Retail passion. Innovative solutions.

Retek® Merchandising System™ 11.0.3

Operations Guide Addendum

Corporate Headquarters:

Retek Inc.
Retek on the Mall
950 Nicollet Mall
Minneapolis, MN 55403
USA

888.61.RETEK (toll free US)
Switchboard:
+1 612 587 5000

Fax:
+1 612 587 5100

European Headquarters:

Retek
110 Wigmore Street
London
W1U 3RW
United Kingdom

Switchboard:
+44 (0)20 7563 4600

Sales Enquiries:
+44 (0)20 7563 46 46

Fax:
+44 (0)20 7563 46 10

Customer Support

Customer Support hours

Customer Support is available 7x24x365 via email, phone, and Web access.

Depending on the Support option chosen by a particular client (Standard, Plus, or Premium), the times that certain services are delivered may be restricted. Severity 1 (Critical) issues are addressed on a 7x24 basis and receive continuous attention until resolved, for all clients on active maintenance. Retek customers on active maintenance agreements may contact a global Customer Support representative in accordance with contract terms in one of the following ways.

Contact Method	Contact Information
E-mail	support@retek.com
Internet (ROCS)	rocs.retek.com Retek's secure client Web site to update and view issues
Phone	+1 612 587 5800

Toll free alternatives are also available in various regions of the world:

Australia	+1 800 555 923 (AU-Telstra) or +1 800 000 562 (AU-Optus)
France	0800 90 91 66
Hong Kong	800 96 4262
Korea	00 308 13 1342
United Kingdom	0800 917 2863
United States	+1 800 61 RETEK or 800 617 3835
Mail	Retek Customer Support Retek on the Mall 950 Nicollet Mall Minneapolis, MN 55403

When contacting Customer Support, please provide:

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

Contents

Change made to the batch schedule and Volume 4

The batch schedule and Volume 4 of the Operations Guide have been updated in conjunction with this release. All references to the batch programs below have been removed because these batch programs are no longer part of RMS.

- slocrbld
- sprdrbld
- szonrbld

Sales History Rollup by Department, Class, and Subclass [hstbld]

Design Overview

The sales history rollup routine will extract sales history information for each item from the item_master, and item_loc_hist tables. The history information will be rolled up to the subclass, class, and dept level to be written to: dept_sales_hist, class_sales_hist, and subclass_sales_hist.

For each item, data to be saved includes sales qty, value, gross profit, and sales rate. This data must be collected from several tables including item_master, item_loc_hist and mask_rebuild. Letting the database (server) roll up the totals verse using a loop on the client enhances speed. Using a VIEW that contains all item information for the current week enhances simplicity. Data can then be summed from this single view instead of having to join across all 3 tables in a single select statement.

The rebuild program can be run in one of two ways:

First, if the program is run with a run-time parameter of 'rebuild', the program will read data (dept, class, and subclass) off the manually input mask_rebuild table, which will determine what is rebuilt. This process is used after items are reclassified from one merchandise hierarchy to another. Rebuilding a department will rebuild each class and subclass within the department, thus, only one row is required on mask rebuild for the department. This type of rebuilding process will rebuild data from all dates on the item_master, item_loc_hist table, rolling them to the department, class, and subclass level.

Second, if the program is run with a run-time parameter of 'weekly', the program will build sales information for all dept/class/subclass combinations only for the current end of week date.

mask_rebuild table:

DEPT	CLASS	SUBCLASS	
X	NULL	NULL	Rebuild Department
X	X	NULL	Rebuild Class
X	X	X	Rebuild Subclass

TABLE	INDEX	SELECT	INSERT	UPDATE	DELETE
DEPT_SALES_HIST	No	Yes	Yes	Yes	No
CLASS_SALES_HIST	No	Yes	Yes	Yes	No
SUBCLASS_SALES_ HIST	No	Yes	Yes	Yes	No
ITEM_MASTER	No	Yes	No	No	No
ITEM_LOC_HIST	No	Yes	No	No	No
MASK_REBUILD	No	Yes	No	No	No

TABLE	INDEX	SELECT	INSERT	UPDATE	DELETE
PERIOD	No	Yes	No	No	No
SYSTEM_VARIABLES	No	Yes	No	No	No

Scheduling Constraints

Processing Cycle: PHASE 3 (weekly)
PHASE AD-HOC (weekly)

Scheduling Diagram: Must run after complete weekly sales have been updated by posupld. Also should be re-run on demand when a sales rollup request has been given for a given dept, class or subclass

Pre-Processing: N/A

Post-Processing: hstbld_post()
Truncates the mask rebuild table.

Threading Scheme: DEPT

Restart Recovery:

If program is run for reclassifying Items (first run time parameter = ‘rebuild’), the driving cursor for the program will be:

```
EXEC SQL DECLARE c_rebuild_eow CURSOR FOR
   SELECT im.dept,
          im.class,
          im.subclass,
          ilh.loc,
          to_char(ilh.eow_date,'YYYYMMDD'),
          ilh.week_454,
          ilh.month_454,
          ilh.year_454,
          ilh.sales_type,
          NVL(SUM(NVL(ilh.sales_issues,0)),0),
          NVL(SUM(NVL(ilh.value,0)),0),
          NVL(SUM(NVL(ilh.gp,0)),0)
     FROM item_master im,
          item_loc_hist ilh,
          v_restart_store rs
    WHERE im.item = ilh.item
      AND ilh.sales_type in ('P', 'R', 'C')
      AND ilh.eow_date = to_date(:ps_vdate,'YYYYMMDD')
      AND ilh.loc_type = 'S'
```

```
      AND rs.driver_value = ilh.loc
      AND rs.num_threads  = TO_NUMBER(:ps_num_threads)
      AND rs.thread_val   = TO_NUMBER(:ps_thread_val)
      AND (ilh.loc >= NVL(:ps_restart_store, -999) AND
                  (im.dept >= NVL(:ps_restart_dept,-999) AND
                      NVL(im.class,0) > NVL(:ps_restart_class,-999)
                   )
                )
GROUP BY im.dept,
         im.class,
         im.subclass,
         ilh.loc,
         to_char(ilh.eow_date,'YYYYMMDD'),
         ilh.week_454,
         ilh.month_454,
         ilh.year_454,
         ilh.sales_type
ORDER BY ilh.loc,
         im.dept,
         im.class,
         im.subclass,
         ilh.sales_type;
```

If program is run for current end of week (second run time parameter = ‘weekly’), the driving cursor for the program will be:

```
EXEC SQL DECLARE c_rebuild_dept CURSOR FOR
   SELECT im.dept,
          im.class,
          im.subclass,
          ilh.loc,
          to_char(ilh.eow_date,'YYYYMMDD'),
          ilh.week_454,
          ilh.month_454,
          ilh.year_454,
          ilh.sales_type,
          NVL(SUM(NVL(ilh.sales_issues,0)),0),
          NVL(SUM(NVL(ilh.value,0)),0),
          NVL(SUM(NVL(ilh.gp,0)),0)
```

```
      FROM item_master  im,
           item_loc_hist  ilh,
           hist_rebuild_mask hrm,
           v_restart_store rs
     WHERE im.item = ilh.item
       AND im.dept = hrm.dept
       AND ilh.sales_type in ('P', 'R', 'C')
       AND ilh.loc_type = 'S'
       AND rs.driver_value = ilh.loc
       AND rs.num_threads  = TO_NUMBER(:ps_num_threads)
       AND rs.thread_val   = TO_NUMBER(:ps_thread_val)
       AND (hrm.class IS NULL
            OR (im.class = hrm.class
                AND (hrm.subclass IS NULL
                     OR  im.subclass = hrm.subclass)))
       AND (ilh.loc >= NVL(:ps_restart_store, -999) AND
                     (im.dept >= NVL(:ps_restart_dept,-999) AND
                           NVL(im.class,0) > NVL(:ps_restart_class,-
999)
                      )
                )
  GROUP BY im.dept,
      im.class,
      im.subclass,
      ilh.loc,
      to_char(ilh.eow_date,'YYYYMMDD'),
      ilh.week_454,
      ilh.month_454,
      ilh.year_454,
      ilh.sales_type
  ORDER BY ilh.loc,
           im.dept,
           im.class,
           im.subclass,
           to_char(ilh.eow_date, 'YYYYMMDD'),
           ilh.sales_type;
```

Program Flow

N/A

Shared Modules

N/A

Function Level Description

init()

- Initialize restart recovery
- If processing current end of week, call check_eow_date()
- Initialize structure arrays for subclass_sales_hist insert, subclass_sales_hist update, class_sales_hist insert, class_sales_hist update, and sales_history fetch.
- Check_eow_date()
- Check that vdate is a valid end of week date

process()

- Open driving cursor
- Array fetch cursor
- Loop through array to process records.
- Increment dept and class sales variables for running totals of sales, value, gp, and forecast_sales.
- Call process_subclass().
- If dept/class/store/eow_date/sales_type changes, call process_class().
- If dept/store/eow_date/sales_type changes, call process_dept().
- Call process_subclass(), process_class(), and process_subclass() to process last record fetched (last record is not processed within above loop.
- Call insert_class()
- Call update_class()
- Call insert_subclass()
- Call update_subclass

Process_subclass()

- Fetch previous sales, value, gp, and forecast_sales value from subclass_sales_hist
- If fetch is not found (no record exists), add values to subclass_insert array
- If fetch is found,
  - Add values to subclass_update array
  - Increment delta variables to hold running totals for difference in subclass' sales and updated sales, value and updated value, gp and updated gp, forecast_sales and updated forecast_sales.

Process_class()

- Fetch rowid from class_sales_hist
- If fetch is not found, add values to class_insert array
- If fetch is found,
- Add values to class_update array
- Increment delta variables to hold running totals for difference in class' sales, value, gp, and forecast_sales by adding deltas from subclass' sales, value, gp, and forecast_sales.
- Reset subclass delta variables and class running total variables.

Process_dept()

- Perform update of dept_sales_hist
- If update is not found (record does not exist), perform insert into dept_sales_hist
- Reset class delta variables and dept running total variables.

Insert_class()

- Perform array insert of class_sales_hist.

Update_class()

- Perform array update of class_sales_hist.

Insert_subclass()

- Perform array insert of subclass_sales_hist.

Update_subclass()

- Perform array update of subclass_sales_hist.

