# Table of Contents:

# Data Types:

**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 |
| phone_number | String | Null |
| phone_number_type | String | Null |
| share_phone_number | Boolean | Null |

**Postal Code**

| Attribute | Data Type | Nullable |
|---|---|---|
| city | String | Not Null |
| state | String | Not Null |
| latitude | String | Not Null |
| longitude | String | Not Null |

**Item**

| Attribute | Data Type | Nullable |
|---|---|---|
| item_name | String | Not Null |
| game_type | String | Not Null |
| game_platform | String | Null |
| game_media | String | Null |
| piece_count | Integer | Null |
| item_condition | String | Not Null |
| Item_description | String | Null |
| computer_platform | String | Null |

**Swap**

| Attribute | Data Type | Nullable |
|---|---|---|
| proposal_date | Date | Not Null |
| proposer | Integer | Not Null |
| proposer_item | Integer | Not Null |
| counterparty | Integer | Not Null |
| counterparty_item | Integer | Not Null |
| swap_status | String | Not Null |
| accepted_rejected_date | Date | Null |
| proposer_rating | Integer | Null |
| counterparty_rating | Integer | Null |

# Business Logic Constraints:

### GameSwap Users:

- Users who are new to the GameSwap System need to register before they can use the system.
- Users cannot update their information if they have any unapproved or unrated swaps.

### GameSwap Items:

- A user cannot list a new item if they have more than two unrated swaps, or more than five unaccepted swaps.

### Swap:

- A user cannot swap items with themselves.
- Users must have listed at least one item to be able to swap items with another user.
- Item associated with a pending swap is not available for swapping.
- A user cannot propose a swap if they have more than 2 unrated swaps.
- If a swap is rejected, a new swap between the proposed item and desired item cannot be made.
- Items from Accepted Swaps cannot be listed for swap again.

# Register

## Task Decomp

**Lock Types:** Insertion on User table
**Number of Locks:** Single
**Enabling Conditions:** None
**Frequency:** Very common, all users must register
**Consistency (ACID):** Not critical, order is not critical
**Subtasks:** Mother task not needed. No decomposition needed.


## Abstract Code

- Users can perform **Registration** by clicking ***Register*** on **Login Form**.
- Upon:
  - enter *email* ('$Email'), *password* ('$Password'), *first name*, *last name*, *nickname*, *city*, *state*, *postal code*, and *phone number* (optional) in input fields.
  - Store *email* ('$Email'), *password* ('$Password'), *first name*, *last name*, *nickname*, *city*, *state*, *postal code*, and *phone number* (optional) in **User** table of "GameSwap Database System".

For Postal Code input:
- Jump to **Verify Postal Code** Task

For Phone Number input:
- Enter the $phoneNumber
- Select Phone Number Type
- Check the box if share Phone Number on Swaps

# Validate Postal Code

## Task Decomp

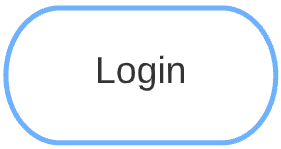
**Lock Types:** Lookup on Postal Code table
**Number of Locks:** Single
**Enabling Conditions:** None
**Frequency:** Very common, part of registration process
**Consistency (ACID):** Not critical, order is not critical
**Subtasks:** Mother task not needed. No decomposition needed.

## Abstract Code

- Triggered from the **Registration** form once the postal code is entered into form.
- Lookup postal code in **PostalCode** table.
- If postal code is not found, notify user that postal code could not be found
- If $city and $state as input from User exist and do not match **PostalCode** City and State, then notify user that postal code validation failed
- If postal code is found, no action is required

# Login

## Task Decomp



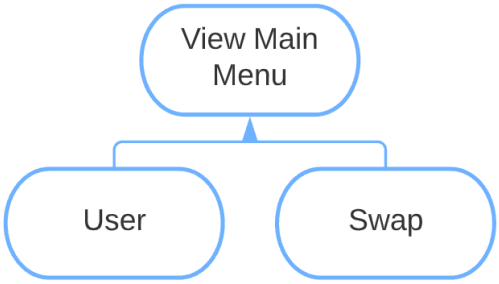

**Lock Types:** Lookup on User table
**Number of Locks:** Single
**Enabling Conditions:** None
**Frequency:** Very common. All users must log in to use the services website provides.
**Consistency (ACID):** Not critical. Order not important.
**Subtasks:** Mother task not needed. No decomposition needed.

