Oracle Utilities Analytics

Developer's Guide

Release 2.5.2.0.1

E65568-02

June 2016

ORACLE

Oracle Utilities Analytics Developer's Guide, Release 2.5.2.0.1

E65568-02

Contents

Contents

Preface

This guide provides instructions for configuring and administering Oracle Utilities Analytics (OUA), including:

Audience

This document is primarily for the developers who would want to extend the functionality of the product based on their custom requirements. This document does not teach Oracle Data Integrator or Oracle Business Intelligence Enterprise Edition fundamentals but expects the users to be familiar with development using Oracle Data Integrator and Oracle Business Intelligence Enterprise Edition.

> **Note**: This document assumes that the developer is using a Unix environment for executing the scripts and commands. A windows machine can also be used for these actions however the "sh" scripts have to be replaced with the corresponding "cmd" scripts.

Prerequisite Knowledge

Oracle Utilities Extractors and Schema and Oracle Utilities Analytics Dashboards use several technologies. You should have knowledge of the following before configuring and administering Oracle Utilities Analytics:

- Oracle Data Warehouse:

  http://docs.oracle.com/cd/E11882_01/server.112/e25554/toc.htm

- Oracle Data Integrator:

  http://docs.oracle.com/cd/E28280_01/nav/odi.htm

- Oracle GoldenGate:

  http://docs.oracle.com/goldengate/1212/gg-winux/index.html

- Oracle WebLogic Server:

http://docs.oracle.com/cd/E23943_01/nav/wls.htm

- Oracle Business Intelligence Enterprise Edition:

  http://docs.oracle.com/cd/E28280_01/nav/bi.htm

How This Guide is Organized

This guide is organized based on the typical flow a user implementing the Oracle Utilities Analytics product goes through.

Related Documents

The following documentation is included with this release:

- *Oracle Utilities Analytics Getting Started Guide*
- *Oracle Utilities Analytics License Information User Guide*
- *Oracle Utilities Analytics Administration Guide*
- *Oracle Utilities Analytics Installation Guide*
- *Oracle Utilities Analytics Quick Install Guide*
- *Oracle Utilities Analytics Release Notes*
- *Oracle Utilities Analytics Dashboards for Oracle Utilities Meter Data Analytics Metric Reference Guide*
- *Oracle Utilities Analytics Dashboards for Oracle Utilities Customer Analytics, Revenue Analytics and Credit & Collections Analytics Metric Reference Guide*
- *Oracle Utilities Analytics Dashboards for Oracle Utilities Exception Analytics Metric Reference Guide*
- *Oracle Utilities Analytics Dashboards for Oracle Utilities Mobile Workforce Analytics Metric Reference Guide*
- *Oracle Utilities Analytics Dashboards for Oracle Utilities Distribution Analytics and Outage Analytics Metric Reference Guide*
- *Oracle Utilities Analytics Dashboards for Oracle Utilities Work and Asset Analytics Metric Reference Guide*
- *Oracle Utilities Analytics Dashboards for Oracle Utilities Operational Device Analytics Metric Reference Guide*
- *Oracle Utilities Extractors and Schema for Oracle Utilities Customer Care and Billing Data Mapping Guide*
- *Oracle Utilities Extractors and Schema for Oracle Utilities Meter Data Management Data Mapping Guide*
- *Oracle Utilities Extractors and Schema for Oracle Utilities Mobile Workforce Management Data Mapping Guide*
- *Oracle Utilities Extractors and Schema for Oracle Utilities Network Management System Data Mapping Guide*
- *Oracle Utilities Extractors and Schema for Oracle Utilities Operational Device Management Data Mapping Guide*
- *Oracle Utilities Extractors and Schema for Oracle Utilities Work & Asset Management Data Mapping Guide*

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
monospace	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

Acronyms

The list of acronyms used in this guide is as explained below:

- **APEX**: Oracle Application Express
- **CC&B**: Oracle Utilities Customer Care and Billing
- **CDC**: Changed Data Capture
- **ELT**: Extraction, Loading and Transformation
- **ETL**: Extraction, Transformation and Loading
- **MDM**: Oracle Utilities Meter Data Management
- **MWM**: Oracle Utilities Mobile Workforce Management
- **NMS**: Oracle Utilities Network Management System
- **OBIEE**: Oracle Business Intelligence Enterprise Edition
- **ODI**: Oracle Data Integrator
- **ODM**: Oracle Utilities Operational Device Management
- **OGG**: Oracle GoldenGate
- **OUA**: Oracle Utilities Analytics
- **OWB**: Oracle Warehouse Builder
- **WAM**: Oracle Utilities Work and Asset Management

Chapter 1

Getting Started

Before starting any customization, you need to create a custom project so that all customizations are isolated from the product components. You should create all the custom objects under the custom project. This chapter includes the following topics:

Naming Conventions

All out of the box objects are prefixed with 'B1' and should not be modified. It is recommended that the client also choose a two character code for prefixing their custom objects to avoid any naming conflicts between the product components and the custom components.

The recommendation is to use the CM prefix for all objects created by you, with CM being a reference to Customer Modification.

Project

You can create a new project for maintaining all custom interfaces, procedures and packages by logging into Oracle Data Integrator Studio. It is recommended that you should use the following folder structure within the project to organize the objects:

- **Facts**: All the **Fact** interfaces should be organized under this folder.
- **Dimensions**: All the **Dimension** interfaces should be organized under this folder.
- **Replication**: All the **Replication View** interfaces should be organized under this folder.
- **Materialized Views**: All the **Materialized View** interfaces should be organized under this folder.

Sub folders can be created based on the product for which the component is being developed. Avoid cross references across the different folders. For example, an interface under the **Dimensions** folder should not refer to an interface in the **Replication** folder.

Model Folder

All custom model objects should reside under a custom model folder. A pattern similar to the folder structures within the project can be followed.

CM Metadata User Procedure

Always use this user procedure to create new entries into the metadata tables. This will make it easier to migrate the same metadata to different environments. This procedure can be used to populate custom labels for the dashboards.

1. Create a procedure **CM_<PROD_FLG>_CREATE_METADATA**.
   Replace the **<PROD_FLG>** with the appropriate edge product code (For example, CCB/ NMS/ MDM/MWM).
2. Add appropriate data population scripts.
   These should be written as merge statements so that existing rows are skipped and only new rows are added. In case the metadata requires corrections, use the update clause of the merge statement. All tasks within the procedure should have the logical schema set to "Metadata". The schema names should not be hardcoded.
3. Create a package **CM_<PROD_FLG>_CREATE_METADATA**.
4. Add the procedure created above as the first step and add the scenario **B1_CFG_METADATA** as a second step.
   The **B1_CFG_METADATA** pulls in the additional metadata from source based on the list of tables to extend the replication.
5. After migrating the CM Project to new environment, execute the custom procedure **CM_<PROD_FLG>_CREATE_METADATA** mentioned above after the addition of the product instance.
   This job should be executed in the newly created context for the product.

Chapter 2

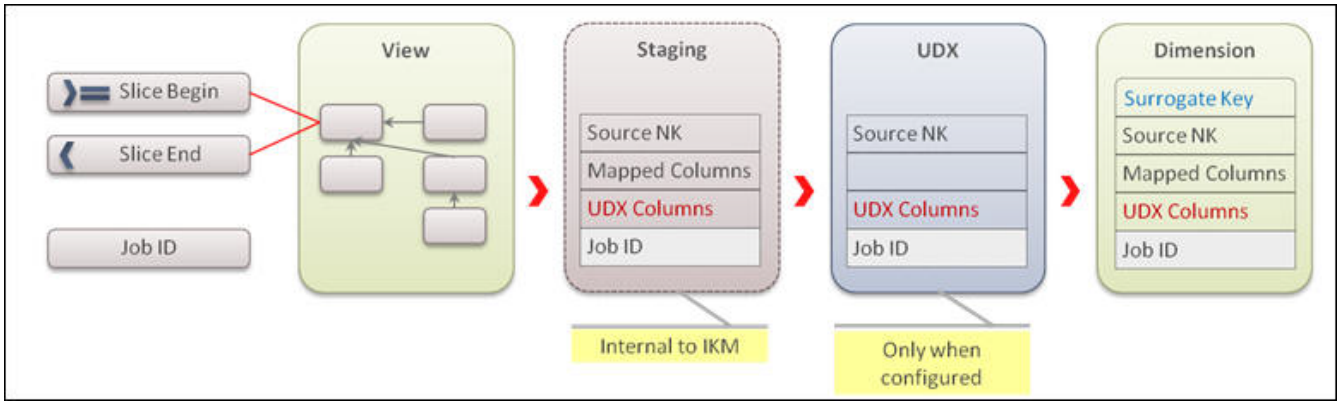
Extending Oracle Utilities Analytics

The Oracle Utilities Analytics product provides a comprehensive set of star schemas in each of its analytics. However, despite the broad analytics, there are few cases when the star schemas need to be extended to suit business analytic needs that are not supported by the base solution. As part of implementation of utility products, utilities may add customized logic that suits their business needs. For example, utilities may introduce additional fields, via configurable characteristics, to objects that were delivered with minimal fields. Because of the highly configurable nature of the edge applications, it is necessary that the Oracle Utilities Analytics product be customizable and extendable. Keeping this in mind, the star schemas have been created with the following extensible attributes in the facts and dimensions:

- Dimension Pattern
- Extending Dimensions
- Fact Pattern
- Extending Facts

Dimension Pattern

The diagram below illustrates the stages of processing in a dimension, and the components utilized in developing a dimension load process.



Dimension Pattern

All the data load processes comprise of a package, and one or more interfaces. The package uses the following mandatory variables as input:

The standard Oracle Data Integrator interfaces process an entire table data set in a single execution. This approach is not scalable for large volumes of data. Oracle Utilities Analytics implements a configuration driven data slicing mechanism to subdivide data into smaller evenly

distributed slices and processes these slices in parallel. The slice start timestamp and the slice end timestamp are calculated values based on configuration and are passed as parameters to each entity load process. This way the same interface is reused for parallel execution.

- **B1_SLICE_BEG_TS**: This is used to pass the slice start timestamp.
- **B1_SLICE_END_TS**: This is used to pass the slice end timestamp.
- **B1_JOB_ID**: This is used to pass the current job identifier.

The first interface is usually a view where filters are applied based on the variables to exclude data that does not fall into the specified range (B1_SLICE_BEG_TS - B1_SLICE_END_TS). This reduces the data processed in one execution. The data from the view is first inserted into a staging table. The staging table contains the following columns:

- Source natural key columns.
- Columns mapped to target or used for filters.
- Columns marked for user extension (for dimensions these are UDF codes and description columns).
- Job identifier to segregate the data from multiple parallel executions of a data load process.

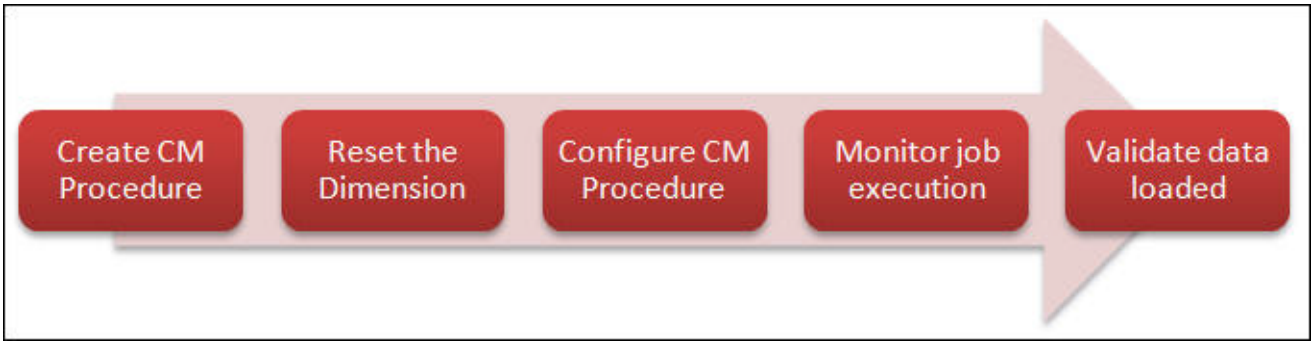
The UDX table (see the section UDX Processing for the details on the UDX table) is created only if the CM procedure has been configured for the entity. This table contains the following types of columns

- Source natural key columns.
- Columns marked for user extension (for dimensions these are UDF codes and description columns).
- Job identifier to segregate the data from multiple parallel executions of a data load process.

The data is finally loaded into the target dimension.

Extending Dimensions

The diagram below illustrates the steps required to extend a dimension.



Extending Dimension

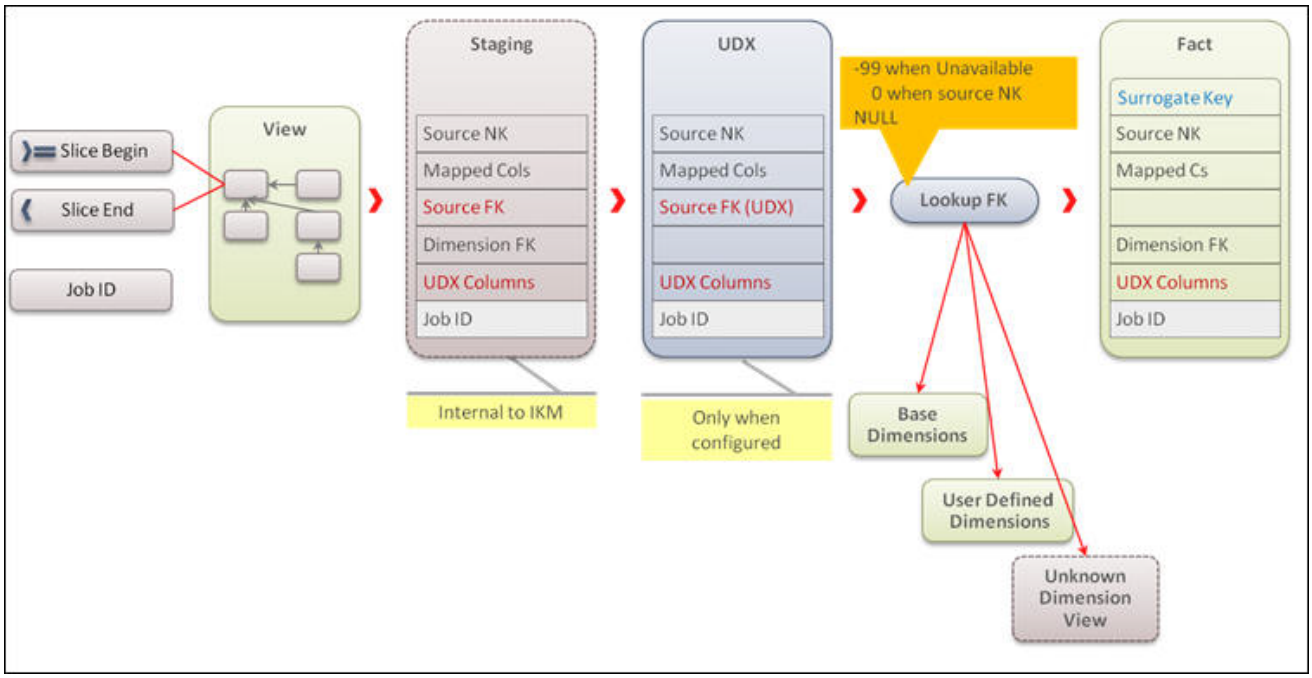
Extending a dimension is a fairly simple process that requires you to create an Oracle Data Integrator procedure that compiles a PL/SQL CM procedure. The CM procedure updates the user defined fields (In general, all the dimensions consist of a minimum of ten UDF columns. These columns are utilized to store additional information from the source systems. For example, UDF1_CD, UDF2_CD, UDF1_DESCR, UDF2_DESCR, and so on) columns based on the input parameters and the natural key of the UDX table.

