

Product Security Rebuild [sprdrbld]

Design Overview

The security features being added to RMS will be maintained in the batch cycle. With each run, the changes made to the data in RMS will be brought under the security features of RMS through the running of 3 batch programs. Sprdrbld.pc will handle the maintenance for the product security data. Items will have different update/select attributes for a given user for any of a number of different functional areas like 'Pricing' or 'Clearances'. For each run, the program will use the security data defined for the user/group/functional area/merchandise level to define whether a user can select or update every single Item covered by the defined rules. The functional document describes the architecture of the security features and how it works. Rules that have a smaller scope overwrite those with a broader scope. For example, a user is assigned to two groups -- one of the groups has no update capability for a given department, while the other group allows updating for a specific class within that department. Which applies? The rule with the lowest item hierarchy in its definition is the rule granting the update capability for the class. Therefore, for every Item in the department and in the class will be allowed to update. For the rest of the items in the department, no updating will be allowed. In addition, if there are conflicting security definitions at the same hierarchy level because a user is associated with more than one group, the user is, as expected, granted the capability.

Performance is a crucial consideration for this program as it involves writing records for different functional areas at the item level for every user in the system. To accomplish this task as efficiently as possible, the program should be built as follows. It will be multi-threaded by department, and use restart_recovery. In the Init routine, an array which will closely resemble the final destination security table, will be sized to handle all the Items in the particular thread running. This array will be loaded with all the Items and used repeatedly for every user/functional area combination. There will be an additional indicator (in addition to the select/update indicators) that will keep track of which Items have a rule affecting them and have therefore been "touched". Each rule will affect certain items in the array and their attributes may be changed multiple times. When they are changed, this indicator will be raised. After all the rules are processed for a given user/functional area, the data in the array that has the "touched" indicator raised will be written out to a SQL Loader file and its indicator reset. This cycle will be repeated until all users and functional areas are exhausted.

This program works by using a view that points to a table with the complete security data. If the Batch with Online Users indicator is set to 'Y', the batch runs and rebuilds the security data, and inserts it to a second permanent table. After it is rebuilt, the view is modified to point to the second permanent table and the first table is truncated. If the Batch with Online Users indicator is set to 'N', and the rebuilt security data is insert into the same table view, which was truncated by the prepost program.

TABLE	INDEX	SELECT	INSERT	UPDATE	DELETE
SEC_USER_GROUP	No	Yes	No	No	No
SEC_GROUP_PROD_M ATRIX	No	Yes	No	No	No
V_RESTART_DEPT	No	Yes	No	No	No
ITEM_MASTER	No	Yes	No	No	No

SYSTEM_VARIABLES	No	Yes	No	No	No
USER_OBJECTS	No	Yes	No	No	No
PUBLIC_DEPENDENCY	No	Yes	No	No	No

Scheduling Constraints

Processing Cycle: Daily

Scheduling Diagram: Must run batch program prepost.pc with parameters *sprdrbld pre* , *sprdrbld.pc*, use SQL load control file *sprdrbld.ctl* to load the output file from *sprdrbld.pc* to database and run *prepost.pc* with parameters *sprdrbld post* in series.

