Oracle® Retail Merchandising System

Operations Guide Addendum

Release 10.1.22

September 2008

Oracle® Retail Merchandising System Operations Guide Addendum, Release 10.1.22

Value-Added Reseller (VAR) Language

(i) the software component known as **ACUMATE** developed and licensed by Lucent Technologies Inc. of Murray Hill, New Jersey, to Oracle and imbedded in the Oracle Retail Predictive Application Server – Enterprise Engine, Oracle Retail Category Management, Oracle Retail Item Planning, Oracle Retail Merchandise Financial Planning, Oracle Retail Advanced Inventory Planning and Oracle Retail Demand Forecasting applications.

(ii) the **MicroStrategy** Components developed and licensed by MicroStrategy Services Corporation (MicroStrategy) of McLean, Virginia to Oracle and imbedded in the MicroStrategy for Oracle Retail Data Warehouse and MicroStrategy for Oracle Retail Planning & Optimization applications.

(iii) the **SeeBeyond** component developed and licensed by Sun MicroSystems, Inc. (Sun) of Santa Clara, California, to Oracle and imbedded in the Oracle Retail Integration Bus application.

(iv) the **Wavelink** component developed and licensed by Wavelink Corporation (Wavelink) of Kirkland, Washington, to Oracle and imbedded in Oracle Retail Store Inventory Management.

(v) the software component known as **Crystal Enterprise Professional and/or Crystal Reports Professional** licensed by Business Objects Software Limited ("Business Objects") and imbedded in Oracle Retail Store Inventory Management.

(vi) the software component known as **Access Via™** licensed by Access Via of Seattle, Washington, and imbedded in Oracle Retail Signs and Oracle Retail Labels and Tags.

(vii) the software component known as **Adobe Flex™** licensed by Adobe Systems Incorporated of San Jose, California, and imbedded in Oracle Retail Promotion Planning & Optimization application.

(viii) the software component known as **Style Report™** developed and licensed by InetSoft Technology Corp. of Piscataway, New Jersey, to Oracle and imbedded in the Oracle Retail Value Chain Collaboration application.

(ix) the software component known as **WebLogic™** developed and licensed by BEA Systems, Inc. of San Jose, California, to Oracle and imbedded in the Oracle Retail Value Chain Collaboration application.

(x) the software component known as **DataBeacon™** developed and licensed by Cognos Incorporated of Ottawa, Ontario, Canada, to Oracle and imbedded in the Oracle Retail Value Chain Collaboration application.

Contents

Preface .... **ix**
Audience ix
Related Documents ix
Customer Support ix
Review Patch Documentation x
Oracle Retail Documentation on the Oracle Technology Network x
Conventions x

1 Introduction **1**

2 Batch Designs **3**
fifgldn1 (Financials General Ledger Download 1) 3
Design Overview 3
Scheduling Constraints 4
Pre/Post Logic Description 4
Restart Recovery 4
Driving Cursor 4
Functional Level Description 5
I/O Specification 5
fifgldn2 (Financials General Ledger Download 2) 5
Design Overview 5
Scheduling Constraints 6
Restart Recovery 6
Driving Cursor 6
Functional Level Description 8
I/O Specification 8
fifgldn3 (Financials General Ledger Download 3) 8
Design Overview 8
Scheduling Constraints 10
Restart Recovery 10
Driving Cursor 11
Program Flow 14
Functional Level Description 16
I/O Specification 16
nwppurge (End Of Year Inventory Position Purge) 16
Functional Area 16
Module Affected 16
Design Overview 16
Scheduling Constraints 16
Restart/Recovery 17
Locking Strategy 17
Security Considerations 17

Performance Considerations 17
Key Tables Affected 17
I/O Specification 17
nwpyearend (End of Year Inventory Position Snapshot) 17
Functional Area 17
Module Affected 17
Design Overview 17
Scheduling Constraints 17
Restart/Recovery 18
Locking Strategy 18
Security Considerations 18
Performance Considerations 18
Key Tables Affected 18
I/O Specification 18
salmaint (Stock Ledger Table Maintenance) 18
Functional Area 18
Module Affected 18
Design Overview 18
Scheduling Constraints 19
Restart/Recovery 19
Locking Strategy 19
Security Considerations 19
Performance Considerations 19
Key Tables Affected 19
I/O Specification 19
stkschedxpld (Scheduled Stock Count Explode) 20
Functional Area 20
Module Affected 20
Design Overview 20
Scheduling Constraints 20
Restart/Recovery 20
Locking Strategy 21
Security Considerations 21
Performance Considerations 21
Key Tables Affected 21
Shared Modules 21
I/O Specification 21
vrplbld (Vendor Replenished Order Build) 22
Design Overview 22
Scheduling Constraints 23
Restart Recovery 23
Program Flow 25
Shared Modules 25

Function Level Description 26
I/O Specification..29

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 10.1.22 documentation set:

- Oracle Retail Merchandising System Release Notes
- Oracle Retail Merchandising System Installation Guide

Customer Support

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

For a base release (".0" release, such as 12.0), Oracle Retail strongly recommends that you read all patch documentation before you begin installation procedures. Patch documentation can contain critical information related to the base release, based on new information 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:

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

- fifgldnld1 (General Ledger Interface)
- fifgldnld2 (General Ledger Interface)
- fifgldnld3 (General Ledger Interface)
- nwppurge (End of Year Inventory Position Purge)
- nwpyearend (End of Year Inventory Position Snapshot)
- salmaint (Stock Ledger Table Maintenance)
- stkschedxpld (Scheduled Stock Count Explode)
- vrplbld (Vendor replenished order build)

For more specific information regarding enhancements and modifications made to the previous Oracle Retail Merchandising System release, see the *Oracle Retail Merchandising System 10.1.22 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.

fifgldn1 (Financials General Ledger Download 1)

Design Overview

This process extracts detailed stock ledger information for certain transaction types on a daily basis in order to bridge the information to an interfaced financial application. The detailed information retains certain reference information, such as stock adjustment reason codes, which are lost during the normal daily roll-up process (saldly.pc).

This process reads from the if_tran_data table, instead of the actual tran_data table itself. Hence, it requires the 'PRE-SALDLY' and 'PRE-FIFGLDN1' run of prepost.pc to have occurred prior to its initiation. It can run in parallel to fifglnd2.pc.

Codes Table Information

Concerning GL data, there are two factors that need to be addressed:

- Some transaction types/amount types need to be bridged at a low level to contain detailed reference information, while others do not.
- There are only certain transaction types/amount types that need to be bridged for a given client. For instance, net sales at retail may be needed for one client, while net sales cost by be needed by another.

This process should allow the flexibility to bridge the required information without coding efforts. Since users can manipulate the codes detail table through a dialogue, this would be a good tool to allow this maintenance.