Once the procedure has been written, configure it and enable the jobs (Refer to the section UDX Processing in the **Chapter 4: Extending Star Schema**). If data has already been loaded, then the

user defined fields are populated for incremental changes. To load the data for all rows, the dimension needs to be reset using the reset scenario. Note that resetting a dimension resets the dependent facts also.

Fact Pattern

The diagram given below illustrates the stages of processing in a fact and the components utilized in developing a fact load process.



All data load processes comprise of a package and one or more interfaces. The package uses the following mandatory variables as input:

- **B1_SLICE_BEG_TS**: This is used to pass the slice start timestamp.
- **B1_SLICE_END_TS**: This is used to pass the slice end timestamp.
- **B1_JOB_ID**: This is used to pass the current job identifier.

The first interface is usually a view where filters are applied based on the variables to exclude data that does not fall into the specified range (B1_SLICE_BEG_TS - B1_SLICE_END_TS). This reduces the data processed in one execution. The data from the view is first inserted into a staging table. The staging table contains the following columns:

- Source natural key columns.
- Columns mapped to target or used for filters.
- Columns marked for user extension (these are UDDGEN, UDM and UDD_KEY columns).There can be multiple columns for each of these types (For example, UDDGEN1, UDM1, UDD1_KEY, UDD2_KEY, and so on).
- Columns required for looking up the foreign keys to dimensions.
- Job identifier to segregate data from multiple parallel executions of a data load process.

The UDX table (see the section UDX Processing for the details on the UDX table) is created only if the CM procedure has been configured for the entity. This table contains the following types of columns:

- Source natural key columns.

- Columns marked for user extension (these are UDDGEN, UDM and UDD_KEY columns, for example, UDDGEN1, UDM1, UDD1_KEY, UDD2_KEY, and so on).
- Columns required for looking up the foreign keys to dimensions.
- Job identifier to segregate data from multiple parallel executions of a data load process.

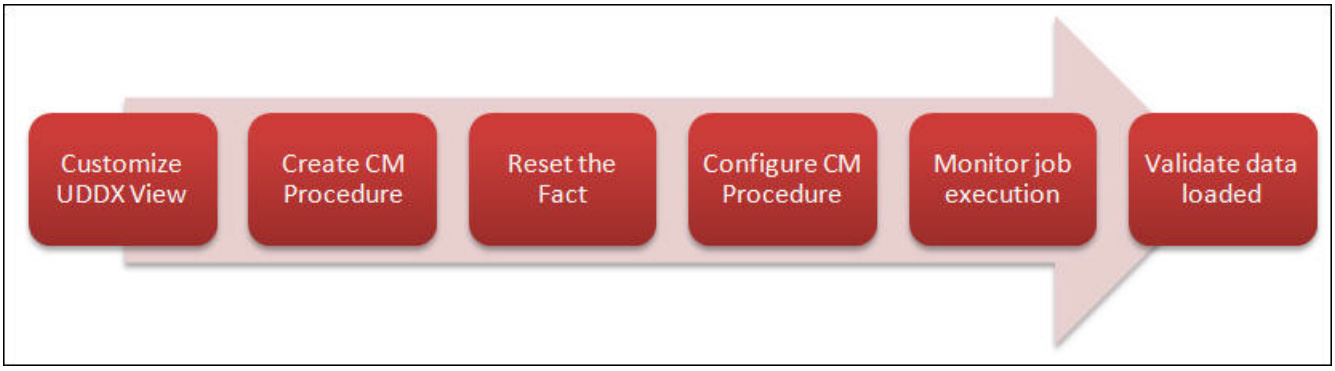
An additional step in the fact processing is the foreign key lookup for dimensions. There are three types of dimensions:

- Base dimensions are the dimensions that are populated out of box.
- User Defined Dimensions (UDDs) are additional dimensions for which a template table is provided in the out of the box product.
- Unknown dimensions are the objects where tables are not provided and you need to create custom dimensions. There is a built in lookup so that custom UDD lookups do not require code change.

The data is finally loaded into the target dimension.

Extending Facts

The diagram below illustrates the steps required to extend a fact.



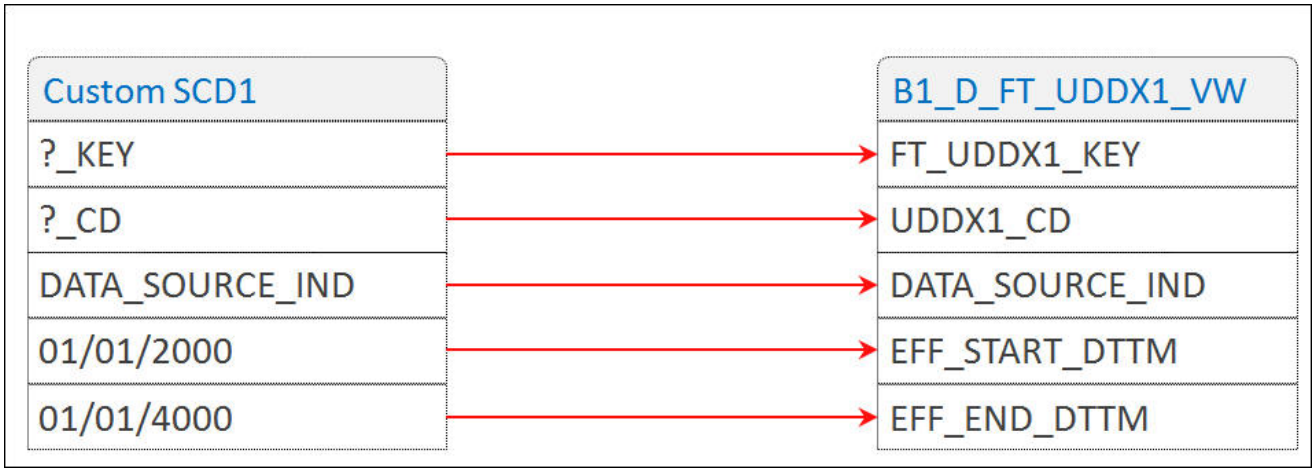
Extending a fact is similar to the process for extending a dimension with an additional step required for custom dimension lookup.

Using a Custom User-Defined Dimension (UDD)

All out of the box facts provide a few UDD_KEY columns for extending the functionality of the fact by associating a custom dimension to the fact (for example, UDD1_KEY, UDD2_KEY, and so on). To utilize this functionality, customize the UDDX view associated with the UDD key.

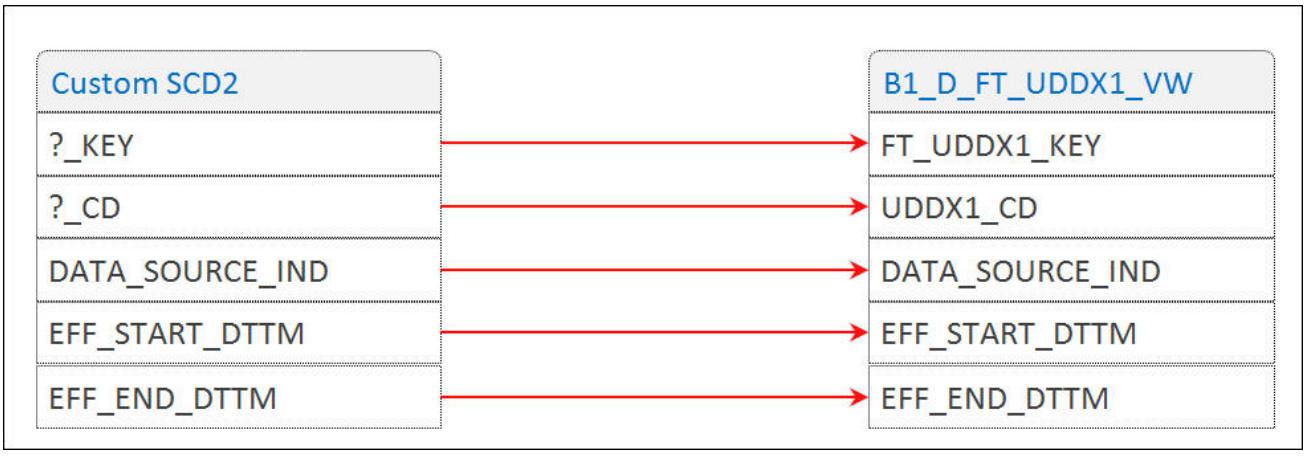
SCD1

If a custom dimension lookup is required, then the UDDX views need to be customized first to refer to a custom dimension as illustrated below. Assuming that the custom dimension is of type 1, and then create the mapping as shown below to override the UDDX view. In the given example, the custom SCD1 dimension is used to link to the CF_FT fact's UDD1_KEY column.



SCD2

Assuming that the custom dimension is of type 2, and then create the mapping as shown below to override the UDDX view. In the given example, the custom SCD2 dimension is used to link to CF_FT fact's UDD1_KEY column.



This ensures that the lookup functions as designed, and now you will have an out of the box fact referring to a custom dimension. Once this is done, create an Oracle Data Integrator procedure that compiles a PL/SQL CM procedure. The CM procedure updates the user defined fields (In general, all the dimensions consist of a minimum of ten UDF columns. These columns are utilized to store additional information from the source systems. For example, UDF1_CD, UDF2_CD, UDF1_DESCR, UDF2_DESCR, and so on) columns based on the input parameters and the natural key of the UDX table.

Once the procedure has been written, configure it and enable the jobs (Refer to the section UDX Processing in the **Chapter 4: Extending Star Schema**). If data has already been loaded, then the user defined fields are only populated for incremental changes. To load the data for all rows, reset the dimension using the reset scenario.

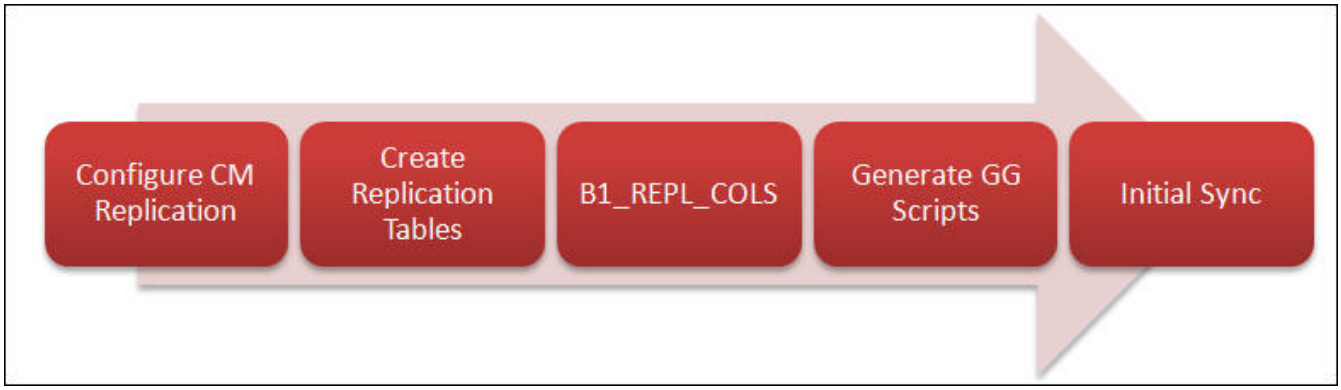
Note: Resetting a dimension resets the dependent facts also.

Chapter 3

Extending Replication

The Oracle Utilities Analytics product allows you to extend the capabilities of the product. The out of the box solution has been set up to enable replication of several tables required for processing and loading data into the data warehouse. Implementer's requirements, however, may vary and they may want to see additional information in their facts and dimensions that are not included in the out of the box solution. Some of these extension requirements may be met by using the tables which are already being replicated out of the box. For others, it may be required to include additional tables in the replication process.

The diagram below illustrates the steps required to include a table for replication that is currently not set up for replication.



Extending Replication

To configure replication, login to the **Administration** user interface, navigate to **Source Table** configuration. Identify the table to be replicated. Set the **CM Replication** flag to "Yes". Perform the Oracle GoldenGate setup and complete the initial synchronization.

This chapter covers the following topics:

- Replicating Custom Tables for Oracle Utilities Application Framework Source Applications
- Replicating Custom Tables for Oracle Utilities Network Management System Source Applications
- Enabling Replication
- Creating a Replicated Table
- Configuring Online and Initial Sync
- Verification of Model Setup

Replicating Custom Tables for Oracle Utilities Application Framework Source Applications

Most of the tables related to the tables used for populating the out of the box star schemas are listed in the metadata configuration "Source Tables". If the table required for extension is not listed here, perform the following steps:

1. Create a procedure **CM_<PROD_FLG>_CREATE_METADATA**. Replace the **<PROD_FLG>** with the appropriate edge product code (For example, CCB/NMS/MDM/MWM).
2. Create a new task for each metadata entry into **B1_OBJECT_MAP**. All the tasks within the procedure should have the logical schema set to "Metadata".
3. Add an entry in the **B1_OBJECT_MAP** setting the **SOURCE_OBJECT_NAME** as the table name and the **TARGET_OBJECT_NAME** as the target fact or dimension, which has some attributes loaded from this table.

   These should be written as merge statements so that existing rows are skipped and only new rows are added. In case the metadata requires corrections, use the update clause of the merge statement. The schema names should not be hardcoded.

   As an example, the merge statement given below is for setting up the tables under a maintenance object in Oracle Utilities Customer Care and Billing for inclusion in the replication process.

   - The Source Product Flag should be the product flag of the source. In this example, it is 'CCB' for Oracle Utilities Customer Care and Billing.
   - The Source Object Name should be the source maintenance object. In this example, the tables are included under the Budget Review maintenance object, it is specified as 'BUD REVIEW', which is the maintenance object code for Budget Review in Oracle Utilities Customer Care and Billing.
   - The Target Object Name should be the ETL view that uses the tables of this maintenance object. In this example, **CM_TEST_VW** is specified as dummy value.

