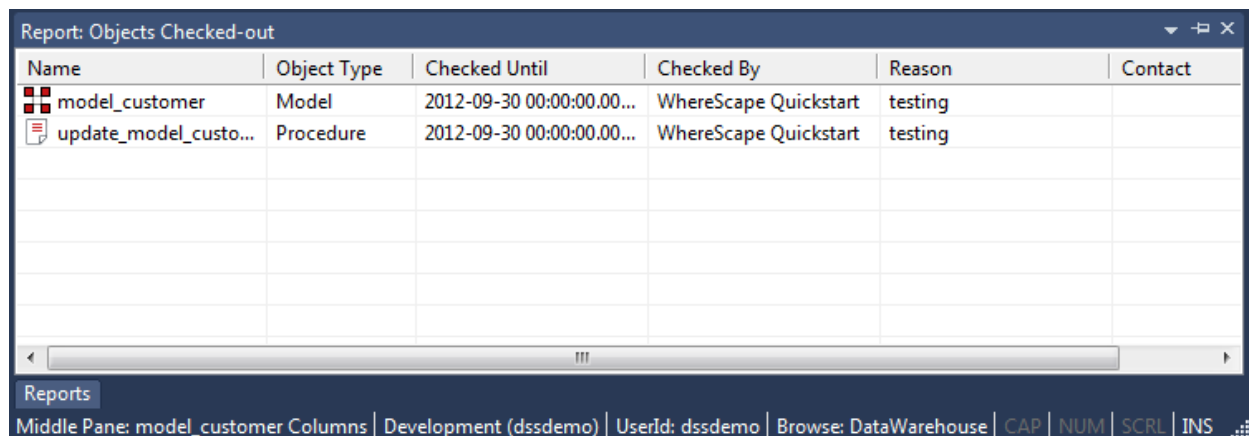
Results

The results of this report are displayed with the following columns:

- Object 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



| Name | Object Type | Checked Until | Checked By | Reason | Contact |
|-----------------------|-------------|---------------------------|-----------------------|---------|---------|
| model_customer | Model | 2012-09-30 00:00:00.00... | WhereScape Quickstart | testing | |
| update_model_custo... | Procedure | 2012-09-30 00:00:00.00... | WhereScape Quickstart | testing | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |

Reports
Middle Pane: model_customer Columns | Development (dssdemo) | UserId: dssdemo | Browse: DataWarehouse | CAP | NUM | SCRL | INS

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
- Model Tables
- Aggregate Tables
- Join Indexes
- Views
- Cubes
- Exports
- Procedures
- Host Scripts

Note: Indexes are not included.

Parameters

There are no parameters for this report.

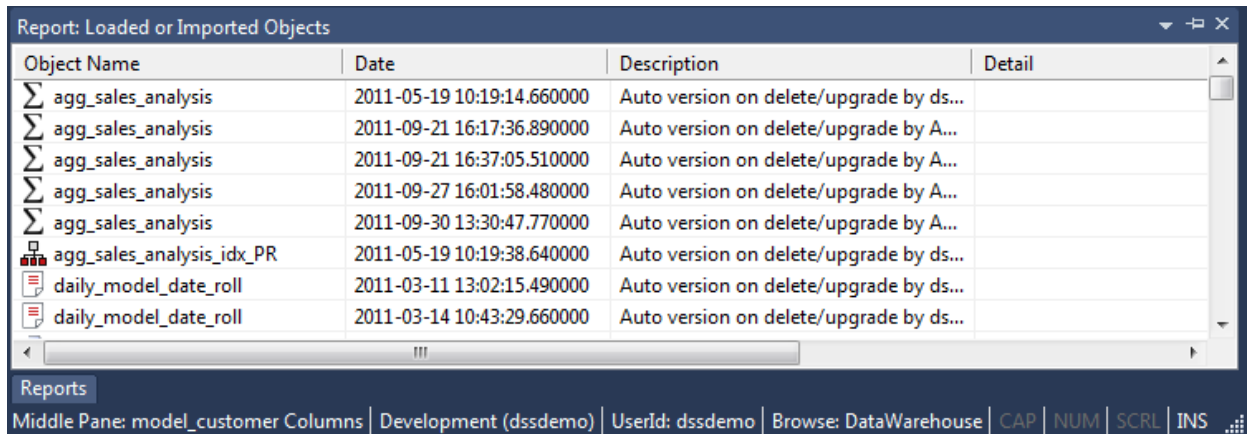
Results

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



Report: Loaded or Imported Objects

| Object Name | Date | Description | Detail |
|---------------------------|----------------------------|---|--------|
| agg_sales_analysis | 2011-05-19 10:19:14.660000 | Auto version on delete/upgrade by ds... | |
| agg_sales_analysis | 2011-09-21 16:17:36.890000 | Auto version on delete/upgrade by A... | |
| agg_sales_analysis | 2011-09-21 16:37:05.510000 | Auto version on delete/upgrade by A... | |
| agg_sales_analysis | 2011-09-27 16:01:58.480000 | Auto version on delete/upgrade by A... | |
| agg_sales_analysis | 2011-09-30 13:30:47.770000 | Auto version on delete/upgrade by A... | |
| agg_sales_analysis_idx_PR | 2011-05-19 10:19:38.640000 | Auto version on delete/upgrade by ds... | |
| daily_model_date_roll | 2011-03-11 13:02:15.490000 | Auto version on delete/upgrade by ds... | |
| daily_model_date_roll | 2011-03-14 10:43:29.660000 | Auto version on delete/upgrade by ds... | |

Reports

Middle Pane: model_customer Columns | Development (dssdemo) | UserId: dssdemo | Browse: DataWarehouse | CAP | NUM | SCRL | INS

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 also appears with **No Job** as their 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

Report: Object-Job Matrix

| Object Name | Action | Job Name | Job Status | Job La: |
|---------------------------|------------|----------------|------------|---------|
| daily_model_date_roll | | No Job | | |
| update_model_customer | | No Job | | |
| update_model_date | | No Job | | |
| update_model_forecast | | No Job | | |
| update_model_order_header | | No Job | | |
| update_model_order_line | | No Job | | |
| update_ods_forecast | | No Job | | |
| update_stage_customer | | No Job | | |
| update_stage_forecast | | No Job | | |
| update_stage_order_header | | No Job | | |
| update_stage_order_line | | No Job | | |
| test | | No Job | | |
| update_dim_time | | No Job | | |
| model_customer | | No Job | | |
| model_date | | No Job | | |
| model_forecast | Statistics | Daily Update | On Hold | 2011-0 |
| model_forecast | Statistics | Daily Update_1 | On Hold | 2011-0 |
| model_forecast | Process | Daily Update_1 | On Hold | 2011-0 |
| model_forecast | Process | Daily Update | On Hold | 2011-0 |
| model_order_header | | No Job | | |
| model_order_line | | No Job | | |
| ods_forecast | | No Job | | |

Reports
 Middle Pane: model_customer Columns | Development (dssdemo) | UserId: dssdemo | Browse: DataWarehouse | CAP | NUM | SCRL | INS

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 a specified object appears in and its action.

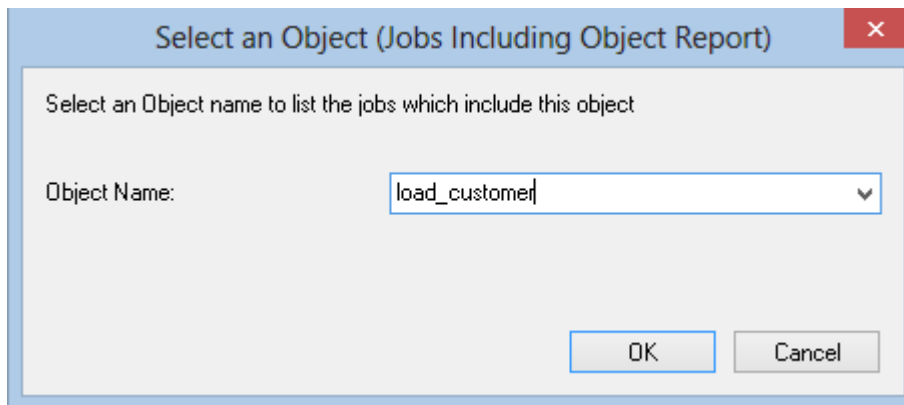
Objects Included

All object types are included in this report.

Parameters

This report has one parameter:

- Object Name



Select an Object (Jobs Including Object Report)

Select an Object name to list the jobs which include this object

Object Name:

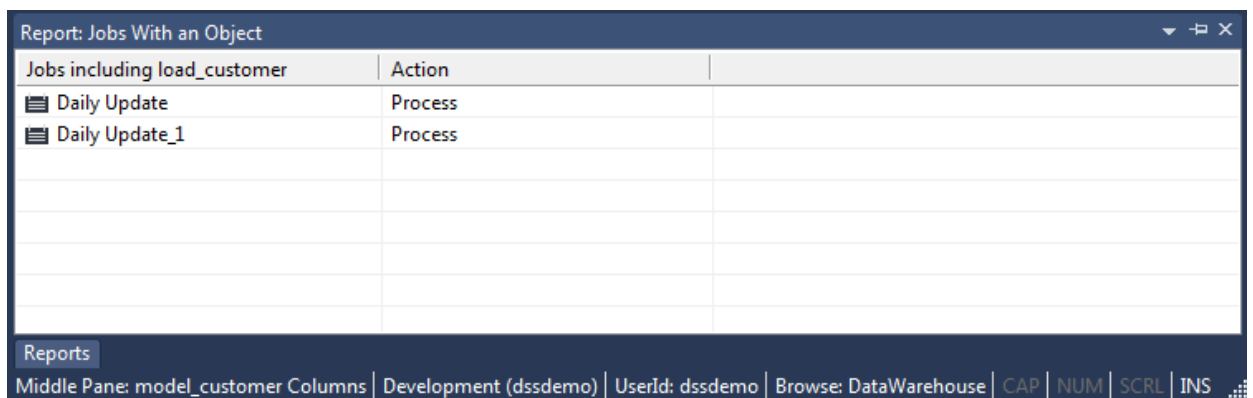
OK Cancel

Results

The results of this report are displayed as a list of jobs with the following columns:

- Jobs including **object_name**
- Action

Report Example



| Jobs including load_customer | Action |
|------------------------------|---------|
| Daily Update | Process |
| Daily Update_1 | Process |
| | |
| | |
| | |
| | |

Reports
Middle Pane: model_customer Columns | Development (dssdemo) | UserId: dssdemo | Browse: DataWarehouse | CAP | NUM | SCRL | INS

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.

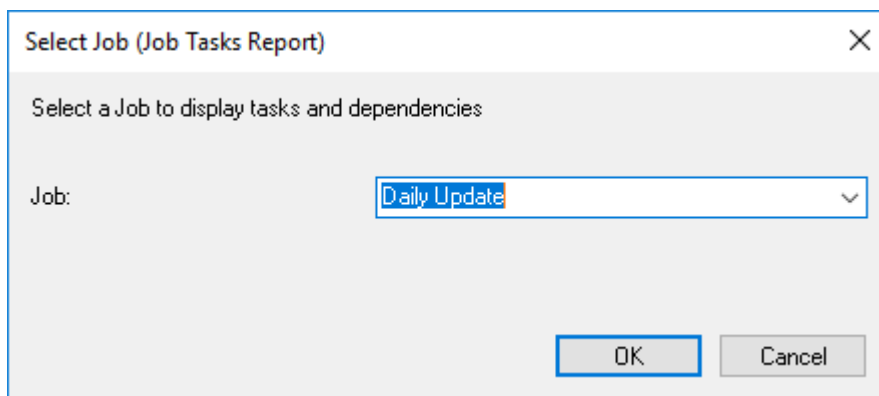
Objects Included

All object types are included in this report.

Parameters

This report has one parameter:

- Job Name



Select Job (Job Tasks Report)

Select a Job to display tasks and dependencies

Job:

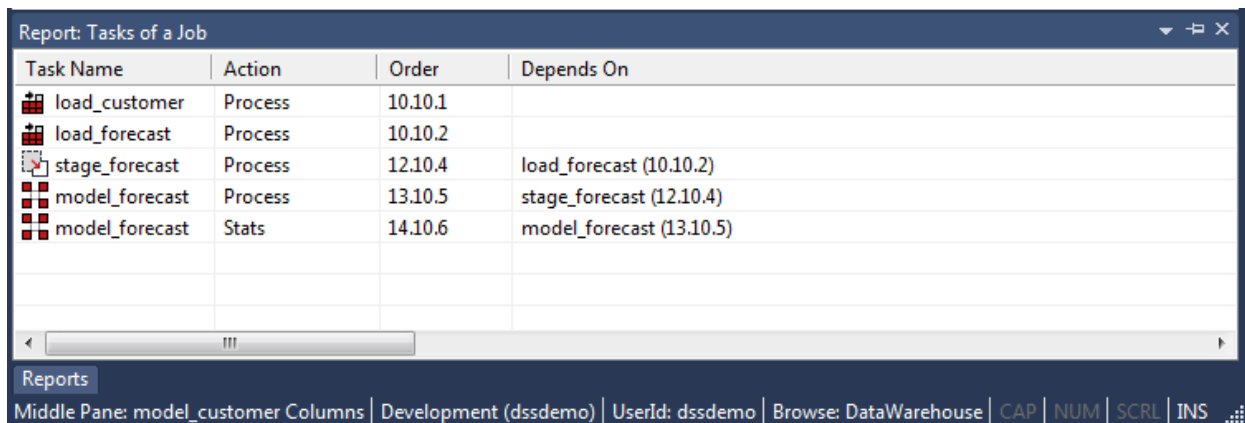
OK Cancel

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



| Task Name | Action | Order | Depends On |
|----------------|---------|---------|--------------------------|
| load_customer | Process | 10.10.1 | |
| load_forecast | Process | 10.10.2 | |
| stage_forecast | Process | 12.10.4 | load_forecast (10.10.2) |
| model_forecast | Process | 13.10.5 | stage_forecast (12.10.4) |
| model_forecast | Stats | 14.10.6 | model_forecast (13.10.5) |

Reports

Middle Pane: model_customer Columns | Development (dssdemo) | UserId: dssdemo | Browse: DataWarehouse | CAP | NUM | SCRL | INS

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 three Operational Reports:

- Object Performance History
- Job Performance History
- Task Performance History

OBJECT PERFORMANCE HISTORY

This report shows the audit trail for a selected object from the scheduler logs.

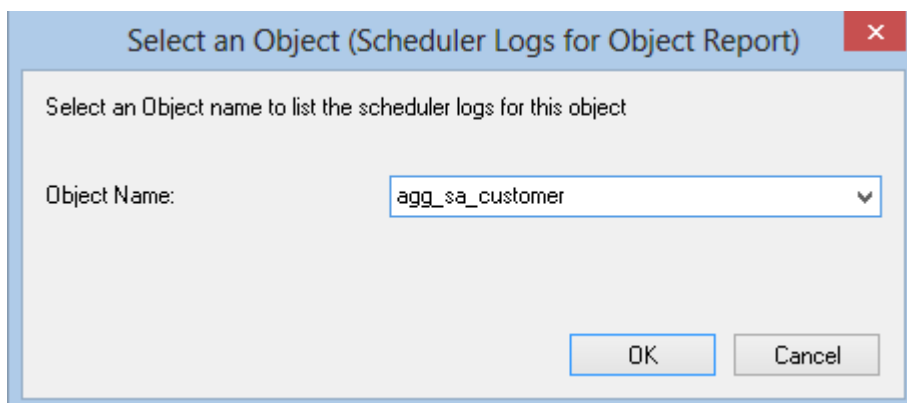
Objects Included

All object types are included in this report.

Parameters

This report has one parameter:

- Object Name

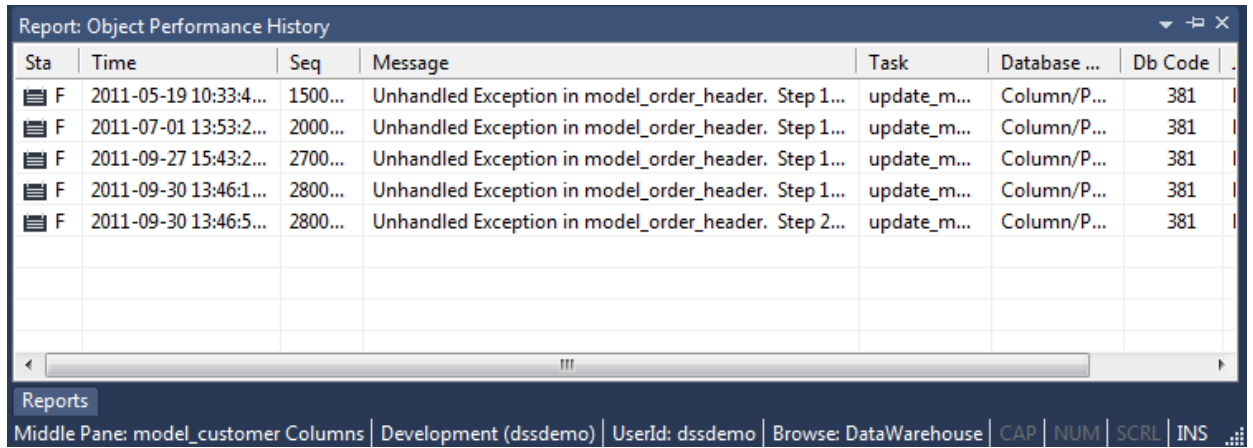


Results

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



The screenshot shows a report window with the title "Report: Object Performance History". The report content is a table with the following columns: Sta, Time, Seq, Message, Task, Database ..., and Db Code. The table contains five rows of data, all indicating "Unhandled Exception in model_order_header. Step 1..." or "Step 2...". The status is "F" for all entries. The task is "update_m...". The database is "Column/P...". The Db Code is "381".

| Sta | Time | Seq | Message | Task | Database ... | Db Code |
|-----|-----------------------|---------|--|-------------|--------------|---------|
| F | 2011-05-19 10:33:4... | 1500... | Unhandled Exception in model_order_header. Step 1... | update_m... | Column/P... | 381 |
| F | 2011-07-01 13:53:2... | 2000... | Unhandled Exception in model_order_header. Step 1... | update_m... | Column/P... | 381 |
| F | 2011-09-27 15:43:2... | 2700... | Unhandled Exception in model_order_header. Step 1... | update_m... | Column/P... | 381 |
| F | 2011-09-30 13:46:1... | 2800... | Unhandled Exception in model_order_header. Step 1... | update_m... | Column/P... | 381 |
| F | 2011-09-30 13:46:5... | 2800... | Unhandled Exception in model_order_header. Step 2... | update_m... | Column/P... | 381 |

Below the table, there is a "Reports" tab and a status bar showing: "Middle Pane: model_customer Columns | Development (dssdemo) | UserId: dssdemo | Browse: DataWarehouse | CAP | NUM | SCRL | INS".

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) of a specified job over time.

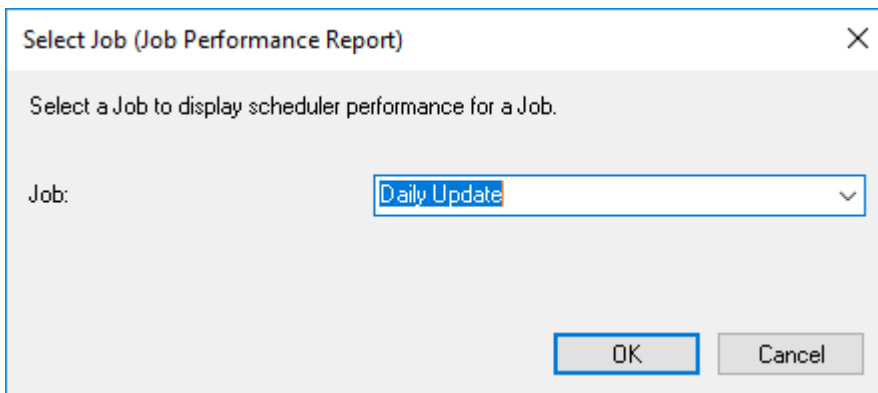
Objects Included

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:

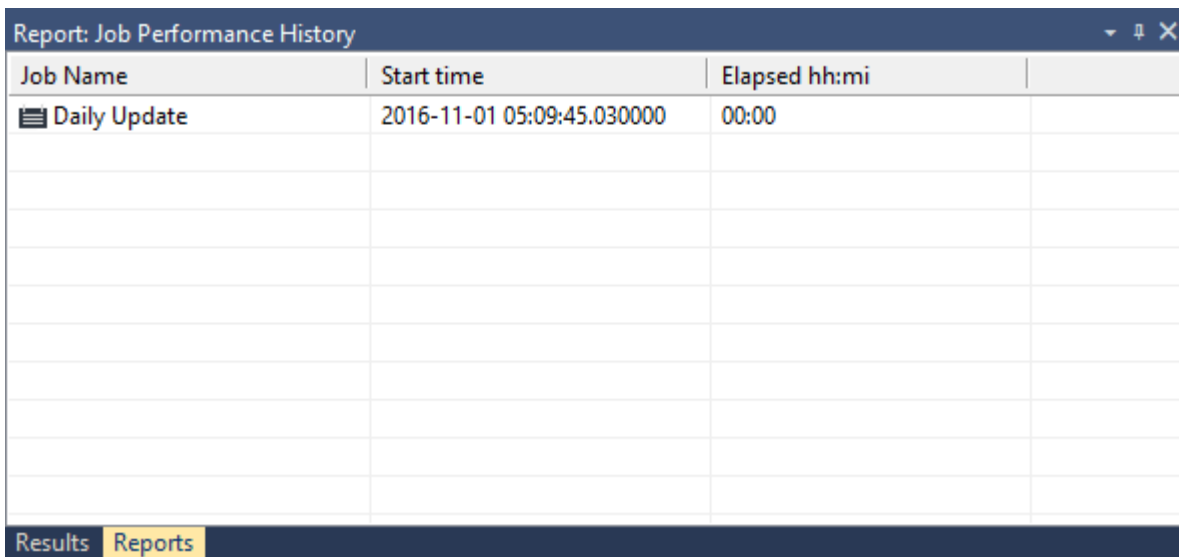
OK Cancel

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



| Job Name | Start time | Elapsed hh:mi |
|----------------|----------------------------|---------------|
| ☰ Daily Update | 2016-11-01 05:09:45.030000 | 00:00 |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |

Results Reports

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) of a specified task within a specified job over time.

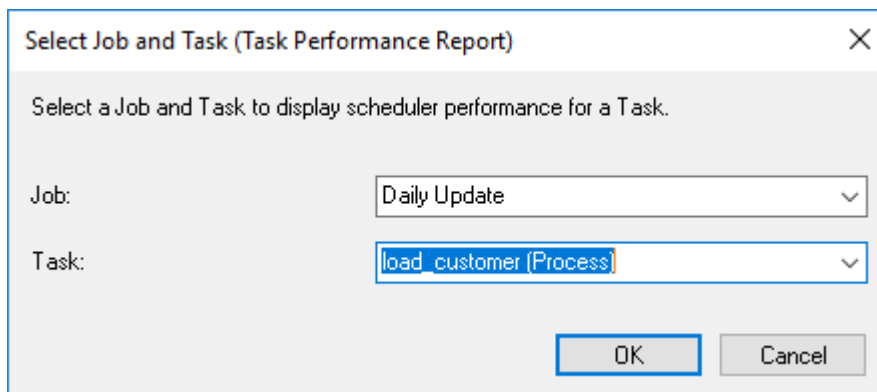
Objects Included

All object types are included in this report.

Parameters

This report has two parameters:

- Job Name
- Task Name (including action)



Select Job and Task (Task Performance Report) [X]

Select a Job and Task to display scheduler performance for a Task.

Job: Daily Update [v]

Task: load_customer (Process) [v]

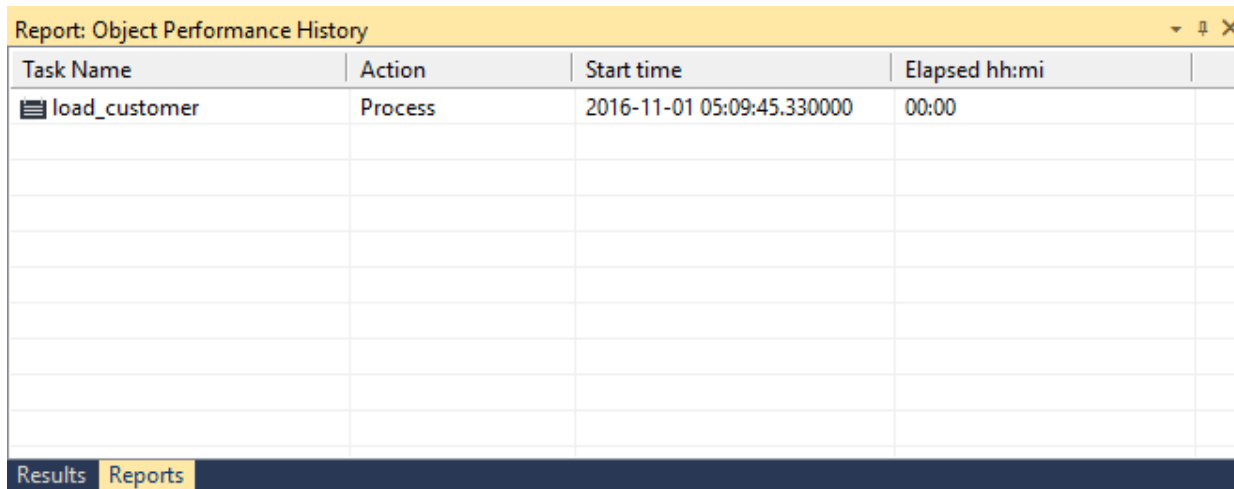
OK Cancel

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



| Task Name | Action | Start time | Elapsed hh:mi |
|---------------|---------|----------------------------|---------------|
| load_customer | Process | 2016-11-01 05:09:45.330000 | 00:00 |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |

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.

CHAPTER 30

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..... | 838 |
| Validate Workflow Data | 838 |
| Validate Table Create Status | 838 |
| Validate Load Table Status..... | 839 |
| Validate Procedure Status | 839 |
| List Meta-data Tables not in the Database | 839 |
| List Database Tables not in the Meta-data | 841 |
| List Tables with no related Procedures or Scripts | 843 |
| List Procedures not related to a Table..... | 844 |
| Compare Meta-data Repository to another | 846 |
| Compare Meta-data Indexes to Database | 849 |
| Teradata: View of Model Validate | 851 |
| List Duplicate Business Key Columns | 853 |
| Query Data Warehouse Objects..... | 854 |

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 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 user_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
- Model Tables
- Aggregate Tables
- Join Indexes
- Views
- Retro Copies (but not Retro Definitions)

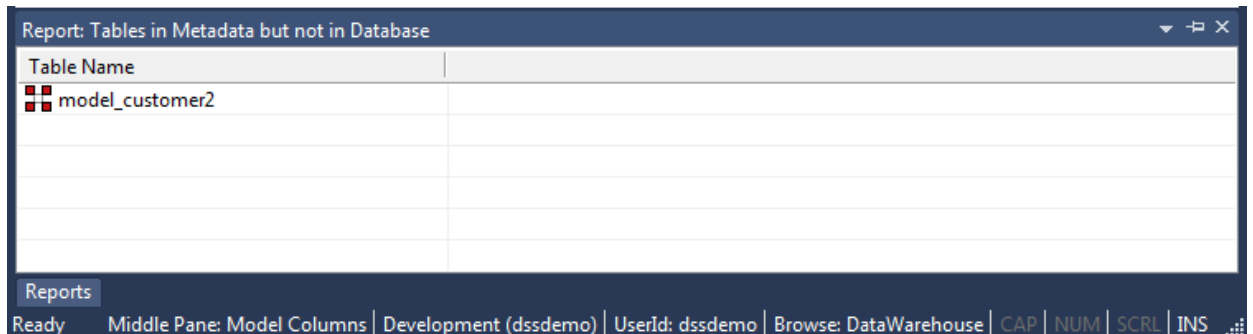
Parameters

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



| Table Name |
|-----------------|
| model_customer2 |
| |
| |
| |
| |

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 exists in the Teradata database but not in the metadata.

Objects Included

The following WhereScape RED object types are included in this report:

- Load Tables
- Stage Tables
- Model Tables
- Aggregate Tables
- Join Indexes
- Views
- 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 Teradata database not in the metadata.

Report Example



The screenshot shows a report window with a dark blue title bar containing the text "Report: Tables in Database but not in Metadata" and standard window controls (minimize, maximize, close). The main content is a table with a light gray header row and a white body. The header row has a single column labeled "Table Name". The first row of the body contains the text "load_promotions" next to a small red folder icon. Below this, there are several empty rows. At the bottom of the window, there is a dark blue footer bar with two tabs: "Reports" (highlighted in yellow) and "Results".

| Table Name |
|-----------------|
| load_promotions |
| |
| |
| |
| |
| |
| |
| |
| |
| |
| |
| |
| |
| |
| |
| |
| |
| |

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.

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

| Procedure/Script name |
|-----------------------|
| daily_date_roll |
| get_dim_date_key |
| |
| |
| |
| |
| |
| |
| |
| |
| |
| |

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 may be specified:

- Optional filter on Local Groups
- Optional filter on Local Projects
- Do detail report
- Only validate procedures that have been modified

Compare two metadata repositories

Compare the current metadata repository with the repository selected below. Enter the connection and logon information for the repository to be compared and press OK.

Odbc Connect TD 15 00

User Name dssdemo

Password *****

Meta Repository dssdemo

Local Groups:

Local Projects:

This is usually a two pass process. The first pass will report each object with validation errors. A more detailed report for listed objects can be obtained by selecting the objects and right clicking detail.

You can however go straight to the detail report by selecting the detail option below.

Detail Report Only

Validate Modified Procedures Only

NOTE: No sizing, tablespace, file group, date, description or documentation comparisons are performed.

WARNING: This will take a very long time for large repositories.

OK Cancel

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*)

The **result set is sortable** by clicking on the appropriate column heading.

Report Example

| Object Name | Comments | Local | Remote |
|------------------------|---|-------|--------|
| DataWarehouse | 5 validation errors | | |
| Sales | Connection name not found | | |
| Windows | 2 validation errors | | |
| Tutorial | Connection name not found | | |
| load_product | Load table not found in remote repository | | |
| load_product_sub_group | Load table not found in remote repository | | |
| load_product_group | Load table not found in remote repository | | |
| load_continent | Load table not found in remote repository | | |
| load_order_line | Load table not found in remote repository | | |
| load_order_header | Load table not found in remote repository | | |
| load_customer | Load table not found in remote repository | | |
| load_country | Load table not found in remote repository | | |
| load_product_line | Load table not found in remote repository | | |

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.

TERADATA: VIEW OF MODEL VALIDATE

This report shows any views built on model tables that do not have the same columns or column properties.

Objects Included

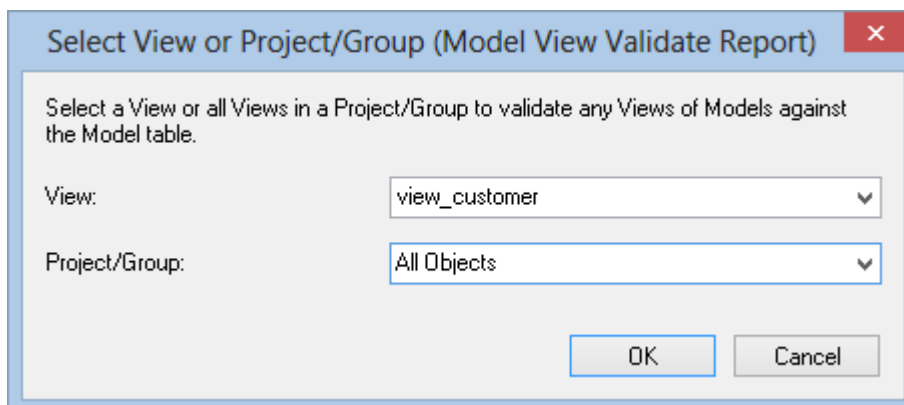
The following WhereScape RED object types are included in this report:

- Views

Parameters

This report has two alternate parameters:

- View Name **OR**
- Project/Group/All Objects



Select View or Project/Group (Model View Validate Report) ✕

Select a View or all Views in a Project/Group to validate any Views of Models against the Model table.

View:

Project/Group:

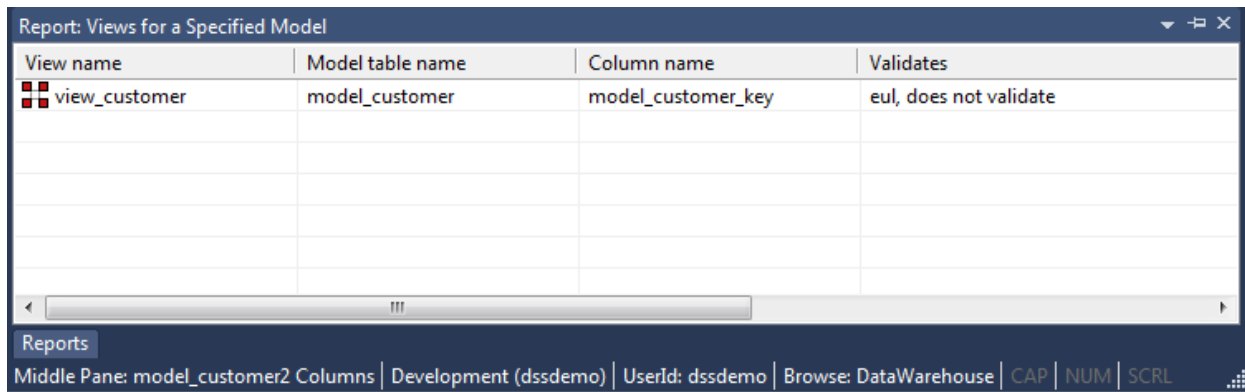
Results

The results of this report are displayed as a list of differing view columns with the following columns:

- View Name
- Model Name (*the name of the model table the view is based on*)
- Column Name (*the column where a difference exists*)
- Validates (*further details of the difference*)

The result set is sortable by clicking on the appropriate column heading.

Report Example



| View name | Model table name | Column name | Validates |
|---------------|------------------|--------------------|------------------------|
| view_customer | model_customer | model_customer_key | eul, does not validate |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |

Reports
Middle Pane: model_customer2 Columns | Development (dssdemo) | UserId: dssdemo | Browse: DataWarehouse | CAP | NUM | SCRL

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
- Model 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

| Table name | Column name |
|------------|--------------------|
| dim_date | calendar_date |
| model_date | calendar_date |
| dim_date | cal_day_in_month |
| model_date | cal_day_in_month |
| dim_date | cal_day_in_week |
| model_date | cal_day_in_week |
| dim_date | cal_day_in_week_no |
| model_date | cal_day_in_week_no |
| dim_date | cal_day_in_year |
| model_date | cal_day_in_year |

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.

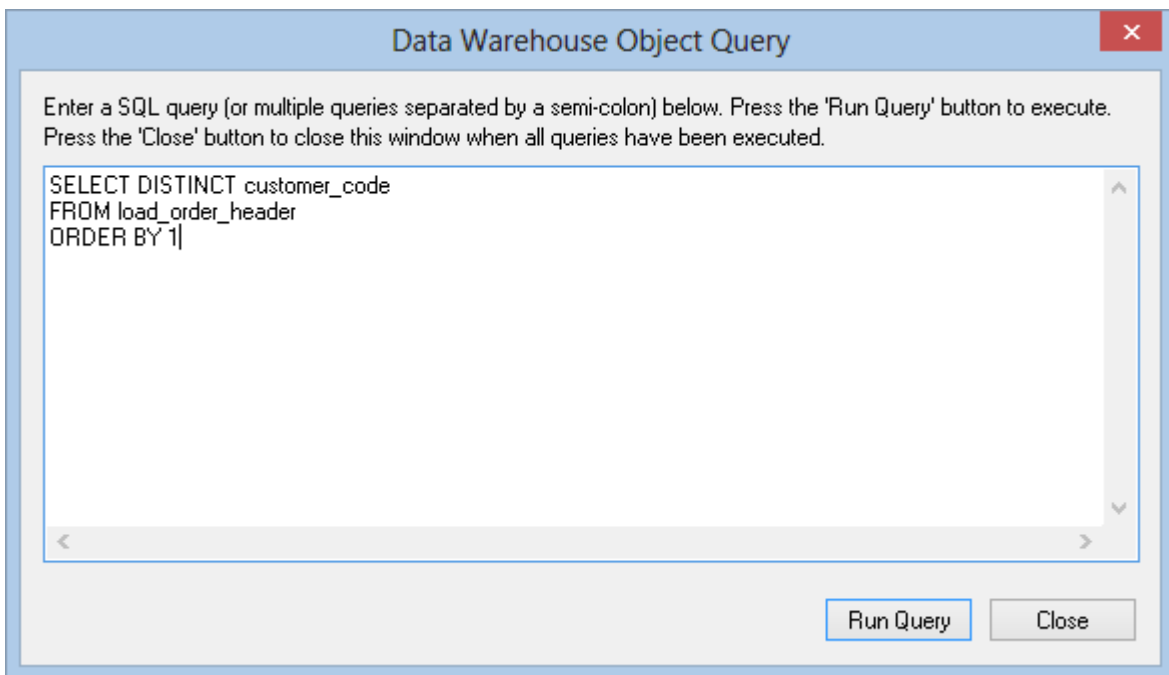
Objects Included

All object types and data warehouse tables can be included in this report.

Parameters

This report has one parameter:

- The SQL Query



Data Warehouse Object Query

Enter a SQL query (or multiple queries separated by a semi-colon) below. Press the 'Run Query' button to execute. Press the 'Close' button to close this window when all queries have been executed.

```
SELECT DISTINCT customer_code  
FROM load_order_header  
ORDER BY 1
```

Run Query

Close

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



The screenshot shows a window titled "Data Warehouse Query" with a standard Windows-style title bar (minimize, maximize, close buttons). The window contains a table with a single row and one column. The value in the cell is "21". The table has a header row and a footer row. The footer row contains two tabs: "Reports" and "Results".

| 21 |
|----|

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.

CHAPTER 31

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

IN THIS CHAPTER

| | |
|---------------------------------|-----|
| Applications | 857 |
| Importing Object Metadata | 865 |
| Importing Language Files..... | 867 |
| Data Warehouse Testing | 868 |

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. A Teradata application can only be loaded into a Teradata metadata repository, etc).

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 they are listed in the application. This way if there are procedure dependencies, ensure 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

Build Deployment Application

Application
Objects to Add/Replace
Objects to Delete

This process builds the files necessary to allow the deployment of a data warehouse solution. These files are read and processed by the Setup Administrator utility. Specify the application identification details and then select the objects to be deployed to and/or deleted from another repository.

Output Directory: C:\Training\RED\Module17

Application Identifier: Orders Application Version: 1

Application Name: Sales Orders Project Release

Description: First Release of Orders into Production

You can select or enter a previous application file as a starting point for this application.

Previous Application:

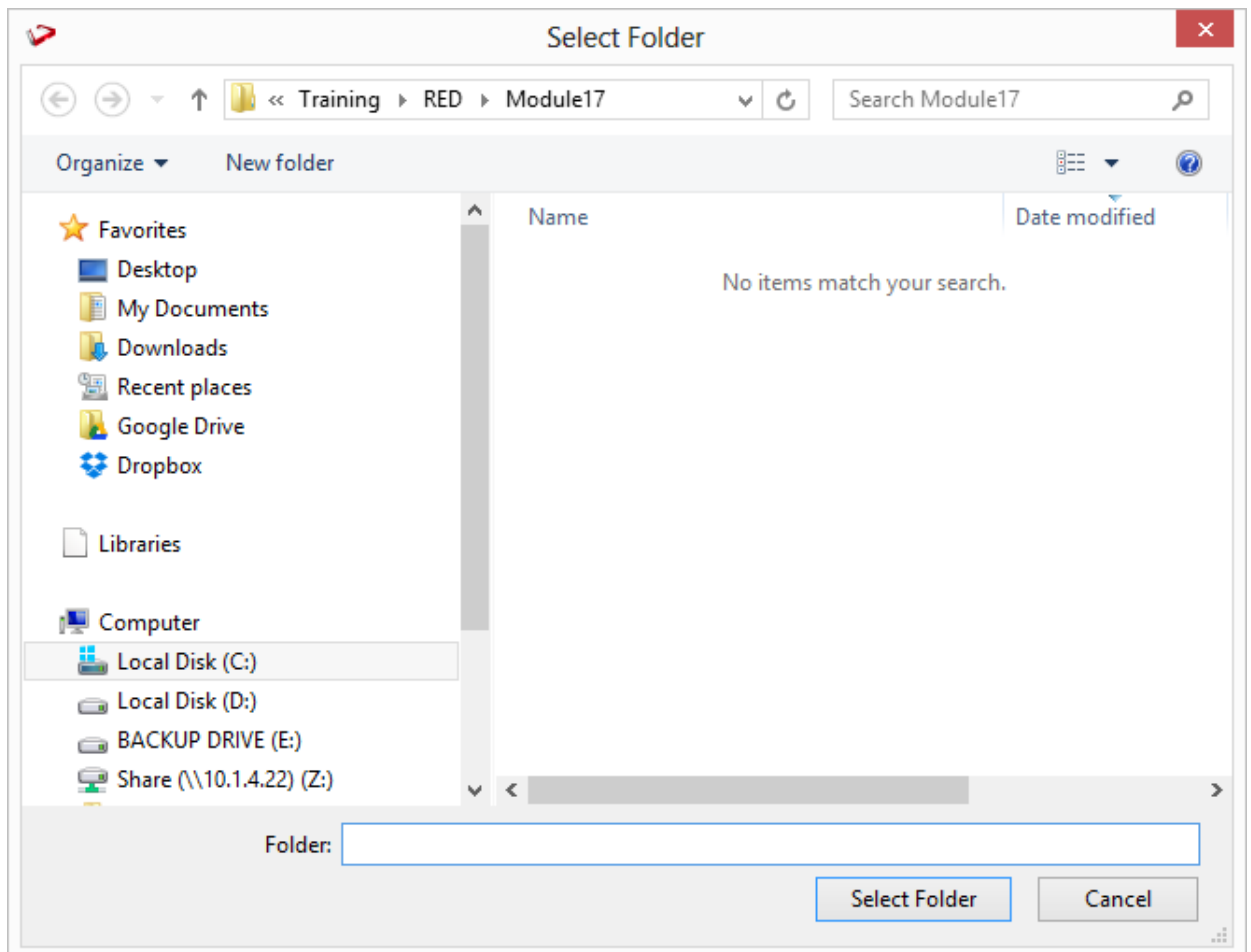
Pre Application Load SQL. (the following optional SQL statement will be issued before the application load commences):

Post Application Load SQL. (the following optional SQL statement will be issued after the application load completes):

Output Directory

The directory to which the application files will be written. By default this will be the WhereScape program directory.

- 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

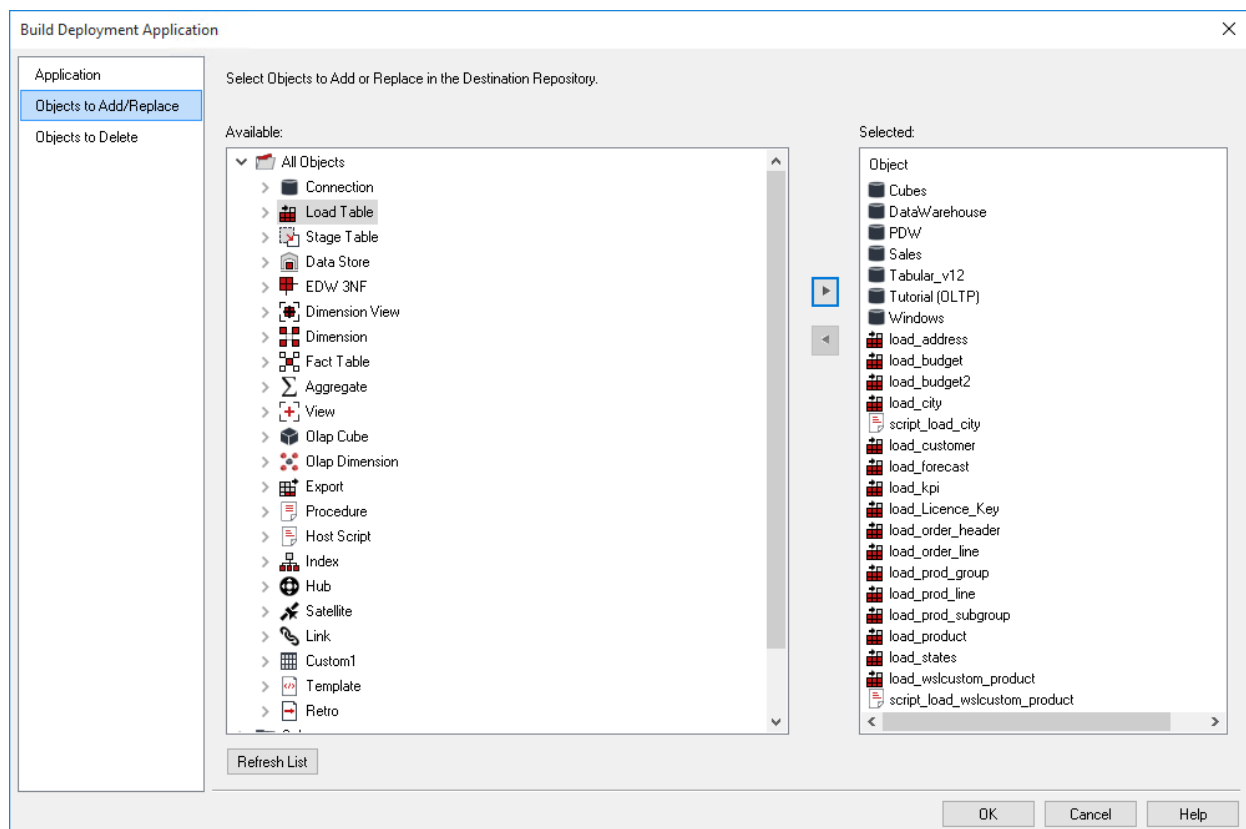
Objects can be moved from the left object tree by double clicking on an object name or by using the > button. This tab allows you to select the objects to add or replace in the destination repository.

NOTE: Maximum number of objects in an application

5000 objects (including jobs)

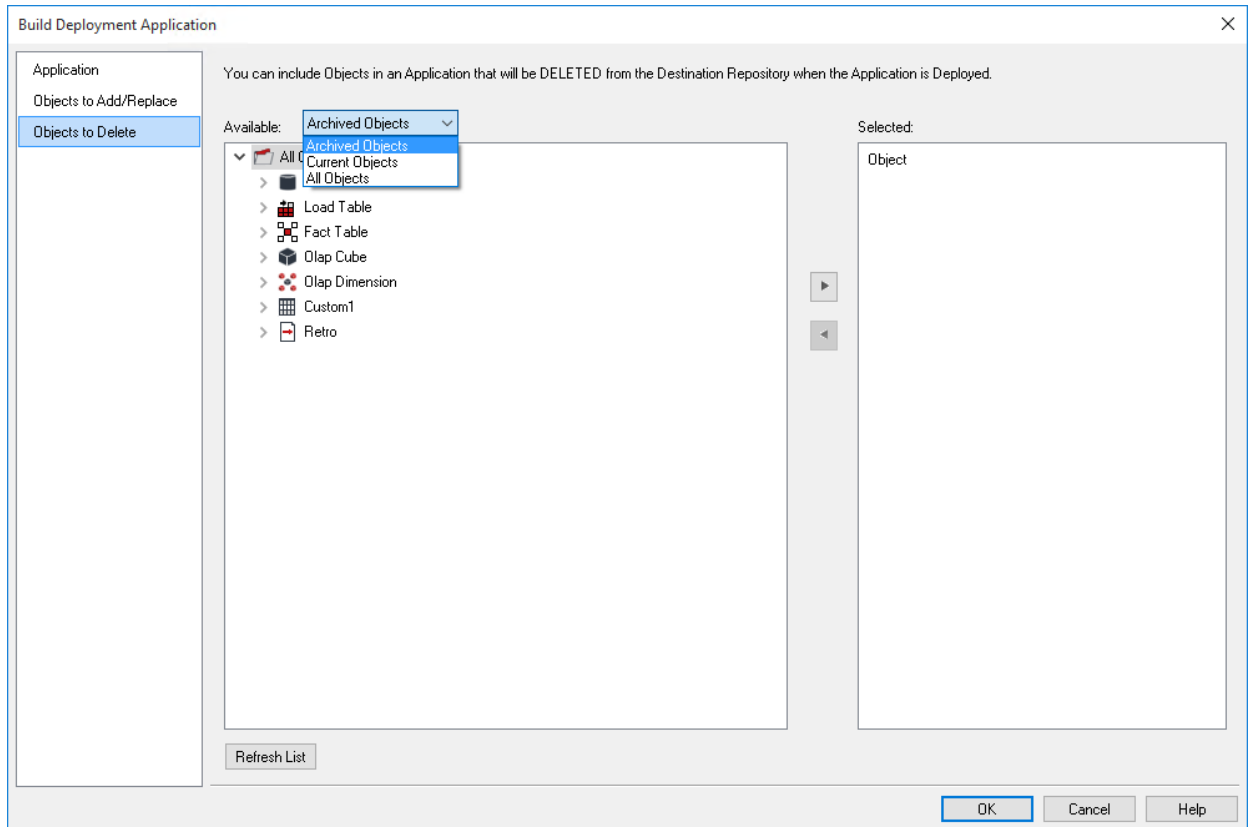
2000 source views of views

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 were created. (e.g. a Teradata application can only be loaded into a Teradata 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 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.

Import Objects from Another Meta Repository [X]

You have chosen to import objects from another MetaRepository. Two methods are supported, and described below:

IMPORT, imports objects without overwriting any existing objects. You are given the option of renaming the new objects if in conflict.

REFRESH, imports objects replacing any existing object of the same name. A version of the replaced object is created.

Enter the uppercase word **IMPORT** or **REFRESH** in the first window, a username and password with access rights to the source repository and the source MetaRepository to import from.

Odbc Connect

User Name

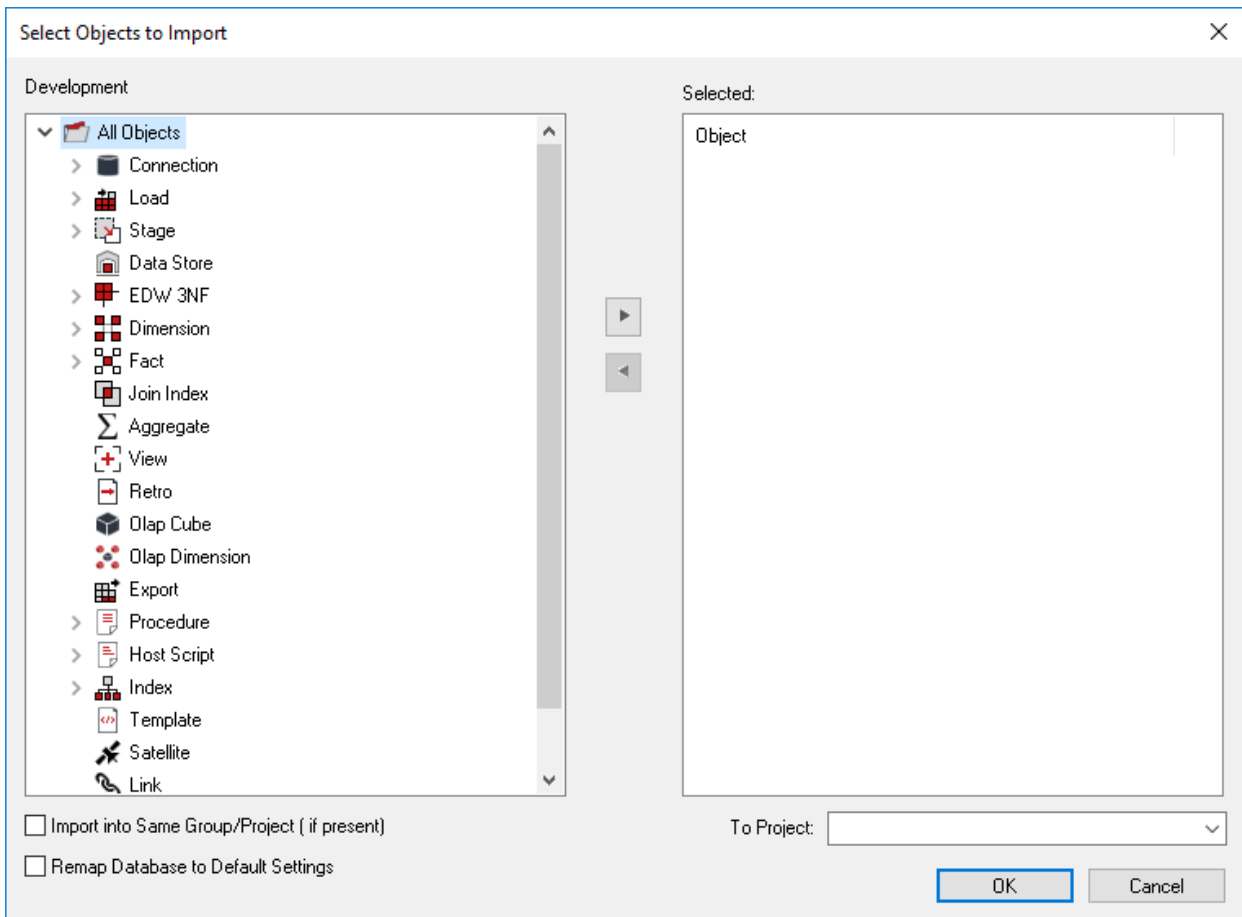
Password

Meta Repository

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 a Teradata application can only be loaded into a Teradata 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.

