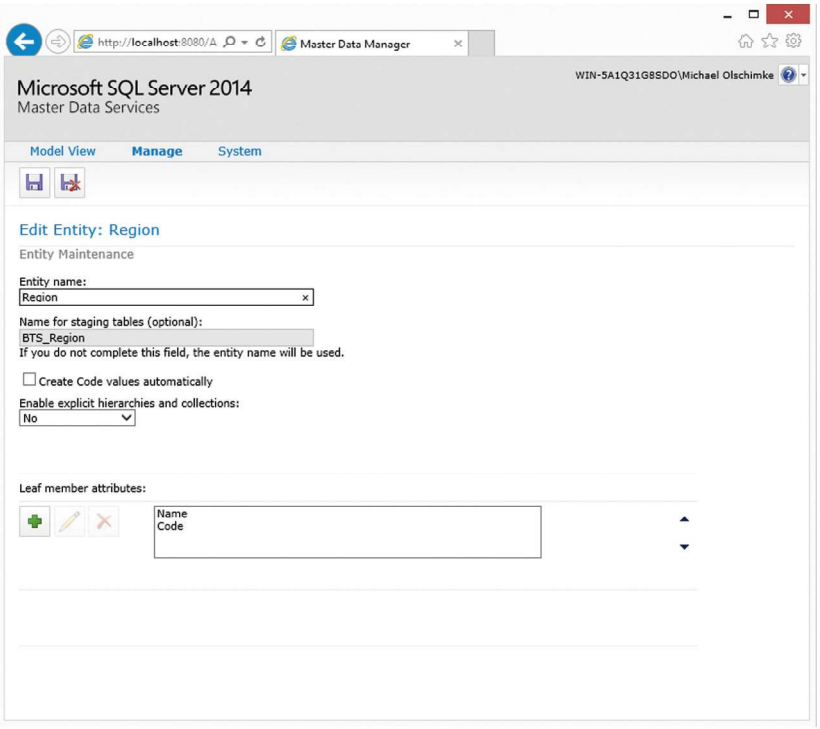

FIGURE 9.17

Edit MDS entities.

Saving the new entity using the save button allows the user to define the attributes of the new entity. In order to do so, select the new entity in the list of entities of the model and select the edit button. The next page (Figure 9.17) allows modifying the entity properties, along with the definition of the entity attributes.

As Figure 9.17 shows, most of the properties that have been set during the entity creation process can be modified, except the name of the staging table. This table was created during the creation of the entity and cannot be modified, as it became an important technical foundation for the entity.

To create new attributes, select the button with the green plus sign, in the **Leaf member attributes** section to the bottom of the page. The next page allows setting up the properties of the new attribute (Figure 9.18).

The page allows selecting one of the attribute types, which have been described in section 9.6.1. In addition, the dialog asks the user to provide an attribute name, and the number of pixels used for displaying the attribute in the Web front-end. Depending on the selected attribute type, additional properties have to be provided by the user to set up the attribute. This includes the data type and length of the value, as shown in Figure 9.17. It is also possible to activate change tracking, a feature that can be used to trigger business rules, which are discussed in the next section.

After saving the new attribute, the same procedure can be used to create additional attributes for the entity. The entity definition is completed when the entity is saved as a whole in the **Edit entity** page.

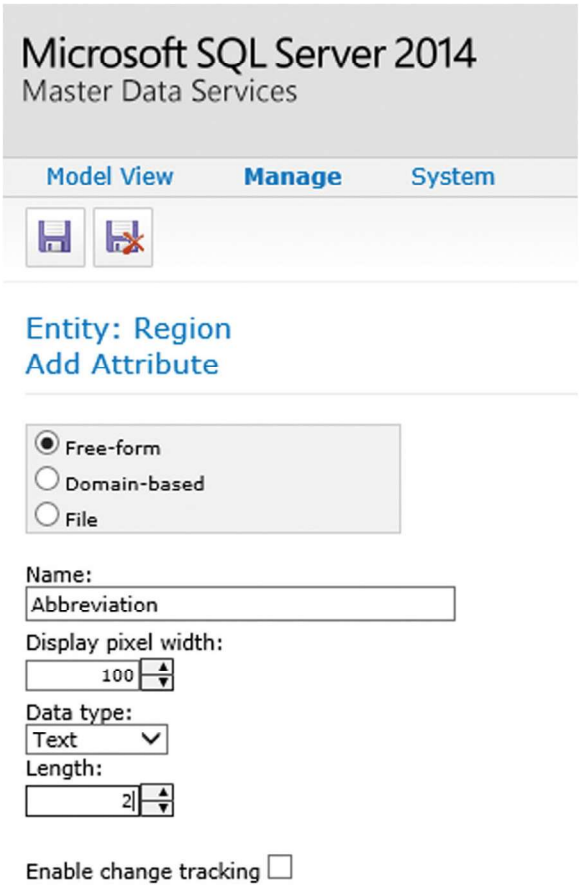

FIGURE 9.18

Add attribute to entity.

9.7.2 CREATING BUSINESS RULES

Business rules are a powerful feature of MDS that help to validate the master data that is created or modified by the business user. They implement logic that is executed when the master data changes and can enforce a given value format for attributes or the existence of data in specified attributes. It is also possible to trigger actions using business rules.

In order to create a business rule in MDS, the **Business Rules** entry in the **Manage** main menu in **System Administration** is used. The following page allows the user to define the entity for which business rules should be created, as shown in Figure 9.19.

The page asks for the model, the entity and the member type to display the current business rules for. In addition, it is possible to filter for business rules that are defined for specific attributes. Note that it is not possible to change the member type for standard entities. Selecting the button with the green plus symbol creates a new business rule on the entity. Figure 9.20 shows the same page as in the previous figure with the newly created business rule.

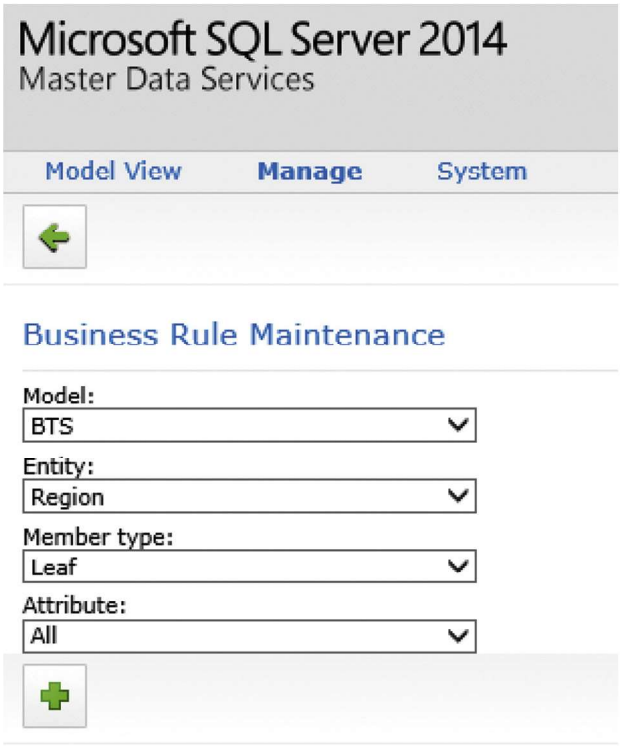

FIGURE 9.19

Business rule maintenance in MDS.

The grid on the page can be used to change the name of the business rule and add a description by double-clicking the respective cell in the grid. It is also possible to modify the selected business rule by selecting the edit button, which is the third button from the left in the icon bar. When creating new business rules, the status is **Rule not defined**, which means that the business rule needs to be modified in order to provide a definition of the business rule.

After selecting the button, the page shown in Figure 9.21 is used to define the business rule.

Business rules are expressed in MDS by two elements:

- **Conditions:** a condition selects the members to apply the business rule to. If no condition is set, the business rule affects all members in the entity.
- **Actions:** the action defines what should happen with the member that is included in the condition. In many cases, it defines what should be checked, for example the length of an attribute, as shown in Figure 9.21.

MDS allows changing the value during modification, set default values, validating values or starting external workflows. When validating values, it is possible to check a number of properties, for example string length, uniqueness, or the input range of numeric values. Thus, business rules present a powerful means to ensure the quality and correctness of master data in MDS.

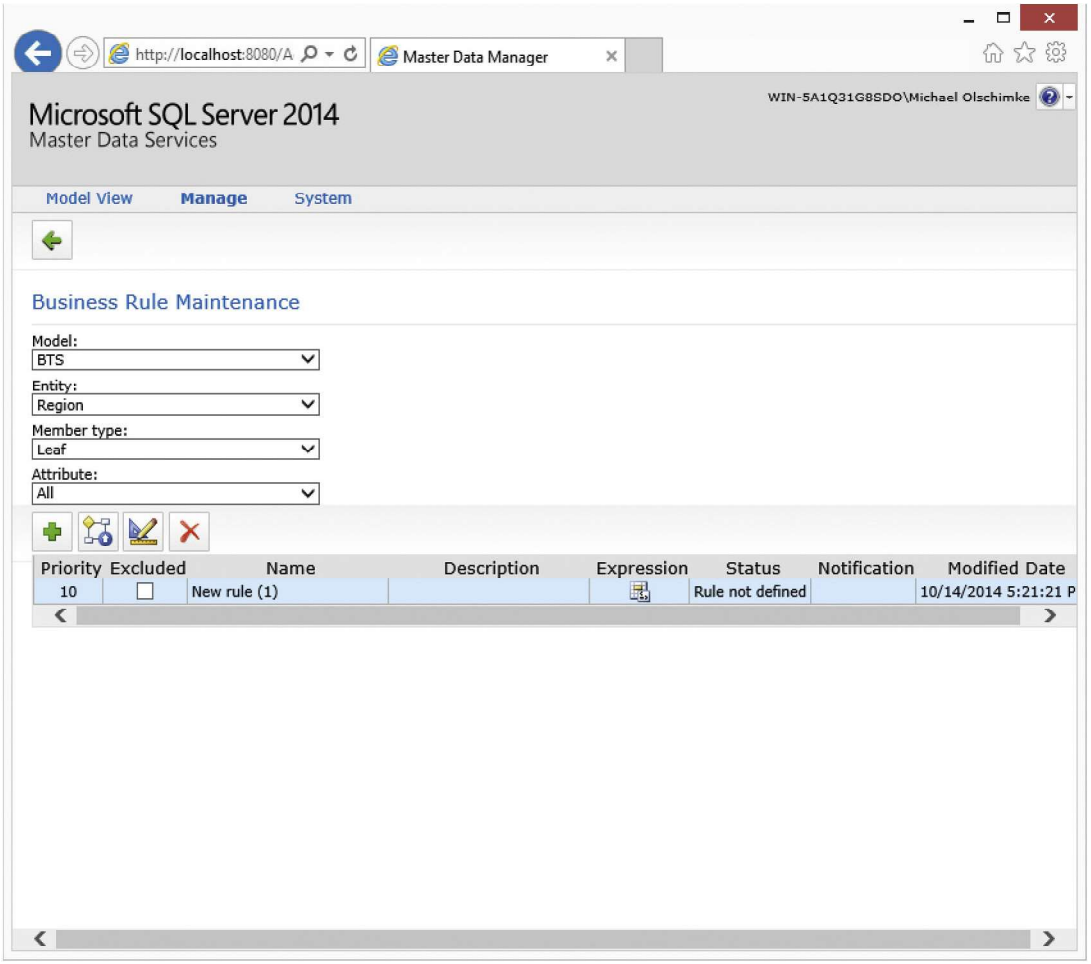

FIGURE 9.20

Managing business rules in MDS.

9.8 IMPORTING A MODEL

As already stated, the companion Web site of this book provides a number of MDS models, which are used throughout the book. In order to load the models to a local MDS installation, it is required to unzip the models and import them into MDS on the server system.

MDS models are imported using the command line tool MDSModelDeploy.exe which is located under `\Program Files\Microsoft SQL Server\120\Master Data Services\Configuration` of the system drive where Microsoft SQL Server 2014 is installed.

The following command is used to identify which MDS installations are available on the server:

```
MDSModelDeploy listservices
```

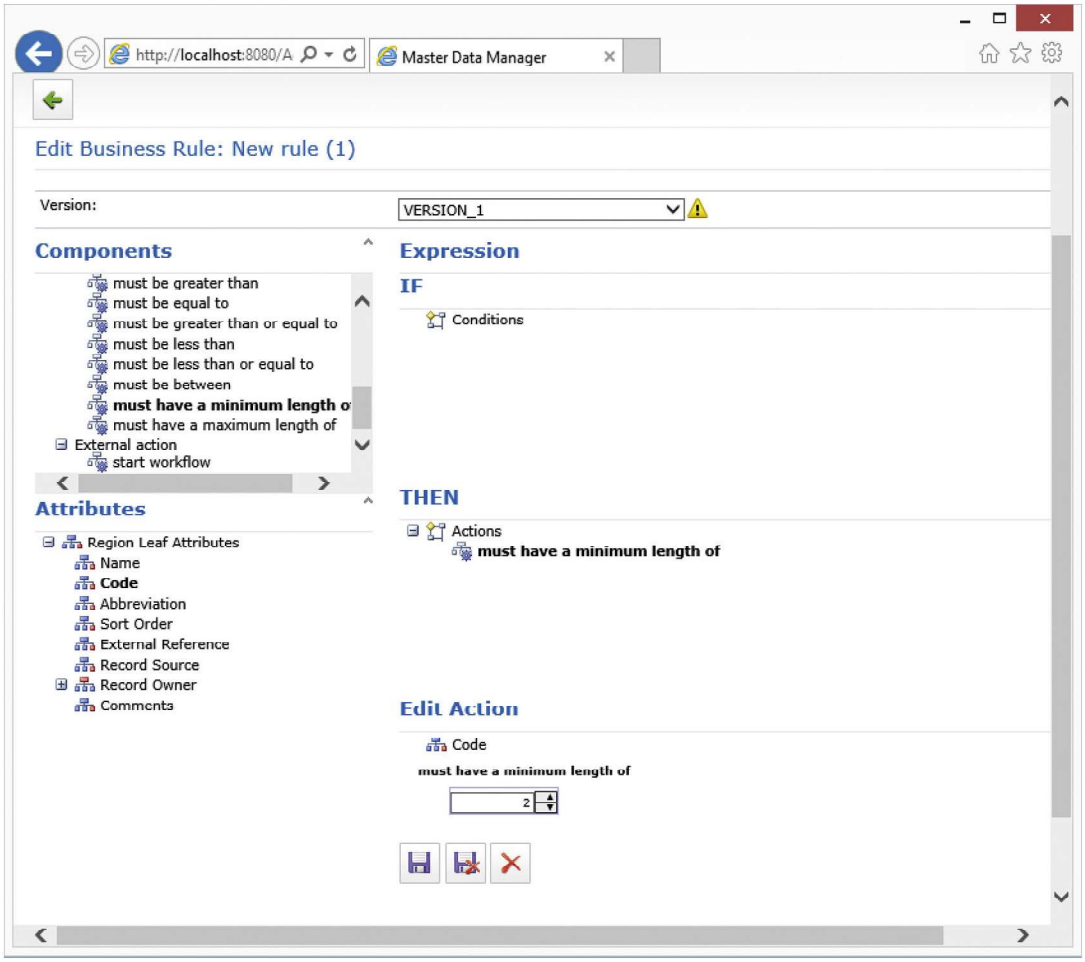

FIGURE 9.21

Modifying a business rule in MDS.

This command returns a list of installations on the local server, including the service name, Web site and virtual path. By default, the first service name is MDS1, which is used throughout this section. The following commands import the master data models of the book to MDS:

```
MDSModelDeploy deployclone -package BTS.xml
MDSModelDeploy deployclone -package FAA.xml
MDSModelDeploy deployclone -package DWH.xml
```

After running these commands, the MDS models are ready to use for the examples in this book. Addititional information is provided at https://msdn.microsoft.com/en-us/library/ff486956.aspx.

9.9 INTEGRATING MDS WITH THE DATA VAULT AND OPERATIONAL SYSTEMS

MDS stores entity members in internal tables within the SQL Server database assigned to MDS. At first glance, the table structure of MDS is complex because the tables have no semantic meaning. It is not possible to quickly identify the internal table for a given entity of a specified model. But it is possible to find out which internal system table holds the data for a given entity by executing the following statement against the MDS database:

```
SELECT
      e.Model_ID
      ,m.Name AS Model_Name
      ,e.Name AS Entity_Name
      ,e.EntityTable
      ,e.StagingBase
  FROM
       mdm.tblEntity e
  LEFT JOIN
       mdm.tblModel m ON (m.ID = e.Model_ID)
  WHERE
    m.Name = 'BTS' AND e.Name = 'Region'
```

This query will return the name of the internal table that holds the master data for the **Region** entity in the **BTS** model. The name is given column 4 **EntityTable**. This information can then be used to truncate the table, which is required when the data becomes corrupt, which sometimes happens during development. The following statement truncates the data:

```
TRUNCATE mdm.tbl_4_76_EN
```

Note that the name of the entity table differs from installation to installation. You should not execute this statement against your MDS database without modifying the table name appropriately.

The described approach should not be used in production. Instead, there are other options for manipulating or accessing the master data in MDS entities. In fact, the managed master data within MDS is of no value for the business as long as it is not integrated with other systems, for example operational systems and the data warehouse. MDS supports two options for integration:

1. **Staging tables [13]:** staging tables are used to load master data from operational systems to MDS. They can also be used to modify or delete entity members programmatically, for example using T-SQL statements or SQL Server Integration Services (SSIS).
2. **Subscription views [13]:** these views are created within MDS and populate the members of an entity. They can be used to integrate master data stored in MDS with operational or analytical systems, called subscribers.

While the first option is to get master data into MDS, the second option is to get master data out of MDS. The data flow of master data is presented in Figure 9.22.

Master data is loaded from operational systems into MDS using staging tables. The figure shows that business users modify and enrich the master data within MDS. The data warehouse then loads the master data from MDS using subscription views into the staging area. However, the same master data, in its enriched form, is also fed back into the operational systems and follows the Total Quality

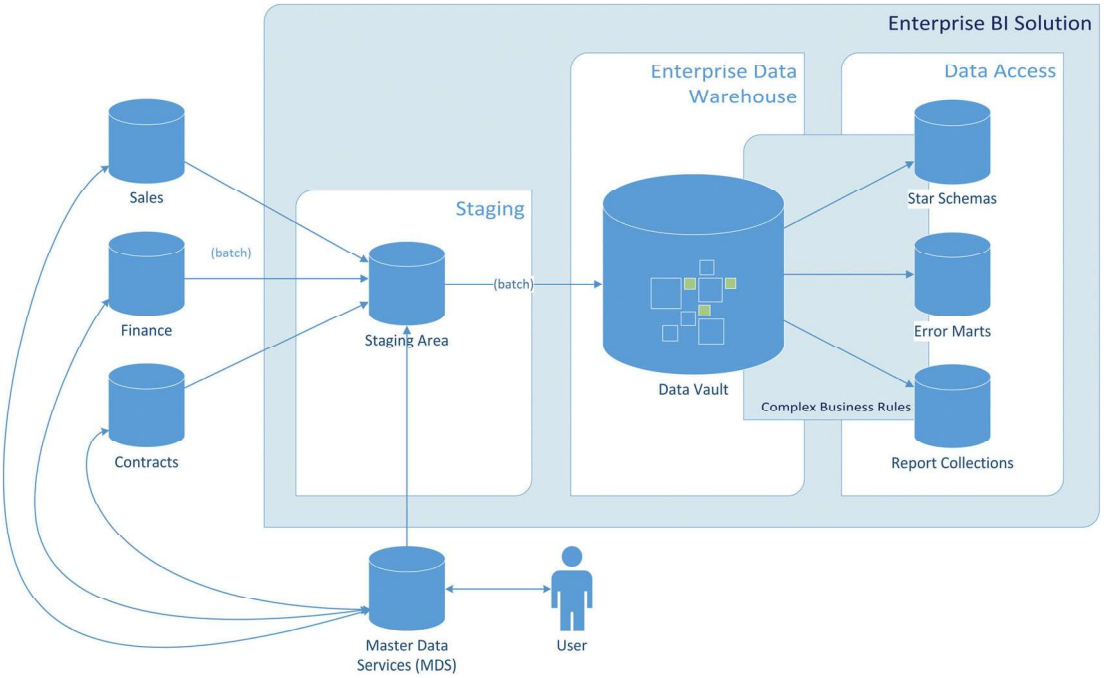

FIGURE 9.22

Closed loop in master data management.

Management (TQM) approach as outlined in section 9.5 and Chapter 3, The Data Vault 2.0 Methodology. Only this closed loop enables business users true self-service because they can now modify the data that is available in both systems, the operational system and the analytical system. In the best case, the master data application should write the master data directly into the Data Vault based EDW layer. However, because the structure of the entities in MDS is fixed and defined by the staging tables and subscription views of MDS, it is treated as just another source system. As a result, the data from MDS is staged and then loaded into reference tables in the Data Vault 2.0 model.

When operational systems deliver master data to MDS, they usually create batches of data within the staging table of MDS. This process is either performed by the operational system itself, if it is aware that MDS is receiving the data and fully supports integration with MDS. In many cases, however, an ETL job, implemented using SSIS, loads the data from the operational source and stages the master data to MDS by creating a new batch in the staging table. Also, there might be multiple source systems that provide master data to MDS, each creating individual batches per load. The batch describes the required modifications to the members within the entity. For example, it describes which members should be added to the entity, which should be updated and which should be deactivated or permanently removed from the entity. Administrative users can apply these batches to MDS by executing them within the Web-based front-end [13] (Figure 9.23).

Each batch is identified within MDS by a batch tag that can be set when the staging table is loaded. The end-user can start selected batches in the **Integration Management** functional area of MDS. The

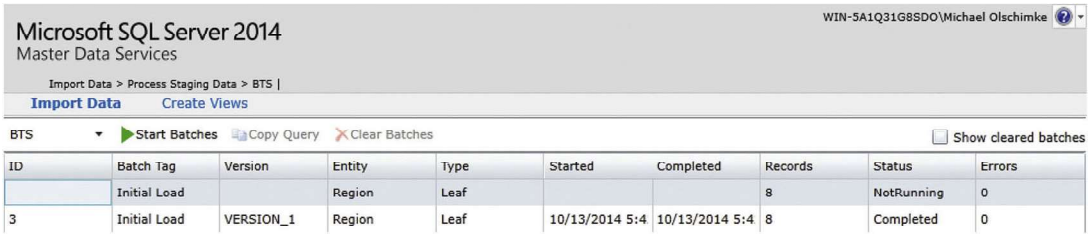

FIGURE 9.23

Manage staging batches in MDS.

Import Data tab serves this purpose, as Figure 9.23 shows. The table on the tab shows the available batches and provides the following columns:

- **ID:** the ID column uniquely identifies a batch by a surrogate key. This key is automatically assigned to the batch by MDS after it has queued to run.
- **Batch Tag:** the batch tag is an alphanumeric identifier that is assigned by the user or process that creates the batch. Examples of such batch tags include "Initial Load" or "CRM 2014-10-08".
- **Version:** this field identifies the entity version to which this batch has been applied.
- **Entity:** the target entity of the batch.
- **Type:** indicates the type of staged data. For the purposes of the book, we will only deal with **Leaf** types, but there are more possible in MDS.
- **Started:** indicates when the staging process has been started.
- **Completed:** indicates when the staging process has been completed.
- **Records:** the number of records within the batch.
- **Status:** indicates the status of the batch, either **NotRunning**, **QueuedToRun**, **Completed** or **Cleared**.
- **Errors:** the number of records that failed to apply. The staging table provides more information on the individual errors on a per-record basis. See section 9.9.1 for more details.

The next sections cover how to create and use staging tables and subscription views in more detail.

9.9.1 STAGE TABLES

While business users can modify master data in MDS using the Web-based front-end or the Microsoft Excel add-in, it is also possible to load new or updated master data from operational systems programmatically, for example using T-SQL or SSIS. It is also possible to delete master data in MDS in the same manner.

The key for programmatic manipulation of master data in MDS is the use of staging tables, which are created within the MDS database when adding new entities. Section 9.7.1 has already covered that it is possible to provide an optional name for the staging table, as shown in Figure 9.24.

The name, in this case **BTS_Region**, can only be set while adding the entity. It is not possible to change the name afterwards. It is a recommended practice to modify the initial name and add the name of the MDS model to the staging table name. The reason for this practice is that MDS uses the provided

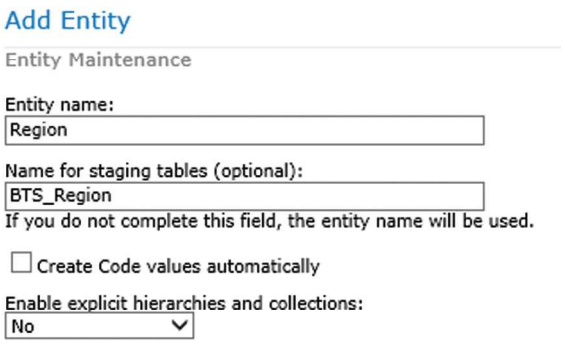

FIGURE 9.24

Providing optional name for staging table.

name and creates staging tables for leaf nodes, consolidated members and relationships. For example, the model distributed on the companion Web site creates many staging tables for leaf notes. By default MDS creates a staging table named **stg.Region_Leaf** for the entity **Region** in the **BTS** model. The issue arises when two different models include an entity called **Region**. In this case, MDS cannot create a staging table with the same name and will warn the user. If the model name is included in the name for the staging tables, this problem cannot arise because all staging table names will be unique.

9.9.1.1 Using T-SQL to Stage Master Data into Microsoft Master Data Services

In order to stage data into MDS using T-SQL only, the following statement can be used to load data into the entity **Region**:

```
INSERT INTO [stg].[BTS_Region_Leaf]
           ([ImportType]
           ,[ImportStatus_ID]
           ,[BatchTag]
           ,[Code]
           ,[Name]
           ,[Abbreviation]
           ,[Sort Order]
           ,[External Reference]
           ,[Record Source]
           ,[Record Owner]
           ,[Comments])
     VALUES
           (0
           ,0
           ,'Initial Load'
           ,'NA'
           ,'North America'
           ,'NA'
           ,1
           ,'http://en.wikipedia.org/wiki/North_America'
           ,'Wikipedia'
           ,'BICC'
           ,'')
```

Table 9.7 Valid ImportType Values [17]

ImportType	Description
0	Create new members. If the staged data is not NULL, existing MDS data will be replaced by staged data. NULL values are ignored. In order to set a NULL value in the target entity, string values in the stage table have to be set to ~NULL~ and number attributes to -98765432101234567890. DateTime attributes have to be set to 5555-11-22T12:34:56.
1	Create new members only. Updating existing MDS data will fail.
2	Create new members and replace existing MDS data with data from the stage table. NULL values in the stage table will overwrite existing MDS values.
3	Deactivate the member with the given Code value in the target entity. The user interface will hide all attributes, hierarchy and collection memberships, and transactions for this member. This action only works if the member is not used as a domain-based attribute value of another member.
4	Delete the member permanently, based on the given Code value. This action deletes all attributes, hierarchy and collection memberships, and the transactions permanently. This action fails if the member is used as a domain-based attribute value of another member.
5	Deactivate the member with the given Code value in the target entity. If the member is used as a domain-based attribute value of another member, this value is set to NULL. This ImportType value works only for leaf members.
6	Delete the member permanently, based on the given Code value. If the member is used as a domain-based attribute value of another member, this value is set to NULL. This ImportType value works only for leaf members.

This statement adds a new member into the staging table for insert. The insert adds the descriptive data for the member and provides some information to the MDS staging process by using one or more of the following MDS columns [17]:

- **ID:** This surrogate key value is assigned by MDS to each member item in the stage table after processing. You should not enter a value in this field.
- **ImportType:** This value determines what will be done if there is already a member in the target entity that has the same **Code** value as the staged entry. Table 9.7 provides a list of valid values for this field.
- **ImportStatus_ID:** This value provides a status of the staging process. When records are inserted into the staging table for processing, this value has to be set to 0. The staging process will use this field to indicate success or failure of the staging process. Table 9.8 provides a list of valid values.

Table 9.8 Valid ImportStatus_ID Values [17]

ImportStatus_ID	Description
0	The entry in the staging table is ready for staging. This value is set by the process that loads data into the staging table.
1	The entry in the staging table was successfully loaded by the staging process. This value is set by MDS after processing.
2	The entry in the staging table was not loaded by the staging process, due to a failure. This value is set by MDS after processing.

- **Batch_ID:** This identifier is assigned automatically by MDS to group staged data into batches. This field is blank if the batch has not been processed.
- **BatchTag:** This is a unique identifier for the batch and is provided by the process that loads the staging table. It is used to execute the batch later and can contain up to 50 characters.
- **ErrorCode:** If the staging process was unable to load the record into the target MDS entity, this field provides an error code that could be used for further analysis.
- **Code:** A unique **Code** value that is used to identify the member in the target entity. This is also required when updating members.
- **Name:** The name of the member.
- **NewCode:** This is only used when updating the **Code** value of the target member.

The following table lists the values which are valid for the **ImportType** field [17]:

The statement at the beginning of this section has staged a record that will eventually insert a new member into the target entity because it used **ImportType** 0. Providing an **ImportStatus_ID** of 0 indicates that the record is ready for staging and the batch tag **Initial Load** is used by the business user to execute the batch in the Web front-end. Similarly, other import types can be executed, for example deactivating members in the target MDS entity. For that purpose, the following statement, which uses **ImportType** 3, can be used to deactivate the created member for North America (NA):

```
INSERT INTO [stg].[BTS_Region_Leaf]
           ([ImportType]
           ,[ImportStatus_ID]
           ,[BatchTag]
           ,[Code])
     VALUES
           (3
           ,0
           ,'Cleanup'
           ,'NA'
)
```

This statement sets the import type and the intial **ImportStatus_ID**. The batch tag **Cleanup** is used to group multiple stage records into one batch that can be executed at once. The **Code** value identifies the entity member that should be deactivated.

In order to execute the batch, an end-user, such as the MDS administrator, can run the batch from the Web front-end, as described in section 9.6.5. However, it is also possible to execute the batch programmatically, by calling a stored procedure in the MDS database:

```
EXEC MDS.stg.udp_BTS_Region_Leaf @VersionName = 'VERSION_1', @LogFlag = 1, @BatchTag =
'Initial Load'
```

There is one stored procedure for each MDS entity. This stored procedure executes the staged data against the target entity. The procedure takes the following parameters [18]:

- **VersionName:** The name of the version that should be updated. Depending on the SQL Server collation setting, this value might be case-sensitive or not.
- **LogFlag:** Determines if the staging process should log transactions. A value of 0 indicates that transactions should not be logged, and a value of 1 indicates that transactions should be logged.
- **BatchTag:** The name of the batch that should be processed by the staging process.

After the batch has been processed, either by manually executing the staging process in the Web front-end or programmatically by calling this stored procedure, the **ImportStatus_ID** field in the stage table is updated from 0 to one of the following values:

If an error has occurred, the **ImportStatus_ID** is set by the staging procedure to the value 2 and the **ErrorCode** is set. In addition, the **Batch_ID** field is updated by a surrogate number that uniquely identifies the batch within MDS.

9.9.1.2 Using SQL Server Integration Services to Stage Master Data into Microsoft Master Data Services

The process shown in the last section can be implemented in SQL Server Integration Services (SSIS) in a similar manner:

Figure 9.25 shows the control flow which consists of the following tasks:

- **Stage BTS_Region (Data Flow Task):** this data flow loads the data from a CSV source (or any operational database) into the MDS staging table, creating a batch as described in the previous section.
- **Initiate Staging Process (Execute SQL Task):** this task calls the stored procedure to execute the batch.

The control flow is intended to load initial data. It is required to modify both the control flow and the data flow to support inserts when master data is already present in the MDS entity; to stage updates to this master data; and to stage deletes. The best approach to support updates and deletes is to modify the provided data flow **Stage BTS_Region** to support inserts and updates at the same time and add an additional data flow to support deletes. We outline a solution at the end of this chapter.

The tasks in the control flow follow the same approach as described in the previous section. The data flow **Stage BTS_Region** is presented in more detail in Figure 9.26.

While this data flow is very basic, it performs all the actions that are required to load the data from an operational system into the MDS staging table. The following components are included in the data flow:

- **BTS_Region (Flat File Source):** this component sources the data from the CSV file *BTS_Region.csv*, provided on the companion Web site. It defines the structure of the file.
- **Add Staging Columns (Derived Column):** this component adds the additional columns required by the staging table.
- **MDS (OLE DB Destination):** this component writes the data in the data flow into the target stage table.

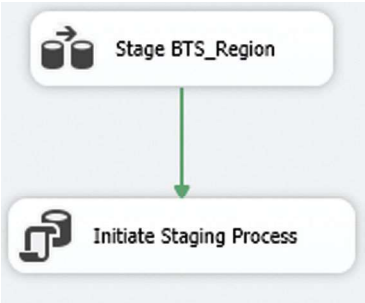

FIGURE 9.25

Control flow for staging master data.