- The **Object Type Flag** is the type of object that is being replicated. In this example, replicating the entire Budget Review MO is specified; hence 'MO' has been specified.

```
merge
 into b1_object_map  tgt
using (select 'CCB'                                prod_flg
           , 'BUD REVIEW'                         source_object_name
           , 'CM_TEST_VW'                         target_object_name
           , 1                                      seq
           , 'MO'                                   object_type_flg
        from dual )  tgt_val
   on (    tgt.prod_flg           = tgt_val.prod_flg
      and tgt.source_object_name = tgt_val.source_object_name
      and tgt.target_object_name = tgt_val.target_object_name
      and tgt.seq                = tgt_val.seq)
when not matched
then insert
     (
       tgt.object_map_id
     , tgt.prod_flg
     , tgt.source_object_name
     , tgt.target_object_name
     , tgt.seq
     , tgt.object_type_flg
     , tgt.char_entity_flg
     , tgt.upd_dttm
     , tgt.upd_user
     , tgt.owner_flg
     )
     values
     (
       b1_object_map_seq.nextval
     , tgt_val.prod_flg
     , tgt_val.source_object_name
     , tgt_val.target_object_name
     , tgt_val.seq
     , tgt_val.object_type_flg
     , null
     , sysdate
     , sys_context('userenv', 'os_user')
     ,'B1');
```

4. The **Insert** statement mentioned below is to specify that the **CM_TEST_VW** ETL view is used to populate the target **CM_F_FT**.

```
merge
 into b1_object_map  tgt
using (select 'CCB'                                prod_flg
           , 'CM_TEST_VW'                         source_object_name
           , 'CM_F_FT'                            target_object_name
           , 1                                      seq
           , 'PRVW'                                 object_type_flg
        from dual )  tgt_val
   on (    tgt.prod_flg           = tgt_val.prod_flg
      and tgt.source_object_name = tgt_val.source_object_name
       and tgt.target_object_name = tgt_val.target_object_name
       and tgt.seq                = tgt_val.seq)
when not matched
then insert
     (
       tgt.object_map_id
     , tgt.prod_flg
     , tgt.source_object_name
     , tgt.target_object_name
     , tgt.seq
     , tgt.object_type_flg
     , tgt.char_entity_flg
     , tgt.upd_dttm
     , tgt.upd_user
     , tgt.owner_flg
     )
     values
     (
       b1_object_map_seq.nextval
     , tgt_val.prod_flg
     , tgt_val.source_object_name
     , tgt_val.target_object_name
     , tgt_val.seq
     , tgt_val.object_type_flg
     , null
     , sysdate
     , sys_context('userenv', 'os_user')
     ,'B1');
```

5. Create a package **CM_<PROD_FLG>_CREATE_METADATA**.
   Add the procedure created in the first step.
   Add the scenario **B1_CFG_METADATA** as a second step.
   Add the scenario **B1_CFG_INSTANCE_JOBS** as third step.
   After migrating the CM Project to new environment, execute the custom procedure **CM_<PROD_FLG>_CREATE_METADATA** mentioned above after the addition of the product instance.
   This job should be executed in the newly created context for the product.

Executing the newly created package in the appropriate context ensures that the required tables are now present in the metadata configuration tables and you can follow the instructions under the section Enabling Replication.

The instructions given here are applicable to all the source products except for Oracle Utilities Network Management System, which does not use the Oracle Utilities Application Framework.

> **Note**: Refer to the section **Mapped Objects** under the **Chapter 2: Oracle Utilities Extractors and Schema** in *Oracle Utilities Analytics Administration Guide*.

Replicating Custom Tables for Oracle Utilities Network Management System Source Applications

Most of the tables related to the tables used for populating the out of the box star schemas are listed in the metadata configuration "Source Tables". If the table required for extension is not listed here, follow the steps below.

1. Create a procedure **CM_NMS_CREATE_METADATA**.
2. Create a new task for each metadata entry into **B1_OBJECT_MAP**. All tasks within the procedure should have the logical schema set to "Metadata".
3. Add an entry in the **B1_OBJECT_MAP** setting the **SOURCE_OBJECT_NAME** as the table name and the **TARGET_OBJECT_NAME** as the target fact or dimension, which has some attributes loaded from this table.

   These should be written as merge statements so that existing rows are skipped and only new rows are added. In case the metadata requires corrections, use the update clause of the merge statement. The schema names should not be hardcoded.

   As an example, the merge statement given below is for setting up the tables under a maintenance object in Oracle Utilities Customer Care and Billing for inclusion in the replication process.

   - The Source Product Flag should be the product flag of the source. In this example, it is 'NMS' for Oracle Utilities Network Management System.
   - The Source Object Name should be the source table. In this example, the table 'CM_XYZ' is included.
   - The Target Object Name should be the ETL view that uses the tables of this maintenance object. In this example, **CM_TEST_VW** is specified as dummy value.

- The **Object Type Flag** is the type of object that is being replicated. In this example, a specific table is replication, hence 'TBL' is been specified.

```
merge
 into b1_object_map  tgt
using (select 'NMS'                                prod_flg
            , 'CM_XYZ'                            source_object_name
            , 'CM_TEST_VW'                        target_object_name
             , 1                                    seq
             , 'TBL'                                object_type_flg
         from dual )  tgt_val
   on (    tgt.prod_flg           = tgt_val.prod_flg
      and tgt.source_object_name = tgt_val.source_object_name
       and tgt.target_object_name = tgt_val.target_object_name
       and tgt.seq                = tgt_val.seq)
when not matched
then insert
     (
       tgt.object_map_id
     , tgt.prod_flg
     , tgt.source_object_name
     , tgt.target_object_name
     , tgt.seq
     , tgt.object_type_flg
     , tgt.char_entity_flg
     , tgt.upd_dttm
     , tgt.upd_user
     , tgt.owner_flg
     )
     values
     (
       b1_object_map_seq.nextval
     , tgt_val.prod_flg
     , tgt_val.source_object_name
     , tgt_val.target_object_name
     , tgt_val.seq
     , tgt_val.object_type_flg
     , null
     , sysdate
     , sys_context('userenv', 'os_user')
     ,'B1');
```

4. The **Insert** statement mentioned below is to specify that the ETL view **CM_TEST_VW** is used to populate the target **CM_F_ZZZ**. The Source Product Flag is NMS here.

```
merge
 into b1_object_map  tgt
using (select 'NMS'                                prod_flg
            , 'CM_TEST_VW'                        source_object_name
            , 'CM_F_ZZZ'                          target_object_name
             , 1                                    seq
             , 'PRVW'                               object_type_flg
         from dual )  tgt_val
   on (    tgt.prod_flg           = tgt_val.prod_flg
      and tgt.source_object_name = tgt_val.source_object_name
       and tgt.target_object_name = tgt_val.target_object_name
       and tgt.seq                = tgt_val.seq)
when not matched
then insert
     (
       tgt.object_map_id
     , tgt.prod_flg
     , tgt.source_object_name
     , tgt.target_object_name
     , tgt.seq
     , tgt.object_type_flg
     , tgt.char_entity_flg
     , tgt.upd_dttm
     , tgt.upd_user
     , tgt.owner_flg
     )
     values
     (
       b1_object_map_seq.nextval
     , tgt_val.prod_flg
     , tgt_val.source_object_name
     , tgt_val.target_object_name
     , tgt_val.seq
     , tgt_val.object_type_flg
     , null
     , sysdate
     , sys_context('userenv', 'os_user')
     ,'B1');
```

5. Create a package **CM_NMS_CREATE_METADATA**.
   Add the procedure created above as the first step.
   Add the scenario **B1_CFG_METADATA** as a second step.
   Add the scenario **B1_CFG_INSTANCE_JOBS** as third step.
   After migrating the CM Project to new environment, execute this newly created package after the addition of the product instance.
   This job should be executed in the newly created context for the product.

Executing the newly created package in the appropriate context ensures that the required tables are now present in the metadata configuration tables and you can follow the instructions under the section Enabling Replication.

Enabling Replication

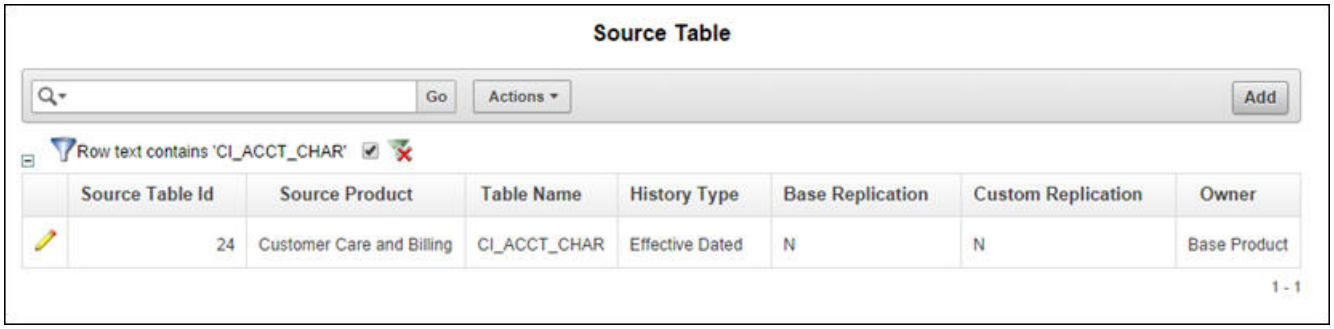
This section describes an example that guides the user through the steps of extending the replication. For the screenshots, the Oracle Utilities Customer Care and Billing product and the table **CI_ACCT_CHAR** has been used for illustration only. The product you implement and the tables to be configured differ and the values should be appropriately modified before performing this exercise.

This section uses the following conventions:

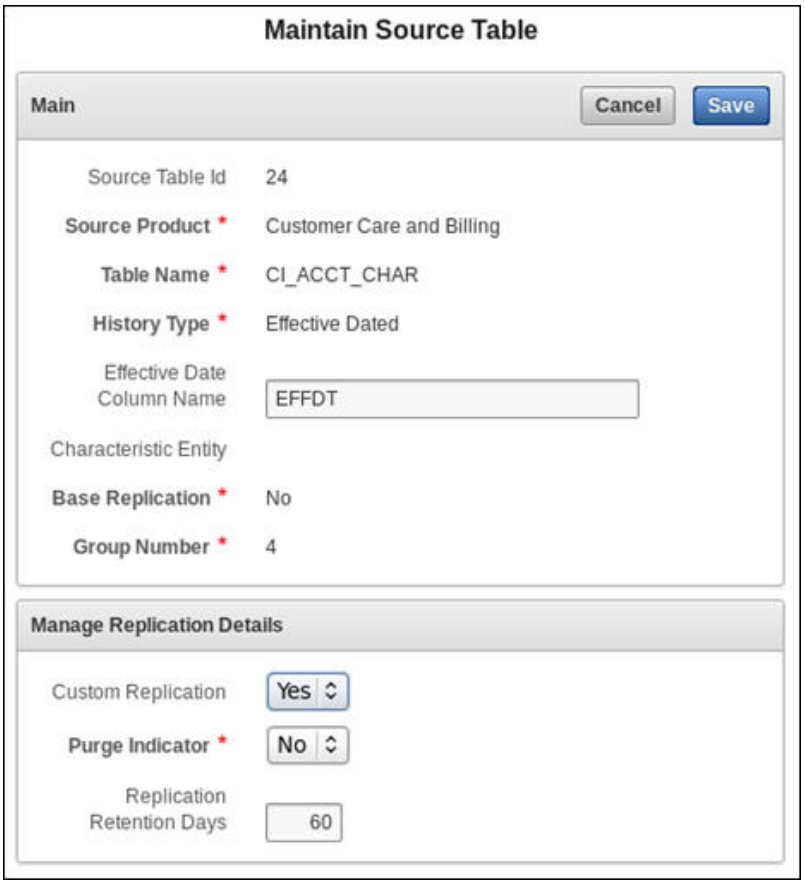
- >> "{Product}" is used to denote the product code (For example, Oracle Utilities Customer Care and Billing, Oracle Utilities Network Management System, Oracle Utilities Operational Device Management, Oracle Utilities Meter Data Management or Oracle Utilities Mobile Workforce Management).
- >> "{Table}" where the table name should be specified.
- >> "{Context}" where the context should be specified.

To enable CM replication, perform the following steps:

1. Open the **Oracle Utilities Administration** user interface in a browser and navigate to **ETL Configuration** > **Source Tables**. Filter by **CI_ACCT_CHAR** and click **Go**.
2. The following page appears.
   Edit the record by clicking on the "Yellow Pencil" icon.



3. The **Maintain Source Table** page appears.
   In the **Maintain Source Table** page, select **Yes** from the **Custom Replication** drop-down list.



4. Click **Save** to save the changes.

Creating a Replicated Table

The configuration changes have been done, but the table has not been replicated yet. The next step is to get the table created in the replication schema by following these step:

1. Follow the steps in sections "**Invoking Source Configuration User Interface**" and "**Using Source Configuration User Interface to Register or Upgrade Sources**" of the *Admin Guide*. Make sure to select **Upgrade Source** option while running the Source Configuration tool.

   This generates the Oracle GoldenGate scripts.

2. Log into SQL Developer and run the query below to verify that the table has been replicated, but there is no data in the table.

```
select *
  from ccb1rep.ci_acct_char;
```

Configuring Online and Initial Sync

The generated Oracle GoldenGate scripts include scripts that should be set up on the source Oracle GoldenGate and scripts that need to be set up on the target Oracle GoldenGate install. To configure online and initial sync, perform the following steps:

1. Run the command mentioned below in the command prompt to go to the directory where the Oracle GoldenGate scripts are generated.
2. Look for a folder named **CCB1AE**. This is the new model that was created by the scripts for processing the new table. You will notice that there is a *Readme.txt* and two folders "src" and "stg".

   ```
   cd <SPLEBASE>/GGScriptsGen
   ```

3. Copy the scripts from "src" folder to the source GoldenGate in the source server.
4. Copy the scripts from "stg" folder to the target GoldenGate in the target server.
5. Open the *Readme.txt*.

   ```
   vi CCB1AE/Readme.txt
   ```

6. Open a new terminal window and use it to follow the steps mentioned in the *Readme.txt* file:
   a. Run the oby file on the source for the new model.
   b. <Source GG Home>/ggsci paramfile <Source GG Home>/diroby/CCB1AES.oby.
   c. Run the oby file on the target for the new model.
   d. <Target GG Home>/ggsci paramfile <Target GG Home>/diroby/CCB1AET.oby.
7. Set the Oracle home by running the command mentioned below and provide Oracle SID when prompted for the ORACLE_SID. This is required for running Oracle GoldenGate command line.

   ```
   . oraenv
   ```

8. Go to the target GoldenGate prompt and check the status of the model job

   ```
   cd <Target GG Home>
   ./ggsci
   ```

   This brings up the Oracle GoldenGate command prompt. Enter "status CCB1AE".
   If you get a message "REPLICAT CCB1AE: RUNNING", then enter "stop CCB1AE".
9. Run B1_SYNC_CONTEXT after verifying that the process "CCB1AE" is stopped as per the instructions in the *Readme.txt* file.

Verification of Model Setup

Verify that the model has been set up. If the record does not exist, then it means that the Oracle GoldenGate scripts for the model CCB1AE were not deployed.

```
select *
 from mdadm.b1_checkpoint
 where group_name = 'CCB1AE';
```

Verify that the table data has been synced up. If there is no entry it means that the scenario B1_SYNC_CONTEXT was not executed or if it was executed, then the Oracle GoldenGate scripts were not deployed at that time.

```
select *
 from mdadm.b1_table_sync
 where model_cd = 'CCB1AE';

Select *
 from ccb1rep.ci_acct_char;
```

Important Note:

- Enable all the replication tables required for customization, and then follow the steps mentioned in the sections Creating a Replicated Table and Configuring Online and Initial Sync.
- Make sure that each model does have more than 100 tables.