I/O Specification

N/A

Technical Issues

N/A

Upload customs tariff files [htsupld]

Design Overview

This batch program will be run whenever an updated US customs tariff file is available (probably twice a year) to upload HTS tariff information from the file into RMS HTS tables. The program will handle both the initial HTS information load as well as mid-year HTS updates that are supplied by the US government. The initial upload is handled by inserting information from the file into the tables; updating information already in the tables is handled by adjusting the effective dates of the existing HTS records and inserting a new set of HTS records into the tables.

Updating HTS records should follow the following guidelines:

- No HTS records with the same HTS and import country should have overlapping effect_from and effect_to dates. Import country is passed as an input parameter to the program, so that the program can support different import countries.
- The new HTS effective dates will never chop up the effective dates of an existing HTS, and there will never be any rollback in dates. Therefore, a new HTS can only start in the middle of an existing HTS or cover a completely different time frame after the existing HTS.
- When loading a new HTS that starts in the middle of an existing HTS, the effect_to date of the existing HTS should be adjusted to one day before the new effect_from date.
- No existing HTS information should be purged by the program. It's the client's responsibility to handle that.

Tables Affected:

TABLE	SELECT	INSERT	UPDATE	DELETE
HTS	Yes	Yes	Yes	Yes
HTS_TAX	No	Yes	Yes	Yes
HTS_FEE	No	Yes	Yes	Yes
HTS_OGA	No	Yes	Yes	Yes
HTS_TARIFF_TREATMENT	Yes	Yes	Yes	Yes
HTS_TT_EXCLUSIONS	No	Yes	Yes	Yes
TARIFF_TREATMENT	Yes	No	No	No
COUNTRY_TARIFF_TREATMENT	Yes	No	No	No
HTS_CHAPTER	Yes	No	No	No
OGA	Yes	No	No	No
UOM_CLASS	Yes	No	No	No
CODE_DETAIL	Yes	No	No	No
QUOTA_CATEGORY	Yes	No	No	No
COUNTRY	Yes	No	No	No

TABLE	SELECT	INSERT	UPDATE	DELETE
HTS_CVD	No	No	Yes	No
HTS_AD	No	No	Yes	No
HTS_REFERENCE	No	No	Yes	No
ITEM_HTS	Yes	Yes	Yes	No
ITEM_HTS_ASSESS	No	No	Yes	No
ORDSKU_HTS	Yes	Yes	Yes	Yes
MOD_ORDER_ITEM_HTS	No	Yes	No	No
PERIOD	Yes	No	No	No
SYSTEM_OPTIONS	Yes	No	No	No
DUAL	Yes	No	No	No
ORDSKU_HTS_ASSESS	No	No	No	Yes
ORDHEAD	Yes	No	No	No
ORDLOC	Yes	No	No	No
ORDSKU	Yes	No	No	No
CE_CHARGES	Yes	No	No	Yes
CE_ORD_ITEM	Yes	No	No	No
ITEM_SUPP_COUNTRY	Yes	No	No	No
ORDSKU_TEMP	Yes	No	No	No

Stored Procedures / Shared Modules (Maintainability)

ITEM_HTS_SQL.DELETE_ASSESS – given the item, hts, import_country_id, origin_country_id, effect_to and effect_from, this function deletes the corresponding record from item_hts_assess.

ITEM_HTS_SQL.DEFAULT_CALC_ASSESS – given the item, hts, import_country_id, origin_country_id, effect_to and effect_from, this function inserts into item_hts_assess, it also will potentially call other package functions and update other tables.

LC_SQL.DELETE_LCORDAPP – given the order_no, this function deletes from lc_ordapply table.

OTB_SQL.ORD_UNAPPROVE – given the order_no, this function updates the otb table.

ITEM_ATTRIB_SQL.GET_STANDARD_UOM – given the item_no, item_type and indicator, this function returns the standard_uom, standard_class, and conv_factor.

UOM_SQL.CONVERT – given the to_uom, from_value, from_uom, item, supplier and origin_country, this function returns the to_value.

SQL_LIB.BATCH_MSG – returns error message information.

ORDER_HTS_SQL.DELETE_ASSESS -- given the order_no and seq_no, this function deletes from the ordsku_hts_assess table.

ORDER_HTS_SQL.DEFAULT_CALC_ASSESS -- given the order_no, seq_no, pack or item, hts, import_country_id, origin_country_id, effect_to and effect_from, this function inserts into ordsku_hts_assess, it also will potentially call other package functions and update other tables.

CE_CHARGES_SQL.INSERT_COMPS – given the ce_id, vessel_id, voyage_flt_ind, order_no, item, pack_item, hts, import_country_id, effect_from, effect_to, cvb_code this function inserts into the ce_charges table.

Function Level Description

main()

- Standard Retek main function. This program takes in four parameters: userid/passwd, input file, reject file, import country id.

init()

- A global variable is used to hold the import country id that is passed in as a program input parameter. Call check_country to make sure that import country exists on the COUNTRY table; return with fatal error if not. It is used as the import country throughout the program.
- Open input file for read and open reject file for write.
- Call retek_init() for restart/recovery initialization.
- If it is a fresh start, call retek_get_record to read the FHEAD line into the fhead structure.
- Fetch vdate from period table.
- Fetch max_item from hts table and max ct from ordsku_hts and max ct from ce_charges.
- Fetch update_item_hts_ind and update_order_hts_ind from the system_options table
- Call check_spi to make sure that ‘C1’ and ‘C2’ exist in the TARIFF_TREATMENT table as SPI’s. ‘C1’ and ‘C2’ are default tariff treatments for every HTS. Return with fatal error if not.

file_process()

- Call function retek_get_record in a while loop to read the THEAD line into the thead structure:
  - if the record type returned is ‘FTAIL’, exit the loop;
  - set a save point.
  - If the record type returned is ‘THEAD’, read the THEAD line into the thead
  - structure that contains V1, V2, V3, V4 fields. The V4 record is not currently used in RMS/RTM.
  - If the record type returned is other than ‘FTAIL’ or ‘THEAD’, give a fatal error (wrong record type).
  - Call function process_THEAD to further process data contained in the THEAD. Set process error flag to indicate non-fatal process error.
  - Call function retek_get_record in a while loop to read the TDETL
  - line into the tdetl structure:
    - if the record type returned is ‘TTAIL’, exit to the outer loop to continue reading THEAD records if any exists;

        - if the record type returned is other than ‘TDETL’ or ‘TTAIL’, give a fatal error (wrong record type).
        - Call function process_TDETL to further process data contained in the TDETL.
    - Set process error flag to indicate non-fatal process error.
    - If update_item_hts_ind = “Y”,
        - If tran_code is “A” or “R”, call item_hts_update function. “A” stands for Update only and “R” stands for Replace. In both of these cases (as opposed to the other possibility of “D” for Delete) item tables will need to be updated.
    - If update_order_hts_ind = “Y”, call ordsku_hts_search function.
    - If process error flag is set. Rollback database process to the save point. Write rejected records to the reject file.
    - Call restart_force_commit to perform intermittent commit for restart/recovery.

process_THEAD()

- Fill the hts_keys structure with data from THEAD.
- After filling in the hts_keys, verify that effect_from < effect_to date. If not, reject the record right away. Call valid_all_numeric function to check effect_from, and effect_to field. If invalid reject the record. This function processes the information in V1, V2 and V3 records based on the transaction code ("A","R", "D") in the V1 record. It compares the new effective dates against those of any existing HTS records with the same HTS code and import country.
- If the transaction code type is ‘A’, insert a record into the HTS table; if the transaction code type is ‘R’, update the HTS record that has the same HTS code, import country id, effect_from and effect_to dates
- For transaction code "A":
    - If new HTS covers a time period different than and after any existing HTS, or no HTS exists for the given HTS/import country, is a valid record for inserting.
    - If new the HTS record is overlapping with existing record and its effect_from date > existing record and effect_to >= existing effect_to date, it is a valid record. Process is as follows:
        1. Insert an HTS record with the same data as the existing overlapping HTS, except that the effect_to date should be 1 day before the effect_from date of the new HTS record.
        2. Update the effect_to date of all corresponding child records to 1 day before the effect_from date of the new HTS record.
        3. Insert new hts to the related tables.

Detailed technical description:

- Call function validate_hts_update to verify that the record is valid for insert/update to the database or reject to the reject file. For the valid record call hts_child_update function to prepare child table processing.
- Call hts_table_insert function to insert record to the hts table. if any invalid information exists, write to error file.

- Call hts_oga_insert function to insert record/s to the hts_oga table. if any invalid information exists, write to error file.
- Call hts_spi_insert function to insert record/s to the hts_tariff_treatment table. if any invalid information exists, write to error file.
- Call hts_gsp_insert function to insert record/s to the hts_tt_exclusions table. if any invalid information exists, write to error message log file.
- Set process error flag if non fatal error occurs. Return error flag.
- For transaction code "R":
  1. Search for the HTS with the same HTS, import country id, effect_from and effect_to dates. If no record found, reject the record.
  2. If a record is found, delete the following child table records with the same HTS, import country id, effect_from and effect_to dates.
- Insert to update the HTS table and re-insert child table information from the input file.

Detailed Techincal Description:

- Call function search_hts_update to find record that can be updated in the database tables.
- If one exists, prepare child tables for processing.
- Call hts_table_insert function to insert record to the hts table. if any invalid information exists, write to error file.
- Call hts_oga_insert function to insert record/s to the hts_oga table. if any invalid information exists, write to error file.
- Call hts_spi_insert function to insert record/s to the hts_tariff_treatment table. if any invalid information exists, write to error file.
- Call hts_gsp_insert function to insert record/s to the hts_tt_exclusions table. if any invalid information exist, write to error message log file.
- Set process error flag if non fatal error occurs. Return error flag.
- For transaction code "D":
  - Seach for the HTS with same HTS, import country id , effect_from and effect_to dates.
  - If a record is found update HTS and all its child records to yesterday.

Note: Since the dates are still presented in 2-digit year in the 99 tape, we assume that the year coming in as 00-49 means 2000-2049, and 50-99 means 1950-1999. The customs uses ‘999999’ to mean Dec 31 st , 2039.)

