Oracle® Retail Merchandising System

Operations Guide Addendum

Release 10.1.23

May 2009

Oracle® Retail Merchandising System Operations Guide Addendum, Release 10.1.23

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
- Functional integration dataflow across the enterprise

This Operations Guide Addendum should be used in conjunction with previously released Oracle Retail Merchandising System 10.x documentation.

Audience

Anyone with an interest in developing a deeper understanding of the underlying processes and architecture supporting Oracle Retail Merchandising System 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 and other systems across the enterprise.
- System analysts and system operations personnel:
  - Who are looking for information about Oracle Retail Merchandising System's processes internally or in relation to the systems across the enterprise.
  - Who operate Oracle Retail Merchandising System regularly.
- Integrators and implementation staff with overall responsibility for implementing Oracle Retail Merchandising System.

Related Documents

For more information, see the following documents in the Oracle Retail Merchandising System Release 10.1.23 documentation set:

- *Oracle Retail Merchandising System Installation Guide*
- *Oracle Retail Merchandising System Release Notes*
- *Oracle Retail Merchandising System Batch Schedule*

Customer Support

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

https://metalink.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

Review Patch Documentation

If you are installing the application for the first time, you install either a base release (for example, 13.0) or a later patch release (for example, 13.0.2). If you are installing a software version other than the base release, be sure to read the documentation for each patch release (since the base release) before you begin installation. Patch documentation can contain critical information related to the base release and code changes that have been made since the base release.

Oracle Retail Documentation on the Oracle Technology Network

In addition to being packaged with each product release (on the base or patch level), all Oracle Retail documentation is available on the following Web site (with the exception of the Data Model which is only available with the release packaged code):

http://www.oracle.com/technology/documentation/oracle_retail.html

Documentation should be available on this Web site within a month after a product release. Note that documentation is always available with the packaged code on the release date.

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

The information in this document reflects modifications and updates to the *Oracle Retail Merchandising System 10.0 Operations Guide* and any subsequent RMS 10.x.x Operations Guide Addendums. Using this document in conjunction with the *Oracle Retail Merchandising System 10.0 Operations Guide* provides retailers with a complete overview of the application.

For the RMS 10.1.23 release, the following batch designs have been added in order to address documentation bugs:

- Order Purge [ordprg]
- Reclassification of Item [reclsdly]
- Item Requisition Extraction [reqext]

For more specific information regarding enhancements and modifications made to the previous Oracle Retail Merchandising System release, see the *Oracle Retail Merchandising System 10.1.23 Release Notes*.

2 Batch Designs

Retailers should refer to these sections in lieu of the corresponding batch designs in the RMS 10.0 Operations Guide or any subsequent RMS 10.x.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.

Order Purge [ordprg]

Functional Area

Purchase Orders

Module Affected

ORDPRG.PC

Design Overview

The purpose of this module is to remove old orders and allocations from the system.

If the import indicator on the SYSTEM OPTIONS table (import_ind) is 'N' and if invoice matching is not installed, then all details associated with an order are deleted when the order has been closed for more months than specified in UNIT_OPTIONS (order_history_months). If invoice matching is installed, then all details associated with an order are deleted when the order has been closed for more months than specified in UNIT_OPTIONS (order_history_months). Orders are deleted only if shipments from the order have been completely matched to invoices or closed, and all those invoices have been posted.

If the import indicator on the SYSTEM OPTIONS table (import_ind) is 'Y' and if invoice matching is not installed, then all details associated with the order are deleted when the order has been closed for more months than specified in UNIT_OPTIONS (order_history_months) , as long as all ALC records associated with an order are in 'Processed' status, specified in ALC_HEAD (status). If invoice matching is installed, then all details associated with an order are deleted when the order has been closed for more months than specified in UNIT_OPTIONS (order_history_months), as long as all ALC records associated with an order are in 'Processed' status, specified in ALC_HEAD (status), and as long as all shipments from the order have been completely matched to invoices or closed, and all those invoices have been posted.

This program will also create a PO header flat file to interface with the Nautilus system. When orders are deleted, a record with the action type = 'D'eleted will be written to an output file. Nautilus will then process this file and delete the PO from the warehouse's database to maintain consistency between the host and warehouse environment.

Removed existing driving cursor declaration, open, fetch and close (the same logic has been moved to Prepost ordprg_pre).

A new driving cursor added using doc_purge_queue which is populated in ordprg_pre.

Added a new function ORDER_SQL.VALIDATE_DOC call to validate the PO (to decide candidate for purging or not).

Removed purge_alloc function, since stand-alone allocations (with and w/o shipments) will be purged in tsfprg.