Chapter 4

Extending Star Schema

The data warehouse schema in Oracle Utilities Analytics (OUA) covers a wide range of reporting requirements. You often require adding additional data elements to meet site specific requirements. Oracle Utilities Analytics allows such extensions to the schema through the use of user-defined constructs, such as User Defined Fields (UDFs), User Defined Measures (UDMs), User Defined Degenerate Dimensions (UDDGENs), User Defined Foreign Keys (UDDFKs) and User Defined Dimensions (UDDs). Using these constructs, you can extend the star schemas that are delivered along with the product. This chapter includes:

User Extensible Columns

The predefined facts and dimensions are provided with a set of user extensible columns, which can be used for extending the existing entities. These columns include:

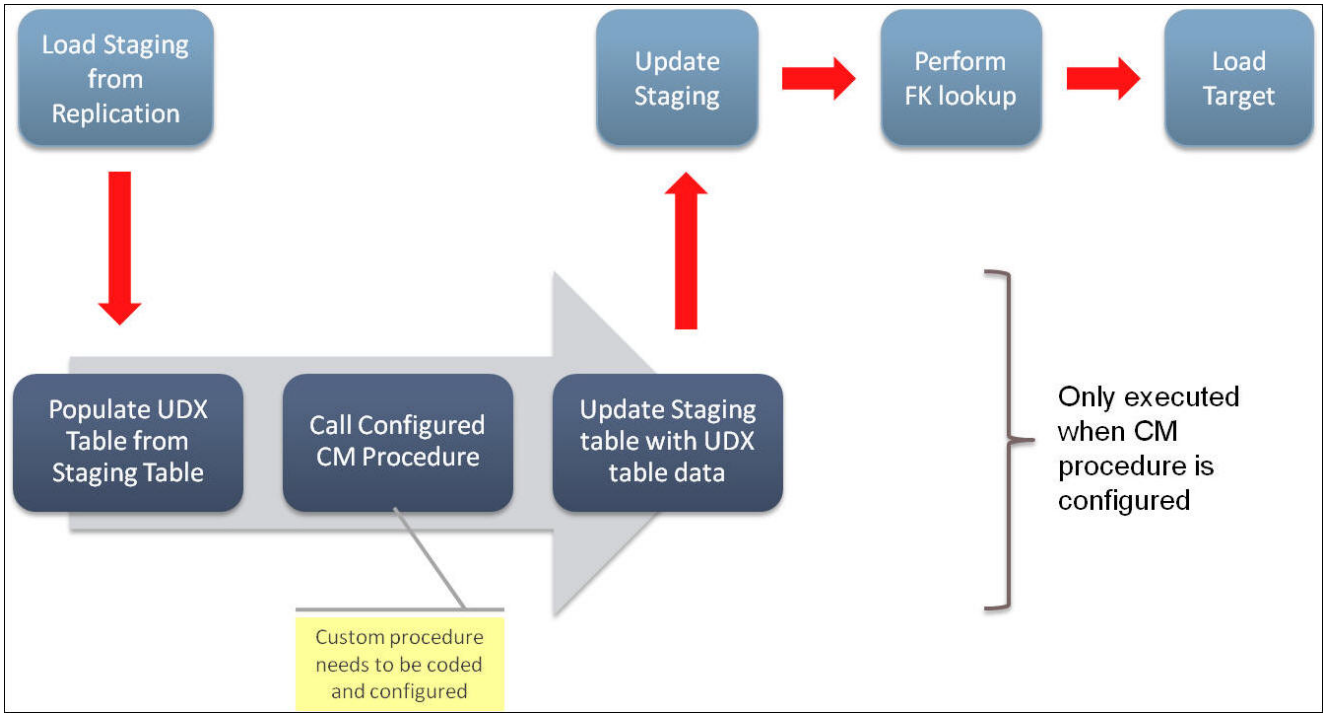
- **User Defined Field**: The User Defined Fields (UDFs) reside on the dimension tables in the star schemas. In general, all the dimensions consist of a minimum of ten UDF columns. These columns can be utilized to store additional information from the source systems.
- **User Defined Measure**: The User Defined Measure (UDM) column is used to support the storage of implementation-specific measures not provided in the out of the box facts.
- **User Defined Degenerate Dimension**: The User Defined Degenerate Dimension (UDDGEN) columns reside directly on the fact table and can be used to store the dimension attributes, which do not fit into a particular dimension, but is required for analytical purposes.
- **User Defined Foreign Key Dimensions**: These are empty foreign keys attributes which are not associated with the out of the box dimensions. This allows you to reuse an existing dimension or to create accustom dimension and build a reference in the fact.
- **User Defined Dimension**: User Defined Dimensions (UDDs) are empty dimensions that are delivered along with the star schemas in Oracle Utilities Analytics.

In addition to utilizing these extensible columns, you can create custom facts and dimensions to achieve their additional analytic requirements.

UDX Processing

In Oracle Utilities Analytics, the process of extending out of the box dimensions and facts is very simple. This relies on a configurable procedure with a predefined signature. All entities have been set up with a functionality that executes the custom procedure if configured.

The diagram below illustrates the processing logic when the user exit procedure is executed. All interfaces process data using staging tables. The first step is loading a staging table using a source view. Once the staging table is loaded, configurations are looked up and if the CM procedure has been configured for the job, then a UDX table is created. The UDX table contains a natural key and all user extensible columns. The table acts as a template and you are expected to update the UDX columns based on the natural key columns and the input parameters.



UDX Processing

After the CM procedure is executed successfully, the data is copied back into the staging table. If the entity being extended is a fact, then user-defined foreign keys are referenced again. After this, the final data is loaded into the target entity.

Your task is reduced to only writing a CM procedure and configuring it.

Populating User-Defined Columns

The Oracle Utilities Analytics product allows you to extend the functionality of the dimensions and facts using user defined columns. Oracle Utilities Analytics allows the creation of PL/SQL procedures to extend the columns. You are utilizing Oracle Data Integrator to define the custom procedure.

The benefits of utilizing Oracle Data Integrator to create the procedure are:

- Schema names do not need to be hardcoded.

- Easier to deploy (Execute in the appropriate context).
- Easy to deploy for multiple instances of the same source product.

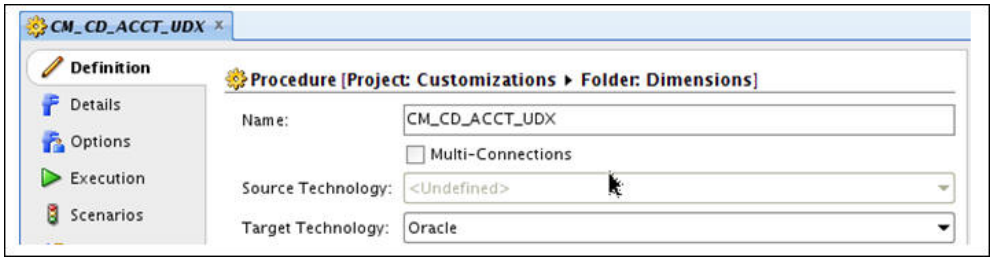
This section covers the following topics:

Creating CM Procedure

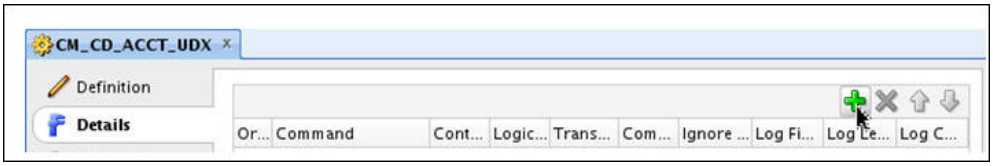
This section walks you through the process of extending the CD_ACCT dimension using a sample data. Again, for purpose of this example, we are assuming that CCB1 is the context defined for the CCB source attached to the Oracle Utilities Analytics product.

To create CM procedure, perform the following steps:

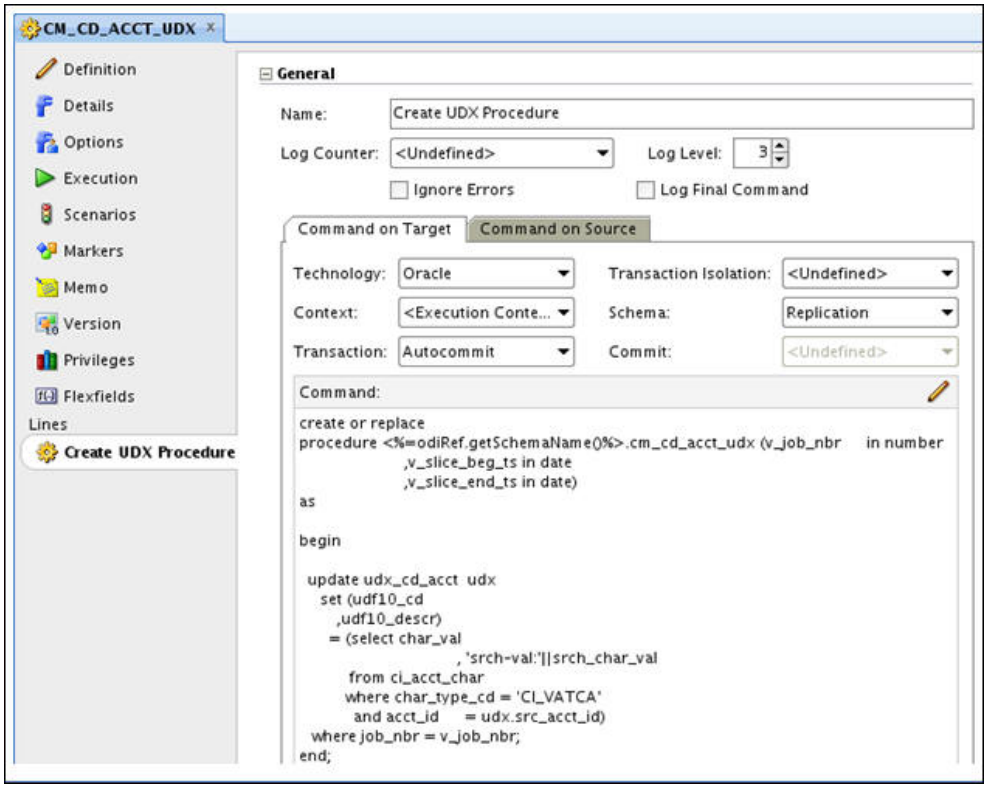
1. Expand the **Dimensions** folder. The folder contains these object types: "Packages", "Interfaces" and "Procedures".
2. Select the **Procedures** node and right-click. Select the **New Procedure** option in the menu that appears.
3. In the editor window, enter the name of UDX in the **Name** field (For example, CM_CD_ACCT_UDX). Select "Oracle" from the **Target Technology** drop-down list.



4. Click on the **Details** tab on the left navigation and click the green + on the top-right corner of the page.



5. Enter the name for the step and select "Replication" from the **Schema** drop-down list as shown below.



6. Paste the code mentioned below into the **Command** section:

```
create or replace
procedure <%=odiRef.getSchemaName()%>.cm_cd_acct_udx
          (v_job_nbr      in number
          ,v_slice_beg_ts in date
          ,v_slice_end_ts in date)
as
begin

  update udx_cd_acct  udx
     set (udf10_cd
         ,udf10_descr)
       = (select char_val
                , 'srch-val:'||srch_char_val
            from ci_acct_char
           where char_type_cd = 'CI_VATCA'
             and acct_id      = udx.src_acct_id)
   where job_nbr = v_job_nbr;
end;
```

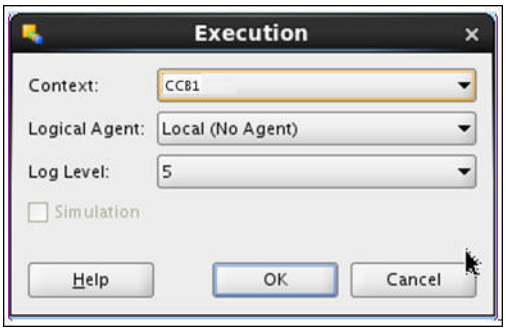
7. Save the changes and click on the green triangle to execute. The **Execution** popup appears. Select "CCB1" from the **Context** drop-down list and click **OK**.

8. Navigate to the main **Operator** tab and expand **Date** > **Today** to view the status of the execution.

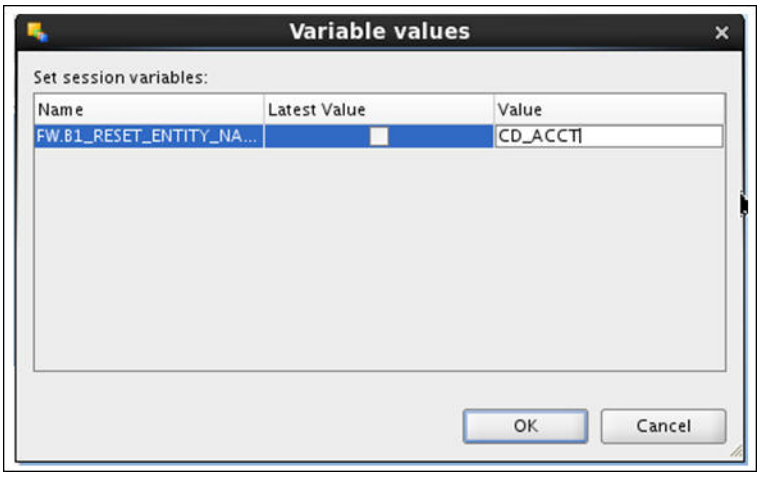
Resetting Dimensions

Since the dimension has already been loaded, you need to reset to empty state before you reload with the customization in place by performing the following steps:

1. Navigate to **Designer** > **Load Plans and Scenarios**> **Framework** > **B1_RESET_ENTITY**.
2. Right-click on the **B1_RESET_ENTITY** entity and select **Execute** option in the menu that appears.
3. Select 'CCB1' as the **Context**.

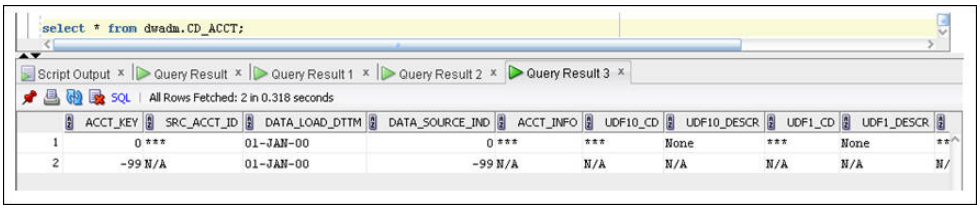


4. In the **Startup Variable**, enter "CD_ACCT" for the entity name.



5. Go back to the **Oracle Utilities Administration** user interface to verify that the entity has been disabled.

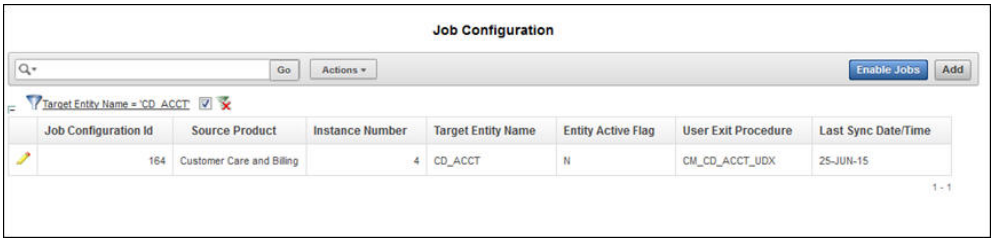
6. Connect to SQL Developer and query the dimension to verify that the all rows except the default 0 and -99 records have been deleted.



