

Oracle Utilities Analytics Warehouse

Developer's Guide

Release 2.8.0.1.0

F75543-01

December 2022

Oracle Utilities Analytics Warehouse Release 2.8.0.1.0 Developer's Guide

F75543-01

Copyright © 2011, 2022 Oracle and/or its affiliates.

Contents

Preface	i
Prerequisite Knowledge.....	ii
Related Documents.....	ii
Conventions.....	iii
Abbreviations.....	iii
Documentation Accessibility.....	iv
Documentation Roadmap.....	iv
Chapter 1	
Getting Started	1-1
Creating a Project.....	1-2
Creating a Model Folder.....	1-2
Using CM Metadata User Procedure.....	1-2
Chapter 2	
User Extension Methods	2-1
Extending Dimensions.....	2-3
Fact Patterns.....	2-3
Extending Facts.....	2-5
Using Custom User-Defined Dimensions (UDD).....	2-5
Custom Dimensions.....	2-6
Custom Facts.....	2-7
Chapter 3	
Extending Replication	3-1
Adding Custom Tables for OUAF-Based Source Applications.....	3-2
Adding Custom Tables for Oracle Utilities Network Management System.....	3-5
Enabling Replication.....	3-7
Creating Replicated Tables.....	3-8
Executing Initial Sync.....	3-11
Verifying Model Setup.....	3-12
Chapter 4	
Extending Star Schema	4-1
UDX Processing.....	4-2
Populating User-Defined Columns.....	4-3
Creating CM Mappings.....	4-4
Creating CM Packages.....	4-6
Resetting Dimensions.....	4-8
Configuring CM Scenarios.....	4-9
Monitoring Job Execution.....	4-10
Validating Data Load.....	4-10
Populating User Defined Foreign Keys.....	4-11
Creating CM Views.....	4-11
Creating CM Mappings.....	4-12

Creating CM Packages.....	4-13
Configuring CM Scenarios.....	4-13
Star Schema	4-14
Custom Dimensions.....	4-15
Creating Dimension Table.....	4-16
Importing Dimension into Model.....	4-18
Importing Replicated Table into Replication Model.....	4-20
Creating Replication Key View in Dimension Model.....	4-21
Creating Mapping for Key Views in Dimension Model.....	4-22
Creating Loading Views in Dimension Model.....	4-24
Creating Mapping for Loading Views.....	4-25
Creating Package for Loading Views.....	4-29
Creating Staging Table in the Dimension Model.....	4-29
Creating Mapping in Dimension Model.....	4-31
Creating Package in Dimension Model.....	4-34
Configuring Entities in Dimension Model.....	4-35
Configuring Jobs in Dimension Model.....	4-35
Monitoring Job Execution.....	4-36
Validating the Data Loaded.....	4-36
Custom Facts.....	4-37
Creating Fact Tables.....	4-38
Importing Fact Tables into Model.....	4-38
Importing Replicated Tables into Fact Model.....	4-39
Creating Key Tables in Fact Model.....	4-41
Creating Mapping for Key Tables in Fact Model.....	4-42
Creating Loading Views in Fact Model.....	4-46
Creating Mapping to Loading Views for Fact Model.....	4-47
Creating Aggregate Tables in Fact Model.....	4-50
Creating Mapping to Load Aggregate Tables in Fact Model.....	4-52
Creating Staging Tables in Fact Model.....	4-55
Creating Error Tables in Fact Model.....	4-56
Creating Mapping to Load Facts.....	4-58
Creating Packages in Fact Model.....	4-63
Configuring Entities in Fact Model.....	4-64
Specifying Dependencies in Fact Model.....	4-64
Configuring Jobs in Fact Model.....	4-66
Monitoring Job Executions.....	4-67
Custom Materialized Views.....	4-67
Creating Mapping for Materialized View.....	4-67
Creating Packages for Materialized View.....	4-68
Configuring Entities for Materialized View.....	4-68
Specifying Dependencies for Materialized View.....	4-69
Configuring Jobs for Materialized View.....	4-69
Monitoring Job Execution.....	4-70

Chapter 5

Extending Analytics.....	5-1
Modifying the RPD File.....	5-2
Customizing Answers.....	5-2
Customizing the Report Labels.....	5-3
Creating New Analytics	5-3
Creating New Answers.....	5-3
Adding New Labels.....	5-4
Customizing Hierarchy Levels.....	5-4

Chapter 6

Migrating Environments	6-1
Presentation Catalog.....	6-2

Repository	6-2
Migrating ODI Components	6-3
CM Project	6-3
CM Models.....	6-4
CM Metadata.....	6-4

Preface

Welcome to the Oracle Utilities Analytics Warehouse Developer's Guide.

This guide focuses on how you can get started with configuring and administering Oracle Utilities Analytics Warehouse (OUAW), Release 2.8.0.1.0. It provides instructions to extend the product, replication, and star schemas, so you can carry out an out-of-the-box implementation.

In the preface:

- [Audience](#)
- [Prerequisite Knowledge](#)
- [Related Documents](#)
- [Conventions](#)
- [Abbreviations](#)
- [Documentation Accessibility](#)
- [Documentation Roadmap](#)

Audience

This guide is primarily for the developers extending the functionality of the product for implementations based on their custom requirements. It does not teach Oracle Data Integrator (ODI) or Oracle Analytics Server (OAS) fundamentals but expects the users to be familiar with development using ODI and OAS.

The developers are expected to be proficient in the following technologies:

- Oracle Data Integrator
- Oracle Analytics Server
- Oracle GoldenGate
- Oracle Database
- Oracle WebLogic

Note: It is assumed 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, “sh” scripts have to be replaced with the corresponding “cmd” scripts.

Prerequisite Knowledge

Oracle Utilities Analytics Warehouse uses several technologies. It is assumed that you have a working knowledge of the following to install and configure Oracle Utilities Analytics Warehouse.

- [Oracle Data Warehouse](#)
- [Oracle GoldenGate](#)
- [Oracle Data Integrator](#)
- [Oracle GoldenGate Monitor](#)
- [Oracle WebLogic Server](#)
- [Oracle Analytics Server](#)

Related Documents

Refer to the [Oracle Utilities Analytics Warehouse Installation and Configuration Checklist](#) for high-level steps to install and configure the Oracle Utilities Analytics Warehouse product. **Note** that this checklist includes information for 2.7.0.2 and is applicable for 2.8.0.1 also.

The following documentation is included in this release.

Installation, Administration, and Release Notes

- *Oracle Utilities Analytics Warehouse Release Notes*
- *Oracle Utilities Analytics Warehouse Getting Started Guide*
- *Oracle Utilities Analytics Warehouse License Information User Manual*

- *Oracle Utilities Analytics Warehouse Installation and Configuration Guide*
- *Oracle Utilities Analytics Warehouse Quick Install Guide*
- *Oracle Utilities Analytics Warehouse Developer's Guide*

Metric Reference Guides

Refer to the [Oracle Utilities Analytics Warehouse](#) documentation on Oracle Help Center for details about the metric reference guides included in this release.

Data Mapping Guides

Refer to the [Oracle Utilities Analytics Warehouse](#) documentation on Oracle Help Center for details about the data mapping guides included in this release.

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.
<i>italic</i>	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.

Abbreviations

The following table lists the commonly used abbreviations used in this document:

Abbreviation	Expanded Form
OUAW	Oracle Utilities Analytics Warehouse
APEX	Oracle Application Express
CCB	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

Abbreviation	Expanded Form
OAS	Oracle Analytics Server
ODI	Oracle Data Integrator
ODM	Oracle Utilities Operational Device Management
OGG	Oracle GoldenGate
OWB	Oracle Warehouse Builder
WAM	Oracle Utilities Work and Asset Management
OAAF	Oracle Utilities Application Framework

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the [Oracle Accessibility Program](#) website.

Access to Oracle Support

Oracle customers have access to electronic support through My Oracle Support. For more information, visit [My Oracle Support](#) or [Oracle Accessibility Learning and Support](#) if you are hearing impaired.

Documentation Roadmap

This guide is organized based on the typical flow you need to follow during the Oracle Utilities Analytics Warehouse implementation. Use the following documentation roadmap to find the information that you need to implement Oracle Utilities Analytics Warehouse.

- [Getting Started](#): Find out what you need to begin customizing the product.
- [Chapter 2: User Extension Methods](#): Gain a high-level understanding of the characteristics and extensible attributes needed to customize and extend the product.
- [Chapter 3: Extending Replication](#): Understand the replication capabilities of the product. This chapter discusses the replication of tables required for processing and loading data into the data warehouse.
- [Chapter 4: Extending Star Schema](#): Find out how a schema can be extended using 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).
- [Chapter 5: Extending Analytics](#): Explains how to use Oracle Analytics Server to extend the analytics in Oracle Utilities Analytics Warehouse.
- [Chapter 6: Migrating Environments](#): Discusses about the environments needed to carry out the implementation.

Chapter 1

Getting Started

Before beginning any product customization, create a custom project so that all the customizations are isolated from the product components. Then, create the custom objects under the custom project.

This chapter includes the following to proceed with customization:

- [Object Naming Convention](#)
- [Creating a Project](#)
- [Creating a Model Folder](#)
- [Using CM Metadata User Procedure](#)

Object Naming Convention

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

Use 'CM' as a prefix for all objects that you create (CM references to Customer Modification).

Creating a Project

Login to Oracle Data Integrator Studio to create a new project to maintain all custom mappings, procedures, and packages.

The project should include the following folder structure to organize objects:

- **Facts:** To organize all fact mappings.
- **Dimensions:** To organize all dimension mappings.
- **Replication:** To organize all replication view mappings.
- **Materialized Views:** To organize all materialized view mappings.

Create these folders for each product. Avoid cross referencing across different folders.

For example: A mapping under the **Dimensions** folder should not refer to a mapping in the **Replication** folder.

Creating a Model Folder

All custom model objects reside in a custom model folder. The structure of the model folder is similar to that of a custom project. For more details, refer to [Creating a Project](#).

Using CM Metadata User Procedure

Use the CM metadata user procedure to create new entries in the metadata tables. It helps in migrating the same metadata to different environments. This procedure is used to populate custom labels for the dashboards.

To execute the CM metadata procedure:

1. Create a **CM_<PROD_FLG>_CREATE_METADATA** procedure.

Replace **<PROD_FLG>** with the appropriate source application code. For example: CCB, NMS, MDM, or MWM.

2. Add the appropriate data population scripts.

These should be written as merge statements, so the 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.

3. Create a **CM_<PROD_FLG>_CREATE_METADATA** package.
4. Add the procedure created in step 1 and then add the **B1_CFG_METADATA** scenario.

B1_CFG_METADATA pulls additional metadata from the source based on the list of tables to extend the replication.

5. After migrating the CM Project to a new environment, add the product instance.
6. Execute the **CM_<PROD_FLG>_CREATE_METADATA** custom procedure.

This job should be executed in the context of the product.

Chapter 2

User Extension Methods

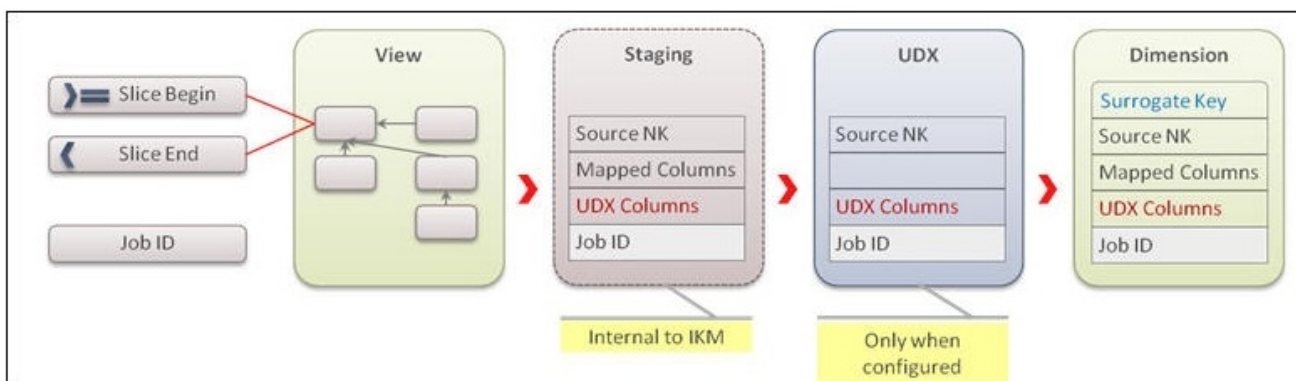
Amongst other extensibility options, Oracle Utilities Analytics Warehouse supports extending schemas. Some of the source systems are highly customizable and customers can extend the edge applications to utilize additional attributes or functionalities that are not in the out-of-the-box solution. This necessitates Oracle Utilities Analytics Warehouse to be flexible and capable of handling additional attributes or other extensibility options.

To make sure this is possible, the star schemas have been created with the following extensible attributes in the facts and dimensions.

- [Dimension Patterns](#)
- [Extending Dimensions](#)
- [Fact Patterns](#)
- [Extending Facts](#)
- [Using Custom User-Defined Dimensions \(UDD\)](#)
- [Custom Dimensions](#)
- [Custom Facts](#)

Dimension Patterns

The following figure illustrates the stages of processing in a dimension and the components utilized in developing a dimension load process.



The data load processes comprise of a package and one or more mapping. The package uses **B1_JOB_ID** mandatory variable as an input. It is used to pass the current job identifier.

The first mapping is usually a view where filters are applied based on the variables to exclude data that does not fall into the specified range. It reduces the data processed in one execution. The data from the view is first inserted into a *staging* table. The staging table includes the following:

- 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 (refer to the [UDX Processing](#) section for more details about UDX tables) is created only if the CM procedure has been configured for the entity. It includes the following:

- 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 following figure illustrates the steps required to extend a dimension.

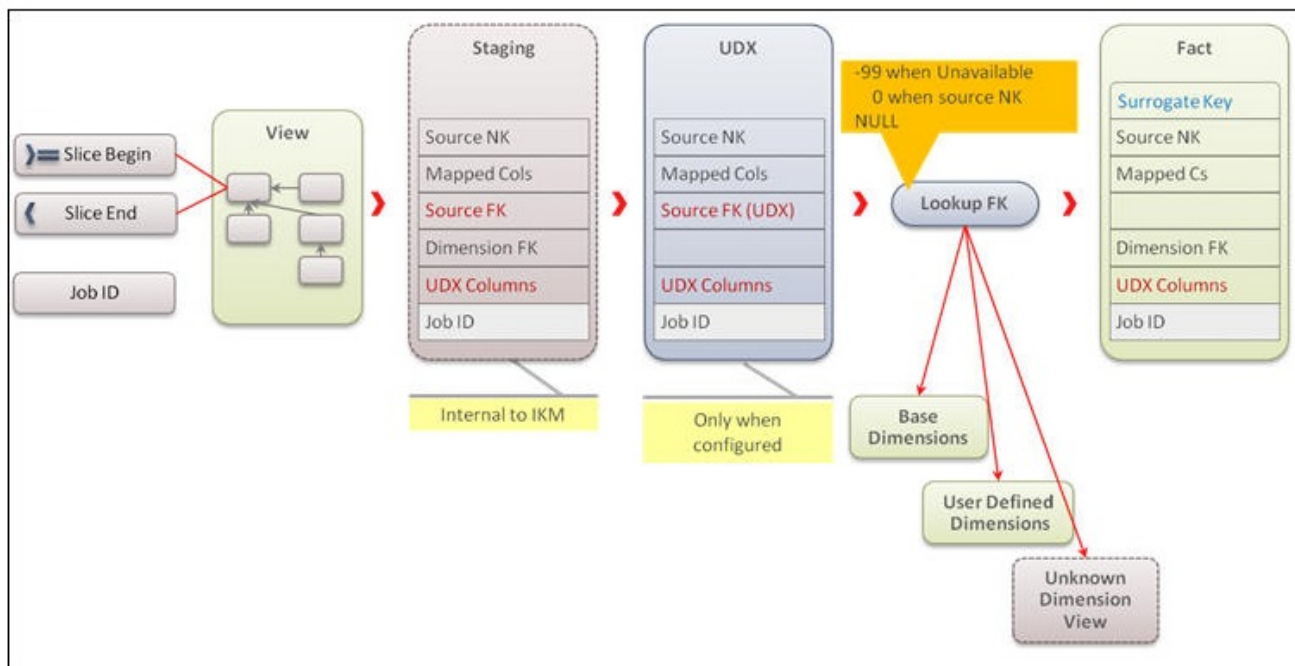


To extend a dimension create an Oracle Data Integrator mapping using the UDX table as source and target along with other source tables. The CM mapping updates the user defined fields (UDF?_CD and UDF?_DESCR) columns based on the input parameters and natural key of the UDX table.

After writing the package using the CM mapping, configure it and enable the jobs (Refer to the [UDX Processing](#) section in [Chapter 4: Extending Star Schema](#)). If data has already been loaded, the user-defined fields are populated for incremental changes. To load the data for all rows, reset the dimension using the reset scenario. Note that resetting a dimension resets the dependent facts also.

Fact Patterns

The following figure illustrates the stages of processing in a fact and the components utilized in developing a fact load process.



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

- **B1_JOB_ID**: Passes the current job identifier.

- **B1_DEF_MISSING_KEY:** Passes the -99th key value for late arriving dimension.
- **B1_DEF_NULL_KEY:** Passes the 0th key value for non-existing dimension value.

The first mapping is usually a view where filters are applied based on the variables to exclude data that does not fall into the specified range. This reduces the data processed in one execution.

The data from the view is first inserted into a *staging* table. The staging table includes the following:

- 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).
- 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 (refer to the [UDX Processing](#) section for more details about UDX tables) is created only if the CM procedure has been configured for the entity. This table includes the following:

- Source natural key columns.
- Columns marked for user extension (these are UDDGEN, UDM and UDD_KEY columns)
- Columns required for looking up the foreign keys to dimensions.
- Job identifier to segregate data from multiple parallel executions of a data load process.

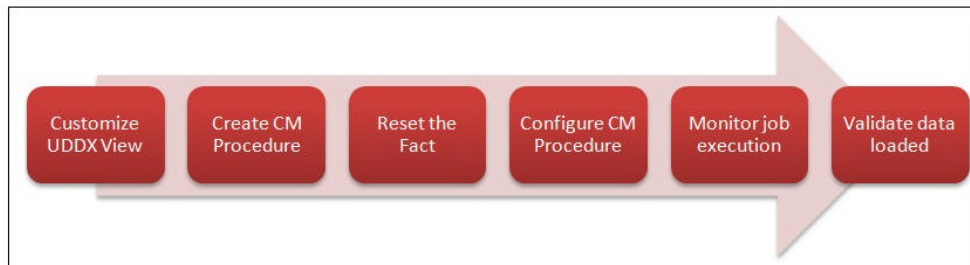
An additional step in the fact processing is the foreign key lookup for dimensions. There are three types of dimensions:

- *Base dimensions* are populated out of the box.
- *User-Defined Dimensions* (UDDs) are additional dimensions for which a template table is provided in the out-of-the-box product. Refer to the [Using Custom User-Defined Dimensions \(UDD\)](#) section for information about user-defined dimensions.
- *Unknown dimensions* are the objects where tables are not provided and custom dimensions have to be created. There is a built-in lookup so that custom UDD lookups do not require any code change.

The data is finally loaded into the target dimension.

Extending Facts

The following figure illustrates the steps required to extend a fact.



The procedure to extend a fact is similar to that of extending a dimension, but includes custom dimension lookup as well. Refer to the [Extending Dimensions](#) section for more details.

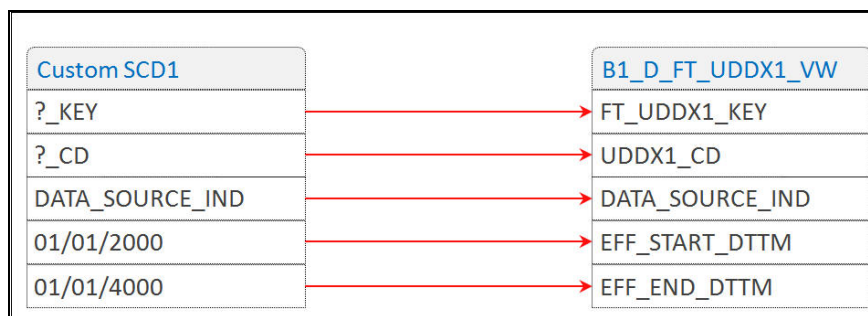
Using Custom User-Defined Dimensions (UDD)

For a custom dimension lookup, customize the UDDX views first to refer to a custom dimension as illustrated below.

SCD1

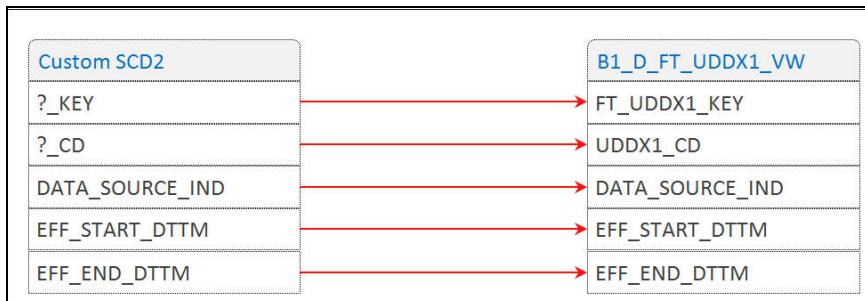
If a custom dimension lookup is required, the UDDX views have to be customized to refer to a custom dimension.

Assuming that the custom dimension is of type 1, 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, 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.



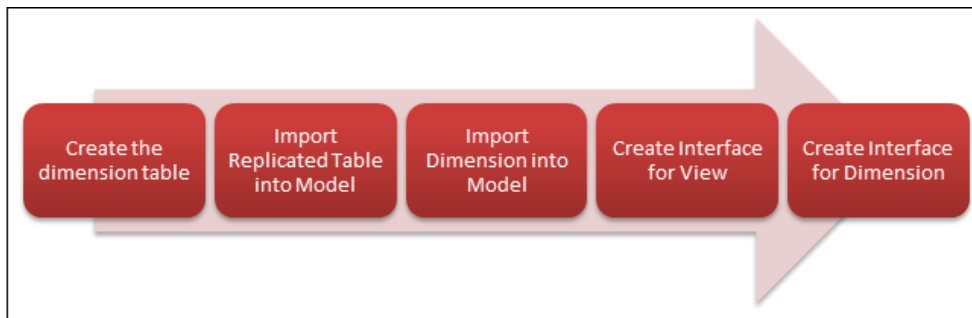
The lookup functions as designed and the out-of-the-box fact refers to a custom dimension. Then, create an Oracle Data Integrator package with the ODI CM mapping. The CM mapping updates the user-defined field columns based on the input parameters and natural key of the UDX table.

Note: The dimensions consist of a minimum of ten UDF columns. These columns are used to store additional information from the source systems. For example: UDF1_CD, UDF2_CD, UDF1_DESCR, UDF2_DESCR, etc.

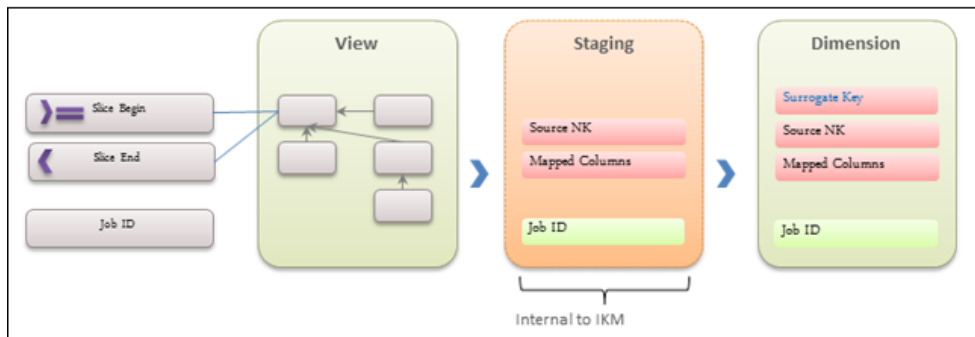
After writing the CM package, configure it and enable the jobs (Refer to the [UDX Processing](#) section in [Chapter 4: Extending Star Schema](#)). If data has already been loaded, user-defined fields are populated for incremental changes. To load the data for all rows, reset the fact using the reset scenario.

Custom Dimensions

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

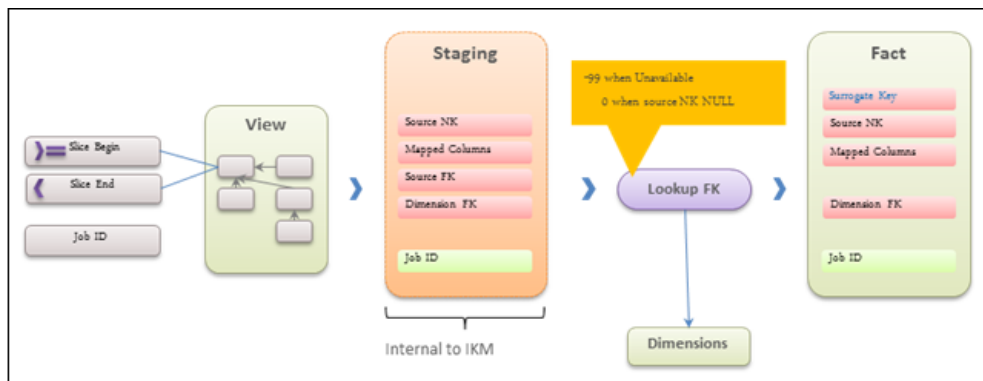


Create the table and a sequence in the database. The dimension table should have a surrogate primary key and a unique key which includes the data source indicator and a column from the source.



Custom Facts

The following diagram shows the pattern to be used while developing the ODI components for a custom dimension. It is similar to the out of the box pattern with the user extension component excluded.



Chapter 3

Extending Replication

Oracle Utilities Analytics Warehouse allows to extend the capabilities of the product. The out-of-the-box solution enables replication of several tables required for processing and loading data into the data warehouse.

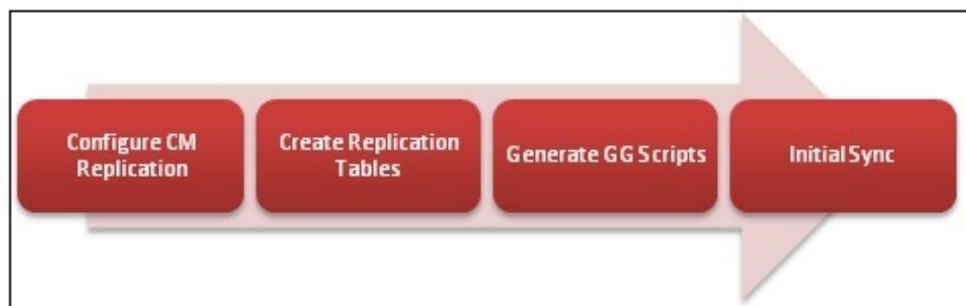
However, the implementer's requirements may vary and additional information might be needed in facts and dimensions that are not included in the out-of-the-box solution. Some of these extension requirements may be met by using the tables that are already being replicated out of the box. For others, additional tables may need to be included in the replication process.

This chapter covers the following:

- [Including Tables](#)
- [Adding Custom Tables for OUAF-Based Source Applications](#)
- [Adding Custom Tables for Oracle Utilities Network Management System](#)
- [Enabling Replication](#)
- [Creating Replicated Tables](#)
- [Executing Initial Sync](#)
- [Verifying Model Setup](#)

Including Tables

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



To configure the replication:

1. Login to the **Administration** user interface.
2. Navigate to **Source Table** configuration and identify the table to be replicated.
3. Set the **CM Replication** flag to “Yes”.
4. Set up Oracle GoldenGate and complete the initial synchronization.

Adding Custom Tables for OUAF-Based Source Applications

Most of the tables related to tables used for populating the out-of-the-box star schemas are listed in the metadata configuration “Source Tables”. It is possible that the table required to be extended is not listed.

To include the table to be extended in the source tables list:

1. Create a procedure **CM_<PROD_FLG>_CREATE_METADATA**. Replace **<PROD_FLG>** with the appropriate edge product code.

For example: CCB/NMS/MDM/MWM/WAM

2. Create a new task for each metadata entry into **B1_OBJECT_MAP**. The tasks within the procedure should have the logical schema set to “Metadata”.

B1_OBJECT_MAP requires two entries - one entry mapping the MO to a custom view and the second entry mapping the custom view to the target custom fact or dimension.

Step 3 creates the first entry and step 4 creates the second entry.