The code type that is created for this process is the General Ledger – Single Transaction 'GLST' type. This includes transaction types/amount types to go through the SKU-level detailed processing if the feed to GL is to be daily. The code for this is a three-character field, the first two characters determining the transaction code, the third character determining if the cost or retail component of the amount is to be bridged ('C' for Cost, 'R' for Retail, and 'B' for Both). For instance, if the code 01R is on the table, the net sales retail will be bridged at the skudetail level.

There are two other codes types in the total GL interface: General Ledger – Rolled Transaction 'GLRT' and General Ledger – Monthly Transaction 'GLMT'. These are part of the FIFGLDN2 and FIFGLDN3 designs.

Note that there are several risks in the maintenance of these codes types:

- The codes table is not masked, and one could incorrectly enter an R12 instead of 12R, etc.

  This results in not sending the code 12 GL transactions.

- '12R' could be entered in more than one codes type. This results in duplicate processing, although at different levels of roll-up. There is no cross-checking to verify that a code entered in one code type is not in the other.

Daily-Level GL Feed

The fifgldn1.pc program reads the if_tran_data table for each transaction type/amount type in the codes table, and posts it to a Oracle Retail general ledger staging table (stg_fif_gl_data) at the SKU detail level, complete with reference fields. These reference fields are bridged to the financial application for segregation of stock adjustment codes, mapping transfers to transfer numbers, etc., depending on the client need.

The fifgldn1.pc program can run in parallel to the saldly process, but must run after the if_tran_data load process, as it processes rows on if_tran_data.

The local currency is retrieved for all locations from their corresponding store/warehouse rows.

Scheduling Constraints

N/A

Pre/Post Logic Description

Processing Cycle: Daily, possibly in parallel to fifgldn2.pc, after PREPOST (PRE-SALDLY) and PREPOST (PRE-FIFGLDN1).

Pre-Processing:

Prepost for PRE-SALDLY. Prepost for PRE-FIFGLDN1 (this creates indexes)

Post-Processing

Threading scheme: Based on store/wh.

Restart Recovery

Logical Unit of Work (recommended Commit check points)

Driving Cursor

This program is fully restartable. The LUW is based upon dept, class, and subclass.

```
Select distinct
Td.sku,
Td.Dept,
Td.Class,
Td.Subclass,
Td.Store,
Td.Wh,
Td.Tran_date,
Td.Tran_code,
Td.Adj_code,
Td.Total_cost,
Td.Total_retail,
Td.Ref_no_1,
Td.Ref_no_2,
St.currency_code,
Wh.currency_code,
cd.code
from if_tran_data td,
code_detail cd,
store st,
wh
v_restart_store_wh
where (cd.code_type = 'GLST')
and (substr(cd.code, 1, 2) = td.tran_code)
and (td.store = st.store (+)
and td.wh = wh.wh (+) )
and v.driver_name = :os_restart_driver_name
and v.driver_value = :td.store
and v.num_threads = :oi_restart_num_threads
and v.thread_val = :oi_restart_thread_val
and (td.dept > NVL(:ora_restart_dept, td.dept-1) OR
(td.dept = :ora_restart_dept AND
(td.class > :ora_restart_class OR
(td.class = :ora_restart_class AND
(td_subclass > :ora_restart_subclass)))))
order by dept,
class,
subclass;
```

Functional Level Description

All database interactions required and error handling considerations. All required stock ledger data must be processed to the GL interface table within Oracle Retail. Errors are handled in the normal batch method.

I/O Specification

N/A

fifgldn2 (Financials General Ledger Download 2)

Design Overview

This process summarizes stock ledger data from the transaction staging table if_tran_data, based on the level of information required, and posts it to a staging table within Oracle Retail for transfer to a financial application's general ledger.

This process reads from the if_tran_data table, instead of the actual tran_data table itself. Hence, it requires the 'PRE-SALDLY' and 'PRE-FIFGLDN1' run of prepost.pc to have occurred prior to its initiation. It can run in parallel to fifglnd1.pc.

Codes Table Information

The code type that is created for this process is the General Ledger – Rolled Transaction 'GLRT' type. This includes transaction types/amount types to go through the roll-up processing if the feed to GL is to be daily. The code for this is a three-character field, the first two characters determining the transaction code, the third character determining if the cost or retail component of the amount is to be bridged ('C' for Cost, 'R' for Retail, and 'B' for Both). For instance, if the code 01R is on the table, the net sales retail is bridged at the skudetail level.

Note that there are several risks in the maintenance of these codes types:

- The codes table is not masked, and one could incorrectly enter an R12 instead of 12R, etc. This results in not sending the code 12 GL transactions.

- ‘12R’ could be entered in more than one codes type. This results in duplicate processing, although at different levels of roll-up. There is no cross-checking to verify that a code entered in one code type is not in the other.

Daily-Level GL Feed

- The fifgldn2.pc program reads and summarizes the if_tran_data table for all transaction types/amount types in the codes table, and posts it to an Oracle Retail general ledger staging table (stg_fif_gl_data) at the department, class, or subclass level.
- The fifgldn2.pc program can run in parallel to the saldly process, as well as the fifgldn1.pc process, but must run after the if_tran_data load process, as it processes rows on if_tran_data.
- The local currency is retrieved for all locations from their corresponding store/warehouse rows.

Scheduling Constraints

Schedule Information	Description
Processing Cycle	Daily, possibly in parallel to saldly, after PREPOST(‘pre-saldly’ and ‘prefifgldn1’)
Scheduling Diagram	N/A
Pre-Processing	N/A
Post-Processing	N/A
Threading Scheme	store/wh

Restart Recovery

Logical Unit of Work (recommended Commit check points)

Driving Cursor

This program is fully restartable. It uses three different driving cursors, based on the input parameter (DEPT, CLASS, SUBCLASS).