Configuring CM Procedure

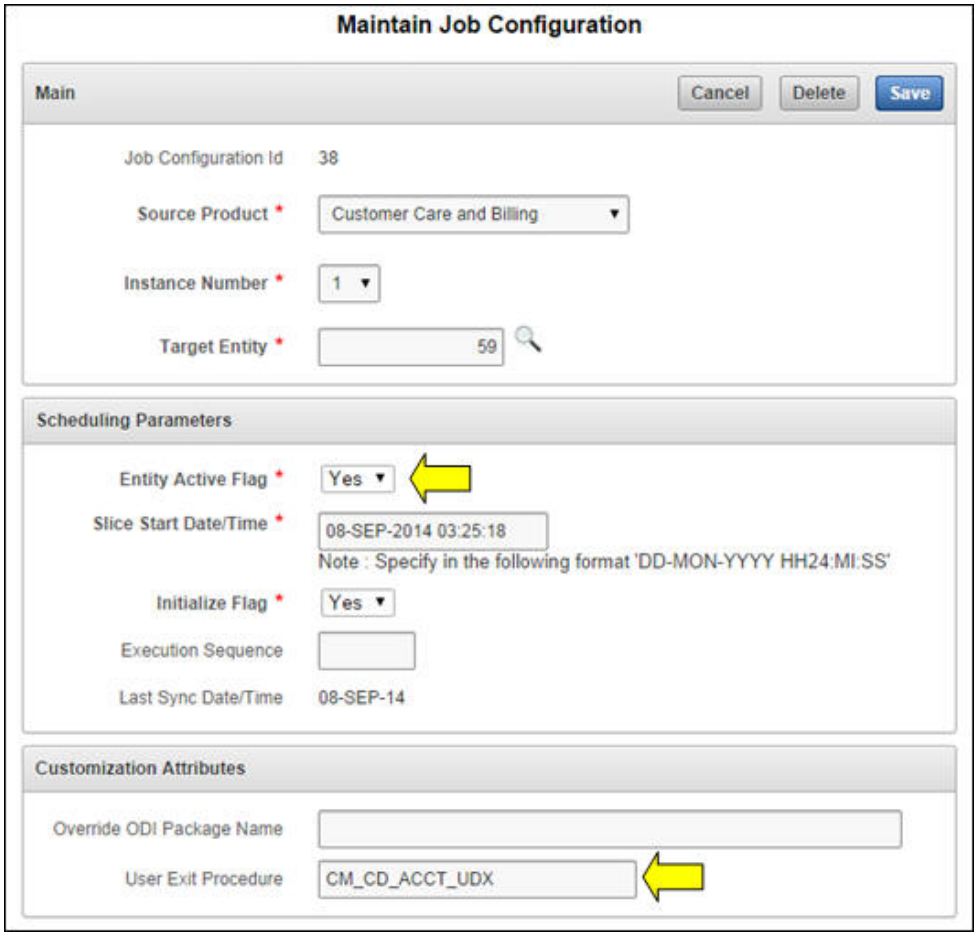
Follow these instructions to configure the user extension procedure for the account dimension:

1. Login to the **Oracle Utilities Analytics Administration** user interface.
2. Click on the **ETL Configuration** tab.
3. Click on the **Job Configuration** menu item.
4. Enter **CD_ACCT** and click **Go** to filter the data.



5. Click on the "Yellow Pencil" icon to go to the **Edit** page.
6. Enter "CM_CD_ACCT_UDX" in the **User Exit Procedure** field.
7. Ensure the **Active Flag** is set to 'Yes'.

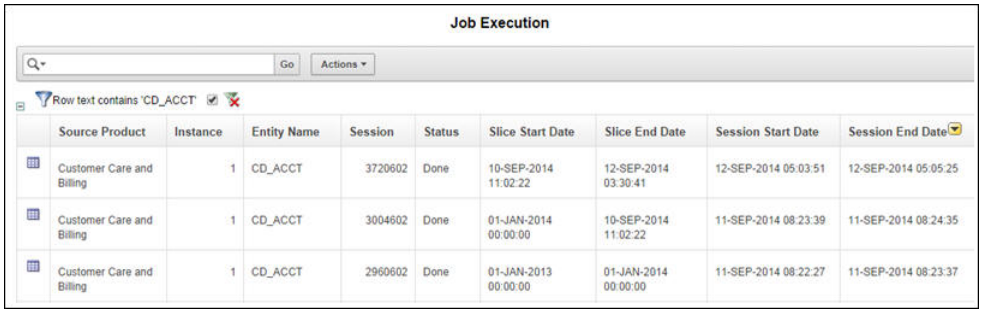
8. Click **Save** to save your configuration changes.



Monitoring Job Execution

Now that the job has been configured for customization and activated, you can monitor the job execution using the **Administration** user interface or using SQL Developer. Below are the steps to monitor the job execution from the **Administration** user interface:

1. Login to the **Oracle Utilities Analytics Administration** user interface.
2. Click on **ETL Job Execution** tab.
3. Enter "CD_ACCT" and click **Go** to filter the data.
   To see the latest execution, you can sort by session end date so that the latest execution appears on the top.



Alternatively, perform these instructions to monitor using SQL developer:

1. Connect to the target database using SQL Developer.

2. Monitor the job executions for the account dimension using the query mentioned below:

```
select *
  from mdadm.b1_jobs_vw
 where entity_name = 'CD_ACCT';
```

Validating Data Load

Follow the steps below to validate that data has been loaded into the customized columns.

1. Check which rows have the **udf10_cd** and **udf10_descr** columns populated using the query mentioned below:

```
select src_acct_id
     , udf10_cd
     , udf10_descr
  from dwadm.CD_ACCT
 where acct_key not in (0,-99)
   and udf10_cd is not null;
```

2. Compare the data in the dimension with the data in the base table **ci_acct_char** using the query mentioned below:

```
select acct_id
     , char_val
     , srch_char_val
  from ccb1rep.ci_acct_char
 where char_type_cd = 'CI_VATCA' ;
```

Populating User-Defined Foreign Keys

This section describes the steps to extend the Out of the box facts with the custom dimension. Create the custom dimension first, and then load the custom dimension. Thereafter, customize the fact load to use the custom dimension and populate the custom dimension key.

Before performing these tasks, perform all the tasks described in the section Custom Dimensions. This section covers the following topics:

- Creating CM View
- Creating CM Procedure
- Configuring CM Procedure

Creating CM View

Follow the instructions below to create an interface that is used to wrap the custom dimension created in the task given above.

1. Open the **Oracle Data Integrator** client and navigate to the **Customization** project.
1. Right-click on the **Interface** and click on the **New Interface** option in the menu that appears.
2. Enter the name of the procedure and enter a brief description of the procedure in the **Description** section.
3. Click on the **Mapping** tab on the bottom of the page to go to the **Edit Mapping** page.
4. On the left navigation, go to **Models** > **Customizations** > **UDX Dimension**.
5. Select and drag the custom dimension into the **Source** section.
6. In the **Properties Inspector**, change the Alias to **UDDX1**.

7. Click on the title bar of the section **Target Datastore** and in the **Properties Inspector** tab, enter the UDDX view name for the name of the target object. The naming convention of the UDDX view is **B1_D_<FACT NAME>_UDDX1_VW** to populate the **UDD1_KEY** of the fact. For example, if the fact name is **CF_ARREARS**, then to populate **UDD1_KEY**, the view name would be **B1_D_ARREARS_UDDX1_VW**. To populate **UDD2_KEY**, the view name would be **B1_D_ARREARS_UDDX2_VW**.
8. Keeping the **CTRL** key clicked, select the columns, surrogate key, natural key and effective dated columns.
9. Right-click on any selected column and in the contextual menu that appears click on **Add Column to Target Table**. This brings all the columns in the target of the interface.
10. Click on the **Flow** tab at the bottom. This takes you to the **Flow Editor** page where the KM selection can be done.
11. In the **Properties Inspector**, select the "IKM BI View Generation" **Global Integration Knowledge Modules (IKM)**. Leave the properties as the default and click **Save**.
12. Execute the interface and go to the **Operator** to view the status.
    The job should execute successfully and the view should be created.
13. Verify the view data by executing the following query in SQL Developer. The data from the view and the custom dimension should match.

```
select *
  from {Target}.uddx view
```

Creating CM Procedure

Follow the instructions below to create the CM procedure for a fact:

1. Login to Oracle Data Integrator client.
2. Navigate to the **Designer** tab.
3. Expand the **Project** "Customizations".
4. Create the **Facts** folder and expand the folder.
5. Right-click on the **Procedures** and select the "New Procedure" option in the menu that appears.
6. Enter the name of the procedure in the **Name** field and in the **Description** field, enter **a** brief description.
7. Click on the **Details** tab on the left hand side navigation section. This opens the **Editor** window. Click on the green + to add a new step.
8. Enter the name for the step and select "Replication" from the **Schema** drop-down list.
9. In the **Command** section, enter the logic to update the UDX table of the fact. The UDD(n)_DSI should be updated.
10. Click **Save** to save the changes and click on the green triangle to execute. The **Execution** popup appears. Select the context from the **Context** drop-down list and click **OK**.
11. Navigate to the main **Operator** tab and expand **Date** > **Today** to view the status of the execution.

Configuring CM Procedure

Follow the instructions below to configure the user extension procedure for the fact.

1. Open the Oracle Utilities Analytics Administration user interface in a browser and login.
2. Click on the **ETL Configuration** tab.
3. Click on the **Job Configuration** menu item.
4. Enter the procedure name in the **User Exit Procedure** field and click **Go** to filter the data.
5. Click on the **Pencil** icon to bring up the edit page.
6. Click **Save.**
7. Enable the job by setting the **Entity Active Flag** to **Yes**.
8. Monitor the job execution and verify the data is in the final fact.

Star Schema

The star schema is perhaps the simplest data warehouse schema. It is called a star schema as the entity-relationship diagram of this schema resembles a star with points radiating from a central table. The center of the star consists of a large fact table. The end points of the star are the dimension tables.

A star query is a join between a fact table and a number of dimension tables. Each dimension table is joined to the fact table using a primary key to foreign key join. However, the dimension tables are not joined to each other. The optimizer recognizes star queries and generates efficient execution plans. It is not mandatory to have any foreign keys on the fact table for star transformation to take effect.

A typical fact table contains keys and measures. A star join is a primary key to foreign key join of the dimension tables to a fact table. The main advantages of a star schema are as follows:

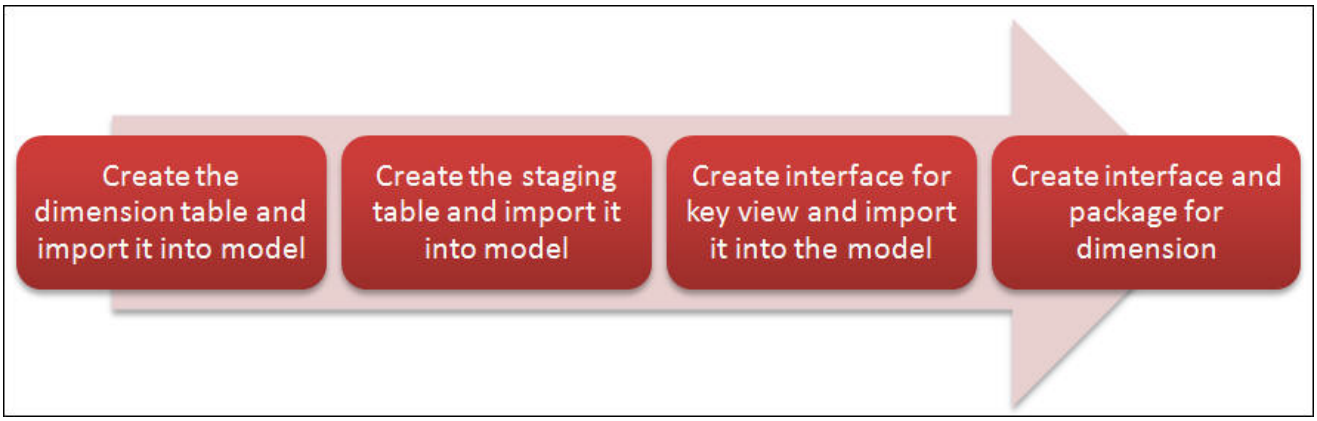
- Provides a direct and intuitive mapping between the business entities analyzed by the end users and the schema design.
- Provides highly-optimized performance for the typical star queries.
- Widely supported by a large number of business intelligence tools, which may anticipate or even require that the data warehouse schema contain dimension tables.

The star schemas are used for both simple data marts as well as very large data warehouses. Once the model has been designed, Oracle Data Integrator can used to create the interfaces and package to load the data into the star schema.

> **Note**: For the details regarding data modeling, refer to the **Chapter 19: Schema Modeling Techniques** of the *Oracle® Database Data Warehousing Guide 11g Release 2* guide.

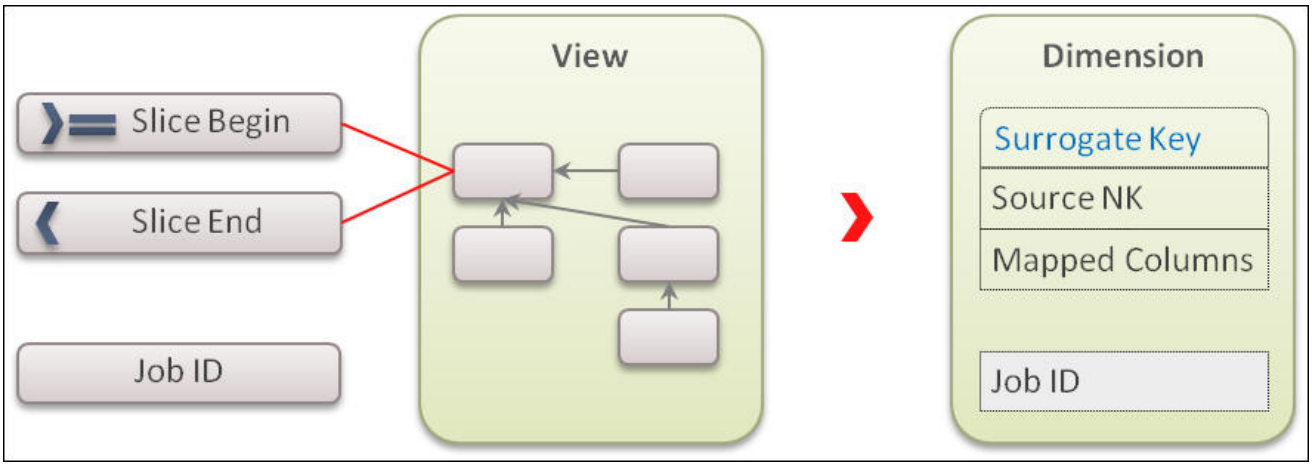
Custom Dimensions

The diagram below illustrates the steps required to create a custom dimension.



Create a table and a sequence in the database. The dimension table has a surrogate primary key and a unique key, which includes the data source indicator and a column from the source.

The below diagram shows the pattern to be used while developing the Oracle Data Integrator components for a custom dimension. This is similar to the out of the box pattern with the user extension component excluded.



