Table of Contents:

GameSwap DataTypes

DataTypes

user

Attribute	Data Type	Nullable
email	String	Not Null
password	String	Not Null
first_name	String	Not Null
last_name	String	Not Null
nickname	String	Not Null

address

Attribute	Data Type	Nullable
city	String	Not Null
state	String	Not Null
postal_code	String	Not Null
latitude	Float	Not Null
longitude	Float	Not Null

phone

Attribute	Data Type	Nullable
phone_number	String	Null
phone_type	String	Null
is_shared	Boolean	Null

item

Attribute	Data Type	Nullable
title	String	Not Null
swap_status	String	Not Null
condition	String	Not Null
description	String	Null
item_number	Integer	Not Null

video_game

Attribute	Data Type	Nullable
platform	String	Not Null
media	String	Not Null

computer_game

Attribute	Data Type	Nullable
os	String	Not Null

jigsaw

Attribute	Data Type	Nullable
piece_count	Integer	Not Null

swap

Attribute	Data Type	Nullable
propose_date	Date	Not Null
swap_status	String	Not Null
swap_id	String	Not Null

accepted_swap

Attribute	Data Type	Nullable
accepted_date	Date	Not Null
proposer_rating	Float	Null
counterparty_rating	Float	Null

rejected_swap

Attribute	Data Type	Nullable
rejected_date	Date	Not Null

GameSwap Constraints

Business Logic Constraints

User

- Users who are new to GameSwap must register first.
- Users who have an existing GameSwap account will not be able to register.
- Users who have more than two unrated swaps or more than five unaccepted swaps cannot list a new item.
- Registered users should not be able to update their profile if they have any unapproved swaps or unrated swaps and show a message if they attempt to do so.
- An email can only be registered once in the system. Nicknames do not have this requirement.
- Users cannot update their email address.
- Users can only have a single unique phone number that is not used by anyone in the system.
- Users cannot swap with themselves.
- Users with no listed items may browse but cannot swap.

Swap

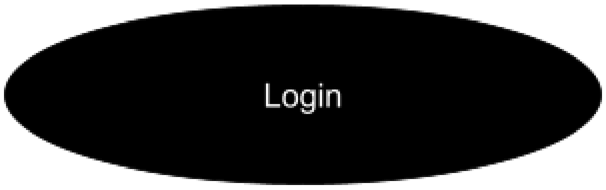
- Specific item for item swap cannot be proposed if swap is rejected.
- Swaps are completed if both proposer and counterparty rate each other.
- Contact information should be shown after a swap is accepted.
- To mark the swap as completed, after swapping items, both users must rate each other, on a scale of 0-5

Item

- Items which are not available for swapping cannot be included in search results.
- If an item does not have a description, the description field should not exist.
- Items associated with a pending swap (a proposed swap not yet accepted or rejected) are not available for swapping.
- Items which were previously paired in a swap between two users cannot be paired in a future swap together.
- Any item which has been part of a successful swap cannot be swapped again in the future.
- A user can enter an item (which was already swapped) into the system as a new item listing (which may have different/new information, such as an updated condition or description) for another swap.

Task Decomposition with Abstract Code

Login Form



Task Decomposition:

- **Lock Types:** Read-only on user table
- **Number of Locks:** Single
- **Enabling Conditions:** None
- **Frequency:** Medium
- **Consistency (ACID):** Not critical
- **Subtasks:** Mother task is not needed. No decomposition needed.

Abstract Code:

- If user has an account, then:
    - User enters *email* or *phone_number*, *password* input fields.
    - Upon:
        - Click ***Enter*** button
            - If user record is found in user table but user.*password* != *password*:
                - Go back to **Login Form** with an error message displaying "Password is incorrect".
            - Else if user record is not found in user table
                - Go back to **Login Form** with an error message displaying "Account not found".
            - Else:
                - Store login information user.*email* as session variable `$email`
                - Go to **Main Menu Form**
- Else if user does not have an account in user table, then:
    - User clicks on ***Register*** button
    - Go to **User Registration Form**

User Registration Form

Task Decomposition:

- **Lock Types:** Read/write on user or phone table
- **Number of Locks:** Two, one to read if user has already registered in user or phone table, and one to write user if new account into user table
- **Enabling Conditions:** Triggered by ***Register*** button
- **Frequency:** Low
- **Consistency (ACID):** Critical. User cannot use email/phone that already exists.
- **Subtasks:** Mother task is not needed. No decomposition needed.