If ‘DEPT’ is the input parameter, the following cursor is used:

```
Select td.Dept,
-1,
-1,
td.Store,
td.Wh,
td.tran_date,
Td.Tran_code,
sum(Td.Total_cost),
sum(Td.Total_retail),
cd.code,
';'||to_char(td.dept)
from if_tran_data td,
code_detail cd,
v_restart_store_wh v
where (cd.code_type = 'GLRT')
and (substr(cd.code,1,2)=td.tran_code)
and v.driver_name = :os_restart_driver_name
and v.driver_value = :td.store
and v.num_threads = :oi_restart_num_threads
and v.thread_val = :oi_restart_thread_val
and td.dept > NVL(:ora_restart_dept, td.dept-1)
group by td.store,
td.wh,
td.dept,
td.tran_code,
td.tran_date,
cd.code
order by td.dept
```

If 'CLASS' is the input parameter, the following cursor is used:

```
Select td.Dept,
Td.class,
-1,
td.Store,
td.Wh,
td.tran_date,
Td.Tran_code,
sum(Td.Total_cost),
sum(Td.Total_retail),
cd.code,
';'||to_char(td.dept)||';'||to_char(td.class)
from if_tran_data td,
code_detail cd,
v_restart_store_wh v
where (cd.code_type = 'GLRT')
and (substr(cd.code,1,2)=td.tran_code)
and v.driver_name = :os_restart_driver_name
and v.driver_value = :td.store
and v.num_threads = :oi_restart_num_threads
and v.thread_val = :oi_restart_thread_val
and (td.dept > NVL(:ora_restart_dept, td.dept-1) OR
(td.dept = :ora_restart_dept AND
(td.class > :ora_restart_class)))
group by td.store,
td.wh,
td.dept,
td.class,
td.tran_code,
td.tran_date,
cd.code
order by td.dept,
td.class
```

If 'SUBCLASS' is the input parameter, the following cursor is used:

```
Select td.Dept,
Td.class,
Td.subclass,
td.Store,
td.Wh,
td.tran_date,
Td.Tran_code,
sum(Td.Total_cost),
sum(Td.Total_retail),
cd.code,
';'||to_char(td.dept)||';'||to_char(td.class)||';'||to_char(td.subclass)
from if_tran_data td,
code_detail cd,
v_restart_store_wh v
where (cd.code_type = 'GLRT')
and (substr(cd.code,1,2)=td.tran_code)
and v.driver_name = :os_restart_driver_name
and v.driver_value = :td.store
and v.num_threads = :oi_restart_num_threads
and v.thread_val = :oi_restart_thread_val
and (td.dept > NVL(:ora_restart_dept, td.dept-1) OR
(td.dept = :ora_restart_dept AND
(td.class > :ora_restart_class OR
(td.class = :ora_restart_class AND
(td.subclass > :ora_restart_subclass)))))
group by td.store,
td.wh,
td.dept,
td.class,
td.subclass,
td.tran_code,
td.tran_date,
cd.code
order by td.dept,
td.class,
td.subclass
```

Functional Level Description

All required stock ledger data must be processed to the GL interface table within Oracle Retail. Errors are handled using the normal batch techniques.

I/O Specification

N/A

fifgldn3 (Financials General Ledger Download 3)

Design Overview

This process summarizes stock ledger data from the monthly stock ledger table month_data, based on the level of information required, and posts it to a staging table within Oracle Retail for transfer to a financial application's general ledger.

Since this process reads from the month_data table, it requires the sales monthly process SALMTH to have run. It can run any time after the close of the last day of the month.

Codes Table Information

The code type that has is used for this process is the General Ledger – Monthly Transaction 'GLMT' type. This includes transaction types/amount types to go through the roll-up processing if the feed to GL is to be monthly. The code for this is a three-character field. The first two characters determine the transaction code, the third character determines if the cost or retail component of the amount is to be bridged ('C' for Cost, 'R' for Retail, and 'B' for Both). For instance, if the code 01R is on the table, the net sales retail is bridged at the department, class, or subclass level.

Since the monthly amounts are no longer maintained as transaction codes, this process needs to map certain amounts back to tran_codes. Some amounts are calculated from existing amount fields, and are not present on the tran_data level. These amounts are given tran_codes specifically for bridging to a general ledger. The following list is the field mapping for the process:

Month_data field Tran

Code	Tran Code	Cost/Retail	Comment
OPN_STK_RETAIL	50	Retail	Monthly Calculated Figure
OPN_STK_COST	50	Cost	Monthly Calculated Figure
STOCK_ADJ_RETAIL	22	Retail	
STOCK_ADJ_COST	22	Cost	
PURCH_RETAIL	20	Retail	
PURCH_COST	20	Cost	
RTV_RETAIL	24	Retail	
RTV_COST	24	Cost	
FREIGHT_COST	26	Cost	
TSF_IN_RETAIL	30	Retail	
TSF_IN_COST	30	Cost	
TSF_OUT_RETAIL	32	Retail	
TSF_OUT_COST	32	Cost	
NET_SALES_RETAIL	01	Retail	
NET_SALES_RETAIL_EX_VAT	02	Retail	
NET_SALES_COST	01	Cost	
RETURNS_RETAIL	04	Retail	
RETURNS_COST	04	Cost	
MARKUP_RETAIL	11	Retail	
MARKUP_CAN_RETAIL	12	Retail	
CLEAR_MARKDOWN_RETAIL	16	Retail	
PERM_MARKDOWN_RETAIL	13	Retail	
PROM_MARKDOWN_RETAIL	15	Retail	
MARKDOWN_CAN_RETAIL	14	Retail	
SHRINKAGE_COST	51	Cost	Monthly Calculated Figure
SHRINKAGE_RETAIL	51	Retail	Monthly Calculated Figure
EMPL_DISC_RETAIL	60	Retail	
WORKROOM_AMT	80	Retail	
CASH_DISC_AMT	81	Retail	
CLS_STK_RETAIL	52	Retail	Monthly Calculated Figure
CLS_STK_COST	52	Cost	Monthly Calculated Figure
GROSS_MARGIN_AMT	53	Retail	Monthly Calculated Figure
COST_VARIANCE_AMT	70	Cost	
HTD_GAFS_RETAIL	54	Retail	Monthly Calculated Figure
HTD_GAFS_COST	54	Cost	Monthly Calculated Figure

Code	Tran Code	Cost/Retail	Comment
INTER_STOCKTAKE_SALES_AMT	55	Cost	Monthly Calculated Figure
INTER_STOCKTAKE_SHRINK_AMT	56	Cost	Monthly Calculated Figure
STOCKTAKE_MTD_SALES_AMT	57	Cost	Monthly Calculated Figure
STOCKTAKE_MTD_SHRINK_AMT	58	Cost	Monthly Calculated Figure
STOCKTAKE_BOOKSTK_RETAIL	59	Retail	Monthly Calculated Figure
STOCKTAKE_BOOKSTK_COST	59	Cost	Monthly Calculated Figure
STOCKTAKE_ACTSTK_RETAIL	61	Retail	Monthly Calculated Figure
STOCKTAKE_ACTSTK_COST	61	Cost	Monthly Calculated Figure
RECLASS_IN_COST	34	Cost	
RECLASS_IN_RETAIL	34	Retail	
RECLASS_OUT_COST	36	Cost	
RECLASS_OUT_RETAIL	36	Retail	

The tran_codes of 50-59 and 61-69 need to be reserved specifically for this interface.

Note that there are several risks in the maintenance of these codes types:

- The codes table is not masked, and one could incorrectly enter an R12 instead of 12R, etc. This results in not sending the code 12 GL transactions.
- '12R' could be entered in more than one codes type. This results in duplicate processing, although at different levels of roll-up or different processing times (daily versus monthly). There is no cross-checking to verify that a code entered in one code type is not in the other.