This section covers the following:

- Creating Staging Tables in Model
- Creating Interface for Key View
- Importing View into the Model
- Creating Interface
- Creating Package
- Configuring Entity
- Configuring Job
- Monitoring Job Execution

Creating Staging Tables in Model

The staging table is created by the scheduling process prior to executing the interface. This is done to optimize parallel execution of multiple slices of the same entity load. To do this, the definition of the staging table structure needs to be present in a model under the Staging folder. The staging table structure is similar to the target table structure with the addition of a few columns. The staging table should have an **IND_UPDATE** column in addition to the columns used in the mapping. Perform the following steps for creating a staging table in a model:

1. Navigate to **Models** > **Customizations**.
2. Right-click on **Customizations** and select **New Model** option in the menu that appears.
3. In the **Name** field, enter "Staging".
4. Specify the value in the **Code** field.
5. Select the **Technology** as "Oracle".
6. Select the **Logical Schema** as "Staging".
7. Save the model.
8. Right-click on the model **Staging** and select **New Datastore** option in the menu that appears.
9. In the **Definition** tab, enter the name of the staging table. The naming convention of the staging table is "STG_" prefixed to entity name.
10. Specify the same table name in the **Resource Name** field.
11. Navigate to **Columns** tab.
12. Click on "+" button on right hand corner and add the columns in the datastore.
    The columns present in the dimension tables have to be present in the staging table.
    In addition to the above column, "IND_UPDATE" column has to be added.
    The data type of column "IND_UPDATE" should be "CHAR(1)".
13. Save the Datastore.
14. Navigate to the **Flexfields** tab.
15. Uncheck the checkbox in the **Default** column.
16. Specify the value "STG" in the **Value** column for the record "B1 Object Type".
17. Specify the entity name (dimension name) in the **Value** column for the record "B1 Target Entity Name".
18. Save the datastore.
19. Close the **Datastore and Model** tab.

Creating Interface for Key View

The key view is created for the fact so that the incremental data for the fact can be filtered based on the key view. The driving tables from the replication schema have to be included and column as part of natural key of the fact have to be created in the **Key** view. In addition to the natural key, include the **JRN_SLICING_TS** column from the source tables in the **Key** view. This view has to be generated in the Replication schema. Follow the instructions mentioned below to create an interface for generating the view for the key columns.

1. Navigate to the **Customizations** project.
2. Create a new folder named **Replication**.
3. Expand the **Replication** folder.
4. Right Click on the **Interface** and select "New Interface" option in the menu that appears.

5. In the **Interface Editor** page, enter the new name in the **Interface Name** field.
   Select the Context in the **Optimization Context** drop-down list and select "Oracle: Replication" in the **Selecting Staging Area** drop-down list.
   Do not check the **Staging Area Different From Target** checkbox.
6. Navigate to **Model** > **Customization** > **Replication**.
7. Drag the tables from the model.
8. Click on the **Target Datastore** to edit the name of the **Target View**.
9. Add the Column Name and datatype under the Target View Name (Natural key of the fact).
10. Add the **JRN_SLICING_TS** column with the Date as datatype after the natural key columns are added.
11. Map the target view columns from the dragged table.
    In case there are multiple driving tables, then add the new dataset with the operator as "UNION".
    The number of data sets should be equal to the number of driving tables selected.
    A primary driving table should be identified and should always be the first dataset.

    For subsequent data sets, include the filter JRN_UPDATE_DTTM > to_date(#B1_EXTRACT_START_DTTM,'YYYYMMDD').
12. Map the column in all the datasets for all the driving tables to target view.
13. Navigate to **Flow** tab and select the **Integration Knowledge Modules** (**IKM**) as "IKM BI View Generation".
14. Click **Save**.
15. Generate scenario for the interface and execute the scenario in the context.

Importing View into the Model

Perform the following steps:

1. Navigate to **Designer** > **Models** > **Customizations**.
2. Open the **Replication** Model.
3. Click on the **Reverse Engineer** tab on the left side menu. Select the context from the **Context** drop-down list and in the **Mask** field, enter the Key View Name.
4. Save the model, and Click the **Reverse Engineer** button.

Creating Interface

Follow the instructions below to create an interface for loading the data from the source view into the new dimension

1. Login to Oracle Data Integrator client.
2. Navigate to the **Dimension** folder under the **Customizations** project. Create the **Dimension** folder if the folder does not exist.
3. Right-click on the **Interface** and select **New Interface** option in the menu that appears.
4. In the **Interface Editor**, enter the name of the interface name in the **Name** field.
   Select a context from the **Optimization Context** drop-down list.
   Select "Oracle. Target" from the **Selecting Staging Area** drop-down list.
5. Navigate to **Models** > **Customizations** > **Target**. Expand the model and drag the **Dimension** table into the target area of the interface editor.

6. Navigate to **Models** > **Customizations** > **Replication**. Expand the model and drag the key view into the source area of the Interface Editor.
7. Oracle Data Integrator will ask if you want to do automatic mapping. Click **Yes** and all columns with the same name will be automatically mapped.
8. Select the **JRN_SLICING_TS** column and drag it outside the table. This creates a new filter. Copy the condition below in the **Property Inspector**, for the filter.

```
 JRN_SLICING_TS >= to_date(#B1_SLICE_BEG_TS,'YYYYMMDDHH24MISS')
and JRN_SLICING_TS  < to_date(#B1_SLICE_END_TS,'YYYYMMDDHH24MISS')
```

9. Drag and drop other tables from the model in the source area and join based on the logic.
10. Map the target table (dimension table) columns with the source tables.
11. Select the column **JOB_NBR** and in the **Property Inspector**, enter "#B1_JOB_ID". Ensure that **UD8** checkbox is selected.
12. Select the column **UPDATE_DTTM** and in the **Property Inspector**, enter the latest of all the effective start date in case of history maintained table and update date for the override table.
13. Navigate to the **Flow** tab and select the **Integration Knowledge Modules (IKM)** as "IKM BI Dimension Load (SCD-II)" if the target table is type 2 dimension, else select the **Integration Knowledge Modules (IKM)** as "IKM BI Dimension Load (SCD-I)" if the target table is type 1 dimension.
14. Click **Save**.

Creating Package

Follow the instructions below to create a package for the new dimension.

1. Navigate to **Designer** > **Customizations** > **Dimensions** > **Packages**.
2. Right-click on the **New Package.**
3. In the **Package Editor** page, enter the name of the package.
4. Click on the **Diagram** tab at the bottom of the editor.
   From the global objects section, drag the variables **B1_JOB_ID, B1_SLICE_BEG_TS** and **B1_SLICE_END_TS** into the editor.
   Change the variables **B1_JOB_ID**, **B1_SLICE_BEG_TS** and **B1_SLICE_END_TS** to declare variable.
   If the Dimension is type 2 dimension, then drag the variable **B1_HIGH_DATE** into the editor. Change the variable **B1_HIGH_DATE** to refresh variable.
5. Drag and drop the interface into the editor and connect them all together.
6. Click the **Save** to save the changes and close the package editor window.
7. Navigate to the packages folder and expand it. You will see the new package displayed here.
8. Right-click on the **Generate Scenario**.
9. A popup dialog appears asking for the scenario name. Click **OK**
10. Another popup dialog appears asking you to select the startup variables. Uncheck **B1_HIGH_DATE** if present as this is a refresh variable and click **OK**.
11. Expand the package and you will notice scenarios and under scenarios you can see the scenario object that was generated.

Configuring Entity

Follow the instructions below to configure a new entity for the custom dimension.

1. Open the Oracle Utilities Analytics Administration User Interface in a browser and login.
2. Click on the **ETL Configuration** tab.
3. Click on the **Target Entity** menu item.
4. Click on the **Add** button on the right.
5. A page appears. Enter the details of the entity here.
6. Enter the values appropriate for the dimension name.

Configuring Job

Follow the instructions below to configure a job for the custom dimension.

1. Open the Oracle Utilities Analytics Administration User Interface in a browser and login.
2. Click on the **ETL Configuration** tab.
3. Click on the **Job Configuration** menu item.
4. Click on the **Add** button on the right.
5. A page appears. Enter the details of the job here.
6. Select product from the **Source Product** drop-down list and select instance number.
7. Click on the **Search** icon for the **Target Entity** field. In the popup search window, enter dimension name and click **Go**.
8. Click on the ID value and it will take you back to the **Job Addition** page with the target entity ID populated.
   Set the Slice Start Date/Time as 01-Jan-2000 or the extract date to which the source instance is configured and click the **Add** button to create the job configuration entry.
9. Enable the job while saving the new entry.

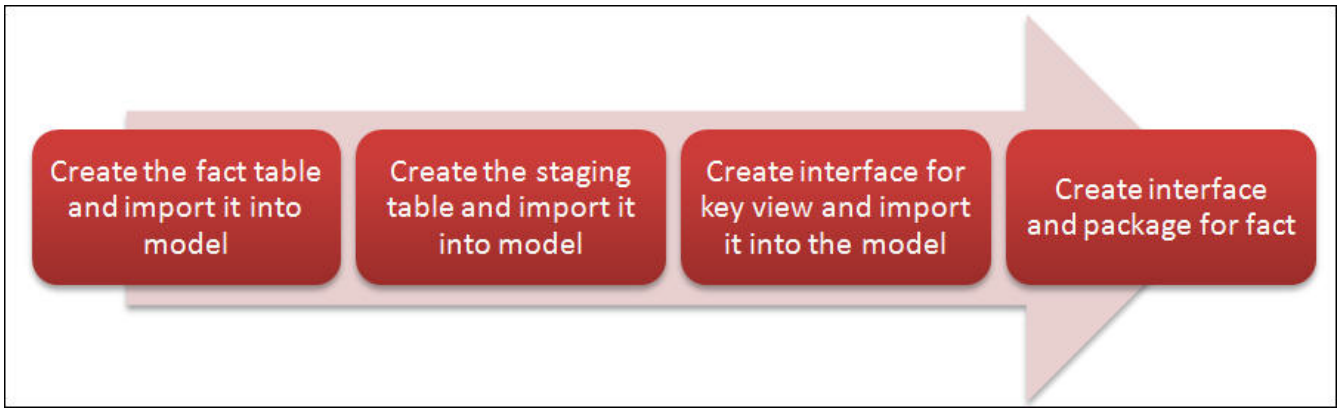
Monitoring Job Execution

Now that the job has been configured for customization and activated, you can monitor the job execution using the Administration User Interface or using SQL developer. Below are the steps to monitor the job execution from the Administration User Interface.

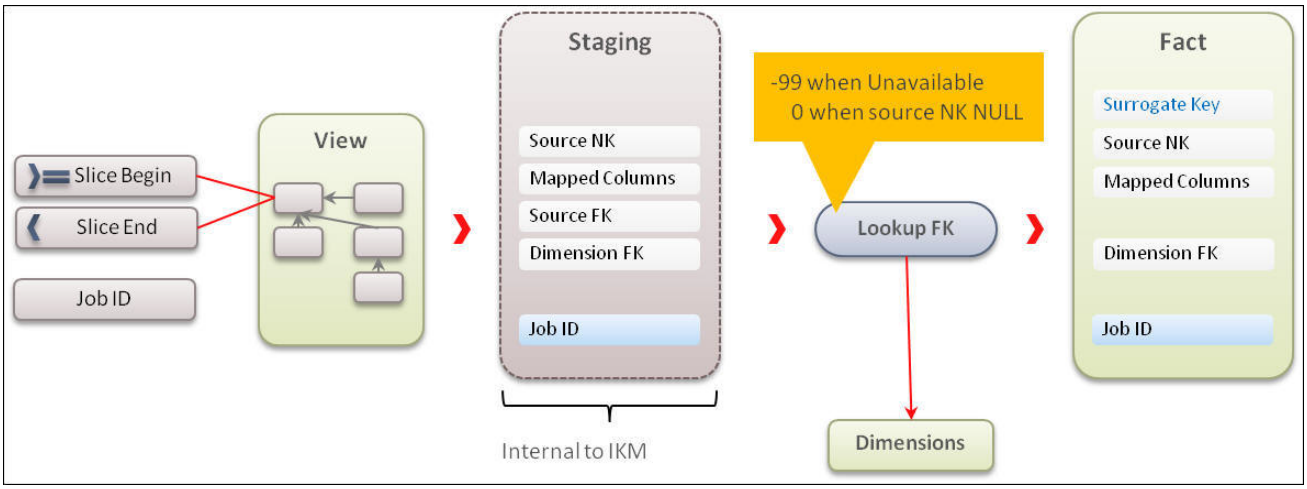
1. Open the Oracle Utilities Analytics Administration User Interface in a browser and login.
2. Click on the **ETL Job Execution** tab.
3. Enter fact name and click **Go** to filter the data.
   To see the latest execution you can sort by session end date so that the latest execution appears on top.

Custom Facts

The diagram below illustrates the steps required to create a custom fact.



The below diagram shows the pattern to be used while developing the Oracle Data Integrator components for a custom dimension. This is similar to the out of the box pattern with the user extension component excluded.



This section covers the following:

- Importing Fact Table into the Model
- Creating Interface for Key View
- Importing View into the Model
- Creating Aggregate Table in Model
- Creating Staging Table in Model
- Creating Interface to Aggregate Data
- Creating Interface to Load Fact
- Creating Package
- Configuring Entity
- Specifying Dependency for Fact
- Configuring Job
- Monitoring Job Execution

Importing Fact Table into the Model

To import the fact table into model, perform the following steps:

1. Login to Oracle Data Integrator client.
2. Navigate to the **Custom Model**.
3. Create a model for facts.
4. Select the **Technology** as "Oracle" and **Logical Schema** as "Target".
5. Click on the **Reverse Engineer** tab on the left side menu. Select context from the **Context** drop-down list and in the **Mask** field, enter the fact name.
6. Save the model and click on the **Reverse Engineer** button.

Creating Interface for Key View

The key view is created for the fact so that the incremental data for the fact can be filtered based on the key view. The driving tables from the replication schema have to be included and column as part of natural key of the fact have to be created in Key view. In addition to the natural key include the JRN_SLICING_TS column from the source tables in the Key view. This view has to be generated in the Replication schema. Follow the instructions below to create an interface for generating the view for the key columns.

1. Navigate to the **Customizations** project.
2. Create a new folder named **Replication**.
3. Expand the **Replication** folder.
4. Right-click on the **Interface** and select **New Interface** option in the popup menu.
5. In the **Interface Editor** page, enter Interface Name in the **Name** field.
   Select the **Context** as the "Optimization Context" and select "Oracle: Replication" in the **Selection Staging Area** drop-down list.
   Do not check the "Staging Area Different From Target**"**.
6. Navigate to **Model** > **Customization** > **Replication**.
7. Drag the tables from the model.
8. Click on the **Target Datastore** to edit the name of the target view.
9. Add the column name and datatype under the target view name (Natural key of the fact).
10. Add the **JRN_SLICING_TS** column with DATE as the datatype after the natural key columns are added.
11. Map the target view columns from the dragged table.
12. In case there are multiple driving tables, then add new dataset with operator as "UNION". The number of data sets should be equal to the number of driving tables selected. A primary driving table should be identified and should always be the first dataset. For subsequent data sets, include the filter JRN_UPDATE_DTTM > to_date(#B1_EXTRACT_START_DTTM,'YYYYMMDD')
13. Map the column in all the datasets for all the driving tables to the Target View.
14. Navigate to **Flow** tab and select the **Integration Knowledge Modules (IKM)** as "IKM BI View Generation".
15. Click **Save**.
16. Generate scenario for the interface and execute the scenario in the context.

