

Batch Detail Design

I. Functional Area

Complex Deals Management

II. Module Affected

Precostcalc2.pc

III. Design Overview

This batch module is responsible for data maintenance tasks that are necessary before running costcalc.

Unprocessed records in RECLASS_COST_CHG_QUEUE drive this program. The driving cursor is a set of nine cursors whose goal is to make sure the DEAL_SKU_TEMP table is inserted for three scenarios:

- If an unprocessed record exists on RECLASS_COST_CHG_QUEUE, create record(s) on DEAL_SKU_TEMP for the RECLASS_COST_CHG_QUEUE event. For reclassification events no location is given, therefore these events are blown out to all item-location locations before inserting to DEAL_SKU_TEMP.
- If an unprocessed record exists on RECLASS_COST_CHG_QUEUE, create record(s) on DEAL_SKU_TEMP for any events on the FUTURE_COST table that the RECLASS_COST_CHG_QUEUE event affects (make the match using item, supplier, origin country id, location, and start date). For reclassification events (not cost change) no location is given and therefore these events are blown out to all item-location locations before seeking FUTURE_COST matches.
- If a record exists on DEAL_SKU_TEMP that affects one or more FUTURE_COST records insert for those records on FUTURE_COST into DEAL_SKU_TEMP to make sure they are recalculated.

The program will also update RECLASS_COST_CHG_QUEUE events that it processed so that the record's process flag reflects the fact that it has been processed. To avoid primary key violations in DEAL_SKU_TEMP, check if an item, location, supplier, origin country id, and start date combination exists on DEAL_SKU_TEMP in the driving cursors. When the program is done, it should delete any records from RECLASS_COST_CHG_QUEUE that did not get processed. (This happens when somebody did a cost change or reclassified a component item of a case UPC. This record will not be picked up by the driving cursor and should not remain on RECLASS_COST_CHG_QUEUE.) Also, all records that are of type G or N should also be deleted since once they have been inserted into DEAL_SKU_TEMP. There is no reason for them to remain in RECLASS_COST_CHG_QUEUE. This should improve the performance of the program by keeping the size of RECLASS_COST_CHG_QUEUE as small as possible.

The LUW of this module is a single record from any one of the driving cursors, which all pick up item/supplier/origin country/location/active date combinations from RECLASS_COST_CHG_QUEUE, and FUTURE_COST.

