

Data Info Leaders

DW Architect

User Guide

August 2010

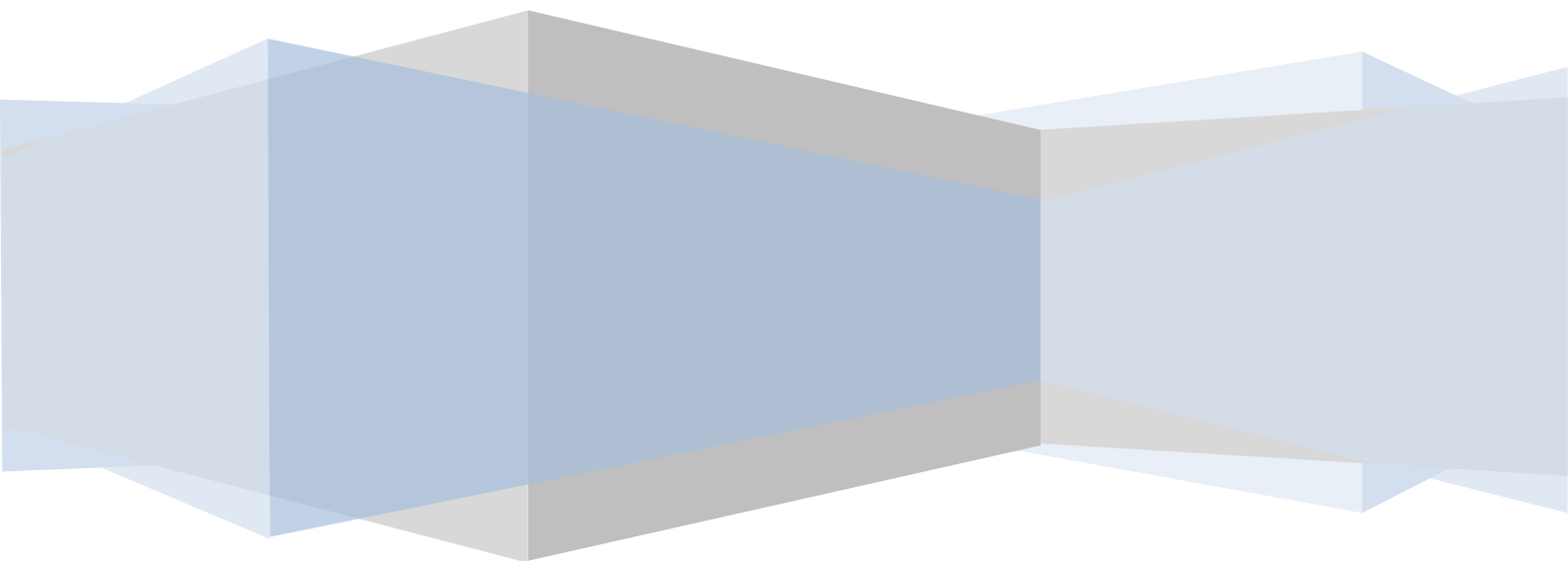


TABLE OF CONTENTS

CONFIGURATION.....	3
SET UP CONFIGURATION	3
Connections.....	3
Configuration	3
Generated Code Output Location.....	4
CREATING A NEW DW ARCHITECT PROJECT	4
CREATING A DIMENSION.....	5
Import5	
Dimension Tab.....	5
Dimension Attributes Tab.....	6
To add an Attribute	6
To Delete an Attribute	6
To Change an Attribute	7
CUSTOM MANAGEMENT COLUMNS	7
ADVANCED TAB	7
ORGANIZING YOUR FACT TABLES AND DIMENSIONS WITH SUB FOLDERS	7
CREATING A FACT TABLE	9
Import9	
Fact Table Tab.....	9
Fact Table Measures Tab	10
To Delete a Measure	10
To Change a Measure	10
ASSOCIATING DIMENSIONS.....	10
CUSTOM MANAGEMENT COLUMNS	11
DEFINING EXTRACT TRANSFORM AND LOAD PROCESSES (ETL)	12
DEFINING EXTRACTS.....	12
Overview Tab.....	12
Source Tab.....	12
Mapping Tab	13
Advanced Tab.....	13
DEFINING FACT/DIMENSION TRANSFORMS.....	14
Overview Tab.....	14
Staging Sources.....	14
Match On	16
Dimension Lookups	16
Transformation Tab.....	17
Advanced Tab.....	17
EXPRESSION LANGUAGE.....	18
STANDARD MANAGEMENT COLUMNS	19
REFERENCE DATA	20
LOAD FREQUENCY	20

CREATING A MANY TO MANY RELATIONSHIP	20
CONFORMED DIMENSIONS	20
GENERATING CODE	21
CONFIGURATION	21
<i>Connections</i>	21
<i>Configuration</i>	22
<i>Generate and Deploy</i>	22
GENERATING THE CODE	22
GENERATING CODE VIA THE COMMAND LINE	23
CUSTOM GENERATION ENGINES AND TEMPLATES	23
<i>Custom Generation Templates</i>	23
<i>Custom Generation Engines</i>	23
DEPLOYING CODE	24
CONFIGURATION	24
<i>Connections</i>	24
<i>Generate and Deploy</i>	24
DEPLOYING THE CODE	24
DEPLOYING CODE VIA THE COMMAND LINE	25
CUSTOM DEPLOYMENT ENGINES	25
SOURCE CONTROL AND MIGRATION	25
DATE RANGE TRANSACTION FACT TABLE PATTERN	26
IMPLEMENTATION	27
<i>Control Properties (Not Shown)</i>	27
<i>Staging Tables</i>	28
<i>Extract</i>	28
<i>Warehouse Table</i>	28
<i>Fact Updates Transform</i>	28
<i>Fact Inserts Transform</i>	28
<i>View/DSV</i>	29
<i>Build</i>	29
DEFINING THE META DATA	29
<i>Fact Table</i>	29
<i>Staging Tables</i>	29
DW BATCH MANAGER	30
WORKFLOW	30
CONFIGURATION	31
EXECUTING THE BATCH	31
CUSTOM TASK EXECUTION PROVIDERS	31
SUPPORT32	
SUPPORT	32

CONFIGURATION

SET UP CONFIGURATION

Create a configuration file that points to your Staging and Data Warehouse databases.. You can have a different configuration for each developer machine and environment.

1. Open the DW Architect Management Console under **All Program>Data Info Leaders** from your start menu.
2. Select **File>New** to create a new configuration file.
3. Change the configuration as described below.
4. Click **File>Save** and save the configuration file.

CONNECTIONS

On the Connections tab, use the configure buttons to create connection strings to your Data Warehouse and Staging Database..

Note! The Databases need to exist on the database server before you can set them as your Data Warehouse and Staging databases.

CONNECTION STRINGS

The connection strings table allows you to specify environment specific connections strings for the source system connections specified in your DW Architect project. This way you can define different source connections for each environment, i.e. development, test, production etc.

1. In the Name field, type the name of the connection from your project. Note this must 'Connection Manager Name' in the Connection Manager dialog in your DW Architect Project exactly.
2. Use the configure button to configure your connection string. Alternatively you can type or paste a connection string into the connection string field. To manually set the Server Name and Provider fields, view the code of your connection in the DW Architect project, and use the values of the ServerName and Provider elements.

