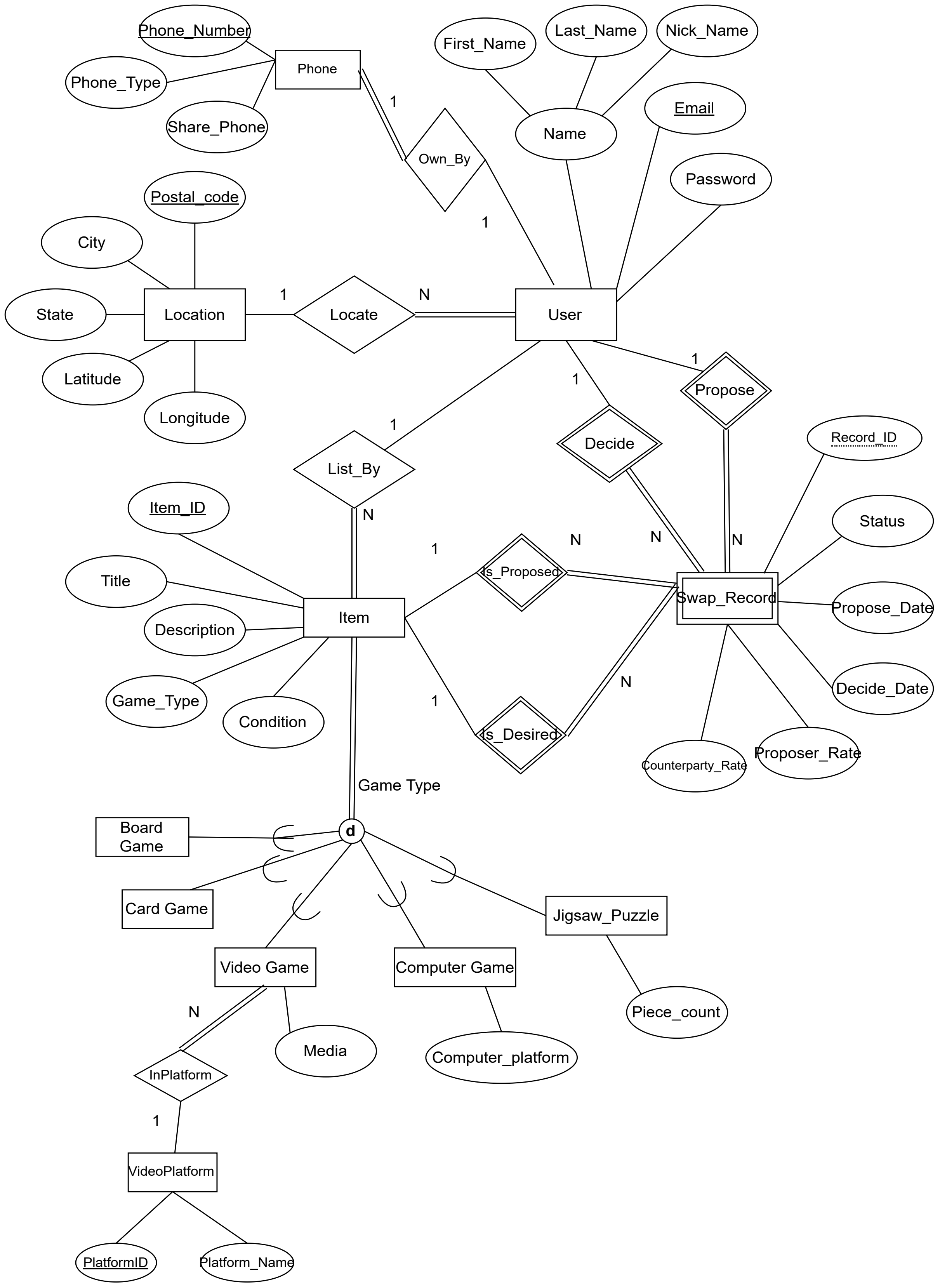
Table of Contents

EER Diagram


Phone_Number
Phone_Type
Share_Phone
Phone
1
Own_By
1
First_Name
Last_Name
Nick_Name
Name
Email
Password
Postal_code
City
State
Location
Latitude
Longitude
1
Locate
N
User
1
Propose
1
Decide
1
List_By
N
Record_ID
Status
Item_ID
Title
Description
Game_Type
Condition
Item
1
Is_Proposed
N
N
N
Swap_Record
Propose_Date
Decide_Date
1
Is_Desired
N
Counterparty_Rate
Proposer_Rate
Game Type
d
Board Game
Card Game
Video Game
Computer Game
Jigsaw_Puzzle
Piece_count
N
Media
Computer_platform
InPlatform
1
VideoPlatform
PlatformID
Platform_Name

Supplement notes on EER diagram

Supplement notes on EER diagram

In most cases, there are many equivalent ways to represent the same entity or relationship and this supplement note provides some additional descriptions/explanations on our EER diagram.
General Naming rules

- We follow the textbook recommendations, use singular nouns for entities.
- If given, we try to directly use the (entity/attribute/relationship) name listed in the instruction file (e.g.: *Password*). Otherwise, we create the name ourselves and the general principle is to make the names meaningful and foster easy understanding. E.g.: ***Is_Proposed*** relationship between item and ***Swap_Record*** suggests that this item is associated with the proposer (i.e. it is provided by proposer), while ***Is_Desired*** relationship between item and ***Swap_Record*** suggests that this item is what the proposer desired in the ***Swap_Record***.
- For (entity/attribute/relationship) names involving only one word, only the first character is capitalized; for names involving more than one word, words are connected with _ and first character for each word is capitalized.