## Abstract Code

- Show "***Login***", "***Register"*** buttons.
- User enters *email* ($Email) or *phone number* ($Number) and *password* ($Password) input fields
- If data validation is successful for both username/phone number and password input fields, then:
    - When ***Login*** button is clicked or ***Enter*** is pressed
        - If User record is found but user.password != $Password
            - Go back to **Login** form, with "wrong password" error message
        - Else:
            - Store login information as session variable $UserID
            - Go to **View Main Menu** form
- If *email* or *phone numbe*r for the user is invalid
    - Display **Login** form, with "email/phone number not registered" error message
- If "***Register***" button is clicked, Jump to **Register** task

# View Main Menu

## Task Decomp




**Lock Types:** Lookup User Name in User table, Lookup on Swap table for User Average Rating, Unaccepted Swaps, and Unrated Swaps
**Number of Locks:** Couple different schema constructs are needed
**Enabling Conditions:** Trigger by successful login or registration
**Frequency:** Very common, similar frequency to login
**Consistency (ACID):** Not critical, order is not critical
**Subtasks:** Separate tasks are needed for reading from User and Swap tables

## Abstract Code

- Show **"*List Item*"**, "***My Items***", "***Search Items***", "***Swap History***", "***Update my info***", "***Logout***", "***Unaccepted Swaps***", "***Unrated* Swaps**" buttons.
- Display "Welcome" message including **User** FirstName and **User** LastName
- Display "My Rating" with calculated average of all user's swap ratings underneath
- Display $numPendingSwaps underneath ***"Unaccepted Swaps***" button
    - If the user has more than 5 unaccepted swaps, ($numPendingSwaps > 5)
        - Display $numPendingSwaps in bold and red
    - If any of the unaccepted swaps are older than 5 days, ($CurrentDate - **Swap** ProposedDate > 5)
        - Display $numPendingSwaps in bold and red
- Display $numberOfUnratedSwaps underneath ***"Unrated Swaps"*** button.
    - If the user has more than 2 unrated swaps, ($numberOfUnratedSwaps > 2)
        - Display $numberOfUnratedSwaps in bold and red
- Upon:
    - Click ***List Item*** button – Jump to the **List Item** task.
    - Click ***My Items*** button – Jump to the **View My Items** task.
    - Click ***Search Items*** button – Jump to **Search Items** task.
    - Click ***Swap History*** button – Jump to **View Swap History** task.

- Click ***Update my info*** button – Jump to **Update User Information** task
- Click ***Unrated Swaps*** button:
  - If user has unrated swaps ($numberOfUnratedSwaps > 0) - jump to **Rate Swaps** task
  - Else do nothing
- Click ***Unaccepted Swaps*** button:
  - If user has unaccepted swaps ($numPendingSwaps > 0) - jump to **Accept/Reject Swaps** task
  - Else do nothing
- Click ***Log Out*** button – Invalidate login session and go back to **Login** form

## List an Item

### Task Decomp


**Lock Types:** Lookup and Write on Item table
**Number of Locks:** Couple locks needed, one for lookups and one for insertions
**Enabling Conditions:** Enabled by a user's login and having less than three unrated swaps or less than 6 unaccepted swaps
**Frequency:** Very common
**Consistency (ACID):** Not critical, order is not critical
**Subtasks:** Separate tasks are needed for reading and writing into the Item table. No mother task is needed.

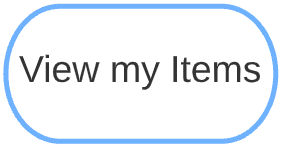
### Abstract Code

- Show *Game Type* field, *Title* field, *Condition* field, *Description* field, and ***List Item*** button
- If user selects "Jigsaw Puzzle" as *Game Type:*
  - Display *Piece Count* field
- If user selects "Video Game*"* as *Game Type:*

    - Display *Game Platform* field
    - Display *Media* field
- If user selects "Computer Game" as *Game Type**:***
    - Display *Computer Platform* field
- Upon Clicking List Item:
    - If any of the displayed fields have incomplete/invalid information entered:
        - User cannot list item, display error message
    - If the user has more than two unrated swaps ($UnratedSwapsCount> 2):
        - User cannot list item, display error message
    - If the user has more than five unaccepted swaps ($numPendingSwaps > 5):
        - User cannot list item, display error message
    - Else:
        - Item information is saved
            - Information provided in *Game Type* field is stored to **Item** gameType
            - Information provided in *Title**e*** is stored in **Item** gameTitle
            - Information provided in *Condition* is stored in **Item** itemCondition
            - Information provided in *Description**n*** is stored in **Item** itemDescription
        - Item number is generated and stored in **Item** ItemID
        - Display Success Message showing the item's **Item** ItemID

