Oracle® Retail AI Foundation Cloud Services

Operations Guide

Release 25.1.101.0

G25244-02

March 2025

Oracle Retail AI Foundation Cloud Services Operations Guide, Release 25.1.101.0

G25244-02

Contents

Send Us Your Comments

Oracle Retail Insights and AI Foundation Cloud Services Operations Guide

Oracle welcomes customers' comments and suggestions on the quality and usefulness of this document.

Your feedback is important, and helps us to best meet your needs as a user of our products. For example:

- Are the implementation steps correct and complete?
- Did you understand the context of the procedures?
- Did you find any errors in the information?
- Does the structure of the information help you with your tasks?
- Do you need different information or graphics? If so, where, and in what format?
- Are the examples correct? Do you need more examples?

If you find any errors or have any other suggestions for improvement, then please tell us your name, the name of the company who has licensed our products, the title and part number of the documentation and the chapter, section, and page number (if available).

> **Note:**
>
> Before sending us your comments, you might like to check that you have the latest version of the document and if any concerns are already addressed. To do this, access the Online Documentation available on the Oracle Technology Network Web site. It contains the most current Documentation Library plus all documents revised or released recently.

Send your comments to us using the electronic mail address: retail-doc_us@oracle.com

Please give your name, address, electronic mail address, and telephone number (optional).

If you need assistance with Oracle software, then please contact your support representative or Oracle Support Services.

If you require training or instruction in using Oracle software, then please contact your Oracle local office and inquire about our Oracle University offerings. A list of Oracle offices is available on our Web site at `http://www.oracle.com`.

Preface

This Operations Guide provides critical information about the processing and operating details of the Retail Insights and AI Foundation Cloud Services, including the following:

- Standalone and Adhoc batch processes
- Integration processes

Audience

This guide is for:

- Systems administration and operations personnel
- Systems analyst
- Integrators and implementers

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.

Customer Support

To contact Oracle Customer Support, access My Oracle Support at the following URL:

https://support.oracle.com

When contacting Customer Support, please provide the following:

- Product version and program/module name
- Functional and technical description of the problem (include business impact)
- Detailed step-by-step instructions to re-create
- Exact error message received
- Screen shots of each step you take

Oracle Help Center (docs.oracle.com)

Oracle Retail Product documentation is available on the following website https://docs.oracle.com/en/industries/retail/html

Comments and Suggestions

Please give us feedback about Oracle Retail Help and Guides. You can send an e-mail to: retail-doc_us@oracle.com

Oracle Retail Cloud Services and Business Agility

The Oracle Retail Insights and AI Foundation Cloud Services are hosted in the Oracle Cloud with the security features inherent to Oracle technology and a robust data center classification, providing significant uptime. The Oracle Cloud team is responsible for installing, monitoring, patching, and upgrading retail software.

Included in the service is continuous technical support, access to software feature enhancements, hardware upgrades, and disaster recovery. The Cloud Service model helps to free customer IT resources from the need to perform these tasks, giving retailers greater business agility to respond to changing technologies and to perform more value-added tasks focused on business processes and innovation.

Oracle Retail Software Cloud Service is acquired exclusively through a subscription service (SaaS) model. This shifts funding from a capital investment in software to an operational expense. Subscription-based pricing for retail applications offers flexibility and cost effectiveness.

1 Introduction

This document is intended to guide a Retail Analytics and Planning Cloud Services implementer through the internal operations of key areas of the AI Foundation platform that they will need to interact with during a project, such as ad hoc batch processes and integration programs. All programs are located within the Process Orchestration and Monitoring (POM) application and the reader is expected to be familiar with that tool.

This guide includes the following topics:

- **Retail AI Foundation Data Standalone Batch Processes** - This chapter provides an overview of each AIF data batch program or process flow in the Standalone set of jobs in POM, the input and output tables involved in the process, and any dependencies or usage details to consider before running them.
- **Retail AI Foundation Cloud Services Standalone Batch Processes -** This chapter provides an overview of each AI Foundation Cloud Services batch program or process flow available in the Standalone set of jobs in POM. The primary purpose of the AI Foundation ad hoc programs is to integrate data from either RI, flat files, or Innovation Workbench.
- **Retail AI Foundation Data Standalone Process Flows** - This chapter provides a set of cross-reference tables showing how programs in the AIF data standalone processes are linked to each other, such as the staging and load jobs to move a single file into the database from start to finish. This should be used to disable all unneeded jobs in the adhoc load processes for files you are not trying to load.
- **Data Validation Framework -** This chapter explains the data validation procedures associated with foundation input files. The data validation framework checks for common mistakes and issues in the incoming data files and either fails the process or outputs warnings to the database, depending on the issue.
- **Support Utilities** - This chapter describes the self-service utilities used for environment maintenance and cleanup. Implementers should be aware of the utilities available to them and leverage them during the project, as needed.

2

AI Foundation Data Standalone Processes

The primary function of standalone processes in the AI Foundation Data (AIF DATA) schedule is to load history data in a new environment for use in one or more applications on the platform. These process flows group together multiple, related programs that load data files, stage them in the database, and transform them into multiple target tables in the data warehouse. Processes are also available for integrations with Planning applications (MFP, AP, and so on) and Xstore.

Adjustments History Load

Module Name	HIST_CSV_ADJUSTMENTS_LOAD_ADHOC
Description	Loads the `ADJUSTMENT.csv` file into the data warehouse and populates key data tables used to integrate with other systems for history data.
Dependencies	HIST_ZIP_FILE_LOAD_ADHOC
Business Activity	Historical Data Load

Design Overview

The history load process for Inventory Adjustment transactions accepts an input file at the transaction level using the file specification for `ADJUSTMENT.csv`. It assumes the file has already been moved into place using the `HIST_ZIP_FILE_LOAD_ADHOC` process. This process imports the file into a preprocessing table in the database, transforms it to RI’s internal staging tables, then loads it into the base fact (item/location/day), as well as the week aggregate used for integrations (item/location/week). The Reason dimension is also seeded with records if the reason code and reason description are provided on the transactions.

Note:

This process does not currently populate BI aggregate tables. Those jobs need to be run separately after each execution of this process if it is necessary to use this data for reporting in RI.

Key Tables Affected

Table	Usage
W_ADJUSTMENT_FTS	File Input
W_REASON_DS	Staging
W_DOMAIN_MEMBER_DS_TL	Staging
W_RTL_INVADJ_IT_LC_DY_FS	Staging
W_RTL_INVADJ_IT_LC_DY_F	Output (Base Fact)
W_RTL_INVADJ_IT_LC_WK_A	Output (Aggregate)

Table	Usage
W_DOMAIN_MEMBER_LKP_TL	Output (Reason Descriptions)

Aggregate Fact History Load

Module Name	HIST_AGGR_FACT_LOAD_ADHOC
Description	Loads pre-aggregated fact data from flat files into the data warehouse for data that is above the item/location intersection.
Dependencies	HIST_ZIP_FILE_LOAD_ADHOC
Business Activity	Historical Data Load

Design Overview

The history load process for aggregate fact data is intended for new customers or migrating customers that cannot provide item/location level history for sales, inventory, and other areas. Pre-aggregated data can be loaded into one of four identical file interfaces that support a wide variety of measures across all functional areas. Each interface must have data for a single intersection (such as subclass/area/week) and the intersection must be configured in `C_ODI_PARAM_VW` before running the loads. This process will take care of importing the files and loading the data into data warehouse tables for storage.

There are jobs included in this process to validate the partition structure of these tables, because they support flexible partitioning by day or week levels. The validator jobs (such as `W_FACT1_PROD1_LC1_T1_F_VALIDATOR_JOB`) verify whether the partition structure is correct relative to the configuration in `C_ODI_PARAM_VW`. If the configuration and table structures do not match, it will automatically drop and re-create all partitions in the expected format. This will result in dropped data as well, so it is important to verify the configuration before loading any data.

When using these tables, the overall flow of data is as follows:

1. Update the configuration table (`C_ODI_PARAM_VW`) from the Control & Tactical Center to specify your base intersections for each table.
   a. The parameter names for these tables are prefixed with `RI_FACT`, such as `RI_FACT1_PROD_LEVEL` for the product level of the `FACT1` table.
   b. The complete list of values you may put into these parameters is provided below.

Product (PROD)	Organization (ORG)	Calendar (CAL)
CMP	COMPANY	YEAR
DIV	CHAIN	HALFYEAR
GRP	AREA	QUARTER
DEPT	REGION	PERIOD
CLS	DISTRICT	WEEK
SBC	LOCATION	DAY
ITEM	CHANNEL	GREGORIANYEAR
ALL	PLANNING_CHANNEL	GREGORIANQUARTER
FLEX1 -FLEX20	PRICE_ZONE	GREGORIANMONTH

Product (PROD)	Organization (ORG)	Calendar (CAL)
	ALL	GREGORIANDAY

2. Update the table partition configuration in `C_MODULE_EXACT_TABLE` by setting the `PARTITION_COLUMN_TYPE` and `PARTITION_INTERVAL` as `WK` for week data or `DY` for day data. Calendar levels above week level do not require partitioning as it's assumed the data volume will be low.
3. Enable the `FACT1` through `FACT4` modules (based on your intended usage) in `C_MODULE_ARTIFACT` by setting the `PARTITION_FLG` and `ACTIVE_FLG` to `Y`.
4. Use the process `CREATE_PARTITION_ADHOC` to re-execute the partitioning programs for any configuration changes made above, unless you have not yet loaded any calendar data, in which case the `CALENDAR_LOAD_ADHOC` process will also perform partitioning for these tables.
5. Use this process (`HIST_AGGR_FACT_LOAD_ADHOC`) to populate staging tables and move data from staging (`FS`) to target (`F`) tables. Enable all jobs relating to the table(s) you will be loading (Each table load has a `COPY`, `STG`, `VALIDATOR`, `TMP`, and `F` job, and all of them should be enabled).
   a. Records may be rejected due to bad/missing data in the 3 base dimensions supported on the facts (product, location, calendar). The job does not fail if rejects occur; it will load any valid records.
   b. Rejected records will be copied to `E$` tables such as `E$_W_RTL_FACT1_PROD1_LC1_T1_TMP`. `E$` tables are created dynamically when records get rejected so the table may not exist initially.
   c. `E$` tables will contain the full rejected record, which you may insert back into the staging (`FS`) tables later to attempt to load them again.
   d. `E$` tables will not purge or drop data unless you perform a full schema cleanup or database clone from another environment.
6. If sending the data to a Planning application, use the processes `LOAD_PDS_FACT1_AGGR_PROCESS_ADHOC` through `LOAD_PDS_FACT4_AGGR_PROCESS_ADHOC` to export the contents of each table to PDS. The integration tables for PDS will have similar names as the source tables, only adding `PDS` in the name. For example, `W_RTL_FACT1_PROD1_LC1_T1_F` is loaded to `W_PDS_FACT1_PROD1_LC1_T1_F` by the process `LOAD_PDS_FACT1_AGGR_PROCESS_ADHOC`.

Key Tables Affected

Table	Usage
W_RTL_FACT1_PROD1_LC1_T1_FS	Staging
W_RTL_FACT2_PROD2_LC2_T2_FS	Staging
W_RTL_FACT3_PROD3_LC3_T3_FS	Staging
W_RTL_FACT4_PROD4_LC4_T4_FS	Staging
W_RTL_FACT1_PROD1_LC1_T1_F	Output
W_RTL_FACT2_PROD2_LC2_T2_F	Output
W_RTL_FACT3_PROD3_LC3_T3_F	Output
W_RTL_FACT4_PROD4_LC4_T4_F	Output

Allocation History Load

Module Name	HIST_ALCDETAIL_LOAD_ADHOC
Description	Loads a full snapshot of allocations data from `W_RTL_ALC_DETAILS_DS.dat` and `W_RTL_ALC_IT_LC_DY_FS.dat` to initialize the dimension and fact data before the nightly batch is enabled.
Dependencies	HIST_ZIP_FILE_LOAD_ADHOC
Business Activity	Historical Data Load

Design Overview

Data regarding allocations of merchandise is split between two interfaces, the dimension file `W_RTL_ALC_DETAILS_DS.dat` and the fact file `W_RTL_ALC_IT_LC_DY_FS.dat`. This process can be used to load full snapshots of historical or currently active allocations to the data warehouse outside the nightly batch cycle. The two files must be in sync, meaning that every allocation record on the detail file must have a record in the header file. The header file is always a full snapshot of all allocations that should appear as currently active in the data warehouse, meaning that if any allocation records are no longer sent on `W_RTL_ALC_DETAILS_DS.dat`, they will be marked as inactive/closed in the data warehouse table (`CURRENT_FLG = N`) and should no longer appear in the files.

This data is used primarily for Retail Insights reporting and for Inventory Planning Optimization (IPO).

Key Tables Affected

Table	Usage
W_RTL_ALC_DETAILS_DS	Staging
W_RTL_ALC_DETAILS_D	Output
W_RTL_ALC_IT_LC_DY_FS	Staging
W_RTL_ALC_IT_LC_DY_F	Output

Batch Individual File Reprocessing

Module Name	REPROCESS_ZIP_FILE_PROCESS_ADHOC
Description	Looks for the `RI_REPROCESS_DATA.zip` file and unpacks it, moving any files to the incoming directory for batch processes.
Dependencies	None
Business Activity	Nightly Batch Processing

Design Overview

This process moves and unloads a ZIP file (specifically `RI_REPROCESS_DATA.zip`) so that the file contents may be added to an in-progress nightly batch run of the AIF DATA schedule. The ZIP file may contain one or multiple files. It only needs to contain the files that you wish to update for the current batch run. Unlike the other ZIP file processes, this process does not archive or delete existing files in the system, so it can safely be used repeatedly to upload new files on top of existing data.

Batch Nightly File Reprocessing

Module Name	NIGHTLY_ZIP_FILE_PROCESS_ADHOC
Description	Looks for the `RAP_DATA.zip` file and any other nightly ZIP file name and unpacks it, deleting all existing nightly files and copying in the new ones.
Dependencies	None
Business Activity	Nightly Batch Processing

Design Overview

This process moves and unloads nightly ZIP files (such as `RAP_DATA.zip`) so that the file contents may be used for an in-progress nightly batch run of the AIF DATA schedule. The ZIP file(s) must contain all data files you need for a nightly batch run. This process contains the exact same jobs as the nightly AIF DATA batch and the primary purpose is to let you reload a new ZIP file when your current nightly batch has failed or you've accidentally provided the wrong upload and need to replace it.

The program uses the `C_LOAD_DATES` table to support restartability if it runs multiple times for the same or different ZIP files, based on the following scenarios:

- The `C_LOAD_DATES` entry it creates has `PACKAGE_NAME='ZIPUNLOAD'`.
- When you run it as part of the nightly batch or as part of this ad hoc job, it creates the entry in the `C_LOAD_DATES` table.
- When you have a "Success" status in `C_LOAD_DATES`, then you can rerun the job by placing a new zip file; it will then unzip it in the internal `Expand` directory (retaining older extracted files in the `Expand` directory if it is not overwritten).
- When you have a status like "Failed", then rerunning will mark it as "Success" and retain the files in the `Expand` directory; it will not erase existing files.
- When you don't have any entry in the `C_LOAD_DATES` table, then running it erases all the directories, then unzips your file into the internal location and processes them as needed.

Customer Loyalty Load

Module Name	HIST_CUST_LYL_LOAD_ADHOC
Description	Loads Customer Loyalty dimension and fact data for use in Retail Insights reporting.
Dependencies	HIST_ZIP_FILE_LOAD_ADHOC
Business Activity	Historical Data Load

Design Overview

This process will load the Customer Loyalty dimension and fact files. This data is used for Retail Insights reporting only. All jobs in the process should be enabled before running it.

Key Tables Affected

Input Table	Target Table
W_RTL_CUST_LYL_PROG_DS	W_RTL_CUST_LYL_PROG_D
W_RTL_CUST_LYL_ACCT_DS	W_RTL_CUST_LYL_ACCT_D
W_RTL_CUST_LYL_AWD_ACCT_DS	W_RTL_CUST_LYL_AWD_ACCT_D
W_RTL_CUST_LYL_TRX_LC_DY_FS	W_RTL_CUST_LYL_TRX_LC_DY_F
W_RTL_CUST_LYL_AWD_TRX_DY_FS	W_RTL_CUST_LYL_AWD_TRX_DY_F
W_RTL_CUST_LYL_TRX_LC_DY_F	W_RTL_CUST_LYL_PROG_LC_DY_A

Data Security Load

Module Name	RAF_SEC_FILTER_LOAD_ADHOC
Description	Copies the data security staging table data (which is populated from IW) into the target tables, such as `RAF_SEC_USER`.
Dependencies	None
Business Activity	Application Administration

Design Overview

This process loads data for AIF data security functionality. The tables populated by this process limit what data an end user can see in certain AIF applications, such as RI and PMO. This data load flow only accepts data from IW, and is an alternative to sending flat files as part of the nightly batch process. This is only a replacement for the flat file load; if there are any jobs downstream in the applications that must be run, those are still required. The steps to use this process are:

1. Implement data security integration in Innovation Workbench/APEX to retrieve the users, groups, and data filter definitions from an external source. You may use REST APIs or custom file loads for this integration.
2. Develop SQL statements or procedures to insert your data into the staging tables (listed below).
3. Run the `RAF_SEC_FILTER_LOAD_ADHOC` process, which will truncate the target tables and insert your newly staged data. If a staging table is empty, then it will not truncate the target table.

The entire process could be automated by establishing REST APIs that post into IW tables, a stored procedure that pushes the data from IW into the RAF staging tables, and then adding a `DBMS_SCHEDULER` job that runs the POM process. Because a truncate-and-load process is used, you must maintain the full set of data security records somewhere to push into the AIF tables.

Key Tables Affected

Staging Table	Target Table
RAF_SEC_USER_STG	RAF_SEC_USER
RAF_SEC_GROUP_STG	RAF_SEC_GROUP

Staging Table	Target Table
RAF_SEC_USER_GROUP_STG	RAF_SEC_USER_GROUP
RAF_FILTER_GROUP_MERCH_STG	RAF_FILTER_GROUP_MERCH
RAF_FILTER_GROUP_ORG_STG	RAF_FILTER_GROUP_ORG

Deal Actuals History Load

Module Name	HIST_DEAL_LOAD_ADHOC
Description	Loads the `W_RTL_DEALACT_IT_LC_DY_FS.dat` file into the data warehouse and populates the target fact table for BI reporting.
Dependencies	HIST_ZIP_FILE_LOAD_ADHOC
Business Activity	Historical Data Load

Design Overview

The history load process for Deal Actuals accepts an input file at the deal/item/location/date level using the file specification for `W_RTL_DEALACT_IT_LC_DY_FS.dat`. It assumes the file has already been moved into place using the `HIST_ZIP_FILE_LOAD_ADHOC` process. This process imports the file into a staging table in the database and loads it into the base fact (item/location/day) table for reporting.

Key Tables Affected

Table	Usage
W_RTL_DEALACT_IT_LC_DY_FS	Staging
W_RTL_DEALACT_IT_LC_DY_F	Output (Base Fact)

Deal Income History Load

Module Name	HIST_CSV_DEAL_INCOME_LOAD_ADHOC
Description	Loads the `DEAL_INCOME.csv` file into the data warehouse and populates key data tables used to integrate with other systems for history data.
Dependencies	HIST_ZIP_FILE_LOAD_ADHOC
Business Activity	Historical Data Load

Design Overview

The history load process for Deal Income transactions accepts an input file at the transaction level using the file specification for `DEAL_INCOME.csv`. It assumes the file has already been moved into place using the `HIST_ZIP_FILE_LOAD_ADHOC` process. This process imports the file into a preprocessing table in the database, transforms it to RI's internal staging tables, and then loads it into the base fact (item/location/day) as well as the week aggregate used for integrations (item/location/week).

Key Tables Affected

Table	Usage
W_RTL_DEALINC_IT_LC_DY_FTS	File Input
W_RTL_DEALINC_IT_LC_DY_FS	Staging
W_RTL_DEALINC_IT_LC_DY_F	Output (Base Fact)
W_RTL_DEALINC_IT_LC_WK_A	Output (Aggregate)

Default Calendar Initialization

Module Name	AUTO_GEN_CALENDAR_LOAD_ADHOC
Description	Automatically generates a generic NRF fiscal calendar and sets up the data warehouse database with it.
Dependencies	None
Business Activity	Initial System Setup

Design Overview

The auto-generated calendar process does not require any input files. Instead, it uses an internal calendar definition based on the National Retail Federation (NRF) 4-5-4 business calendar to populate the Retail Insights data model with basic calendar information. The NRF calendar typically starts around the first week of February and runs for 52 or 53 weeks, depending on the year. The default calendar starts from January 2017 and extends for approximately 30 years. It automatically includes 53-week years where appropriate and follows the NRF guidelines for fiscal weeks and periods.

This process performs all the necessary transform and load jobs required to set up the RAP data warehouse calendar. This process should only be used if you cannot get a business calendar definition from any other source, and the retailer does not want to provide a file themselves. Once this process runs, you can disable `W_MCAL_PERIOD_DS_JOB` in your nightly batch if you do not intend to ever provide a calendar file directly.

This process also populates the Gregorian system calendar at the same time the fiscal calendar is loaded. The Gregorian calendar requires additional start and end date parameters from `C_ODI_PARAM` to define the time range to generate. It must be greater than the range of time in the fiscal calendar. Output tables that start with `W_MCAL_` are mainly used for fiscal calendar generation, while the other tables, such as `W_DAY_D`, are used for the Gregorian calendar. All output tables must be successfully populated with calendar data to use the platform.

Key Tables Affected

Table	Usage
W_MCAL_PERIOD_DS	Staging
W_TIME_OF_DAY_D	Output
W_DAY_D	Output
W_YEAR_D	Output

Table	Usage
W_QTR_D	Output
W_MONTH_D	Output
W_WEEK_D	Output
W_MINUTE_OF_DAY_D	Output
W_MCAL_CONFIG_G	Output
W_MCAL_CAL_D	Output
W_MCAL_PERIOD_D	Output
W_MCAL_DAY_D	Output
W_MCAL_WEEK_D	Output
W_MCAL_YEAR_D	Output
W_MCAL_QTR_D	Output
W_RTL_MCAL_DAY_SHIFT_D	Output
W_RTL_MCAL_DAY_UNSHIFT_D	Output
W_RTL_MCAL_DAY_GUNSHIFT_D	Output
W_RTL_MCAL_DAY_CUSTOM_D	Output
W_RTL_MCAL_WEEK_SHIFT_D	Output
W_RTL_MCAL_WEEK_UNSHIFT_D	Output
W_RTL_MCAL_PERIOD_SHIFT_D	Output
W_RTL_MCAL_PERIOD_UNSHIFT_D	Output

ETL Business Date Update

Module Name	LOAD_CURRENT_BUSINESS_DATE_ADHOC
Description	Override the current business date used for loading data into the data warehouse.
Dependencies	None
Business Activity	Batch Administration

Design Overview

This process updates the current business date in the foundation data warehouse. The current business date usually reflects the most recent data loaded into the platform. For example, if the last nightly batch imported data is for February 2nd, 2023, the current business date will be that same date. Specific to Retail Insights, this date is also used whenever calculating a repository variable like `CurrentDate` or `CurrentWeek`. There may be times during an implementation of an Analytics and Planning solution where you need to alter this date, such as when you are loading historical data for a specific past date and want the data warehouse to reflect that date as the current date. In such cases, you must use this process to set the business date as the first step before running those other processes.

The `ETL_BUSINESS_DATE_JOB` within this process requires a date as an input parameter. The date must be in the format `YYYY-MM-DD`. When you run the process from the POM UI, ensure you edit the parameters for this job first and enter the date as the only input value.

Sample payload when using an API to call the process:

```
{
"cycleName":"Adhoc",
"flowName":"Adhoc",
"processName":"LOAD_CURRENT_BUSINESS_DATE_ADHOC",
"requestParameters":"jobParams.ETL_BUSINESS_DATE_JOB=2017-12-31"
}
```

Key Tables Affected

Table	Usage
W_RTL_CURR_MCAL_G	Output

Fix Unusable Indexes

Module Name	FIX_UNUSABLE_INDEX_ADHOC
Description	Repair broken database indexes that may result from erasing and reloading dimension or fact data repeatedly
Dependencies	None
Business Activity	Batch Administration

Design Overview

This process executes a single job, `FIX_UNUSABLE_INDEX_JOB`, which repairs any database indexes in an unusable state. The most common reason to execute this job is that you have encountered the error:

```
ORA-26026: unique index <INDEX NAME> initially in unusable state
```

When this error occurs, you must run this process to correct it. If you have a failed batch job that encountered this error, you may restart that job from POM after running this process. This job also runs as part of the AIF DATA weekly maintenance process `RI_MAINTENANCE_ADHOC`, which should be scheduled to run every week before your nightly batches.

Flexible Fact Load

Module Name	FLEXFACT_LOAD_ADHOC
Description	Loads the flexible fact tables for use in AIF applications and Retail Insights reporting.
Dependencies	HIST_ZIP_FILE_LOAD_ADHOC
Business Activity	Historical Data Load

Design Overview

This process populates the flex fact tables that are used in both AI Foundation to display custom attributes and measures and in Retail Insights reporting on custom fact data. Prior to loading flex fact data, you must configure the proper data levels in `C_ODI_PARAM` to match the hierarchy levels used in the files. Refer to the *Retail Insights Implementation Guide* for additional details on configuring flex facts.

There is a job included in this process to validate the partition structure of the `FLEXFACT1` table, because it supports flexible partitioning by day or week levels. The validator job (`W_RTL_FLEXFACT1_F_VALIDATOR_JOB`) verifies whether the partition structure is correct relative to the configuration in `C_ODI_PARAM_VW`. If the configuration and table structures do not match, it automatically drops and re-creates all partitions in the expected format. This will result in dropped data as well, so it is important to verify the configuration before loading any data.

Key Tables Affected

Input Table	Output Table
W_RTL_FLEXFACT1_FS	W_RTL_FLEXFACT1_F
W_RTL_FLEXFACT2_FS	W_RTL_FLEXFACT2_F
W_RTL_FLEXFACT3_FS	W_RTL_FLEXFACT3_F
W_RTL_FLEXFACT4_FS	W_RTL_FLEXFACT4_F

Gift Card Sales Load

Module Name	HIST_GCN_TRX_LOAD_ADHOC
Description	Loads Gift Card Sales fact data for use in Retail Insights reporting.
Dependencies	HIST_ZIP_FILE_LOAD_ADHOC
Business Activity	Historical Data Load

Design Overview

This process loads the Gift Card Sales fact file. This fact data is used for Retail Insights reporting only. All jobs in the process should be enabled before running it.

Key Tables Affected

Table	Usage
W_RTL_GCN_TRX_LC_DY_FS	Staging
W_RTL_GCN_TRX_LC_DY_F	Output

History Data Cleanup

Module Name	HIST_DATA_CLEANUP_ADHOC

Description	Erase all data from Inventory and Price tables in RI, in order to restart your history load for those interfaces.
Dependencies	None
Business Activity	Historical Data Load

Design Overview

This process erases all data from select functional areas (currently Inventory Position and Pricing facts). The purpose of the process is to reset the environment if the data currently loaded is invalid or unwanted, and you'd like to start over with empty tables.

Note:

It does not erase partition structures, so you need to load data for the same range of dates already available.

It also does not reset the `C_HIST_LOAD_STATUS` table, so you will need to update that before loading any new data.

History Data File Upload

Module Name	HIST_ZIP_FILE_LOAD_ADHOC
Description	Looks for the `RIHIST_RMS_DATA.zip` file and unpacks it, moving any files to the incoming directory for batch processes.
Dependencies	None
Business Activity	Historical Data Load

Design Overview

This process moves and unloads a ZIP file (specifically `RIHIST_RMS_DATA.zip`) so that the file contents may be used for one or more history and seeding load jobs. The ZIP file may contain one or multiple files. This process is a prerequisite to running any history or seeding load programs.

The first job in this process waits a set period of time for the ZIP file to be uploaded, and it fails if it is not received in that time (4 hours by default). The second job moves the ZIP file to the internal server location and unzip it. It deletes any files previously in the destination folder, unzip the new file, and move the ZIP file to an archive when complete. It fails if the ZIP does not contain any data files, as there is nothing for it to move.

History Data Master Flow

Module Name	RI_FLOW_ADHOC
Description	Ad hoc flow for loading all major foundation fact files in a single process that can be scheduled to run repeatedly in the same day for automated history loads.

Dependencies	None
Business Activity	Historical Data Load

Design Overview

The `RI_FLOW_ADHOC` process in the AIF DATA schedule is a replacement for the `RI_INTRADAY_CYCLE` found in earlier versions. It provides a single master process flow that can load the most common RAP foundation fact files and populate all data warehouse tables using one or multiple input zip files. For example, you may upload the files named `RAP_DATA_HIST.zip.1`, `RAP_DATA_HIST.zip.2`, and `RAP_DATA_HIST.zip.3` at the same time to Object Storage for iterative processing starting with the first file in numerical order. When you provide multiple zip files, you will also schedule the `RI_FLOW_ADHOC` process to run multiple times and, each time, it will pick up the next ZIP file in the sequence. Make sure you only schedule a number of recurring flow executions equal to the number of ZIPs you will upload. If you run the flow without any ZIPs remaining to be processed, all jobs will still attempt to execute with 0 rows of data in the staging tables. This will cause the flow to fail as some loads are designed to stop the process if there is no data available.

The process flow cannot be executed without first configuring which jobs to run. You may configure the flow by enabling or disabling sets of jobs by process name, and then, within the processes you are enabling, you may selectively disable jobs for tables you are not using. The processes included in the flow are listed below in order of execution (first to last). You should disable the processes you do not need before running or scheduling the flow.

Process Name	Usage
FLOW_LOAD_START_PROCESS	Required, non-functional jobs for logging/ dependency purposes.
CLEANUP_C_LOAD_DATES_FLOW_PROCESS	Required, clears prior run statuses from `C_LOAD_DATES` table.
ZIP_FILE_LOAD_FLOW_PROCESS	Required, selects a ZIP file for the current run and unpacks it.
STG_SALES_LOAD_ADHOC_PROCESS	Stages the `W_RTL_SLS_TRX_IT_LC_DY_FS.dat` and `W_RTL_SLSPK_IT_LC_DY_FS.dat` files; disable if not using this file.
STG_CSV_SALES_LOAD_ADHOC_PROCESS	Stages the `SALES.csv` and `SALES_PACK.csv` files; disable if not using these files.
STG_RTV_LOAD_ADHOC_PROCESS	Stages the `W_RTL_INVRTV_IT_LC_DY_FS.dat` file; disable if not using this file.
STG_CSV_RTV_LOAD_ADHOC_PROCESS	Stages the `RTV.csv` file; disable if not using this file.
STG_INVADJ_LOAD_ADHOC_PROCESS	Stages the `W_RTL_INVADJ_IT_LC_DY_FS.dat` file; disable if not using this file.
STG_CSV_INVADJ_LOAD_ADHOC_PROCESS	Stages the `ADJUSTMENT.csv` file; disable if not using this file.
STG_INVRC_LOAD_ADHOC_PROCESS	Stages the `W_RTL_INVRC_IT_LC_DY_FS.dat` file; disable if not using this file.
STG_CSV_INVRC_LOAD_ADHOC_PROCESS	Stages the `RECEIPT.csv` file; disable if not using this file.

Process Name	Usage
STG_INVTSF_LOAD_ADHOC_PROCESS	Stages the `W_RTL_INVTSF_IT_LC_DY_FS.dat` file; disable if not using this file.
STG_CSV_INVTSF_LOAD_ADHOC_PROCESS	Stages the `TRANSFER.csv` file; disable if not using this file.
STG_MARKDOWN_LOAD_ADHOC_PROCESS	Stages the `W_RTL_MKDN_IT_LC_DY_FS.dat` file; disable if not using this file.
STG_CSV_MARKDOWN_LOAD_ADHOC_PROCESS	Stages the `MARKDOWN.csv` file; disable if not using this file.
STG_DEAL_INCOME_ADHOC_PROCESS	Stages the `W_RTL_DEALINC_IT_LC_DY_FS.dat` file; disable if not using this file.
STG_CSV_DEAL_INCOME_ADHOC_PROCESS	Stages the `DEAL_INCOME.csv` file; disable if not using this file.
STG_INV_LOAD_ADHOC_PROCESS	Stages the `W_RTL_INV_IT_LC_DY_FS.dat` file; disable if not using this file.
STG_INVPK_LOAD_ADHOC_PROCESS	Stages the `W_RTL_INVPK_IT_LC_DY_FS.dat` file; disable if not using this file.
STG_INVOOS_LOAD_ADHOC_PROCESS	Stages the `INVENTORY_OOS.csv` file; disable if not using this file.
STG_CSV_INV_LOAD_ADHOC_PROCESS	Stages the `INVENTORY.csv` file; disable if not using this file.
STG_CSV_INVPK_LOAD_ADHOC_PROCESS	Stages the `INVENTORY_PACK.csv` file; disable if not using this file.
STG_PRICE_LOAD_ADHOC_PROCESS	Stages the `W_RTL_PRICE_IT_LC_DY_FS.dat` file; disable if not using this file.
STG_CSV_PRICE_LOAD_ADHOC_PROCESS	Stages the `PRICE.csv` file; disable if not using this file.
STG_ONORD_ADHOC_PROCESS	Stages the `W_RTL_PO_ONORD_IT_LC_DY_FS.dat` file; disable if not using this file.
STG_CSV_ONORD_ADHOC_PROCESS	Stages the `ORDER_DETAIL.csv` file; disable if not using this file.
STG_INVU_LOAD_ADHOC_PROCESS	Stages the `W_RTL_INVU_IT_LC_DY_FS.dat` file; disable if not using this file.
STG_SALES_WF_ADHOC_PROCESS	Stages the `W_RTL_SLSWF_IT_LC_DY_FS.dat` file; disable if not using this file.
STG_CSV_SALES_WF_ADHOC_PROCESS	Stages the `SALES_WF.csv` file; disable if not using this file.
RESET_ETL_THREAD_VAL_ADHOC_PROCESS	Required; updates your staging data to remove any multi-threading parameter values that are no longer supported.
ADHOC_REFRESH_RADM_PROCESS	Required; collects statistics on your tables before starting the target table loads.
SALES_LOAD_ADHOC_PROCESS	Processes all the data from Sales and Sales Pack files (CSV or DAT) into the target data warehouse tables and aggregates.

Process Name	Usage
RTV_LOAD_ADHOC_PROCESS	Processes all the data from the RTV file (CSV or DAT) into the target data warehouse tables and aggregates.
INVADJ_LOAD_ADHOC_PROCESS	Processes all the data from the Adjustment file (CSV or DAT) into the target data warehouse tables and aggregates.
INVRC_LOAD_ADHOC_PROCESS	Processes all the data from the Receipt file (CSV or DAT) into the target data warehouse tables and aggregates.
INVTSF_LOAD_ADHOC_PROCESS	Processes all the data from the Transfer file (CSV or DAT) into the target data warehouse tables and aggregates.
MARKDOWN_LOAD_ADHOC_PROCESS	Processes all the data from the Markdown file (CSV or DAT) into the target data warehouse tables and aggregates.
DEAL_INCOME_ADHOC_PROCESS	Processes all the data from the Deal Income file (CSV or DAT) into the target data warehouse tables and aggregates.
INV_LOAD_ADHOC_PROCESS	Processes all the data from the Inventory file (CSV or DAT) into the target data warehouse tables and aggregates.
INVPK_LOAD_ADHOC_PROCESS	Processes all the data from the Inventory Pack file (CSV or DAT) into the target data warehouse tables and aggregates.
INVOOS_LOAD_ADHOC_PROCESS	Processes all the data from the Inventory OOS file into the target data warehouse tables and aggregates.
PRICE_LOAD_ADHOC_PROCESS	Processes all the data from the Price file (CSV or DAT) into the target data warehouse tables and aggregates.
ONORD_LOAD_ADHOC_PROCESS	Processes all the data from the Purchase Order file (CSV or DAT) into the target data warehouse tables and aggregates.
INVU_LOAD_ADHOC_PROCESS	Processes all the data from the Unavailable Inventory file into the target data warehouse tables and aggregates.
SALES_WF_ADHOC_PROCESS	Processes all the data from the Sales Wholesale/ Franchise file (CSV or DAT) into the target data warehouse tables and aggregates.
FACT_FLOW_END_PROCESS	Required; non-functional jobs for logging/ dependency purposes.
FLOW_LOAD_END_PROCESS	Required; non-functional jobs for logging/ dependency purposes.

Once you've disabled all the unused jobs at the process level, you may want to review the remaining jobs and disable table loads that are not needed for your implementation. Unless you are implementing Retail Insights, you do not need any aggregate table above the item/ location/week level intersections. For example, you will need all sales tables containing `IT_LC_DY` or `IT_LC_WK` for item/loc/day and week intersections. You may not need the sales aggregates such as `W_RTL_SLS_SC_DY_A` which is a BI aggregate for reporting, unless you are implementing Retail Insights.

If you are calling the process from Postman or Curl, then you may use a payload like the following to trigger the process flow:

```
{
  "cycleName" : "Adhoc",
  "flowName" : "RI_FLOW_ADHOC",
  "requestType" : "POM Scheduler"
}
```

Initial Base Cost Seeding

Module Name	SEED_CSV_W_RTL_BCOST_IT_LC_DY_F_PROCESS_ADHOC
Description	Loads a full snapshot of base cost data from `COST.csv` to initialize the positional data before a nightly batch can be enabled.
Dependencies	HIST_ZIP_FILE_LOAD_ADHOC
Business Activity	Nightly Batch Preparation

Design Overview

The seeding load process for Base Cost data accepts an input file at the item-location-date-supplier level using the file specification for `COST.csv`. It assumes the file has already been moved into place by the `HIST_ZIP_FILE_LOAD_ADHOC` process. This process imports the file into a preprocessing table in the database, transforms it to RI's internal staging tables, then loads it into the base fact (item/location/day). This process is only for the base cost, a separate process loads the net cost, if required.

Note:

Seeding processes require a full snapshot of data for a single date, which covers all item/location combinations that should have a starting position for this fact. The seeding process must load data for the day before the nightly batch is going to run. Alternatively, you can include the full snapshots of data in your very first nightly batch and skip the seeding steps. This causes the nightly batch to take a significantly longer time to execute but avoids the manual load processes for all the positional facts.

Key Tables Affected

Table	Usage
W_COST_FTS	File Input
W_RTL_BCOST_IT_LC_DY_FS	Staging
W_RTL_BCOST_IT_LC_G	Output
W_RTL_BCOST_IT_LC_DY_F	Output

Initial Base Cost Seeding (Legacy)

Module Name	SEED_W_RTL_BCOST_IT_LC_DY_F_PROCESS_ADHOC
Description	Loads a full snapshot of base cost data from `W_RTL_BCOST_IT_LC_DY_FS.dat` to initialize the positional data before a nightly batch can be enabled.
Dependencies	HIST_ZIP_FILE_LOAD_ADHOC
Business Activity	Nightly Batch Preparation

Design Overview

The seeding load process for Base Cost data accepts an input file at the item-location-date-supplier level using the file specification for `W_RTL_BCOST_IT_LC_DY_FS.dat`. It assumes the file has already been moved into place using the `HIST_ZIP_FILE_LOAD_ADHOC` process. This process imports the file into a preprocessing table in the database, transforms it to RI's internal staging tables, then loads it into the base fact (item/location/day).

Note:

Seeding processes require a full snapshot of data for a single date, which covers all item/location combinations that should have a starting position for this fact. The seeding process must load data for the day before the nightly batch runs. Alternatively, you can include the full snapshots of data in your very first nightly batch and skip the seeding steps. This causes the nightly batch to take a significantly longer time to execute but avoids the manual load processes for all the positional facts.

Key Tables Affected

Table	Usage
W_RTL_BCOST_IT_LC_DY_FS	Staging
W_RTL_BCOST_IT_LC_G	Output
W_RTL_BCOST_IT_LC_DY_F	Output

Initial Calendar Load

Module Name	CALENDAR_LOAD_ADHOC
Description	Runs all calendar creation and load processes to set up or update the system and fiscal calendars in the data warehouse.
Dependencies	HIST_ZIP_FILE_LOAD_ADHOC
Business Activity	Initial System Setup

Design Overview

The calendar load ad hoc process performs all the necessary stage, transform, and load jobs to set up the data warehouse calendars. It takes as input:

1. A calendar data file (`CALENDAR.csv`) uploaded and unpacked using the `HIST_ZIP_FILE_LOAD_ADHOC` process
2. Optional last-year mapping files to define shifted and unshifted calendars when reporting on LY data
3. System calendar start and end dates in `C_ODI_PARAM`

The calendar data must be in the form of a fiscal calendar (for example, a 4-5-4 or 13-period calendar). It must be at the period level of detail (not the day level) and should include start and end date ranges for the period, quarter, and year levels on each record. The RAP data warehouse currently supports a single, hard-coded calendar ID (Retail Calendar~41) that should be used in the file's first column (`MCAL_CAL_ID`). Optional mapping files for this-year-to-last-year mappings may be provided if the business uses a custom definition of LY in reporting and analytics. These mappings control which range of dates are returned when pulling LY metrics in RI, such as when a fiscal week in the current year should be mapped to a different week in LY. Default mappings are created by the process if no data is provided.