Abstract Code:

- User enters *email*, *nickname*, *password*, *city*, *first_name*, *last_name*, *state* , *postal_code* in required input fields.
- If user inputs *phone_number*, then:
  - If user selects checkbox
    - This user's phone.*disclosure_choice* == true upon write
  - Else
    - This user's phone.*disclosure_choice* == false upon write
  - User selects *phone_type* of *phone_number* in dropdown
- Upon:
  - Click ***Register*** button
    - If *postal_code* is not on the list of valid postal codes, then show the error message "Postal code invalid."
    - Else if any *email* == user.*email* or *phone_number* == phone.*phone_number* , then show error message "User email or phone number is already registered"
    - Else write user's input into user and phone table
      - Jump to **Login Form**

Main Menu Form

Task Decomposition:

- **Lock Types:** Read-only on user, item, swap and accepted_swap table
- **Number of Locks:** Three. One to read user information, another to obtain user items, and another to read statistics.
- **Enabling Conditions:** Upon correct information and successful login
- **Frequency:** High
- **Consistency (ACID):** Not critical
- **Subtasks:** Mother task is not needed. No decomposition required.

Abstract Code:

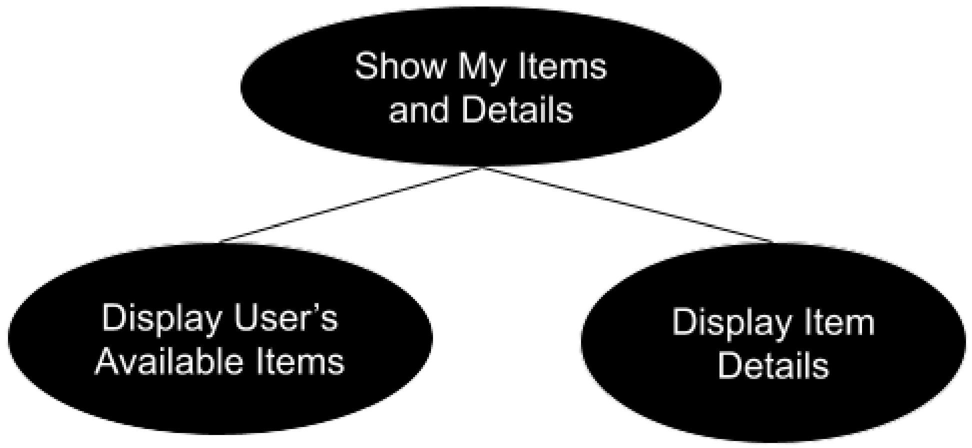
- Query user's *first_name* and *last_name* from user table and display a welcome message
- Display the following statistics:
  - "My Rating" using average of all ratings associated with the current user from accepted_swaps table:
    - Display "None" if no ratings have been made for the user's items.
  - "Unaccepted Swaps" using swap table:
    - If the number of "Unaccepted Swaps" greater than zero, create a clickable link can jump to **Accept/Reject Swaps Form.**
    - If any swaps are more than five days old, or the user has more than five "Unaccepted Swaps", print the number in bold and in red.
  - "Unrated Swaps" using accepted_swaps table*:*
    - If the number of "Unrated Swaps" greater than zero, create a clickable link can jump to **Rate Swaps Form.**
    - If the number of "Unrated Swaps" greater than 2, print the number in bold and red.
- Show "***List Item***", "***My items***", "***Search items***", "***Swap history***", "***Update my info***", and "***Logout***" tabs.
- Upon:
  - Click ***List Item*** button- Jump to **Listing an Item Form.**
  - Click ***My items*** button- Jump to **Display User's Available Items** task.
  - Click ***Search items*** button- Jump to **Search Items Form**.
  - Click ***Swap history*** button- Jump to **Swap History Form.**
  - Click ***Update my info*** button- Jump to **Update User Information Form**.
  - Click ***Logout*** button- Invalidate login session and jump again to the **Login Form**.

Listing an Item Form

Task Decomposition:
- **Lock Types:** Write on item table, read on swap table
- **Number of Locks:** Two. One to write new item into item table, another to pull user's unrated or unaccepted swaps from the swap table.
- **Enabling Conditions:** Click ***List Item*** button from **Main Menu Form.**
- **Frequency:** Low
- **Consistency (ACID):** Order is critical for item indexing
- **Subtasks:** Mother task is not needed. No decomposition needed.