Detailed Technical Description:

- Call function search_hts_reset to find updateable record in the hts table. If one exists, insert new hts record.
- Call function hts_child_update to update all the child records, then delete the existing hts record.

validate_hts_update()

- Call out c_hts_date_invalid cursor to select HTS records which starts before or on the same day as any existing HTS, or starts after and ends before any existing HTS:

  effect_from >= new effect_from OR

  effect_from < new effect_from and effect_to > new effect_to

  If record exists:

  - Call out c_hts_date_invalid2 cursor to select HTS records which starts before any existing HTS and ends on Dec 31 st , 2039.

  If record is not found, Write the record to the reject file, write an error message to the message log file, and return to the calling function with a non-fatal error.

  Else, set indictor =true (so that the existing record will be truncated to end 1 day before new HTS starts).

- New HTS starts after and overlaps with an existing HTS: effect_from < new effect_from and effect_to >= new effect_from or new HTS starts after old end date and therefore does not overlap at all. The ranges are completely separate.

  This is a valid record, and a most likely scenario. Fetch the effect_from and effect_to of the existing HTS. Insert a new record with effect_from date same as existing overlapping hts record and effect_to date is 1 day before the new effect_from date to hts table.

- Call function hts_child_update function to update effect_to date of all child records to 1 day before the new effect_from date.

- Delete the old record from hts table.

search_hts_update()

- Search for the HTS with the same HTS, import country id, effect_from and effect_to dates. If no record found, reject the record.

- If a record is found, delete the following child table records with the same HTS, import country id, effect_from and effect_to dates:

  HTS_TT_EXCLUSIONS

  HTS_TARIFF_TREATMENT

  HTS_OGA

  HTS_TAX

  HTS_FEE

  

  Note: HTS table record cannot be deleted due to the other child tables on HTS: ITEM_HTS, ITEM_HTS_ASSESS, ORDSKU_HTS, HTS_CVD, HTS_AD, HTS_REFERENCE, HTS_CHAPTER. The information on these tables won't be loaded in the HTS upload process.

search_hts_reset()

- Search for the HTS with the same HTS, import country id, effect_from and effect_to dates. If no record found, reject the record.
- Insert into HTS, all the same information, but inserting yesterday as the new to_date.
- If a record is found, call hts_child_update function to update the records in the child tables with effect_to date to yesterday:

hts_child_update()

This function updates the effect_to date of the existing overlapping HTS record on child tables. Since the child tables have referential constraints on the effective dates of the parent table HTS.

- Update the effect_to date of all corresponding child records to 1 day before the effect_from date of the new HTS record.

The following child tables should be updated:

- HTS_TARIFF_TREATMENT
- HTS_TT_EXCLUSIONS
- HTS_AD
- HTS_CVD
- HTS_OGA
- HTS_REFERENCE
- HTS_TAX
- HTS_FEE
- ITEM_HTS
- ITEM_HTS_ASSESS
- ORDSKU_HTS
- CE_CHARGES

Note: Since table HTS_TT_EXCLUSIONS has a foreign key on the effect_to date of table HTS_TARIFF_TREATMENT, we cannot update the effect_to date of HTS_TARIFF_TREATMENT directly. Likewise, insert an HTS_TARIFF_TREATMENT record with the new effect_to date first; then update the effect_to date of the HTS_TT_EXCLUSIONS table; at the end delete the HTS_TARIFF_TREATMENT record with the original effect_to date.

- Call delete_ord_temp_tables and pass in the value “-1” because thereis no known order_no at this point.

item_hts_update()

- Call size_item_array function to allocate space for the items
- Fetch item, origin_country_id and status from item_hts into struct
- If no data found, call free_itemlist and go to the next record. If data is found, Loop
  - If tran_code = “A” the item will need to be inserted with the same data as the fetched record but with new effect_to and effect_from dates.
    1 Insert dates into item_hts.
    2 Delete old record from item_hts
    3 Call the package SQL Delete assess to delete the old records from item_hts_assess.
  - If tran_code = “R”
    1 Call SQL Delete_assess to delete the old records from item_hts_assess
  - Call SQL Default_calc_assess to update the item_hts_assess table (ie insert record with new dates and recalculate)
  - Call ECL_CALC_SQL.CALC_COMP to recalculate expenses based on new assesses.
  - Insert into mod_order_item_hts a new record with same data but new dates.
  - Call free_itemlist

ordsku_hts_search()

- Call size_ord_array function to allocate space for the order information
- Fetch values from ordhead, ordsku_hts, ordsku and ordloc into struct (all necessary values to be able to do a complete insert into the mod_order_item, ordsku_hts, and ordsku_hts_assess tables.
- If no data found, call free_ordlist and go to next record. If data is found, Loop
  - If order status = “A”, (the order needs to be updated) set status from approved back to worksheet by calling SQL functions (LC_SQL.DELETE_LCORDAPP and OTB_SQL.ORD_UNAPPROVE).
  - Insert into mod_order_item_hts table (just the order_no and indicator set to ‘Y’)
  - Call ordsku_hts_update
  - Call free_ordlist

ordsku_hts_update()

- Call size_ce_array to allocate space for the custom entry information
  - Fetch custom entry values from ce_ord_item, ce_head, item_supp_country into struct
  - If no data found, call ordhts_update. If data is found:
  - If CE status = “W”, (worksheet status)

    1 Call ordhts_update

    2 Loop for each custom entry record
    Call ce_update
  - If status != “W” then the quantity cleared will need to be compared to the total quantity. In order to do that they will need to be converted to the standard uom format

    1 Loop
    Call uom_convert to get the total quantity.
    If total_qty < qty_ordered
    Call ordhts_update
  - Call free_ceordlist

ordhts_update()

- If tran_code = “A” or “D”
  - Delete old record (record with old dates) from ordsku_hts_assess
  - Delete old record (record with old dates) from ordsku_hts
  - Insert record with new dates into ordsku_hts
- Else if tran_code = “A”
  - Insert record with new dates into ordsku_hts
- Else if tran_code = “D”
  - Call SQL Delete_assess by calling order_del_assess function
  - Call SQL calc_comp
  - If the item is a pack item check to see if a record already exists on mod_order_item_hts – if it does not, insert one with the pack_item
  - If it is not a pack item, insert with item_no into mod_order_item_hts.
  - Return 0
- Else if tran_code = “R”
  - Call delete_ord_temp_tables and pass in the order_no.
- Call SQL Delete_assess by calling order_del_assess function
- Call ORDER_HTS_SQL.DEFAULT_CALC_ASSESS with either the pack_no or item_no depending on if it is a pack or not.
- Call ELC_CALC.CALC_COMP
- If it is a pack item insert into mod_order_hts with the pack_no
- If it is not a pack item, insert into mod_order_item_hts with the item_no

ce_update()

- Delete from ce_charges.
- If it is a “D”, call CE_CHARGES_SQL.INSERT_COMPS

hts_table_insert()

Before inserting into or updating the HTS table,

- Call function check_chapter to make sure that the chapter already exists on the
- HTS_CHAPTER table. If not, reject the record;
- Call check valid_all_numeric function to check unit for all numeric value.
- Call function check_uom to make sure that the UOMs (UOM1, UOM2, UOM3) already exist on the UOM_CLASS table. Reject the record if UOM does not exist.
- Call function check_duty to make sure that the duty code already exists on the
- CODE_DETAIL table. If not, reject the record.
- Call valid_all_numeric function to verify that the quota is all numeric. Then calling function check_quota to make sure that the quota category already exists on the QUOTA_CATEGORY table. If not, reject the record.

  Update the existing hts record with the updated hts_desc, chapter, units, units_1, units_2, units_3, duty_comp_code, more_hts_ind, quota_cat, quota_ind, ad_ind, cvd_ind.

Insert the following into the HTS table:

- hts: tariff number (V1c)
- import_country_id: import country from the program input parameter
- effect_from: begin effective date (V1e)
- effect_to: end effective date (V1f)
- hts_desc: commodity description (V1l)
- chapter: 1 st 4 (leftmost) digits of tariff number
- units: number of reporting units (V1g)
- units_1: first unit of measure (V1h) (If the number of reporting units is zero, this should be defaulted to ‘X’)
- units_2: second unit of measure (V1I) –NULL if not given
- units_3: third unit of measure (V1j)—NULL if not given
- duty_comp_code: duty code (V1k)
- more_hts:Y if additional tariff indicator (V2j is ‘R’, N otherwise
- quota_cat: category number (V3h) but only if quota indicator (V3g) is 1
- quota_ind ‘Y’ if there is a quota,’N’ otherwise
- ad_ind ‘Y’ if the anti-dumping flag (V3f) is 1, N otherwise
- cvd_ind ‘Y’if the countervailing duty flag (V2k) is 1, N otherwise

hts_oga_insert()

For each OGA code, call function check_oga to verify that the OGA code exists on the OGA table. If not, reject the record; otherwise, call hts_oga_insert to insert into HTS_OGA.

- Insert the following into the HTS_OGA table:
- hts: tariff number (V1c)
- import_country_id: import country from the program input parameter
- effect_from: begin effective date (V1e)
- effect_to: end effective date (V1f)
- code: OGA code from OGA codes field (V3f)
- reference_id: NULL
- comments: NULL

hts_spi_insert()

For each SPI, call function check_spi to check if the SPI exists on the tariff_treatment table; if not, reject the record. Call function hts_tariff_treatment_insert to insert into HTS_TARIFF_TREATMENT. In addition to the SPI records in V3, ‘C1’ and ‘C2’ are default tariff_treatments for every HTS. So, two extra records should be inserted into HTS_TARIFF_TREATMENT with SPI codes ‘C1’ and ‘C2’. ‘C1’ takes the special_duty_rate from V1 and Column 1 rates from V2; ‘C2’ takes Column 2 rates from V2. Before inserting, call function check_spi to make sure that the SPI code (tariff treatment) exists on the TARIFF_TREATMENT table; reject the record if it does not.

Call valid_all_numeric function to check specific_rate, ad_rate, other_rate for all numeric value. If not, reject the record.

Reject HTS lines that have rate greater than 9999999999. A brief explanation of why this is done is located at the end of the function level description section.