3. Add an entry in **B1_OBJECT_MAP** setting **SOURCE_OBJECT_NAME** as the MO name and **TARGET_OBJECT_NAME** as the target fact or dimension, which has attributes loaded from this table.

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

For example: The following merge statement sets the tables under a maintenance object in Oracle Utilities Customer Care and Billing for inclusion in the replication process.

- **Source Product Flag** is the product flag of the source. In this example, it is 'CCB' for Oracle Utilities Customer Care and Billing.
- **Source Object Name** is 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.
- **Target Object Name** is the ETL view that uses the tables of this maintenance object. In this example, **CM_TEST_VW** is specified as dummy value.
- **Object Type Flag** is the type of object to be replicated. In this example, replicating the entire Budget Review MO is specified; hence 'MO' has been specified.

```
merge
into bl_object_map tgt
using (select 'CCB'
           , 'BUD REVIEW'
           , 'CM_TEST_VW'
           , 1
           , 'MO'
       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
(
    bl_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. Run the following **Insert** statement to specify that **CM_TEST_VW** ETL view populates 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 the **CM_<PROD_FLG>_CREATE_METADATA** package.
 - a. Add the procedure created in step 1.
 - b. Add the **B1_CFG_METADATA** scenario and then add the **B1_CFG_INSTANCE_JOBS** scenario.
 - c. After migrating the CM Project to a new environment, execute the custom procedure **CM_<PROD_FLG>_CREATE_METADATA** after adding the product instance.

This job should be executed in the context for the product.

Executing this package in the appropriate context ensures that the required tables are present in the metadata configuration tables. For instructions, refer to the [Enabling Replication](#) section.

These instructions are applicable to all source applications except Oracle Utilities Network Management System, which does not use Oracle Utilities Application Framework (OUAF).

Note: For more details, refer to the **Mapped Objects** section in *Oracle Utilities Analytics Warehouse Installation and Configuration Guide*.

Adding Custom Tables for Oracle Utilities Network Management System

Most of the tables related to tables used for populating the out-of-the-box star schemas are listed in the metadata configuration “Source Tables”. It is possible that the table required to be extended is not listed.

To include the table to be extended in the source tables list:

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

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

For example: The following merge statement sets the tables under a maintenance object in Oracle Utilities Customer Care and Billing for inclusion in the replication process.

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

```
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. Execute the following **Insert** statement to specify that the ETL view **CM_TEST_VW** populates the target **CM_F_ZZZ**. The Source Product Flag is 'NMS'.

```

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 the **CM_NMS_CREATE_METADATA** package.
- Add the procedure created in the steps mentioned above.

- b. Add the **B1_CFG_METADATA** scenario.
- c. Add the **B1_CFG_INSTANCE_JOBS** scenario.
- d. After migrating the CM Project to new environment, execute the package after adding the product instance.

This job should be executed in the context for the product.

Executing the created package in the appropriate context ensures that the required tables are present in the metadata configuration tables. For instructions, refer to the [Enabling Replication](#) section.

Enabling Replication

This section describes an example that guides you through the steps to extend the replication.

Important! The screens used in this section are taken from Oracle Utilities Customer Care and Billing and the **CI_ACCT_CHAR** table is used for illustration only. The application and tables to be configured differ in the implementation. Ensure that the values are appropriately modified before performing this exercise.

The following conventions are used in the procedure:

- >> “{Product}” - Used to denote the product code.

For example: Oracle Utilities Customer Care and Billing, Oracle Utilities Network Management System, Oracle Utilities Work and Asset Management, Oracle Utilities Meter Data Management, or Oracle Utilities Mobile Workforce Management

- >> “{Table}” - Specify the table name.
- >> “{Context}” - Specify the context.

To enable CM replication:

1. Open the **Oracle Utilities Administration** user interface in a browser.
2. Navigate to **ETL Configuration > Source Tables**. Filter by **CI_ACCT_CHAR** and click **Go**.
3. On the **Source Table** page, click the edit icon and edit the record.

Source Table Id	Source Product	Table Name	History Type	Base Replication	Custom Replication	Owner
24	Customer Care and Billing	CI_ACCT_CHAR	Effective Dated	N	N	Base Product

- On the **Maintain Source Table** page, select **Yes** from the **Custom Replication** drop-down list.

The screenshot shows the 'Maintain Source Table' form with the following data:

Field	Value
Source Table Id	24
Source Product *	Customer Care and Billing
Table Name *	CI_ACCT_CHAR
History Type *	Effective Dated
Effective Date Column Name	EFFDT
Characteristic Entity	
Base Replication *	No
Group Number *	4
Custom Replication	Yes
Purge Indicator *	No
Replication Retention Days	60

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

Creating Replicated Tables

With the configuration changes complete, the next step is to replicate the table by creating it in the replication schema.

To create a replica table in the replication schema:

- On the **Configuration Type** page, select **Upgrade Source** and click **Next**.

The **Source Product** page shows all the registered source contexts.

- Select the required source product from the **Source Product** list and click **Next**.

The **Source Details** page shows the previously configured values for the selected source.

- Modify the values s required and click **Next**.

The table below provides a brief description of the fields on this page:

Field Name	Description	Value
DB Host	Source database host name	
DB Port	Source database port	Default port is 1521
DB Service Name	Source database service name	

Field Name	Description	Value
DB Home Path	Source database home installed location. If Oracle GoldenGate for source is not installed on the source database server, provide the Oracle client home location on the server on which Oracle GoldenGate is installed.	
Drill Back URL	Drill back URL for the source database	
DB Schema Name	Source schema name	
Extract Start Date (YYYYMMDD)	Date from which data should be extracted from the source	
Socks Proxy	Socks proxy host and port separated by a ':'	Provide the value only if a socks proxy has been setup. Else, leave the field blank.

Important! While upgrading a source registered using Oracle Utilities Analytics versions prior to 2.7.0, values for the database schema name and drillback URL do not appear by default. These parameters must be entered to proceed with the upgrade of the source.

The **GoldenGate Details** page shows values for the selected source that were configured previously.

4. Modify the values where required and click **Next**.

The table below provides a brief description of the fields on this page.

Field Name	Description	Value
Host	Source GoldenGate server host	
Home Path	Oracle GoldenGate installed location on the source database server	Example: opt/local/ggs_home
Source Database Home	Source database home installed location	
Manager Port	Port number on which Oracle GoldenGate Manager is running on the Oracle GoldenGate host.	Default dynamic min port is 7830. Default dynamic max port is 7880.
Encryption Algorithm	Algorithm configured in Oracle GoldenGate on the source server	AES128

Field Name	Description	Value
Encrypt Key	Encrypt Key configured in Oracle GoldenGate on the source server	Provide encryptkey created while setting up GoldenGate on source database server. For details, refer to the Setting up Oracle GoldenGate on the Source Database Server section in the <i>Oracle Utilities Analytics Warehouse Installation and Configuration Guide</i> .
Shared Secret	Shared secret key configured in Oracle GoldenGate on the source server	For instructions to get this value, refer to the Generating the Shared Secret Password section in the <i>Oracle Utilities Analytics Warehouse Installation and Configuration Guide</i> .
GoldenGate Owner User	User name of GoldenGate Owner user	
GoldenGate Owner Password	Password of GoldenGate Owner user	

Important: While upgrading a source registered using Oracle Utilities Analytics versions prior to 2.7.0, parameter values for Oracle GoldenGate Owner User and Oracle GoldenGate Owner Password are not populated by default. These parameters must be re-entered to proceed with upgrade of the source.

- On the **Source JAgent Details** page, enter the following details in the respective fields. Click **Next**.

Field Name	Description	Value
JAgent Host	Host of Oracle GoldenGate JAgent	
JAgent GoldenGate	Oracle GoldenGate installed location where GoldenGate JAgent is running	Example: /opt/local/ggs_12.1.2.1.0
JAgent Port	Port number on which Oracle GoldenGate JAgent is running on the GoldenGate host	
JAgent User	JAgent user name	
JAgent Wallet Password	JAgent Wallet password	
Confirm JAgent Wallet Password	Re-enter JAgent Wallet password to confirm	

- On the **Configuration Summary** page, the log file location details are displayed. Click **Configure**.

The **Configuration Progress** page shows the status of the configuration.

- Click **Next**.

The **Completion Summary** page shows the log file location details.

- Click **Finish**.

Upon completion, the status of source registration is shown in a prompt. The detailed logs of the operation are available in the logs/system/deployodi.log file on the Oracle Utilities Analytics Home page.

- Login to SQL Developer and run the following query to verify that the table has been replicated, but there is no data in the table.

```
select *
  from ccblrep.ci_acct_char;
```

Executing Initial Sync

After the source is configured, the replication schema needs to be loaded with the current data from the source. This is done in the initial sync process. It is triggered by executing the B1_SYNC_CONTEXT ODI scenario.

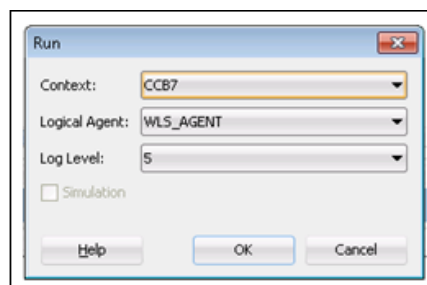
The B1_SYNC_CONTEXT scenario can be triggered in the following ways:

- Using ODI Studio
- Using ODI Console Web Application

Using ODI Studio

To execute B1_SYNC_CONTEXT using the ODI Studio:

- Login to the ODI Studio.
- On the **Designer** tab navigate to the **Load plans and Scenario** folder.
- Expand the **Framework** folder.
- Right-click **B1_SYNC_CONTEXT Version 001** and click **Run**.
- In the **Run** window, select appropriate values for **Context** and **Logical Agent** respectively. Example: Select 'CCB7' as the **Context** and 'WLS Agent' as the **Logical Agent**.



- Click **OK** to save the values.

Using ODI Console Web Application

To run B1_SYNC_CONTEXT using the ODI console web application:

- Login to ODI console.

The ODI console is deployed when the WebLogic agent for ODI is created. The URL format is as below:

```
http://<Weblogic Host>:<Managed Server port>/odiconsole
```

- Login to the Work repository using the 'SUPERVISOR' credential.
- In the browser, navigate to Runtime > Scenario/Load Plan > Folders > Framework.
- Right-click **B1_SYNC_CONTEXT - 001** and click **Execute**.

The B1_SYNC_CONTEXT scenario is executed.

Verifying Model Setup

After the initial sync process is complete:

- Verify that the model is set up.

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

If the record does not exist, it indicates that the Oracle GoldenGate scripts for CCB1AE model were not deployed.

- Verify that the table data is in sync.

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

```
select *
  from ccblrep.ci_acct_char;
```

If there is no entry it indicates that the B1_SYNC_CONTEXT scenario was not executed. Or, if it was executed, the Oracle GoldenGate scripts were not deployed at that time.

Important:

- Enable all the replication tables required for customization and follow the steps mentioned in the sections [Creating Replicated Tables](#) and [Executing Initial Sync](#).
- Ensure that each model does not include more than 100 tables.

Chapter 4

Extending Star Schema

The data warehouse schema in Oracle Utilities Analytics Warehouse covers a wide range of reporting requirements. Often additional data elements are required to meet site-specific requirements. Oracle Utilities Analytics Warehouse allows such extensions to the schema through the user-defined constructs, such as User Defined Fields, User Defined Measures, User Defined Degenerate Dimensions, User Defined Foreign Keys, and User Defined Dimensions. Using these constructs, the star schemas delivered along with the product can be extended.

This chapter includes the following:

- [User Extensible Columns](#)
- [UDX Processing](#)
- [Populating User-Defined Columns](#)
- [Populating User Defined Foreign Keys](#)
- [Star Schema](#)
- [Custom Dimensions](#)
- [Custom Facts](#)
- [Custom Materialized Views](#)

User Extensible Columns

Predefined facts and dimensions are provided with a set of user extensible columns that are used to extend the existing entities. These columns include the following:

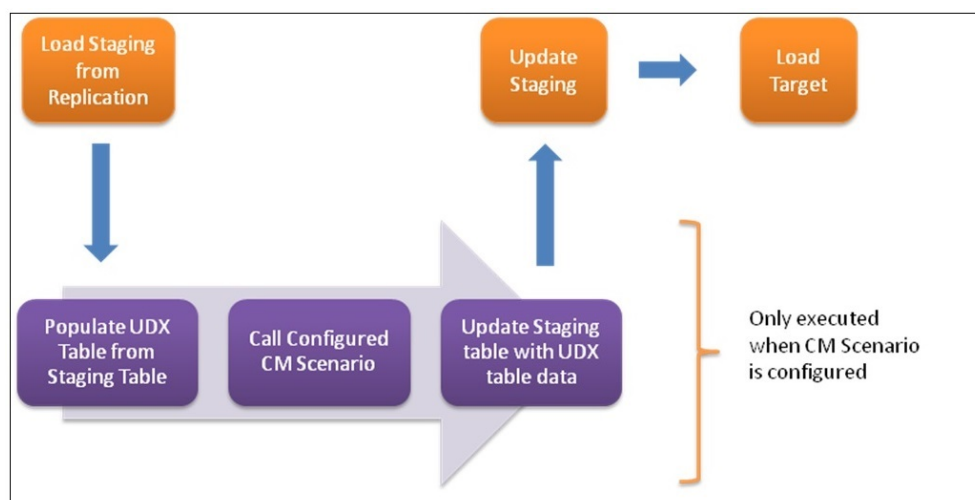
- **User Defined Field:** Resides on the dimension tables in the star schemas. In general, all the dimensions consist of a minimum of ten UDF columns. These columns can be utilized to store additional information from the source systems.
- **User Defined Measure:** Supports the storage of implementation-specific measures that are not provided in the out-of-the-box facts.
- **User Defined Degenerate Dimension:** Reside directly on the fact. They store the dimension attributes that do not fit into a particular dimension, but are required for analytical purposes.
- **User Defined Foreign Key Dimensions:** Empty foreign key attributes not associated with the out-of-the-box dimensions. They allow you to reuse an existing dimension or to create a custom dimension and build a reference in the fact.
- **User Defined Dimension:** Empty dimensions that are delivered along with the star schemas in Oracle Utilities Analytics Warehouse.

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 Warehouse, extending the out-of-the-box dimensions and facts relies on a configurable package with a predefined signature. All entities are set up with a functionality that executes the custom package, if configured.

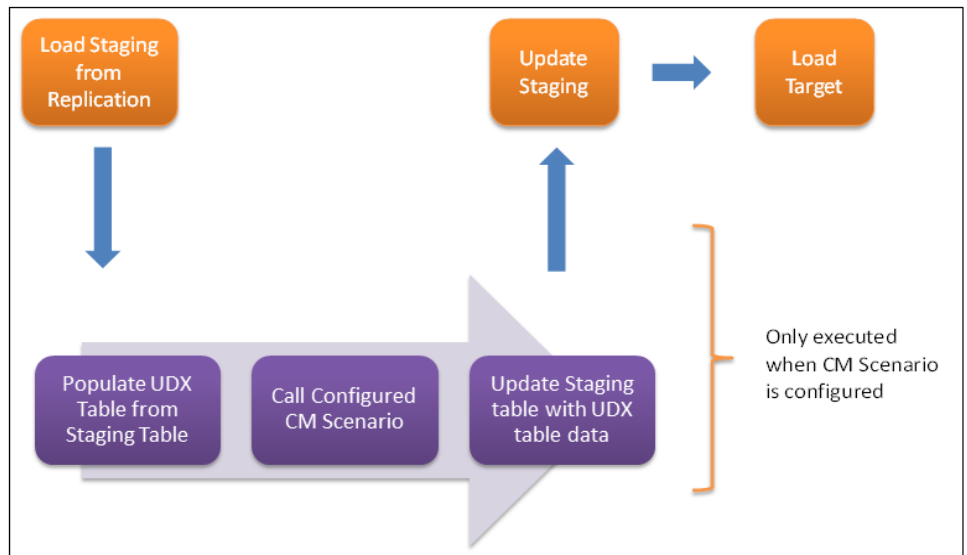
The following figure illustrates the processing logic when the user exit procedure is executed.



All mappings process data using staging tables. The steps in the process are listed as below:

1. The staging table is loaded using a source view.
2. After the staging table is loaded, configurations are looked up.

If the CM package scenario is configured for the job, a UDX table is created. The UDX table contains a natural key and all user extensible columns. The table acts as a template. Update the UDX columns based on the natural key columns and the input parameters.



3. After the CM package scenario 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.

4. The final data is loaded into the target entity.

Note that the process is simplified and reduced to only creating a CM package scenario and configuring it.

Populating User-Defined Columns

The functionality of the dimensions and facts can be extended using user defined columns. ODI-based mapping and package are created to extend the columns, where ODI is used to define the custom package.

Use ODI to create the package for the following reasons:

- Schema names need not be hardcoded.
- Easier to deploy (execute in the appropriate context).
- Easy to deploy for multiple instances of the same source system.

This section includes the following:

- [Creating CM Mappings](#)
- [Creating CM Packages](#)
- [Resetting Dimensions](#)
- [Configuring CM Scenarios](#)
- [Monitoring Job Execution](#)
- [Validating Data Load](#)

Creating CM Mappings

This section describes the process of extending the CD_ACCT dimension using sample data.

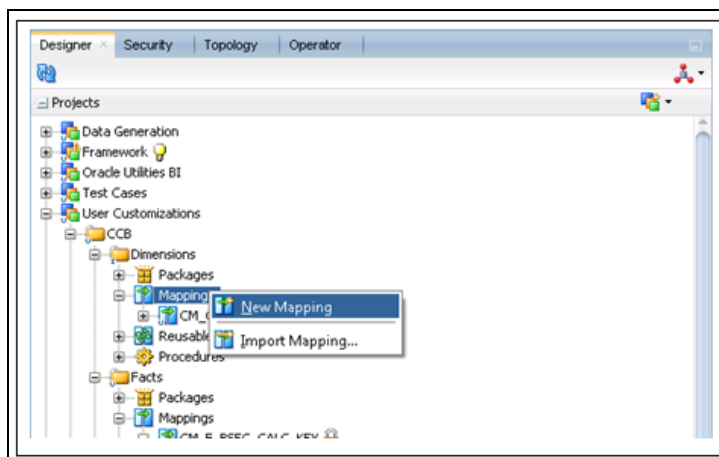
Note: For this example, assume ‘CCB1’ to be the context defined for Oracle Utilities Customer Care and Billing source attached to Oracle Utilities Analytics Warehouse.

To create a CM procedure:

1. Login to the Oracle Data Integrator client.
2. On the **Designer** tab, navigate to **Projects > User Customizations > <product_name> > Dimensions > Mapping**.

In this example, ‘CCB’ is the product.

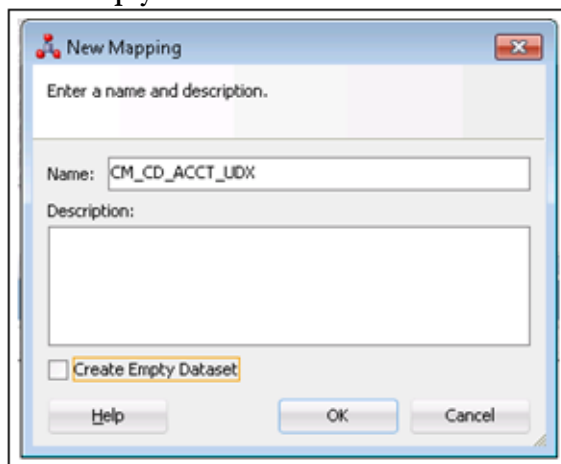
3. Right-click **Mapping** and select **New Mapping** from the menu.



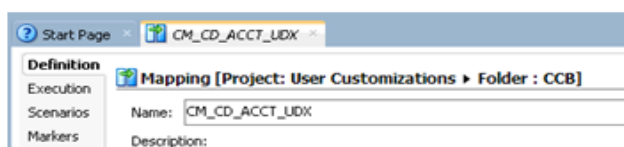
4. In the **New Mapping** window, enter the name of UDX in the **Name** field.

For example: CM_CD_ACCT_UDX

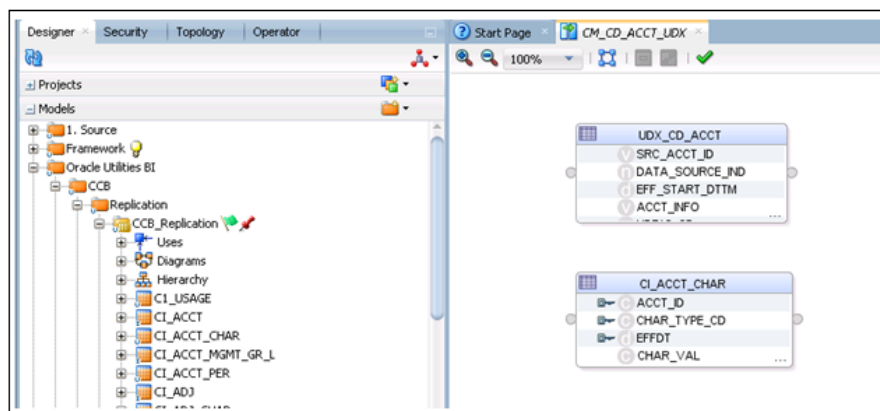
- Unselect the **Create Empty Dataset** checkbox.



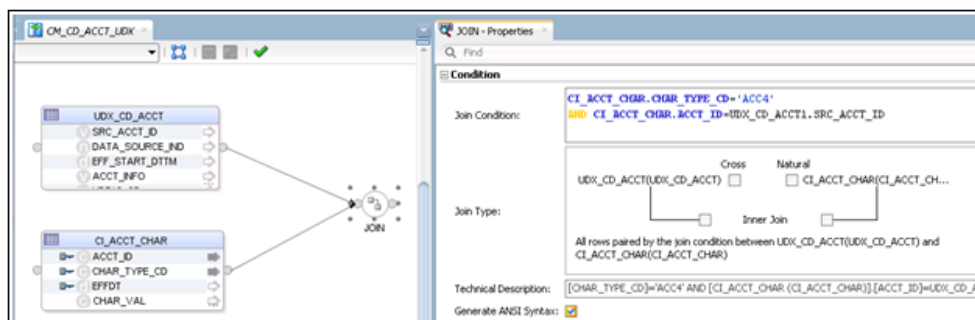
A new mapping “CM_CD_ACCT_UDX” is created.



- Click the **Logical** tab to view the table structure.
- From the **Models** section, drag and drop the UDX and replication tables in the designer pane.

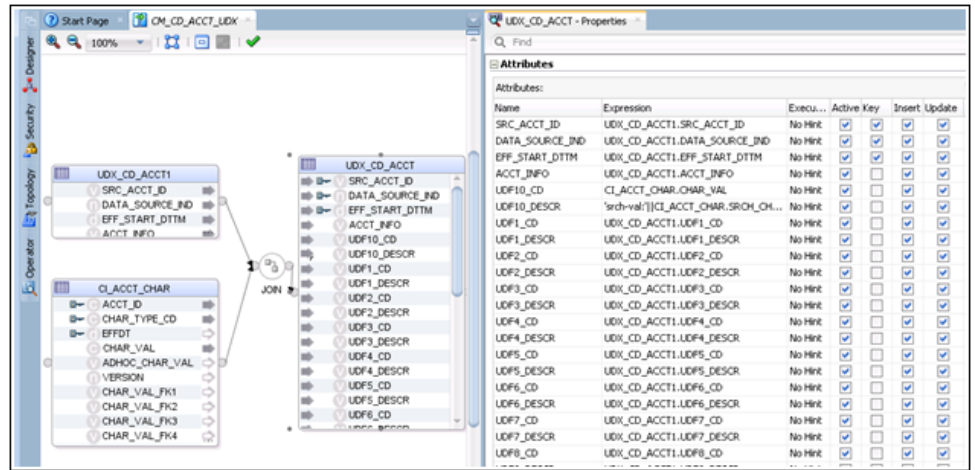


- Join the UDX and replication table.

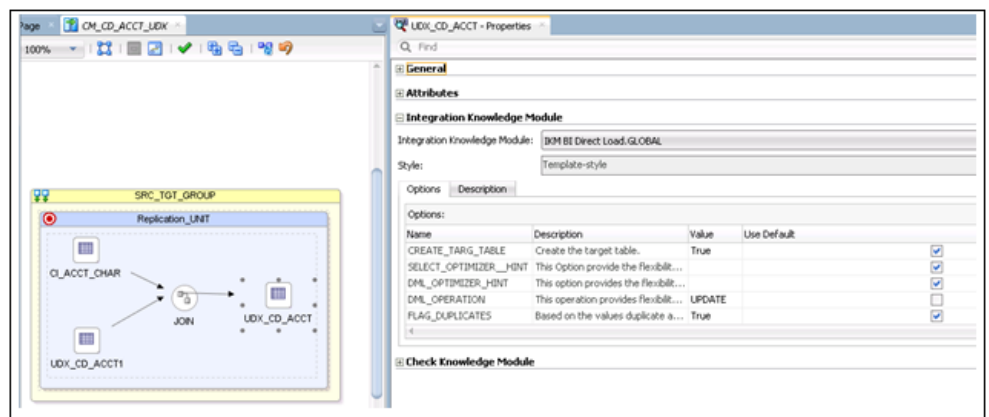


- Drag and drop the UDX_CD_ACCT target data store.

Select UDX as the target table, and then select the appropriate key on the UDX table as defined in the CD_ACCT dimension. The logic to populate UDX should be taken care in the mapping accordingly.



- On the **Physical** tab, select the optimization context.
- Select the target table (UDX_CD_ACCT) and select **IKM BI Direct Load** from the **Integration Knowledge Module** drop-down list.
- On the **Options** tab, set **DML_OPERATION** to **UPDATE** instead of **MERGE**.
- Unselect the **CREATE_TARG_TABLE** option since the UDX table is already created.



The custom mapping is successfully created. A custom package can be created using this mapping.

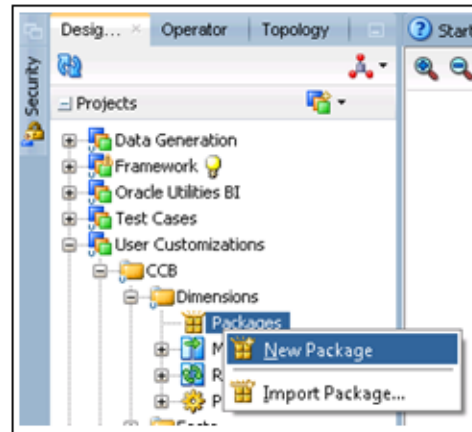
Creating CM Packages

To create a custom package for the existing custom mapping:

- Login to the Oracle Data Integrator client.
- Navigate to **Designer > User Customizations > <product_name> > Dimension > Packages**.

In this example, 'CCB' is the product name.

3. Right-click **Packages** and select **New Package** from the menu.

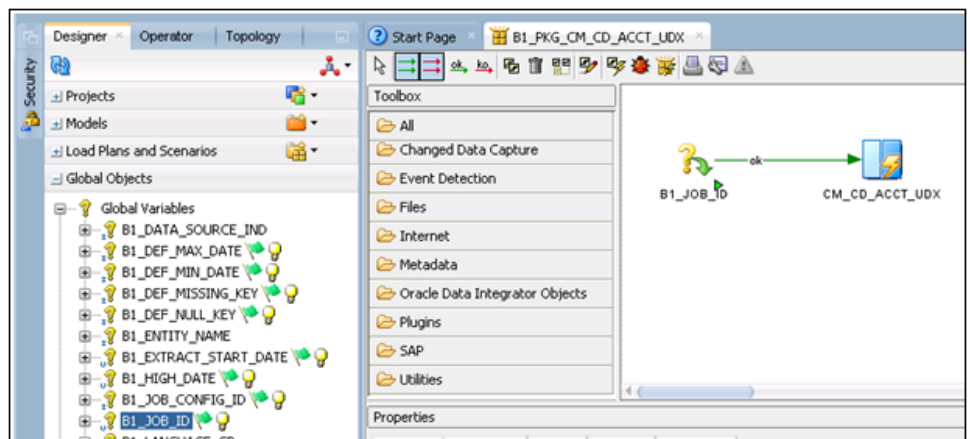


4. In the **Package Editor** window, enter the name of UDX in the **Name** field

For example: B1_PKG_CM_CD_ACCT_UDX

Important! Note that “B1” in the UDX name is taken as an example. Ensure the package name does not start with “B1”.

5. Click the **Diagram** tab at the bottom of the editor.
6. From the **Global Objects** section, drag and drop the 'B1_JOB_ID' variable into the editor.
7. Change 'B1_JOB_ID' to declare the variable.
8. Drag and drop the CM mapping (existing mapping) into the editor and connect them in sequence.



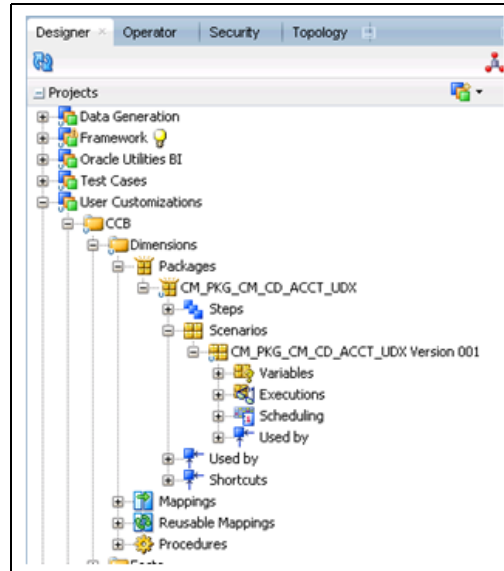
9. Click **Save** and close the package editor window.
10. Navigate to the **Packages** folder and expand it.

The new package is shown.

11. Right-click the package and select **Generate Scenario**.
12. Enter the scenario name and click **OK**.

13. Select the startup variables and click **OK**.
14. In the **Projects** section, navigate to **User Customizations > CCB > Dimensions > Packages**.
15. Expand the package created.

The scenario object created is shown.

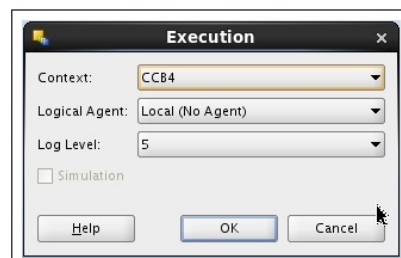


Resetting Dimensions

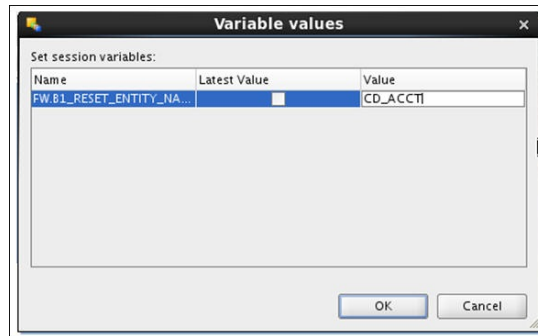
Since the dimension is already loaded, reset it to the empty state before reloading it with customization in place.

To reset the dimensions:

1. Login to the Oracle Data Integrator client.
2. On the **Designer** tab, navigate to **Load Plans and Scenarios > Framework > B1_RESET_ENTITY**.
3. Right-click **B1_RESET_ENTITY** and select **Execute**.
4. On the **Execution** window, select **CCB4** as the **Context** and then click **OK**.



- On the **Variable values** window, enter 'CD_ACCT' as the entity name. Click **OK**.



- Navigate to Oracle Utilities Analytics Warehouse Administration to verify that the entity has been disabled.

Job Configuration Id	Source Product	Instance Number	Target Entity Name	Entity Active Flag	User Exit Procedure	Last Sync DateTime
164	Customer Care and Billing	4	CD_ACCT	N	CM_CD_ACCT_UDX	25-JUN-15

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

ACCT_KEY	SRC_ACCT_ID	DATA_LOAD_DTTM	DATA_SOURCE_IND	ACCT_INFO	UDF10_CD	UDF10_DESCR	UDF1_CD	UDF1_DESCR
1	0 ***	01-JAN-00	0 ***	***	None	***	None	***
2	-99 N/A	01-JAN-00	-99 N/A	N/A	N/A	N/A	N/A	N/A

- Click **Save** to save the configuration changes.

Configuring CM Scenarios

After resetting the dimension, configure the user extension procedure. Below are the steps to configure Account (CD_ACCT) dimension.

To configure the user extension procedure for the account dimension:

- Login to Oracle Utilities Analytics Warehouse Administration.
- On the **ETL Configuration** tab, click **Job Configuration**.
- Enter **CD_ACCT** and click **Go** to filter the data.
- Click the edit icon to edit the details for the product instance for which the UDX has to be populated. There are different jobs for the same entity for different product instances.
- On the **Maintain Job Configuration** page, enter the Custom Package Name (example: B1_PKG_CM_CD_ACCT_UDX) in the **User Exit Procedure** field.
- Ensure **Active Flag** is set to 'Yes'.
- Click **Save** to save the configuration changes.

Monitoring Job Execution

Now that the job is configured for customization and activated, monitor the job execution using the **Administration** user interface or using SQL Developer.

To monitor the job execution from the **Administration** user interface:

1. Login to Oracle Utilities Analytics Warehouse Administration.
2. On the **ETL Job Execution** tab, enter “CD_ACCT” and click **Go** to filter the data.

To see the latest execution, sort by the session end date.

Job Execution								
Source Product	Instance	Entity Name	Session	Status	Slice Start Date	Slice End Date	Session Start Date	Session End Date
Customer Care and Billing	1	CD_ACCT	3720602	Done	10-SEP-2014 11:02:22	12-SEP-2014 03:30:41	12-SEP-2014 05:03:51	12-SEP-2014 05:05:25
Customer Care and Billing	1	CD_ACCT	3004602	Done	01-JAN-2014 00:00:00	10-SEP-2014 11:02:22	11-SEP-2014 08:23:39	11-SEP-2014 08:24:35
Customer Care and Billing	1	CD_ACCT	2960602	Done	01-JAN-2013 00:00:00	01-JAN-2014 00:00:00	11-SEP-2014 08:22:27	11-SEP-2014 08:23:37

Alternatively, use SQL Developer to monitor the job execution:

1. Connect to the target database using SQL Developer.
2. Monitor the job executions for the account dimension using the below query:

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

Validating Data Load

To validate the data load into customized columns:

Note: The queries below are based on the illustrated example. They need to be modified as per the logic used in the UDX

1. Identify the rows in which ‘udf10_cd’ and ‘udf10_descr’ columns are populated. Run the below query:

