

Oracle Financial Services Analytical Applications Data Model Utilities User Guide



Release 8.1.2.0.0

F84901-04

March 2026

The Oracle logo, consisting of a solid red square with the word "ORACLE" in white, uppercase, sans-serif font centered within it.

ORACLE®

Copyright © 2026, Oracle and/or its affiliates.

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish, or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

If this is software, software documentation, data (as defined in the Federal Acquisition Regulation), or related documentation that is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, then the following notice is applicable:

U.S. GOVERNMENT END USERS: Oracle programs (including any operating system, integrated software, any programs embedded, installed, or activated on delivered hardware, and modifications of such programs) and Oracle computer documentation or other Oracle data delivered to or accessed by U.S. Government end users are "commercial computer software," "commercial computer software documentation," or "limited rights data" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, the use, reproduction, duplication, release, display, disclosure, modification, preparation of derivative works, and/or adaptation of i) Oracle programs (including any operating system, integrated software, any programs embedded, installed, or activated on delivered hardware, and modifications of such programs), ii) Oracle computer documentation and/or iii) other Oracle data, is subject to the rights and limitations specified in the license contained in the applicable contract. The terms governing the U.S. Government's use of Oracle cloud services are defined by the applicable contract for such services. No other rights are granted to the U.S. Government.

This software or hardware is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications that may create a risk of personal injury. If you use this software or hardware in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure its safe use. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software or hardware in dangerous applications.

Oracle®, Java, MySQL, and NetSuite are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Intel and Intel Inside are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. AMD, Epyc, and the AMD logo are trademarks or registered trademarks of Advanced Micro Devices. UNIX is a registered trademark of The Open Group.

This software or hardware and documentation may provide access to or information about content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services unless otherwise set forth in an applicable agreement between you and Oracle. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services, except as set forth in an applicable agreement between you and Oracle.

Contents

1 About This Content

2 Introduction

3 Object Management

3.1	Boundaries and Limitations	1
3.2	Adding Dimension Tables and Key Dimension (Leaf) Registration	2
3.2.1	Adding Dimension Tables	2
3.2.1.1	Key Dimension Tables	3
3.2.1.2	Simple Dimension Tables	9
3.2.2	Adding Dimension Column To Required Objects	13
3.2.3	Assigning Processing Key Property	13
3.2.4	Uploading ERwin Model	14
3.2.5	Leaf Registration	14
3.2.5.1	Leaf Registration Procedure	14
3.2.5.2	Executing Leaf Registration Procedure	14
3.2.6	Modify Unique Indexes	16
3.2.7	Executing Object Registration Validation	16
3.3	Adding Custom Instrument Tables	16
3.3.1	Super-class Entities	17
3.3.2	Steps to Create Custom Instrument Table	18
3.3.3	Setting Table Classifications	19
3.3.4	Unique Index	20
3.3.5	Portfolio Selection	20
3.3.5.1	Adding a new User Defined column as a Portfolio column	20
3.3.6	Object Registration Validation	21
3.4	Adding Custom Transaction Tables	21
3.4.1	Super-class Entities	21
3.4.2	Steps to Create Custom Transaction Table	22
3.4.3	Setting Table Classifications	22
3.4.3.1	Setting Processing Key Property	23
3.4.4	Unique Index	23

3.4.5	Object Registration Validation	24
3.5	Adding Custom Lookup Tables	24
3.5.1	Steps to Create Lookup Table	24
3.5.2	Setting Column Properties	24
3.5.3	Setting Table Classifications	25
3.5.4	Registering Lookup Tables and Validation	25
3.5.5	Lookup Table Driver Definition	25
3.6	Adding Management Ledger Class tables	28
3.6.1	Super-class Entities	29
3.6.2	Creating Custom Ledger Class Table	30
3.6.3	Setting Table Classifications	30
3.6.4	Setting Processing Key Property	31
3.6.5	Unique Index	31
3.6.6	Removing the Dimensions	32
3.6.7	Object Registration Validation	33
3.7	Object Registration and Validation	33
3.7.1	User-Assignable Table Classification	34
3.7.2	Requirement For Table Classification	35
3.7.3	Validation Procedure	40
3.7.4	Executing the Validation Procedure	41
3.7.5	Exception Messages	41
3.8	Alternate Rate Output Columns	42
3.8.1	User-Defined Properties	42
3.8.2	Uploading the Model	43
3.9	User Defined Properties	43
3.10	Modifying the Precision of Balance Columns In Ledger Stat	47

4 Utilities

4.1	Reverse Population	1
4.1.1	Tables As Part Of Reverse Population	2
4.1.2	Reverse Population Procedure	3
4.1.3	Executing the Reverse Population Function	3
4.1.4	Exception Messages	5
4.2	Product Instrument Mapping	5
4.2.1	Tables Requiring Synchronization	5
4.2.2	Product Instrument Table Map Procedure	6
4.2.3	Executing the PRODUCT_INSTRUMENT_TABLE_MAP	6
4.2.4	Exception Messages	7
4.3	Instrument Synchronization	7
4.3.1	Tables Requiring Synchronization	8
4.3.2	Dimension Member Synchronization	8

4.3.3	Codes Synchronization	8
4.3.4	Executing the SYNCHRONIZE_INSTRUMENT	9
4.3.5	Exception Messages	10
4.4	Stage Synchronization	10
4.4.1	Tables Requiring Synchronization	11
4.4.2	Dimension Member Synchronization	11
4.4.3	Code Synchronization	11
4.4.4	Executing the Synchronize Stage	12
4.4.5	Exception Messages	13
4.5	Ledger Data Loader	14
4.5.1	Parameters	14
4.5.2	Undo Mechanism	15
4.5.3	Executing Undo Engine	15
4.5.4	Exception Messages	16
4.6	Data Slicing	16
4.6.1	Process flow	16
4.6.2	Executing the Data Slicing Function	17

5 Data Loaders

5.1	Dimension Loaders	1
5.1.1	Enhancements to Support Alphanumeric Code in Dimensions	3
5.1.2	Tables that are Part Of Staging	4
5.1.3	Populating STG_<DIMENSION>_HIER_INTF Table	4
5.1.4	Dimension Load Procedure	6
5.1.4.1	Dimension Leaf Member Set Up	7
5.1.4.2	Deletion of Dimension Members used in a Hierarchy	7
5.1.5	Setting up Dimension Loader	7
5.1.6	Executing the Dimension Load Procedure	9
5.1.7	Exception Messages	11
5.1.8	Executing the Dimension Load Procedure using Master Table approach	11
5.1.8.1	Approach 1	13
5.1.8.2	Approach 2	13
5.1.9	Updating DIM_<DIMENSION>_B <Dimension>_Code column with values from DIM_<DIMENSION>_ATTR table	14
5.1.10	Truncate Stage Tables	15
5.2	Simple Dimension Loader	16
5.2.1	Simple Dimension Stage Table	16
5.2.2	Configuration of Setup Tables	17
5.2.2.1	REV_DIMENSIONS_B Table	17
5.2.2.2	Setup Table Configuration Mapping	18
5.2.3	Executing the Simple Dimension Load Procedure	19

5.2.4	Exception Messages	20
5.3	Historical Rates Data Loader	20
5.3.1	Tables Related to Historical Rates	20
5.3.2	Populating Stage Tables	21
5.3.3	Executing the Historical Rates Data Loader T2T	21
5.3.4	Re-Load Of Historical Rates	22
5.4	Forecast Rate Data Loader	23
5.4.1	Forecast Rate Data Loader Tables	23
5.4.2	Populating Forecast Rate Stage Tables	24
5.4.3	Forecast Rate Loader Program	26
5.4.4	Executing the Forecast Rate Data Load	27
5.4.4.1	Forecast Rate Loader – Method 1	27
5.4.4.2	Forecast Rate Loader – Method 2	28
5.4.4.3	Forecast Rate Loader - Method 3	30
5.4.5	Exception Messages	30
5.5	Prepayment Rate Data Loader	30
5.5.1	Prepayment Rate Loader Tables	30
5.5.2	Prepayment Rate Data Loader	31
5.5.3	Executing the Prepayment Model Data Loader	33
5.5.4	Exception Messages	34
5.6	Stage Instrument Table Loader	35
5.6.1	Stage Tables	35
5.6.2	Populating Stage Tables	36
5.6.3	Mapping To OFSAA Processing Tables	36
5.6.4	Populating Accounts Dimension	38
5.6.5	Executing T2T Data Movement Tasks	39
5.6.6	Re-Load Of Instrument Data	39
5.7	Customer T2T Loading	40
5.7.1	Dependencies	40
5.7.2	Flow Diagram for Customer T2T	41
5.7.3	Executing T2T Data Movement Task	42
5.8	DIM_Party Population	42
5.8.1	Execution from ICC Batch	42
5.9	Instrument Summary Table	43
5.9.1	Mapping To OFSAA Summary Table	43
5.9.2	Dependencies	44
5.9.3	Executing T2T Data Movement Tasks	44
5.9.4	Re-Load Of Instrument Summary Data	45
5.10	Transaction Summary Table Loader	45
5.10.1	Stage Tables	45
5.10.2	Populating Stage Tables	46
5.10.3	Mapping To OFSAA Processing Tables	46

5.10.4	Dependencies	48
5.10.5	Executing T2T Data Movement Tasks	48
5.10.6	Re-Load Of Transaction Summary Data	48
5.11	Ledger Data Loader	49
5.11.1	Features and Limitations of the Load Procedure	50
5.11.2	Setup for the LEDGER_STAT load utility	51
5.11.2.1	FSI_DATA_IDENTITY insert/update during Ledger_load	53
5.11.2.2	Creating View on LEDGER_STAT table	53
5.11.2.3	Creating Load Table	54
5.11.2.4	Creating Unique Index on Load Table	54
5.11.2.5	Creating Views on Load Table	55
5.11.2.6	Setting up Global Temporary Table	55
5.11.2.7	Tables Related to LEDGER_STAT Load Procedure	56
5.11.2.8	Populating Stage Tables	56
5.11.2.9	Executing LEDGER_STAT Load Procedure	56
5.11.2.10	CALENDAR_MONTHS	57
5.11.2.11	FISCAL_ONE_MONTH	57
5.11.2.12	FISCAL_RANGE	57
5.11.2.13	Executing LEDGER_STAT Load Procedure for MULTI CURRENCIES	58
5.11.3	Exception Messages	58
5.11.4	Tables Cleanup After Truncation Of Ledger_Stat	59
5.12	Cash Flow Loader	59
5.12.1	Cash Flow Loader Table	60
5.12.2	Data Validation	63
5.12.3	Executing Cash Flow Loader	64
5.12.3.1	Method 1	64
5.12.3.2	Method 2	66
5.12.4	Exception Messages	68
5.13	Pricing Management Transfer Rate Population	70
5.14	ALMBI Transformation	71
5.15	Hierarchy Transformation	72
5.15.1	Executing the Hierarchy Flattening Transformation	73
5.16	Dim Dates Population	74
5.17	Fact Ledger Stat Transformation	74
5.18	Financial Element Dimension Population	74
5.18.1	Prerequisites	75
5.18.2	Tables Used by the Financial_Elem_Update Transformation	75
5.18.3	Executing the Financial_Elem_Update Transformation	75
5.18.4	Checking the Execution Status	76
5.19	Payment Pattern Loader	77
5.19.1	Executing the Payment Pattern Loader	79
5.19.2	Exception Messages	80

5.20	GAP Limits Loader	80
5.20.1	GAP Limits Loader Tables	81
5.20.2	Executing the Gap Limit loader	81
5.20.3	Exception Messages	81
5.21	Material Currency Identifier	83
5.21.1	Material Currency Identifier Tables	84
5.21.2	Executing the Material Currency loader	84
5.21.3	Exception Messages	85
5.22	Behaviour Pattern Loader	86
5.22.1	Executing the Behaviour Pattern Loader	90
5.22.2	Exception Messages	90

6 SCD Process

6.1	Type 1	1
6.2	Type 2	2
6.3	Prerequisites	2
6.4	Tables Used by the SCD Component	3
6.5	Executing the SCD Component	6
6.6	Checking the Execution Status	7

1

About This Content

This guide provides information on the Oracle Financial Services Data Model Utilities.

Audience

This guide is intended for the users of Oracle Financial Services Data Model Utilities.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at <http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc>.

Access to Oracle Support

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit <http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info> or visit <http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs> if you are hearing impaired.

Related Resources

See these Oracle resources:

- [Oracle Financial Services Asset Liability Management User Guide](#)
- [Oracle Financial Services Funds Transfer Pricing User Guide](#)
- [Oracle Financial Services Profitability Management](#)
- [Oracle Financial Services Cash Flow Engine Reference Guide](#)

Conventions

The following text conventions are used in this document.

Table 1-1 Conventions

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.

2

Introduction

This document contains various chapters related to Data Model utilities and Data Loaders available within the OFSAA Applications.

3

Object Management

This chapter details the steps involved in adding various client data objects into the model.

Topics:

- [Boundaries and Limitations](#)
- [Adding Dimension Tables and Key Dimension \(Leaf\) Registration](#)
- [Adding Custom Instrument Tables](#)
- [Adding Custom Transaction Tables](#)
- [Adding Custom Lookup Tables](#)
- [Adding Management Ledger Class tables](#)
- [Object Registration and Validation](#)
- [Defining Alternate Rate Output Columns](#)
- [User Defined Properties](#)
- [Modifying the Precision of Balance Columns In Ledger Stat](#)

3.1 Boundaries and Limitations

This section includes the details of Boundaries and Limitations used in Data Model Utilities.

Instrument Table - ID Numbers

ID numbers can have a maximum length of 25 digits.

Dimension Leaf Member Set Up

Dimension Leaf values can have a maximum of 14 digits.

Only 26 key (processing) dimensions are allowed in the database. Examples of seeded key leaf types are Common COA ID, Organizational Unit ID, GL Account ID, Product ID, and Legal Entity ID.

The maximum number of columns that the Oracle database allows in a unique index is 32. This is the overriding constraint. After subtracting IDENTITY_CODE, YEAR_S, ACCUM_TYPE_CD, CONSOLIDATION_CD, and ISO_CURRENCY_CD, this leaves 27 columns available for Key Processing Dimensions (leaf dimensions). BALANCE_TYPE_CD is now part of the unique index so this brings the maximum number of leaf columns down to 26.

Balances

Balances stored in Instrument tables are limited to 999,999,999,999.99. Balances stored in the LEDGER_STAT table are limited to 99,999,999,999.9999. The maximum precision for a balance used in a calculation process is 15 digits, with the range of 1.7e-308 to 1.7e+308. Calculation precision on larger numbers is compromised.

Rates

By default, rates stored in instrument tables are limited to 999.9999 and -999.9999. More precision can be achieved by increasing the number of decimals in the column. Internally, rates are stored with the same precision as balances.

Hierarchy Level Limitation

Hierarchies with 15+ level is NOT supported within any Enterprise Performance Management (EPM) - ALM/FTP/PFT/HM processes.

3.2 Adding Dimension Tables and Key Dimension (Leaf) Registration

The following section details the process in which users can add custom key dimensions to the OFSAA application.

Users can view the registered dimension within the AMHM screens. Also, users can add members and hierarchies for the dimension through AMHM screens. For Simple dimensions, entries should also be made in REV_DESCRIPTION_TABLES for each Instrument and Transaction Table in which that new dimension will occur. Registering a new Key Dimension (called as Leaf in OFSA 4.5) requires the following steps:

1. Add a set of dimension tables to store leaf values in ERwin model.
2. Add the key dimension column to required Entities in ERwin model.
3. Assign the Processing Key Column Property (Key Dimension Columns only).
4. Upload the model.
5. Register the Key Dimension. Modify Unique indexes (Key Leaf Dimension only).
6. Validate tables. Each of these steps is discussed in detail in the following sections.
7. Additionally, perform the following steps to avoid the errors running due to Allocations that use a Portfolio table.
8. Insert a row into REV_COLUMN_REQUIREMENTS for the custom dimension column.
9. Add the custom dimension to the Portfolio classification by following the instructions in the section Adding a new user defined column as a Portfolio column for use in all Instrument tables.

Note

For more information on limitation for number of key (processing) dimensions, see the Doc ID 1478920.1. If an allocation using a Portfolio with expression fails, then you should do manual entries for standard columns in REV_COLUMN_REQUIREMENTS through sql script which comes as part of installer.

3.2.1 Adding Dimension Tables

This section provides details on Key dimension and simple dimension tables.

Topics:

- [Key Dimension Table](#)
- [Simple Dimension Table](#)

3.2.1.1 Key Dimension Tables

Each key dimension contains a set of the following tables:

- DIM_<DIMENSION>_B: Stores leaf and node member codes within the dimension.
- DIM_<DIMENSION>_TL: Stores names of leaf and node and their translations.
- DIM_<DIMENSION>_ATTR: Stores attribute values for the attributes of the dimension.
- DIM_<DIMENSION>_HIER: Stores parent-child relationship of members and nodes that are part of hierarchies.

Note

Replace <DIMENSION> with the keyword representing the key dimension.

Seeded key dimension tables are present in 'ALM-FTP-PFT-HM-BSP – Dimensions' subject area within the ERwin model. The above tables need to be created for the new dimension. For more information on creating dimension tables in ERwin, see leaflet (Adding And Customizing Leaf.pdf).

Note

For ease of use, user can copy an existing set of dimension tables such as for ORG_UNIT dimension and rename the tables (in both physical and logical view) to represent the new dimension.

Table structure of one of the seeded key dimension is given following with remarks on how this can be used as the basis for modeling new key dimensions.

DIM_ORG_UNIT_ATTR

Stores the values of the attributes of the members (leaf and nodes) of the dimension.

Column Name	Logical Column Name	Datatype	NULL	Column Description	Remarks
ORG_UNIT_ID	Organization Unit ID	NUMBER(14)	NOT NULL	Leaf column which stores the id for the organization unit dimension	Column name and description should reflect the new dimension. Datatype and other constraints should be retained.

ORG_UNIT_DISPLAY_CODE	Organization Unit Display Code	NUMBER(14)	NULL	Leaf column which stores the display code for the organization unit dimension	Column name and description should reflect the new dimension. Datatype and other constraints should be retained.
ENABLED_FLAG	Enabled Flag	VARCHAR2(1)	NOT NULL	Store if the item is enabled or not	Internally used and hence should be retained in the same form within the new dimension table.
LEAF_ONLY_FLAG	Leaf or Node Flag	VARCHAR2(1)	NOT NULL	Indicates if the member is leaf only or not	Internally used and hence should be retained in the same form within the new dimension table.
DEFINITION_LANGUAGE	Definition Language	VARCHAR2(4)	NOT NULL	Language that is used to define	Internally used and hence should be retained in the same form within the new dimension table.
CREATED_BY	Created By	VARCHAR2(30)	NOT NULL	Indicates who created this item	Internally used and hence should be retained in the same form within the new dimension table.
CREATION_DATE	Creation Date	TIMESTAMP	NOT NULL	Indicates when was this item created	Internally used and hence should be retained in the same form within the new dimension table.
LAST_MODIFIED_BY	Last Modified By	VARCHAR2(30)	NOT NULL	Indicates who modified this item	Internally used and hence should be retained in the same form within the new dimension table.

LAST_MODIFIED_DATE	Last Modified Date	TIMESTAMP	NOT NULL	Indicates when was this item modified	Internally used and hence should be retained in the same form within the new dimension table.
ORG_UNIT_CODE	ORG_UNIT_CODE	VARCHAR2(20)	NULL	This column is used by staging and contains the alpha-numeric codes for each dimension member. Staging dimension table contains unique alpha-numeric codes and a unique numeric identifier is generated while loading into ALM-FTP-PFT-HM-BSP dimension table.	Column name and description should reflect the new dimension. Datatype and other constraints should be retained.

DIM_ORG_UNIT_TL

Stores the names and descriptions of the members (leaf and nodes) of the dimension in various languages.

Column Name	Logical Column Name	Datatype	NULL	Column Description	Remarks
LANGUAGE	Language	VARCHAR2(4)	NOT NULL	Language	Internally used and hence should be retained in the same form within the new dimension table.
ORG_UNIT_ID	Organization Unit ID	NUMBER(14)	NOT NULL	Leaf column which stores the id for the organization unit dimension	Column name and description should reflect the new dimension. Datatype and other constraints should be retained.

ORG_UNIT_NAME	Organization Unit Name	VARCHAR2(150)	NOT NULL	Leaf column which stores the name for the organization unit dimension	Column name and description should reflect the new dimension. Datatype and other constraints should be retained.
DESCRIPTION	Description	VARCHAR2(255)	NULL	Description of an Item	Internally used and hence should be retained in the same form within the new dimension table.
CREATED_BY	Created By	VARCHAR2(30)	NOT NULL	Indicates who created this item	Internally used and hence should be retained in the same form within the new dimension table.
CREATION_DATE	Creation Date	TIMESTAMP	NOT NULL	Indicates when was this item created	Internally used and hence should be retained in the same form within the new dimension table.
LAST_MODIFIED_BY	Last Modified By	VARCHAR2(30)	NOT NULL	Indicates who modified this item	Internally used and hence should be retained in the same form within the new dimension table.
LAST_MODIFIED_DATE	Last Modified Date	TIMESTAMP	NOT NULL	Indicates when was this item modified	Internally used and hence should be retained in the same form within the new dimension table.

DIM_ORG_UNIT_ATTR

Stores the values of the attributes of the members (leaf and nodes) of the dimension.

Column Name	Logical Column Name	Datatype	NULL	Column Description	Remarks
-------------	---------------------	----------	------	--------------------	---------

ORG_UNIT_ID	Organization Unit ID	NUMBER(14)	NOT NULL	Leaf column which stores the id for the organization unit dimension	Column name and description should reflect the new dimension. Datatype and other constraints should be retained.
ATTRIBUTE_ID	Attribute ID	NUMBER(22)	NOT NULL	Stores attribute id number for a member of a dimension	Internally used and hence should be retained in the same form within the new dimension table.
DIM_ATTRIBUTE_NUMERIC_MEMBER	Numeric Dimension Value	NUMBER(22)	NULL	This field stores the number values for the attribute of a member	Internally used and hence should be retained in the same form within the new dimension table.
DIM_ATTRIBUTE_VARCHAR_MEMBER	Varchar Dimension Value	VARCHAR2(30)	NULL	This field stores the varchar values for the attribute of a member	Internally used and hence should be retained in the same form within the new dimension table.
NUMBER_ASSIGNMENT_VALUE	Numeric Value Of A Member	NUMBER(22)	NULL	This field stores the number values for the attribute of a member	Internally used and hence should be retained in the same form within the new dimension table.
VARCHAR_ASSIGNMENT_VALUE	Varchar Member Value	VARCHAR2(1000)	NULL	This field stores the varchar values for the attribute of a member	Internally used and hence should be retained in the same form within the new dimension table.
DATE_ASSIGNMENT_VALUE	Date Value	DATE	NULL	Date value that is assigned	Internally used and hence should be retained in the same form within the new dimension table.

DIM_ORG_UNIT_HIER

Stores the parent-child relationship of various nodes and leaf within hierarchies of the dimension.

Column Name	Logical Column Name	Datatype	NULL	Column Description	Remarks
HIERARCHY_ID	Hierarchy ID	NUMBER(10)	NOT NULL	Unique Id that is generated for every hierarchy that is created	Internally used and hence should be retained in the same form within the new dimension table.
PARENT_ID	Parent ID	NUMBER(14)	NOT NULL	Column that store the id of the child member	Internally used and hence should be retained in the same form within the new dimension table.
CHILD_ID	Child Member ID	NUMBER(14)	NOT NULL	Store child id number for a dimension	Internally used and hence should be retained in the same form within the new dimension table.
PARENT_DEPTH_NUM	Parent Depth Number	NUMBER(14)	NOT NULL	Stores parent depth number	Internally used and hence should be retained in the same form within the new dimension table.
CHILD_DEPTH_NUM	Child Depth Number	NUMBER(14)	NOT NULL	Stores child depth number	Internally used and hence should be retained in the same form within the new dimension table.
DISPLAY_ORDER_NUM	Display Order Number	NUMBER(14)	NOT NULL	Stores the display order number for the member	Internally used and hence should be retained in the same form within the new dimension table.

SINGLE_DEPT H_FLAG	Single Depth Flag	VARCHAR2(1)	NOT NULL	Indicates if the hierarchy is of single depth or not	Internally used and hence should be retained in the same form within the new dimension table.
CREATED_BY	Created By	VARCHAR2(30)	NOT NULL	Indicates who created this item	Internally used and hence should be retained in the same form within the new dimension table.
CREATION_DA TE	Creation Date	TIMESTAMP	NOT NULL	Indicates when was this item created	Internally used and hence should be retained in the same form within the new dimension table.
LAST_MODIFIE D_BY	Last Modified By	VARCHAR2(30)	NOT NULL	Indicates who modified this item	Internally used and hence should be retained in the same form within the new dimension table.
LAST_MODIFIE D_DATE	Last Modified Date	TIMESTAMP	NOT NULL	Indicates when was this item modified	Internally used and hence should be retained in the same form within the new dimension table.

3.2.1.2 Simple Dimension Tables

Simple dimensions are created to store CODE and Descriptions. These tables are used by the User Interfaces to list values in drop downs / radio buttons, and so on. For Simple dimensions, entries should also be made in REV_DESCRIPTION_TABLES for each Instrument and Transaction Table in which that new dimension will occur. The entries in REV_DESCRIPTION_TABLES are used by Data Element Filters as well as the procedures for Synchronize Instruments and Synchronize Stage.

Each simple dimension contains a set of the following tables:

- CD table: Stores the members for a simple dimension.
- MLS table: Stores the members' multi lingual description.

If you use simple dimensions where _CD column is VARCHAR2, you will need to classify the tables as follows:.

FSI_ACCUMULATION_TYPE_CD, FSI_BILLING_METHOD_CD

The CD table should be classified as

295	Codes User Defined (base tbl)
198	Codes Reserved (base tbl)

The MLS table should be classified as

296	MLS Descriptions User Defined
197	MLS Descriptions Reserved

Table structure of one of these seeded simple dimensions is given in the following section with remarks on how this can be used as the basis for modeling new simple dimensions.

FSI_<DIM>_CD

Stores the ID of the members (leaf and nodes) of the dimension.

Column Name	Logical Column Name	Datatype	NULL	Column Description	Remarks
DIM_CD	Dimension Code	NUMBER(5)	NOT NULL	Leaf column which stores the code for the dimension.	Stores the Dimension Code.
LEAF_ONLY_FLAG	Leaf or Node Flag	VARCHAR2(1)	NOT NULL	Indicates if the member is leaf only or not	Internally used and hence should be retained in the same form within the new dimension table.
ENABLED_FLAG	Enabled Flag	VARCHAR2(1)	NOT NULL	Store if the item is enabled or not	Internally used and hence should be retained in the same form within the new dimension table.
DEFINITION_LANGUAGE	Definition Language	VARCHAR2(4)	NOT NULL	Language that is used to define	Internally used and hence should be retained in the same form within the new dimension table.
CREATED_BY	Created By	VARCHAR2(30)	NOT NULL	Indicates who created this item	Internally used and hence should be retained in the same form within the new dimension table.

CREATION_DATE	Creation Date	TIMESTAMP	NOT NULL	Indicates when was this item created	Internally used and hence should be retained in the same form within the new dimension table.
LAST_MODIFIED_BY	Last Modified By	VARCHAR2(30)	NOT NULL	Indicates who modified this item	Internally used and hence should be retained in the same form within the new dimension table.
LAST_MODIFIED_DATE	Last Modified Date	TIMESTAMP	NOT NULL	Indicates when was this item modified	Internally used and hence should be retained in the same form within the new dimension table.
<DIM>_DISPLAY_CD	Dimension Display Code	VARCHAR2()	NULL	Leaf column which stores the display code for the dimension	Column name and description should reflect the new dimension. Datatype and other constraints should be retained. The length of this column is customizable.

FSI_<DIM>_MLS

Stores the members' multi lingual description.

Column Name	Logical Column Name	Datatype	NULL	Column Description	Remarks
DIM_CD	Dimension Code	NUMBER(5)	NOT NULL	Leaf column which stores the code for the dimension.	Stores the Dimension Code.
LANGUAGE	Language	VARCHAR2(3)	NOT NULL	Language	Internally used and hence should be retained in the same form within the new dimension table.

<DIM>	Dimension	VARCHAR2(40)	NOT NULL	Name of the Dimension	Stores the name of the Dimension.
DESCRIPTION	Description	VARCHAR2(255)	NULL	Description of an Item	Internally used and hence should be retained in the same form within the new dimension table.
CREATED_BY	Created By	VARCHAR2(30)	NOT NULL	Indicates who created this item	Internally used and hence should be retained in the same form within the new dimension table.
CREATION_DATE	Creation Date	TIMESTAMP	NOT NULL	Indicates when was this item created	Internally used and hence should be retained in the same form within the new dimension table.
LAST_MODIFIED_BY	Last Modified By	VARCHAR2(30)	NOT NULL	Indicates who modified this item	Internally used and hence should be retained in the same form within the new dimension table.
LAST_MODIFIED_DATE	Last Modified Date	TIMESTAMP	NOT NULL	Indicates when was this item modified	Internally used and hence should be retained in the same form within the new dimension table.

Example for FSI_<DIM>_CD table:

```
CREATE TABLE <XXXXX>_FSI_<DIM>_CD -- ACME_FSI_ACCT_STATUS_CD (<DIM>_CD NUMBER(5)
-- ACCT_STATUS_CD ,LEAF_ONLY_FLAG VARCHAR2(1) ,ENABLED_FLAG
VARCHAR2(1) ,DEFINITION_LANGUAGE VARCHAR2(4) ,CREATED_BY
VARCHAR2(30) ,CREATION_DATE DATE ,LAST_MODIFIED_BY
VARCHAR2(30) ,LAST_MODIFIED_DATE DATE <dim>_display_CD VARCHAR2() );
```

Example for FSI_<DIM>_MLS table:

```
CREATE TABLE <XXXXX>_FSI_<DIM>_MLS -- ACME_FSI_ACCT_STATUS_CD (<DIM>_CD NUMBER(5)
-- ACCT_STATUS_CD ,LANGUAGE VARCHAR2(3) ,<DIM> VARCHAR2(40) --
ACCT_STATUS ,DESCRIPTION VARCHAR2(255) ,CREATED_BY VARCHAR2(30) ,CREATION_DATE
DATE ,LAST_MODIFIED_BY VARCHAR2(30) ,LAST_MODIFIED_DATE DATE );
```

3.2.2 Adding Dimension Column To Required Objects

Dimension column can be added to the following set of customer Data Objects:

- Tables classified as Instruments and Instrument Profitability
- Tables classified as Transaction Profitability
- Management Ledger table
- Result tables of Asset Liability Management

Types of Dimension can be: Ledger Only, Instruments Only, or Both. If the dimension is classified as 'Ledger Only', the dimension column needs to be added only to Ledger Stat table.

If the dimension is classified Instruments only, the dimension column needs to be added to instruments and Transactions tables. If the dimension is classified as 'Both', the dimension column needs to be added to Ledger Stat table and other tables classified as Instruments and Transactions.

- For adding key dimension column to tables that are classified as 'Instruments' and 'Instrument Profitability', add the column to `LEAF_COLUMNS` super-class table.
- For adding key dimension column to tables that are classified as 'Transaction Profitability', add the column to `TRANS_LEAF_COLUMNS` super-class table.
- For adding key dimension column to Ledger Stat table, add the column to `LEDGER_LEAF_COLUMNS` super-class table.

Note

Columns of super-class tables that are linked to sub-class table are rolled down to the sub-class table during 'Model Upload' operation.

If both super-class and sub-class tables have some common columns, then the sub-class table column is retained and the column from the super-class table will be ignored.

If both sub and super entities have common columns with same properties (such as datatype, size, and so on), then there is no issue with model upload process. If sub and super entities have the common columns with different properties, then there will issue with model upload process.

If you use Asset Liability Management, add the Dimension column directly to the following tables. Use the same column properties as the other dimension columns on the table when adding them. Add the column to the same indexes as the existing dimension columns.

- `FSI_O_RESULT_DETAIL_TEMPLATE`
- `FSI_O_CONS_DETAIL_TEMPLATE`
- `FSI_O_RESULT_MASTER`
- `FSI_O_CONSOLIDATED_MASTER`

3.2.3 Assigning Processing Key Property

1. 'Processing Key' is a column level User Defined Property (UDP) in ERwin model. This property can have two values – Yes or No. Only those objects where the column was added to the unique index are affected.

2. For tables classified as 'Transaction Profitability, this property needs to be set as 'Yes' for one or more of the key dimension columns.
3. For Ledger Stat table, this property needs to be set as 'Yes' for all key dimension columns. Assigning Processing Key Property is not required for Simple Dimension.

3.2.4 Uploading ERwin Model

1. ERwin model with the above changes needs to be uploaded in OFSAAI environment.
2. Uploading the model creates these additional tables and sets these properties within the atomic schema.
3. After upload, user can verify the changes in the schema as well as query OFSAAI metadata tables like REV_COLUMN_PROPERTIES for viewing properties assigned to each column.

For more information on data model upload process, see OFSAAI User Guide

3.2.5 Leaf Registration

Oracle Financial Services Analytical Applications Infrastructure (OFSAAI) provides an Leaf Registration procedure to add the new Key Dimension Column to the Dimensions metadata registry (REV_DIMENSIONS_B, REV_DIMENSIONS_TL).

3.2.5.1 Leaf Registration Procedure

This procedure performs the following:

- Registers key and simple dimension.
- Invalidates all Client Data Objects when key dimension is registered.

3.2.5.2 Executing Leaf Registration Procedure

You can execute this procedure either from SQL*Plus or from within a PL/SQL block or from Batch Maintenance window within OFSAAI framework.

To execute the Leaf Registration, follow these steps:

1. To run the procedure from SQL*Plus, login to SQL*Plus as the Schema Owner. The function requires 19 parameters. The syntax for calling the procedure is:

```
function rev_leaf_registration(batch_run_id varchar2, mis_date varchar2,
memDataType varchar2, dimName varchar2, description varchar2, memberBTableName
varchar2, memberTLTableName varchar2, hierarchyTableName varchar2,
attributeTableName varchar2, memberCol varchar2, memberDispCodeCol varchar2,
memberNameCol varchar2, memberDescCol varchar2, dimTypeCode varchar2,
simpleDimFlag varchar2, keyDimFlag char, writeFlag varchar2, catalogTableType
char, flattenedTableName in varchar2, membercodecol in varchar2 )
```

- **batch_run_id** : any string to identify the executed batch.
- **mis_date** : in the format YYYYMMDD.
- **memDataType** : member data type of Dimension as in NUMBER, VARCHAR2, CHAR.
- **dimName** : name of the dimension to be added (less than 21 chars).
- **description** : description of the dimension (less than 255 chars).

- **memberBTableName** : Member Base Table Name input as either null or a value with suffix '_CD' or '_B'.
- **memberTLTableName** : Member TL Table Name input as either null or name of the table.
- **hierarchyTableName** : Hierarchy Table Name input as either null or name of the table.
- **attributeTableName** : Attribute Table Name input as either null or name of the table.
- **memberCol** : Member Column Name input as either null or name of the column.
- **memberDispCodeCol** : Member Display Code Column Name input as either null or name of the column. For simple dimensions, enter the same field as the memberCol. Do not user display column for simple dimensions.
- **memberNameCol** : Member Name Column input as either null or name of the column.
- **memberDescCol** : Member Description Column input as either null or name of the column.
- **dimTypeCode** : Code for the dimension Type as in 'PROD for product type', 'ORGN for Organizational Unit', 'CCOA for Common Chart of Accounts', 'FINELE for Financial Element', 'GL for General Ledger Account', 'OTHER for any other type'.
- All user defined dimensions will have `DIMENSION_TYPE_CODE` as 'OTHER'. User defined dimensions which are product related will have `DIMENSION_TYPE_CODE` as 'PROD'.
- **simpleDimFlag** : 'Y' or 'N' to determine Simple Dimension.
Simple dimensions are created to store CODE and Descriptions. These tables are used by the User Interfaces to list values in drop downs / radio buttons, and so on. Simple dimensions are not reverse populated.
Example: Country, Currencies, Customer Type.
- **keyDimFlag** : 'Y' or 'N' to determine Key Dimension.
Key dimensions are dimensions which get reverse populated to the legacy tables.
Example: Product, Org Unit, General Ledger.
- **writeFlag** : 'Y' or 'N' to determine whether Dimension should appear in drop down list in Dimension Management > Members.
- **catalogTableType** : 'L', 'B', or 'I' to determine table type for key dimensions.
- For a Simple Dimension, this value should be set to Null.
- **flattenedTableName** : Flattened Table Name input as either null or name of the table.
- **membercodecol**: Alphanumeric Code column. Populates the `MEMBER_CODE_COLUMN` column in `REV_DIMENSIONS_B`. The value provided should be a valid code column from the relevant `DIM_<DIMENSION>_B` (key dimension) or `FSI_<DIM>_CD` (simple dimension) table. For simple dimensions use the display code column.

Example for Key Dimension:

```
Declare num number; Begin num := rev_leaf_registration('BATCH_NO_01',
'20101216', 'NUMBER', 'SIMPLE DIMENSION', 'SIMPLE DIMENSION DESC',
'FSI_DIM_SIMPLE_CD', 'FSI_DIM_SIMPLE_MLS', 'null', 'null', 'SIMPLE_CD',
'SIMPLE_CD', 'SIMPLE_NAME_Dim', 'SIMPLE_DESCRIPTION', 'OTHER', 'Y', 'N', 'Y',
'B', 'FLATTEN_PROD_TABLE', 'SIMPLE_DISPLAY_CODE'); End; Example for Simple
Dimension: Declare num number; Begin num :=
rev_leaf_registration('BATCH_NO_01', '20101216', 'NUMBER', 'SIMPLE DIMENSION',
'SIMPLE DIMENSION DESC', 'FSI_DIM_SIMPLE_CD', 'FSI_DIM_SIMPLE_MLS', 'null',
```

```
'null', 'SIMPLE_CD', 'SIMPLE_DISPLAY_CODE', 'SIMPLE_NAME_Dim',
'SIMPLE_DESCRIPTION', 'OTHER', 'Y', 'N', 'Y', 'B', 'FLATTEN_PROD_TABLE'); End;
```

2. To execute the procedure from OFSAAI Batch Management, create a new Batch with the Task as TRANSFORM DATA and specify the following parameters for the task:
 - **Datastore Type:** Select appropriate datastore from list
 - **Datastore Name:** Select appropriate name from the list
 - **IP address:** Select the IP address from the list
 - **Rule Name:** batch_leaf_registration
 - **Parameter List:** Member Data type , Dimension Name, Dimension Description, Member Base Table Name, Member Translation Table Name, Hierarchy Table Name, Attribute Table Name, Member Column , Member Display Code Column, Member Name Column, Member Description Column , Dimension Type Code , Simple Dimension Flag , Key Dimension Flag , writeFlag, Catalog Table Type , Flatten Table Name

3.2.6 Modify Unique Indexes

1. For tables of 'Transaction Profitability' classification, key dimension column can be part of the unique index. If this column is intended to be part of the unique index, alter the unique index in the schema.
2. For Ledger Stat table, all key dimension columns should form part of the unique index. Hence, alter the unique index in the schema to include this column.

3.2.7 Executing Object Registration Validation

- Since leaf registration invalidates all Client Data Objects, Object Registration Validation procedure needs to be executed to validate the required tables.

For more information on Executing Object Registration Validation, see Object Registration and Validation.

3.3 Adding Custom Instrument Tables

Instrument and Account objects are tables storing financial services information about customers and accounts. These are most commonly used objects for OFSAA processing and reporting operations. There are seeded instrument tables that are packaged as part of each OFSAA.

You can customize or remove any of them during implementation. In some cases, you might also require to add a custom instrument table. For Simple dimensions, entries should also be made in REV_DESCRIPTION_TABLES for each Instrument Table in which that new dimension will occur. The entries in REV_DESCRIPTION_TABLES are used by Data Element Filters as well as the procedures for Synchronize Instruments and Synchronize Stage.

Topics:

- [Super-class Entities](#)
- [Creating Custom Instrument Table](#)
- [Setting Table Classifications](#)
- [Unique Index](#)

- [Portfolio Selection](#)
- [Object Registration Validation](#)

3.3.1 Super-class Entities

Most instrument tables are used for OFSAA processing. OFSAA processing mandates the instrument table to have a certain set of columns.

These columns have been put together in super-class entities. The following are the seeded super-class entities:

- **LEAF_COLUMNS**: contains the key dimension columns that are part of the Instrument tables.
- **BASIC_INSTRUMENT_REQ**: contains the basic instrument columns like `ID_NUMBER`, `IDENTITY_CODE`, and so on.
- **MULTI_CUR_REQ**: contains the columns required for multi-currency processing.
- **CASH_FLOW_EDIT_REQ**: contains the columns required for Cash flow Edit processing.
- **CASH_FLOW_PROC_REQ**: contains the columns required for Cash flow processing.
- **TP_BASIC_REQ**: contains the columns required for Transfer Pricing processing.
- **TP_OPTION_COSTING_REQ**: contains the columns required for Transfer Pricing Option Cost processing.
- **PORTFOLIO_REQ**: contains the columns required for Portfolio table classification.
- **TRANS_LEAF_COLUMNS**: contains the key dimension columns that are part of the transaction tables.
- **LEDGER_LEAF_COLUMNS**: contains the key dimension columns that are part of the Ledger Stat table.
- **BASIC_LEDGER_CLASS_REQ**: contains the columns required for Ledger Class tables example `_Ledger`.

Note

Column Precision for Instrument Table:

You can increase the size of the columns to make them hold a value of larger precision, but the new size will impact FTP and ALM engines as follow:

Values/fields read by the engine are restricted to the size that the c++ variables can hold within the engine memory. In fact, having a value larger than the allowed precision can cause the engine to read the value incorrectly.

Changing the size of the fields that these engines write into does not affect the precision of the results

An upgrade could rollback such changes unless you remember to do a model merge.

Instrument table can link to any of the above super-class entities based on its purpose. For example, if the instrument table is used for Cash Flow Processing, then this table should be linked to the following super-class entities:

- `BASIC_INSTRUMENT_REQ`

- MULTI_CUR_REQ
- LEAF_COLUMNS
- CASH_FLOW_EDIT_REQ
- CASH_FLOW_PROC_REQ

Note

CASH_FLOW_PROC_REQ is required for all instrument tables where conditional assumptions will be applied.

Refer to the following mapping table that specifies the list of super-class entities required for each table classification:

Type of Client Data Object	Table Classification	List of Super-class entities
Instrument	Instrument	BASIC_INSTRUMENT_REQ LEAF_COLUMNS
Instrument	ALM Standard	BASIC_INSTRUMENT_REQ LEAF_COLUMNS MULTI_CUR_REQ CASH_FLOW_EDIT_REQ CASH_FLOW_PROC_REQ
Instrument	TP Cash Flow	BASIC_INSTRUMENT_REQ LEAF_COLUMNS MULTI_CUR_REQ CASH_FLOW_EDIT_REQ CASH_FLOW_PROC_REQ TP_BASIC_REQ
Instrument	TP Non-Cash Flow	BASIC_INSTRUMENT_REQ LEAF_COLUMNS MULTI_CUR_REQ CASH_FLOW_EDIT_REQ TP_BASIC_REQ
Instrument	TP Option Costing	BASIC_INSTRUMENT_REQ LEAF_COLUMNS MULTI_CUR_REQ CASH_FLOW_EDIT_REQ TP_BASIC_REQ TP_OPTION_COSTING_REQ
Instrument	Instrument Profitability	BASIC_INSTRUMENT_REQ MULTI_CUR_REQ PORTFOLIO
Instrument	Portfolio	BASIC_INSTRUMENT_REQ LEAF_COLUMNS MULTI_CUR_REQ PORTFOLIO
Transaction	Transaction Profitability	TRANS_LEAF_COLUMNS
Ledger Stat	Ledger Stat	LEDGER_LEAF_COLUMNS

3.3.2 Steps to Create Custom Instrument Table

The following are the steps involved in creating a custom instrument table:

1. Create a new subject area within the ERwin model.
2. Move the required super-class tables as part of the subject area.

3. Create the custom instrument table in ERwin. Specify logical name, physical name and description for the table. Define any columns that do not come from any of the standard super-class tables as part of the custom instrument table. Specify logical, physical names, domain and other column properties for each column.
4. Create subtype relationship between the custom instrument table and various super-class entities.

Note

User defined tables (custom) tables should not have "PORTFOLIO" keyword in the name of the table.

3.3.3 Setting Table Classifications

Table Classifications can be set for any Client Data Object. Table classification set against each Client Data Object is validated through Object Registration Validation process.

The following are the steps involved in setting table classification properties for the custom instrument table:

1. Choose Physical View within the ERwin model.
2. Go to UDP tab within Table Properties window.
3. Specify 'Yes' against required Table Classifications properties.