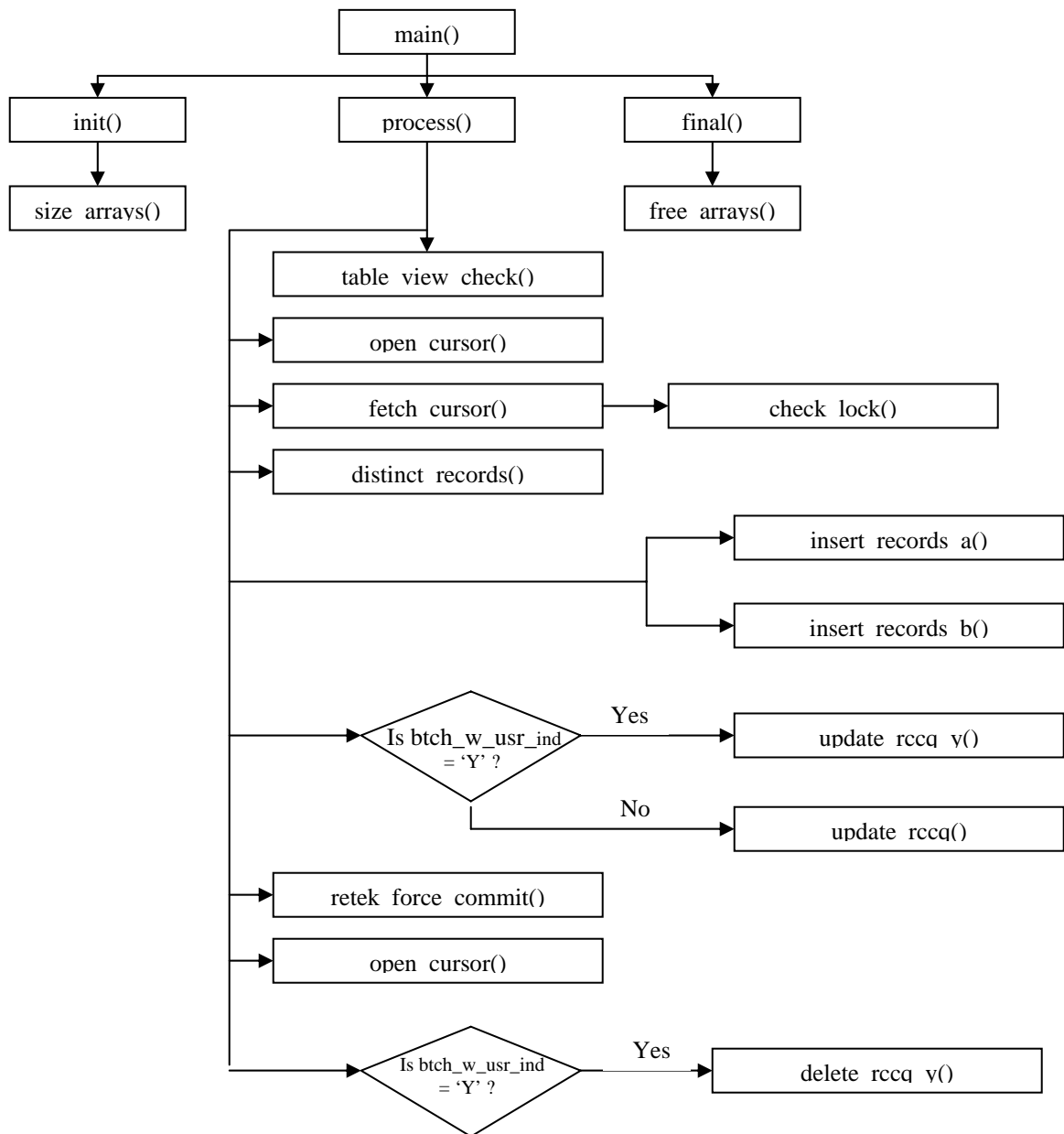
A new indicator has been added to the system_options table called Batch with Online Users (btch_w_usr_ind). If the Batch with Online Users indicator is set to 'Y' this means that users can be online at any given time. Table views were created such that the program will determine the current DEAL_SKU_TEMP table view and then process records from the non-online application view.

IV. Stored Procedures / Shared Modules (Maintainability)

N/A



V. Program Flow



VI. Function Level Description

Declare a fetch array struct to hold array-fetched records from driving cursors.

Declare driving cursors The first eight cursors pick up unprocessed events from RECLASS_COST_CHG_QUEUE that do not exist in DEAL_SKU_TEMP. There are eight of them because each handles a different type of record depending on whether location is given or not, and whether division is given or not. So, the combinations that are possible and therefore need to be fetched are:



| Location given? | Division given? |
|-----------------|-----------------|
| N | N |
| N | Y |
| Y | N |
| Y | Y |

Remember that the driving cursors for every one of the four above described combinations will be again split into two cursors depending on whether the location is a warehouse or a store. The reason so many driving cursors exist is because a single cursor that joined to all the tables at once was so slow that using such a cursor here was not an option. One more cursor is required: the ninth cursor will fetch records from the FUTURE_COST table that have the same item/supplier/origin country/location combination as a DEAL_SKU_TEMP record with an active date that is greater than the DEAL_SKU_TEMP record's start date. This cursor is necessary to ensure that when an event is inserted into DEAL_SKU_TEMP, later events for this event in the FUTURE_COST table may be affected by the event in DEAL_SKU_TEMP and should be re-inserted into DEAL_SKU_TEMP so costcalc can recalculate and reinsert them into FUTURE_COST.

The cursors should not pick up any primary cost pack component items since those are never inserted into DEAL_SKU_TEMP. (See Design Assumptions below.) Also this cursor should never pick up items that are not approved, transaction level, or items that are buyer packs. Deals batch programs do not process those.