Insert the following into the HTS_TARIFF_TREATMENT table:

- hts: tariff number (V1c)
- import_country_id: import country from the program input parameter
- effect_from: begin effective date (V1e)
- effect_to: end effective date (V1f)
- tariff_treatment: SPI code from V3i
- specific_rate: 0,col1 or col2 specific rate, as appropriate (0 for SPI’s,col 1 for col1, col 2 for col2)
- av_rate: 0,col1, or col2 ad valorem rate, as appropriate (0 for SPI’s)
- other_rate: 0,col1, or col2 other rate, as appropriate (0 for SPI’s)

hts_gsp_insert()

For each GSP excluded country, call function check_country_tariff_treatment to check that the country and tariff treatment combination exists on the COUNTRY_TARIFF_TREATMENT table; if not, reject the record.

Insert the following into the HTS_TT_EXCLUSIONS table

- hts: tariff number (V1c)
- import_country_id: import country from the program input parameter
- effect_from: begin effective date (V1e)
- effect_to: end effective date (V1f)
- tariff_treatment: first SPI code from V3i
- origin_country_id: excluded country code from V3d (GSP excluded countries)

check_spi()

Check to see if SPI exists on TARIFF_TREATMENT table; reject the record if it doesn't.

check_country()

Check to see if country exists on COUNTRY table; reject the record if it doesn't.

check_chapter()

Check to see if chapter exists on the HTS_CHAPTER table and reject the record if it doesn't.

check_uom()

Check to see if uom exists on UOM_CLASS table; reject the record if it doesn't.

check_duty()

Check to see if duty code exists on CODE_DETAIL table (check for the code where

code_type='DCMP'); reject the record if it doesn't.

check_quota()

Check to see if the quota_category exists on the QUOTA_CATEGORY table; reject the record if it doesn't.

check_oga()

Check to see if the oga code exists on the OGA table; reject the record if it doesn't.

check_comb_country_tt()

Check to see if the country and tariff_treatment combination exists on the COUNTRY_TARIFF_TREATMENT table; reject the record if it doesn't.

process_TDETL()

- Format the tax line information from tdetl structure.
- Call function process_taxfees, if no non-fatal error in the process_THEAD function.

process_taxfees()

If tax specific rate or tax ad rate is not null, call hts_taxfee_insert to insert the tax rates into HTS_TAX or HTS_FEE tables. If special rates exist on the tax line, call function hts_tariff_treatment_insert to insert into the HTS_TARIFF_TREATMENT table using the ISO country code as the tariff treatment (SPI). If the SPI given on the tax line already exists for the HTS, the record should be updated, as the tax line special rate takes precedence over the V3 line SPI's rate

Call valid_all_numeric function to check tax_specific_rate, tax_av_rate, fee_specific_rate, fee_av_rate for all numeric value, if not reject the record.

Reject HTS lines that have rate greater than 9999999999. A brief explanation of why this is done is located at the end of the function level description section.

hts_taxfee_insert()

If the tax class code is 016,017,018,or 022 it is a tax; insert into HTS_TAX

If the tax class code is 038,053,054,055,056,057,079,090,103 it is a fee; insert into HTS_FEE

Insert the following into the HTS_TAX or HTS_FEE table:

- hts: tariff number (V1c)
- import_country_id: import country from the program input parameter
- effect_from: begin effective date (V1e)
- effect_to: end effective date (V1f)
- tax_type/fee_type: tax class code (V5h)
- tax_comp_code/fee_comp_code: tax comp code (V5i)
- tax_specific_rate/fee_specific_rate: tax specific rate (V5k)
- tax_av_rate/fee_av_rate: tax ad valorem rate (V5l)

hts_tariff_treatment_insert()

Before calling this function, call function check_spi to make sure that the SPI code (tariff treatment) exists on the TARIFF_TREATMENT table; reject the record if it does not.

Insert the following into the HTS_TARIFF_TREATMENT table:

- hts: tariff number (V1c)
- import_country_id: import country from the program input parameter
- effect_from: begin effective date (V1e)
- effect_to: end effective date (V1f)
- tariff_treatment: SPI code from V3i and VDd
- specific_rate: 0,col1 or col2 specific rate, as appropriate (0 for SPI's,col 1 for col1, col 2 for col2)
- av_rate: 0,col1 or col2 ad valorem rate, as appropriate (0 for SPI's)
- other_rate: 0,col1 or col2 other rate, as appropriate (0 for SPI's)

size_item_array()

Allocates space for the item array struct

size_ord_array()

Allocates space for the order array struct

size_ce_array()

Allocates space for the custom entry array struct

free_orditemlist()

Frees the space in the array

free_itemlist()

Frees the space in the array

free_ceordlist()

Frees the space in the array

uom_convert()

- Calls ITEM_ATTRIB_SQL.GET_STANDARD_UOM
- Calls UOM_SQL.CONVERT

order_del_assess()

- Calls ORDER_HTS_SQL.DELETE_ASSESS

delete_ord_temp_tables()

If an order no is not passed in, look at the hts table and see if there is an order that exists for that hts. If so, loop and for each record see if there is a record to delete on the temp tables by calling ORDER_SETUP_SQL.DELTE_TEMP_TABLES.

If the order number was passed in, call ORDER_SETU_SQL.DELETE_TEMP_TABLES right away.

final ()

Restart/recovery close and close input and reject file.

Why HTS lines that have a rate greater than 9999999999 need to be rejected:

For fields specific_rate, av_rate, other_rate, RMS has the data type Number(12,8) and numbers coming in from the customs tape also have 8 implied digits. However, when storing the number into the Retek database, we need to divide the number coming in from the customs tape by 1000000 (left shift 6 digits) instead of 100000000 (left shift 8 digits). This is because Retek stores the percent part of the rate only. In other words, rate 11.5% (0.115) is stored as 11.5 in Retek database, whereas it will come in from the customs tape as 11500000 (=0.115). Therefore, the highest rate that can be represented in Retek is 9999.99999999% (= 99.9999999999, or < 100 times). So we need to reject HTS lines that have rate greater than 9999999999.

Note: This is true for hts spi and hts tax/fee specific_rate and av_rate, except that when 999999999999

Input Specifications

Record Name	Field Name	Field Type	Default Value	Description
FHEAD	Record Descriptor	Char(5)	FHEAD	Describes file line type
	Line number	Number(10)	0000000001	Sequential file line number
	Retek file ID	Char(5)	HTSUP	Describes file type
THEAD	Record Descriptor	Char(5)	THEAD	Describes file line type
	Line number	Number(10)		Sequential file line number
	Transaction id	Number(14)		Unique transaction id
	HTS Line	Char(358)		V1 through V4 records from the customs HTS file concatenated together
TDETL	Record Descriptor	Char(5)	TDETL	Describes file line type
	Line number	Number(10)		Sequential file line number
	Transaction id	Number(10)		Unique transaction id
	Tax/fee line	Char(80)		V5 through VC records from the customs HTS file, each on a separate TDETL line
TTAIL	Record Descriptor	Char(5)	TTAIL	Describes file line type
	Line number	Number(10)		Sequential file line number
	Detail lines	Number(6)		Number of lines between THEAD and TTAIL
FTAIL	Record Descriptor	Char(5)	FTAIL	Describes file line type
	Line number	Number(10)		Sequential file line number
	Transaction Lines	Number(10)		Number of lines between FHEAD and FTAIL

Here is the layout of the original input file:

Note: The input file contains lines of 2400 characters, i.e. the newline character occurs only after every 2400 characters. Each 2400-character line consists of thirty 80-character records. Each 80-character record starts with 'V1' or 'V2' … or 'VD' or blank if the record is completely empty. For each tariff, records V1 and V2 are mandatory; records V3 through VD are optional, which means they can be all blank. Record V4 is not currently used in RMS/RTM. Records V5 through VC contain the tax/fee information for the tariff, and all have the same structure. The lower-case letters in the record name block are as a convenience to cross-reference with the US Customs file description.

Record Name	Field Name	Field Type	Default Value	Description
V1 a	Control identifier	Char(1)	V	Identifies start of record
b	Record type	Char(1)	1	Identifies record type
c	Tariff number	Number(10)		A code located in the Harmonized Tariff Schedule of the United States Annotated (HTS) representing the tariff number. If this number is less than 10 positions, it is left justified
d	transaction code	Char(1)	A, D, R	A code representing the type of transaction. Valid Transaction Codes are: A = Add D = Delete R = Replace
e	begin effective date	char(6)		A numeric date in MMDDYY (month, day, year) format representing the record begin effective date. This date indicates when the record becomes effective.
f	end effective date	char(6)		A numeric date in MMDDYY (month, day, year) format representing the record end effective date. This date indicates the last date the record is effective.

Record Name	Field Name	Field Type	Default Value	Description
g	number of reporting units	number(1)	0,1,or 2 or 3	The number of reporting units required by the Bureau of the Census. In a few instances, units not required by Census may be required to compute duty. In these cases, the Census reporting units are always first, followed by any additional units required to compute the duty.
h	1st reporting unit of measure	char(4)		A code representing the first unit of measure. If the reporting unit is X, no unit of measure is required except for certain tariff numbers in Chapter 99. Valid unit of measure codes are listed in Appendix C.
I	2nd reporting unit of measure	char(4)		A code representing the second unit of measure. Valid unit of measure codes are listed in Appendix C.
j	3rd reporting unit of measure	char(4)		A code representing the third unit of measure. Valid unit of measure codes are listed in Appendix C.
k	duty computation code	char(1)		A code indicating the formula to be used to compute the duty. Valid Duty Computation Codes are listed in Appendix F.
l	commodity description	char(30)		A condensed version of the commodity description that appears in the HTS.
m	column 1 specific rate of duty	Number(12)		The rate of duty that appears in the General column of the HTS. Eight decimal places are implied.

Record Name	Field Name	Field Type	Default Value	Description
n	base rate indicator	char(1)	'B' or blank	A code indicating if the rate contains a base rate. If the base rate indicator is B, the duty rate is a base rate; otherwise, space fill. Not Used in RMS.
o	space fill	char(1)	blank	Space fill. Not used in RMS.
V2 a	Control identifier	char(1)	V	Identifies start of record
b	Record type	char(1)	2	Identifies record type
c	tariff number	Number(10)		A code located in the Harmonized Tariff Schedule of the United States Annotated (HTS) representing the tariff number. If this number is less than 10 positions, it is left justified. This number is the same as that in Record Identifier V1.
d	general column 1 ad valorem percentage	Number(12)		The ad valorem rate of duty that appears in the General column of the HTS. Eight decimal places are implied.
e	column 1 other	Number(12)		The rate of duty that appears in the General column of the HTS that is not an ad valorem rate. Eight decimal places are implied.
f	Column 2 specific rate	Number(12)		The specific rate of duty that appears in Column 2 of the HTS. Eight decimal places are implied.
g	Column 2 ad valorem percentage	Number(12)		The ad valorem rate of duty that appears in Column 2 of the HTS. Eight decimal places are implied.