Once the model is prepared using the above steps, user should upload the ERwin model. After uploading the model, user can check if the custom instrument table has been created in the schema with columns from super-class entities that have been linked to the custom instrument table as well as the columns present in the custom instrument table. Model upload also creates metadata entries within the following Object Registration tables:

- REV_TABLES_B: Contains the list of table names. REV_TABLES_TL: contains the list of table display names and descriptions in various languages.
- REV_TAB_COLUMNS: contains the list of column names.
- REV_TAB_COLUMNS_MLS: contains the list of column display names and descriptions in various languages.
- REV_COLUMN_PROPERTIES: stores the column properties associated with each column.
- REV_TABLE_CLASS_ASSIGNMENT: stores the table classification associated with each table.

Note

In case custom instrument table contains the column in the same name as that of the super-class table, then column present in the custom instrument table will take precedence over the equivalent column of the super-class table. In case multiple super-class tables contain the same column, user should ensure that all the columns have same datatype, as any column can be selected and it is not resolved in any specific order. Physical order of the columns within the custom instrument table is determined in the following way:

- a. Columns present in the custom instrument table.
- b. Columns present in each of the linked super-class table. In case multiple super-class tables are linked to the custom instrument table, columns are rolled down from all super-class tables without any specific order.

Within any table, ERwin maintains three different column orders:

- a. Logical Order: Order of the columns as seen in Logical view of the model
- b. Physical Order: Order of the columns as seen in Physical view of the model.
- c. Database Order: Order of the columns as seen in the Database schema.

3.3.4 Unique Index

Instrument tables require unique index on ID_NUMBER and IDENTITY_CODE column. This unique index needs to be created on the custom instrument table, post-model upload operation.

Transaction tables require unique index on ID_NUMBER, IDENTITY_CODE and one of the key dimension columns. This unique index needs to be created on the custom transaction table, post-model upload operation.

Note

The unique index may not contain any non-Key Dimension columns other than ID_NUMBER and IDENTITY_CODE.

3.3.5 Portfolio Selection

It enables users to define rules with a cross-instrument definition, since the Portfolio table classification contains columns (potentially including user-defined columns) that are common to all instruments. Any other specific instrument table selection (such as Mortgages, and so on) would limit the rule definition to the specific instrument table. During processing of a Portfolio selection from an assumption rule, the engine will substitute the name of a specific table (that is, instruments selected in the parent process rule).

3.3.5.1 Adding a new User Defined column as a Portfolio column

This section provides details on adding a new user defined column as a Portfolio column for use in all Instrument tables. Portfolio is available as a table selection in various modules including Infrastructure objects such as Filters and Expression rules. It is also available in application objects such as Profitability Management Allocation rules, and so on. To add a new user-defined column as a Portfolio column, use the following steps:

1. Include the column in the PORTFOLIO super-type table in the Erwin Data Model to ensure that the column rolls down to all subtype tables.
2. Complete incremental model upload to add the column to all subtype Portfolio tables.
3. Manually insert a row into the Atomic schema REV_PROPERTY_COLUMNS table with TABLE_PROPERTY_CD = 40:

For example, if your new column is APPLE_BRANCH_CD

```
Insert into REV_PROPERTY_COLUMNS(TABLE_PROPERTY_CD,COLUMN_NAME,PROTECTED_FLG)
values(40,'APPLE_BRANCH_CD',1);COMMIT;
```

Data Element Filters created with custom columns will appear in Conditional Assumptions (within Funds Transfer Pricing) when properly registered. When creating a custom column in Instrument table(s)/Cash Flow related table(s), that is, those containing BASIC_INSTRUMENT_REQ/CASH_FLOW_PROC_REQ in the data model, the custom column must have one of the Domains listed as follows: BALANCE, CHAR, CODE, CODE_NUM, DATE, DESCRIPTION, FLAG, NUMBER, NUMERIC, RATE, SWITCH, VARCHAR2, CHAR_RANGE, PCT.

3.3.6 Object Registration Validation

Since leaf registration invalidates all Client Data Objects, Object Registration Validation procedure needs to be executed to validate the required tables. For more information on Object Registration Validation procedure, see Object Registration and Validation

3.4 Adding Custom Transaction Tables

Transaction tables are used within Profitability Management processing. There are seeded transaction tables that are packaged as part of Profitability Management application. You can customize or remove any of them during implementation. In some cases, you might also require to add a custom transaction table.

For Simple dimensions, entries should also be made in REV_DESCRIPTION_TABLES for each Transaction Table in which that new dimension will occur. The entries in REV_DESCRIPTION_TABLES are used by Data Element Filters as well as the procedures for Synchronize Instruments and Synchronize Stage.

Topics:

- [Super-class Entities](#)
- [Steps In Creating Custom Transaction Table](#)
- [Setting Table Classifications](#)
- [Unique Index](#)
- [Object Registration Validation](#)

3.4.1 Super-class Entities

Profitability Management processing mandates the transaction table to have a certain set of columns. These columns have been put together in super-class entities.

The following are the seeded super-class entities:

TRANS_LEAF_COLUMNS : contains the key dimension columns that are part of the Transaction tables.

3.4.2 Steps to Create Custom Transaction Table

The following are the steps involved in creating a custom transaction table:

1. Create a new subject area within the ERwin model.
2. Move `TRANS_LEAF_COLUMNS` into the new subject area.
3. Create the custom transaction table in ERwin.
4. Specify logical name, physical name and description for the table.
5. Define any columns that do not come from any of the standard super-class tables as part of the custom transaction table.
6. Specify logical, physical names, domain and other column properties for each column.
7. Create subtype relationship between the custom transaction table and `TRANS_LEAF_COLUMNS` super-class entity.

3.4.3 Setting Table Classifications

Table Classifications can be set for any Client Data Object. Table classification set against each Client Data Object is validated through Object Registration Validation process.

The following are the steps involved in setting table classification properties for the custom transaction table:

1. Choose Physical View within the ERwin model.
2. Go to UDP tab within Table Properties window.
3. Specify 'Yes' for 'Transaction Profitability' user defined property.

Once the model is prepared using the above steps, user should upload the ERwin model. After uploading the model, user can check if the custom transaction table has been created in the schema with columns from super-class entities that have been linked to the custom transaction table as well as the columns present in the custom transaction table. Model upload also creates metadata entries within the following Object Registration tables:

- `REV_TABLES_B`: Contains the list of table names.
- `REV_TABLES_TL`: contains the list of table display names and descriptions in various languages.
- `REV_TAB_COLUMNS`: contains the list of column names.
- `REV_TAB_COLUMNS_MLS`: contains the list of column display names and descriptions in various languages.
- `REV_COLUMN_PROPERTIES`: stores the column properties associated with each column.
- `REV_TABLE_CLASS_ASSIGNMENT`: stores the table classification associated with each table.
- `REV_TABLE_CLASSIFICATION_TL`: contains the list of table classification, language name and description associated with each table.
- `REV_TABLE_CLASSIFICATION_B`: contains the list of table classification.

Note

For Mortgages, TABLE_CLASSIFICATION_CD = 618 and TABLE_CLASSIFICATION = 'Mortgages' in REV_TABLE_CLASSIFICATION_TL and REV_TABLE_CLASSIFICATION_B tables. For Embedded Options, TABLE_CLASSIFICATION_CD = 617 and TABLE_CLASSIFICATION = 'Embedded Options' in REV_TABLE_CLASSIFICATION_TL and REV_TABLE_CLASSIFICATION_B tables.

- REV_COLUMN_PROPERTY_CD: contains the column property Ids and currency basis associated with each column.
- REV_COLUMN_PROPERTY_MLS: contains the column properties, language name and description associated with each column.

Note

For Economic Value, COLUMN_PROPERTY_CD = 86 and COLUMN_PROPERTY = 'Economic Value' in REV_COLUMN_PROPERTY_MLS and REV_COLUMN_PROPERTY_CD tables. In case custom transaction table contains the column in the same name as that of the super-class table, then column present in the custom transaction table will take precedence over the equivalent column of the super-class table. Physical order of the columns within the custom transaction table is determined in the following way: Columns present in the custom transaction table. Columns present in each of the linked super-class table. Within any table, ERwin maintains three different column orders:

- **Logical Order:** Order of the columns as seen in Logical view of the model.
- **Physical Order:** Order of the columns as seen in Physical view of the model.
- **Database Order:** Order of the columns as seen in the Database schema.

3.4.3.1 Setting Processing Key Property

'Processing Key' user defined property needs to be set for the following columns within the Ledger Class table: Leaf columns that are part of the unique index

The following are the steps to set this property in ERwin:

1. Choose Physical View within the ERwin model.
2. Choose BASIC_LEDGER_CLASS_REQ super-class table.
3. Choose the leaf column that needs to be set 'Processing Key' property.
4. Go to UDP tab in Column Properties window for this column.
5. Specify 'Yes' against 'Processing Key' user-defined property.
6. Choose the custom Ledger Class table.
7. Go to UDP tab in Column Properties window for the columns.
8. Specify 'Yes' against 'Processing Key' user-defined property.

3.4.4 Unique Index

Transaction tables require unique index on the following columns:

- ID_NUMBER
- IDENTITY_CODE
- At-least one of the key dimension columns.

This unique index needs to be created on the custom transaction table, post-model upload operation.

3.4.5 Object Registration Validation

Since leaf registration in-validates all Client Data Objects, Object Registration Validation procedure needs to be executed to validate the required tables. For more information on Object Registration Validation procedure, see Object Registration and Validation.

3.5 Adding Custom Lookup Tables

Lookup tables are used within OFSAA Profitability Management application.

Lookup tables have to be created and registered within OFSAAI, in order to display them in Lookup Table Driver definition of OFSAA Profitability Management application.

Topics:

- [Creating Lookup Table](#)
- [Setting Column Properties](#)
- [Setting Table Classifications](#)
- [Registering Lookup Tables and Validation](#)
- [Lookup Table Driver Definition](#)

3.5.1 Steps to Create Lookup Table

Lookup table has to be created in the ERwin model.

The following are the steps:

1. Open the ERwin model in ERwin Data Modeler tool.
2. Create a new subject area.
3. Create a table and add columns to the table.
4. Lookup table needs to at-least have one primary key column.
5. Lookup table needs to at-least have one numeric non-key column.
6. Such numeric columns will be the return value of the lookup.
7. Specify logical names, comments and primary key for the table.
8. Specify logical names, domains and comments for the column.
9. Domains for the columns can be LEAF, BALANCE, RATE and so on. Save the model.

3.5.2 Setting Column Properties

'Processing Key' is a column level User Defined Property (UDP) in ERwin model.

This property can have two values – Yes or No. 'Processing Key' property needs to be set for all the primary key columns of the lookup table.

That means, this property needs to be set for all columns of the Lookup table except the Return type columns.

'Balance Range' is a column level User Defined Property (UDP) in ERwin model. This property can have two values – Yes or No. Balance Range property needs to be set for the columns that can have range values in the lookup. The following are the steps for setting the above properties:

1. Open the ERwin model in ERwin Data Modeler tool.
2. Go to the Subject Area where lookup table was created.
3. Choose the table and open the columns of the table.
4. Go to **UDP** tab within the column properties for each column.
5. Specify the value for the required user defined properties.

UDP_LOOKUP_RANGE_MINIMUM is a column level User Defined Property (UDP) in ERwin model. This property can have two values – Yes or No. UDP_LOOKUP_RANGE_MINIMUM property needs to be set for the columns that can have minimum values for a range in the lookup. UDP_LOOKUP_RANGE_MAXIMUM is a column level User Defined Property (UDP) in ERwinModel. This property can have two values – Yes or No. UDP_LOOKUP_RANGE_MAXIMUM property needs to be set for the columns that can have maximum values for a range in the lookup.

We do not need to set any other properties corresponding to the other column types viz. the Exact Match or the Hierarchy Match column types, as these column properties are to be set from the Lookup Table Driver definition screen.

6. Save the model.

3.5.3 Setting Table Classifications

Table Classifications can be set for any Client Data Object. Table classification set against each Client Data Object is validated through Object Registration Validation process.

The following are the steps involved in setting table classification properties for the custom lookup table:

1. Choose Physical View within the ERwin model.
2. Go to UDP tab within Table Properties window.
3. Specify Yes for PA Lookup Tables user-defined property.

3.5.4 Registering Lookup Tables and Validation

To register Lookup tables and validation, follow these steps:

1. Upload the model.
2. Execute the object registration validation.

3.5.5 Lookup Table Driver Definition

Post registration and validation, the lookup table is available within Lookup Table Driver definition of OFSAA Profitability Management application.

Following is the criteria for columns to be displayed in the Source - Lookup Mapping grid:

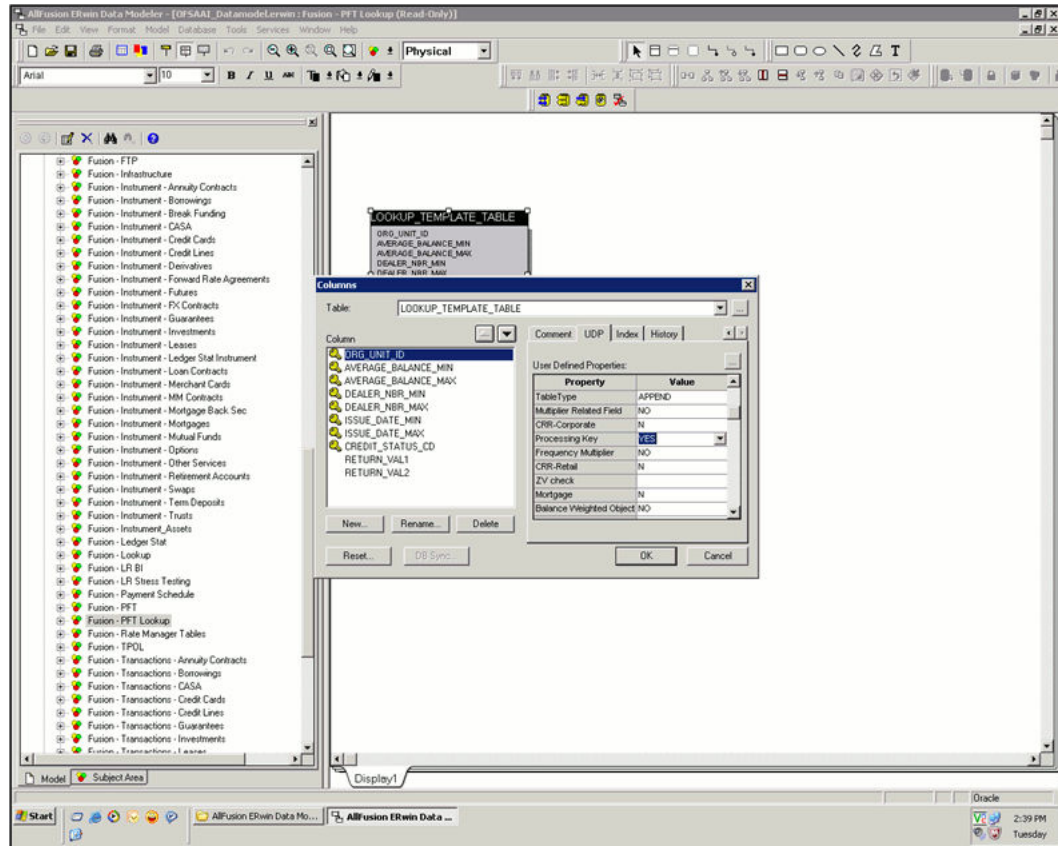
- Column needs to be Primary Key or be part of composite primary key.

- Processing Key user defined property should be set for the column under UDP tab as shown following.

Mapping of Column to Processing Key

Following is the criteria for columns to be part of the Range:

Figure 3-1 Mapping of Column to Processing Key

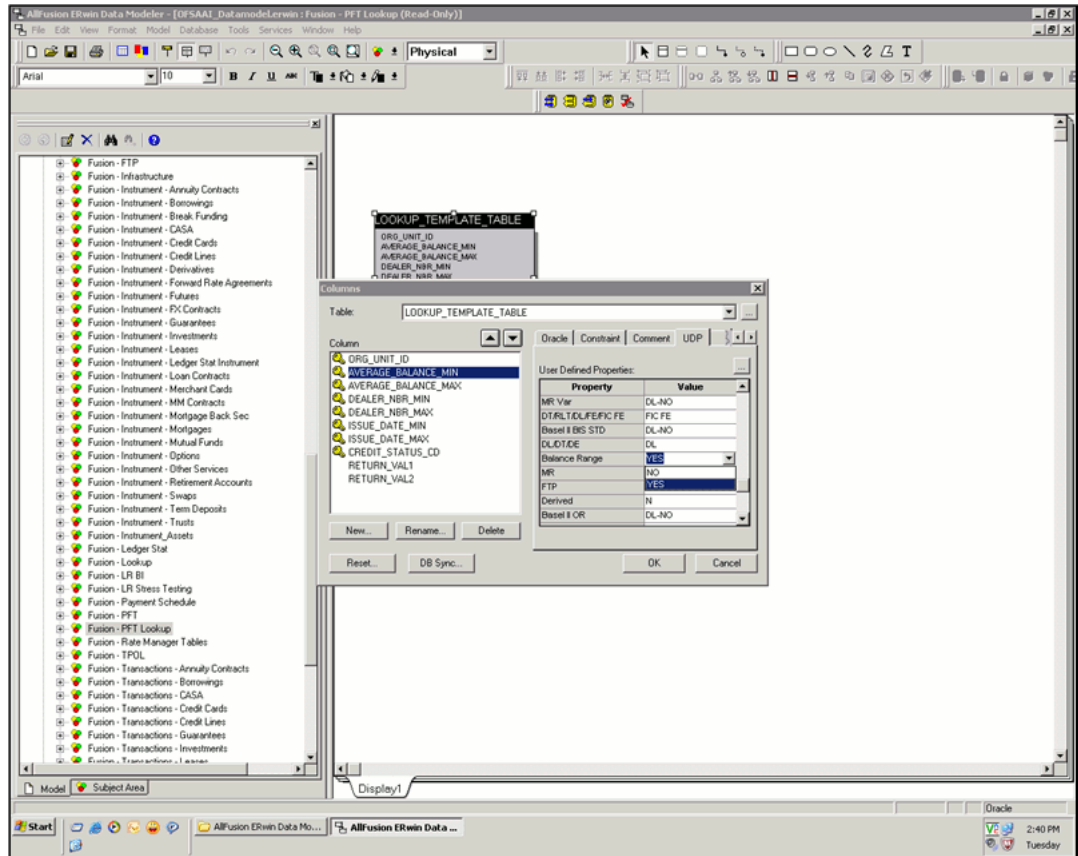


- Balance Range user defined property should be set for the column under UDP tab as displayed.

Mapping of Column for Range Property

Following is the criteria for columns to be part Lookup Return Value:

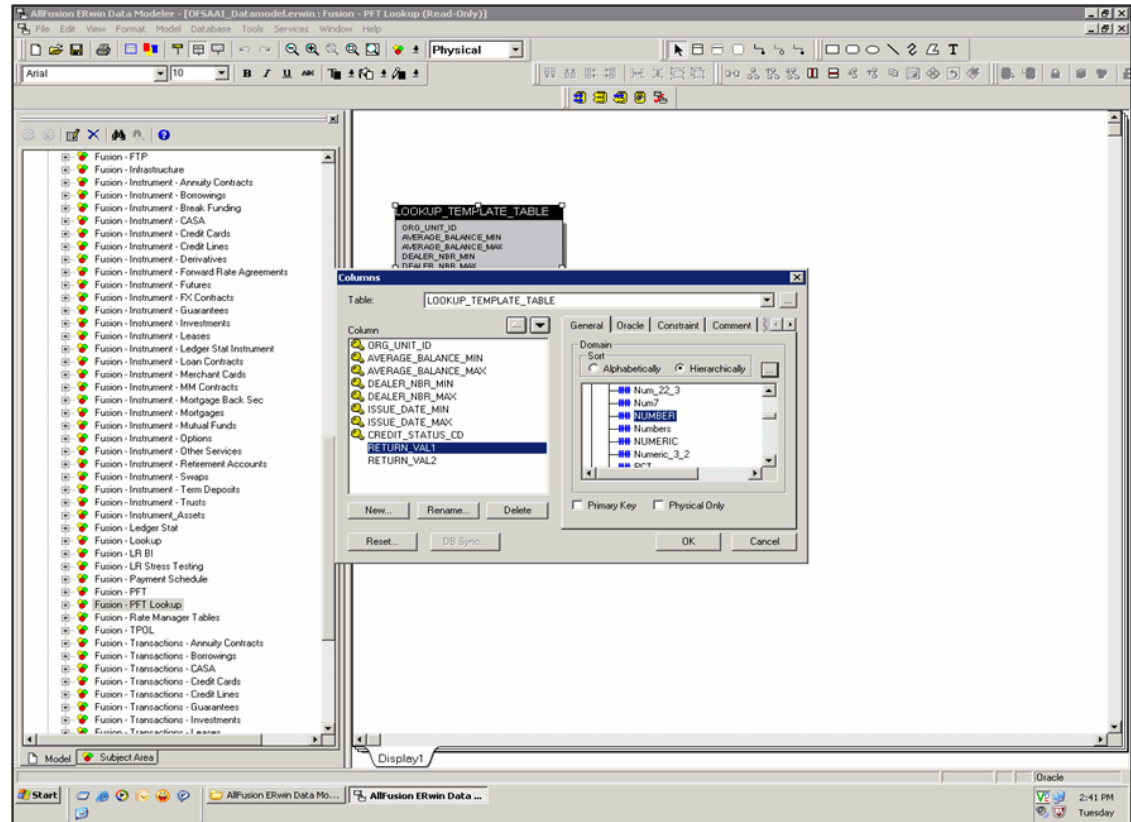
Figure 3-2 Mapping of Column for Range Property



- Column should not be primary key/processing key or be part of composite primary key.
- Column domain should be defined as NUMBER under General Tab as displayed.

Mapping of Column for Look up Return Value

Figure 3-3 Mapping of Column for Look up Return Value



3.6 Adding Management Ledger Class tables

Beginning with release 8, OFSAA Profitability Management supports a Management Ledger table class. Management Ledger tables provide substantially identical functionality to the traditional Ledger/Stat table. Like Ledger/Stat, you may also customize the dimensionality of Management Ledger tables. Additionally, in addition to the seeded Management Ledger table (FSI-D-MANAGEMENT-LEDGER), you may also construct and customize additional ledger tables of the Management Ledger table class.

The seeded FSI-D-MANAGEMENT-LEDGER table is loaded from STG_GL_DATA via a standard T2T rule (T2T_MANAGEMENT_LEDGER). If you choose to build additional Management Ledger tables, you will need to clone T2T_MANAGEMENT_LEDGER to target your new ledger table.

Note

All Custom Dimensions should be added to the PK for STG_GL_DATA.

And all Custom Dimensions should be added to Unique Index of FSI_D_MANAGEMENT_LEDGER.

The following topics are covered in this section:

- [Super-class Entities](#)

- [Creating Custom Ledger Class Table](#)
- [Setting Table Classifications](#)
- [Setting Processing Key Property](#)
- [Unique Index](#)
- [Object Registration Validation](#)

3.6.1 Super-class Entities

Profitability processing mandates the Ledger class table to have a certain set of columns. These columns have been put together in super-class entities.

The following are the seeded super-class entities:

- `BASIC_LEDGER_CLASS_REQ`: Contains the columns required for Ledger Class tables.
- The `FSI_D_MANAGEMENT_LEDGER.ENTERED_BALANCE` column stores entered or transacted balances that correspond to the local currency in which the transactions were booked in your General Ledger.
- The `FSI_D_MANAGEMENT_LEDGER.FUNCTIONAL_BALANCE` column stores balances in the Functional currency of your General Ledger

Note

When Initially loading data from Staging fo Management Ledger, Entered and Functional currency balances should correspond to the values originally booked into your General Ledger

For a mono-currency implementation:

- The Entered balance should equal the Functional balance on each Management Ledger row
- The `ISO_CURRENCY_CD` for each Management Ledger row should match your Functional Currency (from `FSI_DB_INFO`).

For multi-currency implementations:

- When a Management Ledger row's `ISO_CURRENCY_CD` is the same as your Functional Currency, the Entered balance will equal the Functional balance
- The Entered balance will equal the Functional balance for all statistical rows (`ISO_CURRENCY_CD = 002`)
- Generally, the Entered balance will NOT equal the Functional balance for Management Ledger rows where `ISO_CURRENCY_CD` is different than your Functional Currency
- The ratio of Entered balance to Functional balance - the observed exchange rate for the row -- is determined within your source General Ledger. Typically, ending balances will match end-of-month exchange rates while P&L balances may reflect weighted monthly average exchange rates.

Note

For more information on how Entered & Functional balances are used with applications, see the Multicurrency in [the OFS Profitability Management User Guide](#).

3.6.2 Creating Custom Ledger Class Table

The following are the steps involved in creating a custom Ledger Class table:

1. Create subtype relationship between the custom Ledger Class table and BASIC_LEDGER_CLASS_REQ super-class entity.

Note

FINANCIAL_ELEM_ID, ORG_UNIT_ID, GL_ACCOUNT_ID, COMMON_COA_ID, and LEGAL_ENTITY must be present in any Management Ledger table. Create a new subject area within the ERwin model.

2. Create the custom Ledger Class table in ERwin.
3. Specify logical name, physical name and description for the table.
4. Define any columns that do not come from any of the standard super-class tables as part of the custom Ledger Class table.
5. Specify logical, physical names, domain and other column properties for each column.
6. Move BASIC_LEDGER_CLASS_REQ into the new subject area.

3.6.3 Setting Table Classifications

Table Classifications can be set for any Client Data Object. Table classification set against each Client Data Object is validated through Object Registration Validation process.

The following are the steps involved in setting table classification properties for the custom Ledger Class table:

1. Choose Physical View within the ERwin model.
2. Go to UDP tab within Table Properties window.
3. Specify 'Yes' for 'Ledger Class' user defined property.

Once the model is prepared using the above steps, user should upload the ERwin model. After uploading the model, user can check if the custom Ledger Class has been created in the schema with columns from super-class entities that have been linked to the custom Ledger Class table as well as the columns present in the custom Ledger Class table. Model upload also creates metadata entries within the following Object Registration tables:

- REV_TABLES_B: Contains the list of table names.
- REV_TABLES_TL: contains the list of table display names and descriptions in various languages.
- REV_TAB_COLUMNS: contains the list of column names
- REV_TAB_COLUMNS_MLS: contains the list of column display names and descriptions in various languages.
- REV_COLUMN_PROPERTIES: stores the column properties associated with each column.

- **REV_TABLE_CLASS_ASSIGNMENT**: stores the table classification associated with each table.
 - In case custom Ledger Class table contains the column in the same name as that of the super-class table, then column present in the custom Ledger Class table will take precedence over the equivalent column of the super-class table.
 - Physical order of the columns within the custom Ledger Class table is determined in the following way:
 - Columns present in the custom Ledger Class table. Columns present in each of the linked super-class table.

Within any table, ERwin maintains three different column orders: Logical Order: Order of the columns as seen in Logical view of the model.

- Physical Order: Order of the columns as seen in Physical view of the model.
- Database Order: Order of the columns as seen in the Database schema.

3.6.4 Setting Processing Key Property

'Processing Key' user defined property needs to be set for the following columns within the Ledger Class table: Leaf columns that are part of the unique index

The following are the steps to set this property in ERwin:

1. Choose Physical View within the ERwin model.
2. Choose **BASIC_LEDGER_CLASS_REQ** super-class table.
3. Choose the leaf column that needs to be set 'Processing Key' property.
4. Go to UDP tab in Column Properties window for this column.
5. Specify 'Yes' against 'Processing Key' user-defined property.
6. Choose the custom Ledger Class table.
7. Go to UDP tab in Column Properties window for the columns.
8. Specify 'Yes' against 'Processing Key' user-defined property.

3.6.5 Unique Index

Ledger Class tables require unique index on the following columns:

- IDENTITY_CODE
- FISCAL_YEAR
- CONSOLIDATION_CD
- ISO_CURRENCY_CD
- LEGAL_ENTITY_ID
- BALANCE_TYPE_CD
- FISCAL_MONTH
- Key dimension columns

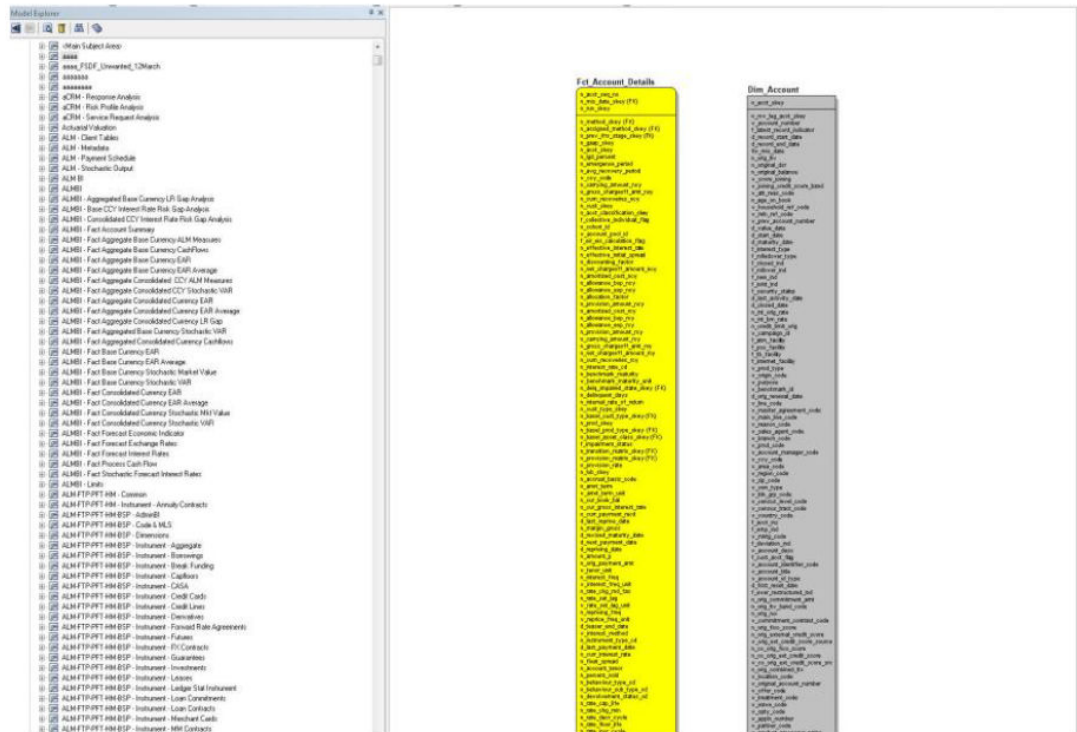
This unique index needs to be created on the custom Ledger Class table, post-model upload operation.

3.6.6 Removing the Dimensions

To remove the Dimensions, follow these steps:

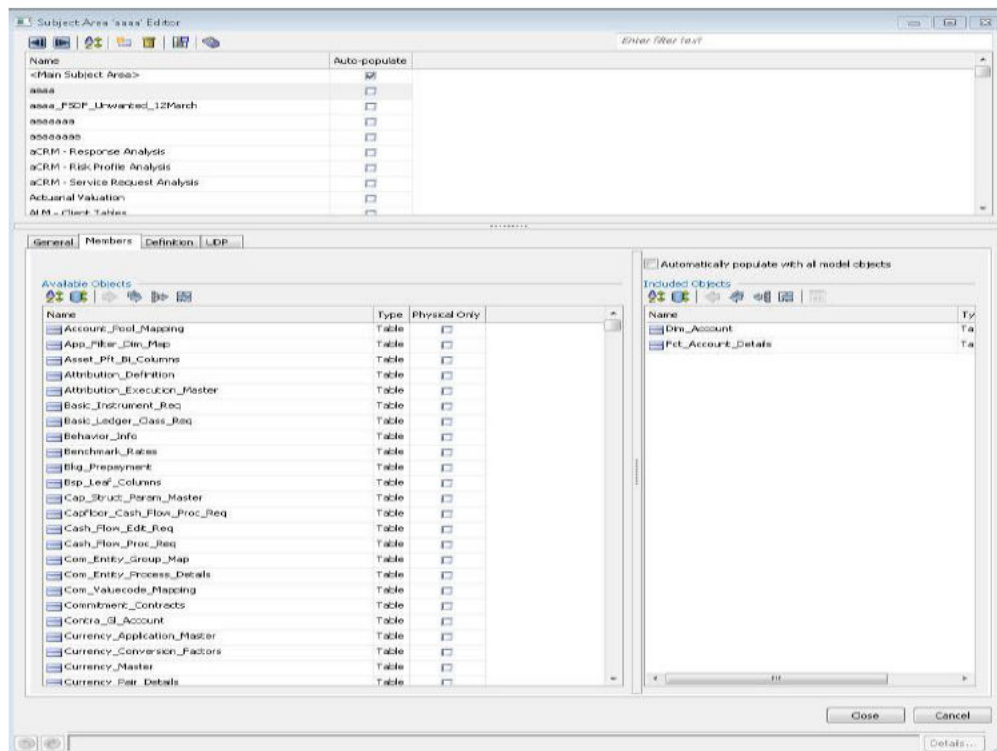
1. Select the subject area within the ERwin model.

Figure 3-4 ERwin model



2. Delete the table. Before deleting the table, check the dependent tables.

Figure 3-5 Dependent tables



3.6.7 Object Registration Validation

Since, leaf registration in-validates all Client Data Objects, the Object Registration Validation procedure needs to be executed to validate the required tables.

3.7 Object Registration and Validation

Table Classifications provide a means to designate how tables are used within the OFSAA suite of applications. Each table classification identifies a specific purpose for which an assigned table is allowed to be used.

Some Table Classifications have requirements that must be satisfied in order for an object to be assigned to the classification. These requirements are designated by Table Properties associated to the Table Classifications. These Table Properties are either specific column name requirements or logic validations.

Table Classification assignments are stored in REV_TABLE_CLASS_ASSIGNMENT.

Note

FSI_D_CUSTOMER table should be registered against Other table Class and Profitability-Other Class in REV_TABLE_CLASS_ASSIGNMENT table. Below SQL statements should be executed explicitly to make the required entries in REV_TABLE_CLASS_ASSIGNMENT :

```
Insert into REV_TABLE_CLASS_ASSIGNMENT
(TABLE_NAME,OWNER,TABLE_CLASSIFICATION_CD,PROTECTED_FLG,VALIDATED_FLAG)
values ('FSI_D_CUSTOMER','<DB OWNER>',30,0,'Y'); Insert into
REV_TABLE_CLASS_ASSIGNMENT
(TABLE_NAME,OWNER,TABLE_CLASSIFICATION_CD,PROTECTED_FLG,VALIDATED_FLAG)
values ('FSI_D_CUSTOMER','<DB OWNER>',350,0,'Y');
where
```

<DB OWNER> is atomic schema user name.

Object Registration is a process of classifying a table with one or more table classifications depending on the purpose of the table. This step is performed within the ERwin model by setting various User Defined Properties for a client data object. Validation procedure validates table class assignment for a client data object and needs to be executed after model upload operation.

Topics:

- [User-Assignable Table Classification](#)
- [Requirement For Table Classification](#)
- [Validation Procedure](#)
- [Executing the Validation Procedure](#)
- [Exception Messages](#)

3.7.1 User-Assignable Table Classification

User-Assignable Table Classifications are those that can be assigned by the administrator to user-defined and client data objects, including the OFSAI Instrument tables. These Table Classifications identify processing and reporting functions for the OFSAA. Some of these Table Classifications have requirements that must be met in order for the classification to be assigned to a table or view.

All User-Assignable Table Classifications are available for assignment within the ERwin model. The following table lists the User-Assignable Table Classifications:

Table 3-1 User-Assignable Table Classifications

Code	Table Classification Name
20	Instrument
50	Ledger Stat
100	Portfolio
200	TP Cash Flow
210	TP Non-Cash Flow
295	Codes User Defined (base tbl)
296	MLS Descriptions User Defined

Table 3-1 (Cont.) User-Assignable Table Classifications

300	Transaction Profitability
310	Instrument Profitability
320	User Defined
330	Data Correction Processing
360	RM Standard
370	TP Option Costing
500	PA Lookup Tables
600	Derivative Instruments
530	Break Funding
197	MLS Descriptions Reserved
198	Codes Reserved (base tbl)
21	Insurance Policy UDP
40	Portfolio Supertype UDP
301	Insurance Transaction Profitability UDP
311	Insurance Policy Profitability UDP
351	Insurance Profitability - Other Class UDP
702	Processing - EPM Prepayment UDP

3.7.2 Requirement For Table Classification

Specific column requirements for each table property can be obtained by querying REV_COLUMN_REQUIREMENTS table.

OFSAAI requires specific table structures, column names and column characteristics for OFSAAI operations. These structures and requirements are embodied by the User-Assignable Table Classifications.

Each Table Classification comprises individual Table Properties that define the requirements for that classification. Table Properties are two distinct types: those encompassing specific column requirements and those encompassing logic requirements via stored procedures.

The following table provides the validation checks that are being done for each of the table classification:

TABLE_CLASSIFICATION_CD	TABLE_CLASSIFICATION	TABLE_PROPERTY	DESCRIPTION	Comments
50	Ledger Stat	Ledger Leaf Column Class	Fields that are part of core modeling dimensions for Fusion PFT	Checks if columns of super-type Ledger Leaf Column Class is present
100	Portfolio	Portfolio Requirements	Dynamic list of Portfolio fields	Checks if columns of super-type Portfolio Requirements is present
200	TP Cash Flow	Basic Instrument Requirements	Instrument Required fields	Checks if columns of super-type Basic Instrument Requirements is present

200	TP Cash Flow	Cash Flow Proc. Requirements	Fields required by TP and ALM Cash Flow processing	Checks if columns of super-type Cash Flow Proc. Requirements is present Note, this is required if Conditional Assumptions are being defined against the table.
200	TP Cash Flow	Cash Flow Edit Requirements	Fields required by Cash Flow Edits in addition to Cash Flow fields	Checks if columns of super-type Cash Flow Edit Requirements is present
200	TP Cash Flow	Multi-Currency Requirements	Fields required for Multi-Currency	Checks if columns of super-type Multi-Currency Requirements is present
200	TP Cash Flow	TP Basic Requirements	Non-cash flow Transfer Pricing fields	Checks if columns of super-type TP Basic Requirements is present
200	TP Cash Flow	Validate Instrument Leaves	Validates that a table has all 'B' leaves	Validation . Check if the table has all the key dimension leaf columns. The leaf columns should be of data type NUMBER
200	TP Cash Flow	Validate Instrument Key	Validate the unique key for Instrument (PA, TP, ALM) tables	Validation . Instrument table should have index present on ID_NUMBER and IDENTITY_CODE column
210	TP Non-Cash Flow	Basic Instrument Requirements	Instrument Required fields	Checks if columns of super-type Basic Instrument Requirements is present
210	TP Non-Cash Flow	Multi-Currency Requirements	Fields required for Multi-Currency	Checks if columns of super-type Multi-Currency Requirements is present
210	TP Non-Cash Flow	TP Basic Requirements	Non-cash flow Transfer Pricing fields	Checks if columns of super-type TP Basic Requirements is present

210	TP Non-Cash Flow	Validate Instrument Leaves	Validates that a table has all 'B' leaves	Validation . Check if the table has all the key dimension leaf columns. The leaf columns should be of data type NUMBER
210	TP Non-Cash Flow	Validate Instrument Key	Validate the unique key for Instrument (PA, TP, ALM) tables	Validation . Instrument table should have index present on ID_NUMBER and IDENTITY_CODE column
300	Transaction Profitability	Basic Instrument Requirements	Instrument Required fields	Checks if columns of super-type Basic Instrument Requirements is present
300	Transaction Profitability	Multi-Currency Requirements	Fields required for Multi-Currency	Checks if columns of super-type Multi-Currency Requirements is present
300	Transaction Profitability	Validate Instrument Leaves	Validates that a table has all 'B' leaves	Validation . Check if the table has all the key dimension leaf columns. The leaf columns should be of data type NUMBER
300	Transaction Profitability	Validate Transaction Key	Validate the unique key for Transaction Profitability tables	Transaction table should have composite index present on ID_NUMBER and IDENTITY_CODE and all the processing key columns.
310	Instrument Profitability	Basic Instrument Requirements	Instrument Required fields	Checks if columns of super-type Basic Instrument Requirements is present
310	Instrument Profitability	Multi-Currency Requirements	Fields required for Multi-Currency	Checks if columns of super-type Multi-Currency Requirements is present
310	Instrument Profitability	Validate Instrument Leaves	Validates that a table has all 'B' leaves	Validation . Check if the table has all the key dimension leaf columns. The leaf columns should be of data type NUMBER

310	Instrument Profitability	Validate Instrument Key	Validate the unique key for Instrument (PA, TP, ALM) tables	Validation . Instrument table should have index present on ID_NUMBER and IDENTITY_CODE column
330	Data Correction Processing	Validate Processing Key	Validate the unique key for Processing tables	Processing Key Column for a table have a matching unique index
360	ALM Standard	Basic Instrument Requirements	Instrument Required fields	Checks if columns of super-type Basic Instrument Requirements is present
360	ALM Standard	Cash Flow Proc. Requirements	Fields required by TP and ALM Cash Flow processing	Checks if columns of super-type Cash Flow Proc. Requirements is present
360	ALM Standard	Cash Flow Edit Requirements	Fields required by Cash Flow Edits in addition to Cash Flow fields	Checks if columns of super-type Cash Flow Edit Requirements is present
360	ALM Standard	Multi-Currency Requirements	Fields required for Multi-Currency	Checks if columns of super-type Multi-Currency Requirements is present
360	ALM Standard	Validate Instrument Leaves	Validates that a table has all 'B' leaves	Validation . Check if the table has all the key dimension leaf columns. The leaf columns should be of data type NUMBER
360	ALM Standard	Validate Instrument Key	Validate the unique key for Instrument (PA, TP, ALM) tables	Validation . Instrument table should have index present on ID_NUMBER and IDENTITY_CODE column
370	TP Option Costing	Basic Instrument Requirements	Instrument Required fields	Checks if columns of super-type Basic Instrument Requirements is present
370	TP Option Costing	Cash Flow Edit Requirements	Fields required by Cash Flow Edits in addition to Cash Flow fields	Checks if columns of super-type Cash Flow Edit Requirements is present

370	TP Option Costing	Multi-Currency Requirements	Fields required for Multi-Currency	Checks if columns of super-type Multi-Currency Requirements is present
370	TP Option Costing	TP Option Costing Requirements	Fields required for Transfer Pricing Option Costing processing	Checks if columns of super-type TP Option Costing Requirements is present
370	TP Option Costing	TP Basic Requirements	Non-cash flow Transfer Pricing fields	Checks if columns of super-type TP Basic Requirements is present
370	TP Option Costing	Validate Instrument Leaves	Validates that a table has all 'B' leaves	Validation . Check if the table has all the key dimension leaf columns. The leaf columns should be of data type NUMBER
370	TP Option Costing	Validate Instrument Key	Validate the unique key for Instrument (PA, TP, ALM) tables	Validation. Instrument table should have index present on ID_NUMBER and IDENTITY_CODE column
500	PA Lookup Tables	Validate PA Lookup	Procedure to check if there is a primary key for the lookup tables.	Validation. All Lookup table should have a primary key present
530	Break Funding	Break Funding Requirements	Fields required as part of TP break funding	Checks if columns of super-type Break Funding Requirements is present
301	Insurance Transaction Profitability UDP	Validate Policy Table Leaves	Validates that a policy table has all 'B' leaves	Validation. Check if the table has all the key dimension leaf columns. The leaf columns should be of data type NUMBER
301	Insurance Transaction Profitability UDP	Validate Policy Trans Key	Validate the unique key for Policy Transaction tables	Transaction table should have composite index present on ID_NUMBER and IDENTITY_CODE and all the processing key columns.

301	Insurance Transaction Profitability UDP	Basic Instrument Requirements	Instrument Required fields	Checks if columns of supertype Basic Instrument Requirements are present
301	Insurance Transaction Profitability UDP	Multi-Currency Requirements	Fields required for Multi-Currency	Checks if columns of supertype Multi-Currency Requirements are present
311	Insurance Policy Profitability UDP	Multi-Currency Requirements	Fields required for Multi-Currency	Checks if columns of supertype Multi-Currency Requirements are present
311	Insurance Policy Profitability UDP	Basic Instrument Requirements	Instrument Required fields	Checks if columns of supertype Basic Instrument Requirements are present
311	Insurance Policy Profitability UDP	Validate Instrument Key	Validate the unique key for Instrument (PA, TP, RM) tables	Transaction table should have composite index present on ID_NUMBER and IDENTITY_CODE and all the processing key columns.
311	Insurance Policy Profitability UDP	Validate Policy Table Leaves	Validates that a policy table has all 'B' leaves	Validation . Check if the table has all the key dimension leaf columns. The leaf columns must be of data type NUMBER
351	Insurance Profitability - Other Class UDP	Profitability - Other Class	Validate the unique key for Profitability - Other Class tables	Validate the unique key for Profitability - Other Class tables

3.7.3 Validation Procedure

The `OFSA_TAB_CLASS_REQ` package contains all of the procedures and supporting functions that validates if a table meets the requirements for a particular Table Classification.