Monthly-level GL Feed

Processes:

- The program fifgldn3.pc reads and summarizes the month_data table for all transaction types/amount types in the codes table, and posts it to an Oracle Retail general ledger staging table (stg_fif_gl_data) at the department, class, or subclass level.
- The program fifgldn3.pc must run after the salmth process, as it processes rows on month_data. The local currency is retrieved for all locations from their corresponding store/warehouse rows.

Scheduling Constraints

Pre/Post Logic Description

Processing Cycle: Monthly, after salmth and sysdte have run.

Restart Recovery

Logical Unit of Work (recommended Commit check points)

Driving Cursor

There is one cursor for DEPT-level roll-up:

```
select md.dept,
-1,
-1,
md.store,
md.wh,
NVL(decode(cd.code, '50R',sum(md.opn_stk_retail),
'50C',sum(md.opn_stk_cost),
'22R',sum(md.stock_adj_retail),
'22C',sum(md.stock_adj_cost),
'20R',sum(md.purch_retail),
'20C',sum(md.purch_cost),
'24R',sum(md.rtv_retail),
'24C',sum(md.rtv_cost),
'26C',sum(md.freight_cost),
'30R',sum(md.tsf_in_retail),
'30C',sum(md.tsf_in_cost),
'32R',sum(md.tsf_out_retail),
'32C',sum(md.tsf_out_cost),
'01R',sum(md.net_sales_retail),
'02R',sum(md.net_sales_retail_ex_vat),
'01C',sum(md.net_sales_cost),
'04R',sum(md.returns_retail),
'04C',sum(md.returns_cost),
'11R',sum(md.markup_retail),
'12R',sum(md.markup_can_retail),
'16R',sum(md.clear_markdown_retail),
'13R',sum(md.perm_markdown_retail),
'15R',sum(md.prom_markdown_retail),
'14R',sum(md.markdown_can_retail),
'51C',sum(md.shrinkage_cost),
'51R',sum(md.shrinkage_retail),
'60R',sum(md.empl_disc_retail),
'80R',sum(md.workroom_amt),
'81R',sum(md.cash_disc_amt),
'52R',sum(md.cls_stk_retail),
'52C',sum(md.cls_stk_cost),
'53R',sum(md.gross_margin_amt),
'70C',sum(md.cost_variance_amt),
'54R',sum(md.htd_gafs_retail),
'54C',sum(md.htd_gafs_cost),
'55C',sum(md.inter_stocktake_sales_amt),
'56C',sum(md.inter_stocktake_shrink_amt),
'57C',sum(md.stocktake_mtd_sales_amt),
'58C',sum(md.stocktake_mtd_shrink_amt),
'59R',sum(md.stocktake_bookstk_retail),
'59C',sum(md.stocktake_bookstk_cost),
'61R',sum(md.stocktake_actstk_retail),
'61C',sum(md.stocktake_actstk_cost),
'34C',sum(md.reclass_in_cost),
'34R',sum(md.reclass_in_retail),
'36C',sum(md.reclass_out_cost),
'36R',sum(md.reclass_out_retail),
0),0),
cd.code,
';'||to_char(md.dept)
from month_data md,
v_restart_store_wh v,
code_detail cd
where cd.code_type = 'GLMT'
```

```
and md.half_no = :last_half_no
and md.month_no = :last_month_no
and md.currency_ind = 'L'
and v.driver_name = :os_restart_driver_name
and v.driver_value = md.store
and v.num_threads = :oi_restart_num_threads
and v.thread_val = :oi_restart_thread_val
and md.dept > NVL(:ora_restart_dept, md.dept - 1)
group by md.store,
md.wh,
md.dept,
cd.code
order by md.dept
```

Also, one for CLASS-level roll-up:

```
SELECT md.dept,
md.class,
-1,
md.store,
md.wh,
NVL(decode(cd.code, '50R',sum(md.opn_stk_retail),
'50C',sum(md.opn_stk_cost),
'22R',sum(md.stock_adj_retail),
'22C',sum(md.stock_adj_cost),
'20R',sum(md.purch_retail),
'20C',sum(md.purch_cost),
'24R',sum(md.rtv_retail),
'24C',sum(md.rtv_cost),
'26C',sum(md.freight_cost),
'30R',sum(md.tsf_in_retail),
'30C',sum(md.tsf_in_cost),
'32R',sum(md.tsf_out_retail),
'32C',sum(md.tsf_out_cost),
'01R',sum(md.net_sales_retail),
'02R',sum(md.net_sales_retail_ex_vat),
'01C',sum(md.net_sales_cost),
'04R',sum(md.returns_retail),
'04C',sum(md.returns_cost),
'11R',sum(md.markup_retail),
'12R',sum(md.markup_can_retail),
'16R',sum(md.clear_markdown_retail),
'13R',sum(md.perm_markdown_retail),
'15R',sum(md.prom_markdown_retail),
'14R',sum(md.markdown_can_retail),
'51C',sum(md.shrinkage_cost),
'51R',sum(md.shrinkage_retail),
'60R',sum(md.empl_disc_retail),
'80R',sum(md.workroom_amt),
'81R',sum(md.cash_disc_amt),
'52R',sum(md.cls_stk_retail),
'52C',sum(md.cls_stk_cost),
'53R',sum(md.gross_margin_amt),
'70C',sum(md.cost_variance_amt),
'54R',sum(md.htd_gafs_retail),
'54C',sum(md.htd_gafs_cost),
'55C',sum(md.inter_stocktake_sales_amt),
'56C',sum(md.inter_stocktake_shrink_amt),
'57C',sum(md.stocktake_mtd_sales_amt),
'58C',sum(md.stocktake_mtd_shrink_amt),
'59R',sum(md.stocktake_bookstk_retail),
'59C',sum(md.stocktake_bookstk_cost),
'61R',sum(md.stocktake_actstk_retail),
'61C',sum(md.stocktake_actstk_cost),
'34C',sum(md.reclass_in_cost),
'34R',sum(md.reclass_in_retail),
'36C',sum(md.reclass_out_cost),
'36R',sum(md.reclass_out_retail),
0),0),
cd.code,
';'||to_char(md.dept)||';'||to_char(md.class)
FROM month_data md,
v_restart_store_wh v,
code_detail cd
WHERE cd.code_type = 'GLMT'
AND md.half_no = :last_half_no
AND md.month_no = :last_month_no
AND md.currency_ind = 'L'
AND v.driver_name = :os_restart_driver_name
AND v.driver_value = md.store
AND v.num_threads = :oi_restart_num_threads
AND v.thread_val = :oi_restart_thread_val
AND (md.dept > NVL(:ora_restart_dept, md.dept - 1) OR
(md.dept = :ora_restart_dept and
(md.class > :ora_restart_class)))
GROUP BY md.store,
md.wh,
md.dept,
md.class,
cd.code
ORDER BY md.dept,
md.class
```