Abstract Code:
- If user has more than two unrated swaps or more than five unaccepted swaps, then:
  - Show a message that they cannot list a new item.
- Else:
  - User selects the item type from the dropdown.
  - If item type is a "Computer Game", then add additional text field for *os*
  - Else if the item type is a "Video Game", then add additional text fields for *platform* and *media*.
  - Else if the item type is a "Jigsaw", then add additional text field for *piece_count*
  - User fills out appropriate additional text fields
  - User enters *title* and *description* (optional) in text fields and selects *condition* from drop down.
  - Upon:
    - Click ***List Item*** button:
      - If there is an error, then:
        - List appropriate error message
      - Else:
        - Save item into item table
        - Assign index to item by the system
        - Show success message with item.*item_number*

My Items Form



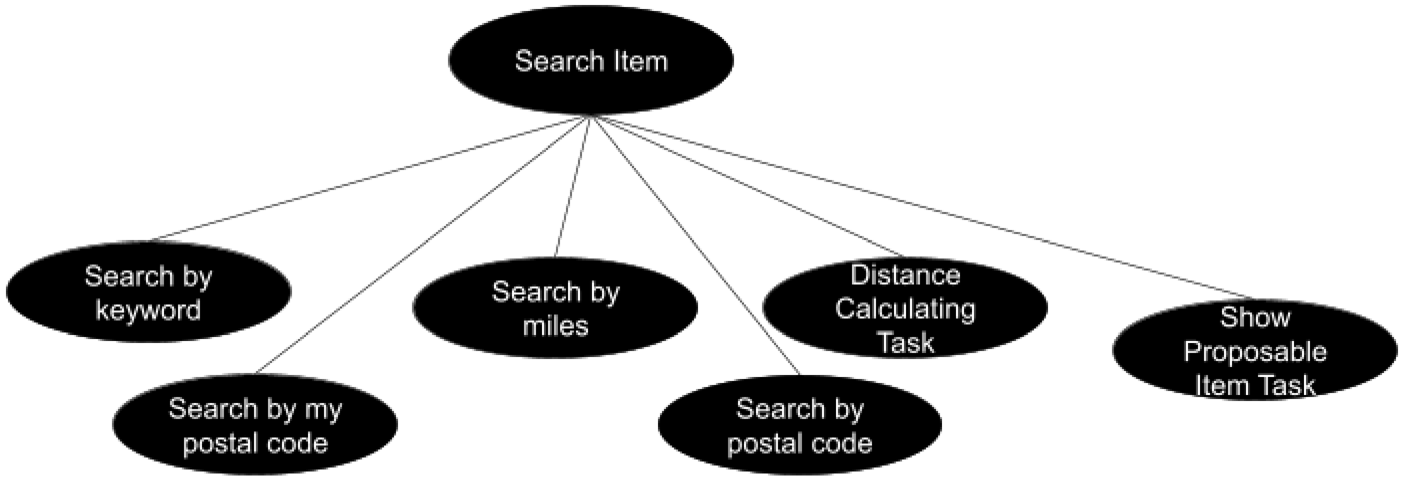
Task Decomposition:

- **Lock Types:** Read-only on user and item
- **Number of Locks:** Two. Read-only locks on user and item
- **Enabling Conditions:** Click ***My items*** button from **Main Menu Form**
- **Frequency:** Medium
- **Consistency (ACID):** Not critical
- **Subtasks:** Mother task is needed, with following sub-tasks:
  - **Display User's Available Items**
  - **Display Item Details**

Abstract Code:

- Run **Display User's Available Items** sub-task:
  - Query about the users and their items using *email* as an identifier from item table:
    - For all the item types (games), query and display the number of items owned by the user.
    - Display the total number of items in the user's possession.
- Sort the items by item number in an ascending order.
- For each item, query and display its title, condition, and description from item table
  - Show the first 100 characters of the description and if the number of characters for the description is more than 100, place an ellipse […].
  - Include details link on each line related to the item number.
- After clicking on details link, run **Display Item Details** sub-task**:**
  - Set $selected_item to be the selected item's item.*item_number* and jump to **View Item Form**

Searching Items Form



Task Decomposition:

- **Lock Types:** Read-only on user,item,address
- **Number of Locks:** 3 read only locks on user,item,address
- **Enabling Conditions:** triggered by when user clicks on ***search item*** button from Main Menu Form
- **Frequency:** Medium
- **Consistency (ACID):** Not critical
- **Subtasks:** Mother task is needed, with following sub-tasks:
    - **Search By Keyword**
    - **Search My Postal**
    - **Search By Postal Codes**
    - **Show Proposable Item**
    - **Search Within Miles**
    - **Distance Calculating**

Abstract Code:

- Generate four radio buttons for the user to select.
- Generate text input fields for search by *keywords* option and search by *postal code* option.
- Generate integer input field for search by *miles* option.
- Generate ***search*** button at the lower right corner of the form.