Logic for document validation in ordprg.pc :-

```
/*
     For each allocation for the given PO
      Check the allocation exists in doc_purge_queue populated in ordprg_pre
        if exists
           for each shipment for the allocation
              check if all the docs in shipment exists in doc_purge_queue (multi
doc shipment)
                 if exists
                    purge shipments
                 else
                    don't purge the po and allocation (I_valid = 0)
           if no shipment exists for allocation
             purge the po and allocation (I_valid = 1)
         else (if doc doesn't exists in doc_purge_queue)
           don't purge the po (I_valid = 0)
       if no allocation exists then purge the PO (I_valid = 1)
*/
```

Tables Affected

TABLE	INDEX	SELECT	INSERT	UPDATE	DELETE
ALC_COMP_LOC	No	No	No	No	Yes
ALC_HEAD	No	Yes	No	No	Yes
ALLOC_CHRG	No	No	No	No	Yes
ALLOC_DETAIL	No	No	No	No	Yes
ALLOC_HEADER	Yes	No	No	No	Yes
ALLOC_REV	No	No	No	No	Yes
APPT_HEAD	No	Yes	No	No	Yes
APPT_DETAIL	No	Yes	No	No	Yes
CE_LIC_VISA	No	No	No	No	Yes
CE_CHARGES	No	No	No	No	Yes
CE_ORD_ITEM	No	Yes	No	No	Yes
CE_SHIPMENT	No	No	No	No	Yes
CE_HEAD	No	No	No	No	Yes
DOC_CLOSE_QUEUE	No	No	No	No	Yes
INVC_DETAIL	Yes	No	No	No	Yes
INVC_HEAD	Yes	Yes	No	No	Yes
INVC_MATCH_WKSHT	Yes	No	No	No	Yes
INVC_MATCH_QUEUE	Yes	No	No	No	Yes
INVC_MERCH_VAT	Yes	No	No	No	Yes
INVC_NON_MERCH	Yes	No	No	No	Yes

TABLE	INDEX	SELECT	INSERT	UPDATE	DELETE
INVC_XREF	Yes	No	No	No	Yes
OBLIGATION	No	No	No	No	Yes
OBLIGATION_COMP	No	No	No	No	Yes
OBLIGATION_COMP_LOC	No	No	No	No	Yes
ORDCUST	No	No	No	No	Yes
ORDHEAD_DISCOUNT	No	No	No	No	Yes
ORDHEAD	Yes	Yes	No	No	Yes
ORDLOC	Yes	No	No	No	Yes
ORDSKU_INVC_COST	No	No	No	No	Yes
ITEM_MASTER	Yes	No	No	No	Yes
LC_ORDAPPLY	No	No	No	No	Yes
DEAL_HEAD	No	No	No	No	Yes
DEAL_DETAIL	No	No	No	No	Yes
DEAL_ITEMLOC	No	No	No	No	Yes
DEAL_THRESHOLD	No	No	No	No	Yes
DEAL_QUEUE	No	No	No	No	Yes
DEAL_CALC_QUEUE	No	No	No	No	Yes
ORD_INV_MGMT	No	No	No	No	Yes
REPL_RESULTS	No	No	No	No	Yes
PERIOD	No	Yes	No	No	No
QC	Yes	No	No	No	Yes
RTV_DETAIL	No	No	No	No	Yes
SHIPMENT	Yes	Yes	No	No	Yes
SHIPSKU	Yes	No	No	No	Yes
SYSTEM_OPTIONS	No	Yes	No	No	No
TRANS_CLAIMS	No	No	No	No	Yes
TRANS_DELIVERY	No	No	No	No	Yes
TRANS_LIC_VISA	No	No	No	No	Yes
TRANS_PACKING	No	No	No	No	Yes
TRANSPORTATION	No	Yes	No	No	Yes
TSFDETAIL	Yes	No	No	No	Yes
TSFHEAD	Yes	No	No	No	Yes
UNIT_OPTIONS	No	Yes	No	No	No
DOC_PURGE_QUEUE	Yes	Yes	No	No	No

Stored Procedures/Shared Modules (Maintainability)

INV_SQL.DELETE_INVC

ORDER_SQL.VALIDATE_DOC

Program Flow

Function Level Description

Delete from the appropriate ordering tables and any tables that may have referential integrity constraints for the fetched order number. Fetch order numbers appropriately based on whether or not invoice matching is installed.

Init()

Select the following fields values:

- invc_match_ind, import_ind, repl_order_history_days, edi_rev_days, rws_ind from the system_options table
- order_history_months from the unit_options table
- vdate from period table

Init_logistics()

Initialize the format of the output file.

Process()

Call del_rev to delete order revision

Open the particular driving cursor based on the indicator inv_match_ind

For each order:

```
Fetch the particular driving cursor based on inv_match_ind
If ordlc.lc_ind = 'Y' then skip and fetch next record;
If import_ind = 'Y' and  alc_head.status ='PR' then
        Call delete_landed_costs;
End if;
Delete from related tables associated with orders being purged by.
Calling del_appts, delete_invc_data, delete_deals and delete_repl_orders
Delete from ordhead table.
```

Removed purge_alloc function, since stand-alone allocations (with and w/o shipments) will be purged in tsfprg.