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

Init(): Standard Retek init function. Calls retek_init() and size_arrays(). If this is a new start, set the restart cursor variable to 1. Declare a cursor that will fetch btch_w_usr_ind from SYSTEM_OPTIONS and vdate from PERIOD.

Process(): This is the main function that does all the work. It will array fetch each of the 9 cursors in the order of their definition above, one at a time until cursor is empty. For each cursor, fetched records will be array inserted into the DEAL_SKU_TEMP table view returned by the table_view_check function(DEAL_SKU_TEMP_A/DEAL_SKU_TEMP_B). Check the value of btch_w_usr_ind:

- If it is set to 'N':
 - RECLASS_COST_CHG_QUEUE will be updated after every second cursor fetch to set the process_flag of covered records to 'Y' and commit logic will be executed. After the last cursor has been fetched empty and all records were processed, delete from RECLASS_COST_CHG_QUEUE all records that belong to this thread and have not been picked up by any of the cursors (their process_flag is still 'N'). Also delete any records whose rec_type is 'G' or 'N'.
- If it is set to 'Y':
 - RECLASS_COST_CHG_QUEUE will be updated after every cursor fetch to set the process_flag of covered records to 'Y' and commit logic will be executed. After the last cursor has been fetched empty and all records were processed, call delete_rccq_y function to delete records from RECLASS_COST_CHG_QUEUE.

Call open_cursor to open the first cursor. (Cursors are opened and fetched in the order that they were declared in, which should be the same order they were described in above.)

- In a while loop (while the no records found indicator is not set):
 - Array fetch records from current driving cursor by calling fetch_cursor().
 - If a lock is encountered in the array of fetched records, skip processing and fetch the next set of records.
 - Call distinct_records() to copy distinct (item/supplier/origin country/location/start date) records from the fetch array (which has RECLASS_COST_CHG_QUEUE's level of distinctness: item/supplier/origin country/location/start date/record type) to the insert array.
 - If the table view is DEAL_SKU_TEMP_A, call insert_records_a() to array insert records from the insert array. Otherwise, call insert_records_b().
 - If btch_w_usr_ind is set to 'N':



- If we fetched empty a cursor with an even number call `update_rccq()` to update RECLASS_COST_CHG_QUEUE's `process_flags` to 'Y' for the records covered by the last two cursors.
 - If `btch_w_usr_ind` is set to 'Y':
 - Call `update_rccq_y()` to update RECLASS_COST_CHG_QUEUE's `process_flags` to 'Y' for the records covered by the current cursor fetch.
 - Call `retck_force_commit()` with the item, supplier and origin country, location, start date, and cursor number of the last record as the argument.
 - If the no records found indicator was set by `fetch_cursor()`, increment the cursor counter. If the cursor counter is under 10, call `open_cursor()` and reset the no records found indicator to 0 along with the total fetched records counter, which should be zeroed since we are about to start fetching from a new cursor. If the cursor number is 10 or above, we are done. Do not call `open_cursor()` and leave the no records found indicator set. This will drop the process out of the while loop.
- Delete from RECLASS_COST_CHG_QUEUE all records for this thread where the `process_flag` is 'N'. (This may happen if the component item of a case UPC was inserted into RECLASS_COST_CHG_QUEUE. The driving cursor will not pick these records up.) Also delete for records whose `rec_type` is 'G' or 'N' regardless of their `process_flag`. We do not need to keep those events around in RECLASS_COST_CHG_QUEUE.

Insert_records_a(): This function performs an array insert into DEAL_SKU_TEMP_A. Its argument is the current array size (number of records fetched into array).

Insert_records_b(): This function performs an array insert into DEAL_SKU_TEMP_B. Its argument is the current array size (number of records fetched into array).