We partition this note into three sections according to three major entities in the EER diagram: ***User***, ***Item***, ***Swap_Record***

User

(1) [Phone] ***Phone*** is NOT presented as an attribute of ***User***, but a separate entity that is connected to ***User*** via relationship ***Own_Phone***. We need this to guarantee that a phone can only be linked to a single user.
(2) [Location] ***Location*** (*Postal_Code*, *City*, *State*) is NOT presented as an attribute of User, but a separate entity that is connected to User via relationship Locate. This is because we want to restrict the valid postal code, city, state to be within the list provided.

Item

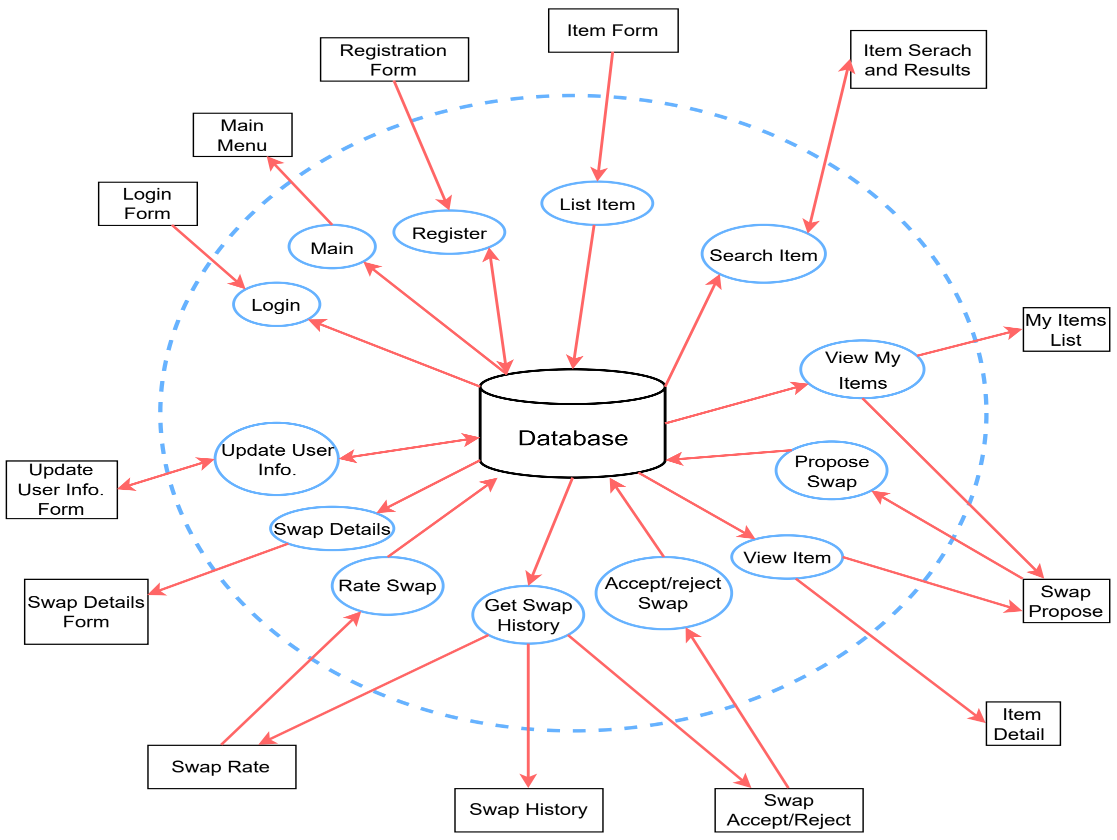
(1) [Item_ID] *Item_ID* is a surrogate key generated by the system. We include it in the EER graph as this attribute is mentioned in the instruction document but it can be removed in the current stage (as suggested by HW submission section on Canvas)
(2) [Game_Type] Item (totally) participates in the specialization to five subtypes: ***Board_Game***, ***Card_Game***, ***Video_Game***, ***Computer_Game***, ***Jigsaw_Puzzle***. In later stages, one separate table/schema will be created for each subtype and we won't create a table for the supertype. We may optionally add an attribute Game_Type to the supertype Item to make it clearer but it is not required. (Alternatively we can also create a table for Item including the common attributes and separate tables for those game types with additional attributes. This way we the Game_Type variable is necessary as both Board_Game and Card_Game do not have additional attributes and no tables are needed for them.)
(3) [Video_Platform] ***Video_Platform*** is NOT presented as an attribute of ***Video_Game*** but a separate entity. The main reason is that this way we can make the list of platforms easier to update, as required in the instruction file. In later stages, *Platform_ID* (this is a surrogate key, it is optional in this stage) will be inserted to ***Video_Game*** schema as a foreign key, so even if we change the name of Platform from "Xbox" to "Ybox" hypothetically, we only need to update

one row in ***Video_Platform*** table rather than directly update all rows involving "Xbox" in ***Video_Platform***. Similar practices can also be adopted for other attributes with pre-defined limited set of values such as *Computer_Platform*, *Condition*, *Media* but it is not required, since we can also just define the set of choices in a checkbox in front end.

