Table of Contents:

GameSwap Abstract Code with SQL:

Register

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* in User table of "GameSwap Database System".

```
INSERT INTO `user` (`email`, `password`, `first_name`, `last_name`,
`nickname`, `postal_code`) VALUES (`$email`, `$password, `$firstName`,
`$lastName`, `$nickname`, `$postalCode`);
```

- If the user enters a phone number ($phoneNum), store the number, number type ($numType), shareable flag ($numShareable) in the PhoneNumber table.

```
INSERT INTO phonenumber(`email`, `number`, `number_type`,
`share_phone_number`) VALUES (`$email`, `$phoneNum`, `$numberType`,
`$numShareable`);
```

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

Verify Postal Code

Abstract Code

- Triggered from the **Registration** form once the postal code is entered into form.
- Lookup postal code in PostalCode table.

```
SELECT postal_code FROM `postalcode` WHERE postal_code=`$postalCode`;
```

- 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

```
SELECT postal_code FROM `postalcode` WHERE postal_code=`$postalCode`
AND city=`$city` AND state=`$state`;
```

- If postal code is found, no action is required

Login

Abstract Code

- Show "***Login***", "***Register***" buttons.
- User enters email or phone number ($userLoginInput) and *password* ($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

```
SELECT `user`.password FROM `user` JOIN phonenumber ON `user`.email =
phonenumber.email WHERE (`user`.email= `$userLoginInput` OR
phonenumber.number=`$userLoginInput`);
```

  - 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 number* 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

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

```
SELECT COALESCE((((SELECT AVG(swap_proposer_rating) AS RATING_AVG FROM
swap WHERE swap_status = 'Accepted' AND proposer_email = `$userId` AND
swap_proposer_rating IS NOT NULL) + (SELECT AVG(swap_counterparty_rating) AS
RATING_AVG FROM swap WHERE swap_status = 'Accepted' AND counterparty_email =
`$userId` AND swap_counterparty_rating IS NOT NULL)) /2), ((SELECT
AVG(swap_proposer_rating) AS RATING_AVG FROM swap WHERE swap_status =
'Accepted' AND proposer_email = `$userId` AND swap_proposer_rating IS NOT NULL)),
((SELECT AVG(swap_counterparty_rating) AS RATING_AVG FROM swap WHERE
swap_status = 'Accepted' AND counterparty_email = `$userId` AND
swap_counterparty_rating IS NOT NULL)), 'None') AS USER_RATING_AVG;
```

- Display $numPendingSwaps underneath ***"Unaccepted Swaps***" button

```
SELECT COUNT(*), (DATEDIFF(CURRENT_DATE, swap.proposal_date)) AS
PENDING_DAYS FROM swap WHERE swap_status = 'Pending' AND counterparty_email =
`$userId`;
```

  - 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.

```
SELECT COUNT(*) FROM swap WHERE swap_status='Accepted' AND ((proposer_email =
`$userId` AND swap_proposer_rating IS NULL) OR (counterparty_email = `$userId` AND
swap_counterparty_rating IS NULL));
```

  - 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

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):

```
SELECT COUNT(*) FROM swap WHERE swap_status='Accepted' AND ((proposer_email =
`$userId` AND swap_proposer_rating IS NULL) OR (counterparty_email = `$userId` AND
swap_counterparty_rating IS NULL));
```

        - User cannot list item, display error message
    - If the user has more than five unaccepted swaps ($numPendingSwaps > 5):

```
SELECT COUNT(*) FROM swap WHERE swap_status = 'Pending' AND counterparty_email
= `$userId`;
```

        - User cannot list item, display error message
    - Else:
        - Item information is saved
            - Information provided in *Game Type* field ($enteredType) is stored to Item gameType. *Game platform* ($enteredGamePlatform), *media* ($media), *computer platform* ($computer_platform), *piece*($piece) are stored in Item.game_platform, Item.media, Item.computer_platform, Item.piece respectively.
            - Information provided in *Title ($*enteredTitle) is stored in Item gameTitle

- Information provided in *Condition* ($condition) is stored in Item itemCondition
- Information provided in *Description* ($description) is stored in Item itemDescription

```
INSERT INTO item (`email`, `TYPE`, `title`, `game_platform`,
`media`, `computer_platform`, `piece`, `condition`,
`description`)
VALUES (`$userId`, `$enteredType`, `$enteredTitle`,
`$enteredGamePlatform`, `$media`, `$computer_platform`,
`$piece`, `$condition`, `$description`);
```

- Item number is generated and stored in Item ItemID
- Query to get the last inserted ID, which will be the newly created item id.

```
SELECT LAST_INSERT_ID();
```

- Display Success Message showing the item's Item ItemID

View My Items

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.

```
SELECT `type` AS GAME_TYPE, COUNT(*) AS GAME_COUNT FROM `item`
where email = `$userID` GROUP BY `type`;
```

- 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.

```
SELECT item_no, type, title, `condition`, description FROM `item` WHERE email =
`$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

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 ($enteredKeyword)