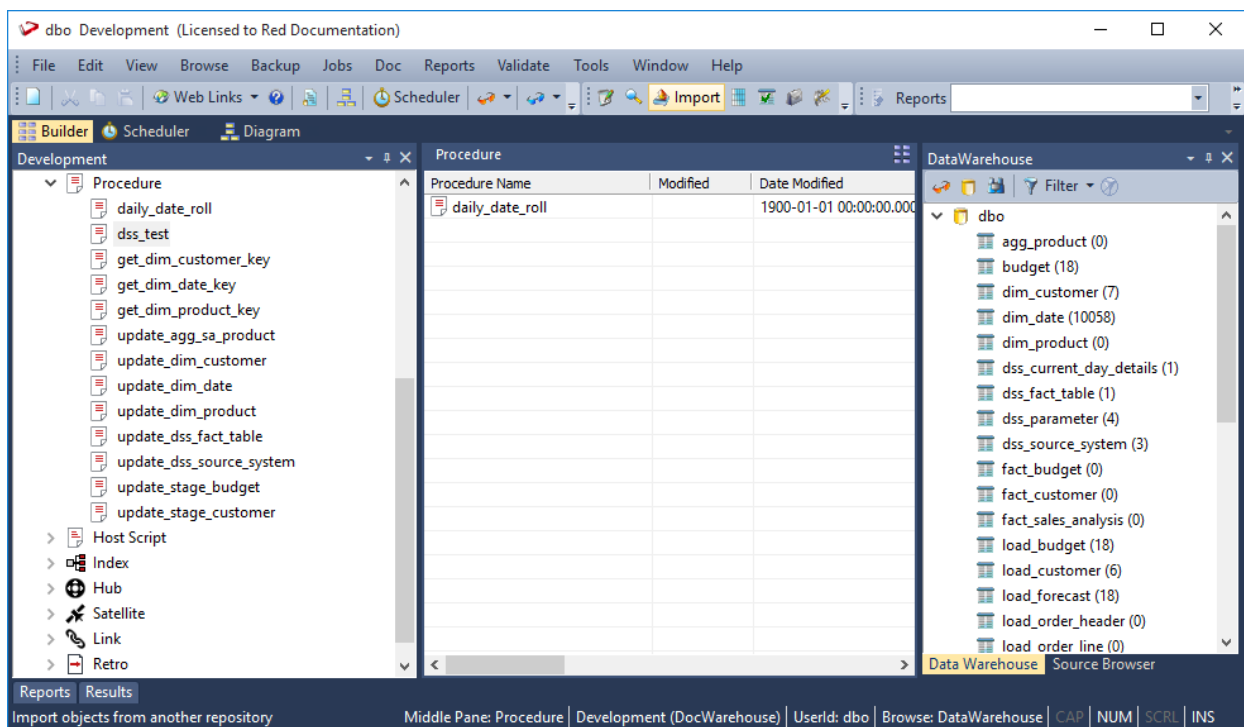
DATA WAREHOUSE TESTING

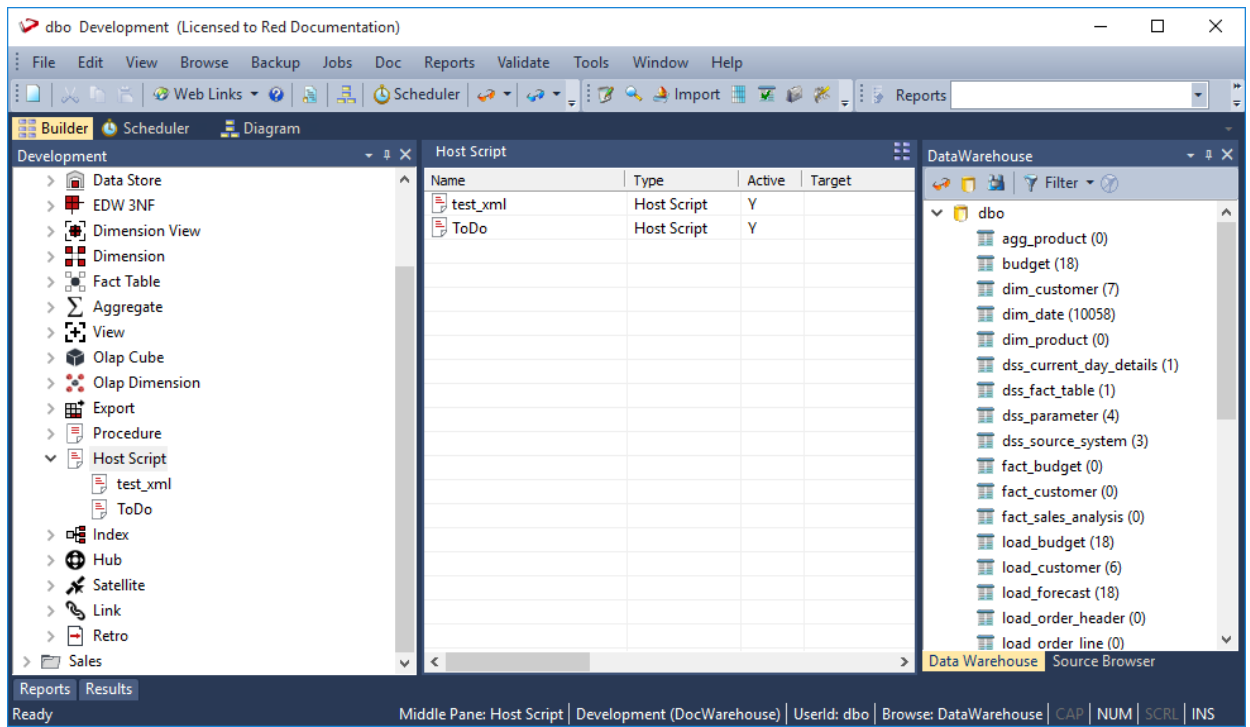
Note: Applications can only be loaded into the same relational database type from which they were created. (For example, a Teradata application can only be loaded into a Teradata database).

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.

Job Definition ✕

Job Name:

Description:

Frequency:

Start Date:

Start Time:

Maximum Threads:

Scheduler:

Dependent On:

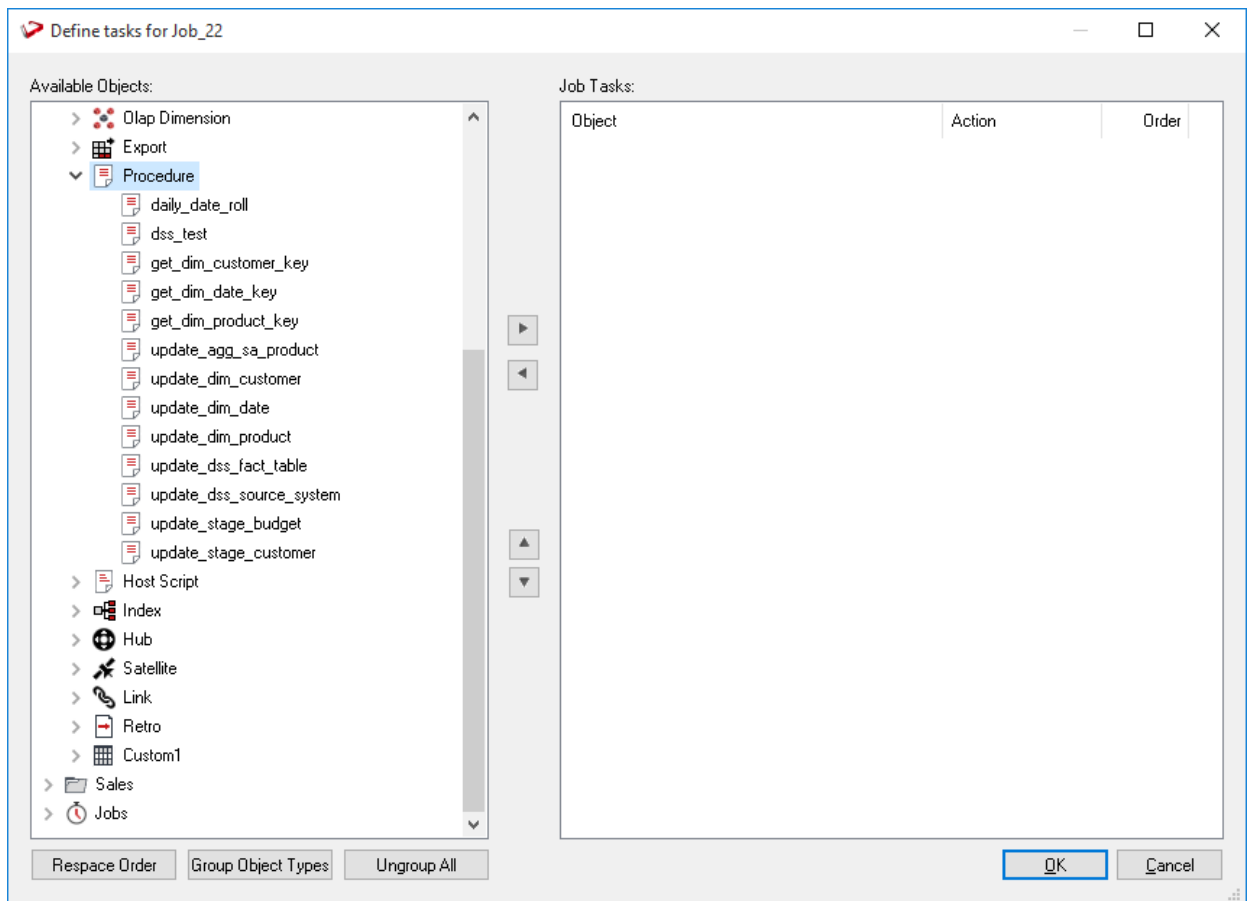
Logs Retained: This field lets you set the number of logs that are retained for this job before an automatic delete and archive occurs. 0 = keep all logs (default action)

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:

- 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**.

CHAPTER 32

BACKING UP AND RESTORING METADATA

This chapter covers the moving, 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.

Various methods exist for getting new or changed metadata from one repository into another repository. These methods are covered in the applications, and managing multiple repositories' sections.

IN THIS CHAPTER

| | |
|--------------------------------|-----|
| Backup using DB Routines | 873 |
| Restoring DB Backups | 875 |
| Unloading Metadata | 876 |
| Loading an Unload..... | 878 |

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 using the main WhereScape RED tool.

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.

Teradata Windows Backups

A Teradata Arcmain based backup can be taken from the WhereScape RED tool by selecting the **Backup/Export the metadata (Teradata Arc)** menu option. This option assumes that the Teradata Client on the PC is the same version as that of the Teradata database where the RED meta repository is stored. The backup may not work if the versions differ.

When executed this menu option attempts to locate the Teradata Arc utility. This utility is normally called 'arcmain'. If 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 |
|-----------------|---|
| wsl_arc_out.bat | Windows command file containing the arc command. |
| wsl_arc_out.log | Log file of the arcmain session. |
| wsl_arc_out.ctl | Control file for the arcmain session. Contains an entry for each meta table to be exported. |
| wsl_arc_in.bld | Template for building the Windows command file used to restore the metadata. |
| wsl_arc_in.cmd | Template for building the arcmain control file used to restore the metadata. |
| wsl_arc.dbl | Empty file for Teradata. |

| File name | Purpose |
|--------------------|---|
| wsl_arc.drp | Command file for dropping all metadata tables. (Used in the restore prior to a reload). |
| wsl_arc.pro | A file containing each procedure stored in the metadata. |
| wsl_arc.seq | Empty file for Teradata. |
| RED0001 -> REDnnnn | Teradata Arcmain files, one for each RED metadata table being backed up. |

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.

WhereScape Unload and Load

The remaining Windows backup option is the WhereScape generic unload and load options. These menu options can be used when the metadata must be sent back to support to help resolve a problem. These options have the advantage that they are database, and database version independent, so can be used to backup the metadata regardless of the version of the database client running on the PC. It is possible to Load the metadata from a different database. For example, the meta data from a Teradata unload can be loaded into an Oracle or SQL Server database. There is, however, additional work in order to successfully move the metadata in this fashion. If such a move is required please contact WhereScape support.

RESTORING DB BACKUPS

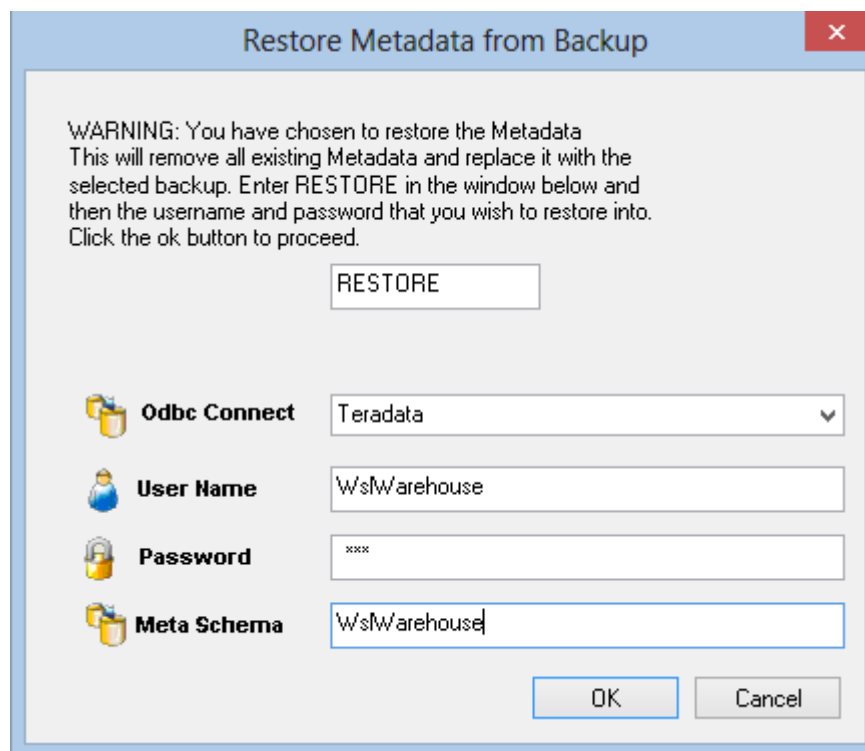
WhereScape RED metadata can be restored from a prior backup. The restore takes place through the WhereScape RED tool.

Note: If transferring the export files via ftp, ensure that the REDxxxx files are transferred in binary mode and that the other files are transferred in ASCII mode.

Teradata Windows Restore

Select the menu option **Backup/Restore Metadata (from Teradata Arc)** to begin the restore process.

A dialog box will appear. The word RESTORE needs to be entered along with the username 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. The metabase is the database where the meta repository exists.



Restore Metadata from Backup

WARNING: You have chosen to restore the Metadata
This will remove all existing Metadata and replace it with the selected backup. Enter RESTORE in the window below and then the username and password that you wish to restore into. Click the ok button to proceed.

RESTORE

Odbc Connect: Teradata

User Name: WslWarehouse

Password: xxxx

Meta Schema: WslWarehouse

OK Cancel

Once the **OK** button is clicked a new dialog box will appear asking for a selection of the export files. Browse to the directory where the export is located and select the wsl_arc_in.bld file. Once the wsl_arc_in.bld 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.

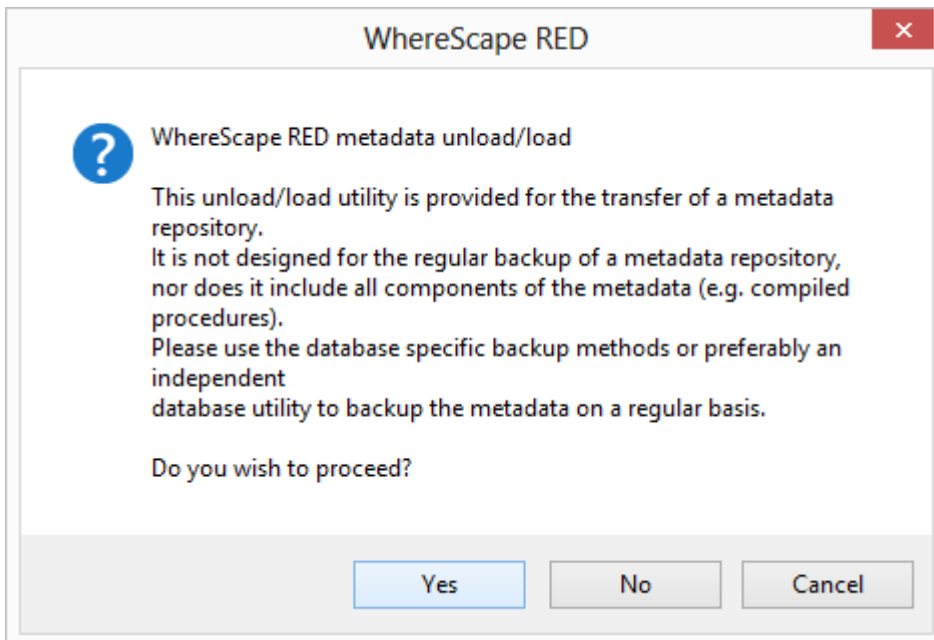
After performing a cross platform unload and load:

- 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.

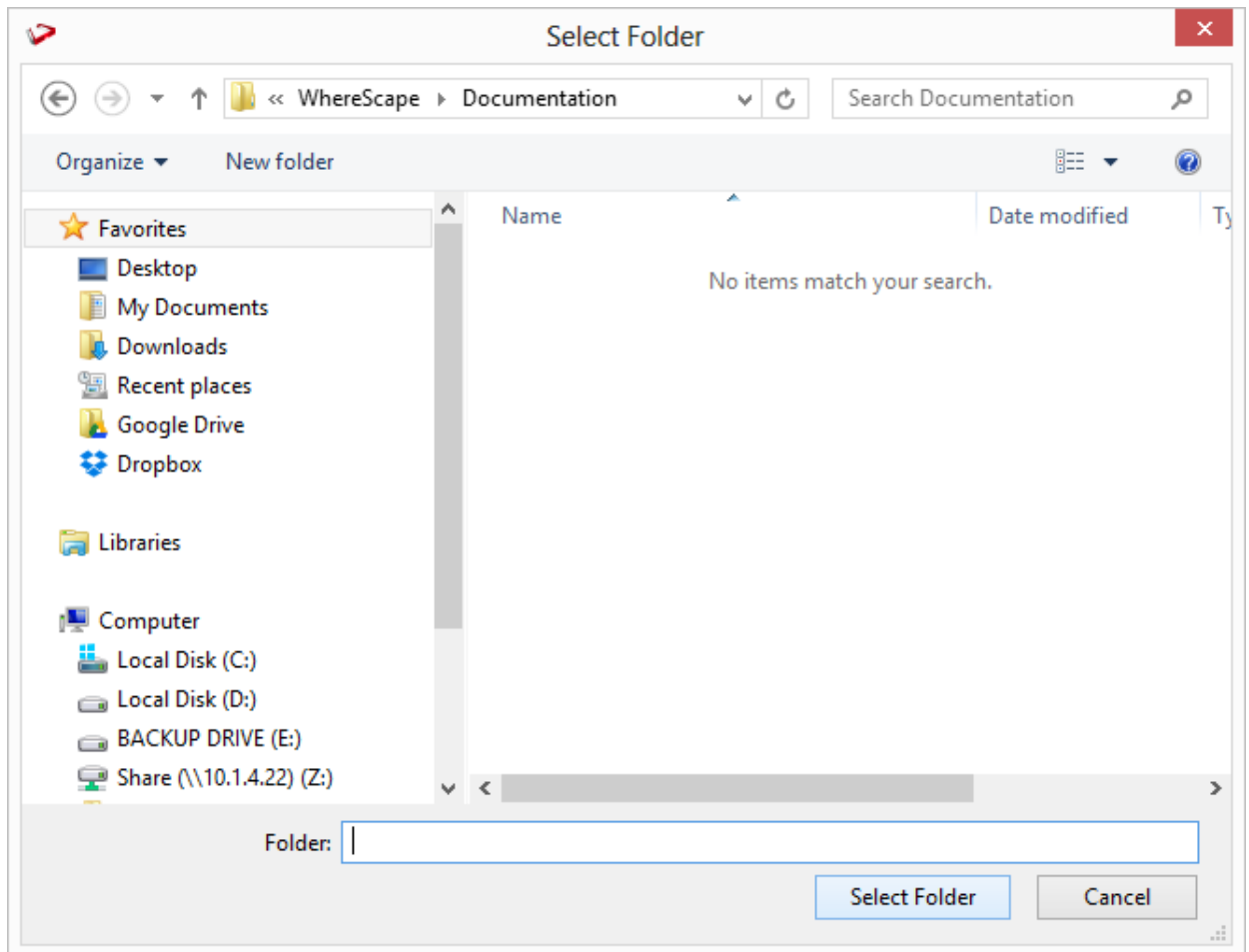
Performing an Unload

An unload can be performed within the WhereScape RED tool by selecting the **Backup/Unload the metadata to disk** menu option.

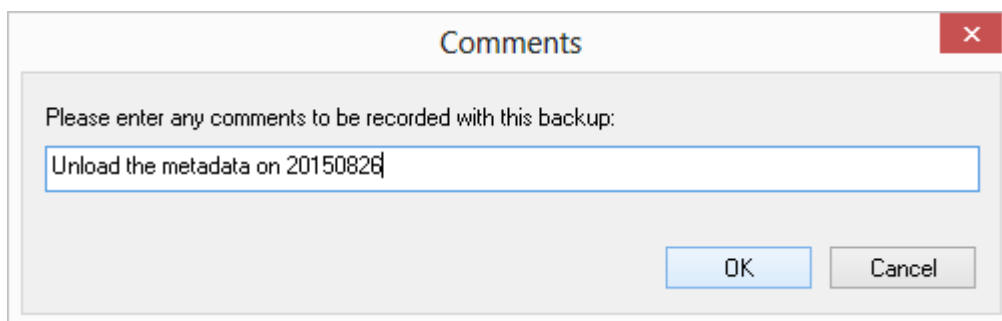
A popup window asks for confirmation to proceed with the unload. Click **Yes**.



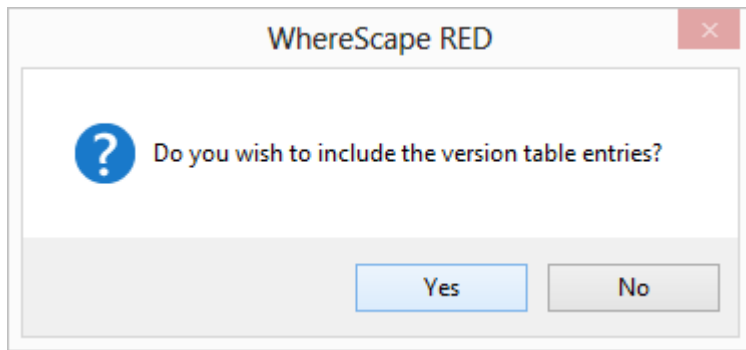
An additional pop-up window asks for a file name for the export. A directory should be chosen and a name entered. Click **Save**.



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.

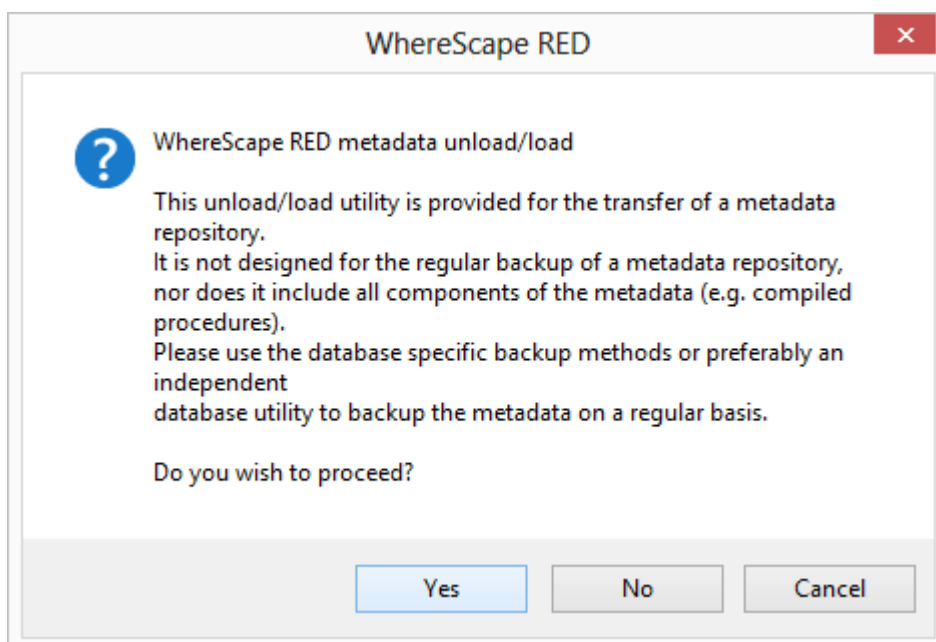
LOADING AN UNLOAD

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 username 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**.

Restore Metadata from Backup

WARNING: You have chosen to restore the Metadata
This will remove all existing Metadata and replace it with the
selected backup. Enter RESTORE in the window below and
then the username and password that you wish to restore into.
Click the ok button to proceed.

RESTORE

Odbc Connect WsWarehouse_TD

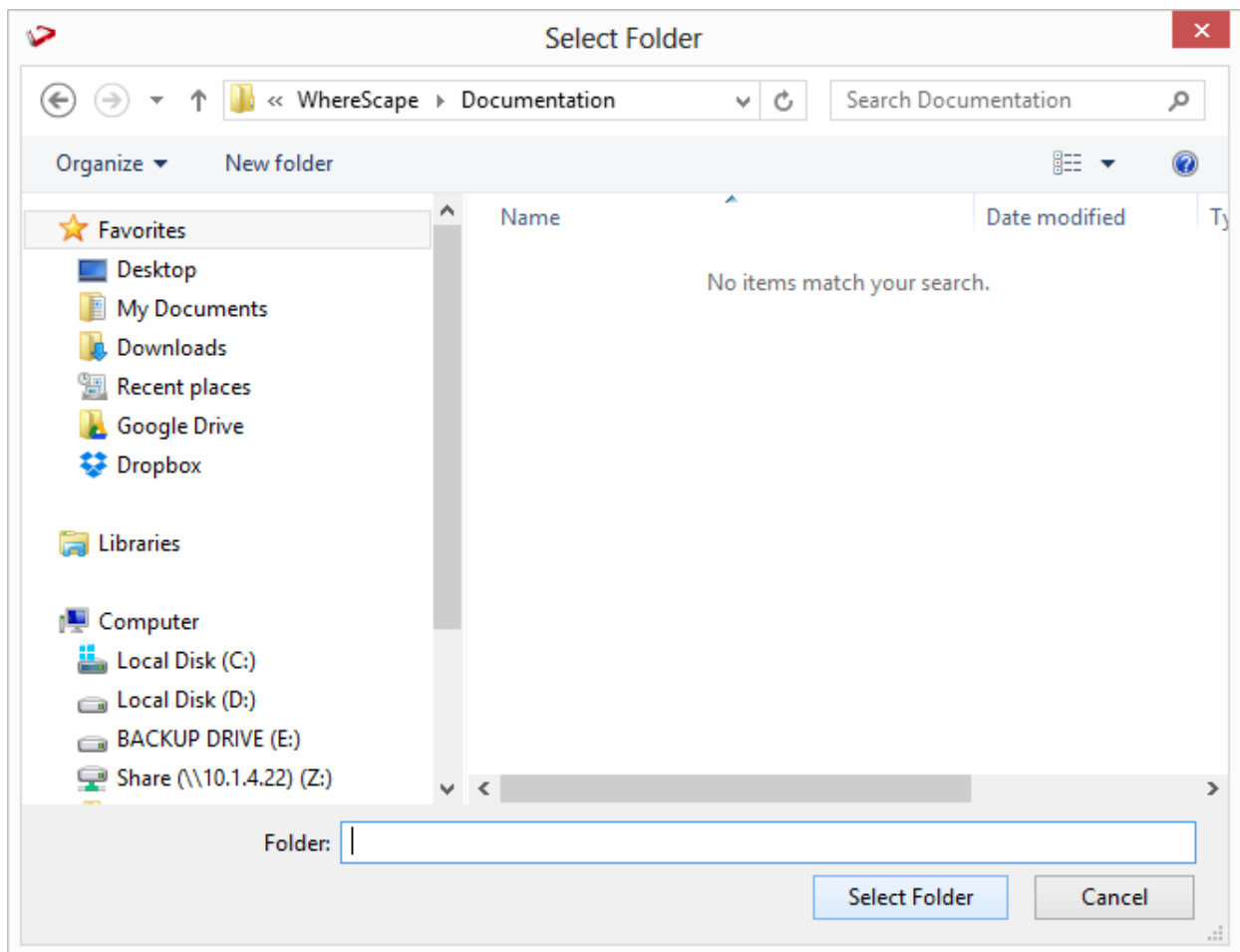
User Name dssdemo

Password *****

Meta Database dssdemo

OK Cancel

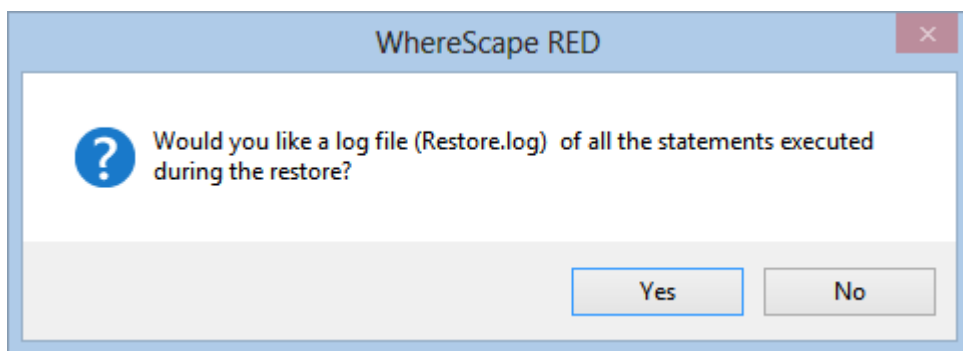
The next dialog box ask for the folder for the metadata to be loaded from. Browse to the contents of the directory where the unload is located and click **Select Folder**.



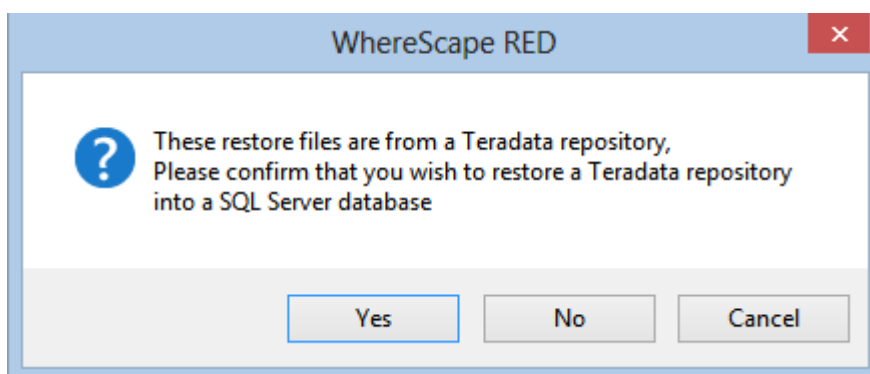
Confirm the load will overwrite the existing repository by clicking **Yes**:



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.

CHAPTER 33

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..... | 883 |
| Validating Source (Load) Tables..... | 885 |
| Validating Procedures | 886 |
| Altering Tables | 887 |
| Validating Indexes..... | 889 |
| Recompiling Procedures..... | 889 |

VALIDATING TABLES

The metadata as stored and maintained by 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.

| Table Name | Results |
|--------------------|---|
| load_customer | Validates OK |
| load_forecast | Validates OK |
| load_order_header | Validates OK |
| load_order_line | Validates OK |
| load_state | Table:meta->load_state column data not found, |
| model_customer | dss-> address - varchar(40), |
| model_date | Validates OK |
| model_forecast | meta-> product_line - char(20), |
| model_order_header | Validates OK |
| model_order_line | Validates OK |
| ods_forecast | Validates OK |
| stage_customer | Validates OK - but different column order |
| stage_forecast | Validates OK |
| stage_order_header | Validates OK |
| stage_order_line | Validates OK |

Reports
Middle Pane: model_customer Columns | Development (dssdemo) | UserId: dssdemo | Browse: No source system | CAP | NUM | SCRL

In this example we see five different scenarios.

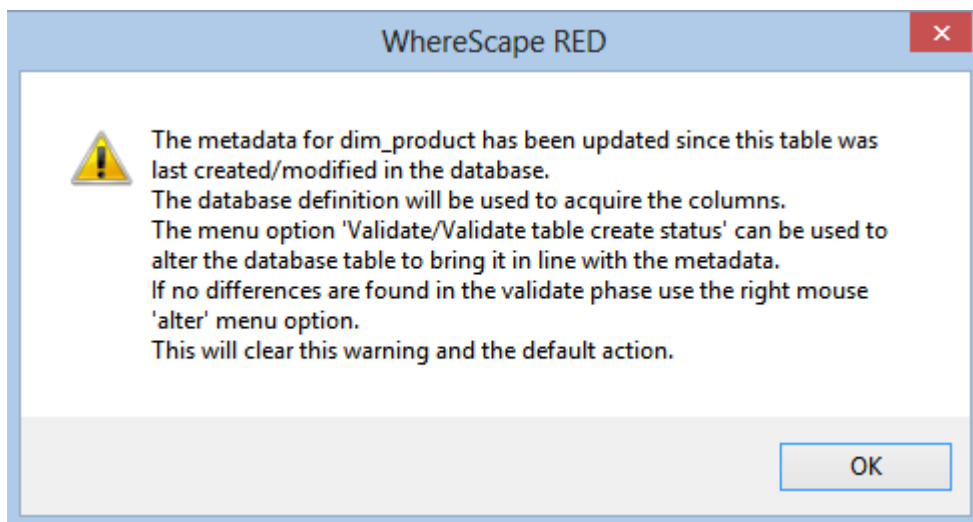
- 1 The metadata for table **load_customer** matches the physical table in the database.
- 2 The metadata for the table **model_forecast** does not match the physical table. The metadata has an additional column called 'product_line'. This column was not found in the physical table. The table can be altered if desired. See the next section on Altering Tables.
- 3 The physical database table **model_customer** has an additional column not found in the metadata. The column is 'address'. The table can be altered if desired. See the next section on Altering Tables.
- 4 The table **stage_customer** 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 alter of the table. The table must be re-created if the order is important.

- 5 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. model_table to create view_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:

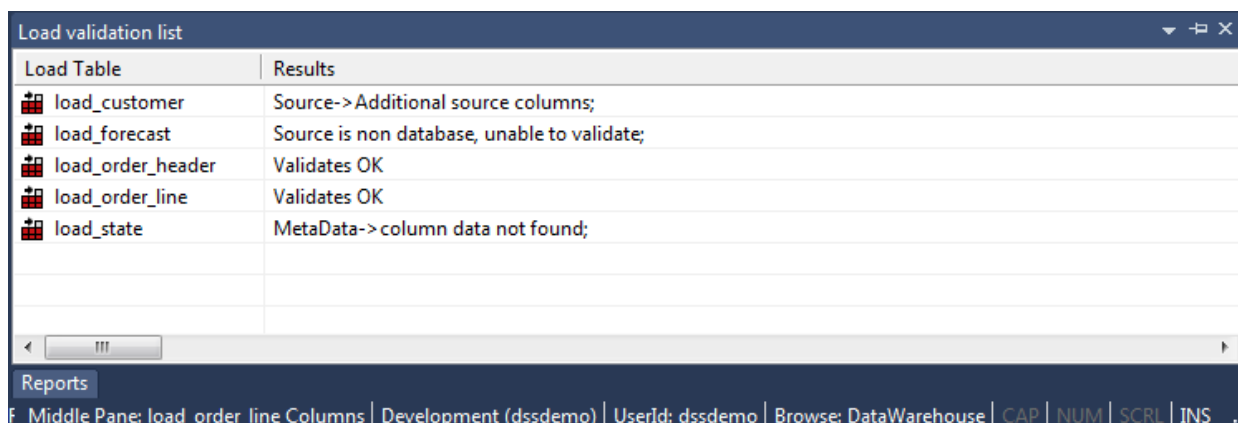


If a subsequent validate of the table in question shows that it validates, this message will mean that the dates are somehow out of sync. This can occur for example after an import where the metadata has been replaced, but the underlying table still matches the metadata. Another common occurrence is where a new column is added and then deleted. To prevent the message from re-occurring in such a situation, proceed as follows.

Use the right-click menu to 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:



| Load Table | Results |
|-------------------|---|
| load_customer | Source-> Additional source columns; |
| load_forecast | Source is non database, unable to validate; |
| load_order_header | Validates OK |
| load_order_line | Validates OK |
| load_state | MetaData-> column data not found; |

Reports
 Middle Pane: load_order_line Columns | Development (dssdemo) | UserId: dssdemo | Browse: DataWarehouse | CAP | NUM | SCRL | INS

The table **load_forecast** is a Windows file load and as such cannot be validated.

The table **load_customer** 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 (stage and model) 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.

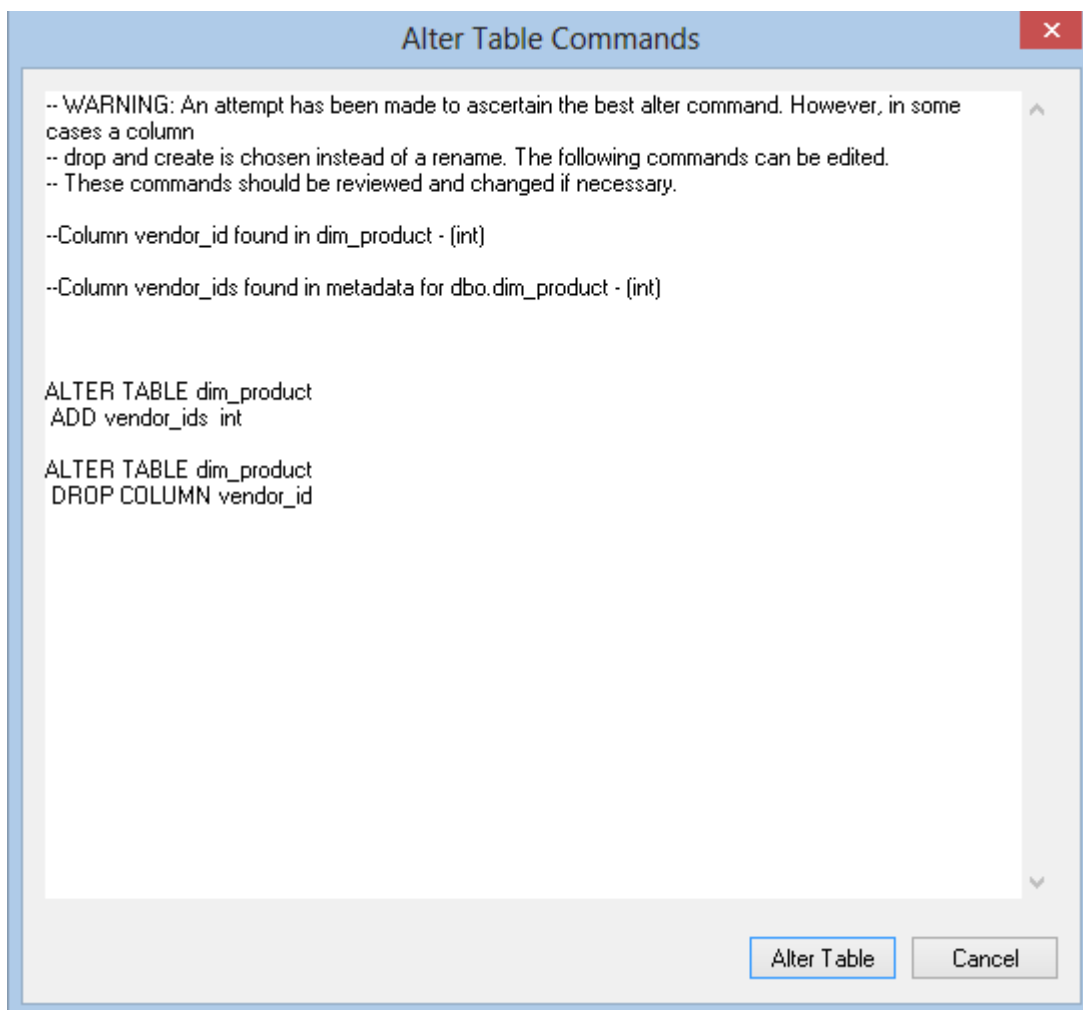
Where a procedure fails to compare the Procedure, the Editor must be used to find the actual differences. Start the editor by double clicking on the procedure name in the left pane. Use the **Tools/Compare to User Source** menu option to display the differences between the procedure in the metadata and 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 the right-click menu option from the object name. Then in the middle pane (the validation listing) select the table that has not validated. Position on the table name and using the right mouse, select the **Alter Table** pop-up menu option. 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. The new column 'State' will be added to the table. Comments at the top of the screen, show 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.

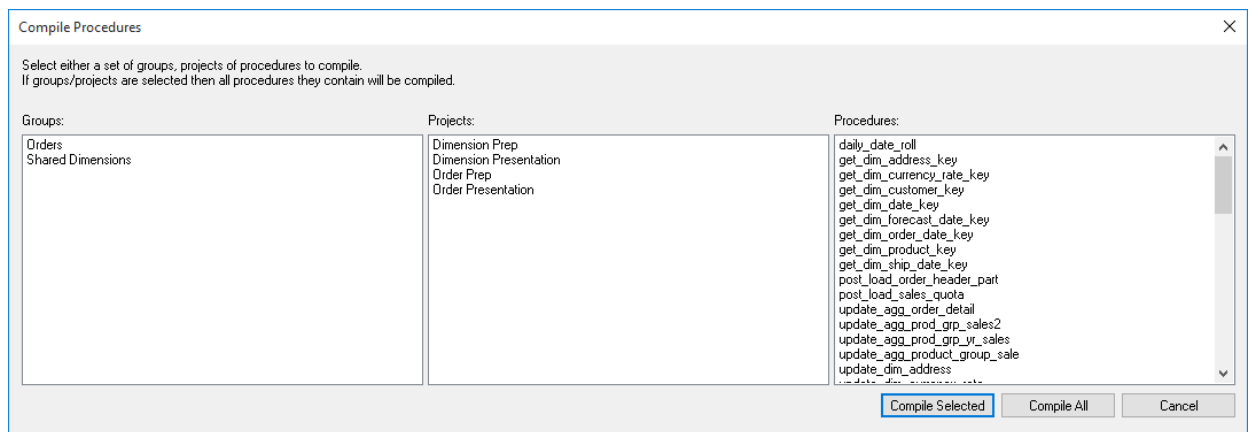
Clicking the **Alter Table** button will proceed to alter the table. In effect, it will execute any command in the window.

VALIDATING INDEXES

Index validate is not available on Teradata.

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:



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.

Compile Selected

A selected group of procedures may be compiled. Selection is done via standard Windows multi selection using the shift and CTRL keys. By selecting projects or groups, it is possible to compile all procedures associated with the projects or groups. Click on the **Compile Selected** button to compile/re-compile the selected procedures.

Compile All

Click on the **Compile All** button to compile/re-compile all procedures in the metadata repository.

Compile Missing

The **Compile Missing** button compiles all procedures in the metadata repository that do not exist in the metadata database in Teradata.

CHAPTER 34

CALLABLE ROUTINES

IN THIS CHAPTER

| | |
|--|-----|
| Introduction to Callable Routines..... | 892 |
| Ws_Api_Glossary | 900 |
| Ws_Connect_Replace..... | 902 |
| Ws_Job_Abort | 904 |
| Ws_Job_Clear_Archive | 906 |
| Ws_Job_Clear_Logs | 908 |
| Ws_Job_Clear_Logs_By_Date..... | 911 |
| Ws_Job_Create | 914 |
| Ws_Job_CreateWait | 918 |
| Ws_Job_Dependency..... | 922 |
| Ws_Job_Release | 925 |
| Ws_Job_Restart | 928 |
| Ws_Job_Schedule | 931 |
| Ws_Job_Status | 934 |
| Ws_Load_Change..... | 939 |
| Ws_Maintain_Indexes..... | 943 |
| Ws_Version_Clear..... | 946 |
| WsParameterRead | 949 |
| WsParameterReadF | 950 |
| WsParameterReadG..... | 952 |
| WsParameterWrite | 954 |
| WsWrkAudit | 955 |
| WsWrkAuditBulk..... | 957 |
| WsWrkError..... | 961 |
| WsWrkErrorBulk..... | 963 |
| WsWrkTask..... | 967 |

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 |
|---|---|
| <i>Ws_Api_Glossary</i> (on page 900) | Adds an entry to the documentation glossary. |
| <i>Ws_Connect_Replace</i> (on page 902) | Replaces the contents of a connection with details from another connection. |
| <i>Ws_Job_Abort</i> (on page 904) | Aborts a job if it is in a running state. |
| <i>Ws_Job_Clear_Archive</i> (on page 906) | Purges archived job logs that are older than the specified age in days. |
| <i>Ws_Job_Clear_Logs</i> (on page 908) | Archives job logs when the maximum number of current logs to retain is exceeded. |
| <i>Ws_Job_Clear_Logs_By_Date</i> (on page 911) | Archives job logs that are older than the specified age in days. |
| <i>Ws_Job_Create</i> (on page 914) | Creates a job based on an existing job and optionally starts it immediately. |
| <i>Ws_Job_CreateWait</i> (on page 918) | Creates a job based on an existing job and schedules it to start later. |
| <i>Ws_Job_Dependency</i> (on page 922) | Adds or removes a child-to-parent dependency between two jobs to control the child job. |
| <i>Ws_Job_Release</i> (on page 925) | Starts a job if it is in a holding or waiting state. |
| <i>Ws_Job_Restart</i> (on page 928) | Starts a job if it is in a failed state. |
| <i>Ws_Job_Schedule</i> (on page 931) | Schedules a job if it is in a holding or waiting state. |
| <i>Ws_Job_Status</i> (on page 934) | Returns the current status of a job. |
| <i>Ws_Load_Change</i> (on page 939) | Changes the Connection or Schema of a load table. |
| <i>Ws_Maintain_Indexes</i> (on page 943) | Drops and/or builds database indexes that are defined in the WhereScape RED metadata. |

| Routine Name | Description |
|--|---|
| <i>Ws_Version_Clear</i> (on page 946) | Purges metadata versions for all objects that do not meet the specified retention criteria. |
| <i>WsParameterRead</i> (on page 949) | Returns the value and comment (for most RDBMS) of a WhereScape RED metadata Parameter. |
| <i>WsParameterReadF</i> (on page 950) | Returns the value of a WhereScape RED metadata Parameter. [SQL Server only] |
| <i>WsParameterReadG</i> (on page 952) | Returns the value of a "global" WhereScape RED metadata Parameter that relates to a load table. |
| <i>WsParameterWrite</i> (on page 954) | Updates the value and comment of a WhereScape RED metadata Parameter or creates it. |
| <i>WsWrkAudit</i> (on page 955) | Records a message in the Audit Log. |
| <i>WsWrkAuditBulk</i> (on page 957) | Records multiple messages in the Audit Log. |
| <i>WsWrkError</i> (on page 961) | Records a message in the Error/Detail Log. |
| <i>WsWrkErrorBulk</i> (on page 963) | Records multiple messages in the Error/Detail Log. |
| <i>WsWrkTask</i> (on page 967) | 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 | Teradata |
|--|-----------|
| <i>Ws_Api_Glossary</i> (on page 900) | Procedure |
| <i>Ws_Connect_Replace</i> (on page 902) | Procedure |
| <i>Ws_Job_Abort</i> (on page 904) | Procedure |
| <i>Ws_Job_Clear_Archive</i> (on page 906) | Procedure |
| <i>Ws_Job_Clear_Logs</i> (on page 908) | Procedure |
| <i>Ws_Job_Clear_Logs_By_Date</i> (on page 911) | Procedure |
| <i>Ws_Job_Create</i> (on page 914) | Procedure |
| <i>Ws_Job_CreateWait</i> (on page 918) | Procedure |
| <i>Ws_Job_Dependency</i> (on page 922) | Procedure |
| <i>Ws_Job_Release</i> (on page 925) | Procedure |
| <i>Ws_Job_Restart</i> (on page 928) | Procedure |
| <i>Ws_Job_Schedule</i> (on page 931) | Procedure |
| <i>Ws_Job_Status</i> (on page 934) | Procedure |
| <i>Ws_Load_Change</i> (on page 939) | Procedure |
| <i>Ws_Maintain_Indexes</i> (on page 943) | Procedure |

| Routine Name | Teradata |
|--|-----------------|
| <i>Ws_Version_Clear</i> (on page 946) | Procedure |
| <i>WsParameterRead</i> (on page 949) | Procedure |
| <i>WsParameterReadF</i> (on page 950) | Procedure |
| <i>WsParameterReadG</i> (on page 952) | Procedure |
| <i>WsParameterWrite</i> (on page 954) | Procedure |
| <i>WsWrkAudit</i> (on page 955) | Procedure |
| <i>WsWrkAuditBulk</i> (on page 957) | Procedure |
| <i>WsWrkError</i> (on page 961) | Procedure |
| <i>WsWrkErrorBulk</i> (on page 963) | Procedure |
| <i>WsWrkTask</i> (on page 967) | 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. |

| Input | Parameter Name | Description |
|-----------------|----------------|---|
| 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. |
| 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 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:

-- ODBC Example 1 (single line).

```
{ CALL Ws_Connect_Replace( 0,'Test Job Name', 'Test Task Name', 0, 0, 'REPLACE', 'Connection1', 'Connection2', ?, ?, ? )};
```

-- ODBC 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 |
|----------|--|
| Teradata | Basic Teradata Query (BTEQ) i.e. bteq. |

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 or 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 or not the glossary term was successfully added/deleted. |

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 |

| Parameter Name | Datatype | Mode |
|----------------|--------------|------|
| 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  
);
```

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 <i>Callable Routines Common Input</i> (on page 896) |
| 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. |

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  
);
```

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. |

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.'