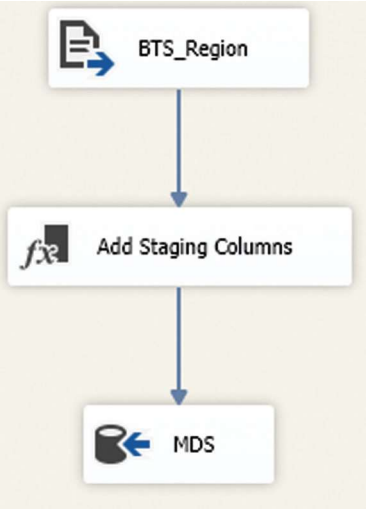

FIGURE 9.26

Data flow for staging master data.

Because the target (as implemented in this book) expects Unicode data, it is required that the data in the data flow use a UTF-8 code page. The package on the companion Web site ensures this by modifying the flat file source **BTS_Region** to provide only text columns that use the data type **Unicode string [DT_WSTR]**. In addition, the code page on the **General** tab has been modified to **65001 (UTF-8)**. Figure 9.27 shows the configuration of columns to meet the requirements of the staging table.

Note that the required data type depends on the code page used by MDS. This setting might differ and requires another data type in the source. Also, there are other ways to change the data type once it has been sourced from the source file or source table: it is also possible to use a **Data Conversion** component to modify the code page.

The component **Add Staging Columns** adds the derived columns which are required by the staging table.

Figure 9.28 shows how the following columns are added to the data flow:

- **ImportType:** setting the import type to 0 is a good practice that works for both the initial load and subsequent loads.
- **ImportStatus_ID:** as described in the previous section, this value is set to 0 to indicate data that has to be loaded from the staging table.
- **BatchTag:** this value is set to Initial Load to identify the loaded master data of the initial load as a group.

Because the data in this data flow loads the regions entity in the BTS model through the **BTS_Region_Leaf** staging table, the OLE DB destination is set to the appropriate target, as Figure 9.29 shows.

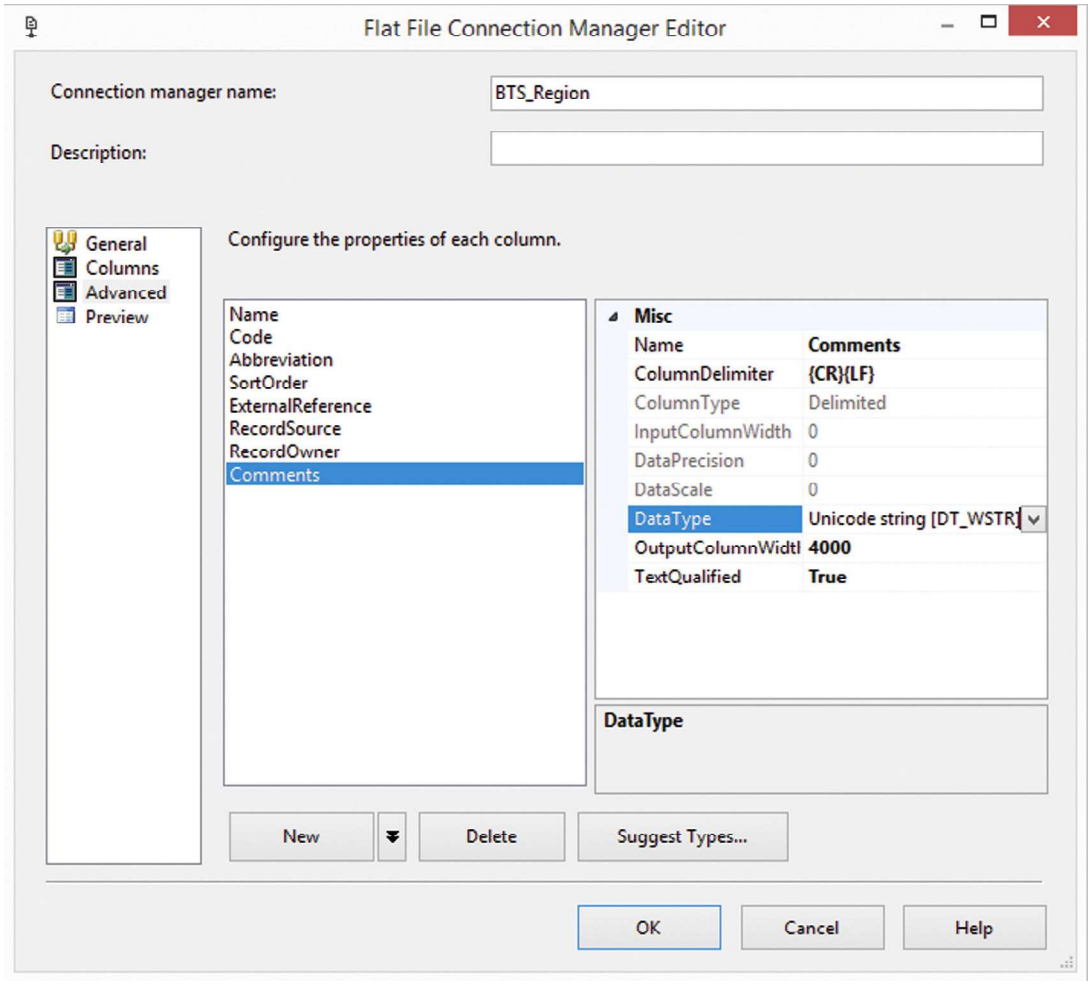

FIGURE 9.27

Setting up the source data types in the flat file connection manager editor.

The name of the destination table is set to **[stg].[BTS_Region_Leaf],** which is the staging table of the **Region** entity in the **BTS** model. All other values on this tab are left as they are. The **Mappings** tab is used to map the columns in the data flow to the columns in the destination.

Figure 9.30 shows this mapping in more detail. Note that the destination columns **NewCode** and **ErrorCode** have no input column, because **NewCode** is only required during updates (and only if the code of the entity member should be changed) and the **ErrorCode** column is used by MDS to provide any error codes during the staging process.

The control flow in Figure 9.25 included another task to initiate the staging process. This task calls the stored procedure that is responsible for loading the master data in the stage table into the MDS

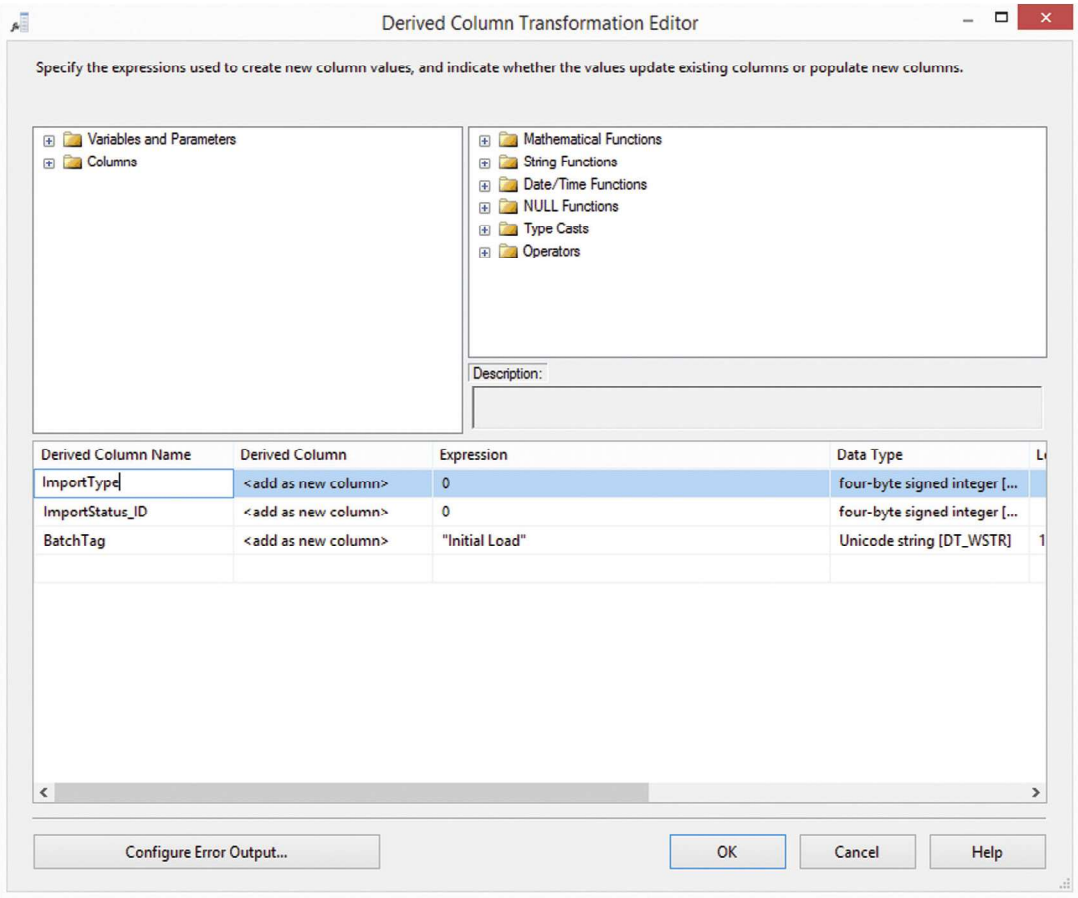

FIGURE 9.28

Adding stage columns in the derived column transformation editor.

entity. This task is executed just after the stage table has been populated with instructions (insert, update, delete). Figure 9.31 shows how this task is configured.

The SQLStatement property of the task is set to the statement that was used in the previous section to initiate the staging process:

```
EXEC MDS.stg.udp_BTS_Region_Leaf @VersionName = 'VERSION_1', @LogFlag = 1, @BatchTag =
'Initial Load'
```

This task completes the basic SSIS control flow for loading master data from operational systems to MDS. In order to use this control flow in production, it needs to be extended in two ways:

- **Support updates and deletes:** in order to support updates and deletes, two additional data flows should be implemented to separate the loading processes from each other. Detecting updates

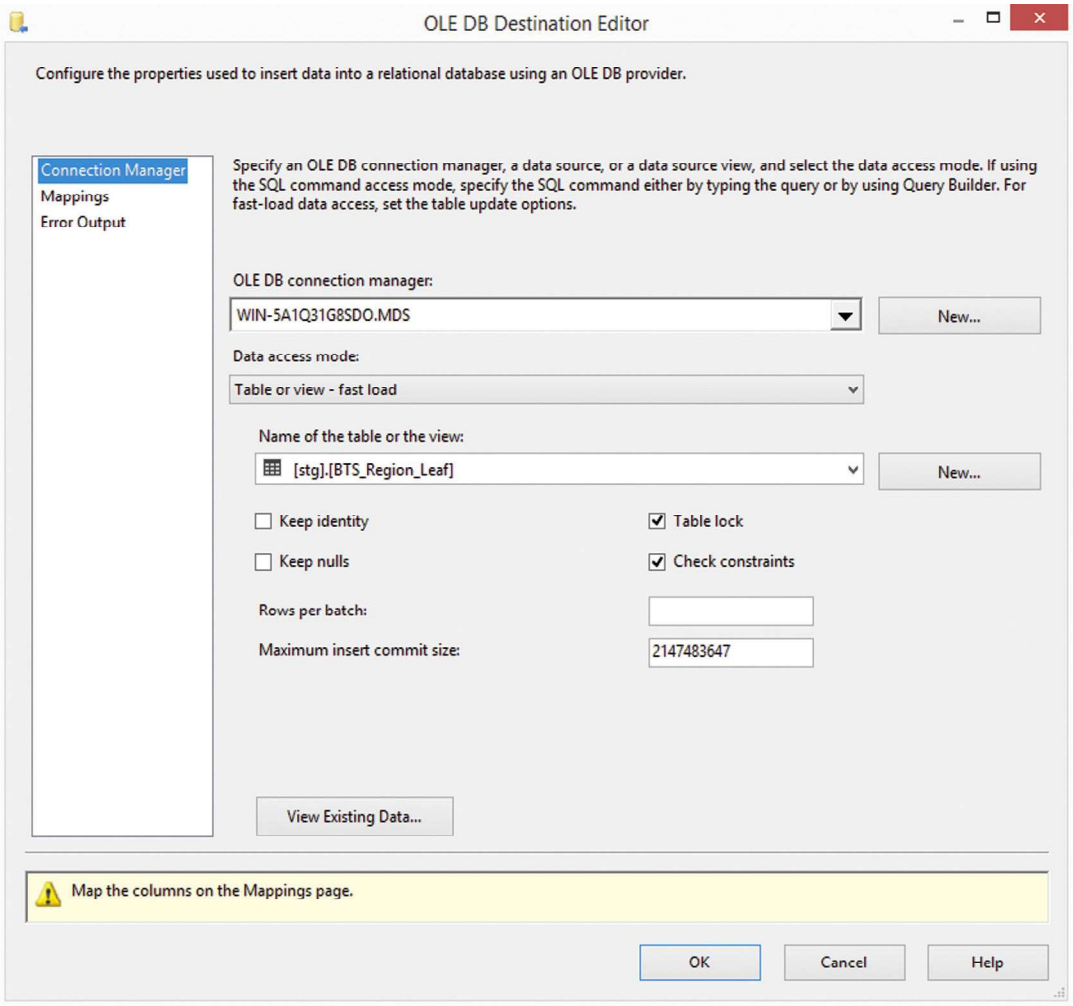

FIGURE 9.29

Setting up the destination in the OLE DB destination editor.

and deletes requires a lookup into the master data in the target entity by using the appropriate subscription view. If there is already a member with the same code as in the source, it should be updated.

- **Log errors to the Error Mart:** this needs to be implemented in the OLE DB destination component, but also in the data flow itself.

Because each source system should be loaded independently from each other, and MDS entities might be sourced from multiple source systems, the data flow should use an **ImportType** setting that

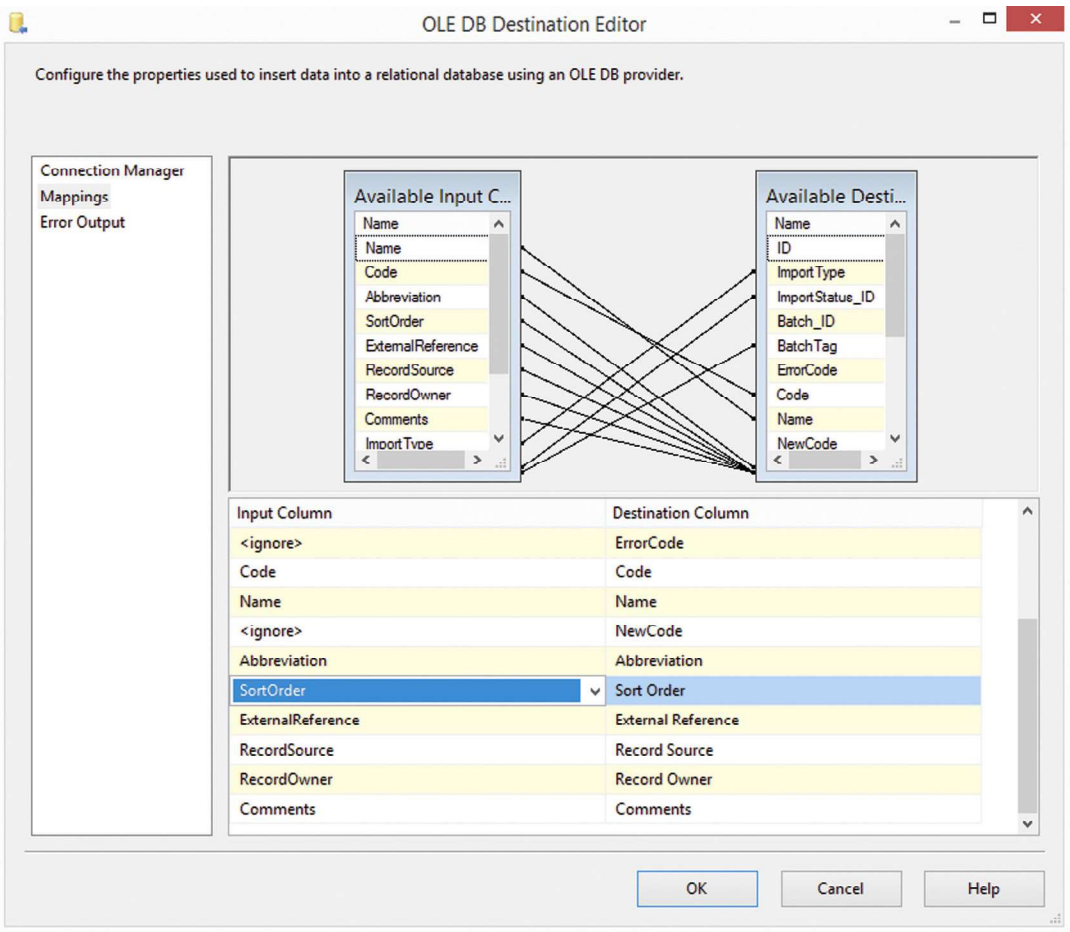

FIGURE 9.30

Mapping columns to destination in the OLE DB destination editor.

allows updating only some of the columns, which is the case with the import type 0: if one of the staging columns is set to a NULL value, the column value is ignored in the staging process. Therefore, only those values are overwritten which are not NULL. Also, not all columns in the MDS entity are sourced from operational source systems. For example, the **sort order** is often set in MDS itself. Using an import type of 2 requires this value to be set in the stage table. Import type 0 allows setting the column to NULL and it will be ignored in the staging process. Therefore, setting **ImportType** to 0 is the preferred method in most cases.

Deleted record detection often requires a lookup to the target entity using the appropriate subscription view, at least if the source system does not provide an audit log that can be used to detect deletes or if the source system does not provide a delta where deletes are marked. If the source

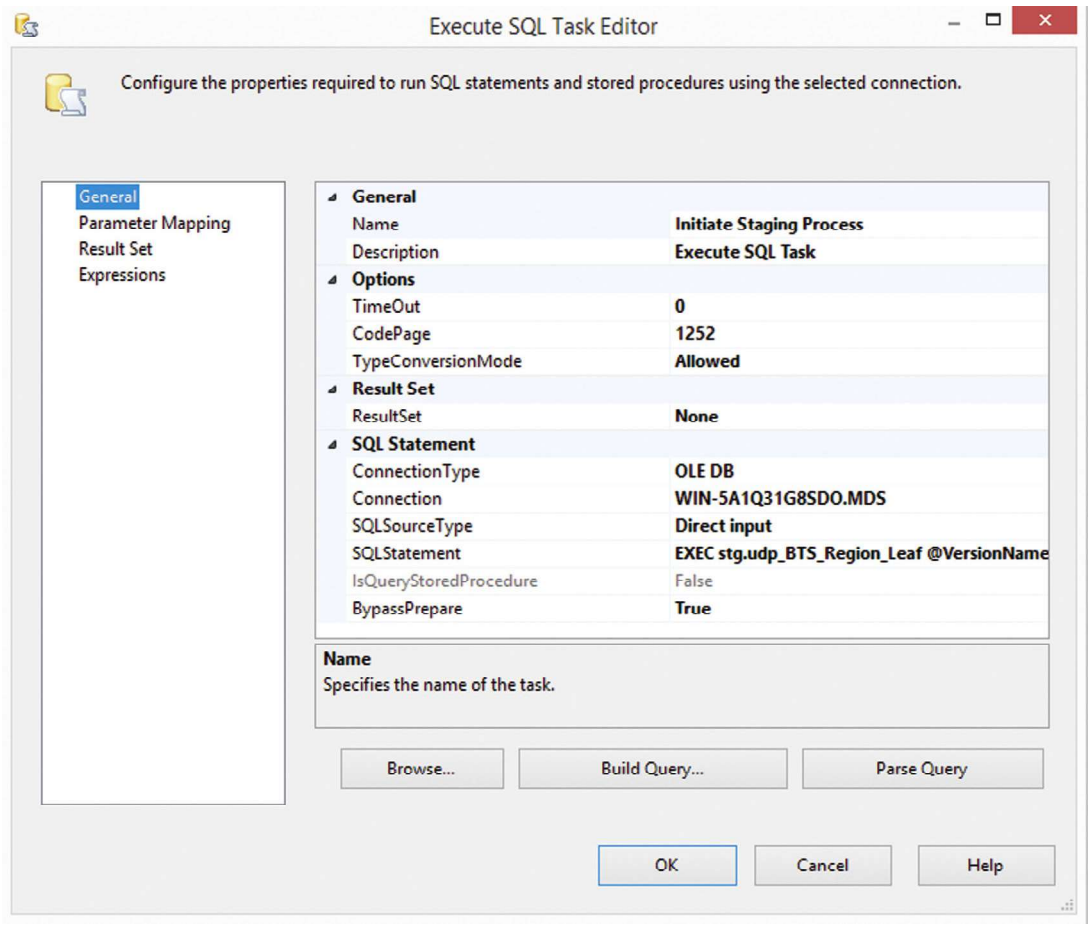

FIGURE 9.31

Setting up the stored procedure.

system provides only full loads of the master data currently used by the operational system, a lookup is required. If the target MDS entity has a member that is not available in the source, the member's **Code** value should be added to the staging table and marked as a delete or deactivation operation by using an import type value between 3 and 6, depending on whether the member in the target entity should be permanently deleted or only deactivated in the user interface and if this operation should also be performed when the member is used as a domain-based attribute value in another entity. The decision how to implement the actual loading of master data depends on these circumstances.

Chapter 10, Metadata Management, shows a modified version of this package that will log errors to the Error Mart.

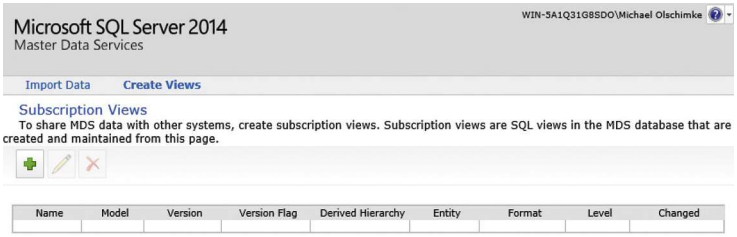

FIGURE 9.32

List available subscription views in MDS.

9.9.2 SUBSCRIPTION VIEWS

As already mentioned in the introduction to section 9.9, subscription views are used to provide master data to external systems, which are called master data subscribers. Thus, the subscription view serves as a tool of master data integration. A subscription view publishes the master data of one entity in the given version. From a technical perspective, the subscription view is just a T-SQL view that populates the data from the internal entity tables covered in section 9.9. However, it is not possible to create the subscription view in T-SQL. Instead, the subscription view is created in the MDS Web front-end. Administrative users use the functional area **Integration Management** for this purpose. The available subscription views are shown on the tab labeled **Create Views** (Figure 9.32).

In order to create a new subscription view, the first button, which shows a green plus button, is used. Figure 9.33 shows the page that follows this action and which is used to set up the subscription view to be added.

The page allows the following options for the new subscription view:

- **Subscription view name:** the name of the new subscription view. Similar to stage tables, the name of the view should be unique throughout MDS. Therefore, the name should include the name of the model. We use the following naming convention throughout this book: <Model name>_<Entity name>_<Subscriber name>.
- **Model:** the model of the entity to be published. The value is set using a combo box that provides a list of available models.
- **Entity:** the entity to be published by the subscription view. A combo box also provides this value.
- **Derived hierarchy:** it is also possible to publish a derived hierarchy. However, for data warehousing, it is often more applicable to publish the individual levels of the hierarchy on its own. This way, the normalized master data becomes available to the data warehouse.
- **Version:** the version of the entity to be published. Each subscription view is created for specific subscribers, which might require an older version of the entity.
- **Version flag:** instead of publishing a specific version, it is also possible to flag a specific version for publication. The advantage of the version flag is that it can be changed when a version gets locked. Therefore, the subscription views that publish the data for a version flag don't need to be updated [19].
- **Format:** for entities, only the option **Leaf members** is available. There are more options available for derived hierarchies, but this is out of the scope for this book.
- **Level:** the level is used for derived hierarchies to indicate the hierarchy level to be published.

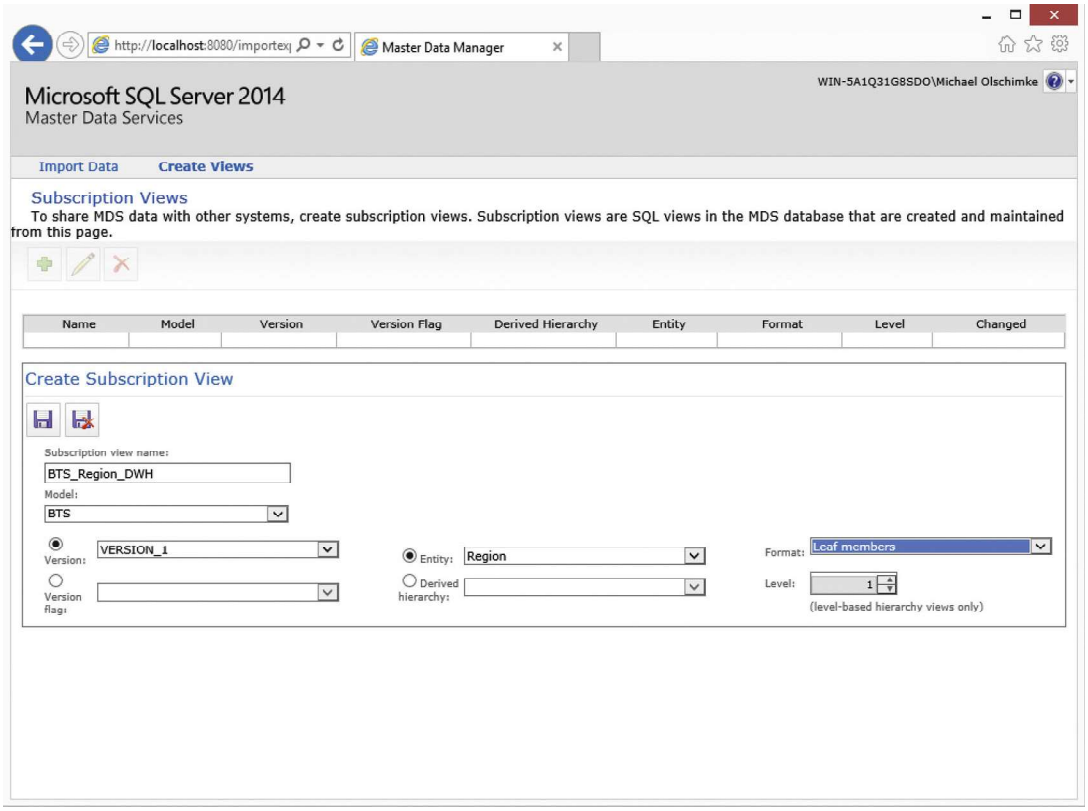

FIGURE 9.33

Create subscription view.

The version is actually an important property for setting up the subscription view. Consider the following issue: multiple operational systems are using the same entity on airport type. One day, the entity is modified by changing the members within the entity to reflect changes in the definition of an airport type (Table 9.9).

Instead of using the two members in Table 9.9, the entity is modified and uses the members in Table 9.10.

However, this modification requires that some operational systems, which subscribe to this entity, be modified as well, in order to make sure that they can actually work with the new members. For

Table 9.9 Airport Types on Day One

Code	Name	Abbreviation	Sort Order
1	Public	P	1
2	Military	M	2

Table 9.10 Airport Types on Day Two

Code	Name	Abbreviation	Sort Order
1	Military	M	1
2	Public Hub	H	2
3	Public Regional	R	3

example, an operational system might have some business logic built in that distinguishes between public and military airports only and does not support a third airport type. It might even have hard-coded the abbreviations in its business logic. Note that abbreviation P is gone after the modification. To support the third airport type, software developers have to modify the business logic, which also requires testing the modified subscribing application. In some cases, it is not possible to rollout the change to all subscribers at the same time. Instead, some subscribers are updated, and some expect the old master data. Providing both versions of the master data entity can mitigate this problem. In order to cope with this issue, multiple versions of the master data in MDS are created and maintained, as described in section 9.6.4. One subscription view is created per entity version and subscriber. The view returns the master data from the selected entity version to the associated subscriber. This maximizes the independence between subscribing applications because it is possible to set the version that is published to the subscribing application individually.

The T-SQL view is available in the MDS database as soon as the **Save** button is selected on the page. In order to retrieve the master data from the Region entity in the BTS model, the following T-SQL statement can be used:

```
SELECT [ID]
      ,[MUID]
      ,[VersionName]
      ,[VersionNumber]
      ,[VersionFlag]
      ,[Name]
      ,[Code]
      ,[ChangeTrackingMask]
      ,[Abbreviation]
      ,[Sort Order]
      ,[External Reference]
      ,[Record Source]
      ,[Record Owner_Code]
      ,[Record Owner_Name]
      ,[Record Owner_ID]
      ,[Comments]
      ,[EnterDateTime]
      ,[EnterUserName]
      ,[EnterVersionNumber]
      ,[LastChgDateTime]
      ,[LastChgUserName]
      ,[LastChgVersionNumber]
      ,[ValidationStatus]
  FROM [MDS].[mdm].[BTS_Region_DWH]
```

The above statement uses the attribute names that have been set up during entity creation. It is not possible to modify these names when creating subscription views. Remember that they include spaces to improve the usability for business users when they modify master data in the Web front-end or using the Microsoft Excel add-in. T-SQL supports spaces in column names by enclosing the column names with square brackets.

Note that the domain-based attributes are translated into multiple columns. For example, the **record owner** column shows that three columns are created:

- **Record Owner_Code:** this column provides the **code** value of the member that is referenced in the domain-based attribute.
- **Record Owner_Name:** this column provides the name of the member that is referenced in the domain-based attribute.
- **Record Owner_ID:** the internal sequence number that identifies the member which is referenced in the domain-based attribute.

Each domain-based attribute will have a **code**, a **name** and an **ID** column. In addition to the attributes from the entity and these additional columns from domain-based attributes, the subscription view populates metadata columns that provide useful information. The following columns are added [13]:

- **ID:** an internal sequence number that identifies the member. This column is not well documented by Microsoft.
- **MUID:** an internal GUID that identifies the member. This column is not well documented by Microsoft.
- **VersionName:** this column indicates the name of the member's version. This value never changes if the subscription view is based on a given version.
- **VersionNumber:** the number of the member's version. This value never changes if the subscription view is based on a given version.
- **VersionFlag:** this column indicates the current version flag for the displayed member. If the view is based on a version flag, this value will never change. Instead, the version name and number will be updated.
- **ChangeTrackingMask:** this column is used when change tracking is enabled for attributes in this entity. It is not well documented by Microsoft.
- **EnterDateTime:** the date and time when the member was created in the entity.
- **EnterUserName:** the domain and name of the user who has created the member in the entity.
- **EnterVersionNumber:** this column indicates the number of the initial version of the member.
- **LastChgDateTime:** the data and time when the member was updated the last time.
- **LastChgUserName:** the domain and name of the user who has modified the member the last time.
- **LastChgVersionNumber:** this column indicates the number of the version this member was last changed.
- **ValidationStatus:** indicates if the business rules have been completed without any errors. Possible values are listed in Table 9.11.

Derived hierarchies should not be published for data warehousing using the **derived hierarchy** option of this page. This option hides some useful information, such as the validation status, but denormalizes some attributes from domain-based attributes. However, the best place for this denormalization

Table 9.11 Validation Status [20]

Validation Status	Description
Waiting to be validated	This validation status is reserved for new members and indicates that they have not been validated yet.
Waiting to be revalidated	The member values or the associated business rules have changed and the members await revalidation.
Validation succeeded	The members have been successfully validated. There were no errors.
Validation failed	The members have been validated, but not all business rules have passed. At least one business rule has raised an error that needs to be fixed before the data can be committed.
Waiting for dependent member revalidation	This validation status is reserved for consolidated members. In order to validate them, the leaf members have to be validated first.

to take place is when the dimension is being built on the way from the Raw Data Vault to the Business Vault or Information Mart. Therefore, it is recommended to export the levels on an individual basis using subscription views on the leaf member level.

Chapter 11, Data Extraction, will demonstrate how to load master data into Data Vault entities, including reference tables, using the subscription views created within this chapter.