When user choose one of four search options:

- If the user chooses search by keyword option, and input *keywords,* then click on ***search*** button, jump to **Search By Keyword** sub-task:
    - Query user input keywords against item.*title* and then item.*description* from item table, when there is a match in either attribute, run **Show Proposable Item** sub-task

- If the user chooses in my postal code option, and then click on ***search***, jump to **Search My Postal** sub-task:
    - Using current user's address.*postal_code*, find all other users who have the same postal code, match all these user's items and run **Show Proposable Item** sub-task**.**

- If the user chooses with X miles of me search options, user input *miles*, then click on ***search***, jump to **Search Within Miles** sub-task.
    - With user input *miles,* query address table and run **Distance Calculating** sub-task**,** flag postal code where results from distance calculation is less or equal to *miles.*
    - Then query user table where users live in these flagged postal code
    - Match all these user's items and run **Show Proposable Item** sub-task**.**
- If the user chooses the search by postal code option, input *postal code*, then click on ***search***, jump to **Search By Postal Codes** sub-task.
    - User input *postal code,* find all other users who live in *postal code*, match all these user's items and run **Show Proposable Item** sub-task
- **Show Proposable Item**
    - Query item table with given *item_number*
    - if *item_swap_status* is available for swap,
        - Display item's *item_number, item_type, title, condition, description*(show only first 100 characters)
    - Run **Distance Calculating** sub-task**.**
    - Sort by distance and item number in an ascending order.
    - If coming from **Search By Keyword** sub-task**,** highlight matching attribute fields with blue color.
    - Set session variable $selected_item to current item.*item_number*
    - Generate ***detail*** button link to this item's **View Item Form** using $selected_item
- **Distance Calculating**
    - Read current user's address.*longitude* and address.*latitude.*
    - Read target user's address.*longitude* and address.*latitude.*
    - Calculate the distance between using above info.
    - Return and display the distance

View Items Form

Task Decomposition:

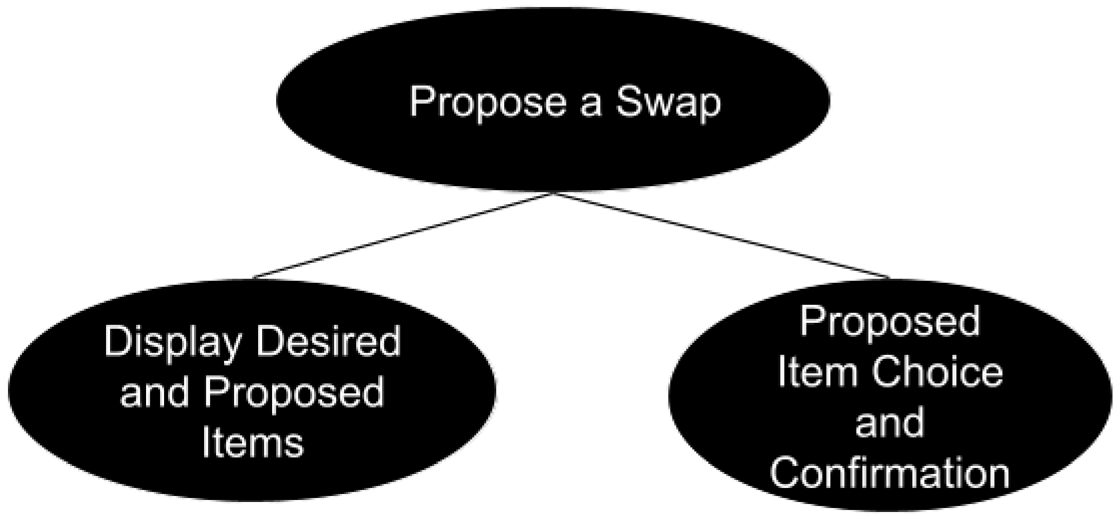
- **Lock Types:** Read-only on user, swap and item table
- **Number of Locks:** Three.
- **Enabling Conditions:** when user clicks on detail link from **My Items Form** or **Search For Item Form**
- **Frequency:** Medium
- **Consistency (ACID):** Not critical
- **Subtasks:** Mother task is not needed. No decomposition.

Abstract Code:

- When the user enters the **View Items Form** from another form, it will pass the $selected_item session variable.
    - Look up the selected item in the item table such that item.*item_number* is equal to $selected_item. Using this record, display:
        - item.*title*, item.*description*, item.*condition*, item.item_*swap_status*, and item.*item_type*
        - If item.*item_type* is ‘computer_game’: display computer_game.*os*
        - If item.*item_type* is ‘video_game’: display video_game.*platform* and video_game.*media*
        - If item.*item_type* is ‘jigsaw’: display jigsaw.*piece_count*
        - Use the item table to look up the item’s owner; if user.*user_email* is equal to $email, it belongs to the current user. If it does not belong to the current user:
            - Calculate and display distance to the current user by looking up:
                - The item’s related user and that user’s user.*latitude* and user.*longitude*
                - Look up the current user’s user.*latitude* and user.*longitude* using the $email session variable.
                - Highlight the calculated distance based on the following rules:
                    - 0-25 miles: green
                    - 25-50 miles: yellow

            - 50-100 miles: orange
            - 100+ miles: red
        - Display the item owner's user.*nickname*
        - Looking at the current user's swaps, if they have less than or equal to 2 unrated swaps or less than or equal to5 unaccepted swaps and the current item is available for swapping:
            - If the user clicks on ***Propose Swap*** button, run the **Propose a Swap Form**
    - If the user clicks ***Exit*** button, clear the $selected_item variable and return to the previous form.