This process populates the Gregorian system calendar at the same time the fiscal calendar is loaded. The Gregorian calendar requires additional start and end date parameters from `C_ODI_PARAM` to define the time range to generate. It must be greater than the range of time in the fiscal calendar. The calendar generation process does not support a 53-week year as the starting year, so it's recommended to make the start date of the Gregorian calendar at least 1 year earlier than the start of the fiscal calendar, which avoids improperly formed data in the fiscal calendar if the 53-week year is the first year.

Output tables that start with `W_MCAL_` are mainly used for fiscal calendar generation, while the other tables such as `W_DAY_D` are used for the Gregorian calendar. All output tables must be successfully populated with calendar data in order to use the platform. Validate the data closely after running this process to ensure nothing is missing or incorrect in the generated calendar data.

Key Tables Affected

Table	Usage
W_MCAL_PERIOD_DTS	Input
W_RTL_MCAL_DAY_SHIFT_DS	Input
W_RTL_MCAL_DAY_UNSHIFT_DS	Input
W_RTL_MCAL_DAY_GUNSHIFT_DS	Input
W_RTL_MCAL_WEEK_SHIFT_DS	Input
W_RTL_MCAL_WEEK_UNSHIFT_DS	Input
W_MCAL_PERIOD_DS	Staging
W_TIME_OF_DAY_D	Output
W_DAY_D	Output
W_YEAR_D	Output
W_QTR_D	Output

Table	Usage
W_MONTH_D	Output
W_WEEK_D	Output
W_MINUTE_OF_DAY_D	Output
W_MCAL_CONFIG_G	Output
W_MCAL_CAL_D	Output
W_MCAL_PERIOD_D	Output
W_MCAL_DAY_D	Output
W_MCAL_WEEK_D	Output
W_MCAL_YEAR_D	Output
W_MCAL_QTR_D	Output
W_RTL_MCAL_DAY_SHIFT_D	Output
W_RTL_MCAL_DAY_UNSHIFT_D	Output
W_RTL_MCAL_DAY_GUNSHIFT_D	Output
W_RTL_MCAL_DAY_CUSTOM_D	Output
W_RTL_MCAL_WEEK_SHIFT_D	Output
W_RTL_MCAL_WEEK_UNSHIFT_D	Output
W_RTL_MCAL_PERIOD_SHIFT_D	Output
W_RTL_MCAL_PERIOD_UNSHIFT_D	Output

Initial Calendar Staging (Legacy)

Module Name	CALENDAR_STG_LOAD_ADHOC
Description	Stages the `W_MCAL_PERIOD_DS.dat` file for the ad hoc calendar load programs.
Dependencies	HIST_ZIP_FILE_LOAD_ADHOC
Business Activity	Initial System Setup

Design Overview

This process looks for the `W_MCAL_PERIOD_DS.dat` file placed on the server by a history zip file upload and imports it to a staging table for use in the `CALENDAR_LOAD_ADHOC` process.

Key Tables Affected

Table	Usage
W_MCAL_PERIODS_DS	File Input

Initial Competitor Price Seeding

Module Name	SEED_W_RTL_COMP_PRICE_IT_LC_DY_F_PROCESS_ADHOC

Description	Loads a full snapshot of competitor price data from `W_RTL_COMP_PRICE_IT_LC_DY_FS.dat` to initialize the positional data before a nightly batch can be enabled.
Dependencies	HIST_ZIP_FILE_LOAD_ADHOC
Business Activity	Nightly Batch Preparation

Design Overview

The seeding load process for Competitor Price fact data accepts an input file at the item-location-competitor store-date level using the file specification for `W_RTL_COMP_PRICE_IT_LC_DY_FS.dat`. It assumes the file has already been moved into place by the `HIST_ZIP_FILE_LOAD_ADHOC` process. This process imports the file into a staging table, then loads it into the base fact (item/location/comp store/day).

Note:

Seeding processes require a full snapshot of data for a single date, which covers all item/location combinations that should have a starting position for this fact. The seeding process must load data for the day before the nightly batch runs. Alternatively, you can include the full snapshots of data in your very first nightly batch and skip the seeding steps. This causes the nightly batch to take a significantly longer time to execute but avoids the manual load processes for all the positional facts.

Key Tables Affected

Table	Usage
W_RTL_COMP_PRICE_IT_LC_DY_FS	File Input
W_RTL_COMP_PRICE_IT_LC_DY_F	Output
W_RTL_COMP_STORE_DS	File Input
W_RTL_COMP_STORE_D	Output

Initial Dimension Load

Module Name	RI_DIM_INITIAL_ADHOC
Description	Runs all core dimension load programs in AIF DATA schedule in POM to stage, transform, and load dimension data to the foundation data warehouse tables.
Dependencies	HIST_ZIP_FILE_LOAD_ADHOC
Business Activity	Historical Data Load

Design Overview

This process runs the dimension load programs needed to initialize the data model with the core dataset needed for history and seed loads. Not all dimensions supported by RI or AIF are part of the initial load process, only those that are used in some way for history or downstream application processing. The process will stage and load all the files in a single flow; no other processes are needed to load the dimensions. The jobs used by the process are the same as the ones in the nightly batch so this also validates the file quality and correctness.

The process has three distinct types of jobs:

- File import jobs that take a CSV input and load it to the database pre-staging tables (usually tables ending in `DTS` or `FTS`)
- Staging jobs that transform the raw inputs to the required formats and perform any defaulting of values on data columns
- Load jobs that move the staging data to internal target tables

The tables below are broken out by each type, so you can review the inputs and outputs for each block of jobs.

Files to Pre-Staging Tables

Input File	Output Table
PRODUCT.csv	W_PRODUCT_DTS
PRODUCT_ALT.csv	W_PRODUCT_ALT_DTS
ORGANIZATION.csv	W_INT_ORG_DTS
ORGANIZATION_ALT.csv	W_ORGANIZATION_ALT_DTS
EXCH_RATE.csv	W_EXCH_RATE_DTS
CALENDAR.csv	W_MCAL_PERIODS_DTS
SUPPLIER.csv	W_SUPPLIER_DTS
EMPLOYEE.csv	W_EMPLOYEE_DTS
PROD_LOC_ATTR.csv	W_PROD_LOC_ATTR_DTS
PROD_LOC_REPL.csv	W_INVENTORY_PRODUCT_ATTR_DTS
ATTR.csv	W_ATTR_DTS
PROD_ATTR.csv	W_PRODUCT_ATTR_DTS
SEASON.csv	W_RTL_SEASON_PHASE_DTS
PROD_SEASON.csv	W_RTL_SEASON_PHASE_IT_DTS
STORE_COMP.csv	W_RTL_LOC_COMP_MTX_DTS
CODES.csv	W_RTL_CODE_DTS
PROD_PACK.csv	W_RTL_ITEM_GRP2_DTS
DIFF_GROUP.csv	W_DIFF_GROUP_DTS
ADJUSTMENT.csv	W_ADJUSTMENT_FTS
PROMOTION.csv	W_RTL_PROMO_EXT_DTS
PROMO_DETAIL.csv	W_RTL_PROMO_IT_LC_DTS
ORDER_HEAD.csv	W_ORDER_HEAD_FTS
REPL_DISTRO.csv	W_RTL_REPL_DISTRO_IT_LC_DS
REPL_REV_INT.csv	W_RTL_REPL_REV_INT_IT_LC_DS
REPL_LT_INT.csv	W_RTL_REPL_LT_INT_IT_LC_DS

Pre-Staging to Staging Tables

These processes apply all of the transformation scripts needed to take simplified interface (SI) data for dimensions and map it to the internal data model staging tables. The simplified interfaces are a one-to-many mapping to the internal data warehouse structures for dimensions, so this intermediate step is required to transform the incoming data and make it usable downstream.

Input Table	Output Table
W_PRODUCT_DTS	W_PROD_CAT_DHS
W_PRODUCT_DTS	W_PRODUCT_ATTR_DS
W_PRODUCT_DTS	W_PRODUCT_DS
W_PRODUCT_DTS	W_PRODUCT_DS_TL
W_PRODUCT_DTS	W_RTL_PRODUCT_BRAND_DS
W_PRODUCT_DTS	W_RTL_PRODUCT_BRAND_DS_TL
W_PRODUCT_DTS	W_RTL_IT_SUPPLIER_DS
W_PRODUCT_DTS	W_DOMAIN_MEMBER_DS_TL
W_PRODUCT_ALT_DTS	W_PRODUCT_FLEX_DS
W_INT_ORG_DTS	W_INT_ORG_DS
W_INT_ORG_DTS	W_INT_ORG_DS_TL
W_INT_ORG_DTS	W_INT_ORG_DHS
W_INT_ORG_DTS	W_DOMAIN_MEMBER_DS_TL
W_INT_ORG_DTS	W_RTL_CHANNEL_DS
W_INT_ORG_DTS	W_INT_ORG_ATTR_DS
W_INT_ORG_DTS	W_RTL_CHANNEL_CNTRY_DS
W_ORGANIZATION_ALT_DTS	W_ORGANIZATION_FLEX_DS
W_EXCH_RATE_DTS	W_EXCH_RATE_GS
W_MCAL_PERIODS_DTS	W_MCAL_PERIOD_DS
W_SUPPLIER_DTS	W_PARTY_ATTR_DS
W_SUPPLIER_DTS	W_PARTY_ORG_DS
W_EMPLOYEE_DTS	W_EMPLOYEE_DS
W_PROD_LOC_ATTR_DTS	W_RTL_IT_LC_DS
W_INVENTORY_PRODUCT_ATTR_DTS	W_INVENTORY_PRODUCT_ATTR_DS
W_ATTR_DTS	W_RTL_PRODUCT_ATTR_DS
W_ATTR_DTS	W_RTL_PRODUCT_ATTR_DS_TL
W_ATTR_DTS	W_DOMAIN_MEMBER_DS_TL
W_ATTR_DTS	W_RTL_PRODUCT_COLOR_DS
W_PRODUCT_ATTR_DTS	W_RTL_ITEM_GRP1_DS
W_RTL_SEASON_PHASE_DTS	W_RTL_SEASON_DS
W_RTL_SEASON_PHASE_DTS	W_RTL_PHASE_DS
W_RTL_SEASON_PHASE_DTS	W_DOMAIN_MEMBER_DS_TL
W_RTL_SEASON_PHASE_IT_DTS	W_RTL_SEASON_PHASE_IT_DS
W_RTL_LOC_COMP_MTX_DTS	W_RTL_LOC_COMP_MTX_DS

Input Table	Output Table
W_RTL_CODE_DTS	W_RTL_CODE_DS
W_RTL_ITEM_GRP2_DTS	W_RTL_ITEM_GRP2_DS
W_DIFF_GROUP_DTS	W_RTL_DIFF_GRP_DS
W_DIFF_GROUP_DTS	W_RTL_DIFF_GRP_DS_TL
W_ADJUSTMENT_FTS	W_REASON_DS
W_ADJUSTMENT_FTS	W_DOMAIN_MEMBER_DS_TL
W_RTL_PROMO_EXT_DTS	W_RTL_PROMO_EXT_DS
W_RTL_PROMO_IT_LC_DTS	W_RTL_PROMO_IT_LC_DS
W_ORDER_HEAD_FTS	W_RTL_PO_DETAILS_DS

Staging to Target Tables

Input Table	Output Table
W_DOMAIN_MEMBER_DS_TL	W_DOMAIN_MEMBER_LKP_TL
W_EMPLOYEE_DS	W_EMPLOYEE_D
W_EXCH_RATE_GS	W_EXCH_RATE_G
W_INT_ORG_DS	W_INT_ORG_D
W_INT_ORG_DHS	W_INT_ORG_DH
W_ORGANIZATION_FLEX_DS	W_ORGANIZATION_FLEX_D
W_PARTY_ATTR_DS	W_PARTY_ATTR_D
W_PARTY_ORG_DS	W_PARTY_ORG_D
W_PARTY_PER_DS	W_PARTY_PER_D
W_PROD_CAT_DHS	W_PROD_CAT_DH
W_PRODUCT_ATTR_DS	W_PRODUCT_ATTR_D
W_PRODUCT_DS	W_PRODUCT_D
W_PRODUCT_FLEX_DS	W_PRODUCT_FLEX_D
W_REASON_DS	W_REASON_D
W_RTL_ALC_DETAILS_DS	W_RTL_ALC_DETAILS_D
W_RTL_BUYER_DS	W_RTL_BUYER_D
W_RTL_CHANNEL_DS	W_RTL_CHANNEL_D
W_RTL_CHANNEL_CNTRY_DS	W_RTL_CHANNEL_CNTRY_D
W_RTL_CO_HEAD_DS	W_RTL_CO_HEAD_D
W_RTL_CO_LINE_DS	W_RTL_CO_LINE_D
W_RTL_CO_SHIP_METHOD_DS	W_RTL_CO_SHIP_METHOD_D
W_RTL_CO_SHIP_TYPE_DS	W_RTL_CO_SHIP_TYPE_D
W_RTL_COMP_STORE_DS	W_RTL_COMP_STORE_D
W_RTL_CONS_METADATA_GS	W_RTL_CONS_METADATA_G
W_RTL_COUPON_DS	W_RTL_COUPON_D
W_RTL_DIFF_GRP_DS	W_RTL_DIFF_GRP_D
W_RTL_DIFF_RNG_DS	W_RTL_DIFF_RNG_D

Input Table	Output Table
W_RTL_DISCOUNT_TYPE_DS	W_RTL_DISCOUNT_TYPE_D
W_RTL_IT_SUPPLIER_DS	W_RTL_IT_SUPPLIER_D
W_RTL_ITEM_GRP1_DS	W_RTL_ITEM_GRP1_D
W_RTL_LOC_STOCK_CNT_DS	W_RTL_LOC_STOCK_CNT_D
W_RTL_NON_MERCH_CODE_DS	W_RTL_NON_MERCH_CODE_D
W_RTL_ORG_FIN_DS	W_RTL_ORG_FIN_D
W_RTL_PHASE_DS	W_RTL_PHASE_D
W_RTL_PO_DETAILS_DS	W_RTL_PO_DETAILS_D
W_RTL_PRICE_CLR_IT_LC_DS	W_RTL_PRICE_CLR_IT_LC_D
W_RTL_PRODUCT_ATTR_DS	W_RTL_PRODUCT_ATTR_D
W_RTL_PRODUCT_BRAND_DS	W_RTL_PRODUCT_BRAND_D
W_RTL_PROMO_DS_TL	W_RTL_PROMO_D_TL
W_RTL_PROMO_IT_LC_DS	W_RTL_PROMO_IT_LC_D
W_RTL_PROMO_CE_IT_LC_DS	W_RTL_PROMO_IT_LC_D
W_RTL_REPL_DISTRO_IT_LC_DS	W_RTL_REPL_DISTRO_IT_LC_D
W_RTL_REPL_REV_INT_IT_LC_DS	W_RTL_REPL_REV_INT_IT_LC_D
W_RTL_REPL_LT_INT_IT_LC_DS	W_RTL_REPL_LT_INT_IT_LC_D
W_RTL_SEASON_DS	W_RTL_SEASON_D
W_RTL_SEASON_PHASE_IT_DS	W_RTL_SEASON_PHASE_IT_D
W_RTL_TNDR_TYPE_DS	W_RTL_TNDR_TYPE_D
W_STATUS_DS	W_STATUS_D

Initial Inventory Pack Seeding

Module Name	SEED_CSV_W_RTL_INVPK_IT_LC_DY_F_PROCESS_ADHOC
Description	Loads a full snapshot of inventory pack data from `INVENTORY_PACK.csv` to initialize the positional data before a nightly batch can be enabled.
Dependencies	HIST_ZIP_FILE_LOAD_ADHOC
Business Activity	Nightly Batch Preparation

Design Overview

The seeding load process for Inventory Pack data accepts an input file at the item-location-date level using the file specification for `INVENTORY_PACK.csv`. It assumes the file has already been moved into place by the `HIST_ZIP_FILE_LOAD_ADHOC` process. This process imports the file into a preprocessing table in the database, transforms it to internal staging tables, then loads it into the base fact (item/location/day).

Note:

Seeding processes require a full snapshot of data for a single date. This snapshot needs to cover all item/location combinations that should have a starting position for this fact. The seeding process must load data for the day before the nightly batch runs. Alternatively, you can include the full snapshots of data in your first nightly batch and skip the seeding steps. This causes the nightly batch to take a significantly longer time to execute, but avoids the manual load processes for all the positional facts.

Key Tables Affected

Table	Usage
W_RTL_INVPK_IT_LC_DY_FTS	File Input
W_RTL_INVPK_IT_LC_DY_FS	Staging
W_RTL_INVPK_IT_LC_G	Output
W_RTL_INVPK_IT_LC_DY_F	Output
W_RTL_INVPK_IT_LC_WK_A	Output

Initial Inventory Seeding

Module Name	SEED_CSV_W_RTL_INV_IT_LC_DY_F_PROCESS_ADHOC
Description	Loads a full snapshot of inventory data from `INVENTORY.csv` to initialize the positional data before a nightly batch can be enabled.
Dependencies	HIST_ZIP_FILE_LOAD_ADHOC
Business Activity	Nightly Batch Preparation

Design Overview

The seeding load process for Inventory data accepts an input file at the item-location-date level using the file specification for `INVENTORY.csv`. It assumes the file has already been moved into place by the `HIST_ZIP_FILE_LOAD_ADHOC` process. This process imports the file into a preprocessing table in the database, transforms it to internal staging tables, then loads it into the base fact (item/location/day).

Note:

Seeding processes require a full snapshot of data for a single date, which covers all item/location combinations that should have a starting position for this fact. The seeding process must load data for the day before the nightly batch runs. Alternatively, you can include the full snapshots of data in your first nightly batch and skip the seeding steps. This causes the nightly batch to take a significantly longer time to execute, but avoids the manual load processes for all the positional facts.

Key Tables Affected

Table	Usage
W_RTL_INV_IT_LC_DY_FTS	File Input
W_RTL_INV_IT_LC_DY_FS	Staging
W_RTL_INV_IT_LC_G	Output
W_RTL_INV_IT_LC_DY_F	Output

Initial Inventory Seeding (Legacy)

Module Name	SEED_W_RTL_INV_IT_LC_DY_F_PROCESS_ADHOC
Description	Loads a full snapshot of inventory data from `W_RTL_INV_IT_LC_DY_FS.dat` to initialize the positional data before a nightly batch can be enabled.
Dependencies	HIST_ZIP_FILE_LOAD_ADHOC
Business Activity	Nightly Batch Preparation

Design Overview

The seeding load process for Inventory data accepts an input file at the item-location-date level using the file specification for `W_RTL_INV_IT_LC_DY_FS.dat`. It assumes the file has already been moved into place by the `HIST_ZIP_FILE_LOAD_ADHOC` process. This process imports the file to internal staging tables, then load it into the base fact (item/location/day).

Note:

Seeding processes require a full snapshot of data for a single date, which covers all item/location combinations that should have a starting position for this fact. The seeding process must load data for the day before the nightly batch runs. Alternatively, you can include the full snapshots of data in your very first nightly batch and skip the seeding steps. This causes the nightly batch to take a significantly longer time to execute but avoids the manual load processes for all the positional facts.

Key Tables Affected

Table	Usage
W_RTL_INV_IT_LC_DY_FS	File Input
W_RTL_INV_IT_LC_G	Output
W_RTL_INV_IT_LC_DY_F	Output

Initial Net Cost Seeding

Module Name	SEED_CSV_W_RTL_NCOST_IT_LC_DY_F_PROCESS_ADHOC
Description	Loads a full snapshot of net cost data from `COST.csv` to initialize the positional data before a nightly batch can be enabled.
Dependencies	HIST_ZIP_FILE_LOAD_ADHOC
Business Activity	Nightly Batch Preparation

Design Overview

The seeding load process for Net Cost data accepts an input file at the item-location-date-supplier level using the file specification for `COST.csv`. It assumes the file has already been moved into place by the `HIST_ZIP_FILE_LOAD_ADHOC` process. This process imports the file into a preprocessing table in the database, transforms it to internal staging tables, then loads it into the base fact (item/location/day). This process is only for the net cost; a separate process loads the base cost, if required.

Note:

Seeding processes require a full snapshot of data for a single date, which covers all item/location combinations that should have a starting position for this fact. The seeding process must load data for the day before the nightly batch runs. Alternatively, you can include the full snapshots of data in your very first nightly batch and skip the seeding steps. This causes the nightly batch to take a significantly longer time to execute, but avoids the manual load processes for all the positional facts.

Key Tables Affected

Table	Usage
W_COST_FTS	File Input
W_RTL_NCOST_IT_LC_DY_FS	Staging
W_RTL_NCOST_IT_LC_G	Output
W_RTL_NCOST_IT_LC_DY_F	Output

Initial Net Cost Seeding (Legacy)

Module Name	SEED_W_RTL_NCOST_IT_LC_DY_F_PROCESS_ADHOC
Description	Loads a full snapshot of net cost data from `W_RTL_NCOST_IT_LC_DY_FS.dat` to initialize the positional data before a nightly batch can be enabled.
Dependencies	HIST_ZIP_FILE_LOAD_ADHOC
Business Activity	Nightly Batch Preparation

Design Overview

The seeding load process for Net Cost data accepts an input file at the item-location-date-supplier level using the file specification for `W_RTL_NCOST_IT_LC_DY_FS.dat`. It assumes the file has already been moved into place using the `HIST_ZIP_FILE_LOAD_ADHOC` process. This process imports the file into a preprocessing table in the database, transforms it to internal staging tables, then loads it into the base fact (item/location/day).

Note:

Seeding processes require a full snapshot of data for a single date, which covers all item/location combinations that should have a starting position for this fact. The seeding process must load data for the day before the nightly batch runs. Alternatively, you can include the full snapshots of data in your very first nightly batch and skip the seeding steps. This causes the nightly batch to take a significantly longer time to execute, but avoids the manual load processes for all the positional facts.

Key Tables Affected

Table	Usage
W_RTL_NCOST_IT_LC_DY_FS	File Input
W_RTL_NCOST_IT_LC_G	Output
W_RTL_NCOST_IT_LC_DY_F	Output

Initial Price Seeding

Module Name	SEED_CSV_W_RTL_PRICE_IT_LC_DY_F_PROCESS_ADHOC
Description	Loads a full snapshot of price data from `PRICE.csv` to initialize the positional data before a nightly batch can be enabled.
Dependencies	HIST_ZIP_FILE_LOAD_ADHOC
Business Activity	Nightly Batch Preparation

Design Overview

The seeding load process for Price data accepts an input file at the item-location-date level using the file specification for `PRICE.csv`. It assumes the file has already been moved into place by the `HIST_ZIP_FILE_LOAD_ADHOC` process. This process imports the file into a preprocessing table in the database, transforms it to internal staging tables, then loads it into the base fact (item/location/day).

Note:

Seeding processes require a full snapshot of data for a single date, which covers all item/location combinations that should have a starting position for this fact. The seeding process must load data for the day before the nightly batch runs. Alternatively, you can include the full snapshots of data in your very first nightly batch and skip the seeding steps. This causes the nightly batch to take a significantly longer time to execute, but avoids the manual load processes for all the positional facts.

Key Tables Affected

Table	Usage
W_RTL_PRICE_IT_LC_DY_FTS	File Input
W_RTL_PRICE_IT_LC_DY_FS	Staging
W_RTL_PRICE_IT_LC_G	Output
W_RTL_PRICE_IT_LC_DY_F	Output

Initial Price Seeding (Legacy)

Module Name	SEED_W_RTL_PRICE_IT_LC_DY_F_PROCESS_ADHOC
Description	Loads a full snapshot of price data from `W_RTL_PRICE_IT_LC_DY_FS.dat` to initialize the positional data before a nightly batch can be enabled.
Dependencies	HIST_ZIP_FILE_LOAD_ADHOC
Business Activity	Nightly Batch Preparation

Design Overview

The seeding load process for Price data accepts an input file at the item-location-date level using the file specification for `W_RTL_PRICE_IT_LC_DY_FS.dat`. It assumes the file has already been moved into place by the `HIST_ZIP_FILE_LOAD_ADHOC` process. This process imports the file into a staging table, then loads it into the base fact (item/location/day).

Note:

Seeding processes require a full snapshot of data for a single date, which covers all item/location combinations that should have a starting position for this fact. The seeding process must load data for the day before the nightly batch runs. Alternatively, you can include the full snapshots of data in your very first nightly batch and skip the seeding steps. This causes the nightly batch to take a significantly longer time to execute but avoids the manual load processes for all the positional facts.

Key Tables Affected

Table	Usage
W_RTL_PRICE_IT_LC_DY_FS	File Input
W_RTL_PRICE_IT_LC_G	Output
W_RTL_PRICE_IT_LC_DY_F	Output

Initial Purchase Order Seeding

Module Name	SEED_CSV_W_RTL_PO_ONORD_IT_LC_DY_F_PROCESS_ADHOC
Description	Loads a full snapshot of purchase order data from `ORDER_HEAD.csv` and `ORDER_DETAIL.csv` to initialize the positional data before a nightly batch can be enabled.
Dependencies	HIST_ZIP_FILE_LOAD_ADHOC
Business Activity	Nightly Batch Preparation

Design Overview

The seeding load process for Purchase Order data accepts two input files at the order header and order detail levels using the file specifications for `ORDER_HEAD.csv` and `ORDER_DETAIL.csv`. It assumes the files have already been moved into place by the `HIST_ZIP_FILE_LOAD_ADHOC` process. This process imports the files into preprocessing tables in the database, transforms them to internal staging tables, then loads them into the base dimension and facts. The dimension is loaded first to support loading the fact table against those foreign keys.

Note:

Seeding processes require a full snapshot of data for a single date, which covers all purchase orders and item/location combinations that should have a starting position for this fact. The seeding process must load data for the day before the nightly batch runs. Alternatively, you can include the full snapshots of data in your very first nightly batch and skip the seeding steps. This causes the nightly batch to take a significantly longer time to execute but avoids the manual load processes for all the positional facts.

Key Tables Affected

Table	Usage
W_ORDER_HEAD_FTS	File Input
W_RTL_PO_DETAILS_DS	Staging
W_RTL_PO_DETAILS_D	Output
W_ORDER_DETAIL_FTS	File Input
W_RTL_PO_ONORD_IT_LC_DY_FS	Staging

Table	Usage
W_RTL_PO_ONORD_IT_LC_DY_F	Output

Initial Purchase Order Seeding (Legacy)

Module Name	SEED_W_RTL_PO_ONORD_IT_LC_DY_F_PROCESS_ADHOC
Description	Loads a full snapshot of purchase order data from `W_RTL_PO_ONORD_IT_LC_DY_FS.dat` to initialize the positional data before a nightly batch can be enabled.
Dependencies	HIST_ZIP_FILE_LOAD_ADHOC
Business Activity	Nightly Batch Preparation

Design Overview

The seeding load process for Purchase Order fact data accepts an input file at the item-location-date level using the file specification for `W_RTL_PO_ONORD_IT_LC_DY_FS.dat`. It assumes the file has already been moved into place by the `HIST_ZIP_FILE_LOAD_ADHOC` process. This process imports the file into a staging table, then loads it into the base fact (item/location/day). It assumes the dimension has already been loaded separately using the initial dimension loads.

Note:

Seeding processes require a full snapshot of data for a single date, which covers all item/location combinations that should have a starting position for this fact. The seeding process must load data for the day before the nightly batch runs. Alternatively, you can include the full snapshots of data in your very first nightly batch and skip the seeding steps. This causes the nightly batch to take a significantly longer time to execute but avoids the manual load processes for all the positional facts

Key Tables Affected

Table	Usage
W_RTL_PO_ONORD_IT_LC_DY_FS	File Input
W_RTL_PO_ONORD_IT_LC_DY_F	Output

Initial Purchase Order Allocation Seeding

Module Name	SEED_W_RTL_PO_ONALC_IT_LC_DY_F_PROCESS_ADHOC
Description	Loads a full snapshot of purchase order allocation data from `W_RTL_PO_ONALC_IT_LC_DY_FS.dat` to initialize the positional data before a nightly batch can be enabled.
Dependencies	HIST_ZIP_FILE_LOAD_ADHOC

Business Activity	Nightly Batch Preparation

Design Overview

The seeding load process for Purchase Order Allocation fact data accepts an input file at the item-location-date level using the file specification for `W_RTL_PO_ONALC_IT_LC_DY_FS.dat`. It assumes the file has already been moved into place by the `HIST_ZIP_FILE_LOAD_ADHOC` process. This process imports the file into a staging table, then loads it into the base fact (item/location/day). It assumes the dimension has already been loaded separately using the initial dimension loads.

Note:

Seeding processes require a full snapshot of data for a single date, which covers all item/location combinations that should have a starting position for this fact. The seeding process must load data for the day before the nightly batch runs. Alternatively, you can include the full snapshots of data in your very first nightly batch and skip the seeding steps. This causes the nightly batch to take a significantly longer time to execute but avoids the manual load processes for all the positional facts.

Key Tables Affected

Table	Usage
W_RTL_PO_ONALC_IT_LC_DY_FS	File Input
W_RTL_PO_ONALC_IT_LC_DY_F	Output

Intercompany Margin History Load

Module Name	HIST_CSV_ICMARGIN_LOAD_ADHOC
Description	Loads the `IC_MARGIN.csv` file into the data warehouse and populates key data tables used to integrate with other systems for history data.
Dependencies	HIST_ZIP_FILE_LOAD_ADHOC
Business Activity	Nightly Batch Preparation

Design Overview

The history load process for Intercompany Margin transactions accepts an input file at the item/location/day level using the file specification for `IC_MARGIN.csv`. It assumes the file has already been moved into place by the `HIST_ZIP_FILE_LOAD_ADHOC` process. This process imports the file into a preprocessing table in the database, transforms it to RI's internal staging tables, then loads it into the base fact (item/location/day) as well as the week aggregate used for integrations (item/location/week).

Key Tables Affected

Table	Usage
W_RTL_ICM_IT_LC_DY_FTS	File Input
W_RTL_ICM_IT_LC_DY_FS	Staging
W_RTL_ICM_IT_LC_DY_F	Output (Base Fact)
W_RTL_ICM_IT_LC_WK_A	Output (Aggregate)

Inventory History Current Position Load

Module Name	HIST_INV_GENERAL_LOAD_ADHOC
Description	Copies the ending positions of inventory history for the last week into the General (G) table for the purpose of testing the data and integrations within RAP.
Dependencies	HIST_INV_LOAD_ADHOC
Business Activity	Nightly Batch Preparation

Design Overview

This process takes the final week of inventory data loaded using the `HIST_INV_LOAD_ADHOC` process and copies it into the table for current inventory positions (`W_RTL_INV_IT_LC_G`). This program uses an `INSERT` statement, so it cannot be re-run multiple times without first truncating the table. The purpose of this program is to test any integrations or reports that use this table prior to actually running nightly batches, when it would normally be populated. The most common use case is for Inventory Planning Optimization testing, which uses this table to get the current inventory position during ad hoc and weekly batch runs.

Key Tables Affected

Table	Usage
W_RTL_INV_IT_LC_DY_F	Input
W_RTL_INV_IT_LC_G	Output

Inventory History Load

Module Name	HIST_INV_LOAD_ADHOC
Description	Processes any staged inventory history data for end-of-week snapshots, starting from the last processed week.
Dependencies	HIST_STG_CSV_INV_LOAD_ADHOC
Business Activity	Historical Data Load

Design Overview

The inventory history load process supports loading of end-of-week inventory snapshots over a long period of time to populate the data warehouse with historical data. It requires the inventory data to already be staged into the database by one of the available staging processes. Multiple weeks of inventory can be provided in a single file, though it is recommended to not load more than one month at a time unless the volumes are low. Every record in the data must be for a week-ending date; other dates in the file will not work using this process.

The inventory history processes in POM support loading data for two datasets, inventory positions and inventory pack positions. When using CSV file formats, this is the `INVENTORY.csv` and `INVENTORY_PACK.csv` files. When using DAT file formats, this is the `W_RTL_INV_IT_LC_DY_FS.dat` and `W_RTL_INVPK_IT_LC_DY_FS.dat` files.

The `C_HIST_LOAD_STATUS` configuration table controls the actions taken by the process. Before running the process for the first time, you must set up this table for the following:

- Set the history load date (`HIST_LOAD_LAST_DATE`) to be the very latest date you expect to load history for (this can be changed later if needed to load more weeks). The date must be a week-ending date and should have `00:00:00` as the timestamp after saving the date to the database table..
- Disable any aggregate (`_A`) tables you do not wish to populate by setting `ENABLED_IND` to `N`. When loading data only for AI Foundation or Planning, you only need the history temporary table (`W_RTL_INV_IT_LC_DY_HIST_TMP`), base fact (`W_RTL_INV_IT_LC_DY_F`) and week aggregate (`W_RTL_INV_IT_LC_WK_A`). If using inventory pack data, then additionally enable the `TMP`, `F`, and `A` tables for that data flow as well. For RI, all tables should be enabled and loaded.

Once setup is complete, begin processing files from the earliest week-ending date you plan to load. You must start from the beginning of the history and load data sequentially. You cannot load data out of order and you cannot load the same week multiple times without first erasing the data from your database. After a week is loaded successfully, the `C_HIST_LOAD_STATUS` records are updated with the most recent load status and date. You may choose to load both inventory and pack data at the same time or you can run the process only for inventory positions first, and then switch the configurations to run for inventory pack data second.

If you will be loading inventory history after you have already started nightly batches, then you must also change two parameters in `C_ODI_PARAM_VW` from the Control Center:

- `INV_NIGHTLY_BATCH_IND` – Change this to `Y` to indicate that nightly batches have been run but you are planning to load history for prior dates.
- `INV_LAST_HIST_LOAD_DT` – Set this to the final week of history data you plan to load, which must be a week-ending date and must be before the nightly batches were started.

Key Tables Affected

Table	Usage
C_HIST_LOAD_STATUS	Configuration
W_RTL_INV_IT_LC_DY_FS	Input
W_RTL_INVPK_IT_LC_DY_FS	Input
W_RTL_INV_IT_LC_DY_F	Output
W_RTL_INVPK_IT_LC_DY_F	Output

Table	Usage
W_RTL_INV_IT_LC_G	Output
W_RTL_INVPK_IT_LC_G	Output
W_RTL_INV_IT_LC_WK_A	Output
W_RTL_INVPK_IT_LC_WK_A	Output
W_RTL_INV_IT_RG_DY_A	Output
W_RTL_INV_IT_DY_A	Output
W_RTL_INV_IT_WK_A	Output
W_RTL_INV_SC_LC_DY_A	Output
W_RTL_INV_CL_LC_DY_A	Output
W_RTL_INV_DP_LC_DY_A	Output
W_RTL_INV_SC_LC_DY_CUR_A	Output
W_RTL_INV_SC_DY_A	Output
W_RTL_INV_SC_DY_CUR_A	Output
W_RTL_INV_SC_LC_WK_A	Output
W_RTL_INV_CL_LC_WK_A	Output
W_RTL_INV_DP_LC_WK_A	Output
W_RTL_INV_SC_LC_WK_CUR_A	Output
W_RTL_INV_SC_WK_A	Output
W_RTL_INV_SC_WK_CUR_A	Output

Inventory History Staging

Module Name	HIST_STG_CSV_INV_LOAD_ADHOC
Description	Stages the `INVENTORY.csv` and `INVENTORY_PACK.csv` files for the ad hoc inventory load programs.
Dependencies	HIST_ZIP_FILE_LOAD_ADHOC
Business Activity	Historical Data Load

Design Overview

This process looks for the `INVENTORY.csv` and `INVENTORY_PACK.csv` files placed on the server by a history zip file upload, moves them into a preprocessing table, and transforms them for use by the `HIST_INV_LOAD_ADHOC` process.

Note:

The inventory files used for history data must contain only week-ending dates and must be full, weekly snapshots of data.

Key Tables Affected

Table	Usage
W_RTL_INV_IT_LC_DY_FTS	File Input
W_RTL_INVPK_IT_LC_DY_FTS	File Input
W_RTL_INV_IT_LC_DY_FS	Output
W_RTL_INVPK_IT_LC_DY_FS	Output

Inventory History Staging (Legacy)

Module Name	HIST_STG_INV_LOAD_ADHOC
Description	Stages the `W_RTL_INV_IT_LC_DY_FS.dat` and `W_RTL_INVPK_IT_LC_DY_FS.dat` files for the ad hoc inventory load programs.
Dependencies	HIST_ZIP_FILE_LOAD_ADHOC
Business Activity	Historical Data Load

Design Overview

This process looks for the `W_RTL_INV_IT_LC_DY_FS.dat` and `W_RTL_INVPK_IT_LC_DY_FS.dat` files placed on the server by a history ZIP file upload and loads them for use by the `HIST_INV_LOAD_ADHOC` process.

Note:

The inventory files used for history data must contain only week-ending dates and must be full, weekly snapshots of data.

Key Tables Affected

Table	Usage
W_RTL_INV_IT_LC_DY_FS	File Input
W_RTL_INVPK_IT_LC_DY_FS	File Input

Inventory Out of Stock Load

Module Name	HIST_INV_OOS_LOAD_ADHOC
Description	Stages and loads the `INVENTORY_OOS.csv` file for out of stock and outlier indicators.
Dependencies	HIST_ZIP_FILE_LOAD_ADHOC
Business Activity	Historical Data Load

Design Overview

This process looks for the `INVENTORY_OOS.csv` file placed on the server by a history zip file upload, moves it into a preprocessing table, and transforms it into the target table for use in AI Foundation loads.

Note:

The inventory OOS file must contain only week-ending dates; other day dates will not be accepted into the interface

Key Tables Affected

Table	Usage
W_RTL_INVOOS_IT_LC_WK_FS	Input
W_RTL_INVOOS_IT_LC_WK_F	Output

Inventory Reclass History Load

Module Name	HIST_CSV_INVRECLASS_LOAD_ADHOC
Description	Loads the `INV_RECLASS.csv` file into the data warehouse and populates key data tables used to integrate with other systems for history data.
Dependencies	HIST_ZIP_FILE_LOAD_ADHOC
Business Activity	Historical Data Load

Design Overview

The history load process for Inventory Reclass transactions accepts an input file at the item/location/day level using the file specification for `INV_RECLASS.csv`. It assumes the file has already been moved into place by the `HIST_ZIP_FILE_LOAD_ADHOC` process. This process imports the file into a preprocessing table in the database, transforms it to internal staging tables, then loads it into the base fact (item/location/day) as well as the week aggregate used for integrations (item/location/week).

Key Tables Affected

Table	Usage
W_RTL_INVRECLASS_IT_LC_DY_FTS	File Input
W_RTL_INVRECLASS_IT_LC_DY_FS	Staging
W_RTL_INVRECLASS_IT_LC_DY_F	Output (Base Fact)
W_RTL_INVRECLASS_IT_LC_WK_A	Output (Aggregate)

Inventory Refresh from Merchandising

Module Name	RDE_INSERT_FULL_INV_POS_ADHOC
Description	Prepares the RDE nightly extract for inventory positions to pull all possible item/locations for the next run as a means of refreshing RAP application inventory from the source without using separate ad hoc jobs.
Dependencies	None
Business Activity	Nightly Batch Maintenance

Design Overview

This process contains only one job, which is named `RDE_INSERT_FULL_INV_POS_JOB`. The purpose of this job is to query the Merchandising Foundation Cloud Service (MFCS) table `ITEM_LOC` and extract all valid item/locations into a temporary table. The list of item/locations is filtered to the same set of items that the nightly batch would allow from MFCS to RAP (only active and approved items). The next time that the inventory position extract `RDE_EXTRACT_FACT_P7_INVILDSDE_JOB` runs, it will include all of these item/locations even if their inventory values did not change since yesterday. This process is only applicable when MFCS is on version 23 or greater, as the job needs to directly query MFCS data that is being replicated to the Analytics & Planning database.

Key Tables Affected

Table	Usage
ITEM_LOC	MFCS Source Table
ITEM_MASTER	MFCS Source Table
RA_INV_IT_LC_EXT	RDE Temp Table

Inventory Reload

Module Name	INV_RELOAD_PROCESS_ADHOC
Description	Provides an automated way to reload a single week of historical inventory as a way of correcting bad data on the inventory position fact table
Dependencies	None
Business Activity	Data Correction

Design Overview

This process contains only one job, which is named `INV_RELOAD_JOB`. This job deletes a week of data from your inventory position fact tables and inserts the data found on `W_RTL_INV_IT_LC_DY_FS` in its place. The process expects a single week of inventory data (with `DAY_DT` equal to a week-ending date) to be loaded into the `W_RTL_INV_IT_LC_DY_FS` staging table. This job deletes that specific week of data from the `W_RTL_INV_IT_LC_DY_F` and `W_RTL_INV_IT_LC_WK_A` tables and then inserts the staging table data. If you then need to move this data to downstream applications, you must also run the associated ad hoc processes to load inventory data to those solutions. The process makes use of the `C_HIST_LOAD_STATUS` table to manage the data reload so that it overwrites the `HIST_LOAD_LAST_DATE` for tables being processed and may update other fields based on the reload activity. The `W_RTL_INV_IT_LC_DY_F` and `W_RTL_INV_IT_LC_WK_A` tables are always loaded by this process, because all customers are expected to be using these two core tables regardless of the RAP applications they subscribe to.