Distinct_records(): This function copies records from the fetch array (which has RECLASS_COST_CHG_QUEUE's level of distinctness: item/supplier/origin country/location/start date/record type) to the insert array (which has DEAL_SKU_TEMP's level of distinctness: item/supplier/origin country/location/start date) so that primary key/index violation errors are avoided on subsequent inserts into DEAL_SKU_TEMP. This function will need local static strings to store any previous call's last copied DEAL_SKU_TEMP primary key so that it can be compared to the next call's first fetch array element's DEAL_SKU_TEMP primary key fields (item/supplier/origin country/location/start date). This is necessary if an array fetch cut the return of two identical item/supplier/origin country/location/start date records into two array blocks because of the max commit counter. We want to keep track of the fact that the item/supplier/origin country/location/start date record in question was already in the previous block. If this block still has that item/supplier/origin country/location/start date in the first field(s), don't copy it into the insert array again.

Open_cursor(): This function consists of a case-switch statement that depending on the argument of the function (an integer representing the number of the cursor that needs to be opened) will open the appropriate cursor.

Fetch_cursor(): This function consists of a case-switch statement that depending on the argument of the function (an integer representing the number of the cursor that needs to be fetched) will fetch the appropriate cursor and returns the number of records fetched, along with an indicator if the no data found indicator has been set by the fetch or not.

If the `btch_w_usr_ind` is set to 'Y', the number of records fetched is greater than 0 and the cursor is not 9:

- Insert into the RECLASS_CC_QUEUE_LOCK_TEMP table item/supplier/origin_country_id/start_date/location/rec_type/rowid records fetch from the driving cursor.
- Call `check_lock()`.
- Check the value returned by the `check_lock` function. If a lock is encountered, insert the array of records into the `batch_lock_log` table.
- Delete records from the SUP_DATA_LOCK_TEMP table.

Update_rccq(): This function will be called for every other cursor that was fetched empty and it will update RECLASS_COST_CHG_QUEUE records' `process_flag` to 'Y' covered by the last two cursors. This strange method of updating those records was necessary for restart/recovery reasons. Since two cursors are used to fetch for a single RECLASS_COST_CHG_QUEUE record (i.e.: no location or division given), one fetching for warehouse locations the other for store locations, the process flag of such a record can not be updated to Y until both cursors were fetched for this record. Otherwise the stores would not get picked up if we update the process flag right after warehouses were fetched. The update



statements cover all records in RECLASS_COST_CHG_QUEUE for which the most recent two cursors were fetched for, not only the ones for which the cursors actually returned records. A cursor may not return records, even though the RECLASS_COST_CHG_QUEUE records were completely valid, simply because the returned values already exist in DEAL_SKU_TEMP and the cursors have a not exists clause.

Update_rccq_y(): This function will be called for every cursor fetch where no record lock has been encountered and will update RECLASS_COST_CHG_QUEUE records' process flag to 'Y'.

Check_lock(): This function will declare a cursor that will lock records on RECLASS_COST_CHG_QUEUE table based on rowid.

Table_view_check(): This function returns the permanent table that is set as the current DEAL_SKU_TEMP table view.

Delete_rccq_y(): This function will declare a cursor that will retrieve RECLASS_COST_CHG_QUEUE records for this thread where the process_flag is 'N' or rec_type is 'G' or 'N' regardless of their process_flag.

- Open the c_delete_reclass cursor.
- In a while loop (while the no record found indicator is not set)
 - Array fetch records from the current driving cursor and insert into the RECLASS_CC_QUEUE_LOCK_TEMP table.
 - Call check_lock().
 - Check the value returned by the check_lock function. If a lock is encountered, insert the array of records into the batch_lock_log table and then delete records from the RECLASS_CC_QUEUE_LOCK_TEMP table.
 - Issue a commit.
- Close the c_delete_reclass cursor.

Size_arrays(): Sizes the fetch array to the commit size.

Free_arrays(): Frees fetch array.

Final(): Standard Retek final function. Calls free_arrays() and retek_close().