Call ORDER_SQL.VALIDATE_DOC function which will validate the given PO and decides whether it's need to be purged or not.

Delete_invc_data()

Call INVC_SQL.DELETE_INVC function to delete the invoice data for the specific orders being purged.

Del_rev()

Delete records from the tables ordloc_rev, ordsku_rev, ordhead_rev and alloc_rev associated with the orders which have been closed for more days than specified in edi_rev_days(in table UNIT_OPTIONS). But deleting occurs only:

- when a letter of credit is not present(ordlc.lc_ind='N').
- Import indicator equals 'N'. Or

- import indicator equals 'Y', and the landed costs are completed (alc_head.status = 'PR'). In this case, purge these landed costs before deleteing the above tables.

Also, before deleting from these tables, purge all related transportation and custom entries.

Purge_transport()

Delete from the table transportation for specific orders being purged as well as child records from tables missing_doc, trans_packing, trans_delivery, trans_claims and trans_lic_visa(based on transportation_id).

Purge_customs_entry()

Delete from customs entry for specific orders being purged.

Delete_landed_costs()

Delete landed costs and obligations as well as their child records for specific orders being purged. Involved tables include: alc_head, alc_comp_loc, obligation, obligation_comp, obligation_comp_loc.

Write_logistic_details()

Write the deleted order to the output file with action type = 'D'eleted.

Delete_deals()

Delete all PO-specific deals assigned to the order being purged. PO-specific deals are identified by the existence of a value in deal_head.order_no.

Del_repl_orders ()

Delete records from the table ord_inv_mgmt and repl_results associated with the order being purged.

Final()

Close the output file.

Del_appts()

Deletes records from appt_detail, first saving distinct appt/loc combination into a local array that is dynamically sized based on the number of records to be deleted from appt_head. Then array deletes records based on the array from appt_head. Also deletes from doc_close_queue. Calls size_appt_array() to size the appt_head delete array.

Size_appt_array()

Sizes the array used to hold appt_head appt/loc info between deletes from appt_detail and appt_head.

Input Specifications

Driving cursor:

```
EXEC SQL DECLARE c_orders CURSOR FOR
 SELECT dpq.doc,
        NVL(lc.lc_ind,0)
   FROM doc_purge_queue dpq,
        ordlc lc
  WHERE doc = lc.order_no(+)
    AND doc_type = 'P'
```

```
      ORDER by 1;

    EXEC SQL DECLARE c_check_import_ind CURSOR FOR
       SELECT import_order_ind
         FROM ordhead
        WHERE order_no = :ps_order_no;
```

Output Specifications

Output file: Format will be as in table given below. Each order is a separate transaction; multiple orders can be given in each file. The skeleton of file is:

FHEAD (file identification – only one line per file) REQUIRED

FDETL detail lines – one line for every record on ordhead. NOT REQUIRED.

FTAIL (total number lines) One line per file. REQUIRED.

Output File

Record Name	Field Name	Field Type	Default Value	Description
File Header	File Type Record Descriptor	CHAR(5)	FHEAD	Value that identifies the record type.
	Line Identifier	NUMBER(10)		identifies file line number
	File Type Definition	CHAR(4)	POHD	identifies file type
	File Create Date	CHAR(14)		YYYYMMDDHH24MISS format
File Detail	File Type Record Descriptor	CHAR(5)	"FDETL"	Value that identifies the record type.
	Line Number	NUMBER(10)		identifies file line number
	Action type	CHAR(1)	D	identifies the action type 'D'elete
	Location	CHAR(4)		ordloc.location number
	File create date	CHAR(12)	vdate	
	Order number	CHAR(8)		
	Vendor	CHAR(7)		
	Preassigned Flag	CHAR(1)		
	Dev not before	CHAR(8)		
	Dev not after	CHAR(8)		
	Shipping terms	CHAR(3)		
	Buyer code	CHAR(12)		
File Trailer	File Type Record Descriptor	CHAR(5)	FTAIL	Value that identifies the record type.
	Line number	NUMBER(10)		identifies file line number
	File Record Counter	NUMBER(6)		number of records/transactions processed in current file

Scheduling Considerations

Schedule Information	Description
Processing Cycle	PHASE AD-HOC (monthly)
Scheduling Diagram	N/A
Pre-Processing	YES
Post-Processing	N/A
Threading Scheme	N/A (single threaded)

Reclassification of Item [reclsdly]

Functional Area

Reclassification

Module Affected

RECLSDLY.PC

Design Overview

The Item Reclassification batch program is executed in order to reclassify items from one department, class or subclass to another. This reclassification of items into different merchandise hierarchy level is initiated or requested online in the item reclassification dialog, with an effective date specified. This program reads in the reclassification requests that are effective the following day, and for each item being reclassified, the following functions are executed:

- Checks if the item is forecastable and if it is, then checks for the existence of a domain. If the item is forecastable and no domain association to the new merchandise hierarchy level exists, reject the item (i.e. the item will not be reclassified).
- Updates the appropriate item table, e.g. item_master, with the new merchandise hierarchy.
- If an item is reclassified, the product securities of the item are then updated.