```
SELECT UT.item_no, UT.TYPE, UT.title, UT.condition, UT.description,
DT.DISTANCE_CALC_MILES FROM
(SELECT `user`.email, item.item_no, item.TYPE, item.title, item.condition, item.description
FROM `user` JOIN item ON `user`.email = item.email WHERE (item.title LIKE
`%$enteredKeyword %` OR item.description LIKE `%$enteredKeyword %`) AND
`user`.email <> `$userId`) UT
JOIN
(SELECT T1.USER1_EMAIL, T2.USER2_EMAIL, ROUND((6371 * 0.621371 * (2 *
ATAN((SQRT((POWER((SIN(((RADIANS(T2.USER2_LAT)) -
(RADIANS(T1.USER1_LAT))) / 2)), 2) +  (COS((RADIANS(T1.USER1_LAT)))) *
(COS((RADIANS(T2.USER2_LAT)))) * POWER((SIN(((RADIANS(T2.USER2_LON)) -
(RADIANS(T1.USER1_LON))) / 2)), 2)))), (SQRT(1 -
(POWER((SIN(((RADIANS(T2.USER2_LAT)) - (RADIANS(T1.USER1_LAT))) / 2)), 2) +
(COS((RADIANS(T1.USER1_LAT)))) * (COS((RADIANS(T2.USER2_LAT)))) *
POWER((SIN(((RADIANS(T2.USER2_LON)) - (RADIANS(T1.USER1_LON))) / 2)),
2))))))),2) AS DISTANCE_CALC_MILES
FROM
    (SELECT U1.email AS USER1_EMAIL, U1.nickname AS USER1_NICKNAME,
U1.postal_code AS USER1_POSTAL_CODE, P1.latitude AS USER1_LAT, P1.longitude
USER1_LON, P1.City AS USER1_CITY, P1.State AS USER1_STATE FROM `user` U1
JOIN postalcode P1 ON U1.postal_code = P1.postal_code) T1
    JOIN
    (SELECT U2.email AS USER2_EMAIL, U2.nickname AS USER2_NICKNAME,
U2.postal_code AS USER2_POSTAL_CODE, P2.latitude AS USER2_LAT, P2.longitude
USER2_LON, P2.City AS USER2_CITY, P2.state AS USER2_STATE FROM `user` U2
JOIN postalcode P2 ON U2.postal_code = P2.postal_code) T2
    ON T1.USER1_EMAIL <> T2.USER2_email) DT
ON UT.email = DT.USER2_EMAIL WHERE DT.USER1_EMAIL =`$userId` ORDER BY
DISTANCE_CALC_MILES, item_no;
```

- If “*In my postal code*” is selected:
  - Find items located within users postal code

```
SELECT UT.item_no, UT.TYPE, UT.title, UT.condition, UT.description,
DT.DISTANCE_CALC_MILES FROM
(SELECT `user`.email, item.item_no, item.TYPE, item.title, item.condition, item.description
FROM `user` JOIN item ON `user`.email = item.email WHERE `user`.email <> `$userId`) UT
JOIN
(SELECT T1.USER1_EMAIL, T2.USER2_EMAIL, T1.USER1_POSTAL_CODE,
T2.USER2_POSTAL_CODE, ROUND((6371 * 0.621371 * (2 *
ATAN((SQRT((POWER((SIN(((RADIANS(T2.USER2_LAT)) -
(RADIANS(T1.USER1_LAT))) / 2)), 2) +  (COS((RADIANS(T1.USER1_LAT)))) *
(COS((RADIANS(T2.USER2_LAT)))) * POWER((SIN(((RADIANS(T2.USER2_LON)) -
(RADIANS(T1.USER1_LON))) / 2)), 2)))), (SQRT(1 -
(POWER((SIN(((RADIANS(T2.USER2_LAT)) - (RADIANS(T1.USER1_LAT))) / 2)), 2) +
(COS((RADIANS(T1.USER1_LAT)))) * (COS((RADIANS(T2.USER2_LAT)))) *
POWER((SIN(((RADIANS(T2.USER2_LON)) - (RADIANS(T1.USER1_LON))) / 2)),
2))))))),2) AS DISTANCE_CALC_MILES
FROM
    (SELECT U1.email AS USER1_EMAIL, U1.nickname AS USER1_NICKNAME,
U1.postal_code AS USER1_POSTAL_CODE, P1.latitude AS USER1_LAT, P1.longitude
USER1_LON, P1.City AS USER1_CITY, P1.State AS USER1_STATE FROM `user` U1
JOIN postalcode P1 ON U1.postal_code = P1.postal_code) T1
    JOIN
    (SELECT U2.email AS USER2_EMAIL, U2.nickname AS USER2_NICKNAME,
U2.postal_code AS USER2_POSTAL_CODE, P2.latitude AS USER2_LAT, P2.longitude
USER2_LON, P2.City AS USER2_CITY, P2.state AS USER2_STATE FROM `user` U2
JOIN postalcode P2 ON U2.postal_code = P2.postal_code) T2
    ON T1.USER1_EMAIL <> T2.USER2_email WHERE T1.USER1_POSTAL_CODE =
T2.USER2_POSTAL_CODE) DT
ON UT.email = DT.USER2_EMAIL WHERE DT.USER1_EMAIL =`$userId`
ORDER BY DISTANCE_CALC_MILES, item_no;
```

- If *“Within ‘X’ miles of me”* is selected:
  - Calculate distance from user using entered number ($userSearchMiles)
  - Find items within that range

```
SELECT UT.item_no, UT.TYPE, UT.title, UT.condition, UT.description,
DT.DISTANCE_CALC_MILES FROM
```