Record Name	Field Name	Field Type	Default Value	Description
h	Column 2 other rate	Number(12)		The rate of duty that appears in Column 2 of the HTS that is not an ad valorem rate or a specific rate. Eight decimal places are implied.
i	countervailing duty flag	char(1)	blank or 1	A code of 1 indicating the tariff number is subject to countervailing duty; otherwise, space fill.
j	additional tariff indicator	char(1)	blank or 'R'	A code indicating if an additional tariff number may be required with this tariff number. Refer to the Harmonized Tariff Schedule of the United States Annotated (HTS) for more specific information on which HTS numbers require additional HTS numbers to be reported. This indicator is R when an additional tariff number may be required; otherwise, space fill.
k	Miscellaneous Permit/License Indicator	char(2)		A code indicating if a tariff number may be subject to a miscellaneous permit/license number.
l	space fill	char(4)	blanks	Not used in RMS.
V3 a	Control identifier	char(1)	V	identifies start of record
b	Record type	char(1)	3	identifies record type
c	tariff number	Number(10)		A code located in the Harmonized Tariff Schedule of the United States Annotated (HTS) representing the tariff number. If this number is less than 10 positions, it is left justified. This number is the same as the number in Record Identifier V1.

Record Name	Field Name	Field Type	Default Value	Description
d	GSP excluded countries	char(20)		The International Organization for Standardization (ISO) country code that indicates countries not eligible for preferential treatment under GSP. Up to ten 2-position country codes can be reported. If countries are excluded from GSP, the Special Programs Indicator (SPI) Code contained in this record (positions 53-64) is A*. Valid ISO country codes are listed in Appendix B.
e	OGA codes	char(15)		Codes that indicate special requirements by other Federal Government agencies must or may apply. Up to five 3-position OGA codes can be provided.
f	anti-dumping flag	char(1)	1 or blank	A code of 1 indicating the tariff number is subject to an antidumping duty; otherwise, space fill.
g	quota indicator	char(1)	1 or blank	A code of 1 indicating the tariff number may be subject to quota. If the tariff number is not subject to quota, space fill.
h	category number	char(6)		A code located in the HTS indicating the textile category assigned to the tariff number. If there is no textile category number, space fill.

Record Name	Field Name	Field Type	Default Value	Description
I	special program indicators	char(28)		A code indicating if a tariff number is subject to a special program. Up to fourteen 2-position codes can be reported. Left justify. The SPI codes are not reported in any particular sequence. If more than fourteen 2-position codes are required, they are reported on the VD record.
NEWLINE			\n	
V4 a	Control identifier	char(1)	V	identifies start of record Entire V4 record not used in RMS.
b	Record type	char(1)	4	identifies record type
c	tariff number	number(10)		A code located in the Harmonized Tariff Schedule of the United States Annotated (HTS) representing the tariff number. If this number is less than 10 positions, it is left justified. This number is the same as the number reported in Record Identifier V1.
d	value edit code	char(3)		A code representing the value edit.
e	value low bounds	number(10)		A value representing the minimum value edit. Five decimal places are implied. If this record contains date edits (positions 36-53), space fill.
f	value high bounds	number(10)		A value representing the maximum value edit. Five decimal places are implied. If this record contains date edits (positions 36-53), space fill.

Record Name	Field Name	Field Type	Default Value	Description
g	entry date restriction	number(1)	0,1, or 2	A code representing the first entry date restriction code.
h	beginning restriction date	char(4)		A numeric date in MMDD (month and day) format representing the first begin restriction date used in the edit. If this record contains a value edit (positions 13-35), space fill.
I	end restriction date	char(4)		A numeric date in MMDD (month and day) format representing the first end restriction date used in the edit. If this record contains a value edit (positions 13-35), space fill.
j	entry date restriction 2	number(1)	0,1, or 2	A code representing the second entry date restriction code.
k	beginning restriction date 2	char(4)		A numeric date in MMDD (month and day) format representing the second begin restriction date used in the edit. If this record contains a value edit (positions 13-35), space fill.
l	end restriction date 2	char(4)		A code located in the Harmonized Tariff Schedule of the United States Annotated (HTS) representing the tariff number. If this number is less than 10 positions, it is left justified. This number is the same as the number reported in Record Identifier V1.
m	country of origin	char(2)		A code representing the value edit.

Record Name	Field Name	Field Type	Default Value	Description
n	space filler	char(2)	blanks	A value representing the minimum value edit. Five decimal places are implied. If this record contains date edits (positions 36-53), space fill.
o	quantity edit code	char(3)		A value representing the maximum value edit. Five decimal places are implied. If this record contains date edits (positions 36-53), space fill.
p	low quantity	number(10)		A code representing the first entry date restriction code.
q	high quantity	number(10)		A numeric date in MMDD (month and day) format representing the first begin restriction date used in the edit. If this record contains a value edit (positions 13-35), space fill.
V5 a	Control identifier	char(1)	V	identifies start of record
b	Record type	char(1)	5,6,7,8,9,A,B, C	identifies record type
c	tariff number	number(10)		A code located in the Harmonized Tariff Schedule of the United States Annotated (HTS) representing the tariff number. If this number contains less than 10 positions, it is left justified. This number is the same as the number reported in Record Identifier V1.

Record Name	Field Name	Field Type	Default Value	Description
d	Country code	char(2)		A code representing the country. Valid ISO country codes are listed in Appendix B. E followed by a space (Caribbean Basin Initiative), and J followed by a space (Andian Trade Preference Act), and R followed by a space (Caribbean Trade Partnership Act), are also valid codes for special rates. Countries eligible for E and J are indicated in the ACS country code file and the Harmonized Tariff Schedule of the United States - Annotated (HTS).
e	specific rate	number(12)		The specific rate of duty listed in the Special column of the HTS. Eight decimal places are implied.
f	ad valorem rate	number(12)		The ad valorem rate of duty listed in the Special column of the HTS. Eight decimal places are implied.
g	Other rate	number(12)		The rate of duty listed in the Special column of the HTS that is not a specific or ad valorem rate. Eight decimal places are implied.
h	tax/fee class code	char(3)		A code representing the tax/fee class. Valid tax/fee class codes are listed in Appendix B.
I	tax/fee comp code	char(1)		A code indicating the first tax/fee computation formula. Computation formulas are presented in Appendix F.

Record Name	Field Name	Field Type	Default Value	Description
j	tax/fee flag	number(1)		A code indicating a tax/fee is required. Valid Tax/Fee Flag Codes are: 1 = Tax/fee required 2 = Tax/fee may be required Not used in RMS.
k	tax/fee specific rate	number(12)	blank if no value	The specific rate of duty required to compute taxes and/or fees. Eight decimal places are implied.
l	tax/fee ad valorem	number(12)	blank if no value	The ad valorem rate of duty required to compute taxes and/or fees. Eight decimal places are implied.
m	space fill	char(1)	blank	Space fill.
V6 through VC records have the same fields as the V5 record.				
VD a	Control identifier	char(1)	V	identifies start of record
b	Record type	char(1)	D	identifies record type
c	tariff number	number(10)		unique tariff number
d	Special Program Indicator (SPI) Code	char(32)		A code indicating if a tariff number is subject to a special program. Up to sixteen additional 2-position codes can be reported. Left justify. The SPI codes are not reported in any particular sequence
e	Filler	char(36)		Space fill.

Output Specifications

N/A

Scheduling Considerations

Processing Cycle: Ad hoc

Scheduling Diagram: Run anytime as needed.

Pre-Processing: after hts upload conversion (ushts2rms – PERL script).

Post-Processing: None

Threading Scheme: None

Restart Recovery

This program supports Retek standard intermittent commit and file upload restart/recovery. Recommended commit counter is 2000 (commit after every 2000 tariff records are read). Input file names must end in a ".1" for the restart mechanism to properly parse the file name. Since there is only 1 input file to be uploaded, only 1 thread is used. A reject file is used to hold records that have failed processing. The user can fix the rejected records and process the reject file again.

Order purge [ordprg]

Design Overview

The purpose of this module is to remove old orders 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). Orders will only be deleted if all allocations associated, if any, have been closed. 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 allocations associated have been closed, 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) and allocations associated to the order, if any, have been closed. 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), all allocations associated to the order, if any, have been closed, 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.

Tables Affected:

TABLE	INDEX	SELECT	INSERT	UPDATE	DELETE
ALC_COMP_LOC	Yes	No	No	No	Yes
ALC_HEAD	Yes	Yes	No	No	Yes
ALLOC_CHRG	Yes	No	No	No	Yes
ALLOC_DETAIL	No	No	No	No	Yes
ALLOC_HEADER	Yes	Yes	No	No	Yes
ALLOC_REV	Yes	No	No	No	Yes
APPT_DETAIL	Yes	Yes	No	No	Yes
APPT_HEAD	No	Yes	No	No	Yes
CARTON	No	No	No	No	Yes
CE_CHARGES	Yes	No	No	No	Yes
CE_FORMS	Yes	No	No	No	Yes

