# Oracle® Retail Merchandising System

Operations Guide Addendum

Release 11.0.11

May 2007

ORACLE

Oracle® Retail Merchandising System Operations Guide Addendum, Release 11.0.11



Primary Author: Nathan Young

# Contents

# Preface

Oracle Retail Operations Guides are designed so that you can view and understand the application's 'behind-the-scenes' processing, including such information as the following:

- Key system administration configuration settings
- Technical architecture

## Audience

Anyone with an interest in developing a deeper understanding of the underlying processes and architecture supporting RMS functionality will find valuable information in this guide. There are three audiences in general for whom this guide is written:

- Business analysts looking for information about processes and interfaces to validate the support for business scenarios within RMS and other systems across the enterprise.
- System analysts and system operations personnel:
  - Who are looking for information about RMS processes internally or in relation to the systems across the enterprise.
  - Who operate RMS regularly.
- Integrators and implementation staff with overall responsibility for implementing RMS.

## Related Documents

For more information, see the following documents in the Oracle Retail Merchandising System Release 11.0.11 documentation set:

- Oracle Retail Merchandising System Installation Guide
- Oracle Retail Merchandising System Release Notes
- Oracle Retail Merchandising System Data Model
- Oracle Retail Merchandising System Batch Schedule

## Customer Support

- https://metalink.oracle.com

When contacting Customer Support, please provide:

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

## Conventions

**Navigate:** This is a navigate statement. It tells you how to get to the start of the procedure and ends with a screen shot of the starting point and the statement "the Window Name window opens."

> **Note:** This is a note. It is used to call out information that is important, but not necessarily part of the procedure.

```
This is a code sample
    It is used to display examples of code
```

A hyperlink appears like this.

# 1
# Introduction

## Overview

The information in this document reflects modifications and updates to the *Oracle Retail Merchandising System 11.0 Operations Guide* and any subsequent RMS 11.0.x Operations Guide Addendums. (The RMS 11.0 Operations Guide is the most recent release of the full Operations Guide for the 11.0 release of RMS.) Using this document in conjunction with the *Oracle Retail Merchandising System 11.0 Operations Guide* provides retailers with a complete overview of the application.

For more specific information regarding enhancements and modifications made to the previous Oracle Retail Merchandising System release, see the Oracle Retail Merchandising System 11.0.11 Release Notes.

# 2

# Batch Designs

Retailers should refer to these sections in lieu of the corresponding batch designs in the RMS 11.0 Operations Guide or any subsequent RMS 11.0.x Operation Guide Addendums.

Batch designs describe how, on a technical level, an individual batch module works and the database tables that it affects. In addition, batch designs contain file layout information that is associated with the batch process.

## Distro Price Change Publish [distropcpub]

### Functional Area

Pricing/Transfers/Allocations

### Module Affected

DISTROPCPUB.PC

### Design Overview

The DISTROPCPUB.PC program will get price change information for any allocations and transfers and write the information to the corresponding queue table. This program will ensure that Oracle Retail Warehouse Management will have access to any item/location unit retail information that is changed after an allocation or transfer has been published.

This program will loop through the PRICE_HIST table, selecting records whose unit retail will change for vdate+1, and transaction type is in 4 (single unit retail was changed) or 11(single unit retail and multi-unit retail were changed). It will then search for allocations and transfers with matching item/locations. When a match is found, depending on the distro type, the program calls allocation or transfer publishing logic to insert the data into the allocation or transfer queue table, so that the RIB can publish the change to the warehouse system.

### Scheduling Constraints

| Schedule Information | Description |
|---|---|
| Processing Cycle | PHASE 3 – Daily |
| Scheduling Considerations | This program should run after RPM price event execution batch process. |
| Pre-Processing | N/A |
| Post-Processing | N/A |
| Threading Scheme | Multithreading based on store. |

### Restart/Recovery

The logical unit of work is store. The driving cursor retrieves all item/locations that have price changes in effect from the next day. It also gets all of the component items of the non-sellable packs that have price changes.

### Locking Strategy

N/A

### Security Considerations

N/A

### Performance Considerations

N/A

### Key Tables Affected

| Table | Select | Insert | Update | Delete |
|---|---|---|---|---|
| PERIOD | Yes | No | No | No |
| PRICE_HIST | Yes | No | No | No |
| V_RESTART_STORE | Yes | No | No | No |
| V_PACKSKU_QTY | Yes | No | No | No |
| ITEM_MASTER | Yes | No | No | No |
| ALLOC_HEADER | Yes | No | No | No |
| ALLOC_DETAIL | Yes | No | No | No |
| TSFHEAD | Yes | No | No | No |
| TSFDETAIL | Yes | No | No | No |
| ORDHEAD_REV | Yes | No | No | No |
| ORDHEAD | Yes | No | No | No |
| ALLOC_MFQUEUE | No | Yes | No | No |
| TSF_MFQUEUE | No | Yes | No | No |

### Shared Modules

N/A

### I/O Specification

N/A

# EDI Location Address to Vendor Download [edidladd]

## Design Overview

The purpose of this module is to download addresses of stores and warehouses to vendors. The address fields in the tables store and wh were removed and were referenced instead to the table addr.

The output file format will be a standard Retek file format that will be translated into EDI format by the Gentran translator. Addresses will be downloaded in two different scenarios. The program will download changes made to store or warehouse addresses into a flat file. Further, if a system_options table flag (addr_catalog) is set to true (Y), then the addresses of all stores and warehouses will be downloaded into a different file to be sent to suppliers. This program will run nightly.

When a store or warehouse address is changed a flag will be set to true to indicate that that location should have its address changes submitted to suppliers. When the changes occur on the location forms (store.fmb or wh.fmb form), the addr_change field is set to true (Y) on the location tables (store or wh). After each changed location address is processed by the EDIDL838.pc program its addr_change flag is reset to N.

The addr_catalog flag for sending address catalogs is set on the sys_ctrl.fmb form. When this is set a catalog listing of all store and warehouse addresses are sent to suppliers, then the addr_catalog field on system_options will be reset to N.

## Tables Affected:

| TABLE | INDEX | SELECT | INSERT | UPDATE | DELETE |
|---|---|---|---|---|---|
| ADDR | No | Yes | No | Yes | No |
| ADD_TYPE | No | Yes | No | No | No |
| ADD_TYPE_MODULE | No | Yes | No | No | No |
| PERIOD | No | Yes | No | No | No |
| STORE | No | Yes | No | No | No |
| SYSTEM_OPTIONS | No | Yes | No | Yes | No |
| WH | No | Yes | No | No | No |

## Stored Procedures / Shared Modules (Maintainability)

None**.**

## Program Flow

N/A

## Function Level Description

Init()

Get system variables: current date and catalog indicator flag. Open output files: for location address changes, address change rejects, canonical catalog listing, and catalog listing rejects. (only open catalog and catalog reject files if a catalog is to be written). Call restart_init. Set up format strings for ouput records.

Process()

For each record fetched from the driving cursor, call a function to process location address changes, and if the catalog indicator flag is set to true to write catalog listing of all locations.

Write FTAIL record to file.

The column addr_catalog in table system_options is then updated to 'N' if its previous value was 'Y'.

Write_records_loc()

This function should write out the address change record to the output file and update the flag on the location table.

Write_records_cat()

This function writes address change records to the catalog file.

Final()

Call restart/recovery close.