## View My Items

### Task Decomp



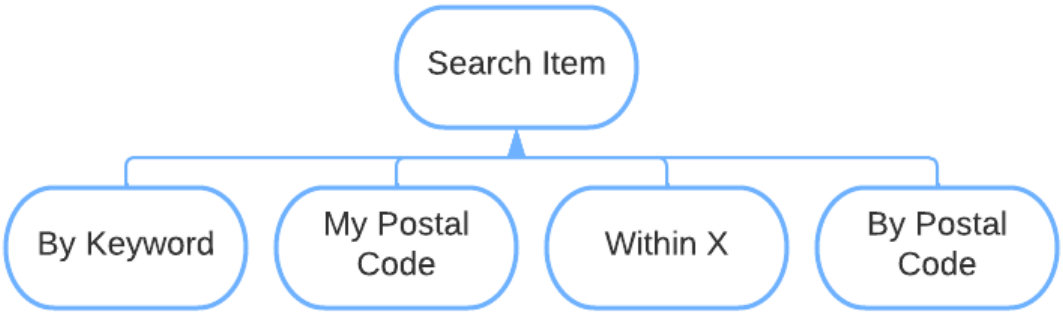

**Lock Types:** Lookup on Item table for all available items and their count by game type
**Number of Locks:** Single
**Enabling Conditions:** User is Logged in
**Frequency:** Very Common. Used by every logged in user
**Consistency (ACID):** Not critical, order is not critical
**Subtasks:** Mother task not needed. No decomposition needed.

### Abstract Code

- $UserID is the ID of the current user using the system from the HTTP Session/Cookie.
- Query the items table to get all distinct game_types and their count for the current logged in user using $userID.
- Display all the distinct item types as table heading with their respective count as table data and the total count at the end.
- Query the items table to get all available items for the current logged in user using $userID.
- For each resulting item:
    - Display **Item** ItemID, Title, GameType, Condition, and Description and a ***Detail*** link to view the detail of the item
- While no button is pressed, do nothing.
- When a button is pushed, then do the following:
    - Click ***Detail*** Link for an item:
        - Jump to **Item Details Form**

## Search Item

### Task Decomp



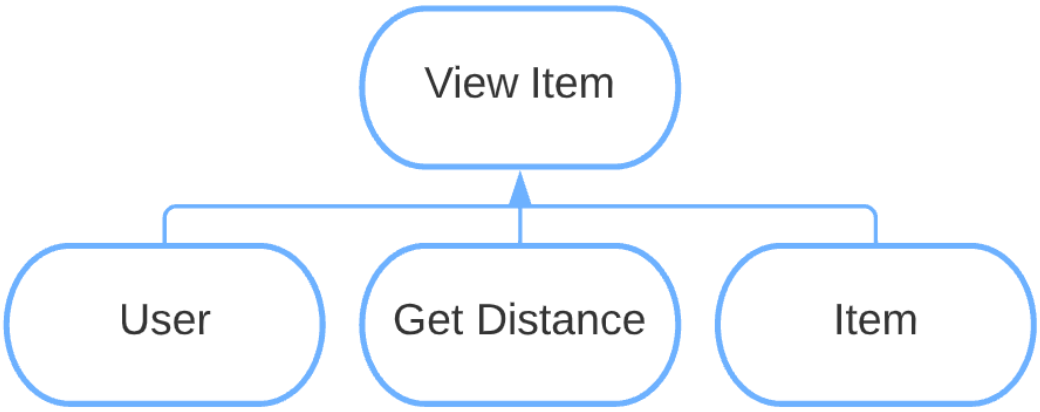

**Lock Types:** Lookups on Item, User and Postal Code tables
**Number of Locks:** Several different schema constructs are needed
**Enabling Conditions:** Enabled by a user's log in
**Frequency:** Fairly common
**Consistency (ACID):** Not critical, order is not critical
**Subtasks:** Separate tasks are needed for each input field in **Search Form**. No mother task is needed.