Generation Tip: The standard generation templates use these connection string definitions to generate SQL Server Linked Servers. The Provider element is used to determine the format of the Create command, and the Server Name is used in the command.

CONFIGURATION

The configuration tab is used to set some defaults for Data Warehouse tables.

1. **Data Warehouse Database Schema:** Use this field to specify the schema name of Data Warehouse tables in the Data Warehouse database. The default is dbo.
2. **Dimension Table Prefix:** The dimension table prefix is applied to the front of the table name of every dimension table in your schema. This allows you to keep your design free of implementation details.

3. **Fact Table Prefix:** The dimension table prefix is applied to the front of the table name of every fact table in your schema. This allows you to keep your design free of implementation details.
4. **Custom Meta Data:** The custom Meta Data table can be used to create custom meta data that can be used in the generation, deployment or batch execution processes.

GENERATED CODE OUTPUT LOCATION

Set the path that is used to store the code generated from your project. Select the Generate/Deploy tab and use the Browse button next to the “Default Generation Output Path” field to select the directory that generate code will be stored.

This location is also used by the deploy process as the source directory of code to deploy.

CREATING A NEW DW ARCHITECT PROJECT

This section describes how to create a new project.

1. Open Visual Studio. From the file menu **New>Project** Solution.
2. Select ‘Data Info Leaders Projects’ in the Project Types Tree.
3. Select DW Architect Project from the templates list. If you have just installed DW Architect You may need to restart Visual Studio before the DW Architect Project Template appears.
4. Give the project a name and directory as per normal.

CREATING A DIMENSION

To create a dimension, right click the 'Dimensions' folder (or one of its sub folders) and select Create New.

There are two ways to create a dimension. By far the quickest way is to import the Meta Data from a Source System, and then edit as required. If the Source Meta data is not available, then you can manually enter the information.

IMPORT

Use the Import button on the Dimension tab to import the Meta Data for the Dimension from a Source System.

You must first create a connection to the Source System through a Connection Manager prior to import. To Import the Meta Data.

1. Click the **Import** button.
2. Select the **Connection** where the source table resides.
3. Select the **Table** and press OK.

The dimension name will default to the source table name, and the dimension attributes will be populate with the column names and data types from the source table. Some connection types don't return column data types.

DIMENSION TAB

1. **Dimension Name.** The Dimension Name is the name of your table in your data warehouse relational database. Spaces in the name will be converted to _ automatically when you save. The prefix for the table name can be set in the configuration utility.
2. **Cube Dimension Name.** This is the friendly name the user would see through a Cube browser. Spaces are not replaced in this name.
3. **Load Freq.** The frequency at which the dimension is loaded from its source. Load frequencies can be configured through the Reference Data> Load Frequency file.
4. **Dimension Description.** A description of the contents of the dimension. optional.
5. **Reference.** This is a free form text you can use to reference the dimension design back to another design document, perhaps a business requirements document.

DIMENSION ATTRIBUTES TAB

You can add, delete and modify dimension attributes through the dimension's Attribute tab.

TO ADD AN ATTRIBUTE

Click in the empty row at the end of the list.

- **Attribute Name.** Enter the attribute name. Spaces are replaced with _ when the dimension is saved, and the first letter of each word changed to upper case. The Name must be unique within the dimension.
- **Data Type.** Select a data type from the dropdown list, or just type the data type. The data type will be validated to ensure it is a valid SQL Server data type when the data type field loses focus. If the data type is invalid, a dialog box will display, and the field cleared.
- **Column Type.** Choose the column type for the attribute. If the attribute is part of the Natural/Business key of the dimension, then select 'Business Key'. The business key can be a composite key. That is, more than one attribute can have a type of Business Key. Normally the business key is what your ETL process will use to look up the dimension to identify if source rows already exist in the dimension. It is the primary key of the source table.
- **SCD Type.** SCD Type is an acronym for Slowly Changing Dimension Type. The list of types correspond to the SCD types set out by Ralph Kimball. When the Attribute type is 'Attribute' this field is enabled and is required. Select the appropriate SCD type for the attribute. The SCD type selected will affect how the DDL is generated for the attribute (using the supplied templates). Type 3 and 6 SCD Type attribute will have additional <attribute name>_Prev columns created for them.
- **Merge, Replace, Reuse.** The Merge, Replace, Reuse indicator is used to identify Business keys that can be merged, replaced or reused. An example might be a product code, which can be used to represent a product for a period of time, the product then goes out of production, and then at a later time the product code is used to represent another product. This meta data can be used in an ETL process which captures these types of events. If 'Merge, Replace, Reuse' is selected then an additional <attribute name>_Prev columns is created for them, using the supplied templates.
- **Description.** Add a description of the attribute. Generally the description should make sense to an end user. Descriptions are used in the generated Data Dictionary.
- **Parent.** The parent indicator is used to indicate that the attribute is the natural key of the corresponding parent row in a parent-child hierarchy. That is, each row keeps its parent identifier in this attribute column. Parent identifiers can be composite keys, so more than 1 attribute can be marked as a Parent. The presence of a Parent indicator on any attribute will cause a Parent Surrogate Key to be generated in the DDL using the supplied DDL templates. Only a single Parent surrogate key is produced regardless of how many attributes are marked as Parent.
- **Reference.** This is a free form text you can use to reference the attribute back to another design document, or perhaps a report the attribute appears on.

TO DELETE AN ATTRIBUTE

Click the 'delete' link next to the attribute in the Attributes table. Changes are only made permanent after the document is saved.

TO CHANGE AN ATTRIBUTE

Simply find the attribute in the list, click in the field you want to change, and change as desired. Attribute name changes are generated in the DDL as non destructive rename script. Changes are only made permanent after the document is saved.

CUSTOM MANAGEMENT COLUMNS

What are Management Columns? Management columns are columns that are added to your Fact and Dimension tables to help manage the data, during ETL. Management Columns include things like:

- Batch Identifier.
- Row Status (Current - Not Current).
- Row Effective Dates.
- Checksum (if you use checksums in your ETL)

DW Architect allows you to specify Custom management columns on an individual Fact/Dimension basis. A set of Standard Management Columns is specified for all Fact/Dimension tables. See section Standard Management Columns for an explanation of Standard Management Columns.

Custom Management columns can be used for Fact and Dimensions that have custom ETL that requires additional management columns for its operation.

Custom Management Columns can be added, changed and deleted through the Management Columns tab > Custom Management Columns table in the Fact and Dimension editors.

ADVANCED TAB

- **Custom Surrogate Key Name.** This field allows the you to specify a custom name for the surrogate key of this dimension. This is useful for in a conforming dimension scenario. See section Conformed Dimensions for more details. The standard DDL templates use the indicator and name to generate the correct surrogate key column name in the DDL.
- **Custom Surrogate Data Type.** This field allows the you to specify a custom data type for the surrogate key of this dimension. This is useful for in a conforming dimension scenario. See section Conformed Dimensions for more details. The standard DDL templates use the indicator and data type to generate the correct surrogate key column data type in the DDL.
- **Surrogate Key is Auto Generated.** Un-tick this option if the ETL manages surrogate keys, instead of the database auto generating surrogate keys. The standard DDL templates use the indicator to set IDENTITY on dimension keys.
- **Generate DDL.** Un-tick this option if you don't want DW Architect to generate the DDL for this table.
- **Conformed.** Meta data indicating the Dimension is a conformed Dimension from another Warehouse. This can be used in the Generation step to generate Conforming code.
- **Dimension Tags.** Custom Meta Data for the Dimension. Tag the dimension with a tag Name and Tag Value.

