

Batch Detail Design

I. Functional Area

Complex Deals Management

II. Module Affected

costcalc.pc

III. Design Overview

This program is responsible for maintaining information on the FUTURE_COST table based on records on the DEAL_SKU_TEMP table.

This batch program will calculate the net cost, net net cost, and dead net net cost for all items that are on the DEAL_SKU_TEMP table (which should contain all items or items in hierarchies on deals that are on the DEAL_QUEUE table, which will contain deals that are about to be approved, unapproved, or closed—any action that would potentially change which deals affect an item). Also items with new item-location relationships, future cost changes or merchandise hierarchy reclassifications or the cancellation of such events will appear here along with appearing on the RECLASS_COST_CHG_QUEUE table. All active deals for each item will be used in the calculation along with any future reclassification or cost change information. Once calculated, the costs will be inserted into the FUTURE_COST table.

If the Batch with Online Users indicator is set to 'Y', it will bulk lock the FUTURE_COST records before deleting the processed records. Information about the records that were not processed due to locking will be inserted to the BATCH_LOCK_LOG table. A check will also be performed to determine which permanent table the DEAL_SKU_TEMP view points to. The permanent table that is not the current view will be the one to be used throughout the batch program.

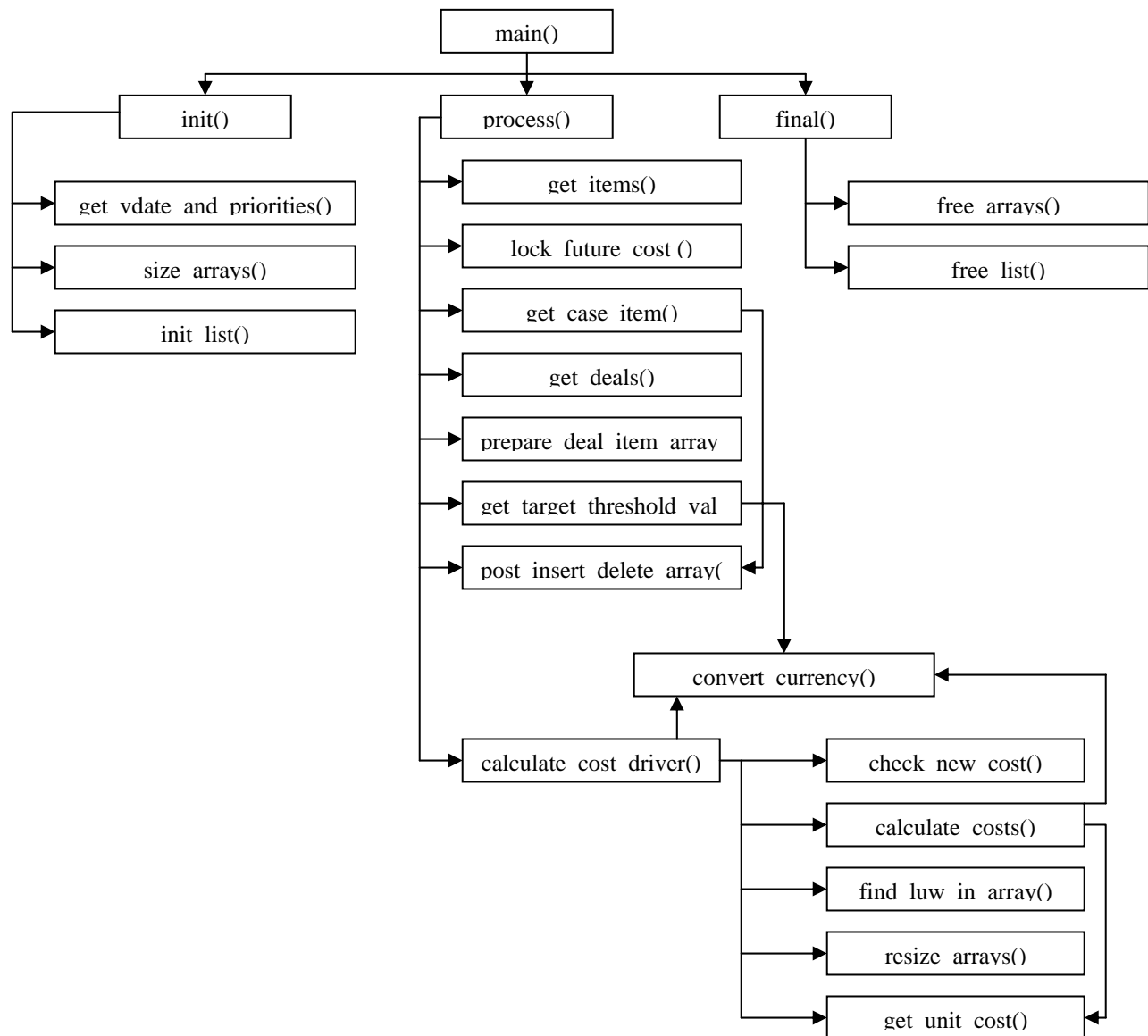
Logical unit of work: item/supplier/origin country/location/start date.