VII. Input Specifications

'Table-To-Table'

Select data from:

| Table Name | Column Name | Column Type | Transformation |
|------------------------|-------------------|--------------|---------------------------|
| RECLASS_COST_CHG_QUEUE | ITEM | VARCHAR2(25) | NONE |
| RECLASS_COST_CHG_QUEUE | SUPPLIER | NUMBER(10) | NONE |
| RECLASS_COST_CHG_QUEUE | ORIGIN_COUNTRY_ID | VARCHAR2(3) | NONE |
| RECLASS_COST_CHG_QUEUE | START_DATE | DATE | NONE |
| RECLASS_COST_CHG_QUEUE | LOCATION | NUMBER(10) | Only if REC_TYPE is not R |
| RECLASS_COST_CHG_QUEUE | DIVISION | NUMBER(4) | Only if REC_TYPE is R |
| RECLASS_COST_CHG_QUEUE | GROUP | NUMBER(4) | Only if REC_TYPE is R |
| RECLASS_COST_CHG_QUEUE | DEPT | NUMBER(4) | Only if REC_TYPE is R |
| RECLASS_COST_CHG_QUEUE | CLASS | NUMBER(4) | Only if REC_TYPE is R |
| RECLASS_COST_CHG_QUEUE | SUBCLASS | NUMBER(4) | Only if REC_TYPE is |



| | | | |
|------------------------|-------------------|--------------|---------------------------|
| | | | R |
| RECLASS_COST_CHG_QUEUE | REC_TYPE | VARCHAR2(1) | NONE |
| GROUPS | DIVISION | NUMBER(4) | Only if REC_TYPE is not R |
| DEPS | GROUP_NO | NUMBER(4) | Only if REC_TYPE is not R |
| ITEM_MASTER | DEPT | NUMBER(4) | Only if REC_TYPE is not R |
| ITEM_MASTER | CLASS | NUMBER(4) | Only if REC_TYPE is not R |
| ITEM_MASTER | SUBCLASS | NUMBER(4) | Only if REC_TYPE is not R |
| ITEM_MASTER | ITEM_PARENT | VARCHAR2(25) | NONE |
| ITEM_MASTER | ITEM_GRANDPARENT | VARCHAR2(25) | NONE |
| ITEM_MASTER | DIFF_1 | VARCHAR2(10) | NONE |
| ITEM_MASTER | DIFF_2 | VARCHAR2(10) | NONE |
| ITEM_MASTER | DIFF_3 | VARCHAR2(10) | NONE |
| ITEM_MASTER | DIFF_4 | VARCHAR2(10) | NONE |
| ITEM_LOC | LOC | NUMBER(10) | Only if REC_TYPE is R |
| ITEM_LOC | LOC_TYPE | VARCHAR2(1) | NONE |
| AREA | CHAIN | NUMBER(4) | Only if LOC_TYPE is S |
| REGION | AREA | NUMBER(4) | Only if LOC_TYPE is S |
| DISTRICT | REGION | NUMBER(4) | Only if LOC_TYPE is S |
| STORE | DISTRICT | NUMBER(4) | Only if LOC_TYPE is S |
| FUTURE_COST | ACTIVE_DATE | DATE | NONE |
| DEAL_SKU_TEMP | ITEM | VARCHAR2(25) | NONE |
| DEAL_SKU_TEMP | SUPPLIER | NUMBER(10) | NONE |
| DEAL_SKU_TEMP | ORIGIN_COUNTRY_ID | VARCHAR2(3) | NONE |
| DEAL_SKU_TEMP | DIVISION | NUMBER(4) | NONE |
| DEAL_SKU_TEMP | GROUP_NO | NUMBER(4) | NONE |
| DEAL_SKU_TEMP | DEPT | NUMBER(4) | NONE |
| DEAL_SKU_TEMP | CLASS | NUMBER(4) | NONE |
| DEAL_SKU_TEMP | SUBCLASS | NUMBER(4) | NONE |
| DEAL_SKU_TEMP | ITEM_PARENT | VARCHAR2(25) | NONE |
| DEAL_SKU_TEMP | ITEM_GRANDPARENT | VARCHAR2(25) | NONE |
| DEAL_SKU_TEMP | DIFF_1 | VARCHAR2(10) | NONE |
| DEAL_SKU_TEMP | DIFF_2 | VARCHAR2(10) | NONE |
| DEAL_SKU_TEMP | DIFF_3 | VARCHAR2(10) | NONE |
| DEAL_SKU_TEMP | DIFF_4 | VARCHAR2(10) | NONE |
| DEAL_SKU_TEMP | CHAIN | NUMBER(4) | NONE |
| DEAL_SKU_TEMP | AREA | NUMBER(4) | NONE |
| DEAL_SKU_TEMP | REGION | NUMBER(4) | NONE |
| DEAL_SKU_TEMP | DISTRICT | NUMBER(4) | NONE |
| DEAL_SKU_TEMP | LOCATION | NUMBER(10) | NONE |
| DEAL_SKU_TEMP | LOC_TYPE | VARCHAR2(1) | NONE |
| SYSTEM_OPTIONS | BTCH_W_USR_IND | VARCHAR2(1) | NONE |
| PERIOD | VDATE | DATE | NONE |