The package performs the following validations:

- VALIDATE_INST_KEY**
 This procedure validates if a table has `ID_NUMBER` and `IDENTITY_CODE`, or `ID_NUMBER`, `IDENTITY_CODE` and `AS_OF_DATE` as its unique index and if the Processing key designated in Column Properties is `ID_NUMBER`, `IDENTITY_CODE`.
- UPDATABLE_INST_REQ_FIELDS**
 This procedure checks that all of the Instrument Required Fields are also listed as updatable in `USER_UPDATABLE_COLUMNS` for the specified table or view.
- VALIDATE_INST_LEAVES**
 This procedure will validate a table has all the required leaf columns

- **VALIDATE_TRANS_KEY**
This procedure validates if a table has `ID_NUMBER` and `IDENTITY_CODE` and one or more 'B' Leaf Columns in its unique index and that these columns match the Processing key designated in Column Properties.
- **VALIDATE_CORR_KEY**
This procedure will validate a table has a unique index with updatable columns.
All the above procedures return a success or failure status. The `REV_TAB_CLASS_ASSIGNMENT` table is updated as 'Y' if a table is successfully validated and 'N' in case of failure.

3.7.4 Executing the Validation Procedure

You can execute this procedure either from SQL*Plus or from within a PL/SQL block or from Batch Maintenance window within OFSAAI framework.

1. To run the procedure from SQL*Plus, login to SQL*Plus as the Atomic Schema Owner. The syntax for calling the procedure is:

```
set serveroutput off Declare Output number; Begin Output :=
fsi_batchtableclassreq(pbatchid, pms_date); End;
```

Note

Since the package contains huge number of `dbms_output` statements, user should either increase the output buffer size or disable the server output.

For Example:

```
set serveroutput off Declare Output number; Begin Output :=
fsi_batchtableclassreq('INFODOM_INSTRUMENT_TABLE_VALIDATION_20131205_1',
'20131205'); End;
```

2. To execute the procedure from OFSAAI Batch Maintenance, create a new Batch with the Task as TRANSFORM DATA and specify the following parameters for the task:

Datastore Type: Select appropriate datastore from list

Datastore Name: Select appropriate name from the list

IP address: Select the IP address from the list

Rule Name: Batch_Table_Class_Req

Parameter List: Batch Identifier and MISDATE

3.7.5 Exception Messages

The `OFSA_TAB_CLASS_REQ` packages throws the following exceptions.

- **Exception 1:** FAILED: Table Property 1030 - Validate Correction Key
This exception occurs when no valid unique index found.
- **Exception 2:** FAILED: Table Property 1030 - Validate Correction Key
This exception occurs when Processing Key Column Properties do not match unique index
- **Exception 3:** FAILED: Table Property 1030 - Validate Transaction Key
This exception occurs when no valid unique index found.
- **Exception 4:** FAILED: Table Property 1000 - Validate Instrument Leaves

This exception occurs when one or more Leaf Columns are missing or incorrectly registered. Check if the datatype of the LEAF columns is NUMBER and domain of these columns is LEAF.

3.8 Alternate Rate Output Columns

This section details the steps required for defining Alternate Rate Output columns within the OFSAA Fund Transfer Pricing Application.

The following topics are covered in this section:

- [Setting User Defined Properties in ERwin](#)
- [Uploading the model and object registration](#)

3.8.1 User-Defined Properties

The following are the user-defined properties that are available for identifying columns required for alternate rate output:

- Transfer Pricing Output (Column Property – 80)
- Option Cost Output (Column Property – 81)
- Other Adj Spread Output (Column Property – 82)
- Other Adj Amount Output (Column Property – 83)
- Economic Value Output (Column Property – 86)
- Liquidity_rate_column (Column Property – 95)
- Liquidity_amount_column (Column Property – 96)
- Basis_rate_column (Column Property – 97)
- Basis_amount_column (Column Property – 98)
- Pricing_rate_column (Column Property – 99)
- Pricing_amount_column (Column Property – 100)

User needs to assign one of the above properties to the columns that need to be used as Alternate Rate Output columns within the Fund Transfer Pricing application.

The following are the steps to set the user-defined property to the column:

1. Open the ERwin file in ERwin Data Modeler tool.
2. Go to Main Subject Area.
3. Go to Physical View.
4. Choose the entity that contains the alternate rate output column. This entity can also be a super-type (like TP_BASIC_REQ).
5. Select the column and open the column properties for the column.
6. Go to UDP tab within column properties.
7. Select YES for one of the above user-defined properties.
8. Save the model.

Note

Setting the user-defined property of the columns within a super-type entity will apply to all the entities that are related to the super-type.

3.8.2 Uploading the Model

To upload the Model, follow these steps:

1. Upload the model in OFSAAI and perform object registration.
2. After uploading the model, you can execute the following query to check if the user-defined properties are set for the columns.

```
select * from rev_column_properties where column_property_cd in (80,81,82,83)
where TABLE_NAME = <<table_name>>Replace <<table_name>> with the relevant table
name and column name in the above query
```

3. Execute the same. Above query returns the columns that are used for alternate rate outputs.

3.9 User Defined Properties

User Defined Properties are set for tables and columns within ERwin.

Table Level User Defined Properties

The following user defined properties can be set for the table:

UDP Name	Description	List of values
Instrument	Property to identify if the table is classified as a basic instrument table. (that is, Instrument table classification code 20)	YES / NO
TP Cash Flow	Property to identify if the table is classified as 'TP Cash Flow' for the purpose of generating Transfer Pricing rates using cash flow methods.	YES / NO
TP Non Cash Flow	Property to identify if the table is classified as 'TP Non-Cash Flow' for the purpose of generating Transfer Pricing rates using non cash flow methods.	YES / NO
Transaction Profitability	Property to identify if the table is classified as 'Transaction' for the purpose of executing allocation rules.	YES / NO
Portfolio	Property to identify if the table is classified as 'Portfolio'.	YES / NO
User Defined	Property to identify if the table is classified as 'User Defined' table for storing multi-lingual descriptions for codes.	YES / NO

Ledger Stat	Property to identify if the table is classified as 'Ledger Stat' for the purpose of executing allocation rules.	YES / NO
ALM Standard	Property to identify if the table is classified as 'ALM Standard' for the purpose of executing ALM cash flow engine to generate cash flows.	YES / NO
TP Option Costing	Property to identify if the table is classified as 'TP Option Costing' for the purpose of generating Transfer Pricing rates with option costing.	YES / NO
Break Funding	Property to identify if the table is classified as 'Break Funding' for the purpose of generating Break funding charges using Transfer Pricing engine.	YES / NO
MLS Descriptions Reserved	Property to identify if the table is classified as 'Reserved' table for storing multi-lingual descriptions for codes.	YES / NO
Codes Reserved (base tbl)	Property to identify if the table is classified as 'Reserved' table for storing codes of simple dimensions.	YES / NO
Codes User Defined (base tbl)	Property to identify if the table is classified as 'User-defined' table for storing codes of simple dimensions.	YES / NO
PA Lookup Tables	Property to identify if the table is classified as 'Lookup Table' for the purpose of defining lookup table allocation rules.	YES / NO
Instrument Profitability	Property to identify if the table is classified as 'Instrument' for the purpose of executing allocation rules.	YES / NO
Derivative Instruments	Property to identify if the table is classified as 'Derivatives' for the purpose of executing ALM cash flow engine to generate cash flows for derivative instruments.	YES / NO
Data Correction Processing	Property to identify if the table is classified as 'Data Correction Processing' for the purpose of executing Cash Flow Edits engine.	YES / NO

Column Level User Defined Properties

The following user defined properties can be set for the column:

UDP Name	Description	List of values
----------	-------------	----------------

Balance Range	Property to identify if the column within a table classified as 'PA Lookup Table' must be displayed under 'Range' within Lookup table definition.	YES / NO
Balance	Property to identify if the column is of type 'Balance'.	YES / NO
Standard Rate	Property to identify if the column is of type 'Standard Rate'.	YES / NO
Balance Weighted Object	Property to identify if the column is of type 'Balance Weighted Object'.	YES / NO
Processing Key	Property to identify if this column is used as a 'Processing Key' within the instrument, transaction and ledger_stat table.	YES / NO
Frequency Multiplier	Property to identify if the column is used to store 'Frequency'. This property is used in Filters UI within OFSAAL.	YES / NO
Multiplier Related Field	Property to specify the name of the column that is used to store the multiplier for the corresponding 'Frequency' column. This property is used in Filters UI within OFSAAL.	Text
Related Field	Property to specify the name of the column that is used to store the multiplier for the corresponding 'Term' column. This property is used in Filters UI within OFSAAL.	Text
Term Multiplier	Property to identify if the column is used to store 'Term'. This property is used in Filters UI within OFSAAL.	YES / NO
Column Alias	Property to specify an alias for the column. This is used within the staging loader program for loading LEDGER_STAT table.	Text
Statistic	Property to identify if the column is of type 'Statistic'.	YES / NO
Transfer Pricing Output	Property to identify if the column must be set as an alternate output column for writing transfer rates by transfer pricing engine.	YES / NO
Option Cost Output	Property to identify if the column must be set as an alternate output column for writing option costing output by transfer pricing engine.	YES / NO
Other Adj Spread Output	Property to identify if the column must be set as an alternate output column for writing other adjustment spread by transfer pricing engine.	YES / NO

Other Adj Amount Output	Property to identify if the column must be set as an alternate output column for writing other adjustment amount by transfer pricing engine.	YES / NO
UDP_LOOKUP_RANGE_MINIMUM	Property to identify minimum range column within a table classified as 'PA Lookup Table'. For eg: LOOKUP_TEMPLATE_TABLE.AVERAGE_BALANCE_MIN column.	YES / NO
UDP_LOOKUP_RANGE_MAXIMUM	Property to identify maximum range column within a table classified as 'PA Lookup Table'. For eg: LOOKUP_TEMPLATE_TABLE.AVERAGE_BALANCE_MAX column.	YES / NO
UDP_EXPORT_PFT_OUTPUT	Property to identify PFT Output Columns in instrument tables and FSI_D_INST_SUMMARY table. For eg: CALL_CENTER_EXP, COMPLAINCE_EXP etc.	YES / NO
UDP_EXPORT_FTP_OUTPUT	Property to identify FTP Output Columns in instrument tables and FSI_D_INST_SUMMARY table For eg: OTHER_ADJUSTMENTS_AMT, OTHER_ADJUSTMENTS_RATE etc.	YES / NO
UDP_EXPORT_FTP_OTHERS	Property to identify columns used by FTP in instrument tables and FSI_D_INST_SUMMARY table For eg: AVG_BOOK_BAL, CUR_BOOK_BAL etc.	YES / NO
UDP_EXPORT_EPM_KEY_DIMS	Property to identify EPM Key Processing dimension columns in instrument tables and FSI_D_INST_SUMMARY table. For eg: COMMON_COA_ID, PRODUCT_ID etc.	YES / NO
UDP_EXPORT_EPM_KEY_COLUMNS	Property to identify mandatory key columns for EPM in instrument tables and FSI_D_INST_SUMMARY table. For eg: AS_OF_DATE, ISO_CURRENCY_CD, IDENTITY_CODE, INSTRUMENT_TYPE_CD etc.	YES / NO
UDP_EXPORT_EPM_OTHERS	Property to identify other columns for EPM in instrument tables and FSI_D_INST_SUMMARY table. For eg: ACCOUNT_OFFICER_CD, INTEREST_INC_EXP etc.	YES / NO

3.10 Modifying the Precision of Balance Columns In Ledger Stat

Steps to modify the Precision

1. Open the ALM, FTP, or PFT model using All Fusion ERwin Data Modeler.
2. Switch to ALM-FTP-PFT-HM-BSP – Ledger Stat subject area.
3. Select Logical view.
4. Edit the Ledger Stat table by double clicking the table in the Logical Layer.
5. Change the data type in Datatype tab to the revised precision and scale (example, NUMBER (22, 3)) for the following columns: Month 01 Amount, Month 02 Amount, Month 03 Amount and so on YTD 01 Amount, YTD 02 Amount, YTD 03 Amount and so on.
6. Save the changes. Select the Physical view.
7. Click LEDGER_STAT table and view the datatype of columns – MONTH_01 till MONTH_12 and YTD_01 till YTD_12. The data type of these columns should display the new precision and scale.
8. Save the model as xml in All Fusion Repository Format.
9. Perform incremental model upload. NOTE In case, users decrease the precision and scale for the columns, such columns should not have any values during model upload.

4

Utilities

This chapter details the steps involved in executing various data model utilities that are available within OFSAA.

Topics:

- [Reverse Population](#)
- [Product Instrument Mapping](#)
- [Instrument Synchronization](#)
- [Stage Synchronization](#)
- [Ledger Load Undo](#)
- [Data Slicing](#)

4.1 Reverse Population

Reverse population procedure populates dimension members, attributes and hierarchies from new dimension tables to OFSA legacy set of dimension tables. ALM, TP and PFT engines refer to OFSA legacy tables for retrieving dimension member information.

Topics:

- [Tables As Part Of Reverse Population](#)
- [Reverse Population Procedure](#)
- [Executing the Reverse Population Function](#)
- [Exception Messages](#)

Note

From PFT 8.0.4.0.14 release, the following entry will be seeded into SETUP_PARAMETERS_MATER table:

```
PARAM_SEQ PARAM_APP_ID PARAM_NAME PARAM_VALUE <SEQ NO> ALL
FAIL_ON_HIERARCHY_MAX_DEPTH_ALL NO
```

Allowed values are YES or NO.

Default value for PARAM_VALUE is NO.

Reverse population hierarchy loader looks for a record in SETUP_PARAMETERS_MASTER table with PARAM_NAME = FAIL_ON_HIERARCHY_MAX_DEPTH_ALL

If FAIL_ON_HIERARCHY_MAX_DEPTH_ALL parameter value is found in the SETUP_PARAMETERS_MASTER table and PARAM_VALUE is set to NO, then continue with current behavior.

If FAIL_ON_HIERARCHY_MAX_DEPTH_ALL parameter value is found in the SETUP_PARAMETERS_MASTER table and PARAM_VALUE is set to YES, abort the loader and batch execution fails.

If FAIL_ON_HIERARCHY_MAX_DEPTH_ALL parameter is not found in SETUP_PARAMETERS_MASTER table then continue with existing behavior.

While saving a hierarchy from AMHM screen with more than 15 levels, even if the parameter FAIL_ON_HIERARCHY_MAX_DEPTH_ALL is set to YES, hierarchy gets saved and reverse population does not perform (which is the existing behavior).

On running the reverse population batch (DIMENSION_HIERARCHY_LOAD) for a hierarchy with more than 15 levels, the batch and task execution fails when the parameter FAIL_ON_HIERARCHY_MAX_DEPTH_ALL is set to YES. Also, an error is logged in FSI_MESSAGE_LOG table.

If the parameter is set to NO (or if the parameter is not available at all), then the reverse population batch will be successful, and only a warning message will be logged in FSI_MESSAGE_LOG which indicates the hierarchy has more than 15 levels.

4.1.1 Tables As Part Of Reverse Population

Dimension data is stored in the following set of tables:

- DIM_<DIMENSION>_B: Stores leaf and node member codes within the dimension.
- DIM_<DIMENSION>_TL: Stores names of leaf and node and their translations.
- DIM_<DIMENSION>_ATTR: Stores attribute values for the attributes of the dimension.
- DIM_<DIMENSION>_HIER: Stores parent-child relationship of members and nodes that are part of hierarchies.

Data present in the above set of dimension tables are transformed into the following set of OFSA Legacy tables.

The reverse population routine synchronizes the dimension data between the new dimension tables and the OFSA Legacy tables. Reverse population occurs automatically if enabled in the

AMHMConfig.properties file. In the AMHMConfig.properties file, set the Parameter value to Y for a specific Dimension Id. The setting in the AMHMConfig.properties only impacts dimension values entered through the interface. Reverse population must be executed as a batch for bulk loading.

Reverse population will automatically occur with object migration for key dimension members on the Target instance if AMHMConfig.properties has a property with Key=HIERARCHY_REVERSE_POP-<Infodom in Upper Case>-<Dimension ID> Value=Y.

For more information on how to define the reverse populate parameters in the AMHMConfig.properties file, see [Oracle Financial Services Analytical Applications Infrastructure \(OFSAI\) User Guide](#).

- OFSA_LEAF_DESC: Stores the description of leaf members that are part of the dimension.
- OFSA_NODE_DESC: Stores the description of nodes that are used within the hierarchy.
- OFSA_DETAIL_LEAVES: Stores the attributes of Common COA dimension.
- OFSA_DETAIL_OTHER_COA: Stores the attributes of GL or Product or any other key dimension.
- OFSA_DETAIL_ELEM_B/OFSADetailElem_MLS: Stores the attributes of Financial Elements dimension.
- OFSA_IDT_ROLLUP: Stores the hierarchy as level-based.
- OFSA_LEVEL_DESC: Stores the hierarchy levels.

Reverse population is done for all key dimensions that are configured within the OFSAI framework.

4.1.2 Reverse Population Procedure

The REVERSE_POPULATION package populates the OFSA legacy dimension tables from new dimension tables.

The procedure performs the following functions:

- Gets the list of source and target tables. The source tables for given dimension is stored in REV_DIMENSION_B table. The OFSA target table for a given dimension is stored in OFSA_CATALOG_OF_LEAVES.
- The REVERSE POPULATION transposes the seeded attributes, leaf members and hierarchy data stored in the form of rows (new dimension table structure) to columns (OFSA).
- All exception messages are logged in the FSI_MESSAGE_LOG table.

4.1.3 Executing the Reverse Population Function

You can execute this function from either within a PL/SQL block or from Batch Maintenance window within OFSAI framework.

1. Members Reverse Population Function

To run the function with a PL/SQL block, execute following:

```
fsi_batchMemberLoad(batch_run_id varchar2, mis_date varchar2, pDimensionId  
varchar2, pMemberId varchar2, pMode varchar2)
```

where BATCH_RUN_ID is any string to identify the executed batch.

MIS_DATE in the format YYYYMMDD.

pDIMENSIONID is the dimension id.

pMEMBERID can be null. If value is provided, only that member id gets reverse populated. If pMode value is 1, it means fresh insert, if value is 2 means update, and if value is 3 means delete. In batch mode, you can prefer to use 2.

For Example:

```
Declare num number;Begin num := fsi_batchmemberload
('INFODOM_20100405', '20100405', 1, null, 2);End;
```

2. To execute the procedure from OFSAAI Batch Maintenance, create a new Batch with the Task as TRANSFORM DATA and specify the following parameters for the task:

Datastore Type: Select appropriate datastore from list

Datastore Name: Select appropriate name from the list

IP address: Select the IP address from the list

Rule Name: Batch_Member_Load

Parameter List: Dimension ID, Member id, pMode

3. Hierarchy Reverse Population

```
Function fsi_batchhierarchyload(batch_run_id varchar2, mis_date varchar2,
pDimensionId varchar2, pHierarchyId varchar2, pMode varchar2)
```

where BATCH_RUN_ID is any string to identify the executed batch.

MIS_DATE in the format YYYYMMDD. pDIMENSIONID is the dimension id. pHIERARCHYID can be null.

If value is provided, only that Hierarchy gets reverse populated. If pMode value is 1, it means fresh insert, if value is 2 means update, and if value is 3 means delete. In batch mode, you can prefer to use 2.

For Example:

```
Declare num number;Begin num :=
fsi_batchhierarchyload('INFODOM_20100405', '20100405', 1, null, 2);End;
```

4. To execute the procedure from OFSAAI Batch Maintenance, create a new Batch with the Task as TRANSFORM DATA and specify the following parameters for the task:

Datastore Type: Select appropriate datastore from list

Datastore Name: Select appropriate name from the list

IP address: Select the IP address from the list

Rule Name: Batch_Hier_Load

Parameter List: Dimension ID, Hierarchy id, pMode

Note

The reverse population fsi_batchMemberLoad and fsi_batchHierarchyLoad should be executed after fn_drmdataloader. The fsi_batchMemberLoad reverse populates the members and the fsi_batchHierarchyLoad reverse populates the hierarchies to the legacy structures.fn_drm

Note

DataLoader supports 15 + level and any hierarchy with greater than 15 level cannot be used in ALM/PFT/FTP process. fsi_batchHierarchyLoad does not support 15 + level, however, this method will work in case of 15+ level hierarchies; it will only skip such hierarchies, with a suitable message in the log table(s). The Hierarchy greater than 15 level are not supported within ALM/FTP/PFT/HM processes, OFSA_IDT_ROLLUP will not be populated, however nothing prevents the EPM application UIs from rendering 15+ level hierarchies .rev_batchHierFlatten supports a maximum of 20 levels including leaf.

4.1.4 Exception Messages

The Reverse Population procedure may cause some exceptions to appear. The text and explanation for each of these exceptions follows. If you call the procedure from a PL/SQL block you may want to handle them so that your program can proceed.

- **Exception 1:** Error. While getting dimension details
This exception occurs when the reverse population procedure cannot find any data configured in the driver table (REV_DIMENSIONS_B).
- **Exception 2:** Error. While generating hierarchy Query
This exception occurs when there is a problem generating hierarchy query dynamically.
- **Exception 3:** Error. While populating Nodes
This exception occurs when there is an error populating the OFSA_NODE_DESC table

4.2 Product Instrument Mapping

ALM and TP processes can be based on a set of data tables or a set of products. In case products are selected, ALM and TP engine internally gets the list of data tables mapped to these products and processes those data tables. During the period-ending load cycle, data is loaded into your Data Objects such as Instrument tables. During this load process, all the distinct members of 'Product' type dimension that are present within each data table will be stored in a separate table (FSI_M_PROD_INST_TABLE_MAP) by executing Product Instrument mapping procedure.

Topics:

- [Tables Requiring Synchronization](#)
- [Product Instrument Table Map Procedure](#)
- [Executing the PRODUCT_INSTRUMENT_TABLE_MAP Procedure](#)
- [Exception Messages](#)

4.2.1 Tables Requiring Synchronization

Product-instrument table mapping is required only for Instrument tables. Instrument tables are defined as all tables with the Instrument Table Classification (table_classification_cd in (20,600,200,210)) on which all of the defined Leaf Columns exist.

4.2.2 Product Instrument Table Map Procedure

This function gives exact mapping of a particular 'Product' stored in multiple Instrument table, and mapping is stored in FSI_M_PROD_INST_TABLE_MAP for given AS_OF_DATE. The function outputs the mapping information only if the corresponding 'Product' definition exists in the corresponding dimension table. The procedure performs the following functions:

1. Gets the list of 'Product' type dimensions from dimension registry table (REV_DIMENSIONS_B).
2. Gets the list of Instrument tables from REV_TABLE_CLASS_ASSIGNMENT.
3. Fetches the distinct set of members for each 'Product' type dimension from all instrument tables for a given AS_OF_DATE.
4. Stores the above set into a mapping table (FSI_M_PROD_INST_TABLE_MAP).
5. The function outputs message in the message log if the member definition which exists in the Instrument table is not found in the respective dimension table.
6. After the Product-Instrument table mapping utility run is completed, you should query the mapping table to look for dimension members that are present as part of each instrument table.

4.2.3 Executing the PRODUCT_INSTRUMENT_TABLE_MAP

You can execute this procedure either from SQL*Plus or from within a PL/SQL block or from Batch Maintenance window within OFSAAI framework.

1. To run the procedure from SQL*Plus, login to SQL*Plus as the Schema Owner.
2. The procedure requires 3 parameters: Batch Id (which can be used to see the log of the procedure executed), MISDATE and the AS_OF_DATE.
3. Identify the table name parameter by enclosing it in single quotes and uppercase, as shown in the following two examples.

The syntax for calling the procedure is: `Declare output number; Begin Output:= fn_Product_Instrument_Map ('Batch_Id', 'MISDATE', 'AS_OF_DATE'); End;`

AS_OF_DATE is the date for which mapping is required.

MISDATE is the date for which batch is run.

Both MISDATE and AS_OF_DATE should be passed as 'YYYYMMDD' format.

An example of running the function from SQL*Plus for the FSI_D_TERM_DEPOSITS table follows:

```
SQL> var output number; SQL> execute :output:= fn_Product_Instrument_Map
('Batch_Id', '20100131', '19991231');
```

To execute the stored procedure from within a PL/SQL block or procedure, see the example that follows.

Call the procedure as often as required to synchronize all of your instrument tables.

The appropriate table parameters are enclosed in single quotes.

```
SQL> declare output number; begin output:= fn_Product_Instrument_Map
('Batch_Id', 'MISDATE', 'AS_OF_DATE') end; /
```

- To execute the procedure from OFSAAI Batch Maintenance, create a new Batch with the Task as TRANSFORM DATA and specify the following parameters for the task:

Datastore Type : Select appropriate datastore from list

Datastore Name : Select appropriate name from the list

IP address : Select the IP address from the list

Rule Name : Product_Inst_Mapping

Parameter List : AS_OF_DATENOTE

Note

BATCHID and MISDATE will be passed explicitly in Batch Maintenance.

4.2.4 Exception Messages

The Product to Instrument Mapping function may cause two exceptions to appear. The text and explanation for each of these exceptions follows. If you call the function from a PL/SQL block you may want to handle them so that your program can proceed.

- Exception 1:** Table does not exist
The exception message reads:
Table 'TABLE_NAME' does not exist.
This exception occurs when the function does not find the Instrument table.
- Exception 2:** Column does not exist
The exception message reads:
Column 'Column_Name' does not exist in the instrument table 'Table_Name' while processing dimension 'Dimension ID'.
This error occurs when leaf column does not exist in the Instrument table.

4.3 Instrument Synchronization

During the period-ending load cycle, data is loaded into Client Data Objects such as Instrument tables and the LEDGER_STAT table. During this load process, it is possible for new, unidentified Dimension and Code values to be loaded into these tables.

The Instrument Synchronization procedure identifies these new Dimension and Code values and inserts default description entries for them into the appropriate tables. The procedure performs both of these synchronizations simultaneously. OFSAAI requires that all Dimension and Code values have a corresponding description. This is required for any OFSAA reporting operation to return the correct results. It also ensures that Hierarchies work properly within the OFS analytical applications.

Topics:

- [Tables Requiring Synchronization](#)
- [Dimension Member Synchronization](#)
- [Code Synchronization](#)
- [Executing the Synchronize Stage Procedure](#)
- [Exception Messages](#)

4.3.1 Tables Requiring Synchronization

Dimension member and Code value synchronization is required only for Instrument and LEDGER_STAT tables. Instrument tables are defined as all tables with the Instrument Table Classification (table_classification_cd = 20) on which all of the defined Key Dimension Columns exist.

4.3.2 Dimension Member Synchronization

The SYNCHRONIZE_INSTRUMENT procedure synchronizes the dimension member tables and the hierarchy tables with LEDGER_STAT and instrument tables, using default values for member descriptions and other information columns. You can then add the correct data to the new dimension members in AMHM member maintenance. The procedure performs the following functions:

1. Checks the specified table (LEDGER_STAT or instrument) for new dimension members in each of that table's key dimension columns and adds the new dimension value as leaf members to the respective dimension member tables.
2. Adds the new dimension member to the corresponding attribute tables with default values for mandatory attributes.
3. When new dimension members are added to the dimension tables these members include 'No Description' in the DESCRIPTION column and contain default values for mandatory attributes.
4. Reverse populates the newly added dimension members into legacy OFSA tables. During reverse population, new members are created as orphan members, under corresponding hierarchies.
5. After the SYNCHRONIZE_INSTRUMENT utility run is completed you should look for any new dimension members using the AMHM member maintenance UI and enter the correct descriptions and other member information. You should also look at the orphan node of each Hierarchy for new dimension members and move these members to the appropriate branch in the rollup.

4.3.3 Codes Synchronization

The SYNCHRONIZE_INSTRUMENT procedure identifies code values in Instrument and LEDGER_STAT tables for which a corresponding description does not exist and inserts a default description into the appropriate Code Description object. This applies only to CODE columns categorized as User-Editable or User-Defined (see the table classification). CODE columns for which OFSAA reserves all of the values are not updated by this procedure. The procedure displays a warning message for any unidentified values in CODE columns where OFSAA reserves the entire range.

For each CODE column (REV_DATA_TYPE_CD equals 3) on the specified object, the SYNCHRONIZE_INSTRUMENT procedure queries from REV_DESCRIPTION_TABLES to identify the object storing the corresponding descriptions. If the resulting object is a User-Editable or User-Defined Code Description object (checks from REV_TABLE_CLASS_ASSIGNMENT table), then the procedure inserts a default description for any code values for which a description record does not already exist. If the resulting object is an OFSAA Reserved Code Description object, then the procedure outputs a warning message indicating how many invalid code values exist in the specified Instrument or LEDGER_STAT table in the message log (FSI_MESSAGE_LOG).

For example, if you are synchronizing the FSI_D_TERM_DEPOSITS table, the procedure queries all of the CODE columns on this table. An example of a Reserved CODE column is

ACCRUAL_BASIS_CD. If the procedure finds any code values in this column that are not present in the corresponding Code Description object (FSI_ACCRUAL_BASIS_CD), it outputs an error message indicating the number of invalid values present. OFSAA Reserved Code Description objects are identified by the following SQL statement:

```
select table_name from rev_table_class_assignment
where table_classification_cd = 197;
```

An example of a User-Editable CODE column is SIC_CD. If the procedure finds any code values in SIC_CD in the FSI_D_TERM_DEPOSITS table that do not have a description in FSI_SIC_MLS, it creates a default description 'No Description' for each value. It is then up to the users to update these descriptions as appropriate. User-Editable Code Description objects are identified by the following SQL statement:

```
select * from rev_description_tables
where table_name = 'FSI_D_TERM_DEPOSITS'
and description_table_name not in
(select table_name from rev_table_class_assignment
where table_classification_cd = 197)
```

4.3.4 Executing the SYNCHRONIZE_INSTRUMENT

You can execute this procedure either from SQL*Plus or from within a PL/SQL block or from Batch Maintenance window within OFSAAI framework.

1. To run the procedure from SQL*Plus, login to SQL*Plus as the Schema Owner.
2. The procedure requires 2 parameters - table name to be synchronized and the As of Date.
3. Identify the table name parameter by enclosing it in single quotes and uppercase, as shown in the following two examples. The syntax for calling the procedure is:

```
Declare output number;Begin synchronize_instrument('Batch_Id', 'TABLE_NAME',
output)End;
```

where

table_name is either:

The name of an Instrument table

LEDGER_STAT

An example of running the stored procedure from SQL*Plus for the FSI_D_TERM_DEPOSITS table follows:

```
SQL> var output number;SQL>
synchronize_instrument('INFODOM_20101231', 'FSI_D_TERM_DEPOSITS', :output);
```

4. To execute the stored procedure from within a PL/SQL block or procedure, see the example that follows. Call the procedure as often as required to synchronize all of your instrument tables. The appropriate table name and AS_OF_DATE is enclosed in single quotes.

```
SQL> declare output number; begin
synchronize_instrument('INFODOM_20101231', 'LEDGER_STAT', output); end; /
```

5. To execute the procedure from OFSAAI Batch Maintenance, create a new Batch with the Task as TRANSFORM DATA and specify the following parameters for the task:

Datastore Type: Select appropriate datastore from list

Datastore Name: Select appropriate name from the list

IP address: Select the IP address from the list

Rule Name: fn_Synchronize_Instrmts

Parameter List: Instrument Table Name or LEDGER_STAT

4.3.5 Exception Messages

The SYNCHRONIZE_INSTRUMENT procedure may cause some exceptions to appear. The text and explanation for each of these exceptions follows. If you call the procedure from a PL/SQL block you may want to handle them so that your program can proceed.

- **Exception 1:** Table is not an Instrument or LEDGER_STAT table
The exception message reads:

ORA-20002 Cannot process: table_name is not an OFSA Instrument or Ledger type table having all leaf columns.

This exception occurs when the table_name parameter is not designated as an Instrument table or LEDGER_STAT table in the OFSAA Metadata. The procedure identified such tables based upon the Table Classification (Instrument or LEDGER_STAT).

- **Exception 2:** Table has invalid seeded FINANCIAL_ELEM_ID values
The exception message reads:

ORA-20004 Cannot process: table_name has new FINANCIAL_ELEM_ID values that are within seeded range (less than 10000).

This error occurs when user-defined leaf values are found in the DIM_FINANCIAL_ELEMENTS_B table within the FDM Reserved seeded data range. The FDM seeded data range for OFSA_LEAF_DESC is WHERE LEAF_NUM_ID=0 and LEAF_NODE<10000. If more records are found in this range than the seeded count for FDM version, the Synchronize Instrument procedure displays the error message and terminates. Delete any user-defined Financial Element leaf values within the FDM seeded data range in order to resolve this problem.

- **Exception 3:** Description table does not exist
The exception message reads:

Note

'Description Table Name' code table could not be synchronized due to :ORA-00942: table or view does not exist. These tables must be synchronized manually. Failure to do so may result in inaccurate reports.

This error occurs while inserting into the description table when user defined values are found in the Code column in dimension member and description table does not exist.

4.4 Stage Synchronization

The Stage Synchronization procedure identifies the new Code values from given stage table which will be treated as source and inserts default description entries for them into the appropriate Dimension and CD/MLS tables. The procedure performs both of these synchronizations simultaneously.

OFSAAI requires that all Dimension and Code values have a corresponding description. This is required for any OFSAA reporting operation to return the correct results.

Topics:

- [Tables Requiring Synchronization](#)
- [Dimension Member Synchronization](#)
- [Codes Synchronization](#)
- [Executing the SYNCHRONIZE_INSTRUMENT Procedure](#)
- [Exception Messages](#)

4.4.1 Tables Requiring Synchronization

Dimension member and Code value synchronization is required only for Instrument and LEDGER_STAT tables. Instrument tables are defined as all tables with the Instrument Table Classification (table_classification_cd = 20) on which all of the defined Key Dimension Columns exist.

4.4.2 Dimension Member Synchronization

The SYNCHRONIZE_STAGE procedure synchronizes the dimension member tables and Stage tables using default values for member descriptions and other information columns. This procedure performs the following functions:

1. Checks the specified table (Stage Table) for new dimension members in each of that table's key dimension columns and adds the new dimension value as leaf members to the respective dimension member tables.
2. Adds the new dimension member to the corresponding attribute tables with default values for mandatory attributes.
3. When new dimension members are added to the dimension tables these members include 'No Description_<ID column Value>' in the DESCRIPTION column and contain default values for mandatory attributes.

4.4.3 Code Synchronization

The SYNCHRONIZE_STAGE procedure identifies code values in stage tables for which a corresponding description does not exist in key dimension or CD/MLS tables and inserts a default description into the appropriate Code Description object. This applies only to CODE columns categorized as User-Editable or User-Defined (see the table classification).

CODE columns for which OFSAA reserves all of the values are not updated by this procedure. This procedure displays a warning message for any unidentified values in the CODE columns where OFSAA reserves the entire range.

If the resulting object is a User-Editable or User-Defined Code Description object (checks from REV_TABLE_CLASS_ASSIGNMENT table), then the procedure inserts a default description for any code values for which a description record does not already exist.

If the resulting object is an OFSAA Reserved Code Description object, then the procedure outputs a warning message indicating how many invalid code values exist in the specified Stage table in the message log file FSI_MESSAGE_LOG.

For example, if you are synchronizing the STG_INVESTMENTS table, the procedure queries all of the CODE columns on this table. An example of a Reserved CODE column is

ACCRUAL_BASIS_CD. If the procedure finds any code values in this column that are not present in the corresponding Code Description object (FSI_ACCRUAL_BASIS_CD), it gives output as an error message indicating the number of invalid values present.

OFSAI Reserved Code Description objects are identified by the following SQL statement:

```
SELECT distinct member_base_table_name, description_table_name FROM
rev_description_tables,REV_DIMENSIONS_B WHERE
rev_description_tables.member_base_table_name=REV_DIMENSIONS_B.Member_b_Table_Nam
e and
rev_description_tables.description_join_column_name=REV_DIMENSIONS_B.MEMBER_DISPL
AY_CODE_COL AND stg_table_name = 'STG_INVESTMENTS' AND
rev_dimensions_b.member_code_column is not null AND description_table_name IN
(SELECT table_name FROM rev_table_class_assignment WHERE table_classification_cd
= 197) AND STG_CD_COLUMN_NAME IN (SELECT column_name FROM user_tab_columns WHERE
table_name = 'STG_INVESTMENTS');
```

An example of a User-Editable CODE column is BRANCH_CD. If the procedure finds any code values in BRANCH_CD in the FSI_BRANCH_CD table that do not have a description in FSI_BRANCH_MLS, it creates a default description 'No Description_<CD Column Value>' for each value. It is now, up to the users to update these descriptions as appropriate.

User-Editable Code Description objects are identified by the following SQL statement:

```
SELECT distinct member_base_table_name, description_table_name FROM
rev_description_tables,REV_DIMENSIONS_B,user_tables WHERE
rev_description_tables.member_base_table_name=REV_DIMENSIONS_B.Member_b_Table_Nam
e and
rev_description_tables.description_join_column_name=REV_DIMENSIONS_B.MEMBER_DISPL
AY_CODE_COL and
rev_description_tables.member_base_table_name=user_tables.Table_Name AND
stg_table_name = 'STG_INVESTMENTS' AND rev_dimensions_b.member_code_column is not
null AND description_table_name NOT IN (SELECT table_name FROM
rev_table_class_assignment WHERE table_classification_cd = 197);
```

4.4.4 Executing the Synchronize Stage

You can execute the SYNCHRONIZE_STAGE procedure either from SQL*Plus or from within a PL/SQL block or from Batch Maintenance window within OFSAI framework.

1. To run the procedure from SQL*Plus, logon to SQL*Plus as the Schema Owner. The procedure requires two parameters: The table name to be synchronized and As of Date.
2. Identify the table name parameter by enclosing it in single quotes and uppercase, as shown in the following two examples. The syntax for calling the procedure is:

```
Declare output number;Beginfsi_sync_stage('Batch_Id','TABLE_NAME', output)End;
```

where table_name is the name of an Stage table.

An example of running the stored procedure from SQL*Plus for the STG_INVESTMENTS table follows:

```
SQL>var output
number;SQL>fsi_sync_stage('INFODOM_20101231','STG_INVESTMENTS',:output);
```

To execute the stored procedure from within a PL/SQL block or procedure, see the example that follows. To call the procedure as often as required to synchronize all of your stage tables, the appropriate table name and AS_OF_DATE is enclosed in single quotes.

```
SQL> declareoutput
number;beginfsi_sync_stage('INFODOM_20101231','STG_INVESTMENTS', output);end;/
```

3. To execute the procedure from OFSAAI Batch Maintenance, create a new Batch with the task as TRANSFORM DATA and specify the following parameters for the task:

DatSTORE Type: Select the appropriate datSTORE type from the list.

DatSTORE Name: Select the appropriate datSTORE name from the list.

IP address: Select the IP address from the list.

Rule Name: synchronize_stage

Parameter List: Stage Table Name

4.4.5 Exception Messages

The SYNCHRONIZE_INSTRUMENT procedure may cause some exceptions to appear. The text and explanation for each of these exceptions follows. If you call the procedure from a PL/SQL block you may want to handle them so that your program can proceed.

- **Exception 1:** Table is not an Instrument or LEDGER_STAT table

The exception message reads:

ORA-20002 Cannot process: table_name is not an OFSA Instrument or Ledger type table having all leaf columns.

This exception occurs when the table_name parameter is not designated as an Instrument table or LEDGER_STAT table in the OFSAA Metadata. The procedure identified such tables based upon the Table Classification (Instrument or LEDGER_STAT).

- **Exception 2:** Table has invalid seeded FINANCIAL_ELEM_ID values

The exception message reads:

ORA-20004 Cannot process: table_name has new FINANCIAL_ELEM_ID values that are within seeded range (less than 10000).

This error occurs when user-defined leaf values are found in the DIM_FINANCIAL_ELEMENTS_B table within the FDM Reserved seeded data range. The FDM seeded data range for OFSA_LEAF_DESC is WHERE LEAF_NUM_ID=0 and LEAF_NODE<10000. If more records are found in this range than the seeded count for FDM version, the Synchronize Instrument procedure displays the error message and terminates. Delete any user-defined Financial Element leaf values within the FDM seeded data range in order to resolve this problem.

- **Exception 3:** Description table does not exist

The exception message reads:

Note

'Description Table Name' code table could not be synchronized due to :ORA-00942: table or view does not exist. These tables must be synchronized manually. Failure to do so may result in inaccurate reports.

This error occurs while inserting into the description table when user defined values are found in the Code column in dimension member and description table does not exist.

4.5 Ledger Data Loader

The `LEDGER_STAT` load utility is an Oracle stored procedure used to load your ledger data into the Oracle Financial Services Analytical Applications (OFSA) `LEDGER_STAT` table.

Topics:

- [Features of the Load procedure](#)
- [Setup for the LEDGER_STAT Load utility](#)
- [Exception Messages](#)
- [Tables Cleanup after Truncation of Ledger_Stat](#)

There are three types of load tables that can be used for loading ledger data.

- Type I (`FISCAL_ONE_MONTH`): Load table contains `ONE_MONTH` column for storing data corresponding to one of the twelve fiscal months.
- Type II (`FISCAL_RANGE`): Load table contains M1 to M12 columns for storing data corresponding to twelve fiscal months.
- Type III (`CALENDAR_MONTHS`): Load table contains `AS_OF_DATE` for storing data corresponding to an as-of-date. While Type II table contains ledger data across fiscal months in a single row, Type III contains the same information in multiple rows. Type III supports calendar dates and data can be for one or multiple dates.

ASCII Ledger data is loaded into any of the above staging or load tables using F2T component of OFSAI framework. This component can be used for loading any flat file data into tables. For more information on how to load data using F2T, see OFSAI User Guide.

`LEDGER_STAT` load utility is a PL/SQL procedure and loads data from the above staging tables into `LEDGER_STAT` table, based on the configuration. Runtime parameters, such as the name of the load table, which all columns to load, ADD or REPLACE update functionality, and whether or not to create offset records are passed as parameters to the procedure and these are inserted into the Load Batch table (`FSI_LS_LOAD_BATCH`).

The procedure is implemented as an Oracle PL/SQL stored procedure so it can be invoked from SQL*Plus or Batch execution screen within OFSAI Batch Maintenance component. Input parameters are read from the batch/parameter table and validated for correctness, completeness and consistency before the load begins. Parameter errors are written to a Message column in the batch/parameter table and `FSI_MESSAGE_LOG` table. Runtime statistics are written to the batch/parameter record following completion of the load for that record.

Note

For supporting loading `LEDGER_STAT` from Type III staging table, a global temporary table (GTT) is created within database. Data is moved from global temporary table into `LEDGER_STAT` table.

4.5.1 Parameters

The following are the parameters to the UNDO engine:

- Batch Run ID (Typical format is `INFODOM_BATCHNAME_MISDATE_EXECUTIONSEQUENCE`)

- IdentityCode-As Of Date
- Mode Of Execution

Mode of execution for undoing the ledger load is 'L'. Identity Code and As Of Date are passed in the second parameters with a Hyphen (-) in between.

OFSAAI Batch execution framework is used to invoke the Undo engine.

4.5.2 Undo Mechanism

- Undo Engine will set the `STATUS_FLAG` column in `FSI_DATA_IDENTITY` table to 'U' to indicate the start of operation.
- The engine code reads all the records from `FSI_DATA_IDENTITY` table. For each record that is read, it checks whether

```
SOURCE_TYPE = 0
```

```
TABLE_NAME = 'ledger_stat'
```

```
IDENTITY_CODE = <as entered by user>, and
```

```
AS_OF_DATE = <as entered by user>
```

After reading all the records from `FSI_DATA_IDENTITY` table, if a matching record is not found then an error message is logged in the `FSI_MESSAGE_LOG` table. However, if a matching record is found, then the Undo engine starts the undo process as detailed following.

- Based on the `IDENTITY_CODE` and Year specified in the `AS_OF_DATE`, engine prepares and executes an update query to set the amount for the month specified in the `AS_OF_DATE` to zero and attaches a decode statement to calculate the Year To Date amount values from the Period Start month to Period End month. It also attaches any data filter if present to this query.
- Engine also prepares and executes a delete query on `LEDGER_STAT` table, to delete all the records for which all the month values are 0 and `IDENTITY_CODE` equals to the value input by user. All entries relevant for the `IDENTITY_CODE` are also deleted from `FSI_DATA_IDENTITY` table.
- If the undo fails for any reason, status would be set as 'C'. If Undo is completed successfully, the entry will be removed from `FSI_DATA_IDENTITY` table.