IV. Stored Procedures / Shared Modules (Maintainability)

CURRENCY_SQL.CONVERT – convert an amount in deal currency to the equivalent amount in supplier currency if necessary, or vice versa.



V. Program Flow



VI. Function Level Description

Main(): Standard Retek main function. Validates input parameters, calls init, process and final. Logs appropriate message.

Init():

- Retrieve the vdate from the period table (use as calculation date for inserts into deal_cost table), get priority indicators (deal_type_priority, deal_age_priority—these determine annual first vs. promotional first, and oldest first vs. newest first ordering for the driving cursor), btch_w_usr_ind from system_options by calling get_vdate_and_priorities().
- Allocate memory for the deal fetch and cost arrays by calling size_arrays() and initialize the linked list for deal target values by calling init_list().
- If the Batch with Online Users indicator is set to 'Y', costcalc.pc's records in the BATCH_LOCK_LOG will be deleted if the batch is run for the first time.



- Restart/recovery initialization by calling `retex_init()`.
- Call the `table_view_check()` function to determine which permanent table the `DEAL_SKU_TEMP` view points to.

Process():

- In a while loop:
 - Call `get_items()` to array fetch records from the `DEAL_SKU_TEMP` table.
 - Lock the corresponding records in the `FUTURE_COST` table if `system_options.btc_h_w_usr_ind` is Y.
 - Loop through the array fetched records and for each:
 - Call `get_case_item()` if the simple pack indicator for the current item is Y.
 - In a while loop:
 - Call `get_deals()` to array fetch deals for the current item.
 - Call `prepare_deal_item_array()` to copy the current item and the deals found for the current item into one single array.
 - Loop through the records in the array holding the deal item combinations and for each record:
 - Call `get_target_threshold()` if the current deal's threshold is not quantity (type Q).
 - Call `calculate_cost_driver()` to get the net, net net, and dead net net cost, and create an insert array that includes the net/net net/dead net net cost information AND the location information.
 - If commit point reached, call `post_insert_delete_records()` to insert the costs into the `FUTURE_COST` table for each LUW, and to delete processed records from the `DEAL_SKU_TEMP_A` or `DEAL_SKU_TEMP_B` table (depending on which permanent table the view `DEAL_SKU_TEMP` is currently not pointing to).
 - After each set of deals has been processed, call the restart commit logic.
 - After loop, call `post_insert_delete_records()` to make sure the final set of records get inserted too.

Get_items():

This function will return data from two separate driving cursors when it is called. The function masks all other db operation requirements from calling entity. The function keeps track of which cursor is being fetched and whether the cursor is open or not. The function will open the active cursor when called if the cursor is not open. The function will array fetch the cursor for data. If no data is found, the program will determine if there was no data at all, no data in this array fetch, or there was data and there was data in this fetch but not to the full array size. Depending on the circumstance, the function will either return a no records found indicator (last cursor fetched and no data found) or will attempt to open/array fetch the next still available cursor. If this second fetch returned records, the function will not return a set indicator for no data found, and therefore 'pretends' that the source is still dumping data.