```
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’. Run the below query:

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

Populating User Defined Foreign Keys

This section describes the steps to extend out-of-the-box facts with custom dimension. Create the custom dimension and load it. Customize the fact load to use the custom dimension and populate the custom dimension key.

Important! Before performing these tasks, complete the steps mentioned in the [Custom Dimensions](#) section.

The section includes the following:

- [Creating CM Views](#)
- [Creating CM Mappings](#)
- [Creating CM Packages](#)
- [Configuring CM Scenarios](#)

Creating CM Views

To create a mapping used to wrap the existing custom dimension:

1. In the Oracle Data Integrator client, navigate to the **Customization** project.
2. Right-click **Mappings** and click **New Mapping**.
3. Enter “CM_D_ARREARS_UDDX1_VW” in the Name field.

“CM_D_ARREARS_UDDX1_VW” is taken as example. Replace it with the custom name.

4. Enter “Override out of the box UDDX1 view for arrears fact” in **Description**.
5. Click the **Mapping** tab at the bottom of the page to go to the **Edit Mapping** page.
6. On the left pane, navigate to **Models > Customizations > UDX Dimension**.
7. Select and drag the CM_D_ARREARS_UDDX1 custom dimension into the **Logical Tab** section.
8. In **Property Inspector** window, modify the **Name** to ‘UDDX1’.
9. On the left pane, navigate to **Models > Oracle Utilities BI > {Product Flag} > Dimensions**.

The naming convention of the UDDX view is **B1_D_<FACT NAME>_UDDX1_VW** to populate the **UDD1_KEY** of the fact.

10. Select the **View**, **Drag**, and **Drop** in the **Logical** tab section.

For example: If the fact name is **CF_ARREARS**, 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**.

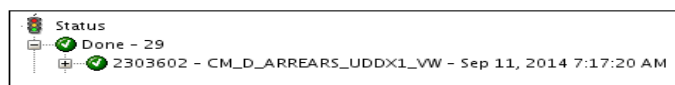
11. Map the **Target** columns with Custom dimension.

Note for Type I dimensions! Use 01-Jan-1900 as EFF_START_DTTM and 01-Jan-4000 as EFF_END_DTTM.

12. Navigate to **Physical** tab.
13. Click the target table.

14. In **Properties Inspector** in Integration Knowledge Module, select “IKM BI View Generation” from the **IKM Selector** drop-down list. Do not modify the remaining fields.
15. Click **Save** to save the changes.
16. Execute the mapping and go to the **Operator** to view the status.

The job executes successfully and the view is created.



17. 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 Mappings

This section describes the process of extending CF_ARREARS for User Defined Foreign Key.

Note: In this example, assume ‘CCB1’ to be the context defined for Oracle Utilities Customer Care and Billing source attached to Oracle Utilities Analytics Warehouse.

To create a CM mapping:

1. Login to the Oracle Data Integrator client.
2. On the **Designer** tab, navigate to **Projects > User Customizations > <product_name> > Facts > Mapping**.

In this example, 'CCB' is the product.

3. Right-click **Mapping** and select **New Mapping** from the menu.
4. In the **New Mapping** window, enter the name of UDX in the **Name** field. For example: CM_CF_ARREARS_UDX
5. Unselect the **Create Empty Dataset** checkbox.
6. Click the **Logical** tab to view the table structure.
7. From the **Models** section, drag and drop the UDX and replication tables in the designer pane.
8. Join the UDX and replication table.
9. Drag and drop the UDX_CF_ARREARS target data store.
10. Select UDX as the target table and select the appropriate key on the UDX table as defined in the CF_ARREARS fact. The logic to populate UDX should be taken care in the mapping accordingly.
11. On the **Physical** tab, select the optimization context.
12. Select the target table (UDX_CF_ARREARS) and select **IKM BI Direct Load** from the **Integration Knowledge Module** drop-down list.
13. On the **Options** tab, set **DML_OPERATION** to **UPDATE** instead of **MERGE**.

14. Unselect the **CREATE_TARG_TABLE** option since the UDX table is already created.

The custom mapping is successfully created. A custom package can be created using this mapping. For instructions, refer to the [Creating CM Packages](#) section.

Creating CM Packages

To create a custom package for the existing custom mapping:

1. Login to the Oracle Data Integrator client.
2. Navigate to **Designer > User Customizations > <product_name> > Fact > Packages**.

In this example, 'CCB' is the product name.

3. Right-click **Packages** and select **New Package** from the menu.
4. In the **Package Editor** window, enter the name of UDX in the **Name** field.

Example: CM_PKG_CM_CF_ARREARS

5. Click the **Diagram** tab at the bottom of the editor.
6. From the **Global Objects** section, drag and drop the 'B1_JOB_ID' variable into the editor.
7. Change 'B1_JOB_ID' to declare the variable.
8. Drag and drop the CM mapping (existing mapping) into the editor and connect them in sequence.
9. Click **Save** and close the package editor window.
10. Navigate to the **Packages** folder and expand it. The new package is shown.
11. Right-click the package and select **Generate Scenario**.
12. Enter the scenario name and click **OK**.
13. Select the startup variables and click **OK**.
14. In the **Projects** section, navigate to **User Customizations > CCB > Fact > Packages**.
15. Expand the package created.

The scenario object created is shown.

Configuring CM Scenarios

To configure the user extension procedure for a fact:

1. Login to the Oracle Utilities Analytics Administration user interface.
2. Click the **ETL Configuration** tab and click **Job Configuration**.
3. Enter the procedure name in the **User Exit Procedure** field and click **Go** to filter the data.

For example: CM_CF_ARREARS_UDX

Job Configuration							
Job Configuration Id	Source Product	Instance Number	Target Entity Name	Entity Active Flag	User Exit Procedure	Last Sync Date/Time	
217	Customer Care and Billing	4	CF_ARREARS	N	-	-	1 - 1

- Click the edit icon to edit the details.
- Set the **User Exit Procedure** (for example: CM_CF_ARREARS_UDX) and click **Save**.

Maintain Job Configuration

Cancel Delete Save

Job Configuration Id 91

Source Product * Customer Care and Billing

Instance Number * 1

Target Entity * 139

Scheduling Parameters

Entity Active Flag * Yes

Slice Start Date/Time * 01-Jan-2010 00:00:00
Note : Specify in the following format 'DD-MON-YYYY HH24:MI:SS'

Initialize Flag * Yes

Execution Sequence

Last Sync Date/Time 01-SEP-14

Customization Attributes

Override ODI Package Name

User Exit Procedure CM_CF_ARREARS_UDX

- Set the **Entity Active Flag** to **Yes** to enable the job.
- 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 is joined to a fact using a primary key to foreign key join. However, the dimensions 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 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:

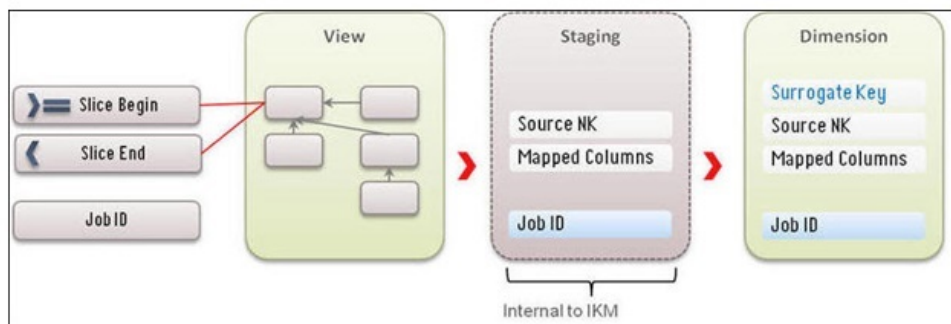
- Provides a direct and intuitive mapping between the business entities analyzed by the end users and schema design.
- Provides highly-optimized performance for the typical star queries.
- Widely supported by a large number of business intelligence tools, which may anticipate or even require that the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

The star schemas are used for both simple data marts, as well as very large data warehouses. After the model is designed, Oracle Data Integrator can be used to create the mappings and package to load the data into the star schema.

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

Custom Dimensions

A custom dimension is created in the database and populated using the pattern illustrated in the figure below.



This section provides the steps required to create a custom dimension and load data into it.

Note: The following steps are explained using the Arrears table (CM_D_ARREARS) from Customer Care and Billing as an example.

1. [Creating Dimension Table](#)
2. [Importing Dimension into Model](#)
3. [Importing Replicated Table into Replication Model](#)
4. [Creating Replication Key View in Dimension Model](#)
5. [Creating Mapping for Key Views in Dimension Model](#)
6. [Creating Loading Views in Dimension Model](#)
7. [Creating Mapping for Loading Views](#)
8. [Creating Package for Loading Views](#)
9. [Creating Staging Table in the Dimension Model](#)

10. [Creating Mapping in Dimension Model](#)
11. [Creating Package in Dimension Model](#)
12. [Configuring Entities in Dimension Model](#)
13. [Configuring Jobs in Dimension Model](#)
14. [Monitoring Job Execution](#)
15. [Validating the Data Loaded](#)

Creating Dimension Table

This section describes the procedure to create a Type-II slowly changing dimension. The dimension should have a primary key. In this example, it is the surrogate key column and a sequence is used to generate the values for this key. A Type II dimension should have a unique key comprising a column from source, the data source indicator, effective start timestamp, and effective end timestamp.

To create a dimension table:

1. Connect to the database using SQL Developer.
2. Run the script below to create the dimension table in the target schema:

```
create table dwadm.cm_d_arrears_uddx1
(
  arrears_uddx1_key      number(10)
, uddx1_cd               varchar2(30)
, attribute1            varchar2(60)
, attribute2            varchar2(60)
, attribute3            varchar2(60)
, attribute4            varchar2(60)
, attribute5            varchar2(60)
, data_source_ind       number(6)
, eff_start_dttm        date
, eff_end_dttm          date
, job_nbr               numeric(15)
, update_dttm           date
, primary key           (arrears_uddx1_key)
);
```

3. Run the script below to create the unique composite key for the Type II dimension:

```
create unique index dwadm.cm_d_arrears_uddx1_uk
on dwadm.cm_d_arrears_uddx1(uddx1_cd
, eff_start_dttm
, eff_end_dttm
, data_source_ind);
```

4. Create the sequence used to generate the surrogate key values.

```
create sequence dwadm.cm_d_arrears_uddx1_seq
start with 1
increment by 1;
```

5. Insert a row for the default 0 key record to handle nulls in the dimension foreign keys.

```
insert into dwadm.cm_d_arrears_uddx1
(
  arrears_uddx1_key
, uddx1_cd
```

```

,attribute1
,attribute2
,attribute3
,attribute4
,attribute5
,data_source_ind
,eff_start_dttm
,eff_end_dttm
,job_nbr
,update_dttm
)
values
(0
,'***'
,'***'
,'***'
,'***'
,'***'
,'***'
,'***'
,0
,to_date('01/01/2000','mm/dd/yyyy')
,to_date('01/01/4000','mm/dd/yyyy')
,0
,sysdate);

commit;

```

6. Insert a row for the default -99 key record for automatic reprocessing of Late Arriving Dimensions:

```

insert into dwadm.cm_d_arrears_uddx1
(
arrears_uddx1_key
,uddx1_cd
,attribute1
,attribute2
,attribute3
,attribute4
,attribute5
,data_source_ind
,eff_start_dttm
,eff_end_dttm
,job_nbr
,update_dttm
)
values
(-99
,'N/A'
,'N/A'
,'N/A'
,'N/A'
,'N/A'
,'N/A'
,-99
,to_date('01/01/2000','mm/dd/yyyy')
,to_date('01/01/4000','mm/dd/yyyy')
,-99
,sysdate);

commit;

```


Importing Dimension into Model

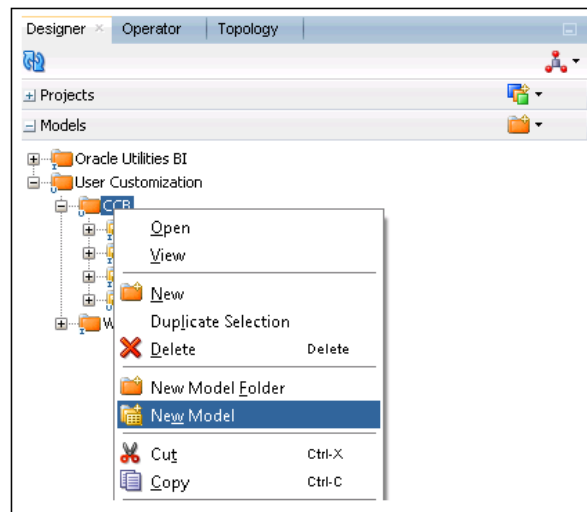
After the custom dimension is created in the database, import the dimension in the custom model folder created for customization.

1. Login to the Oracle Data Integrator client.
2. On the **Designer** tab, navigate to the **Models** section in ODI.
3. Navigate to the **User Customization** folder, and then to the folder with the product name for which the customization is done.

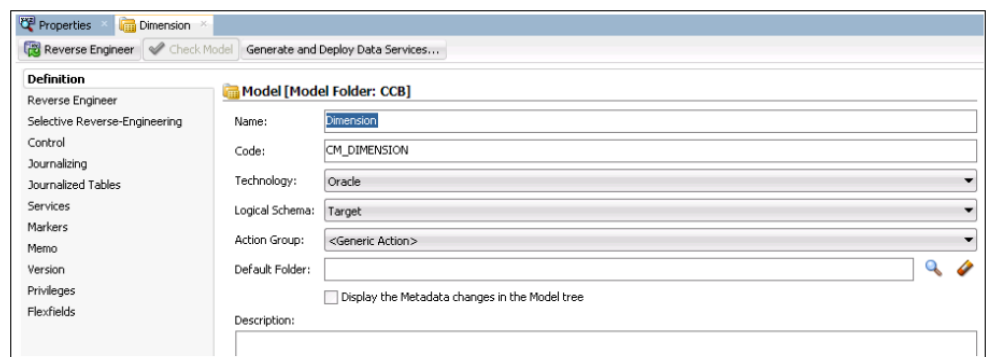
In this example, it is 'CCB'.

4. Right-click on the product folder and select **New Model**.

The **New Model** window opens.

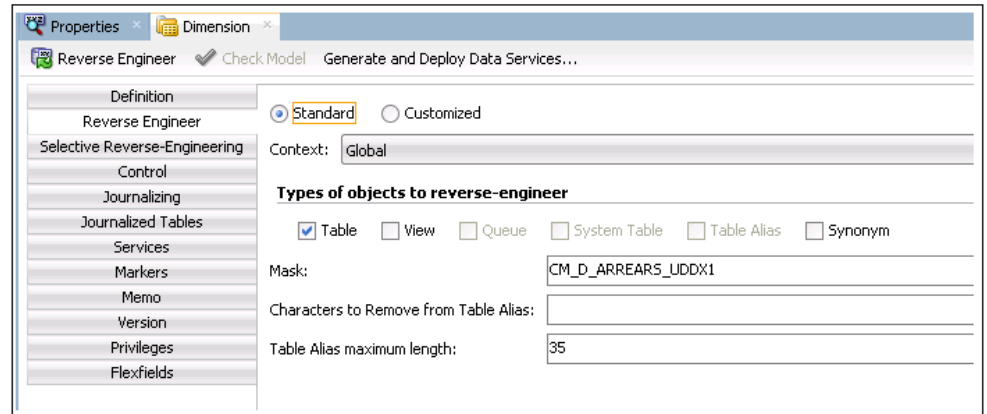


5. In the **Name** field, enter **Dimension**.
6. Specify the code.
7. Select the **Technology** as "Oracle" and **Logical Schema** as "Target".
8. Click **Save** to save the model.



9. Right-click and open the dimension model.
10. Navigate to the **Reverse Engineer** tab.
11. In the **Types of objects to reverse-engineer** section, select **Table**.
12. Enter CM_D_ARREARS_UDDX1 in the **Mask** field.

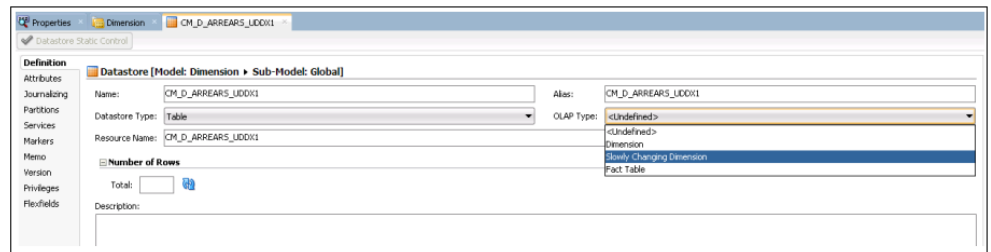
- Clear the **Characters to Remove from Table Alias** field.



- Click **Reverse Engineer**. The dimension table is reversed in the model.

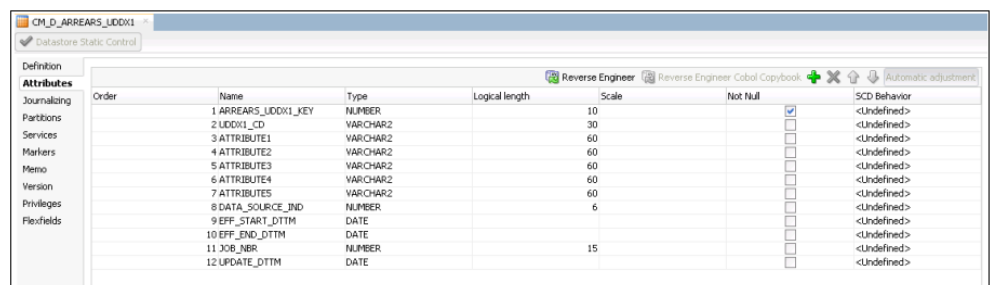
Once the dimension has been imported into the model, set its properties.

- Expand the **Dimension** model and double-click it to open the editor window.
- In the **Definition** tab, from the **OLAP Type** drop-down list, select **Slowly Changing Dimension**.



- Save the changes and navigate to the **Attributes** section of the data store.

Change the SCD behavior for all columns.

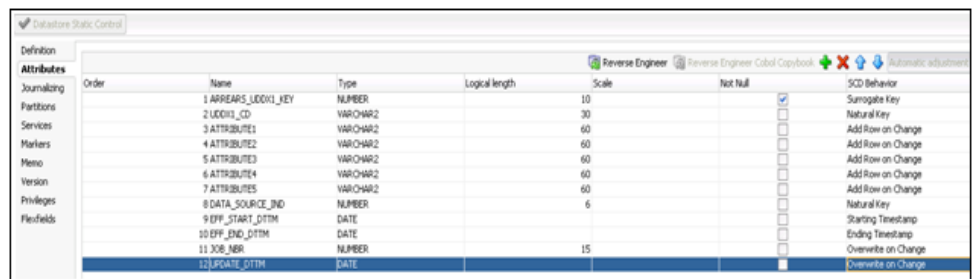


- For each of the attributes, set the **SCD Behavior** as shown below:

Attribute	SCD Behavior
ARREARS_UDDX1_KEY	Surrogate Key
UDDX1_CD	Natural Key
DATA_SOURCE_IND	Natural Key
EFF_START_DTTM	Starting Timestamp

Attribute	SCD Behavior
EFF_END_DTTM	Ending Timestamp
ATTRIBUTE1	Add Row on Change
ATTRIBUTE2	Add Row on Change
ATTRIBUTE3	Add Row on Change
ATTRIBUTE4	Add Row on Change
ATTRIBUTE5	Add Row on Change
JOB_NBR	Overwrite On Change
UPDATE_DTTM	Overwrite On Change

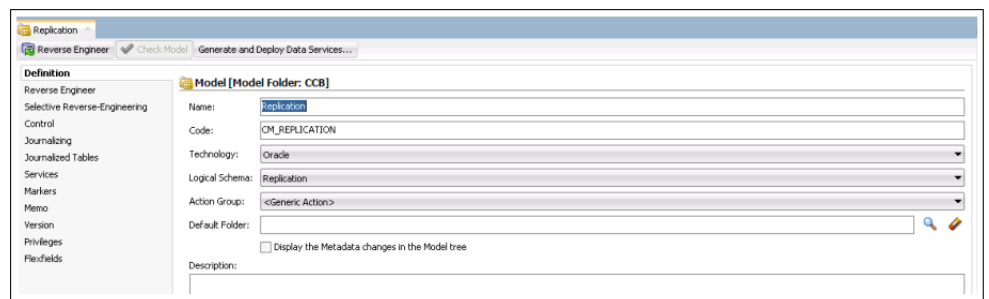
The figure below illustrates the **Attributes** section.



Importing Replicated Table into Replication Model

To import the replicated table into model:

1. Login to Oracle Data Integrator client.
2. On the **Designer** tab, navigate to the **Models** section in ODI.
3. Navigate to the **User Customization** folder, and then to the folder with the product name which is customized.
4. Right-click the product folder and select **New Model**.
The **New Model** window opens.
5. In the **Name** field, enter “Replication”.



6. Click the **Reverse Engineer** tab and select “CCB7” in the **Context** field. Enter “%CI_ACCT_CHAR” in the **Mask** field.

7. Save the model and click **Reverse Engineer**.

The reverse engineering action is executed.

Creating Replication Key View in Dimension Model

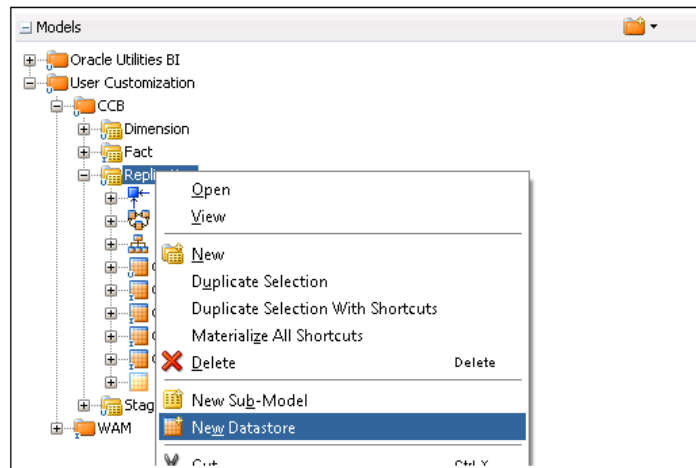
A key view is created for the dimension so that the incremental data for the fact can be filtered based on the key view. The key view should comprise the natural key of the dimension and the JRN_SLICING_TS column that stores the JRN_SLICING_TS column values from the driving tables that are used to create the view.

To create a replication key view in model for the dimension:

1. Login to the Oracle Data Integrator client.
2. Navigate to **Models > User Customizations**, and then navigate to the relevant product folder.