Propose a Swap Form


Propose a Swap
Display Desired and Proposed Items
Proposed Item Choice and Confirmation

Task Decomposition:

- **Lock Types:** Read-only lookups on user and item tables; Write lock on swap and item table
- **Number of Locks:** Multiple locks to read user info, item info, and two write locks one for inserting new swap, one for writing to item
- **Enabling Conditions:** Lookup enabled by a sufficient swap rating; User clicks on ***Propose Swap*** button. Swap proposal capability enabled by ability to view available items to propose; Insert of swap proposal date information is triggered by completing proposal confirmation
- **Frequency:** Medium
- **Consistency (ACID):** Consistency is critical for several reasons:

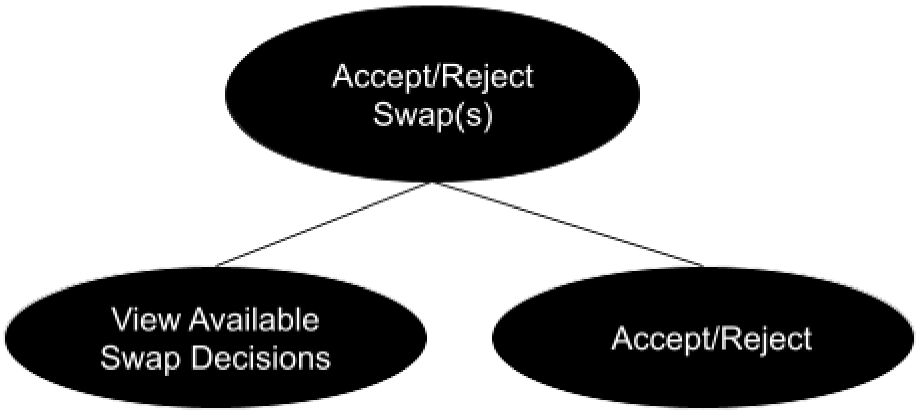
- Swap ratings must be current in order to correctly display or hide **Propose a Swap Form**.
- Up-to-date address information on the user table is critical in order to display a distance warning message if needed.
- Up-to-date desired item information is needed to confirm item availability and accurate information.

- **Subtasks:** Mother task is needed, with following sub-tasks:
  - **Display Desired and Proposed Items**
  - **Proposed Item Choice and Confirmation**

Abstract Code:

- Run **Display Desired and Proposed Items** sub-task:
  - Read desired item from swap table and display on form.
  - Query to get Proposer's and Counterparty's address.*latitude* and address.*longitude* values.
    - If distance between Proposer and Counterparty is >= 100.00 miles, display a warning message containing the distance at the top of the form in red.
  - Query item table to get all of the User's associated available items and display the following information within the form: item.*item_number*, item.*item_type*, item.*title*, item.*condition*.
  - Order list of items by ascending item.*item_number*
- Run **Proposed Item Choice and Confirmation** sub-task:
  - Show ***Select*** radio button for each item displayed.
    - If ***Select*** button is clicked, populate the radio button and show ***Confirm*** button.
      - If ***Confirm*** button is clicked, generate a date stamp and insert into the swap table.
      - Show a message letting the user know swap has started and generate an ***OK*** button.
      - If the user clicks on the button, take the user back to **Main Menu Form**.
    - Else, if no items are chosen OR all previously selected items are unselected by the user, do not show the ***Confirm*** button.

Accept/Reject Swaps Form



Task Decomposition:

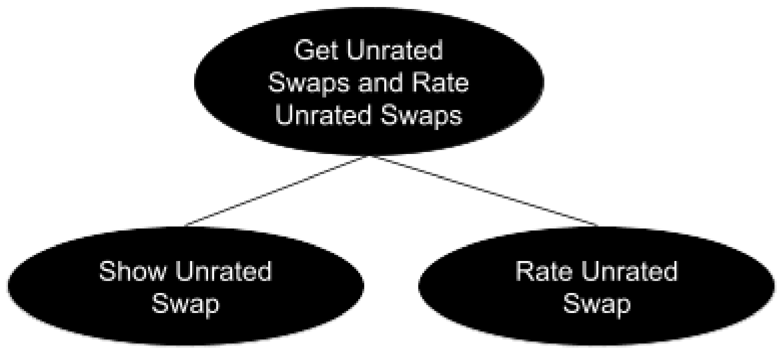
- **Lock Types:** Read lookups on user and swap tables; Write lock on swap table
- **Number of Locks:** Two read locks and one write lock
- **Enabling Conditions:**
    - When the user clicks on unaccepted swap from **Main Menu Form**.
    - Enabled by previously proposed swaps, with updates to the user table
    - Enabled by selection of ***Accept*** or ***Reject*** buttons for each available swap confirmation.
- **Frequency:** Low
- **Consistency (ACID):** Not critical. Once a swap is proposed, it cannot be removed, so no concern over a proposed swap being removed at the same time as acceptance or rejection.
- **Subtasks:** Mother task is needed, with following sub-tasks:
    - **View Available Swap Decisions**
    - **Accept/Reject**