This process is designed for customers that provide full weekly snapshots of their inventory positions, such that deleting a week of data from the internal tables can be based solely on the `DAY_DT` in the staging table and no other date. This job does not update the `W_RTL_INV_IT_LC_G` current positions, as it assumes you are only trying to reload prior weeks of inventory, not the current week. The data is assumed to be correct in `W_RTL_INV_IT_LC_G`; if it is not, you must push in new inventory data using nightly batch processing to correct any records in the current week. For example, if the current week has some non-zero positions that need to be zeroed out after your data correction, you need to include those zero-balance rows in a nightly batch file so they can be both applied to `W_RTL_INV_IT_LC_G` and updated for the current week in the other data warehouse tables and downstream applications.

Key Tables Affected

Table	Usage
W_RTL_INV_IT_LC_DY_FS	Input Table
W_RTL_INV_IT_LC_DY_F	Target Table
W_RTL_INV_IT_LC_WK_A	Target Table

Inventory Selling Date Seeding

Module Name	LOAD_W_RTL_INV_IT_LC_G_FIRST_SOLD_DT_ADHOC
Description	Calculates the initial value of First Sold Date for all item/locations in inventory, based on sales history data.
Dependencies	SEED_CSV_W_RTL_INV_IT_LC_DY_F_PROCESS_ADHOC
Business Activity	Historical Data Load

Design Overview

This process populates the fields `W_RTL_INV_IT_LC_G.FIRST_SOLD_DT` and `LAST_SOLD_DT` with values, using your historical sales data to calculate the first time each item/location with stock on hand was sold. This process should only run after all inventory and sales history is completely loaded and you are ready to begin nightly batches. If this process does not run, then all item/locations will start with a first/last selling date of the first transaction to occur on it in nightly batch runs. These date values are used by the AI Foundation Cloud Services (Lifecycle Pricing Optimization) as an input to determine item lifecycles from the history data in RI.

Key Tables Affected

Table	Usage
W_RTL_SLS_TRX_IT_LC_DY_F	Input
W_RTL_INV_IT_LC_G	Output

List Server Objects

Module Name	RI_LIST_ENV_OBJECTS_ADHOC_PROCESS
Description	Perform pod listings, get list of files in Object Store, as well as get directory listings of files in the key batch file directories on the application pods.
Dependencies	None
Business Activity	Batch Administration

Design Overview

This process provides a utility job for listing the files present on the application server in certain directories relating to batch processing and data movement. It can list files in object storage and internal server folders, depending on the parameters passed into the job from POM. This process is specifically for the RI application server and associated object storage, which is used for the AIF DATA batches. This process can aid in debugging batch issues where you need to know which files are currently being used on the server or which files have just been placed on the server by recently executed jobs.

As parameters, one of the following must be passed in from the POM UI:

- --ls:<pod directory>
- --ftspath:<object store directory>
- --podlist:<pod name pattern>

For example, you can add the following parameter to the job to get the list of files in the `batch/expand` directory:

```
--ls:batch/expand
```

Markdown History Load

Module Name	HIST_CSV_MARKDOWN_LOAD_ADHOC
Description	Loads the `MARKDOWN.csv` file into the data warehouse and populates key data tables used to integrate with other systems for history data.
Dependencies	HIST_ZIP_FILE_LOAD_ADHOC
Business Activity	Historical Data Load

Design Overview

The history load process for Markdown transactions accepts an input file at the item/location/day level using the file specification for `MARKDOWN.csv`. It assumes the file has already been moved into place by the `HIST_ZIP_FILE_LOAD_ADHOC` process. This process imports the file into a preprocessing table in the database, transforms it to internal staging tables, then loads it into the base fact (item/location/day) as well as the week aggregate used for integrations (item/location/week).

Key Tables Affected

Table	Usage
W_MARKDOWN_FTS	File Input
W_RTL_MKDN_IT_LC_DY_FS	Staging
W_RTL_MKDN_IT_LC_DY_F	Output (Base Fact)
W_RTL_MKDN_IT_LC_WK_A	Output (Aggregate)

Market Data Load

Module Name	HIST_MARKET_LOAD_ADHOC
Description	Loads all dimension and fact data relating to Market Items, Market Attributes, and Market Sales subject areas. These tables are used for Retail Insights reporting and Advanced Clustering (AC) analysis.
Dependencies	HIST_ZIP_FILE_LOAD_ADHOC
Business Activity	Historical Data Load

Design Overview

This process loads all the dimensions, facts, and aggregate tables relating to Market Data. Market data comes from a non-Oracle source and represents consumer buying behavior and products available on the market that are related to your own merchandise. It is used in Retail Insights reporting, as well as specific metrics in Advanced Clustering where you may compare performance of your store clusters against market performance in specific product categories. Market data is updated infrequently, such as once a month or quarter, so you may choose to exclude it from your nightly batches and only load it using this ad hoc process.

Key Tables Affected

Input Table	Target Table
W_RTL_MARKET_PRODUCT_DS	W_RTL_MARKET_PRODUCT_D
W_RTL_MARKET_PROD_DHS	W_RTL_MARKET_PROD_DH
W_RTL_MARKET_PRODUCT_DS_TL	W_RTL_MARKET_PRODUCT_D_TL
W_RTL_MARKET_PROD_BRAND_DS_TL	W_RTL_MARKET_PROD_BRAND_D_TL
W_RTL_MARKET_PRODUCT_MTX_DS	W_RTL_MARKET_PRODUCT_MTX_D

Input Table	Target Table
W_RTL_MARKET_PROD_ATTR_DS	W_RTL_MARKET_PROD_ATTR_D
W_RTL_MARKET_PROD_ATTR_MTX_DS	W_RTL_MARKET_PROD_ATTR_MTX_D
W_RTL_MARKET_PROD_BRAND_DS	W_RTL_MARKET_PROD_BRAND_D
W_RTL_MARKET_PROD_DH_MTX_DS	W_RTL_MARKET_PROD_DH_MTX_D
W_RTL_MKTSLS_TA_CH_CNG_WK_FS	W_RTL_MKTSLS_TA_CH_CNG_WK_F
W_RTL_MKTSLS_TA_CH_HG_WK_FS	W_RTL_MKTSLS_TA_CH_HG_WK_F
W_RTL_MKTSLS_TA_CH_CNG_WK_F	W_RTL_MKTSLS_TA_CMG_CS_QR_A
W_RTL_MKTSLS_TA_CH_CNG_WK_F	W_RTL_MKTSLS_TA_CMG_QR_A
W_RTL_MKTSLS_TA_CH_CNG_WK_F	W_RTL_MKTSLS_TA_CL_CS_QR_CUR_A
W_RTL_MKTSLS_TA_CH_CNG_WK_F	W_RTL_MKTSLS_TA_CL_QR_CUR_A

Nightly Batch Status Cleanup

Module Name	C_LOAD_DATES_CLEANUP_ADHOC
Description	Erases the execution status of nightly batch programs. This is required to run a nightly process outside of a batch.
Dependencies	None
Business Activity	Batch Administration

Design Overview

This process erases records from the `C_LOAD_DATES` database table. Any time a job runs as part of the nightly batch, or a job runs that is included in both nightly and ad hoc processing, a status record is inserted into `C_LOAD_DATES`. The job is then blocked from executing again while this record exists, as a safety measure when restarting batch processes that failed midway through execution. During initial dimension loads, you may need to execute the same jobs multiple times to work through file or data issues. In that case, you may execute this process before each run to clear the status of prior runs from the database.

Note:

This process should only run during history and initial data loads or at the guidance of Oracle Support. It should not be run during regular nightly batch processing. Clearing `C_LOAD_DATES` while the batch is running normally could cause data corruption, as it would allow the same jobs to run multiple times for the same business date.

Key Tables Affected

Table	Usage
C_LOAD_DATES	Delete

Plan Data Integration

Module Name	LOAD_PLANNING1_DATA_ADHOC LOAD_PLANNING2_DATA_ADHOC LOAD_PLANNING3_DATA_ADHOC LOAD_PLANNING4_DATA_ADHOC LOAD_PLANNING5_DATA_ADHOC
Description	Extracts data from the MFP and AP Plan Export interfaces to RI's internal planning tables.
Dependencies	CLEANUP_C_LOAD_DATES_PLANNING_ADHOC
Business Activity	RI Integrations

Design Overview

This set of processes moves Merchandise Financial Planning (MFP) and Assortment Planning (AP) export data from the data exchange (RDX) layer to internal staging tables, then triggers the AIF DATA load programs for planning data. Each process contains the end-to-end flow of data for a single interface. Use these processes to perform integration testing and plan data validations during an RI and MFP/AP implementation, or to trigger an on-demand refresh of plan data in RI outside the normal batch cycle. If you run these on the same day as a normal batch run, or you run them multiple times, you must run the cleanup process shown in the dependencies prior to each run.

Key Tables Affected

Table	Usage
MFP_PLAN1_EXP	Input
W_RTL_PLAN1_PROD1_LC1_T1_FS	Staging
W_RTL_PLAN1_PROD1_LC1_T1_F	Output
MFP_PLAN2_EXP	Input
W_RTL_PLAN2_PROD2_LC2_T2_FS	Staging
W_RTL_PLAN2_PROD2_LC2_T2_F	Output
MFP_PLAN3_EXP	Input
W_RTL_PLAN3_PROD3_LC3_T3_FS	Staging
W_RTL_PLAN3_PROD3_LC3_T3_F	Output
MFP_PLAN4_EXP	Input
W_RTL_PLAN4_PROD4_LC4_T4_FS	Staging
W_RTL_PLAN4_PROD4_LC4_T4_F	Output
AP_PLAN1_EXP	Input
W_RTL_PLAN5_PROD5_LC5_T5_FS	Staging
W_RTL_PLAN5_PROD5_LC5_T5_F	Output

Planning Dimension Export

Module Name	LOAD_PDS_DIMENSION_PROCESS_ADHOC
Description	Exports all supported dimensions from the data warehouse to the data exchange schema for Planning.
Dependencies	RI_DIM_INITIAL_ADHOC
Business Activity	RI Integrations

Design Overview

This process runs all the planning data schema dimension exports from the data warehouse to the data exchange layer, where PDS batch processes can pick up and load the data the rest of the way. Each time the exports run, the data is truncated and inserted as full snapshots. Planning exports do not support incremental or delta extracts for dimensions. The programs apply various filters and criteria to the export data to align with Planning Data Schema requirements for dimensions, as described in the *RAP Implementation Guide*. The programs only export specific columns from each dimension, based on the downstream application needs. Review the PDS integration tables in detail to understand which data will be exported.

Key Tables Affected

Input Table	Output Table
W_PRODUCT_D	W_PDS_PRODUCT_D
W_PRODUCT_D_TL	W_PDS_PRODUCT_D
W_PROD_CAT_DH	W_PDS_PRODUCT_D
W_PRODUCT_ATTR_D	W_PDS_PRODUCT_D
W_DOMAIN_MEMBER_LKP_TL	W_PDS_PRODUCT_D
W_INT_ORG_D	W_PDS_ORGANIZATION_D
W_INT_ORG_D_TL	W_PDS_ORGANIZATION_D
W_INT_ORG_DH	W_PDS_ORGANIZATION_D
W_DOMAIN_MEMBER_LKP_TL	W_PDS_ORGANIZATION_D
W_INT_ORG_ATTR_D	W_PDS_ORGANIZATION_D
W_MCAL_DAY_D	W_PDS_CALENDAR_D
W_EXCH_RATE_G	W_PDS_EXCH_RATE_G
W_RTL_ITEM_GRP1_D	W_PDS_PRODUCT_ATTR_D
W_DOMAIN_MEMBER_LKP_TL	W_PDS_PRODUCT_ATTR_D
W_RTL_PRODUCT_ATTR_D	W_PDS_UDA_D
W_DOMAIN_MEMBER_LKP_TL	W_PDS_UDA_D
W_RTL_PRODUCT_ATTR_D	W_PDS_DIFF_D
W_RTL_PRODUCT_ATTR_D_TL	W_PDS_DIFF_D
W_RTL_ITEM_GRP2_D	W_PDS_PRODUCT_PACK_D
W_RTL_CUSTSEG_D	W_PDS_CUSTSEG_D
W_INVENTORY_PRODUCT_ATTR_D	W_PDS_REPL_ATTR_IT_LC_D

Input Table	Output Table
W_INT_ORG_D_CFA	W_PDS_ORG_ATTR_STR_D
W_INT_ORG_ATTR_D	W_PDS_ORG_ATTR_STR_D
W_ORGANIZATION_FLEX_D	W_PDS_ORG_ATTR_STR_D
W_INT_ORG_D_CFA	W_PDS_ORG_ATTR_NBR_D
W_INT_ORG_D_CFA	W_PDS_ORG_ATTR_DT_D
W_PRODUCT_D_CFA	W_PDS_PRODUCT_ATTR_STR_D
W_PRODUCT_D_CFA	W_PDS_PRODUCT_ATTR_NBR_D
W_PRODUCT_D_CFA	W_PDS_PRODUCT_ATTR_DT_D
W_RTL_IT_LC_D_CFA	W_PDS_PROD_ORG_ATTR_STR_D
W_RTL_IT_LC_D_CFA	W_PDS_PROD_ORG_ATTR_NBR_D
W_RTL_IT_LC_D_CFA	W_PDS_PROD_ORG_ATTR_DT_D

Planning Fact Export

Module Name	LOAD_PDS_FACT_PROCESS_ADHOC
Description	Exports all supported facts from the data warehouse to the data exchange schema for Planning.
Dependencies	HIST_SALES_LOAD_ADHOC HIST_INV_LOAD_ADHOC HIST_CSV_ADJUSTMENTS_LOAD_ADHOC HIST_CSV_INVRECEIPTS_LOAD_ADHOC HIST_CSV_MARKDOWN_LOAD_ADHOC HIST_CSV_INVRTV_LOAD_ADHOC HIST_CSV_TRANSFER_LOAD_ADHOC HIST_CSV_DEAL_INCOME_LOAD_ADHOC HIST_CSV_ICMARGIN_LOAD_ADHOC HIST_CSV_INVRECLASS_LOAD_ADHOC
Business Activity	RI Integrations

Design Overview

This process runs all the planning data schema fact exports from the data warehouse to the data exchange layer, where PDS batch processes pick up and load the data the rest of the way. Each run of these jobs inserts to the target tables with a new `RUN_ID`. Old runs are preserved for a configurable period of time (such as 7 days) to ensure PDS has adequate time to retrieve the data before it is erased. All fact exports are incremental and send only the current week’s data based on when it was posted into the data warehouse. This means the exports include all back-posted transaction data regardless of the transaction date, as long as it was posted to RI in the current fiscal week.

The range of dates exported by this process is tracked and configured from the table `C_SOURCE_CDC`. This table can be edited from the Control & Tactical Center to alter the range of dates exported in one batch execution, such as when you are sending historical data to MFP, or when you need to refresh the PDS data for more than a week. The table is automatically updated after every run to reflect the most recent export dates. The next export begins from the last date/time used. The date values are based on the `W_UPDATE_DT` column in the source tables, which tracks the last time a record was modified, regardless of the business transaction date on the record.

Key Tables Affected

Input Table	Output Table
W_RTL_SLS_IT_LC_WK_A	W_PDS_SLS_IT_LC_WK_A
W_RTL_SLS_IT_LC_WK_A	W_PDS_GRS_SLS_IT_LC_WK_A
W_RTL_SLSWF_IT_LC_WK_A	W_PDS_SLSWF_IT_LC_WK_A
W_RTL_INV_IT_LC_WK_A	W_PDS_INV_IT_LC_WK_A
W_RTL_INVU_IT_LC_WK_A	W_PDS_INVU_IT_LC_WK_A
W_RTL_PO_ONORD_IT_LC_DY_F	W_PDS_PO_ONORD_IT_LC_WK_A
W_RTL_PO_ONORD_IT_LC_DY_F	W_PDS_PO_ONORD_IT_LC_DY_F
W_RTL_MKDN_IT_LC_WK_A	W_PDS_MKDN_IT_LC_WK_A
W_RTL_INVADJ_IT_LC_WK_A	W_PDS_INVADJ_IT_LC_WK_A
W_RTL_INVRC_IT_LC_WK_A	W_PDS_INVRC_IT_LC_WK_A
W_RTL_INVTSF_IT_LC_WK_A	W_PDS_INVTSF_IT_LC_WK_A
W_RTL_INVRTV_IT_LC_WK_A	W_PDS_INVRTV_IT_LC_WK_A
W_RTL_INVRECLASS_IT_LC_WK_A	W_PDS_INVRECLASS_IT_LC_WK_A
W_RTL_DEALINC_IT_LC_WK_A	W_PDS_DEALINC_IT_LC_WK_A
W_RTL_ICM_IT_LC_WK_A	W_PDS_ICM_IT_LC_WK_A
W_RTL_TSF_IT_LC_DY_F	W_PDS_TSF_IT_LC_DY_F
W_RTL_ALC_IT_LC_DY_F	W_PDS_ALC_IT_LC_DY_F

Planning Initial Inventory Export

Module Name	LOAD_PDS_FACT_INITIAL_PROCESS_ADHOC
Description	Exports a full snapshot of historical inventory to the data exchange schema for Planning.
Dependencies	HIST_INV_LOAD_ADHOC
Business Activity	RI Integrations

Design Overview

This process exports inventory history from the data warehouse to Planning. The base inventory extract for PDS only sends the current week's inventory, as the data is positional in the data warehouse and the current week reflects all current values on the fact. This process can send a range of weeks at one time by configuring the start date and end date in `C_SOURCE_CDC` for this interface (where `TABLE_NAME = W_RTL_INV_IT_LC_WK_A`). The start and end dates must be week-ending dates and must match a range of weeks loaded as inventory history into RAP. If the job is run without configuring `C_SOURCE_CDC` with valid dates, it will fail with the error "no data found". All weeks of data are written for a single Run ID in the output table. Running the PDS import process consumes the entire range of data into their inventory facts.

Key Tables Affected

Table	Usage
C_SOURCE_CDC	Configuration
W_RTL_INV_IT_LC_WK_A	Input
W_PDS_INV_IT_LC_WK_A	Output

Planning Load Cleanup

Module Name	CLEANUP_C_LOAD_DATES_PLANNING_ADHOC
Description	Erases the execution status of planning batch programs. This is required to run a program multiple times for the same business date.
Dependencies	None
Business Activity	RI Integrations

Design Overview

This process erases records from the `C_LOAD_DATES` database table. Any time a job runs as part of the nightly batch, or a job is included in both nightly and ad hoc processing, a status record is inserted into `C_LOAD_DATES`. The job is then blocked from executing again while this record exists, as a safety measure when restarting batch processes that failed midway through execution. During initial planning integration loads, you may need to execute the same jobs multiple times to work through file or data issues. In that case, you may execute this process before each run to clear the status of prior runs from the database.

Note:

This process should only run during history and initial data loads, or at the guidance of Oracle Support. It should not run during regular nightly batch processing. Clearing `C_LOAD_DATES` when the batch is running normally could cause data corruption, as it would allow the same jobs to run multiple times for the same business date.

Key Tables Affected

Table	Usage
C_LOAD_DATES	Delete

POS Sales Integration

Module Name	LOAD_POSLOG_DATA_ADHOC
Description	Integrates data from Xstore, received through the POSLOG broadcaster services, into the RI data model.
Dependencies	None

Business Activity	RI Integrations

Design Overview

Retail Insights supports loading intraday sales transactions from Xstore's string-based XML receiver API. The data loaded by this method is specifically for reporting today's sales before the end-of-day batch processes the full snapshot of audited sales transactions. The sales data from Xstore is not used as a primary source of sales history in Retail Insights, as the system was designed around the concept of a Sales Audit system being used prior to data coming into the data warehouse.

The data first comes to the Retail AI Foundation Cloud Services from Xstore's web service API. The API is configured as part of the AI Foundation Cloud Services, but is used by Retail Insights to get the raw XML POSLOGs into the database for transformation to the RI data model. This process can then move the data from AI Foundation to RI staging tables, and from there to RI's internal data model for BI reports. Refer to the *RI Implementation Guide* for additional details.

Key Tables Affected

Table	Usage
W_RTL_POSLOG_XML_G	Input
W_RTL_SLS_POS_IT_LC_DY_FS	Staging
W_RTL_SLS_POS_IT_LC_DY_F	Output

Price History Load

Module Name	HIST_CSV_PRICE_LOAD_ADHOC
Description	Loads the `PRICE.csv` file into the data warehouse and populates key data tables used to integrate with other systems for history data.
Dependencies	HIST_ZIP_FILE_LOAD_ADHOC
Business Activity	Historical Data Load

Design Overview

The price history load process supports loading of price information over a long period of time to populate the data warehouse with historical data. This process both stages the `PRICE.csv` file into the database and processes it into RI. Multiple weeks of pricing data can be provided in a single file, though it is recommended not to load more than one month at a time, unless the volumes are low. Pricing data must start with a full snapshot of all item/locations on the earliest day in history that you will be loading. This can be loaded by itself to validate the file is formatted and the data is correct. From then on, you can provide only the price change events on the dates that they occur (such as regular and markdown price changes). The price history load will iterate through the provided files one day at a time and load the available price change events for each date in order.

The `C_HIST_LOAD_STATUS` configuration table determines the actions taken by the process. Before running the process for the first time, you must set up this table for the history load date to be the very latest date you expect to load history for (this can be changed later if needed to load more weeks). Once that setup is complete, you can begin processing files from the earliest date you plan to load. You must start from the beginning of the history and load it sequentially. You cannot load data out of order, and you cannot load the same date multiple times without first erasing the data from the database. After a date is loaded successfully, the `C_HIST_LOAD_STATUS` records are updated with the most recent load status and date.

Key Tables Affected

Table	Usage
W_RTL_PRICE_IT_LC_DY_FTS	File Input
W_RTL_PRICE_IT_LC_DY_FS	Staging
W_RTL_PRICE_IT_LC_DY_F	Output
W_RTL_PRICE_IT_LC_G	Output

Price History Load (Legacy)

Module Name	HIST_PRICE_LOAD_ADHOC
Description	Stages and loads the `W_RTL_PRICE_IT_LC_DY_FS.dat` file for pricing history.
Dependencies	HIST_ZIP_FILE_LOAD_ADHOC
Business Activity	Historical Data Load

Design Overview

The price history load process supports loading of price information over a long period of time to populate the data warehouse with historical data. This process stages the `W_RTL_PRICE_IT_LC_DY_FS.dat` file into the database and processes it into RI. Multiple weeks of pricing data can be provided in a single file, though it is recommended not to load more than one month at a time unless the volumes are low. Pricing data must start with a full snapshot of all item/locations on the earliest day in the history that you are loading. This can be loaded by itself to validate the file is formatted and the data is correct. From then on, you can provide only the price change events on the dates that they occur (such as regular and markdown price changes). The price history load iterates through the provided files one day at a time and loads the available price change events for each date in order.

The actions taken by the process are guided by the configuration table `C_HIST_LOAD_STATUS`. Before running the process for the first time, you must set up this table for the history load date to be the very latest date you expect to load history for (this can be changed later if needed to load more weeks). Once that setup is complete, you can begin processing files from the earliest date you plan to load. You must start from the beginning of the history and load it sequentially. You cannot load data out of order and you cannot load the same date multiple times without first erasing the data from your database. After a date is loaded successfully, the `C_HIST_LOAD_STATUS` records are updated with the most recent load status and date.

Key Tables Affected

Table	Usage
W_RTL_PRICE_IT_LC_DY_FS	Staging

Table	Usage
W_RTL_PRICE_IT_LC_DY_F	Output
W_RTL_PRICE_IT_LC_G	Output

Promotion Budget Load

Module Name	HIST_PROMO_FACT_LOAD_ADHOC
Description	Loads promotion budget and actuals for use in Retail Insights reporting.
Dependencies	HIST_ZIP_FILE_LOAD_ADHOC
Business Activity	Historical Data Load

Design Overview

This process loads the promotion budget and promotion actuals fact tables. These tables are only used in Retail Insights reporting and require that the promotion dimension `W_RTL_PROMO_D` is already loaded using the initial dimension load process or nightly batch jobs.

Key Tables Affected

Table	Usage
W_RTL_PRACT_IT_LC_DY_FS	Staging
W_RTL_PRACT_IT_LC_DY_F	Output
W_RTL_PRBDGT_IT_LC_FS	Staging
W_RTL_PRBDGT_IT_LC_F	Output

RDE Grants to APEX

Module Name	RDE_GRANT_MFCS_TO_APEX_ADHOC
Description	Refreshes the grants and synonyms for Merchandising replicated objects that should be exposed to Innovation Workbench.
Dependencies	None
Business Activity	Historical Data Load

Design Overview

This process runs the job `RDE_GRANT_MFCS_TO_APEX_JOB`, which re-applies the necessary grants and objects to allow a user to query Merchandising data from Innovation Workbench. This process assumes that the environment is one in which Merchandising is version 22 or later and the data is being actively replicated using Golden Gate to RAP.

The synonyms are present in the `RABE01USER` user in the database; so, when querying Merchandising objects, you may query a table like this:

```
select * from RABE01USER.ITEM_MASTER
```

Receipts History Load

Module Name	HIST_CSV_INVRECEIPTS_LOAD_ADHOC
Description	Loads the `RECEIPT.csv` file into the data warehouse and populates key data tables used to integrate with other systems for history data.
Dependencies	HIST_ZIP_FILE_LOAD_ADHOC
Business Activity	Historical Data Load

Design Overview

The history load process for Inventory Receipt transactions accepts an input file at the item/location/day level using the file specification for `RECEIPT.csv`. It assumes the file has already been moved into place by the `HIST_ZIP_FILE_LOAD_ADHOC` process. This process imports the file into a preprocessing table in the database, transforms it to internal staging tables, then loads it into the base fact (item/location/day) as well as the week aggregate used for integrations (item/location/week).

Key Tables Affected

Table	Usage
W_RECEIPT_FTS	File Input
W_RTL_INVRC_IT_LC_DY_FS	Staging
W_RTL_INVRC_IT_LC_DY_F	Output (Base Fact)
W_RTL_INVRC_IT_LC_WK_A	Output (Aggregate)

Receipt Expenses History Load

Module Name	HIST_INVRC_EXP_LOAD_ADHOC
Description	Loads the `W_RTL_INVRC_EXP_IT_LC_DY_FS.dat` file into the data warehouse and populates reporting aggregates.
Dependencies	HIST_ZIP_FILE_LOAD_ADHOC
Business Activity	Historical Data Load

Design Overview

The history load process for Inventory Receipt Expenses and Allowances transactions accepts an input file at the item/location/day level using the file specification for `W_RTL_INVRC_EXP_IT_LC_DY_FS.dat`. It assumes the file has already been moved into place by the `HIST_ZIP_FILE_LOAD_ADHOC` process. This process imports the file into the internal staging table, then loads it into the base fact (item/location/day) as well as the week aggregate used for reporting (item/location/week).

Key Tables Affected

Table	Usage
W_RTL_INVRC_EXP_IT_LC_DY_FS	Staging
W_RTL_INVRC_EXP_IT_LC_DY_F	Output (Base Fact)
W_RTL_INVRC_EXP_IT_LC_WK_A	Output (Aggregate)

Rejected Record Analysis

Module Name	W_RTL_REJECT_DIMENSION_TMP_ADHOC
Description	Analyses rejected records in the pricing and inventory position facts for any known causes of rejection, such as missing dimension keys for the records, and outputs a summary for review.
Dependencies	None
Business Activity	Historical Data Load

Design Overview

The rejected record analysis ad hoc process provides a set of queries comparing rejected data to all related dimensional tables. If any dimension keys are found on the rejected data but not in the related tables, a summary of the comparison is output to a database table for review. This tool can help debug invalid input data so it can be corrected and reprocessed. The ad hoc job currently runs for the Sales Transaction, Inventory Position, and Pricing facts, which are the most common history loads performed. The job is run automatically for Inventory and Price loads because they will fail if any records are rejected, but it requires manual setup and execution for Sales Transaction analysis. The modules enabled for the job are listed in the configuration table `W_RTL_REJECT_DIMLKUP_TMP`. The rejected dimension keys are output to `W_RTL_REJECT_DIMENSION_TMP`.

To add the sales module before running the job for transaction loads, follow these steps:

1. Navigate to the Control & Tactical Center's Manage System Configurations screen.
2. Locate the table `C_MODULE_REJECT_TABLE` and check whether there is already a row for `MODULE_CODE=SLS`
3. If there is not a row for SLS, then add a new row with these values for the first 3 column: `SLS`, `E$_W_RTL_SLS_TRX_IT_LC_DY_TMP`, `W_RTL_SLS_TRX_IT_LC_DY_FS`

To run the job from Postman, use the message body like below:

```
{
  "cycleName": "Adhoc",
  "flowName":"Adhoc",
  "processName":" W_RTL_REJECT_DIMENSION_TMP_ADHOC",
  "requestParameters":"jobParams. W_RTL_REJECT_DIMENSION_TMP_JOB=SLS 20230102
20230109"
}
```

The parameters for the job are the `MODULE_CODE` value from the configuration table followed by start and end dates in `YYYYMMDD` format. The dates correspond to the load date that resulted in rejected records on W_ETL_REJECTED_RECORDS. After you run the job, query W_RTL_REJECT_DIMENSION_TMP to see the results.

Key Tables Affected

Table	Usage
W_RTL_REJECT_DIMLKUP_TMP	Configuration
W_RTL_REJECT_DIMENSION_TMP	Output

Rejected Record Cleanup

Module Name	REJECT_DATA_CLEANUP_ADHOC
Description	Purges rejected records from certain E$ tables and populates a list of invalid dimension keys present on the purged data. The invalid keys will be ignored if a related fact history load process is re-run after failing due to these rejections.
Dependencies	None
Business Activity	Historical Data Load

Design Overview

The rejected record cleanup ad hoc process provides a way to clear out rejected data for positional fact history loads (currently inventory and price) that are blocked by having any rejections. The data is erased from the E$ tables and any invalid keys that do not have matching dimensions are written to the C_DISCARD_DIMM output table. If you then re-run the failed history job from POM, the job will ignore all of the discarded dimension keys and proceed to load the rest of the data file for the current day/week of processing. It is important to note that once you discard positional data in this manner, you cannot reload it later: you are declaring the data as unwanted/unusable. If you instead want to reload your data file with corrected records, you would not re-run your current history load job. You would go back and reload dimension and fact files as needed and start a fresh job run.

This job requires an input parameter of INV or PRICE, which tells the job which fact to clean up. The Postman body message format is below.

```
{
  "cycleName": "Adhoc",
  "flowName":"Adhoc",
  "processName":"REJECT_DATA_CLEANUP_ADHOC",
  "requestParameters":"jobParams.REJECT_DATA_CLEANUP_JOB=INV"
}
```

After doing the cleanup, check the C_HIST_LOAD_STATUS table to see where the history job stopped processing. If all steps are marked COMPLETE and the TMP table has a later value for the MAX_COMPLETED_DATE (for example, the TMP table has a date of 04/18/2021 and the other tables show 04/11/2021) then you may simply rerun the POM job to resume the dataload. In this scenario it will use the existing data in the HIST table for week of 04/18/2021 and continue to load those records in the F/A tables (ignoring the dimensions which are discarded).

Key Tables Affected

Table	Usage
E$_W_RTL_INV_IT_LC_DY_TMP1	Input
E$_W_RTL_PRICE_IT_LC_DP_TMP	Input
C_DISCARD_DIMM	Output

Reprocess CSV Files

Module Name	CSV_REPROCESS_ADHOC
Description	Reprocesses customer data files (with `.csv` file extensions) held in the `RI_REPROCESS_DATA.zip` archive.
Dependencies	REPROCESS_ZIP_FILE_PROCESS_ADHOC
Business Activity	Batch Administration

Design Overview

This process provides an on-demand way to load data files that caused failures in the AIF DATA nightly batch cycle and have a file extension of `.csv`; for example, you attempted to load the file `PRODUCT.csv`, but the nightly batch failed on the job `STG_SI_PRODUCT_JOB`. The steps to correct this issue and resume the batch are as follows:

1. Review and correct the file on your local server and generate a new `PRODUCT.csv` file, then add it to `RI_REPROCESS_DATA.zip`.
2. Upload the ZIP file using FTS to the `ris/incoming` prefix, then run the AIF DATA process `REPROCESS_ZIP_FILE_PROCESS_ADHOC` to load it. Verify that all jobs in the process complete successfully before continuing.
3. Select the AIF DATA process `CSV_REPROCESS_ADHOC`, then locate the job named `COPY_SI_PRODUCT_JOB` and run it.
4. If you want to verify the new file is correct, you may also run `STG_SI_PRODUCT_JOB` from within the same process. You can also go directly back to the nightly batch cycle and re-run the failed `STG_SI_PRODUCT_JOB` from there. Re-running the nightly batch job will attempt to stage the file into the database and resume the batch.

Your batch may also fail on the job `DIM_PROD_VALIDATOR_JOB`, which means the `PRODUCT.csv` file loaded successfully but there are other issues with the contents that cannot pass into the data warehouse. When the batch fails here, run all of the steps above (`COPY` and `STG` jobs for the file) plus all of the associated “simplified interface” or “SI” jobs that transform your CSV file data into the internal database structures. In the case of `PRODUCT.csv`, these additional jobs are one or more of the following, depending on what is enabled in your nightly batch:

- SI_W_PROD_CAT_DHS_JOB
- SI_W_PRODUCT_ATTR_DS_JOB
- SI_W_PRODUCT_DS_JOB
- SI_W_PRODUCT_DS_TL_JOB
- SI_W_RTL_IT_SUPPLIER_DS_JOB

- SI_W_RTL_PRODUCT_ATTR_IMG_DS_JOB
- SI_W_RTL_PRODUCT_BRAND_DS_JOB
- SI_W_RTL_PRODUCT_BRAND_DS_TL_JOB

After all the ad hoc jobs complete, return to the nightly batch cycle and re-run the failed `DIM_PROD_VALIDATOR_JOB`. If the data issues are corrected, then the job will complete successfully.

An alternative way to make data corrections specifically when a `VALIDATOR` job fails is to edit the data directly from Innovation Workbench. For example, when the `DIM_PROD_VALIDATOR_JOB` fails, instead of loading a new `PRODUCT.csv` file and re-running all of these jobs, you may instead log into Innovation Workbench and execute SQLs against the records having an issue in the staging tables, like `W_PRODUCT_DS`. This should only be done if you are familiar with the data model and feel comfortable editing the data directly. This may allow you to complete the current batch cycle faster, but you will still need to go back to your source system and correct the file for tomorrow's batch onwards.

Reprocess DAT Files

Module Name	DAT_REPROCESS_ADHOC
Description	Reprocesses customer data files (with `.dat` file extensions) in the `RI_REPROCESS_DATA.zip` file. Also used to import `.dat` files during initial history loads.
Dependencies	REPROCESS_ZIP_FILE_PROCESS_ADHOC, HIST_ZIP_FILE_LOAD_ADHOC
Business Activity	Batch Administration

Design Overview

This process provides an on-demand way to load data files that have a `.dat` file extension. There are two times when you would need to use this process:

- A data file caused failures in the AIF DATA nightly batch cycle and you need to provide a replacement for it.
- You are loading historical data with a `.dat` file extension, typically generated using an older version of Retail Data Extractor (RDE).

For example, you attempt to load the file `W_RTL_REPL_DAY_DS.dat` but the nightly batch failed on the job `W_RTL_REPL_DAY_DS_STG_JOB`. The steps to correct this issue and resume the batch process are as follows:

1. Review and correct the file on your local server and generate a new `W_RTL_REPL_DAY_DS.dat` file, then add it to `RI_REPROCESS_DATA.zip`
2. Upload the ZIP file using FTS to the `ris/incoming` prefix, then run the AIF DATA process `REPROCESS_ZIP_FILE_PROCESS_ADHOC` to load it. Verify that all jobs in the process complete successfully before continuing.
3. Select the AIF DATA process `DAT_REPROCESS_ADHOC`, then locate the `W_RTL_REPL_DAY_DS_COPY_JOB` job and run it.
4. If you want to verify the new file is correct, run `W_RTL_REPL_DAY_DS_STG_JOB` from within the same process. You can go directly back to the nightly batch cycle and re-run the failed `W_RTL_REPL_DAY_DS_STG_JOB` from there. Re-running the nightly batch job will attempt to stage the file into the database and resume the batch.

If you are using the process to load historical data files, then you may just enable and run all jobs in the process after unpacking your ZIP file; this will import the `.dat` files into the database staging layer.

RTV History Load

Module Name	HIST_CSV_INVRTV_LOAD_ADHOC
Description	Loads the `RTV.csv` file into the data warehouse and populates key data tables used to integrate with other systems for history data.
Dependencies	HIST_ZIP_FILE_LOAD_ADHOC
Business Activity	Historical Data Load

Design Overview

The history load process for Inventory Returns to Vendor (RTV) transactions accepts an input file at the item/location/day level using the file specification for `RTV.csv`. It assumes the file has already been moved into place by the `HIST_ZIP_FILE_LOAD_ADHOC` process. This process imports the file into a preprocessing table in the database, transforms it to RI's internal staging tables, then loads it into the base fact (item/location/day) as well as the week aggregate used for integrations (item/location/week).

Key Tables Affected

Table	Usage
W_RTL_INVRTV_IT_LC_DY_FTS	File Input
W_RTL_INVRTV_IT_LC_DY_FS	Staging
W_RTL_INVRTV_IT_LC_DY_F	Output (Base Fact)
W_RTL_INVRTV_IT_LC_WK_A	Output (Aggregate)

RTV History Load (Legacy)

Module Name	HIST_INVRTV_LOAD_ADHOC
Description	Stages and loads the `W_RTL_INVRTV_IT_LC_DY_FS.dat` file for return-to-vendor history.
Dependencies	HIST_ZIP_FILE_LOAD_ADHOC
Business Activity	Historical Data Load

Design Overview

The history load process for Inventory Returns to Vendor (RTV) transactions accepts an input file at the item/location/day level using the file specification for `W_RTL_INVRTV_IT_LC_DY_FS.dat`. It assumes the file has already been moved into place by the `HIST_ZIP_FILE_LOAD_ADHOC` process. This process will import the file into RI's internal staging tables and then load it into the base fact (item/location/day) as well as the week aggregate used for integrations (item/location/week).

Key Tables Affected

Table	Usage
W_RTL_INVRTV_IT_LC_DY_FS	Staging
W_RTL_INVRTV_IT_LC_DY_F	Output (Base Fact)
W_RTL_INVRTV_IT_LC_WK_A	Output (Aggregate)

Sales History Load

Module Name	HIST_SALES_LOAD_ADHOC
Description	Processes any staged sales history data and runs all aggregation programs for a specified history range.
Dependencies	HIST_STG_CSV_SALES_LOAD_ADHOC
Business Activity	Historical Data Load

Design Overview

The sales history load process supports loading of sales transaction data over a long period of time to populate the data warehouse with historical data. It requires the sales data to already be staged into the database using one of the available staging processes. Multiple weeks of sales can be provided in a single file, though it is recommended to not load more than one month at a time unless the volumes are low. This process populates all sales tables in the data warehouse (but not AIF applications), both for integration and BI reporting purposes. If you are not using RI for reporting, disable the aggregation table programs in POM (except the `W_RTL_SLS_IT_LC_WK_A` aggregate) before running the process.