```
(SELECT `user`.email, item.item_no, item.TYPE, item.title, item.condition, item.description
FROM `user` JOIN item ON `user`.email = item.email WHERE `user`.email <> `$userId`) UT
JOIN
(SELECT T1.USER1_EMAIL, T2.USER2_EMAIL, ROUND((6371 * 0.621371 * (2 *
ATAN((SQRT((POWER((SIN(((RADIANS(T2.USER2_LAT)) -
(RADIANS(T1.USER1_LAT))) / 2)), 2) +  (COS((RADIANS(T1.USER1_LAT)))) *
(COS((RADIANS(T2.USER2_LAT)))) * POWER((SIN(((RADIANS(T2.USER2_LON)) -
(RADIANS(T1.USER1_LON))) / 2)), 2)))), (SQRT(1 -
(POWER((SIN(((RADIANS(T2.USER2_LAT)) - (RADIANS(T1.USER1_LAT))) / 2)), 2) +
(COS((RADIANS(T1.USER1_LAT)))) * (COS((RADIANS(T2.USER2_LAT)))) *
POWER((SIN(((RADIANS(T2.USER2_LON)) - (RADIANS(T1.USER1_LON))) / 2)),
2))))))),2) AS DISTANCE_CALC_MILES
FROM
    (SELECT U1.email AS USER1_EMAIL, U1.nickname AS USER1_NICKNAME,
U1.postal_code AS USER1_POSTAL_CODE, P1.latitude AS USER1_LAT, P1.longitude
USER1_LON, P1.City AS USER1_CITY, P1.State AS USER1_STATE FROM `user` U1
JOIN postalcode P1 ON U1.postal_code = P1.postal_code) T1
    JOIN
    (SELECT U2.email AS USER2_EMAIL, U2.nickname AS USER2_NICKNAME,
U2.postal_code AS USER2_POSTAL_CODE, P2.latitude AS USER2_LAT, P2.longitude
USER2_LON, P2.City AS USER2_CITY, P2.state AS USER2_STATE FROM `user` U2
JOIN postalcode P2 ON U2.postal_code = P2.postal_code) T2
    ON T1.USER1_EMAIL <> T2.USER2_email) DT
ON UT.email = DT.USER2_EMAIL WHERE DT.USER1_EMAIL =`$userId` AND
DT.DISTANCE_CALC_MILES <= `$userSearchMiles`  ORDER BY
DISTANCE_CALC_MILES, item_no;
```

- If *"In postal code"* is selected:
  - Perform **Validate Postal Code** task
  - If postal code entered is invalid
    - Display "Invalid Postal Code" error message
  - Else If postal code ($enteredPostalCode) entered is valid
    - Find items

```
SELECT UT.item_no, UT.TYPE, UT.title, UT.condition, UT.description,
DT.DISTANCE_CALC_MILES FROM
(SELECT `user`.email, item.item_no, item.TYPE, item.title, item.condition, item.description
FROM `user` JOIN item ON `user`.email = item.email WHERE `user`.email <> `$userId`) UT
JOIN
(SELECT T1.USER1_EMAIL, T2.USER2_EMAIL, T1.USER1_POSTAL_CODE,
T2.USER2_POSTAL_CODE, ROUND((6371 * 0.621371 * (2 *
ATAN((SQRT((POWER((SIN(((RADIANS(T2.USER2_LAT)) -
(RADIANS(T1.USER1_LAT))) / 2)), 2) +  (COS((RADIANS(T1.USER1_LAT)))) *
(COS((RADIANS(T2.USER2_LAT)))) * POWER((SIN(((RADIANS(T2.USER2_LON)) -
(RADIANS(T1.USER1_LON))) / 2)), 2)))), (SQRT(1 -
(POWER((SIN(((RADIANS(T2.USER2_LAT)) - (RADIANS(T1.USER1_LAT))) / 2)), 2) +
(COS((RADIANS(T1.USER1_LAT)))) * (COS((RADIANS(T2.USER2_LAT)))) *
POWER((SIN(((RADIANS(T2.USER2_LON)) - (RADIANS(T1.USER1_LON))) / 2)),
2))))))),2) AS DISTANCE_CALC_MILES
FROM
    (SELECT U1.email AS USER1_EMAIL, U1.nickname AS USER1_NICKNAME,
U1.postal_code AS USER1_POSTAL_CODE, P1.latitude AS USER1_LAT, P1.longitude
USER1_LON, P1.City AS USER1_CITY, P1.State AS USER1_STATE FROM `user` U1
JOIN postalcode P1 ON U1.postal_code = P1.postal_code) T1
    JOIN
    (SELECT U2.email AS USER2_EMAIL, U2.nickname AS USER2_NICKNAME,
U2.postal_code AS USER2_POSTAL_CODE, P2.latitude AS USER2_LAT, P2.longitude
USER2_LON, P2.City AS USER2_CITY, P2.state AS USER2_STATE FROM `user` U2
JOIN postalcode P2 ON U2.postal_code = P2.postal_code) T2
    ON T1.USER1_EMAIL <> T2.USER2_email WHERE T2.USER2_POSTAL_CODE =
`$enteredPostalCode`) DT
ON UT.email = DT.USER2_EMAIL WHERE DT.USER1_EMAIL =`$userId`
ORDER BY DISTANCE_CALC_MILES, item_no;
```

- 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

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 ($itemId) from the clicked link

```
SELECT * FROM item WHERE item_no = `$itemId`;
```

- Store item’s UserID in $ownerUserId

- Display Item Number, Item Title/name, Item-specific attributes, Item descriptions
- Get current logged in User using $UserID
- Check to see if the item belongs to another user, by comparing the Owner User's Id with the id of current logged in user:
    - o If item belongs to another user ($ownerUserId != $UserId):
        - ▪ Query the user table using $ownerUserId to gather the Owner user's nickname, city, state, postal code, and Rating.