| | | | |
|----------------------------|---------------------|---------------|------|
| RECLASS_CC_QUEUE_LOCK_TEMP | ITEM | VARCHAR2(25) | NONE |
| RECLASS_CC_QUEUE_LOCK_TEMP | SUPPLIER | NUMBER(10) | NONE |
| RECLASS_CC_QUEUE_LOCK_TEMP | ORIGIN_COUNTRY_ID | VARCHAR2(3) | NONE |
| RECLASS_CC_QUEUE_LOCK_TEMP | START_DATE | DATE | NONE |
| RECLASS_CC_QUEUE_LOCK_TEMP | REC_TYPE | VARCHAR2(1) | NONE |
| RECLASS_CC_QUEUE_LOCK_TEMP | ROW_ID | ROWID | NONE |
| USER_OBJECTS | OBJECT_NAME | VARCHAR2(128) | NONE |
| USER_OBJECTS | OBJECT_ID | NUMBER | NONE |
| PUBLIC_DEPENDENCY | REFERENCE_OBJECT_ID | NUMBER | NONE |

VIII. Output Specifications

'Table-To-Table'

Delete from: RECLASS_COST_CHG_QUEUE, RECLASS_CC_QUEUE_LOCK_TEMP, and BATCH_LOCK_LOG.

Update data on:

| Table Name | Column Name | Column Type | Transformation |
|------------------------|--------------|-------------|----------------|
| RECLASS_COST_CHG_QUEUE | PROCESS_FLAG | VARCHAR2(1) | Set to Y. |

Insert into:

| Table Name | Column Name | Column Type | Transformation |
|----------------------------|-------------------|--------------|----------------|
| DEAL_SKU_TEMP | ITEM | VARCHAR2(25) | N/A |
| DEAL_SKU_TEMP | SUPPLIER | NUMBER(10) | N/A |
| DEAL_SKU_TEMP | ORIGIN_COUNTRY_ID | VARCHAR2(3) | N/A |
| DEAL_SKU_TEMP | START_DATE | DATE | N/A |
| DEAL_SKU_TEMP | DIVISION | NUMBER(4) | N/A |
| DEAL_SKU_TEMP | GROUP_NO | NUMBER(4) | N/A |
| DEAL_SKU_TEMP | DEPT | NUMBER(4) | N/A |
| DEAL_SKU_TEMP | CLASS | NUMBER(4) | N/A |
| DEAL_SKU_TEMP | SUBCLASS | NUMBER(4) | N/A |
| DEAL_SKU_TEMP | ITEM_PARENT | VARCHAR2(25) | N/A |
| DEAL_SKU_TEMP | ITEM_GRANDPARENT | VARCHAR2(25) | N/A |
| DEAL_SKU_TEMP | DIFF_1 | VARCHAR2(10) | N/A |
| DEAL_SKU_TEMP | DIFF_2 | VARCHAR2(10) | N/A |
| DEAL_SKU_TEMP | DIFF_3 | VARCHAR2(10) | N/A |
| DEAL_SKU_TEMP | DIFF_4 | VARCHAR2(10) | N/A |
| DEAL_SKU_TEMP | CHAIN | NUMBER(4) | N/A |
| DEAL_SKU_TEMP | AREA | NUMBER(4) | N/A |
| DEAL_SKU_TEMP | REGION | NUMBER(4) | N/A |
| DEAL_SKU_TEMP | DISTRICT | NUMBER(4) | N/A |
| DEAL_SKU_TEMP | LOCATION | NUMBER(10) | N/A |
| DEAL_SKU_TEMP | LOC_TYPE | VARCHAR2(1) | N/A |
| RECLASS_CC_QUEUE_LOCK_TEMP | ITEM | VARCHAR2(25) | N/A |
| RECLASS_CC_QUEUE_LOCK_TEMP | SUPPLIER | NUMBER(10) | N/A |
| RECLASS_CC_QUEUE_LOCK_TEMP | ORIGIN_COUNTRY_ID | VARCHAR2(3) | N/A |
| RECLASS_CC_QUEUE_LOCK_TEMP | START_DATE | DATE | N/A |
| RECLASS_CC_QUEUE_LOCK_TEMP | LOCATION | NUMBER(10) | N/A |