Swap_Record

(1) [Swap_Record] In our preferred EER diagram, the ***Swap_Record*** entity will be created at the time when a user propose a swap and it covers all possible status (pending/accepted/rejected/completed). The advantage is that this entity is a one stop shop and the structure is simple and the draw back is that some attributes will be left as NULL (e.g.: *Decide_Date*, *Proposer_Rate* etc) and the swap record will be modified later as users proceed with the swap request. Alternatively, we can only choose to create an intermediate entity called Proposal, whose records are generated at the time of proposal. When counterparty accept/rejects a proposal, we generate a new record for ***Swap_Record*** (we can choose to delete the record in Proposal or not) containing attributes such as *Decide_Date*. The advantage of this alternative is that we won't have many missing values but the disadvantage is that the additional layer complicates the problem.
(2) [Owner] ***Swap_Record*** is a weak entity owned by ***User*** and ***Item*** via four relationship: ***Propose***, ***Decide***, ***Is_Proposed***, ***Is_Desired***. A concrete example: User A proposes swap record with counterparty B, who needs to Decide on it later. In this record, item C (currently owned by A) ***Is_Proposed*** by A and item D (currently owned by B) ***Is_Desired*** by A. Note that the relationship with item is NOT sufficient to identify ***Swap_Record*** because it is not enough to identify the proposer and counterparty since the owner if an item changes after a swap or so. On the other hand, relationship with user is NOT sufficient to identify ***Swap_Record*** because the same user may have multiple items listed.
(3) [Record_ID] This is a surrogate key (partial identifier), it is optional for the current EER and can be ignored

IFD Diagram


Registration Form
Item Form
Item Serach and Results
Main Menu
Login Form
Main
Register
List Item
Search Item
Login
My Items List
View My Items
Database
Update User Info. Form
Update User Info.
Propose Swap
Swap Details
View Item
Swap Propose
Rate Swap
Get Swap History
Accept/reject Swap
Swap Details Form
Item Detail
Swap Rate
Swap History
Swap Accept/Reject

Supplement notes on IDF diagram

Supplement notes on IDF diagram

In most cases, there are many equivalent ways to represent the same operations/tasks as well as their relationships with relevant forms. This supplement note provides some additional descriptions/explanations on our IDF diagram.
In general the tasks in the IDF diagram follows the Functionality listed in the instruction file. The only exception is Menu/Navigation (which is not listed in the instruction file), which is the main page automatically shown upon log in. The similar task is not included in the provided IFD example for GTOnline project but it is included in the provided report example for GTOnline project.
(1) Register: need user to input information, and query information (check is it's valid) from database and also write information to database (new user record)
(2) View My Items: this task is first used in the My Items List window. In addition, we also need to query my items in the swap propose window.
(3) View Item: this task is first used in the Item Detail window. In addition, it is also used in the swap propose window
(4) Get Swap History: this task is first used in the Swap History window. In addition, it is also used in the Swap Rate window and Swap Accept/Reject window (need additional filters such as only look at pending swaps)

Data Types:

User

Attribute	Data	Nullable
Email	String	Not Null
Password	String	Not Null
First_Name	String	Not Null
Last_Name	String	Not Null
Nick_Name	String	Not Null
Locate	String <digits only, foreign key from Location>	Not Null

Note: Email is the identifier for User entity; Locate is a foreign key from Location entity to indicate the location of the user and it can be ignored at this stage

Phone

Attribute	Data	Nullable
Phone_Number	String <digits only>	Not Null
Phone_Type	String	Not Null
Share_Phone	Boolean	Not Null
Own_By	String <foreign key from User>	Not Null

Note: Phone_Number is the identifier for Phone entity; Own_By is a foreign key from User entity to indicate the owner of the phone and it can be ignored at this stage

Location

Attribute	Data	Nullable
Postal_Code	String <digits only>	Not Null
City	String	Not Null
State	String	Not Null
Latitude	Float	Not Null
Longitude	Float	Not Null

Note: Postal_Code is the identifier for Location entity

Item

Attribute	Data	Nullable
Item_ID	Integer	Not Null
Title/Name	String	Not Null
Condition	String	Not Null
Description	String	Null
List_By	String <foreign key from User>	Not Null

Five subtypes of items (each of the subtype inherits all the attributes of Item, we only list the incremental attribute in the tables below to avoid repetition)

Board_Game

Attribute	**Data**	**Nullable**
No additional attributes beyond attributes inherited from Item		

Card_Game

Attribute	**Data**	**Nullable**
No additional attributes beyond attributes inherited from Item		

Jigsaw_Puzzle

Attribute	**Data**	**Nullable**
Piece_Count	Integer	Not Null

Computer_Game

Attribute	**Data**	**Nullable**
Computer_Platform	String	Not Null

Video_Game

Attribute	**Data**	**Nullable**
Media	String	Not Null
In_Platform	Integer <foreign key from Video_Platform>	Not Null

Note: Item_ID is the identifier for all subtypes, it is a surrogate key and can be ignored at this stage (though this one is explicitly mentioned in the instruction file). List_By is a foreign key from User to indicate the user who list this item and it can be ignored at this stage. Board_Game and Card_Game has no additional attributes beyond the ones inherited from the supertype Item; In_Platform is a foreign key from Video_Platform to indicate the platform of the video game and it can be ignored at this stage (as noted previously, we do it this way to make the platform list of video game more easily updatable. We can also directly make it an attribute of Video_Game, similar to Media or Computer_Platform, or alternatively we can use similar strategy for Media and Computer_Platform connect using a "In_Media", "In_Platform" relationship).

Video_Platform

Attribute	**Data**	**Nullable**
Platform_ID	Integer <surrogate key>	Not Null
Platform	String	Not Null

Note: Platform_ID is the identifier for Video_Platform, it is a surrogate key and can be ignored at this stage.