Key Tables Affected

Table	Usage
W_RTL_SLS_TRX_IT_LC_DY_FS	Input
W_RTL_SLSPK_IT_LC_DY_FS	Input
W_RTL_SLS_TRX_IT_LC_DY_F	Output (Base Fact)
W_RTL_SLSPK_IT_LC_DY_F	Output (Base Fact)
W_RTL_SLS_IT_LC_WK_A	Aggregate (for integrations)
W_RTL_SLS_IT_LC_DY_A	Aggregate (for BI reporting)
W_RTL_SLS_IT_LC_GMH_A	Aggregate (for BI reporting)
W_RTL_SLS_SC_LC_DY_A	Aggregate (for BI reporting)
W_RTL_SLS_SC_LC_WK_A	Aggregate (for BI reporting)
W_RTL_SLS_CL_LC_DY_A	Aggregate (for BI reporting)
W_RTL_SLS_CL_LC_WK_A	Aggregate (for BI reporting)
W_RTL_SLS_DP_LC_DY_A	Aggregate (for BI reporting)
W_RTL_SLS_DP_LC_WK_A	Aggregate (for BI reporting)
W_RTL_SLS_IT_DY_A	Aggregate (for BI reporting)

Table	Usage
W_RTL_SLS_IT_WK_A	Aggregate (for BI reporting)
W_RTL_SLS_SC_DY_A	Aggregate (for BI reporting)
W_RTL_SLS_SC_WK_A	Aggregate (for BI reporting)
W_RTL_SLS_LC_DY_A	Aggregate (for BI reporting)
W_RTL_SLS_LC_WK_A	Aggregate (for BI reporting)
W_RTL_SLS_IT_LC_DY_SN_A	Aggregate (for BI reporting)
W_RTL_SLS_IT_LC_WK_SN_A	Aggregate (for BI reporting)
W_RTL_SLS_IT_DY_SN_A	Aggregate (for BI reporting)
W_RTL_SLS_IT_WK_SN_A	Aggregate (for BI reporting)
W_RTL_SLS_SC_LC_DY_CUR_A	Aggregate (for BI reporting)
W_RTL_SLS_SC_LC_WK_CUR_A	Aggregate (for BI reporting)
W_RTL_SLS_CL_LC_DY_CUR_A	Aggregate (for BI reporting)
W_RTL_SLS_DP_LC_DY_CUR_A	Aggregate (for BI reporting)
W_RTL_SLS_CL_LC_WK_CUR_A	Aggregate (for BI reporting)
W_RTL_SLS_DP_LC_WK_CUR_A	Aggregate (for BI reporting)
W_RTL_SLS_SC_DY_CUR_A	Aggregate (for BI reporting)
W_RTL_SLS_CL_DY_CUR_A	Aggregate (for BI reporting)
W_RTL_SLS_DP_DY_CUR_A	Aggregate (for BI reporting)
W_RTL_SLS_SC_WK_CUR_A	Aggregate (for BI reporting)
W_RTL_SLS_CL_WK_CUR_A	Aggregate (for BI reporting)
W_RTL_SLS_DP_WK_CUR_A	Aggregate (for BI reporting)
W_RTL_SLSPK_IT_LC_WK_A	Aggregate (for BI reporting)
W_RTL_SLSPK_IT_DY_A	Aggregate (for BI reporting)
W_EMPLOYEE_D	Supporting Dimension (for BI reporting)
W_PARTY_PER_D	Supporting Dimension (for BI reporting)
W_RTL_CO_HEAD_D	Supporting Dimension (for BI reporting)
W_RTL_CO_LINE_D	Supporting Dimension (for BI reporting)

Sales History Staging

Module Name	HIST_STG_CSV_SALES_LOAD_ADHOC
Description	Stages the `SALES.csv` file for the ad hoc sales load programs.
Dependencies	HIST_ZIP_FILE_LOAD_ADHOC
Business Activity	Historical Data Load

Design Overview

This process looks for the `SALES.csv` and `SALES_PACK.csv` files placed on the server by a history ZIP file upload, moves them into a preprocessing table, and transforms the data for use by the `HIST_SALES_LOAD_ADHOC` process.

Key Tables Affected

Table	Usage
W_RTL_SLS_TRX_IT_LC_DY_FTS	File Input
W_RTL_SLSPK_IT_LC_DY_FTS	File Input
W_RTL_SLS_TRX_IT_LC_DY_FS	Output
W_RTL_SLSPK_IT_LC_DY_FS	Output

Sales History Staging (Legacy)

Module Name	HIST_STG_SALES_LOAD_ADHOC
Description	Stages the `W_RTL_SLS_TRX_IT_LC_DY_FS.dat` file for the ad hoc sales load programs.
Dependencies	HIST_ZIP_FILE_LOAD_ADHOC
Business Activity	Historical Data Load

Design Overview

This process looks for the `W_RTL_SLS_TRX_IT_LC_DY_FS.dat` file placed on the server by a history ZIP file upload and loads it for use by the `HIST_SALES_LOAD_ADHOC` process.

Key Tables Affected

Table	Usage
W_RTL_SLS_TRX_IT_LC_DY_FS	File Input

Sales Tender Load

Module Name	HIST_SALES_TEND_LOAD_ADHOC
Description	Processes any staged sales tender data into the sales tender fact table, including processing of related dimensions used by the fact.
Dependencies	HIST_STG_SALES_TEND_LOAD_ADHOC
Business Activity	Historical Data Load

Design Overview

The sales tender load process supports loading of sales transaction tender data over a long period of time to populate the data warehouse with historical data. It requires the sales tender data to already be staged in the database using the separate staging process. Multiple weeks of tender data can be provided in a single file, though it is recommended to not load more than one month at a time unless the volumes are low. This process populates all sales tender tables in the data warehouse, both for integration and BI reporting purposes.

Key Tables Affected

Table	Usage
W_RTL_TRX_TNDR_LC_DY_FS	Input
W_EMPLOYEE_D	Output (Dimension Seeding)
W_RTL_TRX_TNDR_LC_DY_F	Output (Base Fact)

Sales Tender Staging

Module Name	HIST_STG_SALES_TEND_LOAD_ADHOC
Description	Processes a sales tender data file into the sales tender fact staging table.
Dependencies	HIST_ZIP_FILE_LOAD_ADHOC
Business Activity	Historical Data Load

Design Overview

This process looks for the `W_RTL_TRX_TNDR_LC_DY_FS.dat` file placed on the server by a history ZIP file upload and loads it for use by the `HIST_SALES_TEND_LOAD_ADHOC` process.

Key Tables Affected

Table	Usage
W_RTL_TRX_TNDR_LC_DY_FS.dat	Input File
W_RTL_TRX_TNDR_LC_DY_FS	Output

Sales Wholesale/Franchise Staging

Module Name	HIST_STG_CSV_SALES_WF_LOAD_ADHOC
Description	Processes the sales for wholesale/franchise data file into the sales WF fact staging table.
Dependencies	HIST_ZIP_FILE_LOAD_ADHOC
Business Activity	Historical Data Load

Design Overview

This process looks for the `SALES_WF.csv` file placed on the server by a history ZIP file upload and loads it for use by the `HIST_SALES_WF_LOAD_ADHOC` process.

Key Tables Affected

Table	Usage
W_RTL_SLSWF_IT_LC_DY_FTS	File Input
W_RTL_SLSWF_IT_LC_DY_FS	Staging

Sales Wholesale/Franchise Load

Module Name	HIST_SALES_WF_LOAD_ADHOC
Description	Processes any staged sales wholesale/franchise data into the sales wholesale/ franchise fact table and aggregates.
Dependencies	HIST_STG_CSV_SALES_WF_LOAD_ADHOC
Business Activity	Historical Data Load

Design Overview

The sales wholesale/franchise load process supports loading of sales measures for wholesale/ franchise locations over a long period of time to populate the data warehouse with historical data. It requires the sales data to already be staged in the database using the separate staging process. Multiple weeks of sales data can be provided in a single file. This process populates all sales wholesale/franchise tables in the data warehouse, for both integration and BI reporting purposes.

Key Tables Affected

Table	Usage
W_RTL_SLSWF_IT_LC_DY_FS	Staging
W_RTL_SLSWF_IT_LC_DY_F	Base Fact
W_RTL_SLSWF_IT_LC_WK_A	Aggregate (PDS Integration)
W_RTL_SLSWF_IT_DY_A	Aggregate (BI Reporting)
W_RTL_SLSWF_IT_WK_A	Aggregate (BI Reporting)
W_RTL_SLSWF_SC_LC_DY_A	Aggregate (BI Reporting)
W_RTL_SLSWF_SC_LC_WK_A	Aggregate (BI Reporting)
W_RTL_SLSWF_SC_LC_DY_CUR_A	Aggregate (BI Reporting)
W_RTL_SLSWF_SC_LC_WK_CUR_A	Aggregate (BI Reporting)

Shipments History Load

Module Name	SEED_CSV_W_RTL_SHIP_IT_LC_DY_F_PROCESS_ADHOC
Description	Loads a full snapshot of shipment data from `SHIPMENT_HEAD.csv` and `SHIPMENT_DETAIL.csv` to initialize the dimension and fact data before the nightly batch is enabled.

Dependencies	HIST_ZIP_FILE_LOAD_ADHOC
Business Activity	Historical Data Load

Design Overview

Data regarding shipments of merchandise is split between two interfaces, the dimension file `SHIPMENT_HEAD.csv` and the fact file `SHIPMENT_DETAIL.csv`. This process can be used to load full snapshots of historical or currently active shipments to the data warehouse outside of the nightly batch cycle. The two files must be in sync, meaning that every shipment record on the detail file must have a record in the header file. The header file is always a full snapshot of all shipments that should appear as currently active in the data warehouse, meaning that if any shipment records are no longer sent on `SHIPMENT_HEAD.csv`, they will be marked as inactive/closed in the data warehouse table (`CURRENT_FLG = N`) and should no longer appear in the files.

Key Tables Affected

Table	Usage
W_RTL_SHIP_DETAILS_DTS	File Input (`SHIPMENT_HEAD.csv`)
W_RTL_SHIP_DETAILS_DS	Staging
W_RTL_SHIP_DETAILS_D	Output
W_RTL_SHIP_IT_LC_DY_FTS	File Input (`SHIPMENT_DETAIL.csv`)
W_RTL_SHIP_IT_LC_DY_FS	Staging
W_RTL_SHIP_IT_LC_DY_F	Output

Stock Count Load

Module Name	HIST_STOCK_COUNT_LOAD_ADHOC
Description	Loads the stock count dimension and facts for Retail Insights reporting.
Dependencies	HIST_ZIP_FILE_LOAD_ADHOC
Business Activity	Historical Data Load

Design Overview

This process loads the stock count dimension and fact tables for use in Retail Insights reporting. The fact tables are for counts coming from two different sources, systemic counts (INVSS) from a merchandising solution and perpetual counts (INVPS) from a perpetual/real-time inventory solution. The same dimension supports both facts, so the dimension file must be a combination of all stock count header records from any source.

Key Tables Affected

Table	Usage
W_RTL_LOC_STOCK_CNT_DS	Staging

Table	Usage
W_RTL_LOC_STOCK_CNT_D	Output
W_RTL_INVPS_CNT_IT_LC_DY_FS	Staging
W_RTL_INVPS_CNT_IT_LC_DY_F	Output
W_RTL_INVSS_CNT_IT_LC_DY_FS	Staging
W_RTL_INVSS_CNT_IT_LC_DY_F	Output

Stock Ledger Load

Module Name	HIST_STCKLDGR_LOAD_ADHOC
Description	Loads Stock Ledger fact data for use in Retail Insights reporting.
Dependencies	HIST_ZIP_FILE_LOAD_ADHOC
Business Activity	Historical Data Load

Design Overview

This process loads the stock ledger fact files for week- and month-level stock ledgers. These facts are used only in Retail Insights reporting. Before using this process, you must configure it to disable one of the two month-level jobs:

- W_RTL_STCKLDGR_SC_LC_MH_F_JOB
- W_RTL_STCKLDGR_SC_LC_MH_F_GREG_JOB

The first job is used for Fiscal Calendar only, while the second job is used for Gregorian calendar only. Attempting to run both jobs in the same environment will result in failures because you cannot have both calendar types in the table at the same time.

Key Tables Affected

Table	Usage
W_RTL_STCKLDGR_SC_LC_WK_FS	Staging
W_RTL_STCKLDGR_SC_LC_WK_F	Output
W_RTL_STCKLDGR_SC_LC_MH_FS	Staging
W_RTL_STCKLDGR_SC_LC_MH_FS	Output

Store Traffic Load

Module Name	HIST_STTRFC_LOAD_ADHOC
Description	Loads Store Traffic fact data for use in Retail Insights reporting.
Dependencies	HIST_ZIP_FILE_LOAD_ADHOC
Business Activity	Historical Data Load

Design Overview

This process loads the Store Traffic fact file. This fact data is used for Retail Insights reporting only. All jobs in the process should be enabled before running it.

Key Tables Affected

Table	Usage
W_RTL_STTRFC_LC_DY_MI_FS	Staging
W_RTL_STTRFC_LC_DY_MI_F	Output

Supplier Compliance Load

Module Name	HIST_SUPPCM_LOAD_ADHOC
Description	Loads Supplier Compliance fact data for use in Retail Insights reporting.
Dependencies	HIST_ZIP_FILE_LOAD_ADHOC
Business Activity	Historical Data Load

Design Overview

This process loads the Supplier Compliance fact files. This fact data is used for Retail Insights reporting only. All jobs in the process should be enabled before running it.

Key Tables Affected

Table	Usage
W_RTL_SUPPCM_IT_LC_DY_FS	Staging
W_RTL_SUPPCM_IT_LC_DY_F	Output
W_RTL_SUPPCM_IT_LC_WK_A	Aggregate
W_RTL_SUPPCMUF_LC_DY_FS	Staging
W_RTL_SUPPCMUF_LC_DY_F	Output
W_RTL_SUPPCMUF_LC_WK_A	Aggregate

Supplier Invoice Load

Module Name	HIST_SUPP_IVC_LOAD_ADHOC
Description	Loads Supplier Invoice fact data for use in Retail Insights reporting.
Dependencies	HIST_ZIP_FILE_LOAD_ADHOC
Business Activity	Historical Data Load

Design Overview

This process loads the Supplier Invoice fact file. This fact data is used for Retail Insights reporting only. All jobs in the process should be enabled before running it.

Key Tables Affected

Table	Usage
W_RTL_SUPP_IVC_PO_IT_FS	Staging
W_RTL_SUPP_IVC_PO_IT_F	Output

Table Partitioning

Module Name	CREATE_PARTITION_ADHOC
Description	Uses the provided range of dates and the loaded calendar information to generate table partitions across the data warehouse data model.
Dependencies	CALENDAR_LOAD_ADHOC
Business Activity	Initial System Setup

Design Overview

This process may be used after the Calendar load is complete to partition all of your database tables. Tables in Retail Insights are partitioned dynamically based on your fiscal calendar using the days and weeks defined in `W_MCAL_DAY_D` and `W_MCAL_WEEK_D`. This type of partitioning provides optimal performance in BI reporting, where the SQL queries can prune the selected partitions to only those that hold data for your time-based filters and attributes. Without this partitioning in place, batch programs will not insert data into the expected partitions, some programs could fail to load data at all, and BI reporting will have very poor performance.

This process can be run repeatedly to ensure all partitions are created. Each time it runs, it resumes from where it left off, if any partitions still need to be added to the data model. If you have run the process several times and it is now completing in under a minute, then it is no long recreating any new partitions. The functional areas being partitioned should be reviewed in the table `C_MODULE_ARTIFACT`. All tables should be enabled for partitioning, except for tables that have `PLAN` in their naming structure.

Key Tables Affected

Table	Usage
W_MCAL_DAY_D	Input
W_MCAL_WEEK_D	Input
C_ODI_PARAM	Input

Transfer Detail History Load

Module Name	HIST_TSFDETAIL_LOAD_ADHOC
Description	Loads a full snapshot of transfers data from `W_RTL_TSF_DETAILS_DS.dat` and `W_RTL_TSF_IT_LC_DY_FS.dat` to initialize the dimension and fact data before the nightly batch is enabled.
Dependencies	HIST_ZIP_FILE_LOAD_ADHOC
Business Activity	Historical Data Load

Design Overview

Data regarding transfers of merchandise is split between two interfaces, the dimension file `W_RTL_TSF_DETAILS_DS.dat` and the fact file `W_RTL_TSF_IT_LC_DY_FS.dat`. This process can be used to load full snapshots of historical or currently active transfers to the data warehouse outside of the nightly batch cycle. The two files must be in sync, meaning that every transfer record on the detail file must have a record in the header file. The header file is always a full snapshot of all transfers that should appear as currently active in the data warehouse, meaning that if any transfer records are no longer sent on `W_RTL_TSF_DETAILS_DS.dat`, they will be marked as inactive/closed in the data warehouse table (`CURRENT_FLG = N`) and should no longer appear in the files.

This data is not the same as the transfer transactions file (`TRANSFER.csv`). The transfer transactions are aggregated at the item/loc/day level of detail, while these two files are for the individual transfer activities at the lowest level of detail. The aggregated transfer transactions are mainly for Retail Insights and Merchandise Financial Planning, while these transfer detail files are for RI, AI Foundation, and Inventory Planning Optimization usage.

Key Tables Affected

Table	Usage
W_RTL_TSF_DETAILS_DS	Staging
W_RTL_TSF_DETAILS_D	Output
W_RTL_TSF_IT_LC_DY_FS	Staging
W_RTL_TSF_IT_LC_DY_F	Output

Transfer Transaction History Load

Module Name	HIST_CSV_TRANSFER_LOAD_ADHOC
Description	Loads the `TRANSFER.csv` file into the data warehouse and populates key data tables used to integrate with other systems for history data.
Dependencies	HIST_ZIP_FILE_LOAD_ADHOC
Business Activity	Historical Data Load

Design Overview

The history load process for Inventory Transfer transactions accepts an input file at the item/location/day level using the file specification for `TRANSFER.csv`. It assumes the file has already been moved into place by the `HIST_ZIP_FILE_LOAD_ADHOC` process. This process imports the file into a preprocessing table in the database, transforms it to RI's internal staging tables, then loads it into the base fact (item/location/day) as well as the week aggregate used for integrations (item/location/week).

This data is not the same as the transfer details files (`W_RTL_TSF_DETAILS_DS.dat` and `W_RTL_TSF_IT_LC_DY_FS.dat`). The transfer transactions on this file are aggregated at the item/loc/day level of detail, while the transfer detail files are for the individual transfer activities at the lowest level of detail. The aggregated transfer transactions on this file are for Retail Insights and Merchandise Financial Planning.

Key Tables Affected

Table	Usage
W_RTL_INVTSF_IT_LC_DY_FTS	File Input
W_RTL_INVTSF_IT_LC_DY_FS	Staging
W_RTL_INVTSF_IT_LC_DY_F	Output (Base Fact)
W_RTL_INVTSF_IT_LC_WK_A	Output (Aggregate)

Translation Lookup Load (Legacy)

Module Name	W_DOMAIN_MEMBER_LKP_TL_PROCESS_ADHOC
Description	Processes the translatable string data in the `W_DOMAIN_MEMBER_DS_TL.dat` file and loads it into the data warehouse.
Dependencies	HIST_ZIP_FILE_LOAD_ADHOC
Business Activity	Historical Data Load

Design Overview

This process looks for the `W_DOMAIN_MEMBER_DS_TL.dat` file placed on the server by a history ZIP file upload and loads it to the target table in the data warehouse for translatable strings. When using CSV file uploads, all the translatable strings from the CSV files are automatically inserted into this table and loaded in the data warehouse without a second file being provided. However, if you are using legacy files, or you need to update records in this table directly, you can use this process to manually load string lookup records.

Key Tables Affected

Table	Usage
W_DOMAIN_MEMBER_DS_TL	Staging
W_DOMAIN_MEMBER_LKP_TL	Output

3

AI Foundation Data Batch Architecture

The AIF DATA batch schedule is responsible for managing foundation data for the Retail Analytics and Planning solutions and storing it in a shared data warehouse architecture. Understanding the core features of the data warehouse can help you make better use of the jobs and processes available to you. When working with Oracle Support, you may also be asked to modify the behavior of some jobs to resolve issues, and this chapter provides additional information on the AIF DATA processes you will be expected to interact with.

This chapter will cover the following major functions of the AIF DATA batch schedule:

- **File ingestion** – How customer data files are received and processed
- **Data extract** – How data is extracted from a source system within the Oracle Cloud for use in Analytics and Planning applications
- **Data import** – How customer data files are pulled into the Oracle database
- **Data transform and load** – How customer data is transformed and loaded into the data warehouse
- **Data reclassification** – How customer data is reorganized when an item or location is reclassified to a new hierarchy node
- **Data aggregation** – How customer data is aggregated and summarized for use in BI reports
- **Data export** – How customer data is exported from the data warehouse for use in other RAP applications
- **Other supporting processes** – Other technical aspects of the batch processing that supports the overall flow of data throughout the system

File Ingestion

Data files are accepted from external sources using a private cloud instance of Oracle Object Storage (OS). This OS instance is not comparable to the Oracle Public Cloud version: it does not have a user interface and cannot be accessed except through specific programming APIs provided by Oracle (such as the File Transfer Services or FTS). It is used for short-term data storage and retrieval only as a means of transferring files into or out of the Oracle Retail cloud. If you are not providing any files and instead using direct integration from Merchandising Foundation Cloud Service (MFCS) then these programs will not be used.

AIF DATA programs expect customer files to come in the form of ZIP files. The ZIP file must contain only a set of files from the pre-determined list of interfaces that AIF DATA batch supports. It cannot contain a folder structure; all files must be directly packaged into the ZIP file itself. The general steps performed once a ZIP file is uploaded to Object Storage are:

1. The ZIP file is passed through a virus scanner and will not be consumed until the scan completes. A batch program (`ZIP_FILE_WAIT_JOB`) will wait for a maximum of 4 hours for the ZIP file to become available for import and then fail.
2. The ZIP file is unpacked by another batch program (`ZIP_FILE_UNLOAD_JOB`) into a temporary directory where the files will be cleansed using a variety of standard Unix functions including `dos2unix` and methods to apply UTF-8 encoding.

3. After file conversion is completed, all files used in prior batch runs are deleted and the new files are moved to the next stage of processing.
4. Files are compared to a list of required filenames as specified in Retail Home. If any required files are missing, the batch program (`DAT_FILE_VALIDATE_JOB`) will fail. If all required files are found, then the entire set of files are moved into the final server directory where they can be read by later jobs.

Files cannot be consumed by the rest of the batch unless `DAT_FILE_VALIDATE_JOB` is run successfully, so it is a critical program that cannot be skipped or disabled. Even if you have marked all files as optional in Retail Home by disabling the `CONTROLFILES` modules, you must still run `DAT_FILE_VALIDATE_JOB` because it will move the files to the required folder structure on the application server.

Data Extract

When you subscribe to other Oracle Cloud applications that support integration with Retail Analytics and Planning (RAP) applications, such as Merchandising Foundation Cloud Service, you have the option to leverage direct database-to-database extracts that run as part of the AIF DATA batch schedule. These extracts are collectively referred to as the Retail Data Extractor (RDE) component of RAP. They key points of the RDE architecture are:

- Oracle Golden Gate technology is used to take a near-real-time copy of the source system's data and place it on the local ADW instance where RAP is installed
- The local copy of the data is overlaid with database synonyms and permission grants so that AIF DATA batch programs can access it
- Extract programs in the AIF DATA schedule are usually prefixed with `RDE` to more easily identify them
- RDE programs read from the cloned tables, transform the data, and then insert the result to the staging tables in the data warehouse

For all other implementation details on using RDE programs, refer to the *Retail Analytics and Planning Implementation Guide* chapter on "Integration with Merchandising".

Data Import

Data files that have been successfully prepared through the file ingestion process will then go through a two-step process to import the contents into the database. The first step will copy a specific file from the application server staging directory to another location where the database can access it. The second step will perform an External Table Load procedure, which is a standard feature of Oracle Autonomous Data Warehouse (ADW). The file is mapped to a temporary external table name, then the rows of data are read from the file and inserted into a pre-defined target table in our data model.

The file copy step is performed by batch jobs having `COPY` in their names. For example, `W_MCAL_PERIOD_DS_COPY_JOB` will copy the `W_MCAL_PERIOD_DS.dat` file. These jobs have the following parameters in POM that determine the behavior of the process:

- `disJobType` – The internal code for the program type, which affects the API call used when the job is run
- `object` – The filename being written to the destination
- `objectFilepath` – The source path and filename for the copy action

If these `COPY` jobs are not run successfully, then there will not be any file available for the next step to consume, and later steps in the batch will fail as a result.

The external table load step is performed by batch jobs having `STG` in their names. For example, `W_MCAL_PERIOD_DS_STG_JOB` will read the file data from `W_MCAL_PERIOD_DS.dat` and write it to the database. These jobs have the following parameters in POM that determine the behavior of the process:

- `disJobType` – The internal code for the program type, which affects the API call used when the job is run
- `tableName` – The target table in the database where the file data will be written to when this job is executed
- `filename` – The source file that will be mapped to a temporary external table for processing, which must match an object in the `COPY` job parameters
- `connectionType` – The ADW connection tier used for this data load, from one of (low, medium, high)
- `truncateTable` – Instruct the load to truncate the target table before insert (true) or append to the table (false)

You may modify the parameters `connectionType` and `truncateTable` based on guidance from Oracle Support or as needed during an implementation. The connection type of “low” is optimal for low volumes of data, but if you find yourself pushing large amounts of data into a program that uses a low connection, you can change the parameter value to “medium” which can better handle high volumes. Do not make this change to connection type unless you are sure it is necessary and revert the setting to its original value after you are done loading data. You might also want to change `truncateTable` to “false” if you do not want to lose the data currently in a staging table when loading a new file to that table, such as when a customer provides their implementer with an incremental file that needs to be appended to data previously staged for loading. Like the connection type, do not change this setting unless required for a specific use case, and revert it to the default value (true) when finished with your loads.

The external table load (`STG`) programs also use the configuration options provided on context (`CTX`) files to further change the behavior of the data imports. Details on creating context files is provided in the *Retail Analytics and Planning Implementation Guide* chapter on “Data File Generation”. The context files are not directly used; they are pushed into ADW metadata tables using a dedicated POM job (`RI_UPDATE_TENANT_JOB`) and then the external table loads will read from that metadata to configure the imports. This metadata controls the following aspects of the file import: columns to read from the file, column delimiters, data formats, trimming of spaces, header row processing, rejected record tolerance, and any optional parameters supported by the `DBMS_CLOUD` package in ADW.

Transform and Load

Data that has been staged into the data warehouse (either by file imports or direct integrations) may go through additional transformations before being written to the final warehouse tables. When the data is directly loaded using RDE programs, all the transformation is already done and the next step would be to load it into the data warehouse. If the data came from a file with a CSV file extension, it needs additional processing before it can be used.

CSV file formats are provided to reduce the development overhead of creating interfaces for a set of related data warehouse tables. CSV files often combine two or more internal tables into one input file, so additional transformation is needed to split the data fields out after importing the file. The transformation of CSV file data is done using a combination of Oracle Data Integrator (ODI) programs and PL/SQL scripting, and the programs are collectively called

Simplified interface (SI) programs. The SI program's responsibility is to split out the relevant fields from the input to all the other staging tables which require something from that file. For example, the `PRODUCT.csv` file is imported to a table named `W_PRODUCT_DTS`. From here, the SI program `SI_W_PRODUCT_DS_JOB` takes a subset of this data and moves it to `W_PRODUCT_DS`. Another SI program `SI_W_PROD_CAT_DHS_JOB` reads from the same source table and writes to `W_PROD_CAT_DHS`. This continues until all the fields from `W_PRODUCT_DTS` have been parsed out into their designated areas by SI jobs.

The target tables of the SI programs are the same tables directly populated by the RDE extracts, meaning the two data flows are mutually exclusive. Either RDE jobs or file-based jobs can be used to populate data into the data warehouse for a particular interface. Attempting to run both for the same table will result in errors and data conflicts. For additional examples of these data flows, refer to the *Retail Analytics and Planning Implementation Guide* chapter on "Batch Orchestration", under the section "Managing Multiple Data Sources".

After any transformation steps are complete, then loading into the data warehouse target tables can begin. Load jobs take the data in staging tables like `W_PRODUCT_DS` and move them into internal tables like `W_PRODUCT_D`. The load programs are complex and perform many operations to get from the source to the target. The data warehouse applies primary and foreign key constraints, internal sequences for row tracking, audit fields such as insert/update timestamps, historical versions of modified records, data quality validations, and many other functions. The basic load jobs can be broadly split into dimension jobs (for tables ending with `_D`), fact jobs (for tables ending with `_F`), translation lookup jobs (for tables ending with `_TL),` and general load jobs (for tables ending with `_G`). For these programs, the name of the job in the AIF DATA batch schedule will correlate with the target table being loaded. For example, `W_PRODUCT_D_JOB` is responsible for loading the `W_PRODUCT_D` table. Understanding this naming scheme will allow you to connect the jobs being run in the batch schedule with the tables being populated in the data warehouse.

Data Reclassification

A core feature of the data warehouse is its ability to manage reclassifications, which is the term used to describe the reassignment of an item or location from one position in your hierarchies to another. The item and location dimensions (and their hierarchies) are the only ones to support reclassification management. All other dimensions are taken "as-is", meaning the data warehouse will not track major changes done to those records as a reclassification.

As it pertains to the AIF DATA batch schedule, the system applies a reclassification activity across two days, following this overall flow of events:

- Day 1 Batch
  - Incoming product and location data is compared to the data already in the data warehouse and changes to the hierarchy assignments are recorded in temporary tables. This comparison is done automatically and does not require you to tell the system about reclassified records separately.
  - For the initial batch when a reclassified item or location is sent in, the data is applied in the dimension tables like `W_PRODUCT_D` immediately, inserting a new record for the reclassified data and updating the existing record as inactive. Effective dates are assigned to both the old and new records to indicate the periods of time when that hierarchy classification was active. The `CURRENT_FLG` column indicates the latest version of the record.
  - If any fact records are provided for a reclassified item or location, that data is loaded relative to the new record's unique identifier, but existing data remains on the old

record's identifier. Joining the fact and dimension tables together is how you cross-reference the old and new identifiers with the same item or location.

  - A set of jobs prefixed with `FACTOPEN` and `FACTCLOSE` are responsible for maintaining positional facts like inventory and pricing, end-dating the old version of reclassed records and inserting new ones for the new dimension keys, even if the fact itself had no changes on that day. This is necessary to carry forward the positional balances on the new dimension keys, not the inactive ones.

- Day 2 Batch
  - At the very start of the next day's batch processing, a series of jobs having `RECLASS` in the name will take effect. These jobs are responsible for recalculating aggregation tables in the data warehouse to regenerate fact data based on the latest classifications. This is done on the second day's batch due to the amount of time it can take to run, so it's best to do it early in the morning before users enter the system instead of doing it at the end of Day 1's batch.

A side-effect of the split approach is that BI reporting will not reflect a reclassification fully until a day has elapsed and the aggregate tables are recalculated. A user may only notice this if they attempt to report on the current day's fact data at an aggregate level on the same day a reclassification occurs, which will be a small subset of reports (if any). It is still better to run the aggregation programs outside of normal business hours versus running them during the business day in the first batch, which will impact the user experience more than the temporary data inconsistency does. This approach does not impact the other Analytics and Planning applications because they receive their data directly from the item/location fact tables, not the BI aggregate tables.

Data Aggregation

The Analytics and Planning data warehouse supports the Retail Insights solution for all business intelligence and reporting needs. It does this using a comprehensive set of pre-aggregated data tables holding information at levels commonly used in reporting, such as department/week or subclass/store/week. The AIF DATA batch schedule is responsible for maintaining all of the aggregate tables. Each job in the batch schedule ending with `_A_JOB` populates an aggregate table in the data warehouse, with the job name directly corresponding to the table being populated (for example, the `W_RTL_SLS_SC_LC_DY_A_JOB` populates the `W_RTL_SLS_SC_LC_DY_A` table).

It is important to understand how the data warehouse performs aggregations as it can directly impact the way you provide your data to the system. Aggregations are performed using a different approach depending on the type of fact data:

- Transaction facts such as sales and receipts are aggregated incrementally and additively. The incoming data is summarized at each level of aggregation and added to the data already in the target table. This means that the incoming fact data must only contain the values that can be summed with existing records to reach the desired final numbers. Loading data in this manner has the best performance in batch processing as it requires the least number of changes to the aggregate tables. Data will be updated only if it is being altered by an incoming record.
- Positional facts such as inventory are aggregated as complete snapshots of the base fact data for daily aggregates and on the first day of a new week. The item/location records are rolled up to the desired level and then inserted to the aggregation table for the new daily or weekly positions. Weekly positional data is also updated incrementally during the week. The incoming records will update the current week's positions with the changed values (non-additively, replacing the existing data with the new data).

There are multiple distinct batch jobs involved in fact aggregation and all must be enabled in the nightly batch cycles before any BI aggregates can be populated. Using the sales transaction fact aggregate `W_RTL_SLS_IT_DY_A` as an example, the data is processed in the following steps:

- Incremental sales data for the current date will be staged into the `W_RTL_SLS_TRX_IT_LC_DY_FS` table using either RDE programs or a file-based load.
- `W_RTL_SLS_IT_DY_TMP_JOB` aggregates the incoming transaction records from the staging layer to a temporary table `W_RTL_SLS_IT_DY_TMP` at the summarized level of item/day. The internal base fact table `W_RTL_SLS_TRX_IT_LC_DY_F` is not used; aggregate table data always comes from the staging layer only.
- `W_RTL_SLS_IT_DY_A_JOB` will take the data from `W_RTL_SLS_IT_DY_TMP` and load it to `W_RTL_SLS_IT_DY_A` using an additive `MERGE` statement. Existing records will be merged by summing the same column in the source and target (for example, `t.SLS_QTY=s.SLS_QTY+t.SLS_QTY`) while unmatched records will be inserted with all the values from the temporary table. Positional facts such as inventory will use a non-additive merge, overwriting the target column from the source table.
- Other aggregate tables that need item level data such as `W_RTL_SLS_IT_WK_A` will also source from `W_RTL_SLS_IT_DY_TMP` to update their data using a similar merge statement. Week-level aggregates merge the daily data onto the week-level records using the system's fiscal calendar to map the days to weeks.

The most important behavior to understand is that the daily staged data is used to update aggregate tables. A common point of confusion about the aggregation process is the expectation that lower-level aggregate tables directly populate the higher-level aggregates. This leads to the incorrect assumption that all aggregate tables are regenerated from the base fact every day. An implementer might think that a manual data change on a base fact table like `W_RTL_SLS_TRX_IT_LC_DY_F` will automatically appear in the BI aggregates after batch, which is not the case. For aggregates to be updated, the data must be loaded through the staging tables and then processed through the aggregation programs.

Data Exports

The AIF data warehouse is used to export certain datasets to planning applications such as Merchandise Financial Planning (MFP). Separate AIF DATA nightly batch jobs are responsible for pulling data out of the data warehouse tables and exporting it to tables in the retail data exchange (RDX) schema. Only data exported to RDX is available for use in the planning applications, all other data warehouse tables are used only by Retail Insights or AI Foundation processes. This method of exporting data is specific to the planning applications, the way AIF applications get data is by pulling from the data warehouse using AIF APPS batch schedule programs.

Data export programs use a different naming scheme starting with `W_PDS`, such as `W_PDS_PRODUCT_D_JOB`. These jobs use PL/SQL to read from the data warehouse tables and write to RDX schema tables. Export programs operate on a run-based architecture where each execution of the job will create a new Run ID to track the execution and write the complete set of outputs. Runs are managed using tables `RAP_INTF_CFG` and `RAP_INTF_RUN_STATUS` that can be monitored from Innovation Workbench. The full mapping between the source tables in the data warehouse and the target tables in the RDX schema is available in the *Retail Analytics and Planning Implementation Guide* chapter on "Data Processing and Transformations".

Exports from the data warehouse to RDX tables share some common design patterns:

- Export programs, like all other AIF DATA jobs, leverage the `C_LOAD_DATES` table to track the job level run status, and any existing entry in `C_LOAD_DATES` will prevent running the export program again if it has already been run once.
- Exported facts will join on their associated export dimensions to ensure data consistency across tables, such as `W_PDS_SLS_IT_LC_WK_A_JOB` only exporting results for item/locations also present on the `W_PDS_PRODUCT_D` and `W_PDS_ORGANIZATION_D` dimension export tables.
- The source and target tables use the same intersections of data, as defined in the table names. For example, the job `W_PDS_SLS_IT_LC_WK_A_JOB` reads from the table `W_RTL_SLS_IT_LC_WK_A` and writes to `W_PDS_SLS_IT_LC_WK_A`, and both the source and target tables are aggregated at the item/location/week levels.
- Identifiers for the item, location, and date values on the PDS export tables will be converted to their external-facing values, meaning it is not possible to directly join the input and output tables without also using dimension tables to convert the identifiers (the data warehouse tables do not hold the external codes, instead they have foreign key values referring to their associated dimensions).
- Exports of dimension data will be a full snapshot of currently active source records. Exports of positional fact data will be full snapshots of the current positions only (no historical data). Exports of transaction fact data will be incremental based on the change-tracking maintained in the `W_UPDATE_DT` columns of the source tables. Incremental export dates are tracked on the `C_SOURCE_CDC` table.

Supporting Processes

The AIF DATA batch schedule has many programs that support the other major functions described in this chapter but whose purpose is not readily apparent from the job name. Review the table below for a list of such supporting processes and their primary functions within the data warehouse architecture.

Job Name	Primary Functions
*_MILE_MARKER_JOB	Any job having the term “mile marker” in the name is a simple placeholder in POM that allows for easy tracking of key points in the batch sequence. They may also be used to attach job dependencies for related groups of processes that need a single starting or ending point in the batch flow. The job does not perform any operations on the system itself.
VARIABLE_REFRESH_JOB	The underlying batch execution tool Oracle Data Integrator (ODI) uses temporary variables to hold onto any values that are specific to one batch execution, but in some cases the variables can persist across executions unless they are forcibly cleared by this program. If this program does not run, ODI may use invalid variable values from a prior run.

Job Name	Primary Functions
ETL_REFRESH_JOB	Erases the table C_LOAD_DATES, which is used to track the execution status of all AIF DATA programs linked to ODI. If any entry in the C_LOAD_DATES table is not in a “Success” status, then this job will immediately fail, because we assume a non-success entry means the prior batch execution was not properly done and needs manual intervention before we can start the next run.
REFRESH_RADM_JOB	An important part of data warehouse operations is maintaining accurate statistics, like row counts on all database tables. Statistics are used by the database to select the optimal execution plan for a given process. This program dynamically detects tables with altered data and collects new statistics on those tables and their related objects. This job’s runtime will increase relative to the amount of data changing in the system every day. The RADM in the job name refers to the data warehouse user schema, RADM01.
ANAYLZE_TEMP_TABLES_JOB	This process is similar to REFRESH_RADM_JOB, but specifically for temporary tables whose data has likely changed with every batch execution and require regularly updated statistics.
REFRESH_PDS_STATS_JOB	This process is similar to REFRESH_RADM_JOB, but specifically for RDX integration tables between the data warehouse and Planning Data Store (PDS).
RA_ERROR_COLLECTION_JOB	Maintains the table W_ETL_REJECTED_RECORDS, which tracks a summary view of rejected records that occurred in the last nightly batch cycle. This program aggregates the results of the individual errors on the E$_* set of tables and writes the final counts to W_ETL_REJECTED_RECORDS.
RA_BATCHEND_VALIDATION_JOB	Performs one final check on the C_LOAD_DATES table for any jobs that are not in a “Success” status at the end of a batch cycle and will fail if any are found. The intent of programs like this one are to ensure manual validation of batch issues that occurred for a run, so that corrective actions can be taken while all relevant data is still available. If the batch had issues and is allowed to start the next cycle without any review, all temporary data relating to that issue could be lost.

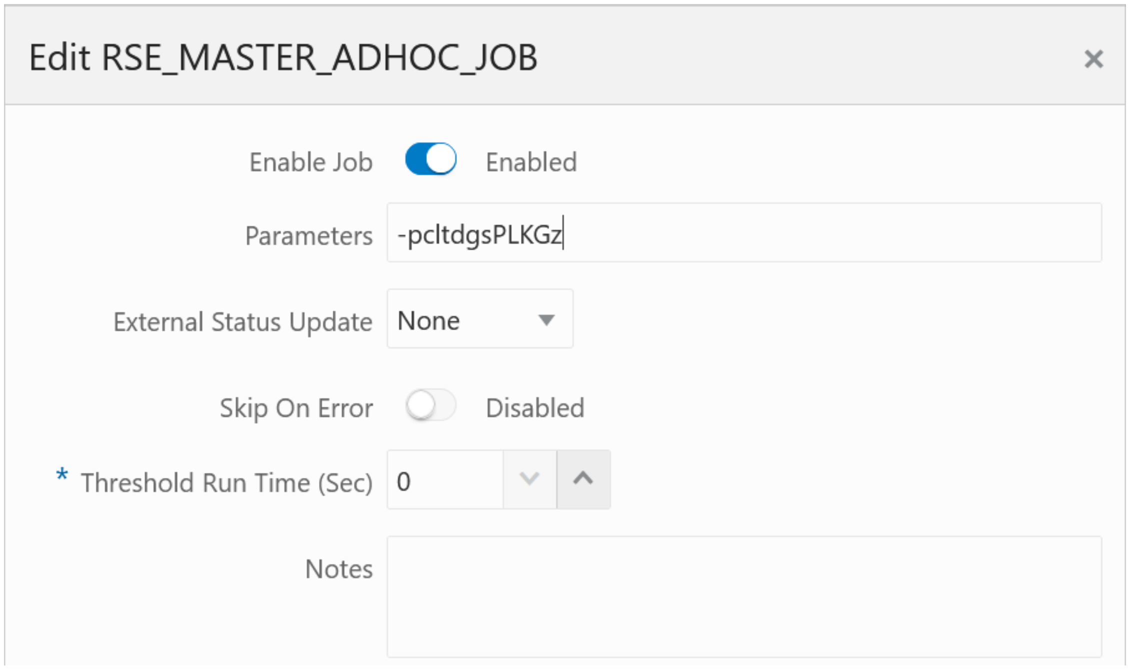
Job Name	Primary Functions
OBIEE_CACHE_CLEAR_JOB	Retail Insights relies on a data caching mechanism that is part of Oracle Analytics Server. This cache needs to be cleared at the end of every nightly batch cycle to ensure the next day's reports do not return stale results from the cache instead of accessing the latest data in the warehouse. The job is only useful to RI customers and could be disabled otherwise.
AGG_UTILITY_PRE_JOB	This program is responsible for updating a dimension lookup temporary table named `W_PRODUCT_D_RTL_CUR_TMP`, which is used as a cross-reference of product dimension records for processes such as aggregation utilities and inactive dimension record cleanup on `W_RTL_ITEM_GRP1_D`.
AGG_UTILITY_ORG_PRE_JOB	This program is responsible for updating a dimension lookup temporary table named `W_INT_ORG_DH_RTL_CUR_TMP`, which is used as a cross-reference of product dimension records for processes such as aggregation utilities and inactive dimension record cleanup.
RESET_ETL_THREAD_VAL_STG_JOB	This program updates the column `ETL_THREAD_VAL`, which is found on many older fact interfaces into the data warehouse by setting them to a fixed value of `1`. Modern data load techniques do not require explicitly setting the number of threads to be used by a process, which is what this column was used for. If a data load has a value other than `1` in this field, it could result in a failure or improper execution of the program.
*_SDE_JOB	Any AIF DATA program ending in `SDE_JOB` refers to a "Source Dependent Extract" program that is pulling data from a source outside the data warehouse and populating an internal data warehouse table. These jobs should be left disabled unless you are familiar with the source system providing data for the job and want it to be extracted (in lieu of any other load method).
LOCALIZATION_REFRESH_JOB	Retail Insights has customizable labels for metrics and folder names which come from Retail Home during the nightly batch, but they are not immediately applied to the Oracle Analytics front-end. This program is responsible for refreshing Oracle Analytics to pick up the latest labels from the database.