Importing View into the Model

To import view into model, perform the following steps:

1. Navigate to **Designer** > **Models** > **Customizations**.
2. Open the **Replication** Model.
3. Click on the **Reverse Engineer** tab on the left side menu. Select the context from the **Context** drop-down list and in the **Mask** field, enter the Key View Name.
4. Save the model and click on the **Reverse Engineer** button.

Creating Aggregate Table in Model

The aggregate table is created by the scheduling process prior to executing the interface. This is done to optimize parallel execution of multiple slices of the same entity load. To do this the definition of the aggregate table structure needs to be present in a model under the Staging folder. The aggregate table structure is similar to the target table structure with the addition of a few columns. The aggregate table should have a JOB_NBR column in addition to the columns used in the mapping. The aggregate table should have "UPDATE_DTTM" column in addition. The "UPDATE_DTTM" column stores the greatest effective start date of the record.

1. Navigate to **Models** > **Customizations** > **Staging.**
2. Create the **Model Staging** if the staging model does not exist.
3. Right-click the model **Staging** and select "New Datastore".
4. In the **Definition** tab, enter the name of the aggregate table name.
5. Specify the same table name in the **Resource Name** field.
6. Navigate to **Columns** tab.
7. Click on "+" button on right hand corner and add the columns in the datastore.
   The columns present in the fact tables have to be present in the aggregate table.
   The dimension key columns have to be renamed to natural key of the dimension.
   For example, if the fact refers to Account dimension, having foreign key column as **ACCT_KEY** in fact.
   In the aggregate table, this column has to be replaced with **ACCT_FK** with the same datatype as that of the natural key of the corresponding dimension.
8. In addition to the above column, **JOB_NBR** column has to be added. The data type of column **JOB_NBR** should be "NUMBER(19)".
9. In addition to the above column **UPDATE_DTTM** column has to be added. The data type of column **UPDATE_DTTM** should be "DATE".
10. Save the Datastore.
11. Navigate to **Flexfields** tab.
12. Uncheck the checkbox in the **Default** column.
13. Specify the value "TMP" in the **Value** column for the "B1 Object Type" record.
14. Specify the entity name (Dimension name) **Value** column for the record "B1 Target Entity Name".
15. Save the datastore.
16. Close the **Datastore and Model** tab.

Creating Staging Table in Model

The staging table is created by the scheduling process prior to executing the interface. This is done to optimize parallel execution of multiple slices of the same entity load. To do this the definition of the staging table structure needs to be present in a model under the Staging folder. The staging table structure is similar to the target table structure with the addition of a few columns. The staging table should have an IND_UPDATE column in addition to the columns used in the mapping.

1. Navigate to **Models** > **Customizations**.
2. Right-click and create a new Model.
3. In the **Name** field, enter "Staging".
4. Specify the value in the **Code** field.
5. Select the **Technology** as "Oracle".
6. Select the **Logical Schema** as "Staging".
7. Save the Model.
8. Right- click the model **Staging** and select "New Datastore".
9. In the **Definition** tab, enter the name of the staging table. The naming convention of the staging table is "STG_" prefixed to the entity name.
10. Specify the same table name **Resource Name** field.
11. Navigate to **Columns** tab.
12. Click on "+" button on right hand corner and add the columns in the datastore.
    The columns present in the fact tables have to be present in the staging table.
    In addition. natural key of the dimension referred in the fact have to be added to staging table.
    For example, if the fact refers to Account dimension, having foreign key column as **ACCT_KEY** in fact.
    In the staging table, there are two columns, one with name **ACCT_FK** with the same datatype as that of the natural key of the corresponding dimension and second **ACCT_KEY** as the same that in the fact.
13. .In addition to the above column, **IND_UPDATE** column has to be added.
    The data type of column **IND_UPDATE** should be "CHAR(1)".
14. In addition to the above column, **UPDATE_DTTM** column has to be added.
    The data type of column **UPDATE_DTTM** should be "DATE".
15. In addition to the above column, **JOB_NBR** column has to be added.
    The data type of column **JOB_NBR** should be "NUMBER(19)".
16. Save the Datastore.
17. Navigate to **Flexfields** tab.
18. Uncheck the checkbox in the **Default** column.
19. Specify the value "STG" in the **Value** column for the record "B1 Object Type".
20. Specify the entity name (Dimension name) **Value** column for the record "B1 Target Entity Name".
21. Save the datastore.
22. Close the **Datastore and Model** tab.

Creating Interface to Aggregate Data

Create a aggregate table and load the data using the key view and all the tables from which the facts are loaded. Include the filter condition on the key view for the incremental data on JRN_SLICING_TS column.

1. Login to Oracle Data Integrator client.
2. Navigate to the **Facts** folder under the **Customizations** project. Create the **Facts** folder if the folder does not exist.
3. Right-click on the **Interface** and select **New Interface** option in the popup menu.
4. In the interface editor, enter Interface name in **Name** field. Select context from the **Optimization Context** drop-down list. Select "Oracle. Target" in the drop-down list for Selecting Staging Area.
5. Navigate to **Models** > **Customizations** > **Staging**. Expand the model and drag the aggregate table into the target area of the interface editor.
6. Navigate to **Models** > **Customizations** > **Replication**. Expand the model and drag the **Key** view into the source area of the interface editor.
7. Oracle Data Integrator will ask if you want to do automatic mapping. Click "Yes" and all columns with the same name will be automatically mapped.
8. Select the **JRN_SLICING_TS** column and drag it outside the table. This will create a new filter. Copy the condition below in the **Property Inspector,** for the filter.

```
   JRN_SLICING_TS >= to_date(#B1_SLICE_BEG_TS,'YYYYMMDDHH24MISS')
and JRN_SLICING_TS  < to_date(#B1_SLICE_END_TS,'YYYYMMDDHH24MISS')
```

9. Drag and drop other tables from model in the source area and join based on the logic
10. Map the **Target** table (Aggregate table) columns with that of the source tables
11. Select the column **JOB_NBR** and in the **Property Inspector**, enter "#B1_JOB_ID". Ensure that **UD8** checkbox is selected.
12. Select the column **UPDATE_DTTM** and in the **Property Inspector**, enter greatest of all the effective start date in case of history maintained table and update date for the override table.
13. Navigate to **Flow** tab and select the **Integration Knowledge Modules (IKM)** as "IKM BI Append Parallel".
14. Click **Save**.

Creating Interface to Load Fact

The fact is loaded from aggregate table by looking up the dimension tables. This interface performs the tasks matching the source natural keys to the appropriate dimension surrogate keys. To create the interface, perform the following steps:

1. Login to Oracle Data Integrator client.
2. Navigate to the **Facts** folder under the **Customizations** project. Create the **Facts** folder if the folder does not exist.
3. Right-click on the **Interface** and select **New Interface** option in the popup menu.
4. In the **Interface Editor** page, enter the name of the interface in the **Name** field. Select context from the **Optimization Context** drop-down list. Select "Oracle. Target" in the drop-down list for **Selecting Staging Area.**
5. Navigate to **Models** > **Customizations** > **Target**. Expand the model and drag the fact table into the target area of the interface editor.

6. Navigate to **Models** > **Customizations** > **Staging**. Expand the model and drag the aggregate table into the source area of the interface editor, provide the alias as 'SRC'
7. Oracle Data Integrator will ask if you want to do automatic mapping. Click **Yes** and all columns with the same name will be automatically mapped.
8. Select the **JOB_NBR** column and drag it outside the table. This creates a new filter. Copy the condition below in the **Property Inspector**, for the filter.

   ```
   SRC.JOB_NBR = #B1_JOB_ID
   ```

9. All joins from the source table to the dimension should be an outer join including all rows from SRC and any rows that are available from dimension.
10. Drag and drop the dimension and join the aggregate table with the dimension tables. The join condition with type 2 dimension is:

    ```
        SRC.DIM_FK = DIM1.SRC_DIM_NK
    and SRC.DATA_SOURCE_IND = DIM1.DATA_SOURCE_IND
    and SRC.UPDATE_DTTM >= DIM1.EFF_START_DTTM
    and SRC.UPDATE_DTTM < DIM1.EFF_END_DTTM
    ```

    The join condition with type 1 dimension is:

    ```
        SRC.DIM_FK = DIM2.SRC_DIM_NK
    and SRC.DATA_SOURCE_IND = DIM2.DATA_SOURCE_IND
    ```

11. Map the **Target** table (**Aggregate** table) columns with that of the source tables.
12. Set the **UD10** flag to true for all dimension KEY columns of the target fact.
13. Select the **Dimension** key and in the **Property Inspector**, enter the transformation as mentioned below by replacing the actual dimension name and the dimension column names:

    ```
    CASE WHEN SRC.DIM_FK IS NULL THEN #B1_NULL_KEY
         WHEN DIM.DIM_KEY IS NULL THEN #B1_MISSING_KEY
         ELSE DIM.DIM_KEY
    END
    ```

14. Select the column **JOB_NBR** and in the **Property Inspector**, enter "#B1_JOB_ID". Ensure that **UD8** checkbox is selected.
15. Navigate to the **Flow** tab and select the **Integration Knowledge Modules (IKM)** as "IKM BI Fact (Merge)".
16. Click **Save**.

Creating Package

Follow the instructions below to create a package for the new fact.

1. Navigate to **Designer** > **Customizations** > **Facts** > **Packages**.
2. Right-click on the **Package** and select **New Package** option in the popup menu.
3. In the **Package Editor** page, enter the name of the package.
4. Click on the **Diagram** tab at the bottom of the editor.
   From the **Global Objects** section, drag the variables **B1_JOB_ID**, **B1_SLICE_BEG_TS, B1_SLICE_END_TS**, **B1_MISSING_KEY** and **B1_NULL_KEY** into the editor.
   Change the variables **B1_JOB_ID, B1_SLICE_BEG_TS, B1_SLICE_END_TS** to declare variable.
   Change the variables **B1_MISSING_KEY** and **B1_NULL_KEY** to refresh variables.
   Now, drag and drop the interface to load the aggregate table and then the interface to load the fact table into the editor and connect them all together in a sequence.

5. Click the **Save** to save the changes and close the **Package Editor** window.
6. Navigate to the **Packages** folder and expand it. You will see the new package displayed here.
7. Right-click on the **Generate Scenario**.
8. A popup dialog appears asking for the scenario name. Click **OK**.
9. Another popup dialog appears asking you to select the startup variables. Uncheck **B1_MISSING_KEY** and **B1_NULL_KEY** as these are refresh variables and click **OK**.
10. Expand the package and you will notice scenarios and under scenarios, you can see the scenario object that was generated.

Configuring Entity

Follow the instructions below to configure a new entity for the custom fact.

1. Open the Oracle Utilities Analytics Administration User Interface in a browser and login.
2. Click on the **ETL Configuration** tab.
3. Click on the **Target Entity** menu item.
4. Click on the **Add** button on the right.
5. Enter the details of the entity in the window.
6. Enter the values appropriate for the fact name.

Specifying Dependency for Fact

Follow the instructions below to specify the dependency for the tact:

1. Add a new step to the user procedure **CM_<PROD_FLG>_CREATE_MEATADATA**.
2. Use the sample mentioned below to create an entry in the entry in the **B1_OBJECT_MAP** for specifying the type 2 dimension that the fact depends on.

3. As an example, the sample merge statement below specifies the base object name. The given sample statement mentions the source object name as the dimension name. The target object name is the fact name. You can specify the source product flag as appropriate. The object type flag is "DMDP" to identify the materialized dependency on the base objects.

```
merge
 into b1_object_map  tgt
using (select 'CCB'                                prod_flg
            , 'DIM2'                            source_object_name
            , 'CM_FACT'                            target_object_name
            , 1                                       seq
            , 'DMDP'                                   object_type_flg
         from dual )  tgt_val
   on (    tgt.prod_flg           = tgt_val.prod_flg
      and tgt.source_object_name = tgt_val.source_object_name
       and tgt.target_object_name = tgt_val.target_object_name
       and tgt.seq                = tgt_val.seq)
when not matched
then insert
     (
       tgt.object_map_id
     , tgt.prod_flg
     , tgt.source_object_name
     , tgt.target_object_name
     , tgt.seq
     , tgt.object_type_flg
     , tgt.char_entity_flg
     , tgt.upd_dttm
     , tgt.upd_user
     , tgt.owner_flg
     )
     values
     (
       b1_object_map_seq.nextval
     , tgt_val.prod_flg
     , tgt_val.source_object_name
     , tgt_val.target_object_name
     , tgt_val.seq
     , tgt_val.object_type_flg
     , null
     , sysdate
     , sys_context('userenv', 'os_user')
     ,'B1');
```

4. Execute the user procedure to create the entries.

Configuring Job

Follow the instructions below to configure a job for the custom fact.

1. Open the Oracle Utilities Analytics Administration User Interface in a browser and login.
2. Click on the **ETL Configuration** tab.
3. Click on the **Job Configuration** menu item.
4. Click on the **Add** button on the right.
5. Add the details of the job in the window.
6. Select product from the **Source Product** drop-down list and select the instance number.
7. Click on the **Search** icon for the **Target Entity** field. In the popup search window, enter fact name and click **Go**.

8. Click on the ID Value and it will take you back to the **Job Addition** page with the target entity ID populated.
   Set the Slice Start Date/Time as 01-Jan-2000 or the extract date to which the source instance is configured and click the **Add** button to create the Job Configuration entry.
9. Enable the job while saving the new entry.

Monitoring Job Execution

Now that the job has been configured for customization and activated, you can monitor the job execution using the Administration User Interface or using SQL developer. Below are the steps to monitor the job execution from the Administration User Interface.

1. Open the Oracle Utilities Analytics Administration User Interface in a browser and login.
2. Click on the **ETL Job Execution** tab.
3. Enter fact name and click **Go** to filter the data.
   To see the latest execution you can sort by session end date so that the latest execution appears on top.

Custom Materialized View

The materialized views are used to store the aggregated data so that the analytics could fetch the data from the materialized views. To create a materialized view on any custom facts, follow the steps mentioned below:

- Creating Interface
- Creating Package
- Configuring Entity
- Specifying the Dependency for the Custom Materialized View
- Configuring Job
- Monitoring Job Execution

Creating Interface

Perform the following steps to create an interface for materialized view:

1. Login to Oracle Data Integrator client.
2. Navigate to the **Materialized Views** folder under the **Customizations** project. Create the **Materialized Views** folder if the folder does not exist.
3. Right-click on the interface and select **New Interface** in the popup menu.
4. In the interface editor, enter Interface name in the **Name** field. Select context from the **Optimization Context** drop-down list. Select "Oracle. Target" in the drop-down list for selecting staging area.
5. Navigate to **Models** > **Target**. Expand the model and drag the source table (dimension or facts) into the target area of the interface editor.
6. Setup the appropriate join conditions between the source tables.
7. Click on the **Target Datastore** to edit the name and specify the materialized view name.
8. Map the **Target Table (Materialized View)** columns with that of the source tables.
9. Navigate to the **Flow** tab and select the **Integration Knowledge Modules (IKM)** as "IKM BI Materialized View".

10. Click **Save**.

Creating Package

Follow the instructions below to create a package for the new fact.

1. Navigate to **Designer** > **Customizations** > **Materialized Views** > **Packages**.
2. Right-click on the **New Package**.
3. In the **Package Editor** page, enter the name of the package.
4. Click on the **Diagram** tab at the bottom of the editor. From the global objects section, drag the variables **B1_JOB_ID** into the editor. Change the variables **B1_JOB_ID** to declare variable.
5. Drag and drop the interface into the editor and connect them all together in sequence.
6. Click **Save** to save the changes and close the **Package Editor** window.
7. Navigate to the packages folder and expand it. You will see the new package displayed here.
8. Right -click on the **Generate Scenario**.
9. A popup dialog appears asking for the scenario name. Click **OK**
10. Expand the package and you will notice scenarios and under scenarios you can see the scenario object that was generated.

