Phase 2 Abstract Code w/SQL

CS 6400 - Spring 2022

Team 077

Table of Contents

Login

Abstract Code

```
- Render *Email/phone number* text input field,
    - When value is changed, set variable $Email to current value
- Render *Password* text input field
    - When value is changed, set variable $Password to current value
- Register ***Login*** button
- Render ***Register*** button
- When ***Register*** button is clicked, go to **Register** task
- When ***Login*** button is clicked:
    - IF USER record with identifier matching '$Email is found
        - IF found USER's password is equal to '$Password'
```

```
SELECT Email, Password
FROM User_
WHERE Email = '$Email' AND Password = '$Password';
```

```
            - Login USER, go to the **View Main Menu** task
        - ELSE (USER's *password* != "$Password')
            - Render error message
    - ELSE: email and password not registered in the database or invalid inputs
        - Render error message
```

Register

Abstract Code

- User enters required text input fields *'$Email'*, *'$Password'*, *'$First_name'*, *'$Last_name'*, *'$Nick_name'* and optional fields *'$Phone_number', '$Phone_type', '$Show_phone_in_swaps'* in registration form
- User selects postal code and the system fills in a matching '$City' and '$State' in the user profile from the ADDRESS entity

```
SELECT post_code, City, State
FROM Address;
```

- When current user hits ***Register*** button
    - Try to insert new USER with values from text input fields

```
INSERT INTO User_
VALUES ('$Email', '$First_name', '$Last_name', '$Nick_name', '$Password',
'$Post_code');

INSERT INTO Phone
VALUES( '$Number', '$Type', '$Shareable', '$Owner');
```

- If successful
    - Go to **View Main Menu** task with "welcome" message on top of the menu that includes the user's first and last name
- Else
    - Show error message from inserting user, e.g. "E-mail already in use" and/or "Phone number already in use"

View Main Menu

Abstract Code

- Subtask Get User:
    - find one USER where user email matches the user email from the page URL / HTTP request / session
    - Save result to variable $user

```
SELECT First_name, Last_name
FROM User_
WHERE Email = '$Email';
```

- Subtask Get User's Swaps:
    - find all ITEMs where user is owner and all SWAPs where user's ITEMs are involved
    - Save result to variable $users_swaps
- Render **<u>Main Menu</u>**
    - Render Welcome Header: "Welcome {'$user.firstname'} {'$user.lastname'}
    - Render My Rating Panel:
        - IF count of $users_swaps is less than one, replaced calculated rating with string "None"
        - ELSE calculated average rating by accumulating all of the ratings of the user from the user's completed swaps and dividing by the number of completed swaps and save to variable $rating. Display "My Rating: { $rating }

```
SELECT AVG(rating) AS My_Rating FROM (
  SELECT Item.ID, Proposer_rating AS rating FROM (
    Accepted_swap
    INNER JOIN Item
    ON Item.ID = Accepted_swap.Proposed_IID AND "$Email" = Item.Owner
  )
  UNION
  SELECT Item.ID, Counterparty_rating AS rating FROM (
    Accepted_swap
    INNER JOIN Item
    ON Item.ID = Accepted_swap.Desired_IID AND "$Email" = Item.Owner
  )
) AS ratings
WHERE ratings.rating IS NOT NULL;
```

- Render Unaccepted Swaps Panel: return “Unaccepted Swaps: {count of swaps where user is counterparty and ‘$swap.status’ is not accepted}
    - **IF** count of unaccepted swaps is greater than 0, wrap count of unaccepted swaps in hyperlink with URL to **Accept/Reject Swaps** form
    - **IF** the count of unaccepted swaps where current date minus ‘$proposal_date’ is greater than 5 days is greater than 0, OR count of unaccepted swaps is greater than 5, wrap hyperlink in styling bold and red

```
SELECT COUNT(*) AS Unaccepted_swaps FROM (
  Pending_swap INNER JOIN Item
  ON
  Pending_swap.Desired_IID = Item.ID AND
  Item.Owner = "$Email"
);
```

- Render Unrated Swaps Panel: return “Unrated Swaps: {count of swaps where (user is ‘$counterparty’ and ‘$counterparty_rating’ is null OR user is ‘$proposer’ and ‘$proposer_rating is null’}”
    - IF count of unrated swaps is greater than zero, wrap count of unrated swaps in hyperlink with URL to **Rate Swaps** task
    - IF count of unrated swaps is greater than zero, wrap hyperlink in styling bold and red

```
SELECT COUNT(*) AS Unrated_Swap FROM (
  SELECT * FROM (
    Accepted_swap
    INNER JOIN Item
    ON Item.ID = Accepted_swap.Desired_IID
    AND "$Email" = Item.Owner
    AND Accepted_swap.Counterparty_rating IS NULL
  )
  UNION
  SELECT * FROM (
    Accepted_swap
    INNER JOIN Item
    ON Item.ID = Accepted_swap.Proposed_IID
    AND "$Email" = Item.Owner
    AND Accepted_swap.Proposer_rating IS NULL
   )
 ) as tbl;
```