REFERENCES

[1] Alex Berson & Larry Dubov: "Master Data Management and Data Governance, Second Edition", p. 5.
[2] Joy Mundy and Warren Thornthwaite: "The Microsoft Data Warehouse Toolkit, Second Edition," pp. 165, 165f.
[3] http://searchdatamanagement.techtarget.com/definition/data-management.
[4] DAMA International: "DAMA-DMBOK Guide: The Data Guide to the Data Management Body of Knowledge," p. 4f.
[5] Duane Nickull: "A Modeling Methodology to Harmonize Disparate Data Models".
[6] Jeremy Kashel, Tim Kent, Martyn Bullerwell: "Microsoft SQL Server 2008 R2 Master Data Services," pp. 8, 13ff, 34, 37, 38f, 40f, 42, 37f, 126f, 132, 141, 94, 104, 104f.
[7] http://code7700.com/thrust_v_power.html
[8] Alex Berson & Larry Dubov: "Master Data Management and Data Governance, Second Edition", p. 7ff.
[9] Reeves, Laura L.: "A Manager's Guide to Data Warehousing", pp. 241–242.
[10] http://www.transtats.bts.gov/homedrillchart.asp
[11] Paul E. McMahon: "Integrating CMMI and Agile Development," pp. 60f, 61f.
[12] Mary Beth Chrissis, et al. "CMMI for Development," p. 133.
[13] Tyler Graham: "Microsoft SQL Server 2012: Master Data Services," Second Edition, pp. 71, 98, 115, 326ff, 327.
[14] http://msdn.microsoft.com/en-us/library/ff486954.aspx.
[15] http://msdn.microsoft.com/en-us/library/ff487062.aspx.
[16] http://msdn.microsoft.com/en-us/library/ff487016.aspx.
[17] http://msdn.microsoft.com/en-us/library/ee633854.aspx.
[18] http://msdn.microsoft.com/en-us/library/hh231028.aspx.
[19] http://msdn.microsoft.com/en-us/library/ff487013.aspx.
[20] https://msdn.microsoft.com/en-us/library/gg471534.aspx.

CHAPTER

METADATA MANAGEMENT 10

In many of our projects, clients ask us how to track metadata, "the data about data." While there are some solutions available to track metadata in data warehouse environments, project teams often work with manual spreadsheets to maintain the metadata required to define the data warehouse, including its artifacts, such as relational tables, information marts, ETL (extract, transform, load) flows, requirements, business rules, etc.

But metadata management, from a Data Vault perspective, also requires the capture of metrics about process execution and errors. This chapter covers the concepts behind capturing metadata, process metrics and error information.

10.1 WHAT IS METADATA?

Our initial definition of metadata as "data about data" is not very helpful. An alternative to this definition is that metadata is all data about other data that is *"needed to promote its administration and use"* [1]. Both definitions are very common definitions from the information technology space but don't provide us with a useful and understandable definition for data warehousing. Another definition, often used in data warehousing, distinguishes metadata by the following two categories [2]:

- **Back room metadata:** this metadata is process related and describes the extraction, cleaning and loading processes. Its main purpose is to help the database administrator (DBA) or the data warehouse team to load the data into the data warehouse. It also helps end-users to understand where data comes from.
- **Front room metadata:** this metadata is more descriptive and is used in query tools and report tools. It primarily benefits end-users and helps them to understand the technical solution when building front-end solutions.

The ultimate goal of metadata management is to describe all artifacts of the data warehouse, not limited to the previous list. There are various examples of metadata that need to be captured in the data warehousing domain, including descriptions of the relational table [1]:

1. **Record layout:** this metadata describes the layout of records in a relational table, including the list of attributes, their relative position and format of data on disk.
2. **Content:** the volume of data within a table or within a load for a particular table (volumetric) [3].
3. **Indexes:** the number and definition of indexes of the table.
4. **Scheduling:** the time schedule when the data in the table is loaded or refreshed.
5. **Usage:** where are the columns in the table being used? What are the dependencies for or on this table?
6. **Referential integrity:** what are the relations from this table to other tables?
7. **General documentation:** usually an unstructured text that describes the purpose of the table.

Often, parts or all of the above metadata is captured in the relational database management system (RDBMS) because database systems provide the means to capture such metadata. In other cases, an entity relationship (E/R) tool is used to capture such information. However, the previous list of metadata isn't even close to being a complete list. Instead, it focuses on only one area of metadata, the metadata about relational structures. However, there is much more metadata in a data warehouse system available, as we have already written in the introduction to this chapter.

Besides the back room and front room categorization of metadata, the metadata in a data warehouse is often categorized into three areas [2,4]:

- **Business metadata:** describes the meaning of data for the business. The Meta Mart covers this metadata and is covered in section 10.2.
- **Technical metadata:** describes the technical aspects of data, including data types, lineage, results from data profiling, etc. This metadata is also covered by the Meta Mart which is covered in section 10.2.
- **Process execution metadata:** provides statistics about running ETL processes, including the number of records sourced and loaded to the destination, number of rows rejected and the time it took to load the data. Collecting ETL process statistics is covered in sections 10.3 and 10.4 when the Metrics Vault and the Metrics Mart are described. In addition, error information is tracked to provide insight into exceptions or stops during ETL process execution. The Error Mart tracks these errors in the Data Vault 2.0 architecture and is covered in section 10.5.

We will follow this definition of metadata through the remainder of this chapter and consequently use this definition throughout this book. The next sections describe the metadata categories in more detail and provide some examples. Note that the list is still only a suggestion and provides only a limited number of metadata examples which we have seen in projects. In practice, each data warehouse team needs to modify the list of metadata that should be tracked to meet the needs of their organization and their project.

It should also be noted that the metadata might change over time because of changes in the underlying definition of the data. This includes all three categories, ranging from changes in the definition of business data (because the business meaning of the data has changed), over technical metadata (because the source systems have changed) to process execution metadata (because the data warehouse has changed).

10.1.1 BUSINESS METADATA

Business metadata describes the meaning of data for the business. Should the business therefore be responsible for business metadata? There are arguments for the assignment of business data to representatives of the business and arguments against it. For example, some argue that business data should be created and maintained during the requirements gathering process by data warehouse business analysts. Others argue that it should be maintained by source system business analysts because most business terms originate from source systems. And another group says that the business metadata should be created and maintained by data modelers when creating logical data models for the data warehouse [2].

The data warehouse team might not be the right party to decide who will be responsible for the business data but, in any case, it needs to decide what business metadata needs to be managed and where.

The list of business metadata that needs to be tracked includes the following metadata related to business definitions [2]:

- **Business column names:** in some cases, the business name might use abbreviations in prefixes or suffixes. Each of these abbreviations should be translated in order to make sense to the business side.
- **Business definitions:** for each attribute, table and other object, including soft business rules of the information mart, there should be a business description of the attribute's, table's or object's business meaning. In some cases, it is not easy to provide a business description for every item. However, this indicates that there is no analytical value of the item for the business and it can be removed from the information mart. If the business demands that the item remain in the information mart, it should be possible for the information consumer to provide a business definition.
- **Ontologies and taxonomies:** these business definitions describe the business objects behind the source data and the relationships to other business objects or the hierarchies that describe the business object itself. It also includes the classification of business objects. Typically, such ontologies and taxonomies are stored in analytical or operational master data management systems. This topic is covered in Chapter 9, Master Data Management.
- **Physical table and column names:** because front-end tools often present information to end-users with references to business names only, the business metadata needs to track the physical names that belong to the business names. Otherwise, the data warehouse team cannot associate business definitions to physical data elements.
- Technical numbering used to identify data elements in the technical model.

This information is required for end-users to understand the information marts they are dealing with; the power users need it in order to find the raw data in which they are interested in the Raw Data Vault; and the ETL developers need this metadata to understand the meaning of the source data when loading the Raw Data Vault and information marts by implementing business rules. In short, these business definitions provide meaning to the source data.

Another set of business metadata includes information about source systems. The following list provides the metadata that needs to be tracked for the data warehouse [2]:

- **Record source:** this metadata describes the record source in business terms, such as "Flight Tracking Database" or "Passenger Information." It should not be a technical reference to the server or database instance. Instead, it should be understandable by the business. In addition, it should include as much detail as possible. For example, instead of using "Passenger Information" for all data sources in the passenger information database, refer to specific data elements, such as "Passenger Information / Personal Module / Home Address Data." Section 10.2.2 describes how to cover such data in the Meta Mart.
- **Table specifications:** In addition to providing the business name of the source system and the detailed data object name, a description should be provided which explains the purpose of the source table, the volume of data in the table and the list of columns and keys (primary and alternate keys). Section 10.2.3 covers how to collect such data in the Meta Mart.
- **(Hard) exception-handling rules:** for each table, there should be a list of potential technical issues provided. This list describes the potential error or data quality issue and how the ETL process should deal with these errors. In Data Vault 2.0, these rules are referred to with the term "hard rule." We describe how to cover such rules in section 10.2.4.

- **Source system business definitions:** this metadata describes the business meaning of source attributes.
- **(Soft) business rules:** source systems implement business rules that modify or generate some of the source data. Other business rules perform data cleansing or prevent the insertion of erroneous data. For example, some business rules aggregate measures as new attributes in the source. When building the data warehouse, ETL developers are interested in these business rules to understand the source system and the business requirements behind it. Some source systems implement business rules within the application itself; others use the RDBMS for this purpose: the use of check constraints, triggers, and referential integrity point to business rule definitions that should be documented. In Data Vault 2.0, these rules are referred to using the term "soft rule." They are covered in more detail in section 10.2.9.

Section 10.2 will demonstrate how to implement these types of metadata with Microsoft SQL Server 2014.

10.1.2 TECHNICAL METADATA

While business metadata describes the meaning of data for the business, technical metadata serves many purposes. The data warehouse team is responsible for creating and maintaining this type of metadata and benefits from it most. Therefore, most technical metadata is around technical components of the data warehouse, such as [2]:

- **Source systems:** this type of metadata provides technical information about the source systems including the source database or flat file location and staging area tables used. See section 10.2.3 for a more detailed description.
- **Data models:** are the physical and logical data models, often presented in a graphical format, and provide information about the relationships between tables. While they are not metadata, they provide an invaluable asset to the data warehouse team.
- **Data definitions:** this list provides technical definitions of all columns in a data source. It including information about the table name, column name, data type, the domain (as enforced by foreign keys, check constraints or the application itself), referential integrity, constraints, defaults, triggers and stored procedures that ensure the data integrity of the source database.
- **Business rules:** the technical definitions of business rules are also considered as technical metadata because they need to be implemented in ETL later on. Similar to business rules in business metadata (refer to section 10.2.8), hard rules should be separated from soft rules. For each rule, there should be a technical description (unlike a more business-oriented description in the business metadata). There are also business rules that describe encryption or decryption requirements, among other data protection and security considerations.
- **Volumetrics:** there should be information about the table size and growth patterns of the source tables to estimate the workload of the data warehouse for this source table. It also helps to predict the growth patterns for hardware acquisition cycles of 6 to 12 months into the future.
- **Ontologies and taxonomies:** technical metadata should also provide information about ontologies and taxonomies, including abbreviations of terms and attributes, relationships, business key designations, peers and parents, hierarchies and re-defines (cross-ontologies matching at a structure level).
- **Data quality:** this kind of metadata provides information about standardization of source data and other data quality metrics.

Section 10.2 will demonstrate how to implement technical metadata in the Meta Mart using Microsoft SQL Server 2014.

10.1.3 PROCESS EXECUTION METADATA

Unlike business or technical metadata, which is provided by the business or source applications, process execution metadata is generated by the data warehouse team and provides insights into the ETL processing for maintenance. The data is used by the data warehouse team or by end-users to better understand the data warehouse performance and results presented in the information marts. There are four different types of process execution metadata, most of them coming from ETL systems such as SQL Server Integration Services (SSIS) [2]:

- **Control flow metadata:** a control flow executes one or more data flows among other tasks. Logging the process execution provides a valuable tool for maintaining or debugging the ETL processes of the data warehouse because it provides information about the data lineage of all elements of the data warehouse.
- **Data flow metadata:** the data flow log provides information about the performance of data flows and how many records have been processed or rejected by each transformation.
- **Package metadata:** a package executes a control flow. The package metadata provides summary information about the running time of the package.
- **Process metadata:** most packages are executed by SQL Server Agent or another scheduling application. The process metadata provides information about the process that has started the package.

While the list uses terminology from Microsoft SSIS, it can be applied to any other ETL tool. This kind of metadata should, on process execution, be held in the Metrics Vault and is usually separated from the business and technical metadata in the Meta Mart. Implementing a Metrics Vault and a Metrics Mart are covered in section 10.3. The next section covers implementing the Meta Mart.

10.2 IMPLEMENTING THE META MART

The Meta Mart is the central piece for collecting business and technical metadata in the Data Vault 2.0 architecture. As outlined in Chapter 2, it is an independent information mart that is not sourced from the Raw Data Vault. Therefore, there is no Meta Vault that could be used as a source for providing a virtual Meta Mart. Instead, the Meta Mart provides a set of tables that are used to collect the metadata of the data warehouse. The Meta Vault is materialized as it actually stores the metadata. Similar to other information marts in the Data Vault 2.0 architecture, the Meta Mart is modeled in such as way that business can make the most use of it. In some cases, this can be a dimensional model, while in others it is a model in third-normal form. In other cases, a metadata tool is used to present the metadata available to the end-user. In that case, the Meta Mart is the database for this metadata front-end.

There is no need for the Meta Mart to be a relational database: many commercial modeling and ETL tools support the customization of their metadata models in order to capture custom metadata, in addition to dedicated metadata management tools. Extending the model can be helpful to capture information about stage tables, such as the record source and the attribute classification (for example if the attribute from the source system is a business key, link information or descriptive attribute). It is also often possible to extend the model with data lineage capabilities especially added for Data Vault

purposes. For example, it is possible in many tools to provide the source tables of Raw Data Vault Hubs, Links and Satellites. Some of these tools also support the definition of business rules which can then be linked to the Data Vault model or the information mart model.

The next sections assume that no such tool is in place. Instead, alternative solutions are presented which are based on a relational Meta Mart.

10.2.1 SQL SERVER BI METADATA TOOLKIT

One such front-end tool is the SQL Server BI Metadata Toolkit, available on CodePlex [5]. It is an update of a tool for SQL Server 2005 which was first published on the MSDN Code Gallery. The toolkit is not production ready, but it serves as an example of a metadata front-end. The toolkit[1]:

1. scans SSIS packages, SSAS databases, SSRS reporting services and relational SQL Server databases
2. extracts the metadata from these components
3. stores them in the metadata database (the Meta Mart), and
4. provides a GUI for visually analyzing the metadata elements of the Microsoft BI stack.

By doing so, the toolkit extracts the metadata on its own and populates the Meta Mart of the data warehouse. Features of the toolkit include data lineage, covering business and technical metadata and providing impact analysis [5].

There are three tools included in the toolkit [6]:

1. **Dependency Analyzer:** this tool reads the contents of SSIS packages, SSAS databases and SQL databases and SSRS report collections. When analyzing SSIS packages, it gathers the metadata about all data flows in the package, about all sources and destinations in the data flows and the columns in the data flow. SSAS databases are analyzed for the included data sources, cubes, data source views, and dimensions. The gathered metadata is written to a selected repository database, usually the Meta Mart. Figure 10.1 shows the database tables that are created by the dependency analyzer.
2. **Dependency Executor:** this tool provides a graphical user interface to the dependency analyzer. Figure 10.2 and Figure 10.3 shows the user interface of this application.
3. **Dependency Viewer:** this tool presents the metadata to the end-user. It displays the information gathered by the dependency analyzer and shows metadata about the analyzed SSIS packages, SSAS databases, SSRS reports, and relational SQL Server databases. Figure 10.4 shows its user interface.

The database model which is created by the dependency analyzer is shown in Figure 10.1.

From a modeling perspective, the database model in this figure is not well designed, for example because primary keys are missing. But because it serves only as an example for a Meta Mart, it still presents an acceptable solution. The following tables are present in the database model:

1. **LookupConnectionID:** this table provides internal information about the connection types in SSIS.
2. **Audit:** this table stores run time information about SSIS data flows. We will implement a similar table when implementing the Metrics Vault in section 10.3.
3. **Version:** this table provides version information about the toolkit itself.
4. **RunScan:** this table stores a row for each analysis run (using the dependency analyzer).
5. **ObjectTypes:** this table provides information about the transformations and tasks available in SSIS and other data elements from other components of the Microsoft BI stack.

[1] https://sqlmetadata.codeplex.com/

6. **Objects:** this table provides metadata information about objects within SSIS packages, in SSAS or relational databases and in reports.
7. **ObjectDependencies:** this table describes the dependencies of objects within the components of the Microsoft BI stack. Objects might be contained by others, reference other objects, or map to other objects.
8. **ObjectAttributes:** this table provides detailed metadata about the object attributes.

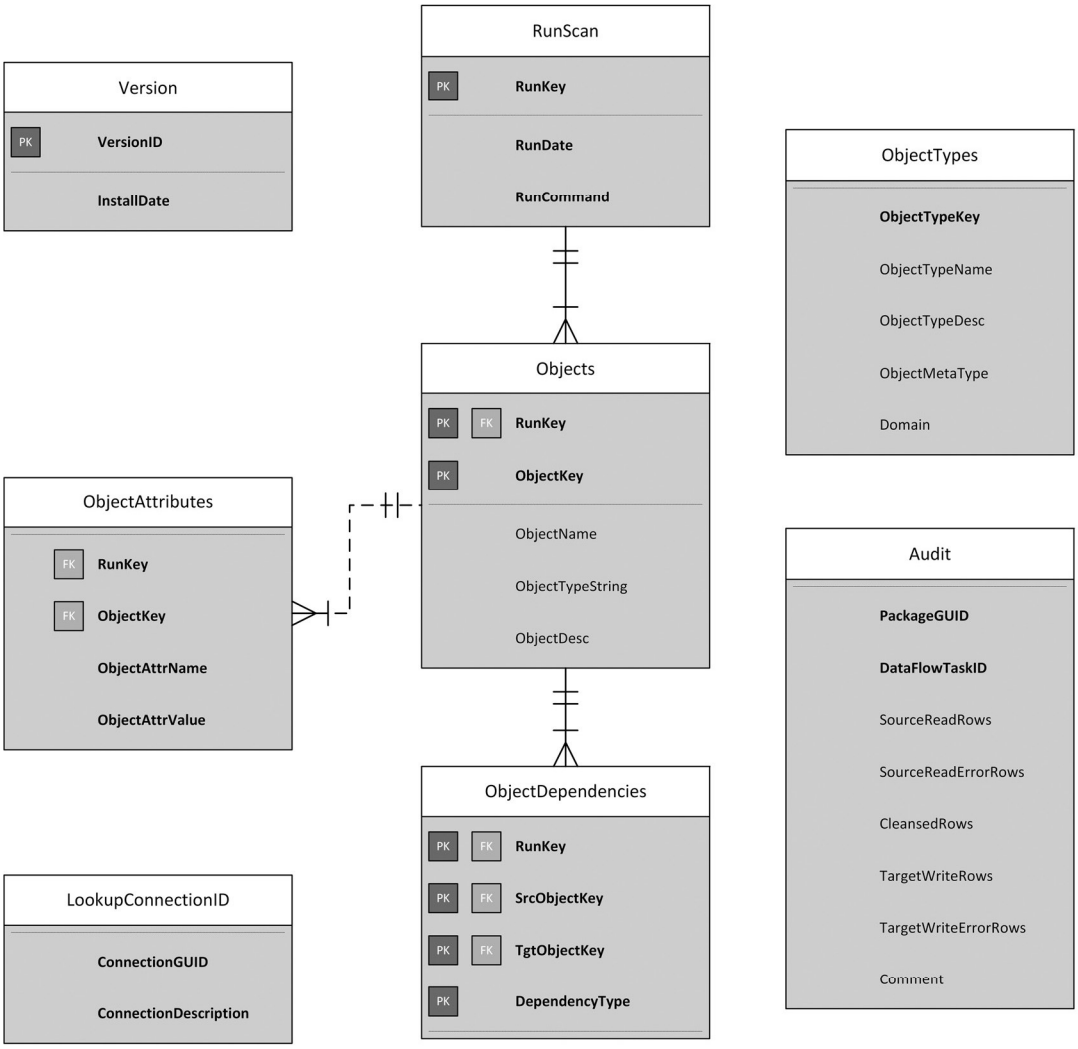

FIGURE 10.1

Database metadata for the SQL Server BI Metadata Toolkit (physical design).

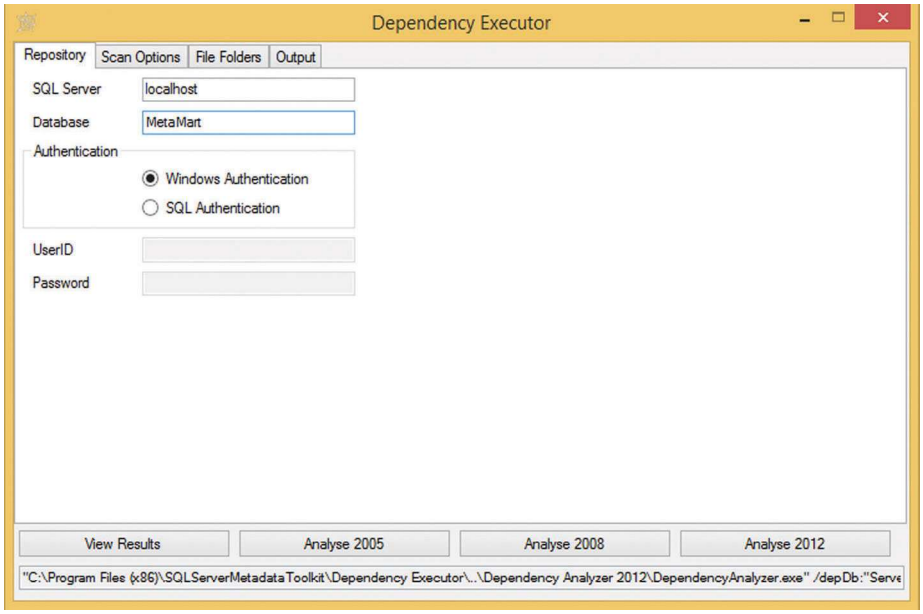

FIGURE 10.2

Setting up the metadata repository in the Dependency Executor.

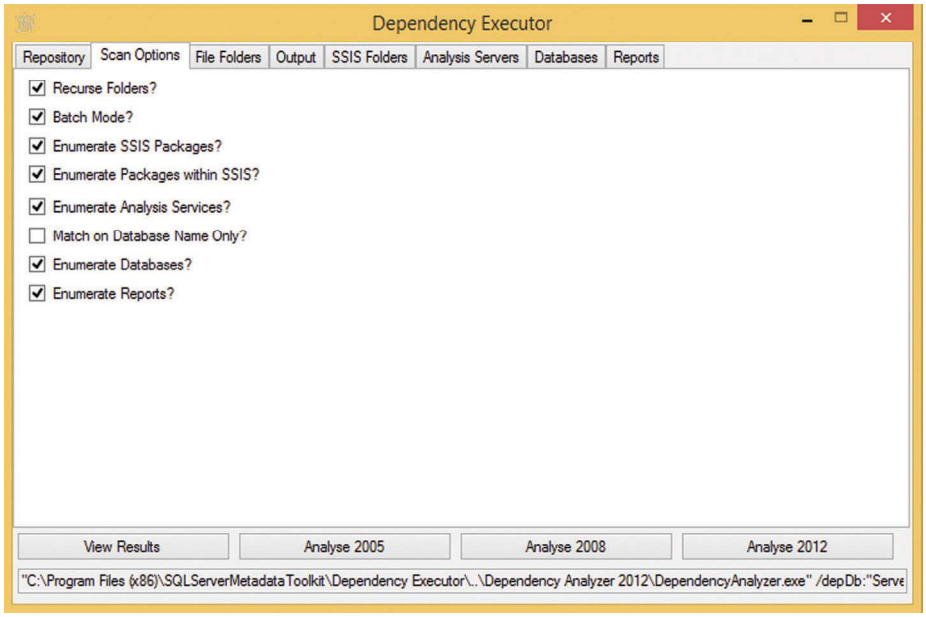

FIGURE 10.3

Setting scan options in the Dependency Executor.

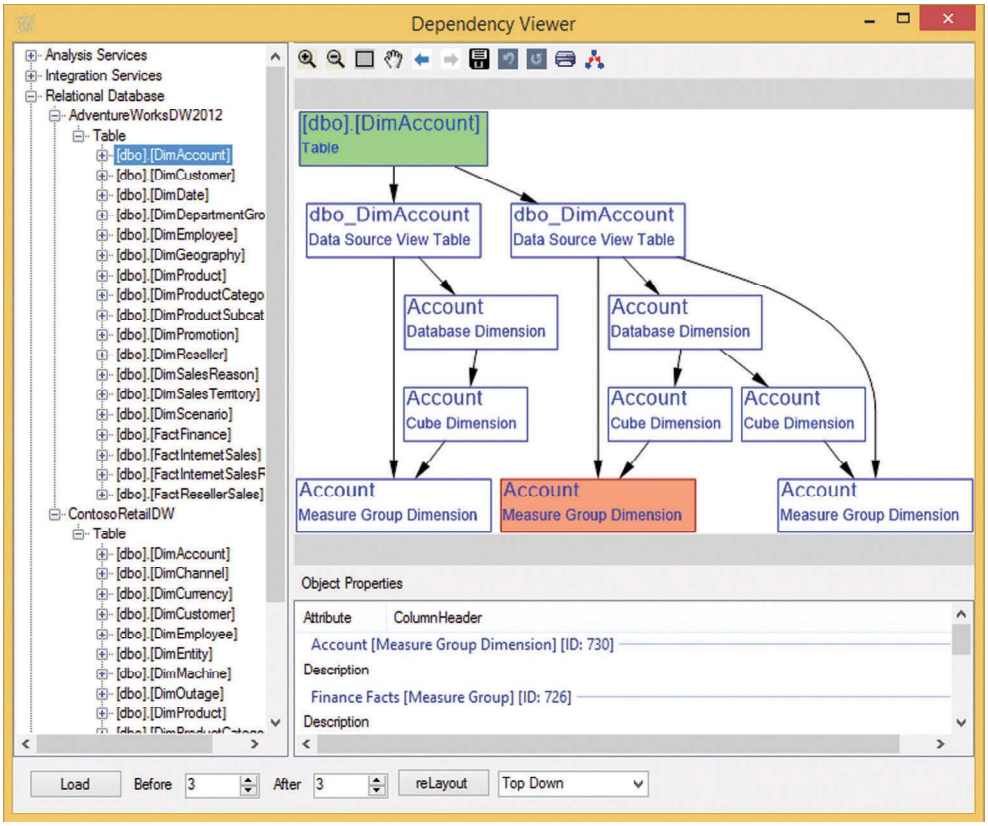

FIGURE 10.4

Dependency viewer of the SQL Server BI Metadata Toolkit.

These tables are populated by the Dependency Analyzer tool which might be run from the command line or using the graphical front-end, the Dependency Executor, as shown in Figure 10.2 and Figure 10.3.

Once the Dependency Executor has performed its analysis, the data is available in the tables presented in Figure 10.1. In order to analyze the dependencies between objects, the following query might be used to receive a readable dependency list:

```
SELECT
  dep.RunKey,
  src.ObjectName AS SrcObjectName,
  tgt.ObjectName AS TgtObjectName,
  dep.DependencyType
FROM
  dbo.ObjectDependencies dep
LEFT JOIN dbo.Objects src
ON dep.RunKey = src.RunKey AND dep.SrcObjectKey = src.ObjectKey
LEFT JOIN dbo.Objects tgt
ON dep.RunKey = tgt.RunKey AND dep.TgtObjectKey = tgt.ObjectKey
```

This query uses the **ObjectDependencies** table as the basis for the query and uses two LEFT JOINS to join the object names for readability. Because of the Dependency Viewer, it is not required to analyze the metadata using SQL queries. Instead, the Dependency Viewer provides a graphical way for the analysis of the metadata stored in the metadata repository (Figure 10.4).

The left tree view of the Dependency Viewer lists all objects found in the Microsoft BI stack. When selecting one of the objects, the object is displayed with all of its dependencies on the central area of the viewer. The bottom area displays the attributes of the selected object for further analysis.

The SQL Server BI Metadata Toolkit provides a powerful example for providing a simple Meta Mart on the technical metadata of the Microsoft BI stack. However, for tracking all artifacts of the enterprise data warehouse, we need to capture more metadata, as described in the previous sections.

10.2.2 NAMING CONVENTIONS

The first step in providing custom metadata is the use of naming conventions. Figure 10.4 from the previous section shows tables from a dimensional information mart that use two prefixes: "Dim" for dimensions and "Fact" for fact tables. By doing so, the table name provides information about the entity type of the table within the dimensional model.

The same approach is often used to denote entities in the Data Vault model. There are multiple options available for abbreviating the various Data Vault 2.0 entities and adding them to the actual identifier of the table. For example, many customers use one of the following abbreviations to name a Data Vault hub:

1. H_Customer
2. HUB_Customer
3. HubCustomer
4. CustomerHub

As you can see, many options exist to use a prefix or a suffix for naming a hub entity. Some customers use a prefix, others a suffix. In some cases, they use Camel Case, while in others they use underscores to separate terms in the table name. Because the options for naming conventions are often limited by development standards already available in the organization [2], project teams who start using Data Vault then only have the choice to decide which prefixes to use for each Data Vault entity. They cannot decide whether they want to use a prefix or a suffix any longer, because this decision has already been made a long time ago, by a prior project or, in larger organizations, a standards committee. In this way, organizations try to standardize their table and attribute namings among multiple projects.

The major question that is left to most project teams is instead: which types of Data Vault entities should have their own prefix? Should there be a prefix for every Data Vault entity type? Or are some entity types the same and should use the same prefix in turn? Table 10.1 lists the Data Vault entity types that should have their own individual abbreviations to be used as a prefix or suffix.

The entity types that should have their own prefix are presented in the first two columns of Table 10.1. Because the Business Vault is modeled after the same principles as the Raw Data Vault, there

Table 10.1 Required Abbreviations for Naming Conventions

Raw Data Vault	Business Vault	Avoid Prefix
Hub	**Business hub**	
Link	**Exploration link**	Nondescriptive links
Same-as-link	Same-as-link	
Hierarchical link	Hierarchical link	
Dependent link	Dependent link	
Nonhistorized link	Nonhistorized link	
Nonhistorized satellite	Nonhistorized satellite	
Satellite	**Computed satellite**	
Multi-active satellites	Multi-active satellites	
Status tracking satellite	Status tracking satellite	
Effectivity satellite	Effectivity satellite	
Record tracking satellite	Record tracking satellite	
Reference table		
Reference table satellite		
	Computed aggregate link	
	PIT table	
	Bridge table	

is often an equivalent of a Raw Data Vault entity in the Business Vault. For example, there might be a nonhistorized satellite in the Business Vault that has reduced the transactions for some reason, thus implementing parts of a business rule or the business rule as a whole. There are some exceptions: reference tables are primarily a concept of the Raw Data Vault, so reference data is usually not modified by a business rule (at least not in our experience). On the other hand, computed aggregate links, PIT tables and bridge tables are concepts of the Business Vault. In other cases, there is an equivalent entity in the Business Vault that follows the same modeling principles as its Raw Data Vault counterpart, but using a different name: for example, an exploration link is a computed link without any other modifications. The only difference is that the exploration link is not sourced from an operational data source, but includes links in the model that have been artificially generated (for example through a data mining algorithm). As a result of these observations, organizations tend to introduce an individual abbreviation for Business Vault entities, especially because it helps them to separate raw data from computed information.

In most cases, a different entity structure demands its own abbreviation. However, there are some exceptions to this rule: the same-as-link or the hierarchical link, for example. While they follow the standard structure of a Data Vault link, they serve a special purpose. Therefore, they should

have their own abbreviation. The same applies to status tracking satellites or effectivity satellites which follow standard Data Vault structures for satellites as well. One exception to this rule is the nondescriptive link (or fact-less fact). It is a special link, but only because it has no satellites. The link itself, the relationship between two business keys, is the fact. The reason why this link should not have its own abbreviation is that a satellite could be attached to the link later on. For example when data from a new operational system is sourced and, all of a sudden, this source system provides data that describes this link (thus adding a Raw Data Vault satellite to the link). By doing so, the nondescriptive link transforms into a standard link and should use the same abbreviation as the standard links. The same applies to cases when computed satellites are added to the nondescriptive link.

When naming Raw Data Vault satellites, it is recommended to include an abbreviation for the source system because, in many cases, there are multiple satellites hanging off the parent hub or link. If our recommendation from Chapter 4, Data Vault 2.0 Modeling, is followed, each source system should be loaded to its dedicated satellite or satellites. This avoids the drawbacks of overloaded satellites as described in that chapter. But if each source system creates a new satellite on a hub, they need different names to distinguish them from each other, both in a logical view and in physical implementation. The logical diagram in Figure 10.5 shows this issue.

Three source systems provide descriptive data for the passenger. They are abbreviated as TRV (travel system), CRM (customer relationship system) and SEC (security). Using this simple naming convention makes it clear that the table is a satellite, to which parent hub each satellite belongs, and where the data comes from. The next issue is satellite splitting: if there are source columns that change frequently, the recommendation from Chapter 4 is to split the satellites by rate of change. For example, if the satellite **SatPassengerCRM** in Figure 10.5 is split by rate of change, the logical model shown in Figure 10.6 results from this split.