4.5.3 Executing Undo Engine

To execute Undo Engine, follow these steps:

1. To execute the engine from OFSAAI Batch Maintenance, create a new Batch with the Task as RUN EXECUTABLE and specify the following parameters for the task:

Datastore Type: Select appropriate datastore from list

Datastore Name: Select appropriate name from the list

IP address: Select the IP address from the list

Parameter List: `./LEDGER_LOAD_UNDO.sh, <Identity Code>-<As_of_Date>, 'L'`

2. To execute the engine from command line, the following is the syntax:

```
./LEDGER_LOAD_UNDO.sh<parameters>
```

Parameters: `<Batch_Run_Id> <IdentityCode>-<As_of_date> 'L'`

Note

`AS_OF_DATE` should be passed in mm/dd/yyyy format. Ledger Load Undo can be executed with both `Wait =Y` (Synchronous) or `N` (Asynchronous).

4.5.4 Exception Messages

The ledger undo program throws both user defined exceptions and Oracle database related exceptions. These exception messages could be seen in `FSI_MESSAGES_LOG` table with the help of the `Batch_Run_Id` which was used during execution. The exception list includes all possible validations on the parameters that were passed and database related exceptions.

4.6 Data Slicing

Data Slicing is a utility that will segment instrument data into equal parts by populating a numeric value into the `DATA SLICE ID` column. Data slicing should be used together with Multi-processing which is described in detail under Appendix F – Process Tuning in the [FTP User Guide](#) and Appendix B – Performance Tuning in the [ALM User Guide](#).

The purpose of segmenting data into equal parts is to balance the data volumes which are handled by each sub-process that is launched when multi-processing is enabled. The goal of multi-processing is to efficiently utilize the maximum amount of processing power of the application server during peak processing which leads to significantly shorter overall processing time. Through benchmark testing, we have found that breaking instrument data down into equal segments via the `Data Slice ID` column, is the most efficient way to use multi-processing. The alternative is to use one or more key dimension columns such as `Organization Unit` and `Product ID`. The shortcoming of using these other dimensions for segmenting data is that data is not evenly distributed across these dimensions so you end up with a few large segments and a large number of small segments which is not optimal for processing. Because each data segment is handled by the engine via a dedicated sub process, evenly distributing the data into equal segments provides the best results.

Data Slices are utilized by the FTP, ALM and BSP engines only when multi-processing is enabled. The default multi-process setting is 1 Process. This means a single sub process is launched when an FTP, ALM or BSP process is started and this one sub process will iterate through all of the data until processing is complete. Using a single process is fine for implementation and testing, but in production, users should identify the number of sub processes that will lead to the best performance. During Process Tuning, users can increase the number of sub processes to 2, 4, 8 or greater numbers depending on the number of CPU's available on the server. In multi-processing, a value of 8 processes means that 8 sub processes are automatically launched and each sub process is responsible for processing all of the data for an entire data segment (`Data Slice Cd`). When a sub process finishes processing the data for a segment, a new sub process is launched and handles the next available data segment. This process repeats until all data segments have been processed. In terms relative improvements in performance, we have observed that a multi process value of 2 is approximately 2x faster than one, 4 is approximately 4x faster and 8 is approximately 8x faster. We have noticed diminishing returns as the number of processes increases, so users will need to iterate on the number of processes setting to find the optimal value.

4.6.1 Process flow

To utilize this functionality the following steps must be performed before executing an ALM, BSP or FTP process with multi-processing enabled:

- Create members for the Data slicing dimension through the dimension management screen (Dimension Management -> Members). Members should be numeric values like 1, 2, 3, and so on. It is advised that the maximum number of members be limited to the number of processors (CPU's) available on the Application Server for executing ALM, BSP and FTP processes in the environment.
- Execute the database procedure 'Upd_DataSlice_Dim_Code' (via a seeded Batch DATA_SLICE_POPULATE: This procedure updates the Data Slice Id column in every instrument table with the values defined in step 1. Instrument records are equally distributed among all members. For more information, refer to Executing the Data Slicing Function section.

Note

Numeric values are assigned individually within each instrument table. For example, if you have 4000 records in table 1 and 4000 in table 2 and 4 unique data slice values, each table will get updated with 1 through 4 data slices.

- Setup multi-processing (Navigate to Common Object Maintenance and select Process Tuning) Select the Data Slice dimension under the Data Slicing Columns section. See the [ALM](#) and [FTP](#) user guides for more detailed steps.

4.6.2 Executing the Data Slicing Function

You can execute this procedure from the Batch processing screen (Common Object Maintenance and select Operations).

1. A seeded batch `INFODOMNAME_DATA_SLICE_POPULATE` is provided with the Task component of TRANSFORM DATA. The following are parameters for the task:

Datstore Type: Select appropriate datstore from list

Datstore Name: Select appropriate name from the list

IP address: Select the IP address from the list

Rule Name: `Upd_DataSlice_Dim_Code`

Parameter List: Number Of Processes, List of Table names, Column Name.

The last two parameter values must be enclosed in single quote

Example: `8, 'FSI_D_MORTGAGES, FSI_D_BORROWINGS', 'DATA_SLICE_ID'`

2. If you want to execute this procedure directly from the database, for example using SQL Developer or a similar tool, the function `fn_POPULATE_DATA_SLICE_CODE` can be used. This function requires five parameters: `BATCH_RUN_ID`, `AS_OF_DATE`, `IN_NPROC`, `IN_SOURCETABLELIST`, `IN_COLUMN`.

The syntax for calling the procedure is:

```
function fn_POPULATE_DATA_SLICE_CODE(BATCH_RUN_ID varchar2 ,AS_OF_DATE IN
Varchar2,IN_NPROC IN Number, IN_SOURCETABLELIST INVarchar2, IN_COLUMN IN
Varchar2)
```

Where,

`BATCH_RUN_ID` is any string to identify the executed batch. When executed from Batch this parameter is automatically generated. When it is executed directly in database then an appropriate value must be passed.

Example:

'OFSALMINFO_DATA_SLICE_1' AS_OF_DATE in YYYYMMDD format enclosed in single quote.

Example:

'20190101'IN_NPROC this numeric value is used to identify the number of processes and will be same as number of members created for Data Slicing dimension.

Example:

IN_SOURCETABLELIST is the list of the tables with comma separated source values. Here, table name should be in single quote. If more than one table names are given then single quote should be at the beginning of list and another at the end of the list, for example :
'FSI_D_MORTGAGES, FSI_D_ BORROWINGS'. \

IN_COLUMN is the name of the column where Data Slide Id gets updated.

DATA_SLICE_ID is the seeded column. You can choose another column if applicable as long as it is present in the instrument tables.

3. Example:
4. Navigate to Common Object Maintenance and select Operations, and then select Batch Maintenance window.
5. Search for the data slicing batch (INFODOMNAME_DATA_SLICE_POPULATE) as displayed:
6. Edit the corresponding task as displayed. Example of Parameter List: 8, 'FSI_D_MORTGAGES, FSI_D_ BORROWINGS', 'DATA_SLICE_ID'
7. Navigate to Batch Execution.
8. Search for the batch as mentioned following and execute it using Batch Execution.

Note

When you are executing the batch, the batch execution date should be the same as the As of Date when the processes are getting executed.

- All the messages (including warning and error) are logged in the FSI_MESSAGE_LOG table.
- It is a mandatory step to execute the above procedure with relevant parameters prior to executing ALM, BSP or FTP processes if you want to make use of this feature.
- This feature is only relevant when multi-processing is enabled and will not result in any performance improvement when number of processes = 1.
- All parameters are mandatory in expected order and format.

5

Data Loaders

This chapter details the steps involved in executing various data loaders that are available within OFSAA. Data loaders move data from staging layer to processing layer.

Topics:

- [Dimension Loaders](#)
- [Simple Dimension Loader](#)
- [Historical Rates Data Loader](#)
- [Forecast Rate Data Loader](#)
- [Prepayment Rate Data Loader](#)
- [Stage Instrument Table Loader](#)
- [Customer T2T Loading](#)
- [DIM_Party Population](#)
- [Instrument Summary Table](#)
- [Transaction Summary Table Loader](#)
- [Ledger Data Loader](#)
- [Cash Flow Loader](#)
- [Pricing Management Transfer Rate Population Procedure](#)
- [ALMBI Transformation](#)
- [Hierarchy Transformation](#)
- [Dim Dates Population](#)
- [Fact Ledger Stat Transformation](#)
- [Financial Element Dimension Population](#)
- [Payment Pattern Loader](#)
- [GAP Limits Loader](#)
- [Material Currency Identifier](#)
- [Behaviour Pattern Loader](#)

5.1 Dimension Loaders

The Dimension Loader procedure populates dimension members, attributes and hierarchies from Staging dimension tables into dimension tables registered within OFSAAI AMHM framework. Users can view the members and hierarchies loaded by the dimension loader through AMHM screens.

Note

The dimension loaders (drmDataLoader, STGDimDataLoader, and simplifiedimloader) load the strings into one target language only, the target language is derived from the database-login-session using USERENV.

Refer to Support Note [1586342.1](#), if Hierarchy Filter is not reflecting correctly after making the changes to underlying Hierarchy.

Topics:

- [Enhancements to Support Alphanumeric Code in Dimensions](#)
- [Tables that are Part Of Staging](#)
- [Dimension Load Procedure](#)
- [Setting up Dimension Loader](#)
- [Executing the Dimension Load Procedure](#)
- [Exception Messages](#)
- [Executing the Dimension Load Procedure using Master Table approach](#)
- [Truncate Stage Tables](#)

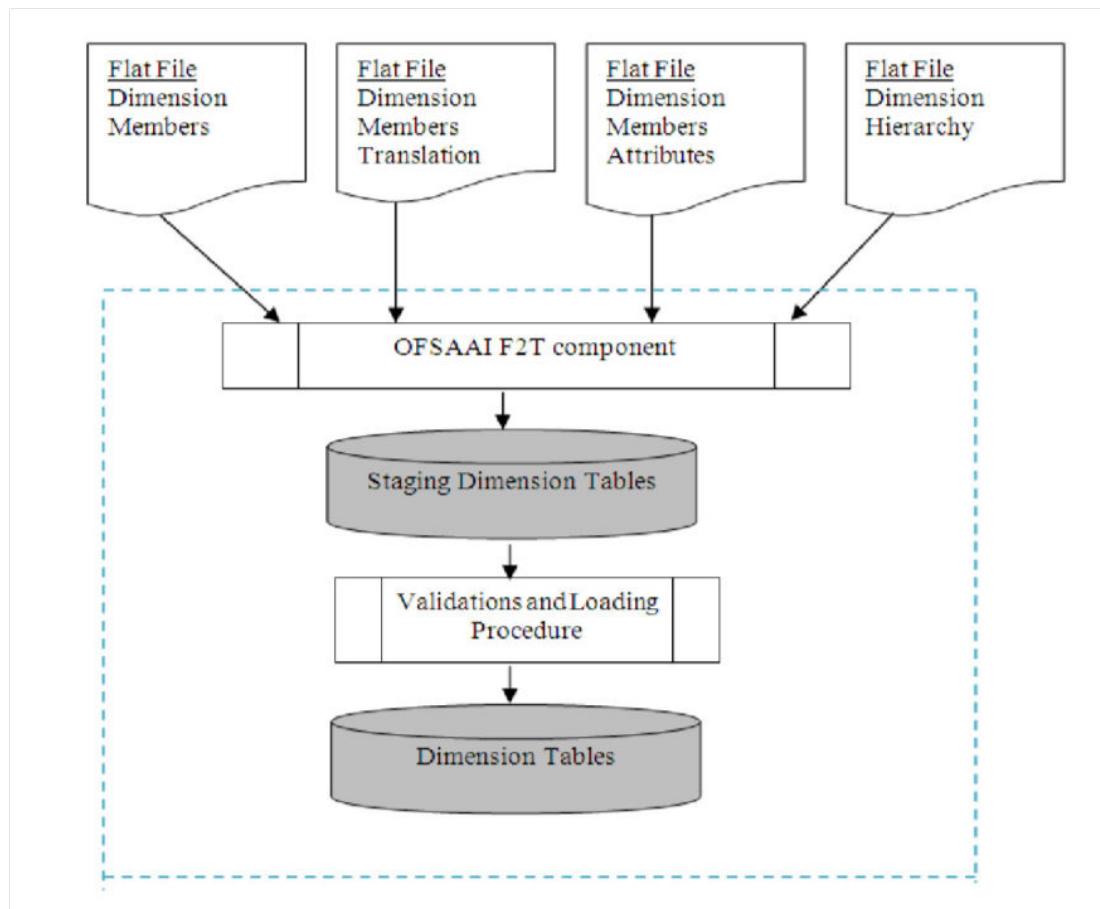
The dimension loader is used to:

- Load dimension members and their attributes from the staging area into Dimension tables that are registered with the OFSAAI AMHM framework.
- Create hierarchies in AMHM.
- Load hierarchical relationships between members within hierarchies from the staging area into AMHM.

Some of the features of the dimension loader are:

- Multiple hierarchies can be loaded from staging tables.
- Validations of members and hierarchies are similar to that of being performed within AMHM screens.
- Members can be loaded incrementally or fully synchronized with the staging tables.

Figure 5-1 Dimension Loader



5.1.1 Enhancements to Support Alphanumeric Code in Dimensions

Note

Dimension Loaders and UIs support capturing an alphanumeric code in addition to the numeric code.

The following Data Model components are required to support dimension member code storage; changes in {6.0/7.3.0/7.3.1} are as follows:

- Release 7.3.1: Dimension Configuration via manual updates to REV_DIMENSIONS_B columns: MEMBER_DATA_TYPE_CODE and MEMBER_CODE_COLUMN. (Also See: OFSAAI Installation & Configuration Guide 7.3 and AI Administration Guide)
- Release 6.0 (7.3): Stage Dimension Interface Table alphanumeric member code column (v_< DIM >_code).
- Release 6.0 (7.3): Stage Dimension Loader Program can directly load alphanumeric member codes

- Release 6.1.1: Some new columns are added to Staging & Processor tables as a part of FSDf. These are not required by EPM applications and not part of the T2T or FSI_D tables.

For further details on display of member codes in the user interfaces, see the [OFSAAI User Guide](#).

5.1.2 Tables that are Part Of Staging

Dimension data is stored in the following set of tables:

- STG_<DIMENSION>_B_INTF: Stores leaf and node member codes within the dimension.
- STG_<DIMENSION>_ TL_INTF: Stores names of leaf and node and their translations.
- STG_<DIMENSION>_ ATTR_INTF: Stores attribute values for the attributes of the dimension.
- STG_<DIMENSION>_ HIER_INTF: Stores parent-child relationship of members and nodes that are part of hierarchies.
- STG_ORG_UNIT_B_INTF: Stores leaf and node member codes within the organization unit dimension.
- STG_ORG_UNIT_TL_INTF: Stores names of leaf and node and their translations for the organization unit dimension.
- STG_ORG_UNIT_ATTR_INTF: Stores attribute values for the attributes of the organization unit dimension.
- STG_ORG_UNIT_HIER_INTF: Stores parent-child relationship of members and nodes that are part of hierarchies for the organization unit dimension.
- STG_HIERARCHIES_INTF: Stores master information related to hierarchies.

Data present in the above set of staging dimension tables are loaded into the following set of dimension tables.

- DIM_<DIMENSION>_ B: Stores leaf and node member codes within the dimension.
- DIM_<DIMENSION>_TL: Stores names of leaf and node and their translations.
- DIM_<DIMENSION>_ATTR: Stores attribute values for the attributes of the dimension.
- DIM_<DIMENSION>_HIER: Stores parent-child relationship of members and nodes that are part of hierarchies.
- REV_HIERARCHIES: Stores hierarchy related information.
- REV_HIERARCHY_LEVELS: Stores levels of the hierarchy.
- REV_HIER_DEFINITIONS: Stores definitions of the hierarchies.

Staging tables are present for all key dimensions that are configured within the OFSAAI framework. For any custom key dimension that is added by the Client, respective staging dimension tables like STG_<DIMENSION>_B_INTF, STG_< DIMENSION>_TL_INTF, STG_<DIMENSION>_ATTR_INTF, and STG_<DIMENSION>_HIER_INTF have to be created in the ERwin model.

5.1.3 Populating STG_<DIMENSION>_HIER_INTF Table

The STG_<DIMENSION>_HIER_INTF table is designed to hold hierarchy structure. The hierarchy structure is maintained by storing the parent child relationship in the table. In the following hierarchy there are 4 levels. The first level node is 100, which is the Total Rollup. The Total Rollup node will have the N_PARENT_DISPLAY_CODE and N_CHILD_DISPLAY_CODE as the same.

Column Name	Column Description
V_HIERARCHY_OBJECT_NAME	Stores the name of the hierarchy
N_PARENT_DISPLAY_CODE	Stores the parent Display Code
N_CHILD_DISPLAY_CODE	Stores the child Display Code
N_DISPLAY_ORDER_NUM	Determines the order in which the structure (nodes, leaves) of the hierarchy should be displayed. This is used by the UI while displaying the hierarchy. There is no validation to check if the values in the column are in proper sequence.
V_CREATED_BY	Stores the created by user. Hard coded as -1
V_LAST_MODIFIED_BY	Stores the last modified by user. Hard coded as -1

Hierarchy Structure

Table 5-1 Simple Data

V_HIERARCHY_OBJECT_NAME	N_PARENT_DISPLAY_CODE	N_CHILD_DISPLAY_CODE	N_DISPLAY_ORDER_NUM	V_CREATED_BY	V_LAST_MODIFIED_BY
INCOME STMT	100	100	1	-1	-1
INCOME STMT	100	1.23E+13	2	-1	-1
INCOME STMT	1.23E+13	1.23E+13	1	-1	-1
INCOME STMT	1.23E+13	10001	1	-1	-1
INCOME STMT	1.23E+13	10002	2	-1	-1
INCOME STMT	1.23E+13	1.23E+13	2	-1	-1
INCOME STMT	1.23E+13	10006	1	-1	-1
INCOME STMT	1.23E+13	10007	2	-1	-1
INCOME STMT	100	1.23E+13	3	-1	-1
INCOME STMT	1.23E+13	1.23E+13	2	-1	-1
INCOME STMT	1.23E+13	30005	1	-1	-1
INCOME STMT	1.23E+13	1.23E+13	1	-1	-1
INCOME STMT	1.23E+13	30006	1	-1	-1
INCOME STMT	1.23E+13	30007	2	-1	-1
INCOME STMT	1.23E+13	30008	3	-1	-1
INCOME STMT	1.23E+13	30009	4	-1	-1
INCOME STMT	100	1.23E+13	4	-1	-1
INCOME STMT	1.23E+13	3912228	1	-1	-1
INCOME STMT	3912228	20020	1	-1	-1
INCOME STMT	3912228	20021	2	-1	-1
INCOME STMT	3912228	20022	3	-1	-1

Column REV_DIMENSIONS_B.MEMBER_CODE_COLUMN

In release 7.3.1: With the introduction of alphanumeric support, REV_DIMENSIONS_B.MEMBER_CODE_COLUMN column becomes important for successful execution of the dimension loader program and subsequent T2Ts. The value in this column should be a valid code column from the relevant DIM_<DIMENSION>_B (key dimension) or FSI_<DIM>_CD (simple dimension) table. The Leaf_registration procedure populates this column. The value provided to the Leaf registration procedure should be the correct DIM_<DIM>_B.<DIM>_CODE or C_DISPLAY_CD column. Setting this will ensure that the values in this column are displayed for both numeric and alphanumeric dimensions as Alphanumeric Code in the UI. Configuration of

an alphanumeric dimension also requires manual update of the REV_DIMENSIONS_B.MEMBER_DATA_TYPE_CODE column.

For more information, see [OFSAAI Installation and Configuration Guide](#).

5.1.4 Dimension Load Procedure

This procedure performs the following functions:

- Gets the list of source and target dimension tables. The dimension tables for a given dimension are stored in REV_DIMENSIONS_B table. The stage tables for a given dimension are stored in FSI_DIM_LOADER_SETUP_DETAILS.
- The parameter Synchronize Flag can be used to completely synchronize data between the stage and the dimension tables. If the flag = 'Y' members from the dimension table which are not present in the staging table will be deleted. If the flag is 'N' the program merges the data between the staging and dimension table.
- The Loader program validates the members/attributes before loading them.

The program validates the number of records in the base members table - STG_<DIMENSION>_B_INTF and translation members table - STG_<DIMENSION>_TL_INTF. The program exits if the number of records does not match

In case values for mandatory attributes are not provided in the staging tables, the loader program populates the default value (as specified in the attribute maintenance screens within AMHM of OFSAAI) in the dimension table.

The program validates for data types of attribute value. For example an attribute that is configured as 'NUMERIC' cannot have non-numeric values.

Dimension Loader validates the attribute against their corresponding dimension table. If any of the attributes is not present, then an error message will be logged in FSI_MESSAGE_LOG table.

Dimension Loader will check the number of records in Dim_<Dim_Name>_B and Dim_<Dim_Name>_TL for the language. In case any mismatch is found, then an error will be logged and loading will be aborted.

- If all the member level validations are successful the loader program inserts the data from the staging tables to the dimension tables

Note

In release 6.0 (7.3) The stage dimension loader program is modified to move alphanumeric code values from STG_< DIMENSION >_B_INTF.V_< DIM >_CODE to DIM_< DIM >_B.< DIM >_CODE column. Previously, DIM_< DIM>_B.< DIM >_CODE column was populated using the fn_updateDimensionCode procedure from the code attributes. With this enhancement users can directly load alphanumeric values.

The fn_updateDimensionCode procedure is still available for users who do not want make any changes to their ETL procedures for populating the dimension staging tables (for example, STG_< DIMENSION >_B_INTF, STG_< DIMENSION>_ATTR_INTF).

- After this, the loader program loads hierarchy data from staging into hierarchy tables.
- In case of hierarchy data the loader program validates if the members used in the hierarchy are present in the STG_<DIMENSION>_B_INTF table.

- The program validates if the hierarchy contains multiple root nodes and logs error messages accordingly, as multiple root nodes are not supported.
- Dimension Loader will check special characters in Hierarchy. Hierarchy name with special characters will not be loaded.
- Following are the list of special characters which are not allowed in Hierarchy Name:
^&\'

After execution of the dimension loader, the user must execute the reverse population procedure to populate OFSA legacy dimension and hierarchy tables.

5.1.4.1 Dimension Leaf Member Set Up

Dimension Leaf values can have a maximum of 14 digits.

Only 26 key (processing) dimensions are allowed in the database. Examples of seeded key leaf types are Common COA ID, Organizational Unit ID, GL Account ID, Product ID, and Legal Entity ID.

The maximum number of columns that the Oracle database allows in a unique index is 32. This is the overriding constraint. After subtracting `IDENTITY_CODE`, `YEAR_S`, `ACCUM_TYPE_CD`, `CONSOLIDATION_CD`, and `ISO_CURRENCY_CD`, this leaves 27 columns available for Key Processing Dimensions (leaf dimensions). `BALANCE_TYPE_CD` is now part of the unique index so this brings the maximum number of leaf columns down to 26.

5.1.4.2 Deletion of Dimension Members used in a Hierarchy

There is an integrity check performed during dimension data loading to confirm if dimension members are included in a hierarchy definition. If they are included, these members should not be deleted from the dimension member pool. If dimension members are deleted or made inactive as part of the data load, the validation will return an error message, *cannot delete a member that is used as part of a hierarchy*.

If you wish to override this validation, an additional parameter can be passed to the Dimension Data Loader program(`fn_drmDataLoader`), for example: `force_member_delete`. The parameter can be set to Y or N. Inputting **Y** allows you to override the used in hierarchy dependency validation. Inputting **N** is the default behavior, which performs the validation check to confirm if members are used in a hierarchy or not.

Below is the function:

```
function fn_drmDataLoader(batch_run_id varchar2, as_of_date varchar2,
pDimensionId varchar2, pSynchFlag char default 'Y', force_member_delete char
default 'N')
```

5.1.5 Setting up Dimension Loader

`FSI_DIM_LOADER_SETUP_DETAILS` table should have record for each dimension that has to be loaded using the dimension loader. The table contains seeded entries for key dimensions that are seeded with the application.

The following are sample entries in the setup table:

Column Name	Description	Sample Value
<code>n_dimension_id</code>	This stores the Dimension ID	1
<code>v_intf_b_table_name</code>	Stores the name of the Staging Base table	<code>Stg_org_unit_b_intf</code>

v_intf_member_column	Stores the name of the Staging Member Column Name	V_org_unit_id
v_intf_tl_table_name	Stores the name of the Staging Translation table	Stg_org_unit_tl_intf
v_intf_attr_table_name	Stores the name of the Staging Member Attribute table	Stg_org_unit_attr_intf
v_intf_hier_table_name	Stores the name of the Staging Hierarchy table	Stg_org_unit_hier_intf
d_start_time	Start time of loader - updated by the loader program.	
d_end_time	End time of loader - updated by the loader program.	
v_comments	Stores Comments.	Dimension loader for organization unit.
v_status	Status updated by the Loader program.	
v_intf_member_name_col	Stores the name of the Member	V_org_unit_name
v_gen_skey_flag	Flag to indicate if surrogate key needs to be generated for alphanumeric codes in the staging. Applicable only for loading dimension data from master tables. Not applicable for loading dimension data from interface tables. Note: Although the application UI may display an alphanumeric dimension member ID, the numeric member ID is the value stored in member-based assumption rules, processing results, and audit tables. Implications for Object Migration: Numeric dimension member IDs should be the same in both the Source and Target environments, to ensure the integrity of any member-based assumptions you wish to migrate. If you use the Master Table approach for loading dimension data and have set it up to generate surrogate keys for members, this can result in differing IDs between the Source and Target and therefore would be a concern if you intend to migrate objects which depend on these IDs.	
v_stg_member_column	Name of the column that holds member code in the staging table. Applicable for loading dimension data from the master tables. (sample value v_org_unit_code) –this appears to be the alphanumeric code	v_org_unit_code v_gl_code v_common_coa_code v_prod_code v_entity_code v_party_id

v_stg_member_name_col	Name of the column that holds member name in the staging table. Applicable only for loading dimension data from master tables. Not applicable for loading dimension data from interface tables.	
v_stg_member_desc_col	Name of the column that holds description in the staging table. Applicable only for loading dimension data from master tables. Not applicable for loading dimension data from interface tables.	
v_stg_intf_member_column	Name of the column that holds member code in the staging table. Applicable for loading dimension data only from the interface tables.	v_org_unit_code v_gl_code v_common_coa_code v_prod_code v_lv_code v_cust_ref_code

Note

Ensure FSI_DIM_LOADER_SETUP_DETAILS.V_STG_MEMBER_COLUMN is updated as mentioned following for Legal Entity and Customer dimensions.

Dimension Loader Approach	V_STG_MEMBER_COLUMN N for Legal Entity	V_STG_MEMBER_COLUMN N for Customer
Using Interface Table (fn_drmDataLoader)	V_LV_CODE	V_CUST_REF_CODE
Using Master Table (fn_STGDimDataLoader)	V_ENTITY_CODE	V_PARTY_ID

5.1.6 Executing the Dimension Load Procedure

You can execute this procedure either from SQL*Plus or from within a PL/SQL block or from the Batch Maintenance window within OFSAAI framework.

1. To run the procedure from SQL*Plus, login to SQL*Plus as the Schema Owner. The function requires four parameters: Batch Run Identifier, As of Date, Dimension Identifier, Synchronize flag (Optional).
2. The syntax for calling the procedure is:

```
function fn_drmDataLoader(batch_run_id varchar2, as_of_date varchar2,
pDimensionId varchar2, pSynchFlag char default 'Y', force_member_delete char
default 'N')
```

where

BATCH_RUN_ID is any string to identify the executed batch.

AS_OF_DATE in the format YYYYMMDD.

pDIMENSIONID dimension id.

pSynchFlag this parameter is used to identify if a complete synchronization of data between staging and dimension table is required. The default value is 'Y'.

Note

With Synch flag N, data is moved from Stage to Dimension tables. Here, an appending process happens. You can provide a combination of new Dimension records plus the data that has undergone change. New records are inserted and the changed data is updated into the Dimension table. With Synch flag Y, the Stage table data will completely replace the Dimension table data. There are a couple of checks in place to ensure that stage_dimension_loader is equipped with similar validations that the UI provides. The Data Loader does a Dependencies Check before a member is deleted. The validation checks, if there are members used in the Hierarchy that are not present in the DIM_< DIM >_B table. This is similar to the process of trying to delete a member from the UI, which is being used in the Hierarchy definition. You are expected to remove or delete such Hierarchies from the UI before deleting a member.

For Example:

```
Declare num number;Begin num := fn_drmDataLoader
('INFODOM_20100405','20100405',1,'Y','N');End;
```

- To execute the procedure from the OFSAAI Batch Maintenance, create a new Batch with the Task as TRANSFORM DATA and specify the following parameters for the task:

Datstore Type: Select appropriate datstore from list

Datstore Name: Select appropriate name from the list

IP address: Select the IP address from the list

Rule Name: fn_drmDataLoader

Parameter List: Dimension ID, Synchronize Flag

The fn_drmdataloader function calls STG_DIMENSION_LOADER package which loads data from the stg_<dimension>_hier_intf to the dim_<dimension>_hier table.

From Release 8.0, RUNIT.sh utility is available to resave the UMM Hierarchy Objects. The data for AMHM hierarchies which is stored in dim_<dimension>_hier table is changed due to the fn_drmdataloader function, so the RUNIT.sh utility is executed to refresh the UMM hierarchies which have been implicitly created due to the AMHM hierarchies. This file resides under ficdb/bin area.

To run the utility directly from the console:

Navigate to \$FIC_DB_HOME/bin of OFSAAI FIC DB tier to execute RUNIT.sh file

The following parameter needs to be provided:

- INFODOM- Specify the information domain name whose hierarchies are to be refreshed. This is the first parameter and mandatory parameter
- USERID- specify the AAI user id who is performing this activity. This is second parameter and mandatory as well
- HIERARCHY- specify the hierarchy code to be refreshed. In case multiple hierarchies need to be refreshed the same can be provided and tilde (~) separated values. This is third parameter and non-mandatory parameter

For example: ./RUNIT.sh,<INFODOM>,<USERID>,<CODE1~CODE2~CODE3>

Note

In case the third parameter is not specified, then all the hierarchies present in the infodomain will be refreshed.

To run the utility through the Operations module:

- a. Navigate to the Operations module and define a batch.
- b. Add a task by selecting the component as RUN EXECUTABLE.
- c. Under Dynamic Parameter List panel, specify `./RUNIT.sh, <INFODOM>, <USERID>, <CODE1~CODE2~CODE3>` in the Executable field.

After saving the Batch Definition, execute the batch to resave the UMM Hierarchy Objects

5.1.7 Exception Messages

The text and explanation for each of these exceptions follows. If you call the procedure from a PL/SQL block you may want to handle these exceptions appropriately so that your program can proceed without interruption.

- **Exception 1:** Error. errMandatoryAttributes
This exception occurs when the stage Loader program cannot find any data default value for mandatory attributes.
- **Exception 2:** Error. errAttributeValidation
This exception occurs when there is a data type mis-match between the attribute value and configured data-type for the attribute.
- **Exception 3:** Error. errAttributeMemberMissing
If there is a mismatch in the count between the member's base and translation table.

5.1.8 Executing the Dimension Load Procedure using Master Table approach

FSI_DIM_LOADER_SETUP_DETAILS table should have a record for each dimension that has to be loaded. The table contains entries for key dimensions that are seeded with the application.

1. The following columns must be populated for user-defined Dimensions.

v_stg_member_column
v_stg_member_name_col
v_stg_member_desc_col

V_STG_INTF_MEMBER_COLUMN column is available in the FSI_DIM_LOADER_SETUP_DETAILS table to avoid manual configuration for Legal Entity and Customer dimensions issues.

fn_drmDataLoader refers to this column (V_STG_INTF_MEMBER_COLUMN) and fn_STGDimDataLoader refers to V_STG_MEMBER_COLUMN.

Following values must be seeded in these columns for Legal entity and Customer dimensions.

For other seeded dimensions the values in both columns will be the same.

Option	Description
V_STG_MEMBER_COLUMN (fn_STGDimDataLoader)	V_STG_INTF_MEMBER_COLUMN (fn_drmDataLoader)
V_ENTITY_CODE	V_LV_CODE
V_PARTY_ID	V_CUST_REF_CODE

2. Additionally, the FSI_DIM_ATTRIBUTE_MAP table should be configured with column attribute mapping data. This table maps the columns from a given master table to attributes.

N_DIMENSION_ID: This stores the Dimension ID

V_STG_TABLE_NAME: This holds the source Stage Master table

V_STG_COLUMN_NAME: This holds the column from the master table

V_ATTRIBUTE_NAME: This holds the name of the attribute the column maps to

V_UPDATE_B_CODE_FLAG: This column indicates if the attribute value can be used to update the code column in the DIM_<Dimension>_B table.

Note

fn_STGDimDataLoader does not use
FSI_DIM_ATTRIBUTE_MAP.V_UPDATE_B_CODE_FLAG

3. You can execute this procedure either from SQL*Plus or from within a PL/SQL block or from the Batch Maintenance window within OFSAAI framework. To run the procedure from SQL*Plus, login to SQL*Plus as the Schema Owner.

The function requires 5 parameters:

Batch Run Identifier , As of Date, Dimension Identifier , MIS-Date Required Flag, Synchronize flag (Optional).

The syntax for calling the procedure is:

```
function fn_STGDimDataLoader(batch_run_id varchar2, as_of_date varchar2,
pDimensionId varchar2, pMisDateReqFlag char default 'Y', pSynchFlag char default 'N')
```

where

BATCH_RUN_ID is any string to identify the executed batch.

AS_OF_DATE in the format YYYYMMDD.

pDIMENSIONID dimension id.

pMisDateReqFlag is used to identify if AS-OF_DATE should be used in the where clause to filter the data.

pSynchFlag is used to identify if a complete synchronization of data between staging and fusion table is required. The default value is 'Y'.

For Example

```
Declare num number;Begin num := fn_STGDimDataLoader
('INFODOM_20100405','20100405',1,'Y','Y');End;
```

4. To execute the procedure from OFSAAI Batch Maintenance, create a new Batch with the Task as TRANSFORM DATA.
5. Specify the following parameters for the task:

Datastore Type: Select appropriate datastore from list

Datastore Name: Select appropriate name from the list

IP address: Select the IP address from the list

Rule Name: fn_STGDimDataLoader

Parameter List: Dimension ID, Mis Date Required Flag , Synchronize Flag

Customer may face a problem while loading customer dimension into AMHM using the Master table approach.

Configuring the setup table for CUSTOMER dimension is pretty confusing while dealing with attributes like FIRST_NAME , MIDDLE_NAME and LAST_NAME.

Most customers would like to see FIRST_NAME , MIDDLE_NAME and LAST_NAME forming the name of the member within the customer dimension.

Currently the STG_DIMENSION_LOADER disallows concatenation of columns. Moreover the concatenation might not ensure unique values. As a solution to this problem we can work on the following options:

- Approach 1
 - a. Create a view on STG_CUSTOMER_MASTER table with FIRST_NAME, MIDDLE_NAME and LAST_NAME concatenated and identify this column as NAME.
 - b. Configure the name column from the view in FSI_DIM_LOADER_SETUP_DETAILS
 - c. Increase the size of DIM_CUSTOMER_TL.NAME column.
 - d. Disable the unique index on DIM_CUSTOMER_TL.
 - e. NAME or append Customer_code to the NAME column.
 - f. The NAME column will be populated into the DIM_CUSTOMER_TL.NAME column.
- Approach 2
 - a. Populate customer_code into the DIM_CUSTOMER_TL.NAME column.

5.1.8.1 Approach 1

As per Approach 1, follow these steps:

1. Create a view on STG_CUSTOMER_MASTER table with FIRST_NAME, MIDDLE_NAME and LAST_NAME concatenated and identify this column as NAME.
2. Configure the name column from the view in FSI_DIM_LOADER_SETUP_DETAILS
3. Increase the size of DIM_CUSTOMER_TL.NAME column.
4. Disable the unique index on DIM_CUSTOMER_TL.NAME or append Customer_code to the NAME column.
5. The NAME column will be populated into the DIM_CUSTOMER_TL.NAME column.

5.1.8.2 Approach 2

As per Approach 2, follow these steps:

- Populate customer_code into the DIM_CUSTOMER_TL.NAME column.

5.1.9 Updating DIM_<DIMENSION>_B <Dimension>_Code column with values from DIM_<DIMENSION>_ATTR table

The stage dimension loader procedure does not insert or update the <Dimension>_code column in the Dim_<Dimension>_B table. This is an alternate method for updating the <Dimension>_Code column in the Dim_<Dimension>_B table, retained to accommodate implementations prior to the enhancement where we enable loading the code directly to the dimension table instead of from the attribute table. It is not recommended for new installations. This section explains how the <Dimension>_code can be updated.

1. A new attribute should be created in the REV_DIM_ATTRIBUTES_B / TL table.

Note

You should use the existing CODE attribute for the seeded dimensions. PRODUCT CODE, COMMON COA CODE, and so on.

2. The fsi_dim_attribute_map table should be populated with values.

The following columns must be populated:

N_DIMENSION_ID (Dimension id)

V_ATTRIBUTE_NAME (The attribute name)

V_UPDATE_B_CODE_FLAG (This flag should be 'Y').

Any given dimension can have only one attribute with V_UPDATE_B_CODE_FLAG as 'Y'. This should only be specified for the CODE attribute for that dimension.

Example:

```
N_DIMENSION_ID 4V_ATTRIBUTE_NAME 'PRODUCT_CODE' V_UPDATE_B_CODE_FLAG 'Y'
V_STG_TABLE_NAME 'stg_product_master' V_STG_COLUMN_NAME 'v_prod_code'
```

Note

The values in V_STG_TABLE_NAME and V_STG_COLUMN_NAME are not used by the fn_updateDimensionCode procedure, however these fields are set to NOT NULL and should be populated.

3. Load STG_<DIMENSION>_ATTR_INTF table with data for the new ATTRIBUTE created.

The attribute values must first be loaded using the stage dimension loader procedure, fn_drmDataLoader, before running this procedure. This procedure will pull values from the DIM_<DIMENSION>_ATTR table. If these rows do not exist for these members prior to running this procedure, the DIM_<DIMENSION>_B.<DIMENSION>_CODE field will not be updated.

Execute the fn_updateDimensionCode function. The function updates the code column with values from the DIM_<DIMENSION>_ATTR table.

4. You can execute this procedure either from SQL*Plus or from within a PL/SQL block or from the Batch Maintenance window within OFSAAI framework.

To run the procedure from SQL*Plus, login to SQL*Plus as the Atomic Schema Owner. The function requires 3 parameters: Batch Run Identifier , As of Date, Dimension Identifier.

The syntax for calling the procedure is:

```
function fn_updateDimensionCode (batch_run_id varchar2, as_of_date varchar2,
pDimensionId varchar2)
```

where

BATCH_RUN_ID is any string to identify the executed batch.

AS_OF_DATE in the format YYYYMMDD.

pDIMENSIONID dimension id

For Example

```
Declare num number;Begin num := fn_updateDimensionCode
('INFODOM_20100405', '20100405',1 );End;
```

You need to populate a row in FSI_DIM_LOADER_SETUP_DETAILS.

For example, for FINANCIAL ELEM CODE, to insert a row into FSI_DIM_LOADER_SETUP_DETAILS, following is the syntax:

```
INSERT INTO FSI_DIM_LOADER_SETUP_DETAILS (N_DIMENSION_ID) VALUES ('0');
COMMIT;
```

5. To execute the procedure from OFSAAI Batch Maintenance, create a new Batch with the Task as TRANSFORM DATA and specify the following parameters for the task:

Datastore Type: Select appropriate datastore from list

Datastore Name: Select appropriate name from the list

IP address: Select the IP address from the list

Rule Name: Update_Dimension_Code

Parameter List: Dimension ID

5.1.10 Truncate Stage Tables

This procedure performs the following functions:

- The procedure queries the FSI_DIM_LOADER_SETUP_DETAILS table to get the names of the staging table used by the Dimension Loader program.
- The MIS Date option only works to the Master Table approach (fn_STGDimDataLoader) dimension loader. It is not applicable to dimension data loaded using the standard Dimension Load Procedure (fn_drmDataLoader).

Executing the Truncate Stage Tables

You can execute this procedure either from SQL*Plus or from within a PL/SQL block or from the Batch Maintenance window within OFSAAI framework.

- To run the procedure from SQL*Plus, login to SQL*Plus as the Schema Owner. The function requires 4 parameters – Batch Run Identifier, As of Date, Dimension Identifier, Mis Date Required Flag. The syntax for calling the procedure is:

```
function fn_truncateStageTable(batch_run_id varchar2, as_of_date varchar2,
pDimensionId varchar2, pMisDateReqFlag char default 'Y')
```

where

- BATCH_RUN_ID is any string to identify the executed batch.
- AS_OF_DATE in the format YYYYMMDD.
- pDIMENSIONID dimension id.

- `pMisDateReqFlag` is used to identify the data needs to be deleted for a given MIS Date. The default value is 'Y'.

For Example

```
Declare num number; Begin num := fn_truncateStageTable
('INFODOM_20100405', '20100405' ,1, 'Y' ); End;
```

- To execute the procedure from OFSAAI Batch Maintenance, create a new Batch with the Task as TRANSFORM DATA and specify the following parameters for the task:
 - **Datastore Type:** Select appropriate datastore from list
 - **Datastore Name:** Select appropriate name from the list
 - **IP address:** Select the IP address from the list
 - **Rule Name:** `fn_truncateStageTable`
 - **Parameter List:** Dimension ID, MIS-Date required Flag

5.2 Simple Dimension Loader

Note

Currently the dimension loader program works only for key dimensions.

Simple Dimension Loader provides the ability to load data from stage tables to Simple dimension tables.

For example, the user can load data into `FSI_ACCOUNT_OFFICER_CD` and `FSI_ACCOUNT_OFFICER_MLS` using the Simple Dimension Loader program.

Simple dimension of type 'writable and editable' can use this loading approach. This can be identified by querying

```
rev_dimensions_b.write_flag = 'Y', rev_dimensions_b.dimension_editable_flag = 'Y'
and rev_dimensions_b.simple_dimension_flag = 'Y'.
```

Topics:

- [Creating Simple Dimension Stage Table](#)
- [Configuration of Setup Tables](#)
- [Executing the Simple Dimension Load Procedure](#)
- [Exception Messages](#)

5.2.1 Simple Dimension Stage Table

You can create stage tables for the required simple dimensions by using the following template:

Table 5-2 STG_<DIM>_MASTER

COLUMN_NAME	DATA TYPE	PRIMARY KEY	NULLABLE
<code>v_< DIM>_display_code</code>	Varchar2(10)	Y	N
<code>d_Mis_date</code>	Date	Y	N

Table 5-2 (Cont.) STG_<DIM>_MASTER

v_Language	Varchar2(10)	Y	N
v_< DIM>_NAME	Varchar2(40)		N
v_Description	Varchar2(255)		N
v_Created_by	Varchar2(30)		Y
v_Modified_by	Varchar2(30)		Y

Here is a sample structure:

Table 5-3 STG_ACCOUNT_OFFICER_MASTER

COLUMN_NAME	DATA TYPE	PRIMARY KEY	NULLABLE
v_acct_officer_display_code	Varchar2(10)	Y	N
d_Mis_date	Date	Y	N
v_Language	Varchar2(10)	Y	N
v_Name	Varchar2(40)		N
v_Description	Varchar2(255)		N
v_Created_by	Varchar2(30)		Y
v_Modified_by	Varchar2(30)		Y

Here are some examples:

- Example For FSI CD/MLS tables:**

```
CREATE TABLE <XXXXX>_FSI_<DIM>_CD -- ACME_FSI_ACCT_STATUS_CD (<DIM>_CD
NUMBER(5) -- ACCT_STATUS_CD ,LEAF_ONLY_FLAG VARCHAR2(1) ,ENABLED_FLAG
VARCHAR2(1) ,DEFINITION_LANGUAGE VARCHAR2(10) ,CREATED_BY
VARCHAR2(30) ,CREATION_DATE DATE ,LAST_MODIFIED_BY
VARCHAR2(30) ,LAST_MODIFIED_DATE DATE <dim>_display_CD VARCHAR2(10) );
```
- Example for FSI_<DIM>_MLS table:**

```
CREATE TABLE <XXXXX>_FSI_<DIM>_MLS -- ACME_FSI_ACCT_STATUS_CD (<DIM>_CD
NUMBER(5) -- ACCT_STATUS_CD ,LANGUAGE VARCHAR2(10) ,<DIM> VARCHAR2(40) --
ACCT_STATUS ,DESCRIPTION VARCHAR2(255) ,CREATED_BY VARCHAR2(30) ,CREATION_DATE
DATE ,LAST_MODIFIED_BY VARCHAR2(30) ,LAST_MODIFIED_DATE DATE );
```

Note

FSI_<DIM>_CD and FSI_<DIM>_MLS should follow the same standards as mentioned above, else Loader will not work as expected.