);

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 <i>Callable Routines Common Input</i> (on page 896) |
| 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. |
| Result Number | Output Result Number: 1 Success. -2 Error. e.g. Due to invalid job name or job not running. -3 Fatal/Unexpected Error. |

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
);
```

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 <i>Callable Routines Common Input</i> (on page 896) |
| 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. |

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 |

| Parameter Name | Datatype | Mode |
|----------------|--------------|------|
| 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
);
```

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 <i>Callable Routines Common Input</i> (on page 896) |
| 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. |

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  
);
```

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 <i>Callable Routines Common Input</i> (on page 896) |
| 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. |

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 |
| 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
, 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 <i>Callable Routines Common Input</i> (on page 896) |
| 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. |

| Input | Description |
|------------------------|---|
| 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. |
| 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. |

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
, 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 <i>Callable Routines Common Input</i> (on page 896) |
| 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 or not 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. |

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 |
| 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
);
```

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 <i>Callable Routines Common Input</i> (on page 896) |
| 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. |

| Output | Description |
|---------------|---|
| 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. |

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
);
```

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 <i>Callable Routines Common Input</i> (on page 896) |
| 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. |

| Output | Description |
|---------------|--|
| 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. |

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
);
```

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 <i>Callable Routines Common Input</i> (on page 896) |
| 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. |

| Output | Description |
|---------------|---|
| 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. |

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
);
```

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 <i>Callable Routines Common Input</i> (on page 896) |
| 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. |

| Output | Description |
|---------------------------------|--|
| Standard Job Status Code | <p>Standard Job Status Code:</p> <p>H - On Hold. The job is on hold. A held job can be edited and/or started.</p> <p>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).</p> <p>B - Blocked. The job is blocked because a previous instance of the same job is still running.</p> <p>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.</p> <p>R - Running. The job is currently running.</p> <p>F - Failed. The job failed due to an error.</p> <p>C - Completed. The job completed successfully (but it may have warnings). A completed job cannot be restarted.</p> <p>G - Failed - Aborted. The job failed and was subsequently aborted. An aborted job cannot be restarted.</p> <p>E - Error Completion.</p> <p>0 - see note below.</p> |

| Output | Description |
|-----------------------------------|---|
| Enhanced Job Status Number | <p>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.</p> <p>1 - On Hold. The job is on hold. A held job can be edited and/or started.</p> <p>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).</p> <p>3 - Blocked. The job is blocked because a previous instance of the same job is still running.</p> <p>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.</p> <p>5 - Running. The job is currently running and no tasks have failed or produced warnings.</p> <p>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.</p> <p>7 - Running with Warnings. The job is currently running and some tasks have produced warnings.</p> <p>8 - Failed. The job failed due to an error.</p> <p>9 - Completed. The job completed without warnings. A completed job cannot be restarted.</p> <p>10 - Completed with Warnings. The job completed with warnings. A completed job cannot be restarted.</p> <p>11 - Failed - Aborted. The job failed and it was subsequently aborted. An aborted job cannot be restarted.</p> <p>12 - Error Completion.</p> |

***** 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*.
-

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 |
| 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
);

```

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 <i>Callable Routines Common Input</i> (on page 896) |
| 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. |

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  
);
```

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 <i>Callable Routines Common Input</i> (on page 896) |
| Table Name | Table Name to process the relevant indexes of. Note: The Table Name is ignored when the optional Index Name is specified. |
| 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.

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
);
```

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 maximum 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 maximum 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 <i>Callable Routines Common Input</i> (on page 896) |
| 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. |
| Maximum Versions per Object to Retain | 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. |
| 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. |

| Output | Description |
|---------------|----------------------------|
| Result Number | Output Result Number: |
| | 1 Success. |
| | -2 Error. |
| | -3 Fatal/Unexpected Error. |

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
);
```

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. |

Teradata Parameters: WsParameterRead

Callable Routine Type: PROCEDURE.

| Parameter Name | Datatype | Mode |
|----------------|-------------|------|
| p_parameter | VARCHAR(64) | IN |

| Parameter Name | Datatype | Mode |
|----------------|---------------|------|
| 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);

```

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. |

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. |

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);
```

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. |

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');
```

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 <i>Callable Routines Common Input</i> (on page 896) 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. |

| Input | Description |
|---------------|--|
| 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. |

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
);
```

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 <i>Callable Routines Common Input</i> (on page 896) 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 <code>SQLCODE</code> . It is optional but recommended to populate this when an error occurs. |
| RDBMS Message | RDBMS-specific message. e.g. The Oracle special variable <code>SQLERRM</code> . 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. |

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  
);
```

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 <i>Callable Routines Common Input</i> (on page 896) 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. |

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
);

```

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 <i>Callable Routines Common Input</i> (on page 896) Note: Refer to the RDBMS-specific parameters for the relative positions (they are NOT declared as the first parameters) |

| Input | Description |
|--------------------------------|--|
| 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. |
| 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. |

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 |

| Parameter Name | Datatype | Mode |
|----------------|----------------|------|
| 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

```

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 <i>Callable Routines Common Input</i> (on page 896) |
| 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. |