For example: CCB

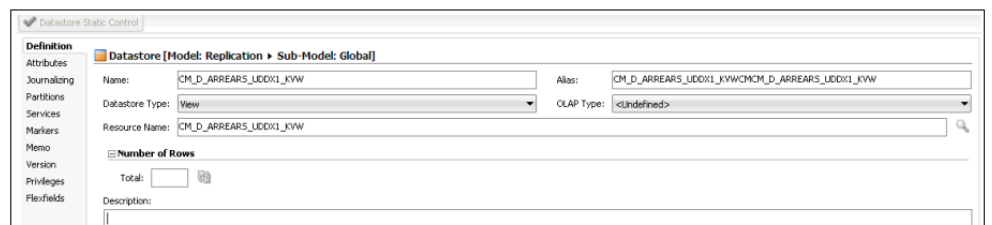
3. Right-click the **Replication** model and select **New Datastore**.



4. On the **Definition** tab, enter the name of the key view as CM_D_ARREARS_UDDX1_KVW.

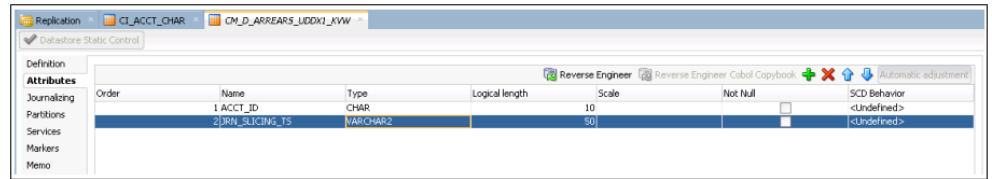
The naming convention of the view is “CM_” prefixed to entity name and suffixed by “_KVW”.

5. In the **Resource Name** field, enter the same name of the key view as in step 4.



6. Navigate to the **Attributes** tab.
7. Click on **+** on the right-hand corner and add the columns in the datastore.

The natural key of the dimension has to be present in the view. In addition to this, the JRN_SLICING_TS column has to be added.



8. Save and close the datastore.

Creating Mapping for Key Views in Dimension Model

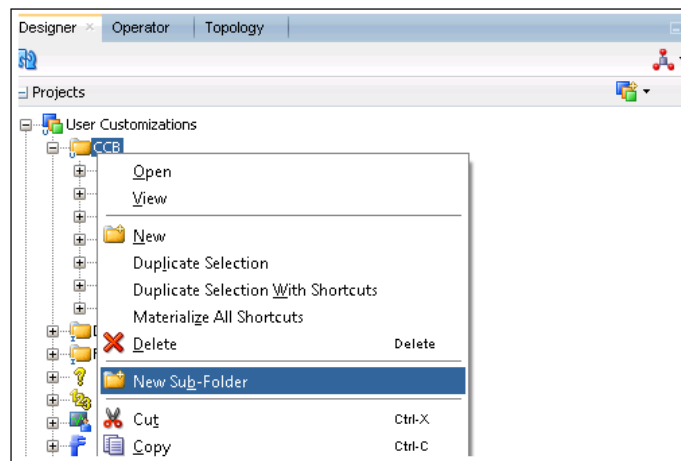
The key view for the dimension has to be generated in the Replication schema.

To create a mapping to generate a view for the key columns:

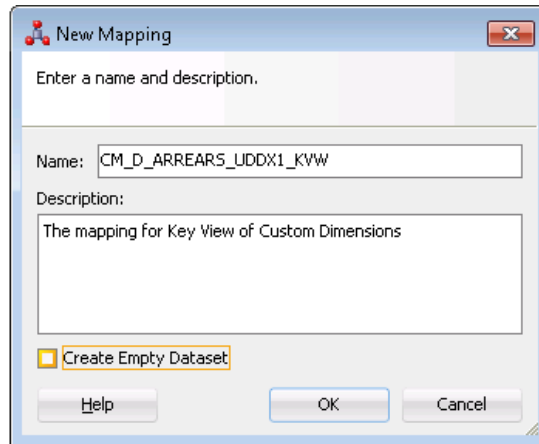
1. On the Oracle Data Integrator client, navigate to **User Customizations**, and then to the relevant product folder.

For example: CCB

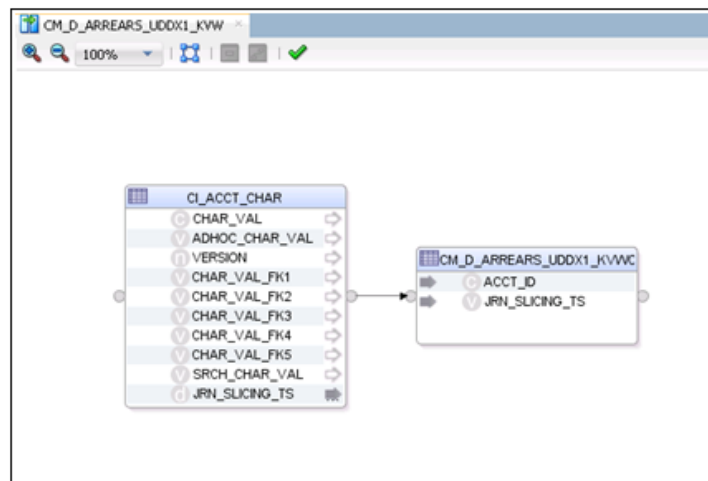
2. Create a new folder named **Replication** and expand it.



3. Right-click on the mapping and select **New Mapping**.
4. In the **New Mapping** window, enter “CM_D_ARREARS_UDDX1_KVW” as the name and uncheck the **Create Empty Dataset** option.



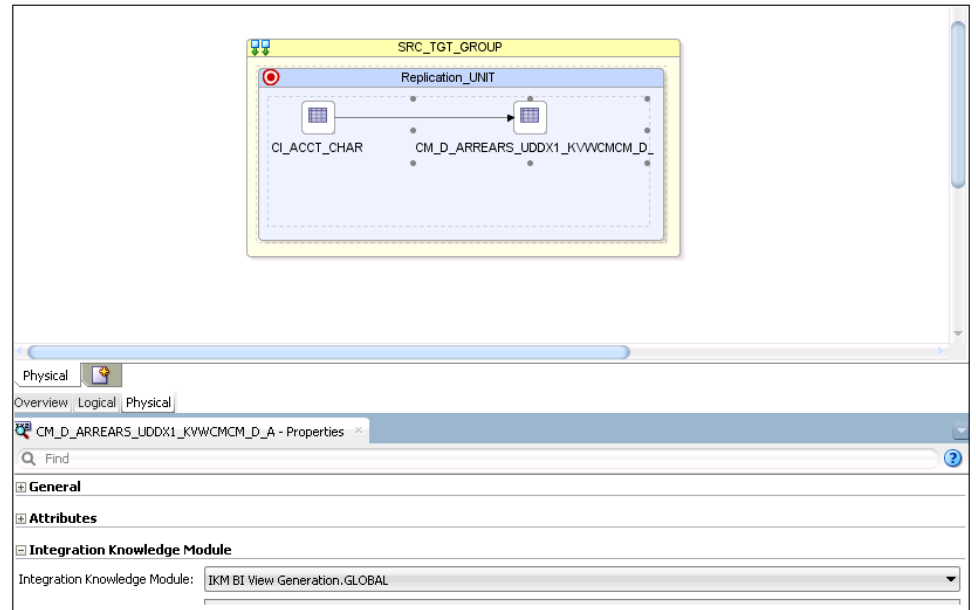
5. Navigate to the **Logical** tab of the mapping editor.
6. On the **Logical** tab, navigate to **Models > User Customizations > Replication**.
7. Drag the CM_D_ARREARS_UDDX1_KVW target replication key view and CI_ACCT_CHAR from the model and drop in the **Logical Design** pane.
8. Click the target view datastore.
9. Map the target view columns from the dragged source table.



10. If there are multiple driving tables, add new data flows in the mapping for every driving table using the SET component.

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')`.

11. Map the relevant column in the datasets for all the driving tables to target view.
12. Navigate to the **Physical Design** tab and set the Context in the **Properties** window.
13. Click the target datastore in the **Physical Design** pane.
14. In the **Integrated Knowledge Module** section, select IKM as “IKM BI View Generation”.



15. Click **Save** to save the mapping.
16. Generate a scenario for the mapping and execute the scenario in the context.

Creating Loading Views in Dimension Model

The loading view is created on top of the source replication tables. The view comprises all columns that are used to populate the dimension table and `IND_UPDATE` and `UPDATE_DTTM` columns.

To create a loading view in the dimension table:

1. On the Oracle Data Integrator client, navigate to **Models > User Customizations**, and then navigate to the relevant product folder.

For example: CCB

2. Right-click the replication model and select **New Datastore**.
3. On the **Definition** tab, enter 'CM_D_ARREARS_UDDX1_VW' as the name of the loading view.

The naming convention of the view is "CM_" prefixed to the entity name and suffixed by "_VW".

4. Specify the same view name again in the **Resource Name** field.
5. Navigate to **Attributes** tab.
6. Click **+** on the right corner and add columns in the datastore.

Add all columns that are required to populate the dimension table.

7. In addition to the above columns, add "IND_UPDATE" column with "CHAR(1)" as the data type.

Add "UPDATE_DTTM" with "DATE" as the data type.

Order	Name	Type	Logical length	Scale	N
1	ARREARS_UDDX1_KEY	NUMBER		10	
2	UDDX1_CD	VARCHAR2		30	
3	ATTRIBUTE1	VARCHAR2		60	
4	ATTRIBUTE2	VARCHAR2		60	
5	ATTRIBUTE3	VARCHAR2		60	
6	ATTRIBUTE4	VARCHAR2		60	
7	ATTRIBUTES	VARCHAR2		60	
8	DATA_SOURCE_IND	NUMBER		6	
9	EFF_START_DTTM	DATE			
10	EFF_END_DTTM	DATE			
11	IND_UPDATE	CHAR		1	
12	UPDATE_DTTM	DATE			

8. Save and close the datastore.

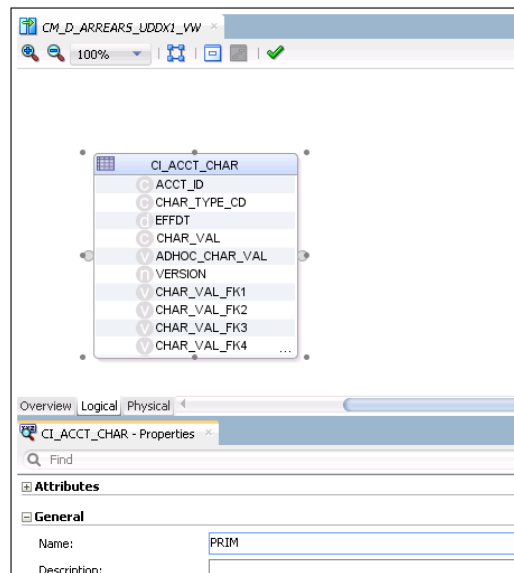
Creating Mapping for Loading Views

To create a mapping to generate the view used as source for the new dimension:

1. On the Oracle Data Integrator client, navigate to **User Customization > <product name> > Replication**.

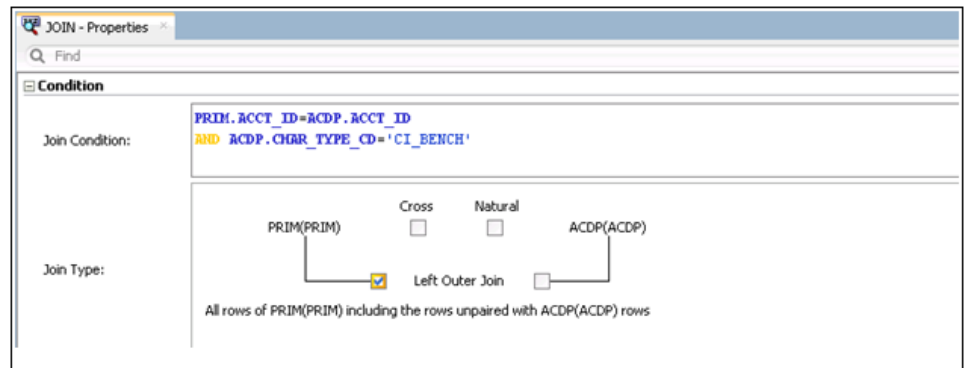
CCB is used as an example in this procedure.

2. Right-click the mapping and select **New Mapping**.
3. In the **New Mapping** window, enter “CM_D_ARREARS_UDDX1_VW” in the **Name** field and provide a description.
4. Navigate to **Model > User Customization > CCB > Replication**.
5. Drag the CI_ACCT_CHAR table into the **Logical Design** pane.
6. In the **Property Inspector**, change the alias name to PRIM.

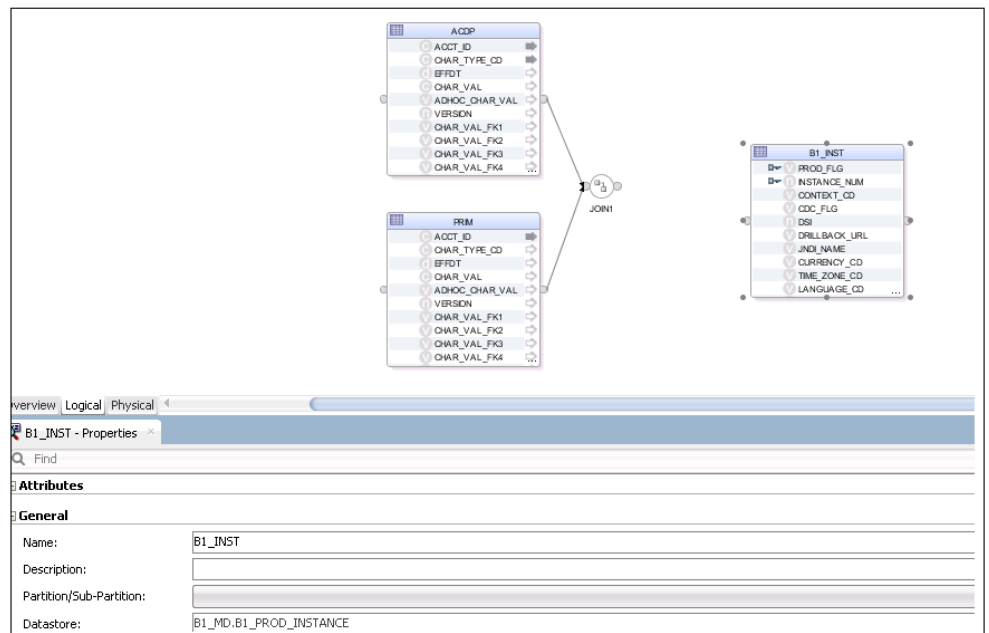


7. Similarly, drag the CI_ACCT_CHAR table again into the **Logical Design** pane. In the property inspector, change the alias name to ACDP.

8. From the **Component** palette, drag the Join component and place it in the **Logical Design** pane.
9. Join the two source tables to the Join component.
10. Edit the Join Condition in the **Properties** window to add the join condition.

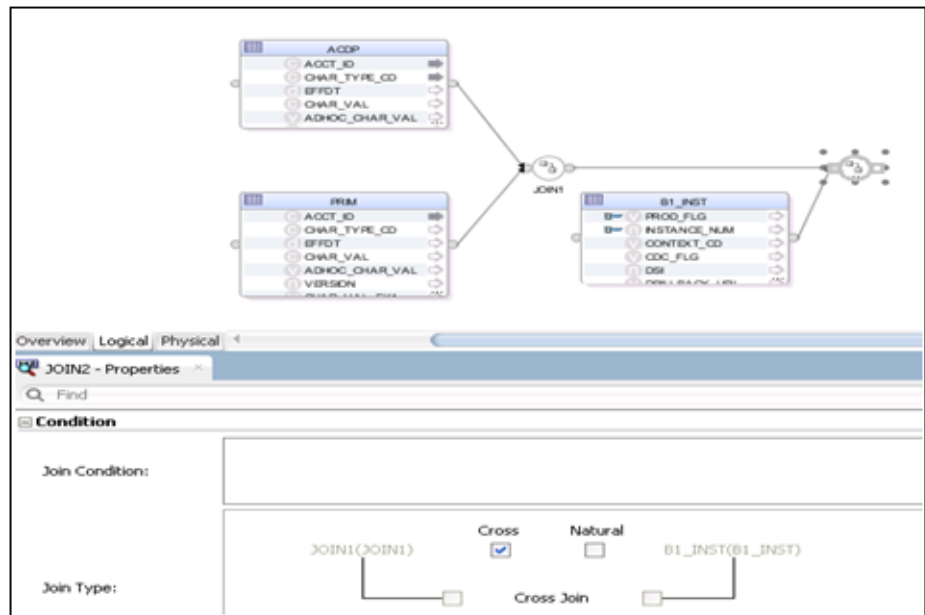


11. Select the **Left Outer Join** and **Use Ordered Join Syntax** check boxes.
12. Navigate to **Models > Framework > Metadata** and expand the model to reveal the tables.
13. Select and drag B1_PROD_INSTANCE into the source section of the mapping editor.
14. In the **Property Inspector**, change the alias to “INST”.

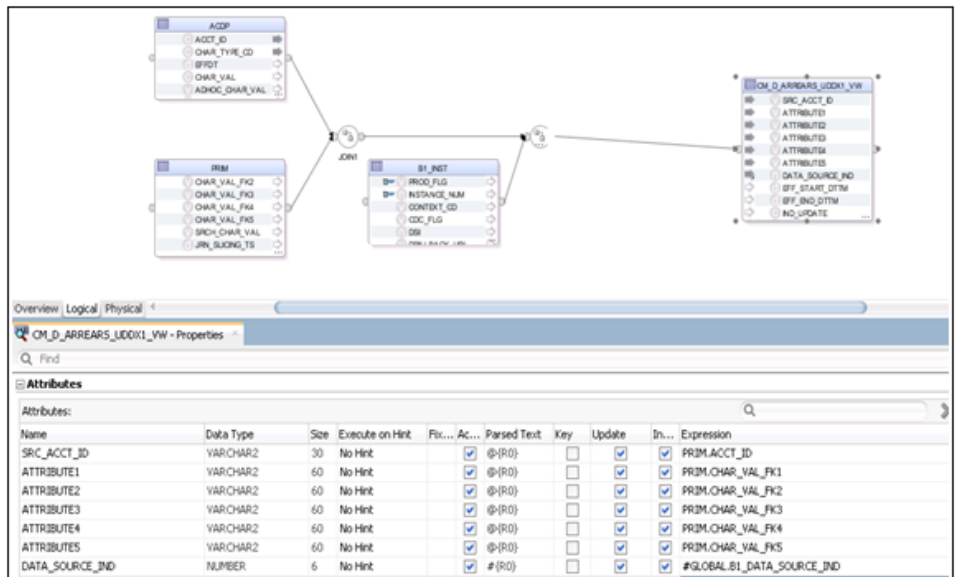


15. Drag another Join component from the **Component** palette and join the output of JOIN1 and B1_INST in this new join component.

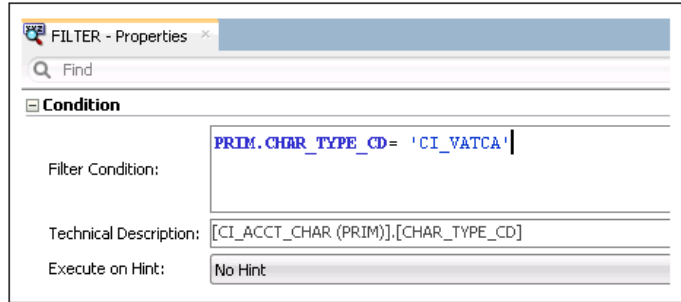
- Click the join created. In the Property Inspector, enter the following condition:
`INST.CONTEXT_CD = '<%=odiRef.getContext("CTX_CODE")%>'`



- Select the columns ACCT_ID from table with alias “PRIM” and map it to the UDDX1_CD column of the target view.
- Map the IND_UPDATE to the JRN_FLAG column from the PRIM alias.
- Map the DATA_SOURCE_IND from the Global Variable B1_DATA_SOURCE_IND.
- Map the other columns as shown.

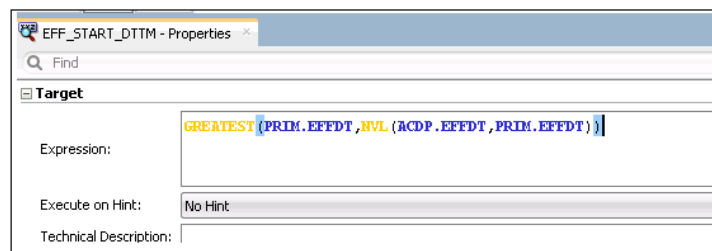


- Select the CHAR_TYPE_CD column from the table with alias “PRIM” and drag it out of the table. A new filter is created.
- In the Property Inspector, enter the condition = 'CI_VATCA'.



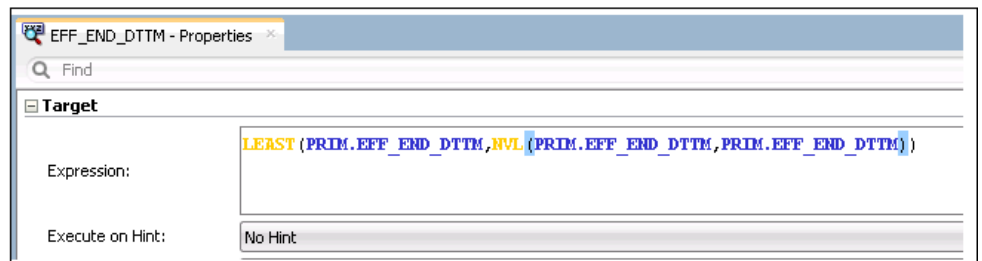
- Click `EFF_START_DTTM` and enter the following expression into the property inspector.

```
GREATEST (PRIM.EFFDT, NVL (ACDP.EFFDT, PRIM.EFFDT) )
```



- Click `EFF_END_DTTM` and replace the expression with the below:

```
LEAST (PRIM.EFF_END_DTTM, NVL (ACDP.EFF_END_DTTM, PRIM.EFF_END_DTTM) )
```



- Select and drag the `CONTEXT_CD` column from `INST` alias to create a filter on it.

- In the Property Inspector, enter the following condition:

```
INST.CONTEXT_CD = '<%=odiRef.getContext("CTX_CODE") %>'
```

- Navigate to **Physical Design** tab and set the **Context** in the **Properties** window.

- Select the target view. In the **Integration Knowledge Module** section, select “IKM BI View Generator” from the drop-down menu.

For the `VW_JOIN_MODE` option, enter “`RECURSIVE_JOINS`”.

- Save the changes and click **Execution**.

- Select `CCB7` for **Context** and click **OK**.

- On the **Operator** tab and expand **Date > Today** to view the status of the execution. A mapping to generate the view used as source for the new dimension is created.

Creating Package for Loading Views

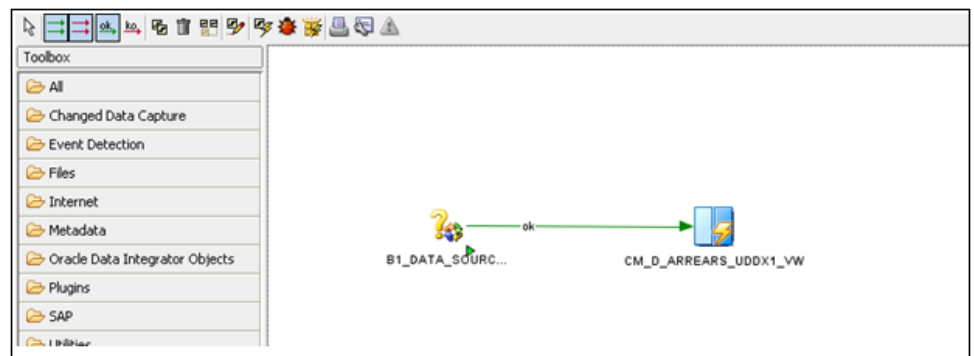
To create a package for the new dimension:

1. On the Oracle Data Integrator client, navigate to **Designer > User Customization > <product_name> > Replication > Package**.

In this example, 'CCB' is used as the product.

2. Right-click and select **New Package**.
3. In the **New Package** window, enter 'CM_PKG_D_ARREARS_UDDX1_VW'.
4. Click the **Diagram** tab at the bottom of the editor.

From the **Global Objects** section, drag B1_DATA_SOURCE_IND into the editor. Drag and drop the CM_D_ARREARS_UDDX1_VW mapping into the editor and connect them.



5. Click **Save** to save the changes and close the package editor.
6. Navigate to the packages folder and expand it. The new package is displayed.
7. Right-click the package and select **Generate Scenario**.
8. Enter the scenario name and then click **OK**.

A package for the new dimension is created.

Creating Staging Table in the Dimension Model

The definition of the staging table structure is under the **Staging** folder. The staging table structure is similar to the target table structure with the addition of a few columns. It should include IND_UPDATE in addition to the columns used in the mapping.

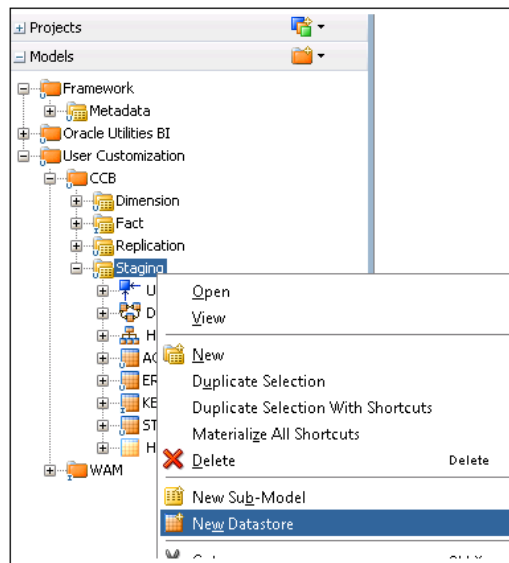
To create a staging table in the dimension model:

1. On the Oracle Data Integrator client, navigate to **Models > User Customization**.
2. Right-click and click **New DataStore** to create a new model.
3. Enter "Staging" in the **Name** field and then specify the code.
4. Select "Oracle" from the **Technology** and "Target" from the **Logical Schema** drop-down lists.
5. Click **Save** to save the model.

- Navigate to **User Customization > <product name> > Staging**.

CCB is used as an example.

- Right-click **Staging** and select **New Datastore**.



- On the **Definition** tab, enter “STG_CM_D_ARREARS_UDDX1” as the name of the staging table.

The naming convention of the staging table is “STG_” prefixed to the entity name.

- Enter “STG_#GLOBAL.B1_JOB_ID” in the **Resource Name** field.

Definition	
Datastore [Model: Staging > Sub-Model: Global]	
Attributes	
Journalizing	Name: <input type="text" value="STG_CM_D_ARREARS_UDDX1"/> Alias: <input type="text" value="STG_CM_D_ARREARS_UDDX1"/>
Partitions	Datastore Type: <input type="text" value="Table"/> OLAP Type: <input type="text" value="<Undefined>"/>
Services	Resource Name: <input type="text" value="STG_#GLOBAL.B1_JOB_ID"/>
Markers	
Memo	
Version	<input type="checkbox"/> Number of Rows
Privileges	Total: <input type="text"/>
Flexfields	Description: <input type="text"/>

The resource name should include the job ID variable so that the staging table is created with job execution number during run time so that there are no performance issues.

- Navigate to the **Attributes** tab.
- Click **+** to add columns to the datastore.

The columns in the dimensions should be present in the staging table. In addition to the above columns, add IND_UPDATE. The data type for IND_UPDATE should be “CHAR(1)”.

Order	Name	Type	Logical length	Scale	Not Null
1	ARREARS_UDDX1_KEY	NUMBER		10	
2	UDDX1_CD	VARCHAR2		30	
3	ATTRIBUTE1	VARCHAR2		60	
4	ATTRIBUTE2	VARCHAR2		60	
5	ATTRIBUTE3	VARCHAR2		60	
6	ATTRIBUTE4	VARCHAR2		60	
7	ATTRIBUTE5	VARCHAR2		60	
8	DATA_SOURCE_IND	NUMBER		6	
9	EFF_START_DTTM	DATE			
10	EFF_END_DTTM	DATE			
11	JOB_NBR	NUMBER		15	
12	UPDATE_DTTM	DATE			
13	IND_UPDATE	CHAR		1	

12. Click **Save** to save the datastore.
13. On the **Flexfields** tab, unselect the **Default** check box.
14. Enter “STG” in the **Value** column for the B1 Object Type record.
15. Enter the entity name (dimension name: CM_D_ARREARS_UDDX1) in the **Value** column for the B1 Target Entity Name record.
16. Click **Save** to save the datastore.

A staging table in the dimension model is created.