Job Name	Primary Functions
RI_HEALTH_CHECK_JOB	This job queries a common platform API to check on the status of various components used within our applications and will fail if any components return a status other than READY or LIVE. A failure here does not necessarily mean the batch will have any problems in other programs, for example the `ords` service may be down, which will only impact Innovation Workbench programs. You can raise an Oracle Service Request if you are unsure how to address the errors provided, but in general the job can be skipped if you are comfortable that the errors shown do not affect your batch.

4

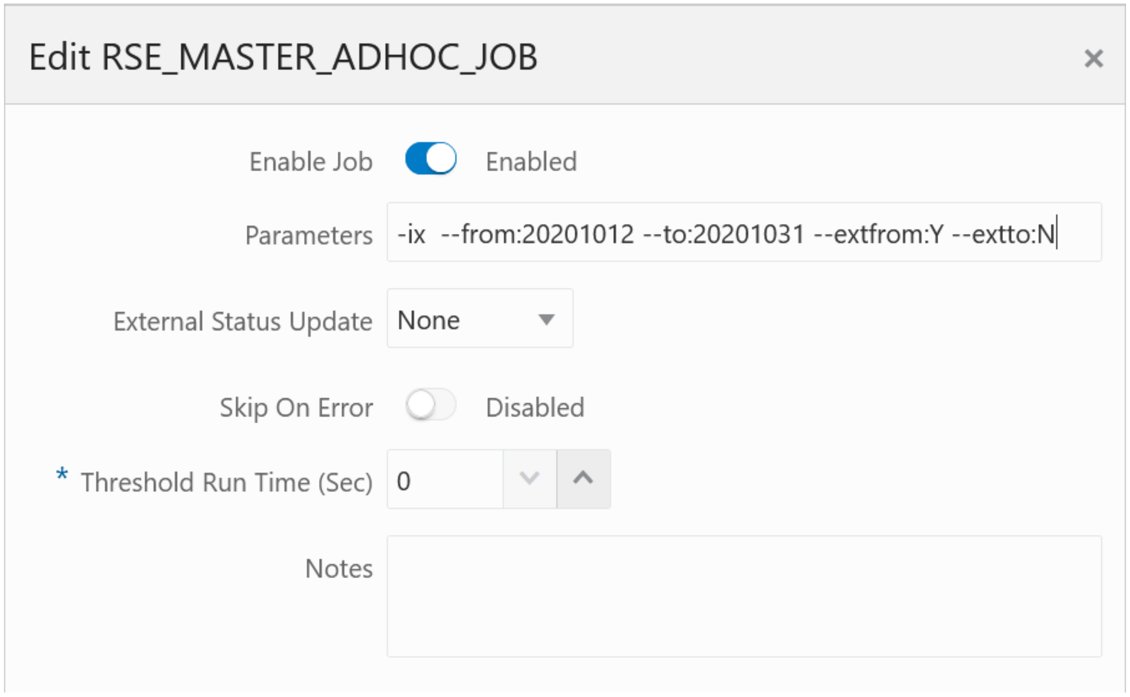
AI Foundation Applications Standalone Processes

The primary function of standalone processes in the AI Foundation Applications (AIF APPS schedule in POM) is to move data from the data warehouse or external sources into the application data models, or to move data out of the platform to send it elsewhere. These process flows differ from the AIF DATA jobs in that most processes contain only one POM job. That job contains many individual programs in it, but the execution flow is determined by parameters passed into the job. This is done by editing the job's parameters from the Batch Monitoring screen in POM:


Edit RSE_MASTER_ADHOC_JOB

Enable Job	Enabled
Parameters	-pcltdgsPLKGz
External Status Update	None
Skip On Error	Disabled
* Threshold Run Time (Sec)	0
Notes	

Each letter in the string refers to a specific program or step in the execution flow, which will be covered in more detail in the sections of this chapter. When multiple parameters are used, such as when start/end dates are provided, the format of those parameters uses double-hyphens and colons as shown here:



Edit RSE_MASTER_ADHOC_JOB

Enable Job	Enabled
Parameters	-ix --from:20201012 --to:20201031 --extfrom:Y --extto:N
External Status Update	None
Skip On Error	Disabled
* Threshold Run Time (Sec)	0
Notes	

This chapter includes the following programs:

- Customer Metrics - Base Calculation
- Customer Metrics - Final Calculation
- Customer Metrics - Loyalty Score
- Fake Customer Identification
- File Export Execution
- File Export Preparation
- Location Ranging
- Master Data Load - AA
- Master Data Load - AC
- Master Data Load - AE
- Master Data Load - Common
- Master Data Load - DT
- Master Data Load - IO
- Master Data Load - Forecast Estimation
- Master Data Load - LPO
- Master Data Load - SO
- Master Data Load - SPO
- Lifecycle Pricing Optimization Run
- Product Location Ranging
- Sales Aggregation - Customer Segment

- Sales Aggregation - Product
- Sales Aggregation - Product Attribute
- Sales Aggregation - Product Hierarchy
- Sales Aggregation - Weekly
- Sales Forecast Aggregation - Product Attribute (Legacy)
- Sales Forecast Aggregation - Product Hierarchy (Legacy)
- Sales Shares - Product Attribute
- Sales Transaction Load

Customer Metrics - Base Calculation

Module Name	RSE_CUST_ENG_METRIC_BASE_ADHOC
Description	Calculate base values for customer engagement metrics.
Dependencies	RSE_SLS_TXN_ADHOC
Business Activity	Analytical Batch Processing

Design Overview

This process aggregates sales transaction data for use in customer engagement metric calculations. The process runs for a range of weeks, depending on which weeks of sales have had a run already performed. It will output the results to a database table for downstream consumption. The `RSE_SLS_TXN_ADHOC` job is normally a prerequisite for this, as it is used to refresh or load additional sales data.

Running this process requires parameters to specify the start and end date range for which data should be processed. The `-s` parameter is for the Start Date and the `-e` parameter provides the End Date. Both are in format `YYYYMMDD`. For example:

```
-s YYYYMMDD -e YYYYMMDD -f Y
```

Key Tables Affected

Table	Usage
RSE_SLS_TXN	Input
RSE_SLS_BASE_ATTR	Output

Customer Metrics - Final Calculation

Module Name	RSE_CUST_ENG_METRIC_CALC_ADHOC
Description	Finalize the customer engagement metrics calculation.
Dependencies	RSE_CUST_ENG_METRIC_BASE_ADHOC
Business Activity	Analytical Batch Processing

Design Overview

This process calculates customer engagement metrics based on numerous inputs, including sales transaction aggregates (for behavioral and predictive metrics) and product attributes (for attribute loyalty metrics). Currently, supported product attributes must have a group type of `BRAND`, `STYLE`, `COLOR`, `LOC_LOYALTY`, or `PRICE_EFF_LOYALTY`, as defined in `RSE_BUSINESS_OBJECT_ATTR_MD`. The `RSE_CUST_ENG_METRIC_BASE_ADHOC` job is normally a prerequisite for this, as it calculates the aggregated customer sales data.

Running this process requires parameters to specify the start and end date range, for which data should be processed. The `-s` parameter is for the Start Date and the `-e` parameter provides the End Date. Both are in format `YYYYMMDD`. For example:

```
-s YYYYMMDD -e YYYYMMDD -f Y
```

Key Tables Affected

Table	Usage
RSE_SLS_BASE_ATTR	Input
RSE_PROD_ATTR	Input
RSE_CUST_ATTR_LOY_DTL	Input
RSE_CUST_SLS_ATTR	Output

Customer Metrics - Loyalty Score

Module Name	RSE_CUST_ATTR_LOY_ADHOC
Description	Calculate customer loyalty score metrics.
Dependencies	RSE_CUST_ENG_METRIC_BASE_ADHOC
Business Activity	Analytical Batch Processing

Design Overview

This process calculates customer engagement loyalty data based on numerous inputs, including sales transactions and product attributes. Currently, supported product attributes must have a group type of `BRAND`, `STYLE`, `COLOR`, `LOC_LOYALTY`, or `PRICE_EFF_LOYALTY`, as defined in `RSE_BUSINESS_OBJECT_ATTR_MD`. The `RSE_CUST_ENG_METRIC_BASE_ADHOC` job is normally a prerequisite for this, as it calculates the aggregated customer sales data.

Running this process requires parameters to specify the start and end date range, for which data should be processed. The `-s` parameter is for the Start Date and the `-e` parameter provides the End Date. Both are in format `YYYYMMDD`. For example:

```
-s YYYYMMDD -e YYYYMMDD -f Y
```

Key Tables Affected

Table	Usage
RSE_SLS_BASE_ATTR	Input
RSE_PROD_ATTR	Input
RSE_CUST_ATTR_LOY_DTL	Output

Data Cleanup Utility

Module Name	AIF_APPS_MAINT_DATA_CLEANUP_ADHOC_PROCESS
Description	Erases database tables within the AIF Apps database schema (`RASE01`)
Dependencies	None
Business Activity	Initial Data Loads

Design Overview

As you are loading and reloading data into AIF applications, you may run into conflicts or constraint violations where you need to purge old data that is causing issues. An ad hoc process is available in the AIF APPS schedule to facilitate this cleanup activity. The job invokes the `rse_data_cleanup.ksh` program. The cleanup that will be done is determined based on the parameters passed into the POM job for each execution. Initially, no parameter is indicated in the job in POM. The user enters the parameters depending on their requirement. This truncates the tables based on the entered parameters. Use the table below to identify the parameters you need to use.

`-h <input value>` This parameter indicates which hierarchy type will be cleaned up (`PRODUCT`, `LOCATION`, `CALENDAR`, `PROMOTION`, or `CUSTSEG`).

Valid values:

- `ALL`- All Hierarchy records will be cleaned up. This includes Product, Location, Calendar, Promotion and Customer Segment Hierarchies. Alternate Hierarchies are also included for location and product.
- `PRODUCT` - Product Hierarchy Data
- `LOCATION` - Location Hierarchy Data
- `CALENDAR` - Calendar Hierarchy Data
- `PROMOTION` - Promotion Hierarchy Data
- `CUSTSEG` - Customer Segment Hierarchy Data

Note:

If this parameter is not indicated, no hierarchies will be cleaned up even if you are using the other parameters to clean app data.

`-r <input value>` This parameter indicates whether all tables referencing the hierarchy IDs directly or indirectly will also be deleted.

Valid values:

- `Y` (Yes)
- `N` (No)

Default value is `Y` to ensure no stranded records will remain. This means any data referencing a hierarchy ID will be purged along with the hierarchy itself. All affected tables will be deleted in full; no data will be preserved.

`-a <input value>` This parameter indicates whether AIF application tables (such as for PMO, SPO, and so on) will be deleted.

Global Values:

- `ALL` - All application tables will be cleaned up.
- `NONE` - Application tables will NOT be cleaned up.

Default value is `ALL` to avoid any stranded records that will no longer work after data is purged. This parameter can also be used to clean up specific application tables that reference the hierarchies (directly or indirectly).

Application Values:

- `CDT` (Customer Decision Tree)
- `CIS` (Advanced Clustering & Segmentation)
- `DT` (Demand Transference)
- `IO` (IPO - Inventory Optimization)
- `MBA` (Affinity Analysis / Market Basket Analysis)
- `PMO` (Lifecycle Pricing Optimization – `PMO_*` tables)
- `PRO` (Lifecycle Pricing Optimization – `PRO_*` tables)
- `RODS` (Retail Operational Data Store)
- `SO` (Space Optimization)
- `SPO` (Size Profile Optimization)

`-o <input value>` This parameter is to indicate whether ONLY the application data will be deleted, but not any hierarchies.

Valid values:

- `Y` (Yes)
- `N` (No)

> **Note:**
>
> Application Data parameter (`-a`) should also be indicated. Default value is `N` if not indicated. PMO and RODS don't have specific app tables, hence they are not covered by this option. All affected tables will be deleted in full; no data will be preserved.

Examples

To clean up all the hierarchy tables together with dependent tables and app tables, add the following as the parameter in POM:

```
-h ALL -r Y -a ALL -o N
```

To clean up only the location hierarchy tables together with AIF Apps dependent tables and app tables, add the following as the parameter in POM:

```
-h LOCATION -r Y -a ALL -o N
```

To clean up only the application tables:

```
-a ALL -o Y
```

To clean up only a specific application (in this example, the CIS - Clustering application tables):

```
-a CIS -o Y
```

Fake Customer Identification

Module Name	RSE_FAKE_CUST_ADHOC
Description	Identify fake customers by looking through sales transaction data, so they can be automatically excluded from some applications.
Dependencies	RSE_SLS_TXN_ADHOC
Business Activity	Sales Preprocessing

Design Overview

This process analyzes sales transaction data looking for “fake” customers, which usually represent excessive sales attributed to a single customer ID. This could be caused by store cards used at the register, corporate cards used by many people, or wholesale transactions involving large numbers of sales. These kinds of transactions can have negative effects on processes like Demand Transference because they are not representative of real customer activity. The threshold for identifying a customer as fake is set using the `RSE_CONFIG` property `FAKE_CUST_DAY_TXN_THRESHOLD`.

Running this routine requires parameters to specify the start and end date range, for which data should be re-processed. The `-s` parameter is for the Start Date and the `-e` parameter provides the End Date. Both are in format `YYYYMMDD`. For example:

```
-s YYYYMMDD -e YYYYMMDD -f Y
```

Key Tables Affected

Table	Usage
RSE_SLS_TXN	Input
RSE_FAKE_CUST	Output

File Export Execution

Module Name	RSE_POST_EXPORT_ADHOC
Description	Runs the export processes for any prepared AI Foundation export files, which includes file movement, zipping, and export to SFTP.
Dependencies	RSE_EXPORT_PREP_ADHOC
Business Activity	Outbound Integrations

Design Overview

This process moves, zips, and exports files from the AI Foundation applications based on the file export type. It accepts a single input parameter for the file frequency type, using one of `DAILY`, `WEEKLY`, `QUARTERLY`, `INTRADAY`, or `ADHOC`. This process is the second step in the data flow and assumes files have already been prepared for export using the dependent process.

File Export Preparation

Module Name	RSE_EXPORT_PREP_ADHOC
Description	Export preparation job for a specific group of AI Foundation export files.
Dependencies	None
Business Activity	Outbound Integrations

Design Overview

This process will prepare a set of export files from the AI Foundation applications based on the file export type. It accepts a single input parameter for the file frequency type, using one of `DAILY`, `WEEKLY`, `QUARTERLY`, `INTRADAY`, or `ADHOC`. This is the first step in the data flow and does not perform the file movement to SFTP; it only prepares the files of the specified type so that the `RSE_POST_EXPORT_ADHOC` process can consume them.

Forecast Aggregates

Module Name	PMO_ACTIVITY_LOAD_ADHOC_PROCESS
Description	Refresh the `PMO_ACTIVITIES` table used by AIF forecasts to manually regenerate the aggregates used for specific run types.
Dependencies	None
Business Activity	Initial Data Loads

Design Overview

This process regenerates aggregate data on the `PMO_ACTIVITIES` table for a subset of forecast run types and time periods. You may want to use this process if you have loaded new historical data into AIF and want it reflected on your existing forecast run types. Make sure the run types are active in the UI before attempting to run this process on them. The process has 2 jobs: `PMO_ACTIVITY_STG_ADHOC_JOB` and `PMO_ACTIVITY_LOAD_ADHOC_JOB`. All of the parameters must be provided on the `STG` job.

Options:

- `-n` Number of weeks to process
- `-f` Force updates to existing data
- `-s` Start date in `YYYYMMDD` format
- `-e` End date in `YYYYMMDD` format
- `-S` Start calendar day ID
- `-E` End calendar day ID
- `-w` Calendar week ID to process
- `-N` New Forecast Run Type Aggregation Flag
- `-r` Forecast Run Type ID
- `-?` Display this usage information

Running this routine requires parameters to specify the start and end date range for weeks of data to process. The `-s` parameter is for the Start Date and the `-e` parameter provides the End Date. Both are in format `YYYYMMDD`. For example:

```
-s YYYYMMDD -e YYYYMMDD -f Y -N N
```

Lifecycle Pricing Optimization Run

Module Name	PRO_OPT_ADHOC
Description	Runs the LPO optimization process outside of the normal batch.
Dependencies	None
Business Activity	Analytical Batch Processing

Design Overview

This process triggers the lifecycle pricing optimization batch processing outside of the normal batch window. All of the necessary steps to calculate optimization results are included in the ad hoc job and no parameters are used. The process triggers the Java libraries on the application server that are responsible for the optimization.

Location Ranging

Module Name	DT_LOC_RANGE_ADHOC

Description	Refresh Location Ranging data for Demand Transference.
Dependencies	DT_PROD_LOC_RANGE_ADHOC
Business Activity	Application Setup

Design Overview

This process calculates SKU Counts for the available ranges of products, for a given CM Group, Store Location, and Week, which may be needed during implementation of Demand Transference when using CM Groups.

Running this routine requires parameters to specify the start and end date range for weeks of data to process. The `-s` parameter is for the Start Date and the `-e` parameter provides the End Date. Both are in format `YYYYMMDD`. For example:

```
-s YYYYMMDD -e YYYYMMDD -f Y
```

Key Tables Affected

Table	Usage
DT_PROD_LOC_STATUS	Input
DT_CM_GRP_LOC_STATUS_AGGR	Output

Master Data Load - AA

Module Name	MBA_MASTER_ADHOC_PROCESS
Description	Run the Affinity Analysis/Market Basket Analysis master script. This is the best way to execute all the initial processing steps for the MBA application module.
Dependencies	RSE_MASTER_ADHOC_PROCESS
Business Activity	Initial Data Loads

Design Overview

This process controls the master set of batch programs for loading data into the Affinity Analysis (also known as Market Basket Analysis or MBA) application. It accepts one or more single-character parameters to control which steps in the process are executed. Multiple steps executed in sequence should be passed as one string.

Options:

- `-A` Process all steps
- `-R <Option>` Resume processing all steps, starting with the step associated with the provided option (see below options) for order
- `-e` Execute MBA ETL routines
- `-c` Execute ARM configuration load routines
- `-a` Execute ARM processes

- `-r` Execute RI ARM processes
- `-b` Execute Baseline processes
- `-?` Display this usage information

Options `-A` and `-R` will enable processing of appropriate steps. Any switch provided more than once, or after a `-A` or `-R` will toggle the switch On/Off. This will enable excluding a small number of steps from processing, without requiring specifying all other switches

Example

`-Aa` will result in running all steps except `-a`

Master Data Load - AC

Module Name	CIS_MASTER_ADHOC_PROCESS
Description	Run the Advanced Clustering/Customer Segmentation master script. This is the best option to run all initial processing steps for the AC/CS modules. NOTE: when running through POM, if any -- options are required, use : instead of = to separate the option from the value.
Dependencies	RSE_MASTER_ADHOC_PROCESS
Business Activity	Initial Data Loads

Design Overview

This process controls the master set of batch programs for loading data into the Advanced Clustering and Customer Segmentation applications. It accepts one or more single-character parameters to control which steps in the process are executed. Multiple steps executed in sequence should be passed as one string.

Options:

- `-A` Process all steps
- `-R <Option>` Resume processing all steps, starting with the step associated with the provided option (see below options) for order
- `-a` Attribute Maintenance
- `-h` Product/Attribute Share Processing
- `-t` Loading cluster templates
- `-v` Setup a new version
- `-s` Update sales data for use by any versions
- `-m` Market Sales Aggregation load
- `-c` Update new versions with all the attribute summary information
- `-?` Display this usage information

Options `-A` and `-R` will enable processing of appropriate steps. Any switch provided more than once, or after a `-A` or `-R` will toggle the switch On/Off. This will enable excluding a small number of steps from processing, without requiring specifying all other switches.

Example

- `-Ah` will result in running all steps except -h
- `--from` Start date of the data processing timeframe. Must be provided in YYYYMMDD format with no spaces. For example, `--from:20170101`. Must be accompanied by the end date and optionally by the `extfrom` flag
- `--to` End date of the data processing timeframe. Must be provided in YYYYMMDD format with no spaces. For example, `--to:20170201`. Must be accompanied by the end date and optionally by the extto flag
- `--extfrom` Optional flag to indicate if the start date must be extended to the start of the week. Accepts Y or N (default). For example, `--extfrom:Y`, with no spaces
- `--extto` Optional flag to indicate if the end date must be extended to the end of the week. Accepts Y or N (default). For example, `--extto:Y`, with no spaces

Master Data Load - AE

Module Name	AE_MASTER_ADHOC_PROCESS
Description	Run the Attribute Extraction master script. This is the best way to trigger all initial processing for the AE application.
Dependencies	RSE_MASTER_ADHOC_PROCESS
Business Activity	Initial Data Loads

Design Overview

This process controls the master set of batch programs for loading data into the Attribute Extraction application. It accepts one or more single-character parameters to control which steps in the process are executed. Multiple steps executed in sequence should be passed as one string.

Options:

- `-A` Process all steps
- `-R <Option>` Resume processing all steps, starting with the step associated with the provided option (see below options) for order
- `-G` Global Lists of Strings loading
- `-C` Product Categories loading
- `-P` Product loading
- `-?` Display this usage information

Options `-A` and `-R` will enable processing of appropriate steps. Any switch provided more than once, or after a `-A` or `-R` will toggle the switch On/Off. This will enable excluding a small number of steps from processing, without requiring specifying all other switches.

Example

`-AGP` will result in running all steps except `-G` and `-P`

Master Data Load - Common

Module Name	RSE_MASTER_ADHOC_PROCESS
Description	Run the AI Foundation Cloud Services common master script. This is the first step that should be run once data has been loaded into RI, and is ready to initialize data needed by all the other application modules.
Dependencies	None
Business Activity	Initial Data Loads

Design Overview

This process controls the master set of batch programs for loading data into the Retail AI Foundation Cloud Services foundation data tables. This process is generally required as the first step in loading data to any AI Foundation application. It accepts one or more single-character parameters to control which steps in the process are executed. Multiple steps executed in sequence should be passed as one string.

Options:

- `-A` Process all steps
- `-R <Option>` Resume processing all steps, starting with the step associated with the provided option (see below options for order)
- `-p` Product Hierarchy
- `-c` CM Group Product Hierarchy
- `-l` Location Hierarchy
- `-t` Trade Area Location Hierarchy
- `-X` Alternate (Flex) Product & Location Hierarchy
- `-d` Calendar Hierarchy
- `-r` Promotion Hierarchy
- `-g` Customer Segment Hierarchy
- `-s` Consumer segment data
- `-P` Product Attributes
- `-L` Location attributes
- `-K` Like Location / Product data load
- `-G` Customer Segment Attributes
- `-z` Price zone ETL
- `-h` Holiday data load
- `-i` Inventory data load
- `-x` Sales transaction data
- `-f` Fake customer data load
- `-k` Fake customer data identification

- `-w` Weekly Aggregate Sales data (Load or Calc)
- `-a` Aggregate Sales data processing
- `-F` Forecast Aggregate Sales data processing
- `-C` Price and Cost data load
- `-u` UDA load
- `-E` Export Group Setup
- `-W` Weather Driven Demand data load
- `-T` Weekly Return transactions
- `-e` Weekly Return Aggregation
- `-S` Weekly Sales Return Price Consolidation
- `-m` Customer Engagement Attribute
- `-o` Forecast Plan Load
- `-b` Budget Allocation Load
- `-O` Order Cost data Load
- `-n` Promotion data Load
- `-D` Daily data Load
- `-U` Supplier, Supplier Item, Daily Supplier Cost, Supplier Inv Mgmt Load
- `-N` Season Phase Item Load
- `-J` Rules Engine data for PRO
- `-j` Rules Engine data for IO
- `-q` Group Flex Load
- `-H` Buyer, Allocation, Purchase Order, Transfer Loads
- `-M` Forecast Spread Profiles Load
- `-V` Forecast Lifecycle Classification Load
- `-v` Exit Dates Load
- `-B` Baseline Sales and Sku/Store Ranging Calculations
- `-?` Display this usage information

Options `-A` and `-R` will enable processing of appropriate steps. The `A` flag indicates to run all steps except the letters following it, while the `R` flag indicates to resume from the letter following it. Any switch provided more than once, or after a `-A` or `-R` will toggle the switch On/Off. This will enable excluding a small number of steps from processing, without requiring specifying all other switches.

Examples:

- `-Act` will result in running all steps except `-c` and `-t`
- `-Rc -t` will result in running all steps starting with `c`, but excluding step `t`
- `-pldgxwa` will result in extracting the product, location, calendar, customer segment, and sales data from the data warehouse and populating all the core AIF aggregates for sales (this is a common set of load steps for first-time runs)

Additional optional flags may be specified after the sequence of steps is provided, as listed below. Date ranges will apply to any step that extracts historical data, such as sales and inventory loads. If no date range is provided, then the job will attempt to determine the range of dates in the data warehouse and extract that entire range. If a step has already extracted data from the data warehouse once, then you must specify dates on additional runs of that step to ensure only that date range is re-extracted.

- `--alt_prod_hier` Run only the alternate (flex) product hierarchy load steps
- `--alt_loc_hier` Run only the alternate (flex) location hierarchy load steps
- `--alt_hier_setup` Run only the alternate (flex) hierarchy setup steps
- `--prioritizefiles` Specifies that data files should be prioritized as the source for a load instead of RI, where it is possible to get data from either source
- `--from` Start date of the data processing timeframe. Must be provided in YYYYMMDD format with no spaces. For example `--from:20170101`. Must be accompanied by the end date and optionally by the `extfrom` flag
- `--to` End date of the data processing timeframe. Must be provided in YYYYMMDD format with no spaces. For example, `--to:20170201`. Must be accompanied by the end date and optionally by the `extto` flag
- `--extfrom` Optional flag to indicate whether the start date must be extended to the start of the week. Accepts `Y` or `N` (default). For example, `--extfrom:Y`, with no spaces
- `--extto` Optional flag to indicate whether the end date must be extended to the end of the week. Accepts `Y` or `N` (default). For example, `--extto:Y`, with no spaces

Master Data Load - DT

Module Name	DT_MASTER_ADHOC_PROCESS
Description	Run the Demand Transference master script. This is the best way to run all the initial processing steps needed by the DT application module. NOTE: when running through POM, if any -- options are required, use : instead of = to separate the option from the value.
Dependencies	RSE_MASTER_ADHOC_PROCESS
Business Activity	Initial Data Loads

Design Overview

This process controls the master set of batch programs for loading data into the Demand Transference application. It accepts one or more single-character parameters to control which steps in the process are executed. Multiple steps executed in sequence should be passed as one string.

Options:

- `-A` Process all steps
- `-R <Option>` Resume processing all steps, starting with the step associated with the provided option (see below options) for order
- `-r` Load Store Sku Ranging Data
- `-l` Aggregate Location Ranging Statistics
- `-b` Calculate Baseline

- `-i` Update model intervals
- `-g` Run Group Load
- `-?` Display this usage information

Options `-A` and `-R` will enable processing of appropriate steps. Any switch provided more than once, or after a `-A` or `-R` will toggle the switch On/Off. This will enable excluding a small number of steps from processing, without requiring specifying all other switches.

Examples

- `-Ab` will result in running all steps except `-b`
- `--from` Start date of the data processing timeframe. Must be provided in YYYYMMDD format with no spaces. For example, `--from:20170101`. Must be accompanied by the end date and optionally by the `extfrom` flag
- `--to` End date of the data processing timeframe. Must be provided in YYYYMMDD format with no spaces. For example, `--to:20170201`. Must be accompanied by the end date and optionally by the `extto` flag
- `--extfrom` Optional flag to indicate if the start date must be extended to the start of the week. Accepts `Y` or `N` (default). For example, `--extfrom:Y`, with no spaces
- `--extto` Optional flag to indicate if the end date must be extended to the end of the week. Accepts `Y` or `N` (default). For example, `--extto:Y`, with no spaces

Master Data Load - Forecast Estimation

Module Name	PMO_MASTER_ADHOC_PROCESS
Description	Run the master script for processing forecast input loads and calculations. This is the way to prepare certain inputs for the forecast such as returns and activities tables.
Dependencies	RSE_MASTER_ADHOC_PROCESS
Business Activity	Initial Data Loads

Design Overview

This process controls the master set of batch programs for loading data into the Forecasting module which is in addition to the common master data load (`RSE_MASTER_ADHOC_PROCESS`). It accepts one or more single-character parameters to control which steps in the process are executed. Multiple steps executed in sequence should be passed as one string.

Options:

- `-A` Process all steps
- `-R <Option>` Resume processing all steps, starting with the step associated with the provided option (see below options) for order
- `-a` Activities
- `-d` Return Data Preparation
- `-c` Return Calculation
- `-h` Holiday load

- `-?` Display this usage information

Options `-A` and `-R` will enable processing of appropriate steps. Any switch provided more than once, or after a `-A` or `-R` will toggle the switch On/Off. This will enable excluding a small number of steps from processing, without requiring specifying all other switches.

The activities load (`-a`) supports date parameters when you are reloading data for a specific historical period. Both parameters should be provided when used.

- `--from` Start date of the data processing timeframe. Must be provided in `YYYYMMDD` format with no spaces. For example `--from:20170101`. Must be accompanied by the end date and optionally by the `extfrom` flag
- `--to` End date of the data processing timeframe. Must be provided in `YYYYMMDD` format with no spaces. For example, `--to:20170201`. Must be accompanied by the end date and optionally by the `extto` flag

Examples

`-Adh` will result in running all steps except `-d` and `-h`

`-a --from:20210502 --to:20210807` will process the historical activities data between 2021-05-02 and 2021-08-07

Master Data Load - IO

Module Name	IO_MASTER_ADHOC_PROCESS
Description	Run the Inventory Planning Optimization-Optimization master script. This is the best option for running all the initial processing steps needed by the Inventory Planning Optimization-Optimization application module.
Dependencies	RSE_MASTER_ADHOC_PROCESS
Business Activity	Initial Data Loads

Design Overview

This process controls the master set of batch programs for loading data into the Inventory Planning Optimization-Inventory Optimization application. It accepts one or more single-character parameters to control which steps in the process are executed. Multiple steps executed in sequence should be passed as one string.

Options:

- `-A` Process all steps
- `-R <Option>` Resume processing all steps, starting with the step associated with the provided option (see below options) for order
- `-a` Replenishment Attributes at Product/Location or Group level
- `-s` Seasons
- `-r` Strategy Rules
- `-C` Supply chain for Distribution Networks
- `-S` Review Schedule
- `-L` Lead Time of Internal Locations

- `-P` Supply chain for Procurement Networks
- `-Q` Supplier Review Cycles
- `-T` Lead Time of Suppliers
- `-M` Internal Order Multiples
- `-m` Supplier Order Multiples
- `-l` Rounding Levels
- `-I` Load IO rules data to rules engine
- `-N` Load N-tier interface data to rules engine
- `-?` Display this usage information

Options `-A` and `-R` will enable processing of appropriate steps. Any switch provided more than once, or after a `-A` or `-R` will toggle the switch On/Off. This will enable excluding a small number of steps from processing, without requiring specifying all other switches.

Example

`-AaP` will result in running all steps except `-a` and `-P`

Master Data Load - LPO

Module Name	PRO_MASTER_ADHOC_PROCESS
Description	Run the Lifecycle Pricing Optimization master script. This is the best option for running all the initial processing steps for the LPO application module.
Dependencies	RSE_MASTER_ADHOC_PROCESS
Business Activity	Initial Data Loads

Design Overview

This process controls the master set of batch programs for loading data into the Lifecycle Pricing Optimization application. It accepts one or more single-character parameters to control which steps in the process are executed. Multiple steps executed in sequence should be passed as one string.

Options:

- `-A` Process all steps
- `-R <Option>` Resume processing all steps, starting with the step associated with the provided option (see below options) for order
- `-b` Baseline
- `-c` Customer Segment Lifetime Value
- `-i` Inventory Aggregation
- `-f` Lifecycle Fatigue
- `-p` Promotion
- `-l` Promotion Lift

- `-C` Price and Cost for Promotion/Markdown Recommendations
- `-e` Price Elasticity
- `-L` Price Ladder
- `-r` Sales Return
- `-s` Season
- `-P` Season Product
- `-d` Season Period
- `-E` Markdown Day of Week
- `-y` Seasonality
- `-D` Model Dates
- `-O` Country Locale
- `-F` Forecast Adjustment
- `-W` Days of Week Profile
- `-u` Load Optimization rules for Promotion/Markdown Recommendations (using interface)
- `-G` Load Optimization rules for Regular Recommendations (using interface)
- `-M` Future Markdowns
- `-U` Product Location CDA Flex Facts for Promotion/Markdown Recommendations
- `-g` Product Location CDA Flex Facts for Regular Recommendations
- `-n` Inventory Aggregation for Regular Recommendations
- `-o` Price and Cost for Regular Recommendations
- `-q` Activities aggregation for Regular Recommendations
- `-S` Load Trailing metrics for Promotion/Markdown Recommendations
- `-?` Display this usage information

Options `-A` and `-R` will enable processing of appropriate steps. Any switch provided more than once, or after a `-A` or `-R` will toggle the switch On/Off. This will enable excluding a small number of steps from processing, without requiring specifying all other switches.

Example

`-AbP` will result in running all steps except `-b` and `-P`

Master Data Load - SO

Module Name	SO_MASTER_ADHOC_PROCESS
Description	Run the Space Optimization master script. This is the best way to run all the initial steps for the SO application module.
Dependencies	RSE_MASTER_ADHOC_PROCESS
Business Activity	Initial Data Loads

Design Overview

This process controls the master set of batch programs for loading data into the Assortment & Space Optimization application. It accepts one or more single-character parameters to control which steps in the process are executed. Multiple steps executed in sequence should be passed as one string.

Options:

- `-A` Process all steps
- `-R <Option>` Resume processing all steps, starting with the step associated with the provided option (see below options) for order
- `-F` Assortment Finalization
- `-a` Assortment
- `-h` Placeholder Product Loading
- `-M` Product Cluster mapping
- `-C` Assortment product location forecast and price/cost
- `-f` Assortment Forecast loading
- `-r` Replenishment Parameters
- `-S` Product Stacking Height Limit
- `-p` Pog Loading
- `-b` Bay/Fixture Loading
- `-y` Display Style Loading
- `-c` Product Fixture Configuration Loading
- `-P` Perform Product Attribute maintenance
- `-m` Assortment Mapping
- `-v` Global Validation
- `-s` Assortment to POG mapping
- `-g` POG Set location creation
- `-?` Display this usage information

Options `-A` and `-R` will enable processing of appropriate steps. Any switch provided more than once, or after a `-A` or `-R` will toggle the switch On/Off. This will enable excluding a small number of steps from processing, without requiring specifying all other switches.

Example

`-AaP` will result in running all steps except `-a` and `-P`

Master Data Load - SPO

Module Name	SPO_MASTER_ADHOC_PROCESS

Description	Run the Size Profile Optimization master script. This is the best way to run all the initial processing steps needed by the SPO application module. NOTE: when running through POM, if any -- options are required, use : instead of = to separate the option from the value.
Dependencies	RSE_MASTER_ADHOC_PROCESS
Business Activity	Initial Data Loads

Design Overview

This process controls the master set of batch programs for loading data into the Size Profile Optimization application. It accepts one or more single-character parameters to control which steps in the process are executed. Multiple steps executed in sequence should be passed as one string.

Options:

- `-A` Process all steps
- `-R <Option>` Resume processing all steps, starting with the step associated with the provided option (see below options) for order
- `-S` Season Data Load
- `-r` Size Range Data Load
- `-s` Size Data Load
- `-p` Product Size Data Load
- `-l` Sub-Size Range Product Location Data Load
- `-?` Display this usage information

Options `-A` and `-R` will enable processing of appropriate steps. Any switch provided more than once, or after a `-A` or `-R` will toggle the switch On/Off. This will enable excluding a small number of steps from processing, without requiring specifying all other switches.

Example

- `-Ar` will result in running all steps except `-r`
- `--from` Start date of the data processing timeframe. Must be provided in YYYYMMDD format with no spaces. For example, `--from:20170101`. Must be accompanied by the end date and optionally by the `extfrom` flag
- `--to` End date of the data processing timeframe. Must be provided in YYYYMMDD format with no spaces. For example, `--to:20170201`. Must be accompanied by the end date and optionally by the `extto` flag
- `--extfrom` Optional flag to indicate if the start date must be extended to the start of the week. Accepts `Y` or `N` (default). For example, `--extfrom:Y`, with no spaces
- `--extto` Optional flag to indicate if the end date must be extended to the end of the week. Accepts `Y` or `N` (default). For example, `--extto:Y`, with no spaces

Product Location Ranging

Module Name	DT_PROD_LOC_RANGE_ADHOC

Description	Refresh Product Location Ranging data for Demand Transference.
Dependencies	W_RTL_IT_LC_D_JOB (in RI)
Business Activity	Application Setup

Design Overview

This process extracts the item/location ranging information from Retail Insights table `W_RTL_IT_LC_D`. This process is also performed in the DT master batch process, but it can be run on its own if you are modifying the data and need to reload it.

Running this routine requires parameters to specify the start and end date range, for which data should be re-processed from the `W_RTL_IT_LC_D` table or from AI Foundation sales tables. The `-s` parameter is for the Start Date and the `-e` parameter provides the End Date. Both are in format `YYYYMMDD`. For example:

```
-s YYYYMMDD -e YYYYMMDD -f Y
```

Key Tables Affected

Table	Usage
W_RTL_IT_LC_D	Input
DT_PROD_LOC_LAST_SLS	Input
DT_PROD_LOC_STATUS	Output

Sales Aggregation – Cumulative Sales

Module Name	PMO_CUMUL_SLS_ADHOC_PROCESS
Description	Creates aggregate cumulative sales data for Lifecycle Pricing Optimization.
Dependencies	RSE_MASTER_ADHOC_PROCESS
Business Activity	Initial Data Loads

Design Overview

This process allows the user to execute the cumulative sales aggregation for Lifecycle Pricing Optimization application in an ad hoc manner. When the user creates a new forecast run type, this aggregation is automatically called as part of “Start Data Aggregation”. This requires that sales aggregations have already been performed using the `RSE_MASTER` ad hoc process, and inventory position/receipts data has already been loaded into the data warehouse and AIF (so that first receipt dates can be used).

Running this process requires parameters to specify the start and end date range for which data should be processed. The `-s` parameter is for the Start Date and the `-e` parameter provides the End Date. Both are in format `YYYYMMDD`. For example:

```
-s YYYYMMDD -e YYYYMMDD -f Y
```

The process has the following list of supported options. All job parameters are passed into the `PMO_CUMUL_SLS_SETUP_ADHOC_JOB` process when invoking it from Postman.

- `-n` Number of weeks to process
- `-f` Force update of existing data - `Y`/`N` (Default)
- `-s` Start date `yyyymmdd`
- `-e` End date `yyyymmdd`
- `-S` Start calendar day ID
- `-E` End calendar day ID
- `-w` Calendar Week ID to process
- `-N` New Forecast Run Type Aggregation Flag - `Y`/`N`

Key Tables Affected

Table	Usage
RSE_SLS_PR_LC_CS_WK	Input
RSE_INV_PR_LC_HIST	Input
RSE_FCST_RUN_TYPE	Input
RSE_FCST_DFLT_PARAMETER	Input
PMO_CUM_SLS	Output

Sales Aggregation - Customer Segment

Module Name	RSE_WKLY_SLS_CUST_SEG_ADHOC
Description	Aggregates Sales Transaction data to Weekly Customer Segment Sales tables.
Dependencies	RSE_SLS_TXN_ADHOC
Business Activity	Initial Data Loads

Design Overview

This process aggregates sales data by customer segment for use in AI Foundation applications. The `RSE_SLS_TXN_ADHOC` job is normally a prerequisite for this, as it is used to refresh or load additional sales data.