### Abstract Code

- Display 4 different options to search by: "*By keyword**”***, "*In my postal code**”, “**Within ‘X’ miles of me**”, “**In postal code:**”.* Display ***Search*** button.
- *Upon pressing **Search** button:*
    - If *“By keyword**”*** is selected:
        - Find items matching entered keyword
    - If "*In my postal code**”*** is selected:
        - Find items located within users postal code
    - If *“Within ‘X’ miles of me”* is selected:
        - Calculate distance from user using entered number
        - Find items within that range
    - If *“In postal code”* is selected:
        - If postal code entered is invalid
            - Display "Invalid Postal Code" error message
        - Else If postal code entered is valid
            - Find items
- If no results found:
    - Display error message: "Sorry, no results found!"
    - Return to **Search Item Form**
- Upon Successful search (results found):
    - Display results, including "Distance**”**, showing the calculated distance from user

## View Item

### Task Decomp




**Lock Types:** Lookups on Item, Postal Code, and User tables
**Number of Locks:** Several different schema constructs are needed
**Enabling Conditions:** User is logged in
**Frequency:** Very common

**Consistency (ACID):** Not critical, order is not critical
**Subtasks:** Separate tasks are needed for reading from three different tables. No mother task is needed.

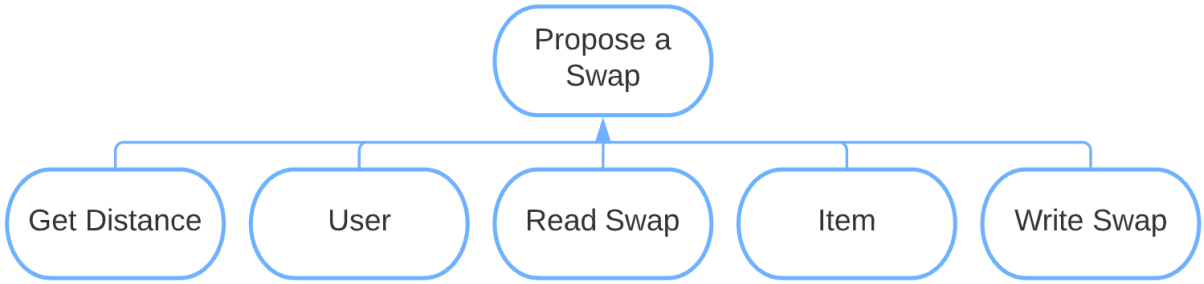
## Abstract Code

- User clicked one of the following:
  - ***Detail Link*** on one of the items listed in the **My Items Form**
  - ***Detail Link*** on one of the items listed in the **Search Result Form**
  - ***Desired Item Title*** link on **Accept/Reject Swap Form**
  - ***Proposed Item Title*** link on **Accept/Reject Swap Form**
  - ***Detail Link*** on one of the items listed in the **Swap History Form**
- Query the items table to gather **Item** Number, **Item** Title/name, **Item**-specific attributes, **Item** descriptions and **Item**'s UserId using itemID from the clicked link
- Store item's UserID in $ownerUserId
- Display **Item** Number, **Item** Title/name, **Item**-specific attributes, **Item** descriptions
- Get current logged in User using $UserID
- Check to see if the item belongs to another user, by comparing the Owner User's Id with the id of current logged in user:
  - If item belongs to another user ($ownerUserId != $UserId):
    - Query the user table using $ownerUserId to gather the Owner user's nickname, city, state, postal code, and Rating.
    - Store Owner User's postal code in $ownerUserPostalCode
    - Display Owner user's Nickname, City, State, Postal code, and Rating in the **View Item Form**.
    - Query the user table to gather the postal code of the current user using $UserId.
    - Store the postal code for the current user in $currentUserPostalCode
    - Calculate and store distance between current user and owner user in $userDistance
    - Display $userDistance in the **View Item Form**.
    - Query the swap table using $UserId to get a count of all the unrated swaps for the current user.
    - Store the count of unrated swaps in $unratedSwapsCount
    - Query the swap table using $UserId to get a count of all unaccepted swaps for the current user.
    - Store the count of unaccepted swaps in $unacceptedSwapsCount
    - Create a new variable $userCanSwap and set it to TRUE

- Check to see if the user has more than 2 unrated swaps or more than 5 unaccepted swaps, and the item is available for swapping
    - If ($unratedSwapsCount >2) or ($unacceptedSwapsCount >5):
        - Set $userCanSwap to FALSE
- If $userCanSwap is Set to TRUE ($userCanSwap == TRUE):
    - Create a new variable $ItemIsAvailable and set it to TRUE
    - Query the swap table to get the count of all accepted or pending swaps where the ItemId is equal to the proposed Item's Id or desired Item's Id.
    - Store this count as $ItemInPlayCount
    - If there are any pending or accepted swap that has this item either as a proposed item or desired item (if $itemInPlayCount >0):
        - Set $ItemIsAvailable to FALSE
- If the User can make a swap and the item is available for swapping (If ($userCanSwap == TRUE and $ItemIsAvailable == TRUE)):
    - Display ***Propose Swap*** button in the **View Item Form**.