The preferred dish information was split from the other passenger data because it changes more frequently than, say, the name. In this case, the name of the entity was changed from Sat**Passenger**CRM to Sat**PassengerPreferredDish**CRM because the separated attribute makes it obvious how to name the new satellite. But what if this becomes more complicated because a couple of attributes are separated that don't necessarily belong together? What if you decide to mechanically separate the attributes of a

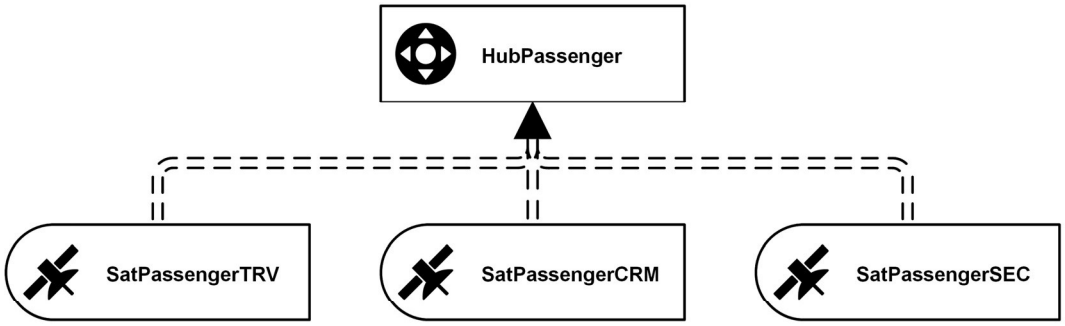

FIGURE 10.5

Multiple satellites on parent hub (logical design).

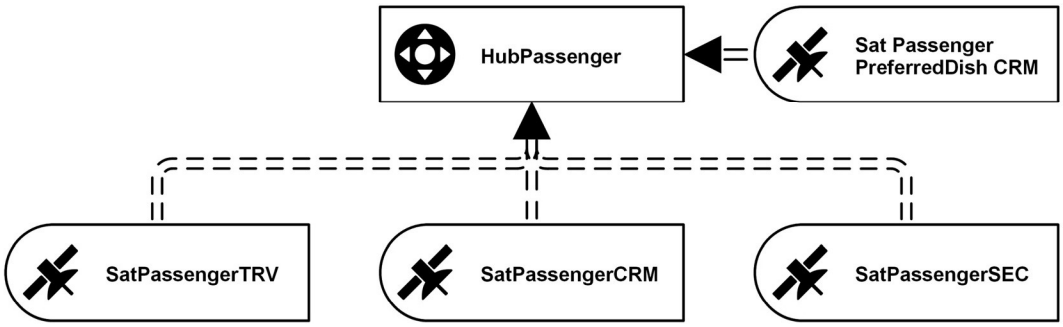

FIGURE 10.6

Multiple satellites after split (logical design).

satellite by some rate of change? To make things more complicated, there are multiple definitions of rate of change as the (incomplete) Table 10.2 demonstrates.

Depending on your understanding of rate of change, one or another (or multiple) definitions might be useful in your project. There should be an abbreviation for each rate of change. For example, if we use the three speed levels slow, medium and fast to denote the rate of change in our satellites, a logical design of our model in Figure 10.5 could look like the design in Figure 10.7.

In this example, the satellite **SatPassengerCRM** has been split into three new satellites: **SatPassengerSCRM**, **SatPassengerMCRM** and **SatPassengerFCRM**. The character before the source system indicates the rate of change for each satellite.

We call this split a mechanical split because it could also be automated: after analyzing the current rate of change for each attribute in a satellite, the algorithm could generate the required DDLs for the new satellites without knowing any business definition of each attribute (thus unable to find a more meaningful name as in the previous example).

Table 10.2 Various Rate of Change Definitions

Schedule	Volatility	Speed
Hourly	Low	Very Slow
Daily	Medium	Slow
Weekly	High	Medium
Monthly		Fast
Quarterly		Very Fast
Semiannual		Light
Yearly		Warp

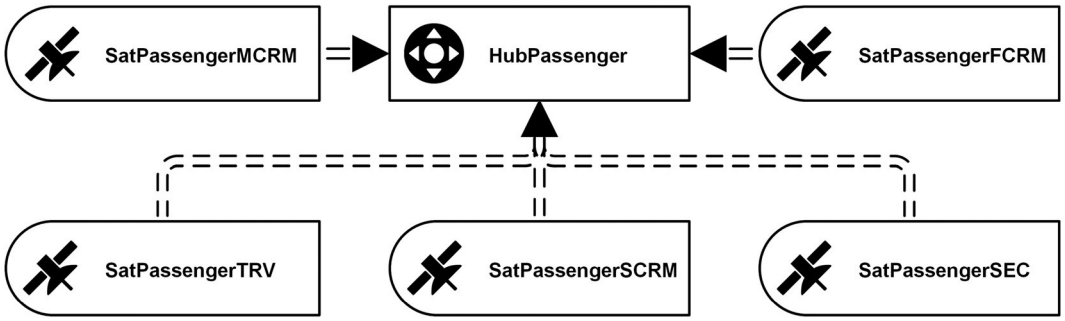

FIGURE 10.7

Multiple satellites after mechanical split (logical design).

The advantage of using naming conventions for the Data Vault tables is that the information about the entity type is encoded in the table name. This information becomes available in all tools that present SQL Server tables to its end-users. The disadvantage, however, is this metadata is not easily retrievable as dedicated metadata, attached to the entity.

10.2.3 CAPTURING SOURCE SYSTEM DEFINITIONS

The purpose of the source system abbreviation in satellites (as described in the previous section) is only to distinguish the table names when multiple source systems provide descriptive data for a business key. It is not a description of the source system.

Instead, much more metadata is required to describe the source system for business and IT usage. For example, the fact that data has been sourced from a source system is important in agile projects, because not all available source system tables are loaded to the data warehouse at once. Instead, only the source systems that provide raw data that is required to build a certain report or other information artifact are sourced into the data warehouse (and only partially, table by table). We source data just in time. Identifying the source system tables that have not been implemented in the data warehouse is required because the data hasn't been loaded in previous sprints. Therefore, these missing source system tables need to be loaded in the current iteration.

Typically, organizations decide to use at least the following metadata attributes to define the source systems sourced by their data warehouse [7]:

- **Source system identifier:** a technical identifier of the source system used to reference the source system in subsequent metadata or other documentation.
- **Source system technical description:** technically describes the source system in a textual format.
- **Source system business description:** describes the source system from a business perspective.
- **Source system version:** the current version of the source system.
- **Source system quality:** describes the quality of the source system (could also use an indicator scale from "poor" to "good" or a percentage value).

- **Data steward:** provides the name and contact information of the data steward responsible for the management of data elements of the operational system.
- **System administrator:** provides the name and contact information of the person who is responsible for administrating the system.

Note that some source systems provide data in separate batches that might not be delivered in the same schedule. For example, some data could be delivered on a daily basis, while other data is delivered in real-time. For that reason, the actual data delivery (a data package) should be defined in another metadata table that includes the following attributes [7]:

1. **Source system identifier:** the technical name of the source system that is providing the data package.
2. **Data package identifier:** the technical name of the data package.
3. **Data package format:** defines how the source system provides its data to the data warehouse: CSV files, XML files, relational access, real-time messages, etc.).
4. **Data package type:** indicates if the package is a full load or delta load.
5. **Delivery schedule:** defines when the data is provided to the data warehouse (e.g., daily between 2 a.m. and 4 a.m.).
6. **Data package technical description:** technically describes the data package in a textual format.
7. **Data package business description:** describes the data package from a business perspective.
8. **Expected data package quality:** describes the quality of the data package (could also use an indicator scale from "poor" to "good" or a percentage value).
9. **Data package refresh type:** indicates the frequency with which the content of the data package are refreshed (e.g., real-time, near-real-time, hourly, daily, monthly, etc.)
10. **Database instance name (relational source):** if the data package is made available using a relational database (for example, if the data warehouse can directly access the operational system to source the data from), the database instance name is required.
11. **Database name (relational source):** if the data package is a relational table, this attribute indicates the database to load the data from.
12. **Delimiter type (flat file source):** if the data package is a flat file, this attribute indicates the delimiter used.
13. **Text delimiter (flat file source):** if the data package is a flat file, this attribute indicates the string delimiter used to separate string elements (usually a double quote).
14. **Header (flat file source):** indicates if a header row is used in a flat file source.
15. **File format (flat file source):** indicates the exact file format, such as CSV, EBCDIC, ASCII, VSAM, etc.
16. **Control flow name:** the name of the control flow that is responsible for loading the data package.

Note that a data package consists of multiple files or relational tables. Depending on the actual source systems and their intended data package formats, there might be need for more metadata attributes. This list should serve only as a starting point. Chapter 11, Data Extraction, lists more potential source systems and data feeds (such as Web sources, social networks, mainframe systems, etc.) and describes the requirements for sourcing them into the data warehouse.

10.2.4 CAPTURING HARD RULES

Hard rules in the data warehouse deal with data type conversions required for loading and technical issues that can arise when staging the data or loading the data from the staging area into the Raw Data Vault. Therefore, they are applied when loading the data into the staging area or loading the data from the staging area to the Raw Data Vault. Examples include:

1. **Assignment of source attribute to target attribute:** in some cases, organizations choose to use a different name in the staging area or in the Raw Data Vault. In this case, the hard rule defines how to map a source field to a target attribute in the staging area or in the Raw Data Vault.
2. **Hard rules that convert a source data type to a target data type:** this practice should be avoided, but in some cases, there is no equivalent data type in the target that directly reflects the source system. For example, an operational system could have been built with Java or .NET types in use, but the data warehouse has decided to use only generic T-SQL data types. A hard rule defines how to perform the required conversion.
3. **Hard rules that ensure how to deal with wrongly sized data:** this type of rule deals with data that is too long to be stored in the destination, for example, because a VARCHAR column in the target is unable to capture the whole source string. Usually, the destination should be able to cover the whole string, but what if the source system was modified and sends more data than expected? In other cases, the incoming data should have an expected number of minimum characters or minimum number of numerical digits. The hard rule defines if and how the data is cut or extended or if the record is rejected.
4. **Invalid data type formats:** because most raw data is not directly sourced from the operational system, but by using comma separated text files (CSV) or other text-based formats. In such a case, the original data is transformed to strings using a predefined format. For example, date data types can be converted from their internal binary representation into UTC strings which present the date using the following format: "YYYY-MM-DD hh:mm:ss[.nnn]" or into a USA style date with the format "mm/dd/yy", omitting the time information and completely changing the structure of the date format. The hard rule does not only define how to convert from a date data type, but also how to deal with strings that don't comply with the expected format. Similar hard rules are required to convert floating numbers, Boolean values, etc.
5. **Unicode:** another practice is to store all string data in the staging area and in the Raw Data Vault using Unicode character sets to make sure that all character data can be stored. The hard rule defines not only the target character set but also any required conversion.
6. **Reformat business keys:** many organizations choose to reformat business keys if there exist multiple formats in operational systems. For example, if a phone number is used as a business key by the business, it might make sense to use a standardized format when loading the hubs to avoid added same-as-links (SAL). Note that the satellites should track the original values as they were stored in the operational systems. This is a hard rule as long as the content of the data doesn't change.
7. **Local smart keys:** In many cases, there are operational systems that are used only in a certain region. In such cases, they often use a local business key, which is extended to a smart key globally. For example, a customer number "12345" for a customer in Germany could be identified as the smart key "DE12345" in the global enterprise. Thus, the business key "DE12345" should

be used in the data warehouse when loading hubs. A hard business rule can be used to convert such local keys into the expected format of the smart key.

8. **Hash keys and hash differences:** the definition of hash keys (on business keys and their relationships) and hash differences (on descriptive attributes) are also defined by hard rules.

Note that the local smart keys should only be used under the described circumstances: to apply the format of the smart key to a local business key. It should be avoided to use a more complicated mapping when applying a hard rule, for example to map the business key "12345" to "XYZ". Because the mapping between these numbers is not a technical mapping but defined by the business (in general), the mapping is actually a soft business rule that should be applied later.

The primary definition of the hard rules for the data warehouse is done using documents that define the hard rules and their applicability. This definition also includes how and where deviations from the expected format and actual errors should be logged. The following metadata attributes should be used to define hard rules:

- **Data flow name (optional):** the data flow that is implementing this hard rule.
- **Name:** a name for the hard rule understandable by the business.
- **Rule identifier:** the technical name of the hard rule.
- **Description:** a technical description of the hard rule.
- **Definition:** the actual definition of the hard rule, using a text-based or graphical notation.
- **Business area:** the functional business area (or business owner) that defines this hard rule.
- **Topic:** a given topic that this hard rule is part of.
- **Task/rule set:** a more detailed task or rule set that this hard rule is part of.
- **Source:** The source of the hard rules definition (for example, project documentation).
- **Implementation type:** Table 10.3 lists the potential rule types.
- **Keywords:** keywords that help to find the hard rule from the business rule collection.
- **Related rules:** relates this hard rule with other hard rules.
- **Example:** provides example inputs and their respective outputs.

The above list provides some commonly used metadata attributes. Not all of them are required in all projects and, in many cases, organizations decide to add additional ones that make sense in their context.

Table 10.3 Hard Business Rule Types [8]

Hard Rule Type	Description
Restriction	These rules define business specific constraints on the data that is allowed or not allowed to use by the business. They state what data is not to enter the staging area or the Raw Data Vault.
Behavior	These rules express how the system should react to given situations, primarily regarding the source systems and their deliveries (such as missing files, etc.).
Deduction	These rules define how to derive corrected data for the staging area or the Raw Data Vault. Note that this rule does not change the data, but fixes data type issues only.
Warning	These rules define what kind of error should be logged and how.
Other	All other rules that don't fit into the standard rule types.

Table 10.4 Hard Rule Definition [9]	
Name:	NULL SAP Date String to DateTime2
Identifier:	HR0012
Description:	SAP provides NULL dates as "00000000" instead of empty strings as Microsoft SQL Server expects it. Therefore, these values are converted to NULL dates instead of direct date representations.
Example:	"00000000" is converted to a NULL date in the target. All other dates are not affected by this hard rule.
Source:	SAP technical interface documentation (Document ID: SAP1234 as of February 3, 2011)
Related Rules:	HR0010: SAP Date String to DateTime2 HR0011: SAP Date String Error Handling

Spreadsheets are often used to define which hard rules apply to which source and target attribute mappings. The form in Table 10.4 can be used to provide a detailed definition of a hard rule.

Because hard rules can be applied when loading the staging area and when loading the Raw Data Vault, the definition is referenced in the mapping table for either or both layers. Section 10.2.5 and 10.2.7 show examples for mapping tables that include such references.

10.2.5 CAPTURING METADATA FOR THE STAGING AREA

We have described in section 10.2.3 that tracking the source systems (that have been loaded into the data warehouse) is important in order to know what raw data is available in the data warehouse and what is still missing. But knowing the source system is not enough: the recommended approach to loading raw data from source systems is to load all the source tables from any source system that provide us the required raw data to build an information artifact, such as a report. For example, if source system A provides required raw data in one table and source system B provides required raw data in two more tables, we source these three tables completely. It is not recommended to source all tables from each source system, because the data contained in these source tables are not required for the current sprint, thus overloading the sprint with unnecessary effort. However, it is also not recommended to source only those attributes in the source tables that provide the required raw data, but load a source table all at once. That way, the overhead for managing the data warehouse contents (the tables already loaded into the data warehouse) is less. It should be clear that managing the data warehouse contents on an attribute level is more effort than on a table level.

For each table in a source system, the following metadata attributes should be managed at a minimum [10]:

1. **Data package identifier:** the technical name of the data package that contains this table or other data structure (for example individual flat file).
2. **Table identifier:** the technical name of the data group, which is a set of data elements (attributes).
3. **Source table schema name:** the name of the source schema.
4. **Source table physical name:** the physical name of the source table.
5. **Table description:** a technical description of the source table.

6. **Table business description:** a detailed textual description of the source table in business terms.
7. **Data quality:** describes the quality of the source table (could also use a indicator scale from "poor" to "good" or a percentage value).
8. **Target table schema name:** the name of the schema in the staging area.
9. **Target table physical name:** the physical name of the staging table.
10. **Data flow name:** the name of the data flow that is loading the table.

Each table consists of multiple source columns and their respective target column mappings, which are described by an additional metadata table [10]:

- **Table identifier:** the technical name of the parent data table.
- **Column identifier:** the technical name of the column in the source table.
- **Source column physical name:** the physical name of the source column in the source table.
- **Source data type:** the data type of the source column.
- **Column description:** the technical description of the column.
- **Column business description:** a detailed textual description of the column in business terms.
- **Column business name:** the common column name that is recognized by business users.
- **Column business alias:** an alternative column name that is recognized by business users.
- **Column acronym name:** a common acronym coding of the column name.
- **Required flag:** indicates if the column is required to have a value (NOT NULL).
- **Computed flag:** indicates if the column is derived from a computed column in the source system.
- **Target column physical name:** the physical name of the target column in the staging area table.
- **Target data type:** the data type of the target column.
- **Is sequence:** indicates if the target column is a sequence column.
- **Is hash key:** indicates if the target column is a hash key.
- **Is hash difference:** indicates if the target column is a hash difference value.
- **Is record source:** indicates if the target column is a record source attribute.
- **Is load date time:** indicates if the target column is a load date time.
- **Hard rules:** references to the hard rules that are applied within the loading process.

It is also possible to denormalize these two tables. While adding redundancy, it is often easier to use.

Note that it should be possible to apply multiple hard rules to a source to stage column mapping. If it is not possible, another option is to define a hard rule that packages multiple hard rules, thus applying multiple hard rules to the referencing mapping. However, such an approach requires more hard rules with fewer chances for reuse.

When sourcing hierarchical files (or tables), additional and more complex metadata is required because the goal of a hierarchical mapping is to map each column (or XML attribute and element) to a relational column in the staging area. Loading hierarchical sources requires normalization for that reason, as Chapter 11, Data Extraction, will demonstrate. One approach to define the metadata for hierarchical sources is to include the hierarchical path in the **source column physical name** and add a **target table physical name** and **target table schema name** (if actually required) to support multiple target tables.

10.2.6 CAPTURING REQUIREMENTS TO SOURCE TABLES

This cross-reference table was introduced in Chapter 3, The Data Vault 2.0 Methodology, and identifies the source tables and optionally the columns that are used by a specific requirement. Table 10.5 shows an example.

Table 10.5 Requirements to Source Tables Cross Reference with Source Attributes

Requirements to Source Map XREF

Requirement Document: MyDoc

Table

Source Table	Source Column	Source Table Physical Name	Source Column Physical Name	Passenger Name Dimension (B2.2)	Preferred Dish Attribute (B3.2.1)
Passenger	Passport Number	PAX	NO	X	X
Passenger	First Name	PAX	FNAME	X	
Passenger	Last Name	PAX	LNAME	X	
Passenger	Title	PAX	TITLE	X	
Passenger	Address	PAX	ADDR		
Flight Reservation	Seat	RES	SEAT		
Flight Reservation	Preferred Dish	RES	DISH		X

One row exists per source column. Each source column is described by some additional information, in this case the physical names. Other options are possible and left to the data warehouse team (for example, the business description). For each requirement, one column is added. Note that not only the name of the requirement is provided but also the technical number that uniquely identifies the requirement in the requirements specification. An X in the cell indicates if the source column is used to implement the requirement. In addition to the table itself, the document references the requirements document thata provides the detailed requirements specification (there might be multiple requirements documents in a project if the team decides to split the requirements up, e.g. per function, etc.).

This cross-reference table can be used in two ways:

1. To identify the requirements that are affected by a change of the source system.
2. To identify the source tables and columns that are needed to be sourced in order to implement a requirement.

Following the general recommendation from 10.2.5, that is, to completely load a table into the data warehouse if at least one column is required by a requirement, we can use the cross-reference table to identify those source tables needed by the iteration that implements a specific requirement.

10.2.7 CAPTURING SOURCE TABLES TO DATA VAULT TABLES

In order to capture metadata for mapping from source tables (in the staging area) to Raw Data Vault tables, multiple metadata tables are recommended. The reason lies in the fact that the mapping for a hub

entity is fundamentally different from link entities and from satellite entities. For example, in order to map from a staging table to a Data Vault hub, the business key needs to be identified only. However, in order to map from a staging table to a Data Vault satellite, all descriptive attributes need to be mapped in addition to all attributes identifying the satellite's parent. To complicate matters, there are significant differences between satellites on hubs and satellites on links, requiring separate tables for both entity types.

Note that the lists of metadata attributes in the following sections are suggestions only. Before maintaining metadata for the data warehouse, the list should be tailored to the actual needs of the data warehouse team. One problem with metadata management is that it often tends to become outdated because many teams stop maintaining the metadata tables. To overcome this problem and speed up the design and development processes of the data warehouse, it is worth investing in modeling and generation tools.

The following definitions also cover only basic Data Vault entities (as covered in Chapter 4). In order to define the metadata for intermediate or advanced concepts, additional metadata attributes might be required.

10.2.7.1 Metadata for Loading Hub Entities

In order to define the metadata for Data Vault hubs, the following attributes are required:

1. **Data flow name:** the name of the data flow that is loading the target hub.
2. **Priority:** a common practice is to source business keys from multiple sources. In this case, the priority can be used to determine the order of the data sources when loading the hub, which might affect the record source to be set in the target hub.
3. **Hub identifier:** the technical name of the target hub.
4. **Target hub table physical name:** the physical name of the target table in the Raw Data Vault.
5. **Source table identifier:** the technical name of the source data table in the staging area.
6. **Source table physical name:** the physical name of the source table in the staging area.
7. **Source column physical name:** the physical name of the source column in the source table that holds the business key.
8. **Source column data type:** the data type of the source column.
9. **Source column required:** indicates if the source column allows NULL values.
10. **Source column default value:** indicates the default value of the source column.
11. **Source column computation:** if the source column is a computed field, provide the expression that computes the column value for documentation purposes.
12. **Business key column description:** the technical description of the business key column.
13. **Business key column business description:** a detailed textual description of the business key column in business terms.
14. **Business key column business name:** the common business key column name that is recognized by business users.
15. **Business key column business alias:** an alternative business key column name that is recognized by business users.
16. **Business key column acronym name:** a common acronym coding of the business key column name.
17. **Business key physical name:** the physical name of the target business key column in the hub table.
18. **Target column number:** The column number of the business key within composite keys. Otherwise 1.
19. **Target primary key physical name:** the physical name of the target primary key column in the hub table.
20. **Target data type:** the data type of the target business key column.

Table 10.6 Metadata for Capturing Source Tables to Data Vault Hub Entities

Hub Identifier	Target Hub Table Physical Name	Source Table Physical Name	Source Column Physical Name	Source Data Type	…	Target Column Physical Name	Target Data Type	Last Seen Date Flag	Hard Rules
H001	HubAirline	AIRLINE	ID	Integer	…	AirlineID	Integer	False	HR33.1
H001	HubAirline	AIRX	NUMBER	BigInt	…	AirlineID	Integer	False	HR1.2.3
H002	HubFlight	FLIGHT	CARRIER	VarChar(2)	…	Carrier	VarChar(2)	False	HR33.1,HR33.2
H002	HubFlight	FLIGHT	NUM	BigInt	…	FlightNum	Integer	False	HR33.1,HR33.3

21. **Last seen date flag:** indicates if a last seen date is used in the hub and should be updated in the loading process.
22. **Hard rules:** references to the hard rules that are applied within the loading process for this business key.

In order to support composite business keys in Data Vault hubs, it is required to provide one line per business key part when dealing with composite business keys. Table 10.6 shows examples for both a hub with only one business key and one hub with a composite key consisting of two elements.

The first hub is hub **Airline** that consists of only one business key attribute **AirlineID**. However, the business keys for this hub are loaded from multiple sources: there is a staging table **AIRLINE** present and another staging table **AIRX**. In this case, the source table identifier (which was introduced in the metadata for the staging table) is the same as the source table physical name. Both source tables provide business keys that need to be loaded in order to include all business keys that are used by the business. Therefore, a hub definition is copied per source table in order to fully define the required metadata to load a hub from all sources.

The second hub loads business keys from only one staging table but is defined by a composite business key. Therefore, two metadata rows are required to fully define the composite key. If multiple source tables provided business keys, these two rows would be copied for each source table.

Because copying rows introduces some redundancy, it might be valuable to investigate normalized structures, especially if metadata tools are involved. If no tools are available, a denormalized structure as proposed in this section might be the more user-friendly structure.

Note that this example doesn't display the descriptive metadata attributes due to space restrictions. However, it shows references to hard rules, which are defined in external documents that describe the hard rules in full detail. Technical numbering is used to reference the hard rules.

10.2.7.2 Metadata for Loading Link Entities

Loading Data Vault links follows a similar pattern compared to hubs but with a little more complexity. The additional complexity is due to the fact that a link table references other hubs to store the relationship between the business keys:

1. **Data flow name:** the name of the data flow that is loading the target link.
2. **Priority:** sometimes, link data is sourced from multiple sources. In this case, the priority can be used to determine the order of the data sources when loading the target link, which might affect the record source to be set in the target link.

3. **Link identifier:** the technical name of the target link.
4. **Target link table physical name:** the physical name of the target table in the Raw Data Vault.
5. **Source table identifier:** the technical name of the source data table in the staging area.
6. **Source column physical name:** the physical name of the source column in the source table that holds the business key.
7. **Source column data type:** the data type of the source column.
8. **Source column required:** indicates if the source column allows NULL values.
9. **Source column default value:** indicates the default value of the source column.
10. **Source column computation:** if the source column is a computed field, provide the expression that computes the column value for documentation purposes.
11. **Source data type:** the data type of the source business key column.
12. **Business key driving flag:** indicates if this business key is part of the driving key (if any).
13. **Business key column description:** the technical description of the business key column.
14. **Business key column business description:** a detailed textual description of the business key column in business terms.
15. **Business key column business name:** the common business key column name that is recognized by business users.
16. **Business key column business alias:** an alternative business key column name that is recognized by business users.
17. **Business key column acronym name:** a common acronym coding of the business key column name.
18. **Hub identifier:** the technical name of the referenced hub.
19. **Hub table physical name:** the physical table name of the reference hub.
20. **Hub reference number:** the number of the hub reference within the sort order of the hub references. This is required to calculate the correct hash key.
21. **Hub primary key physical name:** the physical name of the primary key column in the referenced hub table.
22. **Hub business key physical name:** the name of the business key column in the hub.
23. **Hub business key column number:** the number within the column order of the business key in the hub. Required to calculate the correct hash value.
24. **Hub business key data type:** the data type of the business key column in the referenced hub table. Can be used for automatically applying hard rules.
25. **Target column physical name:** the physical name of the target hash key column in the link table.
26. **Last seen date flag:** indicates if a last seen date is used in the hub and should be updated in the loading process.
27. **Attribute flag:** indicates if the column is an attribute instead of a business key. This is required to define degenerated links (refer to Chapter 4).
28. **Hard rules:** references to the hard rules that are applied within the loading process for this business key.

The number of entries per link depends on multiple factors: first, the number of referenced hubs. For each hub reference there is at least one metadata record required to completely define the link. In addition, if a composite business key defines a hub, the dependent link entry in the metadata table for links requires one record per business key part. Table 10.7 shows a simplified example for a link metadata table.

Table 10.7 Metadata for Capturing Source Tables to Data Vault Link Entities

Link Identifier	Target Link Table Physical Name	Source Table Physical Name	Source Column Physical Name	Source Data Type	…	Hub Table Physical Name	Target Column Physical Name	Hub Business Key Column Number	Hard Rules
L001	LinkFixedBaseOp	FB_OPS	CARRIER	VarChar(2)	…	HubCarrier	CarrierHashKey	1	HR22.1
L001	LinkFixedBaseOp	FB_OPS	AIRPORT	VarChar(3)	…	HubAirport	AirportHashKey	1	HR1.2.5
L002	LinkConnection	CONN	CARRIER	VarChar(2)	…	HubFlightNo	FlightNoHashKey	1	HR22.2
L002	LinkConnection	CONN	FLIGHT	Integer	…	HubFlightNo	FlightNoHashKey	2	HR22.2
L002	LinkConnection	CONN	S_AIRPORT	VarChar(3)	…	HubAirport	SrcAirportHashKey	1	HR1.2.1
L002	LinkConnection	CONN	T_AIRPORT	VarChar(3)	…	HubAirport	TgtAirportHashKey	1	HR1.2.1

The table is simplified because some of the metadata attributes are omitted. The first link **LinkFixedBaseOp** references two hubs: **HubCarrier** and **HubAirport**. Both hubs are defined by simple business keys and not by composite business keys. The second link **LinkConnection** also references two hubs, but one hub, **HubFlightNo**, is defined by a composite business key, consisting of two parts: first the carrier ID and second, the flight number. The second hub **HubAirport** is referenced two times: as source airport of the connection and as a target airport of the connection. For that purpose, both references are stored in different hash keys.

Similar to the metadata table for hubs, presented in the previous section, this table contains redundant metadata in favor of usability. Again, it might be valuable to use a metadata tool with normalized metadata tables.

10.2.7.3 Metadata for Loading Satellite Entities on Hubs

The metadata for satellites contains two types of information:

- **Business keys** that identify the entry in the parent hub entity.
- **Descriptive data** that has to be loaded into the Data Vault satellite for data warehousing purposes.

In order to keep the metadata table as simple as possible, both types of information are provided in the same table: first, the business key information, and second the descriptive data mapping. The mapping table consists of the following metadata attributes:

1. **Data flow name:** the name of the data flow that is loading the target satellite.
2. **Satellite identifier:** the technical name of the target satellite.
3. **Target satellite table physical name:** the physical name of the target table in the Raw Data Vault.
4. **Source table identifier:** the technical name of the source data table in the staging area.
5. **Source column physical name:** the physical name of the source column in the source table that holds the business key or the descriptive data.
6. **Source column data type:** the data type of the source column.
7. **Source column required:** indicates if the source column allows NULL values.
8. **Source column default value:** indicates the default value of the source column.
9. **Source column computation:** if the source column is a computed field, provide the expression that computes the column value for documentation purposes.
10. **Business key driving flag:** indicates if this business key is part of the driving key (if any).
11. **Business key column description:** the technical description of the business key column.
12. **Business key column business description:** a detailed textual description of the business key column in business terms.
13. **Business key column business name:** the common business key column name that is recognized by business users.
14. **Business key column business alias:** an alternative business key column name that is recognized by business users.
15. **Business key column acronym name:** a common acronym coding of the business key column name.
16. **Hub identifier:** the technical name of the referenced hub.

17. **Hub table physical name:** the physical table name of the reference hub.
18. **Hub primary key physical name:** the physical name of the primary key column in the referenced hub table.
19. **Hub business key physical name:** the name of the business key column in the hub.
20. **Hub business key column number:** the number within the column order of the business key in the hub. Required to calculate the correct hash value.
21. **Hub business key column data type:** the data type of the business key column in the referenced hub table. Can be used for automatically applying hard rules.
22. **Target column physical name:** the physical name of the target column (for descriptive data) in the satellite table.
23. **Target column data type:** the data type of the target column.
24. **Target column required:** indicates if the target column is nullable.
25. **Target column default value:** the default value of the target column (this should be defined by a hard rule).
26. **Target column description:** a technical description of the descriptive attribute in the target.
27. **Target column business description:** a textual description of the descriptive attribute in the target, using business terminology.
28. **Target column business name:** the common business name of the descriptive attribute that is recognized by business users.
29. **Target column business alias**: an alternative column name of the descriptive attribute that is recognized by business users.
30. **Target column acronym name:** a common acronym coding of the descriptive attribute's column name.
31. **Hard rules:** references to the hard rules that are applied within the loading process for this business key or descriptive attribute.

In order to create a metadata table that fully describes a hub satellite, one record needs to be added per descriptive data attribute and per business key in the parent hub. The latter is required for identification purposes, as Chapter 12, Loading the Data Vault, will discuss. Table 10.8 shows an example of a satellite definition for a hub with a composite business key.

Similar to the other metadata tables in this section, the table in Table 10.8 has been simplified by omitting some attributes that are not required for the explanation. It shows the definition of a satellite **SatPassenger** that is sourced from a source table called **PAX**. Two of the source columns, **COUNTRY** and **PASSPORT_NO**, are used to identify the entry in the Hub, which is defined by the composite business key consisting of **CountryCode** and **Passport ID**. The business keys are used to calculate the hash key, which is used in the satellite as a reference to the hub's primary key. It is possible to retrieve the corresponding hash key columns from the primary key. Naming conventions are not required to identify this column, because there should only be one hash key in the primary key. The other four fields are of a descriptive nature that means they describe the business key. In order to do so, they are being mapped to the attributes in the satellite.

This table clearly distinguishes between the business key definition and the descriptive data mapping. It is important to include the business key definition in this metadata table because of the required mapping between the source data and the business keys, which is used by the hash key computation when loading the data.