Lastly, there is one for SUBCLASS-level roll-up:

```
SELECT md.dept,
md.class,
md.subclass,
md.store,
md.wh,
NVL(decode(cd.code, '50R',sum(md.opn_stk_retail),
'50C',sum(md.opn_stk_cost),
'22R',sum(md.stock_adj_retail),
'22C',sum(md.stock_adj_cost),
'20R',sum(md.purch_retail),
'20C',sum(md.purch_cost),
'24R',sum(md.rtv_retail),
'24C',sum(md.rtv_cost),
'26C',sum(md.freight_cost),
'30R',sum(md.tsf_in_retail),
'30C',sum(md.tsf_in_cost),
'32R',sum(md.tsf_out_retail),
'32C',sum(md.tsf_out_cost),
'01R',sum(md.net_sales_retail),
'02R',sum(md.net_sales_retail_ex_vat),
'01C',sum(md.net_sales_cost),
'04R',sum(md.returns_retail),
'04C',sum(md.returns_cost),
'11R',sum(md.markup_retail),
'12R',sum(md.markup_can_retail),
'16R',sum(md.clear_markdown_retail),
'13R',sum(md.perm_markdown_retail),
'15R',sum(md.prom_markdown_retail),
'14R',sum(md.markdown_can_retail),
'51C',sum(md.shrinkage_cost),
'51R',sum(md.shrinkage_retail),
'60R',sum(md.empl_disc_retail),
'80R',sum(md.workroom_amt),
'81R',sum(md.cash_disc_amt),
'52R',sum(md.cls_stk_retail),
'52C',sum(md.cls_stk_cost),
'53R',sum(md.gross_margin_amt),
'70C',sum(md.cost_variance_amt),
'54R',sum(md.htd_gafs_retail),
'54C',sum(md.htd_gafs_cost),
'55C',sum(md.inter_stocktake_sales_amt),
'56C',sum(md.inter_stocktake_shrink_amt),
'57C',sum(md.stocktake_mtd_sales_amt),
'58C',sum(md.stocktake_mtd_shrink_amt),
'59R',sum(md.stocktake_bookstk_retail),
'59C',sum(md.stocktake_bookstk_cost),
'61R',sum(md.stocktake_actstk_retail),
'61C',sum(md.stocktake_actstk_cost),
'34C',sum(md.reclass_in_cost),
'34R',sum(md.reclass_in_retail),
'36C',sum(md.reclass_out_cost),
'36R',sum(md.reclass_out_retail),
0),0),
cd.code,
';'||to_char(md.dept)||';'||to_char(md.class)||';'||to_char(md.subclass)
FROM month_data md,
v_restart_store_wh v,
code_detail cd
WHERE cd.code_type = 'GLMT'
AND md.half_no = :last_half_no
AND md.month_no = :last_month_no
AND md.currency_ind = 'L'
AND v.driver_name = :os_restart_driver_name
AND v.driver_value = md.store
AND v.num_threads = :oi_restart_num_threads
AND v.thread_val = :oi_restart_thread_val
AND (md.dept > NVL(:ora_restart_dept, md.dept - 1) OR
(md.dept = :ora_restart_dept and
(md.class > :ora_restart_class or
(md.class = :ora_restart_class and
(md.subclass > :ora_restart_subclass)))))
GROUP BY md.store,
md.wh,
md.dept,
md.class,
md.subclass,
cd.code
ORDER BY md.dept,
md.class,
md.subclass
```

Program Flow

- This program takes the value of system_options.gl_rollup. There is a value of 'D' for department, 'C' for class, or 'S' for subclass.
- To facilitate the dept/class/subclass nature of this program, three integer global bind vars will be declared: Dept_rollup, class_rollup, and sclass_rollup. Depending on the system_options value described above, the appropriate variable is set to 1 and the others to 0 (zero). If/then calls such as "if (dept_rollup) { <statement> };" allow for easy readability of the code.
- In addition to the insert array structure (with fields mirroring the stg_fif_gl_data table structure), one structure (mirroring the **subclass** cursor field list) should be declared. Depending on the level of rollup desired, the class or class and subclass

fields are populated with '-1', but this method is cleaner than having three structures to track.

- stg_fif_gl_data.tran_date is always populated with system_variables.last_eom_date

init() -

- Initialize restart recovery.
- Set up array structures for the driving cursor fetch as well as one for inserts into the STG_FIF_GL_DATA table (dept, class, subclass, location, loc_type, tran_date, tran_code, adj_code, amount, cost_retail_flag, ref_no_1, ref_no_2, sku, currency_code in the structure).
- Normal array sizing should occur [size_arrays()]
- Get value from system_options (as described above). If it is a department level rollup, initialize only the ora_restart_dept var. If it's class, initialize only ora_restart_dept and ora_restart_class, etc.
- Get values from system_vars: last_eom_half_no, last_eom_month_no, & last_eom_date

process() -

- open the appropriate driving cursor
- WHILE(1)
- fetch g_l_restart_max_counter records from appropriate driving cursor into the array
- if NO_DATA_FOUND, this means there were fewer than g_l_restart_max_counter records fetched or none were fetched. Set a flag to eventually break from the loop.
- Calculate the num_records_to_process in the array by subtracting previously processed records from NUM_RECORDS_PROCESSED (this is an automatically kept variable that keeps how many records have been fetched from a cursor).
- If restart_string is NULL (happens on a fresh start only)
- Populate it with the first LUW information
- For 0 to < num_records_to_process, loop
- If LUW has NOT changed
- fill_array()
- ELSE
- Write_array() to stg_fif_gl_data
- Call restart_commit()
- Set restart_string to new LUW
- fill_array() for cost
- Add num_records_to_process to the previously processed records
- If loop breaking flag is set
- break
- END WHILE
- Call write_array() -- this will insert the final LUW from the insert_array

fill_array(array_index, cost_retail_flag) -

- Fetch currency code from store or wh table.
- write cost/retail depending on flag passed in
- increment array_size variable
- if array_size = g_l_restart_max then call resize_array()

write_array() -

- For insert_array_size records, an array insert into stg_fif_gl_data. If -1's are in class and/or subclass, insert them as normal.
- Set insert_array_size counter = 0.
- If there are insertion errors, the program should return fatally.

size_arrays() -

- Allocates the memory for the two array structures. (the fetch array and the insert array)

resize_array() -

- increase the size of the insert array by another g_l_restart_max slots

final() -

- Close off restart recovery processing

Functional Level Description

All database interactions required and error handling considerations

All required stock ledger data must be processed to the GL interface table within Oracle Retail. Errors with the download will require an abort.