TABLE	INDEX	SELECT	INSERT	UPDATE	DELETE
CE_HEAD	Yes	No	No	No	Yes
CE_LIC_VISA	No	No	No	No	Yes
CE_ORD_ITEM	Yes	Yes	No	No	No
CE_ORD_ITEM	Yes	No	No	No	Yes
CE_PROTEST	No	No	No	No	Yes
CE_SHIPMENT	Yes	No	No	No	Yes
DAILY_PURGE	No	No	Yes	No	No
DEAL_CALC_QUEUE	No	No	No	No	Yes
DEAL_DETAIL	Yes	No	No	No	Yes
DEAL_HEAD	Yes	No	No	No	Yes
DEAL_ITEMLOC	Yes	No	No	No	Yes
DEAL_QUEUE	No	No	No	No	Yes
DEAL_THRESHOLD	No	No	No	No	Yes
DOC_CLOSE_QUEUE	Yes	No	No	No	Yes
INVC_HEAD	Yes	Yes	No	No	No
INVC_MATCH_WKSHT	Yes	No	No	No	Yes
INVC_XREF	Yes	No	No	No	Yes
ITEM_MASTER	Yes	Yes	No	No	No
LC_ORDAPPLY	No	No	No	No	Yes
MISSING_DOC	Yes	No	No	No	Yes
OBLIGATION	Yes	No	No	No	Yes
OBLIGATION_COMP	Yes	No	No	No	Yes
OBLIGATION_COMP_LOC	No	No	No	No	Yes
ORD_HEAD	Yes	Yes	No	No	Yes
ORD_INV_MGMT	Yes	No	No	No	Yes
ORD_LC	No	Yes	No	No	No
ORD_XDOCK_TEMP	No	No	No	No	Yes
ORDCUST	Yes	No	No	No	Yes
ORDHEAD_DISCOUNT	No	No	No	No	Yes
ORDHEAD_REV	No	No	No	No	Yes
ORDLOC	Yes	No	No	No	Yes
ORDLOC_DISCOUNT	Yes	No	No	No	Yes

TABLE	INDEX	SELECT	INSERT	UPDATE	DELETE
ORDLOC_EXP	Yes	No	No	No	Yes
ORDLOC_INVC_COST	No	No	No	No	Yes
ORDLOC_REV	No	No	No	No	Yes
ORDLOC_WKSHT	Yes	No	No	No	Yes
ORDSKU	Yes	Yes	No	No	Yes
ORDSKU_HTS	Yes	No	No	No	Yes
ORDSKU_HTS_ACCESS	Yes	No	No	No	Yes
ORDSKU_REV	Yes	No	No	No	Yes
PACK_ITEM	Yes	Yes	No	No	No
PACK_TMPL_HEAD	Yes	Yes	No	No	No
PERIOD	No	Yes	No	No	No
REPL_RESULTS	Yes	No	No	No	Yes
REQ_DOC	Yes	No	No	No	Yes
REV_ORDERS	No	No	No	No	Yes
RTV_DETAIL	Yes	No	No	No	Yes
RUA_RIB_INTERFACE	No	No	No	No	Yes
SHIPMENT	Yes	Yes	No	No	Yes
SHIPSKU	Yes	Yes	No	Yes	Yes
SUP_VIOLATION	No	No	No	No	Yes
SYSTEM_OPTIONS	No	Yes	No	No	No
TIMELINE	Yes	No	No	No	Yes
TRANS_CLAIMS	Yes	No	No	No	Yes
TRANS_DELIVERY	Yes	No	No	No	Yes
TRANS_LIC_VISA	Yes	No	No	No	Yes
TRANS_PACKING	Yes	No	No	No	Yes
TRANS_SKU	Yes	No	No	No	No
TRANSPORTATION	Yes	Yes	No	No	No
UNIT_OPTIONS	No	Yes	No	No	No
WO_DETAIL	Yes	No	No	No	Yes
WO_HEAD	Yes	Yes	No	No	Yes

Stored Procedures / Shared Modules (Maintainability)

INV_SQL.DELETE_INVC

ORDER_SETUP_SQL.DELETE_TEMP_TABLES

Program Flow

N/A

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

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 letter of credit is present, then the order cannot be purge
- If import order, landed cost implications must be considered before purging the order
- Call delete_landed_costs to delete landed cost records associated with the order;
- Call del_appts to delete associated records from all appt_* tables
- Call purge_transport to delete transport records for order with many transports
- Call purge_custom_entry to delete customs entry records for order with many customs entries
- Insert records into daily_purge so pack records related to the current order_no will be deleted by the dlyprg batch run. Insert only occurs if packs on current order are not found on any other order
- Delete RTV detail records
- Delete Shipment detail records
- Delete Carton records
- Delete Work Order detail records
- Delete Work Order primary records
- Delete Allocation Charges records

- Delete Allocation detail records
- Delete Allocation primary records
- Delete Timeline records
- Delete Order Location records
- Delete Order Location Discount records
- Delete Order Location Exp records
- Delete Order detail HTS Access records
- Delete Order detail HTS primary records
- Delete Requested/Required Document records
- Delete Order Location Rev records
- Delete Order detail Rev records
- Delete Order primary Rev records
- Delete Allocation Rev records
- Delete Order Location Invoice Cost records
- Delete Order detail records
- Delete Order Customer records
- Delete Order Cross-dock Temp records
- If order has invoice matching record
  - Delete Invoice Cross-Reference records
  - Delete Invoice Matching Worksheet records
- Delete Order Location Worksheet records
- Delete Supplier Violation records
- Delete Shipment records
- Delete Rev Order records
- Call delete_deals to delete associated deal records with the order
- Call del_repl_orders to Invoice Order Management and Replenishment Result records
- Delete LC Order Apply records
- Delete Order primary Discount records
- Delete Order primary records
- Delete RUA RIB Interface records
- Call ORDER_SETUP_SQL.DELETE_TEMP_TABLE function to delete the order delete temp tables

delete_invc_data()

Updates SHIPSKU.MATCH_INVC_ID column to NULL for invoices associated with the orders being purged. 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.

final()

Close the output file.

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 , trans_sku, transportation,and trans_lic_visa(based on transportation_id).

purge_customs_entry()

Delete from customs entry for specific orders being purged. The following tables are being deleted from this function:

- CE_CHARGES
- CE_FORMS
- CE_HEAD
- CE_LIC_VISA
- CE_ORD_ITEM
- CE_ORD_ITEM
- CE_PROTEST
- CE_SHIPMENT

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.

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. The following tables are being deleted from this function:

- DEAL_CALC_QUEUE
- DEAL_DETAIL
- DEAL_HEAD
- DEAL_ITEMLOC
- DEAL_QUEUE
- DEAL_THRESHOLD

del_repl_orders ()

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

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 (when Retek's Invoice Matching product is not in use):

```
          SELECT oh.order_no,
                 lc.lc_ind
            FROM ordhead oh,
                 ordlc lc
           WHERE lc.order_no(+) = oh.order_no
             AND ((0 <
(NVL(MONTHS_BETWEEN(TO_DATE(:ps_vdate,'YYYYMMDD'),
                       oh.close_date),0) - :pi_hist_months))
                 OR (oh.status = 'W'
                 AND oh.orig_ind = 0
                 AND oh.contract_no is NULL
                 AND (TO_DATE(:ps_vdate,'YYYYMMDD') - oh.written_date)
>= :pi_repl_order_history_days));
           AND NOT EXISTS (SELECT 'x'
                             FROM  alloc_header  alloc2
```

```
                              WHERE ((alloc2.order_no  = oh.order_no
                            AND alloc2.status !='C')
                                    OR  EXISTS (SELECT 'x'
                                                  FROM alloc_header
alloc3
                                                 WHERE
alloc3.alloc_parent = alloc2.alloc_no
                                                   AND alloc2.order_no
= oh.order_no
                                   AND alloc3.status != 'C'
                                                   AND ROWNUM = 1))
                                 AND ROWNUM = 1);
```

Driving Cursor (when Retek's Invoice Matching product in use):

```
         SELECT distinct oh.order_no,
                lc.lc_ind
           FROM ordhead oh,
                shipment sh,
                shipsku ss,
                invc_head ih,
                ordlc lc
          WHERE oh.order_no = sh.order_no
            AND lc.order_no(+) = oh.order_no
            AND sh.shipment = ss.shipment
            AND ss.match_invc_id = ih.invc_id(+)
            AND (0 <
(NVL(MONTHS_BETWEEN(TO_DATE(:ps_vdate,'YYYYMMDD'),
                       oh.close_date),0) - :pi_hist_months))
            AND ((0 <
(NVL(MONTHS_BETWEEN(TO_DATE(:ps_vdate,'YYYYMMDD'),
                          sh.invc_match_date),0) - :pi_hist_months))
                 OR (sh.invc_match_date IS NULL))
            AND (ss.match_invc_id is null OR ih.status = 'P')
            AND sh.invc_match_status = 'C'
            AND 'C' = (SELECT decode(max(ship.invc_match_status),
min(ship.invc_match_status),'C','X')
                         FROM shipment ship
                        where ship.order_no = oh.order_no
                      group by ship.order_no)
            AND NOT EXISTS (SELECT 'x'
```

```
                              FROM  alloc_header  alloc2
                             WHERE ((alloc2.order_no  = oh.order_no
                           AND alloc2.status !='C')
                                 OR  EXISTS (SELECT 'x'
                                               FROM alloc_header
alloc3
                                              WHERE
alloc3.alloc_parent = alloc2.alloc_no
                                                AND alloc2.order_no
= oh.order_no
                                AND alloc3.status != 'C'
                                               AND ROWNUM = 1))
                               AND ROWNUM = 1)
        UNION
        SELECT oh.order_no,
               lc.lc_ind
          FROM ordhead oh,
               ordlc lc
         WHERE lc.order_no(+) = oh.order_no
           AND oh.status = 'W'
           AND oh.orig_ind = 0
           AND oh.contract_no is NULL
           AND (TO_DATE(:ps_vdate,'YYYYMMDD') - oh.written_date) >=
:pi_repl_order_history_days
           AND NOT EXISTS (SELECT 'x'
                             FROM  alloc_header  alloc2
                            WHERE ((alloc2.order_no  = oh.order_no
                          AND alloc2.status !='C')
                                 OR  EXISTS (SELECT 'x'
                                               FROM alloc_header
alloc3
                                              WHERE
alloc3.alloc_parent = alloc2.alloc_no
                                                AND alloc2.order_no
= oh.order_no
                                AND alloc3.status != 'C'
                                               AND ROWNUM = 1))
                               AND ROWNUM = 1)
        UNION
        SELECT oh.order_no,
```

```
                lc.lc_ind
           FROM ordhead oh,
                ordlc lc
          WHERE lc.order_no(+) = oh.order_no
            AND (0 <
(NVL(MONTHS_BETWEEN(TO_DATE(:ps_vdate,'YYYYMMDD'),
                    oh.close_date),0) - :pi_hist_months))
            AND oh.status = 'C'
            AND NOT EXISTS (SELECT 'x'
                              FROM shipment sh
                             WHERE sh.order_no = oh.order_no);
            AND NOT EXISTS (SELECT 'x'
                              FROM  alloc_header  alloc2
                             WHERE ((alloc2.order_no  = oh.order_no
                           AND alloc2.status !='C')
                                 OR  EXISTS (SELECT 'x'
                                               FROM alloc_header
alloc3
                                              WHERE
alloc3.alloc_parent = alloc2.alloc_no
                                                AND alloc2.order_no
= oh.order_no
                                AND alloc3.status != 'C'
                                               AND ROWNUM = 1))
                               AND ROWNUM = 1);
```