Table 10.8 Metadata for Capturing Source Tables to Data Vault Hub Satellite Entities

Satellite Identifier	Target Satellite Table Physical Name	Source Table Physical Name	Source Column Physical Name	…	Hub Table Physical Name	Target Column Physical Name	Hub Business Key Physical Name	Hub Business Key Column Number	Hard Rules
HS001	SatPassenger	PAX	COUNTRY		HubPassenger		CountryCode	1	HR3.4
HS001	SatPassenger	PAX	PASSPORT_NO		HubPassenger		PassportID	2	HR3.5
HS001	SatPassenger	PAX	FNAME			FirstName			HR3.6.1
HS001	SatPassenger	PAX	LNAME			LastName			HR3.6.2
HS001	SatPassenger	PAX	TITLE			Title			HR3.6.3
HS001	SatPassenger	PAX	SEX			Sex			HR3.6.4

10.2.7.4 Metadata for Loading Satellite Entities on Links

Hub satellite entities, which provide descriptive data for Data Vault links, are very similar to hub satellites. However, they are different in the identification of the entry in the parent link entity. The following attributes are used to define the metadata of Data Vault link satellites:

- **Data flow name:** the name of the data flow that is loading the target satellite.
- **Satellite identifier:** the technical name of the target satellite.
- **Target satellite table physical name:** the physical name of the target table in the Raw Data Vault.
- **Source table identifier:** the technical name of the source data table in the staging area.
- **Source column physical name:** the physical name of the source column in the source table that holds the business key or the descriptive data.
- **Source column data type:** the data type of the source column.
- **Source column required:** indicates if the source column allows NULL values.
- **Source column default value:** indicates the default value of the source column.
- **Source column computation:** if the source column is a computed field, provide the expression that computes the column value for documentation purposes.
- **Business key driving flag:** indicates if this business key is part of the driving key (if any).
- **Business key column description:** the technical description of the business key column.
- **Business key column business description:** a detailed textual description of the business key column in business terms.
- **Business key column business name:** the common business key column name that is recognized by business users.
- **Business key column business alias:** an alternative business key column name that is recognized by business users.
- **Business key column acronym name:** a common acronym coding of the business key column name.
- **Link identifier:** the technical name of the referenced parent link.
- **Link table physical name:** the physical table name of the reference link.
- **Link primary key physical name:** the physical name of the primary key column in the referenced link table.
- **Hub identifier:** the technical name of the referenced hub.
- **Hub table physical name:** the physical table name of the reference hub.
- **Hub reference number:** the number of the hub reference within the sort order of the hub references. This is required to calculate the correct hash key.
- **Hub primary key physical name:** the physical name of the primary key column in the referenced hub table.
- **Hub business key physical name:** the name of the business key column in the hub.
- **Hub business key column number:** the number within the column order of the business key in the hub. Required to calculate the correct hash value.
- **Hub business key column data type:** the data type of the business key column in the referenced hub table. Can be used for automatically applying hard rules.
- **Target column physical name:** the physical name of the target column (for descriptive data) in the satellite table.
- **Target column data type:** the data type of the target column.

- **Target column required:** indicates if the target column is nullable.
- **Target column default value:** the default value of the target column (this should be defined by a hard rule).
- **Target column description:** a technical description of the descriptive attribute in the target.
- **Target column business description:** a textual description of the descriptive attribute in the target, using business terminology.
- **Target column business name:** the common business name of the descriptive attribute that is recognized by business users.
- **Target column business alias:** an alternative column name of the descriptive attribute that is recognized by business users.
- **Target column acronym name:** a common acronym coding of the descriptive attribute's column name.
- **Hard rules:** references to the hard rules that are applied within the loading process for this business key or descriptive attribute.

In order to identify the satellite's parent link, it is required to obtain all business keys from the source system that are part of the link. The business keys become input to the hash key calculation. Other than that, the link satellite structure is similar to the hub satellite structure described in the previous section and contains only descriptive attributes. Table 10.9 shows a link satellite on a link that connects two hubs.

The satellite hangs off a link that connects two hubs: **HubCarrier** on one side and **HubAirport** on the other. The metadata information is required in order to map the source data to the business keys, which is required for hash key calculation. For the same reason, the business keys of a hub are ordered using the **Hub Business Key Column Number**. If one of the hubs were defined by more than one business key (or in other words, a composite business key), there would be two entries for this hub in Table 10.9, sequenced using this number. Similar to the metadata for loading satellite entities on hubs, there is no metadata required to identify the hash key column in the satellite because there should be only one hash key in the primary key of the satellite. The remaining three rows at the bottom of the table provide the metadata for the descriptive data attributes in the same manner as in hub satellites.

10.2.8 CAPTURING SOFT RULES

Section 10.2.4 discussed that hard rules only apply technical changes to the data loaded from the source system, for example ensuring that the data type fits the actual data and has the proper size. Hard rules never change the incoming raw data, except for data type alignment. Instead, the modification of data is left to soft rules, which change the value or the grain of the raw data. For example, content is altered by recalculation or is reinterpreted. The grain of the incoming data is changed by required aggregations to modify the source data in such a way that it fits the expected format. In other cases, a lower grain is required than the source system provides. In this case, the raw data needs to be broken up using predefined approximations defined by the business, using soft business rules.

However, the business changes. And so do the definitions of how the raw data should be modified to create information that is useful for the business, because the business rule definitions document the perception of truth by which the business operates on a day-to-day basis. But this truth changes more often than the market changes, for example when the business learns more about the market.

Table 10.9 Metadata for Capturing Source Tables to Data Vault Link Satellite Entities

Satellite Ident.	Target Satellite Table Physical Name	Source Table Physical Name	Source Column Physical Name	…	Link Ident.	Hub Table Physical Name	Hub Ref. No.	Target Column Physical Name	Hub Business Key Physical Name	Hub BK Column No.	Hard Rules
LS001	SatFixedBaseOp	FB_OPS	CARRIER	…	L001	HubCarrier	1		Carrier	1	HR22.1
LS001	SatFixedBaseOp	FB_OPS	AIRPORT	…	L001	HubAirport	2		Airport	1	HR1.2.5
LS001	SatFixedBaseOp	FB_OPS	NAME		L001			Name			HR1.3.4
LS001	SatFixedBaseOp	FB_OPS	DESC		L001			Description			HR1.3.4
LS001	SatFixedBaseOp	FB_OPS	LOC		L001			Location			HR1.3.4

However, modifying any part of the data warehouse requires changing, or at least testing, all parts downstream of the data warehouse that depend on the modified part. That is the major reason why soft rules should be placed in the architecture as close as possible toward the business, downstream in the data warehouse architecture. Because the definitions of how to modify the raw data to retrieve useful information change so frequently, we need to reduce the number of dependencies on these fluid parts of the data warehouse. And that is why the hard rules described in section 10.2.4 should only modify technical aspects of the raw data, but not the data itself.

When capturing business rules, organizations often depend on free-form descriptions, because the soft rule definitions tend to be complex and require verbal explanations in written form. The following sections will describe some approaches to describing soft rules using text or graphical notations that help the business to define the business rules in an easy to interpret format. Regardless whether the business decides to use an unstructured, free-text format for describing the business rules, some text-based or graphical notation, or a combination of both, in any case, metadata should be captured that describes the collection of soft rules in the data warehouse. Such a metadata table should consist of the following attributes [11,8]:

- **Data flow name (optional):** the data flow that is implementing this business rule.
- **Rule identifier:** the technical name of the business rule.
- **Name:** a name for the soft rule understandable by the business.
- **Description:** a technical description of the business rule.
- **Business description:** a textual description of the soft rule in business terms.
- **Definition:** the actual definition of the business rule, using a text-based or graphical notation.
- **Business area:** the functional business area (or business owner) that defines this business rule.
- **Topic:** a given topic that this business rule is part of.
- **Task/rule set:** a more detailed task or rule set that this business rule is part of.
- **Priority:** the priority of the business rule: either "must-have," "should-have," "could-have," or "won't have."
- **Motivation:** defines the motivation why this business rule is defined, for example "data integrity," "security policy," "customer relationship standards," etc.
- **Source:** The source of the soft rules definition (for example, project documentation).
- **Classification:** Table 10.10 lists the potential classifications.
- **Implementation type:** Table 10.11 lists the potential rule types.
- **Keywords:** keywords that help to find the soft rule from the business rule collection.
- **Defined:** the name of the person who has defined the soft rule and the date of definition.
- **Approved:** the name of the person who approved the soft rule and the date of approval.
- **Related rules:** relates this soft rule with other hard rules.
- **Example:** provides example inputs and their respective outputs.

The **data flow name** is an optional metadata attribute because most business rules are implemented in virtual business vault entities or information mart entities. In this case, the record source column is used to refer to the **rule identifier** as Chapter 12 demonstrates. Note the similarity to the metadata attributes for the definition of hard rules in section 10.2.4. The following table lists the possible rule classifications:

The following table provides commonly used rule types:

Again, this table is similar to the hard rule types in Table 10.3. However, Table 10.3 has been condensed to rule types that make sense to hard rules. In contrast, Table 10.11 provides more rule types, because the modification of data is allowed in soft rules.

Table 10.10 Business Rule Definitions [8]

Rule Classification	Description	Example
Term	A noun or noun phrase with an agreed upon definition.	Passenger "female" Preferred dish
Fact	Connects terms into observations relevant for the business.	Passenger's preferred dish is "vegetarian".
Mandatory constraint	Expresses an unconditional circumstance that must be true or not true to complete the business event with integrity.	A flight must have a valid flight number to be included in the **number of flights** measure.
Guideline	Expresses a warning about a circumstance that should be true or not true to complete the business event.	A flight should have a minimum of 2 passengers to be counted as a flight in the **number of flights** measure.
Action enabler	Initiates another business event, message or other activity if the given conditions are true.	If a passenger doesn't show up for a flight, issue a security warning.
Computation	Defines the calculation rule to retrieve the value of a term, including sum, difference, product, quotient, count, maximum, minimum, and average.	The income per passenger is calculated by dividing the total annual income by the number of annual passengers.
Inference	Establishes a new fact if the given conditions are true.	If a passenger has more than 5000 air miles flown, then the customer is of preferred status.

Table 10.11 Soft Business Rule Types [8]

Soft Rule Type	Description
Restriction	These rules define business specific constraints on the data that is allowed or not allowed to use by the business. They state what data is not to leave the Raw Data Vault.
Behavior	These rules express how the system should react to given situations.
Deduction	These rules define how to derive or calculate useful information from the raw data.
Presentation	They define how to present the information to the business user.
Instruction	These rules define how to handle business events, e.g. how to react to passengers not showing up for a flight.
Warning	These rules define what kind of error should be logged and how.
Other	All other rules that don't fit into the standard rule types.

Similar to metadata tables in other sections of this chapter, the metadata attributes and the values in the tables presented in this section are suggestions only. Each project team or organization has to adapt these lists and come up with their own definitions. Use the information given in this section as a starting point.

10.2.9 CAPTURING DATA VAULT TABLES TO INFORMATION MART TABLE MAPPINGS

The mapping between the Data Vault (both Raw Data Vault and Business Data Vault) to information marts is a complex procedure. In many cases, soft business rules with inputs from the Data Vault and outputs in the information mart are defined and documented (refer to section 10.2.8). A soft rule moves the data from the Data Vault into the information mart by transforming, recalculating, aggregating or interpreting the data. To document the inputs and the outputs of the soft rules, the following attributes should be included in the metadata table:

- **Data flow name:** the name of the data flow that is implementing the soft rule.
- **Target table identifier:** the technical name of the target table (such as dimension or fact table if a dimensional model is being built) based on technical numbering.
- **Target table physical name:** the physical name of the target table in the information mart.
- **Source identifier:** the technical name of the source hub, link, or satellite.
- **Source table physical name:** the physical name of the source table in the Data Vault.
- **Source column physical name:** the physical name of the source column in the source table.
- **Source column data type:** the data type of the source column.
- **Source column required:** indicates if the source column allows NULL values.
- **Target column physical name:** the physical name of the target column (for descriptive data) in the satellite table.
- **Target column data type:** the data type of the target column.
- **Target column required:** indicates if the target column is nullable.
- **Target column default value:** the default value of the target column.
- **Target column description:** a technical description of the target column.
- **Target column business description:** a textual description of the target column, using business terminology.
- **Target column business name:** the common business name of the target column that is recognized by business users.
- **Target column business alias:** an alternative name of the target column that is recognized by business users.
- **Target column acronym name:** a common acronym coding of the column name.
- **Soft rule:** references to the soft rules that are applied within the loading process.

Note that this metadata is not only used to describe the detailed metadata for soft business rule execution towards information marts, but also for additional tables in the Business Vault, such as computed satellites.

Table 10.12 shows an example of a Data Vault tables to information mart tables mapping.

Table 10.12 Metadata for Capturing Data Vault Tables to Information Mart Tables

Target Table Identifier	Source Table Physical Name	Source Column Physical Name	Source Data Type	…	Target Table Physical Name	Target Column Physical Name	Target Data Type	Soft Rule
D001	HubAirline	AirlineHashKey	Char(32)	…	DimAirline	AirlineKey	Char(32)	SR443.2.1
D001	HubAirline	AirlineID	Integer	…	DimAirline	AirlineID	Integer	SR443.2.1
D001	SatAirline	Name	VarChar(50)	…	DimAirline	Name	VarChar(50)	SR443.2.1
D001	SatAirline	Abbreviation	VarChar(3)	…	DimAirline	Abbreviation	VarChar(3)	SR443.2.1
F001	BrConnection	FlightNo	Integer	…	FactConnection	FlightNo	Integer	SR221.2
F001	BrConnection	CarrierHashKey	Char(32)	…	FactConnection	CarrierHashKey	Char(32)	SR221.2
F001	BrConnection	SrcAirport	Char(32)	…	FactConnection	SrcAirport	Char(32)	SR221.2
F001	BrConnection	DestAirport	Char(32)	…	FactConnection	DestAirport	Char(32)	SR221.2
F001	BrConnection	Duration	Integer	…	FactConnection	Duration	Integer	SR221.2

There are two dimensional target tables defined: one dimension **DimAirline** and a fact table **FactConnection**. The dimension is sourced from a Data Vault hub **HubAirline** and an accompanying satellite **SatAirline**. All three source attributes from the hub and the satellite are directly mapped to a target column in the information mart. The soft business rule, which is further defined in **SR443.2.1,** might change the values from the source before writing the data into the dimension.

The loading process of the fact table, identified as F001, is sourced from a bridge table. In this case, all dimension references, business keys (loaded as dimension attributes) and measures are directly sourced from this source bridge table. The hash keys in the bridge table already reference the later dimension hash keys, which works seamlessly: in the case of Type 1 dimensions, as in Table 10.12, the hash key is directly sourced from the hub. In the case of Type 2 dimensions, it would be sourced from PIT tables. Chapter 14, Loading the Dimensional Information Mart, demonstrates how to load virtual and materialized information mart entities.

Note that there is one set of metadata entries per soft rule. For each soft rule, there are multiple inputs and outputs defined. The relationship between inputs and outputs is multiple to multiple (m:n) with both sides of the relationship being optional. It is possible to define an input to the soft rule that has no direct associated output (for example, because the input is used in a filter condition). Similarly, an output might be defined using a constant value or calculated using a complex formula that might not be traced back to a source column. This is the case for the second Carrier target column of **FactConnection** in Table 10.12.

10.2.10 CAPTURING REQUIREMENTS TO INFORMATION MART TABLES

Chapter 3 has already introduced the requirements for information mart tables cross-reference, which is provided again in Table 10.13.

The cross-reference table describes which information mart dimension or fact tables are used by a given report or OLAP item. There are three reports in this example: *Passenger*, *Airplane Utilization* and *Connections*. While the *Passenger* report uses only the *Passenger Information* table in the information mart, the *Connections* report uses the *Connections* and the *Airplanes* table. These entity names are the logical names; the physical names are also provided as a reference. It also references the requirements

Table 10.13 Requirements to Information Mart Table Example

Requirements to Target Map XREF

Requirement Document: MyDoc

Table

Logical Name	Physical Name	Business Key	Passenger (B3.1)	Airplane Utilization (B3.2)	Connections (B3.3)
Passenger Information	PASSENGER		X		
Connections	CONNECTION				X
Airplanes	AIRPLANE			X	X

documentation. It is also possible to add the attributes in the information mart, similar to the requirements to source table in section 10.2.6.

The major value of this cross-reference is:

1. To identify the information mart artifacts which are affected by a change of requirements.
2. To identify the requirements which are affected when changing an information mart artifact.

While these advantages sound interchangeable at first glance, they are important, in both orders. The Data Vault 2.0 methodology embraces change, like other agile methodologies. Requirements are first affected by such change. A change requirement will affect one or more artifacts in the information marts, such as a dimension in an OLAP cube. If this dimension is modified, it affects all requirements that are related to this artifact. Using this cross-reference table, we can track such dependent changes in the data warehouse.

10.2.11 CAPTURING ACCESS CONTROL LISTS AND OTHER SECURITY MEASURES

Most database systems allow the definition of users, roles and access control lists (ACL) on individual databases, tables and views of the data warehouse. However, in order to effectively secure the data in the data warehouse, several actions are required to be performed [12]:

- **Identify the data:** this step is already complete when creating the suggested metadata described in the previous sections.
- **Classify data:** for security purposes, the initial metadata used in step 1 and described in the previous sections should be modified to include a data sensitivity attribute.
- **Quantify the value of data:** in order to estimate the costs of security breaches, the value of data assets needs to be quantified first. It might be hard to determine this value, especially because errors in the data warehouse might lead to erroneous business decisions.
- **Identify data security vulnerabilities:** it is also hard to determine the potential vulnerabilities in a data warehouse, due to the long-lifespan of the data storage and unknown future use of the data.
- **Identify data protection measures and their costs:** for each thread identified in the previous action, the potential remedies are identified and priced.
- **Selecting cost-effective security measures:** the value of the data and the severity of the thread are used to level the identified security measures.
- **Evaluating the effectiveness of security measures:** the effectiveness of the security measures needs to be addressed in the final step.

Without going into more detail regarding these activities, it becomes possible to identify the following metadata attributes that are required to define the security measures:

- **Data sensitivity:** the classification of the data regarding security as defined in step 2. Table 10.14 lists and describes an example of a data classification matrix.
- **Data value:** the value of the data as quantified in step 3.
- **Potential remedies per security vulnerability and data item:** the potential remedies as identified in step 4.
- **Thread severity:** the severity of the thread as identified in step 4.

Table 10.14 provides an example of a data classification matrix that can be used to classify the raw data in the data warehouse:

Table 10.14 Example Data Classifications [13]

	Prohibited	Restricted	Confidential	Unrestricted
Classification Guideline	Data is classified as prohibited if required by law or any other regulation or if the organization is required to self-report to the government or the individual if data has been inappropriately accessed.	Data is classified as restricted if the data would qualify as prohibited but the organization determines that personnel could not effectively work with the data when not stored in the data warehouse.	Data is classified as confidential if it is not classified as prohibited or restricted. Exceptions to this rule are listed under Classification of Common Data Elements.	Data is classified as unrestricted if not classified as prohibited, restricted, or confidential.
Classification of Common Data Elements	Checking account numbers	Export controlled data	Contract data	Data authorized to be available in public
	Credit card numbers	Health information	Emails	Data in the public domain
	Driver license numbers	Passport numbers	Financial data	
	Health insurance policy numbers	Visa numbers	Internal memos	
	Social security numbers		Personnel records	
Access Protocol	Access only with permission and with signed non-disclosure agreements (NDA).	Access restricted only to those permitted under law, regulations or organizational policies and with a need to know for carrying out their assigned work and duties.	Access restricted to user with a need to know and with the permission of the data owner or custodian.	Access to unrestricted data is granted to everyone.
Transmission	Approved encryption is required in order to transmit the data through a network.	Approved encryption is required in order to transmit the data through a network.	Approved encryption is recommended in order to transmit the data through a network.	No encryption is required to transmit unrestricted data through a network.
Storage	It is not allowed to store the data in any computing environment, including the data warehouse, unless especially approved by the organization. If it is explicitly allowed, approved encryption is required. It is not allowed for a third party to process or store the data without explicit approval.	Approved encryption is required to store the data in the data warehouse. It is not allowed for a third party to process or store the data without explicit approval.	Encryption is recommended but not required. The data owner might decide on required encryption. Data might be processed or stored by a third party if a valid contract with the party exists.	No encryption is required.

While Table 10.14 provides a list of data classifications that can be used in data warehouse environments, it is still unclear how these levels can be applied to a Data Vault 2.0 model, among the other metadata attributes.

Classified data might be stored in all Raw Data Vault entities. Business keys might be classified as Table 10.14 shows: a credit card number or social security number by itself is already considered as prohibited and should only be stored in a data warehouse if approved by the organization. In that case, the hub that holds this business key should be elevated to the appropriate sensitivity level, restricted (because the organization allowed the storage of the data in the data warehouse). The same applies to satellites which hold descriptive data that might be classified as restricted or confidential. A best practice is to separate the data by sensitivity level. This can be done using satellite splits. For each resulting satellite, the database engine can enforce separate security controls.

However, the data is not only stored in Raw Data Vault entities. There are dependent entities in the Business Vault and in information marts that provide access to information which is based on the raw data provided by the Raw Data Vault, either in a virtual or materialized form. In both cases, the information should be classified to the same sensitivity level as the underlying raw data. When information is generated from multiple entities in the Raw Data Vault, each of them classified with different sensitivity levels, the most restrictive sensitivity level should be used for the derived information as a whole. For example, if a computed satellite in the Business Vault consolidates raw data from a satellite classified as **unrestricted** and another satellite classified as **confidential**, the computed satellite should be classified as **confidential** as well.

Such a classification of derived entities is not necessary if the grain of data is modified, for example during an aggregation. In this case, the classification of the computed satellite might be actually less restrictive because detailed information is not available anymore [14].

10.3 IMPLEMENTING THE METRICS VAULT

The Metrics Vault is used to capture process metrics from the ETL control and data flows. It is an optional component, not necessary to implement a data warehouse. However, it is very helpful with [15]:

- **Error inspection:** the Metrics Vault is used to identify errors and their causes, which is required to restart the ETL process.
- **Root cause analysis:** using the Metrics Vault, an error can be analyzed in order to find the root cause of the failure and put an improved solution in place in order to prevent the same error from happening again in the future.
- **Performance metrics:** the execution and performance metrics can be used to observe the size and growth rates of data and address negative performance trends before they cause any failure during run-time.

Various metrics are captured in order to support these goals, for example the rows loaded successfully, rows rejected, amount of time to load, etc [2]. Everything related to the following areas should be captured by the Metrics Vault in the highest level of detail:

- **Timing metrics:** the start and end time of the ETL processing, including the elapsed time [2].
- **Performance metrics:** read and write throughput provides information about the ETL performance itself, by reporting the rows per second read from the source or written to the destination [2].

- **Volume metrics:** the total numbers of rows read from the source, written to the destination or rejected during processing provides useful information for debugging and administrative purposes [2].
- **Error metrics:** provide information about the number of rejected or otherwise erroneous records per source or destination. Note that this doesn't include raw data which is left for the Error Mart (refer to section 10.5).
- **Frequency metrics:** involve how often a data source delivers data and how often ETL processes are run.
- **Dependency chains:** dependency chains involve waiting times for dependent ETL processes. These waiting times should be measured as a basis for improvement.

The Metrics Vault should be modeled using the Data Vault 2.0 Model in order to integrate metrics from various sources: while a lot of metrics are sourced from the ETL tool such as SSIS, other metrics might be integrated from the server or network infrastructure. Table 10.15 lists typical metrics of the server and network infrastructure. Often, these sources of metrics data are integrated in an agile fashion, similar to the data warehouse itself. That is due to the fact that, while analyzing errors and performance bottlenecks, more and more potential sources of failure are analyzed during a root cause analysis. Whenever a potential source of failure is analyzed, more and more data needs to be integrated into the Metrics Mart to be available for analysis.

While the process metrics are obtained while the ETL control flow is running, the information from Table 10.15 has to be obtained from the server and network devices. The **WMI Data Reader Task** [20] in SSIS can be used to obtain selected metrics during a running ETL control flow. However, if the infrastructure should be monitored more frequently (for example, to collect CPU usage per second), the database of a monitoring software should be integrated into the Metrics Vault.

Table 10.15 Hardware and Network Infrastructure Metrics [16–18]

Metric Name	Metric Type	Description
Backup Age	Backup	The number of hours a database was last backed up.
Backup Time	Backup	The running time of the last backup.
Log Backup Age	Backup	The number of hours a database log backup was created.
Log Backup Time	Backup	The running time of the last log backup.
Batch Requests	Database	The number of batch requests per second.
Checkpoint Pages	Database	The number of flushed dirty pages per second.
Data Buffer Cache Hit Ratio	Database	The percentage of pages found in the SQL Server buffer pool [19].
Database Online	Database	Indicates if the database is online and accepts queries.
Deadlocks	Database	The number of deadlocks per second.
Free List Stalls	Database	The number of free list stalls per second.
Full Scans	Database	The number of full table scans per second.
Initial Compilations	Database	The number of initial compilations per second.
Latch Wait Time	Database	The average waiting time before a latch request is granted.
Latch Waits	Database	The number of unfulfilled latch-requests per seconds.

(Continued)

Table 10.15 Hardware and Network Infrastructure Metrics [16–18] *(cont.)*

Metric Name	Metric Type	Description
Lazy Writes	Database	The number of lazy writes per second.
Lock Timeouts	Database	The number of lock requests per second, which resulted in a timeout.
Lock Waits	Database	The number of unsatisfied lock requests per second.
Page Life Expectancy	Database	The average time a page is kept in main memory.
Recompilations	Database	The number of recompilations per second.
Total Server Memory	Database	The amount of main memory reserved for the database server.
Transactions	Database	The number of transactions per second.
Availability	Network	The percentage of the time a network resource is available for use.
Delay	Network	The amount of time for a packet to traverse (one-way or round trip) a network, network segment or network device.
Error Rate	Network	The percentage of packets or bits that contain errors on a network link, network segment or network device.
Throughput	Network	The amount of data that can be sent on or through a network resource in a given time period.
Utilization	Network	The percentage of network resource utilization.
Failed Jobs	Scheduler	The number of jobs that did not successfully exist in a given time period.
Free Memory	Server	The total amount of available physical RAM.
Total CPU Usage	Server	The total amount of CPU power currently in use.
Total I/O Usage	Server	The total amount of available I/O currently in use.
Active Users	Session	The total number of active database users.
Average Active Time	Session	The sum of database time over all sessions, divided by the elapsed time.
Connection Time	Session	The total time to login.
Response Time	Session	The response time of a simple query, that is the time it takes to send the query from the client to the database, executing it and returning the result.
Total Users	Session	The total number of users connected to the database.
Data File Auto Growth	Storage	The number of data file auto growth events in a given time period.
Data File Auto Shrinks	Storage	The number of data file auto shrink events in a given time period.
Database File DBCC Shrinks	Storage	The number of DBCC file shrink events in a given time period.
Free Space	Storage	The free space in a database.
Log File Auto Growths	Storage	The number of log file auto growth events in a given time period.
Log File Auto Shrinks	Storage	The number of log file auto shrink events in a given time period.

The next sections demonstrate how to capture ETL and WMI metrics in a running ETL control flow and how to model the underlying Metrics Vault.

10.3.1 CAPTURING PERFORMANCE DATA IN SQL SERVER INTEGRATION SERVICES

When using Microsoft SSIS as the ETL tool, it is easy to capture basic metrics on the processed ETL control and data flows. In order to log messages, select **Logging** in the context menu of the control flow to set up the logging. The dialog in Figure 10.8 will be shown.

By default, there is no logging defined for a SSIS control flow. In order to set up the destination of log events, select one of the following provider types in the selection box [21]:

- **SSIS log provider for SQL Server Profiler:** this provider writes the log events into a trace file that can be imported to SQL Server Profiler. SQL Server Profiler can be used to replay packages in a test environment step by step [15].
- **SSIS log provider for XML files:** this provider writes the log events into a XML file which include schema information that is helpful when importing the events into another application.
- **SSIS log provider for SQL Server:** this provider sends the log events to an OLE DB connection, for example to store the event data in a SQL Server table. When the provider accesses a database

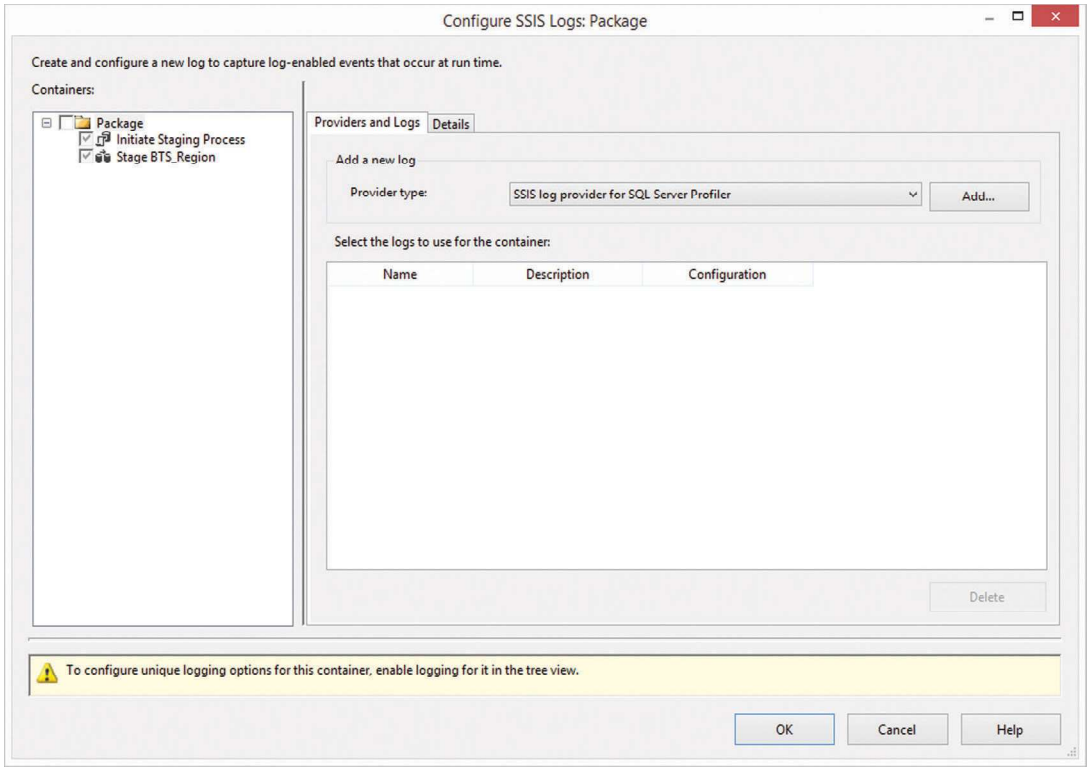

FIGURE 10.8

Configure SSIS logs on package (initial dialog).

for its first time, it checks if a destination table already exists. The destination table (and a stored procedure for writing into it) is created if not [15].

- **SSIS log provider for Windows Event Log:** this provider sends the log events into the Application event store of the Windows Event log. The log is easily accessible by Windows system administrators and can be viewed remotely [15].
- **SSIS log provider for Text files:** this provider writes the log events into a comma-separated file (CSV) which can be easily imported into applications for analysis, including Microsoft Excel or database applications such as Microsoft Access [15].

In addition, it is also possible to develop a custom log provider [22].

To store the log events in a relational Metrics Vault, choose the **SSIS log provider for SQL Server** and select the **Add**… button. A new provider is added to the list below (note that you can configure multiple log providers). Select <**New connection…**> in the selection box behind the configuration column. This will open the dialog to set up a new OLE DB connection (Figure 10.9).

Once the log provider connection has been set up, check the package in the left tree and activate the log provider in the provider list by activating the check mark on the provider (in front of the row), as Figure 10.10 shows.

After setting up the log provider connection to the Metrics Vault, the event types that should be logged have to be selected on the **Details** pane of the dialog (Figure 10.11).

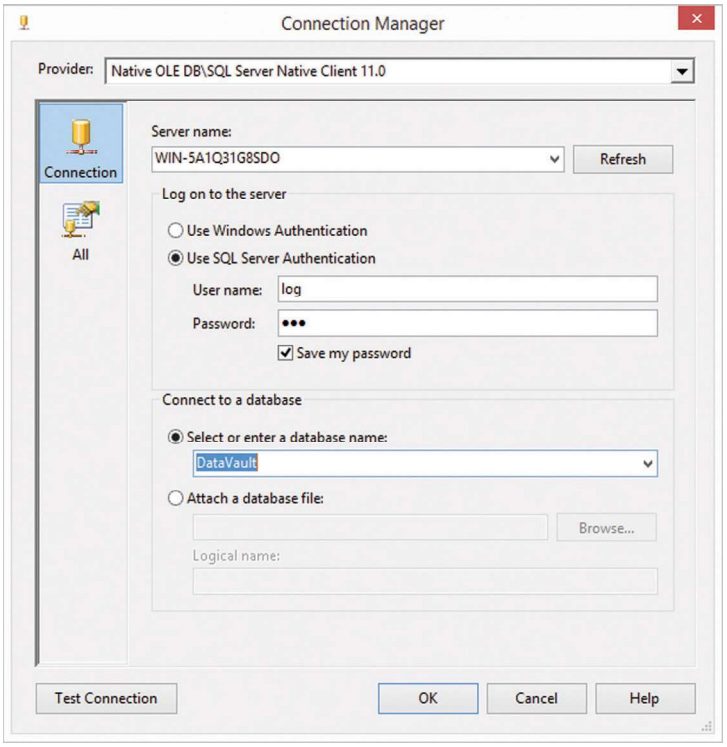

FIGURE 10.9

Connection manager to setup log provider connection.