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
```

CHAPTER 35

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_error | Created from the ws_wrk_error_log table |
| 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 | 971 |
| Ws_admin_v_error | 971 |
| Ws_admin_v_sched | 972 |
| Ws_admin_v_task..... | 973 |

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 <i>Callable Routines Common Input</i> (on page 896) |
| wa_job | See <i>Callable Routines Common Input</i> (on page 896) |
| wa_task | See <i>Callable Routines Common Input</i> (on page 896) |
| wa_status | See <i>Callable Routines Common Input</i> (on page 896) |
| wa_message | the message associated with this audit log |
| wa_db_msg_desc | the database message associated with this audit log |

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 <i>Callable Routines Common Input</i> (on page 896) |
| wd_job | See <i>Callable Routines Common Input</i> (on page 896) |
| wd_task | See <i>Callable Routines Common Input</i> (on page 896) |
| wd_status | See <i>Callable Routines Common Input</i> (on page 896) |
| wd_message | the message associated with this audit log |
| wd_db_msg_desc | the database message associated with this audit log |

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 |

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 |

| Column | Description |
|---------------|--------------------|
| rejected | record rejected |
| errored | record errored |

CHAPTER 36

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 588, on page 976).

For information on importing a data model, see *Importing a Data Model* (on page 986).

IN THIS CHAPTER

| | |
|---|-----|
| Migrating the Data Warehouse Database Platform..... | 976 |
| Importing a Data Model | 986 |
| Re-Targeting Source Tables | 993 |
| Retro Column Properties..... | 995 |

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:

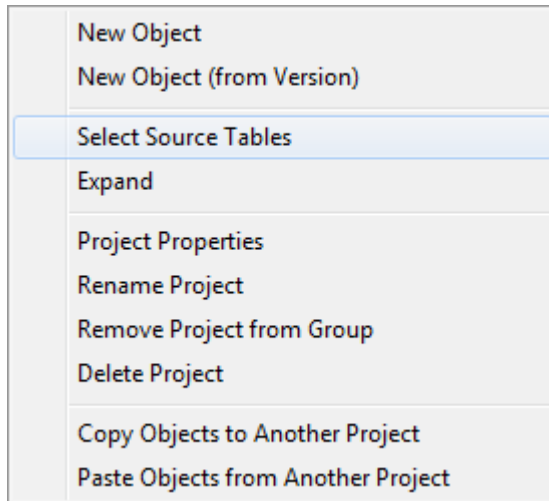
- 1 Create a connection object to the existing warehouse database.
- 2 Create Retro objects based on the source tables in the existing warehouse database.
- 3 Set the Retro objects as Retro Copy type objects.
- 4 Run a Scheduler task to build the Retro Copy objects from the source tables.
- 5 Set the 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:

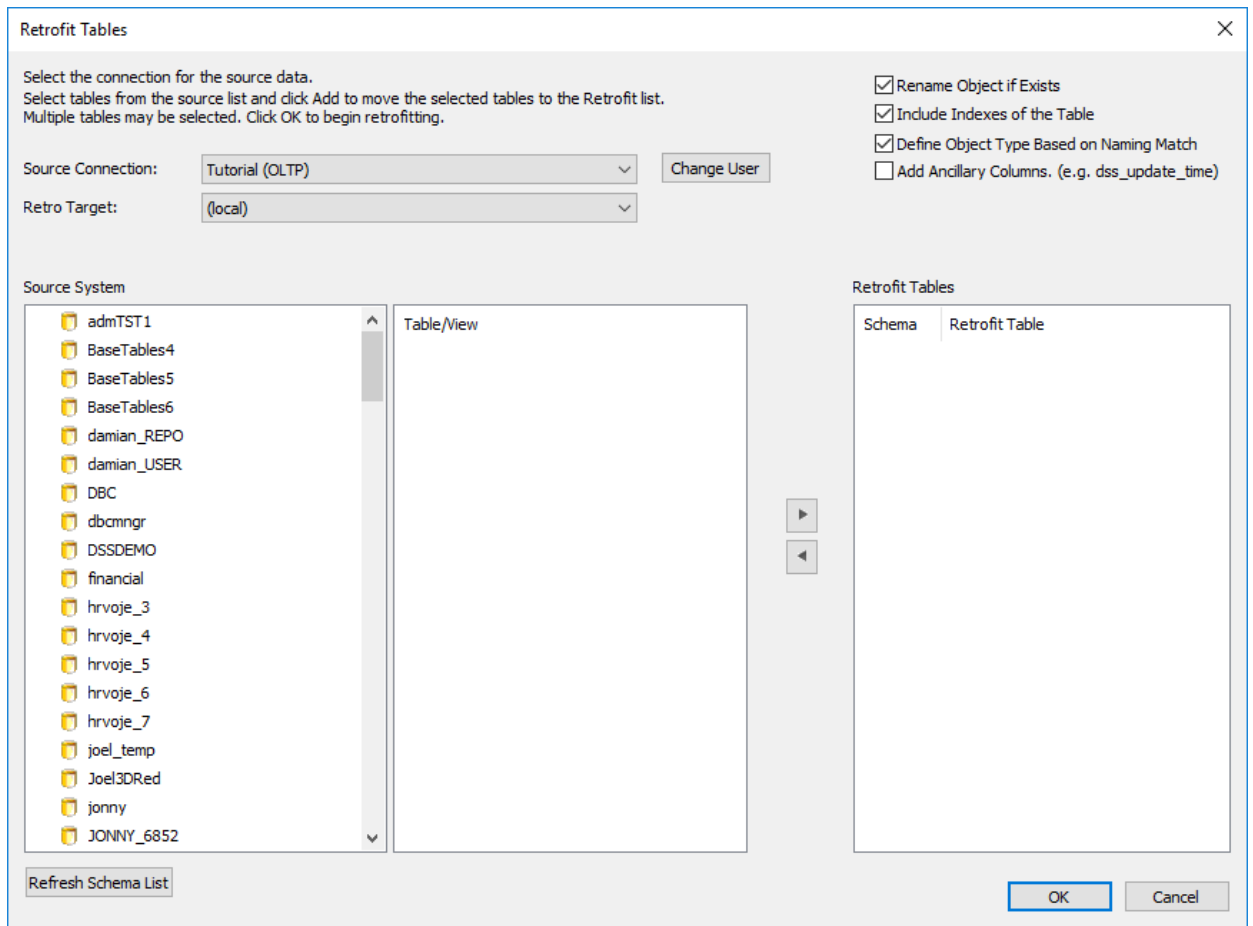
- 1 Create a connection object for the old data warehouse database, populating the following fields:
 - Connection Name
 - Connection Type => **ODBC**
 - ODBC Source
 - Work Directory
 - Extract user name
 - Extract passwordOR
 - Teradata Wallet User ID / Teradata Wallet String

Note: The extract user/ Teradata Wallet user must be able to select from all tables to be migrated.

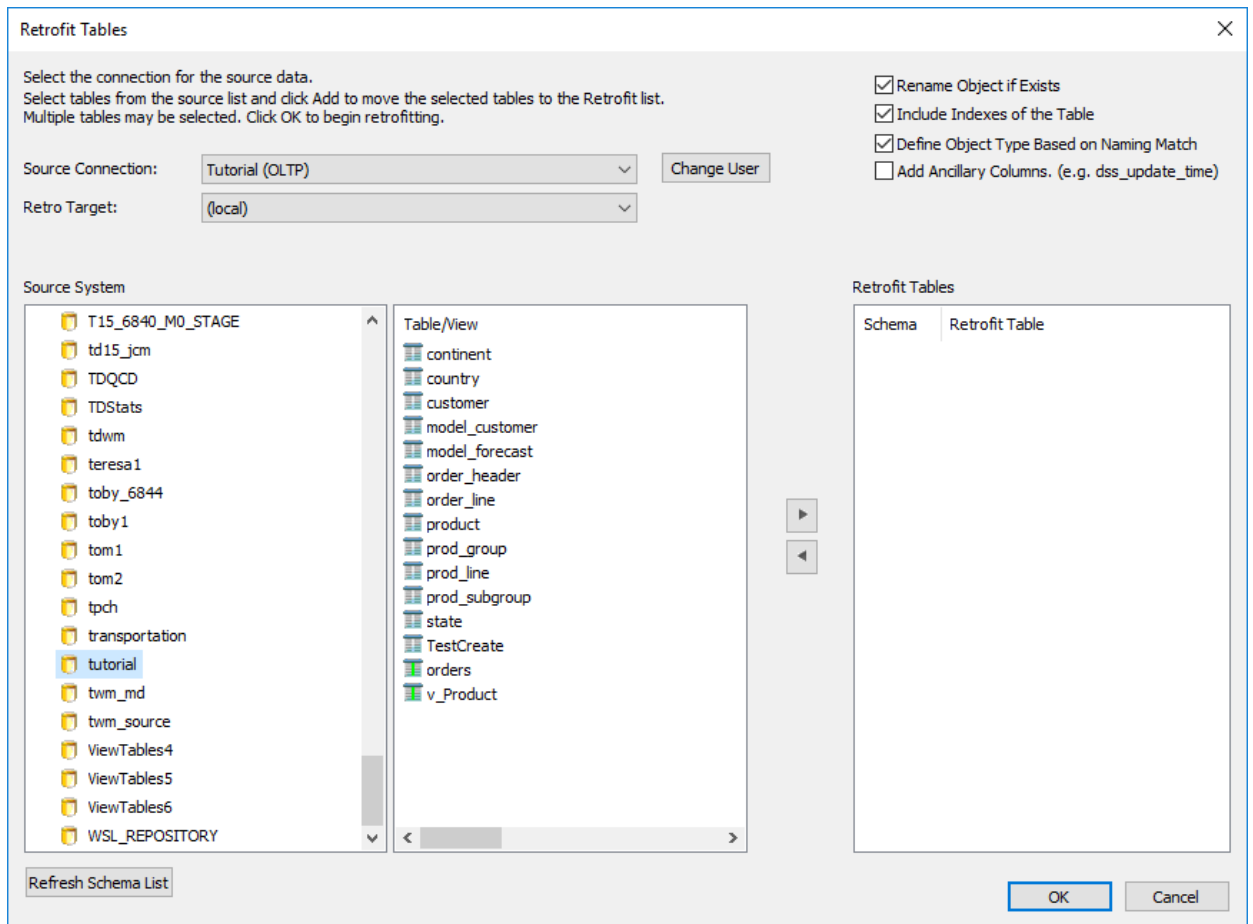
- 2 Ensure all naming standards in **Tools/Options** are set to match the objects being retrofitted. This saves work later.
- 3 Ensure **Enable Retro** is selected in the **Tools/Options/Object Types** menu.
- 4 Right-click on the **Retro object group** in the object tree in the left pane and select **Select Source Tables**.



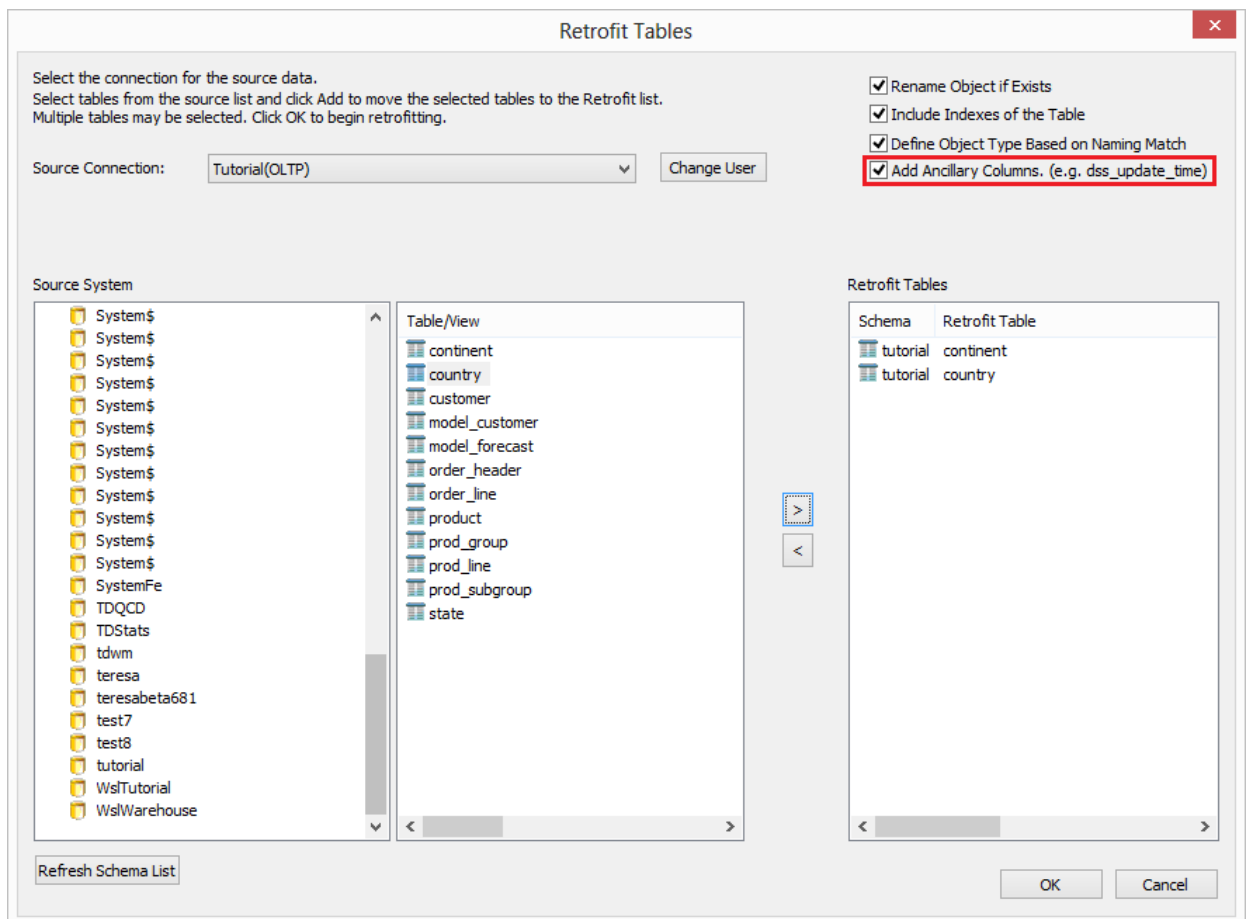
- 5 The Retrofit Tables dialog appears. In the **Source Connection** drop-down list choose the connection set up in step 1. A list of **databases** appears in the left pane.



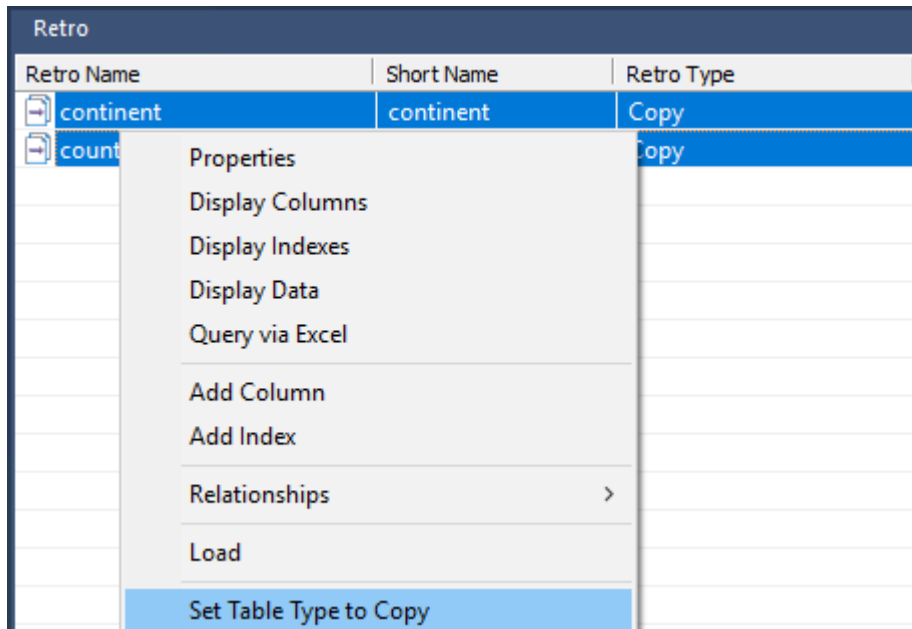
- 6 Double-click on the database/user/schema in the left pane. A list of **tables** in the database is displayed in the middle pane.



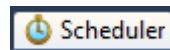
- 7 Select all the required tables from the middle pane list and click > to move them to the right pane. Then click the **Add Ancillary Columns (eg. dss_update_time)** checkbox and click **OK**.



- WhereScape RED acquires the metadata for the tables being migrated and creates a new WhereScape RED Retro object for each.
- 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.



- Click on the **Scheduler** button on the toolbar.



- Create a new job to run straight away.

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:

Description:

Frequency:

Start Date:

Start Time:

Maximum Threads:

Scheduler:

Dependent On:

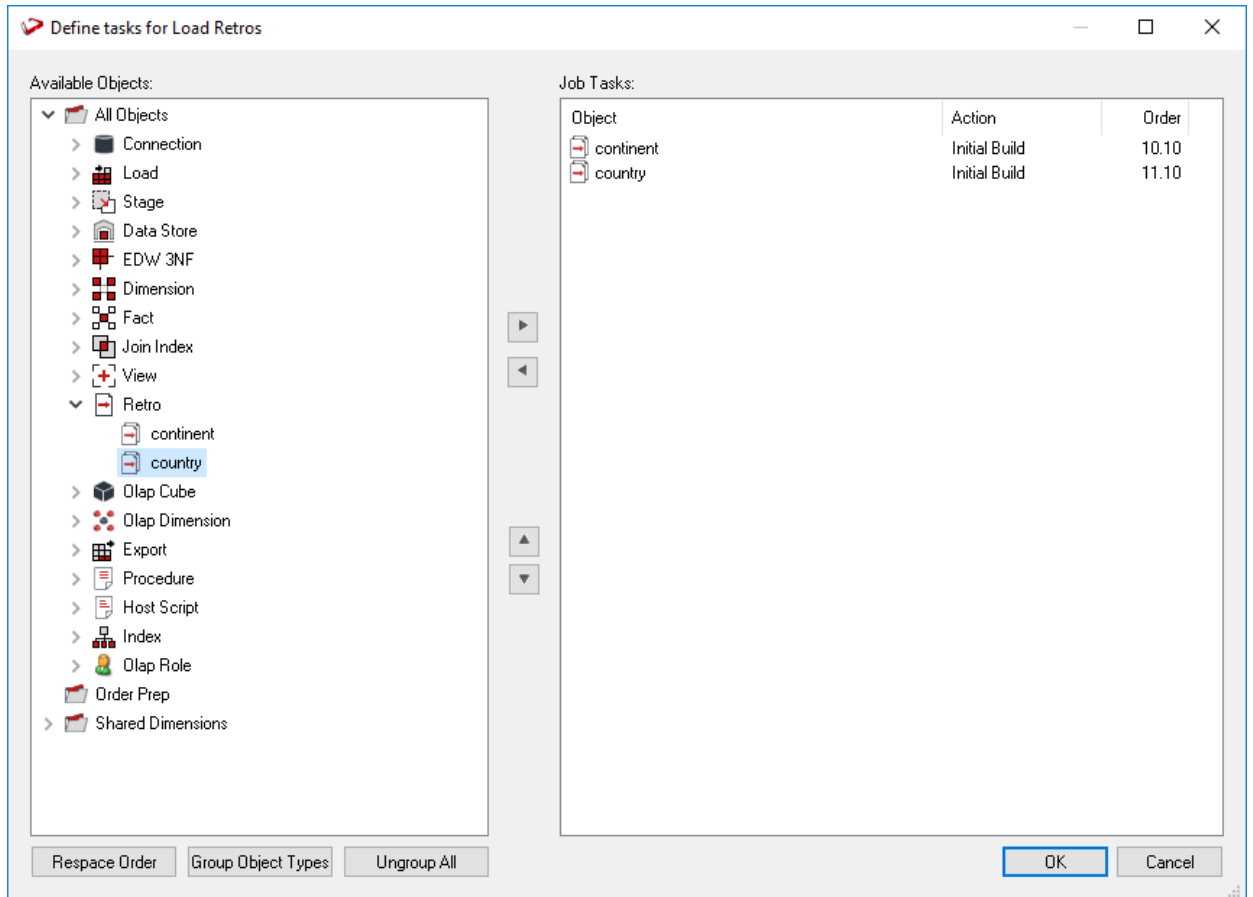
Logs Retained: This field lets you set the number of logs that are retained for this job before an automatic delete and archive occurs. 0 = keep all logs (default action)

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:

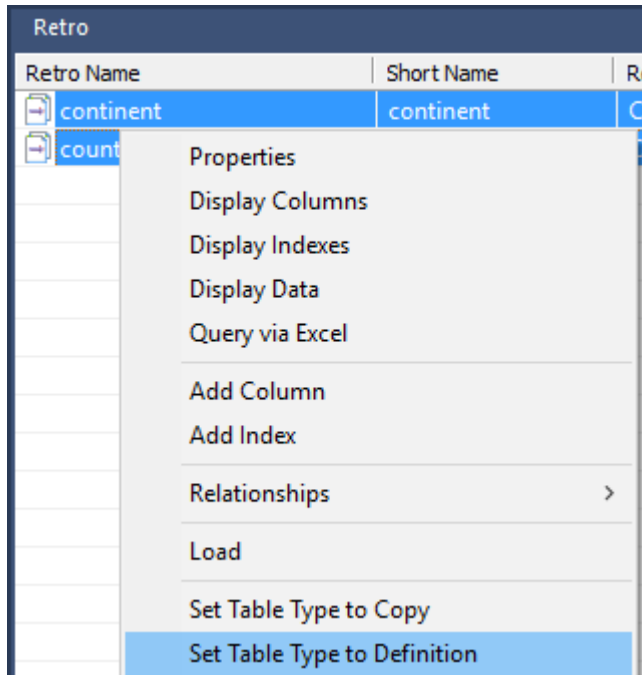
Success Command:

Failure Command:

12 Add all Retro objects created in steps (3) to (9) and click on **Group Object Types**.

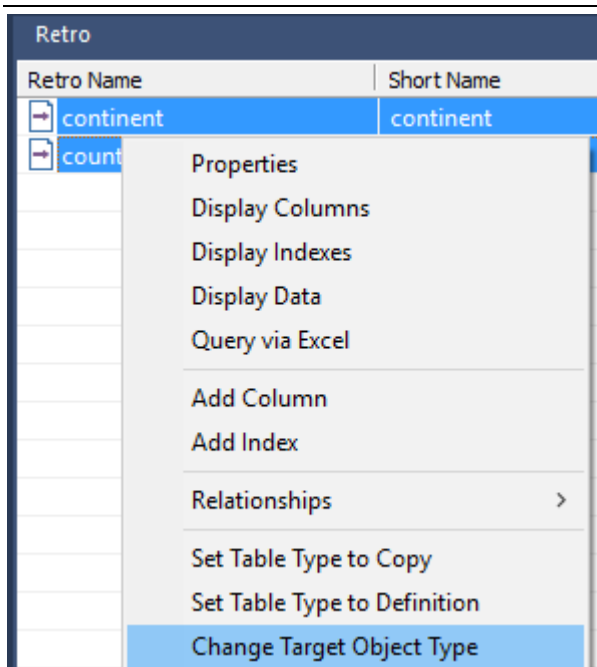


- 13 Once the job has completed, return to the WhereScape RED builder. Double-click on the **Retro Object group**. Select all objects in the middle pane and from the right-click menu select **Set Table Type to Definition**. This indicates the data has been copied into the Retro objects and the Retros can now be converted to the target objects.

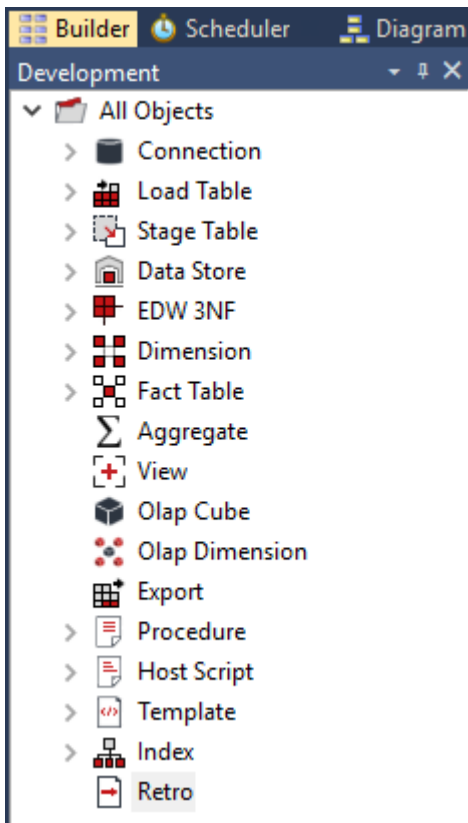


- 14 In the middle pane, select all objects. Right-click and select **Convert to Target Object**. WhereScape RED now converts the **Retro** objects to the appropriate object types.

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-target source tables** (see "**Re-Targeting Source Tables**" on page 993) 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 1010) 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 194) for more information.

IMPORTING A DATA MODEL

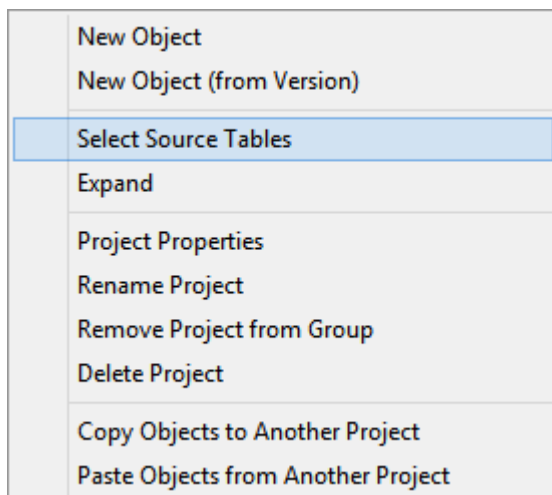
WhereScape RED provides functionality for importing data models from modeling tools.

The process to import a model is:

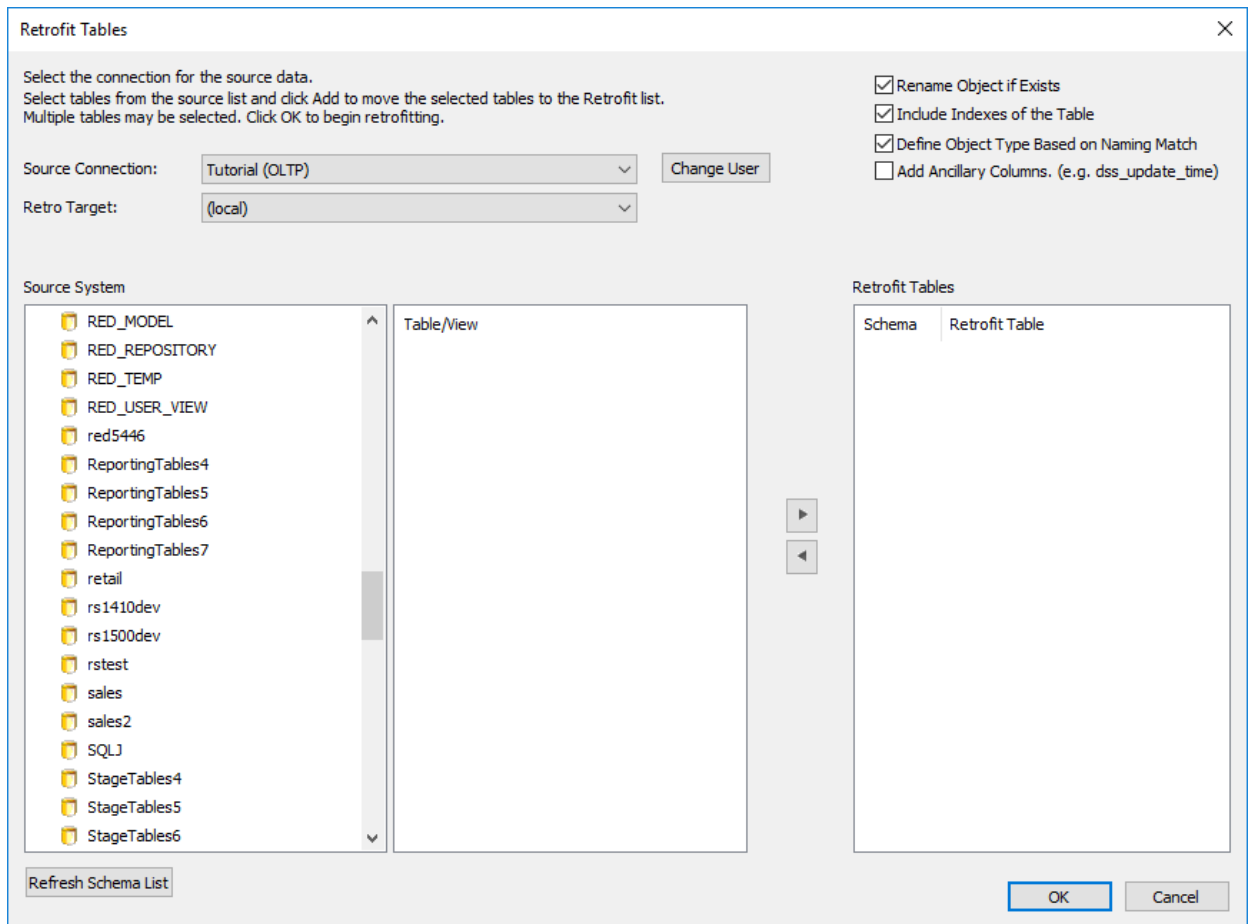
- 1 Create the physical data model in the modeling tool.
- 2 Generate DDL for the physical model in the modeling tool.
- 3 Run the DDL in the data warehouse database to create empty versions of the model tables.
- 4 Retrofit the tables in the dummy database into the WhereScape RED metadata as Retro objects.
- 5 Convert the Retro objects to Dimensions and Facts.

The following instructions outline steps 4 and 5 above:

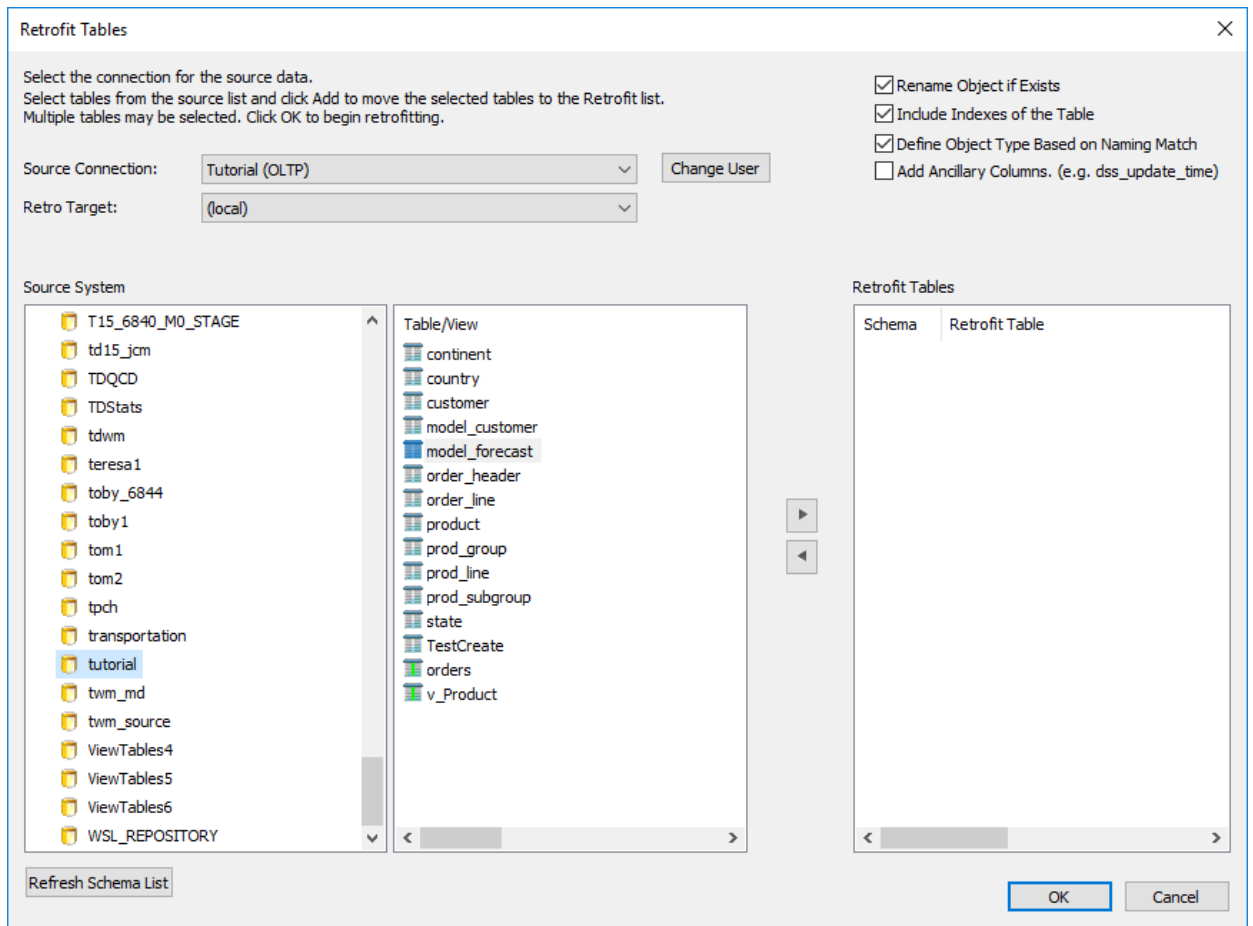
- 1 Right-click on the **Retro object group** in the object tree in the left pane and select **Select Source Tables**.



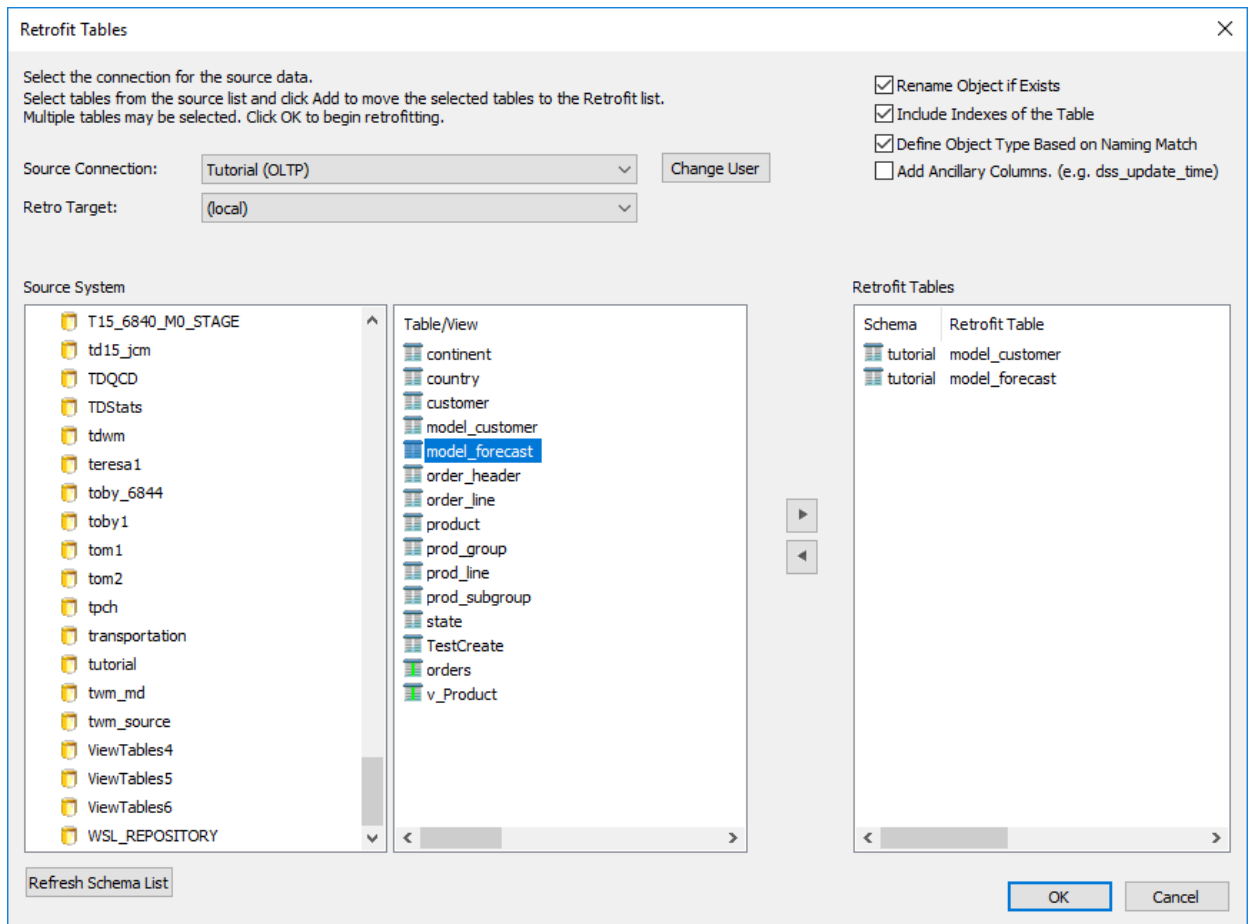
- 2 The Retrofit tables dialog is displayed. In the **Source Connection** drop-down list choose the Data Warehouse connection. A list of **databases** appears in the left pane (your list will be different).



- 3 Double click on the required database in the left pane list. A list of **tables** in the database is displayed in the middle pane.



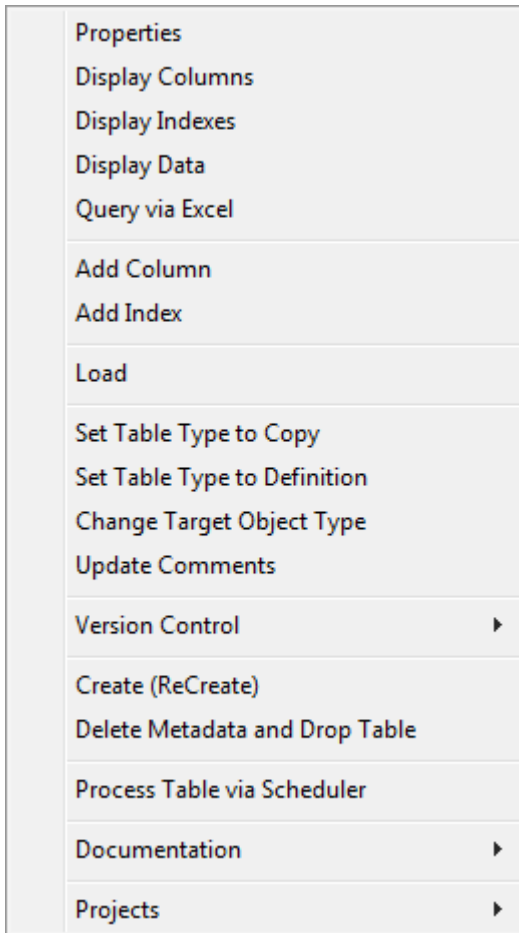
- Click on the required tables in the middle pane list and click > to move them to the right pane. Then click the **Add Ancillary columns (e.g. dss_update_time)** checkbox. Click **OK**.



- Double click on the **Retro object group** in the object tree in the left pane. You should see something like this.

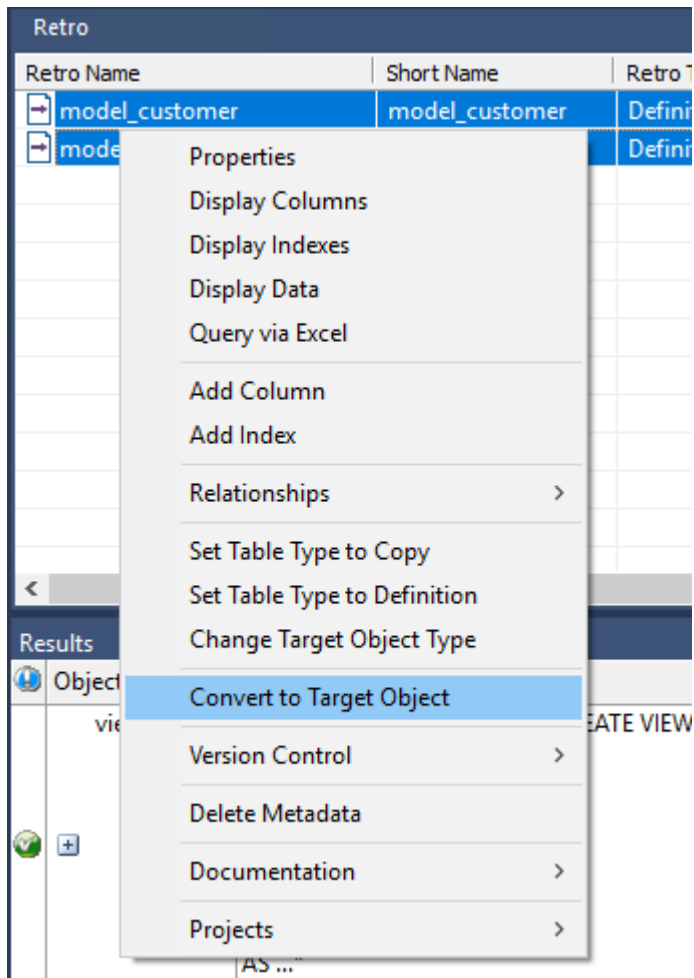
| Retro | |
|----------------|----------------|
| Retro Name | Short Name |
| model_customer | model_customer |
| model_forecast | model_forecast |
| | |
| | |
| | |

- 6 In the middle pane, select the tables. Right-click and select **Set table type to Definition**.

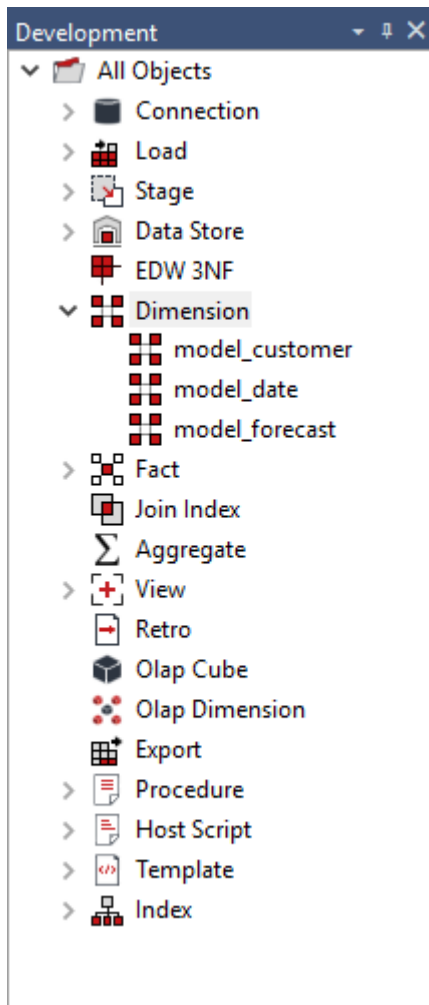


- 7 In the middle pane, select the tables. Right-click and select **Convert to Target Object**.

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.



- 8 The new tables have been imported.



- 9 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:

| Column Name | Display Name | Data Type | Source Table | Source Column |
|-----------------|-----------------|------------------|--------------|-----------------|
| dim_product_key | dim product key | integer ident... | dim_product | dim_product_key |
| description | description | varchar(24) | dim_product | description |
| code | code | numeric(6) | dim_product | code |
| prod_line | prod line | varchar(24) | dim_product | prod_line |
| prod_group | prod group | varchar(24) | dim_product | prod_group |
| subgroup | subgroup | varchar(24) | dim_product | subgroup |
| dss_update_time | dss update time | datetime | | |

They can be re-targeted to the correct source table(s), using the WhereScape RED re-target wizard as follows.

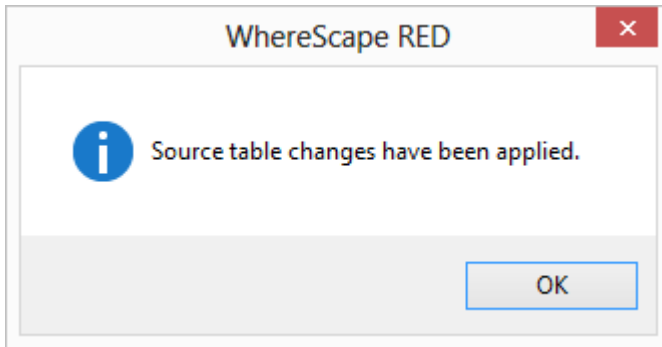
- 1 Right-click on a table object in the left pane and select **Change Column(s)**.
- 2 Select the **Column Source Table** checkbox. Select **dim_product** from the Original Value drop-down list. Select **load_product** from the New Value drop-down list. Click **Apply**.

Change Columns ✕

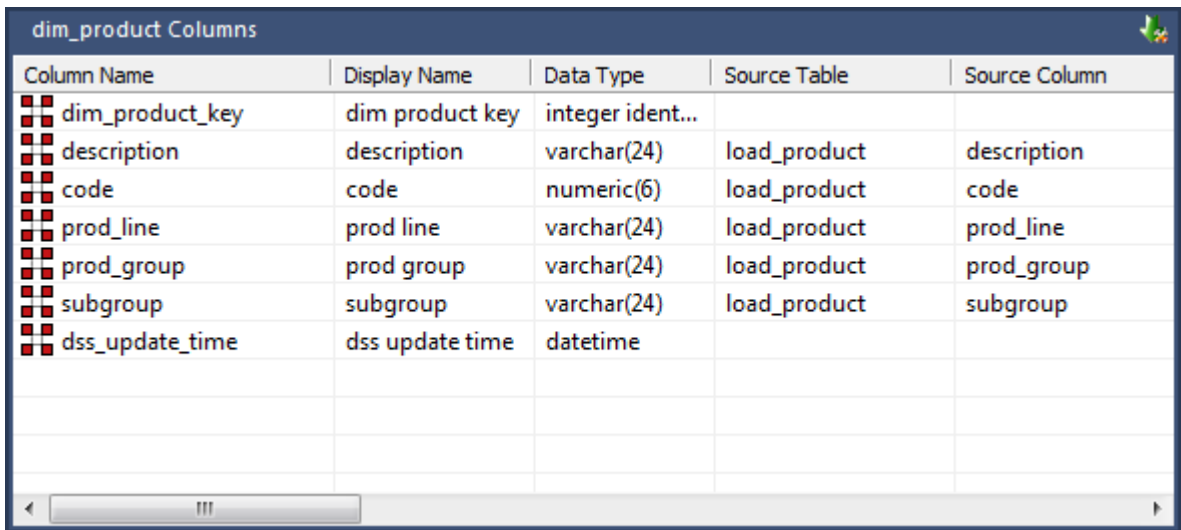
Select the changes to be applied. Note that only the selected columns will have the selected changes applied.

| | | | |
|---|---|---|--|
| Apply Changes to: <input checked="" type="checkbox"/> Column Source Table <input type="checkbox"/> Data Type <input type="checkbox"/> Nulls <input type="checkbox"/> Character Set <input type="checkbox"/> Format String <input type="checkbox"/> Numeric <input type="checkbox"/> Additive <input type="checkbox"/> Attribute <input type="checkbox"/> EUL | Original Value: dim_product _____ _____ _____ _____ _____ | New Value: load_product _____ _____ _____ _____ _____ | <input type="checkbox"/> Update matched columns only |
|---|---|---|--|

- 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.



| Column Name | Display Name | Data Type | Source Table | Source Column |
|-----------------|-----------------|------------------|--------------|---------------|
| dim_product_key | dim product key | integer ident... | | |
| description | description | varchar(24) | load_product | description |
| code | code | numeric(6) | load_product | code |
| 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 |
| 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

The table to which the column belongs. Read only.

Column Name

The database name of the column. This name must conform to database column naming standards. Typically such 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.

Column Title

This defines how the business refers to this column. Although it would not be normal practice to provide end user access to Retro tables, the contents of this field are passed through to the fact table during the drag and drop operation, so this field can be commented for later use in the fact table if desired. This field does not affect the physical table 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.

Column Description

Normally this field will contain the strategy for acquiring or populating a column. It is a useful place to record specific problems with the source data.

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

This numeric field provides the ordering of the columns in the database table 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

This is the database specific data type. It must be a valid data type for the data warehouse database platform. Typical Teradata databases often have integer, numeric(), varchar(), char(), date and timestamp data types. See the database documentation for a description of the data types available. Changing this field alters the table's definition.

Null Values Allowed

This checkbox when set allows nulls to exist in this column. If cleared then nulls are not permitted in the column. Although this affects the physical table definition no action or comparison is made on this field. If you wish to change this attribute for a column in an existing table, then the only supported way is to re-create the table, which may not be desirable.

Default Value

The default value Teradata puts in the column, if the column is not included in the insert statement populating the table.

Character Set

Select Latin or Unicode.

Format

This field is to allow the definition of a format mask for end user tools. It does not affect the physical table definition. This field would not normally be populated.

Character Comparison/Sorting

Indicates the Teradata case of the column; one of: case specific, not case specific, uppercase case specific or uppercase not case specific.

Compress/Compress Value

Indicates the column is compressed and enables the compress value text box. In the compress value text box, you can enter the list of values to use when compressing the column.

Numeric

Indicates that the column in question is numeric when set. This is normally only relevant for fact tables and would not normally be used for Retro tables.

Additive

Indicates that the column in question is additive when set. This is normally only relevant for fact tables and would not normally be used for Retro tables.

Attribute

Indicates that the column in question is an attribute when set. This is normally only relevant for fact tables and would not normally be used for Retro tables.

Business Key

Indicates that the column is part of the primary business key when set. Multiple columns can form the primary business key. This indicator is set and cleared by WhereScape RED during the Retro update procedure generation process. This should not normally be altered.

Key Type

The key type is used internally by WhereScape RED in the building of the update procedure and the generation of indexes. 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 <code>dss_source_system_key</code> 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

Indicates the table from which the column was sourced. 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 Retro table. It is also used in the track back diagrams and in the documentation.

Source Column

Indicates the column in the **Source table** from which the column was sourced. 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.

Join

This field is set by WhereScape RED during the building of the update procedure and should not normally be altered. It indicates that the column in question provides a join to a dimension table. The **Source Table** and **Source Column** fields will provide the dimension'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). The usual dialog for matching the dimension business key to a column or columns in the retro table is not displayed if this option is enabled.

Setting this field to Pre Join activates the **Join Source** field and allows you to select a table from the drop-down list.

Join Source

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

Retro Column continent.continent

Properties

Transformation

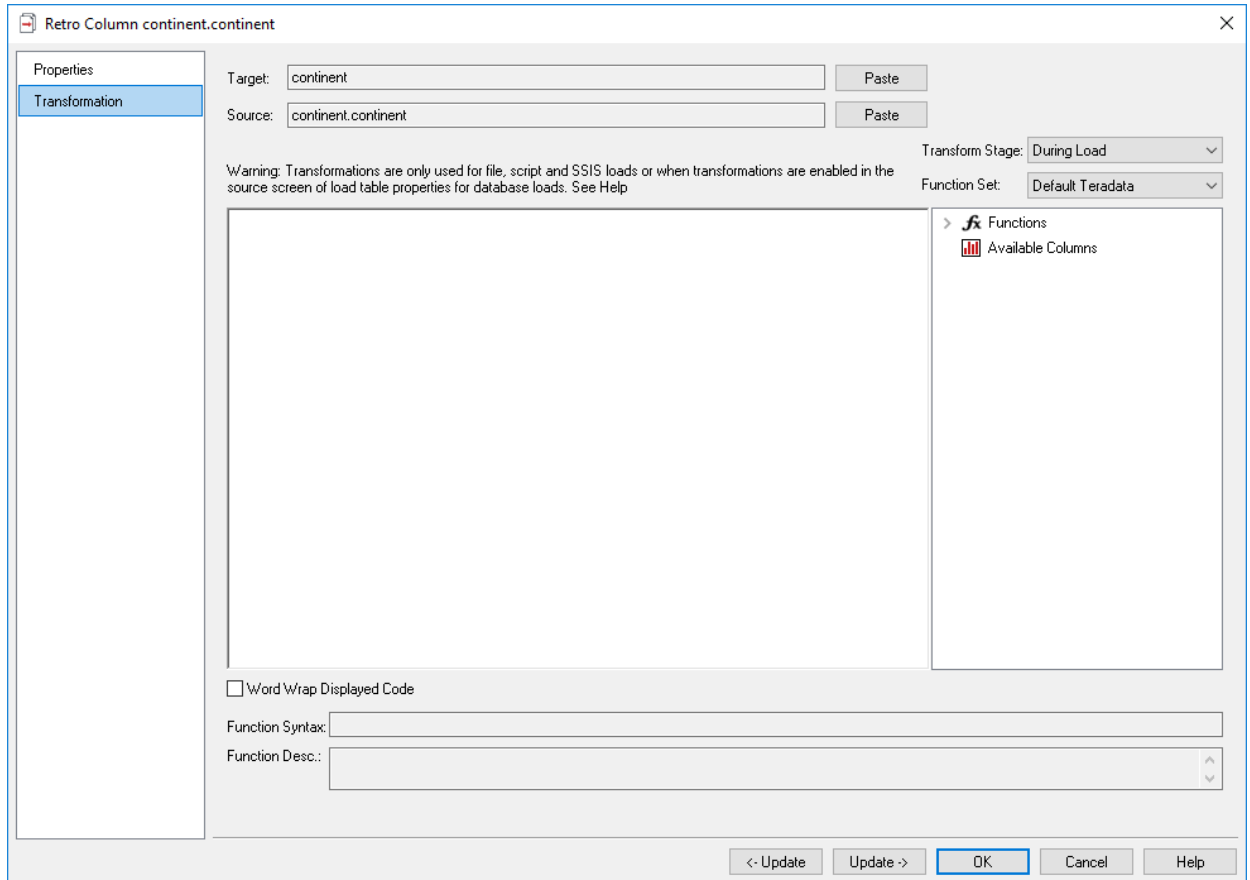
| | |
|------------------------------|-------------------------------------|
| ▲ General | |
| Table Name | continent |
| Column Name | continent |
| Column Title | continent |
| Column Description | |
| ▲ Physical Definition | |
| Column Order | 20 |
| Data Type | varchar(256) |
| Null Values Allowed | <input checked="" type="checkbox"/> |
| Default Value | |
| Character Set | LATIN |
| Format | X(256) |
| Character Comparison/Sorting | |
| Compress | <input type="checkbox"/> |
| ▲ Meta Definition | |
| Numeric | <input type="checkbox"/> |
| Additive | <input type="checkbox"/> |
| Attribute | <input checked="" type="checkbox"/> |
| Business Key | <input type="checkbox"/> |
| Key Type (0,A,B,C,...) | |
| ▲ Source Details | |
| Source Table | continent |
| Source Column | continent |
| Transformation | |
| Join | False |

Column Name
Database-compliant name of the column.
Dialog Opening Value: continent

<- Update Update -> **OK** Cancel Help

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 593) for more details.

CHAPTER 37

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 discussed 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 1002) 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 1003) 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 an 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 | 1002 |
| Integrating | 1003 |

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 stage and model 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 Windows servers where host scripts are currently executed. See *creating a Windows connection* (see "*Windows*" on page 148).
 - 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 135).
 - 5 Incorporate any Host system scripts currently used. See *incorporating host scripts* (see "*Integrating, Host Scripts*" on page 1004).
 - 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 1006)
 - 8 Answer the retrofit questions, and build any required procedures. See *integrate questions* (see "*Integrating, Questions*" on page 1006).
 - 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 1010).
 - 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 windows host scripts can be brought into the WhereScape RED meta data. To incorporate an existing script the process is as follows:

- 1 Create a **Host Script** object using 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.

Host Script monitor_db_mail

Properties

Name: monitor_db_mail Type: Windows script

Purpose:

Owner: dssdemj Delete Lock

Last Update By:

Default Connect:

Edit Lock

Locked For Edit By:

Edit Lock Reason or Last Update: New Script

Timestamps

Created: 2015-11-19 03:12:07 Last Update: Compiled:

OK Cancel Help

- 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 loads**" on page 258) 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:

- Model 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 Staging table group.

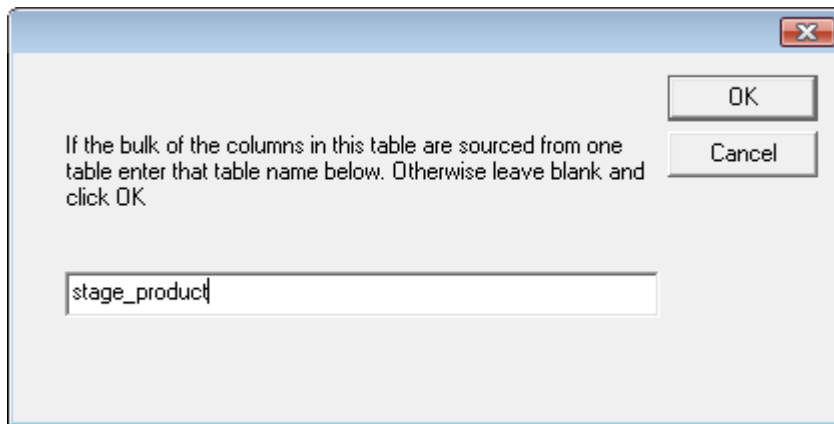
INTEGRATING, QUESTIONS

When a table within the data warehouse schema, that is unknown to RED, is dropped onto a table target the following dialog appears.



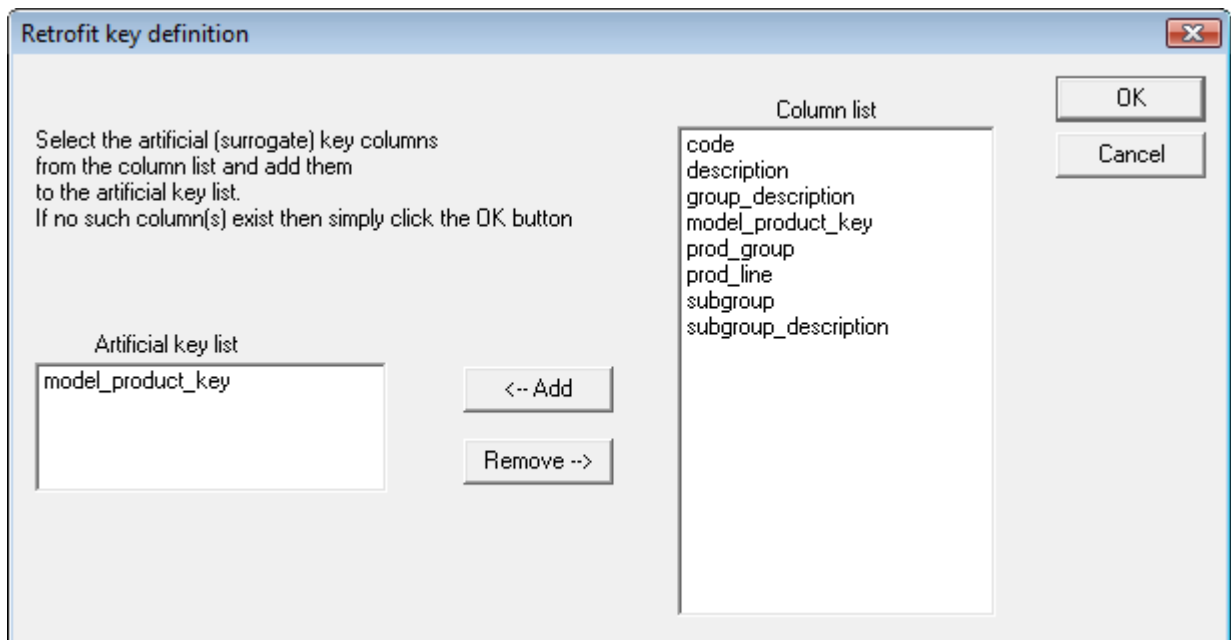
If this is a retro-fit then click Yes to proceed with the retro-fit process. The standard **New Object** dialog will appear and it would be advisable to leave the name of the object unchanged so that it matches the existing table.

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.



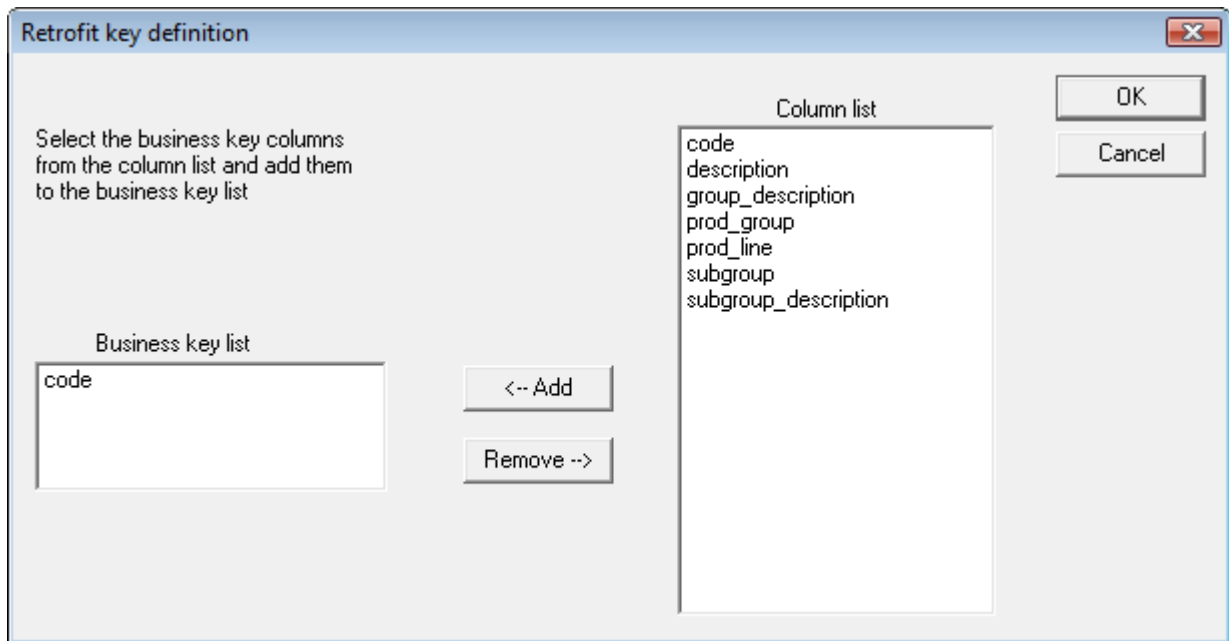
Artificial Key definition

If the table target is a model 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.



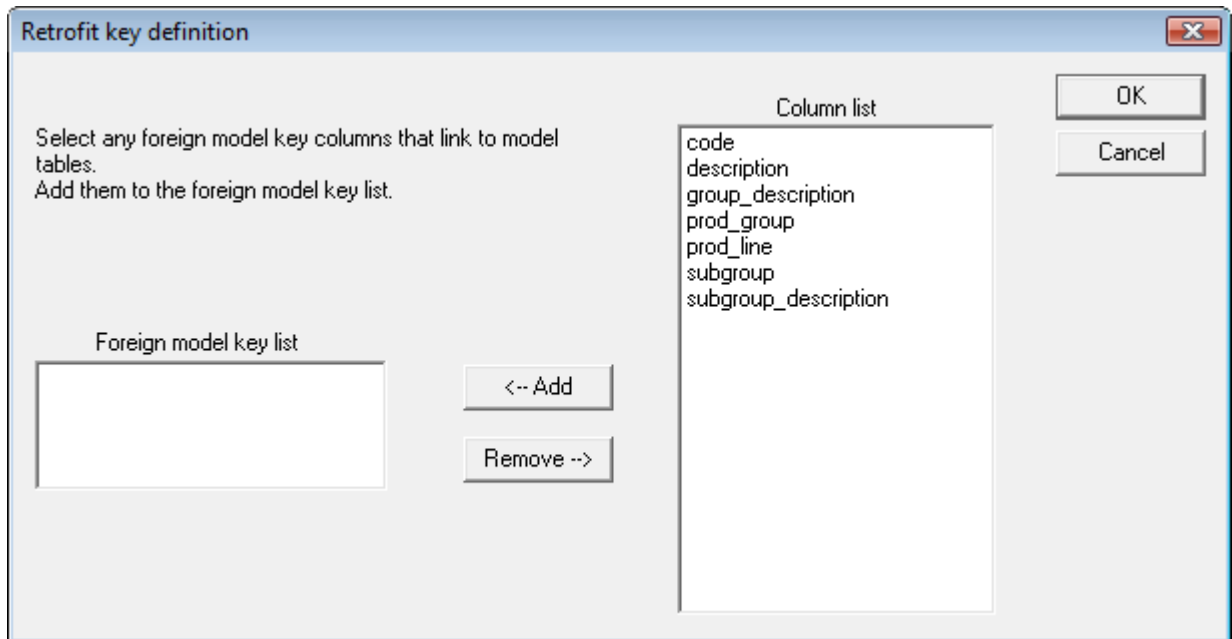
Business Key definition

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.



Foreign Key definition

A dialog will now prompt for the selection of the foreign keys from the table.



Update Procedure

If the target table is a stage, model or aggregate a dialog box will ask if an update procedure 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.