ORGANIZING YOUR FACT TABLES AND DIMENSIONS WITH SUB FOLDERS

Within the Dimension and Fact Table folders you can create sub folders to organize your Dimensions and Fact Tables into logical groups, and make them easier to find. To create a Sub Folder, right click the Dimensions/Fact Tables folder and select **Add>New Folder**.

The application supports a full hierarchy of folders within folders.

CREATING A FACT TABLE

To create a Fact table, right click the 'Fact Tables' folder (or one of its sub folders) and select Create New.

There are two ways to create a Fact Table. By far the quickest way is to import the Meta Data from a Source System, and then edit as required. If the Source Meta data is not available, then you can manually enter the information.

IMPORT

Use the Import button on the Fact Table tab to import the Meta Data for the Fact Table from a Source System.

You must first create a connection to the Source System through a Connection Manager prior to import. To Import the Meta Data.

4. Click the **Import** button.
5. Select the **Connection** where the source table resides.
6. Select the **Table** and press OK.

The Fact Table name will default to the source table name, and the Fact Table Measures will be populated with the column names and data types from the source table. Some connection types don't return column data types.

FACT TABLE TAB

- **Fact Table Name.** The Fact Table Name is the name of your table in your data warehouse relational database. Spaces in the name will be converted to _ automatically when you save. The prefix for the table name can be set in the configuration utility.
- **Measure Group Name.** This is the Measure Group Name the user would see for the Fact Table through a Cube browser. Spaces are not replaced in this name.
- **Load Freq.** The frequency at which the Fact Table is loaded from its source. Load frequencies can be configured through the Reference Data> Load Frequency file.
- **Fact Table Description.** A description of the contents of the Fact Table. Optional.
- **Reference.** This is a free form text you can use to reference the dimension design back to another design document, perhaps a business requirements document.

FACT TABLE MEASURES TAB

- **Measure Name.** Enter the measure name. Spaces are replaced with _ when the Fact Table is saved, and the first letter of each word changed to upper case. The Name must be unique within the dimension.
- **Data Type.** Select a data type from the dropdown list, or just type the data type. The data type will be validated to ensure it is a valid SQL Server data type when the data type field loses focus. If the data type is invalid, a dialog box will display, and the field cleared.
- **Description.** Add a description of the measure. Generally the description should make sense to an end user. Descriptions are used in Data Dictionary generation.
- **Reference.** This is a free form text you can use to reference the measure back to another design document, or perhaps a report the measure appears on.
- **Business Key.** Tick the row(s) that represent the Fact Table's business/natural key. This is used in the definition of the Transform/Load process for the table.

TO DELETE A MEASURE

Click the delete link next to the measure in the Measures table. Changes are only made permanent after the document is saved.

TO CHANGE A MEASURE

Simply find the measure in the measure table, click in the field you want to change, and change as desired. Measure name changes are generated in the DDL as non-destructive rename scripts. Changes are only made permanent after the document is saved.

ASSOCIATING DIMENSIONS

In order to complete the star schema of the fact table you must associate it with one or more of the dimensions in the project. To associate a Fact Table to a dimension, use the Dimensionality tab of the Fact Table.

Select the dimension you want to associate to the fact table in the drop down list in the Dimension column. Click on another cell to ensure the association is recognized. If you intend to associate the dimension as a role play dimension, then give it a role play name. A dimension can be associated to a fact table multiple times, but only once without a role play name. That is, you can associate a dimension to a fact without a role play name once, and every other association must have a role play name and each role play name must be different.

Changes are not saved until the Fact table is saved.

CUSTOM MANAGEMENT COLUMNS

What are Management Columns? Management columns are columns that are added to your Fact and Dimension tables to help manage the data, during ETL. Management Columns include things like:

- Batch Identifier.
- Row Status (Current - Not Current).
- Row Effective Dates.
- Checksum (if you use checksums in your ETL)

DW Architect allows you to specify Custom management columns on an individual Fact/Dimension basis. A set of Standard Management Columns is specified for all Fact/Dimension tables. See section Standard Management Columns for an explanation of Standard Management Columns.

Custom Management columns can be used for Fact and Dimensions that have custom ETL that requires additional management columns for its operation.

Custom Management Columns can be added, changed and deleted through the Management Columns tab > Custom Management Columns table in the Fact and Dimensions editor.

DEFINING EXTRACT TRANSFORM AND LOAD PROCESSES (ETL)

DEFINING EXTRACTS

To define an Extract open the staging table document of the staging table you wish to define the Extract for, and select the **Add** button on the **Extract** tab. The Extract dialog appears.

OVERVIEW TAB

- **Extract Name.** The Extract name will default to the name of the staging table. The Name is used to uniquely identify the Extract and is used by the standard generation templates to name the Extract's stored procedure.
- **Load Frequency.** Set the frequency with which the Extract should execute. Additional Load Frequencies can be defined in the Reference Data/Load Frequency.If document. At present this attribute is not used by the standard generation templates.
- **Extract Pattern.** Select the pattern that will be used to determine the type of code that will be generated for this Extract. Additional Patterns can be defined in the Reference Data/Patterns.rf document with a category of 'Extract'. The standard Extract patterns include:
 - **Truncate.** The Staging table is truncated prior to Extract.
 - **Date Range.** Usually used for the source of high volume Transaction Fact tables, where only a range of data (based on date) is extracted from the source. See the Date Range Transaction Fact Pattern section for more details.
- **Extract Overview.** A description of the extract for documentation purposes.

SOURCE TAB

Defines the source table/query of the Extract.

- **Connection.** First select the connection which contains the table that is the source of the Extract.
- **Source Table View.** Select the name of the table/View that is the source of the Extract.
- **Source Query.** Select the Source Query option if the source of the Extract is a query rather than a single table. Enter your query in the source query text box.

MAPPING TAB

The Mapping tab defines the column mappings between the source columns and the staging table columns. The mappings may also be expressions.

- **Staging Column.** The Staging Column is the name of a Staging table column. The Staging Column is populated from the list of columns defined for the Staging table.
- **Source Column Mapping.** The Source Column Mapping is the name of the Source column that maps to the Staging table column. This field is auto matched based on name, after a source is selected in the Source tab.
- **... Button.** The ... button is used to change the setting for the column, define an expression mapping, and add a description to the mapping for documentation purposes. Click the button and the Extract Mapping dialog appears.
 - **Extract Mapping Description.** A description of the mapping for documentation purposes. Non Mandatory.
 - **Source Column Mapping.** Select a Source Column mapping for the Staging column from the drop down.
 - **Expression.** Write an expression for the mapping. Note the expression can only reference source columns (not staging columns). See the Expression Language section for more details.
 - **Extract Mapping Tags.** Define custom meta Data for the mapping.

ADVANCED TAB

- **Execution Order.** Define an Execution order for this extract in its phase. If this Extract should be run after other Extracts then elevate its Execution order.
- **Extract Tags.** Define custom meta Data for the Extract.