## I/O Specification

Output file format (all characters should be right-padded with blanks and left justified; all numbers should be left-padded with zeroes and right justified)

| Record Name | Field Name | Field Type | Default Value | Description |
|---|---|---|---|---|
| File header | File type record descriptor | Char(5) | FHEAD | Identifies file record type |
| | File line sequence | Number(10) | Start at 1 and increment | Line number of file |
| | File type definition | Char(5) | DLADD | Identifies file source |
| | Purpose code | Char(2) | 04 (change) or 05 (replace) | Add/change location or replace whole list |
| Transaction Detail | File type record descriptor | Char(5) | TDETL | Identifies file record type |
| | File line sequence | Number(10) | Increment | Line number of file |
| | Transaction number | Number(10) | Start at 1,increment | Identifies transaction |

| Record Name | Field Name | Field Type | Default Value | Description |
|---|---|---|---|---|
| | Date | Char(8) | | Period.vdate YYYYMMDD |
| | Store or warehouse | Char(2) | SN (store) or WH (warehouse) | Location type |
| | Location | Number(10) | | Store.store or wh.wh |
| | Location name | Char(20) | | Store..store_name or wh.wh_name |
| | Address line 1 | Char(30) | | Store.store_add1/wh.wh_add1 |
| | Address line 2 | Char(30) | | Store_add2 or wh_add2 |
| | City | Char(20) | | Store.store_city/wh.wh_city |
| | State | Char(3) | | Store.state/wh.state |
| | Postal code | Char(10) | | Store_pcode/wh_pcode |
| | Country | Char(3) | | Store/wh.country_id |
| | Address Type Description | Char(40) | | atp.type_desc |
| File trailer | File type record descriptor | Char(5) | FTAIL | Identifies file record type |
| | Total number lines | Number(10) | | Total lines in file |
| | Total no. TDETL lines | Number(10) | | (total lines – 2) |

## Restart Recovery

Because of the lack of volume and the flexibility requirements of EDI, the program will use Oracle Retail's standard restart/recovery only minimally. The driving query volume is limited to the volume of the store and warehouse tables. Further, the output files that are created are created if they don't exist and are overwritten if they already exist. In the event of a fatal error it is, therefore, reasonable to expect clients to simply restart the job from the beginning without recovery.

Driving cursor:

```
   SELECT :ps_str_type,
          s.store,
          s.store_name,
          a.add_1,
          a.add_2,
          a.city,
          a.state,
          a.post,
          a.country_id,
          a.edi_addr_chg,
          atp.type_desc,
          ROWIDTOCHAR(a.rowid)
     FROM store           s,
          addr            a,
          add_type_module atm,
          add_type        atp
    WHERE a.addr_type        = atm.address_type
      AND a.module           = atm.module
      AND atm.address_type   = atp.address_type
      AND a.key_value_1      = to_char(s.store)
      AND a.edi_addr_chg     = DECODE(:pi_do_all,1,a.edi_addr_chg,'Y')
      AND a.primary_addr_ind = 'Y'
      AND a.module           = 'ST'
UNION ALL
   SELECT :ps_wh_type,
          w.wh,
          w.wh_name,
          a.add_1,
          a.add_2,
          a.city,
          a.state,
          a.post,
          a.country_id,
          a.edi_addr_chg,
          atp.type_desc,
          ROWIDTOCHAR(a.rowid)
     FROM wh              w,
          addr            a,
          add_type_module atm,
          add_type        atp
    WHERE a.addr_type        = atm.address_type
      AND a.module           = atm.module
      AND atm.address_type   = atp.address_type
      AND a.key_value_1      = to_char(w.wh)
      AND a.edi_addr_chg     = DECODE(:pi_do_all,1,a.edi_addr_chg,'Y')
      AND a.primary_addr_ind = 'Y'
      AND a.module           = 'WH';
```

This selects all location information if the catalog is to be processed but only information from locations where the addr_chg indicator is set to Y if the catalog shouldn't be processed. Pi_do_all will be an integer variable that should be initialized as zero and set to 1 just before calling process_catalog.

### Scheduling Constraints

Processing Cycle: PHASE 4 (DAILY)
Scheduling Diagram: N/A
Pre-Processing: N/A
Post-Processing: N/A
Threading Scheme: N/A

## New and Changed Supplier Address Upload [ediupadd]

### Design Overview

The ediupadd.pc batch program will read vendor/supplier sent EDI 838 Profile Data Files. These files will be processed by vendor/supplier and used to update Oracle Retail supplier address information.

Five different types of supplier addresses can be changed via this EDI interface: business, postal, returned to, pick up and payment mailing address. This program always assumes that address information is primary for the address type.

If there is an error with a transaction set, write that transaction to the reject file so it can be fixed and reprocessed later, then the program will continue to the next transaction set. See EDI856 for an example.

| TABLE | INDEX | SELECT | INSERT | UPDATE | DELETE |
|---|---|---|---|---|---|
| STATE | No | Yes | No | No | No |
| SUPPLIER | No | Yes | No | No | No |
| SUPS_ADD | No | No | Yes | Yes | No |

### Scheduling Constraints

Processing Cycle: Ad Hoc
Scheduling Diagram: N/A
Pre-Processing: N/A
Post-Processing: N/A
Threading Scheme: N/A (file based processing—single thread only)

### Restart Recovery

Oracle Retail restart/recovery capability is minimal. The program uses non-fatal error handling to process input files. There is not enough volume to warrant the use of restart recovery. A commit will not occur until the end of file processing and therefore if fatal errors are encountered updates will not have been committed and the program can be restarted without recovery.

### Program Flow

N/A

## Shared Modules

N/A

## Function Level Description

Init()

Open input & output files. Validate file header line. Call Retek API restart_init to initializse restart/recovery process.

Process()

Loop through each line of input file. Call validate_FDETL to validate and format variables.. Write to reject file if variables are not valid. Call update_supplier to update the database with the new addresses.

Update_supplier()

Update sups_add table with new, modified address information. This program presumes that address modification information is regarding a "primary" address. If the action specified is "add", then any existing primary address of that type will be changed so that it is no longer primary, and the new address will be inserted as a primary address of the specified type into the supplier address table. If the action is "update", then the existing primary address of the specified type will be modified to the new passed information (if one does not exist for that type, it will be inserted).

validate_FDETL—validate supplier and address information

## I/O Specification

Input file:

FHEAD File type identification

FDETL Supplier address info

FTAIL End of file marker

| Record Name | Field Name | Field Type | Default Value | Description |
|---|---|---|---|---|
| FHEAD | File record descriptor | Char(5) | FHEAD | Describes file line type |
| | Line number | Number(10) | | Sequential file line number |
| | Gentran_id | Char(5) | | Identifies the file type |
| | File create date | Char(14) | | YYYYMMDDHH24MISS format |
| FDETL | File record descriptor | Char(5) | FDETL | Describes file line type |
| | Line number | Number(10) | | Sequential file line number |
| | Transaction number | Number(10) | | Sequential transaction number |
| | Add or Update | Char(1) | | 'A'dd or 'U'pdate address |

| Record Name | Field Name | Field Type | Default Value | Description |
|---|---|---|---|---|
| | Address type | Char(2) | | Will be translated into sups_add.address_type:<br>01 – Business<br>02 – Postal<br>03 – returns<br>04 – Pick Up (Order)<br>05 – Payment |
| | Supplier | Varchar(10) | | Sups.supplier |
| | Address line 1 | Char(30) | | Sups_add.address_1 |
| | Address line 2 | Char(30) | | Sups_add.address_2 |
| | Address line 3 | Char(30) | | Sups_add.address_3 |
| | Contact name | Char(20) | | Sups_add.contact_name |
| | Contact Phone | Char(20) | | Sups_add.contact_phone |
| | Contact fax | Char(20) | | Sups_add.contact_fax |
| | City | Char(20) | | Sups_add.city |
| | State | Char(3) | | Sups_add.state |
| | Postal code | Char(10) | | Sups_add.post_code |
| | Country | Char(3) | | Sups_add.country_id |
| FTAIL | File record descriptor | Char(5) | | Describes file record type |
| | Line number | Number(10) | | Sequential file line number (total # lines in file) |
| | Number of transactions | Number(10) | | Number of transactions in file |

## Technical Issues

N/A

# Oracle Retail Demand Forecasting Purge [fcstprg]

## Functional Area

Demand Forecasting

## Module Affected

FCSTPRG.PC

## Design Overview

This program deletes data from forecast information tables. Data deletion is performed by partition truncation, table truncation or deletion by domain. The method of deletion is dependent on whether or not the table is partitioned. This program serves to delete data by domains so that they can re-loaded with new forecast information from RDF.