I/O Specification

N/A

nwppurge (End Of Year Inventory Position Purge)

Functional Area

Stock Ledger

Module Affected

NWPPURGE.PC

Design Overview

This program purges the records from the table NWP after a certain amount of years have passed. The number of years is a configurable parameter setup in SYSTEM_OPTIONS.nwp_retention_period.

Scheduling Constraints

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

Restart/Recovery

Restart/recovery is not applicable, but the records will be committed based on the commit max counter setup in the restart control table.

Locking Strategy

N/A

Security Considerations

N/A

Performance Considerations

N/A

Key Tables Affected

Table	Select	Insert	Update	Delete
NWP	Yes	No	No	Yes

I/O Specification

NA

nwpyearend (End of Year Inventory Position Snapshot)

Functional Area

Stock count

Module Affected

NWPYEAREND.PC

Design Overview

This program takes a snapshot of the item's stock position and cost at the end of the year. When the end of year NWP snapshot process runs, it takes a snapshot of stock and weighted average cost (WAC) for every item/location combination currently holding stock. If there is not a record already on the NWP table for an item/location/year combination in the snapshot, a new record is added for that item/location/year combination.

Scheduling Constraints

Schedule Information	Description
Processing Cycle	Phase 4 (Yearly)
Scheduling Considerations	Needs to run on the last day of the year in phase 4.
Pre-Processing	N/A
Post-Processing	N/A

Schedule Information	Description
Threading Scheme	Multithreaded by store_wh

Restart/Recovery

The logical unit of work for this program is set at the location/item level. Threading is done by supplier using the v_restart_store_wh view to thread properly.

The commit_max_ctr field should be set to prevent excessive rollback space usage, and to reduce the overhead of file I/O. The changes are posted when the commit_max_ctr value is reached and the value of the counter is 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
NWP_FREEZE_DATE	Yes	No	No	No
ITEM_MASTER	Yes	No	No	No
NWP	Yes	Yes	Yes	No
ITEM_LOC_SOH	Yes	No	No	No

I/O Specification

N/A

salmaint (Stock Ledger Table Maintenance)

Functional Area

Stock Ledger

Module Affected

SALMAINT.PC

Design Overview

This module is run as either salmaint pre or salmaint post. The salmaint pre functionality adds partitions to the HALF_DATA, DAILY_DATA, WEEK_DATA and MONTH_DATA tables. The salmaint post functionality drops partitions or purges the above tables (if the table is not partitioned) for an old half.

Scheduling Constraints

Schedule Information	Description
Processing Cycle	Ad-Hoc
Scheduling Considerations	N/A
Pre-Processing	N/A
Post-Processing	N/A
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
SYSTEM_OPTIONS	Yes	No	No	No
SYSTEM_VARIABLES	Yes	No	No	No
HALF_DATA	No	No	No	Yes
DAILY_DATA	No	No	No	Yes
WEEK_DATA	No	No	No	Yes
MONTH_DATA	No	No	No	Yes

I/O Specification

N/A

stkschedxpld (Scheduled Stock Count Explode)

Functional Area

Stock Count

Module Affected

STKSCHEDXPLD.PC

Design Overview

This batch program (STKSCHEDXPLD.PC) is used to create the scheduled stock counts for the location. It finds all the stock count schedules, which are set for the location using the SYSTEM_OPTIONS.STAKE_REVIEW_DAYS. The schedule can be set to fire on daily basis or else the user can specify the days (Sunday, Monday, and so on) on which the stock count can be created. In essence, the users specify the cycles such as “every third Monday” and “every second Tuesday and Thursday.”

If the count is a Unit Only Count, then the item list is specified in the detail record. In this case, the item list is exploded out to the SKU, and every SKU is added to the count/item/location tables. The SKU which belongs to the item list in a department/class/subclass is added to the count/item/location tables.

If the count is a unit and amount count, then the department/class/subclass is fully exploded out to the subclass level. All the department/class/subclass combinations are added to the count/location/subclass tables.

If the location is a location list, then the list is exploded out to the locations before the detail record is processed.

If the location is a warehouse, then the already existing warehouse is not added to current count and also the processing is moved to the next location.

If the location is a store and any simple pack exists for an item list on the count, then the simple pack and the item list is added for the count. If the simple pack is on the count, then its component SKU is also added along with any other simple packs containing the SKU which is not already in the count.

Scheduling Constraints

Schedule Information	Description
Processing Cycle	PHASE 0 - Daily
Scheduling Considerations	Run before STKXPLD.PC.
Pre-Processing	N/A
Post-Processing	N/A
Threading Scheme	Multi-threaded by location (store and warehouse).

Restart/Recovery

The logical unit of work for this module is schedule, location. The changes are posted when the commit_max_ctr value is reached.

Locking Strategy

N/A

Security Considerations

N/A

Performance Considerations

N/A

Key Tables Affected

Table	Select	Insert	Update	Delete
STAKE_SCHEDULE	Yes	No	Yes	No
V_RESTART_STORE_WH	Yes	No	No	No
PERIOD	Yes	No	No	No
CODE_DETAIL	Yes	No	No	No
STAKE_HEAD	No	Yes	No	No
STAKE_LOCATION	No	Yes	No	No
STAKE_PRODUCT	No	Yes	No	No
STAKE_PROD_LOC	No	Yes	No	No
STAKE_SKU_LOC	Yes	Yes	No	No
ITEM_MASTER	Yes	No	No	No
DEPS	Yes	No	No	No
SUBCLASS	Yes	No	No	No
PACKITEM	Yes	No	No	No
ITEM_LOC	Yes	No	No	No
SKULIST_DETAIL	Yes	No	No	No
LOC_LIST_DETAIL	Yes	No	No	No
LOCATION_CLOSED	Yes	No	No	No
COMPANY_CLOSED	Yes	No	No	No

Shared Modules

N/A

I/O Specification

N/A

vrplbld (Vendor Replenished Order Build)

Design Overview

Table	Index	Select	Insert	Update	Delete
EDI_ORD_TEMP	No	Yes	No	No	No
ORDHEAD	Yes	Yes	Yes	No	No
ORDSKU	No	No	Yes	Yes	No
ORDLOC	No	No	Yes	No	No
PERIOD	Yes	Yes	No	No	No
STORE	Yes	Yes	No	No	No
WH	Yes	Yes	No	No	No
UNIT_OPTIONS	No	Yes	No	No	No
ITEM_SUPP_COUNTRY	Yes	Yes	No	No	No
SUPP_IMPORT_ATTR	Yes	Yes	No	No	No
SUPS	Yes	Yes	No	No	No
ORDSKU_HTS	Yes	Yes	Yes	No	No
SYSTEM_OPTIONS	No	Yes	No	No	No
DEAL_CALC_QUEUE	No	Yes	Yes	Yes	No
ORD_INV_MGMT	No	No	Yes	No	No

Note: ORDHEAD (select via index) is referenced via the function ORDER_NUMBER_SQL.NEXT_ORDER_NUMBER.