DEFINING FACT/DIMENSION TRANSFORMS

To define a Transform & Load process open the Dimension/Fact table document of the Dimension/Fact table you wish to define the Transform & Load for, and select the **Add** button on the **ETL** tab. The Transform & Load dialog appears.

OVERVIEW TAB

- **Logical Data Map Name.** The Logical Data Map name will default to the name of the Dimension/Fact table. The Name is used to uniquely identify the Transform and is used by the standard generation templates to name the Transform stored procedure.
- **Load Frequency.** Set the frequency with which the Transform should execute. Additional Load Frequencies can be defined in the Reference Data/Load Frequency.If document. At present this attribute is not used by the standard generation templates.
- **ETL Pattern.** Select the code pattern that will be used as the basis for the generated code for this Transform. Additional Patterns can be defined in the Reference Data/Patterns.rf document with a category of 'Transform'. The standard Transform patterns include:
 - **Dimension Patterns:**
 - **Type 1 and 2 Dimension.** This pattern caters for both Type 1 and Type 2 Slowly changing dimension attributes in the same dimension.
 - **Fact Patterns:**
 - **Fact Bridge.** Use this pattern if the fact table is a Fact Bridge participating in a Many to Many star schema join.
 - **Transaction Date Range.** Use this pattern if your Fact table is a high volume Transaction Fact as defined by Ralph Kimball. Transaction facts capture discrete transactions (e.g. sales, inventory pick, phone call etc).
- **Transform Overview.** A description of the Transform for documentation purposes.

STAGING SOURCES

Defines the Staging tables and columns that are the source of the transform.

The staging sources can be a join of a single primary staging table and multiple secondary staging tables. The primary Staging table should be the staging table that has the same grain as your destination Dimension/Fact table.

PRIMARY STAGING TABLE

- **Primary Staging Table.** Select the Staging table that will be the primary source for your transform. The primary source will have the same grain as the destination Fact/Dimension.
- **Staging Table Column List.** This list is automatically populated with all the fields from the Primary Staging table. Delete any columns that are not required.
- **... Primary Staging Button.** Click this button to specify further details for the Primary Staging Table. The Source Staging Table dialog appears.
 - Select the Primary Staging Table columns that are required for the transform.
 - **Filter.** In some cases, you may want to filter the data returned for the primary Staging table in the Transform process. Define the Filter, using an Expression. See the Expression Language section for details of how to refer to Staging and Warehouse tables and columns.
 - **Tag Table.** Define custom meta data for the Primary Staging table.

SECONDARY STAGING TABLES

If data is required from another Staging table in the transform, define the table in the Secondary Staging Tables table, and how it joins to the primary (or other secondary).

- **Staging Table Name.** Select the Secondary Staging table from the drop down.
- **... Secondary Staging Button.** Click this button to specify further details for the Secondary Staging Table. The Source Staging Table dialog appears.
 - Select the Secondary Staging Table columns that are required for the transform.
 - **Join On.** Write an expression describing how the Secondary table joins to either the Primary Staging table, or other Secondary Staging tables. See the Expression Language section for details of how to refer to Staging and Warehouse tables and columns.
 - **Filter.** In some cases, you may want to filter the data returned for the Secondary Staging table in the Transform process. Define the Filter, using an Expression. See the Expression Language section for details of how to refer to Staging and Warehouse tables and columns.
 - **Tag Table.** Define custom meta data for the Primary Staging table.

MATCH ON

The Match On tab defines the columns of the Staging source tables that match to the business key(s) of the target Fact/Dimension for the purposes of identify new and deleted source records. In the standard generated code, where a match is found, further comparison is made based on checksums, to determine if a change has been made to the target row.

There are 2 ways to define the match:

1. Use the Match link below the bottom right of the table to automatically match Staging to Target Business key based on Name.
2. Select the Match manually.
 - **Staging Table.** Select the Staging table that contains the
 - **Staging Column.** Select the matching Staging Column.
 - **... Button.** Click the ... button if it is necessary to define an expression on columns of the Staging source to match to the target business key.

DIMENSION LOOKUPS

This tab only appears for Fact tables. Use this tab to define how the Transform should look-up the corresponding member in the associated Dimension based on Staging columns.

- **... Button.** Click the ... button to open the Dimension Lookup .
 - **Where Table.**

There are 2 ways to define the lookup using a where statement.

- Use the match button at the bottom right of the table to automatically match the Staging to target Dimension business key columns.
- Define the match manually.
 - **Business Key.** The Business Key column is populated from the list of business keys for the Dimension.
 - **Staging Table.** Select the Staging Table that contains the column that maps to the Dimension Business Key.
 - **Staging Column.** Select the Staging column that maps to the Dimension Business Key.
 - **Expression and ... Button.** Define an expression using Staging columns that maps to the Dimension Business Key.
- **Join On.** In some cases it is necessary to define a Join On expression to join the Staging columns to the Dimension table columns.
- **Advanced Tab**
 - **Filter.** In some cases you might want to filter the Dimension prior to applying the lookup. Define the where clause for filtering here.
 - **Dimension Lookup Tags.** Define custom Meta Data for the Dimension Lookup.

TRANSFORMATION TAB

The transformation tab allows you to map Staging table columns to Attributes/Measures during the transformation.

There are 2 ways to define the mapping:

1. Use the Match link below the bottom right of the table to automatically match Staging to Target Attributes/Measures based on Name.
2. Select the Mapping manually.
 - **... Button.** The ... button is used to change the mapping for the column, define an expression mapping, and add a description to the mapping for documentation purposes. Click the button and the Transform Mapping dialog appears.
 - **Transform Mapping Description.** A description of the mapping for documentation purposes. Non Mandatory.
 - **Staging Column Mapping.** Select the Staging table and Staging Column mapping for the Target Attribute/Measure from the drop downs.
 - **Expression.** Write an expression for the mapping. Note the expression can only reference Staging columns. See the Expression Language section for more details.
 - **Transform Mapping Tags.** Define custom meta Data for the mapping.

ADVANCED TAB

- **Execution Order.** Define a Execution order for this Transform in its phase. If this Transform should be run after other Transforms then elevate its Execution order.
- **Transform Tags.** Define custom meta Data for the Transform.
- **Auto Generate Key.** If the surrogate key for the Dimension is a 'smart' key (like 20100616 for a Calendar dimension), select false, and then specify the source Staging column that contains the 'smart' key.

EXPRESSION LANGUAGE

DW Architect supports expressions. The standard generation templates require expressions in SQL syntax, however, if you use custom generation templates, you can define expressions in any language you choose.

The standard templates interpret the following custom syntax into Staging and Warehouse table column names.

Staging Columns: stg(Source_System_Abbrev, Table_Name, Column_Name)

Where

- **Source_System_Abbrev** is the abbreviation for the Source System the table belongs too. Must match the Source System Abbreviation field of the Source System document.
- **Table_Name** is the table name of the Staging Table that contains the Staging Column. Must match the Staging Table name field in the Staging document.
- **Column_Name** is the name of the Staging Column. Must match the Staging Columns tab 'Column Name' field in the Staging document.