```
SELECT FT1.USER2_EMAIL, FT1.USER2_NICKNAME,
FT1.USER2_LOCATION, FT1.USER2_DISTANCE,
FT2.USER2_RATING FROM (SELECT USER1_EMAIL,
USER2_EMAIL, USER2_NICKNAME, CONCAT(USER2_CITY, ", ",
USER2_STATE, " ", USER2_POSTAL_CODE) AS
USER2_LOCATION,
ROUND((6371 * 0.621371 * (2 *
ATAN((SQRT((POWER((SIN(((RADIANS(T2.USER2_LAT)) -
(RADIANS(T1.USER1_LAT))) / 2)), 2) +
(COS((RADIANS(T1.USER1_LAT)))) *
(COS((RADIANS(T2.USER2_LAT)))) *
POWER((SIN(((RADIANS(T2.USER2_LON)) -
(RADIANS(T1.USER1_LON))) / 2)), 2)))), (SQRT(1 -
(POWER((SIN(((RADIANS(T2.USER2_LAT)) -
(RADIANS(T1.USER1_LAT))) / 2)), 2) +
(COS((RADIANS(T1.USER1_LAT)))) *
(COS((RADIANS(T2.USER2_LAT)))) *
POWER((SIN(((RADIANS(T2.USER2_LON)) -
(RADIANS(T1.USER1_LON))) / 2)), 2))))))),2) AS
USER2_DISTANCE
FROM
    (SELECT U1.email AS USER1_EMAIL, U1.nickname AS
USER1_NICKNAME, U1.postal_code AS USER1_POSTAL_CODE,
P1.latitude AS USER1_LAT, P1.longitude USER1_LON, P1.City AS
USER1_CITY, P1.State AS USER1_STATE FROM `user` U1 JOIN
postalcode P1 ON U1.postal_code = P1.postal_code) T1
    JOIN
    (SELECT U2.email AS USER2_EMAIL, U2.nickname AS
USER2_NICKNAME, U2.postal_code AS USER2_POSTAL_CODE,
P2.latitude AS USER2_LAT, P2.longitude USER2_LON, P2.City AS
USER2_CITY, P2.state AS USER2_STATE FROM `user` U2 JOIN
postalcode P2 ON U2.postal_code = P2.postal_code) T2
    ON T1.USER1_EMAIL <> T2.USER2_email WHERE
USER1_EMAIL=`$userId` AND USER2_EMAIL=`$ownerUserId`)
FT1 JOIN
    (SELECT (((SELECT AVG(T1.swap_proposer_rating) FROM
(SELECT swap_proposer_rating FROM swap WHERE
swap_status='Accepted' AND proposer_email = `$ownerUserId` AND
swap_proposer_rating IS NOT NULL) T1 ) + (SELECT
AVG(T2.swap_counterparty_rating) FROM (SELECT
swap_counterparty_rating FROM swap WHERE
swap_status='Accepted' AND counterparty_email = `$ownerUserId`
AND swap_counterparty_rating IS NOT NULL) T2)) / 2) AS
USER2_RATING) FT2;
```

- 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.

```
SELECT postal_code FROM `user` WHERE email = `$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.

```
SELECT COUNT(*) FROM swap WHERE swap_status='Accepted'
AND ((proposer_email = `$userId` AND swap_proposer_rating IS
NULL) OR (counterparty_email = `$userId` AND
swap_counterparty_rating IS NULL));
```

- 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.

```
SELECT COUNT(*) FROM swap WHERE swap_status = 'Pending'
AND counterparty_email = `$userId`;
```

        - 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 $ItemIsAvailable and set it to TRUE
            - Query the swap table to get the count of all accepted or pending swaps where the ItemId ($itemId) is equal to the proposed Item's Id or desired Item's Id.

```
SELECT COUNT(*) FROM swap WHERE (swap_status IN
('Accepted', 'Pending')) AND (proposer_item_id = `$itemId` OR
desired_item_id = `$itemId`);
```

            - 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

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.

```
SELECT COUNT(*) FROM swap WHERE swap_status='Accepted' AND
((proposer_email = `$userId` AND swap_proposer_rating IS NULL) OR
(counterparty_email = `$userId` AND swap_counterparty_rating IS NULL));
```

- 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 ($itemId) to get the desired item's title and item's userId

```
SELECT email, title FROM `item` WHERE item_no = `$itemId`;
```

- Store items's userID in $desiredItemUserId
- Query to get the distance between the current user ($userId) and desired Item's User ($desiredItemUserId)

```
SELECT ROUND((6371 * 0.621371 * (2 *
ATAN((SQRT((POWER((SIN(((RADIANS(T2.USER2_LAT)) -
(RADIANS(T1.USER1_LAT))) / 2)), 2) +
(COS((RADIANS(T1.USER1_LAT)))) *
(COS((RADIANS(T2.USER2_LAT)))) *
POWER((SIN(((RADIANS(T2.USER2_LON)) -
(RADIANS(T1.USER1_LON))) / 2)), 2)))), (SQRT(1 -
(POWER((SIN(((RADIANS(T2.USER2_LAT)) -
(RADIANS(T1.USER1_LAT))) / 2)), 2) +
(COS((RADIANS(T1.USER1_LAT)))) *
(COS((RADIANS(T2.USER2_LAT)))) *
POWER((SIN(((RADIANS(T2.USER2_LON)) -
(RADIANS(T1.USER1_LON))) / 2)), 2))))))),2) AS USER_DISTANCE
FROM
    (SELECT U1.email AS USER1_EMAIL, U1.nickname AS
USER1_NICKNAME, U1.postal_code AS USER1_POSTAL_CODE,
P1.latitude AS USER1_LAT, P1.longitude USER1_LON, P1.City AS
USER1_CITY, P1.State AS USER1_STATE FROM `user` U1 JOIN
postalcode P1 ON U1.postal_code = P1.postal_code) T1
    JOIN
    (SELECT U2.email AS USER2_EMAIL, U2.nickname AS
USER2_NICKNAME, U2.postal_code AS USER2_POSTAL_CODE,
P2.latitude AS USER2_LAT, P2.longitude USER2_LON, P2.City AS
USER2_CITY, P2.state AS USER2_STATE FROM `user` U2 JOIN postalcode
P2 ON U2.postal_code = P2.postal_code) T2
    ON T1.USER1_EMAIL <> T2.USER2_email WHERE USER1_EMAIL =
`$userId` AND USER2_EMAIL = `$desiredItemUserId`;
```