Configuring Entity

Follow the instructions below to configure a new entity for the custom materialized view.

1. Open the Oracle Utilities Analytics Administration User Interface in a browser and login.
2. Click on the **ETL Configuration** tab.
3. Click on the **Target Entity** menu item.
4. Click on the **Add** button on the right.
5. Enter the details of the job in the window.
6. Enter the values for the materialized view name.

Specifying the Dependency for the Custom Materialized View

Follow the instructions below to specify the dependency for materialized view:

1. Add a new step to the user procedure "**CM_<PROD_FLG>_CREATE_MEATADATA**".
2. Use the sample below to create an entry in the entry in the B1_OBJECT_MAP for specifying the base objects that the materialized view depends on.

3. As an example, the sample merge statement below specifies the base object name. The given sample statement mentions the source object name as the base object name. The target object name is the materialized view name. You can specify the source product flag as appropriate. The Object Type flag is "MVDP" to identify the materialized dependency on the base objects.

```
merge
 into b1_object_map  tgt
using (select 'CCB'                              prod_flg
            , 'DIM1'                          source_object_name
            , 'CM_MV'                          target_object_name
            , 1                                     seq
            , 'MVDP'                                 object_type_flg
         from dual )  tgt_val
   on (    tgt.prod_flg           = tgt_val.prod_flg
      and tgt.source_object_name = tgt_val.source_object_name
       and tgt.target_object_name = tgt_val.target_object_name
       and tgt.seq                = tgt_val.seq)
when not matched
then insert
     (
       tgt.object_map_id
     , tgt.prod_flg
     , tgt.source_object_name
     , tgt.target_object_name
     , tgt.seq
     , tgt.object_type_flg
     , tgt.char_entity_flg
     , tgt.upd_dttm
     , tgt.upd_user
     , tgt.owner_flg
     )
     values
     (
       b1_object_map_seq.nextval
     , tgt_val.prod_flg
     , tgt_val.source_object_name
     , tgt_val.target_object_name
     , tgt_val.seq
     , tgt_val.object_type_flg
     , null
     , sysdate
     , sys_context('userenv', 'os_user')
     ,'B1');
```

4. Execute the user procedure to create the entries.

Configuring Job

Follow the instructions below to configure a job for the custom fact.

1. Open the Oracle Utilities Analytics Administration User Interface in a browser and login.
2. Click on the **ETL Configuration** tab.
3. Click on the **Job Configuration** menu item.
4. Click on the **Add** button on the right.
5. Add the details of the job in the window.
6. Select a product from the **Source Product** drop-down list and select the Instance Number.

7. Click on the **Search** icon for the **Target Entity** field. In the popup search window, enter the **Fact Name** and click **Go**.
8. Click on the ID Value and it will take you back to the Job Addition page with the target entity ID populated.
   Set the Slice Start Date/Time as 01-Jan-2000 or the extract date to which the source instance is configured and click **Add** to create the Job Configuration entry.
9. Enable the job while saving the new entry.

Monitoring Job Execution

Now that the job has been configured for customization and activated, you can monitor the job execution using the Administration User Interface or using SQL developer. Below are the steps to monitor the job execution from the Administration User Interface.

1. Open the Oracle Utilities Analytics Administration User Interface in a browser and login.
2. Click on the **ETL Job Execution** tab.
3. Enter the fact name and click **Go** to filter the data.
   To see the latest execution, you can sort by session end date so that the latest execution appears on the top.
4. A new package needs to be created for each materialized view interface.
   The package should contain the global variable **B1_JOB_ID**.

Chapter 5

Extending Analytics

The Analytic Dashboards in Oracle Utilities Analytics (OUA) cover a wide range of reporting requirements. You often might need to see some additional data on the reports to meet site specific requirements. If the data is not available in the star schemas, they can be extracted using one of the support schema extension methods in Oracle Utilities Analytics. Refer to the Chapter 4: Extending Star Schema in this guide for complete details on how this can be done.

With the data available in the star schemas, the additional report requirements can be met either by customizing any of the existing analytics or by adding brand new answers. The below sections describes how to use Oracle Business Intelligence Enterprise Edition (OBIEE) to extend the analytics in Oracle Utilities Analytics product:

- Customizing Existing Analytics
- Creating New Analytics

Customizing Existing Analytics

This section describes how to use Oracle Business Intelligence Enterprise Edition to customize Oracle Utilities Analytics. It includes the following topics:

- Modifying the RPD file
- Customizing the Answers
- Customizing the Report Labels

Modifying the RPD file

All customer modifications must be done in a separate copy of the repository file, which is separate from the product's out-of-the-box repository file. During upgrades to the latest Oracle Utilities Analytics version, any customization done should be merged into the upgraded repository file through the Merge utility of Oracle Business Intelligence Enterprise Edition.

It is recommended that customers use a staging environment for the repository upgrade. However, as long as the customer modifications are done on top of a copy of the base repository file, the Oracle Business Intelligence Enterprise Edition upgrade process should be able to handle most customizations that may be made to the repository file. The simpler the changes, the less complex is the upgrade procedure; hence, it is best to try to limit the changes made to the repository file.

> **Note**: For more information about managing, upgrading and merging repository (.rpd) files, refer to *Oracle Business Intelligence Server Administration Guide*.

Customizing the Answers

For the additional report requirements, if the need is to display additional attributes on an existing report or to include an additional view, then it is recommended to customize the answers delivered with the base product. Create a copy of the base product report and make changes directly to the copy (do not modify the base product report). All user modifications should be saved in a separate custom folder in order to guarantee that any custom modifications are preserved when upgrading to newer versions of Oracle Utilities Analytics later on. The dashboard should be changed to point or refer to the new custom report, or a new custom dashboard can be defined to make use of the customized reports.

> **Note**: The dashboards are overwritten during the upgrade. Any mappings between dashboards and customized answers are lost and must be re-mapped manually. Therefore, you should use a staging environment for upgrade and manually remap dashboards before moving the upgraded customized content into the production environment.
>
> **Note**: For more details about managing, upgrading, and merging presentation catalogs, refer to *Oracle Business Intelligence Presentation Services Administration Guide*.
>
> **Note**: For more details on how to create or edit answers, refer to *Oracle Fusion Middleware User's Guide for Oracle Business Intelligence Enterprise Edition*.

Customizing the Report Labels

You can customize labels or captions on an existing report or report columns. You can provide an override description that is used on the reports instead of the base product description. The override descriptions can be provided via the **Base Field Maintenance** page under the Administration Dashboard in the Oracle Business Intelligence Enterprise Edition Dashboards menu. Once the changes are saved and the cache is cleared, upon the next login, the override descriptions are seen on the report title or the column title.

> **Note**: For more details, refer to the **Maintaining the Administration Dashboards** section in the **Chapter 4: Configuring Oracle Utilities Analytics** in *Oracle Utilities Analytics Administration Guide*.

Creating New Analytics

You can choose to build a completely new report from scratch. This section describes how to use Oracle Business Intelligence Enterprise Edition to add new analytics. It includes the following topics:

- Creating New Answers
- Adding New Labels

Creating New Answers

> **Note**: Before creating new reports, it is recommended to have knowledge of Oracle Business Intelligence Enterprise Edition and Data Warehouse concepts. It is also recommended to have some working knowledge in developing reports using Oracle Business Intelligence Enterprise Edition for a smoother implementation.

Oracle Utilities Analytics provides out-of-the-box dashboards with rich and varied set of analytics for Credit and Collection Analytics, Customer Analytics, Distribution Analytics, Meter Data Analytics, Mobile Workforce Analytics, Outage Analytics, Revenue Analytics, Exception Analytics, and Work and Assets Analytics. However, if required, you can create new answers, or dashboards.

As noted in the above-mentioned section regarding customization of the existing answers, the new answers should also be saved in a separate custom folder so that they are not overwritten when upgrading to newer versions of Oracle Utilities Analytics later on.

You can create field labels for use in their answers, or the labels can be hard coded directly in the answer if there are no multilingual/localization requirements. If the product labels are used in an answer, they can get modified during upgrade to a newer Oracle Utilities Analytics release. At the best, Oracle tries to limit the changes to the existing labels; however, there can be certain situations, when they are updated. Hence, in rare cases, if you are making use of the base labels, then you can expect to have an impact, when the label value changes in a newer release.

Adding New Labels

To use the label mechanism for new answers, the **Custom Field Maintenance** dashboard can be used to add, update, and delete custom labels. These custom labels can then be used in answers as well as in the logical and physical objects in the repository or RPD file.

> **Note**: Only custom field labels, identified by a Customer Modification (CM) owner flag, can be updated or deleted. The new labels are created with a Customer Modification (CM) owner flag. A label that already exists cannot be created, so if a base labels already exists, you can update the override label as described in the preceding section Creating New Answers.
>
> **Note**: For more details, refer to the **Maintaining the Administration Dashboards** section in the **Chapter 4: Configuring Oracle Utilities Analytics** in *Oracle Utilities Analytics Administration Guide*.

Chapter 6

Migrating Environments

Most implementations have multiple environments based on the activities that they carry out. The exact number of environments and the purpose for each of them can vary from implementation to implementation. Below are the typical environments expected during an implementation.

- **Development**: This environment is used to extend the capabilities of the product. This environment is where all custom ELT and custom answers are developed.
- **Acceptance**: This environment is typically used to perform functional validations based on the source system and the custom code. No development happens in this environment.
- **Production**: This environment is used to connect to Oracle Business Intelligence Enterprise Edition and view the dashboards and analytics provided.

The lifecycle of the implementation starts in the Development environments and the code is moved to the Acceptance environment and finally into the Production environment. You may choose to have more than three environments.

This chapter covers the following:

Oracle Business Intelligence Enterprise Edition Components

Migration of Oracle Business Intelligence Enterprise Edition components from one environment to another typically involves moving the two main components of the tool - the presentation catalogs and the repository file. The details of how to migrate each of these and whether a migration is actually needed or not is discussed in the following sections.

Presentation Catalog

You are not expected to modify any of the catalogs delivered with the base product. This being the case the catalogs from base product package can directly be deployed across multiple environments. Follow the same steps as mentioned in the *Oracle Utilities Analytics Installation Guide* in the **Dashboard Component** section.

It is likely for you to extend the analytics by adding some additional dashboards and reports to cater to some additional business requirements. The details for the same have been provided in the Chapter 5: Extending Analytics The important point here is that all new objects should have been saved in a separate folder/catalog other than the base product catalogs.

Now, with the extra objects in place, you need to move these additional catalogs across their environments. The recommended way from Oracle Business Intelligence Enterprise Edition for this is to archive the catalogs from the source environment, move the files across and unarchive them in the target environment. Oracle Business Intelligence Enterprise Edition provides the Catalog Manager utility for this purpose. The utility is available for both windows and unix machines and is installed along with the Oracle Business Intelligence Enterprise Edition product. Follow the below mentioned steps:

1. Start the catalog manager in the source Oracle Business Intelligence Enterprise Edition environment
2. Login in the Online mode. Pick the custom catalogs created and archive them.
3. Save the archived files (.catalog) on the local server.
4. Move these catalog files to the target server through FTP or other available means.
5. Start the **Catalog Manager** in the target Oracle Business Intelligence Enterprise Edition environment in the online mode.
6. Choose the **Unarchive** option and select the catalog files that were moved.

Once the custom catalogs have been successfully deployed, the custom dashboards and reports should start coming up on the target Oracle Business Intelligence Enterprise Edition environment.

> **Note**: For more details on the Catalog Manger, refer to *Oracle Fusion Middleware System Administrator's Guide for Oracle Business Intelligence Enterprise Edition*.

Repository

An implementation can create a custom version of the base product repository file for some additional business requirements. In such cases, the modified repository file needs to be migrated to the other Oracle Business Intelligence Enterprise Edition environments that customers have. The Oracle Utilities Analytics Administration Tool that comes with the Oracle Business Intelligence Enterprise Edition product can be used to save a copy of the repository file.

1. Start the Oracle Utilities Analytics Administration Tool in the source Oracle Business Intelligence Enterprise Edition environment.
2. Open the repository in Online mode.
3. Choose **File menu > Copy As > Repository…**
4. Enter a custom name for the repository such as 'CM_UtilitiesBusinessAnalytics' and save the copy on the local machine.
5. Log into the Oracle Business Intelligence Enterprise Edition Enterprise Manager console of the target Oracle Business Intelligence Enterprise Edition environment.
6. Navigate to **BI Instance > Coreapplication > Deployment**.
7. **Lock and Edit**. The Repository text box will be enabled.
8. Browse to the modified **rpd** file and submit.
9. Provide the RPD password, and then click **Apply.**
10. Activate the changes and then restart Oracle Business Intelligence Enterprise Edition services.

> **Note**: If the database connections to be used by the repository in the target environment are different, then you should first update the connection pool details in the repository file before deploying it on the server.

Oracle Data Integrator Components

Before you begin, note that Oracle Data Integrator imposes certain restrictions on the source for the objects imported. Each environment should have a unique Work Repository ID and Master Repository ID. The Master and Work Repository IDs should be greater than 600. The ID values less than 600 should not be used as these are reserved for the Oracle Utilities Analytics product. Failure to comply to this rule, would result in ID conflicts for you during upgrades to new releases of Oracle Utilities Analytics.

Migrating Oracle Data Integrator components from one environment to another typically involves exporting Oracle Data Integrator objects and importing them in the new environment. The Development environment is the source and all the subsequent environments are the targets where these objects are imported. The export and import are limited to the custom code only.

Each environment should be built or upgraded using the provided installers. After installation or upgrade of individual environments, the custom code can be exported from the Development and imported into the subsequent environments. The following sections provide the instructions that cover the custom code migration from one environment to another.

CM Project

The "Developer Guide" provides detailed instruction on how the custom code should be developed. The first step is the creation of a custom project and ensuring that all custom objects are within this newly created project.

The primary benefit of doing this is that all custom code is completely isolated from the out of the box code. The added benefit is that you can now export the entire CM project and import it in the subsequent environment.

CM Models

In addition to a custom project, you may need to create a custom model folder to organize your custom facts, dimensions, staging or replication objects. These should also be exported from the Development environment into the subsequent environment.

CM Metadata

All the CM metadata created during the customizations should be applied into the subsequent environments. The approach mentioned below should be followed to simplify the process of migrating these.

1. Create a procedure **CM_<PROD_FLG>_CREATE_METADATA**.
2. Replace the **<PROD_FLG>** with the appropriate edge product code (For example, CCB/ NMS).
3. Add appropriate data population scripts.
   These should be written as merge statements so that existing rows are skipped and only new rows are added. In case the metadata requires corrections, use the update clause of the merge statement.

All tasks within the procedure should have the logical schema set to "Metadata". The schema names should not be hard-coded.

In addition, create a package **CM_<PROD_FLG>_CREATE_METADATA**. Add the created procedure as the first step and add the scenario **B1_CFG_METADATA** as a second step. After migration the CM Project to the new environment, execute this step after the addition of the product instance. This job should be executed in the newly created context for the product.