

## **Location Security Rebuild [slocrbld]**

### **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. Slocrbld.pc will handle the maintenance for the location security data. Locations 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/organizational level to define whether a user can select or update every single location 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 region, while the other group allows updating for a specific store within that region. Which applies? The rule applying to the lowest level in the organization hierarchy is the rule granting the update capability for the store. Therefore, for the one store in the region, updates will be allowed. For the rest of the locations in the region, 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.

To accomplish this task as efficiently as possible, the program should be built as follows. It will be multi-threaded by store/warehouse, 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 locations in the particular thread running. This array will be loaded with all the locations 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 locations have a rule affecting them and have therefore been "touched". Each rule will affect certain locations 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 sec\_user\_loc\_matrix table 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_LOC_MATRIX	No	Yes	No	No	No
V_RESTART_STORE_WH	No	Yes	No	No	No
REGION	No	Yes	No	No	No

DISTRICT	No	Yes	No	No	No
STORE	No	Yes	No	No	No
WH	No	Yes	No	No	No
SYSTEM_VARIABLES	No	Yes	No	No	No
SEC_USER_LOC_MAT RIX_A	No	No	Yes	No	No
SEC_USER_LOC_MAT RIX_B	No	No	Yes	No	No
SYSTEM_OPTIONS	No	Yes	No	No	No
USER_OBJECTS	No	Yes	No	No	No
PUBLIC_DEPENDENC Y	No	Yes	No	No	No

## Scheduling Constraints

Processing Cycle: Daily

Scheduling Diagram: Must run batch program `prepost.pc` with parameters `slocrbld pre`, `slocrbld.pc` and `prepost.pc` with parameters `slocrbld post` in series.

Pre-Processing: Prepost with parameters: sloclbld pre

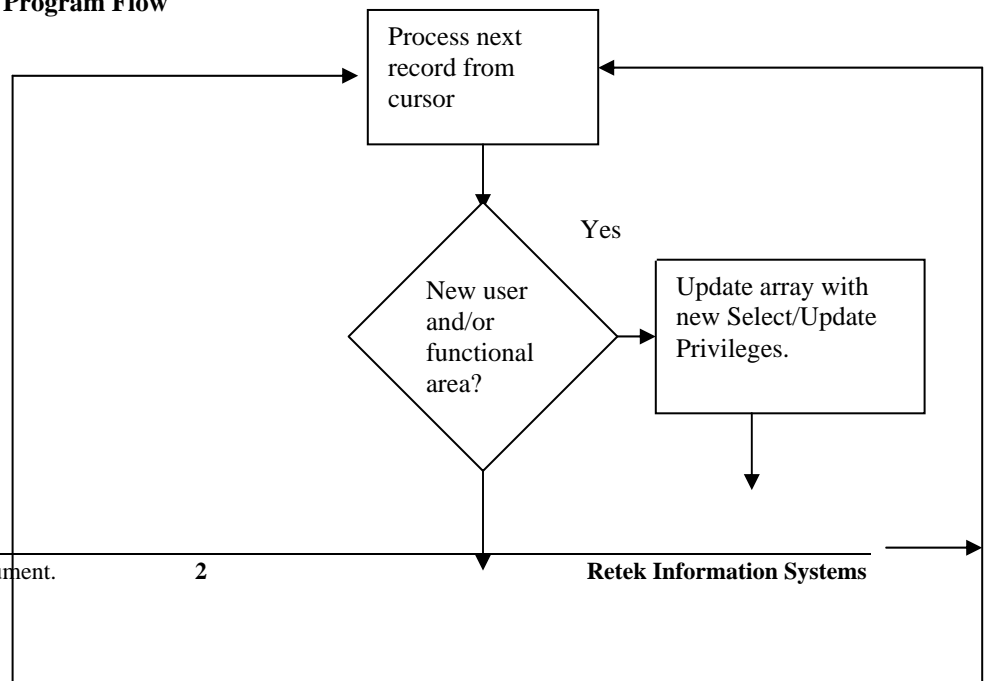
Post-Processing:                      Prepost with parameters: slocrebld post