## Propose a Swap

### Task Decomp




**Lock Types:** Lookups on Item, Postal Code, Swap, and User tables and Write on Swap table
**Number of Locks:** Several different schema constructs are needed
**Enabling Conditions:** User is logged in. User has less than 3 unrated swaps
**Frequency:** Very common
**Consistency (ACID):** Not critical. It is implied lookups must happen before a write can be made.
**Subtasks:** Separate tasks are needed for reading from Item table, Postal Code table and User table and reading and writing to Swap table. No mother task is needed.

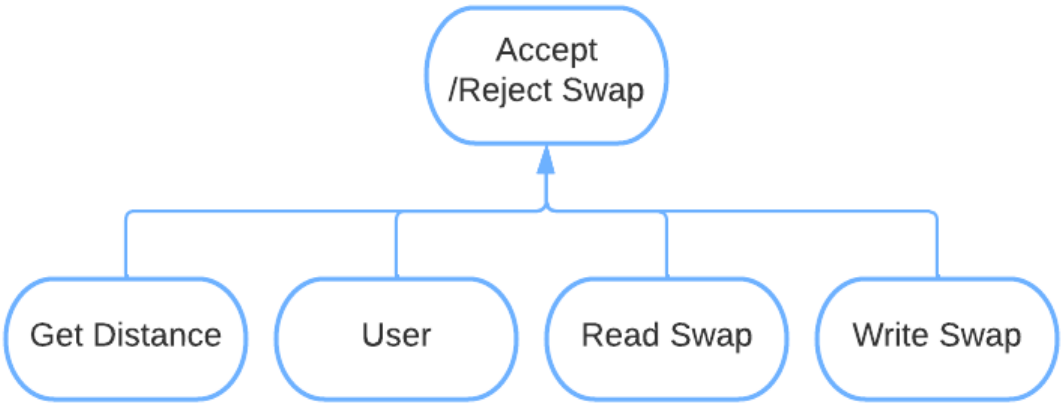
### Abstract Code:

- User clicked on the ***Propose Swap*** button for an available counterparty item in the **View Item Form** for that item.
- Get current logged in User using $UserID
- Query the swap table using $UserId to get a count of all the unrated swaps for the current user.
- Store the count of unrated swaps in $numberOfUnratedSwaps
- If User's number of unrated swaps ($numberOfUnratedSwaps) > 2:
    - Go back to the **View Item Form**, display an error message.
- Else:
    - Query the item table using the itemId to get the desired item's title and item's userId
    - Store items's userID in $desiredItemUserId
    - Query the user table using $desiredItemUserId to get the Postal code of the Desired Item's owner.
    - Store desired item owner's postal code in $desiredItemOwnerPostalcode
    - Query the user table to gather the postal code of the current user using $UserId.
    - Store the postal code for the current user in $currentUserPostalCode
    - Calculate and store the distance between current user and desired owner user in $userDistance
    - If distance between users is greater than 100 miles (if $userDistance > 100.0):
        - Display User Distance ($userDistance) in **Propose Swap Form**
    - Display desired Item Title in **Propose Swap Form**
    - Create an array variable $omitUserItemIds
    - Query the items table to get all the available items for current user ($userID)
    - For each available item:
        - Store the item id in $eachAvailableItemId
        - Query the swap table to get the count of all rejected swaps where (swap proposer user id is equal to current user ID ($userId) and counterparty user id is equal to desired item's user ID(desiredItemUserId) and proposed Item id is equal to $eachAvailableItemId and Desired item id is equal to itemId )OR (swap proposer user id is equal to the desired item user id($desiredItemUserId) and counterparty user id is equal to the current user id ($userId) and proposed item id is equal to itemId and desired item id is equal to $eachAvailableItemId)
        - Store this count in $eachItemAlreadyRejectedcount
        - If the proposed item and desired item have already been rejected before (if $eachItemAlreadyRejectedCount > 0):
            - Add $eachAvailableItemId to $omitUserItemIds
    - Query the items table to get all the available items for currentuser ($userId) whose item id is not in $omitUserItemIds

- Display each item from the query result in the **Propose Swap Form** with a radio button for each item.
- While no button is pressed, do nothing.
- When a button is pushed, then do the following:
  - Click ***Confirm*** button –
    - Add **Swap** using desired item ID (itemId), proposed Item Id (Item Id from the selected radio button), proposer user ID (current user's ID/$userId), counterparty user ID($desiredItemUserId).
    - Display **Swap Created Form**.
    - While no button is pressed, do nothing.
    - When a button is pushed, then do the following:
      - Click ***OK*** button - **View Main Menu**