This program must be run as either the RMS schema owner, or be run by a user that has been granted the following system privileges:

'drop any table'

'alter any table'

## Scheduling Constraints

| Schedule Information | Description |
|---|---|
| Processing Cycle | AD-HOC |
| Scheduling Considerations | N/A |
| Pre-Processing | prepost fcstprg pre - disables indexes |
| Post-Processing | prepost fcstprg post - rebuilds indexes |
| Threading Scheme | N/A |

## Restart/Recovery

N/A

## Locking Strategy

N/A

## Security Considerations

N/A

## Performance Considerations

N/A

### Key Tables Affected

| Table | Select | Insert | Update | Delete |
|---|---|---|---|---|
| ITEM_FORECAST | No | No | No | Yes |
| DEPT_SALES_FORECAST | No | No | No | Yes |
| CLASS_SALES_FORECAST | No | No | No | Yes |
| SUBCLASS_SALES_FORECAST | No | No | No | Yes |

### I/O Specification

N/A

## Oracle Retail Demand Forecasting Rollup by Department, Class and Subclass [fcstrbld_sbc]

### Functional Area

Demand Forecasting

### Module Affected

FCSTRBLD_SBC.PC

### Design Overview

The module rolls up the sales forecast data at subclass and class level to class and department level respectively and inserts the data. The program selects records from the table SUBCLASS_SALES_FORECAST and writes the records to CLASS_SALES_FORECAST and selects the data from CLASS_SALES_FORECAST and writes into DEPT_SALES_FORECAST using the domain ID stored in the table FORECAST_REBUILD. The record in FORECAST_REBUILD is deleted after the record is written to the above destination tables.

### Scheduling Constraints

| Schedule Information | Description |
|---|---|
| Processing Cycle | Phase 3 (Weekly) |
| Scheduling Considerations | After completion of FCSTRBLD.PC. |
| Pre-Processing | N/A |
| Post-Processing | N/A |
| Threading Scheme | N/A |

### Restart/Recovery

Restart/recovery is based on the values stored in restart_bookmark from the last commit prior to failure. The values are for the last domain_id that was not rolled up completely.

### Locking Strategy

N/A

### Security Considerations
N/A

### Performance Considerations
N/A

### Key Tables Affected

| Table | Select | Insert | Update | Delete |
|---|---|---|---|---|
| FORECAST_REBUILD | Yes | No | No | Yes |
| CLASS_SALES_FORECAST | Yes | Yes | No | No |
| DEPT_SALES_FORECAST | No | Yes | No | No |
| SUBCLASS_SALES_FORECAST | Yes | No | No | No |
| STORE | Yes | No | No | No |

### I/O Specification
N/A

## Oracle Retail Demand Forecasting Rollup [fcstrbld]

### Functional Area
Demand Forecasting

### Module Affected
FCSTRBLD.PC

### Design Overview
This program is designed to roll-up new or updated forecasted unit sales data from the item_forecast table. This data will be summarized into the subclass, class and department level sales forecast tables.

### Scheduling Constraints

| Schedule Information | Description |
|---|---|
| Processing Cycle | PHASE 3 (weekly) |
| Scheduling Considerations | N/A |
| Pre-Processing | N/A |
| Post-Processing | prepost fcstrbld post – truncates the FORECAST_REBUILD table |
| Threading Scheme | Threaded by domain id |

### Restart/Recovery
The logical unit of work is a domain id. The program commits each time the rollups (dept, class and subclass) for a domain id is successfully processed.

### Locking Strategy
N/A

### Security Considerations
N/A

### Performance Considerations
N/A

### Key Tables Affected

| Table | Select | Insert | Update | Delete |
|---|---|---|---|---|
| FORECAST_REBUILD | Yes | No | No | Yes |
| SUBCLASS_SALES_FORECAST | Yes | Yes | No | No |
| ITEM_MASTER | Yes | No | No | No |
| ITEM_FORECAST | Yes | No | No | No |
| STORE | Yes | No | No | No |
| CLASS_SALES_FORECAST | Yes | Yes | No | No |
| DEPT_SALES_FORECAST | Yes | Yes | No | No |

### I/O Specification
N/A

## Geocode Hierarchy Upload [gcupld]

### Functional Area
Geocode hierarchy

### Module Affected
GCUPLD.PC

### Design Overview
A geocode identifies a combination of the country, state, county and city in which locations operate.

GCUPLD.PC (geocode hierarchy upload) provides the ability to upload geocodes from an outside source into RMS. This batch module lets retailers delete current geocodes and create new geocodes in the system. A flat file is used to feed the program the additions and deletions to the geocode tables. Validation determines if duplicate records exist, dependencies exist, and the flat file is in the correct format. If errors occur in the validation of the record, it is written out to a reject file to allow further investigation of the record.

## Scheduling Constraints

| Schedule Information | Description |
|---|---|
| Processing Cycle | Ad Hoc |
| Scheduling Considerations | Ad Hoc |
| Pre-Processing | N/A |
| Post-Processing | N/A |
| Threading Scheme | N/A |

## Restart/Recovery

This is a file based upload and a file based restart/recovery logic. The commit_max_ctr field should be set to prevent excessive rollback space usage, and to reduce the overhead of the file I/O. The recommended commit counter setting is 10000 records (subject to change based on implementation).

## Locking Strategy

N/A

## Security Considerations

N/A

## Performance Considerations

N/A

## Key Tables Affected

| Table | Select | Insert | Update | Delete |
|---|---|---|---|---|
| GEOCODE_TEMP | YES | YES | NO | YES |
| DISTRICT_GEOCODES | YES | YES | NO | YES |
| CITY_GEOCODES | YES | YES | NO | YES |
| COUNTY_GEOCODES | YES | YES | NO | YES |
| STATE_GEOCODES | YES | YES | NO | YES |
| COUNTRY_GEOCODES | YES | YES | NO | YES |
| GEOCODE_STORE | YES | NO | NO | NO |
| GEOCODE_TXCDE | YES | NO | NO | NO |

## I/O Specification

### Output File Layout

| Record Name | Field Name | Field Type | Default Value | Description |
|---|---|---|---|---|
| FHEAD | File head descriptor | Char(5) | FHEAD | Describes the file line type |
| | Line id | Char(10) | 0000000001 | Sequential file line number |
| | Gentran ID | Char(4) | 'GCUP' | Identifies which translation Gentran uses |
| | Current date | Char(14) | | File date in YYYYMMDDHH24MISS format |
| FDETL | File record descriptor | Char(5) | FDETL | Describes file line type |
| | Line id | Char(10) | | Sequential file line number |
| | Country Geocode | Char(4) | | Country Geocode |
| | State Geocode | Char(4) | | State Geocode |
| | County Geocode | Char(4) | | County Geocode |
| | City Geocode | Char(4) | | City Geocode |
| | District Geocode | Char(4) | | District Geocode |
| | Geocode Level | Char(6) | | Geocode Level Valid values are: 'CNTRY,'STATE','COUNTY', 'CITY', 'DIST' |
| | Geocode Description | Char(40) | | Geocode Description |
| | Add Delete Ind | Char(1) | | Add/delete Indicator<br>Valid values are: 'A', 'D' |
| FTAIL | File record descriptor | Char(5) | FTAIL | Marks end of file |
| | Line id | Char(10) | | Sequential file line number |
| | Number of lines | Number(10) | | Number of lines in file not counting FHEAD and FTAIL |

# Inventory Adjustment Purge [invaprg]

## Functional Area

Inventory Adjustment

## Module Affected

INVAPRG.PC

## Design Overview

The Inventory Adjustment Purge module deletes all obsolete inventory adjustment records whose adjustment date has elapsed by a pre-determined number of months. The number of months that inventory adjustment records are kept before they are purged by this batch is defined in the SYSTEM_OPTIONS table.

## Scheduling Constraints

| Schedule Information | Description |
|---|---|
| Processing Cycle | AD-HOC (monthly) |
| Scheduling Considerations | N/A |
| Pre-Processing | N/A |
| Post-Processing | N/A |
| Threading Scheme | N/A |

## Restart/Recovery

N/A

## Locking Strategy

N/A

## Performance Considerations

N/A

## Key Tables Affected

| Table | Select | Insert | Update | Delete |
|---|---|---|---|---|
| UNIT_OPTIONS | Yes | No | No | No |
| PERIOD | Yes | No | No | No |
| INV_ADJ | No | No | No | Yes |

## Shared Modules

N/A

## I/O Specification

N/A

# Price History Data Purge [prchstprg.pc]

## Functional Area

Pricing

## Module Affected

PRCHSTPRG.PC

## Design Overview

The PRCHSTPRG program deletes price_hist records, which are older than a number of retention days specified in a new column added to system_options table as system_options.price_hist_retention_days. This program keeps the latest record for the combination of item, location and tran type and deletes the rest of the records, which fall in the specified period of retention days.