Running this process requires parameters to specify the start and end date range, for which data should be processed. The `-s` parameter is for the Start Date and the `-e` parameter provides the End Date. Both are in format `YYYYMMDD`. For example:

```
-s YYYYMMDD -e YYYYMMDD -f Y
```

Key Tables Affected

Table	Usage
RSE_SLS_TXN	Input
RSE_CUSTSEG_CUST_XREF	Input
RSE_CUSTSEG_SRC_XREF	Input
RSE_SLS_PR_LC_CS_WK	Output

Sales Aggregation - Product

Module Name	RSE_WKLY_SLS_PR_AGGR_ADHOC
Description	Calculates Product-based sales aggregate tables.
Dependencies	RSE_WKLY_SLS_ADHOC
Business Activity	Initial Data Loads

Design Overview

This process aggregates sales data by product for use in AI Foundation applications. The `RSE_WKLY_SLS_ADHOC` job is normally a prerequisite for this, as it is used to refresh or load additional sales data.

Running this process requires parameters to specify the start and end date range, for which data should be processed. The `-s` parameter is for the Start Date and the `-e` parameter provides the End Date. Both are in format `YYYYMMDD`. For example:

```
-s YYYYMMDD -e YYYYMMDD -f Y
```

Key Tables Affected

Table	Usage
RSE_SLS_PR_LC_WK	Input
RSE_SLS_PR_WK_A	Output

Sales Aggregation - Product Attribute

Module Name	RSE_WKLY_SLS_PH_ATTR_AGGR_ADHOC
Description	Calculates Product Attribute-based sales aggregate tables.
Dependencies	RSE_WKLY_SLS_ADHOC
Business Activity	Initial Data Loads

Design Overview

This process aggregates sales data by product attribute and product hierarchy levels for use in AI Foundation applications. The `RSE_WKLY_SLS_ADHOC` job is normally a prerequisite for this, as it is used to refresh or load additional sales data.

Running this process requires parameters to specify the start and end date range, for which data should be processed. The `-s` parameter is for the Start Date and the `-e` parameter provides the End Date. Both are in format `YYYYMMDD`. For example:

```
-s YYYYMMDD -e YYYYMMDD -f Y
```

Key Tables Affected

Table	Usage
RSE_SLS_PR_LC_WK	Input
RSE_PROD_ATTR	Input
RSE_SLS_PH_ATTR_LC_WK_A	Output

Sales Aggregation - Product Hierarchy

Module Name	RSE_WKLY_SLS_PH_AGGR_ADHOC
Description	Calculates Product Hierarchy-based sales aggregate tables.
Dependencies	RSE_WKLY_SLS_ADHOC
Business Activity	Initial Data Loads

Design Overview

This process aggregates sales data by product hierarchy levels for use in AI Foundation applications. The `RSE_WKLY_SLS_ADHOC` job is normally a prerequisite for this, as it is used to refresh or load additional sales data.

Running this process requires parameters to specify the start and end date range, for which data should be processed. The `-s` parameter is for the Start Date and the `-e` parameter provides the End Date. Both are in format `YYYYMMDD`. For example:

```
-s YYYYMMDD -e YYYYMMDD -f Y
```

Key Tables Affected

Table	Usage
RSE_SLS_PR_LC_WK	Input
RSE_SLS_PH_LC_WK_A	Output

Sales Aggregation - Weekly

Module Name	RSE_WKLY_SLS_ADHOC
Description	Aggregates Sales Transaction data to week level tables.
Dependencies	RSE_SLS_TXN_ADHOC
Business Activity	Initial Data Loads

Design Overview

This process aggregates sales data by product hierarchy levels for use in AI Foundation applications. The `RSE_SLS_TXN_ADHOC` job is normally a prerequisite for this, as it is used to refresh or load additional sales data.

Running this process requires parameters to specify the start and end date range, for which data should be processed. The `-s` parameter is for the Start Date and the `-e` parameter provides the End Date. Both are in format `YYYYMMDD`. For example:

```
-s YYYYMMDD -e YYYYMMDD -f Y
```

Key Tables Affected

Table	Usage
RSE_SLS_TXN	Input
RSE_SLS_PR_LC_WK	Output

Sales Forecast Aggregation - Product Attribute (Legacy)

Module Name	RSE_SLSFC_PH_ATTR_AGGR_ADHOC
Description	Calculates Product Attribute-based sales forecast aggregate tables.
Dependencies	RSE_SLSFC_PH_AGGR_ADHOC
Business Activity	Initial Data Loads

Design Overview

This process aggregates sales forecast data by product attribute and product hierarchy levels for use in AI Foundation applications. The `RSE_SLSFC_PH_AGGR_ADHOC` job is normally a prerequisite for this, as it is used to refresh or load additional sales forecast data.

Running this process requires parameters to specify the start and end date range, for which data should be processed. The `-s` parameter is for the Start Date and the `-e` parameter provides the End Date. Both are in format `YYYYMMDD`. For example:

```
-s YYYYMMDD -e YYYYMMDD -f Y
```

Note:

This is a legacy process which uses a forecast interface from the data warehouse that has been deprecated.

Sales Forecast Aggregation - Product Hierarchy (Legacy)

Module Name	RSE_SLSFC_PH_AGGR_ADHOC
Description	Calculates Product Hierarchy-based sales forecast aggregate tables.
Dependencies	None
Business Activity	Initial Data Loads

Design Overview

This process aggregates sales forecast data by product hierarchy levels for use in AI Foundation applications.

Running this process requires parameters to specify the start and end date range, for which data should be processed. The `-s` parameter is for the Start Date and the `-e` parameter provides the End Date. Both are in format `YYYYMMDD`. For example:

```
-s YYYYMMDD -e YYYYMMDD -f Y
```

Note:

This is a legacy process which uses a forecast interface from the data warehouse that has been deprecated.

Sales Shares - Product Attribute

Module Name	AC_PROD_ATTR_LOC_SHARE_ADHOC
Description	Calculate product attribute sales shares for use in Advanced Clustering.
Dependencies	RSE_WKLY_SLS_ADHOC
Business Activity	Initial Data Loads

Design Overview

This process aggregates sales shares by product attribute for use in the Advanced Clustering application, specifically for use in clustering by product attribute. The `RSE_WKLY_SLS_ADHOC` job is normally a prerequisite for this, as it is used to refresh or load additional sales data at week level.

You also must choose which attribute mode is applicable for AC. If it is specified as `CDT` in `RSE_CONFIG` property `PERF_CIS_APPROACH`, then this program will expect additional information for CDT-like attribute groups in `RSE_PROD_ATTR_GRP` and `RSE_PROD_ATTR_GRP_VALUE_MAP`. It will also use sales data from `RSE_SLS_PH_ATTR_LC_WK_A`. For any other configuration, these tables are not required and a more generic approach will be taken.

Running this process requires parameters to specify the start and end date range, for which data should be processed. The `-s` parameter is for the Start Date and the `-e` parameter provides the End Date. Both are in format `YYYYMMDD`. For example:

```
-s YYYYMMDD -e YYYYMMDD -f Y
```

Key Tables Affected

Table	Usage
CIS_TCRITERIA_ATTR	Input
CIS_BUS_OBJ_TCRITERIA_ATT_XREF	Input
RSE_PROD_ATTR_GRP	Input
RSE_PROD_ATTR_GRP_VALUE_MAP	Input
RSE_SLS_PH_LC_WK_A	Input
RSE_SLS_PH_ATTR_LC_WK_A	Input
CIS_PROD_ATTR_LOC_SHARE	Output

Sales Transaction Load

Module Name	RSE_SLS_TXN_ADHOC
Description	Performs bulk retrieval of Sales Transaction data.
Dependencies	W_RTL_SLS_TRX_IT_LC_DY_F_JOB (in RI)
Business Activity	Initial Data Loads

Design Overview

This process extracts sales transactions from Retail Insights for use in all AI Foundation applications. The `W_RTL_SLS_TRX_IT_LC_DY_F` table in the data warehouse is the source of this data and the data warehouse must be populated with sales before this program runs.

Running this process requires parameters to specify the start and end date range, for which data should be processed. The `-s` parameter is for the Start Date and the `-e` parameter provides the End Date. Both are in format `YYYYMMDD`. For example:

```
-s YYYYMMDD -e YYYYMMDD -f Y
```

Key Tables Affected

Table	Usage
W_RTL_SLS_TRX_IT_LC_DY_F	Input
RSE_SLS_TXN	Output

5

AI Foundation Data Standalone Process Flows

Standalone processes are separated across several different flows within POM depending on the order in which you need to execute them and the dependencies to be followed. This chapter provides a tabular view of related jobs across the different process flows in the AIF DATA schedule so that you know which jobs are safe to enable or disable, depending on the files you're attempting to load into the platform. It is always a best practice to disable jobs in POM if you are not actively using them, both to reduce runtimes and to avoid loading data you did not intend to process.

Process Flows for DAT Files

The table below shows the standalone process flows for any file with a `.dat` extension. Please note that the following list of jobs should remain enabled and are usually applicable to all batch runs, so they're not included in the table.

- VARIABLE_REFRESH_JOB
- ETL_REFRESH_JOB
- ETL_BUSINESS_DATE_JOB
- W_RTL_CURR_MCAL_G_JOB
- RI_UPDATE_TENANT_JOB (as part of HIST_ZIP_FILE_LOAD_ADHOC)

Table 5-1 DAT File Process Flows

Input Files (DAT)	DAT_REPROCESS_ADHOC	RI_DIM_INITIAL_ADHOC
W_CODE_DS.dat	W_CODE_DS_COPY_JO W_CODE_DS_STG_JOB	W_CODE_D_JOB
W_DOMAIN_MEMBER_DS_TL.dat	W_DOMAIN_MEMBER_DS_TL_COPY_JOB W_DOMAIN_MEMBER_DS_TL_STG_JOB	W_DOMAIN_MEMBER_LKP_TL_JOB
W_EMPLOYEE_DS.dat	W_EMPLOYEE_DS_COPY_JOB W_EMPLOYEE_DS_STG_JOB	W_EMPLOYEE_D_JOB
W_EXCH_RATE_GS.dat	W_EXCH_RATE_GS_COPY_JOB W_EXCH_RATE_GS_STG_JOB	W_EXCH_RATE_G_JOB
W_HOUSEHOLD_DS.dat	W_HOUSEHOLD_DS_COPY_JOB W_HOUSEHOLD_DS_STG_JOB	W_HOUSEHOLD_D_JOB
W_INT_ORG_ATTR_DS.dat W_INT_ORG_DS.dat W_INT_ORG_DS_TL.dat	W_INT_ORG_ATTR_DS_COPY_JOB W_INT_ORG_ATTR_DS_STG_JOB W_INT_ORG_DS_COPY_JOB W_INT_ORG_DS_STG_JOB W_INT_ORG_DS_TL_COPY_JOB W_INT_ORG_DS_TL_STG_JOB	W_INT_ORG_D_TYPE1_JOB W_RTL_ORG_RECLASS_TMP_JOB

Table 5-1 (Cont.) DAT File Process Flows

Input Files (DAT)	DAT_REPROCESS_ADHOC	RI_DIM_INITIAL_ADHOC
W_INT_ORG_DS_CFA.dat	W_INT_ORG_DS_CFA_COPY_JOB W_INT_ORG_DS_CFA_STG_JOB	W_INT_ORG_D_CFA_JOB
W_INT_ORG_DHS.dat	W_INT_ORG_DHS_COPY_JOB W_INT_ORG_DHS_STG_JOB	W_INT_ORG_DH_TYPE1_JOB W_INT_ORG_DH_RTL_TMP_JOB
W_PARTY_ATTR_DS.dat	W_PARTY_ATTR_DS_COPY_JOB W_PARTY_ATTR_DS_STG_JOB	W_PARTY_ATTR_D_JOB
W_PARTY_ORG_DS.dat	W_PARTY_ORG_DS_COPY_JOB W_PARTY_ORG_DS_STG_JOB	W_PARTY_ORG_D_JOB
W_PARTY_ORG_DS_CFA.dat	W_PARTY_ORG_DS_CFA_COPY_JOB W_PARTY_ORG_DS_CFA_STG_JOB	W_PARTY_ORG_D_CFA_JOB
W_PARTY_PER_DS.dat	W_PARTY_PER_DS_COPY_JOB W_PARTY_PER_DS_STG_JOB	W_PARTY_PER_D_JOB
W_PROD_CAT_DHS.dat	W_PROD_CAT_DHS_COPY_JOB W_PROD_CAT_DHS_STG_JOB	W_PROD_CAT_DH_TYPE1_JOB W_PROD_CAT_DH_SC_RTL_TMP_JOB
W_PROD_CAT_DHS_CFA.dat	W_PROD_CAT_DHS_CFA_COPY_JOB W_PROD_CAT_DHS_CFA_STG_JOB	W_PROD_CAT_DH_CFA_JOB
W_PRODUCT_ATTR_DS.dat	W_PRODUCT_ATTR_DS_COPY_JOB W_PRODUCT_ATTR_DS_STG_JOB	W_PRODUCT_ATTR_D_JOB W_RTL_PRODUCT_ATTR_UDA_D_JOB
W_PRODUCT_DS.dat W_PRODUCT_DS_TL.dat	W_PRODUCT_DS_COPY_JOB W_PRODUCT_DS_STG_JOB W_PRODUCT_DS_TL_COPY_JOB W_PRODUCT_DS_TL_STG_JOB	W_PRODUCT_D_TYPE1_JOB W_RTL_PROD_RECLASS_TMP_INITIAL_JOB W_PRODUCT_D_RTL_TMP_JOB W_RTL_PROD_RECLASS_TMP_JOB W_RTL_PRODUCT_AGGR_D_JOB
W_PRODUCT_DS_CFA.dat	W_PRODUCT_DS_CFA_COPY_JOB W_PRODUCT_DS_CFA_STG_JOB	W_PRODUCT_D_CFA_JOB
W_REASON_DS.dat	W_REASON_DS_COPY_JOB W_REASON_DS_STG_JOB	W_REASON_D_JOB
W_RTL_ALC_DETAILS_DS.dat	W_RTL_ALC_DETAILS_DS_COPY_JOB W_RTL_ALC_DETAILS_DS_STG_JOB	W_RTL_ALC_DETAILS_D_JOB
W_RTL_ALLOC_CHRG_DETAILS_DS.dat	W_RTL_ALLOC_CHRG_DETAILS_DS_COPY_JOB W_RTL_ALLOC_CHRG_DETAILS_DS_STG_JOB	W_RTL_ALLOC_CHRG_DETAILS_D_JOB
W_RTL_BUYER_DS.dat	W_RTL_BUYER_DS_COPY_JOB W_RTL_BUYER_DS_STG_JOB	W_RTL_BUYER_D_JOB

Table 5-1 (Cont.) DAT File Process Flows

Input Files (DAT)	DAT_REPROCESS_ADHOC	RI_DIM_INITIAL_ADHOC
W_RTL_CHANNEL_DS.dat	W_RTL_CHANNEL_DS_COPY_JOB W_RTL_CHANNEL_DS_STG_JOB	W_RTL_CHANNEL_D_JOB
W_RTL_CLSTR_GRP_DS.dat	W_RTL_CLSTR_GRP_DS_COPY_JOB W_RTL_CLSTR_GRP_DS_STG_JOB	W_RTL_CLSTR_GRP_D_JOB
W_RTL_CLSTR_HDR_DS.dat	W_RTL_CLSTR_HDR_DS_COPY_JOB W_RTL_CLSTR_HDR_DS_STG_JOB W_DOMAIN_MEMBER_DS_TL_ORASE_JOB	W_RTL_CLSTR_HDR_D_JOB W_DOMAIN_MEMBER_LKP_TL_JOB
W_RTL_CLSTR_GRP_HDR_LC_DS.dat	W_RTL_CLSTR_GRP_HDR_LC_DS_COPY_JOB W_RTL_CLSTR_GRP_HDR_LC_DS_STG_JOB	W_RTL_CLSTR_GRP_HDR_LC_D_JOB
W_RTL_CLSTR_GRP_PRD_DS.dat	W_RTL_CLSTR_GRP_PRD_DS_COPY_JOB W_RTL_CLSTR_GRP_PRD_DS_STG_JOB	W_RTL_CLSTR_GRP_IT_D_JOB
W_RTL_CMG_PRODUCT_MTX_DS.dat	W_RTL_CMG_PRODUCT_MTX_DS_COPY_JOB W_RTL_CMG_PRODUCT_MTX_DS_STG_JOB	W_RTL_CMG_PRODUCT_MTX_D_JOB
W_RTL_CO_HEAD_DS.dat	W_RTL_CO_HEAD_DS_COPY_JOB W_RTL_CO_HEAD_DS_STG_JOB	W_RTL_CO_HEAD_D_JOB
W_RTL_CO_LINE_DS.dat	W_RTL_CO_LINE_DS_COPY_JOB W_RTL_CO_LINE_DS_STG_JOB	W_RTL_CO_LINE_D_JOB
W_RTL_CO_SHIP_METHOD_DS.dat	W_RTL_CO_SHIP_METHOD_DS_COPY_JOB W_RTL_CO_SHIP_METHOD_DS_STG_JOB	W_RTL_CO_SHIP_METHOD_D_JOB
W_RTL_CO_SHIP_TYPE_DS.dat	W_RTL_CO_SHIP_TYPE_DS_COPY_JOB W_RTL_CO_SHIP_TYPE_DS_STG_JOB	W_RTL_CO_SHIP_TYPE_D_JOB
W_RTL_CODE_DS.dat	W_RTL_CODE_DS_COPY_JOB W_RTL_CODE_DS_STG_JOB	W_RTL_CODE_D_JOB
W_RTL_COMP_STORE_DS.dat	W_RTL_COMP_STORE_DS_COPY_JOB W_RTL_COMP_STORE_DS_STG_JOB	W_RTL_COMP_STORE_D_JOB
W_RTL_COUPON_DS.dat W_RTL_COUPON_DS_TL.dat	W_RTL_COUPON_DS_COPY_JOB W_RTL_COUPON_DS_STG_JOB W_RTL_COUPON_DS_TL_COPY_JOB W_RTL_COUPON_DS_TL_STG_JOB	W_RTL_COUPON_D_JOB

Table 5-1 (Cont.) DAT File Process Flows

Input Files (DAT)	DAT_REPROCESS_ADHOC	RI_DIM_INITIAL_ADHOC
W_RTL_CUST_ADDRESS_DS.dat	W_RTL_CUST_ADDRESS_DS_COPY_JOB W_RTL_CUST_ADDRESS_DS_STG_JOB	W_RTL_CUST_ADDRESS_D_JOB
W_RTL_CUST_CUSTSEG_DS.dat	W_RTL_CUST_CUSTSEG_DS_COPY_JOB W_RTL_CUST_CUSTSEG_DS_STG_JOB	W_RTL_CUST_CUSTSEG_D_JOB
W_RTL_CUST_DEDUP_DS.dat	W_RTL_CUST_DEDUP_DS_COPY_JOB W_RTL_CUST_DEDUP_DS_STG_JOB	W_PARTY_PER_D_JOB (data used implicitly during dim load)
W_RTL_CUST_HOUSEHOLD_DS.dat	W_RTL_CUST_HOUSEHOLD_DS_COPY_JOB W_RTL_CUST_HOUSEHOLD_DS_STG_JOB	W_RTL_CUST_HOUSEHOLD_D_JOB
W_RTL_CUSTSEG_ALLOC_DS.dat	W_RTL_CUSTSEG_ALLOC_DS_COPY_JOB W_RTL_CUSTSEG_ALLOC_DS_STG_JOB	W_RTL_CUSTSEG_ALLOC_D_JOB
W_RTL_CUSTSEG_ATTR_DS.dat	W_RTL_CUSTSEG_ATTR_DS_COPY_JOB W_RTL_CUSTSEG_ATTR_DS_STG_JOB	W_RTL_CUSTSEG_D_JOB (data used implicitly during dim load)
W_RTL_DEAL_DS.dat	W_RTL_DEAL_DS_COPY_JOB W_RTL_DEAL_DS_STG_JOB	W_RTL_DEAL_D_JOB
W_RTL_DEPT_CHRG_DETAILS_DS.dat	W_RTL_DEPT_CHRG_DETAILS_DS_COPY_JOB W_RTL_DEPT_CHRG_DETAILS_DS_STG_JOB	W_RTL_DEPT_CHRG_DETAILS_D_JOB
W_RTL_DIFF_GRP_DS.dat W_RTL_DIFF_GRP_DS_TL.dat	W_RTL_DIFF_GRP_DS_COPY_JOB W_RTL_DIFF_GRP_DS_STG_JOB W_RTL_DIFF_GRP_DS_TL_COPY_JOB W_RTL_DIFF_GRP_DS_TL_STG_JOB	W_RTL_DIFF_GRP_D_JOB
W_RTL_DIFF_RNG_DS.dat	W_RTL_DIFF_RNG_DS_COPY_JOB	W_RTL_DIFF_RNG_D_JOB
W_RTL_DIFF_RNG_DS_TL.dat	W_RTL_DIFF_RNG_DS_STG_JOB W_RTL_DIFF_RNG_DS_TL_COPY_JOB W_RTL_DIFF_RNG_DS_TL_STG_JOB	W_RTL_DIFF_RNG_D_JOB
W_RTL_DISCOUNT_TYPE_DS.dat	W_RTL_DISCOUNT_TYPE_DS_COPY_JOB W_RTL_DISCOUNT_TYPE_DS_STG_JOB	W_RTL_DISCOUNT_TYPE_D_JOB

Table 5-1 (Cont.) DAT File Process Flows

Input Files (DAT)	DAT_REPROCESS_ADHOC	RI_DIM_INITIAL_ADHOC
W_RTL_ELC_COMP_DS.dat	W_RTL_ELC_COMP_DS_COPY_JOB W_RTL_ELC_COMP_DS_STG_JOB	W_RTL_ELC_COMP_D_JOB
W_RTL_HOUSEHOLD_COMP_DS.dat	W_RTL_HOUSEHOLD_COMP_DS_COPY_JOB W_RTL_HOUSEHOLD_COMP_DS_STG_JOB	W_RTL_HOUSEHOLD_COMP_D_JOB
W_RTL_HOUSEHOLD_GRP_DS.dat	W_RTL_HOUSEHOLD_GRP_DS_COPY_JOB W_RTL_HOUSEHOLD_GRP_DS_STG_JOB	W_RTL_HOUSEHOLD_GRP_D_JOB
W_RTL_IT_LC_DS_CFA.dat	W_RTL_IT_LC_DS_CFA_COPY_JOB W_RTL_IT_LC_DS_CFA_STG_JOB	W_RTL_IT_LC_D_CFA_JOB
W_RTL_IT_SUPPLIER_DS.dat	W_RTL_IT_SUPPLIER_DS_COPY_JOB W_RTL_IT_SUPPLIER_DS_STG_JOB	W_RTL_IT_SUPPLIER_D_JOB
W_RTL_ITEM_CHRG_DETAILS_DS.dat	W_RTL_ITEM_CHRG_DETAILS_DS_COPY_JOB W_RTL_ITEM_CHRG_DETAILS_DS_STG_JOB	W_RTL_ITEM_CHRG_DETAILS_D_JOB
W_RTL_ITEM_GRP1_DS.dat	W_RTL_ITEM_GRP1_DS_COPY_JOB W_RTL_ITEM_GRP1_DS_STG_JOB	W_RTL_ITEM_GRP1_D_ITEMBRAND_JOB W_RTL_ITEM_GRP1_D_ITEMCOLOR_JOB W_RTL_ITEM_GRP1_D_ITEMDIFF_JOB W_RTL_ITEM_GRP1_D_ITEMFAB_JOB W_RTL_ITEM_GRP1_D_ITEMLIST_JOB W_RTL_ITEM_GRP1_D_ITEMLV_JOB W_RTL_ITEM_GRP1_D_ITEMSCEN_JOB W_RTL_ITEM_GRP1_D_ITEMSIZE_JOB W_RTL_ITEM_GRP1_D_ITEMSTYLE_JOB W_RTL_ITEM_GRP1_D_ITEMUDA_JOB W_RTL_PRODUCT_ATTR_UDA_D_JOB
W_RTL_ITEM_GRP3_DS.dat	W_RTL_ITEM_GRP3_DS_COPY_JOB W_RTL_ITEM_GRP3_DS_STG_JOB	W_RTL_ITEM_GRP3_D_JOB
W_RTL_LOC_LIST_DS.dat	W_RTL_LOC_LIST_DS_COPY_JOB W_RTL_LOC_LIST_DS_STG_JOB	W_RTL_LOC_LIST_D_JOB

Table 5-1 (Cont.) DAT File Process Flows

Input Files (DAT)	DAT_REPROCESS_ADHOC	RI_DIM_INITIAL_ADHOC
W_RTL_LOC_TRAIT_DS.dat	W_RTL_LOC_TRAIT_DS_COPY_JOB W_RTL_LOC_TRAIT_DS_STG_JOB	W_RTL_LOC_TRAIT_D_JOB
W_RTL_LOC_TRAIT_DS_TL.dat	W_RTL_LOC_TRAITS_DS_TL_COPY_JOB W_RTL_LOC_TRAITS_DS_TL_STG_JOB	W_RTL_LOC_TRAIT_D_JOB
W_RTL_LOC_STOCK_CNT_DS.dat	W_RTL_LOC_STOCK_CNT_DS_COPY_JOB W_RTL_LOC_STOCK_CNT_DS_STG_JOB	W_RTL_LOC_STOCK_CNT_D_JOB
W_RTL_MCAL_DAY_DS_CFA.dat	W_RTL_MCAL_DAY_DS_CFA_COPY_JOB W_RTL_MCAL_DAY_DS_CFA_STG_JOB	W_RTL_MCAL_DAY_D_CFA_JOB
W_RTL_NON_MERCH_CODE_DS.dat	W_RTL_NON_MERCH_CODE_DS_COPY_JOB W_RTL_NON_MERCH_CODE_DS_STG_JOB	W_RTL_NON_MERCH_CODE_D_JOB
W_RTL_ORG_FIN_DS.dat	W_RTL_ORG_FIN_DS_COPY_JOB W_RTL_ORG_FIN_DS_STG_JOB	W_RTL_ORG_FIN_D_JOB
W_RTL_PARTY_PER_ATTR_DS.dat	W_RTL_PARTY_PER_ATTR_DS_COPY_JOB W_RTL_PARTY_PER_ATTR_DS_STG_JOB	W_PARTY_ATTR_D_UDA_JOB
W_RTL_PHASE_DS.dat	W_RTL_PHASE_DS_COPY_JOB W_RTL_PHASE_DS_STG_JOB	W_RTL_PHASE_D_JOB
W_RTL_PO_DETAILS_DS.dat	W_RTL_PO_DETAILS_DS_COPY_JOB W_RTL_PO_DETAILS_DS_STG_JOB	W_RTL_PO_DETAILS_D_JOB
W_RTL_PRICE_CLR_IT_LC_DS.dat	W_RTL_PRICE_CLR_IT_LC_DS_COPY_JOB W_RTL_PRICE_CLR_IT_LC_DS_STG_JOB	W_RTL_PRICE_CLR_IT_LC_D_JOB
W_RTL_PROD_HIER_ATTR_LKP_DHS.dat	W_RTL_PROD_HIER_ATTR_LKP_DHS_COPY_JOB W_RTL_PROD_HIER_ATTR_LKP_DHS_STG_JOB	W_RTL_PROD_HIER_ATTR_LKP_DH_JOB W_RTL_PROD_HIER_ATTR_LKP_DH_IM_JOB
W_RTL_PROD_HIER_IMAGE_DS.dat	W_RTL_PROD_HIER_IMAGE_DS_COPY_JOB W_RTL_PROD_HIER_IMAGE_DS_STG_JOB	W_RTL_PROD_HIER_ATTR_LKP_DH_IM_JOB
W_RTL_PRODUCT_ATTR_DS.dat	W_RTL_PRODUCT_ATTR_DS_COPY_JOB W_RTL_PRODUCT_ATTR_DS_STG_JOB	W_RTL_PRODUCT_ATTR_D_JOB

Table 5-1 (Cont.) DAT File Process Flows

Input Files (DAT)	DAT_REPROCESS_ADHOC	RI_DIM_INITIAL_ADHOC
W_RTL_PRODUCT_ATTR_DS_TL.dat	W_RTL_PRODUCT_ATTR_DS_TL_COPY_JOB W_RTL_PRODUCT_ATTR_DS_TL_STG_JOB	W_RTL_PRODUCT_ATTR_D_JOB
W_RTL_PRODUCT_ATTR_IMG_DS.dat	W_RTL_PRODUCT_ATTR_IMG_DS_COPY_JOB W_RTL_PRODUCT_ATTR_IMG_DS_STG_JOB	W_RTL_PRODUCT_ATTR_IMG_D_JOB
W_RTL_PRODUCT_BRAND_DS.dat W_RTL_PRODUCT_BRAND_DS_TL.dat	W_RTL_PRODUCT_BRAND_DS_COPY_JOB W_RTL_PRODUCT_BRAND_DS_STG_JOB W_RTL_PRODUCT_BRAND_DS_TL_COPY_JOB W_RTL_PRODUCT_BRAND_DS_TL_STG_JOB	W_RTL_PRODUCT_BRAND_D_JOB
W_RTL_PRODUCT_IMAGE_DS.dat	W_RTL_PRODUCT_IMAGE_DS_COPY_JOB W_RTL_PRODUCT_IMAGE_DS_STG_JOB	W_PRODUCT_ATTR_D_JOB (data used implicitly by dim load)
W_RTL_PROMO_DS.dat W_RTL_PROMO_DS_TL.dat	W_RTL_PROMO_DS_COPY_JOB W_RTL_PROMO_DS_STG_JOB	W_RTL_PROMO_D_TL_JOB W_PROMO_D_RTL_TMP_JOB
W_RTL_PROMO_CE_DS.dat	W_RTL_PROMO_CE_DS_COPY_JOB W_RTL_PROMO_CE_DS_STG_JOB	W_RTL_PROMO_D_TL_JOB W_PROMO_D_RTL_TMP_JOB
W_RTL_PROMO_CE_IT_LC_DS.dat	W_RTL_PROMO_CE_IT_LC_DS_COPY_JOB W_RTL_PROMO_CE_IT_LC_DS_STG_JOB	W_RTL_PROMO_IT_LC_D_JOB
W_RTL_PROMO_EXT_DS.dat	W_RTL_PROMO_EXT_DS_COPY_JOB W_RTL_PROMO_EXT_DS_STG_JOB	W_RTL_PROMO_D_TL_JOB W_PROMO_D_RTL_TMP_JOB
W_RTL_PROMO_IT_LC_DS.dat	W_RTL_PROMO_IT_LC_DS_COPY_JOB W_RTL_PROMO_IT_LC_DS_STG_JOB	W_RTL_PROMO_IT_LC_D_JOB
W_RTL_REPL_DAY_DS.dat	W_RTL_REPL_DAY_DS_COPY_JOB W_RTL_REPL_DAY_DS_STG_JOB	W_RTL_REPL_DAY_D_JOB
W_RTL_SEASON_DS.dat	W_RTL_SEASON_DS_COPY_JOB W_RTL_SEASON_DS_STG_JOB	W_RTL_SEASON_D_JOB
W_RTL_SEASON_PHASE_IT_DS.dat	W_RTL_SEASON_PHASE_IT_DS_COPY_JOB W_RTL_SEASON_PHASE_IT_DS_STG_JOB	W_RTL_SEASON_PHASE_IT_D_JOB
W_RTL_SUB_IT_LC_DS.dat	W_RTL_SUB_IT_LC_DS_COPY_JOB W_RTL_SUB_IT_LC_DS_STG_JOB	W_RTL_SUB_IT_LC_D_JOB

Table 5-1 (Cont.) DAT File Process Flows

Input Files (DAT)	DAT_REPROCESS_ADHOC	RI_DIM_INITIAL_ADHOC
W_RTL_SUPPLIER_TRAIT_DS.dat	W_RTL_SUPPLIER_TRAIT_DS_COPY_JOB W_RTL_SUPPLIER_TRAIT_DS_STG_JOB	W_RTL_SUPPLIER_TRAIT_D_JOB
W_RTL_TRADE_AREA_DS.dat	W_RTL_TRADE_AREA_DS_COPY_JOB W_RTL_TRADE_AREA_DS_STG_JOB	W_RTL_TRADE_AREA_D_JOB
W_RTL_TRADE_AREA_LOC_MTX_DS.dat	W_RTL_TRADE_AREA_LOC_MTX_DS_COPY_JOB W_RTL_TRADE_AREA_LOC_MTX_DS_STG_JOB	W_RTL_TRADE_AREA_LOC_MTX_D_JOB
W_RTL_TNDR_TYPE_DS.dat	W_RTL_TNDR_TYPE_DS_COPY_JOB W_RTL_TNDR_TYPE_DS_STG_JOB	W_RTL_TNDR_TYPE_D_JOB
W_RTL_TSF_CHRG_DETAILS_DS.dat	W_RTL_TSF_CHRG_DETAILS_DS_COPY_JOB W_RTL_TSF_CHRG_DETAILS_DS_STG_JOB	W_RTL_TSF_CHRG_DETAILS_D_JOB
W_RTL_TSF_DETAILS_DS.dat	W_RTL_TSF_DETAILS_DS_COPY_JOB W_RTL_TSF_DETAILS_DS_STG_JOB	W_RTL_TSF_DETAILS_D_JOB
W_STATUS_DS.dat	W_STATUS_DS_COPY_JOB W_STATUS_DS_STG_JOB	W_STATUS_D_JOB
W_RTL_UDA_METADATA_G.dat	W_RTL_UDA_METADATA_G_COPY_JOB W_RTL_UDA_METADATA_G_STG_JOB	N/A – No additional job required

Process Flows for CSV Files

The table below shows the standalone process flows for any dimension file with a `.csv` extension. CSV files follow a different load path as they are simplified interfaces that are transformed from one input file to many output tables in the data model. The same set of jobs are also present in the Nightly batch cycles, so you can use this section as a reference for nightly batch configuration (ignoring the standalone process names). Please note that the following list of jobs should remain enabled and are usually applicable to all batch runs, so they're not included in the table.

- RI_UPDATE_TENANT_JOB (as part of HIST_ZIP_FILE_LOAD_ADHOC)
- VARIABLE_REFRESH_JOB
- ETL_REFRESH_JOB
- ETL_BUSINESS_DATE_JOB
- W_RTL_CURR_MCAL_G_JOB
- SI_W_DOMAIN_MEMBER_DS_TL_TRUNC_JOB

The W_DOMAIN_MEMBER_DS_TL table works differently from other loaders, as multiple jobs are inserting into the same staging area for different sets of records. The job above is needed at the start of a process flow to truncate the W_DOMAIN_MEMBER_DS_TL table before inserting new records in all later steps in RI_DIM_INITIAL_ADHOC. If you are loading files one at a time, make sure you do not truncate W_DOMAIN_MEMBER_DS_TL excessively. It is only needed at the beginning of a new set of file loads or when starting over after an initial load was done.

Another important note is that you will want to load files in a certain order or together as sets, depending on the data you have available. If possible, you should load all your files as a set, once they become available, rather than reloading one by one every time.