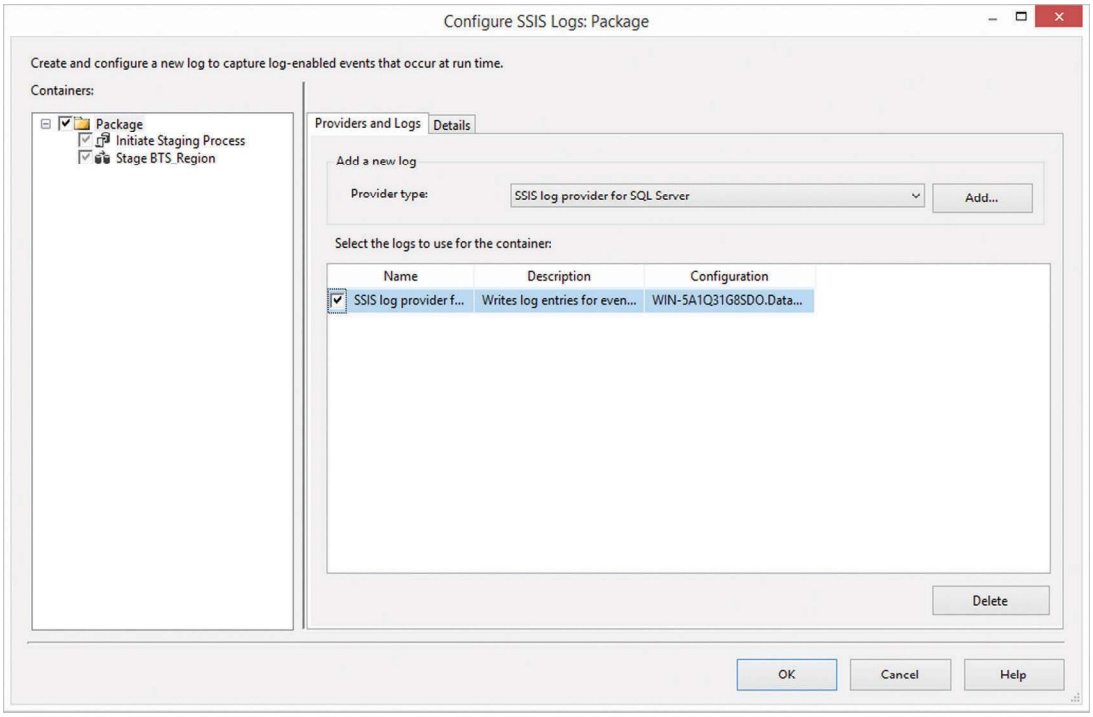

FIGURE 10.10

Configure SSIS logs on package (final dialog).

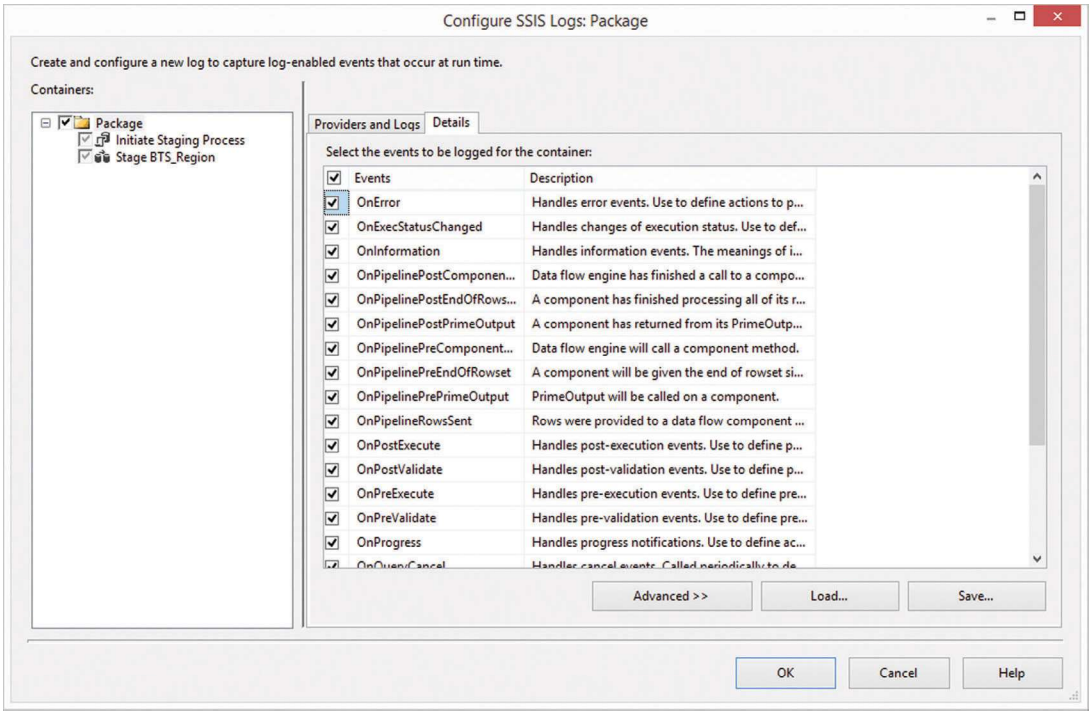

FIGURE 10.11

Log event types available for logging in SSIS.

The following log event types are available [21]:

1. **OnError:** events of this type are raised when errors occur.
2. **OnExecStatusChanged:** events of this type are raised when tasks (not containers) are suspended or resumed during debugging sessions.
3. **OnInformation:** events of this type report the information collected during the validation and execution of executables.
4. **OnPostExecute:** events of this type are raised when executables have finished execution.
5. **OnPostValidate:** events of this type are raised when executables have been validated.
6. **OnPreExecute:** events of this type are raised immediately before executing executables.
7. **OnPreValidate:** events of this type are raised before the validation of executables starts.
8. **OnProgress:** events of this type are raised when executables have progressed in a measurable amount.
9. **OnQueryCancel:** events of this type are raised when it is possible to cancel execution of tasks.
10. **OnTaskFailed:** events of this type are raised when a task fails.
11. **OnVariableValueChanged:** events of this type are raised when a variable changes its value.
12. **OnWarning:** events of this type are raised when a warning occurs.
13. **PipelineComponentTime:** this event type sends a log entry for each phase of validation and execution, including the processing time of each phase.
14. **Diagnostic:** this event type sends diagnostic information.

In addition to these standard event types, many tasks and containers define additional event types that are raised when something happens inside the task or container [21]. For example, the data flow task provides the following event types [23]:

1. **BufferSizeTuning:** this event is raised when the size of the data flow buffer has been changed.
2. **OnPipelinePostEndOfRowset:** events of this type are raised when a component of the data flow has reached the end of the rowset.
3. **OnPipelinePostPrimeOutput:** this event is raised when a component has finished its last prime output.
4. **OnPipelinePreEndOfRowset:** this event is raised before a component of the data flow is reaching the end of the rowset.
5. **OnPipelinePrePrimeOutput:** this event is raised before a component is processing its last prime output.
6. **OnPipelineRowsSent:** this event is raised in order to report the number of rows sent to a component in the data flow task.
7. **PipelineBufferLeak:** this event is raised if a component has not released all buffers, which indicates buffer leaks.
8. **PipelineComponentTime:** this event is used to report the amount of time (in milliseconds) that a component in the data flow has spent in each of its major processing steps (Validate, PreExecute, PostExecute, ProcessInput, and ProcessOutput).
9. **PipelineExecutionPlan:** this event is raised to report the execution plan of the data flow.

10. **PipelineExecutionTrees:** this event is raised to report the execution trees of the layout in the data flow. Execution trees are used to build the execution plan.
11. **PipelineInitialization:** this event is raised to provide initialization information about the data flow task.

Whenever one of the selected events occurs, SSIS will send information about the event to the log provider destination. Using the Advanced button, it is possible to configure the attributes per event that should be logged. The following attributes are logged by default and can be configured (deselected) in the dialog [21]:

- **Computer:** indicates the name of the computer where the event has occurred.
- **Operator:** indicates the user who has launched the package.
- **SourceName:** the name of the container or task in which the event was raised.
- **SourceID:** the unique identifier of the container or task in which the event was raised.
- **ExecutionID:** the GUID of the package instance used for execution.
- **MessageText:** the message text of the logged event.
- **DataBytes:** this byte array stores additional binary data for the event. Used rarely in SSIS logging.

In addition to these optional attributes, the following attributes are always included in the event information [21]:

1. **StartTime:** the start time of the task or container.
2. **EndTime:** the end time of the task or container.
3. **DataCode:** indicates the execution result of the task or container. Possible values are given in Table 10.16.

Note that the **MessageText** attribute is often used to encode other, valuable information, such as the records written to the target. When writing the raw data into the Metrics Vault, the **MessageText** should be split and loaded into separate satellites, following standard Data Vault practices by capturing the raw data in the actual data types of the source system.

Once the event types are selected and the dialog has been confirmed, the selected metrics are captured during execution of the control flow.

In this example, a log user has been created on the database server. When connecting to the database for the first time, SSIS creates two objects in the database:

1. a table named **sysssislog** that becomes the target of all event data [24].
2. a stored procedure **sp_ssis_addlogentry** which is used to write data into the **sysssislog** table.

Table 10.16 Possible DataCode Values

Data Code	Description
0	Success
1	Failure
2	Completed
3	Cancelled

SSIS will call the stored procedure whenever an event occurs that is configured for logging. Interestingly, the table doesn't need to exist; only the stored procedure is required. It is also not required that the table be stored in the **dbo** schema of the database. A modified stored procedure can exist in any schema possible; the only setting that needs to be adjusted is the **default schema** setting of the logging user. That is the reason why another user has been configured in Figure 10.9. The default schema for this user is set to the **log** schema where the stored procedure is located.

The default implementation of the stored procedure just inserts the incoming data into the **sysssislog** table. However, this behavior can be modified to write data into another target or different target structure… the Metrics Vault. The tables shown in Figure 10.12 have been created to capture the events from SSIS.

The central table in the model is a transaction link **TLinkEvent**. It references four hubs: **HubEventType**, **HubComputer**, **HubOperator**, and **HubSource**. In addition, it contains an **id** attribute among some other descriptive attributes that are included in the transaction link. Other descriptive attributes, especially the large attributes such as the **message** and **databytes**, are stored in a transaction satellite **TSatEvent** to keep the transaction link as small as possible. Because the **message** attribute uses a specific format for some event types, additional satellites have been added to capture the message in the best format possible. For example, **TSatDiagnosticExEvent** stores the message in an xml field because this event type is using XML formatted messages. On the other hand, many other SSIS event types are using a text-based format, delimited by colons. To support easy access to the elements of the message, **TSatOnPipelinePreComponentCallEvent** is used to store the separate elements. These satellites are transaction satellites without a **LoadEndDate**, thus not supporting any updates by inserting new versions of the data and end-dating old records. However, for the given task, which is storing unmodifiable events, these entities are the optimal choice. There is one standard satellite in the model, **SatSource,** which allows changing the name of a task or component in SSIS. Referential integrity is implemented using foreign keys but disabled.

The transaction link in the Metric Vault implements the structure described in Chapter 5, Intermediate Data Vault Modeling:

```
CREATE TABLE [log].[TLinkEvent](
       [EventHashKey] [char](32) NOT NULL,
       [LoadDate] [datetime2](7) NOT NULL,
       [RecordSource] [varchar](50) NOT NULL,
       [EventTypeHashKey] [char](32) NOT NULL,
       [ComputerHashKey] [char](32) NOT NULL,
       [OperatorHashKey] [char](32) NOT NULL,
       [SourceHashKey] [char](32) NOT NULL,
       [id] [bigint] NOT NULL,
       [executionid] [uniqueidentifier] NOT NULL,
       [starttime] [datetime] NOT NULL,
       [endtime] [datetime] NOT NULL,
       [datacode] [int] NOT NULL,
 CONSTRAINT [PK_TLinkEvent] PRIMARY KEY NONCLUSTERED
(
       [EventHashKey] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [INDEX]
) ON [DATA]
```

Four hash keys are calculated to reference the hubs in the model. The primary key is on the **EventHashKey**, which is based on the business keys from the referenced hubs and the **id** attribute which

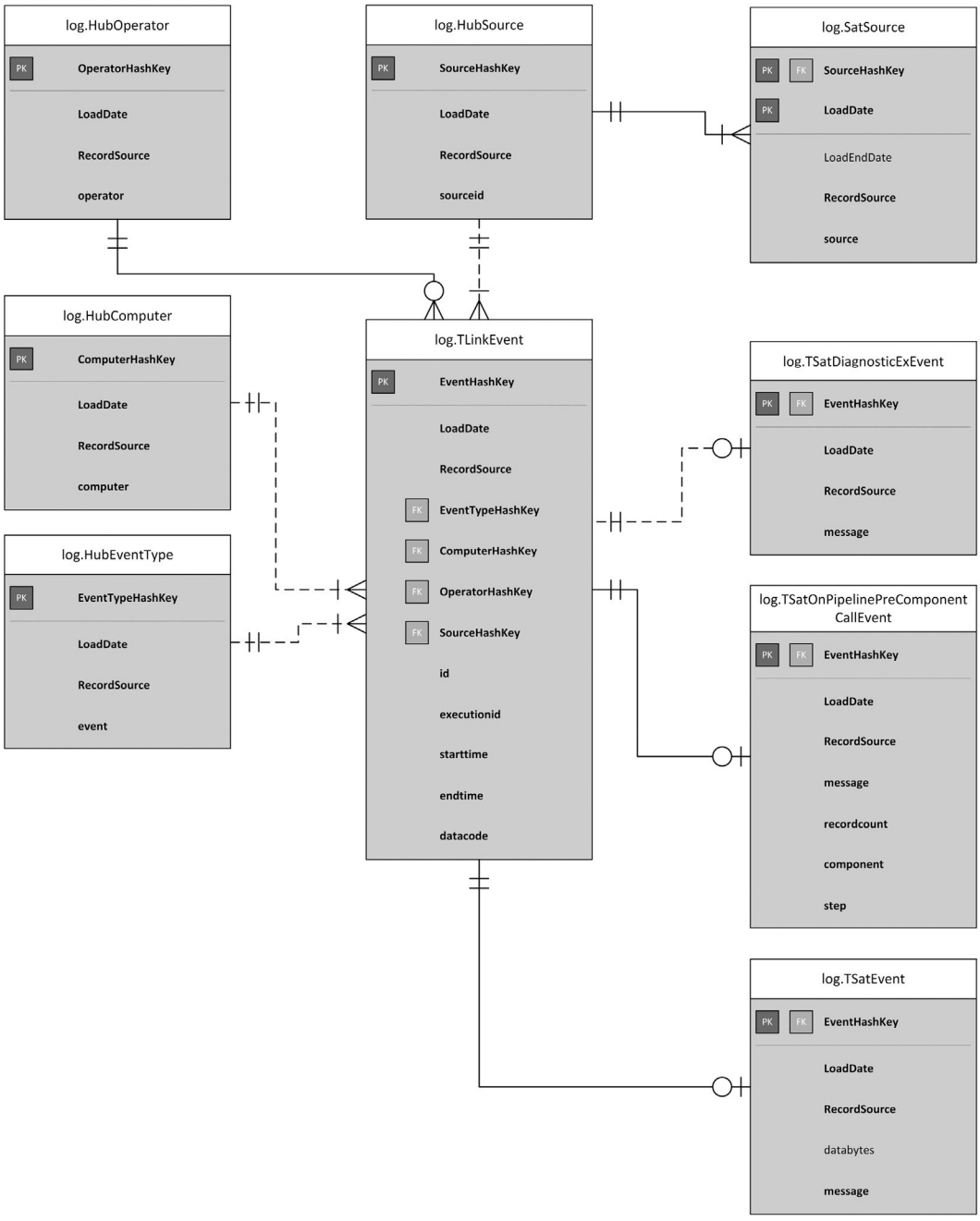

FIGURE 10.12

Metrics Vault for SSIS logging (physical design).

is required for uniqueness of the primary key. The implicit index on the primary key is stored in the **INDEX** filegroup and the data table itself is stored in the **DATA** filegroup. Refer to Chapter 8, Physical Data Warehouse Design, for details. The other attributes of the transaction link (**executionid**, **starttime**, **endtime**, and **datacode**) are included in this table because they are interesting for later aggregations based on the link. Also, they don't increase the size of the row much. The remaining descriptive attributes given to the stored procedure are stored in the satellite **TSatEvent**:

```
CREATE TABLE [log].[TSatEvent](
       [EventHashKey] [char](32) NOT NULL,
       [LoadDate] [datetime2](7) NOT NULL,
       [RecordSource] [varchar](50) NOT NULL,
       [databytes] [image] NULL,
       [message] [nvarchar](2048) NOT NULL,
 CONSTRAINT [PK_TSatEvent] PRIMARY KEY NONCLUSTERED
(
       [EventHashKey] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [INDEX]
) ON [DATA] TEXTIMAGE_ON [DATA]
```

Because this satellite is a nonhistorized satellite, it doesn't contain a **LoadEndDate**. The implicit index on the primary key is stored in the **INDEX** filegroup again, while the data is stored in the **DATA** filegroup. The other satellites are just modifications of this satellite and include only data for specific event types. **SatSource**, however, is a standard satellite, because it allows and tracks modifications in the source system (SSIS in this case):

```
CREATE TABLE [log].[SatSource](
       [SourceHashKey] [char](32) NOT NULL,
       [LoadDate] [datetime2](7) NOT NULL,
       [LoadEndDate] [datetime2](7) NULL,
       [RecordSource] [varchar](50) NOT NULL,
       [source] [nvarchar](1024) NOT NULL,
 CONSTRAINT [PK_SatSource] PRIMARY KEY NONCLUSTERED
(
       [SourceHashKey] ASC,
       [LoadDate] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [INDEX]
) ON [DATA]
```

Other than the **LoadEndDate**, the structure is very similar to the other satellites. The next chapter covers how to calculate the hash keys and how to end-date the satellites, which is required for **SatSource**.

The described model is loaded by modifying the **sp_ssis_addlogentry** stored procedure. Instead of writing all events directly into the **sysssislog** table, the events are split into business keys, relationships and transactions, and descriptive attributes and loaded into the tables of the Metrics Vault. Apart from creating a custom log provider in SSIS, this option is the most feasible one to load the Metrics Vault from SSIS.

One final note on the **LoadDate** in this specific Metrics Vault for SSIS: the recommendation in Chapter 4, Data Vault 2.0 Modeling, is to set the date to a single value for the whole batch in order to identify which records belong together. However, in the case of SSIS logging, each event is reported in a single transaction to the stored procedure. Therefore, each event receives its own load date, following a real-time loading pattern, which is not covered by this book.

In addition to the metrics directly obtained from SSIS using the preceding procedure, more metrics can be retrieved from Integration Services Catalogs, introduced in Microsoft SQL Server 2014. SQL Server maintains this catalog when SSIS packages are deployed to the server, a practice often used in production environments. The following internal views are available and could be integrated into the Metrics Vault [25]:

1. **catalog.catalog_properties:** provides the properties of the Integration Services catalog.
2. **catalog.effective_object_permissions:** provides the effective permissions on all objects in the Integration Services catalog for the current user.
3. **catalog.environment_variables:** provides detailed information about the variables in the environments in the Integration Services catalog.
4. **catalog.environments:** provides details for the environments in the Integration Services catalog.
5. **catalog.execution_parameter_values:** provides the environment variable values in the Integration Services catalog.
6. **catalog.executions:** provides detailed information about package executions in the Integration Services catalog.
7. **catalog.explicit_object_permissions:** provides the explicit permissions assigned to the current user.
8. **catalog.extended_operation_info:** provides extended information for all operations.
9. **catalog.folders:** provides information about the folders in the Integration Services catalog.
10. **catalog.object_parameters:** provides a list of all package and project parameters.
11. **catalog.object_versions:** provides a list of all object versions in the Integration Services catalog. As of Microsoft SQL Server 2014, only project versions are provided.
12. **catalog.operation_messages:** provides logged messages.
13. **catalog.operations:** provides a list of all operations in the Integration Services catalog.
14. **catalog.packages:** provides detailed information about all packages in the Integration Services catalog.
15. **catalog.environment_references:** provides a list of all environment references in all projects.
16. **catalog.projects:** provides a list of all projects in the Integration Services catalog with detailed information.
17. **catalog.validations:** provides a detailed list of project and package validations.

Using the raw data stored in these catalog views, it is possible to create a comprehensive Metrics Vault that integrates metrics from different sources.

However, both sets of metrics are based on SSIS and don't include any information about the hardware and network infrastructure. Such data can be obtained from Windows Management Instrumentation (WMI), provided by Microsoft Windows Server. WMI provides information about the server environment, including the hardware and network infrastructure. Table 10.17 lists the WMI providers currently available.

Table 10.17 Available WMI/MI/OMI Providers [26]

Active Directory	Mobile Device Management Settings	Software Inventory Logging
Application Proxy	MSFT_PCSVDevice	Software Licensing for Windows Vista
BitLocker Drive Encryption (BDE)	MsNetImPlatform	Software License
BITS	NetAdapterCim	Storage Volume
BizTalk	NetDaCim	System Registry
Boot Configuration Data	NetNcCim	System Restore
CIMWin32, Win32, Power Management Events	NetPeerDist	Trusted Platform Module
CIMWin32a	NetQosCim	Trustmon
DcbQosCim	NetSwitchTeam	User Access Logging
Distributed File System (DFS)	NetTCPIP	UserProfileProvider
Distributed File System Replication (DFSR)	NetTtCim	User State Management
Dfsncimprov	NetWNV	View
DhcpServerPSProvider	Network Access Protection	VPNClientPSProvider
Disk Quota	Network Load Balancing (NLB)	WDM
DNS	NFS	WFasCim
Dnsclientcim Provider Classes	Performance Counter	WhqlProvider
DnsClientPSProvider	Performance Monitoring	Win32ClockProvider
DnsServerPSProvider	Ping	Windows Data Access Components (WDAC)
Event Log	Policy	Windows Defender
Formatted Performance Data	Power Meter	Windows Installer
Failover Cluster	Power Policy	Windows Product Activation
Group Policy API	RAMgmtPSProvider	Windows Storage Management
Hyper-V	RAServerPSProvider	Windows System Assessment Tool
Hyper-V (V2)	ReliabilityMetricsProvider	WMI Core
Internet Information Services (IIS)	Remote Desktop Services	Msft_ProviderSubSystem
Internet Protocol Address Management (IPAM) Server	Reporting Services	Win32_Perf
IP Route Provider	Resultant Set of Policy (RSoP)	Win32_PerfFormattedData
Intelligent Platform Management Interface (IPMI)	Security	Win32_PerfRawData
iSCSI Target Server	ServerManager.DeploymentProvider	WMIPerfClass
Job Object	Session	WmiPerfInst
Kernel Trace	Shadow Copy	Work Folders
Live Communications Server 2003	SNMP	
Mobile Device Management Application	SMB Management	

The table also shows the Management Instrumentation (MI) and Open Management Infrastructure (OMI) providers that are available, because they use the same object format. The so-called Management Object Format (MOF) allows querying the data from classes in each provider. Access to the data in these management classes is performed using the WMI Query Language [27]. It is a subset of the ANSI SQL language known from database systems and supported in SSIS by the **WMI Data Reader Task** in the control flow.

10.4 IMPLEMENTING THE METRICS MART

After capturing the raw performance data in the Metrics Vault, the next step is to prepare the data for analysis. To support analysis of performance metrics, a data model is required that provides useful information to end-users. This data model is implemented in the Metrics Mart, which is located downstream towards the user (Figure 10.13).

Figure 10.13 highlights the EDW and the information delivery layer of the data warehouse. The Metrics Vault is implemented as an integral part of the Data Vault model and used exclusively to build the Metrics Mart. By doing so, this approach follows the standard Data Vault 2.0 architecture as outlined in Chapter 2, Scalable Data Warehouse Architectures. The Metrics Mart is an information mart that serves a special purpose and is sourced primarily from the Metrics Vault (instead of the Raw Data Vault and Business Vault). The implementation can use ETL for materializing the Metrics Mart or use virtualization, for example through SQL views. The latter is preferred as the speed to deploy new functionality drastically improves. Both techniques for loading information marts are discussed

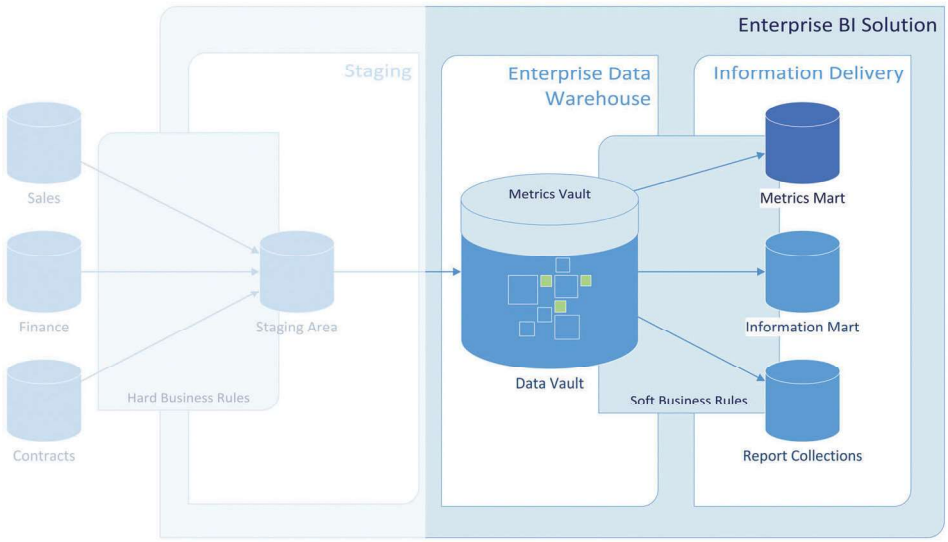

FIGURE 10.13

The Metrics Mart in the Data Vault 2.0 architecture.

in more detail in Chapter 14, Loading the Dimensional Information Mart. Because the Metrics Mart is a special variant of an information mart, the concepts for loading information marts apply for loading the Metrics Mart.

Note that the Metrics Mart can also be sourced from the Raw Data Vault or the Business Vault. In some cases, organizations decide to include business objects into performance measures, for example to measure technical performance metrics related to customers or products. These metrics can help to optimize the data warehouse even further.

The Metrics Mart uses a data model that fits the analysis needs of the end-users the most. This could be a dimensional model, especially if an OLAP cube should be built on top of the Metrics Mart or any other data model. But other data modeling techniques could be used as well, for example if a performance monitoring software is used that expects the data in a different format. It is also possible to feed such a monitoring software package directly from the Metrics Vault or the Metrics Mart, whatever fits the purpose of the task best.

In some cases, it might be helpful to provide the original structure of the SSIS table **sysssislog** in the Metrics Mart. For example, if a third-party tool expects the structure for further analysis. It is easy to provide the original structure by creating a SQL view in the Metrics Mart, sourcing the data directly from the tables in the Metrics Vault. The following T-SQL statement creates such a view:

```
CREATE VIEW sysssislog AS
SELECT
       tle.id,
       het.[event],
       hc.computer,
       ho.operator,
       ss.[source],
       hs.sourceid,
       tle.executionid,
       tle.starttime,
       tle.endtime,
       tle.datacode,
       tse.databytes,
       tse.[message]
FROM
       DataVault.log.TLinkEvent tle
INNER JOIN
       DataVault.log.HubEventType het ON het.EventTypeHashKey = tle.EventTypeHashKey
INNER JOIN
       DataVault.log.HubComputer hc ON hc.ComputerHashKey = tle.ComputerHashKey
INNER JOIN
       DataVault.log.HubOperator ho ON ho.OperatorHashKey = tle.OperatorHashKey
INNER JOIN
       DataVault.log.HubSource hs ON hs.SourceHashKey = tle.SourceHashKey
LEFT JOIN
       DataVault.log.SatSource ss ON ss.SourceHashKey = tle.SourceHashKey
       AND tle.LoadDate >= ss.LoadDate
       AND tle.LoadDate <= ISNULL(ss.LoadEndDate, '9999-12-31 23:59:59.999')
LEFT JOIN
       DataVault.log.TSatEvent tse ON tse.EventHashKey = tle.EventHashKey
       AND tse.LoadDate = tle.LoadDate
```

The statement uses the transaction link TLinkEvent as the primary source because it has the same grain as the original **sysssislog** table. It then joins the business keys from the hubs and other descriptive information to provide the original structure of the table.

In the same manner, another structure required by the end user could be provided. The advantage over the separation of the raw data (in the Metrics Vault) and the information (in the Metrics Mart) is the same as for the Raw Data Vault and information marts: it is possible to integrate other data sources more easily while building required structures in a virtual manner. Thus, it is easy to extend this simple Metrics Vault and Metrics Mart combination by adding additional sources for performance data over the course of the project.

10.5 IMPLEMENTING THE ERROR MART

The Error Mart is another information mart. However, similar to the Meta Mart, the Error Mart is not sourced from the Raw Data Vault or any other source. Instead, it is the primary location to store error information. The error information can come from a variety of sources, but most of it comes from the ETL engine. The Error Mart captures the following types of records:

1. **Records rejected by the staging area:** while the goal of the staging area is to temporarily load all data from the source system, it is not always feasible. In some cases, records have to be rejected because they don't fit into the relational structure of the staging area, for example if transmission errors have occurred.
2. **Records rejected by the Raw Data Vault:** again, the goal of the loading processes for the Raw Data Vault is to capture all raw data, *the good, the bad, and the ugly*. But similar to the staging area, not all data can be captured, especially if it doesn't fit into the relational structure of the Raw Data Vault.
3. **Records not processed by the Business Vault:** the Business Vault processes the raw data from the Raw Data Vault and creates intermediate business rule results. In some cases, soft rules define that the raw data has to comply with given rules, defined by the business. The Error Mart captures which records did not comply with these rules.
4. **Records not processed by the information marts:** because the Business Vault implements business rules only partially, the information marts can reject additional raw records. For that reason, the data rejected by the soft rules implemented in the information mart have to be redirected into the Error Mart as well.

Because soft business rules change over time, the data that is captured by the Error Mart changes over time as well. Interestingly, not all rejected records are bad records. The Error Mart also includes unexpected data that was not expected by the soft rule. In Data Vault, such data is called ugly data. The Error Mart helps to identify ugly data and improve the business rule implementation to capture more data that exists in the operational system and can be processed by some business logic. However, the ultimate goal is to identify the problem as to why the ugly record was not loaded into the Raw Data Vault, fix the issue (e.g., expect it in the loading ETL routines) and load it into the Raw Data Vault.

While the Error Mart can be implemented in the data model preferred by the end-user, many Error Marts are built using a dimension model. Such a model contains the erroneous data in fact tables and includes accompanying dimensions. The next section describes how to set up SSIS to capture erroneous data.

10.5.1 CAPTURING ERRONEOUS DATA IN SQL SERVER INTEGRATION SERVICES

Microsoft SSIS provides the capability to redirect records that have caused a failure in the data flow to an alternative component in the flow. This can include another OLE DB or SQL Server destination. Most data flow components offer an error output that can be used for this purpose.

The error output contains all the columns in the default data flow plus some columns that describe the error. Therefore, all error outputs are different from each other. When building the Error Mart, there are two options:

1. **Create common fact table for all error outputs:** this fact table contains only columns common to all error outputs. It requires the removal of most of the descriptive columns, the raw data, from the fact table.
2. **Create separate fact tables for each error output:** for each error output, a separate fact table is created. It allows the storing of all descriptive data in the corresponding fact table.

The advantage of the first option is that the data model of the Error Mart is fairly simple. However, it doesn't allow us to store the raw data that we might need for error analysis. The second option allows us that but the price is to have a lot of fact tables in the Error Mart which must be analyzed individually. For the sake of this chapter, we will describe a solution that implements option 2 because the goal of the Error Mart is to provide detailed information of the errors in the data warehouse. The production of mere statistics (which is the primary result of option 1) should be left to the Metrics Mart, which can keep error statistics as well.

Consider the staging process for master data in Chapter 9, Master Data Management (Figure 10.14).