Warehouse Columns: whs(dim/fact, Table_Name, Column_Name)

Where

- **dim/fact.** If the table the column belongs to is a dimension, then 'dim', otherwise 'fact'.
- **Table_Name** is the table name of the Warehouse Table that contains the Column. Must match the Dimension/Fact Table name field in the Dimension/Fact document.
- **Column_Name** is the name of the Attribute/Measure. Must match the Attribute Name/Measure Name tab 'Column Name' field in the Dimension/Fact document.

It is necessary to reference staging and warehouse columns in this way, because columns are prefixed and suffixed in the generated code in different ways depending on the circumstances.

STANDARD MANAGEMENT COLUMNS

What are Management Columns? Management columns are columns that are added to your Fact and Dimension tables to help manage the data, during ETL. Management Columns include things like:

- Batch Identifier.
- Row Status (Current - Not Current).
- Row Effective Dates.
- Checksum (if you use checksums in your ETL)

DW Architect has the concept of a set of standard management columns. These are managed through the 'Dimension Standard Management Columns' and 'Fact Standard Management Columns' documents in the Reference Data project folder. You can set up different standard management columns for Facts and Dimensions.

Any management columns you create in these documents will be applied to the Data Warehouse tables when DDL is generated. This includes any changes and deletions of standard management columns. Standard Management columns can be disassociated from individual Fact and Dimension tables through the Fact or Dimension's editor. Custom management columns can also be added to each Fact and Dimension.

To modify Standard management columns, open the appropriate Standard Management Columns document in the Reference Data folder. There you can add, change and delete Standard Management columns.

The DW Architect project template includes a set of standard management columns that are required by the code generated from the default generation templates to operate correctly. They include:

Dimension Standard Management Columns:

- `Batch_Execution_Id`. The Id of the Batch that last added or updated the row in the dimension table.
- `Row_Effective_Date`. The Date the row became effective.
- `Row_End_Date`. The Date the row was no longer effective. This is associated with a change in Latest/Current status.
- `Row_Is_Latest`. Indicates the row is the latest row (or not the latest row) in a series of changes to the underlying member in the dimension.
- `Row_Is_Current`. Indicates the row still appears (or doesn't appear) in the source system.
- `SCD_Type_1_Hash`. The checksum used to identify changes to the dimension member.

Fact Tables Standard Management Columns

- `Batch_Execution_Id`. The Id of the Batch that last added or updated the row in the fact table.
- `Row_Is_Current`. Indicates the row still appears (or doesn't appear) in the source system.
- `Checksum`. The checksum used to identify changes to the fact.

Staging Standard Management Columns:

- **Reject Ind.** Used to indicate row has been rejected based on certain business rule criteria.

REFERENCE DATA

The reference data folder holds simple reference data editors for reference data required by the application. The only editor available in this release is the Load Frequency data.

LOAD FREQUENCY

The Load Frequency Reference data editor determines what values appear in the Load Freq drop down lists in the dimension and fact table editors. The details of the fields are:

- **Load Freq Name:** This is the label that appears in the Dimension and Editor Load Freq drop down lists.
- **Calendar Attribute:** This is the Calendar Attribute from your Calendar/Date Dimension which is associated with the load frequency. For example daily would be associated a day attribute and weekly with a week attribute.
- **Trigger Attribute Value:** This indicates what value in the attribute would trigger the condition for the dimension or fact to be loaded. For example, an End of Week load frequency is triggered when the associated Week Calendar attribute is equal to Sunday. A % character represents a wild card, in the way a % character represents a wild card in a SQL 'like' clause

CREATING A MANY TO MANY RELATIONSHIP

To create a Many to Many relationship between a Fact Table and Dimension you need to create a Group Dimension and an Intermediary Bridge Fact Table.

For more information about this technique you should refer to the 'Data Warehouse Toolkit' book by Ralph Kimball or visit the following TechNet article which explains the same technique in Microsoft terms. <http://technet.microsoft.com/en-us/library/ms345139.aspx>.

1. First create your Group Dimension, without any dimension attributes.
2. Next create a intermediary Bridge Fact Table, this time without any measures. Associate the Fact table with both the target (of the many to many join) Dimension and the Group Dimension.
3. Open your target (of the many to many join) Fact Table, and associate it only with the Group Dimension.

In the Microsoft example in the above link,

- The Transaction Fact Table is the target Fact Table.
- The Customer Dimension is the Target Dimension.
- The CustomerAccount Fact Table is the Intermediary Bridge Fact Table.
- The Account Dimension is the Group Dimension.

CONFORMED DIMENSIONS

If you are a proponent of the Ralph Kimball school of Data Warehouse design, and the Data Warehouse Bus Architecture, you will subscribe to the concept of conformed dimensions.

Put simply, the concept of conformed dimensions states that Data Marts within an Organisation should share common dimensions so that cross Data Mart/Enterprise/Business process analysis can be delivered.

To support this concept DW Architect contains a number of advanced features. These features can be found on the Advanced Tab of the Dimension editor.

If the Data Mart you are building is conforming with a dimension which is managed by another Data Mart, you will want to define it as a dimension in your design, so you can associate fact tables with it.

You may or may not want to generate the DDL for the conformed dimension on your Data Mart depending on the method of conforming you are using. You might ETL the dimension into your Data Mart before processing your dependant fact tables, or point your ETL process for your fact tables directly at the dimension in the other Data Mart.

If you intend to ETL the conformed dimension into your Data Mart, then you will need to define every attribute of the dimension in your design. On the advanced tab you might also want to specify a custom surrogate key, and custom surrogate key data type if the conformed dimension uses a different standard to the standard used by your Data Mart. You will also want to tick the Conformed indicator so your generation process can identify that conforming code needs to be generated for the dimension.

If you intend to access the dimension directly in its resident Data Mart, then you only need to specify the details of the surrogate key and natural key of the conformed dimension in your design. To do this you would create a new dimension in your design, name it appropriately, and then specify a custom surrogate key, and custom surrogate key data type, on the advanced tab, if the conformed dimension uses a different standard to the standard used by your Data Mart. You would also un-tick, the 'Generate DDL' option on the advanced tab, because you do not need to create the table in your Data Mart. The dimension is now available in the Dimensionality tab of Facts you want to associate to the dimension.

GENERATING CODE

Code generation is executed through the DW Management Console. To start the generation process first:

- Open DW Management Console.
- Open the Configuration file for the environment you are generating code for.

Code must be generated for each environment, as each environment will have different source connections.

The act of generating the code is a simple process, however it depends on having the configuration correct before proceeding. The following things need to be configured correctly before Generation will work correctly.

CONFIGURATION

CONNECTIONS

On the Connections tab, use the configure buttons to create connection strings to your Data Warehouse and Staging Database..

Note! The Databases need to exist on the database server before you can set them as your Data Warehouse and Staging databases.

DATA WAREHOUSE DATABASE CONNECTION STRING

The connection string to your target Data Warehouse database.

STAGING DATABASE CONNECTION STRING

The connection string to your target Staging database.

CONNECTION STRINGS TABLE

The connection strings table allows you to specify environment specific connections strings for the source system connections specified in your DW Architect project. This way you can define different source connections for each environment, i.e. development, test, production etc.

1. In the Name field, type the name of the connection from your project. Note this must match 'Connection Manager Name' in the Connection Manager dialog in your DW Architect Project exactly.
2. Use the configure button to configure your connection string. Alternatively you can type or paste a connection string into the connection string field. To manually set the Server Name and Provider fields, view the code of your connection in the DW Architect project, and use the values of the ServerName and Provider elements.