Get_case_item():

This function will look up if the case item is the primary costing pack for an other item. If there is such an other item, this function gets it.

Get_deals():

This function will open/array fetch one of the four cursors that look up deals for an item base on values fetched from the `system_options` table in the `init ()` function. Signals to calling entity if the 'no data found' indicator was set or not. This function will not re-open a cursor until a previous open has been array fetched to no records found.

Prepare_deal_item_array():

This function will merge the current item with deals that were fetched for the current item into one single array. If the current item is different from the last call to this function, a core element is inserted into the target array (to maintain functionality requirements. We still need a record for the item in the merged array, even if no deals were found for the item.)

Check_new_cost():

Fetch from `RECLASS_COST_CHG_QUEUE` to check that the current item/supplier/origin country/location/start date combination has the right cost (which is initially the cost from `ITEM_SUPP_COUNTRY_LOC.UNIT_COST`, but this may be overwritten by this function if a cost change is found affecting this item/supplier/origin country/location before the item/supplier/origin country/location's start date).

Calculate_cost_driver():



This function will drive the process of calculating the net, net net, and dead net net cost, given information on all the deals that apply to a particular sku/supplier/origin country/start date. Each fetched record is passed on to the calculate_costs function to do the actual calculation for each LUW + deal/detail, that is, item/supplier/origin country/location/start date + deal component that applied. For each set of deals for a unique item/supplier/origin country/location/start date, the desired end result is to have one record on FUTURE_COST that will hold the item's costs.

- For each new LUW, reset the flag for 'Fixed Amt value type discount. 'Fixed Amt value type discount should only be applied once for each LUW + loc.
- For each new LUW, reset the merchandise/organization level for 'Exclusive deal class discount. 'Exclusive deal class discount should only be applied once for each LUW.
- Reset the net/net/dead net net costs according to the following rules:
 - If new LUW, set to supplier's original unit cost + RECLASS_COST_CHG_QUEUE cost if future cost change that is earlier than start date of this record is found. This is done by calling check_new_cost()
 - If the same LUW:
 - Check if the flag for 'Exclusive deal class discount is set (previous 'Exclusive discount applied)
 - If NO previous 'Exclusive discount applied, check if this is an 'Exclusive discount:
 - If yes, set net/net/dead net net costs to base cost (supplier's unit cost)
 - If no, set net/net/dead net net costs to costs of loc-independent discounts
 - If previous 'Exclusive discount applied check if this is an 'Exclusive discount with higher merch level or equal merch level but higher org level than the saved merch/org level (only apply the highest merch/org level 'Exclusive discount):
 - If yes, set net/net/dead net net costs to base cost (supplier's unit cost)
 - If no, skip this discount.
- If deal_id and deal_detail_id are not 0, call calculate_costs() to calculate net/net/dead net net costs.
- For the same LUW, the driving cursor has sorted the discounts by cost_appl_ind: 'N' first, 'NN' later, 'DNN' last. For each cost application level, the same business rules are followed. Find this LUW's record in the cost array and update the costs based on whatever new costs were returned from calculate_costs().
- If new LUW, increment the writing index of the cost array (we always write a record into the cost array to keep track of last calculated costs, but change to a new record only if the LUW is changed). Make sure if current item is a UPC pack, a second record is inserted into the cost array for the component item with cost of the case UPC divided by the pack qty.
- Write costs into the current indexed record of the cost array. Use the start_date from DEAL_SKU_TEMP as the active_date. VDATE should be used for the calc_date.

Calculate_costs(): This function performs the actual calculation of the new costs.

Inputs: index of fetch array, target threshold value, current net/net/dead net net costs

Outputs: calculated net/net/dead net net costs

The definition of different net costs is:

net cost = unit cost – components whose cost_appl_ind is 'N'

net net cost = net cost – components whose cost_appl_ind is 'NN'

dead net cost = net net cost – components whose cost_appl_ind is 'DNN'

pricing cost is the base cost, unless pricing_cost_appl_ind is 'Y' in the deal definition, in which case it will be the dead net cost.