Creating Mapping in Dimension Model

To create a mapping to load data from the source view into the new dimension:

1. Create the following metadata entry.

Note: Create the metadata entry before creating the mapping.

```

INSERT INTO MDADM.B1_OBJECT_MAP
( OBJECT_MAP_ID
, PROD_FLG
, SOURCE_OBJECT_NAME
, TARGET_OBJECT_NAME
, SEQ
, UPD_DTTM
, UPD_USER
, OWNER_FLG
, OBJECT_TYPE_FLG)
VALUES ( mdadm.b1_object_map_seq.nextval
, 'CCB'
, 'CM_D_ARREARS_UDDX1_VW'
, 'CM_D_ARREARS_UDDX1'
, '1'
, sysdate
, 'CM'
, 'CM'
, 'PRVW')
Commit;

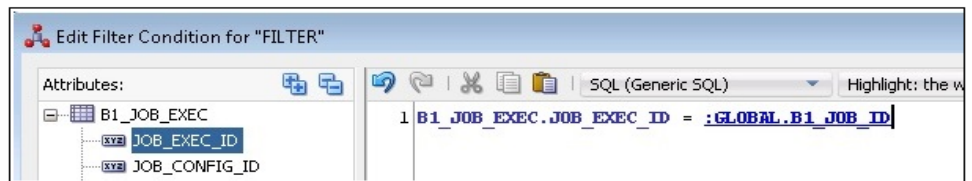
```

2. On the Oracle Data Integrator client, navigate to **User Customization > <product name> > Dimension**.

For example: CCB

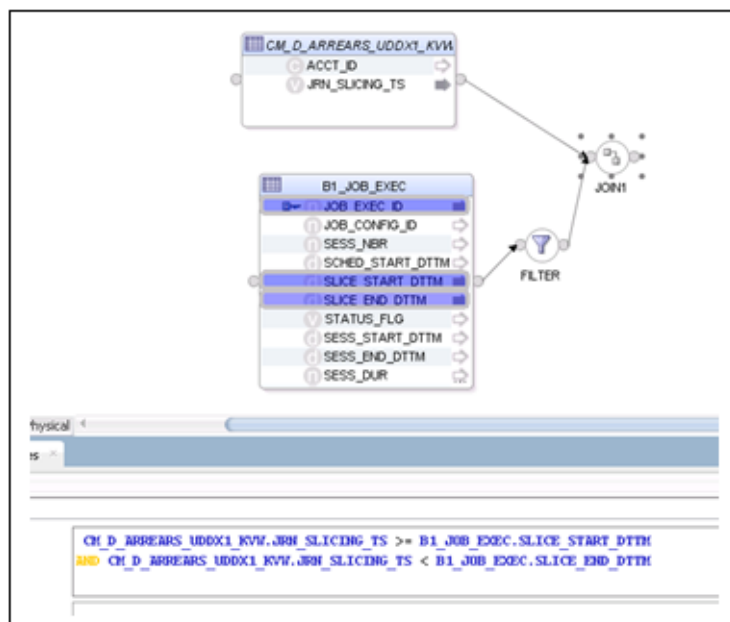
3. Right-click and select **New Mapping**.
4. In the **New Mapping** window, enter “CM_D_ARREARS_UDDX1” as **Name** and provide an appropriate description.
5. Unselect the **Create Empty Dataset** check box.
6. Navigate to **Models > User Customization > <product name>> Dimension**.
7. Expand the model and drag the dimension into the **Logical Design** pane.
8. Navigate to **Models > User Customization > <product name> >Replication**.
9. Expand the model and drag CM_D_ARREARS_UDDX1_VW (the loading view) and CM_D_ARREARS_UDDX1_KVW (the key view) into the **Logical Design** pane.
10. Navigate to **Model > Framework > Metadata**.
11. Drag the B1_JOB_EXEC table to the **Logical Design** pane.
12. Drag the FILTER operator from the **Component** palette and map the output of B1_JOB_EXEC to the filter component.
13. Add the filter condition mentioned below:

B1_JOB_EXEC.JOB_EXEC_ID = :GLOBAL.B1_JOB_ID

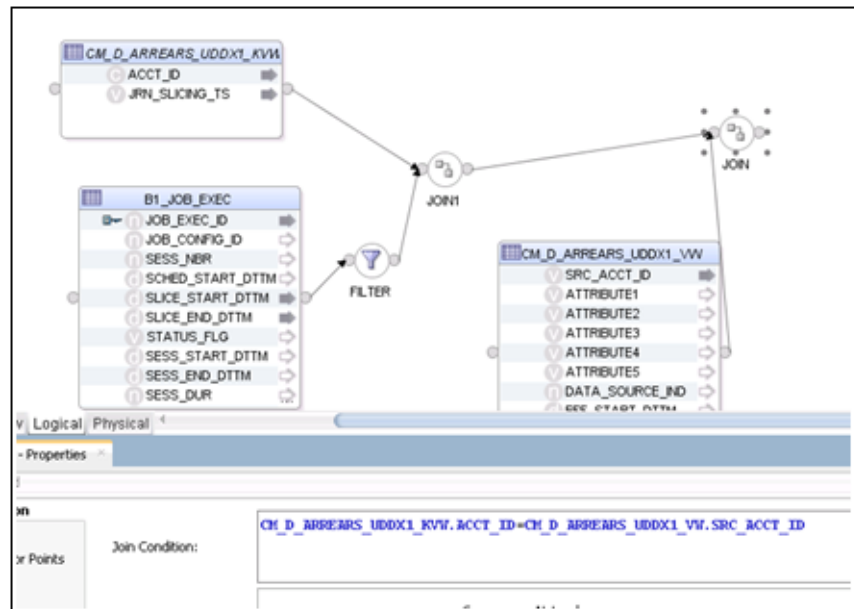


14. Drag the Join component from the **Component** palette and join B1_JOB_EXEC with the CM_D_ARREARS_UDDX1_KVW key view.

This join filters the incremental records only for that slicing period. Name the join as JOIN1.

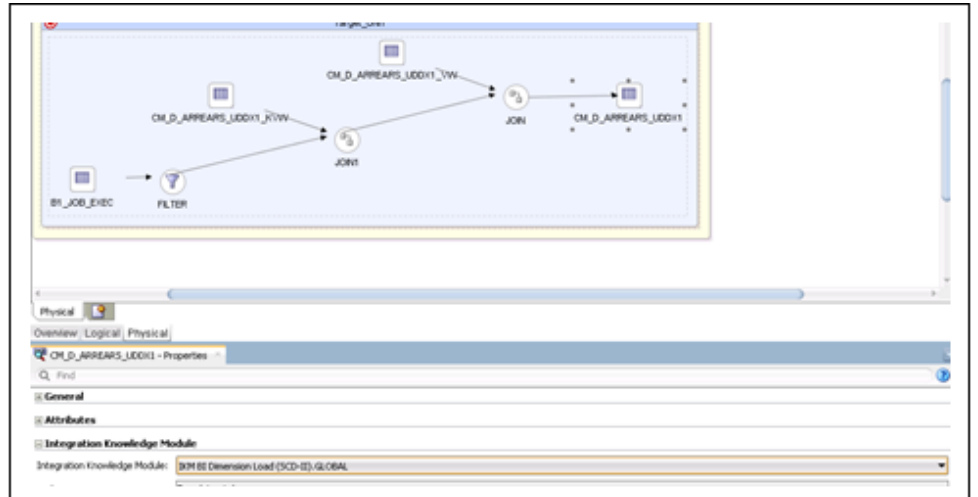


15. Drag another Join component from the **Component** palette. Join CM_D_ARREARS_UDDX1_VW (the loading view) with the output join JOIN1 (from step 14) on the ACCT_ID column.



16. Navigate to **Models > User Customization > CCB > Dimension**. Drag and drop the CM_D_ARREARS_UDDX1 target table on the **Logical Design** pane.
17. Click the ARREARS_UDDX1_KEY column in the target datastore. In the Property Inspector, enter the following code:


```
<%=odiRef.getInfo("DEST_SCHEMA")%>.CM_D_ARREARS_UDDX1_SEQ.NEXTVAL
```
18. Select the JOB_NBR column. In the Property Inspector, enter “#GLOBAL.B1_JOB_ID”.
19. Select the UPDATE_DTTM column. In the Property Inspector, enter “SYSDATE”.
20. Map the other columns from the loading view as appropriate.
21. On the **Physical Design** tab, click the target dimension table.
22. In the **Properties** window set the context.
23. Navigate to the **Integration Knowledge Module** section and select IKM BI Dimension Load (SCD – II).GLOBAL as the KM for mapping.



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

A mapping to load data from the source view into the new dimension is created.

Creating Package in Dimension Model

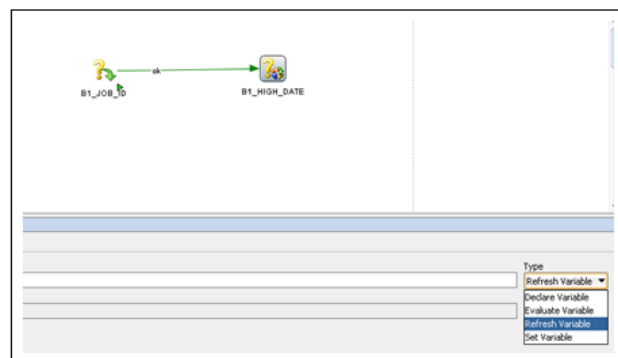
To create a package for the new dimension:

1. On the Oracle Data Integration client, navigate to **Designer > User Customizations > <product name> > Dimension > Packages**.

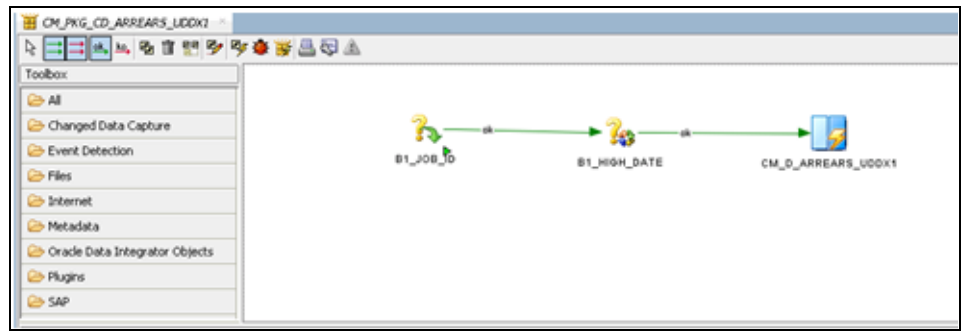
For example: CCB

2. Right-click **Packages** and select **New Package**.
3. In the **New Package** window, enter “CM_PKG_CD_ARREARS_UDDX1” as the package name.
4. Click the **Diagram** tab at the bottom of the editor.
5. From the **Global Objects** section, drag the B1_JOB_ID and B1_HIGH_DATE variables into the editor.

Change the data type for B1_JOB_ID to **Declare Variable** and that for B1_HIGH_DATE to **Refresh Variable**.



6. Drag and drop the CM_D_ARREARS_UDDX1 mapping into the editor and connect them



7. Click **Save** to save the changes and close the package editor window.
8. Navigate to the packages folder and expand it. The new package is displayed.
9. Right-click and select **Generate scenario**.
10. Enter the scenario name and click **OK**.
11. In the **Scenario Variables** window, select the startup variables. Unselect the **Startup Parameter** checkbox for B1_HIGH_DATE (it is a refresh variable) and click **OK**.
12. Expand the package. Under **Scenarios**, the new scenario generated is displayed.

Configuring Entities in Dimension Model

To configure a new entity for a custom dimension:

1. Login to the Oracle Utilities Analytics Warehouse Administration user mapping.
2. On the **ETL Configuration** tab, click **Target Entity**.

Target Entity							
Target Entity Id	Entity Name	Entity Type	Scheduling Time (HH:MI:SS)	Scheduling Interval (mins)	ODI Package Name	Staging Retention Days	Owner
136	CD_UOM	Dimension SCD1	00:00:00	0	B1_PKG_CD_UOM	7	Base Product
134	CD_UCOLPROC_STATUS	Dimension SCD1	00:00:00	0	B1_PKG_CD_UCOLPROC_STATUS	7	Base Product
133	CD_UCOLEVT_TYPE	Dimension SCD1	00:00:00	0	B1_PKG_CD_UCOLEVT_TYPE	7	Base Product
132	CD_TOU	Dimension SCD1	00:00:00	0	B1_PKG_CD_TOU	7	Base Product

3. Click **Add**. The **Maintain Target Entity** page is displayed where you can set up the job details.
4. Enter the appropriate values for the dimension.

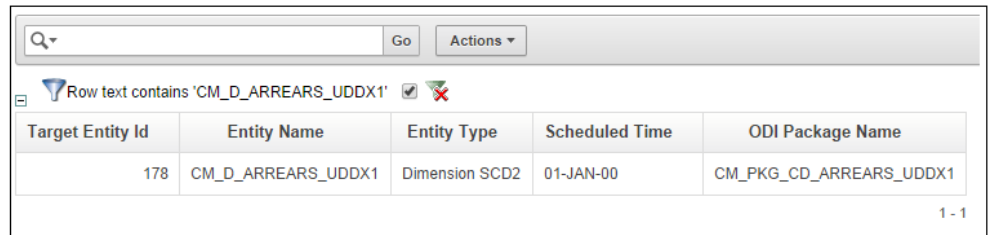
Configuring Jobs in Dimension Model

To configure a job for the custom dimension:

1. Login to Oracle Utilities Analytics Warehouse Administration user mapping.
2. On the **ETL Configuration** tab, click **Job Configuration**.

3. Click **Add**. The **Maintain Job Configuration** page is displayed where you can set up the job details.
4. Select “Customer Care and Billing” from the **Source Product** drop-down list.
5. Select ‘1’ from the **Instance Number** drop-down list.
6. Click the **Search** icon for the **Target Entity** field.

In the search window, enter “CM_D_ARREARS_UDDX1” and click **Go**.



The screenshot shows a search interface with a search bar containing 'CM_D_ARREARS_UDDX1' and a 'Go' button. Below the search bar, there is a filter icon and the text 'Row text contains 'CM_D_ARREARS_UDDX1''. Below this, a table displays the search results.

Target Entity Id	Entity Name	Entity Type	Scheduled Time	ODI Package Name
178	CM_D_ARREARS_UDDX1	Dimension SCD2	01-JAN-00	CM_PKG_CD_ARREARS_UDDX1

1 - 1

7. Click the **Target Entity ID** value.

The ID is populated on the **Maintain Job Configuration** page.

8. Set the **Slice Start Date/Time** as ‘01-Jan-2000’ or the extract date to which the source instance is configured.
9. Click **Add** to create the job configuration entry. The job can be enabled while saving the new entry.

Monitoring Job Execution

After the job is configured for customization and activated, use the Oracle Utilities Analytics Warehouse Administration or SQL Developer to monitor the job execution.

To monitor the job execution from Oracle Utilities Analytics Warehouse Administration:

1. Login to Oracle Utilities Analytics Warehouse Administration.
2. On the **ETL Job Execution** tab, enter “CM_D_ARREARS_UDDX1”.
3. Click **Go** to filter the data. The execution details are displayed.

Sort by the session end date to view the latest execution details.

Validating the Data Loaded

To validate that data is loaded into the custom dimension:

1. Connect to the database using SQL Developer.
2. Use the query below to view the data in the dimension:

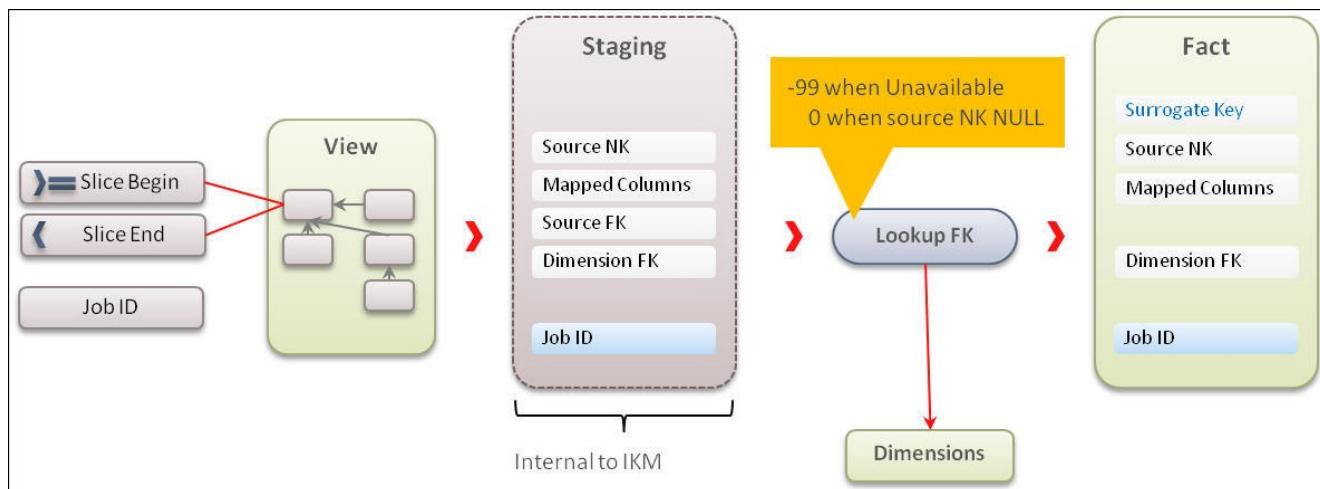
```
select *
from dwadm. CM_D_ARREARS_UDDX1;
```

3. Compare the data in the dimension with the data in the base view using the below query:

```
select *
from ccb1rep.cm_d_arrears_uddx1_vw;
```

Custom Facts

A custom fact is created in the database and populated using the pattern illustrated in the figure below:



The steps required to create a custom fact and load data into it are as follows:

1. [Creating Fact Tables](#)
2. [Importing Fact Tables into Model](#)
3. [Importing Replicated Tables into Fact Model](#)
4. [Creating Key Tables in Fact Model](#)
5. [Creating Mapping for Key Tables in Fact Model](#)
6. [Creating Loading Views in Fact Model](#)
7. [Creating Mapping to Loading Views for Fact Model](#)
8. [Creating Aggregate Tables in Fact Model](#)
9. [Creating Mapping to Load Aggregate Tables in Fact Model](#)
10. [Creating Staging Tables in Fact Model](#)
11. [Creating Error Tables in Fact Model](#)
12. [Creating Mapping to Load Facts](#)
13. [Creating Packages in Fact Model](#)
14. [Configuring Entities in Fact Model](#)
15. [Specifying Dependencies in Fact Model](#)
16. [Configuring Jobs in Fact Model](#)
17. [Monitoring Job Executions](#)

These steps use an example from Oracle Utilities Customer Care and Billing. The Bill segment Calculation (CM_CF_BSEG_CALC) custom fact is populated with the bill segments line calculation amount.

Each bill generated in Oracle Utilities Customer Care and Billing has multiple bill segments, and each bill segment has multiple calculations with different billing. The CM_CF_BSEG_CALC fact has three dimensions - Service Agreement, Premise, and Service Agreement Status.

The other details are:

- Measure = Calculated Amount for each bill segment's line
- Natural Key = Bill Segment, Bill Segment Header Sequence
- Source Table = Bill Segment (CI_BSEG), Bill Segment Calculation (CI_BSEG_CALC), Bill Segment Calculation Line (CI_BSEG_CALC_LN)

Creating Fact Tables

This section describes the procedure to create a fact table using an example.

The fact has a primary key, which is the surrogate key column. Use a sequence to generate the values for this key. The fact has a unique key comprising a column from source, the data source indicator. It includes the bill segment details calculation amount for each header in the bill generated in Oracle Utilities Customer Care and Billing.

To create a fact table:

1. Connect to the Oracle Utilities Analytics Warehouse database using SQL Developer.
2. Run the script below to create a fact table in the target schema:

```
create table dwadm.cm_cf_bseg_calc
(
  bseg_calc_key          number(15)
, bseg_id number(19)
, bseg_hdr_seqnumber(5)
, data_source_ind       number(6)
, bill_nbrnumber(30)
, bseg_cre_dttmdate
, sa_keynumber(15)
, prem_keynumber(15)
, bseg_stat_keynumber(15)
, currency_cdvarchar2(10)
, distribution_cd       varchar2(60)
, bseg_calc_amtnumber(15,2)
, job_nbrnumber(19)
, update_dttmdate
, primary key (bseg_calc_key)
);
```

3. Run the script below to create a unique composite key for the fact:

```
create unique index dwadm.cm_f_bseg_calc_uk
on dwadm. cm_cf_bseg_calc( bseg_id,bseg_hdr_seq data_source_ind);
```

4. Create the sequence used to generate the surrogate key values.

```
create sequence dwadm.cm_cf_bseg_calc_seq
start with 1
increment by 1;
```

Importing Fact Tables into Model

After the custom fact is created, import it into the custom model folder (created previously for customization).

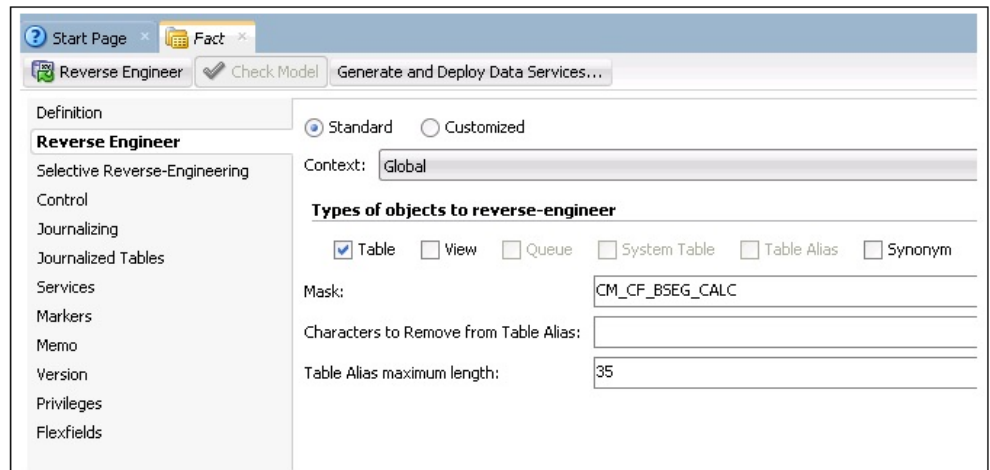
To import a fact into a model folder:

1. Login to the Oracle Data Integrator client.

- On the **Designer** tab, navigate to the **Models > User Customization > <product_name>**.

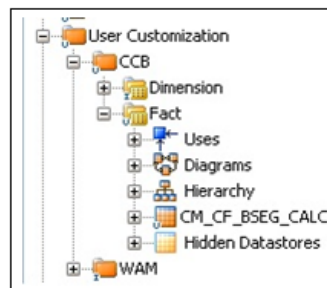
In this example, the product name is ‘CCB’.

- Right-click the product folder and select **Open Fact Model**. The **Open Fact Model** window is displayed.
- On the left pane, click **Reverse Engineer**.
- On the right pane, select the **Table** check box in the **Types of objects to reverse-engineer** section.
- Enter “CM_CF_BSEG_CALC” in the **Mask** field.
- Click **Reverse Engineer** at the top-left corner.



The fact table is reversed in the model.

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



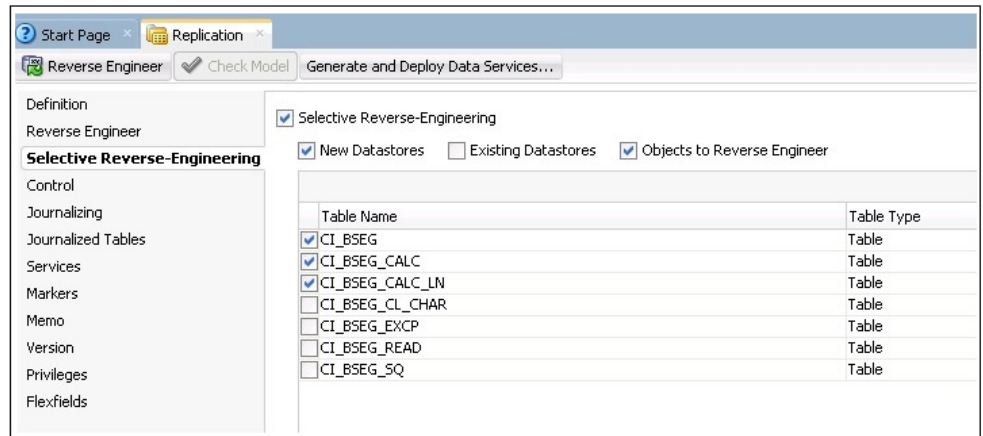
Importing Replicated Tables into Fact Model

To import replicated tables into the fact model:

- Login to the Oracle Data Integrator client.
- On the **Designer** tab, navigate to **Models > User Customization > <product name>** which has to be customized.

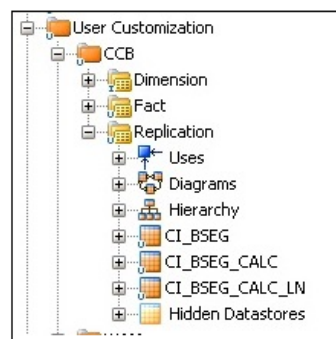
In this example, the product is ‘CCB’.

3. Right-click and select **Open Replication Model**. The **Open Replication Model** window is displayed.
4. On the left pane, click **Reverse Engineer**.
5. On the right pane, do the following:
 - a. Select the **Table** check box in the **Types of objects to reverse-engineer** section.
 - b. Enter “CM_CF_BSEG_CALC” in the **Mask** field.
 - c. Click **Save**.
6. On the left pane, click **Selective Reverse-Engineering**.
7. On the right pane, select the following tables required to load the fact:
 - CI_BSEG
 - CI_BSEG_CALC
 - CI_BSEG_CALC_LN
8. Click **Reverse Engineer** on the top-left corner.



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

The replication tables are reversed in the fact model.



Creating Key Tables in Fact Model

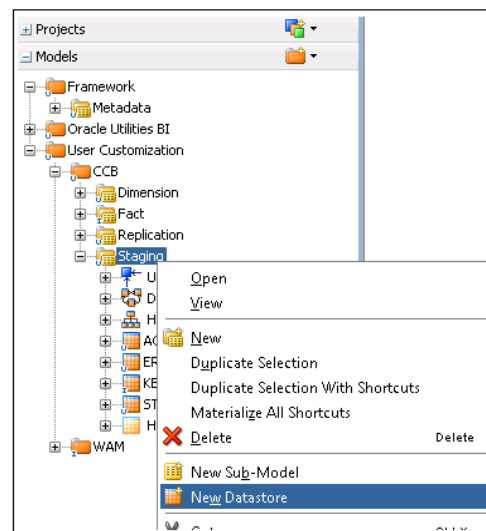
A key table is created to identify the natural key of the entity for incremental loading. It helps to identify specific records for processing instead of scanning the entire replication table.

Ensure the following are taken care while creating a key table in a fact model:

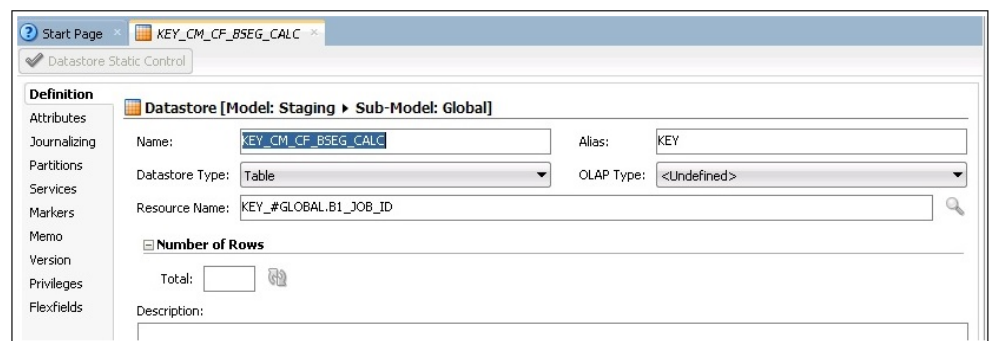
- The resource name of the table is changed to “KEY_#GLOBAL.B1_JOB_ID”.
- The table name can be “KEY_<FACT_NAME>”, but the resource name should be prefixed with the table type followed by the job number.
- Since the table is created at run time, the table name should be suffixed with the job number. It helps in parallel load of the data.
- The key table is created in the model and the flex field is set appropriately.

To create a key table in the fact model:

1. Login to the Oracle Data Integrator client.
2. On the **Designer** tab, navigate to **Models > User Customization > <product_name> > Staging**.
3. Right-click **Staging**, and then select **New Datastore** from the menu.