## Scheduling Constraints

| Schedule Information | Description |
|---|---|
| Processing Cycle | PHASE AD-HOC (daily) |
| Scheduling Considerations | This program is run prior to phase 3 to improve select operations. |
| Pre-Processing | N/A |
| Post-Processing | N/A |
| Threading Scheme | Multi threaded. Threaded by table partition |

## Restart/Recovery

This program will use the commit_max_ctr on the restart_control table to periodically commit SQL delete operations. Restart/Recovery is achieved by processing records that have not been deleted. Table restart_bookmark stores the ps_cur_restart_partition_position for partition position as bookmark_string to restart a thread.

However, in cases where the price_hist table is very large, a particularly large rollback segment may be specified to reduce the risk of exceeding rollback segment space. This will depend on the size of normal rollback segments and the size of the price_hist table.

## Locking Strategy

N/A

## Security Considerations

N/A.

## Performance Considerations

The commit_max_ctr field should be set to prevent excessive rollback space usage, and to reduce the overhead of file I/O. The recommended commit counter setting is 10000 records (subject to change based on experimentation). In case price_hist table is very large then the number of partitions on the table may be increased and then after the number of threads for this program should be increased.

### Key Tables Affected

| Table | Select | Insert | Update | Delete |
|---|---|---|---|---|
| PERIOD | Yes | No | No | No |
| SYSTEM_OPTIONS | Yes | No | No | No |
| PRICE_HIST | No | No | No | Yes |
| DBA_TAB_PARTITIONS | Yes | No | No | No |

### I/O Specification

#### Output File Layout

N/A

## Pre/Post Functionality for Multi-Threadable Programs [prepost]

### Design Overview

The Pre/Post module facilitates multi-threading by allowing general system administration functions (such as table deletions or mass updates) to be completed after all threads of a particular program have been processed. A brief description of all pre- or post-processing functions included in this program can be found in the Function-Level Description section.

This program will take three parameters: username/password to log on to Oracle, a program before or after which this script must run and an indicator telling whether the script is a pre or post function. It will act as a shell script for running all pre-program and post-program updates and purges (the logic was removed from the programs themselves to enable multi-threading & restart/recovery).

For example, to run the pre-program script for the ccext program, the following should be entered on the command line:

```
prepost    user/password    rpl    pre
```

### Tables Affected

| TABLE | SELECT | INSERT | UPDATE | INDEX | DELETE | TRUNCATE | TRIGGER |
|---|---|---|---|---|---|---|---|
| all_constraints | Y | N | N | N | N | N | N |
| all_ind_partitions | Y | N | N | N | N | N | N |
| all_policies | Y | N | N | N | N | N | N |
| alloc_detail | Y | N | N | N | N | N | Y |
| alloc_header | Y | N | N | N | N | N | Y |
| class | Y | N | N | N | N | N | N |
| class_sales_forecast | N | N | N | Y | N | Y | N |
| class_sales_hist | N | N | N | N | Y | N | N |
| class_sales_hist_mth | Y | N | N | N | Y | N | N |
| cost_change_trigger_temp | Y | N | N | Y | N | Y | N |
| cost_susp_head | N | N | Y | N | N | N | N |
| daily_data | Y | N | N | N | N | N | N |
| daily_data_temp | Y | N | N | N | N | Y | N |

| TABLE | SELECT | INSERT | UPDATE | INDEX | DELETE | TRUNCATE | TRIGGER |
|---|---|---|---|---|---|---|---|
| dba_indexes | Y | N | N | N | N | N | N |
| dba_triggers | Y | N | N | N | N | N | N |
| dealfct_temp | N | Y | N | N | N | N | N |
| deal_actuals_forecast | Y | N | N | N | N | N | N |
| deal_actuals_item_loc | Y | Y | N | N | N | N | N |
| deal_bb_no_rebate_temp | N | Y | N | N | N | Y | N |
| deal_bb_rebate_po_temp | N | Y | N | N | N | Y | N |
| deal_bb_receipt_sales_tempp | N | Y | N | N | N | Y | N |
| deal_head | Y | N | N | N | N | N | N |
| deal_item_loc_explode | Y | N | N | N | N | N | N |
| deal_sku_temp | N | N | N | Y | N | Y | N |
| deps | Y | N | N | N | N | N | N |
| dept_sales_forecast | N | N | N | Y | N | Y | N |
| dept_sales_hist | N | N | N | N | Y | N | N |
| dept_sales_hist_mth | Y | N | N | N | Y | N | N |
| domain_class | N | N | Y | N | N | N | N |
| domain_dept | N | N | Y | N | N | N | N |
| domain_subclass | N | N | Y | N | N | N | N |
| edi_daily_sales | N | N | N | N | Y | N | N |
| edi_ord_temp | N | N | N | Y | N | Y | N |
| fif_receiving | N | Y | N | Y | N | Y | N |
| fixed_deal | Y | N | Y | N | N | N | N |
| forecast_rebuild | N | N | N | Y | N | Y | N |
| groups | Y | N | N | N | N | N | N |
| hist_rebuild_mask | Y | N | N | Y | N | Y | N |
| ib_results | N | N | Y | N | N | N | N |
| if_tran_data | Y | N | N | N | N | N | N |
| invc_detail | N | N | Y | N | N | N | N |
| invc_detail_temp | Y | N | N | N | N | Y | N |
| invc_detail_temp2 | N | N | N | N | N | Y | N |
| invc_head | N | N | Y | N | N | N | N |
| invc_head_temp | Y | N | N | N | N | Y | N |
| item_forecast | N | N | N | Y | N | N | N |
| item_loc | Y | N | N | N | N | N | N |
| item_loc_temp | N | Y | N | N | N | Y | N |
| item_master | Y | N | N | N | N | N | N |
| item_supp_country | Y | N | N | N | N | N | N |
| item_supp_country_loc | Y | N | N | N | N | N | N |
| mc_rejections | N | N | N | Y | N | Y | N |
| mod_order_item_hts | N | N | N | Y | N | Y | N |
| on_order_temp | N | N | N | Y | N | Y | N |
| ord_missed | N | N | N | Y | N | Y | N |

| TABLE | SELECT | INSERT | UPDATE | INDEX | DELETE | TRUNCATE | TRIGGER |
|---|---|---|---|---|---|---|---|
| ord_temp | N | N | N | Y | N | Y | N |
| ordhead | Y | N | N | N | N | N | N |
| ordsku | Y | N | N | N | N | N | N |
| packitem | Y | N | N | N | N | N | N |
| period | Y | N | N | N | N | N | N |
| pos_button_head | N | N | Y | N | N | N | N |
| pos_coupon_head | N | N | Y | N | N | N | N |
| pos_merch_criteria | N | N | Y | N | N | N | N |
| pos_mods | N | Y | N | Y | N | Y | N |
| pos_money_ord_head | N | N | Y | N | N | N | N |
| pos_payinout_head | N | N | Y | N | N | N | N |
| pos_prod_rest_head | N | N | Y | N | N | N | N |
| pos_store | N | N | Y | N | N | N | N |
| pos_sup_pay_criteria | N | N | Y | N | N | N | N |
| pos_tender_type_head | N | N | Y | N | N | N | N |
| reclass_cost_chg_queue | Y | Y | Y | N | N | N | N |
| reclass_head | Y | N | N | N | N | N | N |
| reclass_item | Y | N | N | N | N | N | Y |
| reclass_trigger_temp | Y | N | N | Y | Y | Y | N |
| repl_attr_update_exclude | Y | N | N | N | Y | N | N |
| repl_attr_update_head | Y | N | N | N | Y | N | N |
| repl_attr_update_item | Y | N | N | N | Y | N | N |
| repl_attr_update_loc | Y | N | N | N | Y | N | N |
| repl_day | Y | Y | N | N | N | N | N |
| repl_item_loc | Y | Y | N | N | N | N | N |
| repl_item_loc_updates | N | Y | N | Y | N | Y | N |
| rpl_alloc_in_tmp | N | Y | N | N | N | Y | N |
| rpl_distro_tmp | N | Y | N | N | N | Y | N |
| salweek_c_daily | N | Y | N | N | N | Y | N |
| salweek_c_week | Y | Y | N | N | N | Y | N |
| salweek_restart_dept | N | Y | N | N | N | Y | N |
| sec_user_zone_matrix | N | N | N | Y | N | Y | N |
| stage_complex_deal_detail | N | N | N | N | N | Y | N |
| stage_complex_deal_head | N | N | N | N | N | Y | N |
| stage_fixed_deal_detail | N | N | N | N | N | Y | N |
| stage_fixed_deal_head | N | N | N | N | N | Y | N |
| stake_head | Y | N | N | N | N | N | N |
| stake_prod_loc | Y | N | N | N | N | N | N |
| stake_sku_loc | Y | N | N | N | N | N | N |
| store | Y | N | Y | N | N | N | N |
| store_add | Y | N | N | N | Y | N | N |
| subclass_sales_forecast | N | N | N | Y | N | N | N |