Table 5-2 CSV File Process Flows

Input Files (CSV)	RI_DIM_INITIAL_ADHOC (Step 1)	RI_DIM_INITIAL_ADHOC (Step 2)	RI_DIM_INITIAL_ADHOC (Step 3)
ATTR.csv PROD_ATTR.csv	COPY_SI_ATTR_JOB STG_SI_ATTR_JOB COPY_SI_PROD_ATTR_JOB STG_SI_PROD_ATTR_JOB	SI_W_RTL_PRODUCT_ATTR_DS_JOB SI_W_RTL_PRODUCT_ATTR_DS_TL_JOB SI_W_RTL_ITEM_GRP1_DS_JOB STAGING_SI_W_RTL_PRODATTR_ITEM_GRP1_DS_JOB SI_ATTR_W_DOMAIN_MEMBER_DS_TL_JOB SI_PROD_ATTR_W_DOMAIN_MEMBER_DS_TL_JOB SI_W_RTL_PRODUCT_COLOR_DS_JOB	W_RTL_PRODUCT_ATTR_D_JOB W_RTL_ITEM_GRP1_D_ITEMBRAND_JOB W_RTL_ITEM_GRP1_D_ITEMCOLOR_JOB W_RTL_ITEM_GRP1_D_ITEMDIFF_JOB W_RTL_ITEM_GRP1_D_ITEMFAB_JOB W_RTL_ITEM_GRP1_D_ITEMLIST_JOB W_RTL_ITEM_GRP1_D_ITEMLV_JOB W_RTL_ITEM_GRP1_D_ITEMSCEN_JOB W_RTL_ITEM_GRP1_D_ITEMSIZE_JOB W_RTL_ITEM_GRP1_D_ITEMSTYLE_JOB W_RTL_ITEM_GRP1_D_ITEMUDA_JOB W_DOMAIN_MEMBER_LKP_TL_JOB W_RTL_PRODUCT_ATTR_UDA_D_JOB
CODES.csv	COPY_SI_CODES_JOB STG_SI_CODES_JOB	STAGING_SI_W_RTL_CODE_DS_JOB	W_RTL_CODE_D_JOB
DIFF_GROUP.csv	COPY_SI_DIFF_GROUP_JOB STG_SI_DIFF_GROUP_JOB	SI_W_RTL_DIFF_GRP_DS_JOB SI_W_RTL_DIFF_GRP_DS_TL_JOB	W_RTL_DIFF_GRP_D_JOB
EMPLOYEE.csv	COPY_SI_EMPLOYEE_JOB STG_SI_EMPLOYEE_JOB	SI_W_EMPLOYEE_DS_JOB	W_EMPLOYEE_D_JOB

Table 5-2 (Cont.) CSV File Process Flows

Input Files (CSV)	RI_DIM_INITIAL_ADHOC (Step 1)	RI_DIM_INITIAL_ADHOC (Step 2)	RI_DIM_INITIAL_ADHOC (Step 3)
EXCH_RATE.csv	COPY_SI_EXCH_RATE_JOB STG_SI_EXCH_RATE_JOB	SI_W_EXCH_RATE_GS_JOB	W_EXCH_RATE_G_JOB
ORGANIZATION.csv	COPY_SI_ORGANIZATION_JOB STG_SI_ORGANIZATION_JOB	SI_W_INT_ORG_DHS_JOB SI_W_INT_ORG_ATTR_DS_JOB SI_W_INT_ORG_DS_JOB SI_W_INT_ORG_DS_TL_JOB SI_W_RTL_CHANNEL_DS_JOB SI_ORG_W_DOMAIN_MEMBER_DS_TL_JOB SI_W_RTL_CHANNEL_CNTRY_DS_JOB DIM_ORG_VALIDATOR_JOB	W_INT_ORG_DH_TYPE1_JOB W_INT_ORG_D_TYPE1_JOB W_INT_ORG_DH_RTL_TMP_JOB W_RTL_ORG_RECLASS_TMP_JOB W_RTL_CHANNEL_D_JOB W_RTL_CHANNEL_CNTRY_D_JOB W_DOMAIN_MEMBER_LKP_TL_JOB
ORGANIZATION_ALT.csv	COPY_SI_ORGANIZATION_ALT_JOB STG_SI_ORGANIZATION_ALT_JOB	SI_W_ORGANIZATION_FLEX_DS_JOB	W_ORGANIZATION_FLEX_D_JOB
PROD_LOC_ATTR.csv	COPY_SI_PROD_LOC_ATTR_JOB STG_SI_PROD_LOC_ATTR_JOB	SI_W_RTL_IT_LC_DS_JOB	W_RTL_IT_LC_D_JOB
PROD_LOC_REPL.csv	COPY_SI_PROD_LOC_REPL_JOB STG_SI_PROD_LOC_REPL_JOB	STAGING_SI_W_INVENTORY_PRODUCT_ATTR_DS_JOB	W_INVENTORY_PRODUCT_D_JOB
PROD_PACK.csv	COPY_SI_PROD_PACK_JOB STG_SI_PROD_PACK_JOB	STAGING_SI_W_RTL_ITEM_GRP2_DS_JOB	W_RTL_ITEM_GRP2_D_JOB
PROD_SEASON.csv SEASON.csv	COPY_SI_SEASON_JOB STG_SI_SEASON_JOB COPY_SI_PROD_SEASON_JOB STG_SI_PROD_SEASON_JOB	STAGING_SI_W_RTL_PHASE_DS_JOB STAGING_SI_W_RTL_SEASON_DS_JOB STAGING_SI_W_RTL_SEASON_PHASE_IT_DS_JOB SI_SEASON_W_DOMAIN_MEMBER_DS_TL_JOB	W_RTL_SEASON_D_JOB W_RTL_PHASE_D_JOB W_RTL_SEASON_PHASE_IT_D_JOB W_DOMAIN_MEMBER_LKP_TL_JOB

Table 5-2 (Cont.) CSV File Process Flows

Input Files (CSV)	RI_DIM_INITIAL_ADHOC (Step 1)	RI_DIM_INITIAL_ADHOC (Step 2)	RI_DIM_INITIAL_ADHOC (Step 3)
PRODUCT.csv	COPY_SI_PRODUCT_JOB STG_SI_PRODUCT_JOB	SI_W_PROD_CAT_DHS_JOB SI_W_PRODUCT_ATTR_DS_JOB SI_W_PRODUCT_DS_JOB SI_W_PRODUCT_DS_TL_JOB SI_W_RTL_IT_SUPPLIER_DS_JOB SI_W_RTL_PRODUCT_ATTR_IMG_DS_JOB SI_W_RTL_PRODUCT_BRAND_DS_JOB SI_W_RTL_PRODUCT_BRAND_DS_TL_JOB SI_PROD_W_DOMAIN_MEMBER_DS_TL_JOB DIM_PROD_VALIDATOR_JOB	W_PROD_CAT_DH_TYPE1_JOB W_RTL_PROD_HIER_ATTR_LKP_DH_JOB W_PROD_CAT_DH_SC_RTL_TMP_JOB W_RTL_PROD_HIER_ATTR_LKP_DH_IM_JOB W_PRODUCT_D_TYPE1_JOB W_PRODUCT_ATTR_D_JOB W_RTL_PROD_RECLASS_TMP_INITIAL_JOB W_PRODUCT_D_RTL_TMP_JOB W_RTL_PROD_RECLASS_TMP_JOB W_RTL_IT_SUPPLIER_D_JOB W_RTL_PRODUCT_BRAND_D_JOB W_DOMAIN_MEMBER_LKP_TL_JOB W_RTL_PRODUCT_ATTR_UDA_D_JOB
PRODUCT_ALT.csv	COPY_SI_PRODUCT_ALT_JOB STG_SI_PRODUCT_ALT_JOB	SI_W_PRODUCT_FLEX_DS_JOB	W_PRODUCT_FLEX_D_JOB
PROMOTION.csv	COPY_SI_PROMO_EXT_JOB STG_SI_PROMO_EXT_JOB	SI_W_RTL_PROMO_EXT_DS_JOB	W_RTL_PROMO_D_TL_JOB W_PROMO_D_RTL_TMP_JOB
PROMO_DETAIL.csv	COPY_SI_PROMO_DETAIL_JOB STG_SI_PROMO_DETAIL_JOB	SI_W_RTL_PROMO_IT_LC_DS_JOB	W_RTL_PROMO_IT_LC_D_JOB
REPL_DISTRO.csv	W_RTL_REPL_DISTRO_IT_LC_DS_COPY_JOB W_RTL_REPL_DISTRO_IT_LC_DS_STG_JOB	N/A – No SI jobs used	W_RTL_REPL_DISTRO_IT_LC_D_JOB
REPL_REV_INT.csv	W_RTL_REPL_REV_INT_IT_LC_DS_COPY_JOB W_RTL_REPL_REV_INT_IT_LC_DS_STG_JOB	N/A – No SI jobs used	W_RTL_REPL_REV_INT_IT_LC_D_JOB

Table 5-2 (Cont.) CSV File Process Flows

Input Files (CSV)	RI_DIM_INITIAL_ADHOC (Step 1)	RI_DIM_INITIAL_ADHOC (Step 2)	RI_DIM_INITIAL_ADHOC (Step 3)
REPL_LT_INT.csv	W_RTL_REPL_LT_INT_IT_LC_DS_COPY_JOB W_RTL_REPL_LT_INT_IT_LC_DS_STG_JOB	N/A – No SI jobs used	W_RTL_REPL_LT_INT_IT_LC_D_JOB
REPL_ROUND.csv	W_RTL_REPL_ROUND_IT_LC_DS_COPY_JOB W_RTL_REPL_ROUND_IT_LC_DS_STG_JOB	N/A – No SI jobs used	W_RTL_REPL_ROUND_IT_LC_D_JOB
REPL_MULT_SUP.csv	W_RTL_REPL_MULT_SUP_IT_LC_DS_COPY_JOB W_RTL_REPL_MULT_SUP_IT_LC_DS_STG_JOB	N/A – No SI jobs used	W_RTL_REPL_MULT_SUP_IT_LC_D_JOB
REPL_MULT_INT.csv	W_RTL_REPL_MULT_INT_IT_LC_DS_COPY_JOB W_RTL_REPL_MULT_INT_IT_LC_DS_STG_JOB	N/A – No SI jobs used	W_RTL_REPL_MULT_INT_IT_LC_D_JOB
REPL_REV_SUP.csv	W_RTL_REPL_REV_SUP_IT_LC_DS_COPY_JOB W_RTL_REPL_REV_SUP_IT_LC_DS_STG_JOB	N/A – No SI jobs used	W_RTL_REPL_REV_SUP_IT_LC_D_JOB
REPL_LT_SUP.csv	W_RTL_REPL_LT_SUP_IT_LC_DS_COPY_JOB W_RTL_REPL_LT_SUP_IT_LC_DS_STG_JOB	N/A – No SI jobs used	W_RTL_REPL_LT_SUP_IT_LC_D_JOB
REPL_PROC.csv	W_RTL_REPL_PROC_IT_LC_DS_COPY_JOB W_RTL_REPL_PROC_IT_LC_DS_STG_JOB	N/A – No SI jobs used	W_RTL_REPL_PROC_IT_LC_D_JOB
STORE_COMP.csv	COPY_SI_STORE_COMP_JOB STG_SI_STORE_COMP_JOB	STAGING_SI_W_RTL_LOC_COMP_MTX_DS_JOB	W_RTL_LOC_COMP_MTX_D_JOB
SUPPLIER.csv	COPY_SI_SUPPLIER_JOB STG_SI_SUPPLIER_JOB	SI_W_PARTY_ORG_DS_JOB STAGING_SI_W_PARTY_ATTR_DS_JOB	W_PARTY_ATTR_D_JOB W_PARTY_ORG_D_JOB

The table below provides the process flow for fact files in CSV format (applicable in both standalone and nightly cycles).

Input Files (CSV)	File Load	Staging Load	Target Load
ADJUSTMENT.csv	COPY_SI_ADJUSTMENT_JOB STG_SI_ADJUSTMENT_JOB	SI_W_RTL_INVADJ_IT_LC_DY_FS_JOB SI_INVADJ_W_DOMAIN_MEMBER_DS_TL_JOB	W_RTL_INVADJ_IT_LC_DY_F_JOB W_DOMAIN_MEMBER_LKP_TL_JOB

Input Files (CSV)	File Load	Staging Load	Target Load
COST.csv	COPY_SI_COST_JOB STG_SI_COST_JOB	SI_W_RTL_BCOST_IT_LC_DY_FS_JOB SI_W_RTL_NCOST_IT_LC_DY_FS_JOB	W_RTL_BCOST_IT_LC_DY_F_JOB W_RTL_NCOST_IT_LC_DY_F_JOB
DEAL_INCOME.csv	COPY_SI_DEAL_INCOME_JOB STG_SI_DEAL_INCOME_JOB	SI_W_RTL_DEALINC_IT_LC_DY_FS_JOB	W_RTL_DEALINC_IT_LC_DY_F_JOB
IC_MARGIN.csv	COPY_SI_IC_MARGIN_JOB STG_SI_IC_MARGIN_JOB	STAGING_SI_W_RTL_ICM_IT_LC_DY_FS_JOB	W_RTL_ICM_IT_LC_DY_F_JOB
INV_RECLASS.csv	COPY_SI_INV_RECLASS_JOB STG_SI_INV_RECLASS_JOB	STAGING_SI_W_RTL_INVRECLASS_IT_LC_DY_FS_JOB	W_RTL_INVRECLASS_IT_LC_DY_F_JOB
INVENTORY.csv	COPY_SI_INVENTORY_JOB STG_SI_INVENTORY_JOB	SI_W_RTL_INV_IT_LC_DY_FS_JOB	W_RTL_INV_IT_LC_DY_F_JOB
INVENTORY_PACK.csv	COPY_SI_INVENTORY_PACK _JOB STG_SI_INVENTORY_PACK _JOB	SI_W_RTL_INVPK_IT_LC_DY_FS_JOB	W_RTL_INVPK_IT_LC_DY_F_JOB
MARKDOWN.csv	COPY_SI_MARKDOWN_JOB STG_SI_MARKDOWN_JOB	SI_W_RTL_MKDN_IT_LC_DY_FS_JOB	W_RTL_MKDN_IT_LC_DY_F_JOB
ORDER_DETAIL.csv	COPY_SI_ORDER_DETAIL_JOB STG_SI_ORDER_DETAIL_JOB	SI_W_RTL_PO_ONORD_IT_LC_DY_FS_JOB	W_RTL_PO_ONORD_IT_LC_DY_F_JOB
PRICE.csv	COPY_SI_PRICE_JOB STG_SI_PRICE_JOB	SI_W_RTL_PRICE_IT_LC_DY_FS_JOB	W_RTL_PRICE_IT_LC_DY_F_JOB
RECEIPT.csv	COPY_SI_RECEIPT_JOB STG_SI_RECEIPT_JOB	SI_W_RTL_INVRC_IT_LC_DY_FS_JOB	W_RTL_INVRC_IT_LC_DY_F_JOB
RTV.csv	COPY_SI_RTV_JOB STG_SI_RTV_JOB	SI_W_RTL_INVRTV_IT_LC_DY_FS_JOB	W_RTL_INVRTV_IT_LC_DY_F_JOB
SALES.csv	COPY_SI_SALES_JOB STG_SI_SALES_JOB	SI_W_RTL_SLS_TRX_IT_LC_DY_FS_JOB	W_RTL_SLS_TRX_IT_LC_DY_F_JOB
SALES_PACK.csv	COPY_SI_SALES_PACK_JOB STG_SI_SALES_PACK_JOB	SI_W_RTL_SLSPK_IT_LC_DY_FS_JOB	W_RTL_SLSPK_IT_LC_DY_F_JOB
SALES_WF.csv	COPY_SI_SALES_WF_JOB STG_SI_SALES_WF_JOB	STAGING_SI_W_RTL_SLSWF_IT_LC_DY_FS_JOB	W_RTL_SLSWF_IT_LC_DY_F_JOB
SHIPMENT_DETAIL.csv	COPY_SI_SHIPMENT_DETAIL_JOB STG_SI_SHIPMENT_DETAIL_JOB	SI_W_RTL_SHIP_IT_LC_DY_FS_JOB	W_RTL_SHIP_IT_LC_DY_F_JOB

Input Files (CSV)	File Load	Staging Load	Target Load
TRANSFER.csv	COPY_SI_TRANSFER_JOB STG_SI_TRANSFER_JOB	SI_W_RTL_INVTSF_IT_LC_DY_FS_JOB	W_RTL_INVTSF_IT_LC_DY_F_JOB

6

Data Validation Framework

The foundation file interfaces (such as product and organization hierarchies) have a set of validations and error checking jobs that execute with them to ensure the data is accurate, complete, and follows all basic requirements for RAP application usage. Review the contents of this chapter to understand what validations exist and how to reconfigure them per your implementation needs.

Architecture Overview

The validation framework consists of POM batch jobs that execute the validations, and database tables that control the types of validation rules and what happens when the rule is triggered. Some validation rules may cause the POM job to fail, which means the data has a critical issue that needs to be corrected before the batch process can continue. Other rules will simply write warnings to the database but allow the batch to proceed. In both cases, there are tables that can be queried to check the validation results and determine what actions need to be taken.

The table below summarizes the POM jobs that execute the validations:

Table 6-1 Data Validation POM Jobs

Job Name	Summary
DIM_ORG_VALIDATOR_JOB	Executes validations on the Organization Hierarchy data
DIM_PROD_VALIDATOR_JOB	Executes validations on the Product Hierarchy and Product dimension data
DIM_CALENDAR_VALIDATOR_JOB	Executes validations on the Calendar Hierarchy staging data
DIM_CALENDAR_LOAD_VALIDATOR_JOB	Executes validations on the Calendar Period data after the load has been run
DIM_EXCH_RATE_VALIDATOR_JOB	Executes validations on the Exchange Rate staging data
DIM_PROD_ATTR_VALIDATOR_JOB	Executes validations on the Product Attribute staging data
FACT_POSFACT_VALIDATOR_JOB	Executes validations on the positional fact staging data (inventory, POs, and so on). Refer to the POS DATA set of validation rules, such as `POSINVATA_R1` later in this section, for details on what this job is looking for.

The jobs are included both in the nightly batch process flow and in separate ad hoc processes that can be executed as part of your historical data loads.

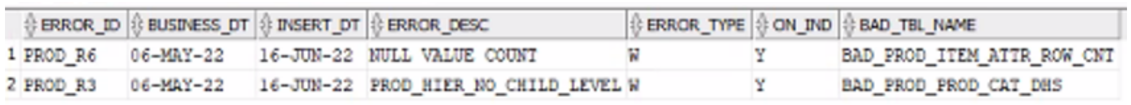
The configuration tables for the validation rules are called `C_DIM_RULE_LIST` and `C_FACT_RULE_LIST`. You can access these tables from the Control & Tactical Center's Manage System Configurations screen. The tables allows you to edit the following fields:

- Set the error message resulting from a validation rule (`ERROR_DESC`)

- Set whether the POM job should have a hard failure or only capture a warning message (`ERROR_TYPE`) with a value of `F` or `W`
- Set whether it is turned on or off (`ON_IND`) with a value of `Y` or `N`

The other important field in this table is the `BAD_TBL_NAME`, which tells you where the results of the validations will be written in the case of any errors or warnings. If a failure or warning does occur, you can directly query the database table listed in `BAD_TBL_NAME` using Data Visualizer or APEX.

Any time you execute one or more of the dimension validation jobs, there is also a database view that summarizes the results from the job executions. This view is `RI_DIM_VALIDATION_V` and can also be queried from DV as needed. An example of the data in this view is shown below:



	ERROR_ID	BUSINESS_DT	INSERT_DT	ERROR_DESC	ERROR_TYPE	ON_IND	BAD_TBL_NAME
1	PROD_R6	06-MAY-22	16-JUN-22	NULL VALUE COUNT	W	Y	BAD_PROD_ITEM_ATTR_ROW_CNT
2	PROD_R3	06-MAY-22	16-JUN-22	PROD_HIER_NO_CHILD_LEVEL	W	Y	BAD_PROD_PROD_CAT_DHS

Using a combination of the data in `RI_DIM_VALIDATION_V` and the specified `BAD_TBL_NAME` table data, you will be able to identify the issues and take corrective action on the source data. In the case of job failures, you will need to reload the data file to proceed. It is also possible to skip the failed validation job in POM, but this should only be done if you have carefully reviewed the validation results and are confident the data will not cause any problems in your target applications.

Resolving Validation Issues

The validation rules scan your input data for a variety of common problems that may result in failures or inconsistencies in downstream applications such as AI Foundation or Planning modules. The table below describes what the rules are checking for and how to resolve the issues.

Table 6-2 Validation Rule Details

Rule ID	Explanation	Resolution
CAL_R1	The `W_MCAL_PERIOD_D` table does not contain any data after loading a calendar file. Your calendar file may have format or data issues that require correction, such as an incorrect value for `MCAL_CAL_ID` or missing dates that prevent it from loading properly.	Reload a corrected data file after reviewing the contents. All start/end date fields must be populated and all other fields should exactly match the file requirements as documented.
CAL_R2	The start and end dates for the fiscal periods, quarters, or years are overlapping, which will result in an invalid calendar.	Create and load a new calendar file where the period/quarter/year start and end dates are exactly aligned and don't overlap or have gaps. Ensure all periods in one quarter/year have the same dates for those columns.

Table 6-2 (Cont.) Validation Rule Details

Rule ID	Explanation	Resolution
CAL_R3	The `START_DT` parameter set on `C_ODI_PARAM_VW` is not less than or equal to your first calendar period start date. This may result in missing calendar data.	Update the `START_DT` parameter from the Control Center to be earlier than your fiscal calendar start date. Detailed explanation of the start and end dates is provided in Chapter 2 of the *RAP Implementation Guide*.
CAL_R4	Your calendar file does not contain at least 2 years prior to the current system date. Many applications on the platform require at least 2 years before and after the current calendar year (5 years total).	Load a new calendar file having at least 2 years of fiscal periods prior to the current year.
PROD_STG_R1	Null descriptions were detected for attributes having a non-null ID column for one of the following: `BRAND_DESC`, `SUPPLIER_DESC`	Populate valid descriptions for all rows in the product data where `BRAND_NAME` or `SUPPLIER_NUM` are populated, and then reload the file.
PROD_R1	Many-to-many relationships exist in your product hierarchy, which is not allowed. This is generally due to the same child ID appearing below multiple parent IDs.	Review all hierarchy levels for instances of the same ID appearing under multiple parents (such as a department belonging to two different divisions or groups) and modify the data to remove the multi-parent issues.
PROD_R2	The same product hierarchy node has multiple descriptions on different rows of the input file.	Modify your product hierarchy file such that any given hierarchy ID has the same description on all rows.
PROD_R3	A node of the product hierarchy has no children under it. This could be due to a reclass that didn't delete the old nodes, or when a new node is added but no items were created yet. Initially, it is set as a Failure (`F`) rule, but you may change it to a warning (`W`) in `C_DIM_RULE_LIST` if you are okay with the data in its current format.	If possible, remove all cases of nodes having no children (for example, if all items are reclassed out of a subclass, delete the old subclass). Some AI Foundation functionality may fail if you attempt to run it on empty nodes; for example, if a user attempts to run CDT on an empty subclass, it cannot complete the run because there will be no data found. MFCS may allow hierarchies to exist without items, but it can lead to user confusion in AIF, so it's best to clean up this data regularly.
PROD_R4	Your product hierarchy levels use alphanumeric characters for the level IDs. This is not allowed if you are implementing Retail Insights; all levels must be numbers.	If you are implementing RI, you must alter your hierarchy to only use numbers for every level above item. Other characters are not allowed.
PROD_R5	You are attempting to delete an item while also sending data for that item in other files on the same batch run. You cannot delete an item if it is still actively sending data on other input interfaces.	Re-send the deleted item file, removing any items that are still active or posting new data to RI. If the item should be deleted, then re-send the other files having that item's data to remove the item from all other files.

Table 6-2 (Cont.) Validation Rule Details

Rule ID	Explanation	Resolution
PROD_R6	This is an informational message and not a hard failure. Null values are being checked for product attribute columns that are critical to the operation of multiple RAP applications. The warning message columns map to the item level (ATTR11), tran level (ATTR12), diff aggregate (ATTR16), and item/parent/grandparent (ATTR13,14,15) fields in the `PRODUCT.csv` file. (which loads into the `W_PRODUCT_ATTR_DS` staging table)	Null values in the `ATTR11` and `ATTR12` counts means that the `ITEM_LEVEL` and `TRAN_LEVEL` input fields are not populated, and these must be populated for 100% of items at all levels. `ATTR13/14/15` represent the item hierarchy of multi-level items and some null values are normal. The only check to perform is whether 100% of rows have nulls; if there are, then no parent/child relationship was set up for any items. `ATTR16` matches with the `DIFF_AGGREGATE` input field (or `DIFF` columns in MFCS) and it must have non-null values for all fashion items (for example, the color IDs for SKUs within a style). MFCS allows you to configure fashion items to not have a diff aggregate, but it is invalid for RAP applications that require diff aggregates to function.
PROD_R7	Invalid hierarchy relationships exist for two or more SKUs having the same item-parents or grandparents but different hierarchy levels. This will break downstream integrations with AIF and Planning.	Correct the hierarchy levels so that all SKUs having the same item-parents also have the same subclass and above hierarchy levels. The `BAD_PRODUCT_DS` table shows multiple messages for this validation depending on where the issue is, such as `PRODUCT L3 AND ITEMPARENT L2 SUBCLASS MISMATCH` for item level 2 and 3 items not having the same subclasses.
PROD_R8	You have more than 1 top level (company) ID, which is not allowed.	Correct the `TOP_PRODCAT_ID` to contain only one value on all rows.
PROD_R9	Your input file contains a different top level (company) ID than what is already in the database.	Correct the `TOP_PRODCAT_ID` to match the company ID already in the system, or erase the data in the system to perform a clean load of new hierarchy data.
PROD_R10	Your input file contains a different top level (company) domain member ID (on `W_DOMAIN_MEMBER_DS_TL`) than what is already in the database.	Correct the `TOP_PRODCAT_ID` to match the company ID already in the system, or erase the data in the system to perform a clean load of new hierarchy data.
PROD_R11	You have more than 1 top level (company) description, which is not allowed.	Correct the `TOP_PRODCAT_DESC` to contain only one value on all rows.
PROD_R12	You have duplicate item IDs when ignoring the case, such as `XYZ` and `xyz`, which is not allowed on the platform. Item IDs are not case-sensitive.	Delete duplicate items such that you only have a single ID across all lowercase/uppercase variations.

Table 6-2 (Cont.) Validation Rule Details

Rule ID	Explanation	Resolution
PROD_R13	You have a one-to-many relationship between the display IDs for a hierarchy level and their unique (UID) identifiers, such as `LVL4_PRODCAT_UID` and `LVL4_PRODCAT_ID`.	For a given value in the unique (UID) columns, you must have one and only one value in the associated display ID column. Correct either the UID or ID columns to ensure they align.
PROD_R14	Invalid hierarchy relationships exist for two or more items having the same item-parents or grandparents but different subclass levels between your incoming product data and the existing data already in `W_PRODUCT_D` and `W_PRODUCT_ATTR_D`. The `BAD_PRODUCT_DS` table shows multiple records for this validation depending on the items involved, using the error message: `EXISTING PRODUCT AND ITEMGRANDPARENT/ ITEMPARENT/CHILD SUBCLASS MISMATCH`.	Correct the hierarchy levels for the incoming items so that all items having the same item-parents also have the same subclass and above hierarchy levels. This rule is for comparing incoming item data and items already in the database from prior loads; you must ensure consistent parents and hierarchy levels across all of them. There is also a supplemental view `W_PRODUCT_ATTR_D_V` you may query to check the item and hierarchy relationships for the mismatching items.
ORG_STG_R1	Null descriptions were detected for attributes having a non-null ID column, for one of the following: `CHANNEL_NAME`, `PLANNING_CHANNEL_NAME`, `STORE_FORMAT_DESC`	Populate valid values for all descriptions where `CHANNEL_ID`, `PLANNING_CHANNEL_ID` or `STORE_FORMAT_ID` has a non-null value, and then reload the file.
ORG_R1	Many-to-many relationships exist in your organization hierarchy, which is not allowed. This is generally due to the same child ID appearing below multiple parent IDs.	Review all hierarchy levels for instances of the same ID appearing under multiple parents (such as a district belonging to two different regions or areas) and modify the data to remove the multi-parent issues.
ORG_R2	The same organization hierarchy node has multiple descriptions on different rows of the input file.	Modify your organization hierarchy file such that any given hierarchy ID has the same description on all rows.
ORG_R3	A node of the organization hierarchy has no children under it. This could be due to a reclass that didn’t delete the old nodes, or when a new node is added but no stores were created yet.	If possible, remove all cases of nodes having no children (for example, if all stores are reclassed out of a district, delete the old district). Some AI Foundation functionality will fail if you attempt to run it on empty nodes.
ORG_R4	Your organization hierarchy levels use alphanumeric characters for the level IDs. This is not allowed if you are implementing Retail Insights; all levels must be numbers.	If you are implementing RI, you must alter your hierarchy to only use numbers for every level of the organization hierarchy. Other characters are not allowed.
ORG_R5	Your location type code (`ORG_TYPE_CODE`) contains invalid values.	Only specific codes `S`, `W`, or `E` are allowed in the `ORG_TYPE_CODE` field, so you must correct any other values and reload the file.

Table 6-2 (Cont.) Validation Rule Details

Rule ID	Explanation	Resolution
ORG_R6	You have more than 1 top level (company) ID, which is not allowed.	Correct the `ORG_TOP_NUM` to contain only one value on all rows of your input file and reload it.
ORG_R7	Your input file contains a different top level (company) ID than what is already in the database, which is not allowed.	Correct the `ORG_TOP_NUM` to match the company ID already in the database (`W_INT_ORG_DH` table) or erase the data in the system to perform a clean load of new hierarchy data, if you do not want to keep your existing dataset.
ORG_R8	Your input file contains a different top level (company) domain member ID or description (on `W_DOMAIN_MEMBER_DS_TL`) than what is already in the database.	Correct the `ORG_TOP_NUM` and `ORG_TOP_DESC` to match the company data already in the database (`W_DOMAIN_MEMBER_LKP_TL` table) or erase the data in the system to perform a clean load of new hierarchy data, if you do not want to keep your existing dataset.
ORG_R9	You have more than 1 top level (company) description, which is not allowed.	Correct the `ORG_TOP_DESC` to contain only one value on all rows of your input file and reload it.
ORG_R10	You are attempting to load a location in `W_INT_ORG_DS` which is already in the data warehouse and flagged inactive or deleted.	You may not reload or reuse location IDs that are already in the data warehouse and have been deleted or inactivated. Delete the location from your input data that is specified in `BAD_ORG_INT_ORG_DS`. If you believe this validation is incorrect you may raise an SR for assistance.
ORG_R11	You are attempting to load a location in `W_INT_ORG_DHS` which is already in the data warehouse and flagged inactive or deleted.	You may not reload or reuse location IDs that are already in the data warehouse and have been deleted or inactivated. Delete the location from your input data that is specified in `BAD_ORG_INT_ORG_DHS`. If you believe this validation is incorrect you may raise an SR for assistance.
ORG_R12	You are attempting to run `W_INT_ORG_DH_JOB` but the staging table `W_INT_ORG_DHS` is empty. The location hierarchy is a full snapshot dimension load and must always have data.	Populate the location hierarchy data into `W_INT_ORG_DHS` table and then re-run the validation job.
EXCH_RATE_R1	Exchange rate dates are overlapping for the same conversion, which will result in multiple rates active for the same date and currency.	Modify the start/end dates for the exchange rate records to ensure there are no overlapping dates. Only one rate may be effective per day/currency combination.

Table 6-2 (Cont.) Validation Rule Details

Rule ID	Explanation	Resolution
EXCH_RATE_R2	Exchange rate dates have gaps which will result in no rate being active for one or more dates.	Modify the start/end dates for the exchange rate records to ensure they have no gaps between one end date and the next start date, for any given currency rate.
EXCH_RATE_R3	You have provided currency conversion in one direction (for example, USD > CAD) but you did not provide it in the alternate direction (CAD > USD).	The system requires that you provide currency rates going in both directions for each currency code pair, to ensure we are always able to convert into and out of any supported currency.
ATTR_R1	There is mismatched data between the `ATTR.csv` and `PROD_ATTR.csv` files	The `ATTR.csv` file must have a header record for all attribute groups and values found in `PROD_ATTR.csv`. Correct the `ATTR.csv` file to match exactly with `PROD_ATTR.csv` or delete the mismatched rows from `PROD_ATTR.csv`.
ATTR_R2	`PROD_ATTR.csv` column `ATTR_GRP_TYPE` contains an invalid type code.	The only valid codes for `ATTR_GRP_TYPE` are (`ITEMDIFF`, `ITEMUDA`, `ITEMLIST`, `COLOR`, and `PRODUCT_ATTRIBUTES`). Correct the `PROD_ATTR.csv` file and reload the data.
ATTR_R3	`ATTR.csv` column `ATTR_TYPE_CODE` contains an invalid type code.	The only valid codes for `ATTR_TYPE_CODE` are (`FF`, `LV`, `DT`, `SIZE`, `FABRIC`, `SCENT`, `FLAVOR`, `STYLE`, `COLOR`, and `DIFF`). Correct the `ATTR.csv` file and reload the data.
ATTR_R4	`ATTR.csv` columns `ATTR_GROUP_ID` or `ATTR_VALUE_ID` contain invalid characters.	Attribute group and value IDs are used as a hierarchy in Planning apps and are restricted from having any spaces, colons, or quotation marks as part of the IDs. Correct the `ATTR.csv` and `PROD_ATTR.csv` files and reload the data.
POSINVDATA_R1	Dates other than the current business date were found on `W_RTL_INV_IT_LC_DY_FS`. The `DAY_DT` is being compared to "select `MCAL_NUM` from `W_RTL_CURR_MCAL_G` where `MCAL_TYPE = 'DT'`".	Daily positional fact data must only contain a `DAY_DT` value matching the current business date; correct the data and reload. This can be a sign that an incorrect file was used in the batch or the dates are out of sync.
POSINVUDATA_R2	Dates other than the current business date were found on `W_RTL_INVU_IT_LC_DY_FS`. The `DAY_DT` is being compared to "select `MCAL_NUM` from `W_RTL_CURR_MCAL_G` where `MCAL_TYPE = 'DT'`".	Daily positional fact data must only contain a `DAY_DT` value matching the current business date; correct the data and reload. This can be a sign that an incorrect file was used in the batch or the dates are out of sync.

Table 6-2 (Cont.) Validation Rule Details

Rule ID	Explanation	Resolution
POSPRICEDATA _R3	Dates other than the current business date were found on `W_RTL_PRICE_IT_LC_DY_FS`. The `DAY_DT` is being compared to “select `MCAL_NUM` from `W_RTL_CURR_MCAL_G` where `MCAL_TYPE = 'DT'`”.	Daily positional fact data must only contain a `DAY_DT` value matching the current business date; correct the data and reload. This can be a sign that an incorrect file was used in the batch or the dates are out of sync.
POSNCOSTDATA _R4	Dates other than the current business date were found on `W_RTL_NCOST_IT_LC_DY_FS`. The `DAY_DT` is being compared to “select `MCAL_NUM` from `W_RTL_CURR_MCAL_G` where `MCAL_TYPE = 'DT'`”.	Daily positional fact data must only contain a `DAY_DT` value matching the current business date; correct the data and reload. This can be a sign that an incorrect file was used in the batch or the dates are out of sync.
POSBCOSTDATA _R5	Dates other than the current business date were found on `W_RTL_BCOST_IT_LC_DY_FS`. The `DAY_DT` is being compared to “select `MCAL_NUM` from `W_RTL_CURR_MCAL_G` where `MCAL_TYPE = 'DT'`”.	Daily positional fact data must only contain a `DAY_DT` value matching the current business date; correct the data and reload. This can be a sign that an incorrect file was used in the batch or the dates are out of sync.
POSPOONORDD ATA_R6	Dates other than the current business date were found on `W_RTL_PO_ONORD_IT_LC_DY_FS`. The `DAY_DT` is being compared to “select `MCAL_NUM` from `W_RTL_CURR_MCAL_G` where `MCAL_TYPE = 'DT'`”.	Daily positional fact data must only contain a `DAY_DT` value matching the current business date; correct the data and reload. This can be a sign that an incorrect file was used in the batch or the dates are out of sync.
POSPOONALCD ATA_R7	Dates other than the current business date were found on `W_RTL_PO_ONALC_IT_LC_DY_FS`. The `DAY_DT` is being compared to “select `MCAL_NUM` from `W_RTL_CURR_MCAL_G` where `MCAL_TYPE = 'DT'`”.	Daily positional fact data must only contain a `DAY_DT` value matching the current business date; correct the data and reload. This can be a sign that an incorrect file was used in the batch or the dates are out of sync.
POSCOMPPRICE DATA_R8	Dates other than the current business date were found on `W_RTL_COMP_PRICE_IT_LC_DY_FS`. The `DAY_DT` is being compared to “select `MCAL_NUM` from `W_RTL_CURR_MCAL_G` where `MCAL_TYPE = 'DT'`”.	Daily positional fact data must only contain a `DAY_DT` value matching the current business date, correct the data and reload. This can be a sign that an incorrect file was used in the batch or the dates are out of sync.

In all cases where you need to correct your file and reload it, you are expected to push only the corrected files into the system using the data reprocessing ad hoc programs in the POM AIF DATA schedule. Refer to the AI Foundation Data Standalone Processes chapter for details on these programs:

1. `REPROCESS_ZIP_FILE_PROCESS_ADHOC` to upload your corrected file(s)
2. `CSV_REPROCESS_ADHOC` or `DAT_REPROCESS_ADHOC` to import the corrected files and transform the data from staging to target tables (only running the required programs, not the entire ad hoc process)

7 Support Utilities

Some support utilities will be exposed for implementers directly in APEX, allowing you to run functions such as database cleanup without Oracle involvement. These utilities may also be used by Oracle Support when responding to Service Requests on your RAP environments. If a process documented here is intended only for Oracle Support usage and not for customers directly, it will be noted in the detailed description.

Data Cleanup Utilities

Data Warehouse Table Cleanup

Because foundation data is always loaded first through the shared data warehouse (also called the `RADM01` schema), implementers often need to erase data from these tables in preparation for a new load. Database functions have been exposed to APEX to allow targeted deletion of data by table name. The delete functions can be disabled by Oracle upon request if you do not want the functionality exposed after customer go-live.

Note:

These utilities are only for data loaded using the AIF DATA schedule jobs in POM. If you have loaded data into one or more AIF applications (AIF APPS schedule) then there is a cleanup job named `AIF_APPS_MAINT_DATA_CLEANUP_ADHOC_PROCESS` in the AIF APPS schedule. Refer to the Data Cleanup Utility section in the AI Foundation Applications Standalone Processes chapter.

The sample command below is the basic method used for table cleanup of a specific table. Specify the schema name and table name to be truncated, then run the PL/SQL block.

```
DECLARE
  SCHEMANAME VARCHAR2(200);
  TABLENAME VARCHAR2(200);
BEGIN
    SCHEMANAME := 'RADM01';
    TABLENAME := 'W_RTL_SLS_TRX_IT_LC_DY_FS';
    RI_SUPPORT_UTIL.CLEAR_SELECTED_RI_TABLES(
        SCHEMANAME => SCHEMANAME,
        TABLENAME => TABLENAME
    );
END;
```

If the process is successful, you will see that the PL/SQL block was successfully executed with no further message or results. If the process encounters any error, it will display the error details in the results panel in APEX. Error details may also be logged in the `RI_LOG_MSG` table unless it is an unexpected failure that was not caught by the program.

To quickly clean the entire database schema instead of individual tables, you may instead call the following command. This command will erase all customer data except for the calendar, system configuration tables, and seed data records. This command will also delete user data from the `C_HIST_LOAD_STATUS` table, which was generated for any history loads. Use this command if you need to reset the environment in preparation for a new dataload using a different dataset:

```
DECLARE
SCHEMANAME VARCHAR2(200);
BEGIN
    SCHEMANAME := 'RADM01';
    RI_SUPPORT_UTIL.CLEAR_SELECTED_RI_TABLES(
        SCHEMANAME => SCHEMANAME
    );
END;
```

Calendar removal is provided as a separate function, because you cannot remove calendar information without also erasing all partitions (which are specific to your currently loaded calendar). The function name is `CLEAR_RI_MCAL_TABLES` and can be called the same way as the schema clear script above, passing in the schema name as the input. Before you perform any calendar cleanup, review the following:

- Partition removal is based on the current partition configuration in `C_MODULE_ARTIFACT`; it will not modify tables that are not enabled for partitioning. Ensure the configuration table reflects your current cleanup needs.
- Because calendar cleanup includes partition removal, you cannot use the system for a new data load without first re-partitioning the system. Refer to the *RAP Implementation Guide* for the steps to reload the calendar.

```
DECLARE
SCHEMANAME VARCHAR2(200);
BEGIN
    SCHEMANAME := 'RADM01';
    RI_SUPPORT_UTIL.CLEAR_RI_MCAL_TABLES(
        SCHEMANAME => SCHEMANAME
    );
END;
```

There is a function named `RI_SUBJECTAREA_TABLE` that erases functional areas of the data warehouse one by one, which can be useful for targeted cleanup of related groups of tables. The function uses the list of tables and subject area names from the database table `C_RI_SUBJECTAREA`, which you can query from APEX to identify which values you want to use. If you run the command for the Price or Inventory Position subject areas, then it will also clean up the `C_HIST_LOAD_STATUS` table for all related entries.

The command syntax is shown below.

```
DECLARE
  SUBJECTAREA_NAME VARCHAR2(200);
  OWNER_NAME VARCHAR2(200);
BEGIN
    OWNER_NAME := 'RADM01';
    SUBJECTAREA_NAME := 'Organization';
    RI_SUPPORT_UTIL.RI_SUBJECTAREA_TABLE(
        SUBJECTAREA_NAME => SUBJECTAREA_NAME,
        OWNER_NAME => OWNER_NAME
  );
END;
```

When you run any of the cleanup commands above, it may take an hour or longer to complete depending upon the amount of data in your schema and the number of partitions requiring deletion. It is recommended to run large cleanup activities from a SQL Script, not from the SQL Commands screen. Add the commands to a script and save it, then execute it separately. Saved scripts that are executed are allowed to run in the background, which means you can navigate away from the page as soon as you start it, and it will continue to run until completed. To monitor the activity after starting a cleanup command or script, you can query the `RI_LOG_MSG` table and check for new log messages:

```
select * from ri_log_msg order by msg_ts desc;
```

If the process is still running, you will see new log entries being added for `ri_support_util` methods. If no recent entries are added and the last set of messages show the `END` messages for a process step, then you can verify that all your tables are cleared and proceed with your implementation activities. When verifying table counts and contents after a cleanup script is run, you must include a hint in your SQL to prevent cached results or stale statistics from being returned.

For example:

```
SELECT /*+ OPT_PARAM('_optimizer_answering_query_using_stats' 'FALSE') */
COUNT(*) from w_product_d;
```

Integration Layer Table Cleanup

Tables used to move data between RAP applications are maintained in a separate database schema called RDX (short for retail data exchange). These tables are not accessible to cleanup packages like `ri_support_util`. A separate package is provided just for these tables, called `rap_support_util`. You may call this package to delete data from the RDX schema, specifically for tables that use `RUN_ID` as the primary key and partition structure, such as `W_PDS_SLS_IT_LC_WK_A`.

The `rap_support_util` package has a single method `PURGE_INTF_RUNS`. This method allows you to delete data from an RDX table fully or for specific `RUN_ID` values. The parameters for the method are:

- `p_app_code` – Required. The code of the application whose integration tables you are purging. Use `%` as a wildcard to purge data regardless of the application source. Valid codes can be found on `RAP_INTF_CFG`.
- `p_intf_name` – Required. The name of the table that needs to be purged. Use `%` as a wildcard to purge all supported tables (recommended to only use this option with a non-wildcard value on `p_app_code`).
- `p_run_id` – Optional. The `run_id` to be deleted.
- `p_max_retained_run_id` – Optional. The max `run_id` that needs to be retained in the table. Data less that this run ID will be purged.
- `p_purge_age_run_id` – Optional. Purge run IDs older than this age.

The first two parameters must always be used, and you may optionally provide one of the other three parameters to further refine the data to be deleted.

To erase all runs in a table, you only need to provide the table name. For example, to erase the `W_PDS_INVRTV_IT_LC_WK_A` table, use the command:

```
DECLARE
    TABLE_NAME VARCHAR2(200);
BEGIN
    TABLE_NAME := 'W_PDS_INVRTV_IT_LC_WK_A';
    RAP_SUPPORT_UTIL.PURGE_INTF_RUNS(p_app_code => '%', p_intf_name =>
TABLE_NAME);
END;
```

To limit the `RUN_ID` values that are purged, pick any one of the other parameters as a third input value after the table name. The run ID and max retained run ID are numbers that should contain a single `RUN_ID` from the available values in the database table. The purge age is a number representing the number of days back from sysdate to purge (any run created before `sysdate-p_purge_age_run_id` will be deleted).

Data Warehouse Partition Cleanup

Most tables in the data warehouse are partitioned using the loaded fiscal calendar, and these partitions are created either through ad hoc processes or automatically in batch. When you have major data changes, such as altering the calendar or reconfiguring the flexible fact tables, you might be required to purge existing partitions as part of data cleanup. There is a function named `REMOVE_ALL_PARTITIONS` that can purge partitions from a single table. It has only one parameter input: the name of the table to purge.

```
DECLARE
    TABLE_NAME VARCHAR2(200);
BEGIN
    TABLE_NAME := 'W_RTL_FLEXFACT1_F';
    RI_SUPPORT_UTIL.REMOVE_ALL_PARTITIONS(p_table_name => TABLE_NAME);
END;
```

Data Delete Utility

During data validations, you may encounter times when a particular subset of data is incorrect, and you wish to reload it. If you do not want to truncate the entire table, you may be able to delete just the segment of data having an issue by using this data deletion utility. This utility allows you to specify ranges of calendar, product, and location values that you need to delete and it will remove only that data, allowing you to reprocess those intersections without reloading the entire history.