Use the cost_appl_ind on deal_detail to figure out whether a deal component contributes to the net, net net, or dead net net cost (the records should already be sorted by cost_appl_ind) and what the initial costs are (initial cost are needed to process 'Cumulative deal class discounts with 'Percentage value type):

- If 'N', the initial net cost is the supplier's original unit cost, and need to update all 3 net costs with the calculated discount
- If 'NN', the initial net net cost is the current net cost, and need to update both net net cost and dead net net cost with the calculated discount.
- If 'DNN', the initial dead net net cost is the current net net cost, and need to update only the dead net net cost with the calculated discount.



Business rules that needs to be followed when applying discounts:

- Deal classes:
 - Cumulative discounts need to be applied to the original unit cost (2% off + 3% off = 5 %off original unit cost)
 - Cascade discounts need to be applied on the result thus far ("current cost")---take 2% off of the unit cost, then take 3% off of that price, for example
 - Deal value types (take N cost calculation for example):
 - for a % discount
 - If 'CS'cade:

$$\text{discount cost} = \text{unit cost} - (\text{unit cost} * \% / 100)$$
 - If 'CU'mulative:

$$\text{discount cost} = \text{unit cost} - (\text{initial unit cost} * \% / 100)$$
 - for an amt discount (first convert amount to be in supplier currency if necessary)

$$\text{discount cost} = \text{unit cost} - \text{amt} \quad (\text{amount discounts are per unit cost already})$$
 - fixed amt: if have fixed amount discount must start with THAT amount rather than the unit cost (convert to supplier currency if necessary)

$$\text{discount cost} = \text{fixed amt} \quad (\text{converted to supplier's currency if necessary})$$
 - quantity discount ("buy some get some at discount") (these are not allowed on rebates)

These are the most complicated. They affect the cost of the get item AND of the buy item, whose cost we also need to get. Both the get item and the buy item will be on deal_itemloc. You should only calculate the cost for whichever item you're presently on (if buy item, just calculate buy item cost; will get the free item separately later, or vice versa). The initial unit cost for the get item should be taken from deal_detail.qty_thresh_free_item_unit_cost (or, if that field is not populated, off of item_supp_country). Before any calculations are done, convert the unit costs into supplier currency if necessary. If a buy/get free discount is encountered, the following things need to happen:

 - Get the get item's initial unit cost. If the buy and get items are the same, then this item's unit cost is the get item's unit cost. If the buy and get items are not the same, then if this item is the get item, use this item's unit cost. Otherwise call get_unit_cost() to get the get item's unit cost.
 - Next get the buy and get item unit cost. The get item unit cost here is different from the above initial get item unit cost since this unit cost may be set from the deal_detail.free_item_unit_cost field if the get item is free (not percent off, or amount off or fixed amount.) So, if this item is the buy item, set the buy item unit cost to this item's unit cost. The get item unit cost is the deal_detail.free_item_unit_cost field if the get item is free (not percent off, or amount off or fixed amount), otherwise it is the buy item unit cost if the buy and get items are the same, if they are not the same, call get_unit_cost() to get it. If the get item unit cost was populated from deal_detail.free_item_unit_cost, call convert_currency() for the get item unit cost if the deal_detail.free_item_unit_cost was stored in a currency other than the get item's supplier's default currency, which is what this program will use for cost calculations. If this item is the get item, set its cost to the deal_detail.qty_thresh_free_item_unit_cost if the get type is free and call convert_currency() as necessary, otherwise set its cost to this item's unit cost. If the buy and get items are the same, set the buy item's unit cost to this item's unit cost. Otherwise call get_unit_cost() to get it.
 - Calculate the discount costs(for whichever is the current item, free or buy)
 - If qty_thresh_buy_target of the buy item < qty_thresh_buy_qty, stop; no discounts are applied
 - Otherwise, figure out how many get items we actually get.
 - If the qty_thresh_recur_ind is 'N':
 - If buy item is same as get item:

$$\text{get qty} = \text{deal_detail.qty_thresh_free_qty} \text{ if buy target} > \text{buy qty. Cut off get qty to buy target minus buy qty if necessary.}$$
 - Else:

$$\text{get qty} = \text{deal_detail.qty_thresh_free_qty} \text{ if buy target} > \text{buy qty}$$
 - If the qty_thresh_recur_ind is 'Y':
 - If buy item = free item:

$$\text{free qty} = \text{qty_thresh_free_qty} * \text{FLOOR}(\text{qty_thresh_buy_target} / (\text{qty_thresh_buy_qty} + \text{qty_thresh_free_qty}))$$
- Add any additional free quantity to get qty where additional free qty =

$$(\text{qty_thresh_buy_target} - (\text{qty_thresh_buy_qty} + \text{qty_thresh_free_qty}) * \text{FLOOR}(\text{qty_thresh_buy_target} / (\text{qty_thresh_buy_qty} + \text{qty_thresh_free_qty}))) -$$



- If buy item different from free item:

$$\text{free qty} = \text{FLOOR}(\text{qty_thresh_buy_target} / \text{qty_thresh_buy_qty}) * \text{qty_thresh_free_qty}$$

- 

Free_arrays(): Free the memory used by the arrays.

Init_list(): Initialize the linked list for target threshold values.

Free_list(): Free the memory used by the linked list for target threshold values.

Add_to_list(): Add a node made of deal/deal detail and the target value to the current position of the linked list.

Get_target_threshold_value(): Given a deal_id and deal_detail_id, fetch the target value from the deal_threshold table (the value where the target_level_ind is 'Y'). Since this function is often called multiple times for the same input (multiple ITEMS of the same deal/deal detail), a linked list is maintained to keep track of target threshold values for different deal/deal detail. The linked list is ordered by the deal/deal detail. This function first tries to get the value from the list (previously fetched from database), if found then job is done. Otherwise, fetch the target value for this deal/deal detail from database and call convert_currency() if the value is currency amount and the deal currency is different from the supplier's currency. The newly fetched value is then saved into the list by calling add_to_list(). Other maintenance functions for the linked list are init_list() (called in init) and free_list() (called in final).

Get_unit_cost(): For a given item/supplier/origin country/location/start date, get the unit cost from ITEM_SUPP_COUNTRY_LOC or RECLASS_COSTCHG_QUEUE depending on whether the latter has a record for the item/supplier/origin country/location/start date or not. If not, look the item/supplier/origin country/location up on the ITEM_SUPP_COUNTRY_LOC table and get the unit costs from there. Since usually the unit cost is fetched by the driving cursor, the function is only called for buy/get type discounts when an item's unit cost is needed.

Convert_currency(): Call CURRENCY_SQL.CONVERT() package to convert an amount in deal currency to equivalent amount in supplier's currency. (This should only be called if the currency types are different—usually they will be the same).

Post_insert_delete_records():

- Array delete all records from the FUTURE_COST table where the delete array's item/supplier/origin country/location matches the record in FUTURE_COST and FUTURE_COST's active date is higher or equal to the start date from the cost array. This step is necessary to prevent primary key violations during insert into the FUTURE_COST table.
- Array insert all records of the cost array into the FUTURE_COST table and array delete processed records, which are also all records of the cost array, from the DEAL_SKU_TEMP table.
- Delete record from DEAL_SKU_TEMP that is associated with the records in the cost array.

Final():

- Call free_arrays() and free_list().
- Call restart/recovery close logic by calling retek_close().

Lock_future_cost(): Locks the FUTURE_COST records based on the pa_item_fetch structure.

Check_lock_future_cost(): Checks if the FUTURE_COST records to be deleted are already locked by another process.

Table_view_check(): Checks which permanent table the DEAL_SKU_TEMP view points to.