- 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 $omitUserItemIds
- Query the items table to get all the items for current user ($userID)

```
SELECT item_no FROM `item` WHERE email = `$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)

```
SELECT COUNT(*) FROM swap WHERE (swap_status IN
('Accepted', 'Rejected')) AND ((proposer_email = `$userID` AND
counterparty_email = `$desiredItemUserId` AND proposer_item_id =
`$eachAvailableItemId` AND desired_item_id = `$itemId`) OR
(proposer_email = `$desiredItemUserId` AND counterparty_email =
`$userId` AND proposer_item_id = `$itemId` AND desired_item_id =
`$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

```
SELECT item_no, type, title, `condition` FROM `item` WHERE email =
`$userId` AND item_no NOT IN (`$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 ($desired_item_Id), proposed Item Id (Item Id from the selected radio button, $selectedItemId), proposer user ID (current user's ID/$userId), counterparty user ID($desiredItemUserId), and today's date as proposal date ($proposalDate).

```
INSERT INTO swap (`proposal_date`,  `proposer_email`,
`proposer_item_id`, `counterparty_email`, `desired_item_id`)
VALUES (`$proposalDate`, `$userId`, `$selectedItemId`,
`$desiredItemUserId`, `$desiredItemId`);
```

    - 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

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

```
SELECT COUNT(*) FROM swap WHERE counterparty_email = `$userId` AND
swap_status = 'Pending';
```

- Store the count of all unaccepted swaps for current user in $numPendingSwaps
  - If $numPendingSwaps > 0:

- Query the swap table to get all unaccepted Swaps for current user ($userId):

```
SELECT proposer_email, counterparty_email, desired_item_id,
proposer_item_id FROM swap WHERE swap_status = 'Pending' AND
counterparty_email = `$userId`;
```

- For each unaccepted swap for the current user ($userId, $desiredItemId, $proposerUserId, $proposedItemId), gather Date, DesiredItem, Proposer, Rating, Distance, and Proposed Item

```
SELECT * FROM
(SELECT swap.proposal_date, swap.desired_item_id, T2.CP_TITLE
AS DESIRED_ITEM, T2.P_EMAIL AS PROPOSER_EMAIL,
T2.P_NICKNAME AS PROPOSER, swap.proposer_item_id,
T2.P_TITLE AS PROPOSED_ITEM FROM swap JOIN (SELECT
T1.CP_TITLE, T1. P_TITLE, T1.P_EMAIL, T1.CP_EMAIL,
T1.CP_INO, T1.P_INO, U1.nickname AS P_NICKNAME FROM
(SELECT IT1.email AS CP_EMAIL, IT1.item_no AS CP_INO,
IT1.title AS CP_TITLE, IT2.email AS P_EMAIL, IT2.item_no AS
P_INO, IT2.title AS P_TITLE FROM item IT1 JOIN item IT2) T1
JOIN(SELECT email, nickname FROM `user`) U1 ON T1.P_EMAIL =
U1.email) T2 ON swap.proposer_email = T2.P_EMAIL AND
swap.counterparty_email = T2.CP_EMAIL WHERE
swap.counterparty_email = `$userId` AND swap.swap_status =
'Pending' AND swap.desired_item_id = T2.CP_INO AND
swap.proposer_item_id = T2.P_INO ) FT1
JOIN
(SELECT ROUND((6371 * 0.621371 * (2 *
ATAN((SQRT((POWER((SIN(((RADIANS(T2.USER2_LAT)) -
(RADIANS(T1.USER1_LAT))) / 2)), 2) +
(COS((RADIANS(T1.USER1_LAT)))) *
(COS((RADIANS(T2.USER2_LAT)))) *
POWER((SIN(((RADIANS(T2.USER2_LON)) -
(RADIANS(T1.USER1_LON))) / 2)), 2)))), (SQRT(1 -
(POWER((SIN(((RADIANS(T2.USER2_LAT)) -
(RADIANS(T1.USER1_LAT))) / 2)), 2) +
(COS((RADIANS(T1.USER1_LAT)))) *
(COS((RADIANS(T2.USER2_LAT)))) *
POWER((SIN(((RADIANS(T2.USER2_LON)) -
(RADIANS(T1.USER1_LON))) / 2)), 2)))))))),2) AS
DISTANCE_CALC_MILES
```