| TABLE | SELECT | INSERT | UPDATE | INDEX | DELETE | TRUNCATE | TRIGGER |
|---|---|---|---|---|---|---|---|
| subclass_sales_hist | N | N | N | N | Y | N | N |
| subclass_sales_hist_mth | Y | N | N | N | Y | N | N |
| sup_data | N | N | N | N | Y | N | N |
| sups_min_fail | N | N | N | Y | N | Y | N |
| system_options | Y | N | N | N | N | N | N |
| system_variables | Y | N | Y | N | N | N | N |
| temp_tran_data | Y | N | N | N | N | Y | N |
| temp_tran_data_sum | N | Y | N | N | N | Y | N |
| tif_explode | N | N | N | Y | N | Y | N |
| tran_data | N | Y | N | N | N | N | N |
| tsf_head | N | N | Y | N | N | N | N |
| vat_code_rates | Y | N | N | N | N | N | N |
| vat_item | Y | N | N | N | N | N | N |
| week_data_temp | N | N | N | N | N | Y | N |
| wh | Y | N | N | N | N | N | N |
| wh_store_assign | N | N | N | N | Y | N | N |

### Scheduling Constraints

Processing Cycle: PHASE ALL (daily)

Scheduling Diagram: See scheduling flow for description of all pre-post requirements in the daily run.

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: N/A (single threaded)

### Restart Recovery

N/A

### Program Flow

N/A

### Shared Modules

FORECASTS_SQL.GET_SYSTEM_FORECAST_IND

UDA_SQL.CHECK_REQD_NO_VALUE

FORECASTS_SQL.GET_DOMAIN

ITEM_ATTRIB_SQL.GET_PACK_INDS

FORECASTS_SQL.GET_ITEM_FORECAST_IND

POS_UPDATE_SQL.POS_INVC_DETAIL_INSERT

CAL_TO_454_LDOM

CAL_TO_454_HALF

CAL_TO_CAL_HALF

CAL_TO_CAL_LDOM

CAL_TO_454_WEEKNO

CAL_TO_CAL_WEEKNO

CAL_TO_454

HALF_TO_CAL_FDOH

HALF_TO_CAL_LDOH

HALF_TO_454_FDOH

HALF_TO_454_LDOH

DBMS_RLS.ENABLE_POLICY

### Function Level Description

Functions to be used by the individual program functions:

modify_indexes()

This function allows indexes to be disabled or rebuilt before and/or after the action that affects them. The individual program passes in the table name and mode (what action to take "disable" or "rebuild") and performs that action. The owner of the index is determined using the synonym_trace function in the library oracle.pc.

get_lock()

This function locks the table that is passed to it. If this function fails to acquire a lock to the specified table, it retries MAX_LOCK_TRIES times before returning a fatal error.

modify_partition_indexes()

This is called by the modify_indexes function to determine if the indexes that need modified are partitioned indexes. If so, then the statement is modified to take that into account to accomplish the action. Index_owner, index_name and mode is passed to this function. Nothing is passed back out.

truncate_table()

The table_name is passed to this function so that it can be truncated. The owner of the table is determined by using the synonym_trace function in the library oracle.pc.

modify_trigger()

Allows triggers to be disabled or enabled before or after certain processes. The table_name, trigger name and mode( "DISABLE" or "ENABLE") are passed to this function and the appropriate action is taken. No values are passed back to the calling function.

alter_constraints()

This function diables, enables, or rebuilds a table constraint based on the table name and the mode passed into it. It is called by vendinv_pre**().**

truncate_user_sec_table()

This is a function used to run the szonrbld pre functions that will truncate the sec_user_zone_matrix table. Disables any indexes prior to the truncation on the associated table and rebuilds/enables them following the truncation.The user running this program for this function must have been granted the 'drop any table' and 'alter any index' system privilege, or be the owning schema user.

get_454_ldom()

This function calls the procedure CAL_TO_454_LDOM to get the 454 last day of month.

get_454_half()

This function calls the procedure CAL_TO_454_HALF to get the 454 calendar half number.

get_next_454_half()

This function calls the procedure CAL_TO_454_HALF to get the next end-of-month 454 calendar half number.

get_next_cal_half()

This function calls the procedure CAL_TO_CAL_HALF to get the next end-of-month half number on the regular calendar.

get_cal_half()

This function calls the procedure CAL_TO_CAL_HALF to get the half number on the regular calendar

get_cal_ldom()

This function calls the procedure CAL_TO_CAL_LDOM to get the end of the month on the regular calendar.

get_454_weekno()

This function calls the procedure CAL_TO_454_WEEKNO to get the 454 week number in half.

get_cal_weekno()

This function calls the procedure CAL_TO_CAL_WEEKNO to get the week number in half on the regular calendar.

get_454_date()

This function calls the procedure CAL_TO_454 to get the 454 calendar week number.

get_cal_fdoh()

This function calls the procedure HALF_TO_CAL_FDOH to get the first day of half.

get_cal_ldoh()

This function calls the procedure HALF_TO_CAL_LDOH to get the last day of half.

get_454_fdoh(void);

This function calls the procedure TO_454_FDOH to get the first day of half in 454 calendar.

get_454_ldoh(void)

This function calls the procedure HALF_TO_454_LDOH to get the last day of half in 454 calendar.

get_tomorrow()

This function gets the next day after the vdate.

get_forecast_ind()

This function cals FORECASTS_SQL.GET_SYSTEM_FORECAST_IND to get the system_forecast_ind.

validate_reclassify()

Validates the reclassification. If the reclassification is rejected, then the data from the RECLASS_TRIGGER_TEMP table is deleted, else the data is inserted into RECLASS_COST_CHG_QUEUE table.

check_stock_count()

This function checks for the existence of a stock count of an item in the STAKE_SKU_LOC or STAKE_PROD_LOC.

check_order()

This function checks for the existence of an order for an item in the ORDHEAD and ORDSKU tables.

check_uda()

This function calls UDA_SQL.CHECK_REQD_NO_VALUE which determines if an item's new hierarchy has any required UDA defaults that the item is not currently associated with.

check_domain_exists()

This function calls FORECASTS_SQL.GET_DOMAIN to check for the existence of the domain for a merchandise hierarchy.

check_forecast()

This function validates the reclassification of an item based on forecast indicator. First, it checks if the item passed is a pack through the package call to ITEM_ATTRIB_SQL.GET_PACK_INDS. Then for non-pack items, it calls FORECASTS_SQL.GET_ITEM_FORECAST_IND to get the item forecast indicator.

delete_reclass_trigger_temp()

This function deletes the records for a given item from the RECLASS_TRIGGER_TEMP.

### Individual Program Functions

rpl_pre()

This function truncates the following tables before replenishment extracts are performed:

- ORD_TEMP
- ORD_MISSED

It also disables any indexes prior to the truncation on the associated tables and rebuilds/enables them following the truncation. The user running this program for this function must have been granted the 'drop any table' and 'alter any index' system privilege, or be the owning schema user.

salweek_pre()

This function truncates, then populates the tables SALWEEK_C_WEEK, SALWEEK_C_DAILY, and SALWEEK_RESTART_DEPT.