VII. Input Specifications

'Table-To-Table'

Select data from:

Table Name	Column Name	Column Type	Transformation
PERIOD	VDATE	DATE	NONE
SYSTEM_OPTIONS	DEAL_TYPE_PRIORITY	VARCHAR2(3)	NONE



SYSTEM_OPTIONS	DEAL_AGE_PRIORITY	NUMBER(10)	NONE
SYSTEM_OPTIONS	BTCH_W_USER_IND	VARCHAR2(1)	NONE
ITEM_LOC	ITEM	VARCHAR2(25)	NONE
DEAL_SKU_TEMP_A	ITEM	VARCHAR2(25)	NONE
DEAL_SKU_TEMP_A	SUPPLIER	NUMBER(10)	N/A
DEAL_SKU_TEMP_A	ORIGIN_COUNTRY_ID	VARCHAR2(3)	NONE
DEAL_SKU_TEMP_A	START_DATE	DATE	NONE
DEAL_SKU_TEMP_A	LOCATION	NUMBER(10)	NONE
DEAL_SKU_TEMP_A	LOC_TYPE	VARCHAR2(1)	NONE
DEAL_SKU_TEMP_A	COST_APPL_IND	VARCHAR2(1)	NONE
DEAL_SKU_TEMP_A	PRICE_COST_APPL_IND	VARCHAR2(1)	NONE
SUPS	CURRENCY_CODE	VARCHAR2(3)	NONE
ITEM_SUPP_COUNTRY_LOC	UNIT_COST	NUMBER(20,4)	NONE
DEAL_SKU_TEMP_B	ITEM	VARCHAR2(25)	NONE
DEAL_SKU_TEMP_B	SUPPLIER	NUMBER(10)	N/A
DEAL_SKU_TEMP_B	ORIGIN_COUNTRY_ID	VARCHAR2(3)	NONE
DEAL_SKU_TEMP_B	START_DATE	DATE	NONE
DEAL_SKU_TEMP_B	LOCATION	NUMBER(10)	NONE
DEAL_SKU_TEMP_B	LOC_TYPE	VARCHAR2(1)	NONE
DEAL_SKU_TEMP_B	COST_APPL_IND	VARCHAR2(1)	NONE
DEAL_SKU_TEMP_B	PRICE_COST_APPL_IND	VARCHAR2(1)	NONE
DEAL_HEAD	DEAL_ID	VARCHAR2(10)	NONE
DEAL_HEAD	CURRENCY_CODE	VARCHAR2(3)	NONE
DEAL_HEAD	CREATE_DATETIME	DATE	NONE
DEAL_HEAD	TYPE	VARCHAR2(6)	NONE
DEAL_DETAIL	DEAL_DETAIL_ID	VARCHAR2(10)	NONE
DEAL_DETAIL	APPLICATION_ORDER	NUMBER(10)	NONE
DEAL_DETAIL	DEAL_CLASS	VARCHAR2(6)	NONE
DEAL_DETAIL	THRESHOLD_VALUE_TYPE	VARCHAR2(6)	NONE
DEAL_DETAIL	QTY_THRESH_BUY_ITEM	VARCHAR2(25)	NONE
DEAL_DETAIL	QTY_THRESH_BUY_QTY	NUMBER(12,4)	NONE
DEAL_DETAIL	QTY_THRESH_RECUR_IND	VARCHAR2(1)	NONE
DEAL_DETAIL	QTY_THRESH_BUY_TARGET	NUMBER(12,4)	NONE
DEAL_DETAIL	QTY_THRESH_GET_ITEM	VARCHAR2(25)	NONE
DEAL_DETAIL	QTY_THRESH_GET_QTY	NUMBER(12,4)	NONE
DEAL_DETAIL	QTY_THRESH_GET_TYPE	VARCHAR2(6)	NONE
DEAL_DETAIL	QTY_THRESH_GET_VALUE	NUMBER(20,4)	NONE
DEAL_DETAIL	QTY_THRESH_FREE_ITEM_UNIT_COST	NUMBER(20,4)	NONE
DEAL_ITEMLOC	MERCH_LEVEL	VARCHAR2(6)	NONE
DEAL_ITEMLOC	ORG_LEVEL	VARCHAR2(6)	NONE
DEAL_THRESHOLD	VALUE	NUMBER(20,4)	NONE
DEAL_THRESHOLD	UPPER_LIMIT	NUMBER(20,4)	NONE
RECLASS_COST_CHG_QUEUE	UNIT_COST	NUMBER(20,4)	NONE
DEAL_THRESHOLD	VALUE	NUMBER(20,4)	NONE