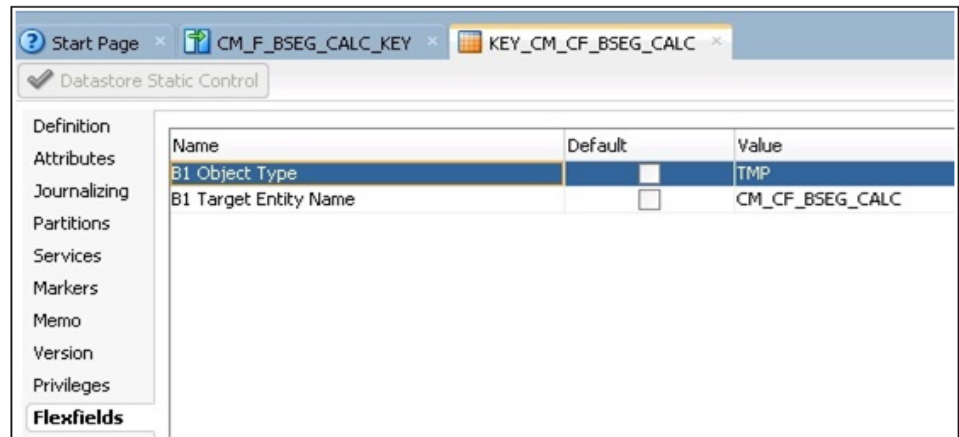
4. In the datastore editor, enter “KEY_CM_CF_BSEG_CALC” and “KEY_#GLOBAL.B1_JOB_ID” in the **Name** and **Resource Name** fields respectively.



5. Click **Flexfields** on the left pane.

- Unselect the **Default** check boxes for **B1 Object Type** and **B1 Target Entity Name** fields respectively.

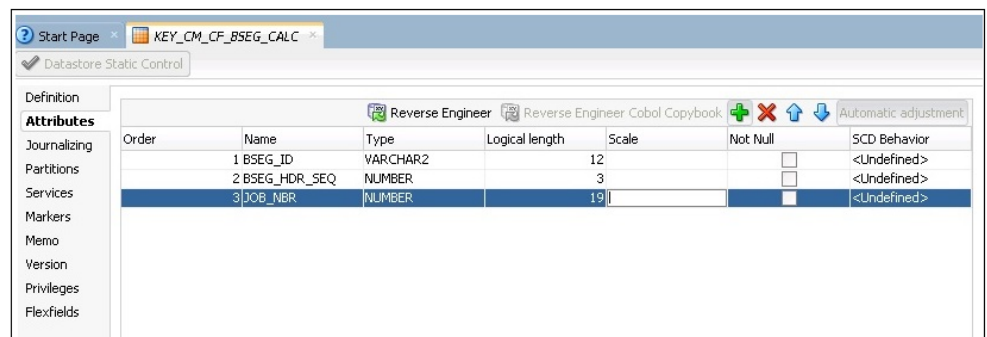
Enter “TMP” and “CM_CF_BSEG_CALC” in the **B1 Object Type** and **B1 Target Entity Name** fields respectively.



- Click the **Attributes** tab on the left pane.
- Click **+** to add columns to the datastore.

Add the columns that are part of the natural key of the fact. Add JOB_NBR in addition to the natural key.

Note: The data type and length of the columns should match to that of the fact table.



- Click **Save** to save the datastore.

Creating Mapping for Key Tables in Fact Model

The key table is created for a fact so that the incremental data for the fact can be filtered based on the key table. The driving tables from the replication schema are included, and columns are created as part of the natural key of the fact. The fact is generated in the Staging schema.

If there are multiple driving tables, the key table data is populated from all the driving tables. Using the “Union” option, distinct data is populated in the key table.

To create a mapping for loading the key table for the fact:

- Login to the Oracle Data Integrator client.

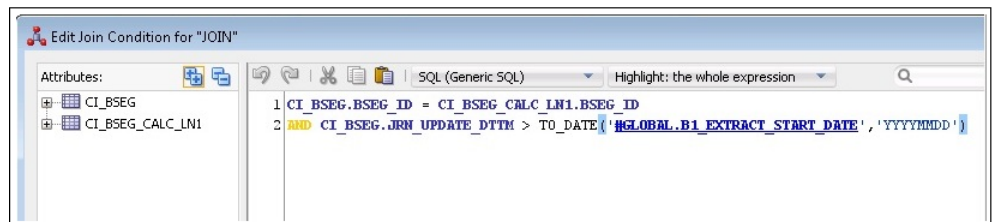
2. On the **Designer** tab, navigate to **Models > User Customization > <product_name> > Fact > Mapping**.

In this example, 'CCB' is the product.

3. Right-click **Mapping** and select **New Mapping** from the menu.
4. In the mapping editor, do the following:
 - a. Enter "CM_F_BSEG_CALC_KEY" in the **Name** field.
 - b. Unselect the **Create Empty Dataset** check box.
 - c. Click **OK**.
5. On the **Designer** tab, navigate to **Models > User Customization > CCB > Replication**.
6. Expand the model and drag the tables to the mapping editor.
7. Drag the CI_BSEG_CALC_LN and CI_BSEG driving tables to the mapping editor.
8. Drag the CI_BSEG_CALC_LN table again to the mapping editor.

Since the CI_BSEG driving table does not include the combination of natural key, the table should be joined with CI_BSEG_CALC_LN to get the natural key in the key table.

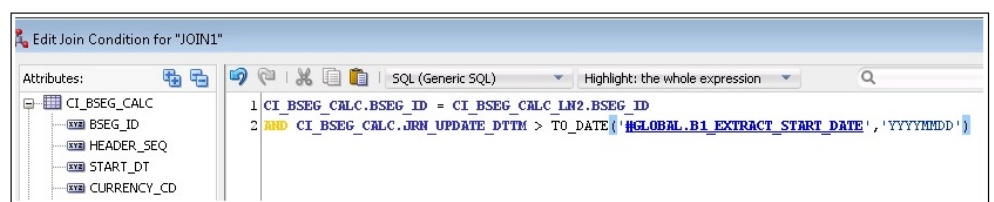
9. In the **Component** window, select the JOIN component and enter the name as "JOIN".
10. Map the CI_BSEG and CI_BSEG_CALC_LN tables as input to the join and specify the join condition in the expression.



11. Drag the CI_BSEG_CALC table to the mapping editor.
12. Drag the CI_BSEG_CALC_LN table again to the mapping editor.

Since CI_BSEG_CALC table does not have the combination of natural key of the fact, it should be joined with CI_BSEG_CALC_LN to get the natural key in the key table.

13. In the **Component** window, select the JOIN component and enter the name as "JOIN1".
14. Map the CI_BSEG_CALC and CI_BSEG_CALC_LN table as input to the "JOIN1" and specify the join condition in the expression.

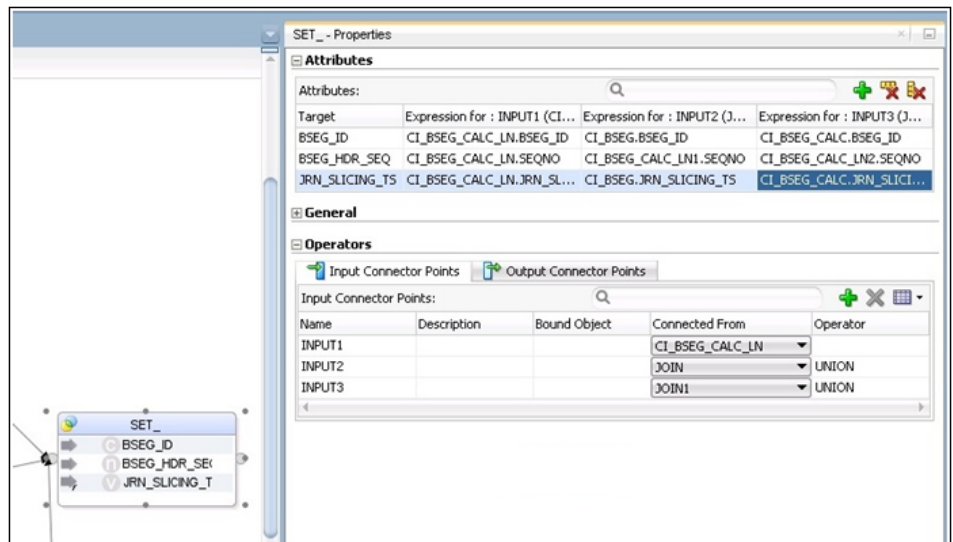


15. Select the components and the SET operator. Drag them to the mapping editor and name the set as “SET_”.
16. Click the SET component on the mapping editor. The **Properties** window is displayed.
17. Add one more input connections to the SET operator and map them to one of the sources.

For example: INPUT1 > CI_BSEG_CALC_LN, INPUT2 > JOIN, INPUT3 > JOIN1

18. On the left pane, click **Attributes**. Click + to add columns to the SET operator.
19. Add the natural key of the fact and “JRN_SLICING_TS” column to the SET operator.

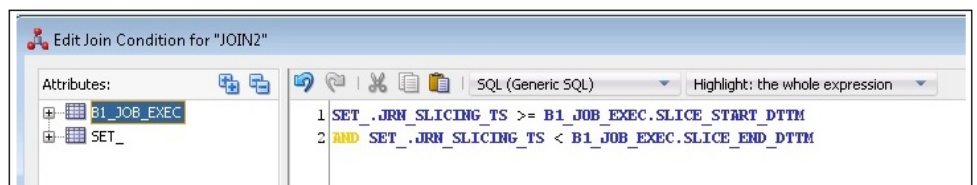
The column names are displayed. For each INPUT connection, there is an EXPRESSION, and the columns are mapped appropriately.



20. On the Designer, navigate to **Models > Framework > Metadata**.
21. Drag the B1_JOB_EXEC table to the mapping editor.
22. Drag the FILTER operator from the **Component** window and map the input of Filter to the output of B1_JOB_EXEC.
23. Add the filter condition as below:

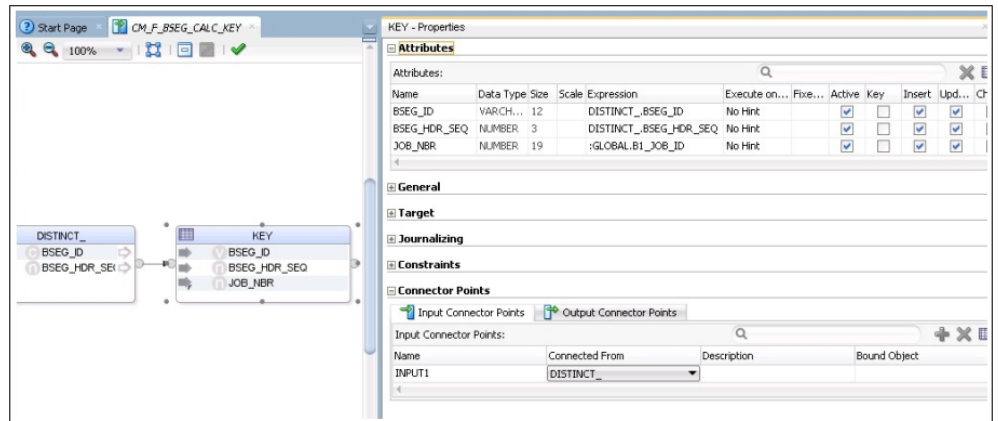

```
B1_JOB_EXEC.JOB_EXEC_ID = :GLOBAL.B1_JOB_ID
```
24. Join B1_JOB_EXEC with the output of the SET operator.

This join filters the incremental records only for that slicing period.

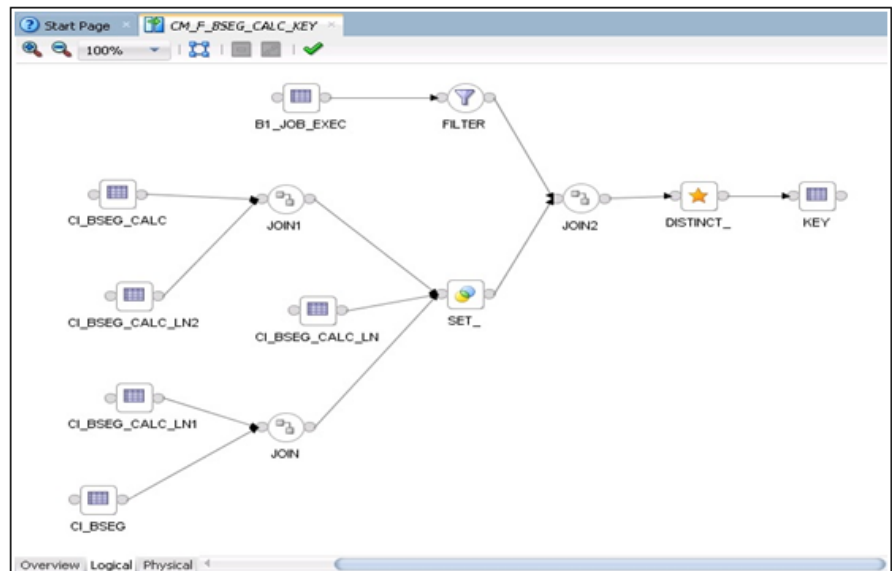


The output of JOIN2 is redirected to distinct only on the natural key of the fact, so that the duplicate keys are loaded into the KEY table.

25. Drag the KEY table from the model and map the natural key from DISTINCT operator to the Target table.
26. Select the key columns from the KEY table in the **Properties** window.
27. In the **Attributes** section, select the respective KEY check box for all the columns that are part of the natural key.

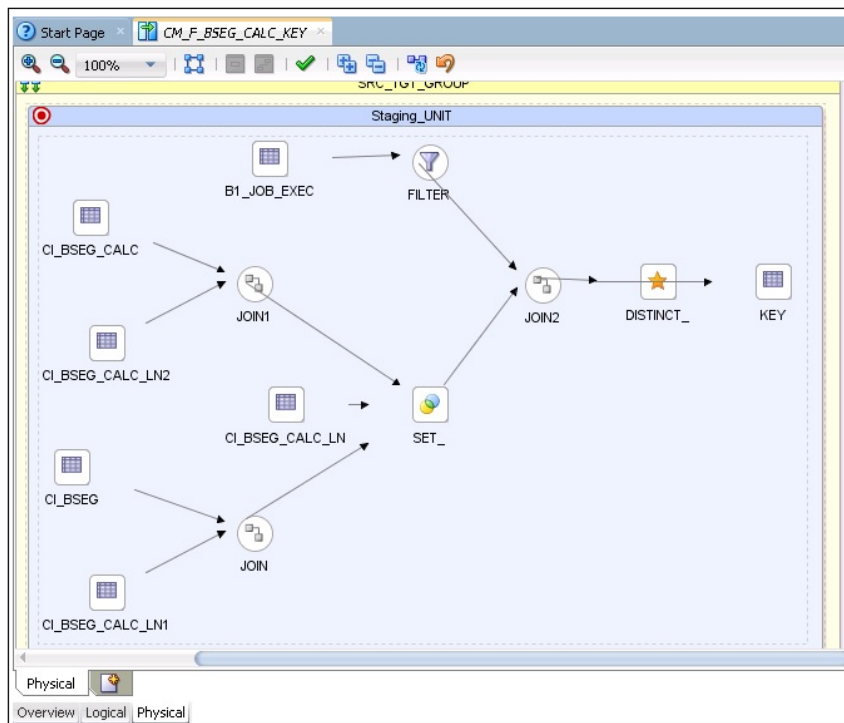


The figure below shows the logical mapping of the tables.



28. On the **Physical** tab, select the context and save the mapping so that the physical mapping diagram is visible.

In this example, select “CCB7”.



29. Click the target table and open the **Properties** window. Do the following:
 - a. Select “IKM BI Direct load” from the **Integration Knowledge Module** drop-down list.
 - b. Modify the “DML_OPERATION” option to “INSERT”.
30. Click **Save** to save the mapping.

Creating Loading Views in Fact Model

The fact loading view is created in the replication model. Since the table is created at run time, the table name is suffixed with the job number. It helps in the parallel data load.

To create a loading view in the fact model:

1. Login to the Oracle Data Integration client.
2. On the **Designer** tab, navigate to **Models > User Customization > <product_name> > Replication**.

In this example, ‘CCB’ is the product.

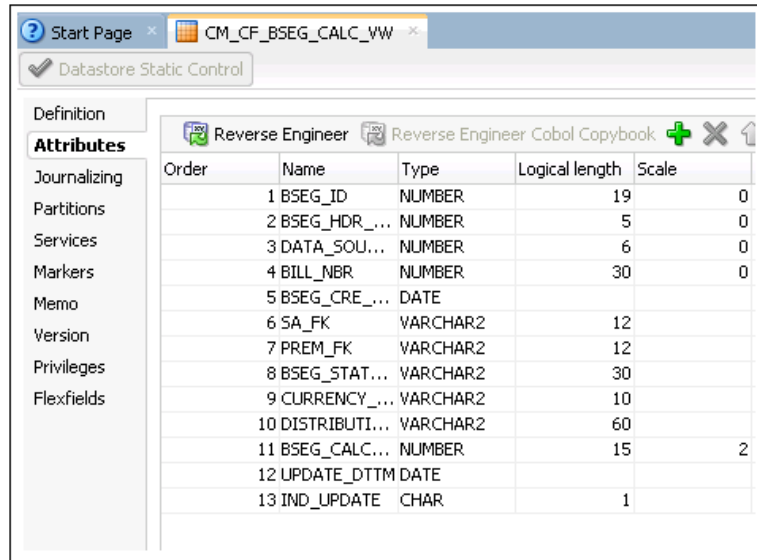
3. Right-click **Replication** and select **New Datastore** from the menu.
4. On the datastore editor, enter the view name “CM_CF_BSEG_CALC_VW” for the name.
5. Navigate to **Attributes** tab.
6. Click **+** to add columns to the datastore. Add all columns from the fact table except the dimension keys.

- In the dimension keys, replace “KEY” with “FK”.

The natural key of the dimension in view is needed to lookup the dimension and populate the dimension key in fact. If the dimension’s natural key has more than one column, then the view has the dimension’s natural key. The naming convention of the keys is FK1, FK2, etc.

Note: The data type and length of the columns should match with that of the fact.

- In addition to the above columns, add IND_UPDATE and UPDATE_DTTM columns. The data types of these columns are CHAR(1) and DATE respectively.



- Click **Save** to save the datastore.

Creating Mapping to Loading Views for Fact Model

The loading view for a fact is created to load the data into the fact. The loading view is joined with the KEY table on natural key, so that the data in that slicing period is loaded. The view is generated in the Replication schema.

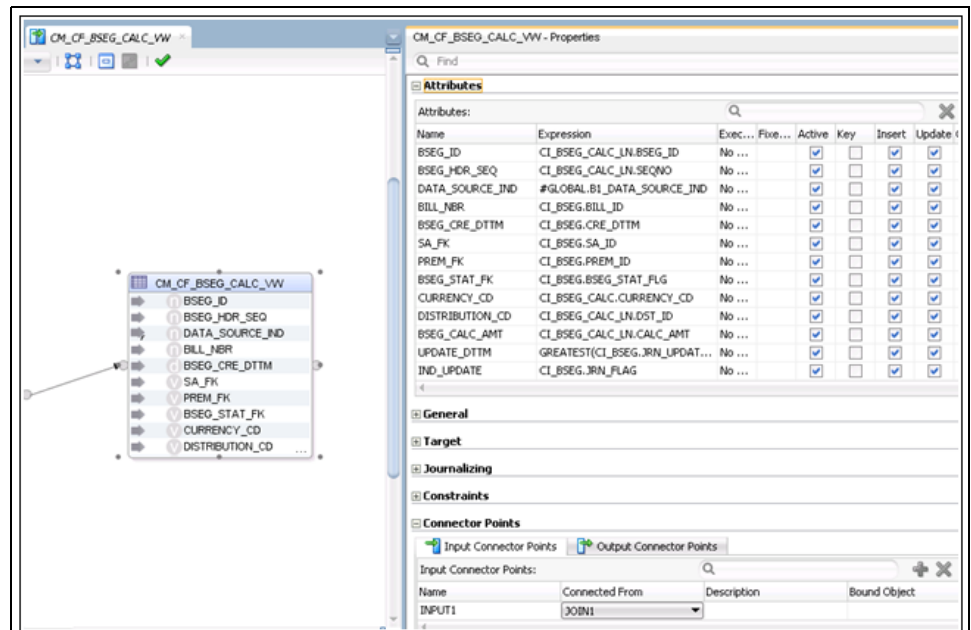
To create a mapping to generate the loading view:

- On the Oracle Data Integrator client, navigate to **Models > User Customization > <product_name>**.

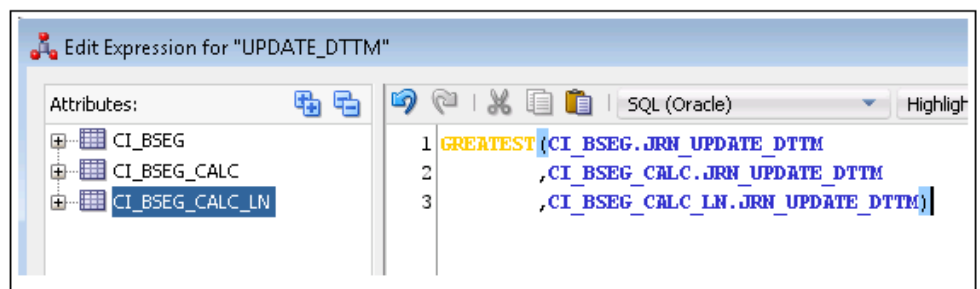
In this example, ‘CCB’ is the product.

- Create a new folder “Replication” and expand it.
- Right-click **Mapping** and select **New Mapping** from the menu.
- In the mapping editor, do the following:
 - Enter “CM_CF_BSEG_CALC_VW” in the **Name** field.
 - Unselect the **Create Empty Dataset** check box.
- Navigate to **Models > User Customization > CCB > Replication**.

6. Drag the tables from the model, and the CI_BSEG_CALC_LN and CI_BSEG replication tables to the mapping editor.
7. Select the JOIN component from the **Component** window and name it as “JOIN”.
8. Map the CI_BSEG and CI_BSEG_CALC_LN tables as input to the join and specify the join condition in the expression.
9. Drag the table CI_BSEG_CALC to the mapping editor.
10. Select the JOIN component from the **Component** window and name it as “JOIN1”.
11. Map the CI_BSEG_CALC and output of JOIN as input to the join component “JOIN1” and specify the join condition in the expression.
12. Drag the View datastore from the model and map the columns from the replication tables as per the logic to populate the columns.



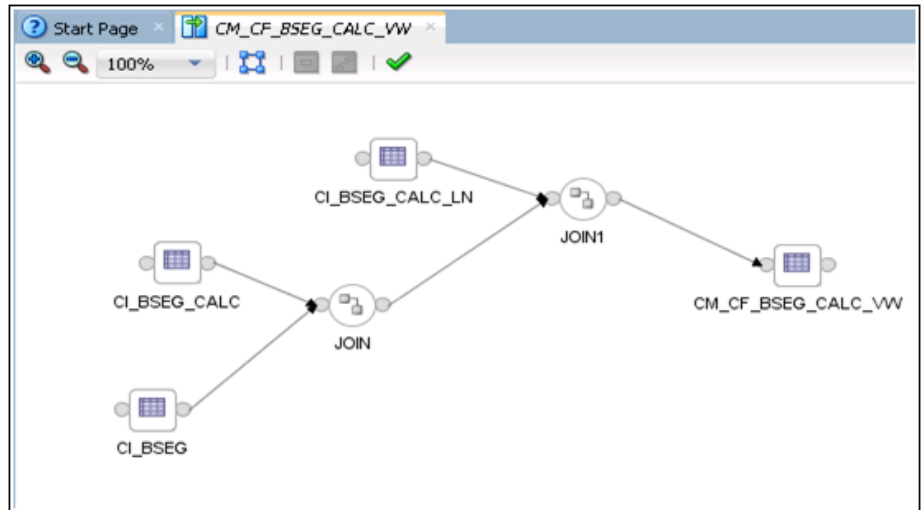
UPDATE_DTTM should be populated as Greatest of JRN_UPDATE_DTTM or JRN_EFF_START_DTTM from all the replication tables. It is populated so that the latest Dimension key of the SCD2 dimension is populated.



DATA_SOURCE_IND is populated from the B1_DATA_SOURCE_IND variable from Global objects.

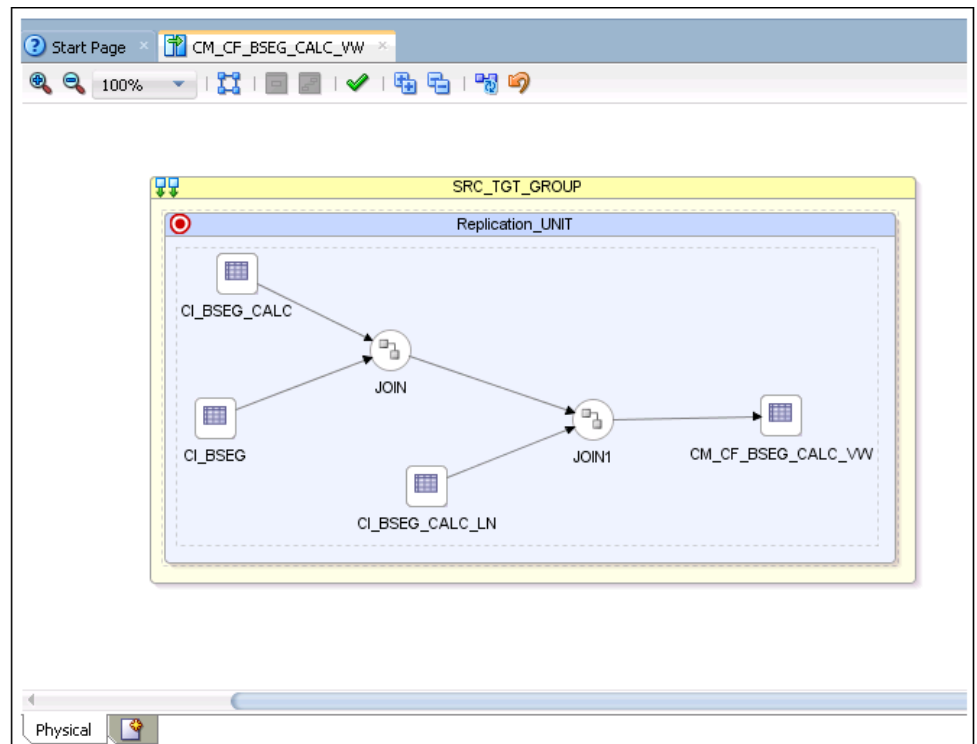
- Map the primary driver table's JRN_FLAG to IND_UPDATE.

The figure below shows the logical mapping.



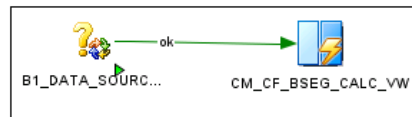
- Navigate to the **Physical** tab and select **Context**.
- Save the mapping so the physical mapping diagram is visible.

In this example, select “CCB7”.



- Click the target table. On the **Properties** window select “IKM BI View Generator” from the **Integration Knowledge Module** drop-down list.
- Click **Save** to save the mapping.
- Create a package with the same name as that of the view name.

- Drag and drop the global variable “B1_DATA_SOURCE_IND”. Drag the mapping and join the steps.



- Click **Save** to save the package.
- Regenerate the scenario for the package. During regeneration, unselect the **Startup Variable** check box.
- Run the scenario in Context so that the view is created in the database.

Creating Aggregate Tables in Fact Model

An aggregate table is created by the scheduling process during the execution of a fact job. It is created to optimize the parallel execution of multiple slices of the same entity load.

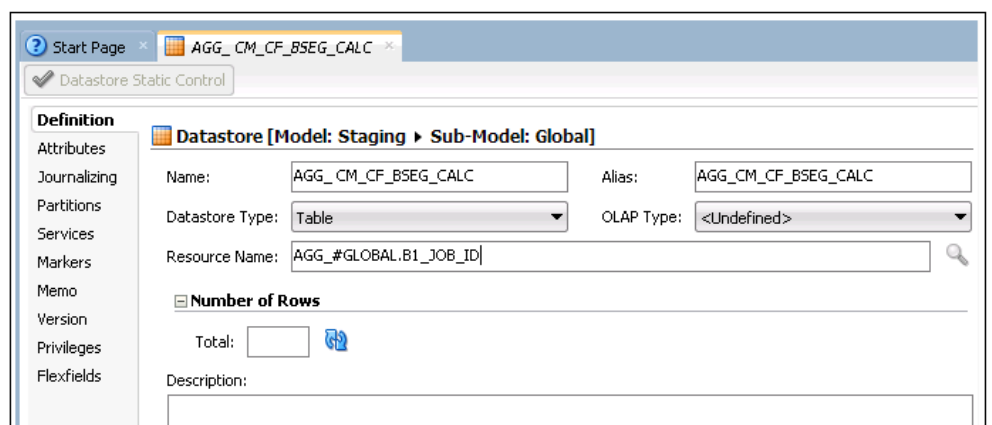
The table name starts with ‘AGG’ and is suffixed with the job number. The table structure should be present in a fact model under the Staging folder. The table name can vary (such as AGG_<FACT_NAME>), but the resource name should be “AGG_#GLOBAL.B1_JOB_ID”.