| | | | |
|----------------------------|--------------|--------------|-----|
| RECLASS_CC_QUEUE_LOCK_TEMP | REC_TYPE | VARCHAR2(1) | N/A |
| RECLASS_CC_QUEUE_LOCK_TEMP | ROW_ID | ROWID | N/A |
| BATCH_LOCK_LOG | PROGRAM_NAME | VARCHAR2(25) | N/A |
| BATCH_LOCK_LOG | TABLE_NAME | VARCHAR2(32) | N/A |
| BATCH_LOCK_LOG | KEY_VALUE1 | VARCHAR2(25) | N/A |
| BATCH_LOCK_LOG | KEY_VALUE2 | VARCHAR2(25) | N/A |
| BATCH_LOCK_LOG | KEY_VALUE3 | VARCHAR2(25) | N/A |
| BATCH_LOCK_LOG | KEY_VALUE4 | VARCHAR2(25) | N/A |
| BATCH_LOCK_LOG | KEY_VALUE5 | VARCHAR2(25) | N/A |
| BATCH_LOCK_LOG | KEY_ROWID | ROWID | N/A |
| BATCH_LOCK_LOG | LOCKED_DATE | DATE | N/A |
| BATCH_LOCK_LOG | THREAD_VAL | NUMBER(10) | N/A |

IX. Scheduling Considerations

This module must be run after ditinsrt and before costcalc in the deals batch cycle.

This module is multi-threaded by supplier. See volume-testing documentation for optimum thread value. (I suggest 15-30 threads.)

X. Locking Strategy

Solution #1 – Bulk Locking

Solution #2 – General Security

XI. Restart/Recovery

This program has restart recovery based on item/supplier/origin country/location/start date/cursor number and is multi-threaded by supplier.

XII. Performance Considerations

The driving cursors should be small enough to be executed fast. If the DEAL_SKU_TEMP table holds too many records as the program runs and the cursors' NOT EXISTS statements are slowing things down because of the size of DEAL_SKU_TEMP (which may very well happen), the only remaining performance enhancement that could go into the program is to simply fetch records from the cursors without checking for duplicates at the time of fetch and check for duplicates at time of insert or handle primary key violations in the insert as a non-fatal error. Or fetch all the records from all the cursors and discretize the fetched records in batch based on DEAL_SKU_TEMP's primary key. These "solutions" may not be a performance enhancement; the second suggestion simply takes away the load of discretizing records from the database and keeps it in the batch. Therefore other programs will not suffer a performance loss from the database being slow while precostcalc2 runs.

XIII. Security Considerations

N/A



XIV. Unit Test Considerations

When program is tested, tester will probably need to run costcalc and prepost for complete results.