VIII. Output Specifications

'Table-To-Table'

Delete data from:

Document: costcalc Master Batch Design.doc
Created: 6/6/2001 3:06 PM by Kevin Hearnen
Last modified: 03/31/04 10:35 AM by zzdalyc

Page
8 of 10

Project Name: <insert project name>
System ID: <insert system ID>
Version #: <insert version #>

Retek Inc.
Midwest Plaza
801 Nicollet Mall
Minneapolis, MN 55402

Retek Confidential Information
Access to this information and documentation is permitted only for its authorized business purpose by authorized personnel subject to confidentiality and nondisclosure provisions.
Printed Material Valid Only As of Print Date 03/31/04 10:35 AM



Table Name	Column Name	Column Type	Transformation
DEAL_SKU_TEMP_A	N/A	N/A	N/A
DEAL_SKU_TEMP_B	N/A	N/A	N/A
FUTURE_COST	N/A	N/A	N/A

The following table will be inserted:

Table Name	Column Name	Column Type	Transformation
FUTURE_COST	ITEM	VARCHAR2(25)	N/A
FUTURE_COST	SUPPLIER	NUMBER(10)	N/A
FUTURE_COST	ORIGIN_COUNTRY_ID	VARCHAR2(3)	N/A
FUTURE_COST	ACTIVE_DATE	DATE	N/A
FUTURE_COST	CURRENCY_CODE	VARCHAR2(3)	N/A
FUTURE_COST	UNIT_COST	NUMBER(20,4)	N/A
FUTURE_COST	LOCATION	NUMBER(10)	N/A
FUTURE_COST	LOC_TYPE	VARCHAR2(1)	N/A
FUTURE_COST	START_IND	VARCHAR2(1)	N/A
FUTURE_COST	BASE_COST	NUMBER(20,4)	N/A
FUTURE_COST	NET_COST	NUMBER(20,4)	N/A
FUTURE_COST	NET_NET_COST	NUMBER(20,4)	N/A
FUTURE_COST	DEAD_NET_NET_COST	NUMBER(20,4)	N/A
FUTURE_COST	PRICING_COST	NUMBER(20,4)	N/A
FUTURE_COST	CALC_DATE	DATE	N/A

IX. Scheduling Considerations

Processing Cycle: Phase II (daily)

Scheduling Diagram: Must be run after ditinsrt.pc, and precostcalc.pc which populate the DEAL_SKU_TEMP table

Threading Scheme: SUPPLIER

X. Locking Strategy

Before processing a group of records, it will be locked first with the no wait clause. If this group of records includes a row that has already been locked by another application, the whole group will be skipped, a flag will be set, information about these records will be written to the batch_lock_log table, and a non-fatal error will be written in the log file. The batch will then continue processing the next group of records.

XI. Restart/Recovery

The module has inherent restart/recovery based on records on the DEAL_SKU_TEMP table.

XII. Performance Considerations

N/A



XIII. Security Considerations

N/A

XIV. Unit Test Considerations

See UTP.

XV. Design Assumptions

All UOMs and currency types in the program and in the target table FUTURE_COST are held in the supplier's primary UOMs and currency.

Before inserting into the FUTURE_COST table, the program will delete all records in FUTURE_COST for the item/supplier/origin country/location/start date and all records where the start date for the same item/supplier/origin country/location is higher than the start date from the cost array in the program. This step is necessary so that records down the line from the current item/supplier/origin country/location/start date on the FUTURE_COST table that are potentially influenced by this record get properly recalculated. This program will recalculate them since they are in DEAL_SKU_TEMP. They were inserted there by precostcalc.pc. Ditinsrt.pc will also insert records into DEAL_SKU_TEMP for LUWs that are covered under deals, which have been approved, unapproved, or closed.

XVI. Outstanding Design Issues

N/A

Issue Description	Priority	Resolution

XVII. Approval and Distribution

The detailed design should be approved by:

Title	Name
Design Lead	

The detailed design should be distributed to:

Title	Name
Quality Control Lead	

XVIII. Appendix

For further questions, contact:

Gabor Tozser

Siobhan McMahon