Abstract Code:

- Run **View Available Swap Decisions** sub-task:
    - Query swap table to get all pending swaps associated with items owned by User, query user table to get user.*nickname*, and query item table to get information related to specific item.
    - Display the following information from these two tables:
        - swap.*propose_date*;
        - ***Desired Item*** as link;
            - If ***Desired Item*** link is clicked – Jump to **View Items Form**
        - user.*nickname*
        - Calculate average of all ratings associated with proposer from accepted_swaps table

        - Distance
            - Read proposer's address.*longitude* and address.*latitude.*
            - Read counterparty's address.*longitude* and address.*latitude.*
            - Calculate the distance between using above info.
            - Return and display the distance
        - ***Proposed Item*** link
            - If ***Proposed Item*** link is clicked – Jump to **View Items Form**
        - Display ***Accept*** and ***Reject*** button for each available swap.
- Run **Accept/Reject** sub-task:
    - If an ***Accept*** button is clicked:
        - Get User's user.*email* and user.*first_name* from user table and display within form or separate dialog box
        - Check User's phone.*phone_number* value from phone table:
            - If null, display message that phone is unavailable.
            - Else, get Sharing Option:
                - If Sharing Option = "Allowed", display phone.*phone_number* and phone.*phone_type* type on form.
            - Generate a date stamp upon clicking ***Accept*** button and insert accepted_swap.*accepted_date* into the accepted_swap table.
            - Update swap.*swap_status* to "accepted."
        - Remove accepted_swap from display list of proposed swaps:
            - If the number of items in swaps list == 0, display **Main Menu Form**.
    - If ***Reject*** button is clicked:
        - Generate a date stamp to reflect swap rejected_swap.*rejected_date* and insert into the rejected_swap table.
        - Update swap.*swap_status* to "rejected."
        - Remove rejected_swap from display list of proposed swaps.
        - If the number of items swaps list == 0, display **Main Menu Form**.

Rate Swaps Form



Task Decomposition:

- **Lock Types**: read lock for item, swap and user, write lock for swap
- **Number of Locks:** 3 read lock, 1 write lock
- **Enabling Conditions:** click on ***rate swap*** button from **Main Menu Form**
- **Frequency:** low to medium
- **Consistency(ACID):** Not critical
- **Subtasks:** Mother task is needed, with following sub-tasks:
  - **Show Unrated Swap**
  - **Rate Unrated Swap**

Abstract Code:

- Run **Show Unrated Swap** sub-task.
- If user select rating for the other user in a swap, run **Rate Unrated Swap** sub-task
- When there is no more unrated swaps, go back to **Main Menu Form**

Show Unrated Swap

- For current user session, query item table where item's owner email is current user email where item.*item_swap_status* is "accepted" AND corresponding rating attributes is null
  - Then query swap where these *item_number* either show up in swap.*being_proposed* OR *swap.being_desired*
  - Then determine user's role and find counter party
- If the item_number shows up in a swap as swap.*being_offered*
  - the role for this user is proposer
  - find other user's nick name by looking up the item_number in swap.*being_desired*, find the user.*nickname* of that item's owner.
- Else if item_number shows up in a swap as swap.*being_desired*
  - the role of the user is counterparty

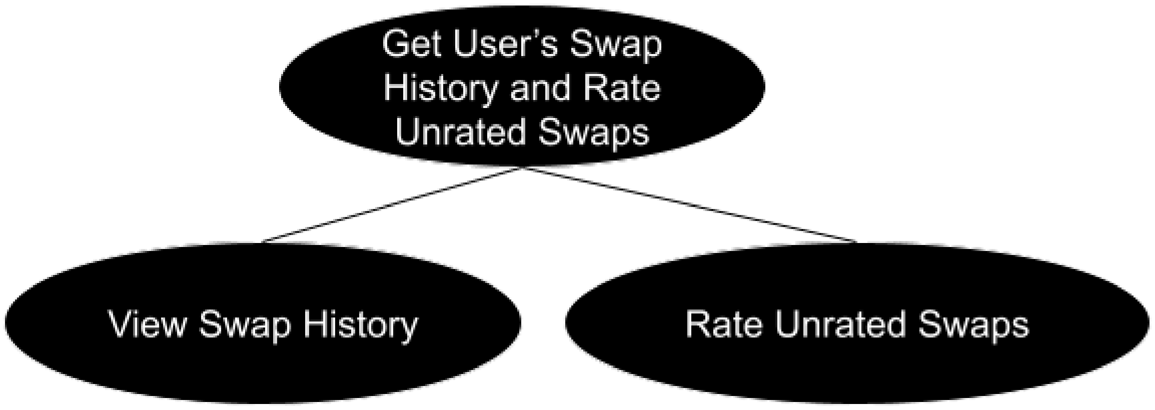
- find other user's nick name by looking up the item_number in swap.*being_offered*, find the user.*nickname* of that item's owner.