The standard generation templates use these connection string definitions to generate SQL Server Linked Servers. The Provider element is used to determine the format of the Create command, and the Server Name is used in the command.

CONFIGURATION

The configuration tab is used to set some defaults for Data Warehouse tables.

- **Data Warehouse Database Schema:** Use this field to specify the schema name of Data Warehouse tables in the Data Warehouse database. The default is dbo.
- **Dimension Table Prefix:** The dimension table prefix is applied to the front of the table name of every dimension table in your schema. This allows you to keep your design free of implementation details.
- **Fact Table Prefix:** The dimension table prefix is applied to the front of the table name of every fact table in your schema. This allows you to keep your design free of implementation details.
- **Custom Meta Data:** The custom Meta Data table can be used to create custom meta data that can be used in the generation, deployment or batch execution processes.

GENERATE AND DEPLOY

Set the path that is used to store the code generated from your project. Select the Generate/Deploy tab and use the Browse button next to the "Default Generation Output Path" field to select the directory that generate code will be stored.

This location is also used by the deploy process as the source directory of code to deploy.

GENERATING THE CODE

1. Open the DW Architect Management Console and select the **Execute>Generate** option.
2. Use the browse button beside the **DW Architect Project File Path** field to set the path of the DW Architect Project file (.xmlproj) that you are generating code for. The file will be in the folder which contains the 'Data Warehouse' folder of the project.

3. Click the **Execute** button.

The application will generate the code and write it to the 'Default Generation Output Path' folder. A list of the generated files and their locations will be displayed in a dialog at the end of generation process.

GENERATING CODE VIA THE COMMAND LINE

If you are automating your deploy and release processes, you might want to automate the Code generation.

The Code Generation can be automated by calling a command line version of the Code generation routine. The syntax of the command is:

```
DWArchitect Management Console <config_file_path> <DWArchitect_Project_Path> [<output file path>]
```

For Example:

- "C:\Program Files (x86)\DataInfoLeaders\DW Architect Management Console\DWArchitect Management Console" Generate "E:\Warehouse\Warehouse.cuf" "E:\Warehouse\Warehouse.xmlProj" out.txt

CUSTOM GENERATION ENGINES AND TEMPLATES

DW Architect supports an open API for developers to create custom Generation Engines and Generation Templates. More information can be found in the API guide.

CUSTOM GENERATION TEMPLATES

DW Architect ships with a single XSLT Generation Engine. This engine takes XSLT generation templates as input. There are 14 XSLT generation templates that will produce the entire Data Warehouse solution. Each template generates code for a different aspect of the solution. E.g. Extracts, Dimension Transforms, Link Servers, Data Warehouse DDL etc.

Developers can develop their own XSLT templates to generate custom code, that better suits their target environment. They can choose to use the standard templates, a combination of standard and custom templates or only custom templates.

CUSTOM GENERATION ENGINES

A custom generation engine can be created, if XSLT is not an appropriate way of generating code for a target platform. For example, SSIS supplies an .NET API for generating SSIS packages. A more appropriate method of generating SSIS packages would be to use this API, rather than XSLT. A developer can create a Custom Generation engine which takes the Meta Data defined in the Data Warehouse project and uses the SSIS API to generate the SSIS packages.

DEPLOYING CODE

Code deployment is executed through the DW Management Console. To start the deployment process first:

- Open DW Management Console.
- Open the Configuration file for the environment you are deploying code to.

The act of deploying the code is a simple process, however it depends on having the configuration correct before proceeding. The following things need to be configured correctly before Deployment will work correctly.

CONFIGURATION

CONNECTIONS

On the Connections tab, use the configure buttons to create connection strings to your Data Warehouse and Staging Database..

Note! The Databases need to exist on the database server before you can set them as your Data Warehouse and Staging databases.

DATA WAREHOUSE DATABASE CONNECTION STRING

The connection string to your target Data Warehouse database.

STAGING DATABASE CONNECTION STRING

The connection string to your target Staging database.

GENERATE AND DEPLOY

Set the path that to the folder that contains the generated code from your project. Select the Generate/Deploy tab and use the Browse button next to the "Default Generation Output Path" field to select the directory that generate code is stored in.

This is the location the deploy process uses as the source directory of code to deploy.

DEPLOYING THE CODE

1. Open the DW Architect Management Console and select the **Execute>Deploy** option.
2. Select the kind of Code you wish to Deploy.
3. Click the **Deploy** button.

The application will deploy the code to the Data Warehouse and Staging databases of the target Data Warehouse Server. The code is read and deployed in folder alphabetical order. I.e. All the code in folder 010 Tables, then 020 Views etc. Only code that matches the file extension defined in the manifest for the Deployment provider is deployed. E.g. The 'SQL and DDL Code' deployment provider specifies the .sql file extension, and therefore only attempts to deploy code files with this extension.

A list of the deployed files will be displayed in a dialog during the deployment process. Any errors are returned in the dialog.

DEPLOYING CODE VIA THE COMMAND LINE

If you are automating your deploy and release processes, you might want to automate the Code deployment.

The Code deployment can be automated by calling a command line version of the Code deployment routine. The syntax of the command is:

```
DWArchitect Management Console Deploy <config_file_path> <TargetType> [<outputFile>]
```

For Example:

- "C:\Program Files (x86)\DataInfoLeaders\DW Architect Management Console\DWArchitect Management Console" Deploy "E:\Warehouse\Warehouse.cuf" "SQL & DDL Code" out.txt

CUSTOM DEPLOYMENT ENGINES

DW Architect supports an open API for developers to create custom Deployment Engines. More information can be found in the API guide.

A custom deployment engine can be created, if a deployment engine doesn't exist for your target environment. The standard deployment engine will deploy SQL and DDL to a Relational DBMS. If you have custom code for another platform, you can create a Deployment Provider to deploy that code to that platform.

SOURCE CONTROL AND MIGRATION

DW Architect eases the issues commonly experienced by Data Warehouse projects when attempting to migrate Data Warehouse projects up through development environments.

Because DW Architect projects can be placed under source control just like any other Visual Studio project it is possible manage a DW Architect project just like any other project. Various trunk and branches can be created with different versions of the project, as required for major or minor releases, patches etc.

Once a DW Architect project is migrated into an environment, you can use the command line version of the DW Management Console Generation routine in an automated build routine to generate the Code for that environment. Alternatively you can generate the Code manually through DW Management Console.

We recommend that you back up any data warehouse before applying any DDL changes. You should also inspect and test the changes in lower environments before applying the changes to production. Ensure that the DDL scripts execute correctly and don't cause any unexpected destructive changes.

DATE RANGE TRANSACTION FACT TABLE PATTERN

The Date Range Transaction Fact table pattern is designed to be a high speed load pattern for 'transaction' style fact tables. Transaction fact tables are typically medium to high volume, with a one to one relationship from a fact to a source row, representing the finest grain of the source. Generally these facts have a low volume of change once created, and their dimensionality is static at creation.

The pattern uses a 'Active Partition Period' to define a boundary between active and stable data, and only operates on an active portion of data, reducing the overall volume of data that needs to be processed. The 'Active Partition Period' is the number of days of source data the pattern must process every Batch. For low volume highly volatile facts (e.g. a Sales Order Fact) a large period might be chosen, say 365 days, to ensure all changes to Sales Orders are captured. For high volume, low volatility facts (e.g. Traffic Detectors data) a short number of days is desirable due to the high volume of data being processed for each day, and the unlikelihood of the data changing once captured.