SALWEEK_C_WEEK is populated with records from the tables DAILY_DATA_TEMP, and WEEK_DATA whose eow_date are between the last eow date and the current eow date.

SALWEEK_C_DAILY is populated with records from the tables DAILY_DATA and DAILY_DATA_TEMP whose eow_date are between the last eow date and the current eow date.

SALWEEK_RESTART_DEPT is populated with the departments, threads, and the count of department records in the SALWEEK_C_WEEK.

salweek_post()

Updates the last end-of-week date on the SYSTEM_VARIABLES table to the run date after all weekly stock ledger data has been processed.

salmth_post()

Updates the following SYSTEM_VARIABLES columns to reflect the current date's values after all monthly stock ledger data has been processed:

- last_eom_half_no
- last_eom_month_no
- last_eom_date
- next_eom_date
- last_eom_start_half
- last_eom_end_half
- last_eom_start_month
- last_eom_mid_month
- last_eom_next_half_no
- last_eom_day
- last_eom_week
- last_eom_month
- last_eom_year
- last_eom_week_in_half

rplapprv_pre()

This function truncates the SUPS_MIN_FAIL table. It disables any indexes prior to the truncation on the associated table and rebuilds/enables it after being truncated. The user running this program for this function must have been granted the 'drop any table' and 'alter any index' system privilege, or be the owning schema user.

rplatupd_pre()

This function truncates the MC_REJECTIONS table so that it is free to hold new mass change rejections. It disables any indexes prior to the truncation on the associated tables and rebuilds/enables them following the truncation. The user running this program for this function must have been granted the 'drop any table' and 'alter any index' system privilege, or be the owning schema user.

rplatupd_post()

This function truncates the holding tables REPL_ATTR_UPDATE_ITEM and REPL_ATTR_UPDATE_LOC after their records have been processed. It disables any indexes prior to the truncation on the associated tables and rebuilds/enables them following the truncation. The user running this program for this function must have been granted the 'drop any table' and 'alter any index' system privilege, or be the owning schema user.

rilmaint_post()

This function locks then truncates the REPL_ITEM_LOC_UPDATES table after these records are processed so the table is free to hold new updates. It disables any indexes prior to the truncation on the associated tables and rebuilds/enables them following the truncation. The user running this program for this function must have been granted the 'drop any table' and 'alter any index' system privilege, or be the owning schema user.

supmth_post()

Deletes records from table SUP_DATA after all daily supplier data records have been rolled up to month level.

sccext_post()

Updates all processed supplier cost change record status to 'Extracted'.

hstbld_pre()

Deletes sales history data for the dept exists in the table hist_rebuild_mask from the three tables subclass_sales_hist, class_sales_hist and dept_sales_hist prior to running hstbld in rebuild mode.

hstbld_post()

This function truncates the holding table MASK_REBUILD after building history records. It disables any indexes prior to the truncation on the associated tables and rebuilds/enables them following the truncation. The user running this program for this function must have been granted the 'drop any table' and 'alter any index' system privilege, or be the owning schema user.

posdnld_post()

This clears the POS_MODS table after all records have been downloaded to the POS. It disables any indexes prior to the truncation on the associated tables and rebuilds/enables them following the truncation. The user running this program for this function must have been granted the 'drop any table' and 'alter any index' system privilege, or be the owning schema user.

poscdnld_post()

This clears the config_status and loc_grp_status in POS_LOC_GRP and sets all values of extract_req_ind to 'N'. It clears the status column in POS_MERCH_CRITERIA. It also sets the status_ind column in POS_STORE to 'N'.

reqext_post()

This function updates the TSFHEAD table and sets the status to 'A', approval_id to 'BATCH', approval_date to the vdate, and the repl_tsf_approve_ind to 'N' where the repl_tsf_approve_ind is equal to 'Y'.

likestore_post()

This function should only be run after both storeadd.pc and all threads of likestore.pc have successfully completed.

In the REPL_ITEM_LOC, table, likestore_post selects and inserts all information from the a like store for the new store.

stkupd_pre()

Calls the stored function DBMS_MVIEW.REFRESH.

stkupd_post()

This function disables the RMS_COL_ITL_UR_AUR trigger of ITEM_LOC.

dtesys_post()

Enables the RMS_COL_ITL_UR_AUR trigger of ITEM_LOC table.

ociroq_pre()

This function truncates the rpl_net_inventory_tmp table, which is populated by the ociroq.c and queried from reqext.pc. This function also inserts records into RPL_DISTRO_TMP values from ALLOC_DETAIL, and ALLOC_HEAD table, and into RPL_ALLOC_IN_TMP values from ALLOC_DETAIL, ALLOC_HEAD, and ORDHEAD table. This function also creates a unique index in these two destination tables.

rplext_post()

Truncates the tables RPL_DISTRO_TMP, and RPL_ALLOC_IN_TMP.

posupld_post()

This updates the columns total_merch_cost , total_qty, invc_qty, INVC_HEAD tables based on the corresponding columns in the INVC_HEAD_TEMP table.

vatdlxpl_post()

This inserts into pos_mods all transaction level items on the vat_item table where the item has a new tran_code. Also, if a sub-transaction level item is on vat_item, it is inserted into the pos_mods table, along with its parent item. These items are not picked up by the vatdlxpl program because the vat_code rate has not changed.

saleoh_pre()

Calculates the next_eom_date, and updates the SYSTEM_VARIABLES.

dealday_pre()

This gets the total sales and purchases from the TEMP_TRAN_DATA table and inserts a new record in TEMP_TRAN_DATA_SUM based on dept, class, subclass, loc_type, location, tran_date, and tran_code.

dealday_post()

Copies the contents of the table TEMP_TRAN_DATA_SUM into TRAN_DATA table. Afterwards, then TEMP_TRAN_DATA_SUM is truncated.

hstbldmth_post()

This is responsible for deleting records in the following tables:

- CLASS_SALES_HIST_MTH
- SUBCLASS_SALES_HIST_MTH
- CLASS_SALES_HIST_MTH
- DEPT_SALES_HIST_MTH

THE FOLLOWING FUNCTIONS SHOULD BE RUN AFTER THE edidlprd PROGRAM!

edidlprd_post()

Deletes old records from the EDI_DAILY_SALES table after they have been processed.

fcstrbld_post()

This truncates the holding table FORECAST_REBUILD after all records have been processed. It disables any indexes prior to the truncation on the associated tables and rebuilds/enables them following the truncation. The user running this program for this function must have been granted the 'drop any table' and 'alter any index' system privilege, or be the owning schema user.

vrplbld_post()

This truncates the EDI_ORD_TEMP table after all replenishment orders have been build from the data held there. Disables any indexes prior to the truncation on the associated tables and rebuilds/enables them following the truncation. The user running this program for this function must have been granted the 'drop any table' and 'alter any index' system privilege, or be the owning schema user.

cntrordb_post()

Sets the last_cont_order_date on system_variables to vdate.

fifgldn1_post()

If Oracle Financials is being used, delete everything from the fif_receiving table and repopulate it from the if_tran_data table. Disables any indexes prior to the truncation on the associated tables and rebuilds/enables them following the truncation. The user running this program for this function must have been granted the 'drop any table' and 'alter any index' system privilege, or be the owning schema user.

fsadnld_post()

Updates the load_sales_ind to 'N' for all records on the appropriate domain table – domain_dept, domain_class, or domain_subclass, where system_options.domain_level = 'D', 'C', or 'S', respectively.

policy_enable()

Enables or disables policies.

whstrasg_post ()

Deletes all warehouse store assignment records from the warehouse store assignment table if the assignment date (wh_store_assign.assign_date) is less than or equal to the current date (period.vdate) minus the warehouse store assignment history days (system_options.wh_store_assign_hist_days).

costcalc_post()

This truncates the deal_sku_temp table. This disables any indexes prior to the truncation on the associated tables and rebuilds/enables them following the truncation. The user running this program for this function must have been granted the 'drop any table' and 'alter any index' system privilege, or be the owning schema user.

tifposdn_post()

This truncates tif_explode table. It disables any indexes prior to the truncation on the associated tables and rebuilds/enables them following the truncation. The user running this program for this function must have been granted the 'drop any table' and 'alter any index' system privilege, or be the owning schema user. It disables any indexes prior to the truncation on the associated tables and rebuilds/enables them following the truncation.

htsupld_pre()

This truncates the mod_order_item_hts table so that reports will be correct and not include data from previous runs of htsupld. It disables any indexes prior to the truncation on the associated tables and rebuilds/enables them following the truncation. The user running this program for this function must have been granted the 'drop any table' and 'alter any index' system privilege, or be the owning schema user.

onordext_pre()

This truncates the on_order_temp table. It disables any indexes prior to the truncation on the associated tables and rebuilds/enables them following the truncation. The user running this program for this function must have been granted the 'drop any table' and 'alter any index' system privilege, or be the owning schema user.

precostcalc_pre()

This processeses records from the COST_CHANGE_TRIGGER_TEMP and RECLASS_TRIGGER_TEMP tables. Reclass_trigger_temp is populated only by database trigger and cost_change_trigger_temp is populated by database trigger and edi_cost_change_sql.create_cost_chg.