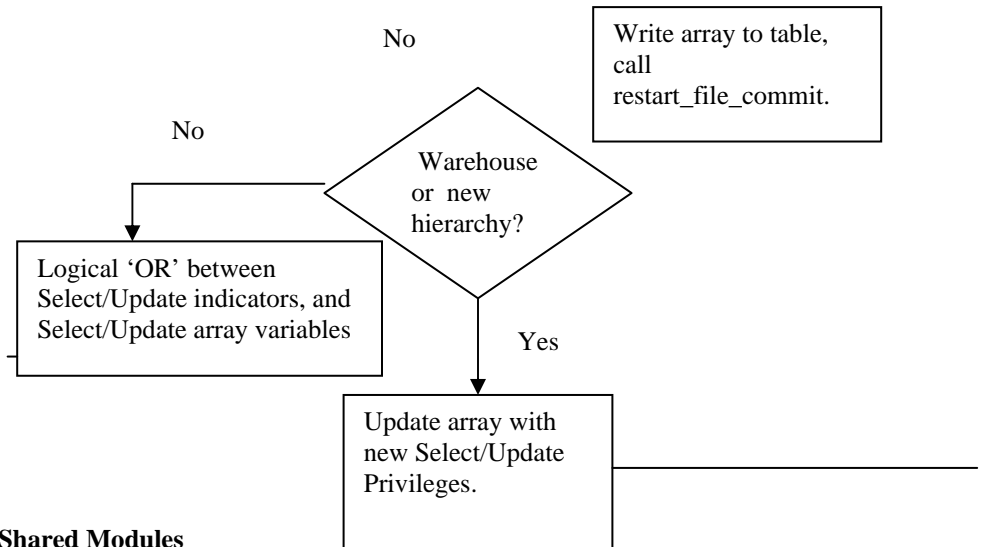
Threading Scheme:                      Store/Warehouse

## 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 store/warehouse combinations that will be affected by the location security rebuild. Large location security rebuilds with thousands of stores/warehouses 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 stores/warehouses 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

#### Driving Cursor:

```
SELECT distinct u.user_id,
    l.column_code,
    l.region,
    l.district,
    l.store,
    l.wh,
    l.select_ind,
    l.update_ind
FROM sec_user_group u,
    sec_group_loc_matrix l,
    district d,
    store s,
    v_restart_store_wh v
WHERE u.group_id = l.group_id
    AND l.region = d.region
    AND l.district is NULL
    AND v.driver_value = s.store
    AND v.num_threads = TO_NUMBER(:ps_restart_num_threads)
    AND v.thread_val = TO_NUMBER(:ps_restart_thread_val)
    AND d.district = s.district
    AND (u.user_id > NVL(:ps_restart_user, '-999')
        OR (u.user_id = :ps_restart_user
            AND l.column_code > :ps_restart_column_code))
UNION
SELECT distinct u.user_id user_id,
    l.column_code column_code,
    l.region region,
    l.district district,
    l.store store,
    l.wh wh,
    l.select_ind,
    l.update_ind
FROM sec_user_group u,
    sec_group_loc_matrix l,
    store s,
```

```
        v_restart_store_wh v
WHERE u.group_id = l.group_id
      AND l.district = s.district
      AND l.store is NULL
      AND v.driver_value = s.store
      AND v.num_threads = TO_NUMBER(:ps_restart_num_threads)
      AND v.thread_val = TO_NUMBER(:ps_restart_thread_val)
      AND (u.user_id > NVL(:ps_restart_user, '-999')
           OR (u.user_id = :ps_restart_user
               AND l.column_code > :ps_restart_column_code))
UNION
SELECT distinct u.user_id user_id,
               l.column_code column_code,
               l.region region,
               l.district district,
               l.store store,
               l.wh wh,
               l.select_ind,
               l.update_ind
FROM sec_user_group u,
     sec_group_loc_matrix l,
     v_restart_store_wh v
WHERE u.group_id = l.group_id
      AND l.store = v.driver_value
      AND v.num_threads = TO_NUMBER(:ps_restart_num_threads)
      AND v.thread_val = TO_NUMBER(:ps_restart_thread_val)
      AND (u.user_id > NVL(:ps_restart_user, '-999')
           OR (u.user_id = :ps_restart_user
               AND l.column_code > :ps_restart_column_code))
UNION
SELECT distinct u.user_id user_id,
               l.column_code column_code,
               l.region region,
               l.district district,
               l.store store,
               l.wh wh,
               l.select_ind,
               l.update_ind
FROM sec_user_group u,
     sec_group_loc_matrix l,
     v_restart_store_wh v
WHERE u.group_id = l.group_id
      AND v.driver_value = l.wh
      AND v.num_threads = TO_NUMBER(:ps_restart_num_threads)
      AND v.thread_val = TO_NUMBER(:ps_restart_thread_val)
      AND (u.user_id > NVL(:ps_restart_user, '-999')
           OR (u.user_id = :ps_restart_user
               AND l.column_code > :ps_restart_column_code))
ORDER BY 1, 2, 3 desc, 4 desc, 5 desc, 6 desc;
```