The approach also uses checksums to compare incoming source records against their equivalent record in the fact table. Where the fact table record doesn't exist or there is a difference in checksums, further processing on that row occurs. Where the checksums are the same, the record is immediately discarded. This approach reduces the amount of processing the process needs to do, as it discards unchanged records as soon as possible.

Because the pattern implements its own change capture, and all inserts, updates and deletes are executed in an atomic way, it is simple to recover after failure. Simply restarting the Batch will see it recover and self-heal its target Fact table.

The pattern is a ledger based pattern. This means that rather than over-write existing rows with updates, additional rows are written to represent the new state of the Fact, and the old row is cancelled out with a negating row. In this way, the history of a Fact is kept, while maintaining the integrity of the aggregation of facts. This pattern means that the majority of updates are either bulk inserts or bulk updates. Bulk inserts and bulk updates are a very efficient way of updating the Fact table, and therefore maintaining high performance.

For a cube to process a fact table, it is only necessary for it to select 'current' records as its source. This will mean that additional records generated in a ledger style fact table will not impact the performance of the cube build.

Our Date Range Transaction Fact table pattern is designed to:

- Be High Speed.
- Handle Restart scenarios and be self-recovering. To this end all updates and inserts are atomic.
- Keep a history of changes to facts.

These patterns are appropriate when:

- Each fact record would usually not change. As is the case with Transactional Fact tables.

These patterns are not appropriate when:

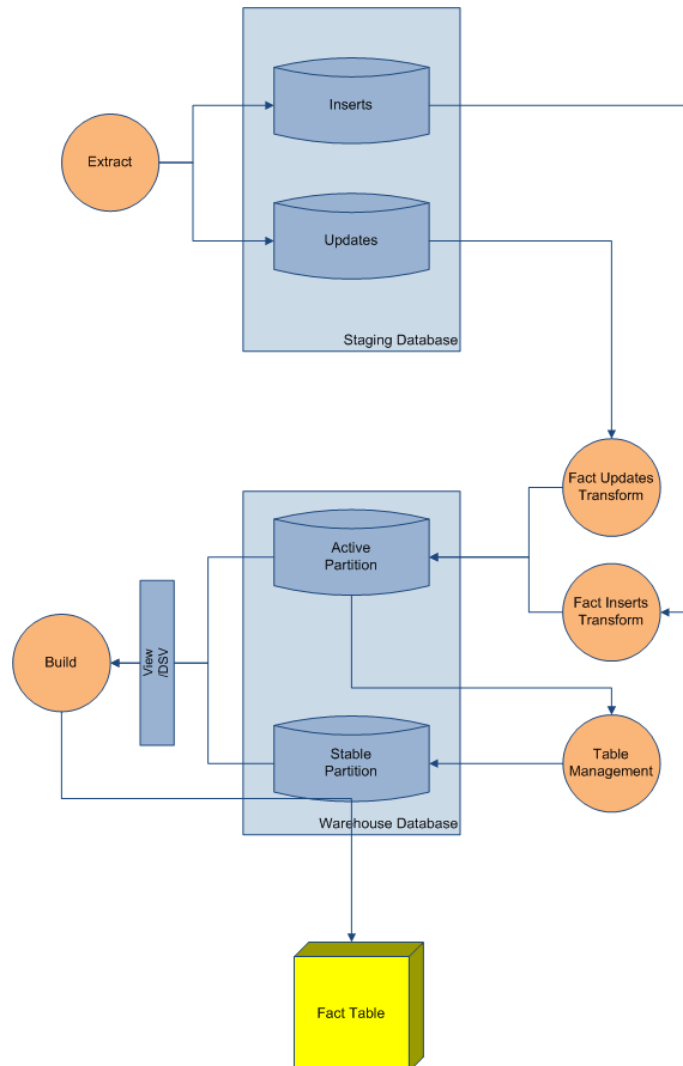
- The fact is expected to have a number of dimensional changes in its life cycle. This is the case with accumulating snapshot Fact Tables. A fact which represents a source record through a set number of steps in its life cycle.
- Very large fact table. The pattern relies on the ability to join on all the surrogate keys, business keys and checksums of each fact table record in the active partition. On very large fact tables this may require an untenable amount of memory.

IMPLEMENTATION

This section describes the various objects that are created and how they interact when a User chooses the Date Range Transaction Fact pattern.

The Meta data required to enable the Date Range Transaction is described in the next section.

The diagram below shows the various objects involved in the Date Range Transaction Fact Pattern.



CONTROL PROPERTIES (NOT SHOWN)

This pattern relies on 3 Control properties:

- **Current_Batch_Effective_Date**, usually the current system date (i.e. The day the Batch is executed)
- **Fact_Active_Partition_Period_Days**. The number of Days of Fact Data that is volatile and is held in the Fact Active partition. Each Fact that has a Transaction Date Range ETL Pattern has its own Control property.
- **Fact_Up_To_Date**. The last day of Data currently held in the Active Partition of the Fact table.

STAGING TABLES

For each staging table identified as the Source of a Date Range Transaction Fact transformation, 2 physical staging tables are created. An 'Updates' staging table to hold source data which falls within the target Fact table's volatile 'Active Partition Period' and is subject to change, and an 'Inserts' table, to hold new source data that has not yet been inserted into the Fact table.

EXTRACT

First an Extract_Start_Date is calculated as:

- $\text{Fact_Up_To_Date} - (\text{Fact_Active_Partition_Period_Days} - 2)$

The Extract procedure runs 2 Extracts.

1. Extract into the 'Update' Staging table. This extracts all data from the source table where the nominated date column is $\geq \text{Extract_Start_Date}$ and $< (\text{Fact_Up_To_Date} + 1 \text{ day})$. This will extract all source data that is still considered volatile to capture any changes that may have occurred in the source data.
2. Extract into the 'Insert' Staging table. This extracts all data from the source table where the nominated date column is $> \text{Fact_Up_To_Date}$ and $< (\text{Batch_Effective_Date} + 1 \text{ Day})$. This will extract all source data that exists in the source table, but has not yet been inserted into the Fact table. It attempts to bring the Fact table up to date with the current system date.

WAREHOUSE TABLE

For each Fact table with a Date Range Transaction Fact transformation defined, 2 physical warehouse tables are created. An Active partition table to hold Fact data which falls within the Fact table's volatile 'Active Partition Period' and is subject to change, and a Stable partition table, which holds Fact data that is considered stable and will no longer change.

FACT UPDATES TRANSFORM

Detecting change and updating Facts is a costly ETL exercise. The Fact Update transformation process only operates on the Active Partition of the Fact data. This is a relatively small subset of all the Fact data. Because the Active Partition is relatively small, the Update process is more efficient. Inserts which are the majority of the change, have been segregated into the Inserts Staging table. Also the size of the Active partition remains relatively stable over time (unless there is a large increase in daily transactions) so the time it takes to make execute the Update remains stable.