The aggregate table is created based on the flex field. The table structure is similar to the target table structure, including a few more columns. There should be an IND_UPDATE column in the table, in addition to the columns used in the mapping. The table should also include an UPDATE_DTTM column which stores the greatest effective start date of the record.

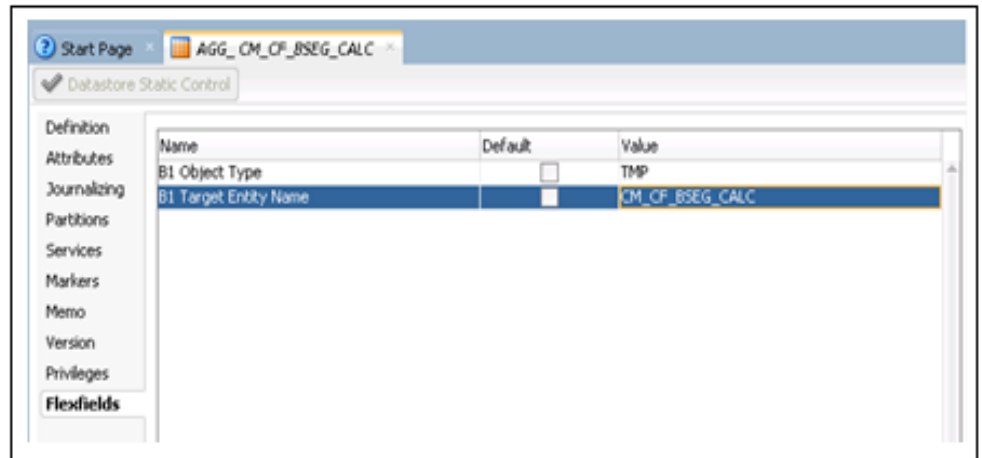
To create an aggregate table in the fact model:

- On the Oracle Data Integrator client, navigate to **Models > User Customization > <product_name> > Staging**.
- Right-click the model and select **New Datastore** from the menu.

Note: Create a staging model if it does not exist.
- On the **Datastore** editor, do the following:
 - Enter “AGG_CM_CF_BSEG_CALC” in the **Name** field.
 - Enter “AGG_#GLOBAL.B1_JOB_ID” in the **Resource Name** field.



4. Click **Flexfields** on the left pane.
5. Unselect the **Default** check box. Enter “TMP” and “CM_CF_BSEG_CALC” in the B1 Object Type and B1 Target Entity Name fields respectively.



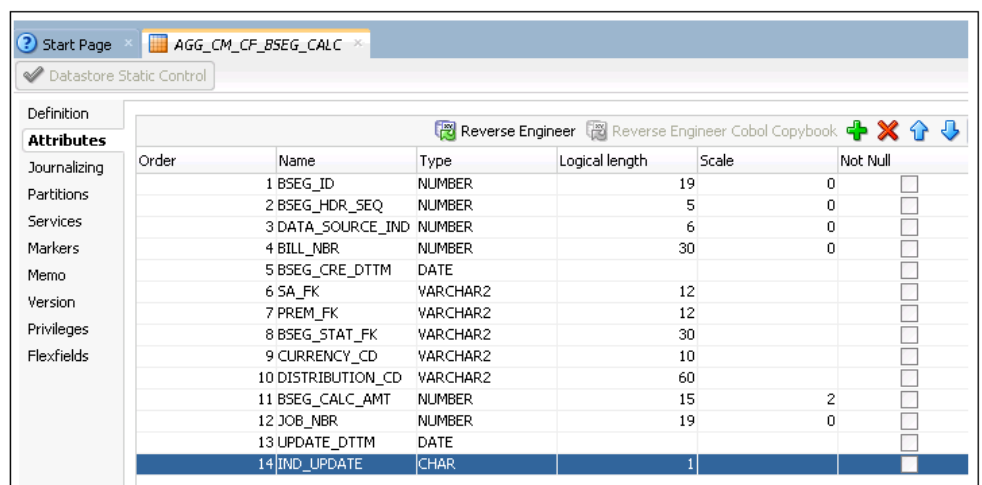
6. Click **Attributes** on the left pane.
7. Click + to add the required columns to the datastore.
8. Add all columns from the fact table except the dimension keys.
9. In the dimension keys, replace “KEY” with “FK”.

The natural key of the dimension in view is needed to lookup the dimension and populate the dimension key in fact. If the dimension's natural key has more than one column, then the view has the dimension's natural key. The naming convention of the keys is FK1, FK2, etc.

Note: The data type and length of the columns should match with that of the fact.

10. In addition to the above columns, add IND_UPDATE and UPDATE_DTTM columns.

The data types of these columns are CHAR(1) and DATE respectively.



11. Click **Save** to save the datastore.

Creating Mapping to Load Aggregate Tables in Fact Model

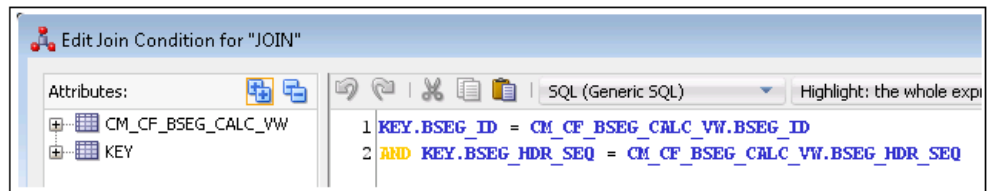
After creating an aggregate table, the data is loaded using the KEY table and loading the view created for the fact.

To create a mapping to load an aggregate table in the fact model:

1. Login to the Oracle Data Integrator client.
2. Navigate to **Projects > User Customization > <product name> > Mapping**.

In this example, 'CCB' is the product.

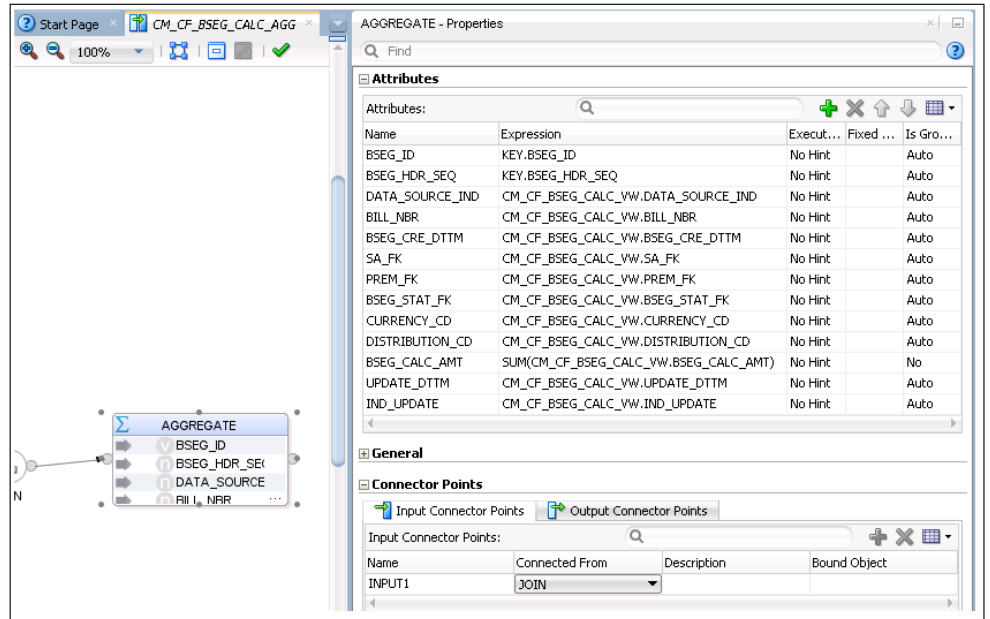
3. Right-click **Mapping** and select **New Mapping** from the menu.
4. On the mapping editor, enter the mapping name in the **Name** field. Unselect the **Create Empty Dataset** check box.
5. On the designer, navigate to **Models > User Customization > CCB > Staging**.
6. Drag the KEY_CM_CF_BSEG_CALC table from the model to the mapping editor.
7. On the designer, navigate to **Models > User Customization > CCB > Replication**.
8. Drag the CM_CF_BSEG_CALC_VW view from the model to the mapping editor.
9. Select the JOIN component from the **Component** window. Enter "JOIN" in the **Name** field.
10. Map the KEY table and loading view as input to the join and specify the join condition in the expression.



11. Drag the AGGREGATE component in the **Components** window and drop it onto the mapping editor.
12. Map the output of the join to the AGGREGATE component.
13. In the AGGREGATE component **Properties** window, click + in the **Attributes** section to add columns.

Note: Ensure that the column names should match with those of the AGGREGATE table.

14. Map all columns from the Loading view and KEY table as applicable.



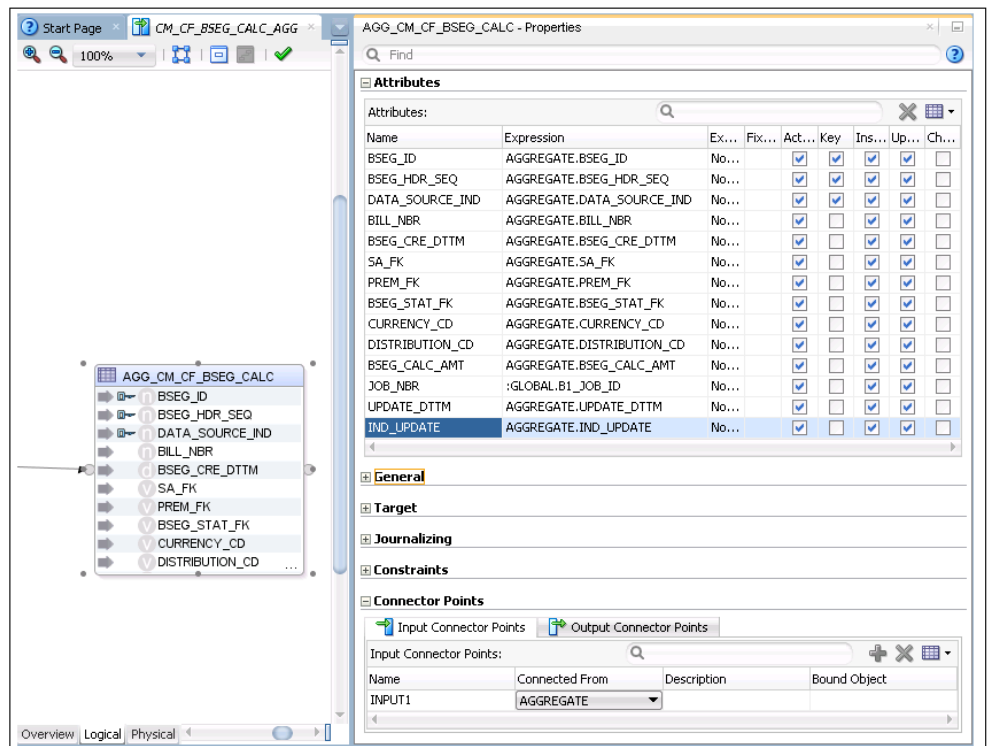
15. On the designer, navigate to **Models > User Customization > CCB > Staging**.

16. Drag the AGG_CM_CF_BSEG_CALC aggregate table from the model.

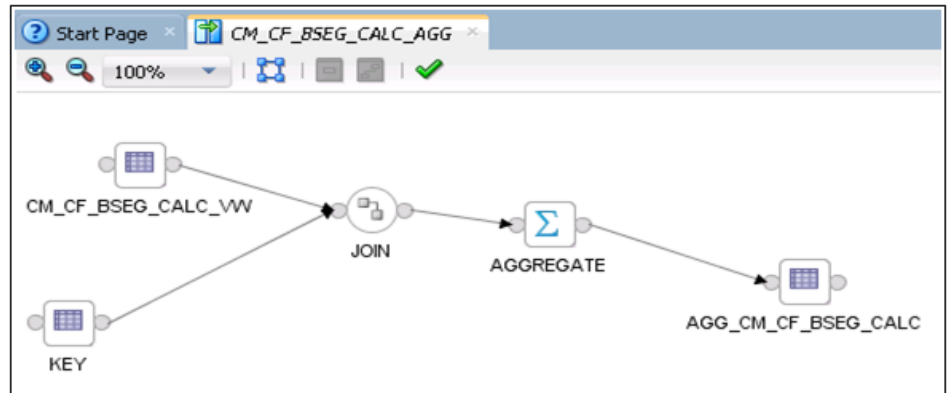
17. Map the output of the AGGREGATE component to the input of the AGG table.

18. Map all columns of the AGGREGTE table.

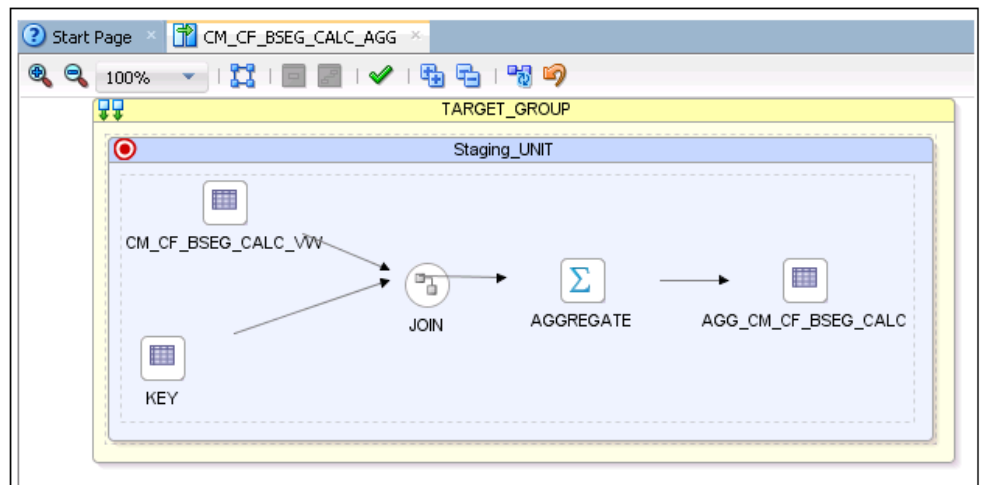
19. Select the KEY columns in the table. Select the respective **KEY** check box for those columns that are part of the natural key.



20. The logical mapping is complete as shown in the figure below.



21. On the designer, navigate to the **Physical** tab. Select the **Context** and **Save** the mapping so that the physical mapping diagram is visible. In this example, select “CCB7”.



22. Click the target table. The **Properties** window is displayed.

23. Select “IKM BI Direct Load” from the **Integrated Knowledge Module** list.

Name	Description	Value	Use Default
CREATE_TARG_TABLE	Create the target table.	True	<input checked="" type="checkbox"/>
SELECT_OPTIMIZER_HINT	This Option provide the fl...		<input checked="" type="checkbox"/>
DML_OPTIMIZER_HINT	This option provides the fl...		<input checked="" type="checkbox"/>
DML_OPERATION	This operation provides fl...	MERGE	<input checked="" type="checkbox"/>
FLAG_DUPLICATES	Based on the values dupli...	True	<input checked="" type="checkbox"/>

24. Click **Save** to save the mapping.

Creating Staging Tables in Fact Model

A staging table is created by the scheduling process prior to the interface execution. It optimizes the parallel execution of multiple slices of the same entity load.

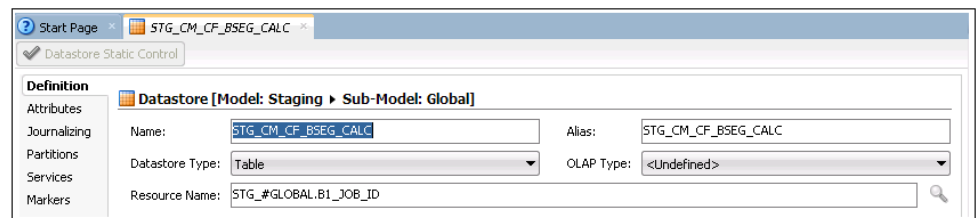
The definition of the staging table structure is under the **Staging** folder. The staging table structure is similar to the target table structure, including a few additional columns. It should include IND_UPDATE in addition to the columns used in the mapping.

To create a staging table in the dimension model:

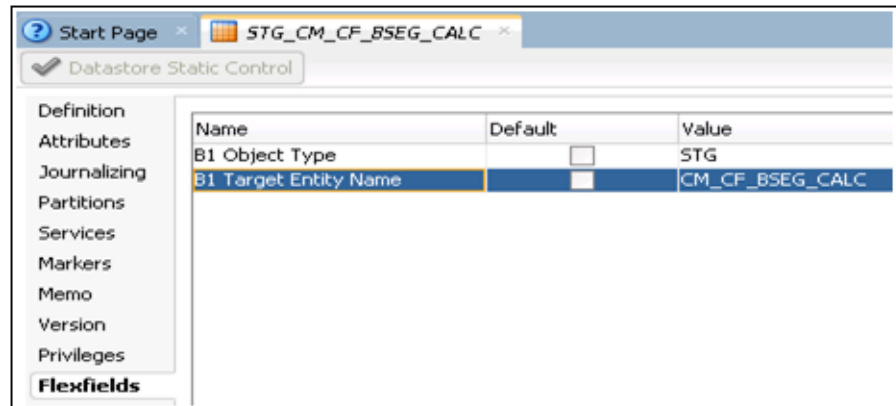
1. On the Oracle Data Integrator client, navigate to **Models > User Customization > <product_name> > Staging**.

In this example, 'CCB' is the product.

2. Right-click the model and select **New Datastore** from the menu.
3. In the Datastore editor, enter the aggregate table name (STG_CM_CF_BSEG_CALC) in the **Name** field. Enter "STG_#GLOBAL.B1_JOB_ID" in the **Resource Name** field.



4. Click the **Flexfields** tab.
5. On the editor, unselect the **Default** check box for all columns. Enter "STG" and "CM_CF_BSEG_CALC" in the **B1 Object Type** and **B1 Target Entity Name** fields respectively.



6. Navigate to **Attributes** tab.
7. Click + on right-hand corner to add columns to the datastore.

Add all columns that are in the fact table. In addition, add the dimension's natural key column. The dimension natural keys (as specified in the aggregate table) should be included in the staging table.

The data type and length of the columns should match with that of the fact table.

8. In addition to the above columns add the following:
 - IND_UPDATE column with CHAR(1) as the data type.
 - UPDATE_DTTM with DATE as the data type.
 - JOB_NBR with NUMBER(19) as the data type.

Order	Name	Type	Logical length	Scale	Not Null
1	BSEG_ID	NUMBER	19	0	
2	BSEG_HDR_SEQ	NUMBER	5	0	
3	DATA_SOURCE_IND	NUMBER	6	0	
4	BILL_NBR	NUMBER	30	0	
5	BSEG_CRE_DTTM	DATE			
6	SA_FK	VARCHAR2	12		
7	PREM_FK	VARCHAR2	12		
8	BSEG_STAT_FK	VARCHAR2	30		
9	CURRENCY_CD	VARCHAR2	10		
10	DISTRIBUTION_CD	VARCHAR2	60		
11	BSEG_CALC_AMT	NUMBER	15	2	
12	SA_KEY	NUMBER	15		
13	PREM_KEY	NUMBER	15		
14	BSEG_STAT_KEY	NUMBER	15		
15	JOB_NBR	NUMBER	19	0	
16	UPDATE_DTTM	DATE			
17	IND_UPDATE	CHAR	1		

9. Click **Save** to save the data store.

Creating Error Tables in Fact Model

An error table is created during the fact job execution, with its table structure similar to that of a staging table.

The error table is populated for late arriving dimensions. If a dimension record is not present in the dimension during the fact load, then the dimension key is populated as -99 and the record is populated in the error table. During the next fact load, data in the error table is looked up in the dimension table to find the records and correct the data in the fact table for the corrected dimension key.

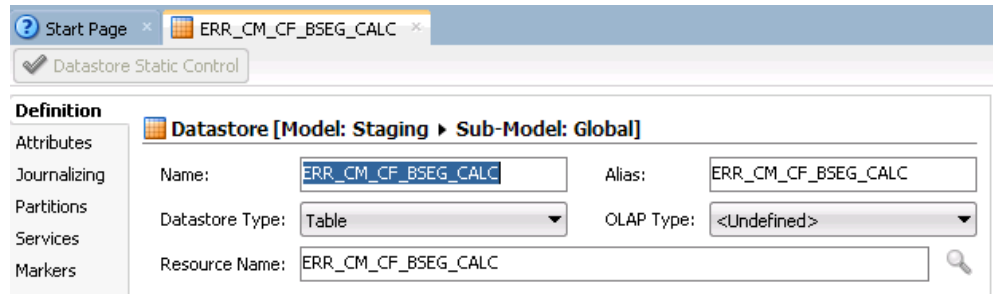
After reprocessing all dimension keys for the fact record, it is deleted from the error table. To do this, ensure that the staging table structure definition is included in a model under the Staging folder. Set the flex fields appropriately.

To create an error table in the fact model:

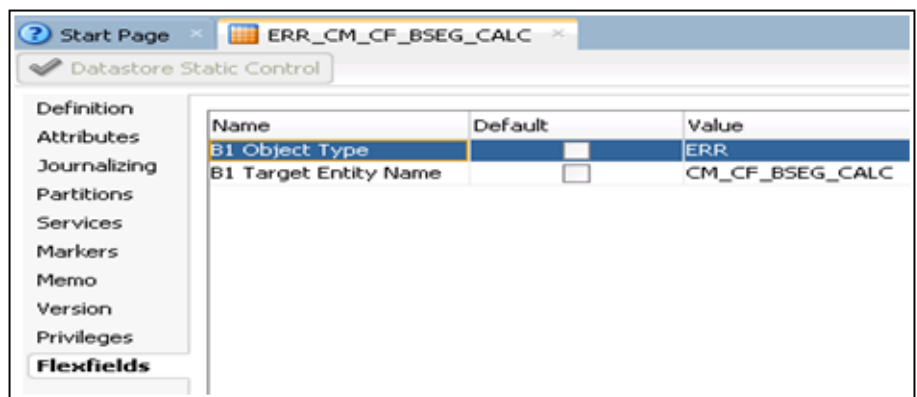
1. Login to the Oracle Data Integrator client.
2. Navigate to **Models > User Customization > <product name> > Staging**.

In this example, 'CCB' is the product.

3. Right-click the model and select **New Datastore** from the menu.
4. On the Datastore editor, enter "ERR_CM_CF_BSEG_CALC" in the **Name** field. Ensure the **Resource Name** is same as that of the error table name.



5. Navigate to the **Flexfields** tab.
6. Unselect the **Default** check box for the respective columns. Enter “ERR” and “CM_CF_BSEG_CALC” in the in **B1 Object Type** and **B1 Target Entity Name** columns respectively.



7. Navigate to the **Attributes** tab.
8. Click + on the right-hand corner to add columns to the datastore.
9. Add all columns from the fact table, and also add the dimension’s natural key column.

The dimension’s natural keys (as specified in the aggregate table) should be included in the staging table.

The data type and length of the columns should match with that of the fact table.

10. In addition to the above columns, add the following:
 - IND_UPDATE column with CHAR(1) as the data type.
 - UPDATE_DTTM with DATE as the data type.
 - JOB_NBR with NUMBER(19) as the data type.

Order	Name	Type	Logical length	Scale
1	BSEG_ID	NUMBER	19	0
2	BSEG_HDR_SEQ	NUMBER	5	0
3	DATA_SOURCE_IND	NUMBER	6	0
4	BILL_NBR	NUMBER	30	0
5	BSEG_CRE_DTTM	DATE		
6	SA_FK	VARCHAR2	12	
7	PREM_FK	VARCHAR2	12	
8	BSEG_STAT_FK	VARCHAR2	30	
9	CURRENCY_CD	VARCHAR2	10	
10	DISTRIBUTION_CD	VARCHAR2	60	
11	BSEG_CALC_AMT	NUMBER	15	2
12	SA_KEY	NUMBER	15	
13	PREM_KEY	NUMBER	15	
14	BSEG_STAT_KEY	NUMBER	15	
15	JOB_NBR	NUMBER	19	0
16	UPDATE_DTTM	DATE		
17	IND_UPDATE	CHAR	1	

11. Click **Save** to save the datastore.

Creating Mapping to Load Facts

The data is loaded into Staging table from the Aggregate table. The Staging table is updated for dimension key by looking up the dimension tables. After the dimension keys are updated, the fact table is loaded with data.

To load data from an Aggregate table to a Staging table, and then to the fact table, follow these instructions:

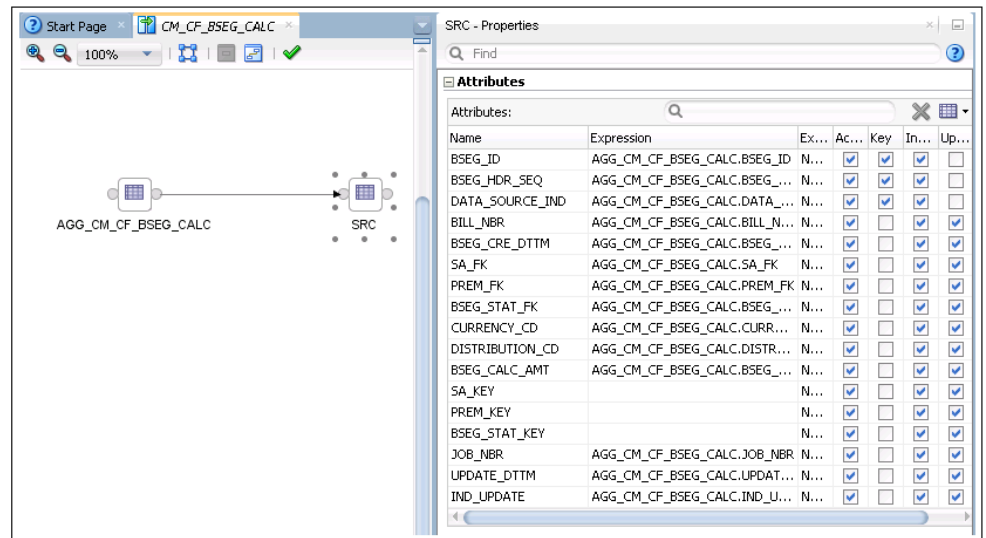
1. Login to the Oracle Data Integrator client.
2. On the Designer, navigate to the **Models > User Customization > <product_name> > Fact**.

In this example, 'CCB' is the product.

3. Right-click **Mapping** and select **New Mapping** from the menu.
4. On the mapping editor, enter "CM_CF_BSEG_CALC" in the Name field. Un-check the **Create Empty Dataset** check box.
5. Navigate to **Models > User Customization > CCB > Staging**.
6. Drag the Aggregate (AGG_CM_CF_BSEG_CALC) and staging (STG_CM_CF_BSEG_CALC) tables and drop them onto the mapping editor.

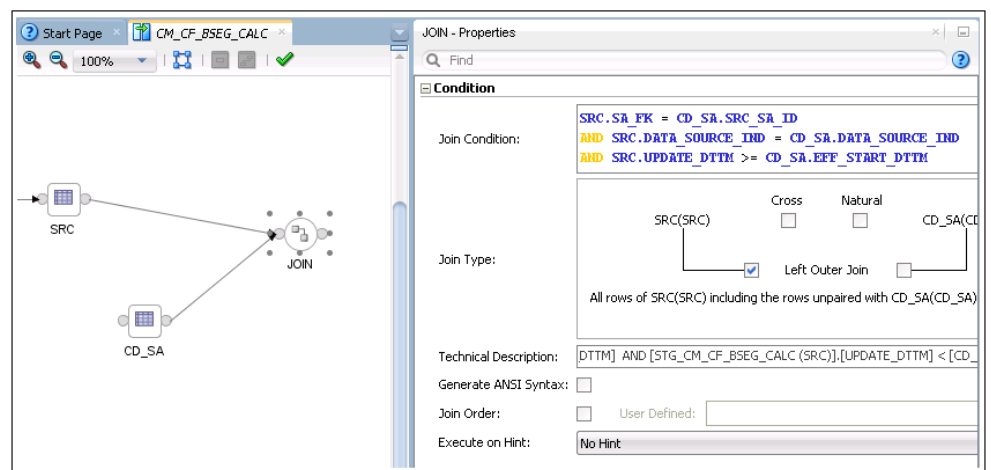
Provide an alias name (for example: "SRC").