5.2.2 Configuration of Setup Tables

The Configuration of Setup Tables can be done with REV_DIMENSIONS_B table:

5.2.2.1 REV_DIMENSIONS_B Table

The REV_DIMENSIONS_B table holds the following target table information:

The target FSI_<DIM>_CD/MLS table can be retrieved from REV_DIMENSIONS_B table as follows:

- `dimension_id`: Holds the id of the simple dimension that needs to be loaded.
- `member_b_table_name`: Holds the name of the FSI_<DIM>_CD target table. For example, FSI_ACCOUNT_OFFICER_CD
- `member_tl_table_name`: Holds the name of the FSI_<DIM>_MLS table name. For example, FSI_ACCOUNT_OFFICER_MLS
- `member_col`: Holds the Column Name for which Surrogate needs to be generated. For example, ACCOUNT_OFFICER_CD
- `member_code_column`: Holds the Name of the joining column name from FSI_<DIM>_CD Display code column. For example, ACCOUNT_OFFICER_DISPLAY_CD
- `key_dimension_flag`: N
- `dimension_editable_flag`: Y
- `write_flag`: Y
- `simple_dimension_flag`: Y

5.2.2.2 Setup Table Configuration Mapping

The FSI_DIM_LOADER_SETUP_DETAILS stores the STG_<DIM>_MASTER table details as follows:

Table 5-4 FSI_DIM_LOADER_SETUP_DETAILS

FSI_DIM_LOADER_SETUP_DETAILS	STG_<DIM>_MASTER
N_DIMENSION_ID	<dimension_id> For example, 617
V_INTF_B_TABLE_NAME	Stage table name For example, STG_ACCOUNT_OFFICER_MASTER
V_GEN_SKEY_FLAG	Default will be 'Y', it generates Surrogate Key. When 'N' then stage display code column will be used as a surrogate key. For example, FSI_ACCOUNT_OFFICER_CD.ACCOUNT_OFFICER_DISPLAY_CD should be numeric.
V_STG_MEMBER_COLUMN	Stores the stage display code column. For example, STG_ACCOUNT_OFFICER_MASTER.v_acct_officer_display_code
V_STG_MEMBER_NAME_COL	Stores the stage column name. For example, STG_ACCOUNT_OFFICER_MASTER.v_Name
V_STG_MEMBER_DESC_COL	Stores the stage description column name. For example, STG_ACCOUNT_OFFICER_MASTER.v_description

5.2.3 Executing the Simple Dimension Load Procedure

There are two ways to execute the simple dimension load procedure: Running Procedure Using SQL*Plus and Simple Dimension Load Procedure Using OFSAAI Batch Maintenance.

1. To run the procedure from SQL*Plus, login to SQL*Plus as the Schema Owner:

```
function fn_simplifiedimloader(batch_run_id VARCHAR2, as_of_date VARCHAR2,
pdimensionid VARCHAR2,pMisDateReqFlag char default 'Y', psynchflag CHAR
DEFAULT 'N') SQLPLUS > declareresult number;beginresult := fn_simplifiedimloader
('SimpleDIIM_BATCH1','20121212','730','N','Y');end;/
```

BATCH_RUN_ID is any string to identify the executed batch.

AS_OF_DATE is in the format YYYYMMDD.

pDIMENSIONID is the dimension ID.

pSynchFlag this parameter is used to identify if a complete synchronization of data between staging and dimension table is required. The default value is 'Y'.

pMisDateReqFlag : Filter will be placed on the input stage table to select only the records which falls on the given as_of_date. Default value is Y. If complete stage table data needs to be considered, then it should be passed 'N'.

Note

With Synch flag N, data is moved from Stage to Dimension tables. Here, an appending process happens. You can provide a combination of new Dimension records plus the data that has undergone change. New records are inserted and the changed data is updated into the Dimension table. With Synch flag Y, the Stage table data will completely replace the Dimension table data.

2. To execute Simple Dimension Loader from OFSAAI Batch Maintenance, a seeded Batch is provided.

The batch parameters are:

Datastore Type: Select the appropriate datastore from list

Datastore Name: Select the appropriate name from the list

IP address: Select the IP address from the list

Rule Name: fn_simplifiedimloader

Parameter : 'Pass the dimension id for which DT needs to be executed, psynchflag'

For example, '730,N,Y

Note

In case of FSI_ACCOUNT_OFFICER_CD query:SELECT dimension_id FROM rev_dimensions_b where member_b_table_name = 'FSI_ACCOUNT_OFFICER_CD'Pass the dimension_id.

Psynchflag: By default it is N, data is moved from Stage to Dimension tables. Here, an appending process happens. You can provide a combination of new Dimension records plus the data that has undergone change. New records are inserted and the changed data

is updated into the Dimension table. With Synch flag 'Y', the Stage table data will completely replace the Dimension table data.

5.2.4 Exception Messages

Below are the list of error messages which can be viewed in view log from UI or fsi_message_log table from back end filtering for the given batch id. On successful completion of each task, messages gets into log table.

In the event of failure, following are the list of errors that may occur during the execution:

- **Exception 1:** When REV_DIMENSIONS_B is not having proper setup details.
Meaning: For Simple Dimension write_flag, simple_dimension_flag, dimension_editable_flag should be Y in rev_dimensions_b for the given Dimension id.
- **Exception 2:** When FSI_DIM_LOADER_SETUP_DETAILS table is not having proper set up details.
Meaning: Setup details are not found for the dimension id.
- **Exception 3:** When Display code Column is non numeric and trying to use as a surrogate key.
Meaning: Display code Column should be numeric as v_gen_skey_flag N

5.3 Historical Rates Data Loader

Historical data for currency exchange rates, interest rates and economic indicators can be loaded into the OFSAA historical rates tables through the common staging area. The T2T component within OFSAA framework is used to move data from the Stage historical rate tables into the relevant OFSAA processing tables. After loading the rates, users can view the historical rate data through the OFSAA Rate Management UI's.

Topics:

- [Tables Related to Historical Rates](#)
- [Populating Stage Tables](#)
- [Executing the Historical Rates Data Loader T2T](#)
- [Re-Load Of Historical Rates](#)

5.3.1 Tables Related to Historical Rates

Historical rates are stored in the following staging area tables:

- **STG_EXCHANGE_RATE_HIST:** This staging table contains the historical exchange rates for Currencies used in the system. Following columns are available:

Table 5-5 STG_EXCHANGE_RATE_HIST

Column Name	Column Details
V_QUOTE_TYPE	VARCHAR2(5 CHAR)
V_RATE_DATA_ORIGIN	VARCHAR2(20 CHAR)

- **STG_IRC_RATE_HIST:** This staging table contains the historical interest rates for the Interest Rate codes used in the system.

- `STG_IRC_TS_PARAM_HIST`: This staging table contains the historical interest rate term structure parameters, used by the Monte Carlo engine.
- `STG_ECO_IND_HIST_RATES`: This staging table stores the historical values for the Economic Indicators used in the system.

Historical rates in OFSAA Rate Management are stored in the following processing tables:

- `FSI_EXCHANGE_RATE_HIST`: This table contains the historical exchange rates for the Currencies used in the system.
- `FSI_IRC_RATE_HIST`: This table contains the historical interest rates for the Interest Rate codes used in the system.
- `FSI_IRC_TS_PARAM_HIST`: This table stores the historical interest rate term structure parameters, used by the Monte Carlo engine.
- `FSI_ECO_IND_HIST_RATES`: This table contains the historical values for the Economic Indicators used in the system.

5.3.2 Populating Stage Tables

Data for historical rates commonly comes from external systems. Such data must be converted into the format of the staging area tables. This data can be loaded into the staging area using the F2T component of the OFSAAI framework. Users can view the loaded data by querying the staging tables and various log files associated with the F2T component.

5.3.3 Executing the Historical Rates Data Loader T2T

You can launch the Historical Rates Data Loader from the following:

- Interest Rates Summary page
- PL/SQL block
- Operations Batch
- To launch from the Interest Rates Summary page:
 - Click the Data Loader icon on the Interest Rates Summary grid toolbar.
 - A warning message will appear: Upload all available Interest Rates and Parameters?
 - Click Yes. The process will load all valid data included in the staging table.

There are four pre-defined T2T mappings configured and seeded in OFSAA for the purpose of loading historical rates. These can be executed from the Batch Maintenance within OFSAAI.

To execute the Historical Exchange Rates Data Loader, create a new Batch and specify the following parameters:

- **Datstore Type:** Select appropriate datstore from the drop down list
- **Datstore Name:** Select appropriate name from the list. Generally it is the Infodom name.
- **IP address:** Select the IP address from the list
- **Rule Name:** `T2T_EXCHANGE_RATE_HIST`
- **Parameter List:** No Parameter is passed. The only parameter is the As of Date selection which is made when the process is executed.

To execute the Historical Interest Rates Data Loader, create a new Batch and specify the following parameters:

- Datastore Type: Select appropriate datastore from the drop down list
- Datastore Name: Select appropriate name from the drop down list
- IP address: Select the IP address from the list
- Rule Name: T2T_IRC_RATE_HIST
- Parameter List: No Parameter is passed. The only parameter is the As of Date selection which is made when the process is executed.

To execute the Historical Term Structure Parameter Data Loader, create a new Batch and specify the following parameters:

- Datastore Type: Select appropriate datastore from list
- Datastore Name: Select appropriate name from the list
- IP address: Select the IP address from the list
- Rule Name: T2T_IRC_TS_PARAM_HIST
- Parameter List: No Parameter is passed. The only parameter is the As of Date selection which is made when the process is executed.

To execute the Historical Economic Indicator Data Loader, create a new Batch and specify the following parameters:

- Datastore Type: Select appropriate datastore from the drop down list
- Datastore Name: Select appropriate name from the drop down list
- IP address: Select the IP address from the list
- Rule Name: T2T_ECO_IND_HIST_RATES
- Parameter List: No Parameter is passed. The only parameter is the As of Date selection which is made when the process is executed.

After executing any of the above batch processes, check the T2T component logs and batch messages to confirm the status of the data load.

The T2T component can fail under the following scenario:

- Unique constraint error: Target table may already contain data with the primary keys that the user is trying to load from the staging area.

5.3.4 Re-Load Of Historical Rates

The T2T component can only perform Insert operations. In case the user needs to perform updates, previously loaded records should be deleted before loading the current records. Function `fn_deleteFusionTables` is used for deleting the records in the target that are present in the source. This function removes rows in the table if there are matching rows in the Stage table. This function requires entries in the `FSI_DELETE_TABLES_SETUP` table to be configured. Configure the following table for all columns that need to be part of the join between the Stage table and Equivalent table.

Users can create new or use existing Data Transformations for deleting a Table. The parameters for the Data Transformation are:

- 'Table to be deleted'
- Batch run ID

- As of Date

Column Name	Column Description	Sample Value
STAGE_TABLE_NAME	Stores the source table name for forming the join statement	STG_LOAN_CONTRACTS
STAGE_COLUMN_NAME	Stores the source column name for forming the join statement	V_ACCOUNT_NUMBER
FUSION_TABLE_NAME	Stores the target table name for forming the join statement	FSI_D_LOAN_CONTRACTS
FUSION_COLUMN_NAME	Stores the target column name for forming the join statement	ACCOUNT_NUMBER

Note

Insert rows in `FSI_DELETE_TABLES_SETUP` for all columns that can be used to join the stage with the equivalent table. In case if the join requires other dimension or code tables, a view can be created joining the source table with the respective code tables and this view can be part of the above setup table.

5.4 Forecast Rate Data Loader

The Forecast Rate Data Loader procedure loads forecast rates into the OFSAA ALM Forecast rates processing area tables from staging tables. In ALM, Forecast Rate assumptions are defined within the Forecast Rate Assumptions UI. The Forecast Rates Data Loader supports the Direct Input and Structured Change methods only for exchange rates, interest rates and economic indicators. Data for all other forecast rate methods should be input through the User Interface. After executing the forecast rates data loader, users can view the information in the ALM - Forecast Rates Assumptions UI.

Topics:

- [Forecast Rate Data Loader Tables](#)
- [Populating Forecast Rate Stage Tables](#)
- [Forecast Rate Loader Program](#)
- [Executing the Forecast Rate Data Load Procedure](#)
- [Exception Messages](#)

5.4.1 Forecast Rate Data Loader Tables

Forecast rate assumption data is stored in the following staging area tables:

- `STG_FCAST_XRATES`: This table holds the forecasted exchange rate data for the current ALM modeling period.

Note

For Direct Input Method, both `N_FROM_BUCKET` and `N_TO_BUCKET` column contain the same bucket number for the record in `STG_FCAST_XRATES` table.

- **STG_FCAST_IRCS**: This table holds the forecasted interest rate data for the current ALM modeling period.
- **STG_FCAST_EI**: This table holds the forecasted economic indicator data for the current ALM modeling period.

Rates present in the above staging tables are copied into the following ALM metadata tables.

- **FSI_FCAST_IRC_DIRECT_INPUT**, **FSI_FCAST_IRC_STRCT_CHG_VAL**.
- **FSI_FCAST_XRATE_DIRECT_INPUT**, **FSI_FCAST_XRATE_STRCT_CHG**.
- **FSI_FCAST_EI_DIRECT_INPUT**, **FSI_FCAST_EI_STRCT_CHG_VAL**

5.4.2 Populating Forecast Rate Stage Tables

- **STG_FCAST_EI**

Table 5-6 STG_FCAST_EI table

v_forecast_name	The Name of the Forecast Rate assumption rule as defined. The Forecast name indicates the Short Description for the Forecast Rate Sys ID as stored in the FSI_M_OBJECT_DEFINITION_TL table. In case the forecast sys id is provided, then populate this field with -1.
v_scenario_name	This field indicates the Scenario Name for which the Forecast Rate data is applicable.
v_economic_indicator_name	This field indicates the Economic Indicator Name for which the Forecast data is applicable.
n_from_bucket	This field indicates the Start Bucket Number for the given scenario.
fic_mis_date	This field indicates the current period As of Date applicable to the data being loaded.
n_fcast_rates_sys_id	The System Identifier of the forecast rate assumption rule to which this data will be loaded. In case forecast name and folder are provided, then populate this field with -1.
v_folder_name	Name of the folder that holds the Forecast Rate assumption rule definition. In case the forecast sys id is provided, then populate this field with -1.
v_ei_method_cd	The Forecast method of economic indicator values include: Direct Input or Structured change. Use DI - For Direct Input or SC - For Structured Change
n_economic_indicator_value	This field indicates the value for the Economic Indicator for the given scenario and time bucket.
n_to_bucket	This field indicates the End Bucket Number for the assumption.

- **STG_FCAST_XRATES**

Table 5-7 STG_FCAST_XRATES table

v_forecast_name	The Name of the Forecast Rate assumption rule as defined. The Forecast name indicates the Short Description for the Forecast Rate Sys ID as stored in the FSI_M_OBJECT_DEFINITION_TL table. In case the forecast sys id is provided, then populate this field with -1.
v_scenario_name	This field indicates the Scenario Name for which the Forecast Rate data is applicable.
v_iso_currency_cd	From ISO Currency Code (like USD, EUR, JPY, GBP) of the forecast rate.
n_from_bucket	This field indicates the Start Bucket Number for the given scenario.
fic_mis_date	This field indicates the As of Date for which the data being loaded is applicable.
n_fcast_rates_sys_id	The System Identifier of the assumption rule to which this data will be loaded. In case forecast name and folder are provided, then populate this field with -1.
v_folder_name	Name of the folder that holds the Forecast Rate assumption rule definition. In case the forecast sys id is provided, then populate this field with -1.
n_to_bucket	This field indicates the End Bucket Number for the given scenario.
v_xrate_method_cd	The Forecast method for exchange rate values include: Direct Input or Structured change. Use DI - For Direct Input or SC - For Structured Change
n_exchange_rate	This field indicates the Exchange rate for the Currency and given bucket Range. The value in N_EXCHANGE_RATE should be the rate used to convert 1 unit of the V_TO_CURRENCY_CD currency to the currency stored in V_FROM_CURRENCY_CD. For example, if V_TO_CURRENCY_CD = 'USD', then enter the exchange rate to convert 1 unit USD to another currency.

- STG_FCAST_IRCS

Table 5-8 STG_FCAST_IRCS table

v_forecast_name	The Name of the Forecast Rate assumption rule as defined. The Forecast name indicates the Short Description for the Forecast Rate Sys ID as stored in the FSI_M_OBJECT_DEFINITION_TL table. In case the forecast sys id is provided, then populate this field with -1.
v_scenario_name	This field indicates the Scenario Name for which the Forecast Rate data is applicable.
v_irc_name	The IRC Name indicates the Name of Interest Rate Code .
n_interest_rate_term	This field indicates the Interest Rate Term applicable for the row of data.

Table 5-8 (Cont.) STG_FCAST_IRCS table

v_interest_rate_term_mult	This field indicates the Interest Rate Term Multiplier for the row of data being loaded.
n_from_bucket	This field indicates the Start Bucket Number for the given scenario.
fic_mis_date	This field indicates the As of Date for which the data being loaded is applicable.
n_fcast_rates_sys_id	The System Identifier of the interest rate code forecast rate definition. In case the forecast name and folder are provided, then populate this field with -1.
v_folder_name	Name of the folder that holds the Forecast Rate assumption rule definition. In case the forecast sys id is provided, then populate this field with -1.
n_interest_rate	This field indicates the Interest Rate Change for the specified Term and for the given scenario.
n_to_bucket	This field indicates the End Bucket Number for the given scenario.
v_irc_method_cd	The Forecast method of interest rate code values include: Direct Input or Structured change. Use DI - For Direct Input or SC - For Structured Change

5.4.3 Forecast Rate Loader Program

The Forecast Rate Loader program updates the existing forecast rates to new forecast rates in the ALM Forecast Rate tables for Direct Input and Structured Change forecasting methods.

Note

The Forecast Rate Loader can only update existing forecast rate assumption rule definitions. The initial Forecast Rate assumption rule definition and initial methods must be created through the Forecast Rates user interface within Oracle ALM.

The Forecast Rates Data Loader performs the following functions:

The User can load forecast rate assumptions for either a specific Forecast Rate assumption rule or multiple forecast rates assumption rules.

- To Load a specific Forecast Rate assumption rule, the user should provide either the Forecast Rate name and a folder name as defined in Oracle ALM or the Forecast Rate System Identifier.
- When the load parameter is to load a specific Forecast Rate assumption rule for a given As of Date, the loader checks for Forecast Name/Forecast Rate System Identifier's presence in the Object Definition Registration Table. If it's present, then the combination of Forecast Name/Forecast Rate system Identifier and As of Date is checked in each of the Forecast Rate Staging Tables one by one.
- The data loading is done from each of the staging tables for the Direct Input and Structured change methods where the Forecast Name and As of Date combination is present.
- When the load parameter is the Load All Option (Y), the Distinct Forecast Name from the 3 staging tables is verified for its presence in Object Definition Registration table and the loading is done for each of the Forecast Names.

- Messages for each of the steps is written into the FSI_MESSAGE_LOG table.

After the Forecast rate loader processing is completed, the user should query the ALM Forecast Rate tables to look for the new forecast rates. Also, the user can verify the data just loaded using the Forecast Rate Assumption UI.

5.4.4 Executing the Forecast Rate Data Load

You can launch the Forecast Data Loader from the following:

- Forecast Rates Summary page
- PL/SQL block
- Operations Batch

5.4.4.1 Forecast Rate Loader – Method 1

You can execute the Forecast Rate Loader two ways. First method is described below:

1. To run the Forecast Rate Loader from SQL*Plus, login to SQL*Plus as the Schema Owner.

The procedure requires six parameters Batch Execution Identifier (batch_run_id)

As of Date (mis_date)

Forecast Rate System Identifier (pObject_Definition_ID)

Option for Loading All or any Specific Forecast Rate assumption rule. If the Load All option is 'N' then either the Forecast Rate Assumption rule Name Parameter with the Folder Name or Forecast Rate Sys ID should be provided else it raises an error (pLoad_all)

Forecast Rate assumption rule Name (pForecast_name)

Folder name (pFolder_Name)

The syntax for calling the procedure is:

```
fn_stg_forecast_rate_loader(batch_run_id varchar2, mis_date varchar2,
pObject_Definition_ID number, pLoad_all char default 'N', pForecast_name
varchar2, pFolder_Name varchar2 p_user_id varchar2 p_appid varchar2)
```

where

BATCH_RUN_ID is any string to identify the executed batch.

mis_date in the format YYYYMMDD.

pObject_Definition_ID -The Forecast Rate System Identifier in ALM

pLoad_all indicates option for loading all forecast rates.

pForecast_Name. This can be null i.e " when the pLoad_all is 'Y' else provide a valid Forecast Rate assumption rule Name.

pFolder_Name indicates the name of the Folder where the forecast rate assumption rule was defined.

p_user_id indicates the user mapped with the application in **rev_app_user_preferences**. This will be used to fetch as of date from **rev_app_user_preferences**. This is a mandatory parameter.

p_appid is the application name. This is a mandatory parameter.

For Example, if the user wants to Load all forecast rates assumption rules defined within a folder, say RTSEG then

```
Declare num number;BeginNum:=
fn_stg_forecast_rate_loader('INFODOM_FORECAST_RATE_LOADER','20100419',null,'Y'
,Null,'RTSEG','ALMUSER1','ALM');End;
```

The loading is done for all forecast rates under folder 'RTSEG' for as of Date 20100419.

Sample Data for STG_FCAST_IRCS to Load all forecast rates defined within a folder

Sample Data for STG_FCAST_XRATES to Load all forecast rates defined within a folder

Sample Data for STG_FCAST_EI to Load all forecast rates defined within a folder

2. If the user wants to Load a specific forecast rate assumption rule, then provide the unique Forecast Rate System Identifier.

```
Declare num number;BeginNum:=
fn_stg_forecast_rate_loader('INFODOM_FORECAST_RATE_LOADER','20100419',10005,'N'
,Null,Null,'ALMUSER1','ALM');End;
```

Sample Data for STG_FCAST_IRCS to load data for specific Forecast Rate providing the Forecast Rate System Identifier

Sample Data for STG_FCAST_XRATES to load data for specific Forecast Rate providing the Forecast Rate System Identifier

Sample Data for STG_FCAST_EI to load data for specific Forecast Rate providing the Forecast Rate System Identifier

Note

To Load data for specific Forecast Rate providing the Forecast Rate System Identifier, the value of Forecast rate Name and Folder Name in the staging tables should be -1.

3. If the user wants to Load a specific forecast rate assumption rule within the Folder providing the name of Forecast Rate as defined in ALM.

```
Declare num number;BeginNum:=
fn_stg_forecast_rate_loader('INFODOM_FORECAST_RATE_LOADER','20100419',Null,'N'
,'LOADER_TEST','RTSEG','ALMUSER1','ALM');End;
```

Sample Data for STG_FCAST_IRCS to Load a specific forecast rate within the Folder providing the name of Forecast Rate as defined in ALM

Sample Data for STG_FCAST_XRATES to Load a specific forecast rate within the Folder providing the name of Forecast Rate as defined in ALM

Sample Data for STG_FCAST_EI to Load a specific forecast rate within the Folder providing the name of Forecast Rate as defined in ALM

If the NUM value is 1, it indicates the load completed successfully, check the FSI_MESSAGE_LOG for more details.

5.4.4.2 Forecast Rate Loader – Method 2

There are two methods to execute Forecast Rate Loader.

1. To execute Forecast Rate Loader from OFSAI Batch Maintenance, a seeded Batch is provided. <INFODOM>_FORECAST_RATE_LOADER is the Batch ID and Forecast Rate Loader is the description of the batch.
2. The batch has a single task. Edit the task.

3. If the user intends to load data for all Forecast Rates under a Folder, then provide the batch parameters as shown.
 - **Datastore Type:** Select the appropriate datastore from list
 - **Datastore Name:** Select the appropriate name from the list
 - **IP address:** Select the IP address from the list
 - **Rule Name:** Forecast_Rate_loader
 - **Datastore Type:** Select the appropriate datastore from list
 - **Datastore Name:** Select the appropriate name from the list
 - **IP address:** Select the IP address from the list
 - **Rule Name:** Forecast_Rate_loader

Sample Data for STG_FCAST_IRCS to Load all forecast rates defined within a folder

Sample Data for STG_FCAST_XRATES to Load all forecast rates defined within a folder

Sample Data for STG_FCAST_EI to Load all forecast rates defined within a folder

4. If the user wants to load data for a specific Forecast Rate assumption rule, provide the Forecast Rate System Identifier, then define the batch parameters.
 - **Datastore Type:** Select the appropriate datastore from list
 - **Datastore Name:** Select the appropriate name from the list
 - **IP address:** Select the IP address from the list
 - **Rule Name:** Forecast_Rate_loader

Sample Data for STG_FCAST_IRCS to load data for a specific Forecast Rate assumption rule, with the Forecast Rate System Identifier already provided

Sample Data for STG_FCAST_XRATES to load data for a specific Forecast Rate assumption rule with the Forecast Rate System Identifier already provided

Sample Data for STG_FCAST_EI to load data for a specific Forecast Rate assumption rule with the Forecast Rate System Identifier already provided

Note

To Load data for specific Forecast Rate assumption rules, provide the Forecast Rate System Identifier and the value of Forecast rate Name and Folder Name in the staging tables should be -1.

5. If the user wants to load data for specific Forecast Rate assumption rules, provide the Forecast Rate Name as defined in ALM, then define the batch parameters as shown.
 - **Datastore Type:** Select an appropriate datastore from list
 - **Datastore Name:** Select an appropriate name from the list
 - **IP address:** Select the IP address from the list
 - **Rule Name:** Forecast_Rate_loader

Sample Data for STG_FCAST_IRCS to Load a specific forecast rate assumption rule, within the Folder, provide the name of Forecast Rate rule as defined in ALM

Sample Data for STG_FCAST_XRATES to Load a specific forecast rate assumption rule, within the Folder, provide the name of Forecast Rate rule as defined in ALM

Sample Data for `STG_FCAST_EI` to Load a specific forecast rate assumption rule, within the Folder, provide the name of Forecast Rate rule as defined in ALM

6. Save the Batch.
7. Execute the Batch for the required `As of Date`.

5.4.4.3 Forecast Rate Loader - Method 3

To execute Forecast Rate Loader from the Forecast Rates Summary page, follow these steps:

1. Click the Data Loader icon on the Forecast Rates Summary page. A warning message will appear: *Upload all available Forecast Rates?*
2. Click Yes. The process will load all valid data included in the staging table.

5.4.5 Exception Messages

The Forecast Rate Data Loader can have the following exceptions:

- Exception 1: Error. While fetching the Object Definition ID from Object Registration Table This exception occurs if the forecast rate assumption rule name is not present in the `FSI_M_OBJECT_DEFINITION_TL` table `short_desc` column.
- Exception 2: Error. More than one Forecast Sys ID is present. This exception occurs when there is more than one Forecast Sys ID present for the given forecast rate assumption rule name.
- Exception 3: Error. Forecast Rate assumption rule Name and As of Date combination do not exist in the Staging Table. This exception occurs when the Forecast Rate assumption rule Name and as of date combination do not exist in the Staging Table.

5.5 Prepayment Rate Data Loader

Prepayment Model assumptions are defined within the Prepayment Model rule User Interfaces in OFSAA ALM, HM and FTP applications. You can input prepayment rates directly through the UI, or import the rates from Excel into the UI. You can also use the Prepayment Rate Data Loader procedure to populate Prepayment Model rates into the OFSAA metadata table from the corresponding staging table. This data loader program can be used to update the Prepayment Model rates on a periodic basis. After loading the prepayment rates, you can view the latest data in the Prepayment Model assumptions UI.

Topics:

- [Prepayment Rate Loader Tables](#)
- [Prepayment Rate Data Loader](#)
- [Executing the Prepayment Model Data Loader](#)
- [Exception Messages](#)

5.5.1 Prepayment Rate Loader Tables

The Loader uses the following staging and target tables:

- `STG_PPMT_MODEL_HYPERCUBE`: This staging table contains prepayment rates for the selected prepayment dimensions.

- **FSI_PPMT_MODEL_HYPERCUBE**: The loader copies rates into this target table for the associated Prepayment Dimension combinations present in the **FSI_M_PPMT_MODEL** table.

5.5.2 Prepayment Rate Data Loader

The Prepayment Rate Data Loader program populates the target OFSAA Prepayment Model table with the values from the staging table. The procedure will load prepayment rate data for a specified Prepayment Model rule or all Prepayment models that are present in the staging table. The program assumes that the Prepayment Model assumptions have already been defined using OFSAA Prepayment Model rule UIs before loading Prepayment Model rates.

The program performs the following functions:

- The program performs certain checks to determine if:

The Prepayment Model dimensions present in staging are the same as those present in the OFSAA Prepayment Model metadata tables.

The bucket members of each of the dimensions present in staging are same as those present in the metadata tables.

The number of records present in the **STG_PPMT_MODEL_HYPERCUBE** table for a Prepayment Model is less than or equal to the maximum number of records that are allowed, which is determined by multiplying the number of buckets per dimension of the Prepayment Model.

PPMT_MDL_SYS_ID	DIMENSION_ID	NUMBER_OF_BUCKETS
20100405	8	2
20100405	4	3

Then the maximum number of records = number of buckets of dimension 8 * number of buckets of dimension 4

That is, maximum number of records = 2 * 3

Therefore, maximum number of records = 6 records

Check is made by Prepayment Rate Data Loader whether the number of records present in **STG_PPMT_MODEL_HYPERCUBE** table for a Prepayment model 20100405 is less than or equal to 6 or not.

- If the above quality checks are satisfied, then the rates present in the Staging table are updated to the OFSAA prepayment model metadata table.
- Any error messages are logged in the **FSI_MESSAGE_LOG** table and can be viewed in OFSAAI Log Viewer UI.

After the Prepayment Rate loader is completed, you should query the **FSI_PPMT_MODEL_HYPERCUBE** table to look for the new rates. Also, you can verify the data using the Prepayment Model Assumption UI.

Populating the data into **STG_PPMT_MODEL_HYPERCUBE**

- **V_PPMT_MDL**: The Name of the Prepayment Model as stored in **FSI_M_OBJECT_DEFINITION_TL** table. If Prepayment Model name is given, also provide the Folder name. If the Prepayment Model System ID is provided, then populate this field with -1.
- **N_ORIG_TERM**: Original term of the contract
- **N_REPRICING_FREQ**: The number of months between instrument repricing

- `N_REM_TENOR`: Remaining term of the contract (in Months)
- `N_EXPIRED_TERM`: Expired term of the contract (in Months)
- `N_TERM_TO_REPRICE`: Repricing term of the contract (in Months)
- `N_COUPON_RATE`: The current gross rate on the instrument
- `N_MARKET_RATE`: Forecast rate representing alternate funding
- `N_RATE_DIFFERENCE`: Spread between the current gross rate and the market rate
- `N_RATE_RATIO`: Ratio of the current gross rate to the market rate
- `N_PPMT_RATE`: User defined prepayment rate for the associated dimension value combination
- `FIC_MIS_DATE`: The As of Date for which the data being loaded is applicable
- `V_FOLDER_NAME`: Name of the Folder which holds the Prepayment Model. If the Prepayment Model System ID is provided, then populate this field with -1. If Folder name is provided, then provide Prepayment Model name as well.
- `N_PPMT_MDL_SYS_ID`: The System Identifier (Object Definition ID) of the Prepayment Model to which this data will be loaded. If Prepayment Model name and Folder are provided, then populate this field with -1.

Column mapping from source to target

Source `STG_PPMT_MODEL_HYPERCUBE` to Target `FSI_PPMT_MODEL_HYPERCUBE` mapping:

`N_ORIG_TERM` -> `ORIGINAL_TERM`

`N_REPRICING_FREQ` -> `REPRICING_FREQ`

`N_REM_TENOR` -> `REMAINING_TERM`

`N_EXPIRED_TERM` -> `EXPIRED_TERM`

`N_TERM_TO_REPRICE` -> `TERM_TO_REPRICE`

`N_COUPON_RATE` -> `COUPON_RATE`

`N_MARKET_RATE` -> `MARKET_RATE`

`N_RATE_DIFFERENCE` -> `RATE_DIFFERENCE`

`N_RATE_RATIO` -> `RATE_RATIO`

`N_PPMT_RATE` -> `PPMT_RATE`

`N_PPMT_MDL_SYS_ID` -> `PPMT_MDL_SYS_ID` when `N_PPMT_MDL_SYS_ID` <> -1, otherwise it performs a lookup in `FSI_M_OBJECT_DEFINITION_TL` based on the Name and Folder provided in the staging table

Example

Based on the `FSI_M_PPMT_MODEL` table, for data in the staging table with Prepayment Model System ID 20100405:

`PPMT_MDL_SYS_ID` `DIMENSION_ID` `NUMBER_OF_BUCKETS`

20100405 8 2

20100405 4 3

The maximum number of records = (number of buckets of dimension 8) * (number of buckets of dimension 4).

That is, maximum number of records = 2 * 3

Therefore, maximum number of records = 6 records.

The Prepayment Rate Data Loader checks whether the number of records present in STG_PPMT_MODEL_HYPERCUBE table for Prepayment Model 20100405 is less than or equal to 6.

If the above quality checks are satisfied, then the rates present in the Staging table are updated to the OFSAA Prepayment Model metadata table.

Any error messages are logged in the FSI_MESSAGE_LOG table.

5.5.3 Executing the Prepayment Model Data Loader

You can launch the Data Loader from the following: Prepayment Models summary page, PL/SQL block, and Operations Batch

1. Prepayment Models summary page: To launch from the Prepayment Models summary page:
 - a. Click the Data Loader icon on the Prepayment Models summary grid toolbar. A warning message will appear: *Upload all available Prepayment Rates?*
 - b. Click Yes. The process will load all valid data included in the staging table.
2. PL/SQL block: To execute the Loader within a PL/SQL block:

- To run the function from SQL*Plus, log in to SQL*Plus as the Schema Owner. The loader requires following parameters:

Batch Execution Name

As Of Date

Object Definition ID

Load All

Model Name

Folder Name

Syntax:

```
fn_ppmt_rate_loader(batch_run_id VARCHAR2, mis_date VARCHAR2,
pObject_Definition_ID NUMBER, pLoad_all CHAR default 'N', pModel_name
VARCHAR2, pFolder_name VARCHAR2)
```

Where:

- BATCH_RUN_ID is any string to identify the executed batch.
- As_of_Date is the execution date in the format YYYYMMDD.
- Object Definition ID is the System Identifier of the Prepayment Model to which this data will be loaded.
- Select Load All if you want to load all the prepayment Models.
- Folder Name is the name of the Folder which holds the Prepayment Model. If the Prepayment Model System ID is provided, then populate this field with -1. If Folder name is provided, then provide Prepayment Model name as well.

For Example:

```

IF (P_OBJECT_DEFINITION_ID = -1) THEN
PRC_STG_PPMT_MODEL_SYSID(P_OBJECT_DEFINITION_ID, P_AS_OF_DATE); ELSE IF
P_LOAD_ALL = 'Y' THEN BEGIN SELECT DISTINCT V_PPMT_MDL, V_FOLDER_NAME BULK
COLLECT INTO V_MODEL_FOLDER_NAME FROM STG_PPMT_MODEL_HYPERCUBE WHERE
TO_CHAR(FIC_MIS_DATE, 'YYYYMMDD') = P_AS_OF_DATE AND N_PPMT_MDL_SYS_ID =
-1; SELECT DISTINCT N_PPMT_MDL_SYS_ID BULK COLLECT INTO
V_OBJECT_DEFINITION_ID FROM STG_PPMT_MODEL_HYPERCUBE WHERE
TO_CHAR(FIC_MIS_DATE, 'YYYYMMDD') = P_AS_OF_DATE AND V_PPMT_MDL = '-1' AND
V_FOLDER_NAME = '-1'; IF (V_MODEL_FOLDER_NAME.COUNT != 0) THEN FOR L_INDEX
IN V_MODEL_FOLDER_NAME.FIRST .. V_MODEL_FOLDER_NAME.LAST LOOP
PRC_STG_PPMT_MODEL_NAME(V_MODEL_FOLDER_NAME(L_INDEX).PPMT_MODEL_NAME,
V_MODEL_FOLDER_NAME(L_INDEX).OBJECT_DEFN_FOLDER_NAME, P_AS_OF_DATE); END
LOOP; END IF; IF (V_OBJECT_DEFINITION_ID.COUNT != 0) THEN FOR L_INDEX IN
V_OBJECT_DEFINITION_ID.FIRST .. V_OBJECT_DEFINITION_ID.LAST LOOP
PRC_STG_PPMT_MODEL_SYSID(V_OBJECT_DEFINITION_ID(L_INDEX), P_AS_OF_DATE);
END LOOP; END IF; ELSE SELECT COUNT(*) INTO V_COUNT_CHECK FROM
STG_PPMT_MODEL_HYPERCUBE WHERE V_PPMT_MDL = P_MODEL_NAME AND V_FOLDER_NAME
= P_FOLDER_NAME AND TO_CHAR(FIC_MIS_DATE, 'YYYYMMDD') = P_AS_OF_DATE; IF
V_COUNT_CHECK != 0 THEN PRC_STG_PPMT_MODEL_NAME(P_MODEL_NAME,
P_FOLDER_NAME, P_AS_OF_DATE);

```

The loader is executed for the given as of date. If the return value (NUM) is 1, this indicates the load completed successfully. Check the FSI_MESSAGE_LOG for more details.

3. Operations Batch: To run from Operations Batch framework:
 - a. You can create a new Batch with the Component = 'TRANSFORM DATA'
 - b. Specify the following parameters for the task:
 - Datstore Type: Select appropriate datastore from list
 - Datstore Name: Select appropriate name from the list
 - IP address: Select the IP address from the list
 - Rule Name: ppmt_rate_loader
 - Parameter List: None

4. View the results.

Any error messages are logged in the FSI_MESSAGE_LOG table. If you launch the Loader from the Prepayment Models Summary page or Operations Batch, you can view processing messages in OFSAAI in the Operations and select View Log UI, where the Component Type = Data Transformation and the Batch Run ID = the ID for your run. You can also spot check results of the load as follows: Query the FSI_PPMT_MODEL_HYPERCUBE table to confirm existence of the new rates. Use the Prepayment Model rule UI to select your rule and View your rates

5.5.4 Exception Messages

The Prepayment Model Rate Loader can have the following exceptions:

- **Exception 1:** Error while fetching the Object Definition ID from Object Definition Table. This exception occurs if the prepayment model name is not present in the FSI_M_OBJECT_DEFINITION_TL table.
- **Exception 2:** Error. More than one prepayment model sys ID is present for the given definition.

This exception occurs when there is more than one Prepayment Model System ID present for the Prepayment Model name in staging.

- **Exception 3:** Error. Data is present in additional dimension ID column than those defined in FSI_M_PPMT_MODEL.
This exception occurs if rates are specified in staging for the dimensions that are not part of the Prepayment Model definition.
- **Exception 4:** The value in the Dimension ID column is not matching with the value present in the corresponding column in metadata table.
This exception occurs if rates are specified in staging for the dimension members that are not part of the Prepayment Model definition.
- **Exception 5:** The number of records for the staging table for a given Prepayment Model Name is more than those calculated by multiplying the number of buckets in FSI_M_PPMT_MODEL table for the given model name.
This exception occurs if there are excess records in staging compared to OFSAA metadata tables for the given Prepayment Model.

5.6 Stage Instrument Table Loader

Data in staging instrument tables are moved into respective OFSAA processing instrument tables using OFSAAI T2T component. After loading the data, users can view the loaded data by querying the processing instrument tables.

Topics:

- [Stage Tables](#)
- [Populating Stage Tables](#)
- [Mapping To OFSAA Processing Tables](#)
- [Populating Accounts Dimension](#)
- [Executing T2T Data Movement Tasks](#)

5.6.1 Stage Tables

Following are examples of some of the various application staging instrument tables:

- STG_LOAN_CONTRACTS: holds contract information related to various loan products including mortgages.
- STG_TD_CONTRACTS: holds contract information related to term deposit products.
- STG_CASA: holds information related to Checking and Savings Accounts.
- STG_OD_ACCOUNTS: holds information related to over-draft accounts.
- STG_CARDS: holds information related to credit card accounts.
- STG_LEASES: holds contract information related to leasing products.
- STG_ANNUITY_CONTRACTS: holds contract information related to annuity contracts.
- STG_INVESTMENTS: holds information related to investment products like bond, equities, and so on.
- STG_MM_CONTRACTS: holds contract information related to short term investments in money market securities.
- STG_BORROWINGS: holds contract information related to various inter-bank borrowings.

- **STG_FX_CONTRACTS**: holds contract information related to FX products like FX Spot, FX Forward, and so on. Leg level details, if any, are stored in various leg-specific columns within the table.
- **STG_SWAPS_CONTRACTS**: holds contract information related to various types of swaps. Leg level details, if any, are stored in various leg-specific columns within the table.
- **STG_OPTION_CONTRACTS**: holds contract information related to various types of options. Leg level details, if any, are stored in various leg-specific columns within the table.
- **STG_FUTURES**: holds contract information related to interest rate forwards and all types of futures. Leg level details, if any, are stored in various leg-specific columns within the table.
- **STG_LOAN_COMMITMENTS**: contains all existing columns from **STG_LOAN_CONTRACTS**

Note

You can modify any existing instrument table to include the columns by adding the **COMMITMENT_CONTRACTS** super type. If you want to execute the Forward Rate transfer pricing pricing against tables in addition to **FSI_D_LOAN_COMMITMENTS**, then add the required columns and do so by adding the **COMMITMENT_CONTRACTS** super type via **ERWIN**.

5.6.2 Populating Stage Tables

Data can be loaded into staging tables through F2T component of OFSAI. After data is loaded, check for data quality within the staging tables, before moving into OFSAA processing tables. Data quality checks can include:

- Number of records between external system and staging instrument tables.
- Valid list of values in code columns of staging.
- Valid list of values in dimension columns like product, organization unit, general ledger, and so on. These members should be present in the respective dimension tables.
- Valid values for other significant columns of staging tables.

5.6.3 Mapping To OFSAA Processing Tables

Following are examples of some of the pre-defined application T2T mappings between the above staging tables and processing tables:

- **T2T_LOAN_CONTRACTS**: for loading data from **STG_LOAN_CONTRACTS** to **FSI_D_LOAN_CONTRACTS**.
- **T2T_MORTGAGES**: for loading data from **STG_LOAN_CONTRACTS** to **FSI_D_MORTGAGES**.
- **T2T_CASA**: for loading data from **STG_CASA** to **FSI_D_CASA**.
- **T2T_CARDS**: for loading data from **STG_CARDS** to **FSI_D_CREDIT_CARDS**.
- **T2T_TD_CONTRACTS**: for loading data from **STG_TD_CONTRACTS** to **FSI_D_TERM_DEPOSITS**.
- **T2T_ANNUITY_CONTRACTS**: for loading data from **STG_ANNUITY_CONTRACTS** to **FSI_D_ANNUITY_CONTRACTS**.
- **T2T_BORROWINGS**: for loading data from **STG_BORROWINGS** to **FSI_D_BORROWINGS**.
- **T2T_FORWARD_CONTRACTS**: for loading data from **STG_FUTURES** to **FSI_D_FORWARD_RATE_AGMTS**.

- T2T_FUTURE_CONTRACTS: for loading data from STG_FUTURES to FSI_D_FUTURES.
- T2T_FX_CONTRACTS: for loading data from STG_FX_CONTRACTS to FSI_D_FX_CONTRACTS.
- T2T_INVESTMENTS: for loading data from STG_INVESTMENTS to FSI_D_INVESTMENTS.
- T2T_LEASES_CONTRACTS: for loading data from STG_LEASES_CONTRACTS to FSI_D_LEASES.
- T2T_MM_CONTRACTS: for loading data from STG_MM_CONTRACTS table to FSI_D_MM_CONTRACTS.
- T2T_OPTION_CONTRACTS: for loading data from STG_OPTION_CONTRACTS to FSI_D_OPTION_CONTRACTS.
- T2T_SWAP_CONTRACTS: for loading data from STG_SWAPS_CONTRACTS to FSI_D_SWAPS.
- T2T_OD_ACCOUNTS: for loading data from STG_OD_ACCOUNTS to FSI_D_CREDIT_LINES.
- T2T_LOAN_COMMITMENTS: for loading data from STG_LOAN_COMMITMENTS to FSI_D_LOAN_COMMITMENTS

COLUMN	SOURCE/OUTPUT COLUMN	DESCRIPTION
COMMIT_START_DATE	From source – Mandatory for Rate	The date on which the Rate Lock period starts
COMMIT_MAT_DATE	From source – Mandatory for Rate	The date on which the Rate Lock period expires. Corresponds to the Loan Origination Date
COMMIT_TERM	From source – Mandatory for Rate	The Rate Lock term period. Equal to COMMIT_MAT_DATE – COMMIT_START_DATE
COMMIT_TERM_MULT	From source – Mandatory for Rate	The Rate Lock term multiplier
COMMIT_FEE_TO_CUST	From source – Optional	The fee that is charged to the customer for the Rate Lock agreement.
COMMIT_OPTION_COST_PCT	Calculated – Output column	The Rate Lock cost expressed as a percentage
COMMIT_OPTION_COST	Calculated – Output column	The calculated Rate Lock cost charged by treasury to the banker – Rate Lock % * Balance
COMMIT_OPTION_TYPE_CD	From source - Mandatory	Refers to Option Type (default = 1). At present, we support only European. If Option Type is European, then compute Rate Lock option cost using the Black Swaption formula. Otherwise do nothing.