- Display swap.*acceptance_date*, current user's role, offering item's title, desired item's title, the other user's nickname, ordered by acceptance date descending
- Generate a drop down menu so the current user can give rating to the other user in the swap.

Rate Unrated Swap

- If current user's role in a swap is proposer,
  - write user chosen value into swap.*counterparty_rating*
- Else if current user's role in a swap is counterparty,
  - write user chosen value into swap.*proposer_rating*

Swap History Form



Task Decomposition:

- **Lock Types:** Read-only on user, item, swap, accepted_swap, and rejected_swap tables; Write lock (insert) on accepted_swap table.
- **Number of Locks:** Several due to multiple schema constructs
- **Enabling Conditions:** Enabled by successfully completed swaps (accepted/rejected).
- **Frequency:** Low; accessed only when a user is interested in seeing their swap history and/or providing ratings for previously completed swaps which were not initially rated.
- **Consistency (ACID):** Not critical. If a swap is completed while viewing the page, the swap history will be updated upon refresh.
- **Subtasks:** Mother task is needed, with following sub-tasks:
  - **View Swap History**
  - **Rate Unrated Swaps**

Abstract Code:

- Run **View Swap History** sub-task:
    - Run query on user and swap tables to derive the following and display on form for both Proposer role and Counterparty role, listed separately:
        - Total swaps proposed
        - Total received
        - Sub-totals for accepted and rejected swaps
        - % rejected
    - If % rejected >= 50.0%, highlight percentage in red.
    - Query swap, item, user, accepted_swaps, and rejected_swaps table to get the User's completed swaps; For each completed swap, display the following information in a table format:
        - swap.*propose_date*
        - accepted_swaps.*accepted_date* or rejected_swaps.*rejected_date*
        - swap.*swap_status*
        - User's role
        - Proposed item title
        - Desired item title
        - Other User's nickname
        - Swap rating:
    - If Swap rating is NULL, display rating mechanism.
        - Run **Rate Unrated Swaps** sub-task:
            - Once rating has been chosen, insert rating into accepted_swap table and refresh page.
                - Once refreshed, run a query for proposer_rating again and display on form.
                - Rate unrated swap by allowing the User to choose the rating they would like to give to the other user:
                    - If myrole == proposer, write to accepted_swap.*counterparty_rating*
                    - Else if myrole == counterparty, write to accepted_swap.*proposer_rating*
    - For each swap listed in history, display a ***Detail*** link:
        - When ***Detail*** link is clicked, display **Swap Details Form** for the swap associated with the link in the table.

Swap Details Form

Task Decomposition:

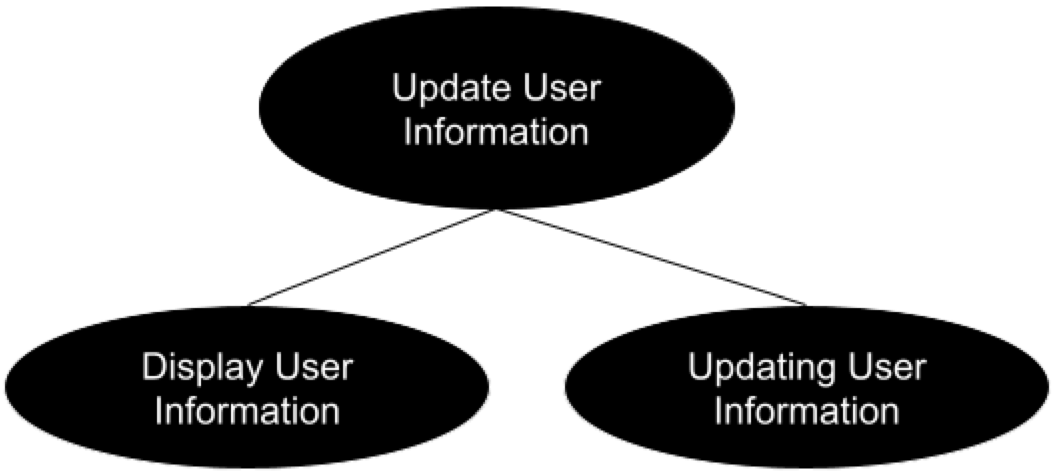
- **Lock Types:** Read on user, item, swap table, write on accepted_swap table
- **Number of Locks:** Four
- **Enabling Conditions:** Accessed only from **Swap History Form**
- **Frequency:** Low.
- **Consistency (ACID):** Not critical
- **Subtasks:** Mother task is not needed. No decomposition needed.