Swap_Record

Attribute	**Data**	**Nullable**
Record_ID	Integer <surrogate key>	Not Null
Proposer	String <foreign key from User>	Not Null
Counterparty	String <foreign key from User>	Not Null

Proposed_Item	Integer <foreign key from Item>	Not Null
Desired_Item	Integer <foreign key from Item>	Not Null
Status	String	Not Null
Propose_Date	String	Not Null
Decide_Date	String	Null
Proposer_Rate	Integer	Null
Counterparty_Rate	Integer	Null

Note: Record_ID is the partial identifier for weak entity Swap_Record, it is a surrogate key and can be ignored at this stage. Proposer is a foreign key from User to indicate the proposer of the swap, it can be ignored at this stage. Counterparty is a foreign key from User to indicate the counterparty to the proposer in this swap, it can be ignored at this stage. Proposed_Item is a foreign key from Item to indicate the item provided by the proposer, it can be ignored at this stage. Desired_Item is a foreign key from Item to indicate the item desired by the proposer, it can be ignored at this stage. Record_ID, Proposer, Counterparty, Proposed_Item, Desired_Item together identify Swap_Record.

Business Logic Constraints

General

1. All information is maintained outside of the application by a database administrator (i.e. administrator does not maintain the database with the same UI)

User

- [Registration] Users who are new to the GameSwap service must register first
- [Registration] Users who already have an existing GameSwap account will not be able to register
- [Login] Users need to log in to the GameSwap service with email or phone before they can make any changes
- [Login] All users can only browse, edit and make changes to the sites that they are assigned
- [Login] Users will be redirected to the main menu after login or register
- [Login] Users will be redirected to the login page (if not log in yet) or main menu (if already log in) if they are trying to access an unauthorized page
- [Email] (Optional/Recommended) Email must follow the format xxx@xxx.xx
- [Name] (Optional/Recommended) First_Name, Last_Name, Nick_Name must be less than 30 characters
- [Name] Different users are allowed to have the same Nick_Name
- [Password] (Optional/Recommended) Password must be at least 6 characters

Location

- [Valid Info] The postal code, city, state information user input must be valid, match existing records

Phone

- [Phone_Number] Phone numbers are formatted as xxx-xxx-xxxx
- [Phone_Type] Phone number type can only be one of the following three: Home, Work, Mobile
- [Unique] (Already reflected in EER) Each user can have up to one phone number and different users cannot share the same phone

Item

- [Description] Item description is optional
- [Condition] Item condition is limited to “Mint”, “Like New”, “Lightly Used”, “Moderately Used”, “Heavily Used”, “Damaged/Missing parts”
- [Game_Type] Game_Type can only be one of the following: “Board Game”, “Card Game”, “Video Game”, “Computer Game”, “Jigsaw Puzzle”
- [Platform] Platform for video game is limited to “Nintendo”, “PlayStation”, “Xbox”
- [Media] Media for video game is limited to “Optical Disc”, “Game Card”, “Cartridge”

- [Platform] Platform for computer game is limited to “Linux”, “macOS”, “Windows”
- [List Item] Users are not allowed to list a new item if that user has more than 2 unrated swaps or more than 5 unaccepted swaps

Swap

- [User] Only users that have listed items will be able to swap items with each other, a user with no listed items may browse items but cannot swap
- [Item] Items with a pending swap or complete swap are not available for swapping
- [Item] After a swap, the owner of the item change and the new owner may choose to list it again with a new item record with updated information
- [User] A user cannot swap items with him/herself
- [Rejection] If a swap is rejected, that specific item-for-item swap cannot be proposed again
- [Acceptance] The counterparty will see the contact information for the proposer if a swap is accepted
- [Rate] Both users must rate each other on a scale of 0-5 after to mark the swap as completed
- [Swap Status] Swap status can only be one of the following four: Pending, Rejected, Accepted (before both side rate the swap), Completed (after both side rate the swap)

Main Menu

- [Display Statistics] My rating, Unaccepted swaps, Unrated swaps statistics will be displayed in the main menu
- [Functionality] Buttons on List Item, My Items, Search Items, Swap History, Update My Info, Logout will be displayed in the main menu
- [Unaccepted Swaps] A link is provided if the number is greater than 0; The number will be in bold and red if any swaps more than 5 days old or user has more than 5 unaccepted swaps
- [Unrated Swaps] A link is provided if the number is greater than 0; The number will be in bold and red if the number is greater than 2

List Item

- [Display Message] Success or Error messages will be displayed if an item is or not successfully listed
- [Listing Form] Only necessary fields associated with different game type are listed

My Items

- [Summary Statistics] A summary statistics counting the number of items in each type will be listed when view my items
- [Details] Link to item detail will be provided for each individual item

Search for Items

- [Search Method] Users are provided with four search options: by keyword, within the user's postal code, within X miles of the user, within a specific postal code entered by the user
- [Search method] Search methods are exclusive
- [Search Results] Only Items available for swapping are included in the search results
- [Not Results Message] No Results Message will be displayed if no items matching the search request and user will return to the search form
- [Postal code] Must be valid, can be found in the database
- [Search Results] Link to item detail will be provided for each individual item, distance to the item owner will be provided

Item Details

- [Button] If the current user does not have more than 2 unrated swaps, or more than five unaccepted swaps, and the item is available for swapping, a "Propose swap" option should be displayed
- [Counterparty] If the item belongs to another user, then their nickname, city, state, postal code should be displayed along with their swapper rating, rounded to hundredths (defaulting to "None" if no ratings exist) and their distance from the user, rounded to tenths, unless they are in the same postal code, in which case the distance should not be displayed
- [Distance] Green background if the distance between 0-25 miles, yellow background if the distance between 25 and 50 miles, orange background if the distance between 50 and 100 miles and red background if above 100 miles

Swap Propose

- [Distance] Warning message shown if the counterparty is >=100 miles
- [Selector] Selectors are used to select the item used in exchange for the desired item

Swap Accept/Reject

- [Display] For each proposal, show the date proposed, desired item's title, proposer's nickname, their rating (rounded to hundredths), distance from user (rounded to tenths), and proposed item title, ordered by proposal date
- [Link] Link to desired item and proposed item is shown
- [Action] Only "Accept" and "Reject" are valid actions
- [Display] If the swap is accepted, display a dialog with the proposer's email, first name, and phone number/type, if available and if sharing option is set.