Using this process, data on regions was sourced from a flat file and loaded into the Master Data Services (MDS). By doing so, data can be loaded into MDS automatically, without typing it in manually

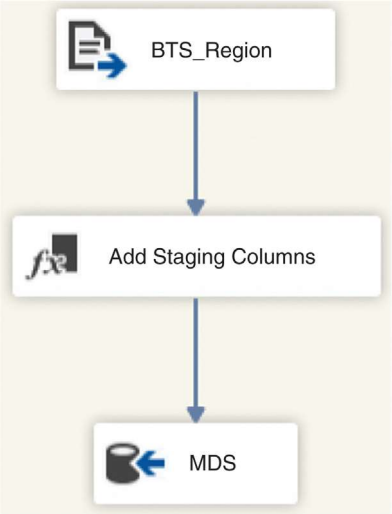

FIGURE 10.14

Data flow for staging master data.

using the Web interface or Microsoft Excel. After adding some columns required by the staging process in the second step, the data is loaded into the appropriate staging table for the **Region** entity in the **BTS** model.

In some cases, the MDS destination might reject records that don't fit into the relational structure. These are technical problems that result from errors in the SSIS implementation, unexpected data or other data-related issues (such as transmission errors in the flat file). On the other hand, MDS business rules which might be defined on the MDS entity could be violated as well. For example, there could be a business rule that requires that the **abbreviation** of the region must be different from the **code** value of the region. If the staged data from the source violates this rule, the record would be marked by MDS after applying the business rules on the entity. MDS would not reject the data. But it is possible to evaluate the result in the subscription view of the target entity and start appropriate measures to deal with invalid data in MDS.

Assuming that the staging table could reject some of the incoming data if there is a hard, technical error, in order to log these erroneous records into the Error Mart, the data flow is extended by redirecting the error output into another destination (Figure 10.15).

All records that have been rejected by the **MDS** destination output in the data flow of Figure 10.15 are redirected into the **Error Mart Destination**. Before doing so, some audit information, gathered from the SSIS engine, is added to the data flow (Figure 10.16).

The **Audit Type** column provides predefined audit information attributes that can be added to the data flow of the error output. On the left side of the table, the corresponding column names are set. In addition to these columns, more audit information can be added using a **Derived Column** component added to the data flow of the error output.

Note that the column names use a readable structure with spaces. In actual projects, the column names should adhere to the standard naming conventions of the data warehouse.

The audit columns should be used as conformed dimensions across the Error Mart. However, to keep things simple, the Error Mart in this example doesn't contain any conformed dimensions, not even

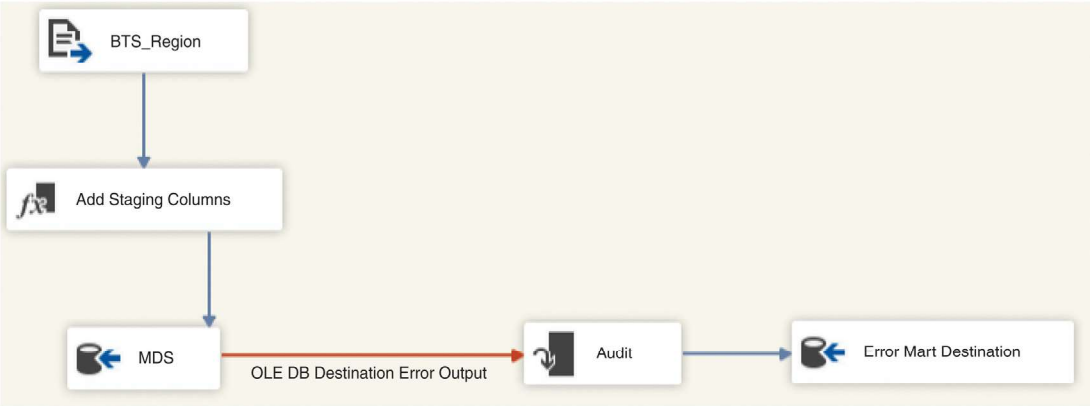

FIGURE 10.15

Data flow for staging master data with Error Mart destination.

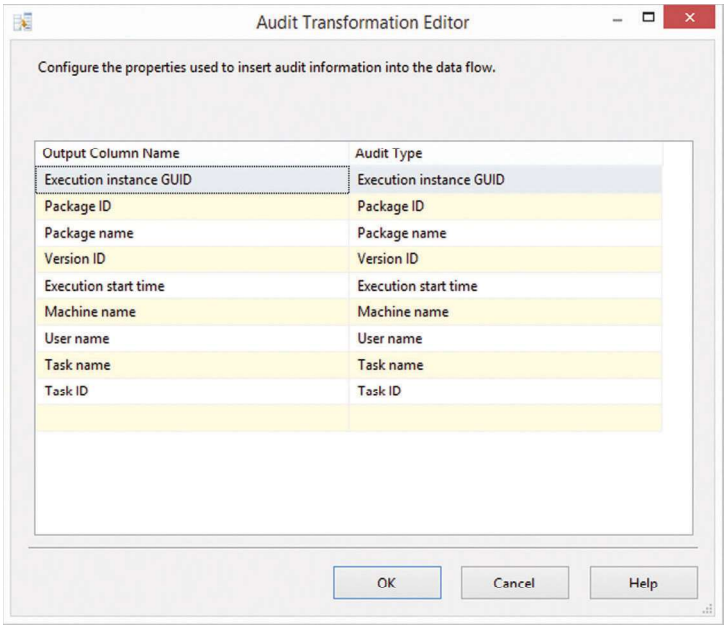

FIGURE 10.16

Audit transformation editor.

dimension tables. All dimensions are directly stored in the fact table, which is not a good practice but works for the purposes of this chapter. A better practice would be to create the following dimension tables:

1. **DimPackage:** containing **Package ID**, **Package name** and **Version ID**. This should be a slowly changing dimension (SCD) type 2 dimension.
2. **DimMachine:** containing **Machine name** and other descriptive information from other sources.
3. **DimUser:** containing **User name** and descriptive information from Active Directory and other sources.
4. **DimTask:** containing **Task name** and **Task ID**. This should be a SCD type 2, because the task name might change (but the task ID doesn't for the same task in SSIS).
5. **DimError:** containing **ErrorCode** and a description of the error. The error description can be obtained by a Script component (refer to [28] for more details).
6. **DimColumn:** providing the **ErrorColumn** and the name of the column. To retrieve the name of the column, a more complicated process is required which is described online (refer to [29]).

The remaining audit columns should be directly added to the fact table in a realistic environment. While using dimension tables is the preferred solution, it would require a more complicated

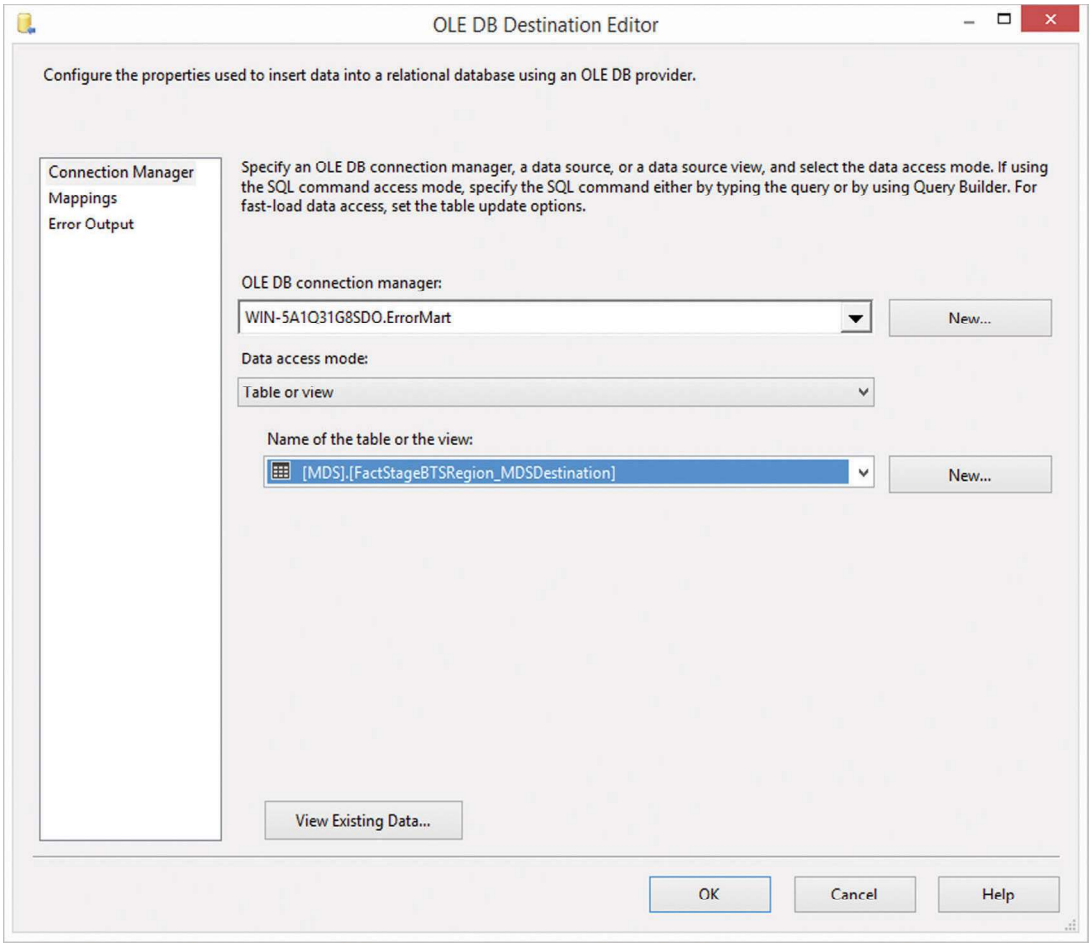

FIGURE 10.17

OLE DB destination editor for the Error Mart destination.

loading process to populate the data in the dimensions as well. Therefore, the Error Mart Destination setup becomes fairly simple. First, the connection to the Error Mart has to be set up (Figure 10.17).

The connection directly accesses the **ErrorMart** database and uses a **table or view** data access mode to the target fact table. The New button in Figure 10.17 allows one to quickly create the target table, based on the data flow of the error output and the added audit columns (Figure 10.18).

Because it is possible to modify the statement directly in the dialog, it is a good place to apply naming conventions to the fact table name. The last step is to map the columns in the data flow to the destination table, which is shown in Figure 10.19.

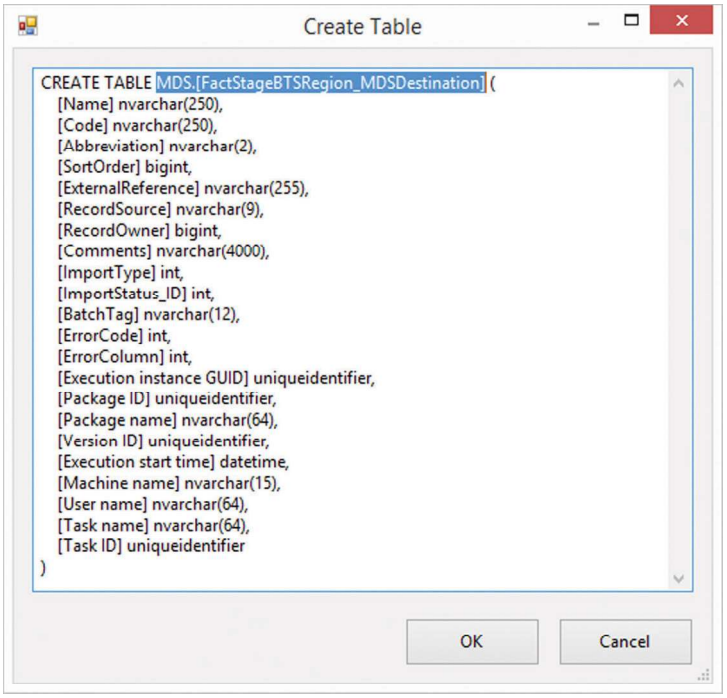

FIGURE 10.18

Create table editor for creating the Error Mart destination.

Because of the simple structure, only a direct mapping between the input columns from the data flow and the destination columns in the target fact table is required.

To implement this solution with dimension tables (as previously described), multiple options are available:

1. **Extend the SSIS data flow:** the SSIS data flow could be extended to populate the dimension tables by adding additional lookup and database destinations to the error output. Use the **Slowly Changing Dimension** component as a starting point for extending the SSIS data flow. However, this approach requires a large number of components to be added to the data flow, which makes it relatively complex.
2. **Create an INSTEAD OF trigger on a view:** replace the table by a view with an accompanying INSTEAD OF trigger that distributes the data in the fact and dimension tables. This approach is transparent to SSIS but requires programming on the database back-end.
3. **Write into a stored procedure:** instead of using a trigger, write your error output to a stored procedure. Use the **SQL command** data access mode in the **OLE DB Destination** component. By doing so, it is not completely transparent, but easier to understand and more visible than an INSTEAD OF trigger.

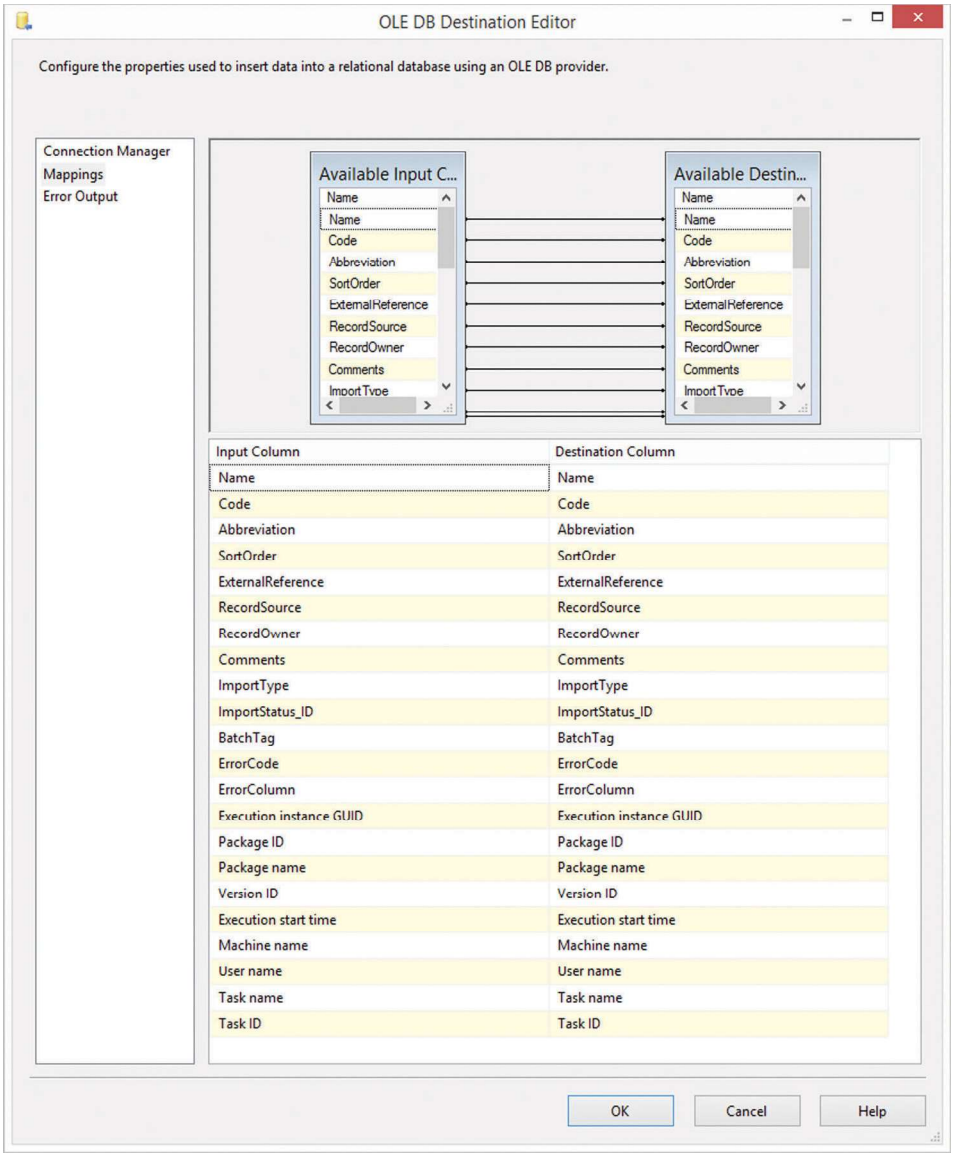

FIGURE 10.19

Column mapping for the Error Mart destination.

Which option you choose depends on a number of factors regarding maintainability and programming requirements. Stick to the solution shown in this chapter if storage is negligible compared to the overhead to maintain the dimension tables. Regardless whether dimension tables are created or if the dimensions are integrated in the fact table, analysis can be performed on the fact tables, either directly or using OLAP cubes in SSAS.

REFERENCES

[1] W. H. Inmon, et al. "Corporate Information Factory," second edition, p. 169, 169f.
[2] Ralph Kimball and Jose Caserta: "The Data Warehouse ETL Toolkit," pp. 124ff, 352, 357, 359, 360ff, 362, 364ff, 367ff, 376, 379.
[3] http://www.merriam-webster.com/dictionary/volumetric
[4] Claudia Imhoff, et al. "Mastering Data Warehouse Design," p. 15.
[5] https://sqlmetadata.codeplex.com/
[6] https://sqlmetadata.codeplex.com/documentation
[7] David Marco, Michael Jennings: "Universal Meta Data Models", pp. 124ff, 134ff.
[8] Barbara von Halle: "Business Rules Applied," pp. 34, 436, 446.
[9] http://agilemodeling.com/artifacts/businessRule.htm
[10] David Marco, Michael Jennings: "Universal Meta Data Models", pp. 125, 126.
[11] Jennifer Stapleton: "DSDM: Business Focused Development," p. 197ff.
[12] Kimmo Palletvuori: "Security of Data Warehousing Server," TKK T-110.5290 Seminar on Network Security.
[13] http://web.stanford.edu/group/security/securecomputing/dataclass_chart.html
[14] Bharat Bhargava: "Security in Data Warehousing".
[15] Brian Knight, et al. "Professional Microsoft SQL Server 2014 Integration Services," p. 622, 622f.
[16] Quest Software: "Spotlight on Oracle 7.6: Getting Started Guide," p. 33ff.
[17] http://labs.consol.de/lang/en/nagios/check_mssql_health
[18] http://www.cse.wustl.edu/~jain/cse567-06/ftp/net_traffic_monitors2
[19] http://logicalread.solarwinds.com/sql-server-buffer-hit-cache-ratio/#.VJ6dOsANA
[20] http://msdn.microsoft.com/en-us/library/ms141744.aspx
[21] http://msdn.microsoft.com/en-us/library/ms140246.aspx
[22] http://msdn.microsoft.com/en-us/library/ms136010.aspx
[23] http://msdn.microsoft.com/en-us/library/ms141122(v(sql.120)
[24] http://msdn.microsoft.com/en-us/library/ms186984.aspx
[25] http://msdn.microsoft.com/en-us/library/ff878135.aspx
[26] http://msdn.microsoft.com/en-us/library/bg126473(v(vs.85).aspx
[27] http://msdn.microsoft.com/en-us/library/aa392902(v(vs.85).aspx
[28] http://msdn.microsoft.com/en-us/library/ms345163(v(sql.120).aspx
[29] http://dougbert.com/blog/post/Adding-the-error-column-name-to-an-error-output.aspx

CHAPTER

DATA EXTRACTION 11

Once the physical environment has been set up (refer to Chapter 8, Physical Data Warehouse Design), the development of the data warehouse begins. This includes master data (as described in Chapter 9, Master Data Management) and the management of metadata (see Chapter 10, Metadata Management). However, the majority of data (regarding the volume and the variety) is periodically loaded from source systems. In addition, more and more data is sourced in real time (not covered by this book) or written directly into the data warehouse (similar to the **sp_ssis_addlogentry** stored procedure in Chapter 10, section 10.3.1).

This chapter covers the extraction from source systems both on premise (such as flat files which provide data extracted from operational systems) and in the Cloud (from Google Drive). Extracting master data from MDS is also covered, both materialized (using SSIS) and virtualized.

Throughout this book, we use example data from the Bureau of Transportation Statistics [1] (BTS) at the U.S. Department of Transportation [2] (DoT). The dataset contains scheduled flights that have been operated by U.S. air carriers. The flights include their actual departure and arrival times. Only those U.S. carriers are covered that account for at least 1% of domestic scheduled passenger revenues [3]. Figure 11.1 shows an overview of the BTS model.

Note that the raw data has been converted into a more usable format for the purpose of this book. For example, the monthly database dumps have been split into daily dumps for ease of use. Also, due to technical issues, some lines have been dropped in some years (for example in 2001 and 2002, among others). You should not conduct research using these files as not all data has been converted – the data sets used in this book are incomplete. If you need files that include all airline data, you should obtain them directly from the BTS Web site [3].

It should also be noted that there are more business keys in the source data that are ignored when building the examples in the book. It is also important to recognize that some of the selected and implemented business keys are far from being optimal; for example, the airport code and the carrier code are not stable because they are reused for new business entities in some cases.

11.1 PURPOSE OF STAGING AREA

When extracting data from source systems, the workload on the operational system increases to some extent. The actual load that is added to the source infrastructure depends on a number of factors, for example how much data is prepared before the extraction or if raw data is exported directly. The required workload is also influenced by the use of bulk exports (i.e., directly exporting the results of a SQL query) or the use of some application programming interface (API). In the worst case (from a

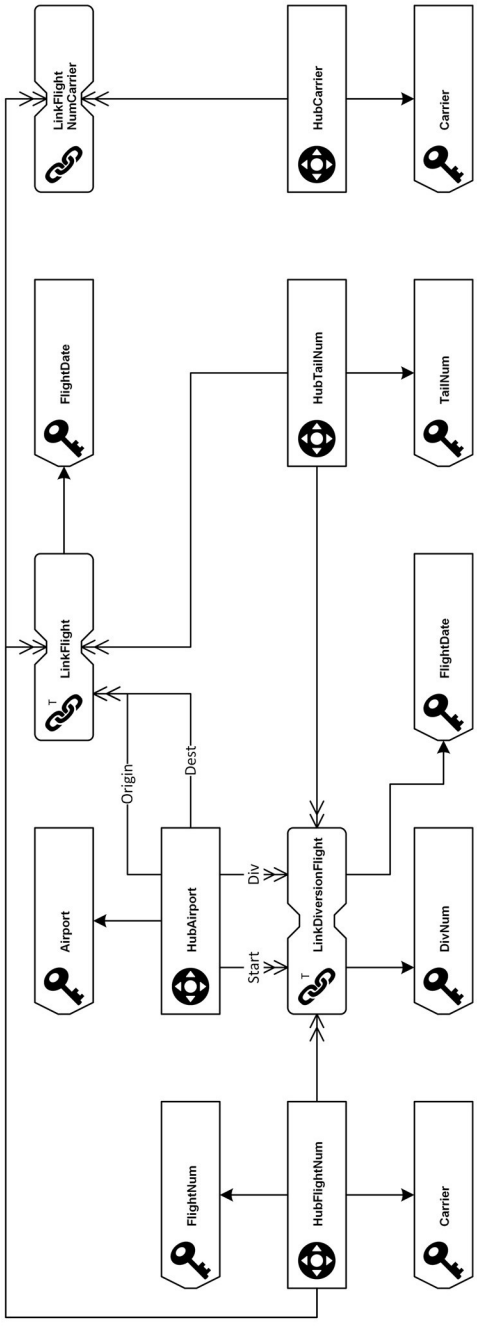

FIGURE 11.1

BTS overview model (logical design).

performance standpoint), all data from the database has to go through an object-relational mapping (ORM) before it is exported into a flat file. In other, similar cases, the relational database is not directly accessible but only some representational state transfer (REST) or Web Service API. Loading the data from such APIs will be a pain on its own, but loading processes for data warehouse systems often involves additional operations, for example lookups to find out the business key for a referenced object or the name of a passenger. If these lookups are directly performed against an operational system, the data warehouse might have an additional burden placed on it. Consider the daily load of 40,000 passenger flight information records for a data warehouse of a midsized airline [4]. Because the business needs detailed information about the aircraft used, a lookup into the aircraft management system with 100 operational aircraft is required. If no lookup caching is enabled, 40,000 lookup operations are required to retrieve the data for the 100 aircraft. Even with lookup caching, more than 100 lookup operations are required to serve all lookups in all data flows.

The primary purpose of the staging area is to reduce the workload on the operational systems by loading all required data into a separate database first. This ingestion of data should be performed as quickly as possible in any format provided. For this reason, the Data Vault 2.0 System of Business Intelligence includes the use of NoSQL platforms, such as Hadoop, as staging areas. Once the raw data has been loaded into the NoSQL platform, it is then sourced and structured once business value and business requirements are established. Due to space limitations, this book will focus on a relational staging area.

The advantage of this staging area is that the data is under technical control of the data warehouse team and can be indexed, sorted, joined, etc. All operations available in relational database systems are available for the staged data. To ensure best usage of the data in the staging area, the data types follow the intended data types from the source system (don't use only varchar data types in the staging area). But another purpose of the staging area is to make data extraction from the operational system easy. We will see in section 11.8 how we violate the recommendation to avoid using only varchars in the staging area in order to retrieve the source data with ease. The data is transformed into the actual data types when loading it into the Data Vault model (refer to Chapter 12, Loading the Data Vault for details).

It is also possible to look up data in a staging area even though, in data warehouse systems built with Data Vault 2.0, this is not often required. Business rules are implemented after the Raw Data Vault and perform their lookups into the Raw Data Vault only. The reason for this is that the staging area provides only a limited history of previous loads. The staging area is transient. It should keep only a limited number of loads in order to deal with erroneous ETL processes downstream (to load the Raw Data Vault). The goal is to prevent data loss if the data cannot be loaded into the data warehouse layer for technical or organizational reasons. For example, not all errors can be fixed within a business day. It is not the goal of the staging area to keep a history of loads. This goal is accomplished by the enterprise data warehouse layer. If the staging area keeps the history for all loads, it requires additional management (consider storage, backup, etc.). As a result, you end up with the management of two data warehouses.

Loading the staging area follows a standard pattern because all incoming data items, such as flat files or relational tables, are sourced independently. To achieve this independence, there is no referential integrity implemented through foreign keys. Each staging table is a direct copy of the incoming data with only some system-generated attributes added. Figure 11.2 shows the template for loading a staging table.

The first step is to **truncate the staging tables**, if only one load should be kept. If the staging area should follow best practices and keep a limited history, the truncation has to take place once the data has been loaded into the Raw Data Vault. For performance reason, it might be advisable to **disable indexes** on the staging area, if there are any implemented. The next step is to perform a **bulk read on**

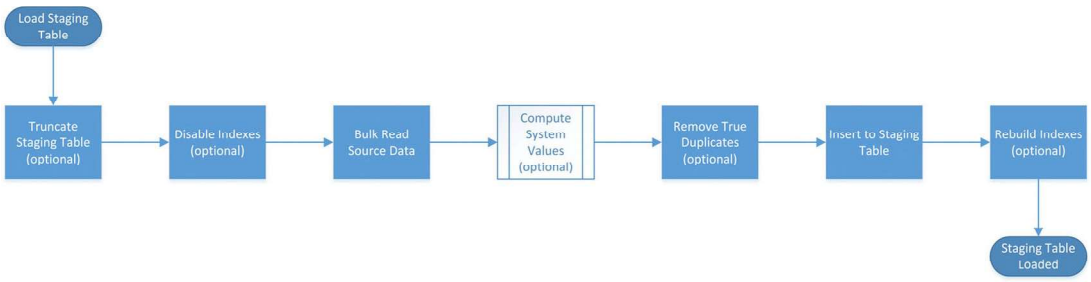

FIGURE 11.2

Template for staging area loading.

the source data from the source file or source database connection. As already mentioned, in order to achieve acceptable performance, the goal should be to read the data in chunks, not single records. If possible, complex APIs, such as the ones introduced by object-relational mapping (ORM), should be avoided. Instead, direct access to the relational tables or their raw data exported to flat files should be preferred. In Microsoft SQL Server, this can be achieved by executing the following T-SQL statement on the staging table [5]:

```
ALTER INDEX ALL ON StageArea.BTS.OnTimeOnTimePerformance
DISABLE;
```

Another purpose of the staging area is to **compute and add system-generated attributes** to the incoming data. By doing this, reuse of the attributes is possible. Figure 11.3 shows the subprocess in more detail. In some cases, true duplicates exist in the source data. If they are kept and loaded into the Raw Data Vault, they would only create additional workload, but no additional changes to hubs and links. Depending on the implementation of the satellite loading processes, they could cause additional problems during change detection. This is why **true duplicates should be removed** from the staging area. Finally, the data is **inserted into the staging table** and **indexes are rebuilt** again. The Microsoft SQL Statement is very similar to the previous one [5]:

```
ALTER INDEX ALL ON StageArea.BTS.OnTimeOnTimePerformance
REBUILD;
```

The following attributes are added in the **Compute System Values** subprocess, which is presented in Figure 11.3.

- **Load date:** the date and time when the batch arrived in the data warehouse (in the staging area). A batch includes all data that arrived as a set. In most cases, this includes all data in all flat files or relational tables from one source system.
- **Record source:** a readable string that identifies the source system and the module or source table where the data comes from. This is used for debugging purposes only and should help your team to trace errors.

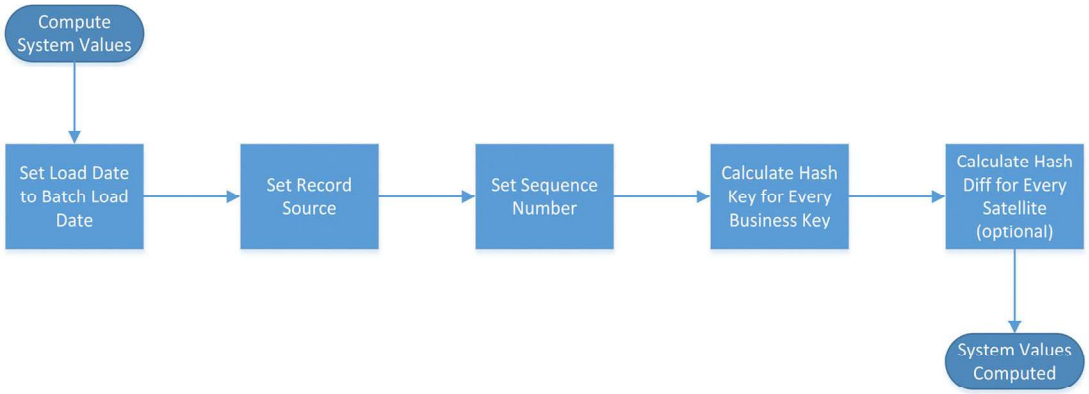

FIGURE 11.3

Compute system values subprocess.

- **Sequence number:** all records in a staging table are sequenced using a unique and ascending number. The purpose of this number is to recognize the order of the records in the source system and identify true duplicates, which are loaded to the staging area but not to downstream layers of the data warehouse. Note that the sequence number is not used for identification purposes of the record.
- **Hash keys:** every business key in the Raw Data Vault is hashed for identification purposes. This includes composite keys and key combinations used in Data Vault links. Chapter 4, Data Vault 2.0 Modeling, discusses how hash keys are used in hubs and links for identification purposes.
- **Hash diffs:** these keys are used to identify changes in descriptive attributes of satellite rows. This is also discussed in Chapter 4.

Because hash value computation consumes CPU power, there is an advantage of reusing the computed value. While the hash function that is used in this process is standardized (examples include MD5 and SHA-1), the input of the hash function has to follow organization-wide, defined standards. These standards have to be set by a governing body in the organization and should be based on the best practices as outlined in the next section.

11.2 HASHING IN THE DATA WAREHOUSE

Traditional data warehouse models often use sequence numbers to identify the records in other tables in the data warehouse and reference them in dependent tables. These numbers are generated in the data warehouse instead of being sourced from operational systems in order to use an independent sequence number that is controlled by the data warehouse.

There are multiple drawbacks with sequence numbers [6]:

- **Dependencies in the loading processes:** in order to load a destination, every dependency has to be loaded first. For example, loading a table that stores the customer addresses requires the loading of the customer table first, in order to look up the sequence number which identifies a particular customer.