Indexes: ORDHEAD(order_no),
ORDSKU(order_no,item)

Function: To continue the process started by the program ediupack of building Oracle Retail orders that reflect the vendor-generated orders as received through the EDI 855.

The program ediupack.pc processes the EDI 855's that have been received from vendors. Rows have been inserted onto the table EDI_ORD_TEMP for use by this program. The EDI_ORD_TEMP table contains all items which were included on the EDI 855, along with the vendor order number with which they were originally associated.

The items are then consolidated so that one Oracle Retail order is created for each vendor-order-number/supplier.

Some details for each item are retrieved from the item/location tables and ITEM_SUPPLIER_COUNTRY and stored in ORDSKU and ORDLOC so that subsequent changes to the item do not affect the outstanding order.

A_order_amt is updated on the OTB table. If the supplier is foreign and OTB is calculated at cost, the added amount is total_duty + total_cost. If the supplier is not foreign and OTB is calculated at cost, the added amount is total_cost. Finally, if OTB is calculated at retail, the added amount is total_retail.

In records written to the ORDHEAD table:

- FREIGHT_TERMS is given the value fetched from SUPS.FREIGHT_TERMS

- ORIG_APPROVAL_DATE is assigned the value of the PERIOD.VDATE
- ORIG_APPROVAL_ID is assigned the value 'EDI855'

If an item is ordered for a location at which it does not have an item/location record, a record is built by calling the stored procedure NEW_ITEM_LOC.

For cross-dock VMI PO creation, it allows having multiple shipping locations for same item/supplier combination. Among multiple shipping locations, one is source warehouse and the other can be store or warehouse, which will be sourced from source warehouse. The quantity given for source warehouse in TSHIP record should be greater than or equal to sum of quantities given in TSHIP record for cross docked locations.

The vendor-order-number is used as the order number if it is a valid RMS pre-issued order number for the supplier. If the vendor-order-number is not a valid RMS pre-issued order number then the order number is allocated automatically on a 'next available' basis.

Note: The vendor minimum order quantity is not considered when writing to the order tables because the quantity is determined by vendor. Thus, this quantity is not stored in the edi_ord_temp table.

Re-run: If this program aborts and terminates normally or terminates abnormally restart after setting restart_flag = 'Y' on restart_program_status table for current restart_name, schema, and thread_val.

Scheduling Constraints

Schedule Information	Description
Processing Cycle	PHASE 3
Scheduling Diagram	N/A
Pre-Processing	N/A
Post-Processing	Vrplbld_post in prepost.pc truncates edi_ord_temp table
Threading Scheme	SUPPLIER

Restart Recovery

The logical unit of work (LUW) for vrplbld.pc is a count of records pulled from the edi_temp_ord table. The number of records is determined by the commit_max_ctr field on the restart_control table. Further, one or more of the key values (vendor_order_no, supplier, dept) must change before a commit event can take place. This ensures that only whole order is saved, that is, that a commit never takes place before all of the order detail records have been processed.

The commit_max_ctr field should be set to prevent excessive rollback space usage. Given the use of multiple threading, the recommended commit counter setting is 5000 records (subject to change based on experimentation).

Two cursors are defined, but only one cursor is opened for fetching (the criteria are based on whether or not departmental level ordering is allowed.)

```
EXEC SQL DECLARE c_items_d CURSOR FOR
    SELECT isc.origin_country_id,
           isc.supp_pack_size,
           NVL(isc.lead_time, 0),
           sia.agent,
           sia.lading_port,
           sia.discharge_port,
           s.currency_code,
           e.vendor_order_no,
           e.supplier,
           e.dept,
           e.wh_or_store_c,
           e.wh_or_store,
           e.item,
           e.item_desc,
           e.ref_item,
           e.unit_retail,
           DECODE(:edi_cost_override_ind, 'Y', iscl.unit_cost, e.unit_cost),
           e.qty_ordered,
           TO_CHAR(e.written_date,   'YYYYMMDD'),
           TO_CHAR(e.not_before_date,'YYYYMMDD'),
           TO_CHAR(e.not_after_date, 'YYYYMMDD'),
           TO_CHAR(e.not_after_date, 'DD'),
           TO_CHAR(e.not_after_date, 'MM'),
           TO_CHAR(e.not_after_date, 'YYYY'),
           ';' || e.vendor_order_no ||
           ';' || TO_CHAR(e.dept)   ||
           ';' || TO_CHAR(e.supplier)
      FROM item_supp_country isc,
           item_supp_country_loc iscl,
           sup_import_attr sia,
           sups s,
           edi_ord_temp e,
           v_restart_supplier rv
     WHERE isc.item               = e.item
       AND isc.primary_country_ind = 'Y'
       AND isc.supplier           = e.supplier
       AND sia.supplier (+)       = e.supplier
       AND s.supplier             = e.supplier
       AND rv.driver_value        = e.supplier
       AND rv.driver_name         = :os_restart_driver_name
       AND rv.num_threads         = :oi_restart_num_threads
       AND rv.thread_val          = :oi_restart_thread_val
       AND (e.vendor_order_no > NVL(:os_restart_vndr_ord_no,-999) OR
              (e.vendor_order_no = :os_restart_vndr_ord_no AND
                 (e.dept > :os_restart_dept OR
                    (e.dept = :os_restart_dept AND
                       e.supplier > :os_restart_supplier)
                 )
              )
           )
  ORDER BY e.vendor_order_no,
           e.dept,
           e.supplier,
           e.wh_or_store_c,
           e.wh_or_store,
           e.item;


EXEC SQL DECLARE c_items_s CURSOR FOR
    SELECT isc.origin_country_id,
           isc.supp_pack_size,
           NVL(isc.lead_time, 0),
           sia.agent,
           sia.lading_port,
           sia.discharge_port,
           s.currency_code,
           e.vendor_order_no,
           e.supplier,
           e.dept,
           e.wh_or_store_c,
           e.wh_or_store,
           e.item,
           e.item_desc,
           e.ref_item,
           e.unit_retail,
           DECODE(:edi_cost_override_ind, ‘Y’, iscl.unit_cost, e.unit_cost),
           e.qty_ordered,
           TO_CHAR(e.written_date,   'YYYYMMDD'),
           TO_CHAR(e.not_before_date,'YYYYMMDD'),
           TO_CHAR(e.not_after_date, 'YYYYMMDD'),
           TO_CHAR(e.not_after_date, 'DD'),
           TO_CHAR(e.not_after_date, 'MM'),
           TO_CHAR(e.not_after_date, 'YYYY'),
           ';' || e.vendor_order_no ||
           ';' || TO_CHAR(e.dept)   ||
           ';' || TO_CHAR(e.supplier)
      FROM item_supp_country isc,
           item_supp_country_loc iscl,
           sup_import_attr sia,
           sups s,
           edi_ord_temp e,
           v_restart_supplier rv
     WHERE isc.item               = e.item
       AND isc.primary_country_ind = 'Y'
       AND isc.supplier           = e.supplier
       AND sia.supplier (+)       = e.supplier
       AND s.supplier             = e.supplier
       AND rv.driver_value  = e.supplier
       AND rv.driver_name  = :os_restart_driver_name
       AND rv.num_threads  = :oi_restart_num_threads
       AND rv.thread_val   = :oi_restart_thread_val
       AND (e.vendor_order_no > NVL(:os_restart_vndr_ord_no,-999) OR
              (e.vendor_order_no = :os_restart_vndr_ord_no AND
                 e.supplier > :os_restart_supplier)
           )
  ORDER BY e.vendor_order_no,
           e.supplier,
           e.wh_or_store_c,
           e.wh_or_store,
           e.item;
```