Note

FSI_D_LOAN_COMMITMENTS - Contains all existing columns from FSI_D_LOAN_CONTRACTS, plus the following columns which support the calculations

You can view the Database Extract definitions by performing the following steps:

Note

The Data Management Tools and Data Ingestion were previously known as Data Integrator Framework and Warehouse Designer respectively. These new terminologies are applicable only for OFSAAI versions 7.3.2.3.0 and above.

1. Navigate to Unified Metadata Manager and select Data Management Tools, and then select Data Ingestion, and then select Database Extracts section.
2. In the left pane, expand the application as defined during application installation and click the Data Source defined during application installation.
3. Expand the required T2T definition to view the Database Extract definitions.

You can view the Source - Target mapping definitions by performing the following steps:

Note

The Data Management Tools and Data Ingestion were previously known as Data Integrator Framework and Warehouse Designer respectively. These new terminologies are applicable only for OFSAAI versions 7.3.2.3.0 and above.

1. Navigate to Unified Metadata Manager and select Data Management, and then select Data Ingestion, and then select Database Extracts.
2. In the left pane, expand the application as defined during application installation and click the Data Source defined during application installation.
3. Expand the required T2T definition to view the extract definition.
4. Click the required Database Extract definition.

The selected Database Extract definition details are displayed with the available Source - Target mappings under the Source - Target Mappings grid.

Note

Staging instrument tables contain alphanumeric display codes for various IDENTIFIER and CODE columns. T2T mapping looks up in respective dimension tables for fetching an equivalent numeric ID and CODE corresponding to the alphanumeric display code. Hence, these dimension tables should be populated with the alphanumeric display code before executing any data movement tasks.

5.6.4 Populating Accounts Dimension

Account Number is an alphanumeric unique identifier within each staging instrument tables. ID_NUMBER is a numeric unique identifier within processing instrument tables. Hence, there is a need to generate a numeric surrogate key for each of the account number. This information is stored in DIM_ACCOUNT table.

You can populate DIM ACCOUNT table using SCD. For more information on Slowly Changing Dimension (SCD) to populate into DIM ACCOUNT, see Document ID [1273210.1](#) (How to Populate the DIM_ACCOUNT Table in OFSAA).

5.6.5 Executing T2T Data Movement Tasks

Before executing T2T data movement tasks, user should ensure that all the dimension tables that are required for instruments data are loaded.

1. The following are some of the mandatory dimensions:

```
DIM_ACCOUNTS
DIM_PRODUCTS_B
DIM_GENERAL_LEDGER_B
DIM_COMMON_COA_B
DIM_ORG_UNIT_B
```

2. Create a new Batch with the Task
3. Specify the following parameters for the task for loading Historical Exchange Rates:

Datastore Type: Select appropriate datastore from the drop down list.

Datastore Name: Select appropriate name from the list. Generally it is the Infodom name.

IP address: Select the IP address from the list.

Rule Name: Select the appropriate T2T name from the above list.

Parameter List: No Parameter is passed. The only parameter is the As of Date Selection while execution.

4. Check T2T component logs and batch messages for checking the status of load.

T2T component can fail because of following cases: Unique constraint error:

Target table may already contain the primary keys that are part of the staging tables.

NOT NULL constraint error: do not have values for NOT NULL columns in the target table.

5.6.6 Re-Load Of Instrument Data

T2T component can only perform Insert operations. In case user needs to perform updates, previously loaded records should be deleted before loading the current records.

Function `fn_deleteFusionTables` is used for deleting the records in the target that are present in the source. This function removes rows in the table if there are matching rows in the Stage table. This function needs `FSI_DELETE_TABLES_SETUP` to be configured. Configure the following table for all columns that need to be part of the join between Stage table and Equivalent table.

Create a new Batch with the Task and specify the following parameters for the task to delete existing records:

- **Datastore Type:** Select appropriate datastore from the drop down list.
- **Datastore Name:** Select appropriate name from the list. Generally it is the Infodom name.
- **IP address:** Select the IP address from the list.
- **Rule Name:** `fn_deleteFusionTables`
- **Parameter List:** 'Table to be deleted'

Batch run ID and As Of Date are passed internally by the batch to the Data Transformation task.

Sample record for FSI_DELETE_TABLES_SETUP table is given following:

Column Name	Column Description	Sample Value
STAGE_TABLE_NAME	Stores the source table name for forming the join statement	STG_LOAN_CONTRACTS
STAGE_COLUMN_NAME	Stores the source column name for forming the join statement	V_ACCOUNT_NUMBER
FUSION_TABLE_NAME	Stores the target table name for forming the join statement	FSI_D_LOAN_CONTRACTS
FUSION_COLUMN_NAME	Stores the target column name for forming the join statement	ACCOUNT_NUMBER

Note

Insert rows in FSI_DELETE_TABLES_SETUP for all columns that can be used to join the stage with the equivalent table. In case if the join requires other dimension or code tables, a view can be created joining the source table with the respective code tables and that view can be part of the above setup table.

5.7 Customer T2T Loading

Data in Transaction Summary table is moved into customer table using OFSAAI T2T component. After loading the data, users can view the loaded data by querying the customer table.

Topics:

- [Dependencies](#)
- [Flow Diagram for Customer T2T](#)
- [Executing T2T Data Movement Task](#)

5.7.1 Dependencies

- DIM_Account: DIM account will be populated using the SCDs/ FN_POPDIMACCOUNT. These 30 tasks represent to the 30 SCD processes where different product processors would be the source, and DIM_ACCOUNT would be the target. Run MAP_REF_NUMs 188 to 217. For more information, see the [Populating Accounts Dimension section](#).
- DIM_<XXXX>_B/TL/HIER/ATTR population: Run DT FN_DRMDATALOADER/ FN_STGDIMDATALOADER to populate B/TL/HIER/ATTR tables of the key dimensions such as Product, Common_COA, GL, Org Unit, Customer, and so on.
- DIM_Party Population: Run the SCD MAP_REF_NUM 168. Dim_PARTY loads party_skey and other customer information. STG_PARTY_MASTER is the source table for Dim_party. Dim_party join to FSI_D_<xxx> table through DIM_CUSTOMER_B. The data flow as DIM_CUSTOMER_B.customer_code = dim_party.v_party_id and dim_party.f_latest_record_indicator = 'Y' and dim_customer_b.customer_id = fsi_d_mortgages.customer_id.
- Run all the Stage to Processing layer T2Ts. This will populate all the processing area tables such as, FI_D_<xxxx>. Example: Run T2T_ANNUITY_CONTRACTS, T2T_MORTGAGES, T2T_MM_CONTRACTS and so on. For more information, see the Stage Instrument Table Loader section.

- Run the Processing to FSI_D_INST_SUMMARY T2Ts.
Example: T2T_INS_SUMM_ANNUITY, T2T_INS_SUMM_BORROWINGS, T2T_INS_SUMM_CASA and so on. For more information, sStage Instrument Table Loader section.

If LRM is integrated with FTP, then perform following steps prior to run actual T2T to populate FSI_D_CUSTOMER T2T

- Dim_Run: Dim_Run table that holds RUN_SKEY and RUN_ID. Finalize the run to pick the right RUN_SKEY mapped to FSI_LRM_ACCT_CUST_DETAILS table.
- FSI_LRM_ACCT_CUST_DETAILS: Populate this table using the T2T
- SETUP_PARAMETERS_MASTER: Manually update this table with finalized RUN_ID in the column Param_value column. Below table shows the default values. PARAM_VALUE should be manually updated. Finalize RUN_ID as in DIM_RUN. T2T expect PARAM_APP_ID to 'FTP' and PARAM_NAME to RUN_ID_FOR_CUSTOMER_T2T_FROM_LRM_TO_FTP. To pick the records from LRM, following row needs to be updated with finalized run_id in PARAM_VALUE. Without this step completion, no rows will be picked from LRM.

PARAM_SEQ	PARAM_APP_ID	PARAM_NAME	PARAM_VALUE
1	FTP	RUN_ID_FOR_CUSTO MER_T2T_FROM_LRM _TO_FTP	-1

Query to find the finalized run id from DIM_RUN:

```
select * from dim_run r where r.f_reporting_flag = 'Y'
```

5.7.2 Flow Diagram for Customer T2T

Instrument summary list of T2Ts

FSI_D_INST_SUMMARY gets loaded with the following T2Ts:

- T2T_INS_SUMM_ANNUITY
- T2T_INS_SUMM_BORROWINGS
- T2T_INS_SUMM_BREAK_FUND_CHG
- T2T_INS_SUMM_CASA
- T2T_INS_SUMM_CREDIT_CARDS
- T2T_INS_SUMM_CREDIT_LINES
- T2T_INS_SUMM_GUARANTEES
- T2T_INS_SUMM_INVESTMENTS
- T2T_INS_SUMM_LDGR_STAT_INST
- T2T_INS_SUMM_LEASES
- T2T_INS_SUMM_LOANS
- T2T_INS_SUMM_MERCHANT_CARDS
- T2T_INS_SUMM_MM_CONTRACTS
- T2T_INS_SUMM_MORTGAGES
- T2T_INS_SUMM_MUTUAL_FUNDS
- T2T_INS_SUMM_OTHER_SERVICES

- T2T_INS_SUMM_RETIREMENT_ACC
- T2T_INS_SUMM_TERM_DEPOSITS
- T2T_INS_SUMM_TRUSTS

5.7.3 Executing T2T Data Movement Task

To execute T2T Data Movement Task, follow these steps:

1. Create a new Batch with the Task
2. Specify the following parameters

Datastore Type: Select appropriate datastore from the drop down list.

Datastore Name: Select appropriate name from the list. Generally it is the Infodom name.

IP address: Select the IP address from the list.

Rule Name: T2T_FSI_D_CUSTOMER

Parameter List: No Parameter is passed. The only parameter is the As of Date Selection while execution.

Note

Take care of following points if LRM is installed:

To pick data from FSI_LRM_ACCT_CUST_DETAILS, RUN_ID needs to be finalized.
That is in DIM_RUN f_reporting_flag ='Y'

Manually update the finalized RUN_ID into setup_parameters_master table (for example -1 is the RUN_ID which needs to be manually overwritten).

Incase RUN_ID is not finalized that is f_reporting_flag not = 'Y' then no data will be picked from FSI_LRM_ACCT_CUST_DETAILS for the given as_of_date.

5.8 DIM_Party Population

DIM_party holds customer level details. STG_PARTY_MASTER is the source table for Dim_party. Party_key uniquely identifies the customer records.

5.8.1 Execution from ICC Batch

To execution DIM_Party population from ICC Batch, follow these steps:

1. Create a new Batch with the Task
2. Specify the following parameters for the task for loading DIM_Party:

Component: Run Executable

Datastore Type: Select appropriate datastore from list

Datastore Name: Select appropriate name from the list

IP address: Select the IP address from the list

Executable: scd,168

Wait: N

Batch Parameter: Y

Note

Instrument tables to instrument Summary T2T pulls the customer related information from Dim party. So, populating Dim party should be done prior to T2T run from Instrument to Instrument Summary T2T execution.

5.9 Instrument Summary Table

The following topics are included in this section:

- [Mapping To OFSAA Summary Table](#)
- [Dependencies](#)
- [Executing T2T Data Movement Tasks](#)
- [Re-Load Of Instrument Summary Data](#)

5.9.1 Mapping To OFSAA Summary Table

The following are the pre-defined T2T mappings between FSI_D_< XXX > tables to FSI_D_INST_SUMMARY tables:

- T2T_INS_SUMM_ANNUITY
- T2T_INS_SUMM_BORROWINGS
- T2T_INS_SUMM_BREAK_FUND_CHG
- T2T_INS_SUMM_CASA
- T2T_INS_SUMM_CREDIT_CARDS
- T2T_INS_SUMM_CREDIT_LINES
- T2T_INS_SUMM_GUARANTEES
- T2T_INS_SUMM_INVESTMENTS
- T2T_INS_SUMM_LDGR_STAT_INST
- T2T_INS_SUMM_LEASES
- T2T_INS_SUMM_LOANS
- T2T_INS_SUMM_MERCHANT_CARDS
- T2T_INS_SUMM_MM_CONTRACTS
- T2T_INS_SUMM_MORTGAGES
- T2T_INS_SUMM_MRTGAGE_BCK_SC
- T2T_INS_SUMM_MUTUAL_FUNDS
- T2T_INS_SUMM_OTHER_SERVICES
- T2T_INS_SUMM_RETIREMENT_ACC
- T2T_INS_SUMM_TERM_DEPOSITS
- T2T_INS_SUMM_TRUSTS

User can view the extract definitions by going through the following steps:

1. Go to Data Integrator and select Source Designer, and then select Define Extracts.
2. Under FUSION_APPSApplication, click Data Source name.
3. Click on any of the T2T definition to view the extract definition. User can view the mapping definitions by going through the following steps:
4. Go to Data Integrator and select Warehouse Designer, and then select Define Mapping.
5. Under FUSION_APPS application, click Data Source Name.
6. Click on any of the T2T definition to view the mapping definition.

5.9.2 Dependencies

- Instrument tables should be loaded before loading the Instrument summary Information related to those instruments.
- Account Number is an alphanumeric unique identifier within each staging tables. `ID_NUMBER` is a numeric unique identifier within processing Instrument tables. Hence, there is a need to look up into a `DIM_ACCOUNT` dimension table for a numeric surrogate key for each of the alphanumeric account number. This dimension table `DIM_ACCOUNT` will be populated as part of the process that loads instrument tables. For more information on loading instrument tables, see Loading Instrument Table Data
- Before executing T2T data movement tasks, user should ensure that all the dimension tables that are required for instruments data are loaded. The following are some of the mandatory dimensions:
 - `DIM_ACCOUNTS`
 - `DIM_PRODUCTS_B`
 - `DIM_GENERAL_LEDGER_B`
 - `DIM_COMMON_COA_B`
 - `DIM_ORG_UNIT_B`

5.9.3 Executing T2T Data Movement Tasks

To execute T2T Data Movement Tasks, follow these steps:

1. Create a new Batch with the Task
2. Specify the following parameters for the task for Loading Instrument Summary table:

Datastore Type: Select appropriate datastore from the drop down list.

Datastore Name: Select appropriate name from the list. Generally it is the Infodom name.

IP address: Select the IP address from the list.

Rule Name: Select the appropriate T2T name from the above list.

Parameter List: No Parameter is passed. The only parameter is the As of Date Selection while execution.

Check T2T component logs and batch messages for checking the status of load. T2T component can fail because of following cases:

- **Unique constraint error:** Target table may already contain the primary keys that are part of the staging tables.
- **NOT NULL constraint error:** Staging table do not have values for mandatory columns of the target table.

5.9.4 Re-Load Of Instrument Summary Data

T2T component can only perform Insert operations. In case user needs to perform updates, previously loaded records should be deleted before loading the current records.

Function `fn_deleteFusionTables` is used for deleting the records in the target that are present in the source. This function removes rows in the table if there are matching rows in the Stage table. This function needs `FSI_DELETE_TABLES_SETUP` to be configured. Configure the following table for all columns that need to be part of the join between Stage table and Equivalent table.

Create a new Batch with the Task and specify the following parameters for the task to delete existing records:

- Datastore Type: Select appropriate datastore from the drop down list.
- Datastore Name: Select appropriate name from the list. Generally it is the Infodomain name.
- IP address: Select the IP address from the list.
- Rule Name: `fn_deleteFusionTables`
- Parameter List: Table to be deleted

Batch run ID and As Of Date are passed internally by the batch to the Data Transformation task.

5.10 Transaction Summary Table Loader

Data in staging Transaction Summary tables are moved into respective OFSAA processing transaction summary tables using OFSAAI T2T component. After loading the data, users can view the loaded data by querying the processing transaction tables.

Topics:

- [Stage Tables](#)
- [Populating Stage Tables](#)
- [Mapping To OFSAA Processing Tables](#)
- [Dependencies](#)
- [Executing T2T Data Movement Tasks](#)
- [Re-Load Of Transaction Summary Data](#)

5.10.1 Stage Tables

Following are examples of various application staging transaction summary tables:

- `STG_LOAN_CONTRACT_TXNS_SUMMARY`: holds transaction summary information related to the loan contracts that are present in staging instrument table for loan contracts, that is `STG_LOAN_CONTRACTS`.
- `STG_CARDS_TXNS_SUMMARY`: holds transaction summary information related to the credit cards present that are present in staging instrument table for credit cards, that is `STG_CARDS`.

- **STG_CASA_TXNS_SUMMARY:** holds transaction summary information related to the checking and saving accounts that are present in staging instrument table for CASA, that is STG_CASA.
- **STG_MERCHANT_CARD_TXNS_SUMMARY:** holds transaction summary information related to the merchant cards that are present in staging instrument table for merchant cards, that is STG_MERCHANT_CARDS.
- **STG_OTHER_SERVICE_TXNS_SUMMARY:** holds transaction summary information related to other services that are present in staging instrument table for other services, that is STG_OTHER_SERVICES.
- **STG_TERMDEPOSITS_TXNS_SUMMARY:** holds transaction summary information related to the term deposits that are present in staging instrument table for term deposits, that is STG_TD_CONTRACTS.
- **STG_TRUSTS_TXNS_SUMMARY:** holds transaction summary information related to the trust accounts that are present in staging instrument table for trusts, that is STG_TRUSTS.

Note

These tables are required for PFT application and used in the Allocation definitions.

5.10.2 Populating Stage Tables

Data can be loaded into staging tables through F2T component of OFSAAI. After data is loaded, check for data quality within the staging tables, before moving into OFSAA processing tables. Data quality checks can include:

- Number of records between external system and staging transaction summary tables.
- Valid list of values in code columns of staging.
- Valid list of values in dimension columns like product, organization unit, general ledger, and so on. These members should be present in the respective dimension tables.
- Valid list of values in dimension columns like product, organization unit, general ledger, and so on. These members should be present in the respective dimension tables.
- Valid values for other significant columns of staging tables.

5.10.3 Mapping To OFSAA Processing Tables

Following are examples of the pre-defined T2T mappings between the above application staging tables and processing tables:

- **T2T_STG_CARDS_TXNS_SUMMARY:** for loading data from STG_CARDS_TXNS_SUMMARY to FSI_D_CREDIT_CARDS_TXNS.
- **T2T_STG_CASA_TXNS_SUMMARY:** for loading data from STG_CASA_TXNS_SUMMARY to FSI_D_CASA_TXNS.
- **T2T_LOAN_CONTRACT_TXNS_SUMMARY:** for loading data from STG_LOAN_CONTRACT_TXNS_SUMMARY to FSI_D_LOAN_CONTRACTS_TXNS.
- **T2T_STG_MERCHANT_CARD_TXNS_SUMMARY:** for loading data from STG_MERCHANT_CARD_TXNS_SUMMARY to FSI_D_MERCHANT_CARDS_TXNS.

- **T2T_STG_OTHER_SERVICE_TXNS_SUMMARY**: for loading data from **STG_OTHER_SERVICE_TXNS_SUMMARY** to **FSI_D_OTHER_SERVICES_TXNS**.
- **T2T_STG_TERMDEPOSITS_TXNS_SUMMARY**: for loading data from **STG_TERMDEPOSITS_TXNS_SUMMARY** to **FSI_D_TERM_DEPOSITS_TXNS**.
- **T2T_STG_TRUSTS_TXNS_SUMMARY**: for loading data from **STG_TRUSTS_TXNS_SUMMARY** to **FSI_D_TRUSTS_TXNS**.

You can view the Database Extract definitions by performing the following steps:

Note

The Data Management Tools and Data Ingestion were previously known as Data Integrator Framework and Warehouse Designer respectively. These new terminologies are applicable only for OFSAI versions 7.3.2.3.0 and above.

User can view the extract definitions by going through the following steps:

1. Navigate to Unified Metadata Manager and select Data Management Tools, and then select, Data Ingestion, and then select Database Extracts section.
2. In the left pane, expand the application as defined during application installation and click the Data Source defined during application installation.
3. Expand the required T2T definition to view the Database Extract definitions.

You can view the Source - Target mapping definitions by performing the following steps:

Note

The Data Management Tools and Data Ingestion were previously known as Data Integrator Framework and Warehouse Designer respectively. These new terminologies are applicable only for OFSAI versions 7.3.2.3.0 and above.

1. Navigate to Unified Metadata Manager and select Data Management Tools, and then select, Data Ingestion, and then select Database Extracts section.
2. In the left pane, expand the application as defined during application installation and click the Data Source defined during application installation.
3. Expand the required T2T definition to view the extract definition.
4. Click the required Database Extract definition.
5. The selected Database Extract definition details are displayed with the available Source - Target mappings under the Source - Target Mappings grid.

Note

Staging transaction summary tables contain alphanumeric display codes for various IDENTIFIER and CODE columns. T2T mapping looks up in respective dimension tables for fetching an equivalent numeric ID and CODE corresponding to the alphanumeric display code. Hence, these dimension tables should be populated with the alphanumeric display code before executing any data movement tasks.

5.10.4 Dependencies

- Instrument tables should be loaded before loading the transaction summary information related to those instruments.
- Account Number is an alphanumeric unique identifier within each staging transaction summary tables. `ID_NUMBER` is a numeric unique identifier within processing transaction summary tables. Hence, there is a need to look up into a `DIM_ACCOUNT` dimension table for a numeric surrogate key for each of the alphanumeric account number. This dimension table `DIM_ACCOUNT` will be populated as part of the process that loads instrument tables. For more information on loading instrument tables, see Loading Instrument Table Data.
- Before executing T2T data movement tasks, user should ensure that all the dimension tables that are required for instruments data are loaded. The following are some of the mandatory dimensions:
 - `DIM_ACCOUNTS`
 - `DIM_PRODUCTS_B`
 - `DIM_GENERAL_LEDGER_B`
 - `DIM_COMMON_COA_B`
 - `DIM_ORG_UNIT_B`

5.10.5 Executing T2T Data Movement Tasks

To execute T2T Data Movement Tasks, follow these steps:

1. Create a new Batch with the Task
2. Specify the following parameters for the task for loading Historical Exchange Rates:

Datastore Type: Select appropriate datastore from the drop down list.

Datastore Name: Select appropriate name from the list. Generally it is the Infodom name.

IP address: Select the IP address from the list.

Rule Name: Select the appropriate T2T name from the above list.

Parameter List: No Parameter is passed. The only parameter is the As of Date Selection while execution.

Check T2T component logs and batch messages for checking the status of load. T2T component can fail because of following cases:

- **Unique constraint error:** Target table may already contain the primary keys that are part of the staging tables.
- **NOT NULL constraint error:** Staging table do not have values for mandatory columns of the target table.

5.10.6 Re-Load Of Transaction Summary Data

T2T component can only perform Insert operations. In case user needs to perform updates, previously loaded records should be deleted before loading the current records.

Function `fn_deleteFusionTables` is used for deleting the records in the target that are present in the source. This function removes rows in the table if there are matching rows in the Stage table. This function needs `FSI_DELETE_TABLES_SETUP` to be configured. Configure the

following table for all columns that need to be part of the join between Stage table and Equivalent table.

Create a new Batch with the Task and specify the following parameters for the task to delete existing records:

- **Datastore Type:** Select appropriate datastore from the drop down list.
- **Datastore Name:** Select appropriate name from the list. Generally it is the Infodomain name.
- **IP address:** Select the IP address from the list.
- **Rule Name:** fn_deleteFusionTables
- **Parameter List:** 'Table to be deleted'

Batch Run ID and As Of Date are passed internally by the batch to the Data Transformation task.

Sample record for FSI_DELETE_TABLES_SETUP table is given following:

Table 5-9 Sample record for FSI_DELETE_TABLES_SETUP table

Column Name	Column Description	Sample Value
STAGE_TABLE_NAME	Stores the source table name for forming the join statement	STG_LOAN_CONTRACTS
STAGE_COLUMN_NAME	Stores the source column name for forming the join statement	V_ACCOUNT_NUMBER
FUSION_TABLE_NAME	Stores the target table name for forming the join statement	FSI_D_LOAN_CONTRACTS
FUSION_COLUMN_NAME	Stores the target column name for forming the join statement	ACCOUNT_NUMBER

Note

Insert rows in FSI_DELETE_TABLES_SETUP for all columns that can be used to join the stage with the equivalent table. In case if the join requires other dimension or code tables, a view can be created joining the source table with the respective code tables and that view can be part of the above setup table.

5.11 Ledger Data Loader

The LEDGER_STAT load utility is an Oracle stored procedure used to load your ledger data into the Oracle Financial Services Analytical Applications (OFSA) LEDGER_STAT table.

Topics:

- [Features of the Load procedure](#)
- [Setup for the LEDGER_STAT Load utility](#)
- [Exception Messages](#)
- [Tables Cleanup after Truncation of Ledger_Stat](#)

There are three types of load tables that can be used for loading ledger data.

- **Type I (FISCAL_ONE_MONTH):** Load table contains ONE_MONTH column for storing data corresponding to one of the twelve fiscal months.

- Type II (`FISCAL_RANGE`): Load table contains M1 to M12 columns for storing data corresponding to twelve fiscal months.
- Type III (`CALENDAR_MONTHS`): Load table contains `AS_OF_DATE` for storing data corresponding to an as-of-date. While Type II table contains ledger data across fiscal months in a single row, Type III contains the same information in multiple rows. Type III supports calendar dates and data can be for one or multiple dates.

ASCII Ledger data is loaded into any of the above staging or load tables using F2T component of OFSAAI framework. This component can be used for loading any flat file data into tables. For more information on how to load data using F2T, see OFSAAI User Guide.

`LEDGER_STAT` load utility is a PL/SQL procedure and loads data from the above staging tables into `LEDGER_STAT` table, based on the configuration. Runtime parameters, such as the name of the load table, which all columns to load, ADD or REPLACE update functionality, and whether or not to create offset records are passed as parameters to the procedure and these are inserted into the Load Batch table (`FSI_LS_LOAD_BATCH`).

The procedure is implemented as an Oracle PL/SQL stored procedure so it can be invoked from SQL*Plus or Batch execution screen within OFSAAI Batch Maintenance component. Input parameters are read from the batch/parameter table and validated for correctness, completeness and consistency before the load begins. Parameter errors are written to a Message column in the batch/parameter table and `FSI_MESSAGE_LOG` table. Runtime statistics are written to the batch/parameter record following completion of the load for that record.

Note

For supporting loading `LEDGER_STAT` from Type III staging table, a global temporary table (GTT) is created within database. Data is moved from global temporary table into `LEDGER_STAT` table.

5.11.1 Features and Limitations of the Load Procedure

The `LEDGER_STAT` load utility is the only supported method for loading your ledger data into the `LEDGER_STAT` table. The `LEDGER_STAT` load utility offers the following features:

- You can load ledger data for one month or for a range of months.
- You can also load ledger data based on calendar as-of-dates.
- A month can be undone individually, using the Ledger Load Undo process. You can do this even though the month to be undone is included in a multiple-month load.
- You can update columns in existing `LEDGER_STAT` rows using either the additive or replacement functionality.
- You can bypass the upsert logic and insert all the rows from the load table using the `INSERT_ONLY` mode. This functionality can be used either for first-time loads or to reload for all months with each load.

Limitations

The following are the limitations.

- Load Table Rows Must Be Unique
- A restriction imposed by the use of bulk SQL (as opposed to row-by-row) processing is that all the rows in the load table(s) must be unique. This means that there is one row in the

load table for one row in LEDGER_STAT. A unique index is created on each load table to enforce this uniqueness and provide acceptable performance.

- Defining Financial Elements in AMHM
- Occasionally, your load table may contain dimension member values for one or more dimensions that are not defined in AMHM. The LEDGER_STAT load procedure loads these rows anyway, except for the rows containing undefined or incompletely defined FINANCIAL_ELEM_ID values.
- Any new values for FINANCIAL_ELEM_ID must first be defined in AMHM before running the load. Specifically, the load procedure needs the AGGREGATE_METHOD value for each FINANCIAL_ELEM_ID value so that the YTD columns in LEDGER_STAT can be computed using the appropriate method.

5.11.2 Setup for the LEDGER_STAT load utility

Setting up and Executing a Type III (or Type 3) Ledger Stat Load Using STG_GL_DATA . The Type 3 load takes data from STG_GL_DATA and transfers it into the LEDGER_STAT table. Steps to follow to setup and run a Type III Ledger Stat Load:

1. Populate STG_GL_DATA The following columns in STG_GL_DATA must be populated with valid values:

Option Column	Description
V_GL_CODE	General Ledger Code value.
FIC_MIS_DATE	This field indicates the current period As of Date applicable to the data being loaded.
V_ORG_UNIT_CODE	Org Unit Code value.
V_SCENARIO_CODE	Populate with a value from the CONSOLIDATION_DISPLAY_CODE column from the FSI_CONSOLIDATION_CD table (ex. ACTUAL, BUDGET).
V_CCY_CODE	ISO Currency Code from FSI_CURRENCIES (ex. USD)
V_PROD_CODE	Product Code value.
V_FINANCIAL_ELEMENT_CODE	Populate with a value from the FINANCIAL_ELEM_CODE column from the DIM_FINANCIAL_ELEMENTS_B table (ex. ENDBAL, AVGBAL).
V_COMMON_COA_CODE	Common COA Code value.
N_AMOUNT_LCY	Balance

The following columns in STG_GL_DATA must be populated because they are defined as NOT NULL but can be defaulted to the value of your choice because they are not used:

V_LV_CODE

V_BRANCH_CODE

F_CONSOLIDATION_FLAG

V_GAAP_CODE

- Verify data exists in the view `STG_GL_DATA_V`. The following SQL statement is used to populate this view:

```
SELECT v_data_origin DS, f_consolidation_flag ACCUM_TYPE, fcc.consolidation_cd
CONSOLIDAT, v_ccy_code ISOCRNCYCD, dfec.financial_elem_id FINANC_ID,
doub.org_unit_id ORG_ID, dglb.gl_account_id GL_ACCT_ID, dccb.common_coa_id
CMN_COA_ID, dpb.product_id PRDCT_ID, fic_mis_date AS_OF_DATE, n_amount_lcy
VALUE, 0 baltypecdFROM STG_GL_DATA SGD, DIM_GENERAL_LEDGER_B
DGLB, DIM_ORG_UNIT_B DOUB, DIM_PRODUCTS_B DPB, DIM_FINANCIAL_ELEMENTS_B
DFEB, DIM_COMMON_COA_B DCCB, FSI_CURRENCIES FC, FSI_CONSOLIDATION_CD FCCWHERE
NVL(n_amount_lcy, 0) <> 0AND SGD.V_GL_CODE = DGLB.GL_ACCOUNT_CODEAND
SGD.V_ORG_UNIT_CODE = DOUB.ORG_UNIT_CODEAND SGD.V_PROD_CODE =
DPB.PRODUCT_CODEAND SGD.V_FINANCIAL_ELEMENT_CODE = DFEB.FINANCIAL_ELEM_CODEAND
SGD.V_COMMON_COA_CODE = DCCB.COMMON_COA_CODEAND SGD.V_CCY_CODE =
FC.ISO_CURRENCY_CDAND SGD.V_SCENARIO_CODE = FCC.CONSOLIDATION_DISPLAY_CODE;
```

As seen in the code above, the view references the `_CODE` columns on the dimension tables. For example, `COMMON_COA_CODE` on `DIM_COMMON_COA_B` and `ORG_UNIT_CODE` on `DIM_ORG_UNIT_B`. These code columns must be populated for data to exist in `STG_GL_DATA_V`.

The `Update_Dimension_Code` (`fn_updatedimensioncode`) program populates these Code columns using data from values in the Code dimension Attribute (for example, `COMMON COA CODE`, `ORG UNIT CODE`, and so on).

The `BALTYPECD` column has a default value of 0 in the View, as this column is not null in `LEDGER_STAT`. `Baltypecd` is not a Dimension. It indicates the credit or debit of the same account details. Since same account can hold both credit and debit, this column should be populated in the source with a value. It is the part of the unique Index and Not Null column in `LEDGER_STAT`.

- If using the Type 3 Ledger Stat Load for the first time, run the GTT table creation procedure.

The GTT table creation procedure creates the Global Temporary Table `LS_LOAD_TABLE_GTT_V`.

The `fn_ledger_load_create_gtt` function creates the table `LS_LOAD_TABLE_GTT_V` and the index `UK_GTT` for use in the Type 3 Ledger Stat Load.

Note

If the GTT table has not been created and you try to execute the Ledger Stat Load, you will get the following error in `FSI_MESSAGE_LOG:WRAPPER_LEDGER_STAT_LOAD`- Error: -942: `ORA-00942: table or view does not exist`

- Populate `FSI_LS_LOAD_BATCH`. You need to populate the following columns:

Option	Description
<code>RUN_FLAG</code>	Y
<code>SEQUENCE</code>	Sequence value (ex. 1)
<code>LOAD_TABLE_NAME</code>	<code>STG_GL_DATA</code>
<code>ONE_MONTH_ONLY</code>	N
<code>UPDATE_MODE</code>	ADD or REPLACE

Option	Description
INSERT_ONLY	Y or N
CREATE_OFFSETS	N
IS_CALENDAR_MONTH	Y
START_CALENDAR_MONTH	Starting date to load in format YYYYMMDD.
END_CALENDAR_MONTH	Ending date to load in format YYYYMMDD.

5. Run the Ledger Stat Load. Use the following command to run the Type 3 Ledger Stat Load in SQL*Plus as the atomic user:

```
DECLARE x NUMBER :=0; BEGIN x := ofsa_util.wrapper_ledger_stat_load('BATCH_ID', 'MIS_DATE', 'TABLE_NAME', 'TABLE_TYPE', 'UPDATE_MODE', 'INSERT_ONLY', 'START_DATE', 'END_DATE') dbms_output.put_line ('The return variable is ' || x); END; DECLARE x NUMBER :=0; BEGIN x := ofsa_util.wrapper_ledger_stat_load('ARALSLOADTYPE3_4', '20110111', 'STG_GL_DATA', 'CALENDAR_MONTHS', 'ADD', 'Y', '20101231', '20101231'); dbms_output.put_line ('The return variable is ' || x); END;
```

After the Ledger Load completes, check the tables `FSI_MESSAGE_LOG` and `FSI_LS_LOAD_BATCH` for errors.

Note

For ledger load table name is `ledger_load` and data source value is the `V_DATA_ORIGIN` from `STG_GL_DATA`. For `Ledger_stat` with the same data source will have same identity code.

5.11.2.1 FSI_DATA_IDENTITY insert/update during Ledger_load

Insert happens into `FSI_DATA_IDENTITY` with new identity code if the new Data origin used in the `STG_GL_DATA`.

Update happens in case same set of Data source used `STG_GL_DATA`.

Updates set happens

`start_time = SYSDATE,`

`end_time = SYSDATE,`

`number_of_entries = no.of entries for the current load`

Here, `STG_GL_DATA` is considered as a source of input for `LEDGER_LOAD`. In case of other source table, the rule is same in populating data into `FSI_DATA_IDENTITY`.

5.11.2.2 Creating View on LEDGER_STAT table

A view is created on the `LEDGER_STAT` table called `LSL`. The purpose of this view is to provide shorter column names for the load procedure. The `LSL` view must contain the same columns as `LEDGER_STAT`. Column alias for each columns within the view should match the `COLUMN_ALIAS` user-defined property that is set for each column of `LEDGER_STAT` table in the ERwin model.

For any user-defined dimensions in your `LEDGER_STAT` you must complete the following steps.

- In ERwin model, look up the `COLUMN_ALIAS` User Defined Property (UDP) for added dimension columns within `LEDGER_STAT` table.
- Specify the value of the property `COLUMN_ALIAS`.
- Modify the view to include new dimension columns. Use the same `COLUMN_ALIAS` that was mentioned in the ERwin model in the load table view.

5.11.2.3 Creating Load Table

This step is applicable for loading ledger data from Type I or Type II load table. Staging table `STG_GL_DATA` (used for Type III load) is packaged with the application. Multiple load tables (Type I or Type II) can be created as required by the System Administrator. Table structure for the Type I and Type II load tables is given in the following sections:

```

-----
-- Uncomment the m1..m12 columns if you plan to load a range of months (Type II Load Table).
-- Add lines for all of the LEDGER_STAT user-defined leaf columns in the place
-- indicated below. Don't forget to add commas if you need to.

CREATE TABLE &load_table_name( ds VARCHAR2(12) NOT NULL, -- data_source year_s
NUMBER(5) NOT NULL, accum_type char(1) NOT NULL, consolidat NUMBER(5) NOT NULL,
isocrncycd VARCHAR2(3) DEFAULT '002' NOT NULL, financ_id NUMBER(14) NOT NULL,
org_id NUMBER(14) NOT NULL, gl_acct_id NUMBER(14) NOT NULL, cmn_coa_id NUMBER(14)
NOT NULL, prdct_id NUMBER(14) NOT NULL, baltypecd NUMBER(5) DEFAULT 0 NOT NULL,
-- -- m1 NUMBER(15,4), -- m2 NUMBER(15,4), -- m3 NUMBER(15,4), -- m4
NUMBER(15,4), -- m5 NUMBER(15,4), -- m6 NUMBER(15,4), -- m7 NUMBER(15,4), -- m8
NUMBER(15,4), -- m9 NUMBER(15,4), -- m10 NUMBER(15,4), -- m11 NUMBER(15,4), --
m12 NUMBER(15,4), -- one_month_amt NUMBER(15,4) -- --
----- -- Other
leaf columns (PROPERTY_COLUMN from REV_COLUMN_PROPERTIES for LEDGER_STAT): --
----- -- . . . -- )
-----

```

5.11.2.4 Creating Unique Index on Load Table

This step is applicable for loading ledger data from Type I or Type II load table. A unique index has to be created on each load table specifying the column alias for each column within the load table. Column alias should match the column alias specified for columns within `LEDGER_STAT` table. `LEDGER_STAT` load procedure identifies the source columns that need to be loaded using the column aliases and not by the physical column names. Column alias for `LEDGER_STAT` columns are specified in the user-defined property (UDP) `COLUMN_ALIAS` within ERwin model. Refer to ERwin model for getting the column alias for each of the `LEDGER_STAT` columns. Definition of the unique index is given following:

```

CREATE UNIQUE INDEX &load_table_name ON &load_table_name ( ds, year_s,
accum_type, consolidat, isocrncycd, financ_id, org_id, gl_acct_id, cmn_coa_id,
prdct_id baltypecd, -----
Include all additional LEDGER_STAT primary key -- leaf columns here (use
PROPERTY_COLUMN from REV_COLUMN_PROPERTIES): ----- . . . -- )

```

The unique key of the load table must be identical to the unique key of `LEDGER_STAT`, with the exception that instead of `IDENTITY_CODE`, which is in `LEDGER_STAT`, the load table has a column called `DS` (Data Source).

5.11.2.5 Creating Views on Load Table

This step is applicable for loading ledger data from Type I or Type II load table. In addition to load tables, views have to be created on the staging tables similar to the view LSL that was created on LEDGER_STAT. A view has to be created on each load table specifying the columns alias for each column within the load table. Column alias should match the column alias specified for columns within LEDGER_STAT table. LEDGER_STAT load procedure identifies the source columns that need to be loaded using the column alias. Column alias for LEDGER_STAT columns are specified in the user-defined property (UDP) COLUMN_ALIAS within ERwin model. See the ERwin model for getting the column alias for each of the LEDGER_STAT columns. View definition is given following:

```

-----
-- Uncomment the m1..m12 columns if you plan to load a range of months (Type II Load table).
-- Add lines for all of the LEDGER_STAT user-defined leaf columns in the place
-- indicated below. Don't forget to add commas if you need to.
-----

CREATE OR REPLACE VIEW &load_table_name._v AS SELECT ds, year_s, accum_type,
consolidat, isocrncyd, financ_id, org_id, gl_acct_id, cmn_coa_id, prdct_id,
baltypecd, -- -- NVL(m1,0) AS m1, -- NVL(m2,0) AS m2, -- NVL(m3,0) AS m3, --
NVL(m4,0) AS m4, -- NVL(m5,0) AS m5, -- NVL(m6,0) AS m6, -- NVL(m7,0) AS m7, --
NVL(m8,0) AS m8, -- NVL(m9,0) AS m9, -- NVL(m10,0) AS m10, -- NVL(m11,0) AS m11,
-- NVL(m12,0) AS m12, -- NVL(one_month_amt,0) AS one -- --
----- -- Other
leaf columns (PROPERTY_COLUMN from REV_COLUMN_PROPERTIES for LEDGER_STAT): --
----- . . . --
FROM &load_table_name WHERE NVL(one_month_amt,0) <> 0; -- -- OR NVL(m1,0) <> 0 --
OR NVL(m2,0) <> 0 -- OR NVL(m3,0) <> 0 -- OR NVL(m4,0) <> 0 -- OR NVL(m5,0) <> 0
-- OR NVL(m6,0) <> 0 -- OR NVL(m7,0) <> 0 -- OR NVL(m8,0) <> 0 -- OR NVL(m9,0) <>
0 -- OR NVL(m10,0) <> 0 -- OR NVL(m11,0) <> 0 -- OR NVL(m12,0) <> 0;

```

In case, the custom dimensions are added to the load table, views need to be modified to reflect the same.

5.11.2.6 Setting up Global Temporary Table

This step is applicable for loading ledger data from Type III. Calendar dates present in the data of Load table are converted to the corresponding Fiscal Year/Month. Conversion from calendar date to fiscal year & month is done based on the START_MONTH column present in FSI_FISCAL_YEAR_INFO table. These derived fiscal year & fiscal month are then inserted in an intermediate Global Temporary Table (GTT) after aggregating the rows of same months/years. Therefore, if 12 rows are present for the same fiscal year each corresponding to a different month, then global temporary table may have maximum of one row corresponding to the fiscal months, these 12 rows represent.

GTT needs to contain valid dimension member identifiers and numeric codes. Since staging table contains alphanumeric identifiers and codes, a view is created on STG_GL_DATA table joining with other relevant dimension and CD/MLS tables before being used in the GTT creation.

Global temporary table can be created in 2 ways:

- **Using PL/SQL**
Declare output number; Begin Output:= fn_ledger_load_create_gtt('BATCH_ID', 'AS_OF_DATE', 'TABLE_NAME'); End; AS_OF_DATE is the date for which GTT is created, in YYYYMMDD format. TABLE_NAME is the staging table name STG_GL_DATA. An example of running the function from SQL*Plus is as follows: SQL> var output number; SQL> execute :output:= fn_ledger_load_create_gtt('BATCH_ID', '20100519', 'STG_GL_DATA');
- **Using OFSAAI Batch Maintenance**
To execute the procedure from OFSAAI Batch Maintenance, run the batch mentioned following and specify the following parameters:
 - **Datastore Type:** Select appropriate datastore from list
 - **Datastore Name:** Select appropriate name from the list
 - **IP address:** Select the IP address from the list
 - **Rule Name:** fn_ledgerLoadGTTCreation
 - **Parameter List:** AS_OF_DATE and TABLE_NAME
TABLE_NAME is the staging table name STG_GL_DATA.
AS_OF_DATE should be passed as 'YYYYMMDD' format.

 **Note**

BATCHID will be passed explicitly in Batch Maintenance. The appropriate table parameters are enclosed in single quotes.

5.11.2.7 Tables Related to LEDGER_STAT Load Procedure

LEDGER_STAT Loader utility uses the following tables:

- **FSI_FISCAL_YEAR_INFO:** The table contains the fiscal year information. This is a setup table.
- **FSI_LS_LOAD_BATCH:** The table contains the parameters for the load batch that needs to be executed for loading ledger data from staging or load table into LEDGER_STAT. This is a setup table.
- **STG_GL_DATA:** The staging table contains the ledger data for various as-of-dates.
- **LEDGER_STAT:** The processing table contains the ledger data for various fiscal months. This is loaded from staging table.

5.11.2.8 Populating Stage Tables

Data for ledger can come from external systems. Such data has to be in the format of the staging table. This data can be loaded into staging through F2T component of OFSAAI framework. Users can view the loaded data by querying the staging tables and various log files associated with F2T component.

5.11.2.9 Executing LEDGER_STAT Load Procedure

You can execute this procedure either from SQL*Plus or from within a PL/SQL block or from Batch Maintenance window within OFSAAI framework.

To run the procedure from SQL*Plus, login to SQL*Plus as the Schema Owner. The procedure/ batch requires the following 9 parameters:

- BATCH_ID- Any unique number to identify the execution run.
- MIS_DATE- Date on which the loading is done expressed in YYYYMMDD format.
- TABLE_NAME- STG_GL_DATA(Type III) or any other load table (TYPE I or TYPE II)
- TABLE_TYPE- FISCAL_ONE_MONTH or FISCAL_RANGE (TYPE I or TYPE II)
- CALENDAR_MONTHS (TYPE III)
- UPDATE_MODE- ADD/REPLACE
- INSERT_ONLY- Y/N
- START_DATE- Calendar start date in YYYYMMDD
- END_DATE- Calendar end date in YYYYMMDD

The input parameter logic for the Type III, Type II and Type I tables.