```
FROM
    (SELECT U1.email AS USER1_EMAIL, U1.nickname AS
USER1_NICKNAME, U1.postal_code AS USER1_POSTAL_CODE,
P1.latitude AS USER1_LAT, P1.longitude USER1_LON, P1.City AS
USER1_CITY, P1.State AS USER1_STATE FROM `user` U1 JOIN
postalcode P1 ON U1.postal_code = P1.postal_code) T1
    JOIN
    (SELECT U2.email AS USER2_EMAIL, U2.nickname AS
USER2_NICKNAME, U2.postal_code AS USER2_POSTAL_CODE,
P2.latitude AS USER2_LAT, P2.longitude USER2_LON, P2.City AS
USER2_CITY, P2.state AS USER2_STATE FROM `user` U2 JOIN
postalcode P2 ON U2.postal_code = P2.postal_code) T2
    ON T1.USER1_EMAIL <> T2.USER2_email WHERE
USER1_EMAIL = `$userId` AND USER2_EMAIL =
`$proposerUserId`) FT2
JOIN
(SELECT COALESCE((((SELECT AVG(swap_proposer_rating) AS
RATING_AVG FROM swap WHERE swap_status = 'Accepted' AND
proposer_email = `$proposerUserId` AND swap_proposer_rating IS
NOT NULL) + (SELECT AVG(swap_counterparty_rating) AS
RATING_AVG FROM swap WHERE swap_status = 'Accepted' AND
counterparty_email = `$proposerUserId` AND
swap_counterparty_rating IS NOT NULL)) /2), ((SELECT
AVG(swap_proposer_rating) AS RATING_AVG FROM swap
WHERE swap_status = 'Accepted' AND proposer_email =
`$proposerUserId` AND swap_proposer_rating IS NOT NULL)),
((SELECT AVG(swap_counterparty_rating) AS RATING_AVG
FROM swap WHERE swap_status = 'Accepted' AND
counterparty_email = `$proposerUserId` AND
swap_counterparty_rating IS NOT NULL)), 'None') AS
USER_RATING_AVG) FT3 WHERE FT1.desired_item_id =
`$desiredItemId` AND FT1.proposer_item_id =`$proposedItemId`;
```

        - Display Date, DesiredItem, Proposer, Rating, Distance, and Proposed Item for each unaccepted swaps along with an ***Accept*** and ***Reject*** Button
    - If $numPendingSwaps == 0:
        - **View Main Menu**
- While no button is pressed, do nothing.
- When a button is pushed, then do the following:
    - Click ***Accept*** button:
        - Update Swap table using the proposed item id ($proposedItemId) and counterparty item id ($desiredItemId) of the Swap for which the ***Accept***

button was clicked.

```
UPDATE swap SET accepted_rejected_date =
CURRENT_DATE, swap_status = ‘Accepted’ WHERE
desired_item_id = `$desiredItemId` AND proposer_item_id =
`$proposed_item_id`;
```

- 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 table using the proposed item id ($proposedItemId) and counterparty item id ($desiredItemId) of the Swap for which the ***Reject*** button was clicked.

```
UPDATE swap SET accepted_rejected_date =
CURRENT_DATE, swap_status = ‘Rejected’ WHERE
desired_item_id = `$desiredItemId` AND proposer_item_id =
`$proposed_item_id`;
```

  - Display **Accept/Reject Swaps Form**
- Click ***DesiredItem*** Link:
  - Jump to **Item Details Form**
- Click ***ProposedItem*** Link:
  - Jump to **Item Details Form**

Rate Swaps

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** from **Main Menu** "Unrated Swaps" link.
        - **Rate Swaps** lookup subtask will be triggered upon visiting this form.
        - Using $UserId, find all unrated swaps in Swap table that current User is a part of.

```
SELECT swap.counterparty_email, swap.proposer_email, swap.desired_item_id,
swap.proposer_item_id, swap.accepted_rejected_date
FROM ` swap `
INNER JOIN `item` ON item.item_no=swap.proposer_item_id OR
item.item_no=swap.desired_item_id
INNER JOIN `user` ON `user`.email=swap.counterparty_email OR
`user`.email=swap.proposer_email
WHERE swap.swap_status='Accepted'
AND ((swap.proposer_email = `$userId` AND swap.swap_proposer_rating IS NULL) OR
(swap.counterparty_email = `$userId` AND swap.swap_counterparty_rating IS NULL))
AND (`user`.email=`$userId`)
AND (item.email=`$userId`);
```

- Display Swap AcceptedRejectedDate
    - Display $acceptedRejectedDate
- Display My Role
    - For My Role, if $userID is in $proposerEmail, then set My Role to Proposer
    - Else, set My Role to CounterParty.
- Display Proposed Item Name
    - Using the $proposedItem, we can query the Item table to get the proposed item's name

```
SELECT item.title from item WHERE item.item_no=`$proposedItem`;
```

- Display Desired Item Name
    - Using the $desiredItem, we can query the **Item** table to get the counter party item's name

```
SELECT item.title from item WHERE item.item_no=`$desiredItem`;
```

- Display Other User nickname
    - Query User table for $otherUser

```
SELECT `user`.nickname from `user` WHERE `user`.email=`$otherUser`;
```

- 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
    - $myRole is Proposer

```
UPDATE `swap` SET `swap_proposer_rating`= `$rating`
WHERE swap.counterparty_email=`$counterpartyEmail`
AND swap.proposer_email=`$proposerEmail`
AND swap.desired_item_id=`$desiredItem`
AND swap.proposer_item_id=`$proposedItem`;
```

- $myRole is CounterParty

```
UPDATE `swap` SET `swap_counterparty_rating`= `$rating`
WHERE swap.counterparty_email=`$counterpartyEmail`
AND swap.proposer_email=`$proposerEmail`
AND swap.desired_item_id=`$desiredItem`
AND swap.proposer_item_id=`$proposedItem`;
```

View Swap History

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**.**

```
(SELECT 'Proposer' AS MY_ROLE, (PA.P_ACCEPTED +
PR.P_REJECTED) AS TOTAL, PA.P_ACCEPTED AS ACCEPTED,
PR.P_REJECTED AS REJECTED,
((PR.P_REJECTED/(PA.P_ACCEPTED + PR.P_REJECTED)) * 100) AS
`Rejected PCNT` FROM