Pre-Processing: Prepost with parameters: *sprdrbld pre*

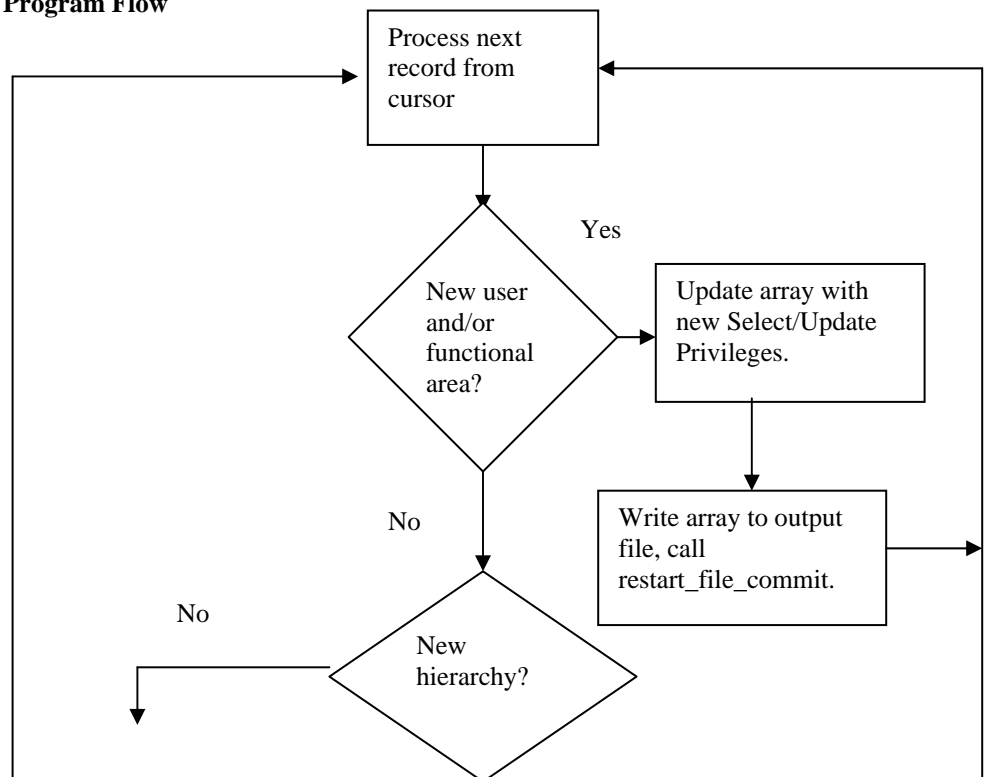
Post-Processing: Prepost with parameters: *sprdrbld post*

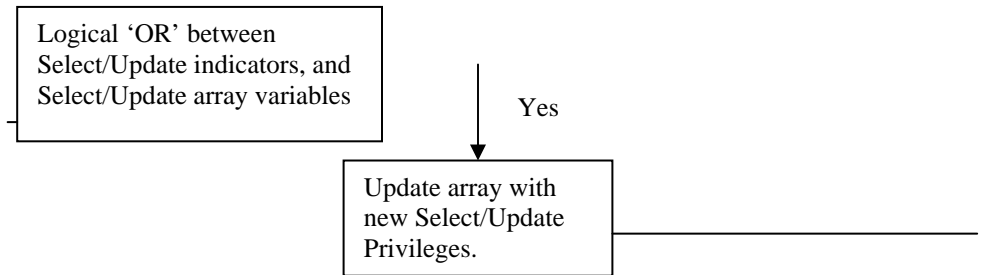
Threading Scheme: Department

Restart Recovery

The logical unit of work for location security rebuild will be the user-functional area (column_code). Restart/recovery will be based on the user-functional area. The restart commit counter will need to be carefully determined by each client according to the number of departments that will be affected by the product security rebuild. Large product security rebuilds with thousands of items need smaller commit counters to avoid reprocessing large amounts of data in the event of program failure. Small location security rebuilds with small amount of items can have much larger commit counters since fewer rows will be inserted into the database each time for one user-functional area.

Program Flow





Shared Modules

N/A

Function Level Description

Main()

Init()

- Check SYSTEM_VARIABLES.update_prd_sec_ind. If the indicator is not set then the program exit normally without further processing.
- Call retek_init() to get restart-recover variables.
- Fetch SYSTEM_OPTIONS.btch_w_usr_ind into a global variable.
- Get_total_skus()
Get total items in the current thread.
- Size_item_array()
Size item array based on the number of items in the current thread. The item array includes dept, class, subclass, level1, level2, level3, select_ind, update_ind and touched columns.
- Load_item_array()
Load all items in the current thread to the item array.
- Table_view_check()
Checks which permanent table is currently set as the view.

Process()

The driving cursor is ordered to return records defining rules for entire department first, and then those for class, and on down. The records are processed in that order. That is to say, first work with the department level rules, then move to the more specific rules so that the rules with the smaller scope take priority over the higher level rules.

- Call size_rule_array() to allocate memory for arrays that store security rules.
- Open the driving cursor in a while loop. Fetch the data into rule array.
- Call set_null_to_field() to set fields to null when those fields' indicators are -1 in the rule array.
- Check if this is a second array fetch or greater, if yes, call process_record() to process the last record in last array fetch and the first record in current array fetch.
- Open a for loop
 - Call process_record() to process the current and last record.

