

Item Requisition Extraction [reqext]

Design Overview

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

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

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

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

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

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

updated to reflect the fact that stock is now reserved for transfer at the warehouse and expected at the store.

Scheduling Constraints

Processing Cycle:	PHASE 3
Scheduling Diagram:	Rplatupd, repladj, prepost ociroq and ociroq need to run before rext so that all replenishment calculation attributes are up to date. Posupld need to run before rext so that all stock information is up to date. Rplext should run after rext, since the ROQ for a warehouse is influenced by any transfers created.
Pre-Processing:	N/A
Post-Processing:	N/A
Threading Scheme:	DEPT

Restart Recovery

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

Program Flow

N/A

Shared Modules

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

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

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

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

Data Structures

repl_info_struct: Holds information fetched from the driving cursor.

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

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

tsfhead_struct: Used to buffer inserts to the TSFHEAD table.

tsfdetail_struct: Used to buffer inserts and updates to the TSFDETAIL table.

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

repl_results_struct: Used to buffer inserts to the REPL_RESULTS table.

domain_struct: Used to cache forecasting domain information.

Function Level Description

General Controlling Functions

main()

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

init()

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

driving_cursor()

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

process()

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

replenish_item()

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

place_tsf_line_item()

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

final()

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

Simple Pack Distribution and Transfer

build_pack_ratio()

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

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

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

first_ratio_pass()

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

next_ratio_pass()

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

adjust_pack_ratio()

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

add_primary_pack()

If an item is flagged to have a primary simple pack, that pack is the only one that will be transferred. This function adds the primary pack to the simple pack

distribution array and assigns it the full share of the ROQ. This is a support function for `build_pack_ratio()`.

get_multi_simple_pack()

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

get_single_sales_hist()

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

add_single_item()

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

calc_pack_dist()

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

calc_pack_tsf_qty()

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

Substitute Item Distribution and Transfer

calc_sub_dist()

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

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

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

- The store's ROQ, total ROQ, availability of the item, and total availability are all decremented by the amount just transferred to prepare for the next item-location's calculation.

get_sub_items()

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

add_master()

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

shift_subs()

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

Database DML Handling

post_all()

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

handle_tsf()

Controls handling of inserts and updates to the Transfer tables.

add_tsfhead()

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

add_tsfdetail()

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

get_next_seq_no()

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

post_tsf()

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

handle_item_loc()

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

add_item_loc()

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

post_item_loc()

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

handle_repl_results()

Controls posting of report information to the REPL_RESULTS table.

add_repl_results()

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

post_repl_results()

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

update_review_date()

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

PL/SQL Stored Procedure Calls

get_wh_current_avail()

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

next_transfer_number()

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

Domain Validation

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

Support Functions

copy_repl_to_store()

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

reset_store_struct()

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

reset_alt_item_struct()

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

Array Sizing

size_repl_info_struct()

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

size_store_struct()

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

size_alt_item_struct()

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

size_tsfhead_struct()

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

size_tsfdetail_struct()

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

size_item_loc_struct()

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

size_repl_results_struct()

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

Database Interaction

Tables Selected From:

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

Tables Inserted To:

REPL_RESULTS
TSFDETAIL
TSFHEAD

Tables Updated:

ITEM_LOC_SOH
REPL_ITEM_LOC
TSFDETAIL

I/O Specification

N/A

Technical Issues

N/A