Currently, the following functional modules are supported:

Name	Module Code
Sales	SLS
Sales Promotion	SLSPR
Sales Pack	SLSPK
Inventory Position	INV

Name	Module Code
Inventory Receipts	INVRC
Inventory Transfers	INVTSF
Inventory Adjustments	INVADJ
Inventory RTVs	INVRTV
Markdowns	MKDN
Wholesale/Franchise	SLSWF

The job for this utility is in the AIF DATA schedule, Standalone process section. The process name is `DATA_DELETE_PROCESS_ADHOC` and has only one job, `DATA_DELETE_JOB`. The job accepts two parameter values:

1. The first value is required and must specify the module code (for example, `SLS`)
2. The second value is optional and may be the value `ALL`. This overrides the configurations for the job and instead performs a full table truncate on the module, similar to the data cleanup utility but only for the specific table covered by this program.

The parameters are provided directly into the POM edit parameters screen or as values on the job parameters in a Postman call. If entered into POM UI, you might enter `SLS ALL` to erase the sales data, or just `SLS` to run the sales delete process for your configuration.

To configure what data will be deleted, there are two configuration tables available from the Manage System Configurations. The first table is `C_MODULE_CLNUP_TABLE`. This lists the available tables in the database to be cleaned up by the utility. You may change the column `CLNUP_IND` to be `Y` or `N`, which enables or disables that table for the cleanup functions. The second table is `C_MODULE_CLNUP_CFG`. This is where you specify the intersections of product, location, and calender to be deleted from the tables. You may insert a row to this table to set the parameters. The columns to update are listed below.

Column	Usage
MODULE_CODE	Enter the module code from the supported list of codes, like `SLS` or `INV`
PROD_LVL_NAME	Enter the product hierarchy level that your IDs will be specified for, including `ITEM`, `SBC`, `CLS`, `DEPT`, `GRP`, or `DIV`
PROD_LVL_ID_LST	Enter a comma-separated list of unique identifiers for product hierarchy nodes (equivalent to `W_PROD_CAT_DH.LEVEL_ID` or `PRODUCT.csv` values like `LVL4_PRODCAT_UID` or `ITEM`) that you want data deleted for.
PROD_LVL_ID_LST2	Optional field for more identifiers if your list exceeds the field length of the prior column.
ORG_LVL_NAME	Enter the location hierarchy level that your IDs will be specified for, including `LOCATION`, `DISTRICT`, `REGION`, `AREA`, `CHAIN`
ORG_LVL_ID_LST	Enter a comma-separated list of unique identifiers for location hierarchy nodes (equivalent to `ORGANIZATION.csv` `ORG_HIER10_NUM` or other org hierarchy values) that you want data deleted for.
ORG_LVL_ID_LST2	Optional field for more identifiers if your list exceeds the field length of the prior column.

Column	Usage
CLNUP_START_DATE	Set the start date for the data cleanup. Use a week-starting date if cleaning week-level tables. Make sure there is no timestamp component (time shows as `00:00:00`).
CLNUP_END_DATE	Set the end date for the data cleanup. Use a week-ending date if cleaning week-level tables. Make sure there is no timestamp component (time shows as `00:00:00`).
CLNUP_IND	Enter `Y` or `N` to enable or disable the configuration record.

The data delete utility will use this information in the following ways:

- When not using the `ALL` option, it truncates partition statements if no prod/loc intersections are provided, or delete statements which are limited by your product, location, and calendar values. All combinations of the specified values are deleted for the given date range.
- When using the `ALL` option, the configuration table is ignored and the tables having `CLNUP_IND=Y` will be truncated entirely.
- Week-level tables are always truncated by weekly partition or deleted by week, partial week cleanup is not possible, which is why week start/end dates should be used in the configuration.

For the hierarchy information, the tables `W_PRODUCT_D_RTL_CUR_TMP` and `W_INT_ORG_DH_RTL_CUR_TMP` are used to select the necessary intersections and keys. These should be populated already by performing hierarchy loads into the system; but if you are having any trouble with the tool deleting the necessary data, check that your hierarchies have been loaded here. Only the latest hierarchy definitions are used, so if reclassifications were performed, it will not delete data for inactive or outdated hierarchy definitions.

Innovation Workbench Process Cleanup

If a process running from Innovation Workbench needs to be terminated for any reason, it is possible to do this from Innovation Workbench itself. As a prerequisite, you will need two pieces of information:

- the account name which invoked the process/query that needs to be terminated (for example: `first.lastname@domainname.com`)
- the minimum required runtime of the database session(s) to be killed (in minutes)

After collecting this information, you will use a package called `RAP_DBADMIN_LOCAL_UTIL`, which is used for program units associated with session management. This package has a routine named `KILL_USER_RUNAWAY_SESSION` which kills long-running sessions for specified client information. It has support for two parameters to help select sessions to be killed. The first parameter (`p_client_info`) is optional and, if not provided, will default to the user who invoked the routine. The second parameter (`p_max_runtime_allowed_min`) is also optional and accommodates the number of minutes a session must be active before being selected for getting killed. This defaults to 60 minutes if not provided. To kill all IW sessions, without regard for runtime, provide a value of 0 for `p_max_runtime_allowed_min`. The routine never terminates the session that invoked the routine.

Based on the parameters provided, sessions that are ACTIVE and meet the search conditions provided will be killed using a command like the following:

```
BEGIN
rap_dbadmin_local_util.kill_user_runaway_session(:userid_to_kill, :minimum_run
time_minutes);
END;
/
```

The variables shown above are placeholders for the values to be used. The process will kill all sessions that are associated with the specified account (except for the session initiating this request), and have been active for more than the specified number of minutes. The runtime parameter allows for easier cleanup of sessions, allowing only the terminating of longer running sessions, and not more-recent sessions.

As an alternative to using this process, it is also possible to schedule your commands to run through `DBMS_SCHEDULER`. Those statements run in the background and can be monitored through the normal data dictionary views associated with `DBMS_SCHEDULER`. If a statement runs through `DBMS_SCHEDULER`, and it needs to be terminated, then instead of using this `RAP_DBADMIN_LOCAL_UTIL` package to kill the session, it is possible to use `DBMS_SCHEDULER.STOP_JOB` to stop the job that was submitted.

Aggregation Utility

The data warehouse has over 100 different tables for pre-calculating data at higher levels of aggregation, mainly for the purpose of BI reporting and analytics. These tables do not need to be populated during initial historical data loads but would be needed before end-users begin accessing data in RI. Some aggregates are also used for PDS integrations (at the item/ location/week level). To populate or update these tables after history loads are complete, an aggregation utility is provided that can use the base intersection of a functional area to calculate all of the higher-level tables.

The utility currently supports the following subject areas in RI:

Name	Module Code
Sales	SLS
Sales Pack	SLSPK
Sales Promotion	SLSPR
Sales Wholesale	SLSWF
Inventory	INV
Inventory Adjustments	INVADJ
Inventory Receipts	INVRC
Inventory Transfers	INVTSF
Inventory Reclasses	INVRECLASS
Inventory Return to Vendor	INVRTV
Markdowns	MKDN
Net Profit	NPROF
Customer Loyalty Transactions	CUST

Within these subject areas, the aggregation does have some limitations on which columns are populated (relative to nightly batches). Aggregate columns that are derived by joining multiple tables together during batch processing are not included in this utility because the data may not be available or accurate for the calculations. This includes:

- Inventory availability columns, such as the counts and amounts based on presentation stock and demo stock levels
- Inventory age and weeks-in-store calculations based on new receipt activity
- Any columns that join to the clearance dimension to get the clearance indicator and markdown event ID for a specific inventory or transaction record

Prerequisites for using the utility (all steps must be completed every time you want to use the utility):

1. Partitioning has been run for the target functional areas such as sales (`SLS`), inventory (`INV`), and so on. Follow the steps in the *RAP Implementation Guide* to perform additional partitioning as needed. If you have not used the utility since the last time you received a product patch, you should re-run the partitioning process again to ensure all tables are partitioned.
2. The base fact for the functional area has already been loaded with data for the entire date range you want to aggregate on. For example, the `W_RTL_SLS_TRX_IT_LC_DY_F` table is loaded before attempting to aggregate it to `W_RTL_SLS_IT_LC_WK_A`.
3. Database statistics have been collected recently using `REFRESH_RADM_JOB` and `ANAYLZE_TEMP_TABLES_JOB` (either as part of an ad hoc data load or automatically as part of nightly batch).

The configuration table to control the utility is `C_RI_AGGREGATION_MAP`, which is available from the Control Center in the AI Foundation user interface. It contains a list of aggregate tables in the data warehouse that can be processed by the utility. For each table you want to load, set the `START_DT` as the earliest date to process and the `END_DT` to the final date to process. The tables are grouped by functional area such as `MODULE_CODE=SLS` so you can update all tables relating to that fact.

When specifying the start/end dates, make sure to consider the calendar level of the table. Day level tables can have any start/end dates because they use daily partitions. Week level (WK) tables should use week starting/ending dates to ensure each full week of data is always aggregated into the table. Similarly, Gregorian month (GMH) tables should use month start/end dates. The utility also has functions to auto-extend your date ranges to encompass full weeks and months even if you make mistakes in the configuration. By default, dates will always be auto-extended so that full weeks/months are always loaded where needed. This can be changed using parameter `RI_AGG_FULL_LOAD_TYPE` on `C_ODI_PARAM_VW` if you only want the dates you specify to be included in the aggregations. Valid values include:

- `F` – full auto-extend of dates
- `FE` – extend end dates only
- `FS` – extend start dates only

N/A (or any other values) – Use only the dates in the mapping table

Once the necessary updates are performed, you will execute an ad hoc process in POM named `AGGREGATION_UTILITY_ADHOC`. This process is a first-time manual run to validate the configuration is working as intended and to set up the temp tables. This process has 3 jobs in it:

`AGG_UTILITY_PRE_JOB` – Calculates a temporary lookup table for product hierarchy relationships

`AGG_UTILITY_ORG_PRE_JOB` – Calculates a temporary lookup table for organization hierarchy relationships

`AGG_UTILITY_JOB` – Performs an aggregation action for a specific table name and run type

The `AGG_UTILITY_JOB` requires two input parameters: the name of the table as found in `C_RI_AGGREGATION_MAP` and the type of aggregation to perform (`FRESH` or `RESTART`). When `FRESH` is specified, it assumes you want to aggregate the entire date range specified in the configuration table, even if it has been run before. If `RESTART` is specified, it will run only from the last completed period (the partition job aggregates one quarter at a time so it will not re-run earlier quarters that already completed). Also use the `RESTART` option if you changed the `END_DT` to some time further in the future and want to only process incomplete dates resulting from the change. In most use cases you can always specify `RESTART` as the option and it will perform the required actions.

Example Postman message body for the process call:

```
{
  "cycleName":"Adhoc",
  "flowName":"Adhoc",
  "requestType":"POM Scheduler",
  "processName":"AGGREGATION_UTILITY_ADHOC",
  "requestParameters":"jobParams.AGG_UTILITY_JOB= W_RTL_SLS_CS_IT_LC_DY_A
RESTART"
}
```

Once you have issued the command to start the process, you may monitor the detailed run status by querying the table `C_BULK_LOAD_STATUS` from APEX. A record will be inserted for each calendar quarter that has been processed until the entire date range is aggregated. The POM job will complete successfully after the table is loaded for all dates. You may then compare the base fact table with the target aggregate and confirm the values have been rolled up as expected.

The aggregate tables must be populated in a specific sequence based on the value in the `AGGREGATION_LEVEL` column in `C_RI_AGGREGATION_MAP`. For each `MODULE_CODE`, the level 1 tables must be populated first, then the level 2 tables, and so on. To automate this execution sequence, there is a separate job available in POM, named `AGG_SRVC_JOB`. The aggregation service job accepts a single input parameter for the `MODULE_CODE` value. The job will execute all tables in the associated record set in `C_RI_AGGREGATION_MAP` for that module, following the `AGGREGATION_LEVEL` sequence as needed. The two `PRE` jobs (`AGG_UTILITY_PRE_JOB` and `AGG_UTILITY_ORG_PRE_JOB`) are prerequisites for this job, so ensure you've already run those at least once before using `AGG_SRVC_JOB`.

Example Postman message body for the process call:

```
{
  "cycleName":"Adhoc",
  "flowName":"Adhoc",
  "requestType":"POM Scheduler",
  "processName":"AGGREGATION_SRVC_ADHOC",
  "requestParameters":"jobParams.AGG_SRVC_JOB=INV"
}
```

If any processes in the `AGG_SRVC_JOB` have a failure or are taking too long to run, you may also check the following tables for more information:

- C_RI_SRVC_REQ_QUEUE – Contains the status of individual service calls invoked by the aggregation process. A status of 6 means success while 7 means failed.
- RI_LOG_MSG – If a process does fail, the detailed trace logs will be written to this table to help you identify the problem. Look for records where PROGRAM_UNIT = RI_AGGREGATION_UTIL.

Database Statistics Utility

A critical part of working with large datasets in Oracle Database is the collection of statistics on your database tables. The POM processes used to load data generally include a job to collect statistics on the entire database schema to ensure stats are always up-to-date. The drawback of this program is that it can take a significant amount of time to run, even if you only need to refresh statistics on a single table. To help implementers collect statistics on specific tables, a utility is provided using the POM standalone program COLLECT_STATS_JOB.

The COLLECT_STATS_JOB accepts a single input parameter for the database module code you wish to gather stats on. The module codes are defined from the configuration table C_MODULE_DBSTATS, which is available from the Control & Tactical Center in the AI Foundation UI. The configuration table will come pre-defined with some core modules that often need stats collected on them using the codes SLS, INV, and PRICE. You have the ability to insert new rows into the table to define your own **custom** values for MODULE_CODE. You may specify any value you wish for the MODULE_CODE, along with one or more tables you plan to collect stats on. You would then pass the MODULE_CODE value into the job parameters to collect stats on your chosen list of tables. TABLE_NAME and MODULE_CODE are the only required values for tables in the RADM01 schema. If you are collecting stats on a temp table (in the RABE01USER schema) then you must also populate the OWNER_TYPE as BATCH.

The C_MODULE_DBSTATS column OP_TYPE provides a way to handle various issues with locked statistics. It accepts one of the following codes:

- SKIP – If a table's statistics are locked, then skip it and continue processing
- C_LOCK – If a table's statistics are locked, unlock it, collect stats, then lock it again
- C_UNLOCK – If a table's statistics are locked, unlock it and collect stats, leaving it unlocked
- UNLOCK – If a table's statistics are locked, unlock them

After reviewing the configuration, you may invoke the job from POM or Postman, providing your MODULE_CODE as the only input parameter.

Example Postman message body for the process call:

```
{
 "cycleName":"Adhoc",
 "flowName":"Adhoc",
 "requestType":"POM Scheduler",
 "processName":"COLLECT_STATS_ADHOC",
 "requestParameters":"jobParams.COLLECT_STATS_JOB=SLS"
}
```

External Table Load Logs

The first step of importing a file into RAP applications is to map the raw file as an external table on the Oracle database. The file is then pulled from the external table into an actual staging table in the target database schema. From a batch job perspective, the external table steps are performed by the jobs having `STG` in the name, such as `W_RTL_CMP_CLOSED_DS_STG_JOB` or `STG_SI_ORGANIZATION_JOB`. Issues that occur during the external table setup and load process result in rejected records on the application server that are not immediately visible to the database, since no data is yet loaded into the system.

To access rejected records from external tables, a temporary link is created in the database that points to the log files. You must use a procedure in the `ri_support_util` package to access this data. The procedure is named `get_file_load_result` and it accepts two input parameters:

1. The log file type, using values `LOG` or `BAD`. `LOG` files are the detailed log messages, while `BAD` files are the actual rejected records from the source data.
2. The numerical sequence of the database object linked to the logs. This is obtained from the error message when a job fails in POM.

Here is an example log message you might get from a failed job in POM:

```
Status check shows failed job, due to [ORA-20003: Reject limit reached, query
table "RADM01"."COPY$124_LOG" for error details
```

The table referenced in this message is actually an external table link to a log file on the server. To access the data, log into Innovation Workbench and call the support utility with this command:

```
create table BATCH_LOG124 as select * from table
(ri_support_util.get_file_load_result('LOG', '124'));
```

Creating a table allows you to preserve the logs without re-querying the application server. If there are rejected records associated with the same load, then there will also be a `BAD` table, which can use the same command but replacing `LOG` with `BAD`. External table logs are temporary, and they will be erased frequently by automated processes. You will need to extract the relevant data from the logs the same day the job fails, or it may be deleted.

Managing Rejected Records

Data Warehouse Rejection Process

A core feature of the foundation data warehouse is the capturing and storage of rejected fact records. A rejected record is one in which the data was able to be staged into the database from a file or other integration, but there was a problem with the data that prevented it from loading into the final data warehouse table. The most common reason for rejection is that one or more of the key columns in the fact record does not have any matching value in the associated dimension tables. For example, you provide a sales transaction with `ITEM = 12340` but there is no such item as part of your product dimension, so the system is unable to load that record.

The general process for rejecting data is:

1. Records are loaded from the staging (`FS`) table to a temporary (`TMP`) table where the data is joined with internal dimensions and foreign keys are obtained.
2. Records are moved from the temporary table to the target fact (`F`) table using either an `INSERT` or `MERGE` statement, depending on the fact program.

3. The program compares the records in the `FS` and `TMP` tables and any differences are written to an error (`E$`) table for review. `E$` tables do not exist when the system is first installed; they are created dynamically at runtime.
4. A summary of the errors is written to the `W_ETL_REJECTED_RECORDS` table when the job completes. Jobs do not fail due to rejected records, they will load any valid data and end successfully.

A list of rejected record tables is provided below for reference. These tables belong to the `RABE01USER` database user, so when querying them you should append the username in front of the table name. If you are attempting to query one of these tables from APEX and you get an error that the table does not exist, then one of two things may be the reason:

- You do not have any rejections yet, so the table has not been created by the load program.
- The table has not been granted to APEX. The ability to select from `E$` tables is given by `RABE_TO_RTLWSP_GRANTS_JOB` and this job must be executed as part of any ad hoc process to refresh the table grants.

Subject Area	Rejected Records Table
Allocation Details	E$_W_RTL_ALC_IT_LC_DY_TMP
Base Cost	E$_W_RTL_BCOST_IT_LC_DY_TMP
Cluster Items	E$_W_RTL_CLSTR_GRP_IT_TMP
Competitor Price	E$_W_RTL_COMP_PRICE_IT_LC_DY_T
Customer Loyalty Awards	E$_W_RTL_CUST_LYL_AWD_TRX_DY_T
Customer Loyalty Transactions	E$_W_RTL_CUST_LYL_TRX_LC_DY_TM
Deal Income	E$_W_RTL_DEALINC_IT_LC_DY_TMP
Fact Aggregate 1	E$_W_RTL_FACT1_PROD1_LC1_T1_TMP
Fact Aggregate 2	E$_W_RTL_FACT2_PROD2_LC2_T2_TMP
Fact Aggregate 3	E$_W_RTL_FACT3_PROD3_LC3_T3_TMP
Fact Aggregate 4	E$_W_RTL_FACT4_PROD4_LC4_T4_TMP
Flex Fact 1	E$_W_RTL_FLEXFACT1_TMP
Flex Fact 2	E$_W_RTL_FLEXFACT2_TMP
Flex Fact 3	E$_W_RTL_FLEXFACT3_TMP
Flex Fact 4	E$_W_RTL_FLEXFACT4_TMP
Gift Card Sales	E$_W_RTL_GCN_TRX_LC_DY_TMP
Intercompany Margin	E$_W_RTL_ICM_IT_LC_DY_TMP
Inventory Adjustments	E$_W_RTL_INVADJ_IT_LC_DY_TMP
Inventory Expenses	E$_W_RTL_INVRC_EXP_IT_LC_DY_TM
Inventory OOS	E$_W_RTL_INVOOS_IT_LC_WK_TMP
Inventory Count (Perpetual)	E$_W_RTL_INVPS_CNT_IT_LC_DY_TM
Inventory Count (Systemic)	E$_W_RTL_INVSS_CNT_IT_LC_DY_TM
Inventory Packs	E$_W_RTL_INVPK_IT_LC_DY_TMP
Inventory Receipts	E$_W_RTL_INVRC_IT_LC_DY_TMP
Inventory Reclass	E$_W_RTL_INVRECLASS_IT_LC_DY_T
Inventory Return to Vendor	E$_W_RTL_INVRTV_IT_LC_DY_TMP
Inventory Transfers	E$_W_RTL_INVTSF_IT_LC_DY_TMP
Inventory Unavailable	E$_W_RTL_INVU_IT_LC_DY_TMP

Subject Area	Rejected Records Table
Inventory Position (when RI_INVAGE_REQ_IND=N)	E$_W_RTL_INV_IT_LC_DY_TMP
Inventory Position (when RI_INVAGE_REQ_IND=Y)	E$_W_RTL_INV_IT_LC_DY_TMP1
Markdowns	E$_W_RTL_MKDN_IT_LC_DY_TMP
Market Sales (Consumer Group)	E$_W_RTL_MKTSLS_TA_CH_CNG_WK_T
Market Sales (Household Group)	E$_W_RTL_MKTSLS_TA_CH_HG_WK_TM
Net Cost	E$_W_RTL_NCOST_IT_LC_DY_TMP
Plan 1	E$_W_RTL_PLAN1_PROD1_LC1_T1_TMP
Plan 2	E$_W_RTL_PLAN2_PROD2_LC2_T2_TMP
Plan 3	E$_W_RTL_PLAN3_PROD3_LC3_T3_TMP
Plan 4	E$_W_RTL_PLAN4_PROD4_LC4_T4_TMP
Plan 5	E$_W_RTL_PLAN5_PROD5_LC5_T5_TMP
Plan 6	E$_W_RTL_PLAN6_PROD6_LC6_T6_TMP
Plan Forecast 1	E$_W_RTL_PLANFC_PROD1_LC1_T1_T
Plan Forecast 2	E$_W_RTL_PLANFC_PROD2_LC2_T2_T
Promotion Actual	E$_W_RTL_PRACT_IT_LC_DY_TMP
Promotion Budget	E$_W_RTL_PRBDGT_IT_LC_TMP
Purchase Order Allocations	E$_W_RTL_PO_ONALC_IT_LC_DY_TMP
Purchase Orders	E$_W_RTL_PO_ONORD_IT_LC_DY_TMP
Price	E$_W_RTL_PRICE_IT_LC_DY_TMP
Replenishment Demand	E$_W_RTL_REPL_DMD_IT_LC_DY_TMP
Replenishment WF Orders	E$_W_RTL_REPL_WF_ORD_IT_LC_DY_
Sales	E$_W_RTL_SLS_TRX_IT_LC_DY_TMP
Sales Consignment	E$_W_RTL_SLSCC_TRX_IT_LC_DY_TM
Sales Discount	E$_W_RTL_SLSDSC_TRX_IT_LC_DY_T
Sales Extensions	E$_W_RTL_SLS_TRX_EXT_IT_LC_DY_
Sales Pack	E$_W_RTL_SLSPK_IT_LC_DY_TMP
Sales Promotion	E$_W_RTL_SLSPR_TX_IT_LC_DY_TMP
Sales Wholesale	E$_W_RTL_SLSWF_IT_LC_DY_TMP
Shipment Details	E$_W_RTL_SHIP_IT_LC_DY_TMP
Stock Ledger (Gregorian Month)	E$_W_RTL_STCKLDGR_SC_LC_MH_G_T
Stock Ledger (Month)	E$_W_RTL_STCKLDGR_SC_LC_MH_TMP
Stock Ledger (Week)	E$_W_RTL_STCKLDGR_SC_LC_WK_TMP
Store Traffic	E$_W_RTL_STTRFC_LC_DY_MI_TMP
Supplier Compliance	E$_W_RTL_SUPPCM_IT_LC_DY_TMP
Supplier Compliance Unfulfilled	E$_W_RTL_SUPPCMUF_LC_DY_TMP
Transaction Tender	E$_W_RTL_TRX_TNDR_LC_DY_TMP
Transfer Details	E$_W_RTL_TSF_IT_LC_DY_TMP
XStore Sales	E$_W_RTL_SLS_POS_IT_LC_DY_TMP

Ad Hoc Rejected Record Reprocessing

Some subject areas have support utilities to aid with reloading records that were rejected because of bad or missing data in nightly batch executions. When records are rejected, they will first be placed into separate tables prefixed with `E$`. From here, you may review the data for issues and go back to the source systems to make corrections and avoid future batch problems. The rejected record utilities supports most fact areas in Retail Insights. The table below shows a few examples of the mapping between rejected record tables and their module code used by the reprocessing programs. The complete list of module codes can be found in the table `C_MODULE_REJECT_TABLE`.

Subject Area	Sub-Module Code	Rejected Records Table
Sales	SLS	E$_W_RTL_SLS_TRX_IT_LC_DY_TMP
Sales Promotion	SLSPR	E$_W_RTL_SLSPR_TX_IT_LC_DY_TMP
Inventory	INV	E$_W_RTL_INV_IT_LC_DY_TMP or E$_W_RTL_INV_IT_LC_DY_TMP1
Price	PRICE	E$_W_RTL_PRICE_IT_LC_DY_TMP

As a prerequisite to running these processes, some one-time cleanup must be done. The inventory and price reload jobs use the `C_HIST_LOAD_STATUS` table in the same manner as historical loads. For this reason, you must erase the values from the `MAX_COMPLETED_DATE` and `HIST_LOAD_STATUS` columns of this table. All rows should show as null values for these fields. This cleanup can be done using the Control & Tactical Center UI.

For inventory only, there are two rejected record tables used by the process, but only one of them is listed in the configuration tables. You do not need to alter the configuration to specify the other `TMP` table; the program will select the correct table automatically based on the value of `RI_INVAGE_REQ_IND` in `C_ODI_PARAM`. `E$_W_RTL_INV_IT_LC_DY_TMP` is used when the parameter is set to `N` (which means the system is not tracking receipt dates or inventory age) while the other table `E$_W_RTL_INV_IT_LC_DY_TMP1` is used when the parameter value is `Y`.

Before attempting to reload any rejections, you will also need to perform another batch run or ad hoc load to correct the associated dimensions, such as adding any missing items or locations. For example, for records that are rejected on Day 1, you must fix the source data and run a normal batch on Day 2; then you are ready to reprocess the older rejections on Day 2+ after the batch cycle. Once the dimensions are fixed, you can follow the steps below to reload the rejected records.

1. Run the `E_FS_RELOAD_JOB` in the process `E_FS_RELOAD_PROCESS_ADHOC`. This job accepts three input values: sub-module code, start date, and end date. The sub-module code is required and comes from the table above (or refer to `C_MODULE_REJECT_TABLE` for a complete list of codes). The dates are optional and specify the range of `CHECK_DATE` values to extract from the `E$` table. If no dates are provided to the job, then it will use `trunc(sysdate-1)` to `trunc(sysdate)` as the start and end date. The values should be entered as parameters on the job in the format:

   ```
   SLS 20230808 20230809
   ```

2. The set of records found for the provided input parameters will be moved from the associated `E$` table to another table prefixed with `ERR`. This table will keep the history of reprocessed records so the data is not lost.
3. Run the job `PROCESS_REJECTED_RECORD_JOB` in the process `PROCESS_REJECTED_RECORD_ADHOC`. This job will move the data from the `ERR` tables to the staging (`FS`) tables.
4. Verify the `FS` table now contains the data you want to load. You also have the ability to directly update the data in the staging tables from Innovation Workbench if any further changes need to be done to make it load successfully.
5. Run the reload process for the data to move the records from the staging table into the data warehouse tables. This will be one of the following ad hoc processes (make sure all jobs in these processes are enabled in Batch Administration before trying to run them):
   - E_SLS_RELOAD_PROCESS_ADHOC
   - E_SLSPR_RELOAD_PROCESS_ADHOC
   - E_INV_RELOAD_PROCESS_ADHOC
   - E_PRICE_RELOAD_PROCESS_ADHOC

For Sales and Sales Promotion, they are kept separate because it's possible to have transactions that were loaded to the base sales tables but were rejected from promotional sales tables. In this case, you might only reprocess the `SLSPR` module, which will not load any new data into the base sales transaction tables. If you see rows rejected on both `SLS` and `SLSPR` `E$` tables, then you want to reprocess both modules, as the tables loaded are differently.

For Inventory and Price, you can see the status of the reload using the `C_HIST_LOAD_STATUS` table, similar to how the historical load is performed. Once a range of dates is loaded successfully in this manner, you cannot go back and reprocess the same records again: the job will not allow you to insert any item/locations that already exist in the fact tables. If you have different, rejected item/location records that still need to be reprocessed, then you must first reset `C_HIST_LOAD_STATUS` to allow past dates to be reprocessed. The inventory reload is also used only to populate the core tables that are common to RAP (`W_RTL_INV_IT_LC_G`, `W_RTL_INV_IT_LC_DY_F` and `W_RTL_INV_IT_LC_WK_A`). For any other inventory aggregates that need to be reloaded, the Aggregation Utility must be used.

The intermediate `ERR` tables used to hold the reload history have a `DELETE_FLG` column to indicate that they've been reloaded to `FS` tables once and should not be used again on future runs of the jobs. If you do want to reprocess the same set of records again to `FS` tables, there is a separate process named `E_FS_RESET_DELETE_FLG_PROCESS_ADHOC` with one job (`E_RESET_DELETE_FLG_JOB`) that accepts a sub-module code, start date, and end date similar to the `E_FS_RELOAD_JOB` parameters. This will change the delete flag back to `N` only for that subset of records, allowing you to start over from step 1 above.

Nightly Rejected Record Reprocessing

The AIF DATA nightly batch process also supports reloading rejected records in a similar manner to the manual reprocessing steps described in Ad Hoc Rejected Record Reprocessing. The steps to follow when you want to load previously rejected records during the nightly batch are:

1. Before the next nightly batch starts, run the `E_FS_RELOAD_JOB` in the process `E_FS_RELOAD_PROCESS_ADHOC`. This job accepts three input values: sub-module code, start date, and end date. The sub-module code is required and comes from the `C_MODULE_REJECT_TABLE`. The dates are optional and specify the range of `CHECK_DATE`

values to extract from the `E$` table. If no dates are provided to the job, then it will use `trunc(sysdate-1)` to `trunc(sysdate)` as the start and end date. The values should be entered as parameters on the job in the format:

```
SLS 20230808 20230809
```

2. The set of records found for the provided input parameters will be moved from the associated `E$` table to another table prefixed with `ERR`. You may repeat `E_FS_RELOAD_JOB` as many times as needed to prepare all your data for processing.
3. During the next nightly batch run, the batch will execute the job `PROCESS_REJECTED_RECORD_JOB` and move all data in the `ERR` tables that has not yet been processed into the associated fact staging (`FS`) tables. This data is combined with your current nightly batch data, so you need to ensure no conflicts will occur between the rejected records and the incoming nightly data.
4. The nightly batch will proceed as usual to load the fact staging tables to the target data warehouse tables. If you have any new rejections (or the records you tried to reprocess were rejected a second time) then these records will be inserted back to their `E$` table with `CHECK_DATE` equal to the current system date.

After processing is complete, the `C_MODULE_REJECT_TABLE` table column `PROCESSED_FLG` is updated to `Y`, indicating that the rejections were processed in batch. The value of this column is `N` if data was prepared by `E_FS_RELOAD_JOB` but not yet processed. Additionally, the `DELETE_FLG` column is updated to `Y` on each `ERR` table that was processed so that those records are not used again by later runs of these procedures.

There are some limitations on this approach to reprocessing rejected data. Any time you reload rejected data, there is a risk that the same item/locations were already updated by a later batch and your attempt to reload data now will cause duplicate records to be loaded. Be sure that the data you are attempting to process was not already loaded before you begin. For positional fact interfaces (such as inventory, purchase orders, price, and costs) only the latest rejected record will be processed if there are multiple for the same set of primary keys but different dates. That latest record will be loaded for the current business date, the same way all other nightly positional data is loaded. If a newer positional record exists in your incoming nightly data, then the same item/location will not be loaded from the rejected records. If an item or location is deleted or closed for a positional record, that record will be ignored. If a positional record was archived due to inactivity (such as inventory remaining at zero for 90 days) then we will ignore those item/locations as well, because the only way such records are rejected is because of invalid/closed dimensions or the rejected record was from before the item/location was archived and therefore it is not reflecting the current position.

Rejected Record Notifications

When records are rejected during a nightly batch, the jobs themselves do not generally fail. The batch will be allowed to complete but the rejected records are placed in separate tables for review. You may enable notifications that will alert your administrator users any time rejections happen in the AIF DATA batch cycles. The notifications are visible anywhere the Notifications panel is available on the left side of the screen, such as in the Retail Home and AI Foundation user interfaces. You can also customize the recipients and behavior from the Manage Notifications screen in Retail Home. Refer to "Notifications Administration" in the *Retail Home Administration Guide* for details. These notifications will be present under the **Retail Insights** application in the dropdown menu, with a notification type code of **AIF_DATA_REJECTION** and a notification name of **AIF Data Rejections**. **AIF DATA** refers to the schedule in POM by the same name, which is what these notifications are issued for.

To enable the rejected record notifications, you must first enable the nightly jobs below in the AIF DATA nightly batch schedule. By default, all jobs should be enabled except `E_INV_REJECT_DATA_NOTIF_JOB`. This job is not needed unless you are configuring the job to fail when rejections are found (using the options described farther below).

Job	Purpose
E_REJECT_DATA_NOTIF_JOB	Checks for rejected records and triggers notifications when they are found. Will only check for rejections in data loaded in the last 24 hours, so that it does not trigger repeatedly for older data.
E_INV_REJECT_DATA_NOTIF_JOB	Checks for rejected records in inventory data and optionally causes the batch to fail immediately, giving you the opportunity to correct the data and reload it before resuming the batch. The batch failure trigger is configured separately, based on the severity configuration.
E_DIMM_LKUP_PROD_CHK_JOB	Compares the loaded product dimension data with the internal lookup table used for downstream integrations and logs any differences for review. Mismatches in the lookup table can cause missing data later in AIF or RPAS apps. The missing records will be in the table `RABE01USER.W_RTL_REJECT_DIMENSION_TMP` with `ERR_PROD_LKP_TMP` as the `TABLE_NAME`.
E_DIMM_LKUP_ORG_CHK_JOB	Compares the loaded organization dimension data with the internal lookup table used for downstream integrations and logs any differences for review. Mismatches in the lookup table can cause missing data later in AIF or RPAS apps. The missing records will be in the table `RABE01USER.W_RTL_REJECT_DIMENSION_TMP` with `ERR_ORG_LKP_TMP` as the `TABLE_NAME`.

Once enabled, the notifications will be issued based on the configuration table `C_MODULE_REJECT_TABLE`. This table has two columns that can be updated:

- `E_NOTIFICATION_ON` – Set to `N` to disable the notifications or `Y` to enable them
- `E_SEVERITY_LEVEL` – Set to `1` to mark the notification as **Critical**, which will also cause the POM job to fail. Set to `2` or `3` for lower severity messages, which will not cause the POM job to fail but will still issue notification messages.

When you set a notification to severity `1` and it causes the batch job to fail, then you must also mark that notification as read to prevent it from causing the job to keep failing in future runs. This can be done on the notifications user interface, either by clicking the **X** icon to clear the notification from the task panel or by opening the full notifications tab and marking them as read using the UI action for it. Refer to the *Retail Home User Guide* chapter on "Notifications" for details on marking notifications as read.

In addition to the Notifications panel in the UI, the messages are also logged in the database if you wish to access them from Innovation Workbench or create a custom service using the data. The table used for the messages themselves is `RADM01.RAF_NOTIFICATION`; you must use the database username as a prefix, because the same table exists for all Oracle users (`RASE01` could be used instead for AIF Apps notifications). The table used to log the job activity is

RI_LOG_MSG; you may query this table where PROGRAM_UNIT = ‘RI_NOTIFICATION_UTIL’ to see when messages are triggered.

Database Hints for SQL Jobs

Oracle Support may need to alter or add to the Oracle SQL hints used by specific programs to improve performance on your dataset. All AIF DATA jobs in ODI support configurable hints using rows added to the C_ODI_PARAM table.

The general process is to insert a row into C_ODI_PARAM with PARAM_NAME set to 'IKM_OPTIMIZER_HINT_INSERT' or 'IKM_OPTIMIZER_HINT_SELECT' and with INTEGRATION_ID set to 'Step/Interface Name'.

Insert statement template:

```
  INSERT INTO c_odi_param (
        row_wid,
        scenario_name,
        scenario_version,
        param_name,
        param_value,
        integration_id,
        created_on_dt,
        change_on_dt
    )
        ( SELECT
            2,
            $ODI_SCENARIO_NAME,
            '001',
            'IKM_OPTIMIZER_HINT_INSERT',
            $HINT_DEFINITION,
            $STEP_NAME,
            sysdate,
            sysdate
        FROM
            dual
        )
```

As an example, we want to add a hint for job step SIL_Retail_SalesTransactionFact inside the scenario 'SIL_RETAIL_SALESTRANSACTIONFACT'. We would run the following statement to add the hint:

```
INSERT INTO c_odi_param (
row_wid,
scenario_name,
scenario_version,
param_name,
param_value,
integration_id,
created_on_dt,
change_on_dt
)
( SELECT
2,
'SIL_RETAIL_SALESTRANSACTIONFACT', -- “Scenario_Name”
'001',
'IKM_OPTIMIZER_HINT_INSERT',
'/* +Append */',
'SIL_Retail_SalesTransactionFact', -- “Step/Interface Name”
sysdate,
sysdate
FROM
dual
)
```

Once a row is added for the first time, it should not be inserted again. Instead, update the `param_value` with the new hint SQL.

Data Model Utilities

Data Warehouse Models

The Innovation Workbench workspace in APEX provides access to internal data warehouse objects by using synonyms. This results in the end-user being unable to directly describe the objects to see their column definitions and other information, because the synonym does not provide any of that information about its underlying table. To access data model information, a utility package named `RI_DATA_MODEL` is provided to query the information in a user-friendly format. Currently, this package provides access to data in the `RADM01` and `RABE01USER` schemas, which covers all tables in the AIF DATA data warehouse.

If you need to query the list of tables in one of the data warehouse schemas, you can use the following commands to do so:

```
select * from table(ri_data_model.ri_table_list('RADM01'));
select * from table(ri_data_model.ri_table_list('RADM01')) WHERE TABLE_NAME
LIKE '%DEAL%';
```

To get the column information for a table, use the following SQL statement, changing the table name in the command as needed:

```
select * from table(ri_data_model.ri_table_desc('W_RTL_SLS_TRX_IT_LC_DY_F'));
```

To get the primary key (PK) information for a table, use the following SQL statement, changing the table name in the command as needed:

```
select * from table(ri_data_model.ri_table_pk('W_RTL_SLS_TRX_IT_LC_DY_F'));
```

To get the foreign key (FK) information for a table, use the following SQL statement, changing the table name in the command as needed:

```
select * from table(ri_data_model.ri_table_fk('W_RTL_SLS_TRX_IT_LC_DY_F'));
```

AI Foundation Models

The AI Foundation applications have a separate utility in Innovation Workbench for reviewing the data models for each solution. There is a help view that can be used to get details about the routines provided. Use the following query:

```
SELECT * FROM rse_data_model_support_help_vw;
```

This will provide a list of the routines that are available, along with a description of the routines and any parameters they support. A summary of the utility functions is provided below.

- `RSE_DATA_MODEL_SUPPORT_UTIL.GET_TABLE_LIST` - Return a list of objects that match the parameter patterns provided
- `RSE_DATA_MODEL_SUPPORT_UTIL.GET_TABLE_PK_LIST` - Return a set of primary keys for one or more objects (`TABLE` and `VIEW`)
- `RSE_DATA_MODEL_SUPPORT_UTIL.GET_TABLE_FK_LIST` - Return a set of foreign keys for one or more objects (`TABLE` or `VIEW`)
- `RSE_DATA_MODEL_SUPPORT_UTIL.GET_TABLE_UK_LIST` - Return a set of unique keys for one or more objects (`TABLE` and `VIEW`)
- `RSE_DATA_MODEL_SUPPORT_UTIL.GET_TABLE_DESC` - Return a list of columns for one or more objects
- `RSE_DATA_MODEL_SUPPORT_UTIL.GET_TABLE_PK` - Return a set of primary keys columns for one or more objects (`TABLE` and `VIEW`)
- `RSE_DATA_MODEL_SUPPORT_UTIL.GET_TABLE_FK` - Return a set of foreign keys columns for one or more objects (`TABLE` and `VIEW`)
- `RSE_DATA_MODEL_SUPPORT_UTIL.GET_TABLE_UK` - Return a set of unique keys columns for one or more objects (`TABLE` and `VIEW`)

Some examples of how to use the routines:

- Get help with a routine:

  ```
  SELECT * FROM rse_data_model_support_help_vw WHERE program_unit =
  'RSE_DATA_MODEL_SUPPORT_UTIL.GET_TABLE_LIST'
  ```

- Execute a routine, providing an optional parameter to help specify the object to return information for:

  ```
  SELECT * FROM RSE_DATA_MODEL_SUPPORT_UTIL.GET_TABLE_LIST ( 'RSE_SLS_%' );
  ```

Any of the parameters listed in the help output can, optionally, be provided to limit the output that the routine provides.