

Price Change POS Extract [pcdnld]

Design Overview

The Price Change Download (pcdnld) module selects permanent price change records, which are set to go into effect a pre-determined number of days after today. The details of these records are then written out to the POS_MODS table, which is used as an interface point to with the point of sale system. The number of days before the price change goes into effect that the details are downloaded to the POS system is determined by the system option pos_extract_days on UNIT_OPTIONS. After this program has processed a price change, the status of the price change is changed to Extracted.

If the Batch with Online Users indicator is set to 'Y', it will bulk lock the price_susp_head records first to check for record locking for a given set of records before performing the necessary updates. Although the price_susp_detail table is updated, bulk locking will not be performed in price_susp_detail since if the price_susp_head table is locked, so will the price_susp_detail table and vice versa. If the records are locked by another application, they will not be processed, thus an update to the price_susp_head and price_susp_detail tables for the set of records will not happen. Otherwise, the records will be processed and an update to the price_susp_head and price_susp_detail tables for the set of records will happen.

Table	Index	Select	Insert	Update	Delete
PERIOD	No	Yes	No	No	No
UNIT_OPTIONS	No	Yes	No	No	No
PRICE_SUSP_HEAD	Yes	Yes	No	Yes	No
PRICE_SUSP_DETAIL	Yes	Yes	No	Yes	No
V_RESTART_PRICE_CHG	Yes	Yes	No	No	No
PRICE_ZONE_GROUP_STORE	Yes	Yes	No	No	No
ITEM_ZONE_PRICE	Yes	Yes	No	No	No
POS_MODS	No	No	Yes	No	No
CLEAR_SUSP_HEAD	Yes	Yes	No	No	No
CLEAR_SUSP_DETAIL	Yes	Yes	No	No	No
CLEAR_RESET_CALC	Yes	Yes	No	No	No
SYSTEM_OPTIONS	No	Yes	No	No	No
BATCH_LOCK_LOG	No	No	Yes	No	Yes
PRICE_DOWNLOAD_LOCK_TEMP	No	Yes	Yes	No	Yes

Scheduling Constraints

Processing Cycle:	Phase 1
Scheduling Diagram:	N/A
Pre-Processing:	N/A
Post-Processing:	Pccrdnld should run only after this module has completed
Threading Scheme:	Price change

Restart Recovery

The restart string is price change + ITEM + store.

Records will be fetched from an outer loop into arrays (sized according to restart_max_ctr on RESTART_CONTROL) and each record will be individually processed in a loop. Commits will occur after each record in the for-loop has been processed and before the next set of records are fetched into the arrays. This can happen because status is part of the select criteria on both the price change header and detail tables.

Program Flow

N/A

Shared Modules

TICKET_SQL.PRICE_CHANGE

Function Level Description

init()

This function will, in addition to performing standard restart tasks, fetch the Batch with Online Users indicator (btch_w_usr_ind) from the system_options table, current date from the period table and the number of days buffer required by the point of sales system for pricing modifications (pos_extract_days). Furthermore, if the btch_w_usr_ind is set to 'Y', it will delete all the records in the batch_lock_log table where the program name is equal to 'pcdnld' and the thread value is equal to the current thread value being used.

process()

This function selects the impending price change records (price change + ITEM + store combinations) for processing. The records are selected into arrays (size determined by commit_max_ctr on the RESTART_CONTROL table) and then each record in the array is processed by a for-loop.

Before going inside the internal loop, it will first check if the btch_w_usr_ind is 'Y'. If the btch_w_usr_ind is set to 'Y', it will insert all the records retrieved from the driving cursor into the price_download_lock_temp global temp table to check for locking errors by calling the check_lock() function. If there is at least one record locked from the set of records retrieved, all the records retrieved will be dumped into the BATCH_LOCK_LOG table for reporting purposes, the batch will not continue processing the current set of records retrieved by not going inside the internal loop and the batch will retrieve another set of records. The checking for locking issues is done for every set of records retrieved depending on the commit max counter. Otherwise, if the btch_w_usr_ind is set to 'N', it will continue processing the given set of records without checking for locking issues.

Inside this internal loop the ITEM information is retrieved from the ITEM_MASTER table (item, status, & pack_ind). If the status is approved, then process_item is called. If the price change number has changed since the last check then the psh_update() function is called to update the header status. When the PRICE_SUSP_DETAIL rowid changes then a row is added to an array that will update the status of the PRICE_SUSP_DETAIL table.

After the internal loop has processed the individual records, the `pos_mods_insert()` function is called to insert POS_MODS records created in the `process_item()` function and the `price_susp_detail` rows that have completed processing are updated by calling the `psd_update()` function.

process_item()

This function will select the current/old retail pricing information from ITEM_ZONE_PRICE. First it is determined whether the item is on clearance at the price change location by calling the `item_on_clearance()` function. If it is, then the function is exited (price information is not retrieved and a POS_MODS record is not written). If the price change is a regular price change then the information can be selected using the zone information (`zone_id` & `zone_group_id`) on the `price_change` tables. Otherwise, the “old” zone information must be retrieved from the PRICE_ZONE_GROUP_STORE table based on the store location of the price change. This table is joined with the ITEM_ZONE_PRICE table to retrieve old retail information. Once the old retail information is retrieved then an array to insert into POS_MODS table is filled in the `fill_pos_mods()` function.

After the call to the `fill_pos_mods` function, the package `TICKET_SQL.PRICE_CHANGE` is called to create the appropriate records for printing tickets based on the price change.

fill_pos_mods()

The main task of this function is to fill elements of arrays with information that will be inserted into the POS_MODS table. In addition to filling the arrays, it will also resize the insert (incrementing the size with the `commit_max_ctr` on RESTART_CONTROL) arrays when the array indexes will be extended.

pos_mods_insert()

Insert into POS_MODS from the insert arrays. The base insert adds item and pricing information derived from the price change tables. An additional insert adds reference item information to the POS_MODS table.

Is_item_on_clearance()

This function checks the clearance suspense and reset tables to see if the current item is on clearance at the current store.

size_input_arrays()

This function sizes two different sets of arrays. The arrays into which the price change information is fetched are allocated memory. Also the array used to hold the PRICE_SUSP_DETAIL rowids whose status will be updated is sized here. The `commit_max_ctr` field, on the RESTART_CONTROL table, determines the size of the arrays.

psd_update()

The PRICE_SUSP_DETAIL table is updated, the status set to ‘E’, for all of the records whose rowids are in the update array.

check_lock()

This function will check if a given set of records in the `price_susp_head` table are locked by another application. If at least one record from the set of records is locked, this function will write a record locking error in the error log and it will return 54. Otherwise, it will return 0 and continue processing. This

function is called for every set of records fetched, depending on the commit max counter.

I/O Specification

N/A

Technical Issues