Stored Procedures/Shared Modules (Maintainability)

FORECASTS_SQL.GET_SYSTEM_FORECAST_IND:

Stored PL/SQL procedure for returning the value of the forecast_ind from the SYSTEM_OPTIONS table.

FORECASTS_SQL.GET_DOMAIN:

Stored PL/SQL procedure for retrieving the domain for a merchandise hierarchy.

RECLASS_SQL.ITEM_PROCESS:

Stored PL/SQL procedure for updating or inserting records into the ITEM_MASTER, POS_MODS, and TRAN_DATA tables. If the item cannot be reclassified, the IO_RECLASS_FAILED variable returns a value of TRUE. Otherwise, it returns a value of FALSE. If reclassification failed, the function returns the reason for failure.

LOC_PROD_SECURITY_SQL.INSERT_USR_SEC:

Stored PL/SQL procedure for creating new records on to the SEC_USER_PROD_MATRIX table for all users that have security access to the new item.

ITEMLIST_MC_REJECTS_SQL.INSERT_REJECTS:

Stored PL/SQL procedure for inserting reclass failed reasons into the MC_REJECTIONS table.

Input Specifications

Select from: V_RESTART_RECLASS, RECLASS_ITEM, RECLASS_HEAD, ITEM_MASTER, PACKITEM, ORDSKU, DEAL_CALC_QUEUE, DEAL_ORDER_TEMP, SYSTEM_OPTIONS, and PERIOD

Output Specifications

'Table-To-Table'

Delete from: RECLASS_HEAD, RECLASS_ITEM, SEC_GROUP_PROD_MATRIX, and SEC_USER_PROD_MATRIX

Insert into: DEAL_ORDER_TEMP

Function Level Description

init()

This function initializes restart/recovery and fetches global options and variables. Calls size_arrays function.

process()

This is the main control function of the program. Each reclass_head record fetched from the driving cursor is checked for domain association to the new merchandise hierarchy if the forecast_ind is 'Y'. This existence check will be used during the call to the RECLASS_SQL.ITEM_PROCESS function to determine if the record will be rejected from reclassification. Calls the process_item function to perform the item reclassification. If reclassification failed, the insert_reject_record and delete_reclass_item functions are called. Otherwise, all order numbers associated to the item are inserted into the DEAL_ORDER_TEMP table by calling update_deal_calc_queue function for later processing during prepost. Also, reclassified items are deleted from the RECLASS_ITEM table and the product securities of the items are updated by calling the process_security function. After processing, the reclass_head record is then deleted from the RECLASS_HEAD table.

delete_reclass_head()

This function deletes the reclassification record from the RECLASS_HEAD table.

delete_reclass_item()

This function deletes the record from the RECLASS_ITEM table based on the row_id.

check_domain_exists()

If forecast_ind is 'Y', this function checks if a domain association to the new merchandise hierarchy exist for the given dept/class/subclass.

process_item()

This function calls the RECLASS_SQL.ITEM_PROCESS function to perform the item reclassification.

size_arrays()

This function sizes the fetched array to the commit size.

process_security()

This function deletes the records of the reclassified item in the SEC_GROUP_PROD_MATRIX and SEC_USER_PROD_MATRIX tables. Calls the LOC_PROD_SECURITY_SQL.INSERT_USR_SEC function.

get_order_numbers()

This function finds all the order numbers that are associated to the input item in the ORDSKU table.

update_deal_calc_queue()

This function is passed an array of order numbers and a long telling the number of order numbers in the array. This function then inserts the order numbers into the DEAL_ORDER_TEMP table, prepost post processing will select from this table and insert into DEAL_CALC_QUEUE table, along with 'N' for the recalc_all_ind and override_manual_ind columns.

order_exists()

This function checks to see whether or not the passed order number already exists in the DEAL_CALC_QUEUE or the DEAL_ORDER_TEMP table. A zero is returned if the order number is not in the DEAL_CALC_QUEUE OR in the DEAL_ORDER_TEMP table, a one if it is in the table, or a negative one if there was a SQL_ERROR found.

insert_reject_record()

This function calls the ITEMLIST_MC_REJECTS_SQL.INSERT_REJECTS function to insert the reclassification failed reason in the MC_REJECTIONS table.

final()

Standard Retek final function. Calls retek_close().