## Accept/Reject Swap

### Task Decomp



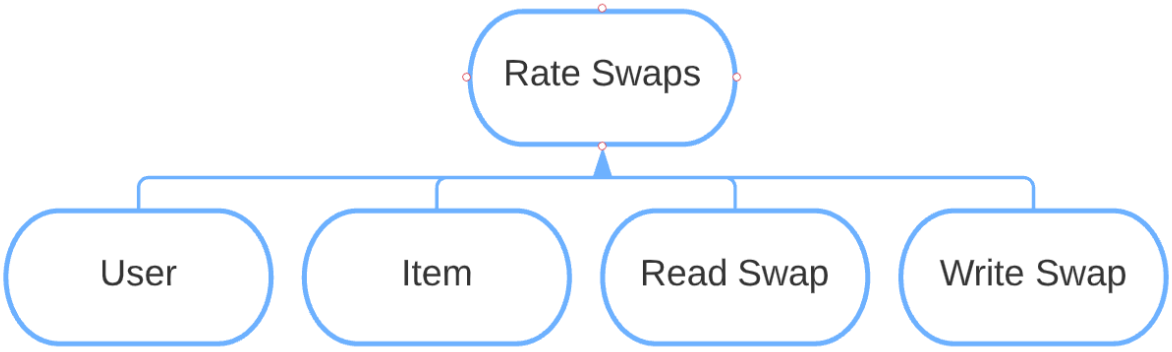

**Lock Types:** Lookup and Update needed for Swap table. Lookups on User and Postal Code table to calculate distance between users.
**Number of Locks:** Several schema constructs are needed.
**Enabling Conditions:** Enabled by a user's login and a proposed swap
**Frequency:** Very common. Used in every swap by the CounterParty user.
**Consistency (ACID):** Not critical. It is implied that a lookup must happen before an update can be made.
**Subtasks:** Separate tasks are needed for looking up swaps, calculating distance between users and updating the database with new swap status. No mother task is needed.

# Abstract Code:

- User clicked on ***Number of Unaccepted Swaps link*** on the **Main Menu Form** or accepted any pending swap
- Get current logged in User Id using $UserID
- Query the swap table to find the count of unaccepted swaps for the current User using $UserID
- Store the count of all unaccepted swaps for current user in $numPendingSwaps
    - If $numPendingSwaps > 0:
        - Query the swap table using $UserId to gather Date, DesiredItem, Proposer, Rating, Distance, and Proposed Item for all unaccepted swaps for the current user
        - Display Date, DesiredItem, Proposer, Rating, Distance, and Proposed Item for each unaccepted swaps along with an ***Accept*** and ***Reject*** Button
    - If $numPendingSwaps == 0:
        - **View Main Menu**
- While no button is pressed, do nothing.
- When a button is pushed, then do the following:
    - Click ***Accept*** button:
        - Update **Swap** table using the SwapID of the Swap for which the ***Accept*** button was clicked.
        - Display **Swap Accepted Form**.
            - While no button is pressed, do nothing.
            - When a button is pushed, then do the following:
                - Click ***OK*** button – Display **Accept/Reject Swaps Form**
    - Click ***Reject*** button:
        - Update **Swap** table using the SwapID of the Swap for which the ***Reject*** button was clicked.
        - Display **Accept/Reject Swaps Form**
    - Click ***DesiredItem*** Link:
        - Jump to **Item Details Form**
    - Click ***ProposedItem*** Link:
        - Jump to **Item Details Form**

# Rate Swaps

## Task Decomp




**Lock Types:** Lookups on User, Item tables. Lookup and Update on Swap table
**Number of Locks:** Several different schema constructs are needed
**Enabling Conditions:** Enabled by user's login
**Frequency:** Fairly Common
**Consistency (ACID):** Not critical. Order not important.
**Subtasks:** Separate tasks are needed for reading and writing into the Swap table and for reading on User and Item tables. No mother task is needed.