Rate Swaps

- [Order] Swap records ordered by acceptance date descending

- [Action] Choosing a rating will record it in the database and remove the swap from the list.
- [No results] If no additional swaps need rating, return the user to the main menu

Swap History

- [Order] Sorted by acceptance/rejection date descending and swap proposed date ascending
- [Display] Detail of each item link will be provided; rate drop down menu shown for unrated swaps
- [Summary] At the top of the form should be a summary, showing the logged in user's total swaps proposed, total received, subtotals for accepted and rejected, and % rejected, rounded to tenths
- [Color] % rejected will be in red background if value is greater than 50%

Update User Info

- [Eligibility] Cannot make any updates and will be shown an error message if user has any unapproved swaps (as proposer or counterparty) or unrated swaps and try to update info
- [Email] Cannot update email
- [Phone] Error message if a user try to change phone number to a number used by another user

Task Decomposition/Abstract Code

General Rules

We follow the rules in the provided example

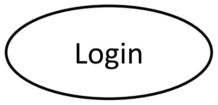
- Form: **Form**
- Button: ***Button***
- Task: **Task**
- Input Fields: *Input_Fields*
- Table: Entity_type
- A certain entity/record/row (the record found in the proceeding operation): entity
- Attribute/column: entity.Attribute or Entity.Attribute
- Input content typed in the input field: “$input_content”
- Strings: “string”
- Tab: **“Tab”**

For subtypes of Item entity, we directly use Item.Attribute to refer to an Attribute that is shared by all subtypes (i.e. when we search for Item.Description, we are searching for Description for all five subtypes, this is equivalent to search one by one but can save us some space). For attribute that is only attached to a specific subtype, we use notation such as Video_Game.Media.

Login

Login

Task Decomp



Lock type: read-only on User table
Number of Locks: Single
Enabling Condition: None
Frequency: Around 200 logins per day
Consistency (ACID): not critical, order is not critical
Subtasks: Mother Task is not needed. No decomposition needed.

Abstract Code

- If ***Register*** button is clicked:
    - Go to **<u>Registration</u>** form
- User enters *email/phone* (“$Email_or_Phone”) and *password* (“$Password”) into input fields
- If data validation is successful for both *email/phone* and *password* input field, then
    - When ***Login*** button is clicked:
        - If “@” in “$Email_or_Phone”:
            - If (“$Email_or_Phone” is not found in User.Email) or (“$Email_or_Phone” is found in User.Email but user.Password != “$Password”):
                - 3.1.1.1.1 Go back to **<u>Login</u>** form, with error message
            - 3.1.1.2 Else
                - 3.1.1.2.1 Store login information as session variable “$UserID”
                - 3.1.1.2.2 Go to **<u>Main Menu</u>**
        - 3.1.2 Else:
            - 3.1.2.1 If (“$Email_or_Phone” is not found in Phone.Phone_Number) or (user.Password!=”$Password” for user that user.Email=phone.Own_By for phone that phone.Phone_Number=”Email_or_Phone”):
                - 3.1.2.1.1 Go back to **<u>Login</u>** form, with error message
            - 3.1.2.2 Else
                - 3.1.2.2.1 Store login information as session variable “$UserID”
                - 3.1.2.2.2 Go to **<u>Main Menu</u>**

Registration

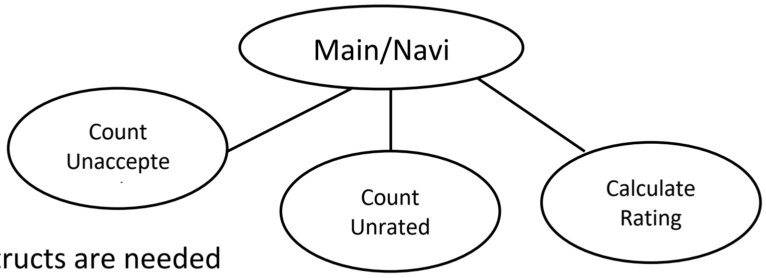
Registration

Task Decomp

Lock type: Write lock-insertion on User
Number of Locks: Single
Enabling Condition: None
Frequency: Around 10 registrations per day
Consistency (ACID): not critical, order is not critical
Subtasks: Mother Task is not needed. No decomposition needed.

Abstract Code

- 1 If data validation is successful for all fields (*Email, Nick_Name, Password, City, First_Name , State, Last_Name, Postal_Code, Phone*), then
  - When ***Register*** button is clicked:
    - If "$Email" is found in User.Email:
      - Go back to ***Register*** form, with error message "Email already used"
    - Else if "$Postal_Code" is not found in Location.Postal_Code:
      - Go back to Register form, with error message "Please enter valid Postal Code"
    - Else if "$Postal_Code" is found in Location.Postal_Code and ("$City"!=location.City or State!=location.State)
      - Go back to ***Register*** form, with error message and suggestion for City and State info
    - Else:
      - Store user information in User
      - Go to **Login** form



Main/Navigation Menu

Main/Navigation Menu

Task Decomp

Lock Types: All are Read-only.
Number of Locks: Several different schema constructs are needed
Enabling Conditions: Trigger by successful login.
Frequency: User Detail and Menu Options have the same frequency. Viewed on the main/home page after every login
Consistency (ACID): not critical, order is not critical.
Subtasks: All tasks must be done, can be done in parallel. Mother task is required to coordinate subtasks. Order is not necessary