Scheduling Considerations

Schedule Information	Description
Processing Cycle	Daily
Scheduling Diagram	This module is scheduled to run daily during Phase 4 of the batch schedule. The requirement is for all the batch modules that update inventory and retail / cost prices to be completed before this module can run.
Pre-Processing	Prepost (reclsdly pre)
Post-Processing	Prepost (reclsdly post)
Threading Scheme	If desired, this program can be threaded by reclass_no to process several reclassifications at once.

Locking Strategy

N/A

Restart/Recovery

The logical unit of work for the reclassification is reclass_no/item. The restart commit counter will need to be carefully determined by each client according to the number of item/store combinations that will be affected by the reclassification since processing is done inside the packages rather than directly from the batch program. Large reclassifications with thousands of items held at many stores need smaller commit counters to avoid reprocessing large amounts of data in the event of program failure. Small reclassifications affecting just a few item/store combinations can have much larger commit counters since fewer rows will be inserted into the database each time an item is processed.

Driving cursor:

The main driving cursor is executed in prepost reclasdly pre function by calling RECLASS_SQL.PRE_PROCESS to perform contract validation before reclassification and inserts rejection message for the problem items. These can't be multithreaded so they are performed in the pre process. RECLASS_ITEM_TEMP and CONTRACT_UPDATE_TEMP tables are populated by this pre process. RECLASS_ITEM_TEMP is used as a driving table in RECLSDLY batch. CONTRACT_UPDATE_TEMP table is used in the POST_PROCESS function to update contract department. GTT_RECLASS_ITEM_TEMP and GTT_CONTRACT_UPDATE_TEMP global temporary tables are used to help the processing.

```
  SELECT ri_rowid,
         reclass_no,
         reclass_item,
         item,
         NVL(item_parent,' '),
         NVL(item_grandparent,' '),
         item_level,
         tran_level,
         new_dept,
         new_class,
         new_subclass,
         old_dept,
         old_class,
         old_subclass,
         new_division,
         new_group,
         old_division,
         old_group,
         NVL(reclass_validated,'N')
    FROM reclass_item_temp rit,
         v_restart_reclass rv
   WHERE rv.driver_value = rit.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 (rit.reclass_no > NVL(:ps_restart_reclass_no, -999) OR
            (rit.reclass_no = :ps_restart_reclass_no AND
             rit.reclass_item >= :ps_restart_item))
     ORDER BY 2,
              3;
```

Item Requisition Extraction [reqext]

Design Overview

Reqext (Item Requisition Extraction) handles automatic replenishment of items from warehouses to stores. It cycles through every item-store combination that is set to be reviewed on the current day, and calculates the quantity of the item that needs to be transferred to the store (if any). In addition, it distributes this *Recommended Order Quantity (ROQ)* over any applicable *alternate items* associated with the item. The program then takes this information and either creates new transfer line items or adds to existing ones.

Alternate items are either *simple packs* or *substitute items.* Simple packs are sellable and orderable packs that contain only a single item, such as a six-pack of cola or twelve-pack of socks. Substitute items are items predefined to be interchangeable with the item being replenished (referred to as the *master item*).

When an item is set up to use simple packs (designated by an indicator on the REPL_ITEM_LOC table), the ROQ must be distributed among these packs according to desirability. If a master item has no simple packs associated with it, it will be requested as itself. If there is only one pack associated with the item (referred to as the *primary simple pack*), then there is no distribution needed – the item will be transferred in this simple pack, since the cost per item for a pack is always less than that of an individual item. If multiple simple packs can be substituted for an item, then the distribution of the ROQ over these packs is determined by comparing the packs' relative sales history. ***Replenishing an item through multiple simple packs can have a severely negative effect on the performance of this program!*** Because the pack distribution depends on access to the huge sales history tables (ITEM_LOC_HIST), it is not recommended that many items be placed on replenishment through multiple simple packs. Whenever possible, it is better to assign a primary simple pack to the item, since this does not require distribution calculation.

If an item is not set up to use simple packs, the program will see if any substitute items are associated with it. If there are no substitute items associated with the master item, it will be transferred alone. If there are substitute items, they will be fetched into a list and the master item placed at either the head or tail end of the list, depending on the *fill priority* (set on the SUB_ITEMS_HEAD table). The priority determines which items are transferred first.

No matter what type of alternate items (if any) are used, the program will account for availability when building transfer line items. For simple packs, the share of ROQ allocated to each pack may be decreased or increased if the source warehouse has a shortage of some packs but a surplus of others. For substitute items, transfer quantities are prorated by calculating the ratio of total availability to total need, and items are transferred in order of priority until all need is filled or until no stock is available.

Once the transfer quantity of an item has been calculated, the transfer line item is posted to the database if **1)** the actual quantity to transfer is greater than zero, and **2)** the replenishment order control indicator for the item-store combination is either Automatic or Semi-Automatic. If it is Manual, a record will be written to another table (REPL_RESULTS) for reporting purposes. If the system-level All Replenishment Results indicator is set to "Yes", all line items will be written to REPL_RESULTS, *even if the quantity to order is zero.* Whenever a transfer line item is placed, the appropriate item-location table (ITEM_LOC_SOH) is updated to reflect the fact that stock is now reserved for transfer at the warehouse and expected at the store.