Output Specifications

N/A

Scheduling Considerations

Processing Cycle: PHASE AD-HOC (monthly)

Scheduling Diagram: N/A

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: N/A (single threaded)

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

Design Overview

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

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

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

prepost user/password rpl pre

Tables affected:

TABLE	SELECT	INSERT	UPDATE	INDEX	DELETE	TRUNCATE	TRIGGER
all_constraints	Y	N	N	N	N	N	N
all_ind_partitions	Y	N	N	N	N	N	N
all_policies	Y	N	N	N	N	N	N
alloc_detail	Y	N	N	N	N	N	Y
alloc_header	Y	N	N	N	N	N	Y
class	Y	N	N	N	N	N	N
class_sales_forecast	N	N	N	Y	N	Y	N
class_sales_hist	N	N	N	N	Y	N	N
class_sales_hist_mth	Y	N	N	N	Y	N	N
cost_change_trigger_temp	Y	N	N	Y	N	Y	N
cost_susp_head	N	N	Y	N	N	N	N
daily_data	Y	N	N	N	N	N	N
daily_data_temp	Y	N	N	N	N	Y	N
dba_indexes	Y	N	N	N	N	N	N
dba_triggers	Y	N	N	N	N	N	N
dealfct_temp	N	Y	N	N	N	N	N
deal_actuals_forecast	Y	N	N	N	N	N	N

TABLE	SELECT	INSERT	UPDATE	INDEX	DELETE	TRUNCATE	TRIGGER
deal_actuals_item_loc	Y	Y	N	N	N	N	N
deal_bb_no_rebate_temp	N	Y	N	N	N	Y	N
deal_bb_rebate_po_temp	N	Y	N	N	N	Y	N
deal_bb_receipt_sales_temp p	N	Y	N	N	N	Y	N
deal_head	Y	N	N	N	N	N	N
deal_item_loc_explode	Y	N	N	N	N	N	N
deal_sku_temp	N	N	N	Y	N	Y	N
deps	Y	N	N	N	N	N	N
dept_sales_forecast	N	N	N	Y	N	Y	N
dept_sales_hist	N	N	N	N	Y	N	N
dept_sales_hist_mth	Y	N	N	N	Y	N	N
domain_class	N	N	Y	N	N	N	N
domain_dept	N	N	Y	N	N	N	N
domain_subclass	N	N	Y	N	N	N	N
edi_daily_sales	N	N	N	N	Y	N	N
edi_ord_temp	N	N	N	Y	N	Y	N
fif_receiving	N	Y	N	Y	N	Y	N
fixed_deal	Y	N	Y	N	N	N	N
forecast_rebuild	N	N	N	Y	N	Y	N
groups	Y	N	N	N	N	N	N
hist_rebuild_mask	Y	N	N	Y	N	Y	N
ib_results	N	N	Y	N	N	N	N
if_tran_data	Y	N	N	N	N	N	N
invc_detail	N	N	Y	N	N	N	N
invc_detail_temp	Y	N	N	N	N	Y	N
invc_detail_temp2	N	N	N	N	N	Y	N
invc_head	N	N	Y	N	N	N	N
invc_head_temp	Y	N	N	N	N	Y	N
item_forecast	N	N	N	Y	N	N	N
item_loc	Y	N	N	N	N	N	N
item_loc_temp	N	Y	N	N	N	Y	N

TABLE	SELECT	INSERT	UPDATE	INDEX	DELETE	TRUNCATE	TRIGGER
item_master	Y	N	N	N	N	N	N
item_supp_country	Y	N	N	N	N	N	N
item_supp_country_loc	Y	N	N	N	N	N	N
mc_rejections	N	N	N	Y	N	Y	N
mod_order_item_hts	N	N	N	Y	N	Y	N
on_order_temp	N	N	N	Y	N	Y	N
ord_missed	N	N	N	Y	N	Y	N
ord_temp	N	N	N	Y	N	Y	N
ordhead	Y	N	N	N	N	N	N
ordsku	Y	N	N	N	N	N	N
packitem	Y	N	N	N	N	N	N
period	Y	N	N	N	N	N	N
pos_button_head	N	N	Y	N	N	N	N
pos_coupon_head	N	N	Y	N	N	N	N
pos_merch_criteria	N	N	Y	N	N	N	N
pos_mods	N	Y	N	Y	N	Y	N
pos_money_ord_head	N	N	Y	N	N	N	N
pos_payinout_head	N	N	Y	N	N	N	N
pos_prod_rest_head	N	N	Y	N	N	N	N
pos_store	N	N	Y	N	N	N	N
pos_sup_pay_criteria	N	N	Y	N	N	N	N
pos_tender_type_head	N	N	Y	N	N	N	N
reclass_cost_chg_queue	Y	Y	Y	N	N	N	N
reclass_head	Y	N	N	N	N	N	N
reclass_item	Y	N	N	N	N	N	Y
reclass_trigger_temp	Y	N	N	Y	Y	Y	N
repl_attr_update_exclude	Y	N	N	N	Y	N	N
repl_attr_update_head	Y	N	N	N	Y	N	N
repl_attr_update_item	Y	N	N	N	Y	N	N
repl_attr_update_loc	Y	N	N	N	Y	N	N
repl_day	Y	Y	N	N	N	N	N
repl_item_loc	Y	Y	N	N	N	N	N

TABLE	SELECT	INSERT	UPDATE	INDEX	DELETE	TRUNCATE	TRIGGER
repl_item_loc_updates	N	Y	N	Y	N	Y	N
rpl_alloc_in_tmp	N	Y	N	N	N	Y	N
rpl_distro_tmp	N	Y	N	N	N	Y	N
salweek_c_daily	N	Y	N	N	N	Y	N
salweek_c_week	Y	Y	N	N	N	Y	N
salweek_restart_dept	N	Y	N	N	N	Y	N
sec_user_zone_matrix	N	N	N	Y	N	Y	N
stage_complex_deal_detail	N	N	N	N	N	Y	N
stage_complex_deal_head	N	N	N	N	N	Y	N
stage_fixed_deal_detail	N	N	N	N	N	Y	N
stage_fixed_deal_head	N	N	N	N	N	Y	N
stake_head	Y	N	N	N	N	N	N
stake_prod_loc	Y	N	N	N	N	N	N
stake_sku_loc	Y	N	N	N	N	N	N
store	Y	N	Y	N	N	N	N
store_add	Y	N	N	N	Y	N	N
subclass_sales_forecast	N	N	N	Y	N	N	N
subclass_sales_hist	N	N	N	N	Y	N	N
subclass_sales_hist_mth	Y	N	N	N	Y	N	N
sup_data	N	N	N	N	Y	N	N
sups_min_fail	N	N	N	Y	N	Y	N
system_options	Y	N	N	N	N	N	N
system_variables	Y	N	Y	N	N	N	N
temp_tran_data	Y	N	N	N	N	Y	N
temp_tran_data_sum	N	Y	N	N	N	Y	N
tif_explode	N	N	N	Y	N	Y	N
tran_data	N	Y	N	N	N	N	N
tsf_head	N	N	Y	N	N	N	N
vat_code_rates	Y	N	N	N	N	N	N
vat_item	Y	N	N	N	N	N	N
week_data_temp	N	N	N	N	N	Y	N
wh	Y	N	N	N	N	N	N

TABLE	SELECT	INSERT	UPDATE	INDEX	DELETE	TRUNCATE	TRIGGER
wh_store_assign	N	N	N	N	Y	N	N

Scheduling Constraints

Processing Cycle: PHASE ALL (daily)

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

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: N/A (single threaded)

Restart Recovery

N/A

Program Flow

N/A

Shared Modules

- FORECASTS_SQL.GET_SYSTEM_FORECAST_IND
- UDA_SQL.CHECK_REQD_NO_VALUE
- FORECASTS_SQL.GET_DOMAIN
- ITEM_ATTRIB_SQL.GET_PACK_INDS
- FORECASTS_SQL.GET_ITEM_FORECAST_IND
- POS_UPDATE_SQL.POS_INVC_DETAIL_INSERT
- CAL_TO_454_LDOM
- CAL_TO_454_HALF
- CAL_TO_CAL_HALF
- CAL_TO_CAL_LDOM
- CAL_TO_454_WEEKNO
- CAL_TO_CAL_WEEKNO
- CAL_TO_454
- HALF_TO_CAL_FDOH
- HALF_TO_CAL_LDOH
- HALF_TO_454_FDOH
- HALF_TO_454_LDOH
- DBMS_RLS.ENABLE_POLICY

Function Level Description

Functions to be used by the individual program functions:

modify_indexes()

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

get_lock()

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

modify_partition_indexes()

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

truncate_table()

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

modify_trigger()

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

alter_constraints()

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

truncate_user_sec_table()

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

get_454_ldom()

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

get_454_half()

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

get_next_454_half()

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

get_next_cal_half()

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

get_cal_half()

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

get_cal_ldom()

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

get_454_weekno()

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

get_cal_weekno()

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

get_454_date()

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

get_cal_fdoh()

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

get_cal_ldoh()

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

get_454_fdoh(void);

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

get_454_ldoh(void)

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

get_tomorrow()

This function gets the next day after the vdate.

get_forecast_ind()

This function cals FORECASTS_SQL.GET_SYSTEM_FORECAST_IND to get the system_forecast_ind.

validate_reclassify()

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

check_stock_count()

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

check_order()

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

check_uda()

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

check_domain_exists()

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

check_forecast()

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

delete_reclass_trigger_temp()

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

Individual Program Functions

rpl_pre()

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

- ORD_TEMP
- ORD_MISSED

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

salweek_pre()

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

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

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

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

salweek_post()

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

salmth_post()

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

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

rplapprv_pre()

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

rplatupd_pre()

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

rplatupd_post()

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

rilmaint_post()

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

supmth_post()

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

sccext_post()