- Run the program with btch_w_usr_ind set to 'N' and there are no locked records.
- Run the program with btch_w_usr_ind set to 'N' and there are locked records.
- Run the program with btch_w_usr_ind set to 'Y' and there are no locked records.
- Run the program with btch_w_usr_ind set to 'Y' and there are some locked records.
- Run the program with btch_w_usr_ind set to 'Y' and some of the locks on the records have been released.
- Run the program with btch_w_usr_ind set to 'Y' and all of the locks on the records have been released.
- Run the program with btch_w_usr_ind set to 'Y' and with restart/recovery.

See program's UTP for further instructions.

XV. Design Assumptions

Background: Costcalc is driven by the DEAL_SKU_TEMP table, which holds item/supplier/origin country/location/active date records that then need to be moved to the FUTURE_COST table with their costs at the date specified on DEAL_SKU_TEMP. Costcalc simply takes these item/supplier/origin country/location/active date records, calculates the cost for this combination and inserts the result into FUTURE_COST, along with a reset date for the item/supplier/country/location record if it has one. A reset date would be a deal closing that caused the cost change in the first place. This reset record on FUTURE_COST is simply the item/supplier/origin country/location/close date and the cost. This design has a few inherent problems. Some deals have no close dates and item reclassifications also have no close dates but potentially may change the cost of the item since different deals may apply due to the new merchandise hierarchy classification. Therefore a record on DEAL_SKU_TEMP with no close date may affect records on FUTURE_COST that have an active date later than the record on DEAL_SKU_TEMP. The solution is to move records from FUTURE_COST that are potentially affected by DEAL_SKU_TEMP records back to DEAL_SKU_TEMP, thus guaranteeing that they get recalculated and the correct price will be set as they are re-inserted into FUTURE_COST by costcalc. This process of checking for affected records and moving them into DEAL_SKU_TEMP is performed by precostcalc2. Also a new table was created called RECLASS_COST_CHG_QUEUE which holds reclassification, cost change, new item-location events for items along with a general record that simply holds an item which needs to be inserted into DEAL_SKU_TEMP (if an item reclassification was cancelled, we still need to send in a record to DEAL_SKU_TEMP to make sure the item's record on FUTURE_COST is re-calculated).

Case UPCs: Component items of case UPCs are never inserted into DEAL_SKU_TEMP. These items will be processed by costcalc as part of their case UPCs. Therefore if a case UPC component is inserted into RECLASS_COST_CHG_QUEUE, the driving cursor should ignore it and the record should be deleted. Also an item should be approved, transaction level and not a buyer pack to qualify for insertion into DEAL_SKU_TEMP.

Three quick examples of what appears in RECLASS_COST_CHG_QUEUE for a re-class, a cost change, a new item-location record or a general record:

Re-classification: I need an ITEM, -1 for LOCATION, SUPPLIER, ORIGIN_COUNTRY_ID, START_DATE, DIVISION, GROUP_NO, DEPT, CLASS, SUBCLASS, REC_TYPE = 'R'.

Cost change: I need an ITEM, LOCATION, SUPPLIER, ORIGIN_COUNTRY_ID, START_DATE, REC_TYPE = 'C'.
(For warehouse locations, only insert virtual warehouses!)



New item location: I need an ITEM, LOCATION, SUPPLIER, ORIGIN_COUNTRY_ID, START_DATE, REC_TYPE = 'N'. (For warehouse locations, only insert virtual warehouses!)

General: Primary key fields only. (These records appear as placeholders for cancelled events. For example, the merchandiser may cancel a future reclassification event in which case the original event's record on RECLASS_COST_CHG_QUEUE would be updated to have a rec_type of 'G' and a process_flag of 'N'. This results in precostcalc2 reinserting this event into DEAL_SKU_TEMP, from there costcalc will re-insert the event into FUTURE_COST. Thus external systems that exported the event earlier will see the event again in FUTURE_COST with potentially a new cost and can export again if necessary. Once such a record has been migrated to FUTURE_COST, there is no need to keep it in RECLASS_COST_CHG_QUEUE too. It may be deleted. Same is true for records with a rec_type 'N'.

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

Jeff Touhey

Siobhan McMahon

Craig Lindner