Scheduling Considerations

Schedule Information	Description
Processing Cycle	PHASE 3
Scheduling Diagram	Rplatupd, repladj, prepost ociroq and ociroq need to run before reqext so that all replenishment calculation attributes are up to date. Posupld need to run before reqext so that all stock information is up to date. Rplext should run after reqext, since the ROQ for a warehouse is influenced by any transfers created.
Pre-Processing	PREPOST REQEXT PRE - Create the TSFHEAD records for unique combination of Warehouse and Store, stock category and department.
Post-Processing	PREPOST REQEXT POST -Delete all the TSFHEAD records created those do not have the TSFDETAILS record created through replenishment. Update transfer status to approved.
Threading Scheme	DEPT

Restart Recovery

The logical unit of work is item, source warehouse. The driving cursor is ordered by item, source warehouse, order control indicator and simple pack indicator. When any of these values change during the course of processing (i.e., the current value is different than that of the previous record), then a transfer will be created, taking total quantities and availability into consideration (see replenish_item(), below).

Program Flow

N/A

Shared Modules

REPLENISHMENT_SQL.GET_STORE_REVIEW_TIME:

Stored PL/SQL procedure for calculating the time between scheduled shipments to a store from a warehouse. This time is used by GET_REPL_ORDER_QTY_SQL in its calculations.

ITEMLOC_QUANTITY_SQL.GET_WH_CURRENT_AVAIL:

Stored PL/SQL procedure for calculating the amount of a given item available at a given warehouse.

NEXT_TRANSFER_NUMBER:

Stored PL/SQL procedure used for getting the next valid transfer number for use in creating new transfers.

RMS_ROUND_TO_PACKSIZE:

Shared C function (see rpl.h) used in rounding an item's quantity up to the size of a simple pack, or for rounding an order quantity up to a receivable pack size.

Data Structures

repl_info_struct:

Holds information fetched from the driving cursor.

store_struct:

Holds information about item-location combinations, used for ROQ and distribution calculations.

alt_item_struct:

Holds information about alternate items associated with a given master item. Used in distribution calculations.

tsfhead_struct:

Used to buffer inserts to the TSFHEAD table.

tsfdetail_struct:

Used to buffer inserts and updates to the TSFDETAIL table.

item_loc_struct:

Used to buffer updates to the item-location tables (RAG_SKUS_ST, RAG_SKUS_WH, WIN_STORE, WIN_WH, PACKWH).

repl_results_struct:

Used to buffer inserts to the REPL_RESULTS table.

domain_struct:

Used to cache forecasting domain information.

Function Level Description

General Controlling Functions

main()

The standard Retek main function, this calls init(), process() and final(), and posts messages to the daily log files.

init()

Initializes the Restart-Recovery API and fetches system-level global variables.

driving_cursor()

Opens, fetches data from, or closes the driving cursor. This is a support function for process().

process()

This function fetches records from the driving cursor (driving_cursor()), passes them to replenish_item() to perform all appropriate actions, and commits work when appropriate (post_all(), restart_commit()).

replenish_item()

The controlling function for replenishment calculations. This function copies records out of the driving cursor buffer (copy_repl_to_store()), and calculates the ROQ for each record (get_repl_order_qty()). If a change in item, source warehouse, order control indicator, or simple pack indicator has occurred, the appropriate functions are called to calculate distribution of need over all appropriate alternate items and stores, and to place the transfers. If item's ROQ is zero or negative, no mater simple pack indicator is on the master item will be used for replenishment (build_pack_ratio(), calc_pack_dist(), calc_sub_dist()).

place_tsf_line_item()

This function takes a item-location combination and a transfer quantity, and actually builds the transfer line item (handle_tsf()). It then updates the item-location tables to reflect the change in stock (handle_item_loc()), and writes a record to the reporting table (handle_repl_results()) when appropriate.

final()

The standard Oracle Retail final function, this closes down the Restart-Recovery API.

Simple Pack Distribution and Transfer

build_pack_ratio()

Calculates distribution of the master item’s recommended order quantity (ROQ) over simple packs. Simple packs are sellable and orderable packs containing only a single item (e.g., six-pack of cola). Since the cost per item will always be less in a pack than singularly, the item will only be ordered in terms of simple packs (if any are applicable). This function tries to divide the total ROQ for the item among all applicable simple packs by using the packs' relative sales history to build a distribution 'mask' containing ratios used to calculate each pack's share of the ROQ. This mask is then adjusted to account for availability (shortages of some packs, surpluses of others).