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 *retro-fitting procedures* (see "*Integrating, Procedures*" on page 1010).

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 or Output | Data Type |
|----------------|-----------------|--------------|
| p_sequence | Input | Integer |
| p_job_name | Input | Varchar(256) |
| p_task_name | Input | Varchar(256) |
| p_job_id | Input | Integer |
| p_task_id | Input | Integer |
| p_return_msg | Output | Varchar(256) |
| p_status | Output | 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 retro-fitting 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 retro-fitting WhereScape generated tables and views a number of additional considerations need to be taken.

Change the properties of all such columns. The key type should be set to 1, and the primary business key checkbox should be set.

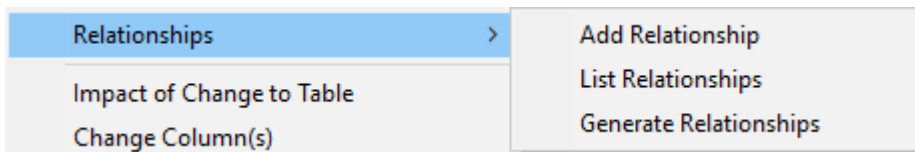
CHAPTER 38

RELATIONSHIP MAINTENANCE

Relationship Maintenance is available for the maintenance of joins between tables; 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 tables.

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.

Add Relationship

Specify the Relationship by selecting a table and column from both the left hand side and the right hand side, and then clicking the Add button.
For a multiple column Relationship, select each pair of columns before clicking the Add button.

Join Type: Primary Join

Object Type:

Table/View:

Column:

Joins:

| | |
|--|--|
| | |
| | |
| | |
| | |
| | |
| | |

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.

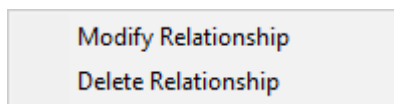
LIST RELATIONSHIPS

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).

| Relationships for ds_customer | | | | | | | |
|-------------------------------|--------|------------|--------------|--------|--------------|--------|--|
| Object | Column | Join Type | Object | Column | Primary Join | Part | |
| ds_customer | code | One to One | dim_customer | code | Y | 1 of 1 | |
| | | | | | | | |
| | | | | | | | |

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.

Modify Relationship ✕

Specify the Relationship by selecting a table and column from both the left hand side and the right hand side, and then clicking the Modify button.
For a multiple column Relationship, select each pair of columns before clicking the Modify button.

Join Type: One to One Primary Join

Object Type: Data Store Load

Table/View: ds_test load_customer

Column:

Joins:

| | |
|-------|------|
| code | code |
| cname | name |
| | |
| | |
| | |

Modify Reset Cancel Help

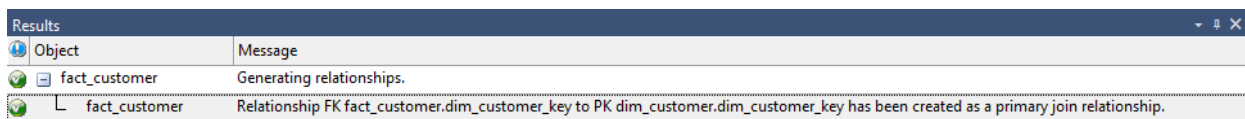
Relationships are edited in this dialog in the same way as the **Add Relationship** dialog above. **Object Types** and **Table names** cannot be modified.

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**.



The screenshot shows a 'Results' window with three rows of data. The first row is a header with 'Object' and 'Message'. The second row shows 'fact_customer' with the message 'Generating relationships.'. The third row shows 'fact_customer' with the message 'Relationship FK fact_customer.dim_customer_key to PK dim_customer.dim_customer_key has been created as a primary join relationship.'.

| Object | Message |
|---------------|---|
| fact_customer | Generating relationships. |
| fact_customer | Relationship FK fact_customer.dim_customer_key to PK dim_customer.dim_customer_key has been created as a primary join relationship. |

CHAPTER 39 UPGRADING RED

The upgrading of RED is discussed at some length in the **Installation and Administration Guide**.

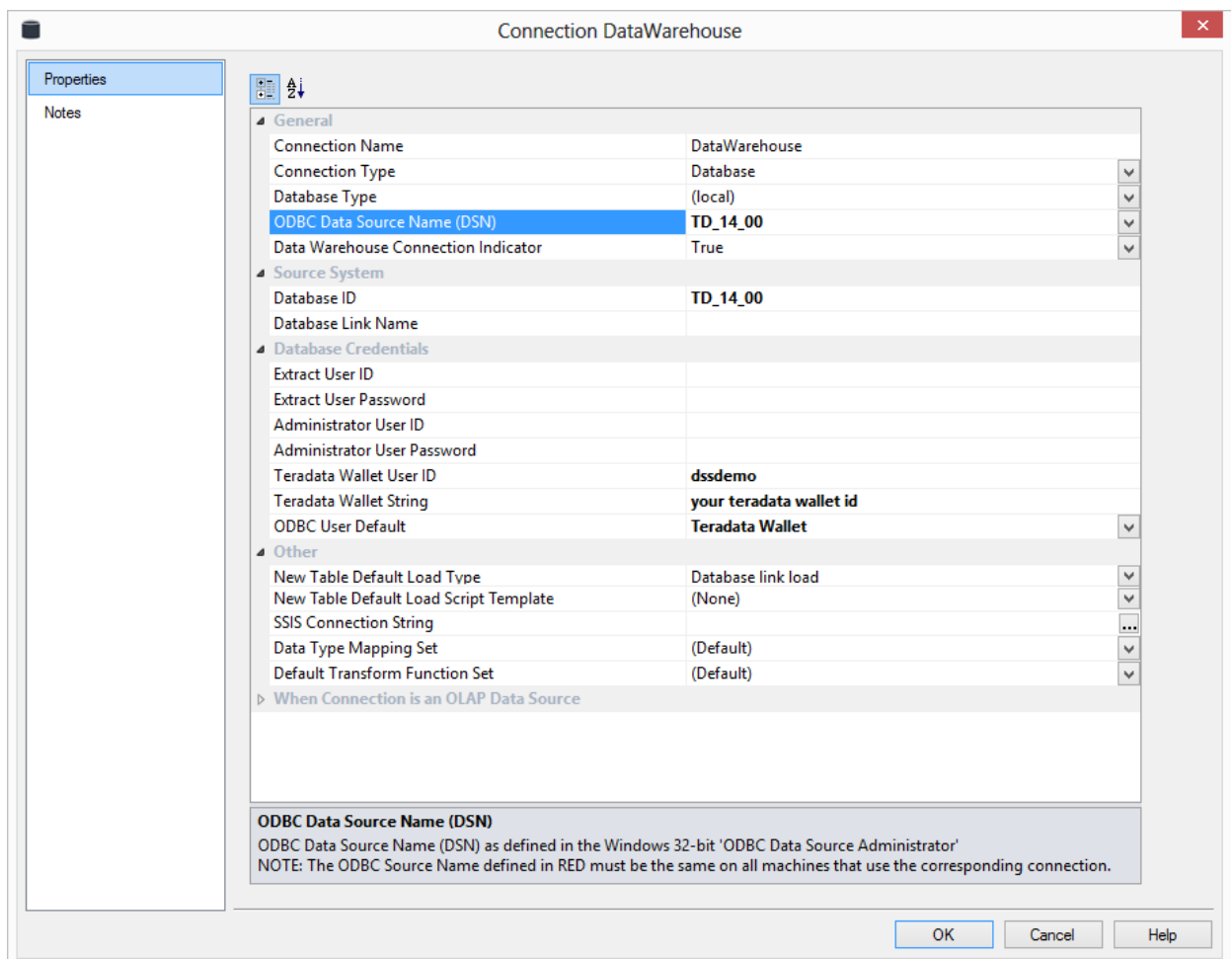
CHAPTER 40 LOGIN CHECKS

The following checks are performed during login; and if necessary, warning messages are displayed:

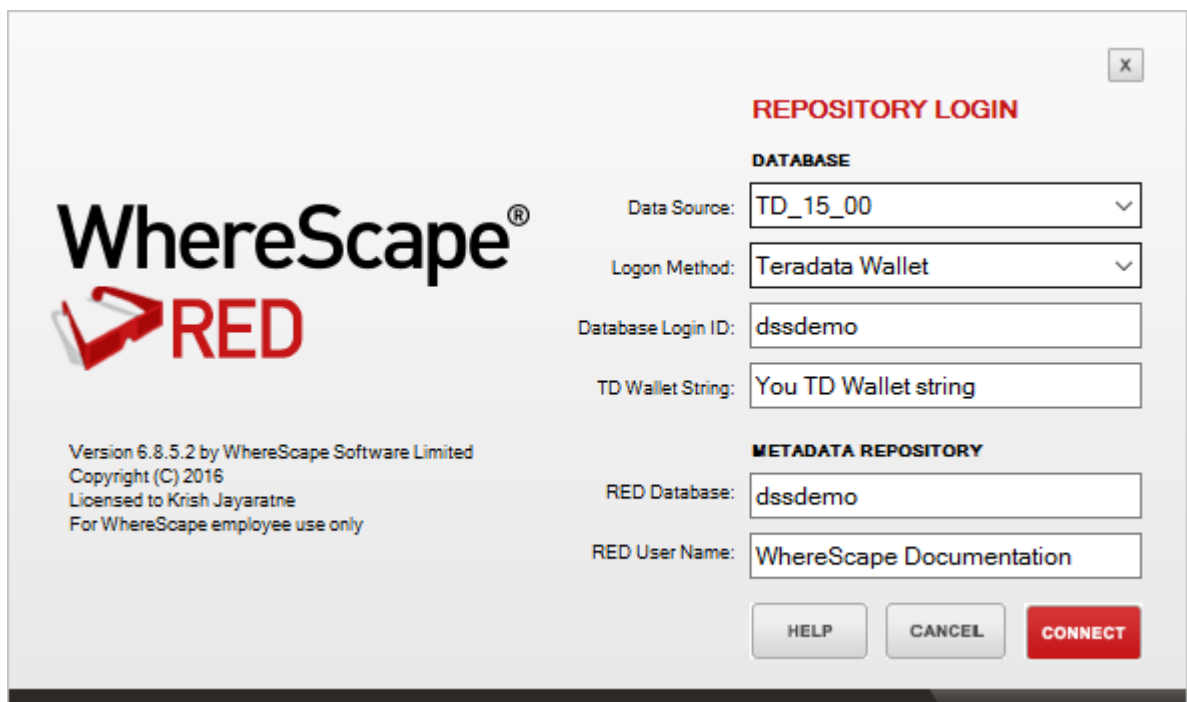
- 1 Warning if the login Data Source does not match the data warehouse connection's ODBC DSN.

You can correct this issue by performing one of the following actions:

- Alter the Data Warehouse Connection **ODBC Source** to match the login **Data Source**.

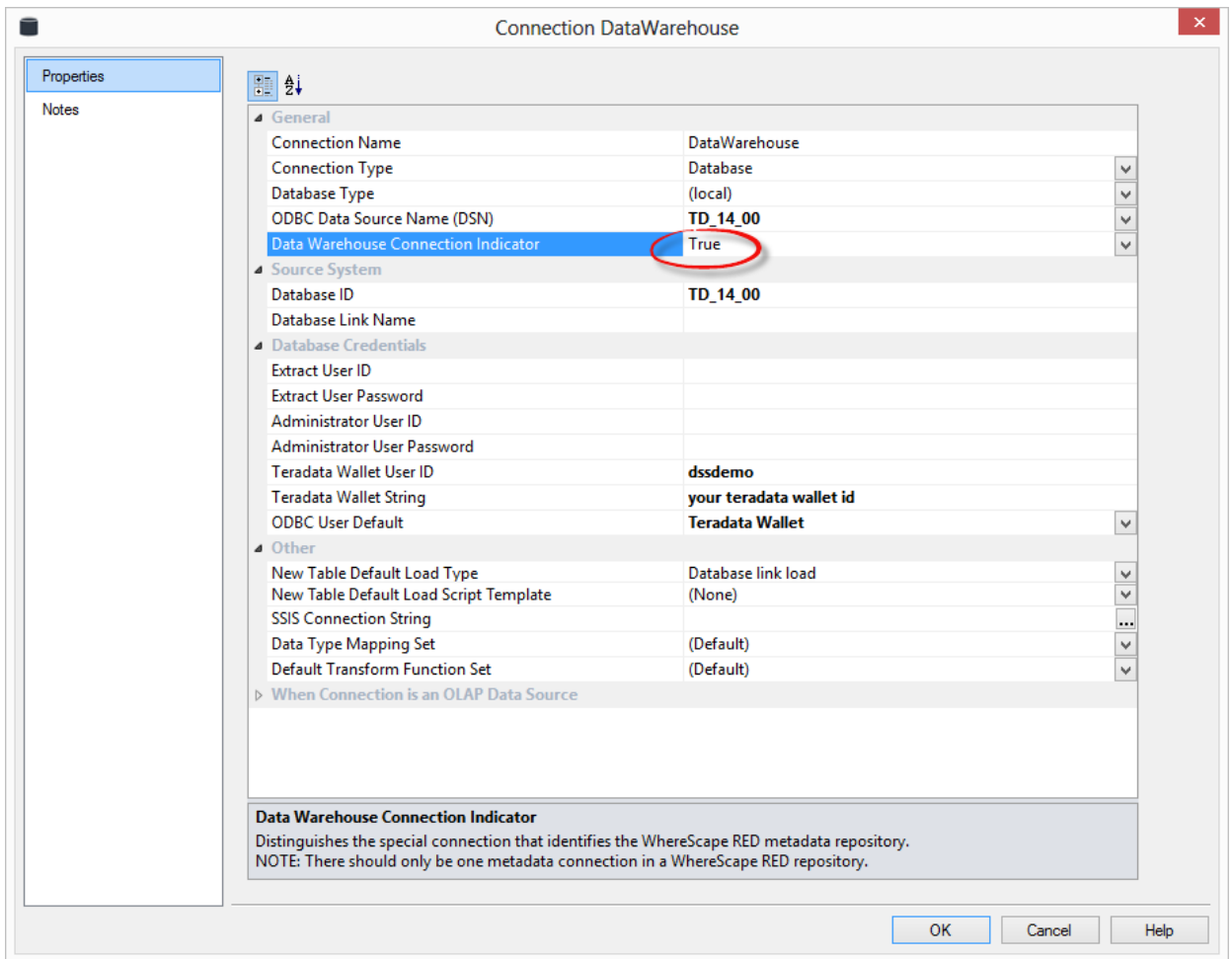


- Log off and log back in using the **Data Source** with the same name as the Data Warehouse Connection **ODBC Source**.



The screenshot shows the 'REPOSITORY LOGIN' dialog box. On the left is the WhereScape RED logo and version information: 'Version 6.8.5.2 by WhereScape Software Limited', 'Copyright (C) 2016', 'Licensed to Krish Jayaratne', and 'For WhereScape employee use only'. On the right, under the 'DATABASE' section, there are four fields: 'Data Source' (TD_15_00), 'Logon Method' (Teradata Wallet), 'Database Login ID' (dssdemo), and 'TD Wallet String' (You TD Wallet string). Under the 'METADATA REPOSITORY' section, there are two fields: 'RED Database' (dssdemo) and 'RED User Name' (WhereScape Documentation). At the bottom right are three buttons: 'HELP', 'CANCEL', and 'CONNECT'.

- 2 Warning if more than one connection has the data warehouse check-box on:
You can correct this issue by editing all the connections and making sure that only one is set to Data Warehouse:



CHAPTER 41

DATA TYPE MAPPINGS

IN THIS CHAPTER

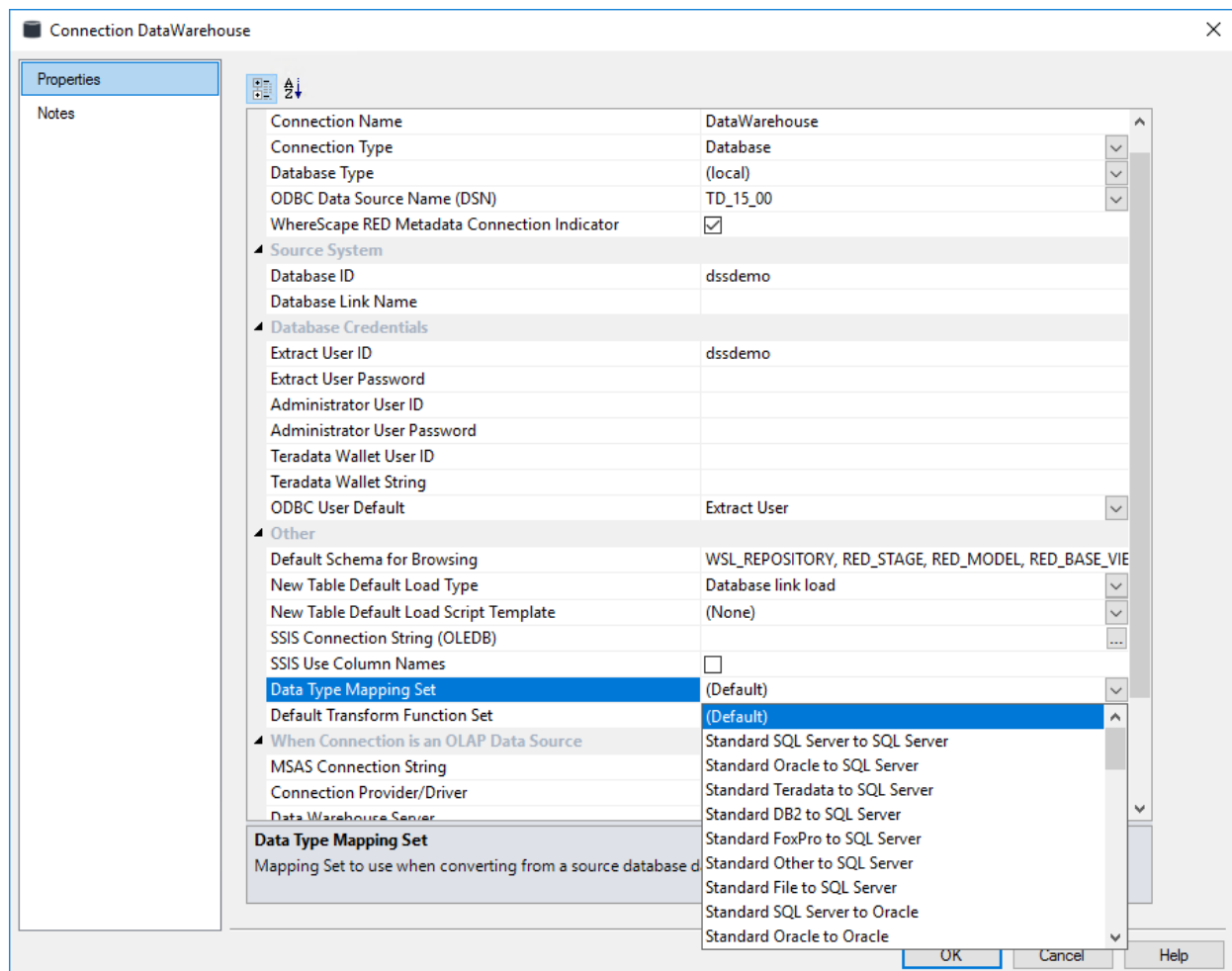
| | |
|--|------|
| Using Data Type Mapping Sets..... | 1022 |
| Maintaining Data Type Mapping Sets | 1024 |
| Loading Data Type Mapping Sets..... | 1044 |
| Exporting Data Type Mapping Sets | 1046 |
| Data Type Mapping Examples..... | 1048 |

USING DATA TYPE MAPPING SETS

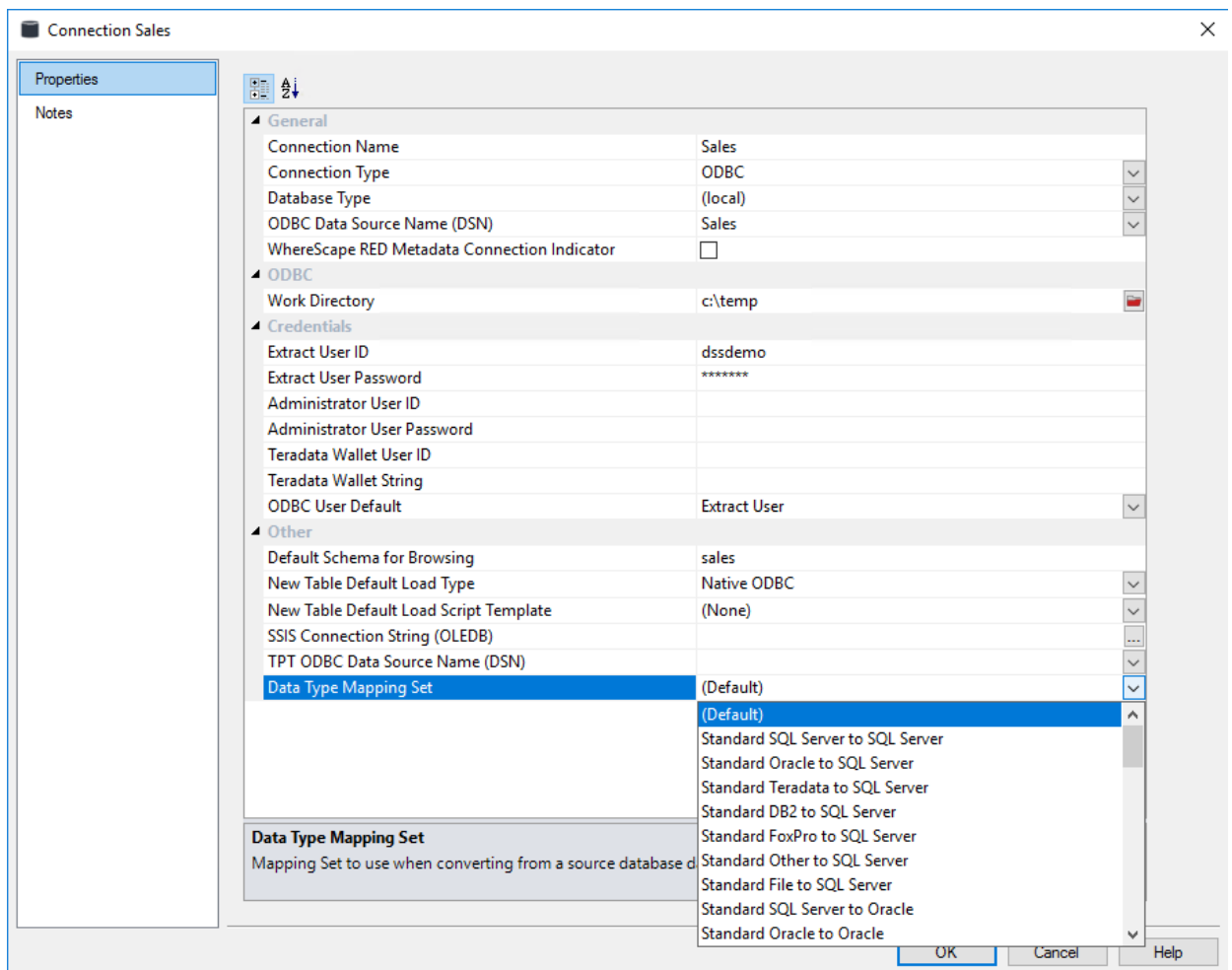
Data type mapping sets contain a list of mappings that are used when loading tables into the data warehouse.

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



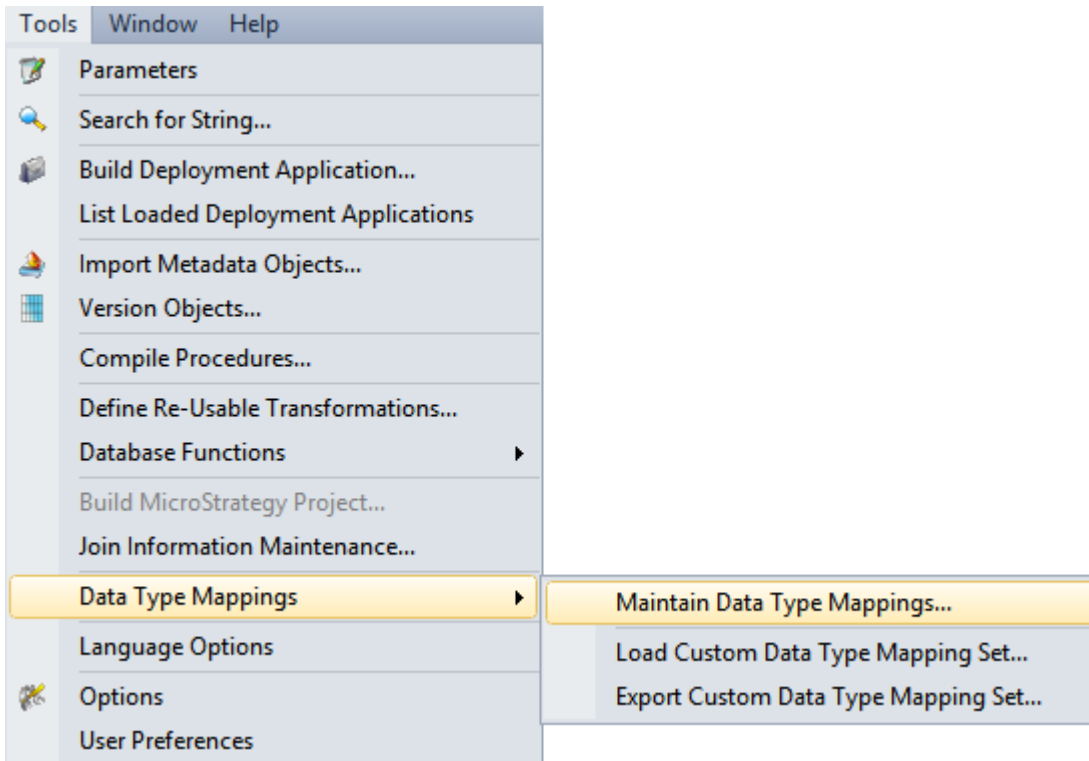
Non-Data Warehouse Connection Properties Dialog



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...**



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 1025)

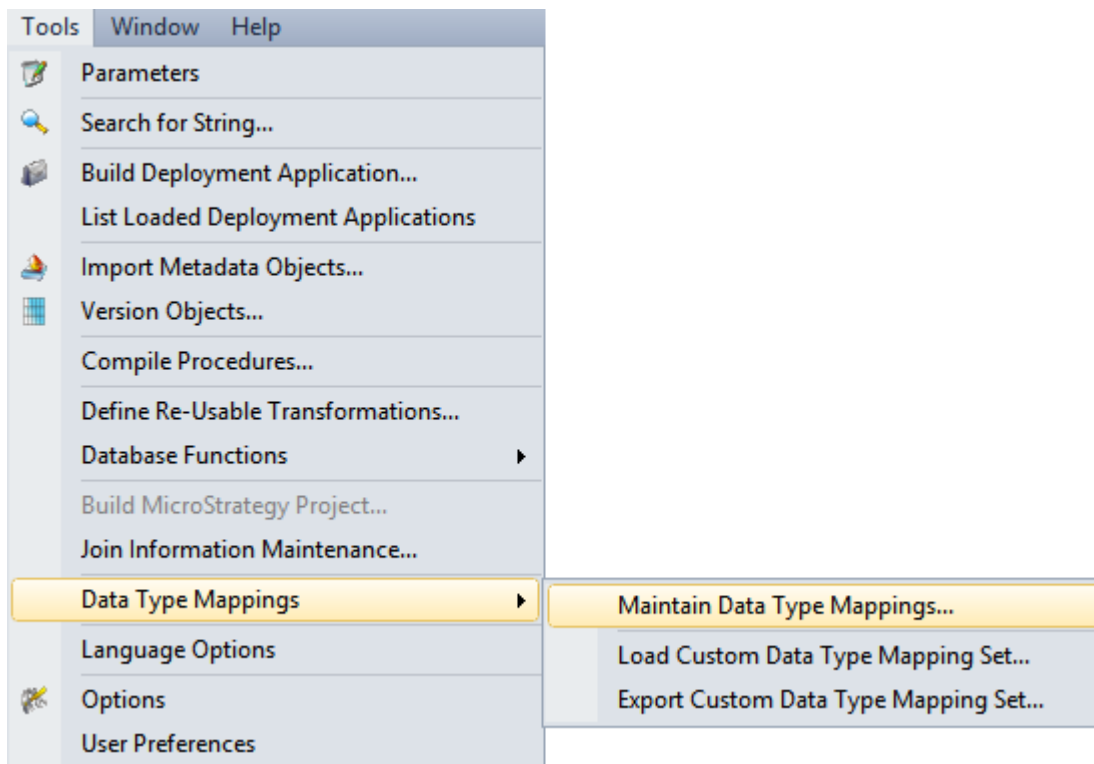
To copy a data type mapping set, see *Copying a Data Type Mapping Set* (on page 1029)

To edit a data type mapping set, see *Editing a Data Type Mapping Set* (on page 1033)

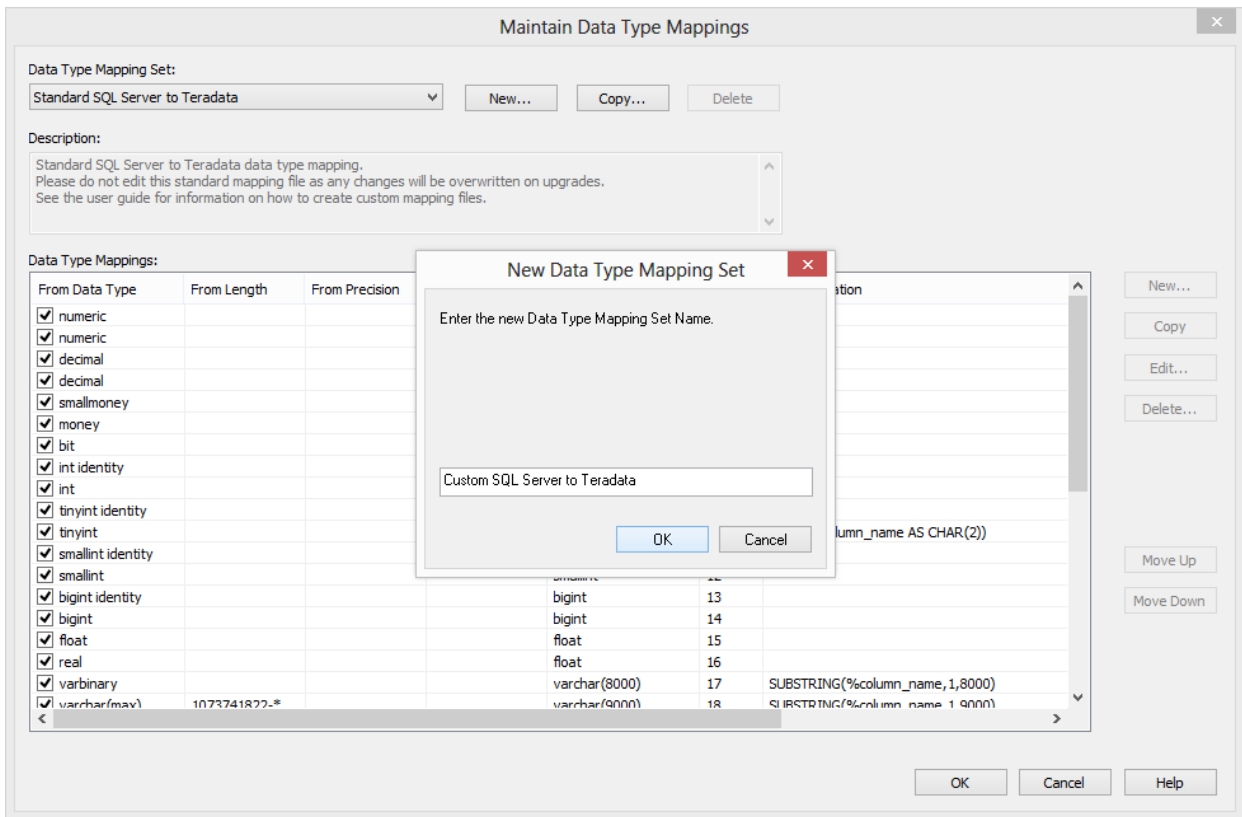
To delete a data type mapping set, see *Deleting a Data Type Mapping Set* (on page 1043)

CREATING A NEW DATA TYPE MAPPING SET

To create a new data type mapping set, select **Tools/Data Type Mappings/Maintain Data Type Mappings...**



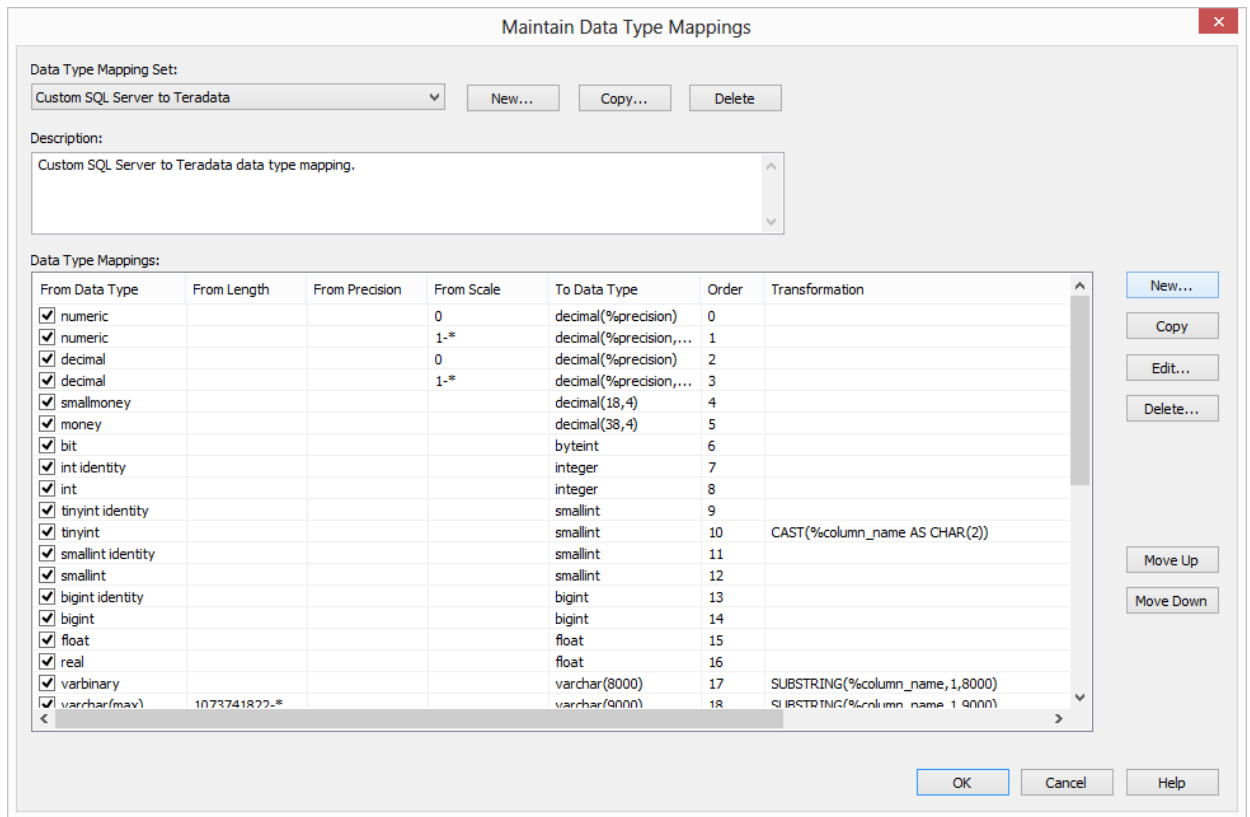
Click on the **New** button, enter the name of the new Mapping Set and click **OK**.



Description

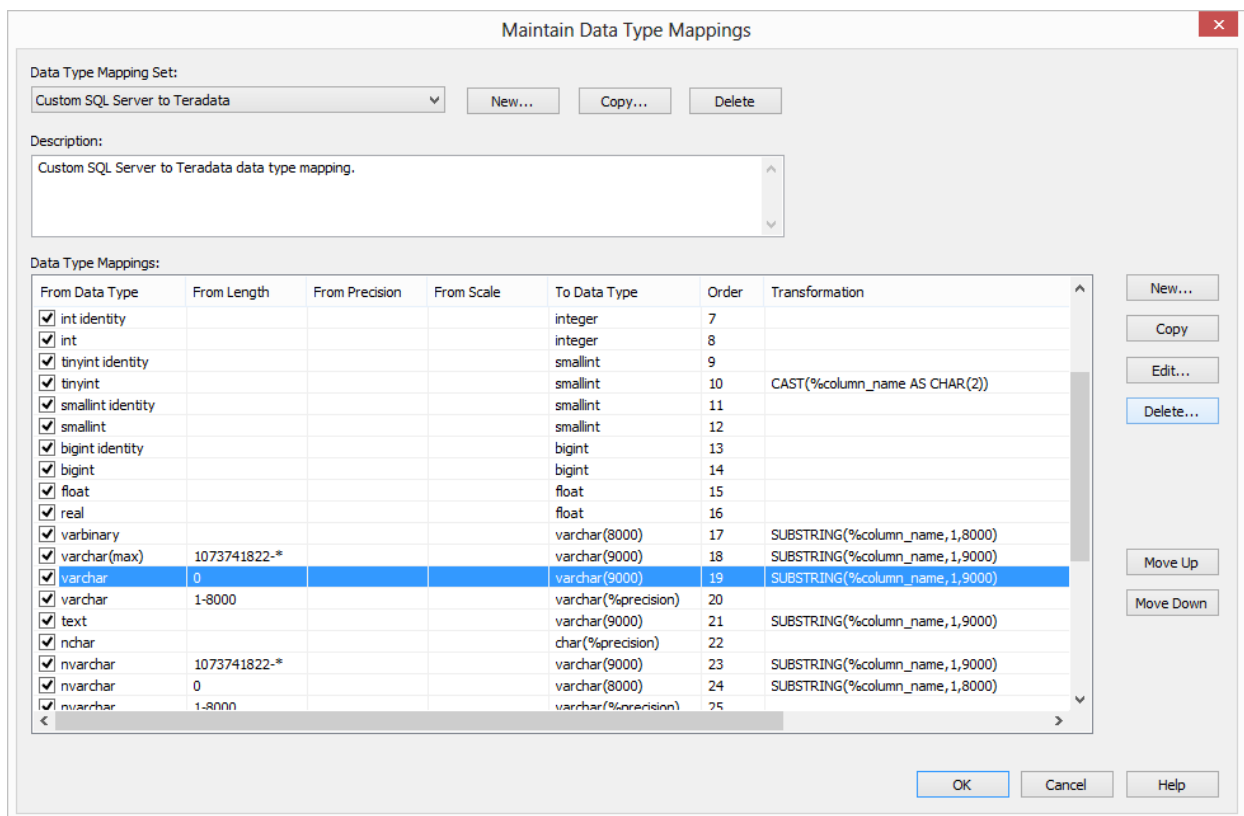
Enter the description for the data type mapping set.

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.

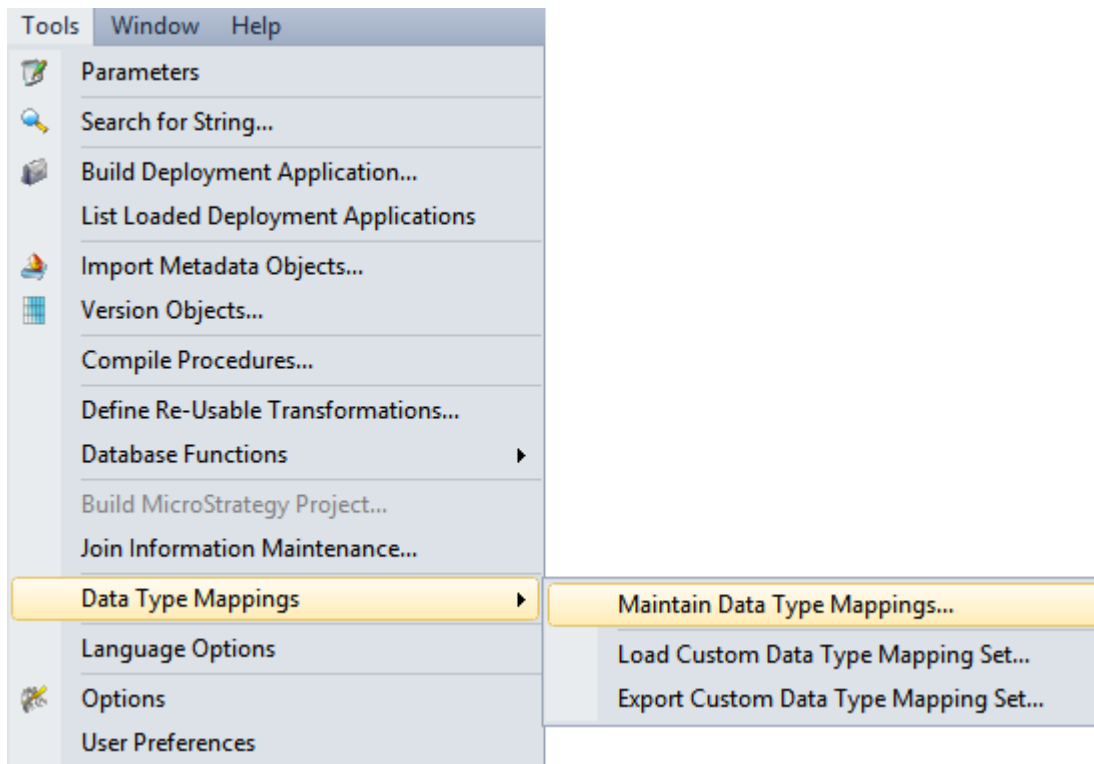
To move a mapping up or down in the list, select a mapping and then click on the **Move Up** or **Move Down** button on the right side of the dialog.



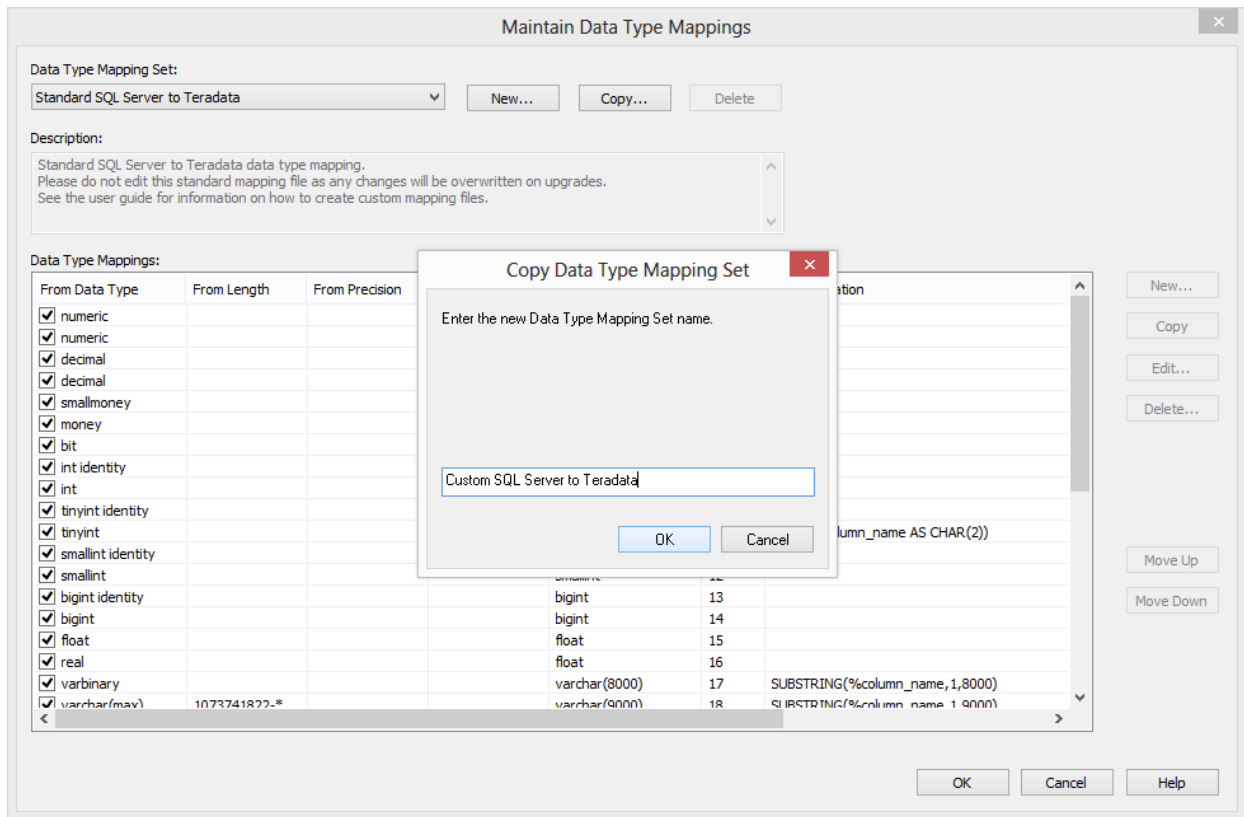
Note: The order of the data type mappings is significant as when loading a table, the procedure checks the data type mappings from top to bottom and stops when a data type and its parameters are correctly matched. A blank parameter means that it will match to anything.

COPYING A DATA TYPE MAPPING SET

To copy an existing data type mapping set, select **Tools/Data Type Mappings/Maintain Data Type Mappings...**



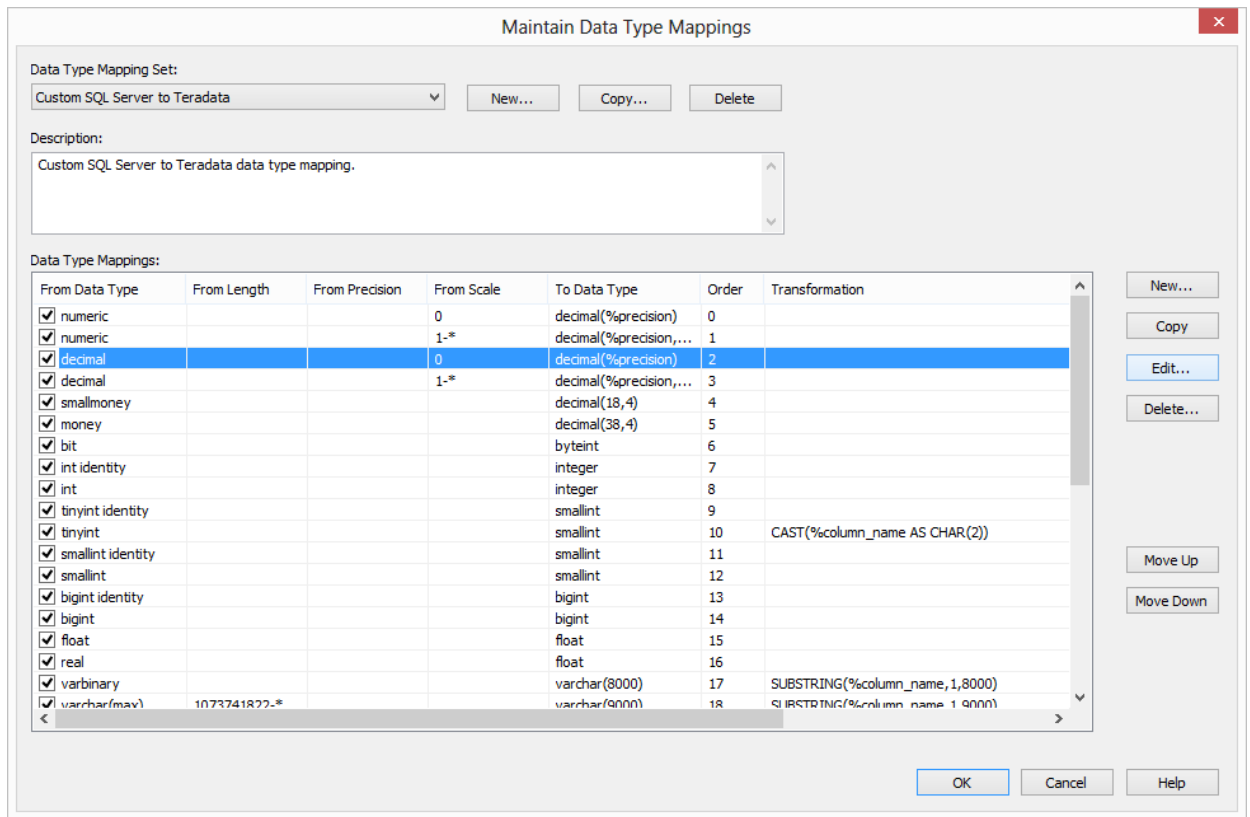
Select the Mapping Set to be copied from the drop-down list and then click on the **Copy** button. Enter the name of the new Mapping Set and click **OK**.



Description

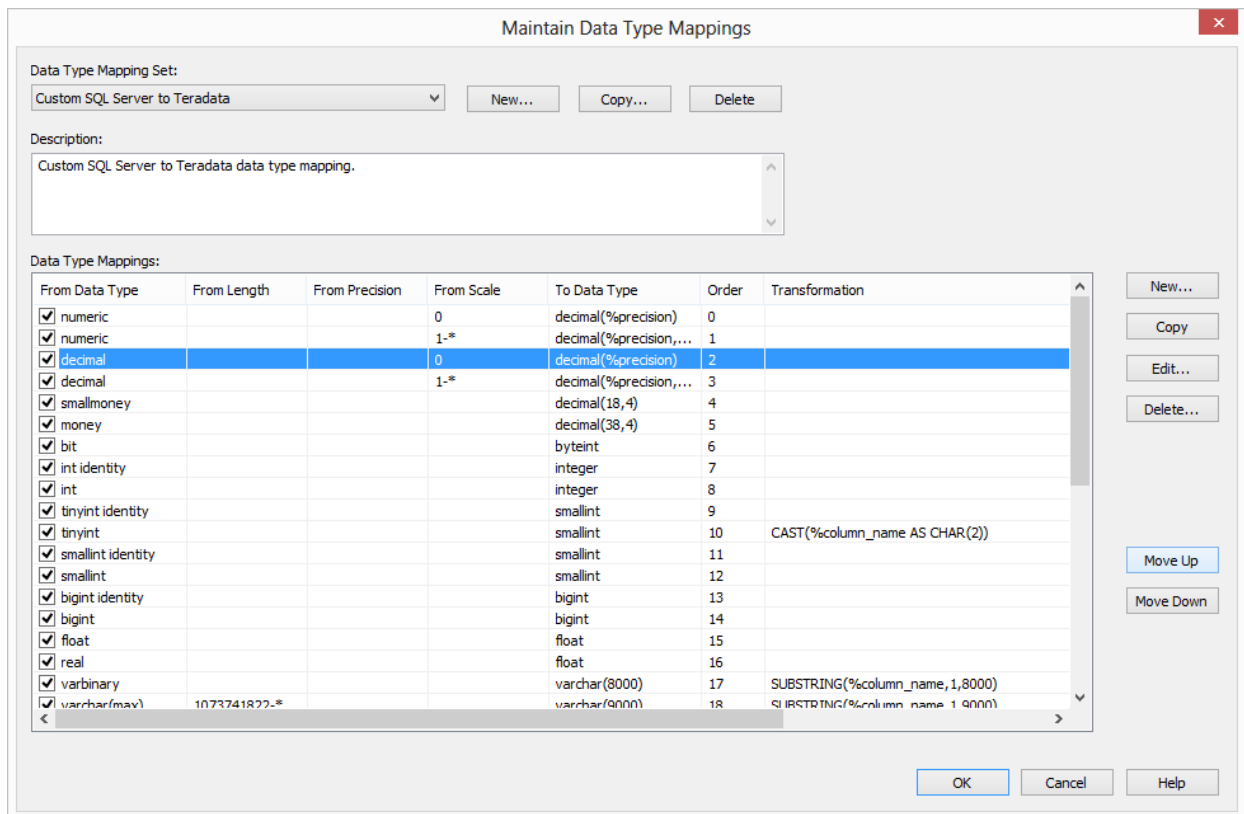
Enter the description for the data type mapping set.

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.

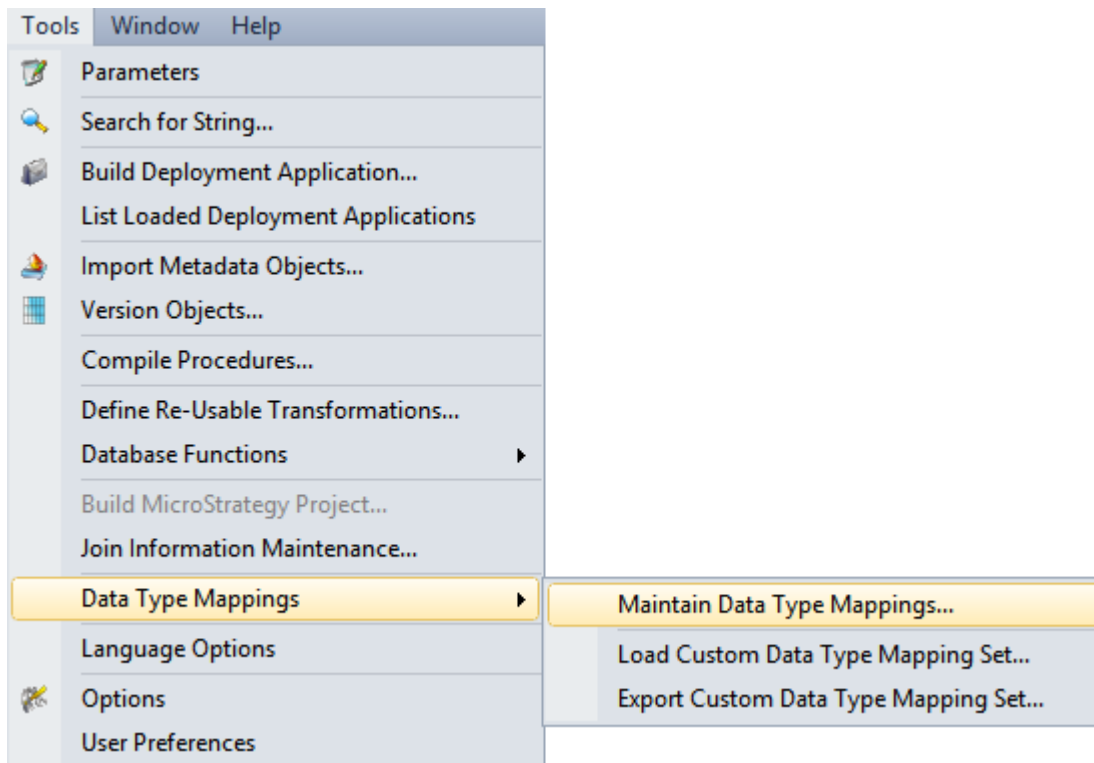


Note: The order of the data type mappings is significant as when loading a table, the procedure checks the data type mappings from top to bottom and stops when a data type and its parameters are correctly matched. A blank parameter means that it will match to anything.

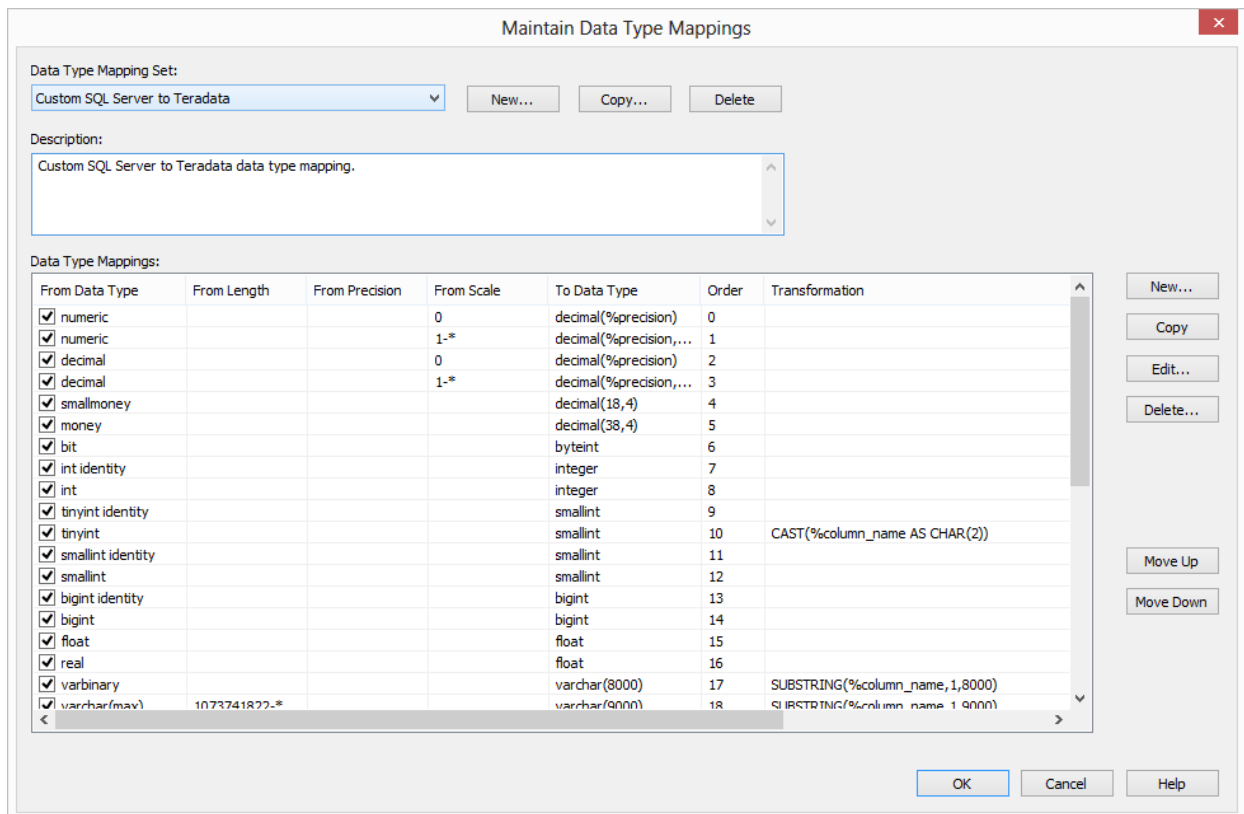
EDITING A DATA TYPE MAPPING SET

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.

Maintain Data Type Mappings

Data Type Mapping Set: Custom SQL Server to Teradata [New...] [Copy...] [Delete]

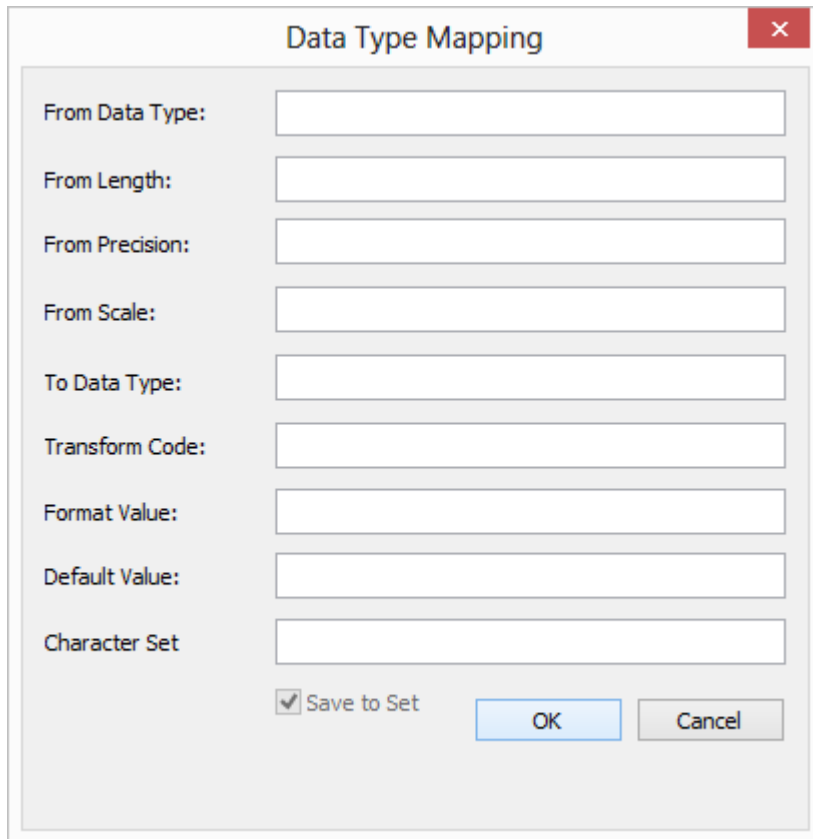
Description: Custom SQL Server to Teradata data type mapping.

| From Data Type | From Length | From Precision | From Scale | To Data Type | Order | Transformation |
|---|--------------|----------------|------------|------------------------|-------|--------------------------------|
| <input checked="" type="checkbox"/> numeric | | | 0 | decimal(%precision) | 0 | |
| <input checked="" type="checkbox"/> numeric | | | 1-* | decimal(%precision,... | 1 | |
| <input checked="" type="checkbox"/> decimal | | | 0 | decimal(%precision) | 2 | |
| <input checked="" type="checkbox"/> decimal | | | 1-* | decimal(%precision,... | 3 | |
| <input checked="" type="checkbox"/> smallmoney | | | | decimal(18,4) | 4 | |
| <input checked="" type="checkbox"/> money | | | | decimal(38,4) | 5 | |
| <input checked="" type="checkbox"/> bit | | | | byteint | 6 | |
| <input checked="" type="checkbox"/> int identity | | | | integer | 7 | |
| <input checked="" type="checkbox"/> int | | | | integer | 8 | |
| <input checked="" type="checkbox"/> tinyint identity | | | | smallint | 9 | |
| <input checked="" type="checkbox"/> tinyint | | | | smallint | 10 | CAST(%column_name AS CHAR(2)) |
| <input checked="" type="checkbox"/> smallint identity | | | | smallint | 11 | |
| <input checked="" type="checkbox"/> smallint | | | | smallint | 12 | |
| <input checked="" type="checkbox"/> bigint identity | | | | bigint | 13 | |
| <input checked="" type="checkbox"/> bigint | | | | bigint | 14 | |
| <input checked="" type="checkbox"/> float | | | | float | 15 | |
| <input checked="" type="checkbox"/> real | | | | float | 16 | |
| <input checked="" type="checkbox"/> varbinary | | | | varchar(8000) | 17 | SUBSTRING(%column_name,1,8000) |
| <input checked="" type="checkbox"/> varchar(max) | 1073741822-* | | | varchar(4000) | 18 | SUBSTRING(%column_name,1,4000) |

[New...] [Copy] [Edit...] [Delete...] [Move Up] [Move Down]

[OK] [Cancel] [Help]

- Enter the required fields and click **OK**.



The screenshot shows a dialog box titled "Data Type Mapping" with a close button (X) in the top right corner. The dialog contains the following fields and controls:

- From Data Type: [Text Input]
- From Length: [Text Input]
- From Precision: [Text Input]
- From Scale: [Text Input]
- To Data Type: [Text Input]
- Transform Code: [Text Input]
- Format Value: [Text Input]
- Default Value: [Text Input]
- Character Set: [Text Input]
- Save to Set
- OK [Button]
- Cancel [Button]

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 1048)

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 1048)

Format Value

The format of this Conversion's Data Type output.

Default Value

The default value of this Conversion's Data Type output.

Character Set

The Database-compliant character set used for storage. Type Latin or Unicode.

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 Teradata

Description: Custom SQL Server to Teradata data type mapping.

| From Data Type | From Length | From Precision | From Scale | To Data Type | Order | Transformation |
|---|--------------|----------------|------------|------------------------|-------|--------------------------------|
| <input checked="" type="checkbox"/> numeric | | | 0 | decimal(%precision) | 0 | |
| <input checked="" type="checkbox"/> numeric | | | 1-* | decimal(%precision,... | 1 | |
| <input checked="" type="checkbox"/> decimal | | | 0 | decimal(%precision) | 2 | |
| <input checked="" type="checkbox"/> decimal | | | 1-* | decimal(%precision,... | 3 | |
| <input checked="" type="checkbox"/> smallmoney | | | | decimal(18,4) | 4 | |
| <input checked="" type="checkbox"/> money | | | | decimal(38,4) | 5 | |
| <input checked="" type="checkbox"/> bit | | | | byteint | 6 | |
| <input checked="" type="checkbox"/> int identity | | | | integer | 7 | |
| <input checked="" type="checkbox"/> int | | | | integer | 8 | |
| <input checked="" type="checkbox"/> tinyint identity | | | | smallint | 9 | |
| <input checked="" type="checkbox"/> tinyint | | | | smallint | 10 | CAST(%column_name AS CHAR(2)) |
| <input checked="" type="checkbox"/> smallint identity | | | | smallint | 11 | |
| <input checked="" type="checkbox"/> smallint | | | | smallint | 12 | |
| <input checked="" type="checkbox"/> bigint identity | | | | bigint | 13 | |
| <input checked="" type="checkbox"/> bigint | | | | bigint | 14 | |
| <input checked="" type="checkbox"/> float | | | | float | 15 | |
| <input checked="" type="checkbox"/> real | | | | float | 16 | |
| <input checked="" type="checkbox"/> varbinary | | | | varchar(8000) | 17 | SUBSTRING(%column_name,1,8000) |
| <input checked="" type="checkbox"/> varchar(max) | 1073741822-* | | | varchar(4000) | 18 | SUBSTRING(%column_name 1 4000) |

Buttons: 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 want to edit and click the **Edit** button.

Maintain Data Type Mappings

Data Type Mapping Set: Custom SQL Server to Teradata

Description: Custom SQL Server to Teradata data type mapping.

| From Data Type | From Length | From Precision | From Scale | To Data Type | Order | Transformation |
|---|--------------|----------------|------------|------------------------|-------|----------------------------------|
| <input checked="" type="checkbox"/> numeric | | | 0 | decimal(%precision) | 0 | |
| <input checked="" type="checkbox"/> numeric | | | 1-* | decimal(%precision,... | 1 | |
| <input checked="" type="checkbox"/> decimal | | | 0 | decimal(%precision) | 2 | |
| <input checked="" type="checkbox"/> decimal | | | 1-* | decimal(%precision,... | 3 | |
| <input checked="" type="checkbox"/> smallmoney | | | | decimal(18,4) | 4 | |
| <input checked="" type="checkbox"/> money | | | | decimal(38,4) | 5 | |
| <input checked="" type="checkbox"/> bit | | | | byteint | 6 | |
| <input checked="" type="checkbox"/> int identity | | | | integer | 7 | |
| <input checked="" type="checkbox"/> int | | | | integer | 8 | |
| <input checked="" type="checkbox"/> tinyint identity | | | | smallint | 9 | |
| <input checked="" type="checkbox"/> tinyint | | | | smallint | 10 | CAST(%column_name AS CHAR(2)) |
| <input checked="" type="checkbox"/> smallint identity | | | | smallint | 11 | |
| <input checked="" type="checkbox"/> smallint | | | | smallint | 12 | |
| <input checked="" type="checkbox"/> bigint identity | | | | bigint | 13 | |
| <input checked="" type="checkbox"/> bigint | | | | bigint | 14 | |
| <input checked="" type="checkbox"/> float | | | | float | 15 | |
| <input checked="" type="checkbox"/> real | | | | float | 16 | |
| <input checked="" type="checkbox"/> varbinary | | | | varchar(8000) | 17 | SUBSTRING(%column_name, 1, 8000) |
| <input checked="" type="checkbox"/> varchar(max) | 1073741822-* | | | varchar(4000) | 18 | SUBSTRING(%column_name 1 4000) |

Buttons: New..., Copy, Edit..., Delete..., Move Up, Move Down, OK, Cancel, Help

- Change any fields as required and click **OK**.

Data Type Mapping [X]

From Data Type:

From Length:

From Precision:

From Scale:

To Data Type:

Transform Code:

Format Value:

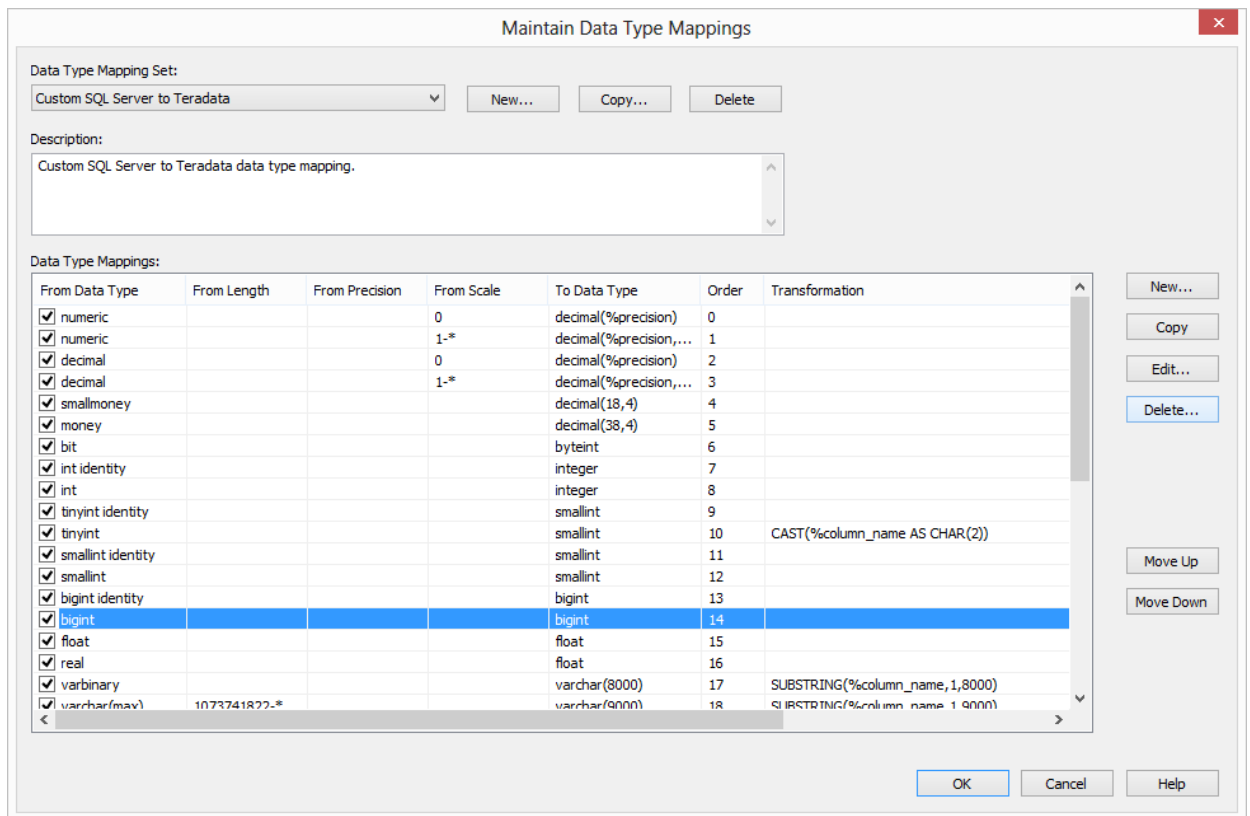
Default Value:

Character Set:

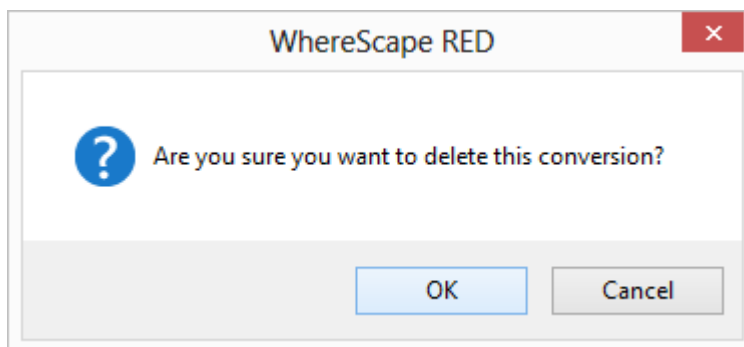
Save to Set

To delete an existing mapping in the data type mapping set:

- Select the mapping and then click the **Delete** button.

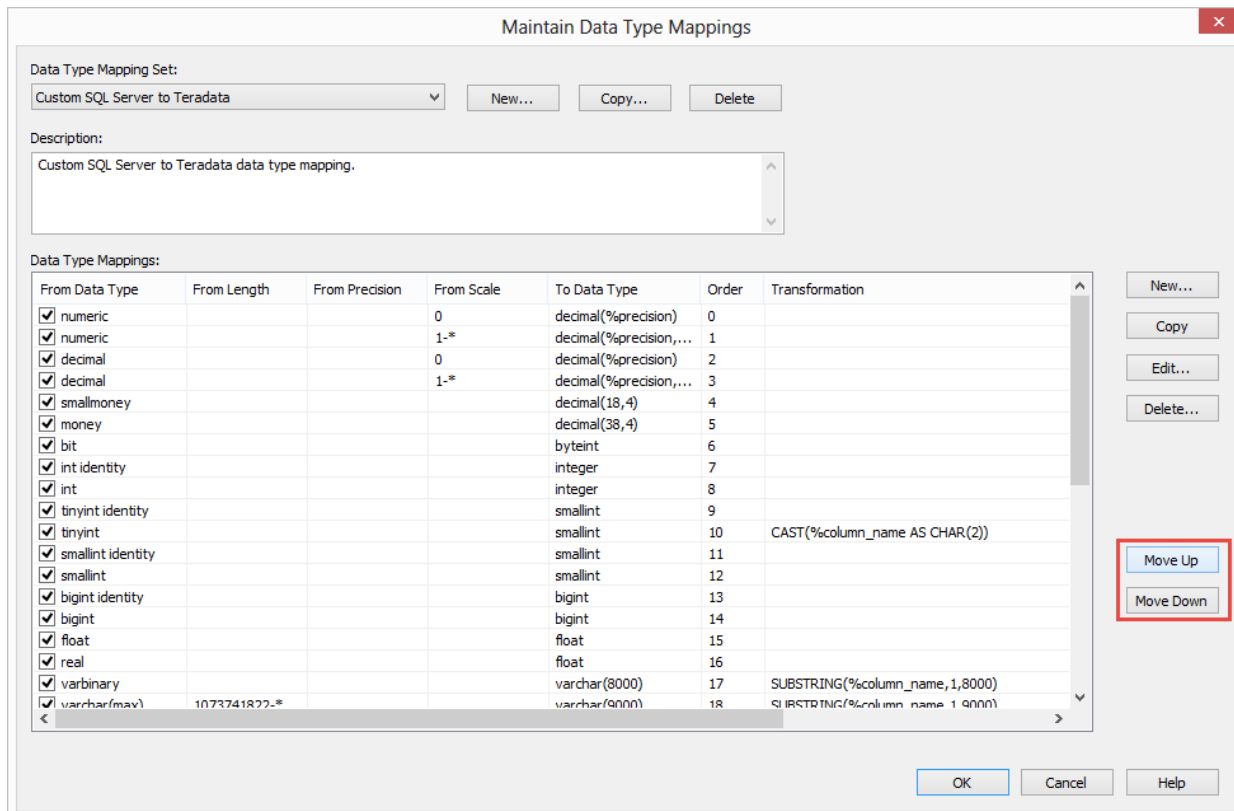


- Click **OK** to delete.



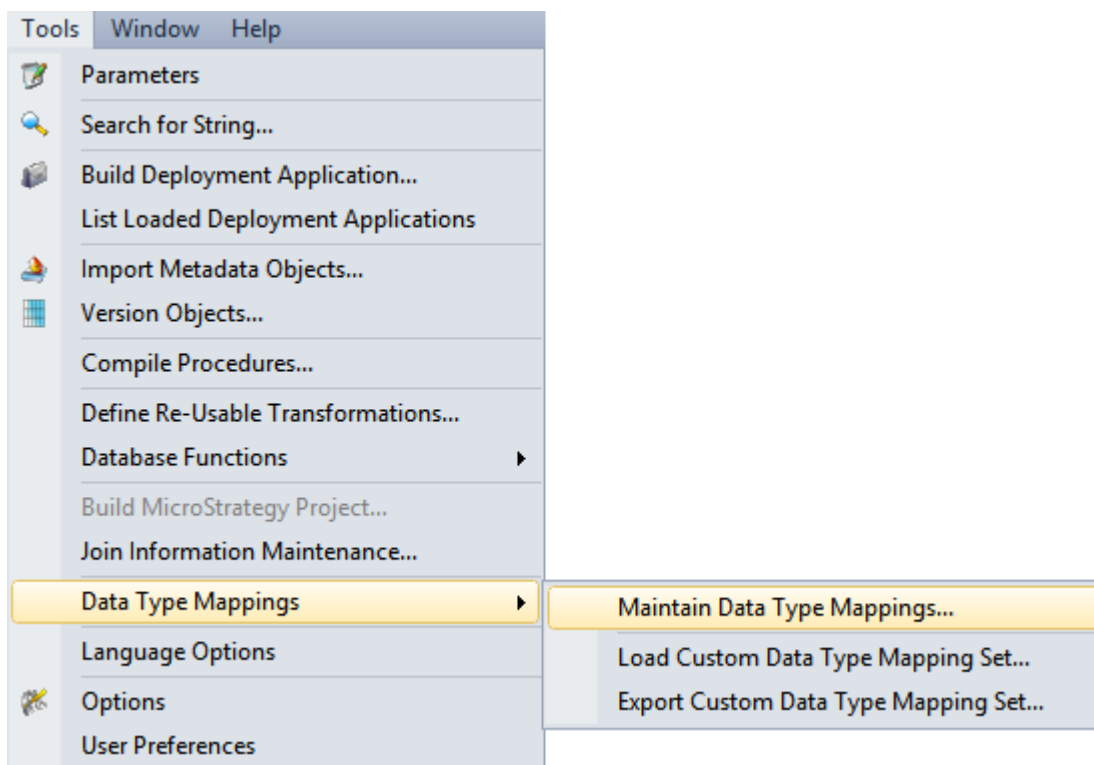
To move a mapping in the data type mapping set up or down in the list:

- Select a mapping and then click on the **Move Up** button on the right to move the mapping up in the list;
- Or click on the **Move Down** button on the right to move the mapping down in the list.

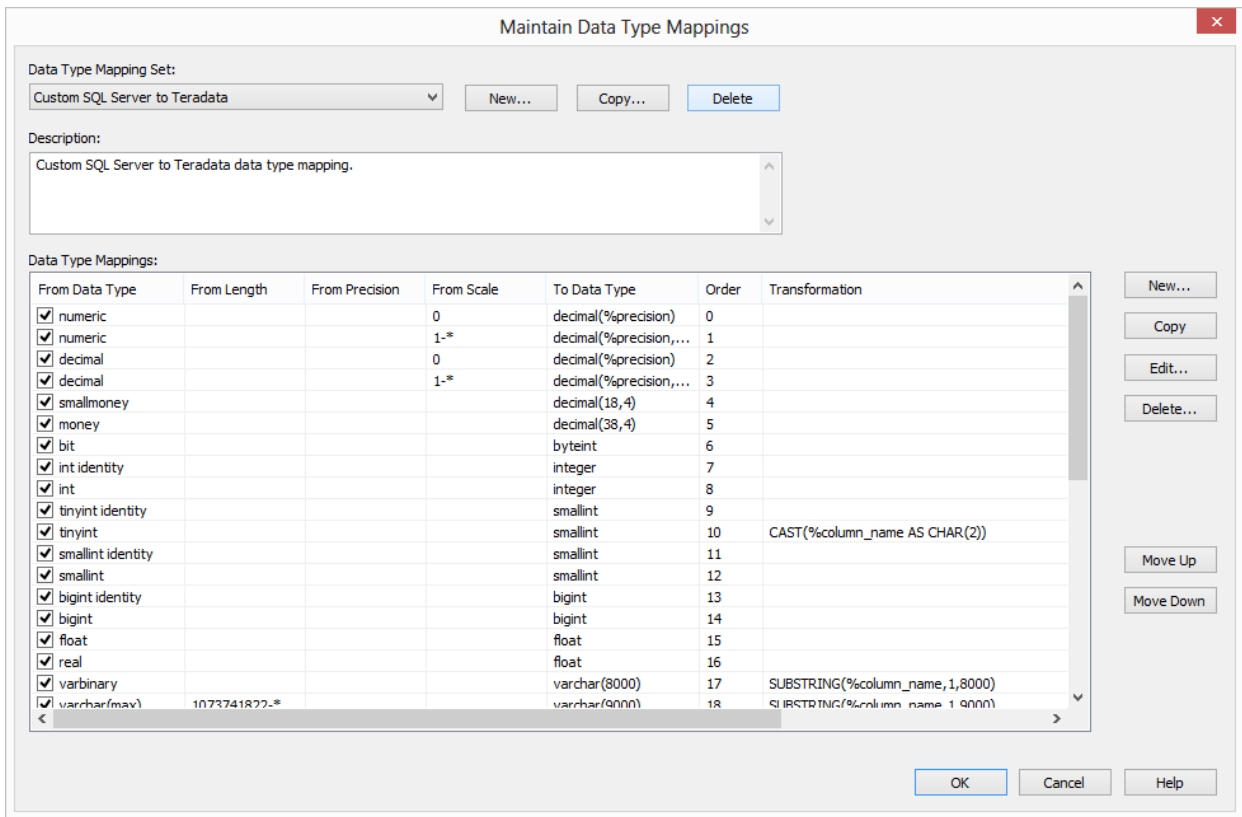


DELETING A DATA TYPE MAPPING SET

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.

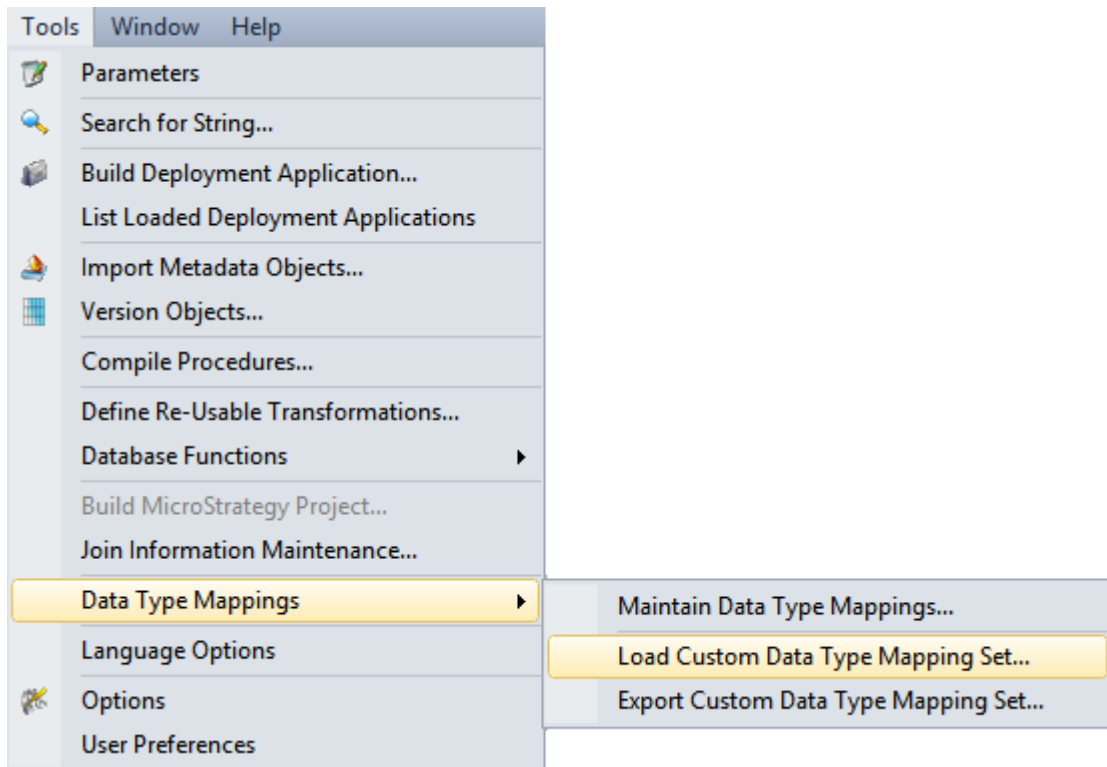


Note: The **Delete** button is disabled for standard data type mapping sets.

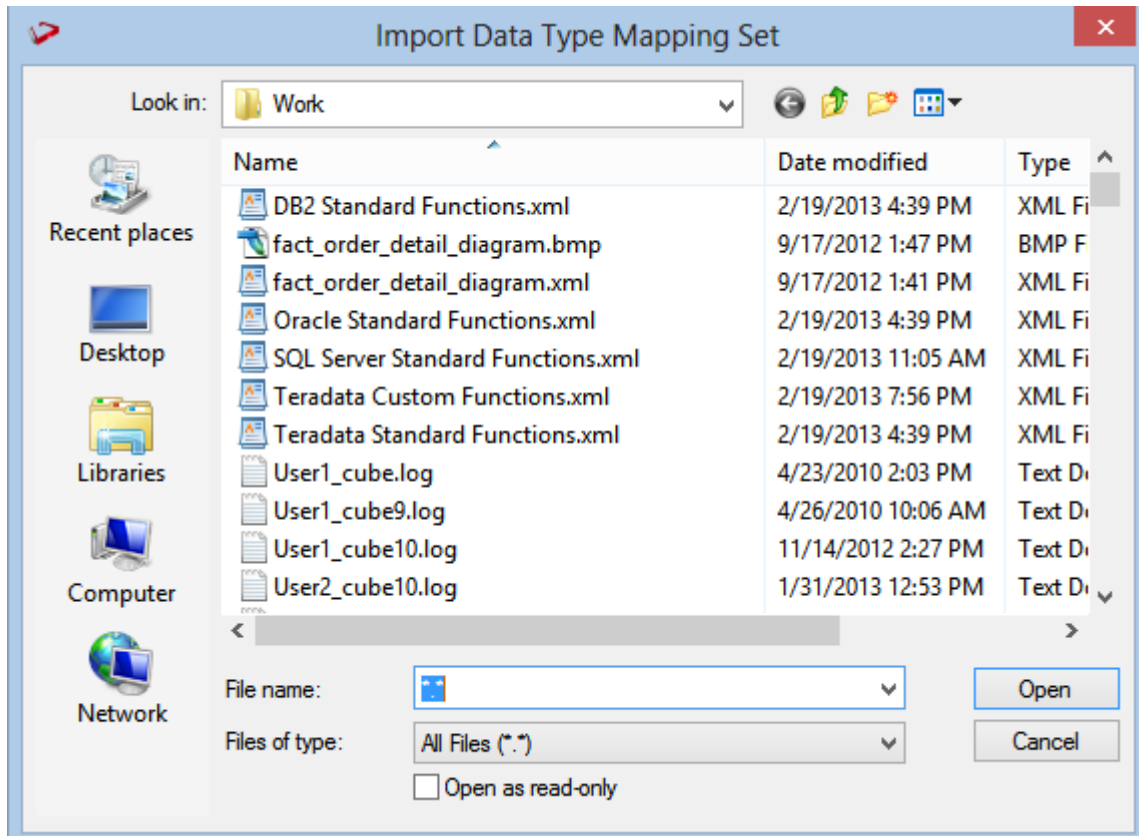
LOADING DATA TYPE MAPPING SETS

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...**



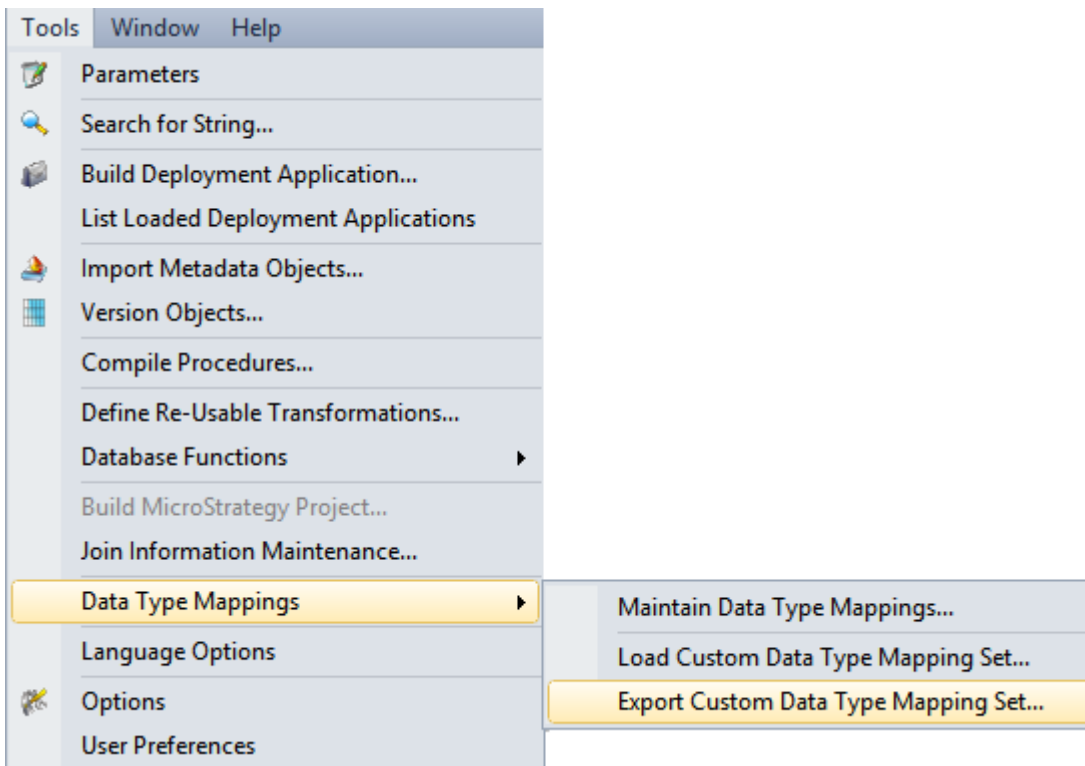
The following dialog is displayed. Select the xml file to load the data type mappings. By default RED expects the xml files to be in **ProgramData\WhereScape\Work**



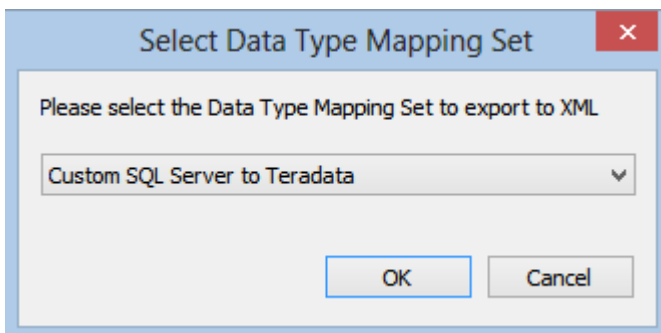
EXPORTING DATA TYPE MAPPING SETS

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.

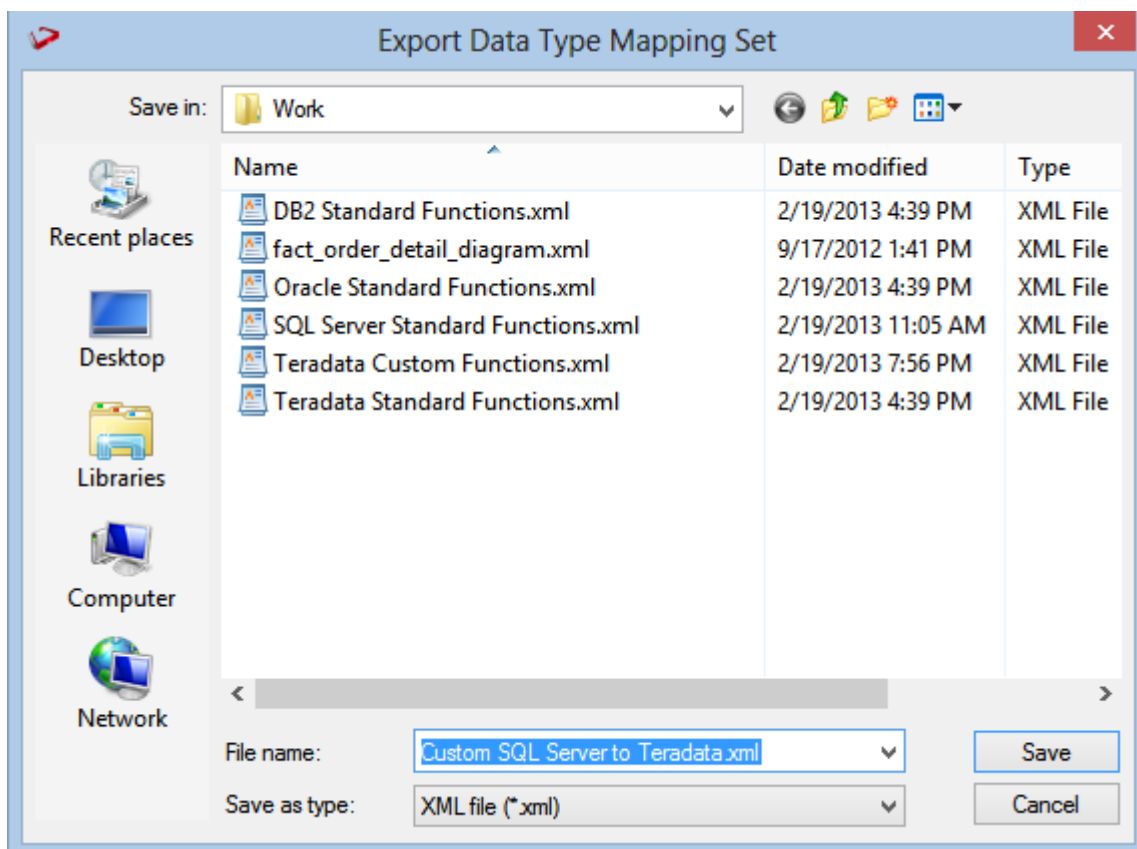
To export a data type mapping set, select **Tools/Data Type Mappings/Export Custom Data Type Mapping Set...**



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.

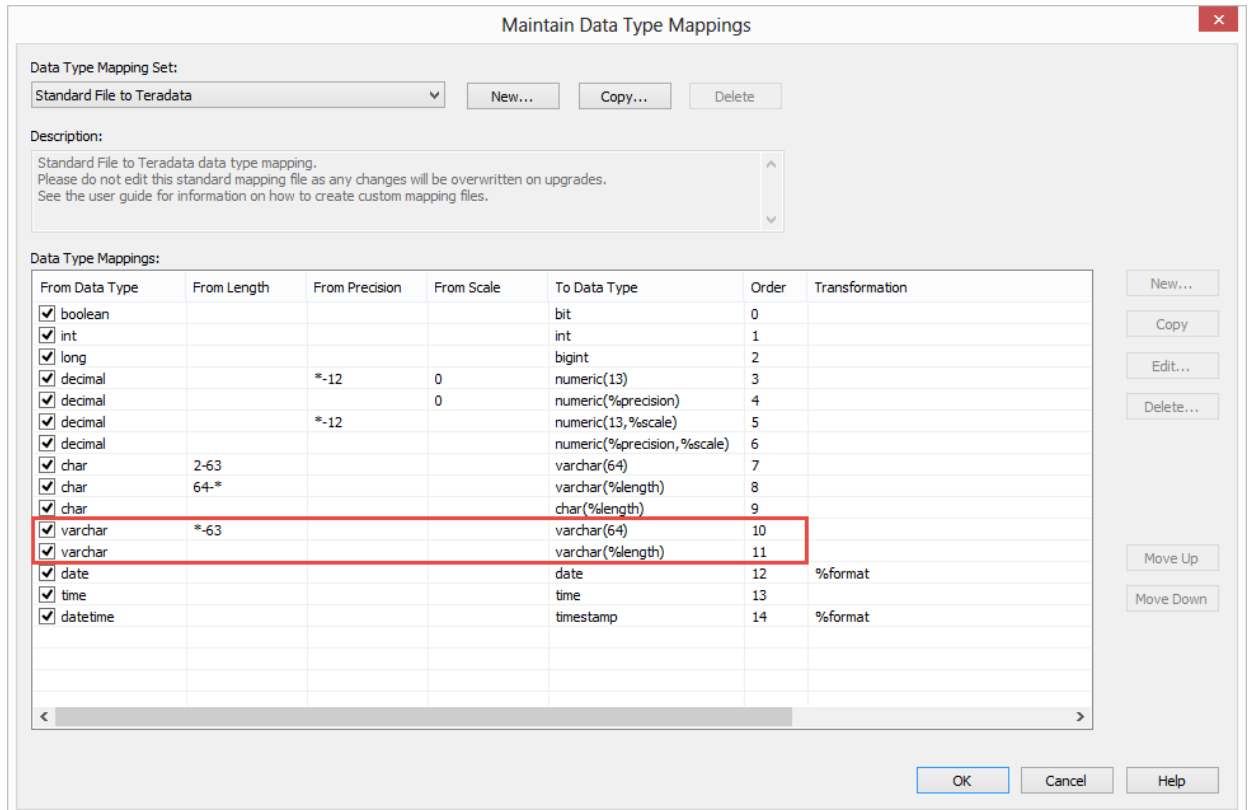
The examples in this topic demonstrate how **Custom** Data Type Mapping Sets can be configured using the following variables:

- %length
- %scale
- %precision
- %table_name
- %column_name
- %format
- %prompt

%length

In the example below, when converting a varchar in a file to Teradata 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 DB2 to Teradata format, we follow the following steps in the given order:

- If the decimal has a scale of zero, the **data type** will become decimal(%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 decimal(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 decimal(%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 decimal(8,3).

Maintain Data Type Mappings ✕

Data Type Mapping Set:
 Standard DB2 to Teradata New... Copy... Delete

Description:
 Standard DB2 to Teradata 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 |
|--|-------------|----------------|------------|----------------------------|-------|----------------|
| <input checked="" type="checkbox"/> decimal | | | 0 | decimal(%precision) | 0 | |
| <input checked="" type="checkbox"/> decimal | | | 1-* | decimal(%precision,%scale) | 1 | |
| <input checked="" type="checkbox"/> smallint | | | | smallint | 2 | |
| <input checked="" type="checkbox"/> integer | | | | integer | 3 | |
| <input checked="" type="checkbox"/> bigint | | | | bigint | 4 | |
| <input checked="" type="checkbox"/> real | | | | float | 5 | |
| <input checked="" type="checkbox"/> double | | | | float | 6 | |
| <input checked="" type="checkbox"/> decfloat | | | | float | 7 | |
| <input checked="" type="checkbox"/> number | | | | float | 8 | |
| <input checked="" type="checkbox"/> long varchar | | | | clob | 9 | |