This function will either insert new records or update existing ones on reclass_cost_chg_queue. Both tables, COST_CHANGE_TRIGGER_TEMP and RECLASS_TRIGGER_TEMP are truncated and their indexes rebuilt at the end of this function. The user running this program for this function must have been granted the 'drop any table' and 'alter any index' system privilege, or be the owning schema user.

reclsdly_pre()

This disables the trigger RMS_TABLE_RCS_BIDR on the reclass_item table. The user running this program for this function must have been granted the 'alter any trigger' system privilege, or be the owning schema user.

Reclsdly_post()

Call the packaged function RECLASS_SQL.POST_PROCESS which does the following tasks:

If the item is moving to a new dept, class, or subclass a record is inserted to the table hist_rebuild_mask.

If the item is reclassified to a new department, and the single_style_po_ind = 'Y' then the order in the table ordhead is updated with the new department.

If the item is reclassified to a new department, then determine if the item is included in an item-list and insert a record into skulist_dept if the item is in an item-list and the new department is not associated to the item-list..

Delete item records associated with the old hierarchy that exist on the DEAL_ITEM_LOC_EXPLODE table.

Also, Clear the dept field in the ordhead table when the item has changed departments

ibcalc_pre()

This updates the status on ib_results to 'U'nprocessed where the status = 'W'orksheet so after ibcalc is run, multiple records in 'W'orksheet status will not exist for each item/location.

fcstprg_pre()

This disables any indexes prior to the truncation on following tables. This is run BEFORE the fcstprg.pc program on PARTITIONED TABLES only:

- ITEM_FORECAST
- DEPT_SALES_FORECAST
- CLASS_SALES_FORECAST
- SUBCLASS_SALES_FORECAST

The user running this program for this function must have been granted the 'alter any index' system privilege, or be the owning schema user.

fcstprg_post()

This rebuilds the indexes following truncation of following tables:

- ITEM_FORECAST
- DEPT_SALES_FORECAST
- CLASS_SALES_FORECAST
- SUBCLASS_SALES_FORECAST

The user running this program for this function must have been granted the 'alter any index' system privilege, or be the owning schema user.

dealinc_pre()

Call get_sys_date()

Call size_arrays()

Loops through the deal actuals item loc table and create any item/loc/order combinations in the table that have previous turnovers but do not exist in future periods.

dealfct_pre()

This inserts details of forecast periods for active deal components that require processing into dealfct_temp table.

dealact_pre_no_rebate()

Truncates the deal_bb_no_rebate_temp table.

Then inserts billback NO Rebate type of deal into deal_bb_no_rebate_temp.

dealact_pre_rebate_po()

Truncates the deal_bb_rebate_po_temp table.

Then inserts billback rebate PO type of deal into deal_bb_rebate_po_temp.

dealact_pre_receipt_sales ()

Truncates the deal_bb_receipt_sales_temp.

Then inserts billback rebate Sales and Receipt type of deal into deal_bb_receipt_sales_temp.

vendinvc_pre()

Truncate the STAGE_COMPLEX_DEAL_HEAD table.

Truncate the STAGE_COMPLEX_DEAL_DETAIL table.

Then inserts complex deals for invoicing into vendinvc_temp.

vendinvf_pre()

Truncate the STAGE_FIXED_DEAL_HEAD table.

Truncate the STAGE_FIXED_DEAL_DETAIL table.

vendinvc_post()

Get vdate.

Call process_deal_head().

vendinvf_post()

Get vdate.

Call process_fixed_deal().

process_fixed_deal()

For each active Fixed Deal record where the Collect End Date is earlier than the vdate, set it's status to Inactive.

process_deal_head()

For each active Deal Head record where Est Next Invoice Date, Close Date, Last Invoice Date and Last EOM Date are earlier than vdate, AND Billing Type is Off Invoice and Invoice processing Logic !='NO', set the Est Next Invoice Date to null.

### I/O Specification

N/A

### Technical Issues

N/A

## Reclassification of Item [reclsdy]

### Design Overview

The Item Reclassification batch program is executed in order to reclassify items from one department, class or subclass to another. The reclassification of items into a different merchandise hierarchy level is initiated or requested online in the Item Reclassification dialog, with an effective date specified. A parameter passed to the program dictates whether validation (P) or validation and execution (E) logic is performed. Each record that is processed is written to the reclass_error_log with a success_ind of 'S' if successful or 'R' if rejected. In the event that the record fails it will also be written to the mc_rejections table which provides details as to the reason for the rejection.

As noted above, the logic executed by the program is directed by the process mode. If the process mode = 'P' (pre-validation), only the logic validating the item reclassification is performed. The item will NOT be reclassified and the only impact to the item will be a record written to the reclass_error_log table. If the item passes the reclassification validation a record is written to the reclass_error_log with success_ind = 'S'. If the item fails the reclassification validation a record is written to the reclass_error_log with success_ind = 'R' and a record is also written to the mc_rejections table. In pre-validation mode the program reads in all reclassification requests for each item being reclassified and performs the following logic:

- check if item is forecastable and if it is, then check for the existence of a domain. If the item is forecastable and no domain association to the new merchandise hierarchy level exists, the item fails validation. This only holds for non-pack Items since pack items are not forecastable.
- check if the item has UDA (user defined attributes) defined for all required UDA's of the new merchandise hierarchy level, if not the item fails validation
- check if the item has UDA defined for all UDA's that have default set up for the new merchandise hierarchy level, if not, a warning message is written to the report, but the item passes validation
- check if the item exists on an approved order. If the item exists on an approved order and the system level variable single_style_po_ind = N, the item fails validation

If the process mode = 'E' the program first performs the same validation logic that is listed for pre-validation mode, and for those items that pass validation the program executes the item reclassification. In execution mode the program reads in reclassification events that are scheduled for tomorrow (vdate +1) or earlier. In addition to the validation noted above the following logic is executed:

- check if the item exists on an approved order. If the item exists on an approved order, the system level variable single_style_po_ind = 'Y', and the item is being moved to a new department the order will be updated with the new department. Also, the OTB table is updated to transfer cancel_amt, approved_amt and received_amt from the old merchandise hierarchy level to the new merchandise hierarchy level
- check if the item is scheduled for a stock count. If the item is scheduled for a stock count, the stock count is scheduled by item-list, and the stock date has not yet been reached update the dept, class, subclass (as appropriate) for the stock count. If the item is scheduled for a unit stock count*, the stock count is scheduled by merchandise hierarchy, and the reclassification is scheduled between the stock lockout date and the stock take date one of three updates will occur:
  - if reclassified item's new dept, class, and subclass and item's old dept, class, subclass are both included in stock count update stake_sku_loc with new dept, class, subclass
  - if reclassified item's new dept, class, and subclass are included in stock count but item's old dept, class, subclass are not included in stock count add the item to stake_sku_loc
  - if reclassified item's new dept, class, and subclass are not included in stock count but item's old dept, class, subclass are included in stock count delete the item from stake_sku_loc

* If the item is scheduled for a unit and dollar stock count the reclassification will fail validation.