This function performs the following steps to optimally distribute the ROQ among any and all simple packs:

- If a primary simple pack was defined for this item, that pack will be the only one used to supply the item (add_primary_pack()).
- If no primary pack was defined, the program will build a list of all simple packs associated with the item (get_multi_simple_pack()).
- If no appropriate simple packs are found, the item will be ordered as itself (add_single_item()).
- The historical sales for all simple packs and the master item are added up.
- The ROQ is distributed among the simple packs by taking the ratio of each pack’s historical sales to the total historical sales (first_ratio_pass()).
- If the first pass through the list did not account for the entire ROQ because of lack of availability for some packs, the program must keep cycling through the items until it has either distributed the ROQ among all available packs or there is simply no available stock left to supply the need (next_ratio_pass()).

first_ratio_pass()

Performs initial distribution of an item's ROQ among its associated simple packs. Calculates each pack's share as a ratio of its historical sales to the total historical sales (adjust_pack_ratio()). The historical sales of the master item are added to those of the simple pack with the lowest cost to give it a greater share of the ROQ. This is a support function for build_pack_ratio().

next_ratio_pass()

This function readjusts the ratios of still-available packs to try and cover the share of ROQ not yet allocated, still distributing the leftover ROQ proportionally by historical sales. This is a support function for build_pack_ratio().

adjust_pack_ratio()

Sets a simple pack's share of the ROQ to reflect its desirability (in terms of historical sales patterns), adjusting for availability. This is a support function for first_ratio_pass() and next_ratio_pass().

add_primary_pack()

If an item is flagged to have a primary simple pack, that pack is the only one that will be transferred. This function adds the primary pack to the simple pack distribution array and assigns it the full share of the ROQ. This is a support function for build_pack_ratio().

get_multi_simple_pack()

Finds all simple packs associated with a given master item and information about them (historical sales, availability, etc). This is a support function for build_pack_ratio().

get_single_sales_hist()

Gets the historical sales of the master item at all stores supplied by the given warehouse for use in calculating distribution among simple packs. Since the master item will only be transferred as a pack, this sales amount will be added to that of the pack with the lowest cost, increasing its share of the ROQ. This is a support function for build_pack_ratio().

add_single_item()

If an item is flagged to use simple packs, but none are found, it will be ordered as itself. This function adds the master item to the simple pack distribution structure and assigns it the full share of the ROQ. This is a support function for build_pack_ratio().

calc_pack_dist()

Once each simple pack's share of the item's ROQ has been calculated in build_pack_ratio(), this function calculates actual transfer quantities and places the transfer line items (place_tsf_line_item()). The function loops through each pack in the list, calculating the amount of the pack to transfer to each store (calc_pack_tsf_qty()). If the total transfer quantity of the pack exceeds its availability at the warehouse, each store will have its quantity reduced by one receivable pack until a reasonable number has been reached. Finally, a transfer line item is placed for the pack to the store.

calc_pack_tsf_qty()

Calculate the actual quantity to transfer for a store based on an alternate item's share of the ROQ at a store, adjusted for any applicable simple pack and/or shipping pack sizes. This is a support function for calc_pack_dist().

Substitute Item Distribution and Transfer

calc_sub_dist()

Calculate distribution of the ROQ over substitute items. Substitute items are items (selected by the user beforehand) that can be requested in place of a given item to cover situations where availability is too low or demand is too high.

After calling get_sub_items() to generate a list of appropriate items for transfer, the function loops through every item-location combination and performs the following steps to make sure that both need and availability are accounted for when placing transfers from the warehouse to the stores:

- If the total availability of all items in the substitute list cannot cover the full need over all stores, then the ratio of the total availability to the total ROQ is calculated. If total availability *can* cover total ROQ, the ratio is set to 1.
- The initial transfer quantity for the item at the location is calculated as the store's need adjusted by the availability ratio, and rounded up to a receivable pack size.
- If there is not enough of the item available at the warehouse to fill the calculated transfer quantity, the quantity will be decremented to an orderable amount.
- The transfer line item is placed by calling place_tsf_line_item().
- The store's ROQ, total ROQ, availability of the item, and total availability are all decremented by the amount just transferred to prepare for the next item-location's calculation.

get_sub_items()

Retrieves substitute items for the current master item and information about them from the database (receiving pack size, availability, etc.). If the fill priority for this set of items (SUB_ITEMS_HEAD.fill_priority) is set to 'M'aster, the master item will be the first one in the list, and will be used first to fill need. If it is set to 'S'ubstitute, the master item will be placed at the tail end of the list. This is a support function calc_sub_dist().

add_master()

Adds the master item to the appropriate position in the substitutes list. This is a support function for get_sub_items().

shift_subs()

If the fill priority for the substitutes list is set to 'M'aster, the master item must be placed at the head of the list. This function clears out the first position by moving each substitute item 'back' a slot. This is a support function for get_sub_items().