| <input checked="" type="checkbox"/> clob | | | | clob | 10 | |
| <input checked="" type="checkbox"/> date | | | | timestamp | 11 | |
| <input checked="" type="checkbox"/> time | | | | time | 12 | |
| <input checked="" type="checkbox"/> timestamp | | | | timestamp | 13 | |

New...
Copy
Edit...
Delete...

Move Up
Move Down

OK Cancel Help

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 varchar in SQL Server to Teradata format, we follow the following steps in the given order:

- If the varchar has a length of 1 or greater, the **data type** will become varchar(%precision); where we substitute the number of digits in the varchar for the variable '%precision'. Thus if the varchar has a length of 14, the resulting data type will be varchar(14).
- If the varchar has a length of 0, the data type will be become varchar(9000) where the transformation will ensure that the correct length of the source string.

Maintain Data Type Mappings

Data Type Mapping Set: Custom SQL Server to Teradata

Description: Custom SQL Server to Teradata data type mapping.

| From Data Type | From Length | From Precision | From Scale | To Data Type | Order | Transformation |
|---|--------------|----------------|------------|---------------------|-------|--------------------------------|
| <input checked="" type="checkbox"/> bigint identity | | | | bigint | 14 | |
| <input checked="" type="checkbox"/> bigint | | | | bigint | 15 | |
| <input checked="" type="checkbox"/> float | | | | float | 16 | |
| <input checked="" type="checkbox"/> real | | | | float | 17 | |
| <input checked="" type="checkbox"/> varbinary | | | | varchar(8000) | 18 | SUBSTRING(%column_name,1,8000) |
| <input checked="" type="checkbox"/> varchar(max) | 1073741822-* | | | varchar(9000) | 19 | SUBSTRING(%column_name,1,9000) |
| <input checked="" type="checkbox"/> varchar | 0 | | | varchar(9000) | 20 | SUBSTRING(%column_name,1,9000) |
| <input checked="" type="checkbox"/> varchar | 1-8000 | | | varchar(%precision) | 21 | |
| <input checked="" type="checkbox"/> text | | | | varchar(9000) | 22 | SUBSTRING(%column_name,1,9000) |
| <input checked="" type="checkbox"/> nchar | | | | char(%precision) | 23 | |
| <input checked="" type="checkbox"/> nvarchar | 1073741822-* | | | varchar(9000) | 24 | SUBSTRING(%column_name,1,9000) |
| <input checked="" type="checkbox"/> nvarchar | 0 | | | varchar(8000) | 25 | SUBSTRING(%column_name,1,8000) |
| <input checked="" type="checkbox"/> nvarchar | 1-8000 | | | varchar(%precision) | 26 | |
| <input checked="" type="checkbox"/> ntext | | | | varchar(9000) | 27 | SUBSTRING(%column_name,1,9000) |
| <input checked="" type="checkbox"/> date | | | | date | 28 | |
| <input checked="" type="checkbox"/> datetime | | | | timestamp(3) | 29 | |
| <input checked="" type="checkbox"/> datetime2 | | | | timestamp | 30 | CAST(%column_name AS CHAR(26)) |
| <input checked="" type="checkbox"/> smalldatetime | | | | timestamp(3) | 31 | |
| <input checked="" type="checkbox"/> time | | | | time | 32 | |

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'.

Maintain Data Type Mappings

Data Type Mapping Set:
 Custom SQL Server to Teradata [New... Copy... Delete]

Description:
 Custom SQL Server to Teradata data type mapping.

Data Type Mappings:

| From Data Type | From Length | From Precision | From Scale | To Data Type | Order | Transformation |
|--|--------------|----------------|------------|---------------------|-------|--|
| <input checked="" type="checkbox"/> varchar | 1-8000 | | | varchar(%precision) | 21 | |
| <input checked="" type="checkbox"/> text | | | | varchar(9000) | 22 | SUBSTRING(%column_name,1,9000) |
| <input checked="" type="checkbox"/> nchar | | | | char(%precision) | 23 | |
| <input checked="" type="checkbox"/> nvarchar | 1073741822-* | | | varchar(9000) | 24 | SUBSTRING(%column_name,1,9000) |
| <input checked="" type="checkbox"/> nvarchar | 0 | | | varchar(8000) | 25 | SUBSTRING(%column_name,1,8000) |
| <input checked="" type="checkbox"/> nvarchar | 1-8000 | | | varchar(%precision) | 26 | |
| <input checked="" type="checkbox"/> varchar | | 1-2 | | varchar(%precision) | 27 | COALESCE(%table_name.%column_name,'U') |
| <input checked="" type="checkbox"/> varchar | | 3-6 | | varchar(%precision) | 28 | COALESCE(%table_name.%column_name,'UNK') |
| <input checked="" type="checkbox"/> varchar | | 7-* | | varchar(%precision) | 29 | COALESCE(%table_name.%column_name,'UNKNOWN') |
| <input checked="" type="checkbox"/> ntext | | | | varchar(9000) | 30 | SUBSTRING(%column_name,1,9000) |
| <input checked="" type="checkbox"/> date | | | | date | 31 | |
| <input checked="" type="checkbox"/> datetime | | | | timestamp(3) | 32 | |
| <input checked="" type="checkbox"/> datetime2 | | | | timestamp | 33 | CAST(%column_name AS CHAR(26)) |
| <input checked="" type="checkbox"/> smalldatetime | | | | timestamp(3) | 34 | |
| <input checked="" type="checkbox"/> time | | | | time | 35 | |
| <input checked="" type="checkbox"/> datetimeoffset | | | | timestamp(3) | 36 | |
| <input checked="" type="checkbox"/> timestamp | | | | timestamp | 37 | |
| <input checked="" type="checkbox"/> image | | | | varchar(8000) | 38 | |
| <input checked="" type="checkbox"/> uniqueidentifier | | | | char(36) | 39 | |

Buttons: New... Copy Edit... Delete... Move Up Move Down OK Cancel Help

%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).

The screenshot shows the 'Maintain Data Type Mappings' dialog box. It features a dropdown menu for 'Data Type Mapping Set' set to 'Standard File to Teradata'. Below this is a 'Description' field with text: 'Standard File to Teradata 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.' The main part of the dialog is a table titled 'Data Type Mappings' with the following columns: From Data Type, From Length, From Precision, From Scale, To Data Type, Order, and Transformation. The table contains 14 rows of mappings. The last three rows are highlighted with red boxes:

| From Data Type | From Length | From Precision | From Scale | To Data Type | Order | Transformation |
|--|-------------|----------------|------------|------------------------|-------|----------------|
| <input checked="" type="checkbox"/> boolean | | | | bit | 0 | |
| <input checked="" type="checkbox"/> int | | | | int | 1 | |
| <input checked="" type="checkbox"/> long | | | | bigint | 2 | |
| <input checked="" type="checkbox"/> decimal | | *-12 | 0 | numeric(13) | 3 | |
| <input checked="" type="checkbox"/> decimal | | | 0 | numeric(%precision) | 4 | |
| <input checked="" type="checkbox"/> decimal | | *-12 | | numeric(13,%scale) | 5 | |
| <input checked="" type="checkbox"/> decimal | | | | numeric(%precision...) | 6 | |
| <input checked="" type="checkbox"/> char | 2-63 | | | varchar(64) | 7 | |
| <input checked="" type="checkbox"/> char | 64-* | | | varchar(%length) | 8 | |
| <input checked="" type="checkbox"/> char | | | | char(%length) | 9 | |
| <input checked="" type="checkbox"/> varchar | *-63 | | | varchar(64) | 10 | |
| <input checked="" type="checkbox"/> varchar | | | | varchar(%length) | 11 | |
| <input checked="" type="checkbox"/> date | | | | date | 12 | %format |
| <input checked="" type="checkbox"/> time | | | | time | 13 | |
| <input checked="" type="checkbox"/> datetime | | | | timestamp | 14 | %format |

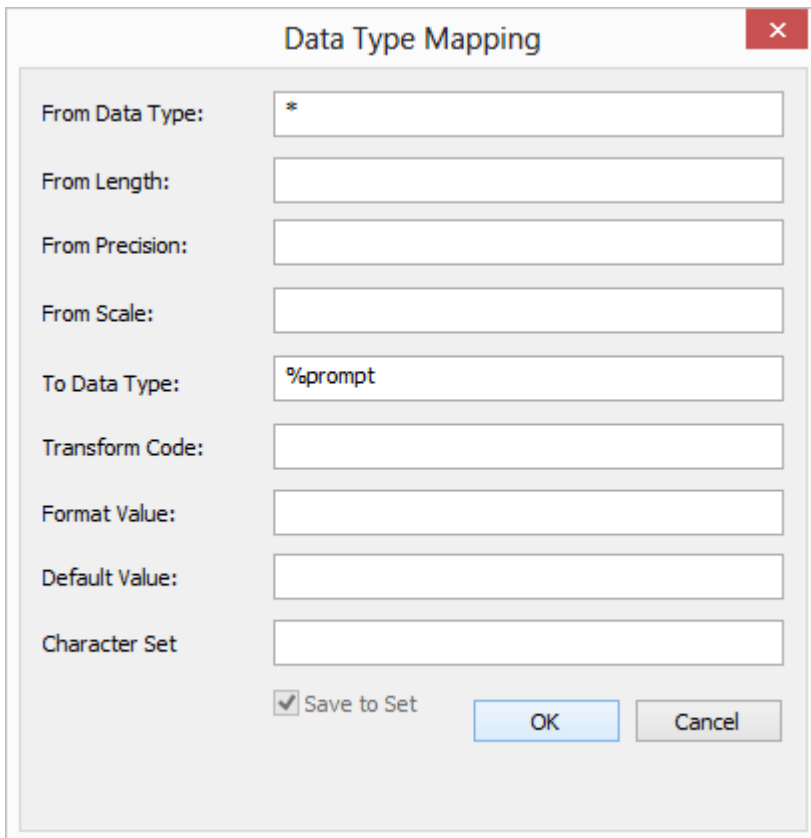
At the bottom of the dialog are 'OK', 'Cancel', and 'Help' buttons. On the right side, there are buttons for 'New...', 'Copy', 'Edit...', 'Delete...', 'Move Up', and 'Move Down'.

%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 placeholder 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.



The screenshot shows a dialog box titled "Data Type Mapping" with a close button (X) in the top right corner. The dialog contains several input fields and a checkbox:

- From Data Type:** * (text input)
- From Length:** (empty text input)
- From Precision:** (empty text input)
- From Scale:** (empty text input)
- To Data Type:** %prompt (text input)
- Transform Code:** (empty text input)
- Format Value:** (empty text input)
- Default Value:** (empty text input)
- Character Set:** (empty text input)
- Save to Set (checkbox)
- OK (button)
- Cancel (button)

- 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.

Data Type Mapping [X]

From Data Type:

From Length:

From Precision:

From Scale:

To Data Type:

Transform Code:

Format Value:

Default Value:

Character Set:

Save to Set

- 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.

Maintain Data Type Mappings

Data Type Mapping Set:
 Custom SQL Server to Teradata New... Copy... Delete

Description:
 Custom SQL Server to Teradata data type mapping.

Data Type Mappings:

| From Data Type | From Length | From Precision | From Scale | To Data Type | Order | Transformation |
|--|-------------|----------------|------------|---------------------|-------|--|
| <input checked="" type="checkbox"/> nvarchar | 0 | | | varchar(8000) | 25 | SUBSTRING(%column_name,1,8000) |
| <input checked="" type="checkbox"/> nvarchar | 1-8000 | | | varchar(%precision) | 26 | |
| <input checked="" type="checkbox"/> varchar | | 1-2 | | varchar(%precision) | 27 | COALESCE(%table_name.%column_name,'U') |
| <input checked="" type="checkbox"/> varchar | | 3-6 | | varchar(%precision) | 28 | COALESCE(%table_name.%column_name,'UNK') |
| <input checked="" type="checkbox"/> varchar | | 7-* | | varchar(%precision) | 29 | COALESCE(%table_name.%column_name,'UN...') |
| <input checked="" type="checkbox"/> ntext | | | | varchar(9000) | 30 | SUBSTRING(%column_name,1,9000) |
| <input checked="" type="checkbox"/> date | | | | date | 31 | |
| <input checked="" type="checkbox"/> datetime | | | | timestamp(3) | 32 | |
| <input checked="" type="checkbox"/> datetime2 | | | | timestamp | 33 | CAST(%column_name AS CHAR(26)) |
| <input checked="" type="checkbox"/> smalldatetime | | | | timestamp(3) | 34 | |
| <input checked="" type="checkbox"/> time | | | | time | 35 | |
| <input checked="" type="checkbox"/> datetimeoffset | | | | timestamp(3) | 36 | |
| <input checked="" type="checkbox"/> timestamp | | | | timestamp | 37 | |
| <input checked="" type="checkbox"/> image | | | | varchar(8000) | 38 | |
| <input checked="" type="checkbox"/> uniqueidentifier | | | | char(36) | 39 | |
| <input checked="" type="checkbox"/> geography | 0 | 0 | 0 | varchar(30) | 40 | |
| <input checked="" type="checkbox"/> * | | | | %prompt | 41 | |
| <input checked="" type="checkbox"/> (New) | | | | | 42 | |

New... Copy Edit... Delete...

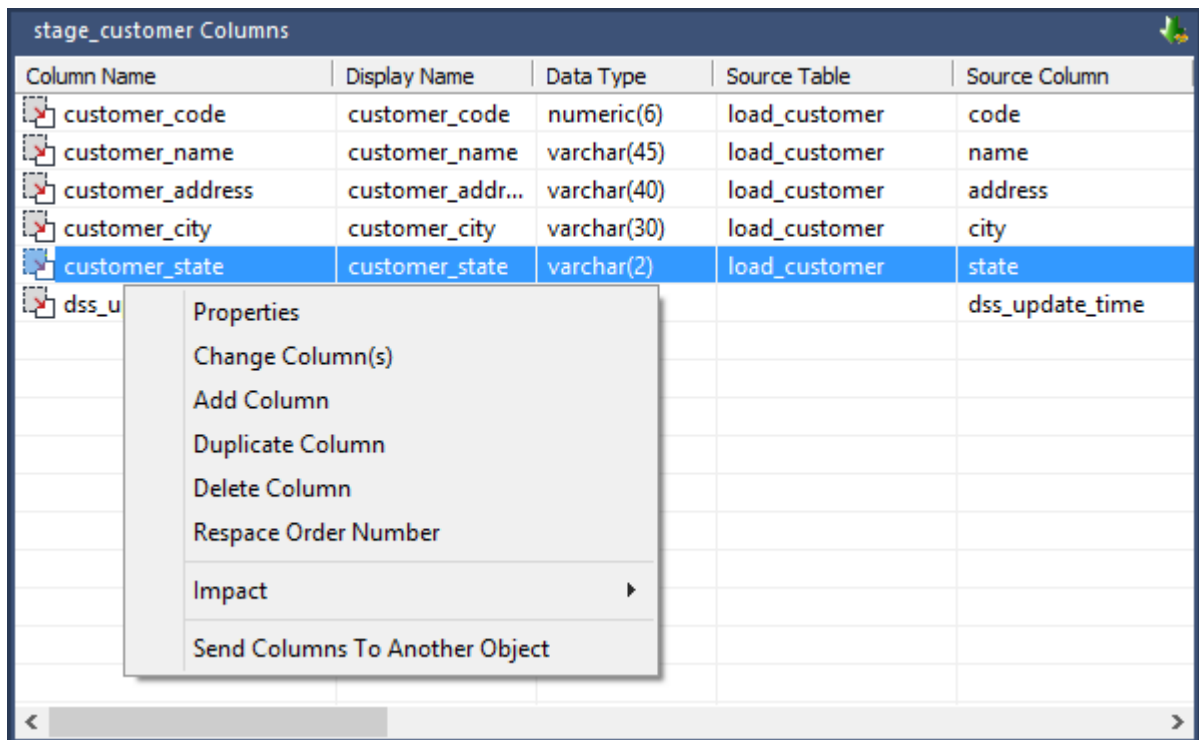
Move Up Move Down

OK Cancel Help

CHAPTER 42

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.

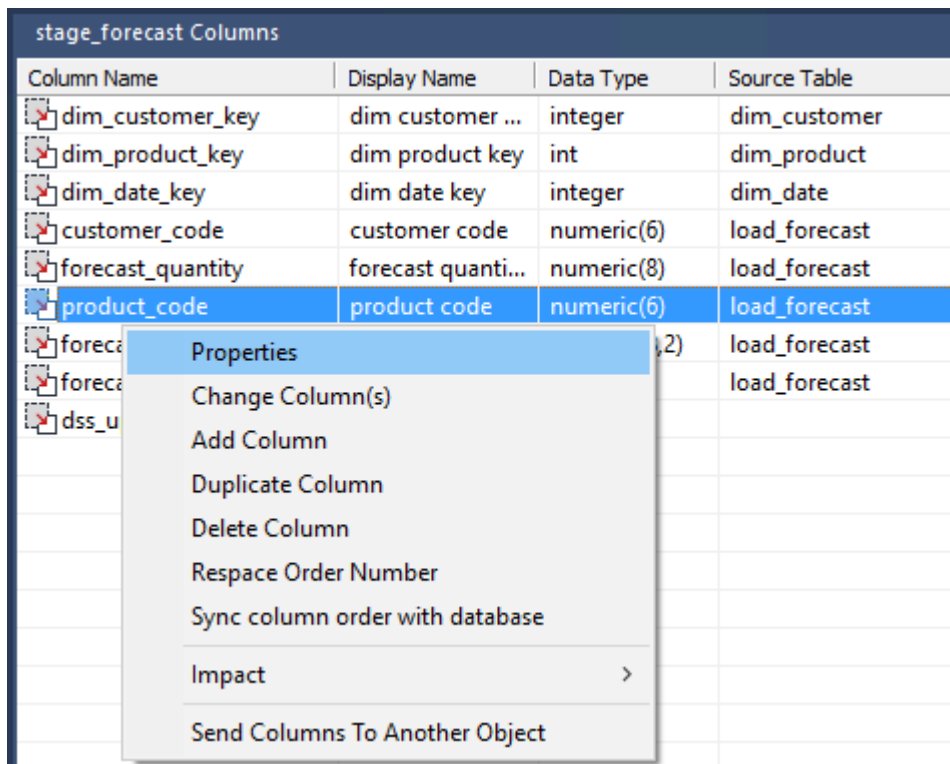


IN THIS CHAPTER

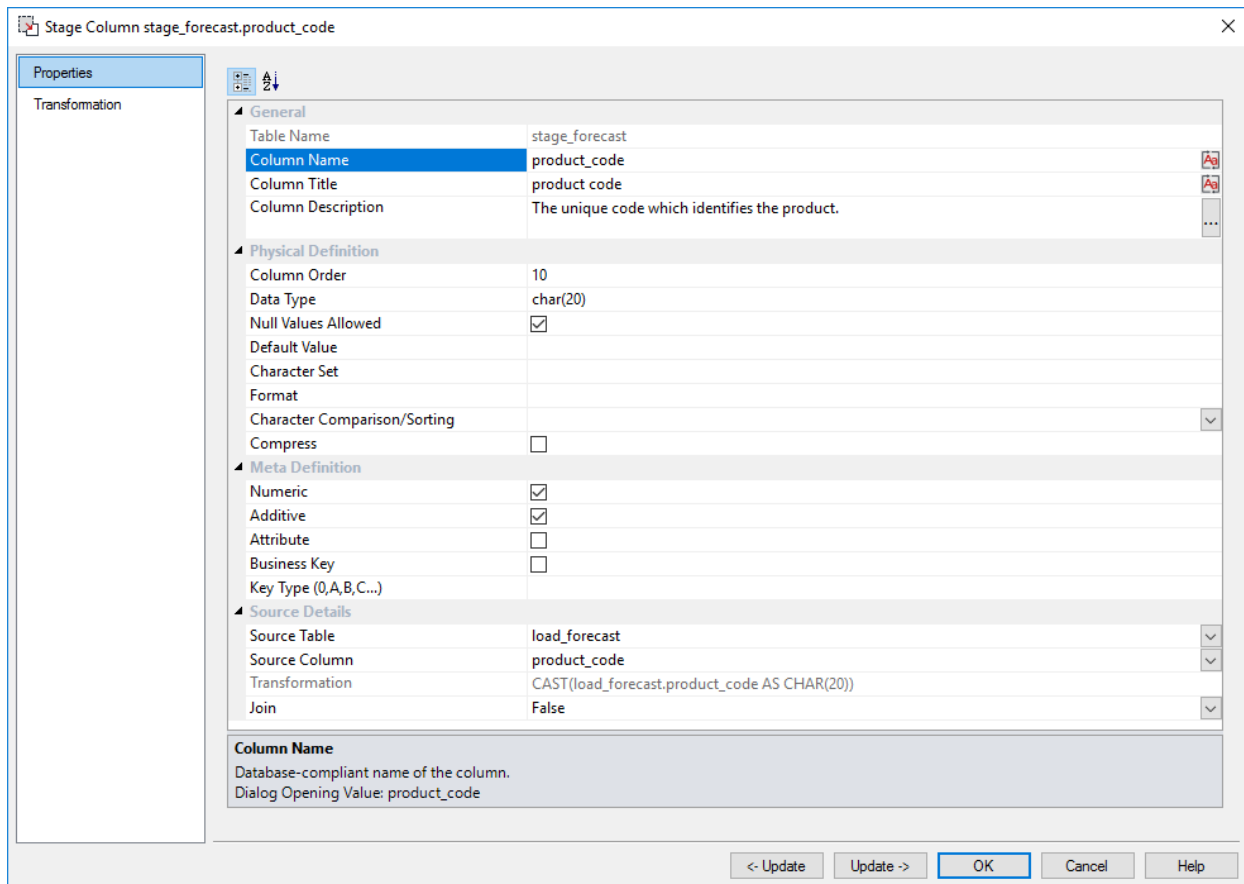
| | |
|--------------------------------------|------|
| Properties | 1059 |
| Change Column(s)..... | 1062 |
| Add Column..... | 1064 |
| Duplicate Column | 1065 |
| Delete Column..... | 1067 |
| Re-space Order Number | 1068 |
| Impact | 1069 |
| Sync Column order with database..... | 1071 |
| Send Columns to Another Object..... | 1072 |

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.

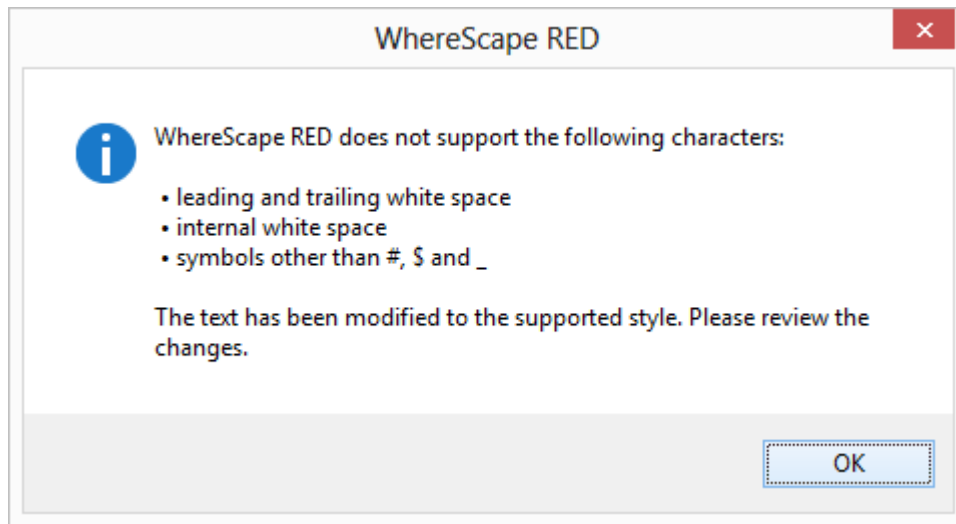


Warning: WhereScape RED does not support the following characters in **Column Names**:

- 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 303).

For Stage tables, see *Stage Table Column Properties* (on page 331).

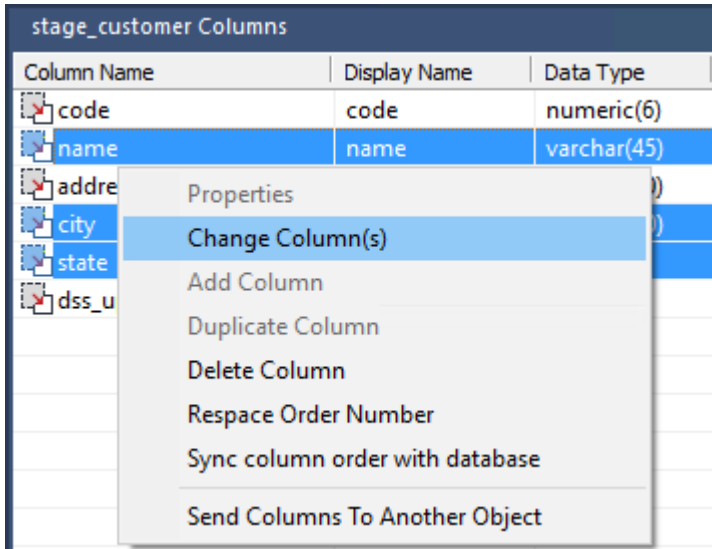
For Model tables, see *Model Column Properties* (see "*Model Table Column Properties*" on page 471).

For Data Store tables, see *Data Store Column Properties* (on page 371).

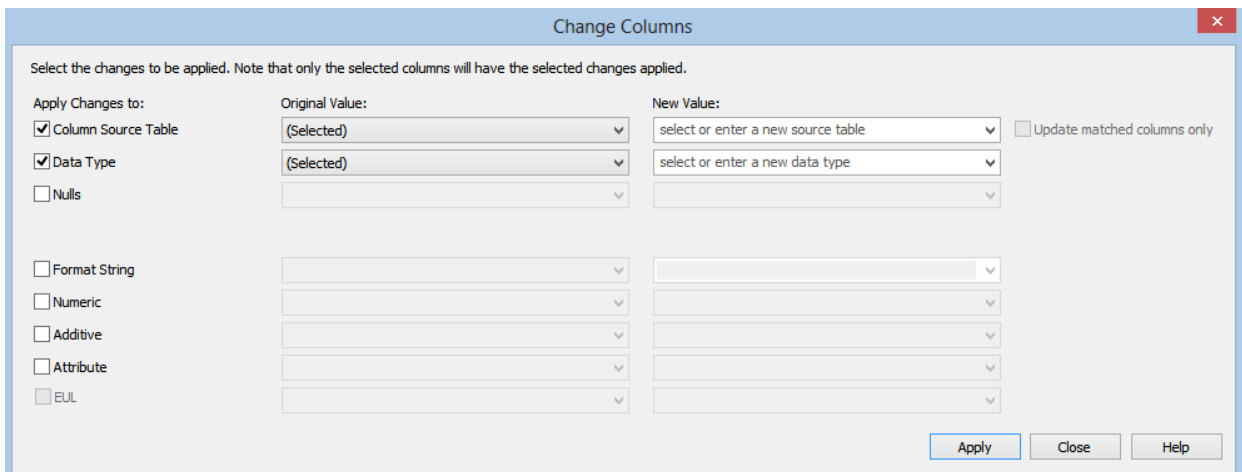
For EDW 3NF tables, see *EDW 3NF Column Properties* (see "*EDW 3NF Table Column Properties*" on page 395).

CHANGE COLUMN(S)

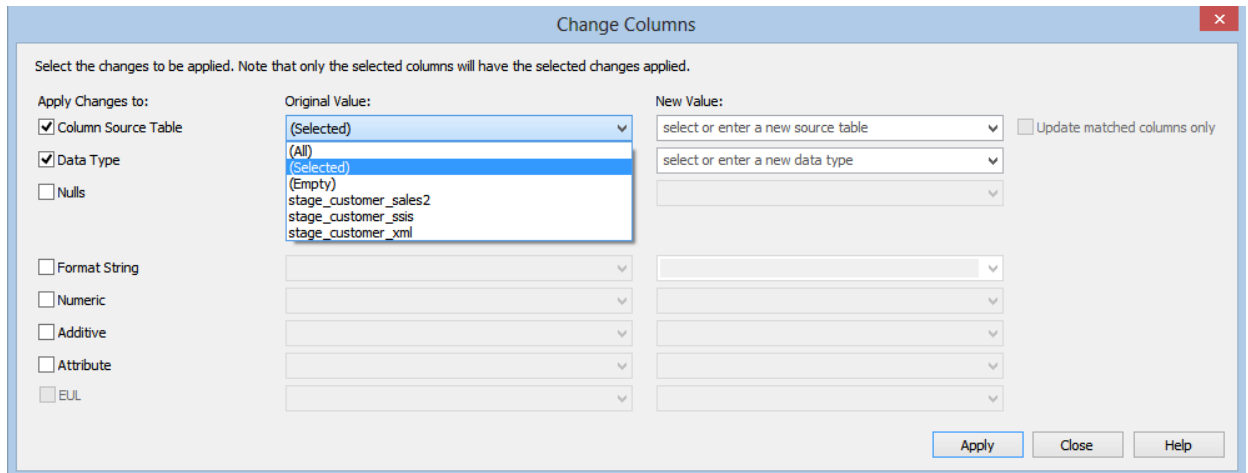
To change the properties for multiple columns, right-click on a column in the middle pane and select **Change Column(s)**.



To change a column property, you first need to select the relevant **checkboxes** on the left. Each checkbox, when selected, allows you to change the value for that field in the column properties.



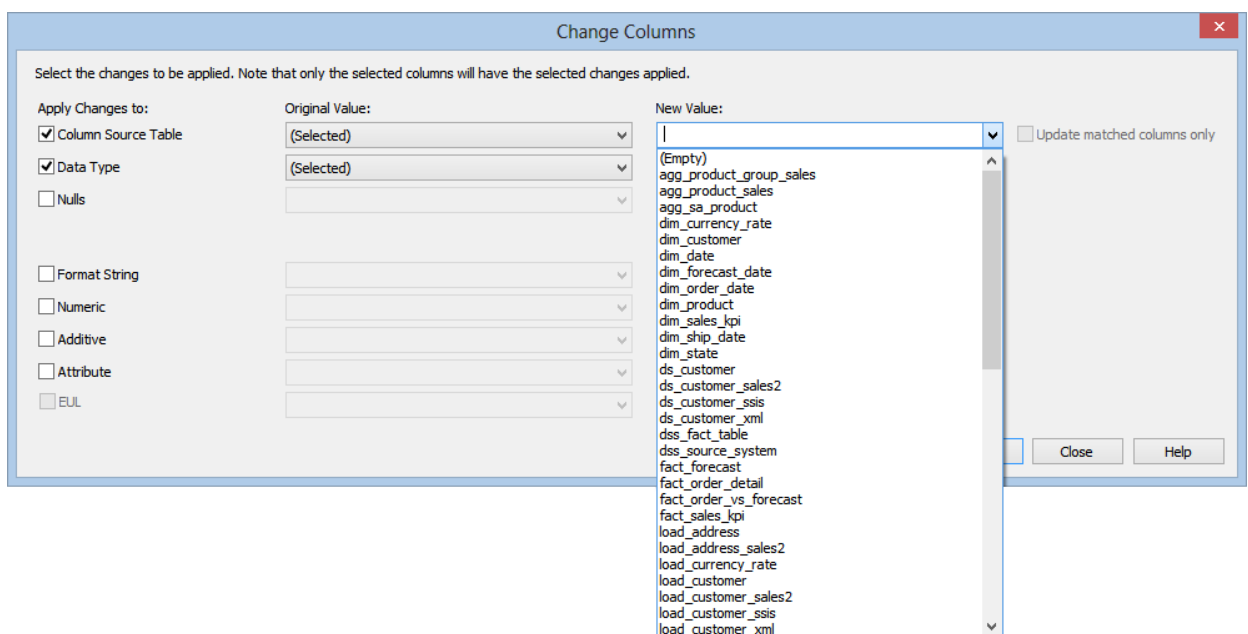
In the **Original Value** column, select the value/s to be changed.



- 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.

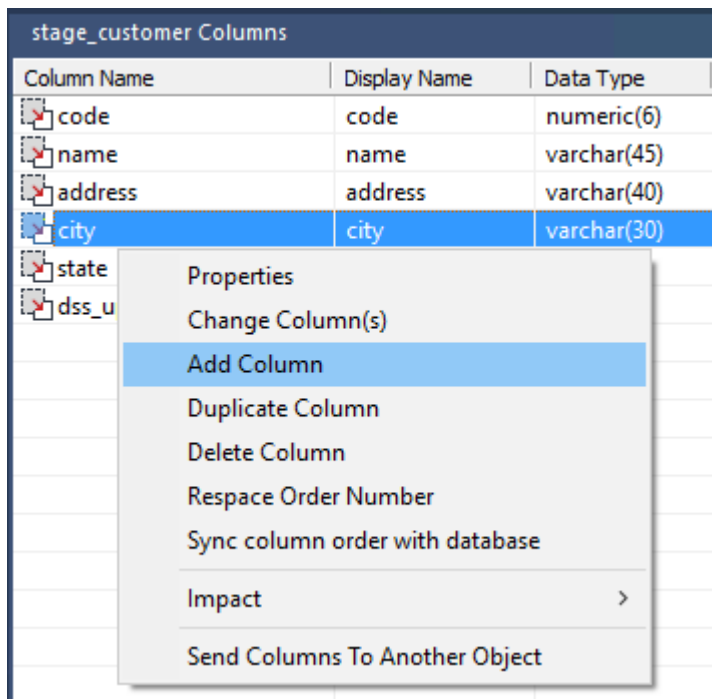
In the **New Value** column, select the new value to be assigned; or key in the new value.



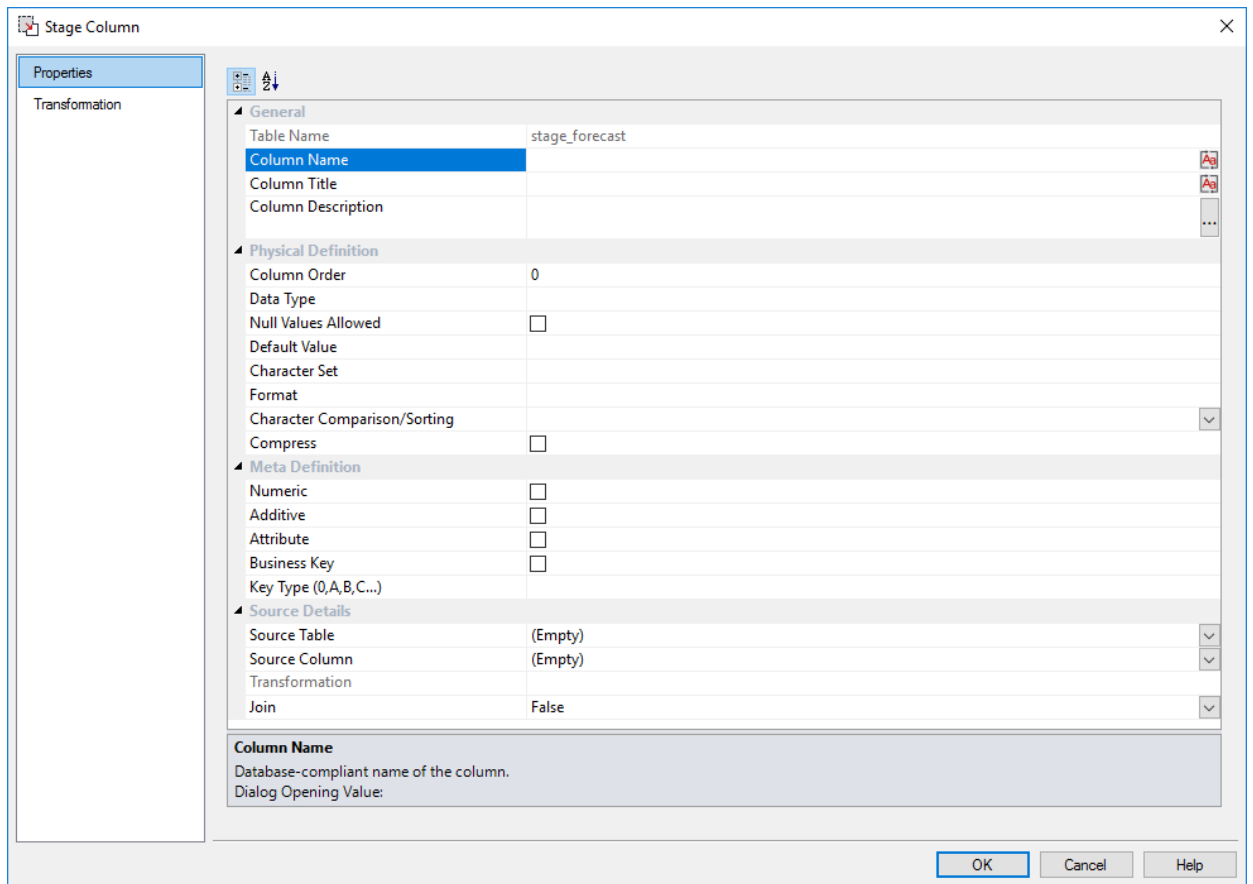
Note: When editing the property **Column Source Table**, selecting the **Update matched columns only** check-box will validate that the selected column name exists in the new source table. If it does not exist, then the update will not take place.

ADD COLUMN

To add a column, right-click on one of the columns in the middle pane and select **Add Column**.

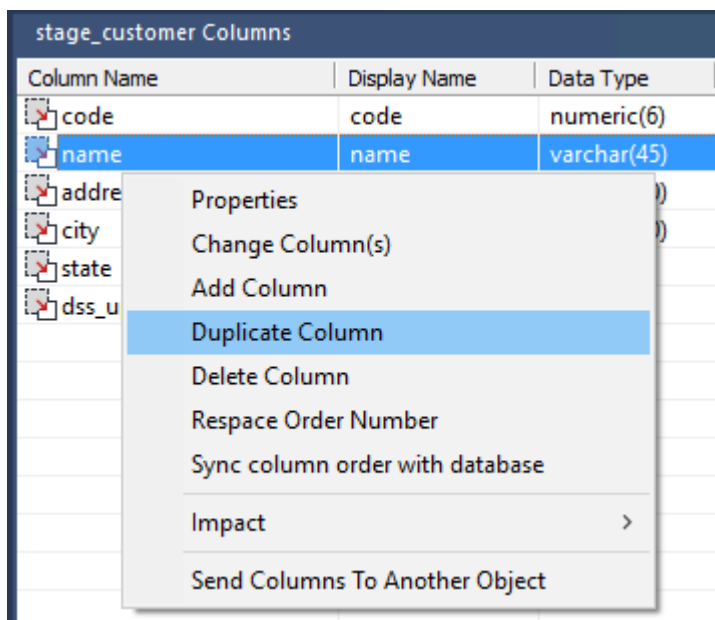


Enter the details to define a new column and click **OK**.



DUPLICATE COLUMN

To duplicate/copy a column, right-click on one of the columns in the middle pane and select **Duplicate Column**.

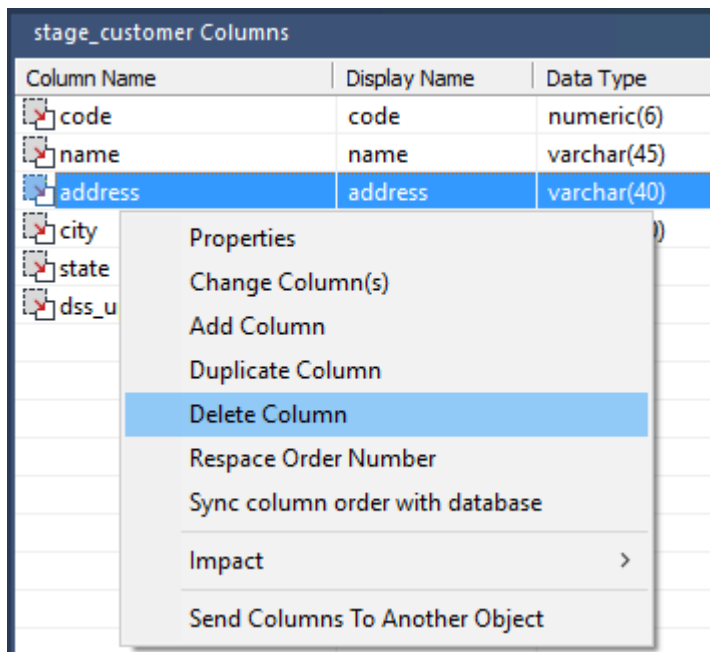


Change the **Column Name** and the **Column Title** and any other properties to define a new column and click **OK**.

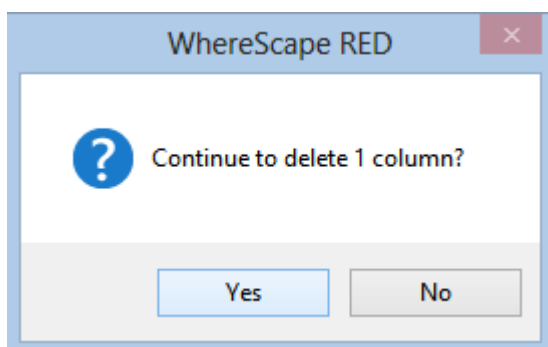
| Section | Property | Value |
|---------------------|------------------------------|---|
| General | Table Name | stage_customer |
| | Column Name | customer_category code |
| | Column Title | customer_category code |
| | Column Description | The full name of the customer. Forms a hierarchy with city and state. |
| Physical Definition | Column Order | 20 |
| | Data Type | varchar(45) |
| | Null Values Allowed | <input checked="" type="checkbox"/> |
| | Default Value | |
| | Character Set | |
| | Format | |
| | Character Comparison/Sorting | |
| Meta Definition | Compress | <input type="checkbox"/> |
| | Numeric | <input type="checkbox"/> |
| | Additive | <input checked="" type="checkbox"/> |
| | Attribute | <input checked="" type="checkbox"/> |
| | Business Key | <input type="checkbox"/> |
| Source Details | Key Type (0,A,B,C,...) | |
| | Source Table | load_customer |
| | Source Column | name |
| | Transformation | |
| Column Name | Join | False |
| | Column Name | Database-compliant name of the column. Dialog Opening Value: name |

DELETE COLUMN

To delete a column, right-click on one of the columns in the middle pane and select **Delete Column**.

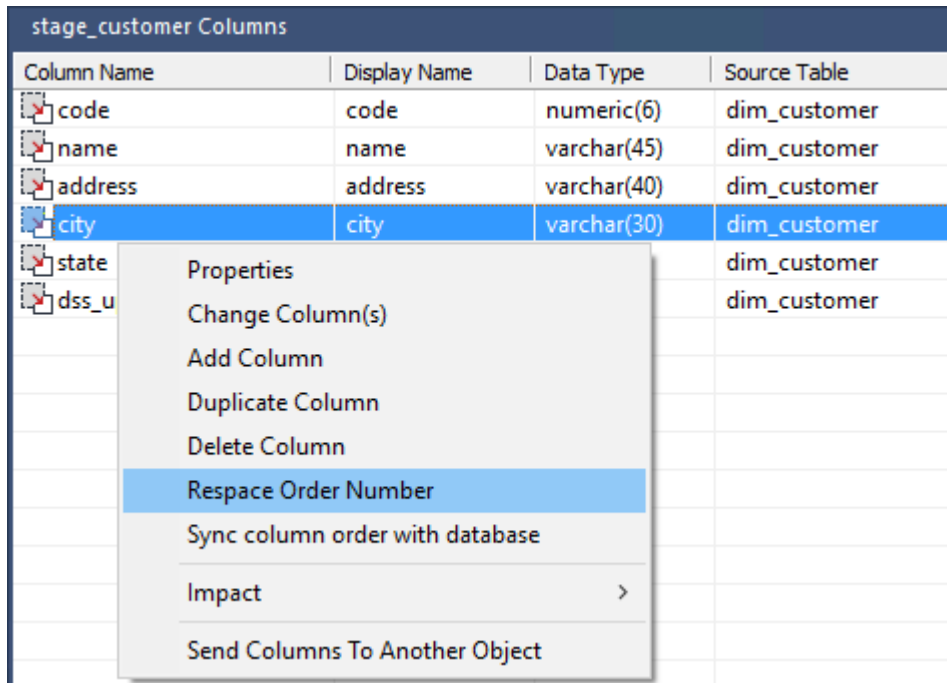


Click **Yes** to continue with the delete.



RE-SPACE ORDER NUMBER

To re-space the column order, right-click on any column in the middle pane and select **Respace Order Number**.



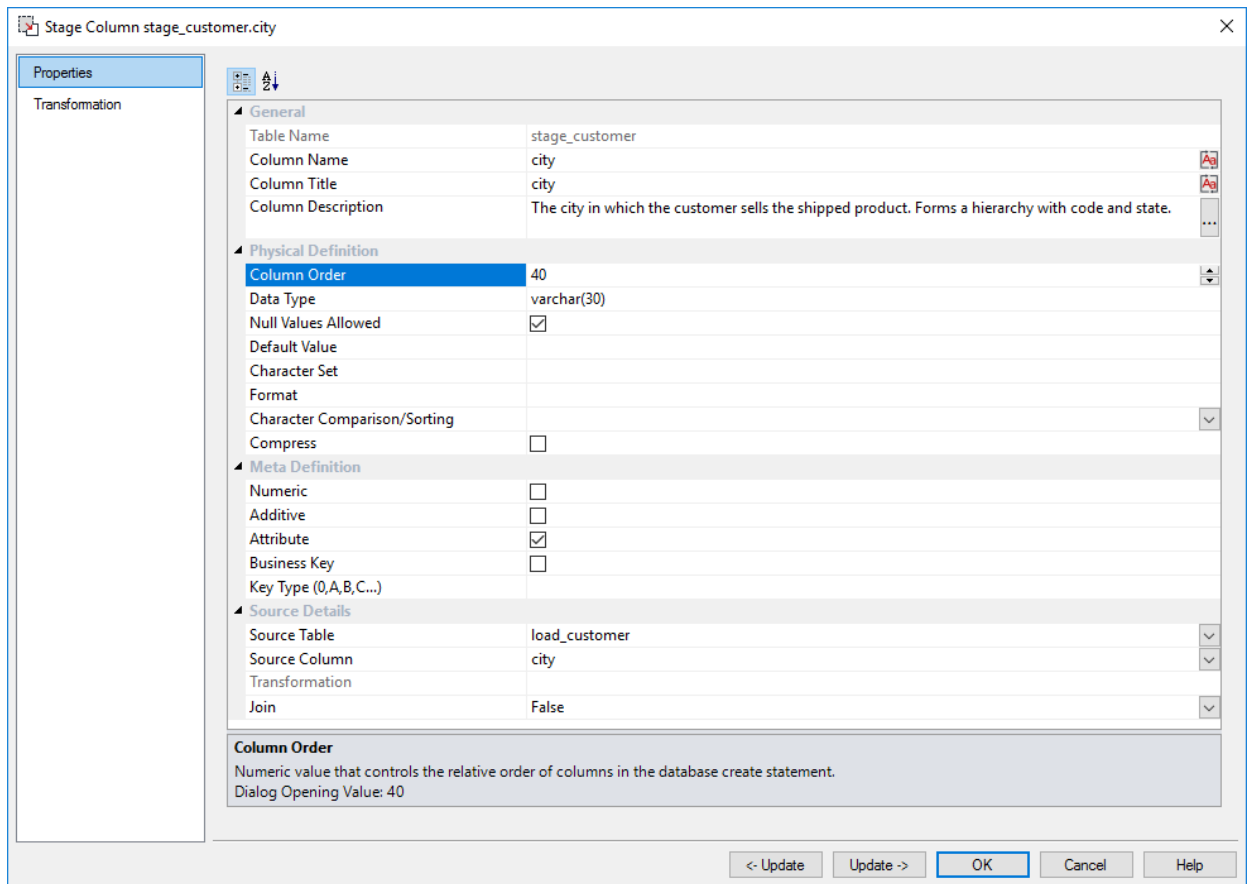
The screenshot shows a table titled "stage_customer Columns" with the following data:

| Column Name | Display Name | Data Type | Source Table |
|-------------|--------------|-------------|--------------|
| code | code | numeric(6) | dim_customer |
| name | name | varchar(45) | dim_customer |
| address | address | varchar(40) | dim_customer |
| city | city | varchar(30) | dim_customer |
| state | | | dim_customer |
| dss_u | | | dim_customer |

A context menu is open over the 'city' column, with the following options:

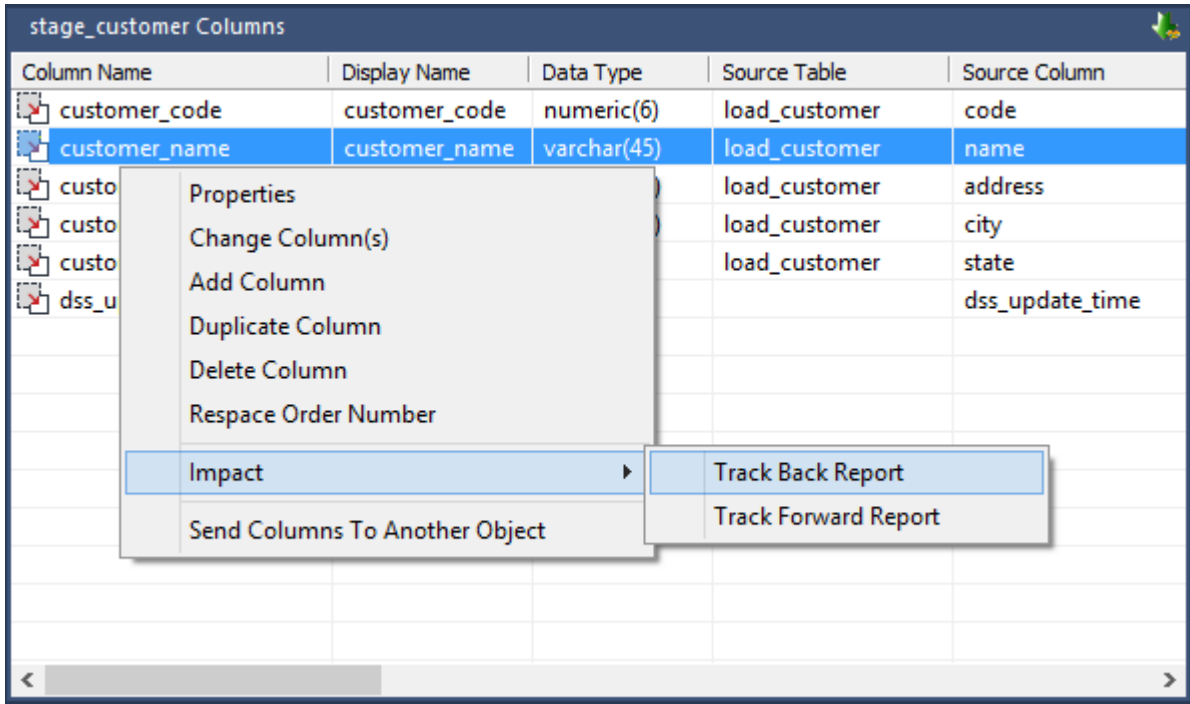
- Properties
- Change Column(s)
- Add Column
- Duplicate Column
- Delete Column
- Respace Order Number
- Sync column order with database
- Impact >
- Send Columns To Another Object

The **Column Order** number for each column will be adjusted so that the column order numbers are evenly spaced.

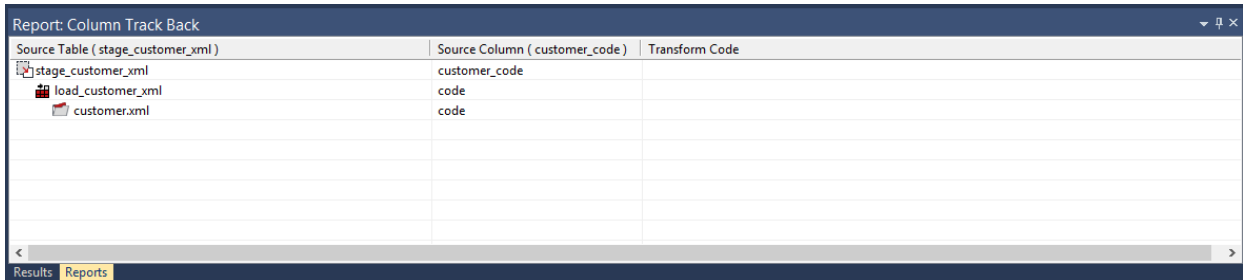


IMPACT

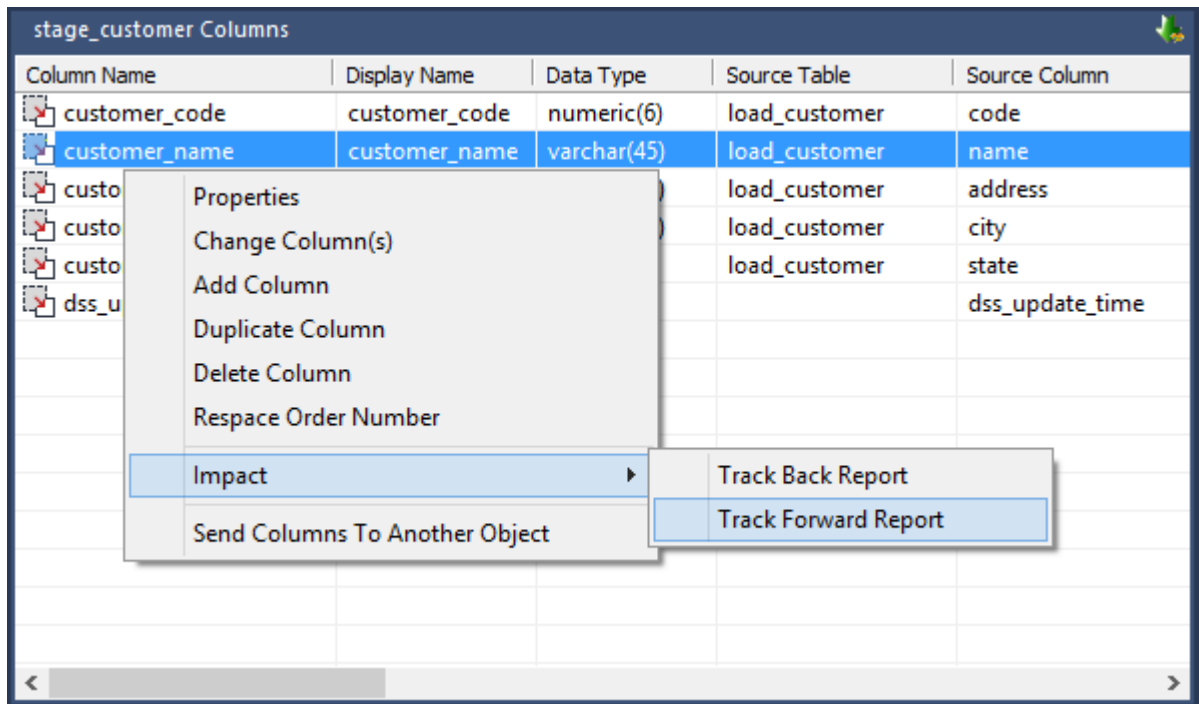
To display a Track Back Report, right-click on a column in the middle pane and select **Impact > Track Back Report**.



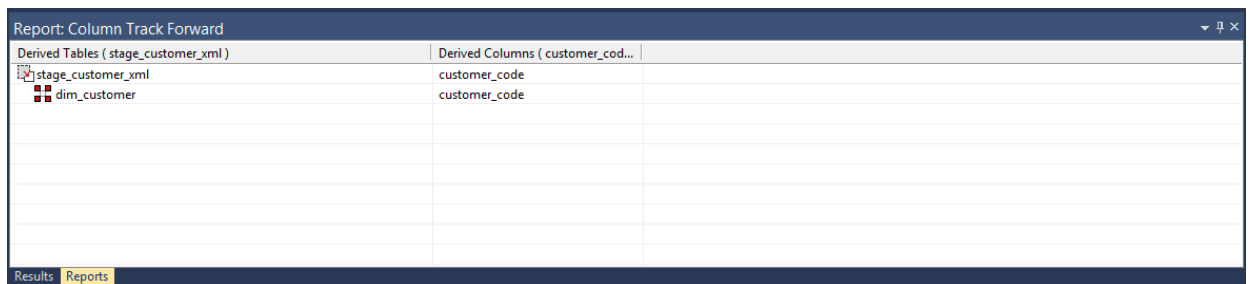
The report will be displayed in the bottom middle pane. This report lists the origins of the selected column. See **Track Back Report** (see "**Column Track-Back**" on page 805).



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 807).

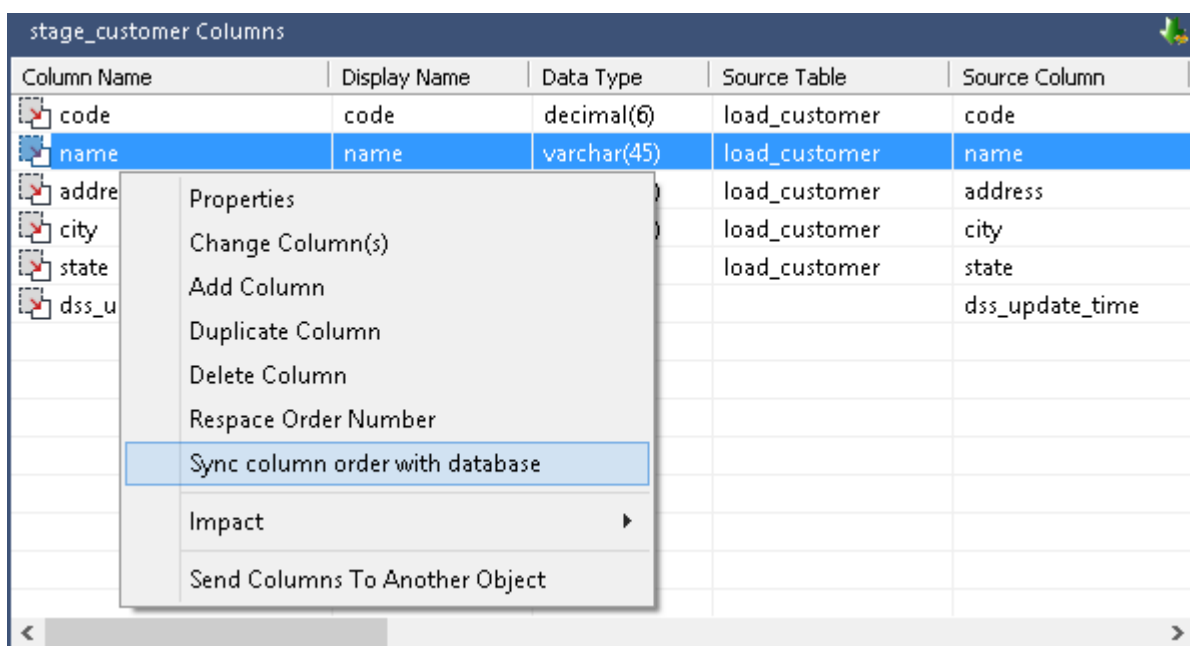


SYNC COLUMN ORDER WITH DATABASE

To synchronize the metadata's column order to match the same order in the physical table in the database:

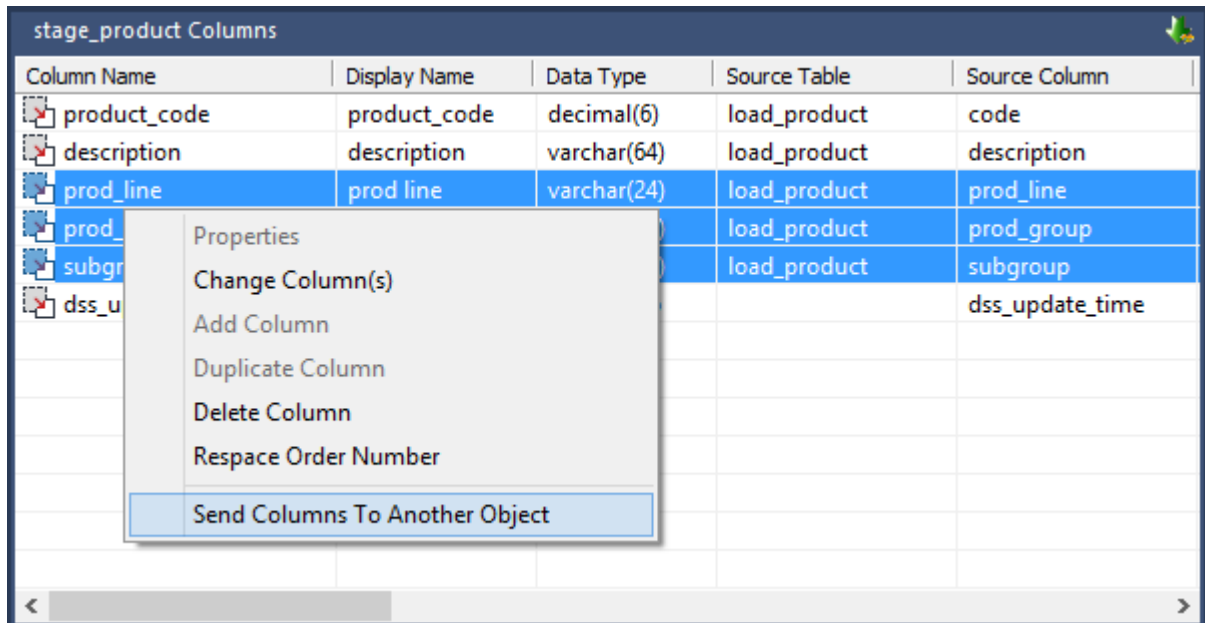
- Right-click on one of the columns in the middle pane and select **Sync the column order with the database**.

This will reorder the metadata columns to match the column order in the database table.



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**.

stage_sales_detail Columns

| Column Name | Display Name | Data Type | Source Table | Source Column |
|--------------------|-------------------|-------------|-------------------|--------------------|
| dim_customer_key | dim customer ... | integer | dim_customer | dim_customer_key |
| dim_product_key | dim product key | integer | dim_product | dim_product_key |
| dim_order_date_key | dim order date... | integer | dim_order_date | dim_order_date_key |
| dim_ship_date_key | dim ship date ... | integer | dim_ship_date | dim_ship_date_key |
| order_number | order number | decimal(12) | load_order_header | order_number |
| order_date | order date | date | load_order_header | order_date |
| customer_code | customer code | decimal(6) | load_order_header | customer_code |
| ship_date | ship date | date | load_order_header | ship_date |
| order_line_no | order line no | decimal(4) | load_order_line | order_line_no |
| product_code | | | load_order_line | product_code |
| unit_sale_price | | | load_order_line | unit_sale_price |
| quantity | | | load_order_line | quantity |
| sales_value | | | load_order_line | sales_value |
| tax | | | load_order_line | tax |
| dss_update_time | | | | dss_update_time |

Context menu for 'order_line_no':

- Properties
- Change Column(s)
- Add Column
- Duplicate Column
- Delete Column
- Respace Order Number
- Impact
- Send Columns To Another Object
- 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_sales_detail Columns | | | | |
|----------------------------|-------------------|---------------|-------------------|--------------------|
| Column Name | Display Name | Data Type | Source Table | Source Column |
| dim_customer_key | dim customer ... | integer | dim_customer | dim_customer_key |
| dim_product_key | dim product key | integer | dim_product | dim_product_key |
| dim_order_date_key | dim order date... | integer | dim_order_date | dim_order_date_key |
| dim_ship_date_key | dim ship date ... | integer | dim_ship_date | dim_ship_date_key |
| order_number | order number | decimal(12) | load_order_header | order_number |