Abstract Code

- Show **"My Rating", "Unaccepted swaps", "Unrated Swaps"** tab and ***Log out, List Item, My Items, Search items, Swap history, Update my info*** button
- Query and count swap_record in Swap_Record where (swap_record.Counterparty=="$UserID") and (swap_record.Status=="Pending"), in **"Unaccepted swaps"**, show color
- Query and count swap_record in Swap_Record where (swap_record.Proposer=="$UserID" and swap_record.Status=="Accepted" and swap_record.Proposer_Rate=NULL) or (swap_record.Counterparty=="$UserID" and swap_record.Status=="Accepted" and swap_record.Counterparty_Rate=NULL), in **"unrated swaps"**, show color
- Query and calculate mean of (swap_record.Counterparty_Rate in Swap_Record where (swap_record.Proposer=="$UserID" and swap.Counterparty_Rate is NOT NULL)) and (swap_record.Proposer_Rate in Swap_Record where (swap_record.Counterparty=="$UserID" and swap.Proposer_Rate is NOT NULL))[1], in **"my rating"**
- Upon
  - Click ***Logout*** button --- Jump to the **Login** task
  - 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 the **Search items** task
  - Click ***Swap history*** button --- Jump to the **Swap history** task
  - Click ***Update my info*** button --- Jump to the **Update User info** task
  - Click Unaccepted swaps link
    - If the number in **"Unaccepted swaps"**>0
      - Jump to the **Accept/Reject Swap** task
    - Else
      - Stay in the **Main** menu
  - Click "Unrated swaps" button
    - If the number in **"Unrated swaps"**0

[1] Here we assume the rating of a user is the average rating given by his/her counterparty in a swap (i.e. how other people rate him/her), instead of how he/she rate other people

        - Jump to the **Rate Swaps** task
    - Else
        - Stay in the <u>**Main**</u> menu

List Item

List Item

Task Decomp

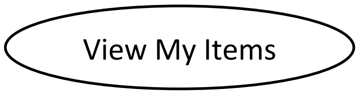
Lock Types: Writer only insertion
Number of Locks: Single
Enabling Conditions: Enabled by User Login
Frequency: Around 1000 listings per day
Consistency: Not critical
Subtasks: Mother Task is not needed. No decomposition needed

Abstract Code

- User clicked on ***List item*** button from <u>**Main**</u> Menu
- If the number in **“Unaccepted swaps”**>5 or the number in **“Unrated swaps”**>2:
    - Show error message
- Else
    - Run the **List item** task, show *GameType, Title, Condition, Description* fields
    - If “$GameType”=”Jigsaw puzzle”
        - Show *PieceCount* field
    - Else if “$GameType”=”Video game”
        - Show *Platform, Media* field
    - Else if “$GameType”==”Computer game”
        - Show *Platform* field
    - If data validation is successful for all fields, then
        - When ***List Item*** button is clicked
            - Store item information in Item Table
            - Show item insertion succeeds information
            - If ***OK*** button is clicked
            - Go to <u>**Main**</u> Menu

View My Items

View My Items



Task Decomp

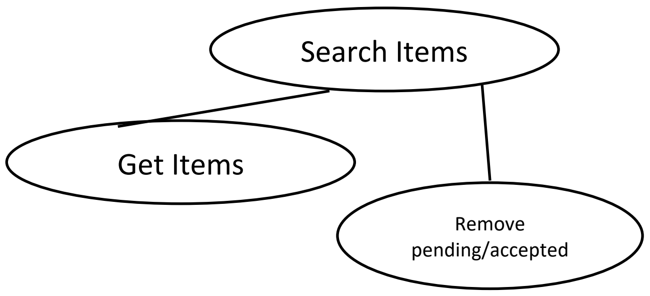
Lock Types: read-only lookups for show own items and show item statistics by game type
Number of Locks: Several different schema constructs are needed
Enabling Conditions: Enabled by a user's login
Frequency: Around 500 views per day
Consistency (ACID): not critical
Subtasks: All tasks

Abstract Code

- User clicked on ***My Items*** button from **Main** Menu:
- Run the **View My Items** task: query for information to get item in Item where (item.List_By=”$UserID”)
- Query item in Item where item.List_By=”$UserID”
- Count and display items owned by current user by game type

Search Items

Search Items



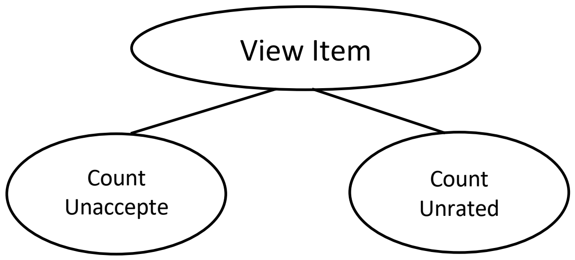
Task Decomp

Lock Types: read-only lookups
Number of Locks: Two
Enabling Conditions: Enabled by a user's login
Frequency: High frequency
Consistency (ACID): not critical
Subtasks: Mother Task is needed. All tasks must be done. Need to finish Get Items first to get a preliminary list of items (query Item and User), then finish Remove Pending/Accepted to remove those items currently in a pending/accepted swap request (query Swap_Record). Order is important. Mother task is required to coordinate subtasks.