5.11.2.10 CALENDAR_MONTHS

- If Start_Date and End_Date are null then month part of MIS_Date is taken for processing a particular month. (Example: if MIS_DATE is 20101231 then the December calendar month data is processed).
- In this case the Start_Date and End_Date becomes optional.

5.11.2.11 FISCAL_ONE_MONTH

- The Start_Date and End_Date parameters will hold numeric values identifying the fiscal month. The value of these parameters will be between 1 and 12 (that is, M1 till M12).
- The Start_Date and End_Date should be same.
- In this case the Start_Date and End_Date are mandatory.

5.11.2.12 FISCAL_RANGE

- The Start_Date and End_Date parameters will hold numeric values identifying the fiscal month. The value of these parameters will be between 1 and 12 (that is, M1 till M12).
- The Start_Date and End_Date parameters will specify the range of fiscal months which are to be processed. Ex: M1 till M6 in case the Start_Date and End_Date values are 1 and 6.
- In this case the Start_Date and End_Date are mandatory.

Ledger Load can be executed in 2 different ways:

- Using PL/SQL:
By using the function

```
ofsa_util.wrapper_ledger_stat_load('BATCH_ID','MIS_DATE', TABLE_NAME',
TABLE_TYPE', 'UPDATE_MODE', 'INSERT_ONLY','START_DATE','END_DATE');
```

Example:

```
DECLARE x NUMBER :=0; BEGIN x :=
ofsa_util.wrapper_ledger_stat_load('batch_id_1','20090202','STG_GL_DATA','CALE
NDAR_MONTHS','ADD','Y','20070430','20080331'); dbms_output.put_line ('The
return variable is ' || x); END;
```

- Using Batch Maintenance
To execute the procedure from OFSAAI Batch Maintenance, create a new Batch with the Task as TRANSFORM DATA and specify the following parameters for the task:
 - Datastore Type: Select appropriate datastore from list
 - Datastore Name: Select appropriate name from the list
 - IP address: Select the IP address from the list
 - Rule Name: fn_ledgerDataLoader
 - Parameter List: <Same as mentioned above in the parameter list>

5.11.2.13 Executing LEDGER_STAT Load Procedure for MULTI CURRENCIES

The data for the Ledger can have more than one currency at a time that is multi currencies input.

To execute multi currencies, you need to enable the flag using the following statement:

```
update fsi_db_info f1 set f1.multi_currency_enabled_flg =1;
commit;
```

Note

FTP supports the following two scenarios for Ledger Migration:

- If multi-currency is not used:
FSI_DB_INFO
Multi_currency_enabled_flg = 0
Currency_type_enabled_flg = 0
Functional_currency_cd = same as iso_currency_cd in ledger and instrument data
Ledger Data (FE 100 and FE 140) must exist in LS and must only be in Functional currency (iso_currency_cd = functional currency and currency_type_cd = 2)
- If multi-currency is used:
FSI_DB_INFO
Multi_currency_enabled_flg = 1
Currency_type_enabled_flg = 1
Functional_currency_cd = same as iso_currency_cd in ledger and instrument data
Instrument Data can be in multiple currencies Ledger Data (FE 100 and FE 140) must exist in LS and must only be in Functional currency (iso_currency_cd = functional currency and currency_type_cd = 2). Exchange rate data should be present in Exchange Rate Direct Access table for converting entered currency to functional currency.

5.11.3 Exception Messages

The ledger load program throws both user defined exceptions and Oracle database related exceptions. These exception messages could be seen in FSI_MESSAGES_LOG table with the help of the batch_id which was used during execution. The exception list includes all possible validations on the parameters that were passed and database related exceptions.

5.11.4 Tables Cleanup After Truncation Of Ledger_Stat

The LEDGER_STAT procedure makes entries into certain audit tables. Whenever the user truncates/deletes the Data from LEDGER_STAT, he needs to additionally remove the auditing entries from the tables FSI_DATA_IDENTITY, FSI_M_SRC_DRIVER_QUERY and FSI_LS_MIGRATION_RESULTS. This procedure enables the user to clean up these audit tables.

Executing the clean up of Ledger_Stat Load Procedure

To run the procedure from SQL*Plus, login to SQL*Plus as the Schema Owner:

```
Fn_ledger_stat_cleanup(batch_run_id VARCHAR2, as_of_date VARCHAR2) SQLPLUS >
declare result number; begin result:=Fn_ledger_stat_cleanup
('LEDGER_CLEANUP_BATCH1','20121212'); end; /
```

BATCH_RUN_ID is any string to identify the executed batch.

AS_OF_DATE in the format YYYYMMDD.

Ledger Stat Clean up Procedure

To execute Ledger Stat Clean up from OFSAAI Batch Maintenance, a seeded Batch is provided.

The following batch parameters are mentioned:

- **Datastore Type:** Select the appropriate datastore from list
- **Datastore Name:** Select the appropriate name from the list
- **IP address:** Select the IP address from the list
- **Rule Name:** Fn_ledger_stat_cleanup

5.12 Cash Flow Loader

At times customers source some of their processed cash flow result data (not payment schedule) from 3rd party information providers or from internal systems. The cash flow data loader provides a way to load externally sourced cashflows into an ALM process and have these aggregate into the ALM result output tables. These external cash flows can be loaded into the Stg_Account_Cash_Flows table and merged with OFSAA generated results by executing Cash Flow Loader.

- Data can be aggregated at Product / Organisation Unit / Currency level or given at account level. Behaviour of Cash Flow Loader is controlled by setup entries in tables SETUP_MASTER and FSI_CASH_FLOW_LOADER_SETUP.
- ALM generates results in both base and consolidated currency. Same capability is available via Cash Flow Loader. Cash flows in both base currency and consolidated currency can be loaded. The loader does not do currency conversion and consolidation. It is expected to be done at source or during data load to staging. This is controlled by currency type column. If only base currency data is given then only RES_DTL and FSI_O_RESULT_MASTER are populated. If consolidated is given then CONS_DTL and FSI_O_CONSOLIDATED_MASTER are also populated.
- The cash flows in stage are mapped to process and scenario which must be configured in ALM. Currently the loader supports only deterministic processes. Stochastic processes are not supported.

- There is no filter for instrument type cd cash flow loader package. However, it validates stage supplied instrument type codes with OOB provided instrument type codes, so that only valid records get processed.
- If irregular cashflows is used in STG_PAYMENT_SCHEDULE table (AMRT Type as 801) , then cash flows are required to be loaded from last payment date till maturity date to populate the first interest payment accurate. This same logic is followed while loading to STG_ACCOUNT_CASH_FLOW as well. STG_ACCT_CASH can have only one instrument type cd to distinguish between the CASH FLOWS loaded from different type of accounts.

5.12.1 Cash Flow Loader Table

- **STG_ACCOUNT_CASH_FLOW:** This table is used to store the cash flow generated by the different sources for loading purpose. This is a staging table where ALM expected cashflows data to be loaded from back office/bank. There is no dependency of FSI_D tables to load this table.

Data is expected as per following details:

Column name	Data expectation
Extraction Date (fic_mis_date)	MIS date for which the given data is valid, also called As of Date
Cash Flow Date (d_cash_flow_date)	Calendar date on which cash flow or other event occurs
Id Number (n_account_id)	This is equivalent of ID_NUMBER in EPM processing tables (ex: FSI_D_LOAN_CONTRACTS) and is used to map cash flows with their corresponding Instrument record. If aggregated cash flows are loaded then this column can be defaulted to -1
Identity Code (n_acct_data_identity_cd)	This is equivalent of IDENTITY_CODE in EPM processing tables (ex: FSI_D_LOAN_CONTRACTS) and stores as of date in number (YYYYMMDD) format
Cash Flow Amount (n_cash_flow_amount)	This column stores the cash flow or other amount, depending on financial element, on event date
Cash Flow Sequence (n_cash_flow_sequence)	Sequence in which event occurs is mentioned here. It can be a running number and is used to identify order in which an event occurs if there are multiple events on same date.
Currency Type Code (n_currency_type_cd)	This column decides whether the given cash flows are for Base (Natural) currency or consolidated (Reporting) currency. Based on it loader will move to either RES_DTL_XXX or CONS_DTL_XXX table. Expected values are: '1' for base/natural (also called entered) currency and '2' for consolidated/reporting (also called functional) currency. Corresponding reference tables are FSI_CURRENCY_TYPE_CD and FSI_CURRENCY_TYPE_MLS - If you have selected Consolidate to Reporting Currency in ALM Process, then following cases are possible with respect to Consolidation Flag, n_currency_type_cd =1, and n_currency_type_cd = 2.

- Case 1: If Consolidation Flag is OFF and N_Currency_Type_Cd = 1, then process will execute successfully and only RES_DTL_XX will get populated.
- Case 2: If Consolidation Flag is OFF, N_Currency_Type_Cd = 1, and N_Currency_Type_Cd = 2, then process will execute successfully and only RES_DTL_XX will populate for records N_Currency_Type_Cd=1 and ignore records of type N_Currency_Type_Cd=2.
- Case 3: If Consolidation Flag is ON and N_Currency_Type_Cd = 1, then Loader would fail as it expects records for N_Currency_Type_Cd=2 when CONSOLIDATED_OUTPUT_FLG is ON.
- Case 4: If Consolidation Flag is ON, N_Currency_Type_Cd 1, and N_Currency_Type_Cd = 2, then it logs as No Data in the instrument table for the given FIC MIS DATE. Loading data to fsi_o_consolidated_master Failed error in FSI_MESSAGE_LOG. You need to load data if Consolidation Flag is ON and N_Currency_Type_Cd = 2. Here, N_Currency_Type_Cd = 2 signifies that the records are of consolidation type and meant for CONT_DTL once processing them. Similarly, N_Currency_Type_Cd = 1 indicates that non-consolidated records in stage and meant for RES_DLT processing.
- Note: If there are records for n_currency_type_cd = 1 and n_currency_type_cd =2 does not records, then Cash Flow Loader cannot use the same set of records loaded for n_currency_type_cd = 1 and convert it for consolidation.

Instrument Type Code (n_instrument_type_cd)

This identifies the type of instrument that is, loan, deposit and so on. for which data is being loaded. Corresponding reference tables are FSI_INSTRUMENT_TYPE_CD and FSI_INSTRUMENT_TYPE_MLS

Scenario Number (n_scenario_no)

An ALM process can have multiple forecast rate scenario. This column indicates the scenario for which data has been loaded. It is used by loader to map data to corresponding scenario of ALM process. Reference tables are FSI_CASH_FLOW_LOADER_SETUP (scenario_num) and DIM_FCST_RATES_SCENARIO (n_scenario_num). Loader takes ALM Process Id as input and then checks corresponding scenario numbers in ALM metadata tables for validation

Account / Contract Code (v_account_number)

This column stores the alpha-numeric unique account or contract number for which data is being loaded. This is generally the unique identifier from operational source systems. Corresponding reference table in DIM_ACCOUNT (v_account_number). If aggregated cash flows are loaded then this column can be defaulted to -1

Cash Flow Type (v_cash_flow_type)	Indicates whether the cash flow is Inflow or Outflow. Values expected are 'I' for inflow and 'O' for outflow. Note that, this column is not used by cash flow loader, hence does not impact any result in ALM. It is used by Liquidity Risk Management (LRM) application and is a mandatory column in table.
Currency Code (v_ccy_code)	Three letter ISO currency code in which the cash flow amount is denominated must be given in this column.
Common Coa Code (v_common_coa_code)	Common Chart of Account code of the account number for which data is loaded must be given here. Corresponding reference table is DIM_COMMON_COA_B (common_coa_code)
Data Origin (v_data_origin)	Code of the source system from where data is obtained is expected here. Corresponding reference table is DIM_DATA_ORIGIN (v_data_source_code)
Financial Element Code (v_financial_element_code)	This indicates the financial element that is, nature of amount loaded. Corresponding reference table is DIM_FINANCIAL_ELEMENTS_B (financial_elem_code)
GL Account Code (v_gl_account_code)	General Ledger Account code of the account number for which data is loaded must be given here. Corresponding reference table is DIM_GENERAL_LEDGER_B (gl_account_code)
Lv Code (v_lv_code)	Legal Entity code of the account number for which data is loaded must be given here. Corresponding reference table is DIM_LEGAL_ENTITY_B (legal_entity_code)
Organization Unit Code (v_org_unit_code)	Organisation or Business Unit code of the account number for which data is loaded must be given here. Corresponding reference table is DIM_ORG_UNIT_B (org_unit_code)
Product Code (v_prod_code)	Product code of the account number for which data is loaded must be given here. Corresponding reference table is DIM_PRODUCTS_B (product_code)

- Output tables: Aggregated Cash Flows will be populated in the following output tables
 - RES_DTL_XX
 - CONS_DTL_XX
 - FSI_O_RESULT_MASTER
 - FSI_O_CONSOLIDATED_MASTER
 XX denotes the Process ID.
- SETUP_MASTER: This table will be used in the case of instrument cash flows. For Instrument cash flows, an entry against V_COMPONENT_VALUE of the SETUP_MASTER table should have values either 0 or 1 which indicate if id numbers or account numbers are provided, respectively.
- FSI_ALM_DETERMINISTIC_PROCESS: This table will be used for loading cash flows in CONSOLIDATED tables. To populate consolidated tables, the CONSOLIDATED_OUTPUT_FLG should be 1 in the fsi_alm_deterministic_process table against the cash flow process ids.

- **FSI_CASH_FLOW_LOADER_SETUP**: This table will have all the process ids for cash flow loader. Only those processes will be executed which have status 'N' in **FSI_CASH_FLOW_LOADER_SETUP** table. In such case, those processes will be already existing into the system.
- **FSI_M_USER_ACTIVE_TIME_BUCKETS**: For cash flow loader, user should be mapped to an active time bucket in the **FSI_M_USER_ACTIVE_TIME_BUCKETS** table.
- **TIME BUCKETS**: The following tables will store the time bucket details:
 - **FSI_TIME_BUCKET_MASTER**
 - **FSI_M_LR_IRR_BUCKETS**
 - **FSI_LR_IRR_BUCKETS_AUX**
 - **FSI_TIME_BKT_ISB**
 - **FSI_TIME_BKT_LR_LRR_DATES**

5.12.2 Data Validation

- Check if Batch run id or mis date or User name is null. If Yes, then write message in **FSI_MESSAGE** log and exit.
- Check if the given user name exists in **FSI_M_USER_ACTIVE_TIME_BUCKETS** table. If user does not exist, then write error message in **FSI_MESSAGE** table and exit.
- The time bucket mapped to user in **FSI_M_USER_ACTIVE_TIME_BUCKETS** table should be present in **FSI_INCOME_SIMULATION_BUCKETS** table. If time bucket is not present in **fsi_income_simulation_buckets** table, then write error message in **fsi_message** table and exit.
- Only financial elements corresponding to Income Simulation Buckets (ISB), Liquidity risk (LR) and Interest Rate Risk (IRR) will be processed.
- If process id is given as parameter, then run the loader program for the given process id. If process id is not given, then the loader program will be run for all the process ids mapped in the set up table with status 'N'.
- Verify that the given process is present in **FSI_ALM_DETERMINISTIC_PROCESS** table and **FSI_M_ALM_PROCESS** table. If not present, then write error message in **fsi_message** table and exit.
- Check if the given process id mapped in **fsi_cash_flow_loader_setup** table is of status 'N'.
- Cash-flow date of each record must correspond correctly to valid time bucket dates in **FSI_TIME_BKT_ISB** and/or **FSI_TIME_BKT_LR_IRR_DATES** tables. However if this check fails, the user will be informed of the improper records through an error message (see [Exception 9](#)).
- Check if the set up table is mapped correctly or not.
 - Check if the given process id mapped to a scenario is present in the **stg_account_cash_flow** table.
 - Check if the entire scenario mapped in the set up table with a functional currency value has its base currency present.
- Check if 'consolidated_output_flag' in the **FSI_ALM_DETERMINISTIC_PROCESS** table is 1. If yes, write to consolidated tables.
- Verify that all the dimension ids given in the **stg_account_cash_flow** table is valid and present in the respective dimension tables.

- In the case of instrument level cash flows, the following conditions should be satisfied to proceed:
In the setup_master table for the V_COMPONENT_CODE =123, the v_component_value should be either 0 or 1 which indicates account number or identity number is given in the STG_ACCOUNT_CASH_FLOW table, respectively.

Identity code and id_number/account number should be present in the stg_account_cash_flow table.

5.12.3 Executing Cash Flow Loader

The user can execute this Cashflow Loader from either SQL*Plus or from within a PL/SQL block or from the Batch Maintenance window within OFSAAI framework.

5.12.3.1 Method 1

Cash Flow Loader can be executed directly from SQL Plus.

1. User must login to the database using schema user id and password. The procedure requires 4 parameters:

As of Date (mis_date)

User Name: Should be present in fsi_m_user_active_time_buckets table

Process id

Batch Execution Identifier (batch_run_id)

```
declare result number :=0;begin result := fn_cash_flow_loader(batch_run_id
=> :batch_run_id, mis_date => :mis_date, p_user_name => :p_user_name,
p_process_sys_id => :p_process_sys_id); if result = 0 then
dbms_output.put_line('Cash Flow Loader Failed'); else
dbms_output.put_line('Cash Flow Loader Succesfully completed'); end; \
```

where

BATCH_RUN_ID is any string to identify the executed batch.

mis_date in the format YYYYMMDD.

P_USER_NAME: The user name present in FSI_M_USER_ACTIVE_TIME_BUCKETS table.

P_PROCESS_SYS_ID can be null or can have value to process specific process id.

2. Case 1. When Process id is null:

```
declare result number :=0;begin result := fn_cash_flow_loader('INFODOM_
CASH_FLOW_LOADER', '20100419', 'ALMUSER', NULL);if result = 0
thendbms_output.put_line('Cash Flow Loader
Failed');elsedbms_output.put_line('Cash Flow Loader Succesfully
completed');end;
```

3. Case 2. When Process id is not null:

```
declare result number :=0;begin result := fn_cash_flow_loader('INFODOM_
CASH_FLOW_LOADER', '20100419', 'ALMUSER', 120003);if result = 0
thendbms_output.put_line('Cash Flow Loader
Failed');elsedbms_output.put_line('Cash Flow Loader Succesfully
completed');end;
```

Figure 5-2 SETUP_MASTER Sample Data

V_COMPONENT_CODE	V_COMPONENT_DESC	V_COMPONENT_VALUE
123	Cash Flow Loader	1

For instrument cash flows, the V_COMPONENT_VALUE should be either 1 or 0. If the value is '1' then Identity code and Account number should be populated in the stg_account_cash_flows table . If the Value is '0' then Identity code and ID Number should be populated in the stg_account_cash_flows table.

Figure 5-3 FSI_CASH_FLOW_LOADER_SETUP Sample Data

RUN_DATE	PROCESS_ID	SCENARIO_NUM	STATUS
12/10/2001	120003	1	N
12/10/2001	120123	1	Y

Note

Only the process id with Status 'N' will process while executing the loader program. After the successful execution of the loader program the value in the STATUS columns will be changed to 'Y' in FSI_CASH_FLOW_LOADER_SETUP for the process id.

Figure 5-4 FSI_M_ALM_PROCESS Sample Data

FM_PROCESS_SYS_ID	ALM_PROCESS_TYPE_CD	FILTER_SYS_ID	FILTER_TYPE	PREPAY_SYS_ID	PROD_CHAR_ID	REPORTING_CURRENCY
120003	1	0	0	0	0	USD
120123	1	0	0	0	0	USD

Figure 5-5 FSI_M_ALM_PROCESS Sample Data

REPORTING_CURRENCY_CD	STATUS	TRANSACTION_SYS_ID	ALT_SOURCE_FLG	BI_TRANSFORM_STATUS
USD	0	0	0	
USD	0	0	0	

Figure 5-6 STG_ACCOUNT_CASH_FLOWS Sample Data

V_ACCOUNT_NUMBER	FIC_MIS_DATE	N_ACCT	N_ACCOU	N_SCENARIO_NO	N_CURRENCY	V_FINANCIAL_ELE	V_DCY	N_CASH_FLOW_SEQL	D_CASH_FLOW_Dt	V_DATA_ORIGI	V_CASH_FLOW_TYPE
ACCT12	12/31/1994	11	0	1	1	GAPRNOFF	USD	34	10/25/1997	SRC	
ACCT12	12/31/1994	11	0	1	1	GAPRNOFF	USD	35	11/25/1997	SRC	
ACCT12	12/31/1994	11	0	1	1	GAPRNOFF	USD	36	12/25/1997	SRC	
ACCT12	12/31/1994	11	0	1	1	GAPRNOFF	USD	37	1/25/1998	SRC	
ACCT12	12/31/1994	11	0	1	1	GAPRNOFF	USD	38	2/25/1998	SRC	
ACCT12	12/31/1994	11	0	1	1	GAPRNOFF	USD	39	3/25/1998	SRC	
ACCT12	12/31/1994	11	0	1	1	GAPRNOFF	USD	40	4/25/1998	SRC	
ACCT12	12/31/1994	11	0	1	1	GAPRNOFF	USD	41	5/25/1998	SRC	
ACCT12	12/31/1994	11	0	1	1	GAPRNOFF	USD	42	6/25/1998	SRC	
ACCT12	12/31/1994	11	0	1	1	GAPRNOFF	USD	43	7/25/1998	SRC	

Instrument Cashflow Data

Figure 5-7 STG_ACCOUNT_CASH_FLOWS Sample Data

V_CASH_FLOW_TYPE	N_CASH_FLOW_AMOUNT	V_ORG_UNIT_CODE	V_PROD_CODE	N_INSTRUMENT_TYPE_CD	V_GL_ACCOUNT_CODE	V_COMMON_COA_CD
...	11000.350	ORG1	...	PRDD1	...	120
...	11000.350	ORG1	...	PRDD1	...	120
...	11000.350	ORG1	...	PRDD1	...	120
...	11000.350	ORG1	...	PRDD1	...	120
...	11000.350	ORG1	...	PRDD1	...	120
...	11000.350	ORG1	...	PRDD1	...	120
...	11000.350	ORG1	...	PRDD1	...	120
...	11000.350	ORG1	...	PRDD1	...	120
...	11000.350	ORG1	...	PRDD1	...	120
...	11000.350	ORG1	...	PRDD1	...	120

Figure 5-8 Aggregate Data Sample Data

V_ACCOUNT_NUMBER	FIC_MIS_DATE	N_ACCT	N_ACCOU	N_SCENARIO_NO	N_CURRENCY	V_FINANCIAL_ELE	V_DCY	N_CASH_FLOW_SEQ	D_CASH_FLOW_DF	V_DATA_ORIG	V_CASH_FLOW_TYPE	N_CASH
0	12/31/2001	*	0	0	3	1 GAPRUNOFF	USD	1	1/31/2001	* SRC2
0	12/31/2002	*	0	0	3	1 GAPRUNOFF	USD	2	4/30/2001	* SRC2
0	12/31/2003	*	0	0	3	1 GAPRUNOFF	USD	3	7/31/2001	* SRC2
0	12/31/2004	*	0	0	3	1 GAPRUNOFF	USD	4	10/31/2001	* SRC2
0	12/31/2005	*	0	0	3	1 GAPRUNOFF	USD	5	1/31/2002	* SRC2
0	12/31/2006	*	0	0	3	1 GAPRUNOFF	USD	6	4/30/2002	* SRC2
0	12/31/2007	*	0	0	3	2 GAPRUNOFF	USD	7	7/31/2002	* SRC2

Figure 5-9 Aggregate Data Sample Data

SH_FLOW_AMOUNT	V_ORG_UNIT_CODE	V_PROD_CODE	N_INSTRUMENT_TYPE_CD	V_GL_ACCOUNT_CODE	V_COMMON_COA_CD
100834348.3	ORG2	...	PRDD2
120833421.3	ORG2	...	PRDD2
91038411.3	ORG2	...	PRDD2
85128473.5	ORG2	...	PRDD2
13048274.34	ORG2	...	PRDD2
13700688.06	ORG2	...	PRDD2
13974701.82	ORG2	...	PRDD2

5.12.3.2 Method 2

To execute the Cash Flow Loader from OFSAAI Batch Maintenance, a seeded Batch is provided.

1. Execution Steps Select <INFODOM>_CASH_FLOW_LOADER as the Batch ID and Cash Flow Loader is the description of the batch.

Figure 5-10 Batch Maintenance

Batch Maintenance

Batch ID Like: OFSALMINFO_cash Batch Description Like:

Module: Last Modification Date: Between And

Batch Name: Add View Edit Delete

Batch ID	Batch Description	Batch Edit/Non Edit
<input type="checkbox"/> OFSALMINFO_CASH_FLOW_LOADER	Cash Flow Loader	E

Page 1 of 1 (1-1 of 1 items) K < > X Records Per Page 15

Task Details

Task ID	Task Description	Metadata Value	Component ID	Precedence
No data found				

Page 0 of 0 (0-0 of 0 items) K < > X Records Per Page 0

- The batch has a single task.

Figure 5-11 Editing Batch

Batch Maintenance

Batch ID Like: OFSALMINFO_cash Batch Description Like:

Module: Last Modification Date: Between And

Batch Name: Add View **Edit** Delete

Batch ID	Batch Description	Batch Edit/Non Edit
<input checked="" type="checkbox"/> OFSALMINFO_CASH_FLOW_LOADER	Cash Flow Loader	E

Page 1 of 1 (1-1 of 1 items) K < > X Records Per Page 15

Task Details

Task ID	Task Description	Metadata Value	Component ID	Precedence
<input checked="" type="checkbox"/> Task1	Cash Flow Loader	CashFlow.Loader	TRANSFORM DATA	

Page 1 of 1 (1-1 of 1 items) K < > X Records Per Page 15

- Edit the task. If the user wants to load all the process ids given in the FSI_CASH_FLOW_LOADER_SETUP table for the given as of date, then Process_id parameter should be null.
- Specify the following parameters:
 - Data store Type:** Select appropriate data store from list
 - Data store Name:** Select appropriate name from the list
 - IP address:** Select the IP address from the list
 - Rule Name:** CashFlowLoader
- If the user wants to process the specific process id mentioned in the FSI_CASH_FLOW_LOADER_SETUP table for the given as of date, then the process_id parameter should be given.
- Specify the following parameters:
 - Data store Type:** Select appropriate data store from list
 - Data store Name:** Select appropriate name from the list
 - IP address:** Select the IP address from the list
 - Rule Name:** CashFlowLoader

Figure 5-12 Task Definition

Task Definition

Save Close

Task ID: Task1 Description: Cash Flow Loader

Components: TRANSFORM DATA

Dynamic Parameters List

Property	Value
Datastore Type	EDW
Datastore Name	OFSALMINFO
Primary IP For Runtime Processes	10.184.156.158
Rule Name	CashFlowLoader
Parameter List	NULL

Audit Panel

Created By: ALMUSER Creation Date: 17 may 2018 13:36:21
Last modified by: ALMUSER Last Modification Date: 17 may 2018 13:36:21

7. Save the batch.
8. Execute the Batch defined for the required **As of Date**.

5.12.4 Exception Messages

All the exceptions will be logged in FSI_MESSAGE_LOG table. Cash Flow Loader program can raise the following exceptions:

- **Exception 1**
Load fails: Cannot pass null for the parameter As of Date, User name, Batch run id
As of date, User Name and Batch run id cannot be passed as null.
- **Exception 2**
Load Fails: User name does not exist in the Table fsi_m_user_active_time_buckets
User name in the Parameter should be mapped with an active time bucket in the fsi_m_user_active_time_buckets table.
- **Exception 3**
Load Fails: Data for the selected As of date does not exist in stg_account_cash_flows Table
Stage Account Cash Flow does not have data for the given As Of Date.
- **Exception 4**
Load Fails: Time bucket sys id is not present in fsi_income_simulation_buckets
Active time bucket mapped to the user name should be present in the Fsi_Income_Simulation_Bucket table.

- **Exception 5**
Load Fails: dates calculation failed

Date Calculation for Fsi_Time_Bkt_Isb and Fsi_Time_Bkt_Lr_Irr_Dates failed for the given time bucket sys id.
- **Exception 6**
Load Fails: The process id process_sys_id does not exist in the Table fsi_m_alm_process

Process id which is either passed as parameter or picked up from the Fsi_Cash_Flow_loader_Setup table should be present in the fsi_m_alm_process table.
- **Exception 7**
Load Fails. The process id process_sys_id does not exist in the Table Fsi_alm_deterministic_process

Process id which is either passed as parameter or picked up from the Fsi_Cash_Flow_loader_Setup should be present in the Fsi_alm_deterministic_process table.
- **Exception 8**
Load Fails. The process id process_sys_id does not exists in the table fsi_cash_flow_loader_setup or the status is not set to N for the process

Process id which is either passed as parameter, either does not exist in the Fsi_Cash_Flow_loader_Setup table or the status is not 'N'.
- **Exception 9**
Date-check failed: Certain cash-flow dates in "STG_ACCOUNT_CASH_FLOWS" for current process are out of range of buckets defined in FSI_TIME_BKT_ISB_DATES/ FSI_TIME_BKT_LR_IRR_DATES table(s). For details, run query: <<QUERY>>

This error is generated when the cash-flow dates in 'STG_ACCOUNT_CASH_FLOWS' are out of the time-bucket date ranges defined in either FSI_TIME_BKT_ISB_DATES or FSI_TIME_BKT_LR_IRR_DATES.

In the actual error message (logged in FSI_MESSAGE_LOG), <<QUERY>> is replaced by a SQL query that the user can copy and execute on the schema in which the batch was run. This will provide the user with an output of the problematic records from staging area.
- **Exception 10**
Load Fail: The Data for the N_SCENARIO_NO mapped to the process process_sys_id does not exist in the STG_ACCOUNT_CASH_FLOWS table

Scenario Number mismatch for the Fsi_Cash_Flow_Loader and Stg_Account_Cash_Flow
- **Exception 11**
Load Fail: Does not have base currency

Base currency should be present in Stg_Account_Cash_Flow.
- **Exception 12**
CONSOLIDATED_OUTPUT_FLG in fsi_alm_deterministic_process table is 1 but no data for n_currency_type_cd in STG_ACCOUNT_CASH_FLOWS table

Consolidated Flag for the process id is set to 1 but n_currency_type_cd in Stg_Account_Cash_Flow is not set.
- **Exception 13**
Load Fail: For the N_SCENARIO_NO mapped in the set up table, the dimension code given is incorrect

Dimensions present in Stg_Account_Cash_Flow are not present in the corresponding dimension tables.
- **Exception 14**

All the account numbers are not present in the STG_ACCOUNT_CASH_FLOWS table
Records in Stg_Account_Cash_Flow do not have account number populated.

- **Exception 15**
All the Identity codes are not present in the STG_ACCOUNT_CASH_FLOWS table
Records in Stg_Account_Cash_Flow do not have identity code populated.
- **Exception 16**
All the Id Numbers are not present in the STG_ACCOUNT_CASH_FLOWS table
Records in Stg_Account_Cash_Flow do not have identity number populated.
- **Exception 17**
Instrument type code given for the process l_process_sys_id is wrong

Instrument code present in the Stage table is not mapped in
FSI_INSTRUMENT_TYPE_MLS
- **Exception 18**
No Data in the instrument table for the given FIC MIS DATE. Loading data to
FSI_O_RESULT_MASTER failed

Instrument table corresponding to instrument code in Stg_Account_Cash_Flow does not
have data for the given As of date.

5.13 Pricing Management Transfer Rate Population

This function populates FSI_M_PROD_TRANSFER_RATE table from
FSI_PM_GENERATED_INSTRMNTS table for particular Effective date.

After executing this procedure, you should query FSI_M_PROD_TRANSFER_RATE table.

Executing the POPULATE_PM_TRANS_RATE_TABLE (earlier known as
POPULATE_TPOL_TRANS_RATE) Procedure

You can execute this procedure either from SQL*Plus or from within a PL/SQL block or from
Batch Maintenance window within OFSAAI framework.

To run the procedure from SQL*Plus, login to SQL*Plus as the Schema Owner.

The procedure requires the following 6 parameters:

- **Batch Id (Batch_Id):** can be used to see the log of the procedure executed.
- **Misdate (Mis_date):** the date for which batch is run.
- **Run Id (p_v_run_id):** Unique Run ID for the run.
- **Process Id (p_v_process_id):** Unique Process ID for the batch.
- **Run Execution Id (p_v_run_execution_id):** Unique Run Execution Id for the Run.
- **Run skkey (p_n_run_skkey):** Unique run skkey generated by the run.

The syntax for calling the procedure is:

```
Declare output number; Begin Output:= POPULATE_PM_TRANS_RATE_TABLE (Batch_Id
varchar2, Mis_date varchar2, p_v_run_id varchar2, p_v_process_id varchar2,
p_v_run_execution_id varchar2, p_n_run_skkey varchar2); End;
```

Mis_date should be passed as 'YYYYMMDD' format.

An example of running the function from SQL*Plus is as follows:

```
SQL> var output number; SQL> execute: output:=
POPULATE_PM_TRANS_RATE_TABLE('Batch_Id', '20100131,' $RUNID=1306182237482',
'$PHID=1228363751510', '$EXEID=RQEXE016','$RUNSK=99');
```

To execute the stored procedure from within a PL/SQL block or procedure, see the example that follows.

```
SQL> declare output number; begin Output:= POPULATE_PM_TRANS_RATE_TABLE
('Batch_Id','Mis_date', 'p_v_run_id','p_v_process_id','p_v_run_execution_id',
p_n_run_skey'); End; /
```

To execute the procedure from OFSAAI Batch Maintenance, create a new Batch with the Task as TRANSFORM DATA and specify the following parameters for the task:

- **Datastore Type** : Select appropriate datastore from the list
- **Datastore Name** : Select appropriate name from the list
- **IP address** : Select the IP address from the list
- **Rule Name** : POPULATE_PM_TRANS_RATE_TABLE

BATCHID and MISDATE will be passed explicitly in Batch Maintenance

5.14 ALMBI Transformation

ALM_BI_TRANSFORMATION data definition transforms the Asset Liability Management (ALM) processing results of an executed ALM process to ALMBI fact tables.

You can execute this procedure either from SQL*Plus or from within a PL/SQL block or from Batch Maintenance window within OFSAAI framework.

1. To run the procedure from SQL*Plus, login to SQL*Plus as the Atomic Schema Owner. The syntax for calling the procedure is

```
set serveroutput on
```

```
DECLARE num number;
```

```
Begin num :=
```

```
FN_ALM_BI_TRANSFORMATION('<P_BATCH_RUN_ID>', '<P_AS_OF_DATE>', '<PID>', '<P_RE_RUN_F
LAG>', '<B_LIMIT_FLAG>'); END;
```

FN_ALM_BI_TRANSFORMATION function requires below parameters:

P_BATCH_RUN_ID is the Batch Run ID typically in the format
<INFODOM>_TRANSFORMALMRESULT_<ASOFDATE AS YYYYMMDD>_X.

P_AS_OF_DATE is the As of Date in the format YYYYMMDD. PID Pass the ALM Process Sys ID for which the transformation has to be done.

P_RE_RUN_FLAG is a re-run flag.

'Y': Yes (This means that the transformation was already done and the user is trying to redo the transformation once again for the ALM process).

'N': No (This means that the user is executing the transformation for the first time for the ALM process).

Run below query to check if Transformation has been run before.

BI_TRANSFORM_STATUS = 1 means it was run before, null means new run.

```
select * from fsi_process_run_history where PROCESS_SYS_ID = <PID>;
```

B_LIMIT_FLG is limit flag. Specify 'Y' or 'N'. The limit should be prepared only if the flag is 'Y'.

For example:

```
set serveroutput on DECLARE num number;
```

```
Begin num :=
```

```
FN_ALM_BI_TRANSFORMATION('OFSALMINFO_TRANSFORMALMRESULT_20230430_3','20230430','7  
12902','Y','N'); END;
```

Note

The values for parameters PID and p_re_run_flag has to be entered in the Parameter List during the batch definition.

Figure 5-13 Example

Property	Value
Datastore Type	EDW
Datastore Name	OFSALMINFO
Primary IP For Runtime Processes	[REDACTED]
Rule Name	ALM_BI_TRANSFORMATION
Parameter List	5022308,'Y','N'

- If the user is trying to do transformation of ALM process 200009 for the first time, then the values that must be entered in the Parameter List are 200009, 'N'.
- If the user is trying to do transformation of ALM process 200011, for which he had already done the transformation, then the values that must be entered in the Parameter List are 200011, 'Y'.

5.15 Hierarchy Transformation

Hierarchy Flattening Transformation is used to move the hierarchy data from the parent child storage structure in EPM AMHM (Attribute, Member and Hierarchy Management) model to a level based storage structure in OFSAA BI applications. In EPM AMHM model, hierarchy data for any hierarchy created on seeded or user defined dimensions using the AMHM is stored within hierarchy tables of respective dimensions. This is moved to the REV_HIER_FLATTENED table in OFSAA BI applications after flattening by the Hierarchy flattening process.

batch_hierTransformation is a seeded Data Transformation program installed as part of the OFSAA BI applications installers.

Note

Refer to Support Note 1586342.1, if Hierarchy Filter is not reflecting correctly after making the changes to underlying Hierarchy.

5.15.1 Executing the Hierarchy Flattening Transformation

You can execute this procedure from SQL Plus/PLSQL/Batch Maintenance window within OFSAAI framework: Using SQL Plus/PLSQL and Using OFSAAI Batch Maintenance

- Using SQL Plus/PLSQL, execute below:

```
function rev_batchHierFlatten(batch_run_id varchar2, mis_date varchar2,
pDimensionId varchar2, pHierarchyId varchar2, )
```

- Function Name: rev_batchHierFlatten
- Parameters: batch_run_id, mis_date, pDimensionId, pHierarchyId
 - batch_run_id: It is the batch run id. Batch Run ID value is passed from the Batch execution UI. Therefore, it is not required to define it as a parameter value in Batch Maintenance.
 - mis_date: This parameter value is passed from the Batch execution UI. Therefore, it is not required to define it as a parameter value in Batch Maintenance. Follow the date format, YYYYMMDD
 - pDimensionId- Enter the Dimension id . To find dimension id, execute the following query in database to find the value and use the value in dimension id column for the dimension name / description to be processed:


```
Select b.dimension_id,t.dimension_name,t.description from
rev_dimensions_b b inner join rev_dimensions_tl t on b.dimension_id =
t.dimension_id and t.dimension_name like '<dimension name>'
```

 Replace <dimension name> in the preceding query with the Dimension Name you find in the UI (Maintenance > Dimension Management) for the dimension on which the Hierarchy you want to flatten is configured.
 - pHierarchyId: Enter Hierarchy id. If all the hierarchies belonging to a dimension are to be processed then, provide NULL as the parameter value. Else, provide the System Identifier of the hierarchy that needs to be transformed.

Execute the following query in database if only a single hierarchy is to be processed and use the value in hierarchy_id column as parameter for the hierarchy to be processed:

```
select b.object_definition_id , short_desc,long_desc from
fsi_m_object_definition_b b inner join fsi_m_object_definition_tl t on
b.object_definition_id = t.object_definition_id and b.id_type = 5
```
 - If all the hierarchies for GL Account dimension must be processed, the parameter list should be given as follows (where '2' is the dimension id for the seeded dimension GL Account):


```
'2',null
```
 - If a particular hierarchy with code 1000018112 must be processed (you can obtain this code by executing the preceding query in the database), the parameter list should be given as follows:


```
'2', '1000018112'
```

```
SQL ExampleSQL> var fn_return_val number;SQL> execute :fn_return_val:=
rev_batchHierFlatten ('Batch1 ', '20091231 ', '2 ', '1000018112');SQL>
print fn_return_val
```

PLSQL Example

```
DECLARE fn_return_val number := null;BEGIN fn_return_val :=
rev_batchHierFlatten('Batch1', '20091231', '2', 1000018112'); IF
fn_return_val = 1 THEN Dbms_output.put_line('Execution status of
batchHierFlatten is' ||fn_return_val || ' --Successful'); ELSIF
fn_return_val = 0 THEN Dbms_output.put_line('Execution status of
batchHierFlatten is' ||fn_return_val || ' --FAILURE'); END IF;EXCEPTION
WHEN OTHERS THEN Dbms_output.put_line('Execution status of
batchHierFlatten is' || SQLCODE || '-' || SQLERRM); END;
```

On successful execution of `rev_batchHierFlatten` function in Database, value returned will be 1 or 0. 1 indicates successful execution and 0 indicates failure in execution. This function will be present in Atomic Schema.

2. To execute the procedure from OFSAI Batch Maintenance, run the following batch and specify the following parameters:
 - **Datastore Type:** Select appropriate datastore from the list
 - **Datastore Name:** Select appropriate name from the list
 - **IP address:** Select the IP address from the list
 - **Rule Name:** batch_hierTransformation
 - **Parameter List:** Dimension ID, Hierarchy ID

5.16 Dim Dates Population

`DIM_DATES_POPULATION` is a seeded Data Transformation which is installed as part of the OFSAA BI applications installers. Time dimension population transformation is used to populate the `dim_dates` table with values between two dates specified by the user.

Note

During data transformation, the data will be loaded into FISCAL columns by reading the start date/end date information from `DIM_FINANCIAL_YEAR` table. Users can enter data manually into `DIM_FINANCIAL_YEAR` table.

5.17 Fact Ledger Stat Transformation

`FSI_LEDGER_STAT_TRM` is a seeded Data Transformation which is installed as part of the OFSAA BI applications installers. Fact Ledger Population transformation is used to populate the `FCT_LEDGER_STAT` table from the Profitability `LEDGER_STAT` table. Database function `LEDGER_STAT_TRM` is called by the function `FSI_LEDGER_STAT_TRM`.

5.18 Financial Element Dimension Population

Financial Element Dimension Population involves populating custom Financial Elements created into `DIM_FINANCIAL_ELEMENT` table from `DIM_FINANCIAL_ELEMENT_B` table.

Topics:

- [Prerequisites](#)
- [Tables Used by the Financial_Elem_Update Transformation](#)
- [Executing the Financial_Elem_Update Transformation](#)

5.18.1 Prerequisites

All the post install steps mentioned in the [Oracle Financial Services Analytical Applications Infrastructure \(OFSAAI\) Installation and Configuration guide](#) and [Oracle Financial Services Profitability Management User Guide](#).

- Application User must be mapped to a role that has seeded batch execution function (BATPRO).
- Seeded and Custom Financial Elements are required to be available in DIM_FINANCIAL_ELEMENTS_B, DIM_FINANCIAL_ELEMENTS_TL tables.
- Before executing a batch check if the following servers are running on the application server (For more information on how to check if the services are up and on and how to start the services if you find them not running, refer to any OFSAA BI User Guide- for example, [Oracle Financial Services Analytical Applications Infrastructure User Guide](#)).
 - Iccserver
 - Router
 - AM Server
 - Messageserver
 - Olapdataserver
- Batches will have to be created for executing the function.

5.18.2 Tables Used by the Financial_Elem_Update Transformation

DIM_FINANCIAL_ELEMENT: This table stores the seeded and custom Financial Elements.

For more details on viewing the structure of the tables, see OFSAA EPM Erwin Data Model.

5.18.3 Executing the Financial_Elem_Update Transformation

To execute the function from OFSAAI Information Command Center (ICC) frame work, create a batch by performing the following steps:

Note

For a more comprehensive coverage of configuration and execution of a batch, see Oracle Financial Services Analytical Applications Infrastructure User Guide.

1. From the Home menu, select Operations, then select **Batch Maintenance**.
2. Click **New Batch (+)** and enter the Batch Name and description.
3. Click **Save**.
4. Select the Batch you have created in the earlier step by clicking on the checkbox in the Batch Name container.

5. Click **New Task (+)**.
6. Enter the Task ID and Description. Select Transform Data, from the components list.
7. Select the following from the Dynamic Parameters List and then click **Save**:
 - **Datastore Type:** Select appropriate datastore from the list
 - **Datastore Name:** Select appropriate name from the list
 - **IP address:** Select the IP address from the list
 - **Rule Name:** Select Financial_Elem_Update from the list of all available transformations. (This is a seeded Data Transformation which is installed as part of the OFSAA BI applications installers. If you don't see this in the list, contact Oracle support)
 - **Parameter List:** OFSAAI Application User Name (See the following for details on Parameter list).
 - **Application User Name:** This is the OFSAAI application user name which the transformation uses for inserting in DIM_FINANCIAL_ELEMENT table. Sample parameter for this task is 'APPUSER'
8. Execute the batch.

The function can also be executed directly on the database through SQLPLUS. Details are:

 - **Function Name :** fn_dim_financial_elem_update
 - **Parameters :** pBatch_Id, pas_of_date, appuser_name
 - **Sample parameter values :** 'Batch1','20091231', 'APPUSER'

5.18.4 Checking the Execution Status

The status of execution can be monitored using the Batch Monitor screen.