## Abstract Code

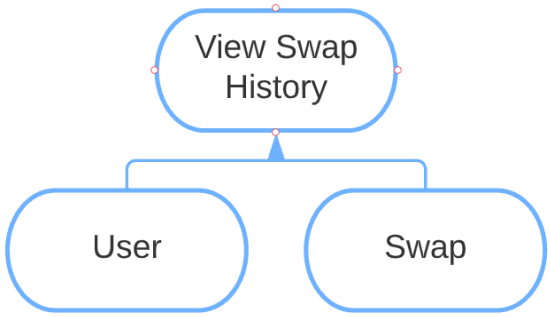
- User can trigger **Rate Swaps** task from three different forms. $UserID is the ID of the current user using the system from the HTTP Session/Cookie.
    - User is currently on **Rate Swaps Form**
        - **Rate Swaps** lookup subtask will be triggered upon visiting this form.
        - Using $UserID, find all unrated swaps in **Swap** table that **User** is a part of.
        - Display **Swap** AcceptedRejectedDate
        - Display My Role
            - For My Role, if $UserID is in **Swap** Proposer, then set My Role to Proposer
            - Else, set My Role to CounterParty.
        - Display Proposed Item Name
            - Using the **Swap** ProposedItemID, we can query the **Item** table to get the proposed item's name
        - Display Desired Item Name

- Using the **Swap** CounterPartyItemId, we can query the **Item** table to get the counter party item's name
- Display Other User Name
  - If $UserId is in **Swap** Proposer, store **Swap** CounterParty in $otherUserID
  - Else store **Swap** Proposer in $otherUserID
  - Use $otherUserID to query the **User** table, to get the name of the other user
- User is currently on **Swap History Form**
- User is currently on **Swap Details Form**

- In all forms, **Rate Swaps** task can be triggered to update into **Swap** table from selecting an option in a dropdown.
  - Dropdown will list rating choices (0-5)
- Once rating choice is selected from *rating dropdown***,** store rating in $rating and start **RateSwaps** subtask for update into **Swap** table
  - Given UserSwapID is ID of current swap whose rating was just entered in by the user, query the **Swap** table for **Swap** SwapID== UserSwapID.
    - If the current User through $UserID is found in **Swap** Proposer, then update **Swap** ProposerRating to rating
    - If the current User through $UserID is found in **Swap** CounterParty, then update **Swap** CounterPartyRating to rating

## View Swap History

### Task Decomp




**Lock Types:** Lookup on Swap table and User table
**Number of Locks:** Couple different schema constructs are needed
**Enabling Conditions:** User is logged in
**Frequency:** Fairly common

**Consistency (ACID):** Not critical. Order not important.
**Subtasks:** Separate tasks are needed for reading from two different tables, User and Swap tables.

## Abstract Code

- User can view the history of all swaps by clicking on ***Swap history*** in the **Main Manu Form** after **Log in.**
- Run **View Swap History** task: Query for user's all swaps by using $UserID from the **Log in**.
  - For the summary of user's all swaps:
    Find and display user's total proposed swaps;
    Find and display user's total received swaps;
    Find and display total accepted swaps;
    Find and display total rejected swaps;
    Find and display % of rejected swaps rounded to tenths**.**
  - For the list of user's all swaps:
    Find and display swap proposed date sorted by ascending order;
    Find and display swap acceptance/Rejection date sorted by descending order;
    Find and Display swap status (accepted or rejected);
    Find and display user's role in proposal (proposer or counterparty);
    Find and display proposed item title;
    Find and display desired item title;
    Find and display other user's nickname;
    Find and display the rating that the user gave to the other user for accepted swaps.

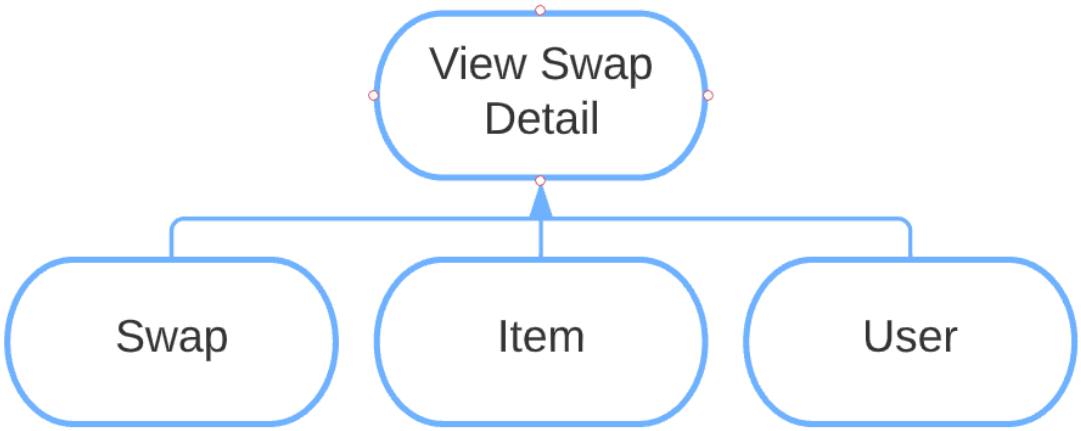
If not rated:

- user should click to ***Main Menu*** – Jump to **Main Menu** Task
- user should click to ***Unrated Swaps*** button on "***Main Menu***" – Jump to **Rate Swaps** task

If needed, click on ***Detail*** to see **Swap Detail Form** for listed swap.