The Fact Update Transform takes the data from the Update staging tables, and uses row checksums to compare the existing Fact records to the newly Staged records to detect Inserts, Updates and Deletes. Detected Updates cause a negating row to be inserted into the Fact active partition table to reverse the existing Fact row, and a new row is inserted. Detected Deletions cause a negating row to be inserted into the Fact active partition table to reverse the existing Fact row. Detected Inserts are inserted into the Active partition.

FACT INSERTS TRANSFORM

The Fact Inserts transform is a simple process of taking the data from the Inserts Staging table, transforming it, and inserting it into the Fact table. Every row in the Primary Staging Insert table becomes a new Fact in the Fact table.

VIEW/DSV

The View/ Data Source View (DSV) combines the data in the Active and Stable partition for processing by Cube Build process.

BUILD

Standard Build Cube Job.

DEFINING THE META DATA

FACT TABLE

In DW Architect, to identify a fact table is using the Date Range Transaction Fact table pattern, add a tag named 'DIL_Active_Partition_Period_Days' to the fact table on the Advanced tab. Set the value to the number of days you want to keep in your active (i.e. Volatile) partition.

The Fact table must have an ETL Transformation with an ETL pattern set to 'Transaction Date Range'. Within this Transformation, you must identify one measure as the Date attribute that is used to determine if a row should be kept in the active partition or moved to the stable partition. To identify this measure, edit it through the Transformation tab and add a tag named 'DIL_Date_Attribute'. Set the value to 'DIL_Date_Attribute' also.

STAGING TABLES

The Primary staging table (The staging table at the same grain as the fact) for a Date Ranged Transaction Fact table must have an ETL Extract with a Pattern set to 'Date Range'.

The Staging table must also identify the Fact table it is the source for. To do this add the following tags to the Advanced tab of the Staging table:

Name	Value
DIL_Target_Table_Type	Fact
DIL_Target_Table	<i>The name of the target Fact Table</i>

Within the Extract, you must identify one source column as the Date attribute that is used to determine if a source row is extracted based on the value of the 'DIL_Active_Partition_Period_Days' tag of the Target Fact table. To identify this source column, edit it through the Mapping tab and add a tag named 'DIL_Date_Attribute'. Set the value to 'DIL_Date_Attribute' also.

Any secondary Staging tables that need to be date ranged can be treated in the same way.

DW BATCH MANAGER

A Batch is executed via the command line. To run the Batch you need 5 things

1. DW Architect Management Console installed with a Full (Server), Trial or Community license. The Trial is time restricted, and the Community edition is restricted to only 20 tasks per day. Enough to run a daily Batch for a small Data Warehouse.
2. A configuration file, created through DW Architect Management Console.
3. Deployed Code.
4. A Batch Workflow file.
5. A .bat command file.

WORKFLOW

A Batch workflow consists of an XML file with custom schema. An example of the schema is shown below. This example is used in our Tutorial and Example data warehouses.

```
<Batch_Schedule>
  <Batch_Type>Adventure Works Nightly</Batch_Type>
  <Schedule>
    <Phase>
      <Phase_Name>Extract</Phase_Name>
      <Phase_Provider>StoredProcExecutionProvider</Phase_Provider>
      <Task>
        <Task_Name>Extract%</Task_Name>
        <Task_Provider>StoredProcExecutionProvider</Task_Provider>
      </Task>
    </Phase>
    <Phase>
      <Phase_Name>Derived Extract</Phase_Name>
      <Phase_Provider>StoredProcExecutionProvider</Phase_Provider>
      <Task>
        <Task_Name>DerivedExtract%</Task_Name>
        <Task_Provider>StoredProcExecutionProvider</Task_Provider>
      </Task>
    </Phase>
    <Phase>
      <Phase_Name>Transform_Dim</Phase_Name>
      <Phase_Provider>StoredProcExecutionProvider</Phase_Provider>
      <Task>
        <Task_Name>Transform_Dim%</Task_Name>
        <Task_Provider>StoredProcExecutionProvider</Task_Provider>
      </Task>
    </Phase>
    <Phase>
      <Phase_Name>Transform_Updates</Phase_Name>
      <Phase_Provider>StoredProcExecutionProvider</Phase_Provider>
      <Task>
        <Task_Name>Transform_Update%</Task_Name>
        <Task_Provider>StoredProcExecutionProvider</Task_Provider>
      </Task>
    </Phase>
    <Phase>
      <Phase_Name>Transform_Inserts</Phase_Name>
      <Phase_Provider>StoredProcExecutionProvider</Phase_Provider>
      <Task>
        <Task_Name>Transform_Insert%</Task_Name>
        <Task_Provider>StoredProcExecutionProvider</Task_Provider>
      </Task>
    </Phase>
  </Schedule>
</Batch_Schedule>
```

```

    <Phase_Name>Table Management</Phase_Name>
    <Phase_Provider>StoredProcExecutionProvider</Phase_Provider>
    <Task>
      <Task_Name>Tbl_Mng%</Task_Name>
      <Task_Provider>StoredProcExecutionProvider</Task_Provider>
    </Task>
  </Phase>
</Schedule>
</Batch_Schedule>

```

The Workflow contains a set of phases which in turn contain a set of tasks. Each phase executes in order, and each task in order within the phase. The task name can be wild carded. Each task execution provider should be written to interpret wild card task names into a set of tasks to execute, as is the case with the StoredProcExecutionProvider.

Tasks are executed by Task execution Providers. A Task Execution providers is required for each Server Type/Platform that batch tasks execute on. The StoredProcExecutionProvider will execute stored procedures on SQL Server. The name in the <Task_Provider> element must match the name in the Task Providers manifest file.

CONFIGURATION

There are 2 additional configuration setting that need to be set prior to execution the batch.

- The path to the directory that contains your Batch workflow files. Set the Batch Schedule path on the Batch tab.
- Concurrent Tasks defines the number of batch tasks that can execute concurrently within a phase of the Batch. The value of Concurrent Tasks configuration setting on the Batch tab determines the number of threads that are started to execute tasks in the batch. The batch processor manages these threads, and which tasks are executed on them at which time. Only tasks within a phase can be executed Concurrently. I.e. all Extracts are executed, then all Dimension Transforms, then all Fact Transforms etc. Over time the batch processor will dynamically adjust. It will determines which tasks are the longest running within a phase, and will start these first in order to shorten the overall length of the batch.

EXECUTING THE BATCH

To execute a batch you must run a command line statement, either directly in a command window, or through a .bat command file. The command must be run as 'Run As Administrator' under Vista or above. The syntax of the command is :

```
"DWArchitect Management Console" <config file> <workflow file> New [output text file]
```

An example command is shown below:

```
"C:\Program Files (x86)\DataInfoLeaders\DW Architect Management Console\DWArchitect
Management Console" Batch "C:\EXAMPLE\PRJ\Adventure Works DW\AW_Vista.cuf"
AW_Workflow.xml New out.txt
```

The progress of the batch is written to the console, or to the output text file, if one is specified.

CUSTOM TASK EXECUTION PROVIDERS

DW Architect supports an open API for developers to create custom Task Execution Providers. More information can be found in the API guide.

A custom Task Execution Provider can be created, if a Task Execution Provider doesn't exist for your target environment. The standard StoredProcExecutionProvider will execute Stored Procedures in a Relational DBMS.

SUPPORT

SUPPORT

For support please email support@datainfoleaders.com or visit our website at www.datainfoleaders.com.