| order_date | order date | date | load_order_header | order_date |
| customer_code | customer code | decimal(6) | load_order_header | customer_code |
| prod_line | prod line | varchar(24) | stage_product | prod_line |
| prod_group | prod group | varchar(24) | stage_product | prod_group |
| subgroup | subgroup | varchar(24) | stage_product | subgroup |
| ship_date | ship date | date | load_order_header | ship_date |
| order_line_no | order line no | decimal(4) | load_order_line | order_line_no |
| product_code | product code | decimal(6) | load_order_line | product_code |
| unit_sale_price | unit sale price | decimal(9,3) | load_order_line | unit_sale_price |
| quantity | quantity | decimal(8) | load_order_line | quantity |
| sales_value | sales value | decimal(13,2) | load_order_line | sales_value |
| tax | tax | decimal(9,2) | load_order_line | tax |
| dss_update_time | dss update time | timestamp | | dss_update_time |

CHAPTER 43 DATABASE FUNCTIONS

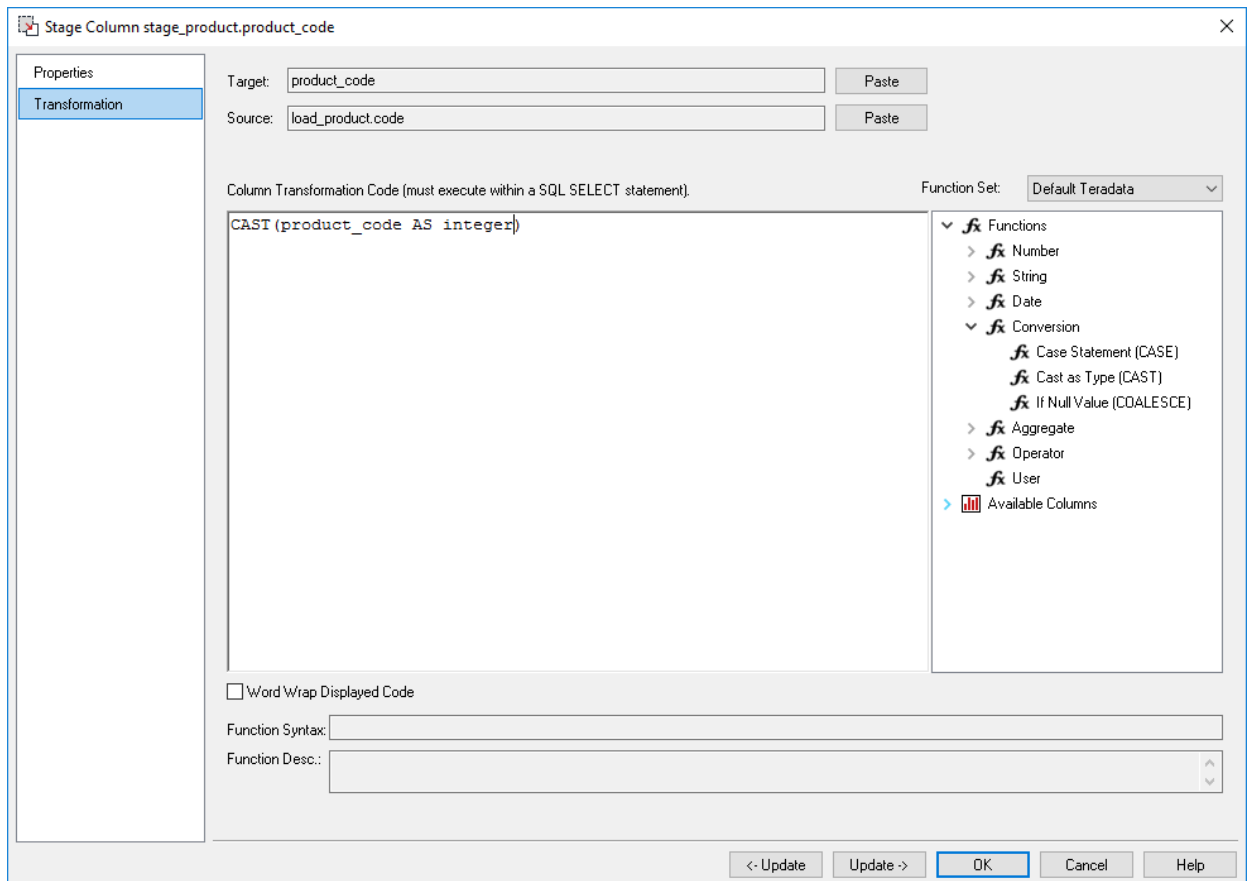
IN THIS CHAPTER

| | |
|--|------|
| Using Database Function Sets | 1077 |
| Maintaining Database Function Sets | 1080 |
| Loading Database Function Sets | 1103 |
| Exporting Database Function Sets | 1106 |

USING DATABASE FUNCTION SETS

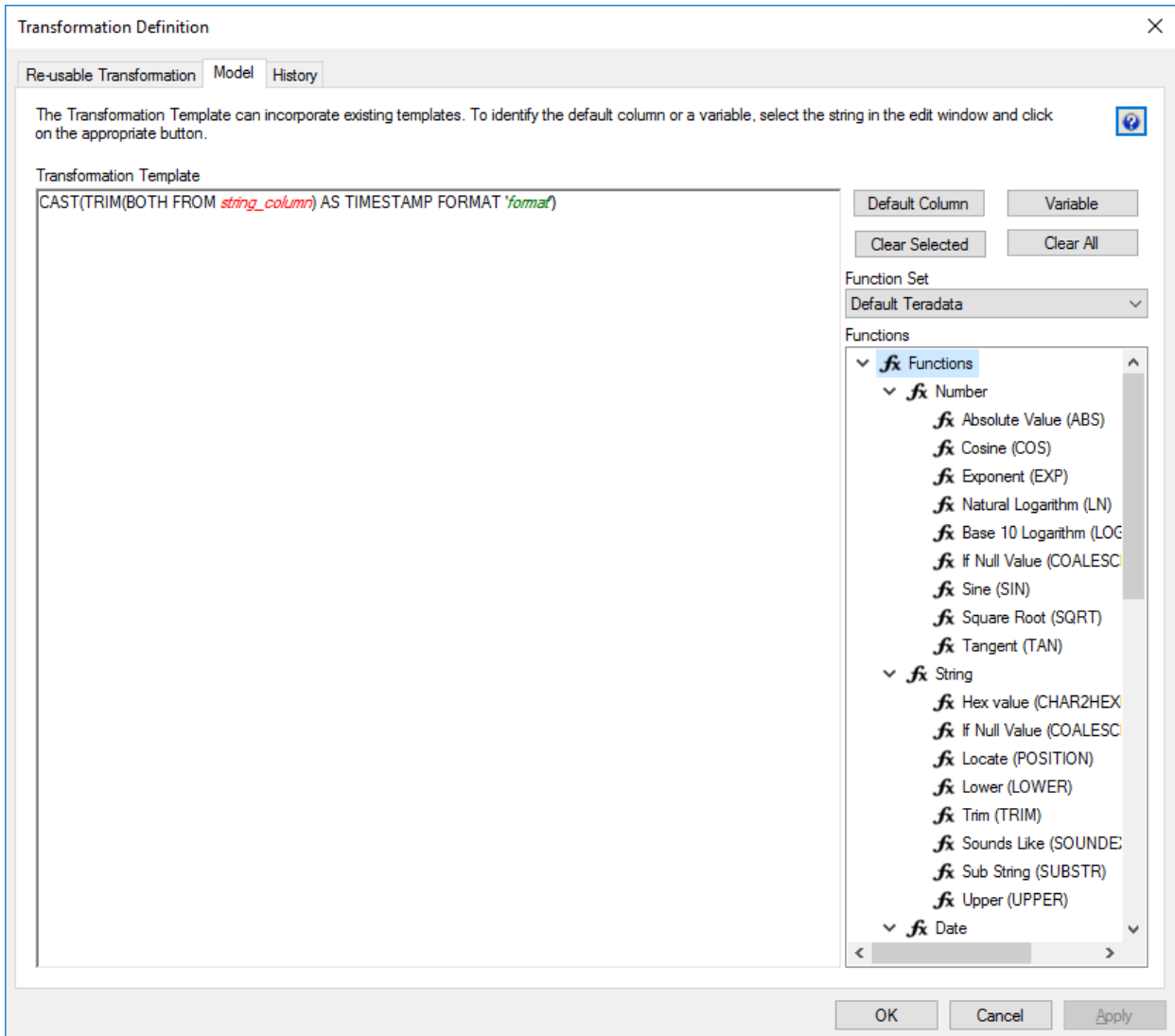
Database function sets contain a list of functions and operators that can be used for building transformations. These function sets may be created, edited, deleted, imported and exported using the **Database Functions** options on the Tools menu.

Column Transformation Properties Dialog



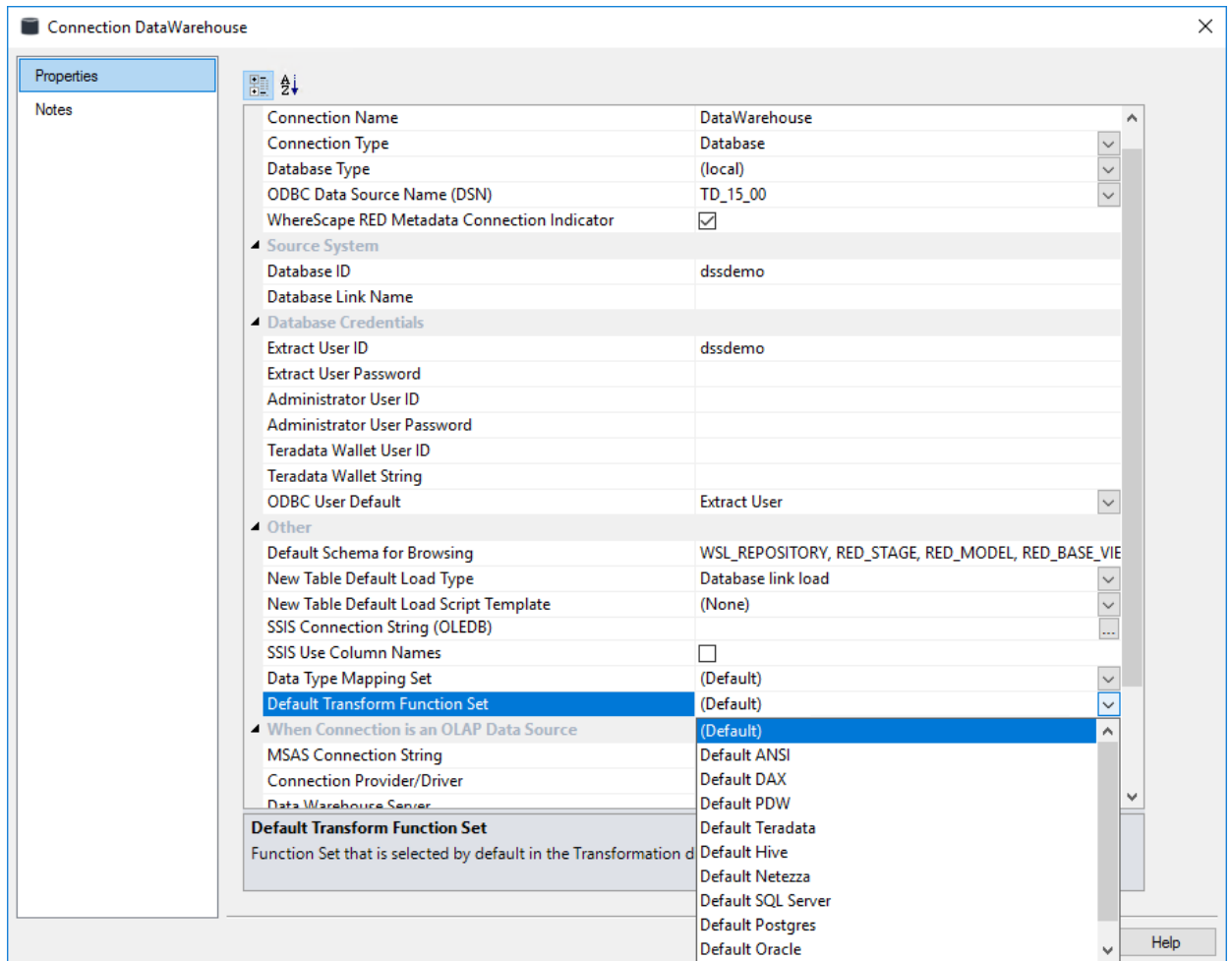
A drop-down list allows the user to select which set of functions are to be displayed in the tree view when creating a transformation on a column of a table.

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...**

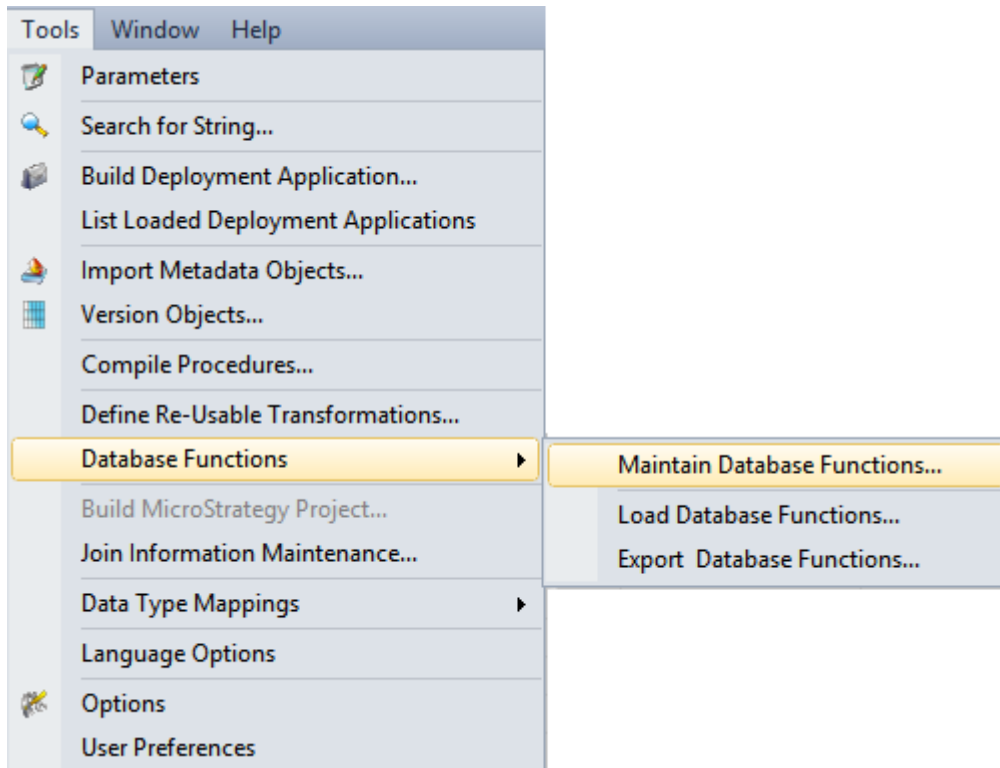
Connection Properties Dialog



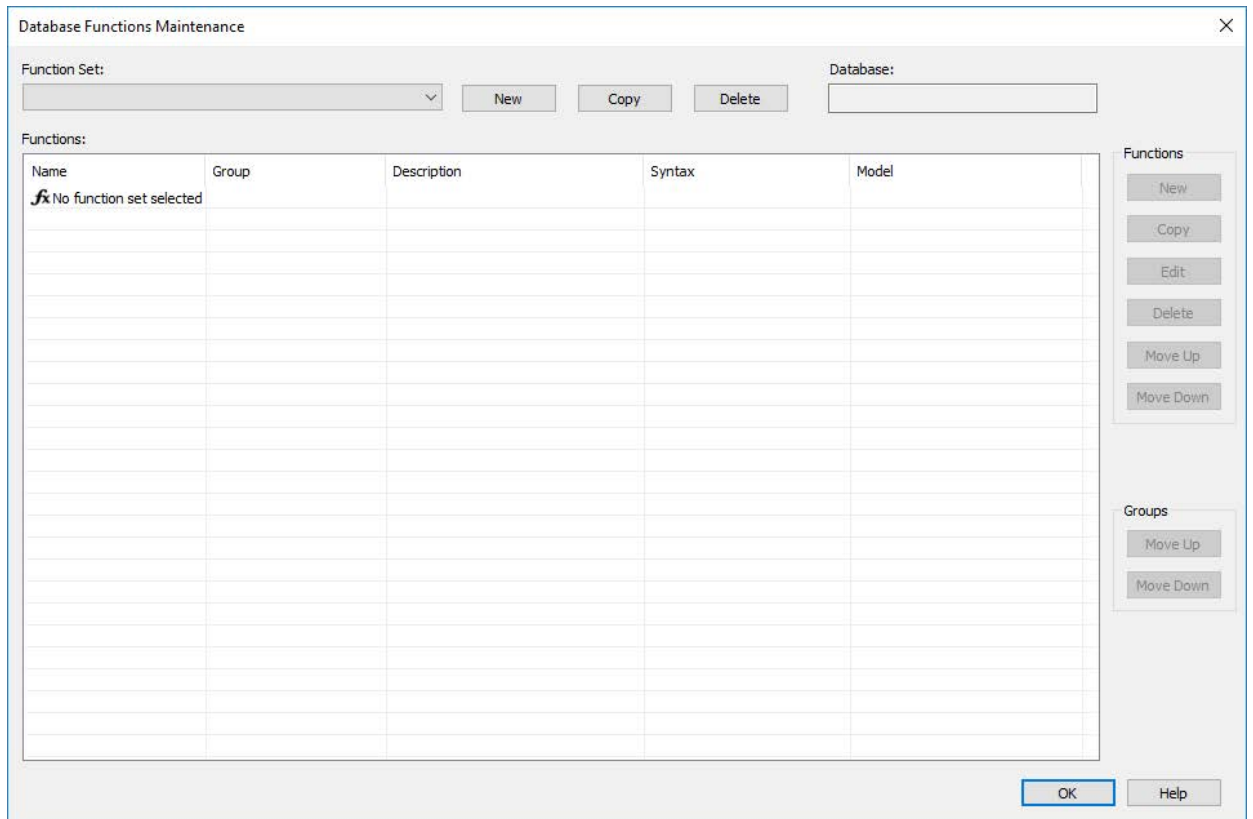
In the Connection Properties dialog, when the **Data Warehouse** check-box is selected, a drop-down list **Default Transform Function Set** is displayed. This is the default set that will be selected in the transformation dialogs above.

MAINTAINING DATABASE FUNCTION SETS

To maintain the database function sets, select **Tools/Database Functions/Maintain Database Functions...**



The following dialog is displayed.



Select a database function set from the **Function Set** drop-down list.

Database Functions Maintenance

Function Set: Default Teradata New Copy Delete Database: Teradata

Functions:

| Name | Group | Description | Syntax | Model |
|--------------------------|------------|--|--------------------------------|-------------------------------------|
| Absolute Value (ABS) | Number | Returns the absolute value of the n... | ABS(numeric_column) | ABS(numeric_column) |
| Cosine (COS) | Number | Returns the trigonometric cosine for ... | COS(numeric_column) | COS(numeric_column) |
| Exponent (EXP) | Number | Returns e raised to the specified po... | EXP(n) | EXP(n) |
| Natural Logarithm (LN) | Number | Returns the natural, or base 'e' loga... | LN(n) | LN(n) |
| Base 10 Logarithm (L... | Number | Returns the base10 logarithm of the... | LOG(number) | LOG(number) |
| If Null Value (COALES... | Number | Returns either the passed column or... | COALESCE(numeric_column,... | COALESCE(numeric_column,def... |
| Sine (SIN) | Number | Returns the trigonometric sine for a... | SIN(numeric_column) | SIN(numeric_column) |
| Square Root (SQRT) | Number | Returns the square root of the valu... | SQRT(numeric_column) | SQRT(numeric_column) |
| Tangent (TAN) | Number | Returns the trigonometric tangent f... | TAN(numeric_column) | TAN(numeric_column) |
| Hex value (CHAR2HE... | String | Returns the hexadecimal value of th... | CHAR2HEXINT(string_column) | CHAR2HEXINT(string_column) |
| If Null Value (COALES... | String | Returns either the passed column or... | COALESCE(string_column,d... | COALESCE(string_column,default... |
| Locate (POSITION) | String | Returns an integer value containing ... | POSITION('search_string' IN... | POSITION('search_string' IN stri... |
| Lower (LOWER) | String | Returns a string where every charac... | LOWER(string_column) | LOWER(string_column) |
| Trim (TRIM) | String | Returns a string which is the passed ... | TRIM(BOTH/TRAILING/LEAD... | TRIM(BOTH/TRAILING/LEADING... |
| Sounds Like (SOUNDEX) | String | Used in a comparison situation. This ... | SOUNDEX(string_column) | SOUNDEX(string_column) |
| Sub String (SUBSTR) | String | Returns a sub string of the passed c... | SUBSTR(string_column, star... | SUBSTR(string_column, start ,co... |
| Upper (UPPER) | String | Returns a string where every charac... | UPPER(string_column) | UPPER(string_column) |
| Add Date parts (INTE... | Date | Returns a date which has a number ... | INTERVAL 'number' date_part | INTERVAL 'number' date_part |
| Add Months (ADD_M... | Date | Returns a date which has had the sp... | ADD_MONTHS(date_column... | ADD_MONTHS(date_column,nu... |
| Date Conversion (CA... | Date | Performs a date conversion or calcul... | CAST(date AS FORMAT ") | CAST(date AS FORMAT ") |
| Extract (EXTRACT) | Date | Returns the integer value of the dat... | EXTRACT(YEAR/MONTH/DA... | EXTRACT(YEAR/MONTH/DAY/H... |
| System Date (CURRE... | Date | Returns the current system date tim... | CURRENT_TIMESTAMP | CURRENT_TIMESTAMP |
| Case Statement (CASE) | Conversion | Allows the use of IF ... THEN ... ELS... | CASE [input_expression] W... | CASE [input_expression] WHEN ... |
| Cast as Type (CAST) | Conversion | Returns the column converted to th... | CAST(column AS data_type) | CAST(column AS data_type) |
| If Null Value (COALES... | Conversion | Returns either the passed column or... | COALESCE(column,default_... | COALESCE(column,default_value) |

Functions: New Copy Edit Delete Move Up Move Down

Groups: Move Up Move Down

Please note that this is a standard function set and may not be edited or deleted. OK Help

To create a database function set, see *Creating a New Database Function Set* (on page 1083)

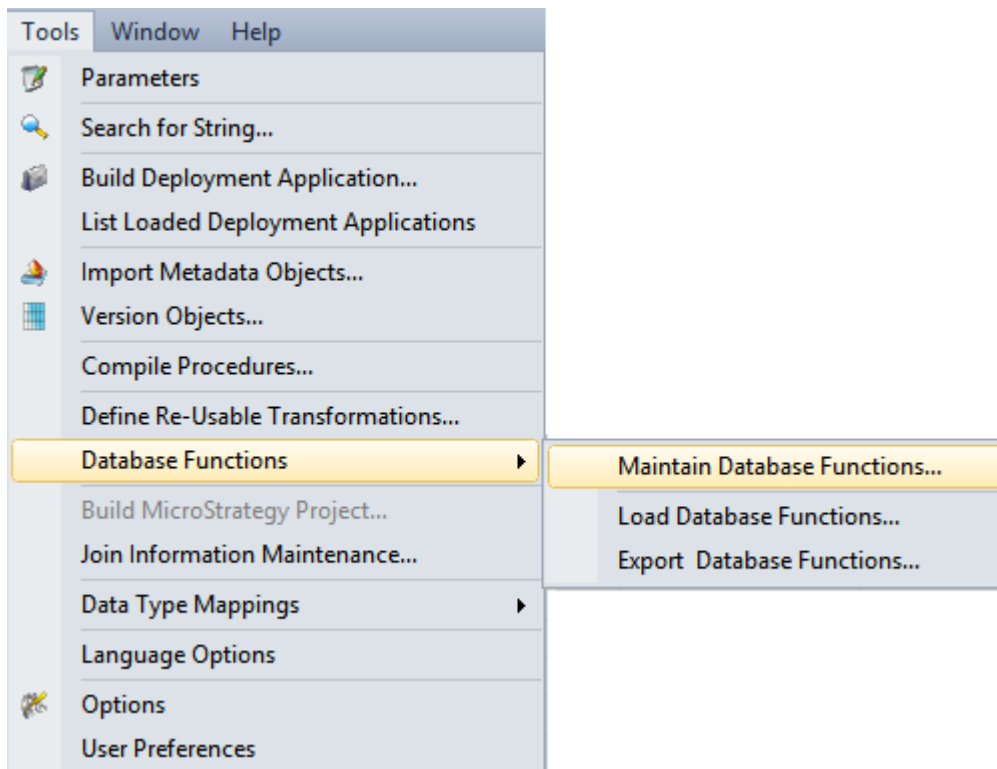
To copy a database function set, see *Copying a Database Function Set* (on page 1086)

To edit a database function set, see *Editing a Database Function Set* (on page 1089)

To delete a database function set, see *Deleting a Database Function Set* (on page 1101)

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.

Database Functions Maintenance

Function Set: Database:

| Name | Group | Description | Syntax | Model |
|--------------------------|------------|--|--------------------------------|-------------------------------------|
| Absolute Value (ABS) | Number | Returns the absolute value of the n... | ABS(numeric_column) | ABS(numeric_column) |
| Cosine (COS) | Number | Returns the trigonometric cosine for ... | COS(numeric_column) | COS(numeric_column) |
| Exponent (EXP) | Number | Returns e raised to the specified po... | EXP(n) | EXP(n) |
| Natural Logarithm (LN) | Number | Returns the natural, or base 'e' loga... | LN(n) | LN(n) |
| Base 10 Logarithm (L... | Number | Returns the base10 logarithm of the... | LOG(number) | LOG(number) |
| If Null Value (COALES... | Number | Returns either the passed column or... | COALESCE(numeric_column,... | COALESCE(numeric_column,def... |
| Sine (SIN) | Number | Returns the trigonometric sine for a... | SIN(numeric_column) | SIN(numeric_column) |
| Square Root (SQRT) | Number | Returns the square root of the valu... | SQRT(numeric_column) | SQRT(numeric_column) |
| Tangent (TAN) | Number | Returns the trigonometric tangent f... | TAN(numeric_column) | TAN(numeric_column) |
| Hex value (CHAR2HE... | String | Returns the hexadecimal value of th... | CHAR2HEXINT(string_column) | CHAR2HEXINT(string_column) |
| If Null Value (COALES... | String | Returns either the passed column or... | COALESCE(string_column,d... | COALESCE(string_column,default... |
| Locate (POSITION) | String | Returns an integer value containing ... | POSITION('search_string' IN... | POSITION('search_string' IN stri... |
| Lower (LOWER) | String | Returns a string where every charac... | LOWER(string_column) | LOWER(string_column) |
| Trim (TRIM) | String | Returns a string which is the passed ... | TRIM(BOTH/TRAILING/LEAD... | TRIM(BOTH/TRAILING/LEADING... |
| Sounds Like (SOUNDEX) | String | Used in a comparison situation. This ... | SOUNDEX(string_column) | SOUNDEX(string_column) |
| Sub String (SUBSTR) | String | Returns a sub string of the passed c... | SUBSTR(string_column, star... | SUBSTR(string_column, start ,co... |
| Upper (UPPER) | String | Returns a string where every charac... | UPPER(string_column) | UPPER(string_column) |
| Add Date parts (INTE... | Date | Returns a date which has a number ... | INTERVAL 'number' date_part | INTERVAL 'number' date_part |
| Add Months (ADD_M... | Date | Returns a date which has had the sp... | ADD_MONTHS(date_column... | ADD_MONTHS(date_column,nu... |
| Date Conversion (CA... | Date | Performs a date conversion or calcul... | CAST(date AS FORMAT ") | CAST(date AS FORMAT ") |
| Extract (EXTRACT) | Date | Returns the integer value of the dat... | EXTRACT(YEAR/MONTH/DA... | EXTRACT(YEAR/MONTH/DAY/H... |
| System Date (CURRE... | Date | Returns the current system date tim... | CURRENT_TIMESTAMP | CURRENT_TIMESTAMP |
| Case Statement (CASE) | Conversion | Allows the use of IF ... THEN ... ELS... | CASE [input_expression] W... | CASE [input_expression] WHEN ... |
| Cast as Type (CAST) | Conversion | Returns the column converted to th... | CAST(column AS data_type) | CAST(column AS data_type) |
| If Null Value (COALES... | Conversion | Returns either the passed column or... | COALESCE(column,default_... | COALESCE(column,default_value) |

Please note that this is a standard function set and may not be edited or deleted.

Enter a **Function Set Name** and select a **Database** from the drop-down list. Click **OK**.

New Database Function Set

Function Set Name:

Database:

Please note that the new function set will only be saved once a function has been added to the set.

Function Set Name

Enter a unique function set name.

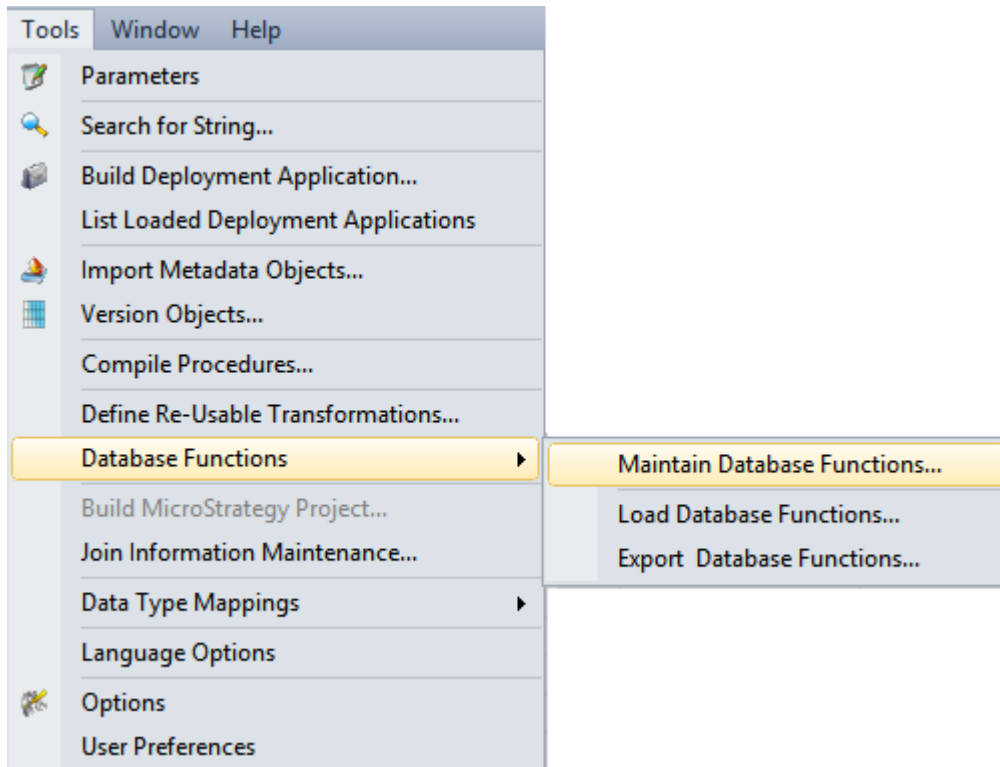
Note: The function set name may not contain the phrase "Standard Functions" as this is reserved for WhereScape supplied function sets.

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.

Database Functions Maintenance

Function Set: Database:

| Name | Group | Description | Syntax | Model |
|--------------------------|------------|--|--------------------------------|-------------------------------------|
| Absolute Value (ABS) | Number | Returns the absolute value of the n... | ABS(numeric_column) | ABS(numeric_column) |
| Cosine (COS) | Number | Returns the trigonometric cosine for ... | COS(numeric_column) | COS(numeric_column) |
| Exponent (EXP) | Number | Returns e raised to the specified po... | EXP(n) | EXP(n) |
| Natural Logarithm (LN) | Number | Returns the natural, or base 'e' loga... | LN(n) | LN(n) |
| Base 10 Logarithm (L... | Number | Returns the base10 logarithm of the... | LOG(number) | LOG(number) |
| If Null Value (COALES... | Number | Returns either the passed column or ... | COALESCE(numeric_column,... | COALESCE(numeric_column,def... |
| Sine (SIN) | Number | Returns the trigonometric sine for a... | SIN(numeric_column) | SIN(numeric_column) |
| Square Root (SQRT) | Number | Returns the square root of the valu... | SQRT(numeric_column) | SQRT(numeric_column) |
| Tangent (TAN) | Number | Returns the trigonometric tangent f... | TAN(numeric_column) | TAN(numeric_column) |
| Hex value (CHAR2HE... | String | Returns the hexadecimal value of th... | CHAR2HEXINT(string_column) | CHAR2HEXINT(string_column) |
| If Null Value (COALES... | String | Returns either the passed column or ... | COALESCE(string_column,d... | COALESCE(string_column,defaul... |
| Locate (POSITION) | String | Returns an integer value containing ... | POSITION('search_string' IN... | POSITION('search_string' IN stri... |
| Lower (LOWER) | String | Returns a string where every charac... | LOWER(string_column) | LOWER(string_column) |
| Trim (TRIM) | String | Returns a string which is the passed ... | TRIM(BOTH/TRAILING/LEAD... | TRIM(BOTH/TRAILING/LEADING... |
| Sounds Like (SOUNDEX) | String | Used in a comparison situation. This ... | SOUNDEX(string_column) | SOUNDEX(string_column) |
| Sub String (SUBSTR) | String | Returns a sub string of the passed c... | SUBSTR(string_column, star... | SUBSTR(string_column, start ,co... |
| Upper (UPPER) | String | Returns a string where every charac... | UPPER(string_column) | UPPER(string_column) |
| Add Date parts (INTE... | Date | Returns a date which has a number ... | INTERVAL 'number' date_part | INTERVAL 'number' date_part |
| Add Months (ADD_M... | Date | Returns a date which has had the sp... | ADD_MONTHS(date_column... | ADD_MONTHS(date_column,nu... |
| Date Conversion (CA... | Date | Performs a date conversion or calcul... | CAST(date AS FORMAT ") | CAST(date AS FORMAT ") |
| Extract (EXTRACT) | Date | Returns the integer value of the dat... | EXTRACT(YEAR/MONTH/DA... | EXTRACT(YEAR/MONTH/DAY/H... |
| System Date (CURRE... | Date | Returns the current system date tim... | CURRENT_TIMESTAMP | CURRENT_TIMESTAMP |
| Case Statement (CASE) | Conversion | Allows the use of IF ... THEN ... ELS... | CASE [input_expression] W... | CASE [input_expression] WHEN ... |
| Cast as Type (CAST) | Conversion | Returns the column converted to th... | CAST(column AS data_type) | CAST(column AS data_type) |
| If Null Value (COALES... | Conversion | Returns either the passed column or ... | COALESCE(column,default_... | COALESCE(column,default_value) |

Please note that this is a standard function set and may not be edited or deleted.

Enter the new **Function Set Name** and select a **Database** from the drop-down list. Click **OK**.

New Database Function Set

Function Set Name:

Database:

Please note that the new function set will only be saved once a function has been added to the set.

Function Set Name

Enter a unique function set name.

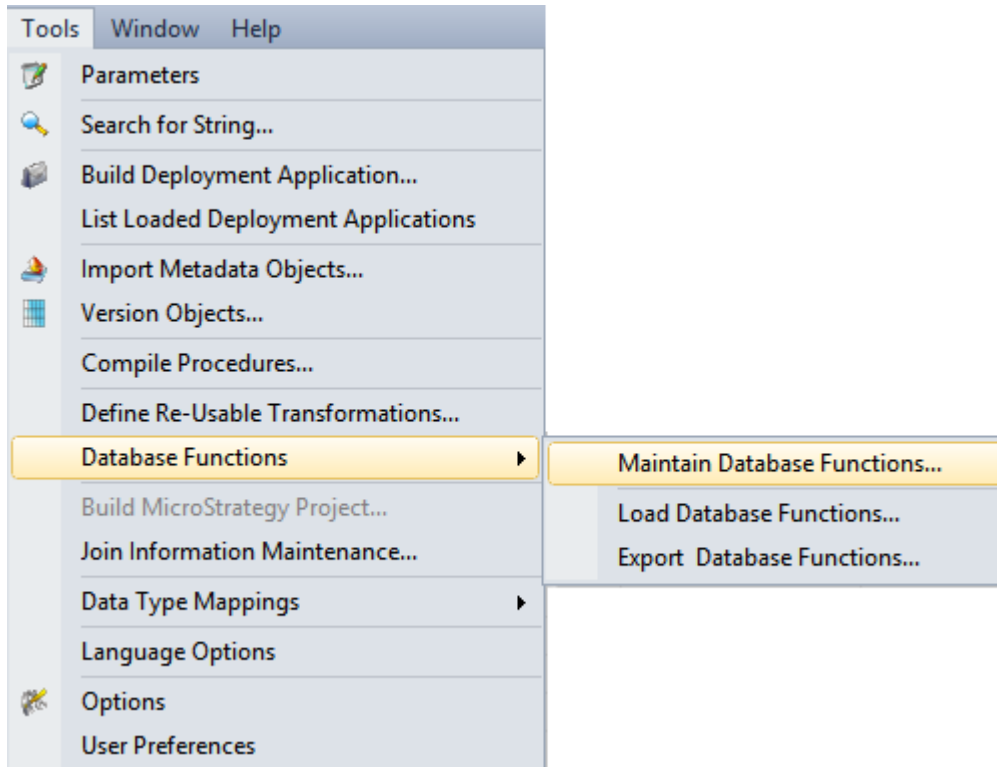
Note: The function set name may not contain the phrase "Standard Functions" as this is reserved for WhereScape supplied function sets.

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.

Database Functions Maintenance

Function Set: Teradata Custom Functions New Copy Delete Database: Teradata

Functions:

| Name | Group | Description | Syntax | Model |
|--------------------------|------------|--|--------------------------------|-------------------------------------|
| Absolute Value (ABS) | Number | Returns the absolute value of the n... | ABS(numeric_column) | ABS(numeric_column) |
| Cosine (COS) | Number | Returns the trigonometric cosine for ... | COS(numeric_column) | COS(numeric_column) |
| Exponent (EXP) | Number | Returns e raised to the specified po... | EXP(n) | EXP(n) |
| Natural Logarithm (LN) | Number | Returns the natural, or base 'e' loga... | LN(n) | LN(n) |
| Base 10 Logarithm (L... | Number | Returns the base10 logarithm of the... | LOG(number) | LOG(number) |
| If Null Value (COALES... | Number | Returns either the passed column or... | COALESCE(numeric_column,... | COALESCE(numeric_column,def... |
| Sine (SIN) | Number | Returns the trigonometric sine for a... | SIN(numeric_column) | SIN(numeric_column) |
| Square Root (SQRT) | Number | Returns the square root of the valu... | SQRT(numeric_column) | SQRT(numeric_column) |
| Tangent (TAN) | Number | Returns the trigonometric tangent f... | TAN(numeric_column) | TAN(numeric_column) |
| Hex value (CHAR2HE... | String | Returns the hexadecimal value of th... | CHAR2HEXINT(string_column) | CHAR2HEXINT(string_column) |
| If Null Value (COALES... | String | Returns either the passed column or... | COALESCE(string_column,d... | COALESCE(string_column,default... |
| Locate (POSITION) | String | Returns an integer value containing ... | POSITION('search_string' IN... | POSITION('search_string' IN stri... |
| Lower (LOWER) | String | Returns a string where every charac... | LOWER(string_column) | LOWER(string_column) |
| Trim (TRIM) | String | Returns a string which is the passed ... | TRIM(BOTH/TRAILING/LEAD... | TRIM(BOTH/TRAILING/LEADING... |
| Sounds Like (SOUNDEX) | String | Used in a comparison situation. This ... | SOUNDEX(string_column) | SOUNDEX(string_column) |
| Sub String (SUBSTR) | String | Returns a sub string of the passed c... | SUBSTR(string_column, star... | SUBSTR(string_column, start ,co... |
| Upper (UPPER) | String | Returns a string where every charac... | UPPER(string_column) | UPPER(string_column) |
| Add Date parts (INTE... | Date | Returns a date which has a number ... | INTERVAL 'number' date_part | INTERVAL 'number' date_part |
| Add Months (ADD_M... | Date | Returns a date which has had the sp... | ADD_MONTHS(date_column... | ADD_MONTHS(date_column,nu... |
| Date Conversion (CA... | Date | Performs a date conversion or calcul... | CAST(date AS FORMAT ") | CAST(date AS FORMAT ") |
| Extract (EXTRACT) | Date | Returns the integer value of the dat... | EXTRACT(YEAR/MONTH/DA... | EXTRACT(YEAR/MONTH/DAY/H... |
| System Date (CURRE... | Date | Returns the current system date tim... | CURRENT_TIMESTAMP | CURRENT_TIMESTAMP |
| Case Statement (CASE) | Conversion | Allows the use of IF ... THEN ... ELS... | CASE [input_expression] W... | CASE [input_expression] WHEN ... |
| Cast as Type (CAST) | Conversion | Returns the column converted to th... | CAST(column AS data_type) | CAST(column AS data_type) |
| If Null Value (COALES... | Conversion | Returns either the passed column or... | COALESCE(column,default_... | COALESCE(column,default_value) |

Functions: New Copy Edit Delete Move Up Move Down

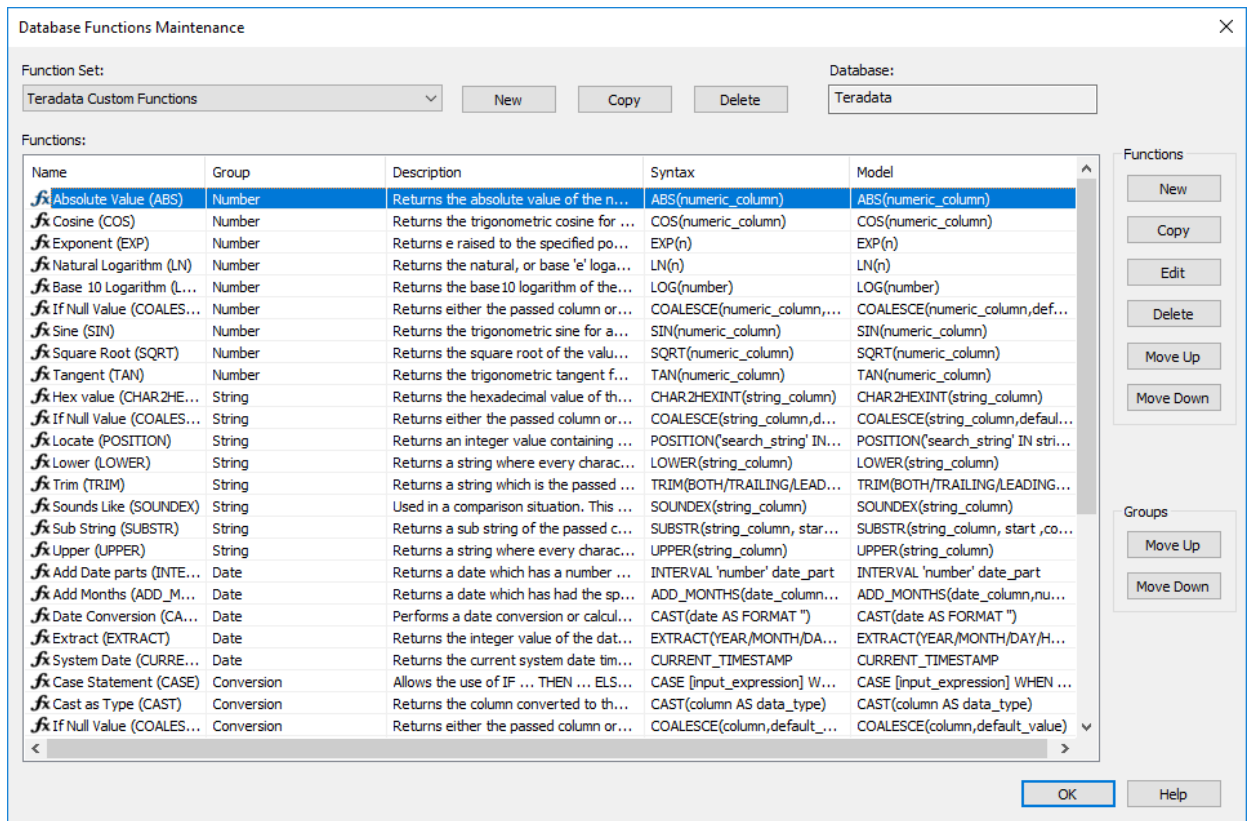
Groups: Move Up Move Down

OK Help

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**.

New Function

Function Name:

Group:

Description:

Syntax:

Model:

To set the Default Column, select it in the Model box above and click the Default Column button.

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.

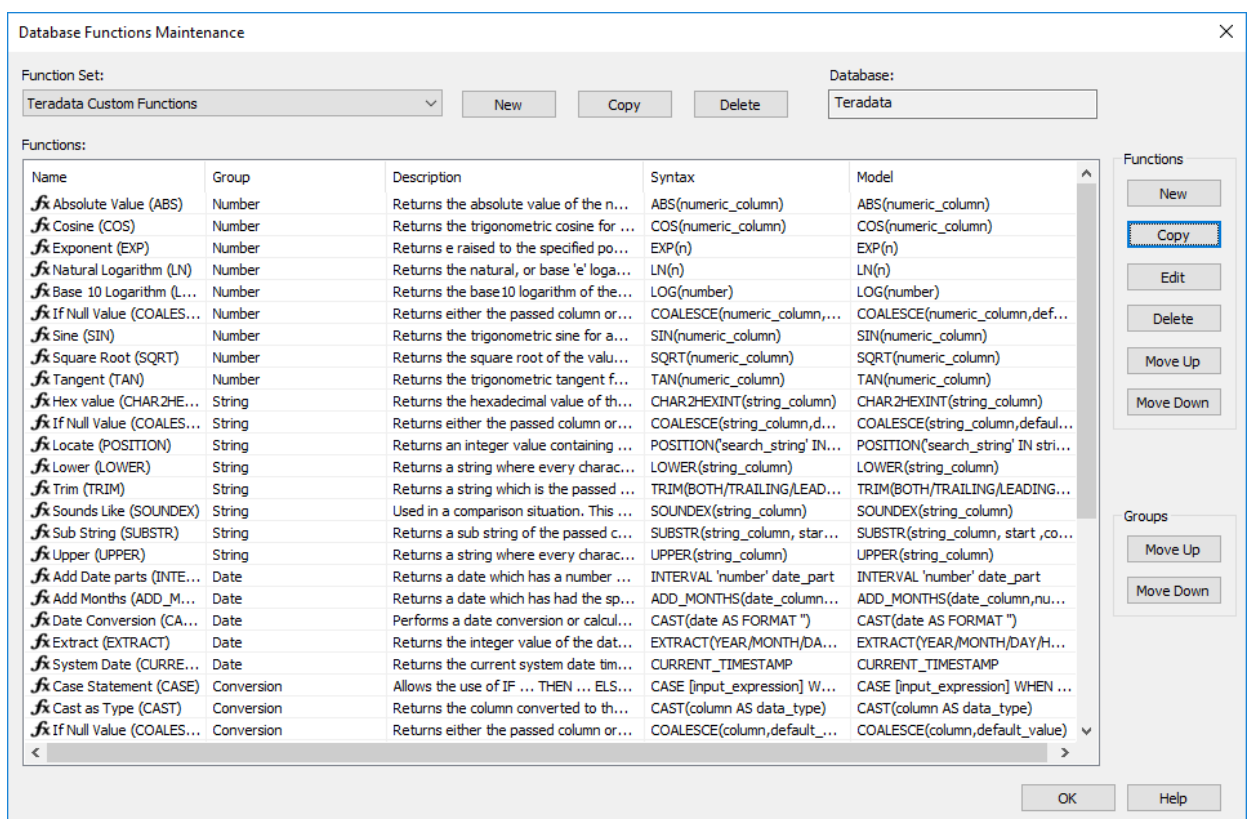


Clear Default Column

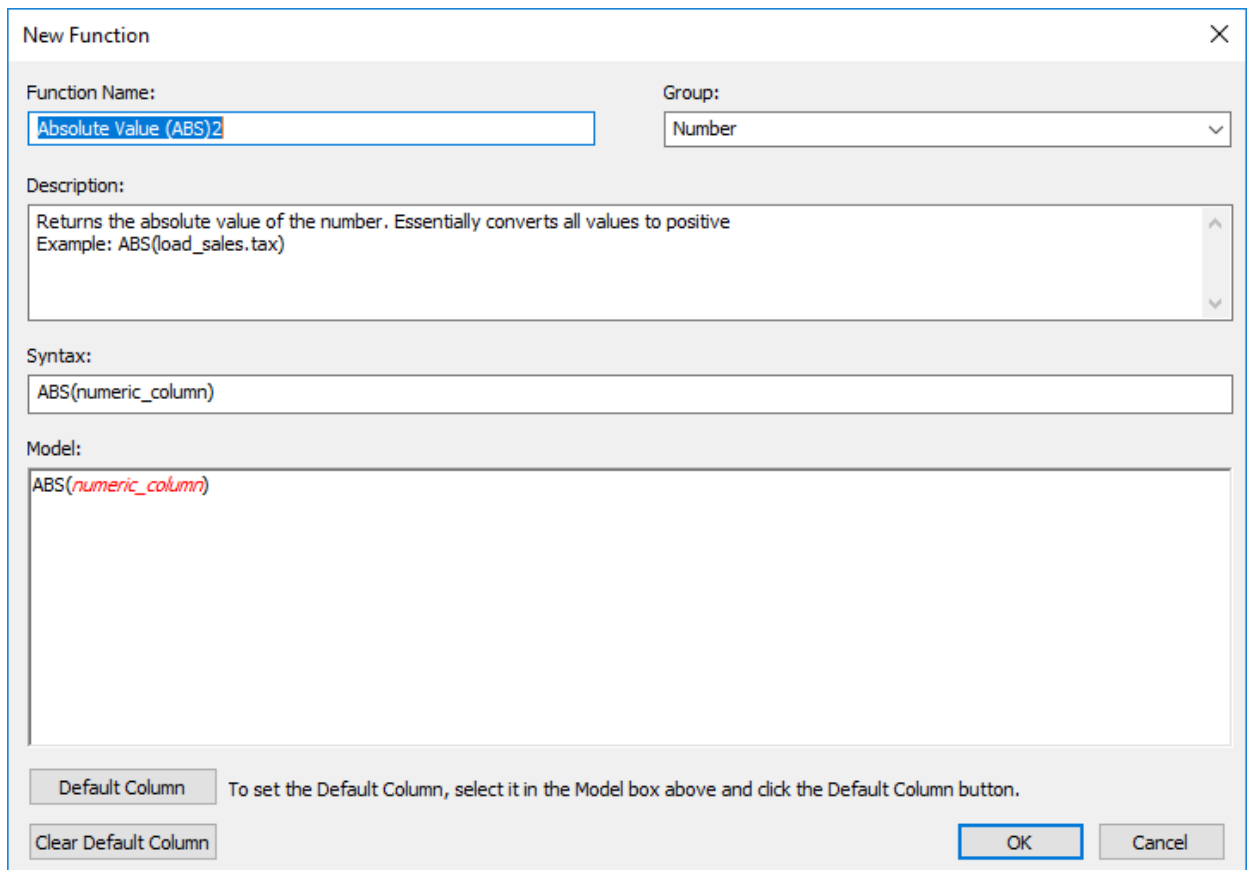
Click this button to clear the default column.

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**.



New Function

Function Name:

Group:

Description: Returns the absolute value of the number. Essentially converts all values to positive
Example: ABS(load_sales.tax)

Syntax:

Model:

To set the Default Column, select it in the Model box above and click the Default Column button.

To edit an existing function in the database function set:

Select the function and then click on the **Edit** button on the right.

Database Functions Maintenance

Function Set: Teradata Custom Functions New Copy Delete Database: Teradata

Functions:

| Name | Group | Description | Syntax | Model |
|--------------------------|------------|--|--------------------------------|-------------------------------------|
| Absolute Value (ABS) | Number | Returns the absolute value of the n... | ABS(numeric_column) | ABS(numeric_column) |
| Cosine (COS) | Number | Returns the trigonometric cosine for ... | COS(numeric_column) | COS(numeric_column) |
| Exponent (EXP) | Number | Returns e raised to the specified po... | EXP(n) | EXP(n) |
| Natural Logarithm (LN) | Number | Returns the natural, or base 'e' loga... | LN(n) | LN(n) |
| Base 10 Logarithm (L... | Number | Returns the base10 logarithm of the... | LOG(number) | LOG(number) |
| If Null Value (COALES... | Number | Returns either the passed column or... | COALESCE(numeric_column,... | COALESCE(numeric_column,def... |
| Sine (SIN) | Number | Returns the trigonometric sine for a... | SIN(numeric_column) | SIN(numeric_column) |
| Square Root (SQRT) | Number | Returns the square root of the valu... | SQRT(numeric_column) | SQRT(numeric_column) |
| Tangent (TAN) | Number | Returns the trigonometric tangent f... | TAN(numeric_column) | TAN(numeric_column) |
| Hex value (CHAR2HE... | String | Returns the hexadecimal value of th... | CHAR2HEXINT(string_column) | CHAR2HEXINT(string_column) |
| If Null Value (COALES... | String | Returns either the passed column or... | COALESCE(string_column,d... | COALESCE(string_column,default... |
| Locate (POSITION) | String | Returns an integer value containing ... | POSITION('search_string' IN... | POSITION('search_string' IN stri... |
| Lower (LOWER) | String | Returns a string where every charac... | LOWER(string_column) | LOWER(string_column) |
| Trim (TRIM) | String | Returns a string which is the passed ... | TRIM(BOTH/TRAILING/LEAD... | TRIM(BOTH/TRAILING/LEADING... |
| Sounds Like (SOUNDEX) | String | Used in a comparison situation. This ... | SOUNDEX(string_column) | SOUNDEX(string_column) |
| Sub String (SUBSTR) | String | Returns a sub string of the passed c... | SUBSTR(string_column, star... | SUBSTR(string_column, start ,co... |
| Upper (UPPER) | String | Returns a string where every charac... | UPPER(string_column) | UPPER(string_column) |
| Add Date parts (INTE... | Date | Returns a date which has a number ... | INTERVAL 'number' date_part | INTERVAL 'number' date_part |
| Add Months (ADD_M... | Date | Returns a date which has had the sp... | ADD_MONTHS(date_column... | ADD_MONTHS(date_column,nu... |
| Date Conversion (CA... | Date | Performs a date conversion or calcul... | CAST(date AS FORMAT ") | CAST(date AS FORMAT ") |
| Extract (EXTRACT) | Date | Returns the integer value of the dat... | EXTRACT(YEAR/MONTH/DA... | EXTRACT(YEAR/MONTH/DAY/H... |
| System Date (CURRE... | Date | Returns the current system date tim... | CURRENT_TIMESTAMP | CURRENT_TIMESTAMP |
| Case Statement (CASE) | Conversion | Allows the use of IF ... THEN ... ELS... | CASE [input_expression] W... | CASE [input_expression] WHEN ... |
| Cast as Type (CAST) | Conversion | Returns the column converted to th... | CAST(column AS data_type) | CAST(column AS data_type) |
| If Null Value (COALES... | Conversion | Returns either the passed column or... | COALESCE(column,default_... | COALESCE(column,default_value) |

Functions: New Copy Edit Delete Move Up Move Down

Groups: Move Up Move Down

OK Help

Change any fields as required and then click **OK**.

Edit Function ✕

Function Name: Group:

Description:
Returns an integer value containing the starting position of the search string, or zero if the search string is not found.
Example: POSITION('MITH' IN load_customer.name)

Syntax:

Model:

To set the Default Column, select it in the Model box above and click the Default Column button.

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.

Database Functions Maintenance

Function Set: Teradata Custom Functions New Copy Delete Database: Teradata

Functions:

| Name | Group | Description | Syntax | Model |
|--------------------------|------------|--|--------------------------------|-------------------------------------|
| Absolute Value (ABS) | Number | Returns the absolute value of the n... | ABS(numeric_column) | ABS(numeric_column) |
| Cosine (COS) | Number | Returns the trigonometric cosine for ... | COS(numeric_column) | COS(numeric_column) |
| Exponent (EXP) | Number | Returns e raised to the specified po... | EXP(n) | EXP(n) |
| Natural Logarithm (LN) | Number | Returns the natural, or base 'e' loga... | LN(n) | LN(n) |
| Base 10 Logarithm (L... | Number | Returns the base10 logarithm of the... | LOG(number) | LOG(number) |
| If Null Value (COALES... | Number | Returns either the passed column or ... | COALESCE(numeric_column,... | COALESCE(numeric_column,def... |
| Sine (SIN) | Number | Returns the trigonometric sine for a... | SIN(numeric_column) | SIN(numeric_column) |
| Square Root (SQRT) | Number | Returns the square root of the valu... | SQRT(numeric_column) | SQRT(numeric_column) |
| Tangent (TAN) | Number | Returns the trigonometric tangent f... | TAN(numeric_column) | TAN(numeric_column) |
| Absolute Value (ABS)2 | Number | Returns the absolute value of the n... | ABS(numeric_column) | ABS(numeric_column) |
| Hex value (CHAR2HE... | String | Returns the hexadecimal value of th... | CHAR2HEXINT(string_column) | CHAR2HEXINT(string_column) |
| If Null Value (COALES... | String | Returns either the passed column or ... | COALESCE(string_column,d... | COALESCE(string_column,default... |
| Locate (POSITION) | String | Returns an integer value containing ... | POSITION('search_string' IN... | POSITION('search_string' IN stri... |
| Lower (LOWER) | String | Returns a string where every charac... | LOWER(string_column) | LOWER(string_column) |
| Trim (TRIM) | String | Returns a string which is the passed ... | TRIM(BOTH/TRAILING/LEAD... | TRIM(BOTH/TRAILING/LEADING... |
| Sounds Like (SOUNDEX) | String | Used in a comparison situation. This ... | SOUNDEX(string_column) | SOUNDEX(string_column) |
| Sub String (SUBSTR) | String | Returns a sub string of the passed c... | SUBSTR(string_column, star... | SUBSTR(string_column, start ,co... |
| Upper (UPPER) | String | Returns a string where every charac... | UPPER(string_column) | UPPER(string_column) |
| Add Date parts (INTE... | Date | Returns a date which has a number ... | INTERVAL 'number' date_part | INTERVAL 'number' date_part |
| Add Months (ADD_M... | Date | Returns a date which has had the sp... | ADD_MONTHS(date_column,nu... | ADD_MONTHS(date_column,nu... |
| Date Conversion (CA... | Date | Performs a date conversion or calcul... | CAST(date AS FORMAT ") | CAST(date AS FORMAT ") |
| Extract (EXTRACT) | Date | Returns the integer value of the dat... | EXTRACT(YEAR/MONTH/DA... | EXTRACT(YEAR/MONTH/DAY/H... |
| System Date (CURRE... | Date | Returns the current system date tim... | CURRENT_TIMESTAMP | CURRENT_TIMESTAMP |
| Case Statement (CASE) | Conversion | Allows the use of IF ... THEN ... ELS... | CASE [input_expression] W... | CASE [input_expression] WHEN ... |
| Cast as Type (CAST) | Conversion | Returns the column converted to th... | CAST(column AS data_type) | CAST(column AS data_type) |

Functions: New Copy Edit Delete Move Up Move Down

Groups: Move Up Move Down

OK Help

Click **Yes** to confirm the delete.

WhereScape RED

Are you sure you want to delete the function Absolute Value (ABS)2

Yes No

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.

Database Functions Maintenance

Function Set: Teradata Custom Functions New Copy Delete Database: Teradata

Functions:

| Name | Group | Description | Syntax | Model |
|--------------------------|------------|--|--------------------------------|-------------------------------------|
| Absolute Value (ABS) | Number | Returns the absolute value of the n... | ABS(numeric_column) | ABS(numeric_column) |
| Cosine (COS) | Number | Returns the trigonometric cosine for ... | COS(numeric_column) | COS(numeric_column) |
| Exponent (EXP) | Number | Returns e raised to the specified po... | EXP(n) | EXP(n) |
| Natural Logarithm (LN) | Number | Returns the natural, or base 'e' loga... | LN(n) | LN(n) |
| Base 10 Logarithm (L... | Number | Returns the base10 logarithm of the... | LOG(number) | LOG(number) |
| If Null Value (COALES... | Number | Returns either the passed column or ... | COALESCE(numeric_column,... | COALESCE(numeric_column,def... |
| Sine (SIN) | Number | Returns the trigonometric sine for a... | SIN(numeric_column) | SIN(numeric_column) |
| Square Root (SQRT) | Number | Returns the square root of the valu... | SQRT(numeric_column) | SQRT(numeric_column) |
| Tangent (TAN) | Number | Returns the trigonometric tangent f... | TAN(numeric_column) | TAN(numeric_column) |
| Hex value (CHAR2HE... | String | Returns the hexadecimal value of th... | CHAR2HEXINT(string_column) | CHAR2HEXINT(string_column) |
| If Null Value (COALES... | String | Returns either the passed column or ... | COALESCE(string_column,d... | COALESCE(string_column,default... |
| Locate (POSITION) | String | Returns an integer value containing ... | POSITION('search_string' IN... | POSITION('search_string' IN stri... |
| Lower (LOWER) | String | Returns a string where every charac... | LOWER(string_column) | LOWER(string_column) |
| Trim (TRIM) | String | Returns a string which is the passed ... | TRIM(BOTH/TRAILING/LEAD... | TRIM(BOTH/TRAILING/LEADING... |
| Sounds Like (SOUNDEX) | String | Used in a comparison situation. This ... | SOUNDEX(string_column) | SOUNDEX(string_column) |
| Sub String (SUBSTR) | String | Returns a sub string of the passed c... | SUBSTR(string_column, star... | SUBSTR(string_column, start ,co... |
| Upper (UPPER) | String | Returns a string where every charac... | UPPER(string_column) | UPPER(string_column) |
| Add Date parts (INTE... | Date | Returns a date which has a number ... | INTERVAL 'number' date_part | INTERVAL 'number' date_part |
| Add Months (ADD_M... | Date | Returns a date which has had the sp... | ADD_MONTHS(date_column,nu... | ADD_MONTHS(date_column,nu... |
| Date Conversion (CA... | Date | Performs a date conversion or calcul... | CAST(date AS FORMAT ") | CAST(date AS FORMAT ") |
| Extract (EXTRACT) | Date | Returns the integer value of the dat... | EXTRACT(YEAR/MONTH/DA... | EXTRACT(YEAR/MONTH/DAY/H... |
| System Date (CURRE... | Date | Returns the current system date tim... | CURRENT_TIMESTAMP | CURRENT_TIMESTAMP |
| Case Statement (CASE) | Conversion | Allows the use of IF ... THEN ... ELS... | CASE [input_expression] W... | CASE [input_expression] WHEN ... |
| Cast as Type (CAST) | Conversion | Returns the column converted to th... | CAST(column AS data_type) | CAST(column AS data_type) |
| If Null Value (COALES... | Conversion | Returns either the passed column or ... | COALESCE(column,default_... | COALESCE(column,default_value) |

Functions: New Copy Edit Delete Move Up Move Down

Groups: Move Up Move Down

OK Help

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.

Database Functions Maintenance

Function Set: Teradata Custom Functions Database: Teradata

Functions:

| Name | Group | Description | Syntax | Model |
|-----------------------------|------------|--|--------------------------------|-------------------------------------|
| fx Absolute Value (ABS) | Number | Returns the absolute value of the n... | ABS(numeric_column) | ABS(numeric_column) |
| fx Cosine (COS) | Number | Returns the trigonometric cosine for ... | COS(numeric_column) | COS(numeric_column) |
| fx Exponent (EXP) | Number | Returns e raised to the specified po... | EXP(n) | EXP(n) |
| fx Natural Logarithm (LN) | Number | Returns the natural, or base 'e' loga... | LN(n) | LN(n) |
| fx Base 10 Logarithm (L... | Number | Returns the base10 logarithm of the... | LOG(number) | LOG(number) |
| fx If Null Value (COALES... | Number | Returns either the passed column or... | COALESCE(numeric_column,... | COALESCE(numeric_column,def... |
| fx Sine (SIN) | Number | Returns the trigonometric sine for a... | SIN(numeric_column) | SIN(numeric_column) |
| fx Square Root (SQRT) | Number | Returns the square root of the valu... | SQRT(numeric_column) | SQRT(numeric_column) |
| fx Tangent (TAN) | Number | Returns the trigonometric tangent f... | TAN(numeric_column) | TAN(numeric_column) |
| fx Hex value (CHAR2HE... | String | Returns the hexadecimal value of th... | CHAR2HEXINT(string_column) | CHAR2HEXINT(string_column) |
| fx If Null Value (COALES... | String | Returns either the passed column or... | COALESCE(string_column,d... | COALESCE(string_column,default... |
| fx Locate (POSITION) | String | Returns an integer value containing ... | POSITION('search_string' IN... | POSITION('search_string' IN stri... |
| fx Lower (LOWER) | String | Returns a string where every charac... | LOWER(string_column) | LOWER(string_column) |
| fx Trim (TRIM) | String | Returns a string which is the passed ... | TRIM(BOTH/TRAILING/LEAD... | TRIM(BOTH/TRAILING/LEADING... |
| fx Sounds Like (SOUNDEX) | String | Used in a comparison situation. This ... | SOUNDEX(string_column) | SOUNDEX(string_column) |
| fx Sub String (SUBSTR) | String | Returns a sub string of the passed c... | SUBSTR(string_column, star... | SUBSTR(string_column, start ,co... |
| fx Upper (UPPER) | String | Returns a string where every charac... | UPPER(string_column) | UPPER(string_column) |
| fx Add Date parts (INTE... | Date | Returns a date which has a number ... | INTERVAL 'number' date_part | INTERVAL 'number' date_part |
| fx Add Months (ADD_M... | Date | Returns a date which has had the sp... | ADD_MONTHS(date_column... | ADD_MONTHS(date_column,nu... |
| fx Date Conversion (CA... | Date | Performs a date conversion or calcul... | CAST(date AS FORMAT ") | CAST(date AS FORMAT ") |
| fx Extract (EXTRACT) | Date | Returns the integer value of the dat... | EXTRACT(YEAR/MONTH/DA... | EXTRACT(YEAR/MONTH/DAY/H... |
| fx System Date (CURRE... | Date | Returns the current system date tim... | CURRENT_TIMESTAMP | CURRENT_TIMESTAMP |
| fx Case Statement (CASE) | Conversion | Allows the use of IF ... THEN ... ELS... | CASE [input_expression] W... | CASE [input_expression] WHEN ... |
| fx Cast as Type (CAST) | Conversion | Returns the column converted to th... | CAST(column AS data_type) | CAST(column AS data_type) |
| fx If Null Value (COALES... | Conversion | Returns either the passed column or... | COALESCE(column,default_... | COALESCE(column,default_value) |

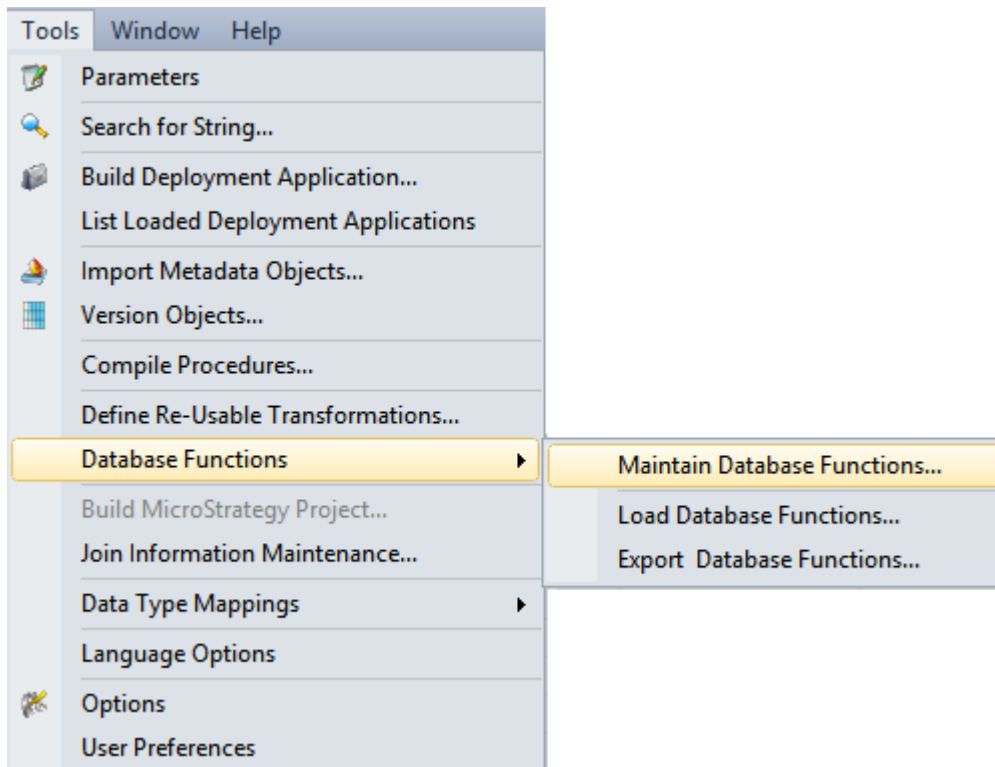
Functions: New Copy Edit Delete Move Up Move Down

Groups: Move Up Move Down

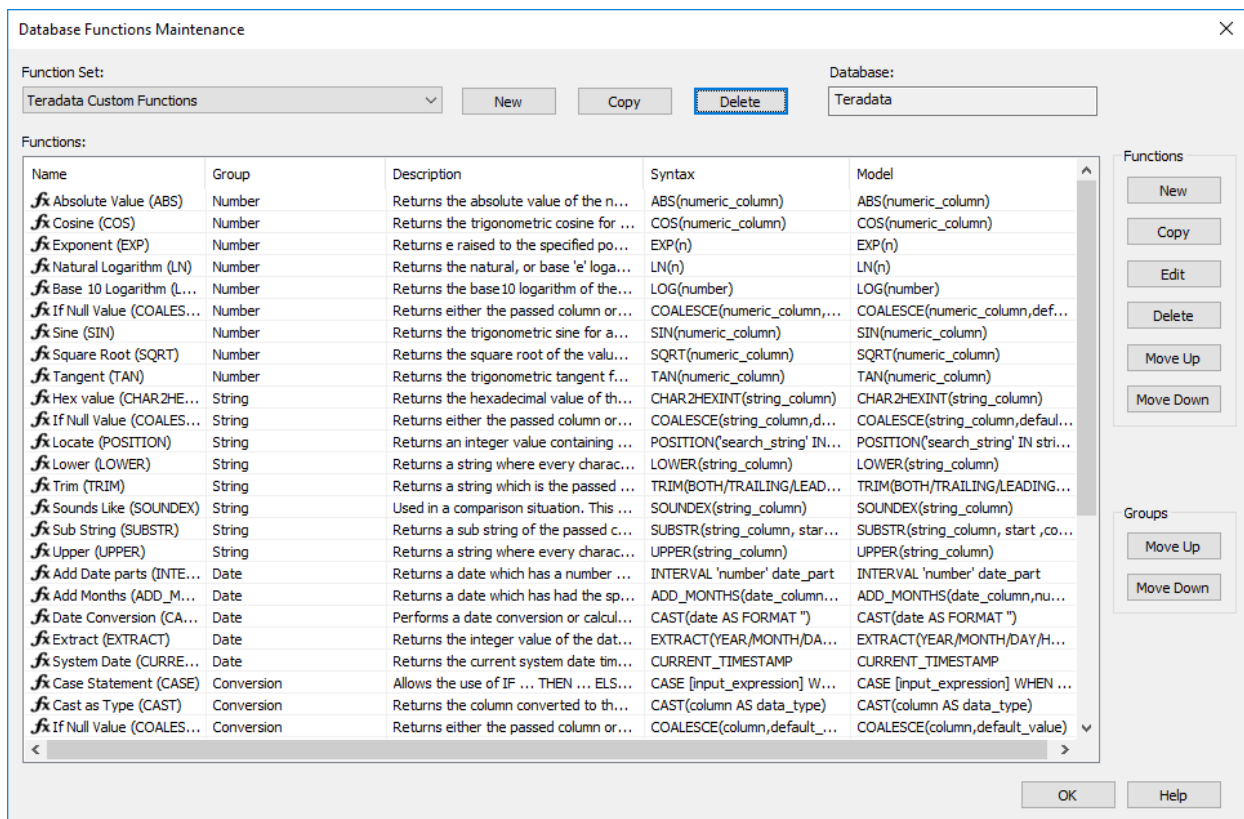
OK Help

DELETING A DATABASE FUNCTION SET

To delete a database function set, select **Tools/Database Functions/Maintain Database Functions...**

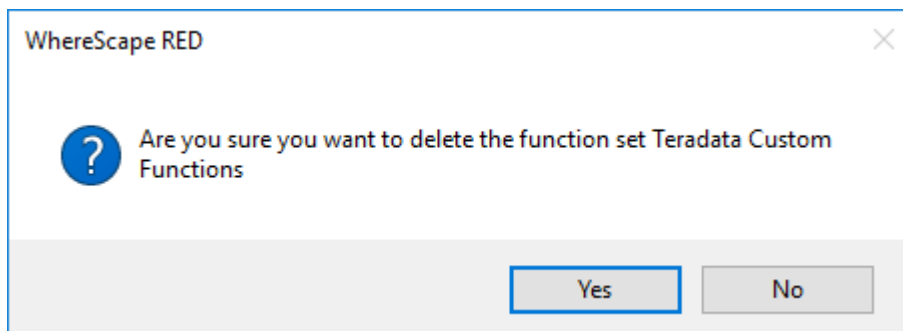


Select a **Function Set** from the drop-down list and click on the **Delete** button.



Note: The **Delete** button is disabled for standard function sets.

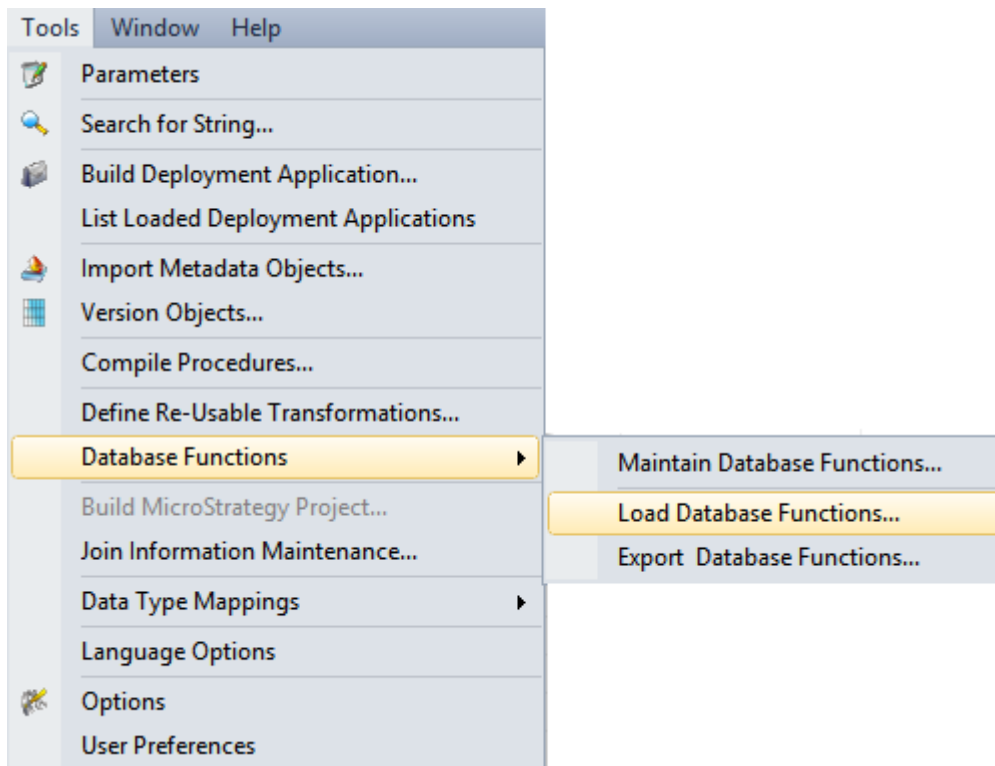
Click **Yes** to confirm the delete.



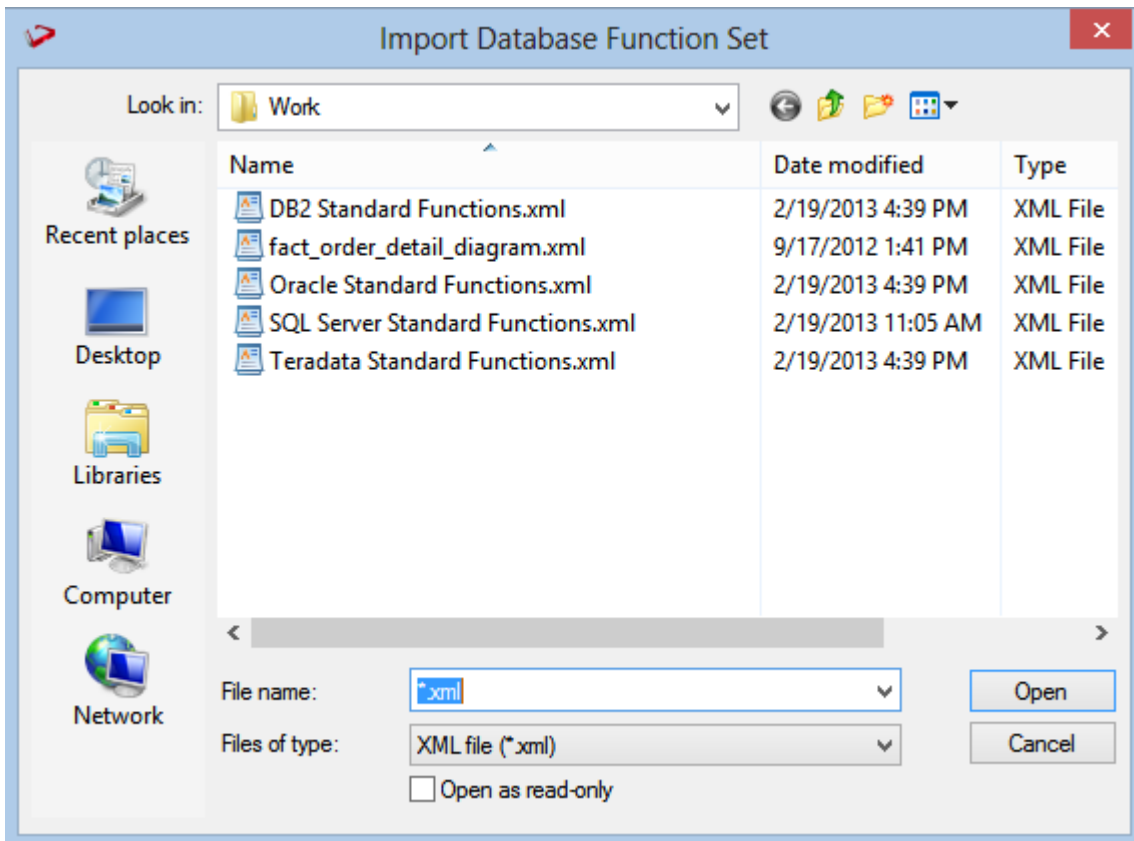
When all functions are deleted, the function set ceases to exist.

LOADING DATABASE FUNCTION SETS

To load a database function set, select **Tools/Database Functions/Load Database Functions...**

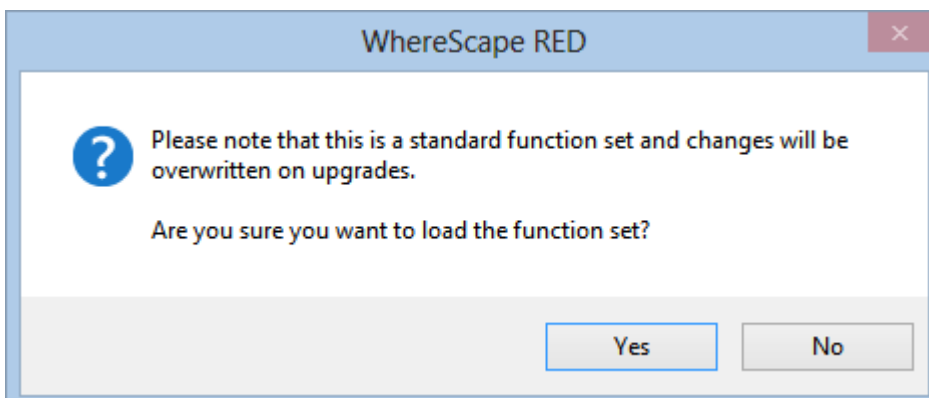


The following dialog is displayed. Select the xml file to load the database functions. By default RED expects the xml files to be in **ProgramData\WhereScape\Work**

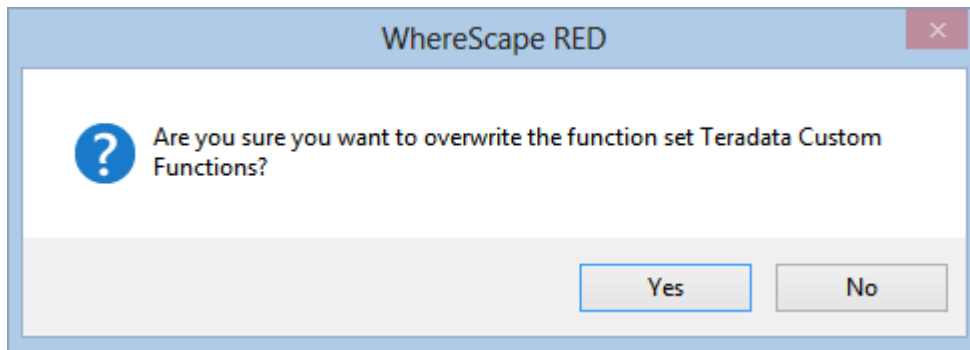


The xml file is validated using the schema definition file at <install directory>\Administrator\Function Sets\Database Function Set.xsd

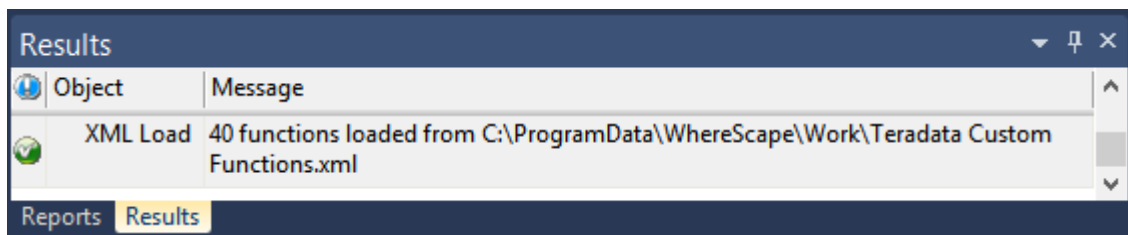
If a function set containing the phrase **Standard Function** is loaded, a warning is displayed:



If an existing function set (not a standard set) is about to be overwritten, a warning is displayed:

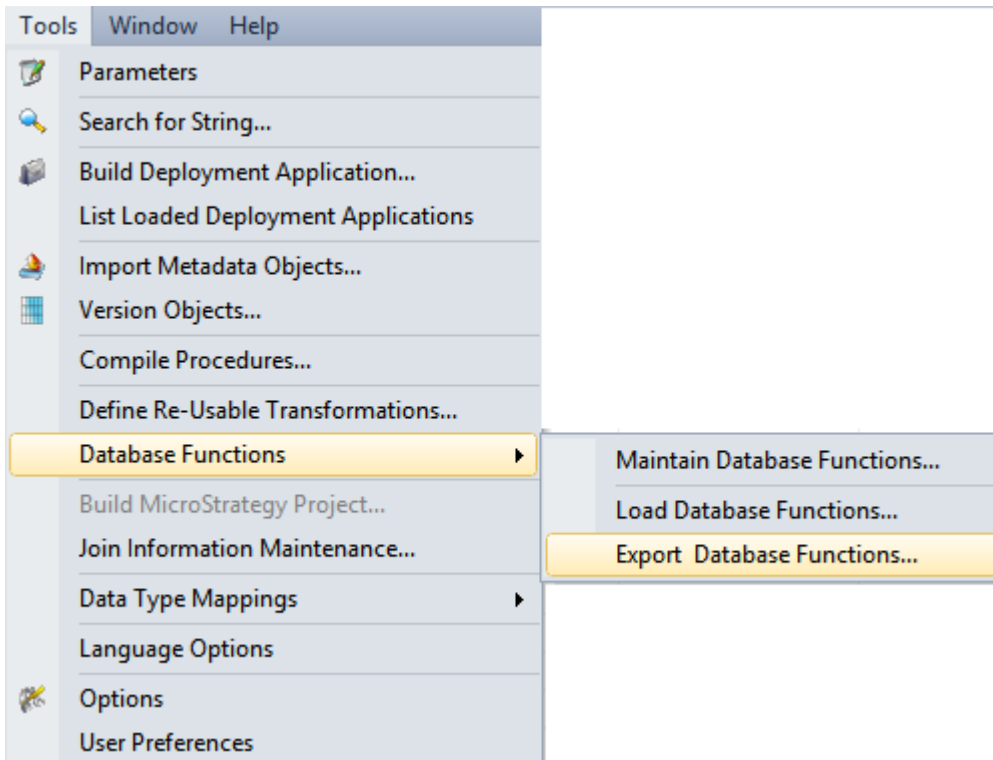


A message is displayed in the results pane.

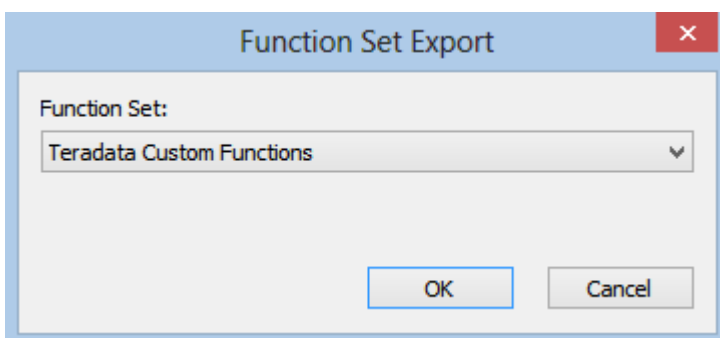


EXPORTING DATABASE FUNCTION SETS

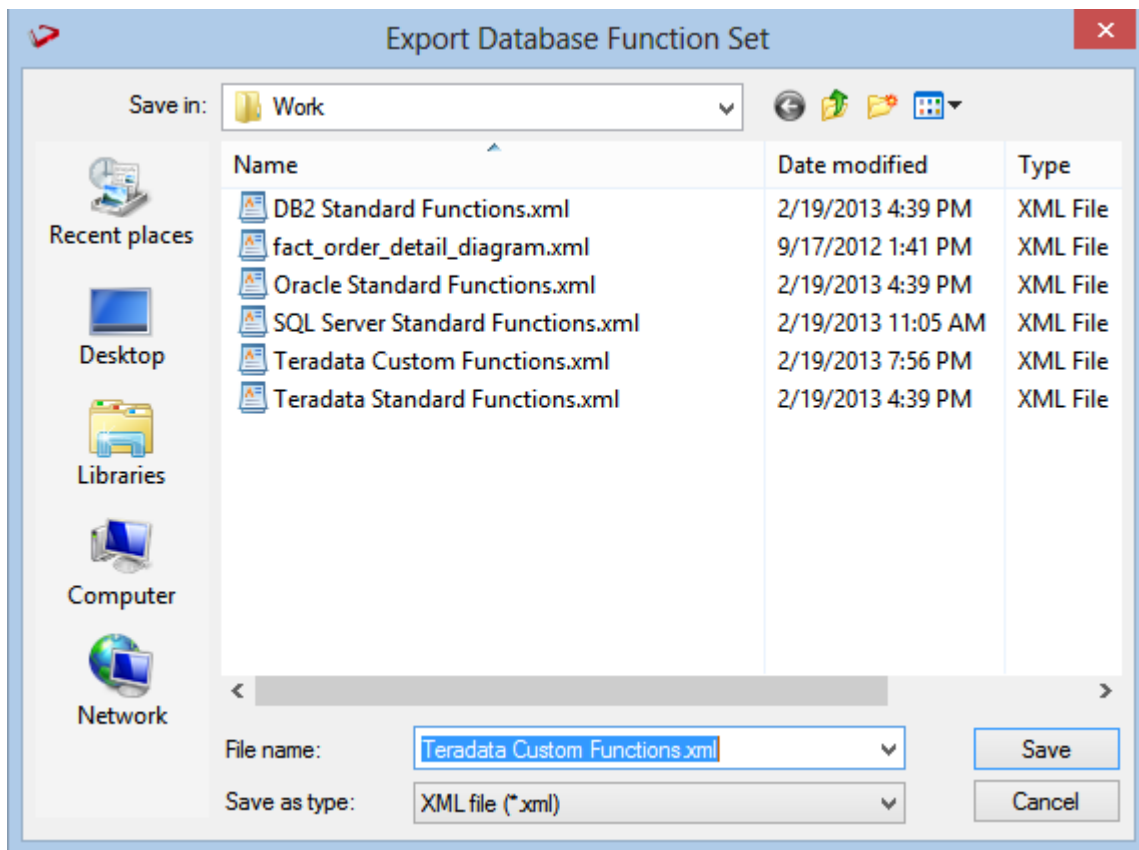
To export Database Function Sets, select **Tools/Database Functions/Export Database Functions...**



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**.



CHAPTER 44

GATHER STATISTICS

Gathering Statistics on a table 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 **Define Statistics** or to **Refresh Full 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 see *Editing Tasks* (see "*Editing Tasks in a Job*" on page 722).

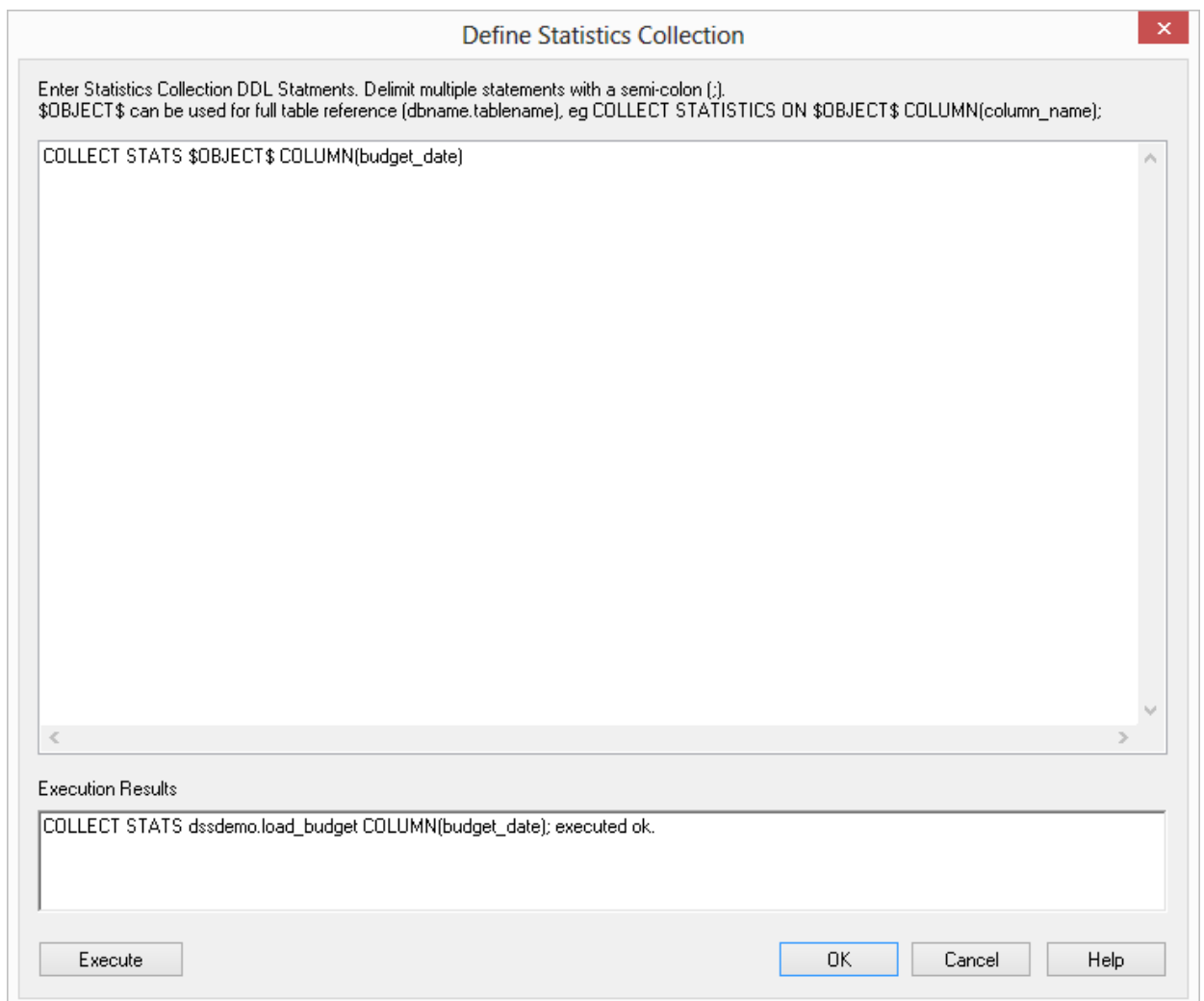
IN THIS CHAPTER

| | |
|-------------------------|------|
| Define Statistics | 1108 |
|-------------------------|------|

DEFINE STATISTICS

In a multiple database environment it is required not to have the database name hard-coded into the COLLECT STATS command. When tables are deployed to a new environment using RED, these hard coded database names are not translated to the database names in the target environment in the COLLECT STATS commands. RED can automatically add the Database Name (storage location) prefix to the table_name in the COLLECT STATS DDL.

- 1 Right click on any table object on the left pane to **Gather Statistics**.
- 2 Find **Gather Statics** on the drop-down menu.
- 3 Options allow choosing either to **Define Statistics** or to **Refresh Full Statistics**.
- 4 Select **Gather Statistics -> Define Statistics**.
- 5 On the Define Statistics Collection dialog, enter the "COLLECT STATS dbname.tablename COLUMN(column_name,);" command by using \$OBJECT\$ COLUMN (table_name).



- 6 Click **Execute** if adding the DDL statement for the first time.

When the DDL is executed, RED substitutes the Database name from the Storage Properties of the object. This improves the portability of objects between environments.

- 7 Click **OK** on the **Define Statistics Collection** dialog to close it.
- 8 Right-click on the table and select **Refresh Full Statistics** to manually refresh the statistics on that table.
- 9 To automate gathering statistics on a table, users can add a statistics task to a job to be processed by the scheduler. For more information about adding statistics tasks to jobs see **Editing Tasks** (see "**Editing Tasks in a Job**" on page 722).