   (SELECT COUNT(*) AS P_ACCEPTED FROM swap WHERE
proposer_email = `$userId` AND swap_status = 'Accepted') PA JOIN

   (SELECT COUNT(*) AS P_REJECTED FROM swap WHERE
proposer_email = `$userId` AND swap_status = 'Rejected') PR)

UNION

(SELECT 'CounterParty' AS MY_ROLE, (CPA.CP_ACCEPTED +
CPR.CP_REJECTED) AS TOTAL, CPA.CP_ACCEPTED AS
ACCEPTED, CPR.CP_REJECTED AS REJECTED,
((CPR.CP_REJECTED/(CPA.CP_ACCEPTED + CPR.CP_REJECTED)) *
100) AS `Rejected PCNT` FROM

(SELECT COUNT(*) AS CP_ACCEPTED FROM swap WHERE
counterparty_email = `$userId` AND swap_status = 'Accepted') CPA JOIN

   (SELECT COUNT(*) AS CP_REJECTED FROM swap WHERE
counterparty_email = `$userId` AND swap_status = 'Rejected') CPR);
```

- 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.

```
SELECT swap.proposal_date, swap.accepted_rejected_date,
swap.swap_status, 'Proposer' AS MY_ROLE, swap.proposer_item_id,
T2.P_TITLE AS PROPOSED_ITEM, swap.desired_item_id,
T2.CP_TITLE AS DESIRED_ITEM, T2.CP_NICKNAME AS
OTHER_USER, swap.swap_proposer_rating AS RATING FROM `swap`
JOIN
(SELECT T1.CP_TITLE, T1. P_TITLE, T1.P_EMAIL, T1.CP_EMAIL,
T1.CP_INO, T1.P_INO, U1.nickname AS CP_NICKNAME FROM
(SELECT IT1.email AS CP_EMAIL, IT1.item_no AS CP_INO, IT1.title
AS CP_TITLE, IT2.email AS P_EMAIL, IT2.item_no AS P_INO, IT2.title
AS P_TITLE FROM item IT1 JOIN item IT2) T1 JOIN
(SELECT email, nickname FROM `user`) U1 ON T1.CP_EMAIL =
U1.email) T2 ON swap.proposer_email = T2.P_EMAIL AND
swap.counterparty_email = T2.CP_EMAIL WHERE (swap.proposer_email
= `$userId`) AND swap.swap_status IN ('Accepted', 'Rejected') AND
swap.desired_item_id = T2.CP_INO AND swap.proposer_item_id =
T2.P_INO
UNION
SELECT swap.proposal_date, swap.accepted_rejected_date,
swap.swap_status, 'CounterProposer' AS MY_ROLE,
swap.proposer_item_id, T2.P_TITLE AS PROPOSED_ITEM,
swap.desired_item_id, T2.CP_TITLE AS DESIRED_ITEM,
T2.P_NICKNAME AS OTHER_USER, swap.swap_proposer_rating AS
RATING FROM `swap` JOIN
(SELECT T1.CP_TITLE, T1. P_TITLE, T1.P_EMAIL, T1.CP_EMAIL,
T1.CP_INO, T1.P_INO, U1.nickname AS P_NICKNAME FROM
(SELECT IT1.email AS CP_EMAIL, IT1.item_no AS CP_INO, IT1.title
AS CP_TITLE, IT2.email AS P_EMAIL, IT2.item_no AS P_INO, IT2.title
AS P_TITLE FROM item IT1 JOIN item IT2) T1 JOIN
(SELECT email, nickname FROM `user`) U1 ON T1.P_EMAIL =
U1.email) T2 ON swap.proposer_email = T2.P_EMAIL AND
swap.counterparty_email = T2.CP_EMAIL WHERE
(swap.counterparty_email = `$userId`) AND swap.swap_status IN
('Accepted', 'Rejected') AND swap.desired_item_id = T2.CP_INO AND
swap.proposer_item_id = T2.P_INO;
```

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

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 that matches current Swap selected in GUI. Display Swap ProposedDate, AcceptedRejectedDate, Status

```
SELECT swap.proposal_date, swap.accepted_rejected_date, swap.swap_status  FROM
swap WHERE swap.desired_item_id=`$desiredItemId` AND
swap.proposer_item_id=`$proposerItemId`;
```

  - Display My Role
    - Display $myRole
  - Display Rating Left
    - If Proposer:

```
SELECT swap.swap_counterparty_rating  FROM swap WHERE
swap.desired_item_id=`$desiredItemId` AND
swap.proposer_item_id=`$proposerItemId`;
```

    - If CounterParty:

```
SELECT swap.swap_proposer_rating  FROM swap WHERE
swap.desired_item_id=`$desiredItemId` AND
swap.proposer_item_id=`$proposerItemId`;
```

  - Use swap information to query Item table for proposer and counter party items. Display Item ItemID, Title, GameType, Condition, and Description for each item.
  - For Proposed Item

```
SELECT item.item_no,item.title,item.TYPE,item.`condition`,item.description FROM
item WHERE item.item_no=`$proposerItemId`;
```

  - For Desired Item

```
SELECT item.item_no,item.title,item.TYPE,item.`condition`,item.description FROM
item WHERE item.item_no=`$desiredItemId`;
```

  - Query the User table to get information on other user

- Display other User Nickname
- If current user is Proposer:

```
SELECT `user`.nickname FROM `user` INNER JOIN swap ON
swap.counterparty_email= `user`.email WHERE swap.proposer_email = `$userId` AND
swap.desired_item_id=`$desiredItemId` AND swap.proposer_item_id=`
$proposerItemId`;
```

- If current user is CounterParty:

```
SELECT `user`.nickname FROM `user` INNER JOIN swap ON swap.proposer_email=
`user`.email WHERE swap.counterparty_email = `$userId` AND
swap.desired_item_id=`$desiredItemId` AND
swap.proposer_item_id=`$proposerItemId`;
```

- Display swap distance
- First, get user emails

```
SELECT counterparty_email, proposer_email FROM swap WHERE
swap.desired_item_id=`$desiredItemId` AND
swap.proposer_item_id=`$proposerItemId`;
```

- Then calculate distance between users

```
SELECT
 ROUND((6371 * 0.621371 * (2 *
ATAN((SQRT((POWER((SIN(((RADIANS(T2.USER2_LAT)) -
(RADIANS(T1.USER1_LAT))) / 2)), 2) +  (COS((RADIANS(T1.USER1_LAT)))) *
(COS((RADIANS(T2.USER2_LAT)))) * POWER((SIN(((RADIANS(T2.USER2_LON))
- (RADIANS(T1.USER1_LON))) / 2)), 2)))), (SQRT(1 -
(POWER((SIN(((RADIANS(T2.USER2_LAT)) - (RADIANS(T1.USER1_LAT))) / 2)),
2) +  (COS((RADIANS(T1.USER1_LAT)))) * (COS((RADIANS(T2.USER2_LAT)))) *
POWER((SIN(((RADIANS(T2.USER2_LON)) - (RADIANS(T1.USER1_LON))) / 2)),
2))))))),2) AS DISTANCE_CALC_MILES
 FROM
    (SELECT U1.email AS USER1_EMAIL, U1.nickname AS USER1_NICKNAME,
U1.postal_code AS USER1_POSTAL_CODE, P1.latitude AS USER1_LAT,
P1.longitude USER1_LON, P1.City AS USER1_CITY, P1.State AS USER1_STATE
FROM `user` U1 JOIN postalcode P1 ON U1.postal_code = P1.postal_code) T1
    JOIN
    (SELECT U2.email AS USER2_EMAIL, U2.nickname AS USER2_NICKNAME,
U2.postal_code AS USER2_POSTAL_CODE, P2.latitude AS USER2_LAT,
P2.longitude USER2_LON, P2.City AS USER2_CITY, P2.state AS USER2_STATE
FROM `user` U2 JOIN postalcode P2 ON U2.postal_code = P2.postal_code) T2
    ON T1.USER1_EMAIL <> T2.USER2_email WHERE
USER1_EMAIL=`$proposerEmail` AND USER2_EMAIL=`$counterPartyEmail`;
```

- If $status is **Accepted**
  - Display other **User** FirstName, Email

```
SELECT `user`.first_name, `user`.email FROM `user` WHERE
`user`.email=`$otherUserId`;
```

  - If User PhoneNumber Shareable==**True**
    - Display User PhoneNumber Number and NumberType

```
SELECT phonenumber.number, phonenumber.number_type FROM `phonenumber`
WHERE phonenumber.email=`$otherUserId` AND share_phone_number=1;
```

Update User Information

Abstract Code

- User can update User's Information by clicking ***Update My Info*** on **Main Menu Form**.
- If user has unapproved swap or unrated swaps
  - Query to get a count of all unapproved for the current logged in user ($userId)

```
SELECT COUNT(*) FROM swap WHERE swap_status = 'Pending' AND
counterparty_email = `$userId`;
```

  - If the count > 0:
    - Display error message "Update not allowed due to unapproved swaps on **Main Menu Form** while clicking on ***Update My Info.***

  - Query to get a count of all unrated swaps for the current logged in user ($userId)

```
SELECT COUNT(*) FROM swap WHERE swap_status='Accepted' AND
((proposer_email = `$userId` AND swap_proposer_rating IS NULL) OR
(counterparty_email = `$userId` AND swap_counterparty_rating IS NULL));
```

  - If the count > 0:

        - Display error message "Update not allowed due to unrated swaps" on **Main Menu Form** while clicking on ***Update My Info.***

- Upon:
    - For non-postalcode and non-phone number update. Update *password* ('$Password'), *first name* ($ newFirstName), *last name* ($newLastName), *nickname* ($newNickName) in input field

    ```
    UPDATE `user` SET `password` = '$newPwd', first_name = `$newFirstName`,
    last_name = `$newLastName`, nickname = `$newNickName` WHERE email =
    `$userId`;
    ```

    - If user tries to update phone number on **Update My Information Form** using by another user:
        - Query the phone number table to see if the number input by the user($userInputNumber) is in use by someone else.

        ```
        SELECT COUNT(*) FROM phonenumber WHERE number =
        `$userInputNumber` AND email <> `$userId`;
        ```

        - If the count is 0, that means the phone number is already in use by some other user in the system.
            - Display error message "Phone Number is in use".
        - Else:
            - Update the phone number ($userInputNumber). Number type ($numberType) and Shareable flag for the number($numberShareable) for the logged in user($userId)

            ```
            UPDATE phonenumber SET `number` = `$userInputNumber`,
            number_type = `$numberType`, share_phone_number =
            `$numberShareable` WHERE email = `$userId`;
            ```

    - If user tries to update postal code ($newPostalCode) on **Update My Information Form:**

        - Verify Postal Code, if none is found, display error message "Postal Code not exist"

        ```
        SELECT COUNT(*) FROM `postalcode` WHERE
        postal_code=`$newPostalCode`;
        ```

- Only if postal code is found. Update PostalCode is allowed in User table

```
UPDATE `user` SET postal_code = `$newPostalCode` WHERE email
= `$userId`;
```