Abstract Code:

- When a user enters the **Swap Details Form** for a given swap:
    - Find the current user in the user table such that user.*user_email* is equal to $email:
    - If user is owner of item being_offered or being_desired. For each swap in which the owner has an item being_offered or being_desired:
        - Using the swap.*swap_id*:
            - Display user.*user_email* of user who owns item being_offered
        - Display item.*title* and *item.description* of item being_offered
        - If user is not the owner of item being_offered:
            - Find the being_offered item's related user and that user's user.*latitude* and user.*longitude*
            - Look up the current user's user.*latitude* and user.*longitude* using the $email session variable.
            - Display the distance calculated using the above variables.
                - Display user.*user_email* of user who owns item that is being_desired
        - Display item.*title* and item.*description* of item being_desired
        - If user is not the owner of item being_desired:
            - The being_desired item's related user and that user's user.*latitude* and user.*longitude*
            - Look up the current user's user.*latitude* and user.*longitude* using the $email session variable.

  - Display the distance calculated using the above variables.
- Display swap.*swap_status* for the current swap
- If swap.*swap_status* is ‘Accepted’:
  - display accepted_swap.*accepted_date*
- else if swap.*swap_status* ‘Rejected’:
  - display rejected_swap.*rejected_date*
- else display swap.*propose_date*
- If swap.*swap_status* == ‘Accepted’
  - If user is the owner of item being_offered:
  - Look up accepted_swap.*proposer_rating*.
  - If value is not null, display.
  - Else if value is null:
    - Take user’s input for *swap_rating*. In swap table, update accepted_swap.*proposer_rating* per the user input.
    - Return to **Swap Details Form** (refresh).
  - Else if user is the owner of item.*being_desired*:
    - If swap.*swap_status* == ‘Accepted’
      - If user is the owner of item being_offered:
      - Look up accepted_swap.*counterparty_rating*. If value is not null, display. If value is null:
      - Take user’s input for *swap_rating*. In swap table, update accepted_swap.*counterparty_rating* per the user input.
    - Return to **Swap Details Form** (refresh).
- If the user has no items involved in swaps, display “No swaps found.” Return to **Main Menu Form**

Update User Information Form



Task Decomposition

- **Lock Types:** Read from user and phone table. Write on user and phone table.
- **Number of Locks:** Several due to two schemas accessed.
- **Enabling Conditions:** Consistent across both tasks: the user must exist.
- **Frequency:** Low
- **Consistency (ACID):** Not critical
- **Subtasks:** Mother task is needed, with following sub-tasks:
  - **Display User Information**
  - **Updating User Information**

Abstract Code:

- When user clicks the ***Update my info*** button from the **<u>Main Menu Form</u>**, run the **Display User Information** sub-task:
    - ■ Query the user table such that $email is equal to user.*user_email* and display the user's user.*first_name*, user.*last_name*, user.*nickname*, and user.*user_email*
  - If user has_a phone find the phone using the user.*user_email* (equal to $email) in the phone table. Display the phone.*phone_number*, phone.*phone_type*, and phone.*is_shared*
  - If ***Delete User Profile*** button is pushed*:* Find and delete row for this user based on user.*user_email* being equal to $email in the user table. Clear $email session variable. Return to **<u>Login Form</u>**
  - Display the ***Change User Information*** button. If clicked, run the **Updating User Information** sub-task:
    - ■ While no buttons are pressed, do nothing.
    - ■ If ***Update Email button*** is pressed:
    - ■ take the user's input

        - replace user.*user_email* with the input in the user table. Update the $email session variable to reflect this change as well.
            - Run **Display User Information** sub-task**.**
        - If ***Update Name Preferences*** button is pressed*:*
            - take the user's input for the three name fields.
            - Find the user in the user table based on user.*user_email* and update user.*first_name*, user.*last_name*, and user.*nickname* in user table with the user's input.
            - Run **Display User Information** sub-task**.**
        - If ***Update Phone Preferences*** button is pressed***:***
            - Take the user's input for phone.*phone_number*, phone.*phone_type*, and phone.*is_shared*.
            - Find the user's phone in the phone table using user.*user_email* and update phone.*phone_number*, phone.*phone_type*, phone.*is_shared* in the phone table with the user's input.
            - Run **Display User Information** sub-task**.**
    - If ***Exit*** button is clicked, return to **Main Menu Form**.