- update item_master with the new merchandise hierarchy
- insert records into pos_mods with a transaction code of 13 (item reclassification) for each item/store combination
- insert records into tran_data with transaction types of 34 (reclassification in) and 36 (reclassification out) for each sku/store combination
- if the reclassification causes a change of domains for the item, the item store tables are updated, setting the last_sales_export_date to NULL. A NULL will result in all the item store's sales history to be downloaded during the sales download process to the external forecasting system. This is required because of the domain change.
- if an item that is part of an item-list is reclassified to a new department, and the new department is not associated to the item-list, insert a record into skulist_dept with the new department/item-list
- insert new and old dept number into hist_rebuild_mask so that the sales history of the dept where the reclassified items moved from/to can be rebuilt later when calling hstbld.pc.

### Tables Affected

| TABLE | INDEX | SELECT | INSERT | UPDATE | DELETE |
|---|---|---|---|---|---|
| DEAL_CALC_QUEUE | No | Yes | Yes | No | No |
| HIST_REBUILD_MASK | No | No | Yes | No | No |
| ITEM_MASTER | No | Yes | No | Yes | No |
| MC_REJECTIONS | No | No | Yes | No | No |
| ORDHEAD | No | Yes | No | Yes | No |
| ORDSKU | No | Yes | No | No | No |
| POS_MODS | No | No | Yes | No | No |
| RECLASS_ERROR_LOG | No | No | Yes | Yes | Yes |
| RECLASS_HEAD | No | Yes | No | No | Yes |
| RECLASS_ITEM | No | Yes | No | No | Yes |
| SYSTEM_OPTIONS | No | Yes | No | No | No |
| TRAN_DATA | No | No | Yes | No | No |

### Stored Procedures / Shared Modules (Maintainability)

FORECASTS_SQL.GET_SYSTEM_FORECAST_IND

FORECASTS_SQL.GET_DOMAIN

RECLASS_SQL.ITEM_PROCESS

ITEMLIST_ATTRIB_SQL.ITEM_IN_SKULIST

ITEMLIST_MC_REJECTIONS_SQL.INSERT_REJECTS

## Function Level Description

**main()**

standard Oracle Retail main function that calls init(), process(), and final()

**init()**

Retrieve system level variables; single_style_po_ind, otb_system_ind and stake_lockout_days

Initialize restart/recovery

Declare a structure of arrays to store rows read in from the driving cursor

Set up arrays to receive fetches in table_process() – memory should be dynamically allocated so that the cursor fetches arrays the size of max counter on restart_control (pi_commit_max_ctr)

Get the current date

Call delete_log() function to purge reclass_error_log of all records with the same process_ind as the program is currently running

**Process()**

Open driving cursor

Get the system forecast indicator

Enter while loop

Fetch pi_commit_max_ctr records at a time into the structure, exiting loop when all rows are processed

If process_mode = 'E':

- Call delete_reclass_head() function to delete previous reclassification event
- Call check_domain_exists() function
- If the item group has changed issue a commit and set a new savepoint
- Call delete_reclass_item() function to delete the reclass item
- If any item in the item group has failed reclassification skip the rest of the items in the group, else call process_item() function to process the item reclassification.
- Finally, if the reclass has not failed, call the delete_dile() function.

Else if process_mode = 'P':

- If the item group has changed set a new savepoint and call the check_domain_exists() and process_item() functions to process the item reclassification validation
- If the item reclassification has failed rollback to the last savepoint and call insert_reject_record() function to write a reject record to the mc_rejections table and call insert_log() function to insert a record to the reclass_error_log with a success_ind = 'R'. If process_mode = 'E' call delete_reclass_item() function to delete the item from reclass_item

Else if item reclassification is successful call insert_log() function to write a record to the reclass_error_log with a success_ind = 'S'.

**Check_domain_exists()**

If system_forecast_ind = 'Y' get domain for new dept/class/subclass

**Process_item()**

Call PL/SQL function RECLASS_SQL.ITEM_PROCESS to perform reclassification validation or validation/execution logic.

**Delete_reclass_head()**

Delete the reclassification event from reclass_head

**Delete_reclass_item()**

Delete the reclassification item from reclass_item

**Insert_log()**

Insert reclass log error record into the reclass_error_log table.

**Insert_reject_record()**

Call PL/SQL function ITEMLIST_MC_REJECTS_SQL.INSERT_REJECTS to insert a record to the mc_rejections table

**Delete_log()**

Delete reclass_error_log records where the process_ind is equal to the process_mode in which the program is currently running

**Size_arrays()**

Size the reclass_recs structure members.

## Input Specifications

Driving cursor:

```
SELECT ROWIDTOCHAR(ri.rowid),
       ri.reclass_no,
       ri.item,      /* This is a level 1 item */
       im.item,
       NVL(im.item_parent,' '),
       NVL(im.item_grandparent,' '),
       im.item_level,
       im.tran_level,
       rh.to_dept,
       rh.to_class,
       rh.to_subclass,
       rh.reclass_date,
       im.dept,
       im.class,
       im.subclass
  FROM v_restart_reclass rv,
       reclass_item ri,
       reclass_head rh,
       item_master im
 WHERE rh.reclass_no   = ri.reclass_no
   AND rh.reclass_date <= decode(:ps_process_mode, 'E', TO_DATE(:ps_vdate,
'YYYYMMDD'), rh.reclass_date)
   AND (ri.item = im.item
        OR ri.item = im.item_parent
```

```
            OR ri.item = im.item_grandparent)
      AND rv.driver_value = rh.reclass_no
      AND rv.driver_name  = :ps_driver_name
      AND rv.num_threads  = TO_NUMBER(:ps_num_threads)
      AND rv.thread_val   = TO_NUMBER(:ps_thread_val)
      AND (rh.reclass_no > NVL(:ps_restart_reclass_no, -999) OR
           (rh.reclass_no = :ps_restart_reclass_no AND
            ri.item > :ps_restart_item))
    UNION
    -- This is for simple pack
    SELECT ROWIDTOCHAR(ri.rowid),
           ri.reclass_no,
           ri.item,
           im.item,
           NVL(im.item_parent,' '),
           NVL(im.item_grandparent,' '),
           im.item_level,
           im.tran_level,
           rh.to_dept,
           rh.to_class,
           rh.to_subclass,
           rh.reclass_date,
           im.dept,
           im.class,
           im.subclass
      FROM v_restart_reclass rv,
           reclass_item ri,
           reclass_head rh,
           packitem pi,
           item_master im
     WHERE rh.reclass_no = ri.reclass_no
       AND rh.reclass_date <= decode(:ps_process_mode, 'E', TO_DATE(:ps_vdate,
'YYYYMMDD'), rh.reclass_date)
       AND im.simple_pack_ind = 'Y'
       AND (im.item = pi.pack_no OR
            im.item_parent = pi.pack_no OR
            im.item_grandparent = pi.pack_no)
       AND EXISTS (SELECT 'x'
                     FROM item_master im1
                    WHERE pi.item = im1.item
                      AND im1.item_level = im1.tran_level
                      AND (ri.item = im1.item
```

```
                OR ri.item = im1.item_parent
                OR ri.item = im1.item_grandparent))
  AND rv.driver_value = rh.reclass_no
  AND rv.driver_name  = :ps_driver_name
  AND rv.num_threads  = TO_NUMBER(:ps_num_threads)
  AND rv.thread_val   = TO_NUMBER(:ps_thread_val)
  AND (rh.reclass_no > NVL(:ps_restart_reclass_no, -999) OR
      (rh.reclass_no = :ps_restart_reclass_no AND
        ri.item > :ps_restart_item))
ORDER BY 2,
              3;
```

### Output Specifications

N/A

### Scheduling Considerations

| | |
|---|---|
| Processing Cycle: | PHASE 3 (daily) |
| Pre-Processing: | cremhierdly.pc |
| Post-Processing: | prepost reclsdly post |
| Threading Scheme: | v_restart_reclass |

### Restart Recovery

The reclsdly.pc batch program has multi-threading capabilities (reclass_no) as well as restart/recovery functionality. The logical unit of work for this program is reclass_no, item.