Abstract Code

- A ratio group (i.e. checkbox that can only select a single choice) (with four options: *By keyword, In my postal code, Within miles, In postal code*) for SearchType with parameter *SearchParam* associates with all four options except *In my postal code* shows up when user clicked on ***My Item*** button from **Main** Menu
- When ***Search*** button if clicked
  - If data validation is successful ("$SearchType" is available and "$SearchParam" is available when necessary)
    - Query user.Locate for user="$UserID" in User
    - If "$SearchType"="By keyword":
      - (Query item in Item where (item.Title Contains "$SearchParam" or item.Game_Type Contains "$SearchParam" or item.Condition Contains "$SearchParam" or item.Description Contains "$SearchParam") Union (Query computer_game in Computer_Game where computer_game.Computer_Platform Contains "$SearchParam") Union (Query video_game in Video_Game where video_game.Media Contains "$SearchParam") Union (Query video_game in Video_Game where video_game.In_Platform in video_platform.Platform_ID for (video_platform in Video_Platform where video_Platform.Platform Contains "$SearchParam"))
      - Query item.List_By in Item to get user and user.Locate in User to get location.Postal_Code in Location for each item and calculate distance and list results
      - The above two steps generate the tentative list of items in this case
    - Else if "$SearchType"="In my postal code":
      - Query user.Locate in User where user.Email="$UserID" and store results as locate, query item in Item for item.List_By=user.Email for user in User where user.Locate=locate and user.Email!="$UserID" , calculate distance (0 in this case)
      - The above step generates the tentative list of items in this case
    - Else if "$SearchType"="Within miles":
      - Query user.Locate in User where user.Email="$UserID", store as locate

        - Query user in User where Distance(user.Locate,locate)<"$SearchType", store as user
        - Query item in Item where item.List_By in user
        - The above steps generate the tentative list of items in this case
    - Else if "$Search Type"=="In postal code"
        - Query user in User where user.Locate=="$SearchParam" and calculate distance based on Location, store as user
        - Query item in Item where item.List_by in user
        - The above steps generate the tentative list of items in this case
    - The tentative list of items is stored in items. Query item in items where (item not in swap_record.Is_Proposed or swap_record.Is_Desired for swap_record in Swap_Record such that swap_record.Status="Pending" or "Accepted" or "Completed")), store this new filtered list as items
- If length of items>0:
    - Display query results
- Else:
    - Return to the search form with error message "Sorry, no results found!"



View Item

View Item

Task Decomp

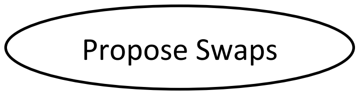
Lock Types: read-only lookups
Number of Locks: Two
Enabling Conditions: Enabled by a user's login, or by an item search
Frequency: High frequency
Consistency (ACID): not critical
Subtasks: All tasks must be done, but can be done in parallel. Mother task is required to coordinate subtasks. Order is not necessary

Abstract Code

- Upon user clicked detail link from item lists
- Query item in Item and display
- Query user in User where user.Email=item.List_By, query location in Location where location.Postal_Code=user.Locate, calculate distance based on location
- Change the background color of Distance based on its value
- Query and count swap_record in Swap_Record where (swap_record.Counterparty=="$UserID") and (swap_record.Status=="Pending"), as **"Unaccepted swaps"**
- Query and count swap_record in Swap_Record where (swap_record.Proposer=="$UserID" and swap_record.Status=="Accepted" and swap_record.Proposer_Rate=NULL) or (swap_record.Counterparty=="$UserID" and swap_record.Status=="Accepted" and swap_record.Counterparty_Rate=NULL), as **"unrated swaps"**,
- If number of unaccepted swaps<=5 and number of unrated swaps<=2
    - Show ***Propose Swap*** button, keep the item that is currently viewed as "$Item", keep the user that list this item as "$OwnerID"

Propose Swaps

Propose Swaps



Task Decomp

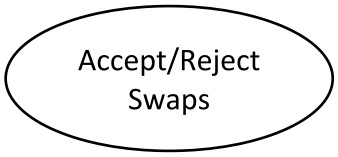
Lock Types: Writer only insertion
Number of Locks: Single
Enabling Conditions: Enabled by a user’s login and detail viewing (“$Item” and “$OwnerID” from View Item task)
Frequency: High frequency
Consistency (ACID): not critical
Subtasks: Mother Task is not needed. No decomposition needed

Abstract Code

- Upon user clicked ***Propose Swap*** button
- Change the background color of Distance based on its value (distance is already calculated before)
- Query item in Item where item.List_By=”$UserID” and item not in (swap_record.Is_Proposed where swap.record.Status=”Accepted” or “Pending” or “Completed”) and item not in (swap_record.Is_Desired where swap.record.Status=”Accepted” or “Pending” or “Completed”)
- Show items as radio check box
- When user click ***Confirm*** button
    - If check box is selected:
        - if exists swap_record in Swap_Record where (swap_record.Is_Proposed=item and swap_record.Is_Desired=”$Item”)
            - Return error message “You cannot send the same swap request that was rejected before again”
        - Else
            - Insert swap_record to Swap_Record, with status as “Pending” and Propose_Time as current time
    - Else:
        - Display error message “Please select an item to swap”

Accept/Reject Swaps

Accept/Reject Swaps



Task Decomp

Lock Types: Writer only update
Number of Locks: Single
Enabling Conditions: Enabled by a user's login and number of unaccepted swaps greater than 0
Frequency: High frequency
Consistency (ACID): not critical
Subtasks: Mother Task is not needed. No decomposition needed