- **Waiting on caches for parent lookups:** assuming sequence numbers instead of hashes would be applied. In this situation, Link and Satellite loads must "wait" for parent table lookup caches (particularly in ETL engines) to load with the cross-maps of business keys to sequence numbers. Hubs and links often contain large sets of rows that need to be cached before the ETL processing can begin. This causes a bottleneck in the loading processes. This can be alleviated or removed by switching the model to Data Vault 2.0 Hash Keys – eliminating the need for lookup caching altogether.
- **Dependencies on serial algorithms:** sequence numbers are often used as surrogate keys but they are what they are: sequences that are serial numbers. In order to generate such a sequence number, a sequence generator is often used, which presents a bottleneck because it needs to be synchronized in order to prevent two sequence numbers with the same value. In Big Data environments, the required synchronization can become a problem because data is coming at high speed and/or volume.
- **Complete restore limitations:** when restoring a parent table (e.g. the customer table from the previous example), the sequence numbers need to be the same as before the restore. Otherwise, the customer references in dependent tables become invalid (in the best case) or wrong (in the worst case).
- **Required synchronization of multiple environments:** if sequence numbers have to be unique across nodes or heterogeneous environments, synchronization of the sequence number generation is required. This synchronization can become a bottleneck in Big Data environments.
- **Data distribution and partitioning in MPP environments:** in MPP environments, the sequence number should not be used for data distribution or partitioning because queries can cause hot spots when getting data out of the MPP environment [7].
- **Scalability issues:** sequence numbers are easy to use but are limited when it comes to scalability. When using sequence numbers in large data warehouses with multiple terabytes or even petabytes of data, the sequence generation can become a bottleneck when loading large amounts of data. Sequences have an upper limit (usually), and in large data sets, or with repeated "restarts" due to error processing, the sequence generators can "run-out" of sequence numbers, and must then be cycled back to one. This is called roll-over, and isn't a preferred situation – nor is it advisable to have this situation in a Big Data system to begin with. The other side of this situation is performance. With Big Data, many times the loading processes need to be run in parallel. Sequence generators, when "partitioned" so they can run in parallel, assign groups of sequences – and can hit the upper limits of the sequence number faster. Why? Each group of sequences (one per partitioned process) requires a block of sequences, and often leaves holes in the sequencing as it loads, thus eating up sequences much quicker than expected.
- **Difference of NoSQL engines:** in many cases, NoSQL engines use hash keys instead of sequence numbers because of the limitations regarding MPP data distribution and scalability, described in the previous bullet points.

Due to these drawbacks and limitations, hash keys are used as primary keys in the Data Vault 2.0 model, thus replacing sequence numbers as surrogate keys. Hash functions ensure that (exactly) the same input (on a bit level) produces the same hash value. The following sections discuss how to use this characteristic of hash functions for generating nonsequential surrogate keys that can be used in data warehousing.

Chapter 4 has outlined some benefits of using hash keys in the Data Vault 2.0 model: because hash keys are calculated independently in loading processes, there are no lookups into other Data Vault 2.0 entities required in order to get the surrogate key of a business key. In general, a lookup into another table requires I/O performance in order to retrieve the sequence number for a given business key. On the other hand, computing a hash key only requires CPU performance, which is often favored over I/O performance because of better parallelization and better resource consumption in general.

The alternative to hash keys is to use business keys as identifying elements in the data warehouse. However, using hash keys provides the advantage that the primary keys (and the referencing columns) can always use a fixed length data type, which has performance advantages in Microsoft SQL Server. This is due to the fixed-length column value that can be stored directly in the data page, requiring no lookups in additional database pages (text/image page) when performing table joins. Similar advantages are seen in other relational database management systems. While business keys in hubs might be shorter than hash keys, hash keys tend to be shorter in link tables (where multiple business keys are combined), a factor often overlooked when discussing the storage requirements of hash keys versus business keys.

But it's not only the identification of business keys that can take advantage of introducing hash calculations in the data warehouse. Another advantage is the use of hash diff values, which can help to speed up column compares on descriptive attributes when loading satellites. This is of special interest for loading large amounts of raw data into satellites and is described in more detail in section 11.2.5.

While the benefits outweigh the drawbacks, the goal of the data warehouse is to reduce the number of hash key calculations in order to reduce CPU computations. Therefore, most hash values (hash keys and hash diffs) are calculated when loading the data from source systems into the stage area. This is not possible in every case, but is a desired strategy that should be achieved in 80% or more of the cases. Figure 11.4 shows potential locations in the loading process where hash values can be calculated if it is not possible to calculate and store the hash keys and hash diff values in the staging area.

Calculating the hash values upstream to the data warehouse increases the reusability of the values in downstream processes towards the business user and prevents recomputation of the same hash key or hash diff value. On the other hand, the reusability of hash diff values is limited because they are only used in one satellite.

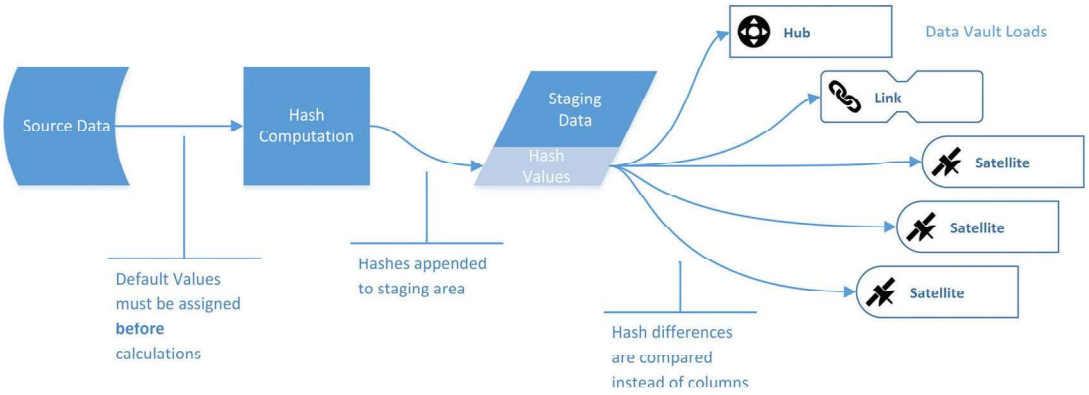

FIGURE 11.4

Stage load hash computation.

11.2.1 HASH FUNCTIONS REVISITED

There are multiple options available when applying hash functions in data warehousing. When selecting the hash function it is important to understand that hashing in the data warehouse doesn't have any intention of encrypting or securing the business key. The only goal is to replace sequential surrogate keys by a hash key alternative [8]. Hash functions have a desired set of characteristics, including [8]:

- **Deterministic:** for a given block of input data, the hash function will always generate the same hash value whenever called.
- **Irreversible:** a hash function is an irreversible, one-way function. It is not possible to derive the source data from the hash value.
- **Avalanche effect:** if the input data is changed only slightly, the hash value should change drastically. Also known as a cascading effect.
- **Collision-free:** any two different blocks of input data should always produce different hash values. The unlikely event of two input blocks generating the same hash value is called a collision. These hash collisions is discussed in more detail in section 11.2.3.

The most common and recommended options for the purpose of hashing business keys include MD5 and SHA-1:

- **MD5 message-digest algorithm (MD5):** The MD5 algorithm was introduced in 1992 as a replacement of MD4. It takes a message of arbitrary length as an input and produces a 128-bit (16-byte) hash value, called a message digest or "signature" of the input. In cryptography, it was commonly used to digitally sign the input, such as an email message or file download to prevent later modifications [9].
- **Secure hash algorithm (SHA):** In 1995, SHA-1 was published as an industry standard for secure hash algorithms and should replace MD5 due to security concerns. SHA-1 is a U.S. Federal Information Processing Standard [10].

Table 11.1 shows some key statistics about the generated hashes of both algorithms and, with SHA-256, an implementation of the secure hash algorithm with a larger message digest size:

The different output lengths are due to the fact that a character in hexadecimal representation can represent only 4 bits. For that reason, the space is doubled when using hexadecimal characters to represent the hash value. Section 11.2.2 discusses the advantages of the hexadecimal hash representation. Section 11.2.3 discusses the advantages of all three hash functions in more detail.

Note that both algorithms are used in cryptography, but they are not encryption algorithms. The hash result is typically encrypted with a private key, as used in public-key cryptosystems, for example

Table 11.1 Key Statistics for Selected Hash Algorithms

	MD5	SHA-1	SHA-256
Max. Input Length	Unlimited	2^{64}-1 bits	2^{64}-1 bits
Output Length (binary)	128 bits (16 bytes)	160 bits (20 bytes)	256 bits (32 bytes)
Output Length (hex)	32 characters/bytes	40 characters/bytes	64 characters/bytes

RSA [11]. By doing so, a signature is created that is associated with the public key of the sender, which can be validated.

Unencrypted MD5 and SHA-1 hashes are often used to validate whether a download has not been compromised or accidentally modified due to transmission errors after downloading the file from the Web site of origin. For that reason, hashes are often seen next to downloads on Web sites, as shown in Figure 11.5.

In this case, there are multiple hash values provided using hexadecimal representation, including MD5, SHA-1 and SHA-256. By using hexadecimal characters, it is possible to include the hash value as a "readable" text directly into the Web site. Otherwise, the hash keys would have to be provided as additional binary file downloads. It is possible to validate the integrity of the downloaded file with a tool such as File Checksum Integrity Verifier (FCIV) from Microsoft [12]. The tool takes the downloaded file as input and reports the MD5 hash for the downloaded file:

```
FCIV -md5 LibreOffice_4.3.5_MacOS_x86-64.dmg
```

If there is no difference between the bits of the file on the server and the bits of the downloaded file on the client, the output of the File Checksum Integrity Verifier should be the same as stated on the Web site in Figure 11.5.

11.2.2 APPLYING HASH FUNCTIONS TO DATA

The approach described in the previous section uses an important characteristic of hash functions: the same input (on a bit level) produces the same, fixed-length hash value. In addition, the hash function produces a "random-like" value for each input. If only one bit changes in the input, the resulting hash value is not even close to the hash value without the changed bit. As described in the previous section, this characteristic is called the avalanche effect [13]. Consider the following examples:

Table 11.2 shows three example inputs and their respective MD5 and SHA-1 hash keys (note that MD5 outputs and SHA-1 outputs cannot be compared due to the different algorithms). Notice the inputs: the first two values are "Data Vault 2.0" and "DataVault 2.0": a space between the first two words was removed. While this change seems to be minimal, it is actually not. The character length is

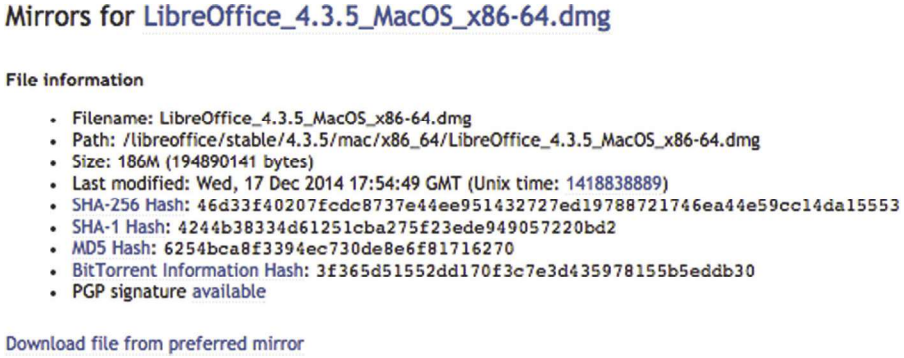

FIGURE 11.5

LibreOffice download with hash values for verification purposes.

different (from 14 characters down to 13) and therefore, the bit representation after the fourth character is completely different, as all following bits shift by eight characters to the left (Table 11.3).

Given their positions, all bold bits have changed, because they have been removed or shifted to the left. Therefore, we would expect this drastic change in the input to be reflected in their hash keys (which is the case in Table 11.2). And the same is true for an actual small change on the bit level (Table 11.4).

Here, only the first character case changes from an upper-case D to a lower-case d. Bitwise, the difference is just one bit that changes in the input, as indicated bold in Table 11.4: the bit on position 3 has changed from 0 to 1. However, even in this case of minor change, the hash values between example 2 and 3 in Table 11.2 are completely different, yet not randomly generated. The hash looks random, but it follows an algorithm that has the goal of ensuring that small changes have a high impact on the output – a desired characteristic in cryptography.

This characteristic is also desired when distributing the data in MPP environments such as Microsoft Analytics Platform System (formerly Microsoft SQL Server Parallel Data Warehouse). The distribution relies on a distribution column (other vendors call this a primary index). The data in this distribution column should be evenly distributed because all rows in a table are distributed according to this data in the column. If the data is skewed, the records will not be evenly distributed across the nodes, which will cause hot spots when answering queries [14]. Because the calculated hash value is "random-like," it is a good candidate for a distribution column. Even similar data is distributed evenly on the nodes, thus ensuring that the maximum possible number of nodes answers queries.

Hash values can be calculated in a number of locations, for example directly in the database (using T-SQL functions), in SSIS, or in a third-party application. However, while the implementation of the hash algorithm should be the same (in the case of SHA-1, this can be ensured through a validation program by the NIST [15]), the resulting hash value depends heavily on the input to the function.

Table 11.2 Hash Examples

#	Input	Output (MD5)	Output (SHA-1)
1	Data Vault 2.0	CCD04E26434D844C002CF7B0914F61EB	47014AD9CAEA430A925F98FF21D7CD420F0E219A
2	DataVault 2.0	D234C1DA50518AC45A432D24AD756553	C11A059BBFD5DAA5A9E7ED594353CCDA13BA9151
3	dataVault 2.0	61CA9F6B4F747C5F09BAAB093FE21AF2	06495EEE633A19F30319457A39CEBD89B275A48B

Table 11.3 Binary Representation of Examples 1 and 2

Input (ASCII)	Binary Representation
Data Vault 2.0	01000100 01100001 01110100 01100001 **00100000 01010110 01100001 01110101 01101100 01110100 00100000 00110010 00101110 00110000**
DataVault 2.0	01000100 01100001 01110100 01100001 **01010110 01100001 01110101 01101100 01110100 00100000 00110010 00101110 00110000**

Table 11.4 Binary Representation of Examples 2 and 3

Input (ASCII)	Binary Representation
DataVault 2.0	01000100 01100001 01110100 01100001 01010110 01100001 01110101 01101100 01110100 00100000 00110010 00101110 00110000
dataVault 2.0	01100100 01100001 01110100 01100001 01010110 01100001 01110101 01101100 01110100 00100000 00110010 00101110 00110000

Even the same sentence can produce different hash values if the input on the bit level is not exactly the same. Raw data types (integers, floats, etc.) make things worse, as the binary representation might differ from system to system. In order to calculate a re-usable hash value, that is a hash value that is the same for the same data input, all raw data types are converted into a string before hashing. That reduces the complexity of the approach to hash the data. The following requirements have to be met by the input string:

1. **Character set:** the character set defines how bits of a string or varchar are interpreted into readable characters. Or in other terms: how bits represent characters. Because commonly available hash functions work with bits instead of characters, the character set plays an important role to ensure that the same characters end up with the same bit representation. This includes Unicode vs. non-Unicode choices.
2. **Leading and trailing spaces:** in many cases, any leading and trailing spaces don't play an important role for the business user of a data warehouse. For example, business keys usually don't include leading or trailing spaces and descriptive data is often trimmed from them as well before presented to the user. Because they don't change the meaning of the input and only the binary representation, they should not be included in the input to the hash function.
3. **Embedded or unseen control characters:** the same applies for any embedded or unseen control characters. Examples for this might be bell codes (BC) [16], carriage return (CR), new line (LF) or backspace sign, which often have no difference to the semantics of the text. Many of these control characters are inserted into text files or databases due to interoperability issues between operational systems.
4. **Data type, length, precision and scale:** the use of different data types, their length, and precision and scale settings produce different binary representations. For example, values stored in a decimal(5,2) or in a decimal(6,2) might be represented as the same characters, but are stored as different binary values. Therefore, one of the recommendations is to convert all data types to strings before hashing them.
5. **Date format:** when converting dates to strings, the question is how to represent dates in a common manner. It is recommended to cast all dates and timestamps into a standardized ISO8601 format [17].
6. **Decimal separator:** another problem when converting data types to strings concerns different regional settings. Depending on the current locale, different decimal and thousands separators are used. For example, the number 1,000.99 (in US format) is represented as 1.000,99 in Germany and 1'000.99 in Switzerland.

7. **Case sensitivity:** as already shown from the previous examples, the character case changes the binary representation of the string. Therefore, the character case has to be taken into consideration when hashing data. Depending on your data, case sensitivity needs to be turned on or off for your input data. For example, in most cases, business keys are case insensitive. Customer DE00042 is the same as de00042. There are exceptions to this rule, for example when business keys are actually case sensitive. The same applies to descriptive data that is often case sensitive, as well. We will discuss such examples in sections 11.2.4 and 11.2.5, respectively. Note that case sensitivity applies to the output as well. Some hash functions (or the accompanying conversion from binary values to a hexadecimal representation) produce a lower-case hex string, while others produce an upper-case hexadecimal string. The best practice is to convert all outputs from hash functions to upper-case.
8. **NULL values:** depending on the system where the hash value is calculated, the binary representation of a NULL value might be different, especially when converting NULL values to varchars or strings. Note that hash values are not always generated in a database but in other software environments such as the .NET framework, a Java Virtual Machine or in a Unix Shell. The recommended approach when dealing with NULL values is to always replace them by empty strings and use delimiters when concatenating multiple fields (for example, when calculating the hash key of a composite business key or the hash diff value of descriptive data).
9. **Lack of delimiters for concatenated fields:** because the recommendation is to replace NULL values by an empty string, there might be examples when different data becomes the same input after converting to strings and concatenating the individual elements of the input. Therefore the use of delimiters is required. This is described in more detail in section 11.2.4.
10. **Order of concatenated fields:** when concatenating fields, the order of the fields plays a vital role. If it is changed, the hash values become different.
11. **Endianness:** the architecture of software and hardware plays another significant role in how bytes are stored in memory. This can become an issue when not all hashes are generated on the same system, for example because some data is being hashed on other systems than the primary data warehouse system. For example, the .NET Framework is using little endian [18] (storing the least-significant byte (LSB) at the lowest memory address), while the Java Virtual Machine is using big endian [19] (storing the most significant byte (MSB) at the lowest memory address) as does Hadoop [20]. Microsoft SQL Server uses big endian in most cases [21]. On the hardware side, Intel® in its 80 × 86 architecture uses little endian [22].
12. **Everything else:** whenever the bit representation of the input string is changed, the hash value changes.

In order to deal with these issues and requirements, a first thought is to create a central function, such as a reusable user-defined function or SSIS script component that calculates the hash value by preparing the input and calling the appropriate hash function. However, this approach is often not enough, because today data is loaded into the data warehouse with various tools, for example an ETL tool such as SSIS, directly in the database (for example the stored procedure created in Chapter 10, Metadata Management) or in tools external to the data warehouse team. Other data is stored in NoSQL environments such as Hadoop and never touches the data warehouse until it is joined when building the information marts. However, if both parties use different approaches for dealing with these requirements, the join will fail, because the hash keys in hubs and links will be

completely different from each other, preventing the joining of data across systems. For that reason, the first task when starting the data warehouse initiative is to create a standards document for hash value generation. The document should not only define which hashing function should be used to calculate the hash values, but also how the input is generated to ensure the same output for the same data. Table 11.5 reviews the standards that need to be established by this document and gives best practices for each.

Note that the recommendation for Endianess and SQL Server is to use Little Endian because the HashBytes function appears to use Little Endian. Refer to section 11.6.3 for a discussion related to SQL Server and SSIS.

Addressing these requirements in the design phase and throughout the whole data warehouse lifecycle is the first step in dealing with technological risks regarding the use of hashes in data warehousing. However, there are additional risks, which are covered in the next section, that need to be mitigated.

11.2.3 RISKS OF USING HASH FUNCTIONS

When using hash keys and hash diff values in the data warehouse, there are some additional issues that need to be considered in the design phase of the project. Ignoring these risks will sooner or later hit back on the data warehouse team.

Table 11.5 Best Practices for Hashing Standards Document

Standard	Best Practice
Hashing Function	MD5
Character set	UTF-8
Leading and trailing spaces	Strip
Control characters	Remove or change to blank (space)
Data type, length, precision and scale	Standardize to regional settings of organization's headquarters
Date format	Standardize to regional settings of organization's headquarters
Decimal separator	Standardize to regional settings of organization's headquarters
Case sensitivity	Business keys: all upper-case; descriptive data: it depends
NULL values	Remove by changing to empty string or other default value
Delimiters for concatenated fields	Semicolon or comma
Endianness	Little Endian (due to SQL Server and the Java Virtual Machine which is used for Hadoop)

11.2.3.1 Hash Collisions

Section 11.2.1 introduced **collision freedom** as a desired characteristic of hash functions. When hashing business keys while loading a hub, we want to prevent the hash function producing the same hash for two different business keys. Such a collision would represent a problem for a data warehouse built with Data Vault 2.0 modeling, because other entities, such as links or satellites, reference the business key using its hash value. If two business keys are using the same hash key, the reference would not be unique.

Because hash functions compress data from a theoretically unlimited input to a fixed-length hash value, it is not possible to prevent a hash collision, which is the same hash value for two arbitrary long inputs. And in fact, there are *random inputs* with the same hash key for any given *meaningful input*. For example, the 128-bit inputs (expressed in hexadecimal notation) shown in Table 11.6 produce the same MD5 hash value.

The common MD5 hash value is `008EE33A9D58B51CFEB425B0959121C9`. This type of hash collision is called a general hash collision. It is not possible to prevent this problem if compression takes place. If all input is random, which means binary blocks of random input, there are various levels of risks, depending on the selected hash function. Figure 11.6 shows a comparison of risks (the odds of a hash collision) for CRC-32, MD5 and SHA-1.

CRC-32 is not a recommended option for a hash function in data warehousing. However, it is included in Figure 11.6 to exemplify why this is the case: the risk of collisions is just too high. If a single hub or a single link contains only 77163 hashed records (a small-sized hub), the risk of a hash collision is 1 in 2 (50%). Using MD5, at least 5.06 billion (5.06×10^9) records need to be included in the hub to get such a risk for collisions. Compare this to SHA-1: in order to reach a risk of 50%, the number of 1.42×10^{24} records have to be added into the *same* hub first. Note that if a collision happens in another Data Vault entity (two different inputs, the same hash value, but different hubs), the collision is not a problem at all because no data is referenced erroneously. Does it mean that there is no risk of hash collisions when only a small number of records to be hashed is involved? No. The risk is just negligible: if a hub contains only ~200 business keys, hashed with MD5, the risk that a meteor would hit your data center is higher than that of a collision. The problem is that there is still a minor risk involved.

It is possible to reduce the risk even further by using the SHA-1 hash function. However, SHA-1 might not be available in all tools used in the data warehousing infrastructure. In these cases, the hashing function might be coded manually if that is possible.

Table 11.6 Example Single-Block Collision [23]

```
4d c9 68 ff 0e e3 5c 20 95 72 d4 77 7b 72 15 87
d3 6f a7 b2 1b dc 56 b7 4a 3d c0 78 3e 7b 95 18
af bf a2 00 a8 28 4b f3 6e 8e 4b 55 b3 5f 42 75
93 d8 49 67 6d a0 d1 55 5d 83 60 fb 5f 07 fe a2

4d c9 68 ff 0e e3 5c 20 95 72 d4 77 7b 72 15 87
d3 6f a7 b2 1b dc 56 b7 4a 3d c0 78 3e 7b 95 18
af bf a2 02 a8 28 4b f3 6e 8e 4b 55 b3 5f 42 75
93 d8 49 67 6d a0 d1 d5 5d 83 60 fb 5f 07 fe a2
```

Figure adapted by author from "Example single-block collision," by Marc Stevens. Copyright Marc Stevens. Reprinted with permission.

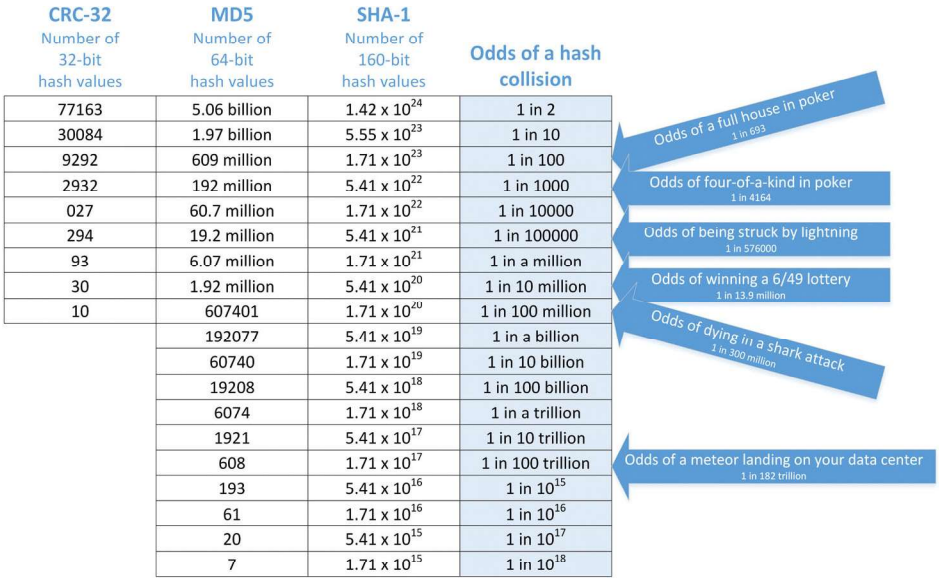

FIGURE 11.6

Probabilities of hash collisions [24].

Figure adapted by author from "Hash Collision Probabilities," by Jeff Preshing. Copyright Jeff Preshing. Reprinted with permission.

However, for data warehousing purposes, the inputs for potential collisions are not random nor binary as in Table 11.6. Instead, they are meaningful: business keys follow a (limited) string format; even descriptive data in satellites has only a limited input to be meaningful. Therefore, the chance for a collision between two meaningful inputs is even lower as presented in Figure 11.6.

In reality, while the risk of a hash collision tends to be very low for MD5 and SHA-1, there is no way to guarantee that there will be no collisions in the operational lifetime of the data warehouse. But it is possible to check for the direct opposite: when designing the data warehouse, it is possible to check if there is already a collision in the initial load. By hashing all business keys of a source file, we can find out if there are already collisions using a given hash function (such as MD5). If that is the case, we can move to a hash function (such as SHA-1) with a larger hash value output (bit-wise) before making the choice of hash function permanently. But having no hash collisions in the initial load has no meaning for the future. The first collision could still occur on the first operational day, by chance.

While it is not possible to prevent collisions, it is possible to detect them at least. When loading hub tables, we can ensure that there is no other business key in the hub having the same hash key. The same applies to link tables where we check for existing business key combinations with the same hash key. It will slow down the loading patterns a little, but ensure that no data becomes corrupted in the data warehouse. There are also techniques to detect a hash diff collision, but the chances for such collision are very low, much lower than on hash keys in hubs and links. Both techniques are described in full detail in Chapter 12, Loading the Data Vault.

11.2.3.2 Storage Requirements

Another consideration includes the required additional storage to keep the hash keys and hash diffs. Using SHA-1 instead of MD5 increases the storage requirements from 32 bytes (MD5 in hexadecimal representation) to 40 bytes (SHA-1). Compare this to a big-integer sequence number that only requires 8 bytes.

Generally, the additional storage requirements are not a real problem in the enterprise data warehouse layer (modeled in Data Vault 2.0 notation). In our experience, the advantages (such as the loading performance) outweigh this disadvantage of consuming more storage.

However, most data warehouse teams prefer (big) integer surrogate keys in information marts, for two reasons: first, business users have direct access to the information mart and might be confused by hash keys in dimension and fact tables. In addition, the storage requirements for using hash keys in fact tables are a concern for many teams. Because fact tables usually refer to multiple dimension tables and contain many rows, the additional storage requirement of a hash key becomes an issue. Consider the example of a fact table that refers to 10 dimension tables. Each fact requires 320 bytes for referencing the 10 dimensions if MD5 is used. If the fact table contains 10 million rows, around 3 GB are required to store only the references. Using SHA-1 requires 3.8 GB for storing the references to the dimension tables. Compare this to 762 MB when using 8-byte integers. If you decide to avoid MD5 in favor of a SHA algorithm, it is recommended to use SHA-1 over SHA-256 (or higher) because of storage requirements. SHA-1 already provides superior resistance against collisions, which makes SHA-256 in most if not all cases irrelevant.

For that reason, information marts are typically modeled using sequence number instead of hash keys (refer to Chapter 7, Dimensional Modeling, for a detailed description).

When storing hashes, avoid using the varchar datatype. Columns that use a varchar datatype might be stored in text pages instead of the main data page under some circumstances. Microsoft SQL Server moves columns with variable length out of the data page if the row size grows over 8,060 bytes [26]. Because hash keys are used for joining, the join performance will greatly benefit from having the hash key in the data page. If the hash key is stored in the text page, it has to be de-referenced first. Columns using a fixed-length datatype are guaranteed to be included in the data page.

On some occasions, data warehouse teams try to save storage by using binary(16) for MD5 hashes or binary(20) for SHA-1 hashes. Doing so limits the interoperability with other systems, such as Hadoop or BizTalk. Therefore, this is not a recommended practice. If you decide to do it anyway, avoid using the varbinary datatype for the same reasons as avoiding the varchar datatype. More interoperability issues are discussed in the next section.

11.2.3.3 Tool and Platform Compatibility

One of the reasons why hash keys have been introduced is that they improve the interoperability between different platforms, such as the relational database and NoSQL environments. By using hash keys in hubs and links, it is possible to integrate data on various platforms, structured in the relational database and unstructured data in NoSQL environments such as Hadoop. For that reason, the recommended data type to store hash keys is varchar because it is easy to read and write by external applications. If other datatypes are used, an on-the-fly conversion might be required which slows down the read/write process and makes it more complex.

Compatibility with other tools and platforms is also the reason to recommend MD5 for hash keys. Many systems, such as ETL tools, are capable of calculating MD5 hash keys. While SHA-1 is the newer algorithm and the recommended hashing function for encryption purposes nowadays, not all tools typically used in the business intelligence domain support it. On the other hand, most ETL tools

and database systems support the MD5 hashing function out of the box. For the same reason, it is not recommended to use MD6 (the direct successor of MD5), SHA-256 or any other hashing algorithm that doesn't have widespread support compared to MD5 or SHA-1.

Before making a decision regarding the choice of hash function, the available tools and their hashing capabilities should be carefully reviewed to avoid surprises in later phases of the project.

11.2.3.4 Maintenance

In the rare case that a collision has occurred, the resolution includes increasing the bit-size of the hash value (hash key or hash diff) by changing the hash function. For example, if MD5 was used when a collision occurred, the recommendation is to move to SHA-1. This upgrade requires increasing the length of the hash key column (for example from 32 characters to 40) and to recalculate the hash values based on the business keys in the hub or link or the descriptive data in the satellites. Consider the recalculation of a hub's hash keys. Because other links and satellites reference these hash keys in order to reference the hub (for example to describe the business key in a satellite), at a minimum, all hash keys in dependent entities have to be recalculated, as well. Theoretically, all other hash keys could be left at the smaller size.

But leaving some hash keys in the smaller hash size would increase the maintenance costs because more management is required. It also hinders any automation efforts or requires more complex metadata to accomplish this task. Therefore, the recommendation is to use only one hash function for calculating all the hashes in the data warehouse.

For the same reason, it should be avoided to use different hash algorithms in the relational data warehouse and unstructured NoSQL environments, such as Hadoop (Figure 11.7).

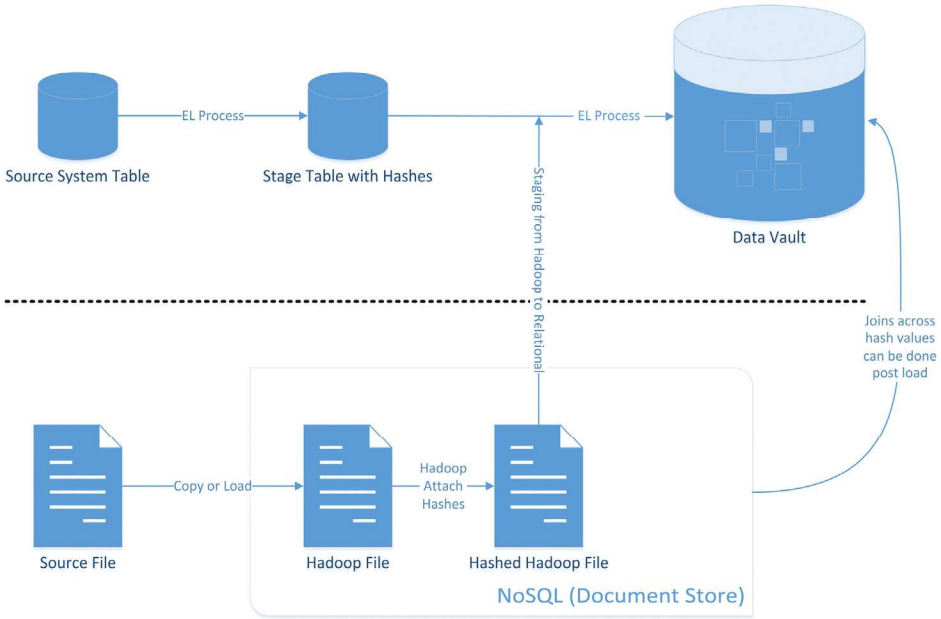

FIGURE 11.7

Integrating distinct parallel load operations.