### **Function Level Description**

Main

Init()

- Check SYSTEM\_VARIABLES.update\_loc\_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\_locs()  
Get total stores and warehouses in the current thread.
- Size\_loc\_array()  
Size location array based on the number of locations in the current thread. The location array includes region, district, loc, loc\_type, select\_ind, update\_ind and touched columns.
- Size\_loc\_insert\_array()  
Size location insert array based on the maximum commit count set in restart\_control table. This array will be used for holding the exact data that is going to be insert into the sec\_user\_loc\_matrix table. The location insert array includes user\_id, column\_code, loc, loc\_type, select\_ind and update\_ind columns.
- Load\_loc\_array()  
Load all the stores/warehouses including organizational info in the current thread to the location 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 regions first, and then those for districts, and on down. The records are processed in that order. That is to say, first work with the region level rules, then move to the more specific rules so that the rules with the smaller scope take priority over the higher level rules. Warehouses are not included in the hierarchy and Each warehouse translates into a rule to be written out.

- 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. The rule array includes

user\_id, column\_code, region, region\_ind, district, district\_ind, store, store\_ind, wh, wh\_ind, select\_ind and update\_ind.

#### 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 region and district. Therefore, all rules for a particular organizational 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/warehouse 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 store/warehouse level according to the last record rule.
  - If it is new:
    - Call update\_array() to blow security rule down to the store/warehouse level according to the last record rule.
    - Call write\_array() to insert the security rules of last record's user/functional area (down to store/warehouse level) to table sec\_user\_loc\_matrix.
    - Call retek\_force\_commit() to commit the output to database and 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 location array according to the input security rule.

- If the input rule is a region level security rule, then loop through the location array, for all the stores within the region, set the select\_inds and update\_inds equal to the input rule's select\_ind and update\_ind, respectively. Set touched column of each processed row to 'Y'.
- If the input rule is a district level security rule, then loop through the location array, for all the stores within the district, set the select\_inds and update\_inds equal to the input rule's select\_ind and update\_ind, respectively. Set touched column of each processed row to 'Y'.
- If the input rule is a store level security rule, then loop through the location array, set the select\_ind and update\_ind of the store equal to the input rule's select\_ind and update\_ind, respectively. Set touched column of each processed row to 'Y'.

- If the input rule is a warehouse level security rule, then loop through the location array, set the select\_ind and update\_ind of the warehouse equal to the input rule's select\_ind and update\_ind, respectively. Set touched column of each processed row to 'Y'.

#### Write\_array()

This function writes out rows with touched = 'Y' in the location array to sec\_user\_loc\_matrix table.

- Create a loop through the location array
  - Check the location insert array, if it is full, call insert\_array\_to\_table().
  - If the touched column of the row is 'Y', copy the row to location insert array. Set the touched column of the row to 'N'.
- If the location insert array is not empty, call insert\_array\_to\_table().

#### Insert\_array\_to\_table()

This function uses array insert to insert the contents in the location insert array to the correct table sec\_user\_loc\_matrix\_a or sec\_user\_loc\_matrix\_b.

#### final():

restart/recovery close

### **I/O Specification**

N/A

### **Technical Issues**

N/A