- Render ***List Items*** button: display hyperlink to **List Item** task page
- Render ***My Items*** button: display hyperlink to **View Item Details** task page
- Render ***Search Items*** button: display hyperlink to **Search Items** task page
- Render ***Swap History*** button: display hyperlink to **View Swap History** task page
- Render ***Update My Info*** button: display hyperlink to **Update My Info** task page
- Render ***Logout*** button: on click, invalidate user session, redirect to **Login** task page

Update User Info

Abstract Code:

- Subtask: Check and render unapproved or unrated swaps exception
    - find SWAPs where the current user is $counterparty or $proposer and $swap_status is not accepted OR where $status is accepted and the user has not rated the other participant. Save result to variable $unapproved_or_unrated_swaps

```
//1)  Unapproved swaps as proposer
```

```
SELECT count(*) AS Unapproved_proposed_swaps
FROM Pending_swap
INNER JOIN Item
ON Pending_swap.Proposed_IID = Item.ID
WHERE Owner = '$Email';
```

```
//2)  unapproved swaps as counterparty
```

```
SELECT count(*) AS Unapproved_received_swaps
FROM Pending_swap
INNER JOIN Item
ON Pending_swap.Desired_IID  = Item.ID
WHERE Owner = '$Email';
```

```
//3) unrated swaps as a proposer
```

```
SELECT count(*) AS Unrated_proposed_swaps
FROM Accepted_swap
INNER JOIN Item
ON Accepted_swap.Proposed_IID = Item.ID
WHERE Owner = '$Email' AND Proposer_rating is NULL;
```

```
//4) unrated swaps as a counterparty
SELECT count(*) AS Unrated_received_swap
FROM Accepted_swap
INNER JOIN Item
ON Accepted_swap.Desired_IID  = Item.ID
```

```sql
WHERE Owner = '$Email' AND Counterparty_rating is NULL;
```

- IF length of '$unapproved_or_unrated_swaps' is greater than 0:
    - display "You have {length of '$unapproved_or_unrated_swaps'} unapproved or unrated swaps. Please resolve pending swaps before updating user information."
- ELSE continue to Get User Data subtask and render **<u>Update User Information</u>** form

- Subtask: Get User Data
    - Get data for current user and save to variable '$user_data'
    - Get address for current user and save to variable '$user_address'

```sql
SELECT Email, First_name, Last_name, User_.Post_code, City, State, Number_,
Type_, Shareable
FROM User_
LEFT JOIN Address
ON User_.Post_code = Address.Post_code
LEFT JOIN Phone
ON Phone.Owner = '$Email'
WHERE Email = '$Email';
```

- Render **<u>Update User Information</u>** form
    - Define form template **<u>Update User Information</u>**:
        - Email, Text Input:
            - Label is "Email"
            - Disabled is True
            - Starting value is $user_data.email
        - *Nickname*, Text Input:
            - Label is "Nickname"
            - Starting value is $user_data.nickname
        - *Password*, Text Input:
            - Label is "Password"
            - Display is "********"
            - Starting value is null
        - *City*, Text Input:
            - Label is "City"
            - Starting value is $user_address.city
        - *First Name*, Text Input:
            - Label is "First Name"
            - Starting value is $user_data.first_name
        - *State*, Text Input:

                - Label is “State”
                - Starting value is $user_address.state
            - *Last Name*, Text Input:
                - Label is “Last Name”
                - Starting value is $user_data.last_name
            - *Postal Code*, Text Input:
                - Label is “Postal Code”
                - Starting value is $user_address.postal_code
            - *Phone Number*, Text Input**:**
                - Label is “Phone number (optional)
                - Starting value is $.phone_number else
            - *Type*, Dropdown List:
                - Label is “Type”
                - Starting value is $user_data.phone_type
            - *Shown Phone Number*, Check Box:
                - Starting value is $user_data.phone_shareable
            - ***Update*** button: Text is “Update”, IF clicked, execute Update User subtask
        - Render **Update User Information** form with values from $user_data and $user_address
- Subtask Update User:
    - Define variable $update_user_payload
    - For each field in **Update User Information** form where value is not null and value is not equal to $user_data.{value_name} append field name and value as key, value to $unapproved_or_unrated_swaps
    - IF $update_user_payload is not empty, update current user’s USER entity with $update_user_payload
        - IF error on updating, display friendly error message:
            - Format and display error to user, e.g. **“**The phone number you provided {update_user_form.phone_number} is already in use. Please use another phone number.”
        - ELSE return to **View Main Menu** task

```
UPDATE User_
SET
Password = '$Password', First_name='$First_name', Last_name='$Last_name',
Nick_name='$Nick_name', Phone_number='$Phone_number',
Phone_type='$Phone_type', Phone_shareable='$show_phone_in_swaps',
Post_code='$Post_code'
WHERE Email = '$Email';
```