- End of for loop
- Copy the last record in the current array fetch to last rule array. Since the last record of an array fetch hasn't been processed until compared to the first record of the next array fetch. However, with each new array fetch, the last record of the previous array fetch is overwritten. Thus here it needs to be copied.
- End of while loop.

Size_rule_array()

This function allocates memory for arrays that store security rules based on the maximum commit count set in table restart_control table.

Set_null_to_field()

This function loops through all the records in rule array and set a field to null when the field's indicator is -1.

Process_record()

This function does the majority of the processing. The data from the driving cursor is ordered by dept, class, subclass, level1, level2, and level3 such that the department level rules are selected first, then the class level, etc. Also, all rules for a particular merchandise hierarchy will be grouped together and processed so that a single security rule will be decided for that particular hierarchy. When multiple records do occur at the same level, the logical OR will be used to determine whether to grant update/select privileges.

- Compare the user/functional area of the current record and the last record :
 - If it isn't new:
 - Compare the hierarchy of the current record and the last record:
 - If it isn't new, call logical_or_indicators() to update the current record's select and update indicators according to the logical 'OR' between the current and last records' indicators.
 - If it is new, call update_array() to blow security rule down to the SKU level according to the last record rule.
 - If it is new:
 - Call update_array() to blow security rule down to the item level according to the last record rule.
 - Call write_array() to output the security rules of last record's user/functional area (down to item level) to SQL load file.
 - Call retek_force_commit() to set book mark in the restart_bookmark table.

Logical_or_indicators()

This function updates the input current record's select and update indicators according to the logical 'OR' between the input current and last records' indicators. For example, if the current record's select indicator is 'N', the last record's select indicator is 'Y', then the current record's select indicator is updated to 'Y'; If the current record's select indicator is 'N', the last record's select indicator is 'N', then the current record's select indicator is kept untouched('N'). If the current record's select indicator is 'Y', no matter what last record's select indicator is, the current record's select indicator is kept untouched('Y'). So does update indicator.

Update_array()

This function updates the item array according to the input security rule. There are five kinds of security rules. They are department, class, subclass, style and level1, level2, and level3 security rules.

- If the input rule is a department level security rule, then loop through the item array, for all the items within the department, set the select_inds and update_inds equal to the input rule's select_ind and update_ind, respectively. Set all three touched indicators of each processed row to 'Y'.
- If the input rule is a class level security rule, then loop through the item array, for all the items within the class, set the select_inds and update_inds equal to the input rule's select_ind and update_ind, respectively. Set all three touched indicators of each processed row to 'Y'.
- If the input rule is a subclass level security rule, then loop through the item array, for all the items within the subclass, set the select_inds and update_inds equal to the input rule's select_ind and update_ind, respectively. Set all three touched indicators of each processed row to 'Y'.
- If the input rule is a level1 level security rule, then loop through the item array, for all the items corresponding to the level1 item (the item, and all its children and grandchildren), set the select_inds and update_inds equal to the input rule's select_ind and update_ind, respectively. Set all three touched indicators of each processed row to 'Y'.
- If the input rule is a level2 level security rule, then loop through the item array, for all the items corresponding to the level2 item (the item, and all its children), set the select_inds and update_inds equal to the input rule's select_ind and update_ind, respectively. Set the level2 and level3 touched indicators of each processed row to 'Y'.
- If the input rule is a level3 level security rule, then loop through the item array and set the select_inds and update_inds equal to the input rule's select_ind and update_ind, respectively. Set the level3 touched indicator of each processed row to 'Y'.

Write_array()

This function writes out rows with touched indicator equals 'Y' in the SKU array to SQL load file.

final():

restart/recovery close

I/O Specification

Each row of the output SQL load file outputfilename.dat corresponds to one record row in the sec_user_prod_matrix table. The format of the output file is as follows:

Permanent_table;column_code;user_id;item;select_ind;update_ind

Example:

A;PPRM;JOHN;10007986;N;N
A;PPRC;CLINTON;10001000;Y;N
A;PPRM;CLINTON;10007986;Y;Y
...

Technical Issues

N/A