Database DML Handling

post_all()

The DML handling functions (handle_tsf(), handle_item_loc(), handle_repl_results()) normally only post information to the database tables when their respective buffers are full. When a commit point is reached, however, all buffers must be flushed to ensure restartability. This function forces all the buffers to be posted to the database.

handle_tsf()

Controls handling of inserts and updates to the Transfer tables.

add_tsfhead()

Deals with transfer header information. Either finds an appropriate transfer to add line items to (matching to/from locations, department and freight code), or creates a new one. passes back the transfer number for use in add_tsfdetail(). This is a supporting function for handle_tsf().

add_tsfdetail()

Deals with transfer detail information. Either finds an appropriate record on the TSFDETAIL table to add quantity to (matching transfer number and item), or creates a new one if none is found. This is a supporting function for handle_tsf().

get_next_seq_no()

Every line item on a transfer has a unique identifier within that transfer. This function gets the next sequence number for a new line item. This is a supporting function for add_tsfdetail().

post_tsf()

Posts transfer information to the database. Inserts to TSFHEAD, inserts and updates to TSFDETAIL. This is a supporting function for handle_tsf().

handle_item_loc()

Whenever a transfer is created or modified, the source location's reserved quantity and the receiving location's expected quantity must be adjusted to reflect the new stock status. This function controls the handling of updates to the RAG_SKUS_ST, RAG_SKUS_WH, WIN_STORE, WIN_WH and PACKWH tables.

add_item_loc()

Adds records to arrays for update of expected and reserved quantities on the item-location tables (RAG_SKUS_ST, RAG_SKUS_WH, WIN_STORE, WIN_WH, PACKWH) based on the appropriate item types. This is a support function for handle_item_loc().

post_item_loc()

Posts item-location stock status changes to the database (RAG_SKUS_ST, RAG_SKUS_WH, WIN_STORE, WIN_WH, PACKWH). This is a support function for handle_item_loc().

handle_repl_results()

Controls posting of report information to the REPL_RESULTS table.

add_repl_results()

Adds records to the replenishment results structure for reporting. This is a supporting function for handle_repl_results().

post_repl_results()

Posts replenishment information to the REPL_RESULTS table. This is a supporting function for handle_repl_results().

update_review_date()

Updates the last_review_date column on the REPL_ITEM_LOC table to reflect the fact that item-location combinations have just been evaluated.

PL/SQL Stored Procedure Calls

get_wh_current_avail()

Gets the available quantity of a given item at a given warehouse. This function is a wrapper for the ITEMLOC_QUANTITY_SQL.GET_WH_CURRENT_AVAIL stored PL/SQL procedure.

next_transfer_number()

Gets the next transfer number in the Oracle stored sequence for creating new transfer headers. This function is a wrapper for the NEXT_TRANSFER_NUMBER stored procedure.

Domain Validation

Domain validation is done in the ociroq.c batch program.

Support Functions

copy_repl_to_store()

Copies a record from the structure holding rows from the driving cursor into a structure holding item-location information for ROQ calculation, distribution, and transfer placement.

reset_store_struct()

Resets summary variables in a store information structure to prepare it for the next set of line items.

reset_alt_item_struct()

Resets summary variables in an alternate item structure to prepare it for the next set of alternates.

Array Sizing

size_repl_info_struct()

Allocates memory to the structure used to buffer fetches from the driving cursor.

size_store_struct()

Allocates memory to the structure used to hold item-location level information.

size_alt_item_struct()

Allocates memory to the structure used to hold information about alternate items (either simple packs or substitute items).

size_tsfhead_struct()

Allocates memory to the structure used to buffer inserts to the Transfer Header table.

size_tsfdetail_struct()

Allocates memory to structures used to buffer inserts and updates to the Transfer Detail table.

size_item_loc_struct()

Allocates memory to structures used to buffer updates of the item-location tables (RAG_SKUS_ST, RAG_SKUS_WH, WIN_STORE, WIN_WH, PACKWH).

size_repl_results_struct()

Allocates memory to the structure used to buffer inserts to the Replenishment Results table.

Database Interaction

Tables Selected From:

- RPL_NET_INVENTORY_TMP
- ITEM_SUPP_COUNTRY
- PACKHEAD
- PACKITEM
- PACKSTORE_HIST
- PERIOD
- RAG_SKUS_ST_HIST
- REPL_DAY
- REPL_ITEM_LOC
- STORE
- SUB_ITEMS_HEAD
- SUB_ITEMS_DETAIL
- SYSTEM_OPTIONS
- TSFDETAIL
- TSFHEAD
- WH

Tables Inserted To:

- REPL_RESULTS
- TSFDETAIL
- TSFHEAD

Tables Updated:

- ITEM_LOC_SOH
- REPL_ITEM_LOC
- TSFDETAIL

I/O Specification

N/A

Technical Issues

N/A