Updates all processed supplier cost change record status to ‘Extracted’.

hstbld_pre()

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

hstbld_post()

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

posdnld_post()

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

poscdnld_post()

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

reqext_post()

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

likestore_post()

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

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

stkupd_pre()

Calls the stored function DBMS_MVIEW.REFRESH.

stkupd_post()

This function disables the RMS_COL_ITL_UR_AUR trigger of ITEM_LOC.

dtesys_post()

Enables the RMS_COL_ITL_UR_AUR trigger of ITEM_LOC table.

ociroq_pre()

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

rplext_post()

Truncates the tables RPL_DISTRO_TMP, and RPL_ALLOC_IN_TMP.

posupld_post()

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

vatdlxpl_post()

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

saleoh_pre()

Calculates the next_eom_date, and updates the SYSTEM_VARIABLES.

dealday_pre()

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

dealday_post()

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

hstbldmth_post()

This is responsible for deleting records in the following tables:

- CLASS_SALES_HIST_MTH
- SUBCLASS_SALES_HIST_MTH
- CLASS_SALES_HIST_MTH
- DEPT_SALES_HIST_MTH

THE FOLLOWING FUNCTIONS SHOULD BE RUN AFTER THE edidlprd PROGRAM!

edidlprd_post()

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

fcstrbld_post()

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

vrplbld_post()

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

cntrordb_post()

Sets the last_cont_order_date on system_variables to vdate.

fifgldn1_post()

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

fsadnld_post()

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

policy_enable()

Enables or disables policies.

whstrasg_post ()

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

costcalc_post()

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

tifposdn_post()

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

htsupld_pre()

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

onordext_pre()

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

precostcalc_pre()

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

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

reclsdly_pre()

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

ibcalc_pre()

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

fcstprg_pre()

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

- ITEM_FORECAST
- DEPT_SALES_FORECAST
- CLASS_SALES_FORECAST
- SUBCLASS_SALES_FORECAST

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

fcstprg_post()

This rebuilds the indexes following truncation of following tables:

- ITEM_FORECAST
- DEPT_SALES_FORECAST
- CLASS_SALES_FORECAST
- SUBCLASS_SALES_FORECAST

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

dealinc_pre()

Call get_sys_date()

Call size_arrays()

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

dealfct_pre()

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

dealact_pre_no_rebate()

Truncates the deal_bb_no_rebate_temp table.

Then inserts billback NO Rebate type of deal into deal_bb_no_rebate_temp.

dealact_pre_rebate_po()

Truncates the deal_bb_rebate_po_temp table.

Then inserts billback rebate PO type of deal into deal_bb_rebate_po_temp.

dealact_pre_receipt_sales ()

Truncates the deal_bb_receipt_sales_temp.

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

vendinvc_pre()

Truncate the STAGE_COMPLEX_DEAL_HEAD table.

Truncate the STAGE_COMPLEX_DEAL_DETAIL table.

Then inserts complex deals for invoicing into vendinvc_temp.

vendinvf_pre()

Truncate the STAGE_FIXED_DEAL_HEAD table.

Truncate the STAGE_FIXED_DEAL_DETAIL table.

vendinvc_post()

Get vdate.

Call process_deal_head().

vendinvf_post()

Get vdate.

Call process_fixed_deal().

process_fixed_deal()

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

process_deal_head()

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

I/O Specification

N/A

Technical Issues

N/A

Transfer purge [tsfprg]

Design Overview

The purpose of this module is to purge transfer records, deleting all rows from the transfer header and detail table based on the number of months of transfer history to be retained. The number of transfer history months to be retained is specified on system options. If the difference in the number of months between today and the date on which the transfer was closed is greater than or equal to the number of transfer history months, the header and detail record are purged. If a transfer has allocations associated to it, all these allocations and associated tier records must be closed first before the transfer records can be purged. Note however, that Mass Return Transfers (MRT) are not processed by this batch program. Purging of MRT records are done by mrtprg.pc.

Tables Affected:

TABLE	INDEX	SELECT	INSERT	UPDATE	DELETE
PERIOD	No	Yes	No	No	No
SYSTEM_OPTIONS	No	Yes	No	No	No
TSFHEAD	No	Yes	No	No	Yes
TSFDETAIL	No	No	No	No	Yes
TSFDETAIL_CHRG	No	No	No	No	Yes
TSF_WO_DETAIL	No	No	No	No	Yes
TSF_WO_HEAD	No	No	No	No	Yes
TSF_XFORM_DETAIL	No	No	No	No	Yes
TSF_XFORM	No	No	No	No	Yes
TSF_PACKING_DETAIL	No	No	No	No	Yes
TSF_PACKING	No	No	No	No	Yes
TSF_ITEM_WO_COST	No	No	No	No	Yes
TSF_ITEM_COST	No	No	No	No	Yes
SHIPMENT	No	No	No	No	Yes
SHIPSKU	No	Yes	No	No	Yes
ORDCUST	No	No	No	No	Yes
SHIPITEM_INV_FLOW	No	No	No	No	Yes
CARTON	No	No	No	No	Yes
APPT_HEAD	No	Yes	No	No	Yes
APPT_DETAIL	No	Yes	No	No	Yes
ALLOC_HEADER	No	Yes	No	No	Yes
ALLOC_DETAIL	No	No	No	No	Yes

TABLE	INDEX	SELECT	INSERT	UPDATE	DELETE
ALLOC_CHRG	No	No	No	No	Yes
DOC_CLOSE_QUEUE	No	No	No	No	Yes
V_RESTART_TRANSFER	No	Yes	No	No	No

Function Level Description

init()

- Call restart_init().
- Initialize arrays

  Fetch tsf_history_months from system_options and vdate from period.

process()

open driving cursor

while

- Loop through the records in the driving cursor feeding them into an array as they are fetched
- When no more records are found, set a flag to false so that the loop will be exited at the end of the processing.
- Determine the number of records processed this time through the loop
- Delete the records in shipitem_inv_flow for all tsf_no fetched into the array
- Loop thru all transfers fetched in the main cursor
  - call del_appts() to delete associated appointments
  - call del_allocs() to delete associated allocations and its tiers
  - open cursor fetching all shipment numbers for the transfer number
  - while(1)
    - fetch the shipment number to the shipment array
    - Delete from shipsku and shipment for all of the shipment numbers in the array
  - end while loop
- End loop
- Delete from ordcust, tsfdetail_chrg, tsf_wo_detail, tsf_wo_head, tsf_xform_detail, tsf_xform, tsf_packing_detail, tsf_packing, tsf_item_wo_cost, tsf_item_cost, tsfdetail, and tsfhead for all of the transfer number in the array.

End while loop

Delete from carton table

Size_arrays()

- Allocate system memory for common update array components.
- The number of rows of each array should be determined by the counter (commit_max_ctr) on the restart_control table. The value will be obtained in the init via the function call to restart_init

final()

- Call retek_close() function

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.

Del_allocs()

Delete records from alloc_chrg, alloc_detail and alloc_header tables. The records deleted are allocations associated to the transfers and its tiers.

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:

```
       SELECT th.tsf_no,
              -1 child_tsf_no,
              th.to_loc,
              th.to_loc_type,
              th.from_loc,
              th.from_loc_type
         FROM tsfhead th,
              v_restart_transfer rv
        WHERE th.mrt_no IS NULL
          AND NOT EXISTS (SELECT 'X'
                          FROM tsfhead th1
                          WHERE th1.tsf_parent_no = th.tsf_no)
          AND (th.status           =  'D'
               OR (th.status       =  'C'
                   AND MONTHS_BETWEEN(TO_DATE(:os_vdate,'YYYYMMDD'),
  th.close_date)
                       >= :ol_tsf_history_mths)
              )
```

```
       AND NOT EXISTS (SELECT 'x'
                         FROM  alloc_header  alloc2
                        WHERE ((alloc2.order_no  = th.tsf_no
                                AND alloc2.status !='C')
                                OR  EXISTS (SELECT 'x'
                                              FROM alloc_header
alloc3
                                             WHERE
alloc3.alloc_parent = alloc2.alloc_no
                                               AND alloc2.order_no
= th.tsf_no
                                       AND alloc3.status != 'C'
                                               AND ROWNUM = 1))
                          AND ROWNUM = 1)
       AND rv.driver_value     = th.tsf_no
       AND rv.num_threads      = TO_NUMBER(:ps_num_threads)
       AND rv.thread_val       = TO_NUMBER(:ps_thread_val)
 UNION ALL
    SELECT th.tsf_no,
           th1.tsf_no  child_tsf_no,
           th.to_loc,
           th.to_loc_type,
           th.from_loc,
           th.from_loc_type
      FROM tsfhead th,
           tsfhead th1,
           v_restart_transfer rv
     WHERE (th.status              =  'D'
           OR (th.status           =  'C'
              AND
MONTHS_BETWEEN(TO_DATE(:os_vdate,'YYYYMMDD'),th.close_date)
                    >= :ol_tsf_history_mths))
       AND (th1.tsf_parent_no      = th.tsf_no
            AND (th1.status        =  'D'
                 OR (th1.status    =  'C'
                     AND
MONTHS_BETWEEN(TO_DATE(:os_vdate,'YYYYMMDD'),th1.close_date) >=
:ol_tsf_history_mths)))
       AND NOT EXISTS (SELECT 'x'
```

```
                          FROM  ALLOC_HEADER  alloc2
                         WHERE ((alloc2.order_no  IN (th.tsf_no,
th1.tsf_no)
                                AND alloc2.status !='C')
                                OR  EXISTS (SELECT 'x'
                                             FROM ALLOC_HEADER
alloc3
                                            WHERE
alloc3.alloc_parent = alloc2.alloc_no
                                              AND alloc2.order_no
IN (th.tsf_no, th1.tsf_no)
                                       AND alloc3.status != 'C'
                                              AND ROWNUM = 1))
                           AND ROWNUM = 1)
       AND rv.driver_value         = th.tsf_no
       AND rv.num_threads          = TO_NUMBER(:ps_num_threads)
       AND rv.thread_val           = TO_NUMBER(:ps_thread_val)
     ORDER BY 1;
```

Scheduling Considerations

Processing Cycle: PHASE AD-HOC
Scheduling Diagram: N/A
Pre-Processing: N/A
Post-Processing: N/A
Threading Scheme: N/A (single threaded)