# View Swap Detail

## Task Decomp




**Lock Types:** Lookup on Swap table, Item table, and User table
**Number of Locks:** Three different schema constructs are needed.
**Enabling Conditions:** User is logged in
**Frequency:** Fairly common
**Consistency (ACID):** Not critical. Order not important.
**Subtasks:** Mother task not needed. No decomposition needed.

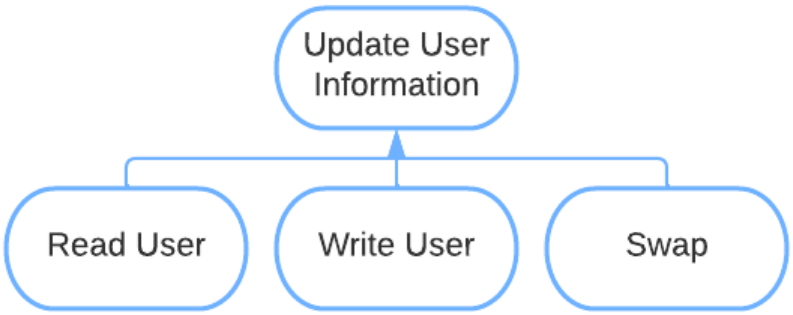
## Abstract Code

- User clicked on the ***details*** button from **Swap Details:**
  - Run the **View Swap Detail** task: Query the Swap, Item, User tables to provide details on swap in question. $UserID is the ID of the current user using the system from the HTTP Session/Cookie.
  - Query the **Swap** table to find a swap with SwapID of $currentSwapID where $currentSwapID is the SwapID of the swap in question. Display **Swap** ProposedDate, AcceptedRejectedDate, Status
  - Display My Role
    - For My Role, if $UserID is in **Swap** Proposer, then set My Role to Proposer
    - Else, set My Role to CounterParty.
  - Display Rating
    - For Rating, if $UserID is in **Swap** Proposer, then set Rating to **Swap** ProposerRating

        - Else, set Rating to **Swap** CounterPartyRating
    - Store **Swap** ProposedItemID and DesiredItemID in $proposedItemID and $desiredItemID, respectively.
    - Store in $otherUserID based on the following
        - If $UserId is in **Swap** Proposer, store **Swap** CounterParty in $otherUserID
        - Else store **Swap** Proposer in $otherUserID
    - Store **Swap** Status in $status
    - Use $proposedItemID and $desiredItemID to query Item table for respective items. Display **Item** ItemID, Title, GameType, Condition, and Description for each item.
    - Query the User table to find user where **User** Email==$otherUserID.
        - Display **User** Nickname and **Swap** SwapDistance
        - If $status is **Accepted**
            - Display **User** FirstName, Email
            - If **User** PhoneNumber Shareable==**True**
                - Display **User** PhoneNumber Number and NumberType

## Update User Information

### Task Decomp




**Lock Types:** Lookup and Insertion on User table. Lookup on Swap table.
**Number of Locks:** Several different schema constructs are needed
**Enabling Conditions:** User is logged in
**Frequency:** Somewhat common
**Consistency (ACID):** Not critical. Order not important.
**Subtasks:** Separate tasks are needed for reading and writing into the User table and reading from Swap table**.** Mother task is not needed.

# Abstract Code

- User can update User's Information by clicking ***Update My Info*** on **Main Menu Form**.
- Upon:
    - Update *password* ('$Password'), *first name*, *last name*, *nickname*, *city*, *state*, *postal code*, and *phone number* (optional) in input field
    - Store updated *password* ('$Password'), *first name*, *last name*, *nickname*, *city*, *state*, *postal code*, and *phone number* (optional) in "GameSwap Database System".
    - If user has unapproved swap or unrated swaps:
        - Display error message "Update not allowed due to unapproved swap or unrated swaps" on **Main Menu Form** while clicking on ***Update My Info.***
    - If user tries to update email address on **Update My Information Form**:
        - Display error message "Updating Email is not allowed".
    - If user tries to update phone number on **Update My Information Form** using by another user:
        - Display error message "Phone Number is in use".
    - If user tries to update postal code on **Update My Information Form:**
        - Jump to **Verify Postal Code** Task