Note

For a more comprehensive coverage of configuration & execution of a batch, see Oracle Financial Services Analytical Applications Infrastructure User Guide.

The status messages in batch monitor are :

N: Not Started

O: On Going

F: Failure

S: Success

The Event Log window in Batch Monitor provides logs for execution with the top row being the most recent. If there is any error during execution, it will get listed here. Even if you see Successful as the status in Batch Monitor it is advisable to go through the Event Log and re-check if there are any errors. The execution log can be accessed on the application server by going to the following directory `$(FIC_DB_HOME)/log/date`. The file name will have the batch execution id.

The database level operations log can be accessed by querying the `FSI_MESSAGE_LOG` table. The batch run id column can be filtered for identifying the relevant log.

Check the `.profile` file in the installation home if you are not able to find the paths mentioned earlier.

Following messages will be available in the `FSI_MESSAGE_LOG` table after executing the batch.

```
Starting to update DIM_FINANCIAL_ELEMENT
```

```
Please provide application user name (In case OFSAAI application user name is not passed as a parameter).
```

```
Successfully Completed.
```

After successful execution of the batch, user can verify custom financial element present in `DIM_FINANCIAL_ELEMENT` table.

5.19 Payment Pattern Loader

The Payment Pattern Loader provides the ability to load bulk payment pattern definitions through a back end procedure. This Loader reads the stage table data, does data quality checks on the same, and load them into `FSI_PAYMENT_PATTERN` and `FSI_PAYMENT_PATTERN_EVENT` tables, if the stage table data is valid.

Following is the stage table to input the payment pattern:

Table 5-10 STG_PAYMENT_PATTERN

Column Name	Column Datatype	Column Null Option	Column Is PK	Column Comment
V_AMRT_TYPE	VARCHAR2(5)	NOT NULL	Yes	Amortization code between 1000 to 69999. Patterns between this range will be consider for payment processing
N_EVENT_ID	NUMBER(5,0)	NOT NULL	Yes	Event Identity Number
N_SPLIT_ID	NUMBER(5,0)	NOT NULL	Yes	Holds number of patterns with in split pattern
V_PATTERN_TYPE	VARCHAR2(40)	NOT NULL	Yes	List of values could be Absolute, Relative, Split
V_TERM_TYPE	VARCHAR2(40)	NOT NULL	Yes	List of values could be Principal and Interest, Principal Only, Interest Only, Level Principal, Final Principal & Interest, Other
V_AMRT_TYPE_D ESC	VARCHAR2(255)	NOT NULL	No	Alpha numeric value
N_PCT_VALUE	NUMBER(8,4)	NULL	No	Percentage applied to each pattern in case of split pattern type

Table 5-10 (Cont.) STG_PAYMENT_PATTERN

V_PAYMENT_EVE NT_MONTH	VARCHAR2(20)	NULL	No	Month in which payment event should occur
N_PAYMENT_EVE NT_DAY	NUMBER(2)	NULL	No	Number of Days Payment type
N_PAYMENT_EVE NT_FREQ	NUMBER(5,0)	NULL	No	Number of times payment event should occur
V_PAYMENT_EVE NT_FREQ_MULT	VARCHAR2(40)	NULL	No	List of values could be Days,Months,Years
N_PAYMENT_EVE NT_REPEAT_VAL UE	NUMBER(5,0)	NULL	No	Holds number of times payment frequency should repeat
N_AMOUNT	NUMBER(14,2)	NULL	No	Amount
V_AMOUNT_TYPE	VARCHAR2(40)	NULL	No	List of values could be % of original Payment,% of current payment,absolute value
V_PAYMENT_TYP E	VARCHAR2(30)	NULL	No	List of values could be Conventional, Level principal, Non-amortizing

The loader program performs the following data quality checks:

- The following values will be checked against the relevant look tables as mentioned
 - Pattern Type: FSI_PATTERN_TYPE_MLS
 - CashFlowType: FSI_PAYMENT_TYPE_MLS (Should accept only 100-Principal and Interest and 300-Interest Only)
 - Month: FSI_MONTHS_MLS
 - Multiplier: FSI_MULTIPLIER_MLS
 - Payment Method: FSI_AMOUNT_TYPE_MLS (For Conventional accept only % of Original Payment, % of Current Payment and Absolute value)
 - Payment Type: FSI_PMT_PATTERN_TYPE_MLS
- While defining any pattern type like relative or absolute, MONTH and DAY combination should be unique
- MONTH and DAY pair should have valid month and day combination, such as January 31 days and February 28 days (Leap year was not considered) and so on.
- If cash flow value is Principal and Interest then N_AMOUNT cannot be blank. If it is Interest only then V_PAYMENT_TYPE and N_AMOUNT should be blank.
- When payment type is a Non-amortizing and Payment pattern is Relative then N_PAYMENT_EVENT_FREQ and N_PAYMENT_EVENT_REPEAT_VALUE should have values which could range between 1 to 9999.

6. One Split pattern can have any number of definition, however the sum of N_PCT_VALUE of all the definition should be 100% and all the payment patterns in the split should be defined.
7. All the following fields should have this validation on place:
 - Day: Positive Integer Number Range from 1 to 31 depends on the month for which day.
 - Percentage: Positive Integer or Decimal Number
 - Frequency: Positive Integer range from 1 to 9999
 - Repeat: Positive Integer range from 1 to 9999
 - Value: Integer numbers from 0 to 9999999999
8. For each payment pattern and payment type combinations fields relevant to that would be populated by the user remaining columns should be populated with default values:
 - Payment Pattern: Absolute
 - Payment Type: Conventional / Level Principal
 - Columns gets populated with user values: Code , Description, Pattern Type, Payment Type, Month, Day, Cash Flow Type, Payment Method, Value, Percentage (in case of Split pattern type)
 - Payment Type: Non amortizing Payment type.
 - Columns gets populated with user values: Code , Description , Pattern Type, Payment Type, Month, Day, Percentage (in case of Split pattern type)
 - Payment Pattern: Relative
 - Payment Type: Conventional / Level Principal
 - Columns gets populated with user values: Code, Description, Pattern Type, Payment Type, Frequency, Multiplier, Repeat, Cash Flow Type, Payment Method, Value, Percentage (in case of Split pattern type)
 - Payment Type: Non amortizing Payment type.
 - Columns gets populated with user values: Code, Description, Pattern Type, Payment Type, Frequency, Multiplier, Repeat, Percentage (in case of Split pattern type).

The loader program defaults values for each column in case values provided by user are not relevant for the pattern and payment patterns they defined.

5.19.1 Executing the Payment Pattern Loader

There are two ways to execute the Payment Pattern Loader procedure: Running Procedure Using SQL*Plus and Payment Pattern Loader Procedure Using OFSAAI Batch Maintenance

1. To run the procedure from SQL*Plus, login to SQL*Plus as the Schema Owner:

```
SQLPLUS > declare
result number;beginresult := fn_paymentpattern
('Payment_Pattern_20121212_1', '20121212');
end;
/
```

BATCH_RUN_ID is any string to identify the executed batch.

AS_OF_DATE is in the format YYYYMMDD

2. To execute Payment Pattern Loader from OFSAAI Batch Maintenance, a seeded Batch is provided.

The batch parameters are:

- **Datastore Type:** Select the appropriate datastore from list.
- **Datastore Name:** Select the appropriate name from the list.
- **IP address:** Select the IP address from the list.
- **Rule Name:** fn_paymentpattern

5.19.2 Exception Messages

Below are the list of error messages which can be viewed in view log from UI or FSI_MESSAGE_LOG table from back end filtering for the given batch id. On successful completion of each task messages gets into log table.

In the event of failure, following are the list of errors that may occur during the execution:

- **Exception 1:** PATTERN TYPE IS NOT MATCHING THE LIST OF VALUES
- **Exception 2:** TERM TYPE IS NOT MATCHING THE LIST OF VALUES
- **Exception 3:** PAYMENT MULTIPLIER FREQ IS NOT MATCHING THE LIST OF VALUES
- **Exception 4:** EVENT MONTH IS NOT MATCHING THE LIST OF VALUES
- **Exception 5:** PMT PATTERN IS NOT MATCHING THE LIST OF VALUES
- **Exception 6:** AMOUNT TYPE IS NOT MATCHING THE LIST OF VALUES
- **Exception 7:** v_amrt_type_cd range between 1000 to 69999
- **Exception 8:** n_payment_event_freq_cd and n_payment_event_repeat_value should have values range between 1 to 9999
- **Exception 9:** N_PCT_VALUE POSITIVE INTEGER OR DECIMAL NUMBER
- **Exception 10:** N_SPLIT_ID POSITIVE NUMBER
- **Exception 11:** If Term Type is PRINCIPAL AND INTEREST, then it should range from 1 to 999999999
- **Exception 12:** If Term Type is INTEREST ONLY AMOUNT and AMOUNT TYPE CD should be blank
- **Exception 13:** ERR while deleting stg_payment_pattern unwanted records
- **Exception 14:** Month and Day pair should have valid combination. Like January 31 days
- **Exception 15:** Pct value should be 100 in case of split pattern for other patterns it should be =0
- **Exception 16:** Error during percentage check
- **Exception 17:** Error during inserting

5.20 GAP Limits Loader

This loader will provide the ability to load user defined GAP Limits through a back end procedure.

For more information, see the [ALM User guide](#) on OHC.

5.20.1 GAP Limits Loader Tables

The Loader uses the following staging and target tables:

- `STG_ALM_GAP_LIMIT_DTL` — This staging table contains preliminary user-provided data that will subsequently undergo data quality checks.
- `FSI_ALM_GAP_LIMIT_DTL` — The loader copies limit bucket-sets into this table after quality checks; only bucket-sets that pass quality checks are populated in this table

5.20.2 Executing the Gap Limit loader

There are two ways to execute the Gap Limit Loader procedure: Running Procedure Using SQL*Plus and Gap Limit Loader Procedure Using OFSAAI Batch Maintenance

1. To run the function from SQL*Plus, log in to SQL*Plus as the Schema Owner.

The loader requires two parameters:

- Batch Execution Name
- As Of Date

Syntax:

```
fn_load_fsi_alm_gap_limits (batch_run_id IN VARCHAR2, as_of_date IN VARCHAR2)
```

For example:

```
SQLPLUS > declare
result number;beginresult := fn_load_fsi_alm_gap_limits
('INFODOM_20100405', '20100405');
end;
/
```

2. To execute Gap Limit Loader from OFSAAI Batch Maintenance, a seeded Batch <INFODOM>_GAP_LIMITS_LOADER is provided.

The batch parameters are:

- **Datastore Type:** Select the appropriate datastore from list.
- **Datastore Name:** Select the appropriate name from the list.
- **IP address:** Select the IP address from the list.
- **Rule Name:** `fn_load_fsi_alm_gap_limits`
- **Parameter List:** None

5.20.3 Exception Messages

Below are the list of error messages which can be viewed in view log from UI or `FSI_MESSAGE_LOG` table from back end filtering for the given batch id. On successful completion of each task messages gets into log table.

In the event of failure, following are the list of errors that may occur during the execution:

- **Exception 1:** ALM GAP Limit Loader exited but no records were inserted into FSI table. All GAP limit bucket-sets in the STG table failed data quality checks and/or there was some internal error

- **Exception 2:** Errors recorded in internal memory but could not be logged in FSI_MESSAGE_LOG. Exiting.
Error messages could not be logged
- **Exception 3:** Could not insert records into internal memory. It may be a count mismatch b/w consolidated_records and pk_iter.
Internal processing of limit bucket-sets failed during execution, please contact support
- **Exception 4:** No Records found in STG Table for As_Of_Date: <DATE>
STG table had no records for the selected date of execution of utility
- **Exception 5:** Error in prc_load_fsi_alm_gap_limits: <ERROR MESSAGE>
Unexpected error in execution
- **Exception 6:** Invalid Legal_Entity_Code: <CODE>
Check if the legal entity code matches with valid legal entities in DIM_LEGAL_ENTITY_B (leaf nodes only and should be enabled)
- **Exception 7:** Invalid Org_Unit_Code: <CODE>
Check if the organisation unit code matches with valid codes in DIM_ORG_UNIT_B(leaf nodes only and should be enabled)
- **Exception 8:** Invalid Currency: <CCY>
Please ensure that the currency code for a bucket set is a valid ISO code
- **Exception 9:** Invalid Currency_Type_Code: <CCY_Type>
Check if the currency type is valid and present in FSI_CURRENCY_TYPE_MLS
- **Exception 10:** Invalid Time_Bucket_Name: <BKT_NAME>
Check if the bucket name provided matches valid entries in FSI_TIME_BUCKET_MASTER
- **Exception 11:** Invalid Bucket_Number: <NUM> for Time_Bucket_Name: <BKT_NAME>
Ensure bucket number in DIM_RESULT_BUCKET corresponds to the bucket name of in the row; Names are case insensitive
- **Exception 12:** Invalid Start_Date_Index<INDEX> for Time_Bucket_Name: <BKT_NAME>
The start date index in FSI_LR_IRR_BUCKETS_AUX matches for the given bucket name
- **Exception 13:** Invalid Forecast_Rate_Rule_Name: <FCST_RULE_NAME>
Ensure the forecast rate rule applied corresponds to those present in FSI_M_OBJECT_DEFINITION_TL; Names are case insensitive
- **Exception 14:** Invalid Scenario_Name: <SCEN_NAME> for Forecast_rate_rule_name: <FCST_RULE_NAME>
The scenario applied should correspond to the forecast rate rule being applied
- **Exception 15:** Invalid Repricing_GAP_Measure: <MEASURE>
Repricing gap measure should be one of 'NET REPRICE GAP', 'CUMULATIVE REPRICE GAP'; case insensitive
- **Exception 16:** Eff_End_Date[<DATE>] must be later than or same as AS_OF_DATE[<EXECUTION_DATE>]
Execution date must always be less than the effective end date of a limit bucket-set
- **Exception 17:** Eff_End_Date[<DATE>] must be later than Eff_Start_Date[<DATE>]
The effective end date of a limit-bucket set must be later than its start date
- **Exception 18:** Invalid Limit_Method: <LIMIT_METHOD>
Bucket limit method must be one of 'ABSOLUTE' or 'RELATIVE'; case insensitive
- **Exception 19:** GAP_Limits cannot be negative
Self-explanatory
- **Exception 20:** GAP lwr_limit must be lesser than upr_limit
Self-explanatory

- **Exception 21:** Bucket Continuity Constraint: Current bucket's lower_limit must be previous bucket's upper_limit+1
Self-explanatory
- **Exception 22:** Bucket Limit_Method Mismatch: All buckets in a set must follow the first bucket's limit method
Self-explanatory
- **Exception 23:** For Limit Method: RELATIVE, first bucket's lower_limit must be 0
Self-explanatory
- **Exception 24:** For Limit Method: RELATIVE, last bucket's upper_limit must be 100
Self-explanatory
- **Exception 25:** For Limit Method: RELATIVE, lower_limit must be b/w [0,upper_limit)
The lower limit of a bucket in a bucket-set must start from previous bucket's lower limit+1 or 0 if it is the first bucket in a bucket-set; the upper limit must be greater than lower limit or be exactly 100 if it is the last bucket in the set.
- **Exception 26:** For Limit Method: RELATIVE, upper_limit must be b/w (lower_limit,100]
The lower limit of a bucket in a bucket-set must start from previous bucket's lower limit+1 or 0 if it is the first bucket in a bucket-set; the upper limit must be greater than lower limit or be exactly 100 if it is the last bucket in the set.
- **Exception 27:** Record will be rejected due to error(s) in row(s) indicated by Bkt_num(s): <BAD_BKT_NUMS>
The row itself passes DQ data quality checks but will still be rejected due to errors elsewhere in its limit bucket-set
- **Exception 28:** Error in Data Quality Validator : dq_validator.
DQ validation function failed
- **Exception 29:** Error in bulk_logging, erroneous rows could not be recorded.
Bulk logging of error messages to FSI_MESSAGE_LOG failed; transactions to FSI_* table will be rolled-back
- **Exception 30:** No good records were found in the STG Table.
All GAP limit bucket-sets in the STG table failed data quality checks
- **Exception 31:** Error in target_insert_update: <ENGINE GENERATED ERROR MESSAGE>.
Possible reason: Two or more exact same bucket sets with typographical differences are present(that is, one bucket has v_scenario_name = A_Lim_Bucket and another has v_scenario_name = a_lim_bucket). It is advisable to avoid stray spaces in strings and keep everything in all-caps
- **Exception 32:** Error in target_insert_update
Function for inserting good records to FSI_* table failed
- **Exception 33:** Could not cleanup old records for batch run id: <BATCH_RUN_ID>
Previous error messages in FSI_MESSAGE_LOG for the same BATCH_RUN_ID as run could not be deleted; The execution may have still gone through

5.21 Material Currency Identifier

As per the standardized approach of IRRBB, the loss in economic value of an equity is calculated for each currency with material exposures. Material exposure is defined as those accounting for more than 5% of either banking book assets or liabilities. Utility moves data from FSI_D_<INSTRUMENT TABLE> to FCT_ALM_SIGNIFICANT_CURRENCY.

For more information, see the [ALM User Guide](#).

5.21.1 Material Currency Identifier Tables

Below is the list of Material Currency Identifier tables:

- FSI_D_CREDIT_CARDS
- FSI_D_MUTUAL_FUNDS
- FSI_D_RETIREMENT_ACCOUNTS
- FSI_D_TERM_DEPOSITS
- FSI_D_ASSET_BACK_SEC
- FSI_D_BORROWINGS
- FSI_D_INVESTMENTS
- FSI_D_ANNUITY_CONTRACTS
- FSI_D_CASA
- FSI_D_LOAN_CONTRACTS
- FSI_D_CREDIT_LINES
- FSI_D_GUARANTEES
- FSI_D_MERCHANT_CARDS
- FSI_D_MORTGAGES
- FSI_D_TRUSTS
- FSI_D_CAPFLOORS
- FSI_D_MM_CONTRACTS
- FSI_D_LEASES
- FSI_D_OTHER_SERVICES
- FSI_D_LEDGER_STAT_INSTRUMENT
- FSI_D_LOAN_COMMITMENTS
- FSI_D_FUTURES
- FSI_D_CAPFLOORS
- FSI_D_SWAPS

5.21.2 Executing the Material Currency loader

There are two ways to execute the Material Currency Loader procedure: Running Procedure Using SQL*Plus and Material Currency Loader Procedure Using OFSAAI Batch Maintenance

1. To run the procedure from SQL*Plus, login to SQL*Plus as the Schema Owner:

For example:

```
SQLPLUS > declare
result number;beginresult := fn_signf_currency_loader
('ALMUSER','Y',0.05,'>','USD',200203,'100',6,5011);
end;
/
```

Here,

- **User Name:** ALMUSER is the user Name for which the task is to be executed
 - **Off-Balance Sheet Flag:** Set the Off-balance sheet flag as 'Y' to enable off-balance sheet instruments, 'N' to disable them.
 - **Material Currency Threshold:** .05 is Material Currency Threshold. 05 is the threshold defined by BASEL norms but the parameter is configurable between [0,1], limits inclusive
 - **Comparison Operator:** Is Comparison Operator; valid choices include [>,>=,<,<=]
 - **Reporting Currency Code:** USD is reporting currency ISO code
 - **LE Hierarchy Id (Optional):** Numeric Id of Legal Entity hierarchy to be used must be given. It accepts only a single hierarchy ID. If this parameter is provided, LE Id_list must be either NULL or contain only one Legal Entity ID.
 - **LE Id list (Optional):** If LE Hierarchy Id is not provided, LE Id list can contain either a single Legal Entity ID or multiple IDs in a comma-separated format. Example: '1,2,3,4'. If LE Hierarchy Id is provided, then LE Id list must contain only one value of either node or leaf member present in hierarchy. Example: '1'.
 - **Product Dimension Id (Optional):** Numeric Id of custom key processing dimension to be used.
 - **Common COA Attribute Id (Optional):** Numeric Id of the Common Chart of Account attribute of the dimension given in Product Dimension Id. This will be used to identify asset and liability instruments. If value is given for Product Dimension Id then Common COA Attribute Id is mandatory.
2. To execute Material Currency Loader from OFSAAI Batch Maintenance, a seeded Batch <INFODOM>_MATERIAL_CURR_IDENTIFICATION is provided.

The batch parameters are:

- **Datastore Type:** Select the appropriate datastore from list.
- **Datastore Name:** Select the appropriate name from the list.
- **IP address:** Select the IP address from the list.
- **Rule Name:** fn_signf_currency_loader
- **Parameter List:** User Name, off-Balance Sheet Flag, Material Currency Threshold, Comparison Operator, Reporting Currency Code, LE Hierarchy Id, LE Id list, Product Dimension Id, and Common COA Attribute Id (for description of the configurable paramters, see above)

5.21.3 Exception Messages

During the course of execution, certain exception messages are logged in FSI_MESSAGE_LOG. These can be viewed from the UI or from the back-end by filtering FSI_MESSAGE_LOG for the current 'batch_run_id'. On successful completion of the task, a message indicating success is logged. In the event of execution failure or intermediate errors, messages from the following list can be logged:

- **Exception 1:** Table '<INSTRUMENT_TABLE_NAME>' not found in current schema. This table will be ignored.
The table may have been removed or currently does not exist in the schema
- **Exception 2:** Error during inserting asset records from '<INSTRUMENT_TABLE_NAME>': <ENGINE_GENERATED_MESSAGE>. Table will be ignored.

The columns types of the table may have changed or columns may have been removed

- **Exception 3:** Error in utility/task "ALM Material Currency Identification". Please check if username is valid. In rare cases, it may be an internal DB error.
The username provided either doesnot exist or the user's product dimension preference is not one of (PRODUCT, GENERAL_LEDGER, or COMMON COA). Very rarely it could be an internal mapping error
- **Exception 4:** Error in utility/task "ALM Material Currency Identification"; off-balance-sheet flag must be 'Y' or 'N'.
Illegal user-input parameter
- **Exception 5:** Error in utility/task "ALM Material Currency Identification" during insert/merge operation on '<ISNTRUMENT_TABLE_NAME>'. OPERATION ABORTED.
Internal error during merging of liability records with asset records in the TMP_* table (TMP table stores intermediate records before final consolidation into the FCT_* table)
- **Exception 6:** Error in utility/task "ALM Material Currency Identification"; Please ensure that 1. material threshold is between [0,1]; 2. a valid comparison operator has been provided; 3. Reporting currency is a valid ISO code.
Illegal user-input parameter(s)
- **Exception 7:** Error in utility/task "ALM Material Currency Identification"; Insert operation on FCT_ALM_SIGNIFICANT_CURRENCY failed, OPERATION ABORTED. Probable reason: foreign key violation in FCT_* table on column 'N_ENTITY_SKEY'.
Internal error during insertion of final records in to the result table due to a Foreign Key Violation
- **Exception 8:** Error in utility/task "ALM Material Currency Identification"; Insert/Update operation on DIM_DATES failed; OPERATION ABORTED.
Internal error because the chosen date of execution does not exist in an internal table of indexed dates and associated information (DIM_DATES)
- **Exception 9:** Unhandled exception in utility "ALM Material Currency Identification"; FCT_ALM_SIGNIFICANT_CURRENCY will be returned to its initial state. Error: <ENGINE_GENERATED_ERROR>
Unexpected internal error during execution; it may be a primary key violation during insertion of records into the FCT_* table.
- **Exception 10:** Could not cleanup old records for batch run id: <BATCH_RUN_ID>
Previous error messages in FSI_MESSAGE_LOG for the same BATCH_RUN_ID as run could not be deleted; utility's execution may have still gone through

5.22 Behaviour Pattern Loader

The Behaviour Pattern Loader provides the ability to load bulk Behaviour pattern definitions through a back end procedure. This Loader reads input data from STG_BEHAVIOUR_PATTERN_NRP table – performs data quality checks on the same – and loads the definitions into FSI_BEHAVIOUR_PATTERN_MASTER and FSI_BEHAVIOUR_PATTERN_DETAIL tables based on following conditions:

- **New BP:** If pattern code in stage table is not already present in FSI table, then insert data after quality check and necessary transformations
- **Existing BP:** If pattern code in stage is already present in FSI table, then compare 'created date'. If date in stage table is higher (that is, more recent) than that in FSI table, then overwrite the definition otherwise skip and log appropriate message.

The utility is called from a Batch called Behaviour Pattern Loader. The utility is designed to be executed for a single AS OF DATE only, that is, it would fetch the data from STAGE table for the given MIS date and push the records to DTL/MASTER table accordingly.

The structure of the Stage table is as follows:

Table 5-11 STG_BEHAVIOUR_PATTERN_NRP

Column Name	Logical Name	Data Type	Null Allowed ?	PK	Column Comments
FIC_MIS_DATE	Extraction Date	DATE	No	Yes	Date on which the behaviour pattern definition was created. Normally indicates the calendar date from which it is valid.
F_REPLICATING_PORTFOLIO_FLAG	Replicating Portfolio Flag	CHAR(1)	Yes	No	This indicates whether the behaviour pattern definition is for replicating portfolio (FTP use case) or not. List of Values are Y for Replicating Portfolio and N for Non-replicating portfolio.
N_PATTERN_CODE	Pattern Code	NUMBER(5)	No	Yes	Code assigned to behaviour pattern definition. This must be a number between 70000 to and 99999
N_PATTERN_PERCENTAGE	Pattern Percentage	NUMBER(22,6)	Yes	No	This stores the percentage of current balance that is used as cash flow on the event date. Within one pattern code sum of percentages must not exceed 100.
N_SEQUENCE_NUMBER	Sequence Number	NUMBER(3)	No	Yes	Within one pattern multiple tenors can be defined. Sequence denotes the order of each tenor.

Table 5-11 (Cont.) STG_BEHAVIOUR_PATTERN_NRP

N_PATTERN_T ENOR	Pattern Tenor	NUMBER(5)	Yes	No	This is the tenor specified in behaviour pattern definition and is used to decide cash flow event. It must be read in conjunction with Tenor Unit.
V_PATTERN_S UBTYPE_CD	Behaviour Sub Type Display Code	VARCHAR2(5)	Yes	No	This indicates the sub-type of behaviour for which pattern is defined. Expected values are: If Behaviour Type Display Code is Non Maturing (NM) then expected values are CR for Core and VL for Volatile; If Behaviour Type Display Code is Non Performing (NP) then expected values are SS for Substandard, DF for Doubtful and L for Loss; If Behaviour Type Display Code is Devolvement and Recovery (DR) then expected values are SD for Sight Devolvement, SR for Sight Recovery, UD for Usance Devolvement, UR for Usance Recovery, U for Usance and S for Sight.

Table 5-11 (Cont.) STG_BEHAVIOUR_PATTERN_NRP

V_CREATED_BY	Created By	VARCHAR2(20)	Yes	No	Identifier for the user or model that created the behaviour pattern definition. It can also denote the system from which definition is sourced.
V_PATTERN_DESCRIPTION	Pattern Description	VARCHAR2(255)	Yes	No	Description for the behaviour pattern definition given by user.
V_PATTERN_NAME	Pattern Name	VARCHAR2(30)	Yes	No	Name of the behaviour pattern definition given by user.
V_PATTERN_TYPE_CD	Behaviour Type Display Code	VARCHAR2(40)	Yes	No	This indicates the type of behaviour for which pattern is defined. Expected values are NM for Non Maturing, NP for Non Performing and DR for Devolvement and Recovery.
V_PATTERN_TENOR_UNIT	Pattern Tenor Unit	VARCHAR2(1)	Yes	No	This indicates the unit in which Tenor is specified. List of values are D for Days, M for Months and Y for Years.

Following checks will be performed on the intermediate data populated by the user in STAGE table. The following list of values will be checked against the relevant look-up tables:

- Pattern Code should be between 70000 and 99999.
- BP Pattern Name:
 - If it is a new pattern code, then name should not be already used by another BP.
 - If it is an existing pattern code, then name should be same as BP existing in FSI table. If names are different, then that from FSI table will be retained.
- Behaviour Type Display Code should be present in table FSI_BEHAVIOUR_TYPE_CD column BEHAVIOUR_TYPE_DISPLAY_CD

- Behaviour Sub Type Display Code should be present in table FSI_BEHAVIOUR_SUB_TYPE_CD column BEHAVIOUR_SUB_TYPE_DISPLAY_CD
- Tenor should be a valid number.
- Tenor unit should be present in table FSI_MULTIPLIER_CD column MULTIPLIER_CD.
- Percentage for one pattern code sum of percentage across all Behaviour Type Display Code must not exceed 100. Sequence for one pattern code sequence number must not repeat.

5.22.1 Executing the Behaviour Pattern Loader

To execute the Behaviour Pattern Loader, follow these steps:

1. To run the procedure from SQL*Plus, login to SQL*Plus as the Schema Owner:

```
SQLPLUS > declare
result number;
begin
result := fn_Behaviour_pattern_loader
('Behaviour_Pattern_20121212_1', '20121212');
end;
/
```

BATCH_RUN_ID is any string to identify the executed batch.

AS_OF_DATE is in the format YYYYMMDD

2. To execute Behaviour Pattern Loader from OFSAAI Batch Maintenance, a seeded Batch is provided.

The batch parameters are:

- **Datastore Type:** Select the appropriate datastore from list.
- **Datastore Name:** Select the appropriate name from the list.
- **IP address:** Select the IP address from the list.
- **Rule Name:** fn_behaviourpattern

5.22.2 Exception Messages

Below is a list of error messages generated during execution. They can be viewed in the 'view log' from UI or in FSI_MESSAGE_LOG table after filtering for the given batch_run_id. Appropriate messages are also logged on successful completion of failure of the utility, as the case may be.

Following error messages may be logged during execution:

- **Exception 1:** No records found in STG_BEHAVIOUR_PATTERN_NRP for <DATE> STG_* table has no records for the chosen execution date
- **Exception 2:** Error in Wrapper_bp_loader:
<ENGINE_GENERATED_ERROR_MESSAGE>
Execution of the utility failed due to some unexpected internal error
- **Exception 3:** Issue in look_up_tbl procedure:
<ENGINE_GENERATED_ERROR_MESSAGE>

Procedure to look-up values against matching tables failed; this is an internal error

- **Exception 4:** Issue in procedure: key_val_look_up:
<ENGINE_GENERATED_ERROR_MESSAGE>
Procedure to look-up key-value pairs against matching tables failed; this is an internal error
- **Exception 5:** ROW # <ROWNUM>: N_PATTERN_CD out of range.
N_PATTERN_CD exceeded 99999
- **Exception 6:** ROW # <ROWNUM>: N_TENOR is not a valid number (must be >0).
Self-explanatory
- **Exception 7:** ROW # <ROWNUM>: BEHAVIOUR TYPE DOES NOT MATCH VALID BEHAVIOUR TYPES
Behaviour Pattern Type for a definition must be one of NM, NP or DR
- **Exception 8:** ROW # <ROWNUM>: TENOR UNIT DOES NOT MATCH VALID TENOR UNITS.
Tenor Units must be one of D, M or Y
- **Exception 9:** ROW # <ROWNUM>: BEHAVIOUR SUB-TYPE EITHER DOES NOT EXIST OR IS INCORRECT FOR GIVEN BEHAVIOUR TYPE.
Behaviour Sub-Type in a given pattern set must be as follows: <BEHAVIOUR TYPE>:: [VALID SUB-TYPES] ::: NM[CR,VL], NP[SS,DF,L] & DR[SD,SR,UD,UR,U,S]
- **Exception 10:** ROW # <ROWNUM>: Warning: Replicating_Portfolio_Flag was NOT NULL but neither Y nor N; Will be replaced with N
Self-explanatory
- **Exception 11:** ROW # <ROWNUM>: New pattern code but name is already in use for PATTERN_CD: <PATTERN_CD_NAME>. Please note names are NOT case sensitive. The pattern will be rejected as two pattern code definitions cannot have the same name
- **Exception 12:** ROW # <ROWNUM>: Warning: PATTERN_CODE already exists in FSI_BEHAVIOUR_PATTERN_MASTER with name: <NAME>. This name will be retained. Only the newer(date) of these two records will be kept.
If a pattern code present in STG_* table already exists in the FSI_*_MASTER table, then the FSI_* table will be updated with the new definition but the name from FSI_*_MASTER will be retained if the date of execution for the batch is older than that for which the record already exists in the MASTER table.
- **Exception 13:** Error in Dq_validator: <ENGINE_GENERATED_MESSAGE>
Function for validating STG_* table records failed; this is an internal error
- **Exception 14:** Definitions with negative N_PATTERN_PERCENTAGE. This PATTERN_CD will be ignored.
Some pattern definitions the STG_* table had negative percentage values
- **Exception 15:** Pattern codes with N_PATTERN_PERCENTAGE violation found (>100). These will be ignored.
Account percentage allocation across behaviour pattern types in a Pattern definitions in STG_* table must add to 100
- **Exception 16:** No violations of N_PATTERN_PERCENTAGE found for any pattern code & behaviour type combination.
The definitions that remained after data quality checks had no anomalies in their pattern percentages
- **Exception 17:** Error in procedure percentage_chk: <ENGINE_GENERATED_MESSAGE>
The percentage checker failed; this is an internal error
- **Exception 18:** Duplicate older Behaviour Pattern Definition in STG_TABLE. It will be ignored.

The pattern definitions in STG_* table will be ignored if they are older than those in MASTER table

- **Exception 19:** Date comparison check passed; either no clashes found between STG_BEHAVIOUR_PATTERN_NRP records and FSI_BEHAVIOUR_PATTERN_MASTER records or all definitions in Stage Table were older.
Self-explanatory
- **Exception 20:** Error in flg_older: <ENGINE_GENERATED_MESSAGE>
Procedure for verification of date-clashes between new records in STG_* and those already present in FSI_* MASTER failed; this is an internal error
- **Exception 21:** Error During Merge Operation in FSI_BEHAVIOUR_PATTERN_MASTER: <ENGINE_GENERATED_MESSGE>
Records could not be inserted into FSI_BEHAVIOUR_PATTERN_MASTER; this is either an internal error or a Primary Key violation
- **Exception 22:** Error During Merge Operation in FSI_BEHAVIOUR_PATTERN_DETAIL: <ENGINE_GENERATED_MESSGE>
Records could not be inserted into FSI_BEHAVIOUR_PATTERN_DETAIL; this is either an internal error or a Foreign Key violation
- **Exception 23:** Could not clean-up old records for batch run id: <BATCH_RUN_ID>
Previous error messages in FSI_MESSAGE_LOG for the same BATCH_RUN_ID as run could not be deleted; The execution may have still gone through

6

SCD Process

SCDs are dimensions that have data that changes slowly, rather than changing on a time-based, regular schedule.

For more information on SCDs, see:

- Oracle Data Integrator Best Practices for a Data Warehouse at <http://www.oracle.com/technetwork/middleware/data-integrator/overview/odi-bestpractices-datawarehouse-whi-129686.pdf>
- Oracle Warehouse Builder Data Modeling, ETL, and Data Quality Guide at http://download.oracle.com/docs/cd/E16338_01/owb.112/e10935/dim_objects.htm

Additional online sources include:

http://en.wikipedia.org/wiki/Slowly_changing_dimension

http://www.oracle.com/webfolder/technetwork/tutorials/obe/db/10g/r2/owb/owb10gr2_gs/owb/lesson3/slowlychangingdimensions.htm

<http://www.oraclebidwh.com/2008/11/slowly-changing-dimension-scd/>

<http://www.informationweek.com/news/software/bi/showArticle.jhtml?articleID=204800027&pgno=1>

<http://www.informationweek.com/news/software/bi/showArticle.jhtml?articleID=59301280>

You can also refer to The Data Warehouse Toolkit: The Complete Guide to Dimensional Modeling by Ralph Kimball and Margy Ross.

The SCD component of the platform is delivered via a C++ executable. The types of SCD handled by the OFSAAI SCD component for OFSAA BI applications installers are Type 1 and Type 2.

6.1 Type 1

The Type 1 methodology overwrites old data with new data, and therefore does not track historical data.

This is useful for making changes to dimension data.

N_PRODUCT_SKE	V_PRODUCT_NAME	D_START_DATE	D_END_DATE	F_LATEST_RECORD_INDICATOR
Y	ME			
1	PL	5/31/2010	12/31/9999	Y

In this example,

- N_PRODUCT_SKEY is the surrogate key column which is a unique key for each record in the dimension table.
- V_PRODUCT_NAME is the product name.
- D_START_DATE indicates the date from which this product record is valid.

- D_END_DATE indicates the date till which this product record is valid.
- F_LATEST_RECORD_INDICATOR with value 'Y', which indicates this is the latest record in the dimension table for this product and 'N' indicates it is not.
- If the V_PRODUCT_NAME column is set as a Type 1 SCD column and if there is a change in the product name to 'Personal Loan' from 'PL' in the above example, in the next processing period, then when SCD is executed for the new processing period the record in the above example changes to:

N_PRODUCT_SKE	V_PRODUCT_NAME	D_START_DATE	D_END_DATE	F_LATEST_RECORD_INDICATOR
Y	ME			
1	Personal Loan	6/30/2010	12/31/9999	Y

6.2 Type 2

The Type 2 method tracks historical data by creating multiple records for a given natural key in the dimensional tables with separate surrogate keys.

With Type 2, the historical changes in dimensional data are preserved. In the above example for the change in product name from 'PL' to 'Personal Loan' if history has to be preserved, then the V_PRODUCT_NAME column has to be set as Type 2 when SCD is processed for the processing period and the change inserts a new record as shown in the following example:

N_PRODUCT_SKE	V_PRODUCT_NAME	D_START_DATE	D_END_DATE	F_LATEST_RECORD_INDICATOR
Y	ME			
1	PL	5/31/2010	12/31/9999	N
1	Personal Loan	6/30/2010	12/31/9999	Y

A new record is inserted to the product dimension table with the new product name. The latest record indicator for this is set as 'Y', indicating this is the latest record for the personal loan product. The same flag for the earlier record was set to 'N'.

6.3 Prerequisites

The Hierarchy Flattening Transformation should have been executed successfully.

- The SCD executable should be present under <installation home>ficdb/bin. The file name is scd and the user executing the SCD component should have execute rights on this file.
- The setup tables accessed by SCD component (SETUP_MASTER, SYS_TBL_MASTER, and SYS_STG_JOIN_MASTER) should have the required entries. The SETUP_MASTER table does not come seeded with the installation; the required entries must be added manually. The required columns are mentioned in the Tables Used by the SCD Component. The tables SYS_TBL_MASTER and SYS_STG_JOIN_MASTER are seeded for the Org unit, GL Account, Product, and Common COA (Chart of Accounts) dimensions along with solution installation and you must only add entries in these tables, if you add new dimensions.
- Database Views with name DIM_<Dimension Name>_V come seeded, for the seeded dimensions which come as part of installation. These views source data from the Profitability dimension tables as well as the flattened hierarchy data. DIM_PRODUCT_V is the view available for the product dimension.

New views will have to be added for any new dimension, added in addition to the seeded dimensions.

6.4 Tables Used by the SCD Component

The following are the database tables and columns used by the SCD component:

- **SETUP_MASTER**
- **V_COMPONENT_CODE**: This column is not used by the OFSEFPA solution. This column acts as a primary key for ALMBI.
- **V_COMPONENT_DESC**: This column value is hard coded in the database view definitions for **DIM_PRODUCT_V**, **DIM_GL_ACCOUNT_V**, **DIM_COMMON_COA_V**, and **DIM_ORG_UNIT_V** to obtain the Hierarchy ID from the **REV_HIER_FLATTENED** table. For this reason, the value for this column should be unique.

Note

The value in **V_COMPONENT_DESC** must exactly match with the value used in the SQL to create the **DIM_<dimension>_V** view. The View SQL contains a section referencing the **SETUP_MASTER** table. You must use the same upper and/or lower case letters in **V_COMPONENT_DESC** as used in this section of the View SQL.

- **V_COMPONENT_VALUE**: This is the hierarchy ID to be processed and this can be obtained by executing the following query:

```
select b.object_definition_id,short_desc,long_desc from fsi_m_object_definition_b b inner join
fsi_m_object_definition_tl t on b.object_definition_id = t.object_definition_id and b.id_type = 5
```

Note

For any newly defined Hierarchy, a row will have to be inserted to this table manually for SCD to process that Hierarchy. You can only specify one Hierarchy for each dimension.

Examples:

V_COMPONENT_CODE	V_COMPONENT_VALUE	V_COMPONENT_DESC
COMMON_COA_HIER	1000063952	COMMON_COA_HIER1
GL_ACCOUNT_HIER	200000808	GL_ACCOUNT_HIER1
ORG_HIER	200282	ORG_UNIT_HIER1
PRODUCT_HIER	1000004330	PRODUCT_HIER1

- **SYS_TBL_MASTER**

The solution installer populates one row per dimension for the seeded dimensions in this table.

Column Name	Data Type	Column Description
MAP_REF_NUM	NUMBER(3) NOT NULL	The Mapping Reference Number for this unique mapping of a Source to a Dimension Table.
TBL_NM	VARCHAR2(30) NOT NULL	Dimension Table Name.
STG_TBL_NM	VARCHAR2(30) NOT NULL	Staging Table Name.

SRC_PRTY	NUMBER(2) NULL	Priority of the Source when multiple sources are mapped to the same target.
SRC_PROC_SEQ	NUMBER(2) NOT NULL	The sequence in which the various sources for the DIMENSION will be taken up for processing.
SRC_TYP	VARCHAR2(30) NULL	The type of the Source for a Dimension, that is, Transaction Or Master Source.
DT_OFFSET	NUMBER(2) NULL	The offset for calculating the Start Date based on the Functional Requirements Document (FRD).
SRC_KEY	NUMBER(3) NULL	

Example:

This is the row inserted by the solution installer for the product dimension.

MAP_REF_NUM	128
TBL_NM	DIM_PRODUCT
STG_TBL_NM	DIM_PRODUCT_V
SRC_PRTY	
SRC_PROC_SEQ	1
SRC_TYP	MASTER
DT_OFFSET	0

Note

For any newly defined dimension, a row will have to be inserted to this table manually.

- **SYS_STG_JOIN_MASTER**

The solution installer populates this table for the seeded dimensions.

Column Name	Data Type	Column Description
MAP_REF_NUM	NUMBER(3) NOT NULL	The Mapping Reference Number for this unique mapping of a Source to a Dimension Table.
COL_NM	VARCHAR2(30) NOT NULL	Name of the column in the Dimension Table.
COL_TYP	VARCHAR2(30) NOT NULL	Type of column. The possible values are given in the following section.
STG_COL_NM	VARCHAR2(60) NULL	Name of the column in the Staging Table.
SCD_TYP_ID	NUMBER(3) NULL	SCD type for the column.
PRTY_LOOKUP_REQD_FLG	CHAR(1) NULL	Column to determine whether Lookup is required for Priority of Source against the Source Key Column or not.

COL_DATATYPE	VARCHAR2(15) NULL	The list of possible values are VARCHAR, DATE, and NUMBER, based on the underlying column datatype.
COL_FORMAT	VARCHAR2(15) NULL	

The possible values for column type (the COL_TYPE column) in SYS_STG_JOIN_MASTER table are:

PK: Primary Dimension Value (can be the multiple of the given Mapping Reference Number)

SK: Surrogate Key

DA: Dimensional Attribute (may be multiple for a given Mapping Reference Number)

SD: Start Date

ED: End Date

LRI: Latest Record Indicator (Current Flag)

CSK: Current Surrogate Key

PSK: Previous Surrogate Key

SS: Source Key

LUD: Last Updated Date/Time

LUB: Last Updated By

Example:

This is the row inserted by the solution installer for the product dimension.

MAP_REF_NUM	128
COL_NM	V_PRODUCT_NAME
COL_TYP	DA
STG_COL_NM	V_PRODUCT_NAME
SCD_TYP_ID	2
PRTY_LOOKUP_REQD_FLG	N
COL_DATATYPE	VARCHAR
COL_FORMAT	

Note

For any newly defined dimension, the column details will have to be inserted to this table manually.

- DIM_< dimension name >_V: The database view which SCD uses as the source.

DIM_PRODUCTS_V

These views come as part of install for the dimensions seeded with the application.

Note: For any newly defined dimension, a view will have to be created, which is similar to that of DIM_PRODUCTS_V.

6.6 Checking the Execution Status

The Batch execution status can be monitored through Batch Monitor section of OFSAAI Operations module.

The status messages in batch monitor are:

N: Not Started

O: On Going

F: Failure

S: Success

1. The execution log can also be accessed on the application server in the directory `$FIC_DB_HOME/log/figen`, where file name will have the Batch Execution ID. The detailed SCD component log can be accessed on the application server in the directory `$FIC_HOME` by accessing the following path `/ftpshare/<infodom name>/logs`.
2. Check the `.profile` file in the installation home if you are unable to find this path.
3. The Event Log window in Batch Monitor section provides execution logs, in which the top row is the most recent. Any errors during the Batch execution are listed in the logs.