7. Map the columns from the Aggregate table to those from the Staging table.



8. Drag and drop the Staging table again.

All joins from the Staging table (SRC) to the dimension should be an outer join including all rows from staging and any rows that are available from dimension.



9. Drag and drop the dimension and join the staging table with the dimension tables.

The join condition with type 2 dimension is as follows:

```
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 as follows:

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

10. Map the natural key columns of the target table (staging table) with that of the staging table. Select the KEY check box in the **Properties** window.

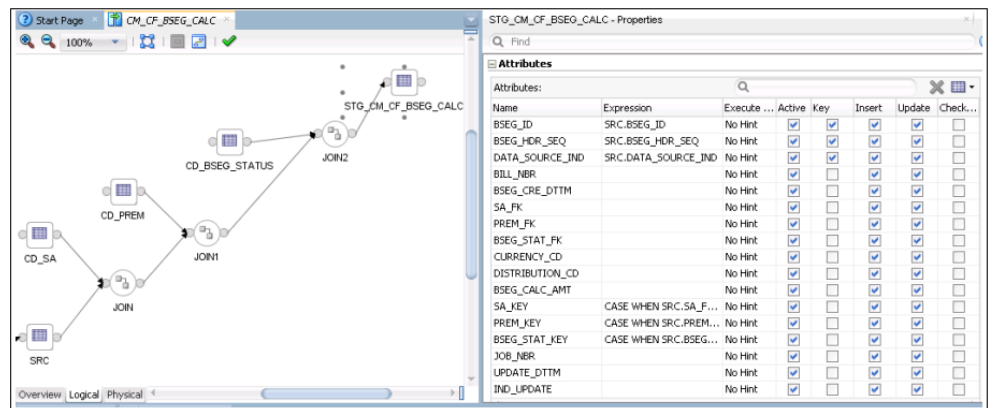
- Select a dimension key and provide the transformation for that key in the properties inspector.

Replace the actual dimension name and dimension column names as follows:

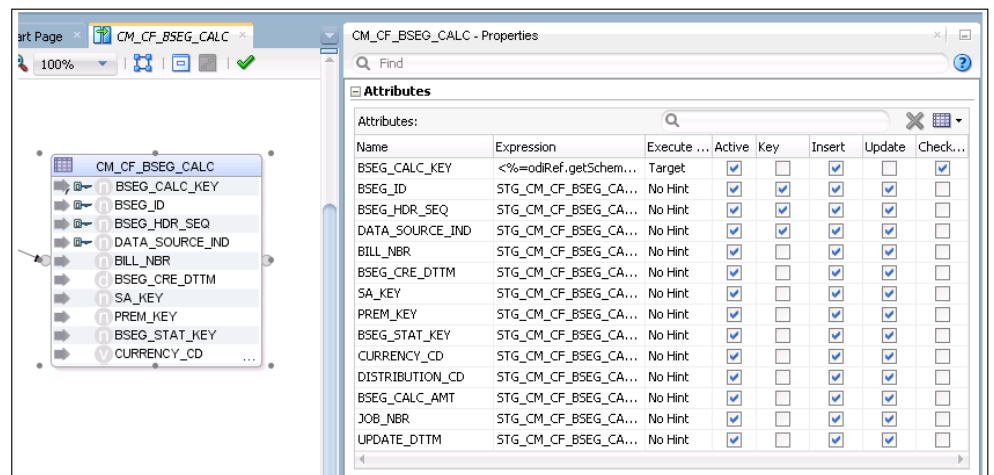
```

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

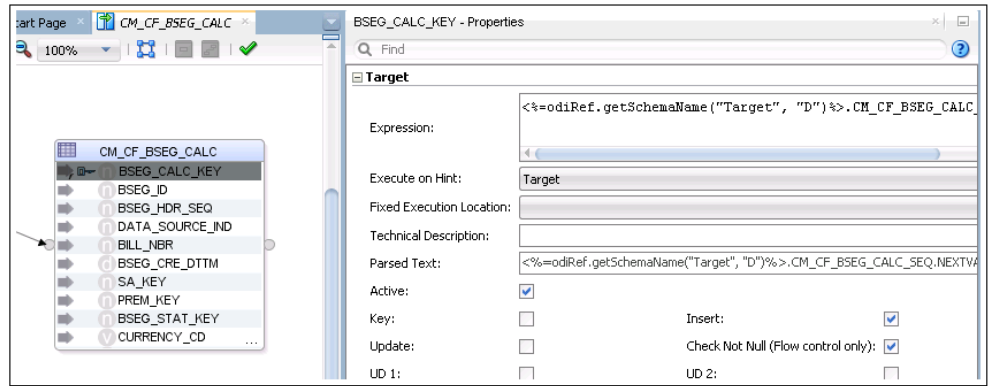
- Repeat step 11 for all dimension keys in the table.



- Drag and drop the fact table from the model.
- Map all columns in the fact table with those in the staging table (the target for the dimension lookup). Select the **Key** check box to mark the key columns.

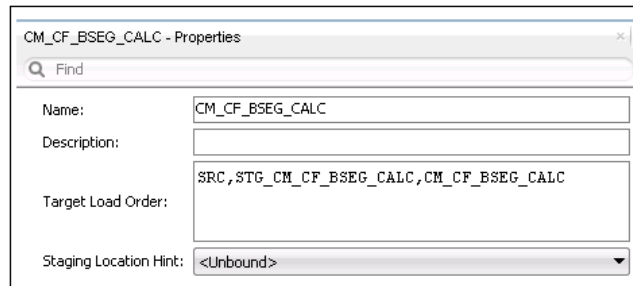


- Map the surrogate key to the sequence. Then, unselect the **Update** check box.

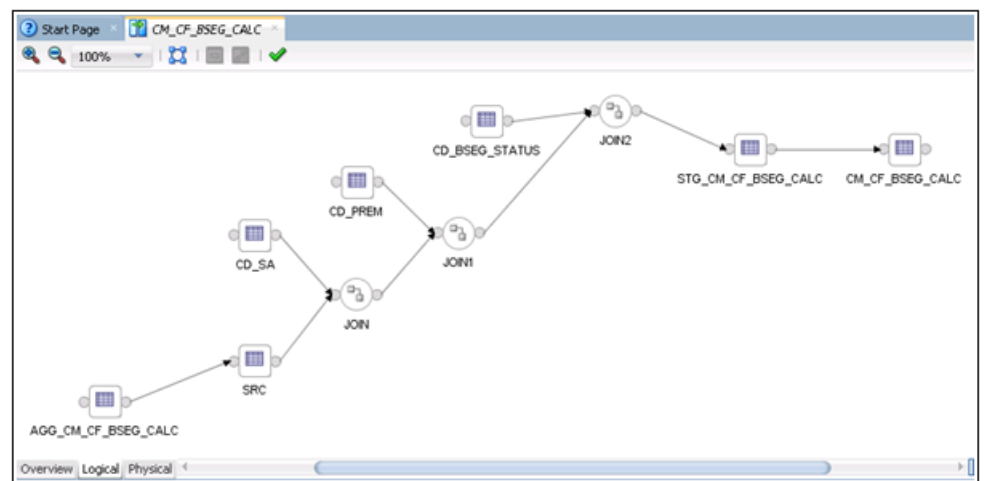


16. Save the mapping.
17. Click the mapping editor to open the **Properties** window for the mapping.
18. Specify the target load order as SRC, staging table, and fact table.

In this example: SRC,STG_CM_CF_BSEG_CALC,CM_CF_BSEG_CALC

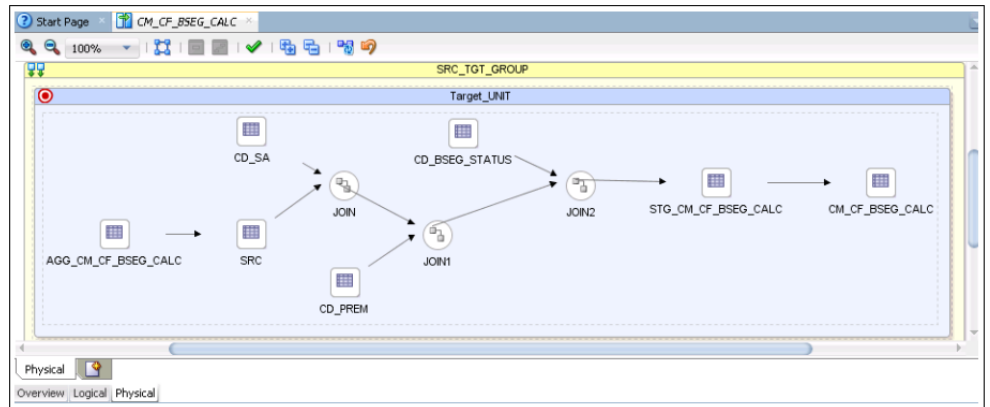


The logical mapping is complete. The figure below shows the mapping.



19. Navigate to the **Physical** tab and select the **Context**.
20. Click **Save** to save the mapping so that the physical mapping diagram is visible.

In this example, select “CCB7”.

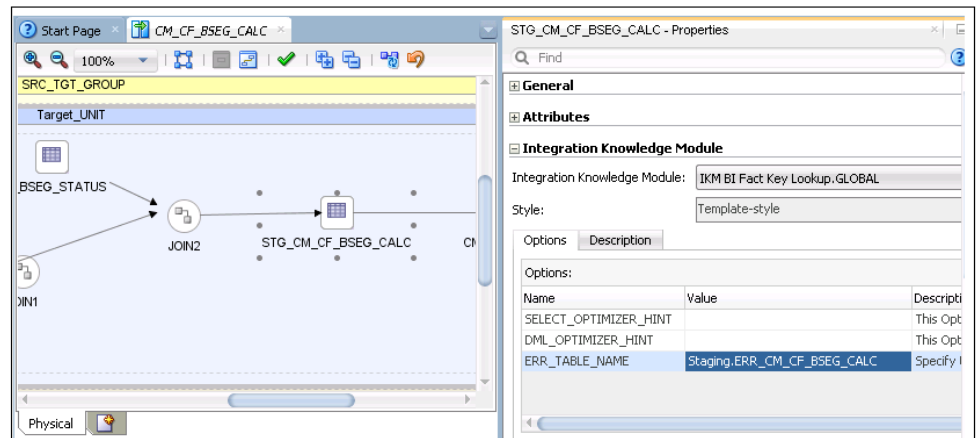


21. On the mapping diagram, click **SRC**. The respective **Properties** window is displayed.
22. Select “IKM BI Direct Load” from the **Integration Knowledge Module** drop-down list.

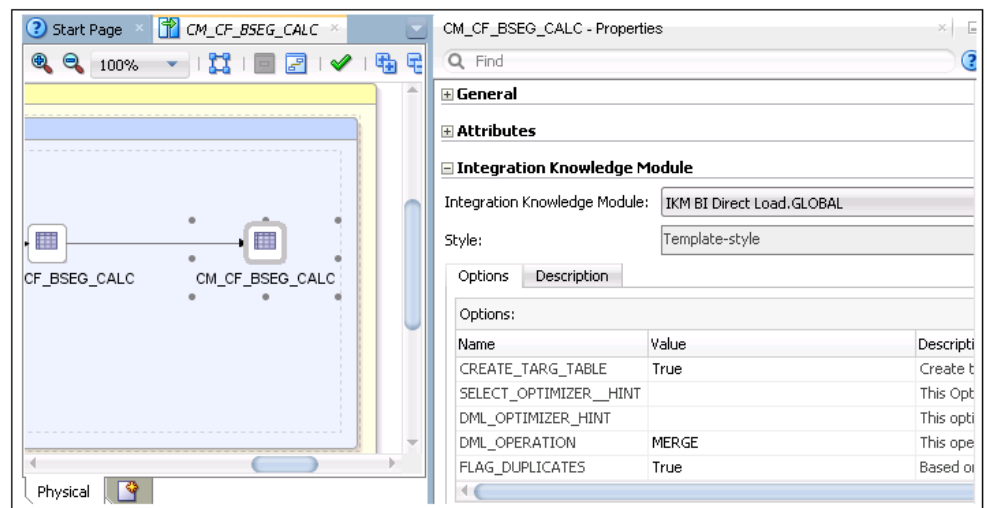
Options			
Name	Description	Value	Use Def
CREATE_TARG_TABLE	Create the target table.	True	
SELECT_OPTIMIZER_HINT	This Option provide the fl...		
DML_OPTIMIZER_HINT	This option provides the fl...		
DML_OPERATION	This operation provides fl...	MERGE	
FLAG_DUPLICATES	Based on the values dupli...	True	

23. Click **STG_CM_CF_BSEG_CALC** staging table. The respective **Properties** window is displayed.
24. Select “IKM BI Fact Key Lookup” from the **Integration Knowledge Module** drop-down list.

Enter “Staging.ERR_CM_CF_BSEG_CALC” in the **Value** field for the **ERR_TABLE_NAME** option. (This option is set to get the error table name).



25. Click the **CM_CF_BSEG_CALC** fact table. The respective **Properties** window is ready.
26. Select “IKM BI Direct Load” from the **Integration Knowledge Module** dropdown list.



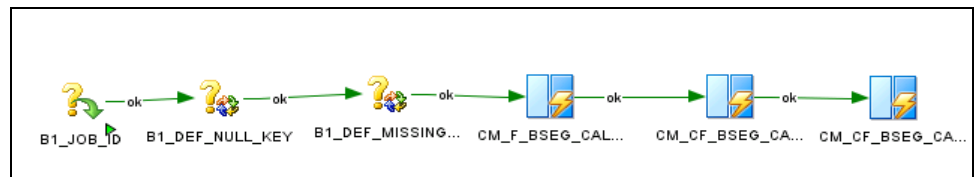
27. Click **Save** to save the mapping.

Creating Packages in Fact Model

To create a package in the new fact model:

1. Login to the Oracle Data Integrator client.
2. On the Designer, navigate to **Models > User Customization > Facts > Packages**.
3. Right-click **Package** and select **New Package** from the menu.
4. On the package editor, enter “CM_PKG_CM_CF_BSEG_CALC” in the **Name** field.
5. Click the **Diagram** tab at the bottom of the editor.

6. In the **Global Objects** section, do the following:
 - a. Drag the following variables into the editor:
 - B1_JOB_ID
 - B1_DEF_MISSING_KEY
 - B1_DEF_NULL_KEY
 - b. Modify the following variables to declare the variable:
 - B1_JOB_ID
 - c. Modify B1_DEF_MISSING_KEY and B1_DEF_NULL_KEY to refresh variables.
 - d. Drag and drop the mapping to load the aggregate table, and then mapping to load the fact table into the editor. Connect them all in a sequence.



- e. Click **Save** to save the changes and close the package editor.
7. Navigate to **Packages** and expand it. The new package is displayed.
8. Right-click **Generate Scenario**. Enter the scenario name and then click **OK**.
9. When prompted, select the startup variables.
10. Unselect B1_DEF_MISSING_KEY and B1_DEF_NULL_KEY (as they are refresh variables) and click **OK**.
11. Expand the package. In the list of scenarios, the new scenario object that was generated is displayed.

Configuring Entities in Fact Model

To configure a new entity in a custom fact:

1. Login to the Oracle Utilities Analytics Warehouse Administration user mapping.
2. On the **ETL Configuration** tab, click **Target Entity**.
3. Click **Add**. The **Main Target Entity** page is displayed.
4. Enter the appropriate values for the entity.

Specifying Dependencies in Fact Model

The dependency for Type 2 dimension should be specified in the fact. It is mentioned in the B1_OBJECT_MAP table for the custom fact.

Below is a sample query to insert the dependency. Ensure to add this merge query in the existing CM procedure to add the metadata in Oracle Utilities Analytics Warehouse.

Note: Refer to the [Configuring CM Scenarios](#) section for instructions to create a CM procedure.

```

merge
  into b1_object_map tgt
  using (select 'CCB'
            , 'CD_SA'
            , 'CM_CF_BSEG_CALC'
            , 1
            , 'DMDP'
            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');

merge
  into b1_object_map tgt
  using (select 'CCB'
            , 'CD_PREM'
            , 'CM_CF_BSEG_CALC'
            , 2
            , 'DMDP'
            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');
COMMIT;

```

Configuring Jobs in Fact Model

To configure a job for the custom fact:

1. Login to the Oracle Data Integrator client.
2. On the designer, navigate to **Load Plans and Scenarios > Accelerators > Oracle Utilities Analytics Warehouse**.
3. Right-click the B1_CFG_INSTANCE scenario and run in the context.

A fact job for that context is created. Also, dependency for the context specific dimensions for this fact job is created.

Alternately, execute the scenario from the Oracle Data Integrator console as follows:

1. Login to the Oracle Data Integrator console.

The Oracle Data Integrator console is deployed when the WebLogic agent for Oracle Data Integrator is created.

The URL format for the console is:

<http://<Weblogic Host>:<Managed Server port>/odiconsole>

2. Login to the Work repository using the ‘SUPERVISOR’ credential.
3. In the browser, navigate to **Runtime > Scenario/Load Plan > Folders > Accelerators > OUA**.
4. Right-click the B1_CFG_INSTANCE scenario and execute it in the context. The scenario is executed successfully.

After executing the scenario successfully, enable the fact job in Oracle Utilities Analytics Warehouse Administration.

1. Login to Oracle Utilities Analytics Warehouse Administration.
2. On the **ETL Configuration** tab, click **Job Configuration**.
3. Search for the custom fact and edit it.
4. Select “Yes” from the **Entity Active Flag** drop-down list. The fact is enabled successfully.

Monitoring Job Executions

After the fact job is configured for customization and activated, monitor the job execution using Oracle Utilities Analytics Warehouse Administration or SQL Developer.

To monitor the fact job execution using Oracle Utilities Analytics Warehouse Administration:

1. Login to Oracle Utilities Analytics Warehouse Administration.
2. On the **ETL Job Execution** tab, enter the fact name.
3. Click **Go** to filter the data.

To view the latest execution, sort by the session end date.

Custom Materialized Views

A materialized view stores the aggregated data, helping the analytics to fetch data from the materialized view.

Note: OOTB materialized views are not provided for Oracle Utilities Network Management Systems and Oracle Utilities Work and Asset Management.

This section provides the steps to create a materialized view on custom facts:

1. [Creating Mapping for Materialized View](#)
2. [Creating Packages for Materialized View](#)
3. [Configuring Entities for Materialized View](#)
4. [Specifying Dependencies for Materialized View](#)
5. [Configuring Jobs for Materialized View](#)
6. [Monitoring Job Execution](#)

Creating Mapping for Materialized View

To create mapping for a materialized view:

1. Login to the Oracle Data Integrator client.
2. On the Designer, navigate to the **Models > User Customization > <product_name>**.

In this example, 'CCB' is the product.

3. Create a **Materialized View** folder (if not already existing).
4. Right-click the mapping and select **New Mapping** from the menu.
5. On the mapping editor, enter the mapping name in the **Name** field.
6. Navigate to **Models > Target**.

Expand the model and drag the source table (dimension or fact) into the target area of the mapping editor.

7. Set up the appropriate join conditions between the source tables.

8. Navigate to **Models > User Customization > CCB > Materialized View**.
9. Drag and drop the materialized view to the mapping editor.

Note: Ensure the materialized view datastore is created in the Oracle Data Integrator model before creating the mapping.
10. Map the **Target Table (Materialized View)** columns with those of the source tables.
11. On the **Flow** tab, select “IKM BI Materialized View” from the **Integration Knowledge Modules (IKM)** drop-down list.
12. Click **Save**.
13. Run the mapping so that the materialized view is created in the database.
14. Reverse the materialized view in Oracle Data Integrator so that the data type is same in both Oracle Data Integrator and database.

Note: If the datastore structure is different in database and Oracle Data Integrator, then execute the materialized view in Upgrade mode. Then, modify the materialized view definition instead of refreshing it.

Creating Packages for Materialized View

To create a package for the new fact:

1. Login to the Oracle Data Integrator client.
2. Navigate to **Designer > Models > User Customization > Materialized View > Packages**.
3. Right-click **Package** and select **New Package** from the menu.
4. On the package editor, enter the package name in the **Name** field.
5. Click the **Diagram** tab at the bottom of the editor.
6. From the **Global Objects** section, drag the B1_JOB_ID variable into the editor.

Modify the B1_JOB_ID variable to declare a variable.

7. Drag and drop the mapping into the editor and connect them in sequence.
8. Click **Save** to save the changes and close the package editor.

Navigate to **Packages** and expand it. The new package is displayed.

9. Right-click **Generate Scenario**. Enter the scenario name and click **OK**.
10. In the **Packages** folder, verify if the scenario object generated is listed.

Configuring Entities for Materialized View

To configure a new entity for a custom materialized view:

1. Login to Oracle Utilities Analytics Warehouse Administration.
2. On the **ETL Configuration** tab, click **Target Entity**. Click **Add**.
3. Enter the fact job details.
4. Enter the materialized view name. Click **Save** to save the details.

Specifying Dependencies for Materialized View

This section provides a sample query to insert the dependency.

```
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');
```

Configuring Jobs for Materialized View

To configure a job for the custom fact:

1. Login to Oracle Utilities Analytics Warehouse Administration.
2. On the **ETL Configuration** tab, select **Job Configuration**.
3. Click **Add** to add the job details.
4. On the **Job Addition** page, select a product from the **Source Product** drop-down list, and then select the **Instance Number**.
5. Click the **Search** icon for the **Target Entity** field. Enter the fact name and click **Go**.
6. Click **ID Value**. On the **Job Addition** page, the target entity ID is populated.
7. Set the **Slice Start Date/Time** to “01-Jan-2000” or the extract date to which the source instance is configured. Click **Add** to create the Job Configuration entry.
8. Enable the job while saving the new entry.

Monitoring Job Execution

After configuring the job for customization and activating it, monitor the job execution using Oracle Utilities Analytics Warehouse Administration or SQL Developer.

To monitor the job execution using Oracle Utilities Analytics Warehouse Administration:

1. Login to Oracle Utilities Analytics Warehouse Administration.
2. On the **ETL Job Execution** tab, enter the fact name and click **Go** to filter the data.

To view the latest execution, sort by the session end date.

Chapter 5

Extending Analytics

The Analytics Dashboards in Oracle Utilities Analytics Warehouse 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 Warehouse.

Refer to the [Extending Star Schema](#) chapter 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 sections below describe how to use Oracle Analytics Server to extend the analytics in Oracle Utilities Analytics Warehouse product:

- [Customizing Existing Analytics](#)
- [Creating New Analytics](#)

Customizing Existing Analytics

This section describes how to use Oracle Analytics Server to customize Oracle Utilities Analytics Warehouse. It includes the following:

- [Modifying the RPD File](#)
- [Customizing 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 Warehouse version, any customization done should be merged into the upgraded repository file through the Merge utility of Oracle Analytics Server.

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 Analytics Server 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 the [Managing Metadata Repositories for Oracle Analytics Server](#) documentation.

Customizing 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 Warehouse 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.

For details about managing, upgrading, and merging presentation catalogs, refer to the **Configure and Manage the Presentation Catalog** section in the [Administering Oracle Analytics Server](#) documentation.

For details about how to create or edit answers, refer to the [Oracle Analytics Server](#) documentation.

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 Analytics Server 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 **Administration Dashboards Maintenance** section in the *Oracle Utilities Analytics Warehouse Installation and Configuration Guide* available in the [Oracle Utilities Analytics Warehouse](#) documentation.

Creating New Analytics

If the additional report requirements are different from any of the base product reports, you can choose to build a completely new report from scratch.

This section describes how to use Oracle Analytics Server to add new analytics, including:

- [Creating New Answers](#)
- [Adding New Labels](#)
- [Customizing Hierarchy Levels](#)

Creating New Answers

Note: Before creating new reports, it is recommended to have knowledge of Oracle Analytics Server and Data Warehouse concepts. It is also recommended to have some working knowledge in developing reports using Oracle Analytics Server for a smoother implementation.

Oracle Utilities Analytics Warehouse 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 described in the [Customizing Existing Analytics](#) section, 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 Warehouse 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 Warehouse 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 presentation 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](#).

For more details, refer to the **Administration Dashboards Maintenance** section in the *Oracle Utilities Analytics Warehouse Installation and Configuration Guide* available in the [Oracle Utilities Analytics Warehouse](#) documentation.

Customizing Hierarchy Levels

Usually, the Control Zone (CD_CTRL_ZONE) and Control Zone Secondary (CD_CTRL_ZONE_SEC) dimensions have data in all the 6 levels (PARENT_NCG_LVL1_NAME to PARENT_NCG_LVL6_NAME columns). If the data is not available for any level, it will be considered a missing level. Data is displayed incorrectly or it is missing.

These Control Zone and Control Zone Secondary prompts can be customized. The dashboards in Oracle Utilities Analytics Warehouse can be adjusted to display the available zone levels in the desired prompts and reports. This zone level mapping is configured in OAS RPD.

For detailed configuration steps, refer to the knowledge article “Customizing Hierarchy Levels in Oracle Utilities Analytics Warehouse (Doc ID 2273874.1)” available on My Oracle Support (<https://support.oracle.com/>).

Chapter 6

Migrating Environments

Most implementations have multiple environments based on the associated activities. 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:** Used to extend the capabilities of Oracle Utilities Analytics Warehouse. All custom ETL and answers are developed in this environment.
- **Acceptance:** Typically used to perform functional validations based on the source system and custom code. This environment does not involve any product development.
- **Production:** Used to connect to Oracle Analytics Server and view the available dashboards and analytics.

The implementation life cycle starts in the Development environment. The code is moved to Acceptance environment, and finally into the Production environment. It is possible to have more than three environments.

This chapter provides information about migrating the OAS and ODI components across environments:

- [Migrating OAS Components](#)
- [Migrating ODI Components](#)

Migrating OAS Components

Migration of Oracle Analytics Server components from one environment to another typically involves moving the two main components of the tool - the presentation catalogs and the repository file.

The following sections focus on how to migrate each of these and whether a migration is actually needed or not.

- [Presentation Catalog](#)
- [Repository](#)

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 Warehouse Installation and Configuration Guide* in the **Dashboard Component** section.

You can extend the analytics by adding some additional dashboards and reports to cater to any some additional business requirements. The details have been provided in the [Extending Analytics](#) section. Make sure that the new objects are saved in a separate folder/catalog other than the base product catalogs. With the extra objects in place, you need to move these additional catalogs across their environments.

The recommended way from Analytics Server is to archive the catalogs from the source environment, move the files across and unarchive them in the target environment. Oracle Analytics Server 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 Analytics Server product.

Follow these steps:

1. Start the Catalog Manager in the source Oracle Analytics Server environment.
2. Login in the Online mode.
3. Pick the custom catalogs created and archive them.
4. Save the archived files (.catalog) on the local server.
5. Move these catalog files to the target server through FTP or other available means.
6. Start the Catalog Manager in the target Oracle Analytics Server environment in the Online mode.
7. Select Unarchive 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 Analytics Server environment.

Note: For more details on the Catalog Manger, refer to the [Oracle Analytics Server](#) documentation.

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 Analytics Server environments that customers have.

The Oracle Analytics Server Administration Tool that comes with the Oracle Analytics Server product can be used to save a copy of the repository file.

1. Start the Oracle Utilities Analytics Warehouse Administration Tool in the source Oracle Analytics Server environment.
2. Open the repository in Online mode.
3. Navigate to **File > Copy As > Repository**.
4. Enter a custom name for the repository (such as CM_UtilitiesBusinessAnalytics) and save the copy on the local machine.
5. Login to the Oracle Analytics Server Enterprise Manager console of the target Oracle Analytics Server environment.
6. Navigate to **BI Instance > Coreapplication > Deployment**.
7. Lock and edit. The **Repository** text box is enabled.
8. Browse to select the modified RPD file and submit it.
9. Provide the RPD password and click **Apply**.
10. Activate the changes and restart Oracle Analytics Server services.

Note: If the database connections used by the repository in the target environment are different, ensure to update the connection pool details in the repository file before deploying it on the server.

Migrating ODI Components

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.

Note: Before importing the code in an environment, purge the execution log if any. Use the **Smart export** and **Smart import** option when moving from one environment to other environments.

CM Project

To develop custom code create a custom project and ensure 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. Also, the entire CM project can be exported and imported into 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.

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.