Abstract Code

- Upon user click ***Unaccepted swaps***
- Query swap_record in Swap_Record where swap_record.Counterparty="$UserID" and swap_record.Status="Pending"
- Query user in User where swap_record.Proposer=user.Email, calculate distance based on location where location.Postal_Code=user.Locate
- Show swap_record and ***Accept*** and ***Reject*** buttons
- If user click ***Accept*** button
    - Change swap_record.Status to "Accepted", swap_record.Decide_Date to current date
    - Return contact information for swap proposer by query user in User where user.Email=swap_record.Proposer
- If user click ***Reject*** button
    - Change swap_record.Status to "Rejected", swap_record.Decide_Date to current date

Swap History

Swap History

Task Decomp

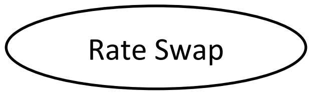
Lock Types: Writer only update
Number of Locks: Single
Enabling Conditions: Enabled by a user’s login
Frequency:
Consistency (ACID): not critical
Subtasks: Mother Task is not needed. No decomposition needed

Abstract Code

- Upon user click ***Swap history*** button
- Query swap_record in Swap_Record where (swap_record.Counterparty=”$UserID” or swap_record.Proposer=”$UserID”) and (swap_record.Status!=”Pending”)
- If swap_record.Status=”Accepted” and ((swap_record.Counterparty=”$UserID” and swap_record.Counterparty_Rate is NULL) or ((swap_record.Proposer=”$UserID” and swap_record.Proposer_Rate is NULL)):
  - Show rating check box
- Count swap_record where swap_record.Counterparty==”$UserID”
  - Count swap_record where swap_record.Counterparty==”$UserID” and swap_record.Status=”Accepted” or “Completed”
  - Calculate rate of rejection and change background color according to this rate
- Count swap_record where swap_record.Proposer==”$UserID”
  - Count swap_record where swap_record.Proposer==”$UserID” and swap_record.Status=”Accepted” or “Completed”
  - Calculate rate of rejection and change background color according to this rate
- If swap_record.Status=”Accepted” and swap_record.Counterparty=”$UserID” and swap_record.Counterparty_Rate is NULL:
  - When user select rating
    - Change swap_record.Counterparty_Rate=“$Rating”
- If swap_record.Status=”Accepted” and swap_record.Proposer=”$UserID” and swap_record.Proposer_Rate is NULL:
  - When user select rating
    - Change swap_record.Proposer_Rate=“$Rating”

Rate Swap

Rate Swap



Task Decomp

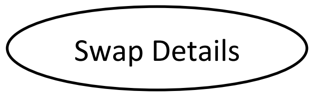
Lock Types: Writer only update
Number of Locks: Single
Enabling Conditions: Enabled by a user's login and number of unrated swaps greater than 0
Frequency: High frequency
Consistency (ACID): not critical
Subtasks: Mother Task is not needed. No decomposition needed

Abstract Code

```
●  Upon user click Unrated swaps link
●  Query swap_record in Swap_Record where (swap_record.Counterparty=”$UserID” and
   swap_record.status==”Accepted” and swap_record.Counterparty_Rate=NULL) or
   (swap_record.Proposer=”$UserID” and swap_record.Status==”Accepted” and
   swap_record.Proposer=NULL)
●  While number of query results is greater than 0
   ■  Show unrated swaps and rating dropdown menu
   ■  If user select “$Rate”
      ◆  If swap_record.Counterparty=”$UserID”
         ●  Change swap_record.Counterparty_Rate=”$Rate”
      ◆  Else
         ●  Change swap_record.Proposer_Rate=”$Rate”
   ■  If swap_record.Counterparty_Rate is NOT NULL and swap_record.Proposer_Rate is
      NOT NULL
      ◆  Change swap_record.Status=”Complete”
   ■  Query swap_record in Swap_Record where (swap_record.Counterparty=”$UserID” and
      swap_record.status==”Accepted” and swap_record.Counterparty_Rate=NULL) or
      (swap_record.Proposer=”$UserID” and swap_record.Status==”Accepted” and
      swap_record.Proposer=NULL)
●  Return to main menu
```

Swap Details

Swap Details



Task Decomp

Lock Types: Read Only
Number of Locks: Single
Enabling Conditions: Enabled by a user’s login and swap_record.proposer is current user or swap_record.counterparty is current user
Frequency: Around 1000 views per day
Consistency (ACID): not critical
Subtasks: Mother Task is not needed. No decomposition needed

Abstract Code

- Upon user click ***Detail*** button in swap history form
- query swap_record in Swap_Record
- query item in Item for swap_record.Proposed_Item and swap_record.Desired_Item
- query user in User for (swap_record.Proposer and swap_record.Counterparty) and (user.Email!=”$UserID”)

Update User Info

Update user info

Task Decomp

Lock Types: write only update
Number of Locks: Single
Enabling Conditions: Enabled by a user's login
Frequency: Infrequent
Consistency (ACID): not critical
Subtasks: Mother Task is not needed. No decomposition needed

Abstract Code

- Upon user click ***Update my Info*** button: Query user in User where user="$UserID" and show the **Update User Info** form (with *Email* not updatable)
- If data validation is successful for all fields (Email, Nick_Name, Password, First_Name, Last_Name), then
    - When ***Update*** button is clicked:
        - If "$Postal_Code" is not found in Location.Postal_Code:
            - Go back to **Update User Info** form, with error message "Please enter valid Postal Code"
        - Else if "$Postal_Code" is found in Location.Postal_Code and ("$City"!=location.City or State!=location.State)
            - Go back to **Update User Info** form, with error message and suggestion for City and State info
        - Else if "$Phone" is found in Phone and phone.Own_By!="$UserID":
            - Go back to **Update User Info** form, with error message "This phone is used by someone else"
        - Else
            - Update user in User for user where user.Email="$UserID" in User
            - Go to **Login** form