Program Flow

N/A

Shared Modules

- NEW_ITEM_LOC
- ORDER_NUMBER_SQL.GET_NEXT_ORDER_NUMBER
- CURRENCY_SQL.GET_RATE
- CAL_TO_454_LDOW
- ORDER_SETUP_SQL.DEFAULT_ORDHEAD_DOCS
- ORDER_SETUP_SQL.DEFAULT_ORDSKU_DOCS
- ORDBA_CALC_SQL.ORDER_COST
- ORDER_EXPENSE_SQL.INSERT_COST_COMP
- ORDER_HTS_SQL.DEFAULT_CALC_HTS

- OTB_SQL.ORD_APPROVE
- ORDER_SETUP_SQL.DEFAULT_ORDER_INV_MGMT_INFO
- ORDER_ATTRIB_SQL.MULTIPLE_LOCS_EXIST
- SUP_INV_MGMT_SQL.GET_PURCHASE_PICKUP

Function Level Description

init()

- select department level orders flag from unit options
- select base country, latest ship days, FOB information, ELC indicator, EDI cost override inidicator, calendar 454 indicator, bill-to location, and multi-channel indicator from system options
- select vdate from period
- call restart initialization logic

process()

- call open_cursor to open appropriate cursor (department level or multi-department orders)
- call fetch_cursor for prime fetch of driving cursor
- while
  - if supplier or vendor order number changed then call create_header
  - if item changed call create_item
  - if multi-channel = ‘N’ or the location fetched is a store call add_loc
  - if multi-channel is on and the location is a warehouse, then distribute:
    - call dist_init
    - call dist_set_distribution_rule
    - for every replenishable warehouse
      - call dist_expected_quantity
      - call dist_distributed_quantity
      - call add_loc
    - call dist_final
  - call add_loc
  - call fetch_cursor
    - if supplier or vendor order number changed then
      - if the ELC indicator is Y, call add_cost_comp
      - call item_defaults
        - call apply_disc
      - call header_defaults
      - call update_header
      - call update_order
      - call check_vmi_order

        - else if the item changed
            - if the ELC indicator is Y, call add_cost_comp
            - call item_defaults
    - call restart commit logic
- end while
- call item_defaults
- call apply_disc
- call header_defaults
- call update_header
- call update_order
- call check_vmi_order

open _cursor()

- open appropriate cursor depending on what type of orders are allowed.

fetch _cursor()

- fetch records from appropriate cursor

get_min_max()

- finds the minimum and maximum lead times for the current order

item_active ()

- returns 1 if item is active (0 if inactive) on item_loc

create_header()

- call function to determine if vendor-order-number is a valid pre-issued order number (ORDER_NUMBER_SQL.CHECK_ORDER_PREISSUE)
- If not pre-issued order number call function to get the next order number (ORDER_NUMBER_SQL.NEXT_ORDER_NUMBER)
- call CAL_TO_454_LDOW procedure to convert order's not_after_date to the end-of-week date to update otb_eow_date field
- call CURRENCY_SQL.GET_RATE to get the exchange rate for the supplier's currency.
- insert record into ordhead

header_defaults()

- Call ORDER_ATTRIB_SQL.MULTIPLE_LOCS_EXIST to retrieve the location if the order is a single location order. Call ORDER_SETUP_SQL.DEFAULT_ORDHEAD_DOCS to default the necessary documents for the order header. Also call ORDER_SETUP_SQL.
- DEFAULT_ORDER_INV_MGMT to appropriate supplier inventory management information to the order with NULL inserted for the scaling constraints, since a vendor generated order is not eligible for scaling. If no values are found for the purchase type or pickup location, call SUP_INV_MGMT_SQL.GET_PURCHASE_PICKUP to retrieve the default values. If no value is found for the pickup date, use the longest lead-time to calculate the pickup date.

update_header()

- Check to see if any items on the order come from a country other than the ordhead import country. If so, update the import_order_ind to 'Y' on ordhead.

update_order()

- Update the Order Header table with the following values
  - Purchase Type
  - Pickup Location (only enter if the Purchase Type is 'FOB' or 'BACK')
  - Pickup Date (only enter if the Purchase Type is 'FOB' or 'BACK') – **if value is later than the Not After Date, set equal to the Not After Date**
  - Earliest Ship Date – minimum of Order/Item ESDs
  - Latest Ship Date – maximum of Order/Item LSDs
- Update Order Item table. Set the pickup location if the purchase type is 'FOB' or 'BACK'.

item_defaults()

- Call ORDER_SETUP_SQL.DEFAULT_ORDSKU_DOCS to default necessary documents for items in order.

apply_disc()

- if edi_cost_override_ind = 'Y' then insert a record into the DEAL_CALC_QUEUE table if the current order number is not in the table, or update the order_appr_ind field to 'Y' if the order number is already in the table.
- call update_OTB

add_cost_comp()

- Call ORDER_EXPENSE_SQL.INSERT_COST_COMP, ORDER_HTS_SQL.DEFAULT_CALC_HTS and ELC_CALC_SQL.CALC_COMP.

create_item ()

- if edi_cost_override_ind = 'Y' insert record into ordsku with the cost_source = 'MANL'. If edi_cost_override_ind != 'Y', insert a record into ordsku with the cost_source = 'NORM';

add_loc ()

- insert into ordloc

set_worksheet_status()

- For a given order number and reject code, update the status on the ordhead table to 'W'orksheet and update the reject code to whatever was passed in.

order_exists()

- For a given order number, check to see whether or not the order number is on the DEAL_CALC_QUEUE table.

final()

- Call restart/recovery close logic.

I/O Specification

N/A