List New Item

Abstract Code:

- Show *Game type* drop down list, *Title* text input field, *Condition* text input field and *Description* text input field. The contents of each field will be accessible via a variable name, such as $game_type.
- If $game_type is selected as “Jigsaw Puzzle”, show *Piece count* field
- If $game_type is selected as “Video Game”, show *Platform* and *Media* fields
- If $game_type is selected as “Computer Game”, show *Platform* field
- If user clicks ***List Item:***
    - Verify that the user has less than 3 unrated swaps, and less than 6 unaccepted swaps. Prompt error if the verification does not pass

```
//unrated_proposed_item
```

```
SELECT Count(*) AS unrated_proposed_item
FROM   Accepted_swap
       LEFT JOIN Item I1
              ON Accepted_swap.Proposed_IID = I1.ID
       LEFT JOIN Item I2
              ON Accepted_swap.Desired_IID = I2.ID
WHERE  ( I1.Owner = '$Email'
         AND Proposer_rating IS NULL )
        OR ( I2.Owner = '$Email'
             AND Counterparty_rating IS NULL );
```

```
//number_of_unaccepted_swap
```

```
SELECT Count(*) AS number_of_unaccepted_swap
FROM   Pending_swap
       LEFT JOIN Item I1
              ON Accepted_swap.Proposed_IID = I1.ID
       LEFT JOIN Item I2
              ON Accepted_swap.Desired_IID = I2.ID
WHERE  I1.Owner = '$Email' OR I2.Owner = '$Email';
```

- Try to insert data from the form into ITEM table (there are)
    - IF success:
        - A success message will be displayed to the user with the item number (generated on insert)

`#case CardGame`

```
START TRANSACTION;

INSERT INTO Item (Title, Condition_, Description, Owner)
 VALUES ('$Title' , '$Condition_', '$Description', '$Email');

SET @last_id_in_table = LAST_INSERT_ID();
SELECT LAST_INSERT_ID();

INSERT INTO CardGame (ID)VALUES (@last_id_in_table);

COMMIT;
```

`#case BoardGame`

```
START TRANSACTION;

INSERT INTO Item (Title, Condition_, Description, Owner)
 VALUES ('$Title' , '$Condition_', '$Description', '$Email');

SET @last_id_in_table = LAST_INSERT_ID();
SELECT LAST_INSERT_ID();

INSERT INTO BoardGame (ID)VALUES (@last_id_in_table);

COMMIT;
```

`#case Jigsaw`

```
START TRANSACTION;

INSERT INTO Item (Title, Condition_, Description, Owner)
 VALUES ('$Title' , '$Condition_', '$Description', '$Email');

SET @last_id_in_table = LAST_INSERT_ID();
```

```
SELECT LAST_INSERT_ID();

INSERT INTO Jigsaw (ID, PieceCount) VALUES (@last_id_in_table,
$PieceCount);

COMMIT;
```

```
#case ComputerGame
```

```
START TRANSACTION;

INSERT INTO Item (Title, Condition_, Description, Owner)
 VALUES ('$Title' , '$Condition_', '$Description', '$Email');

SET @last_id_in_table = LAST_INSERT_ID();
SELECT LAST_INSERT_ID();

INSERT INTO ComputerGame (ID,  Platform) VALUES (@last_id_in_table,
'$Platform');

COMMIT;
```

```
START TRANSACTION;

INSERT INTO Item (Title, Condition_, Description, Owner)
 VALUES ('$Title' , '$Condition_', '$Description', '$Email');

SET @last_id_in_table = LAST_INSERT_ID();
SELECT LAST_INSERT_ID();

INSERT INTO VideoGame (ID, Platform, Media) VALUES (@last_id_in_table,
'$Platform', '$Media');

COMMIT;
```

Search Item

Abstract Code

- Render **Search for Item** form with appropriate input fields for *By Keyword, In My Postal Code, Within X Miles,* and *In Postal Code* searches. The contents of each text input field will be accessible via a variable such as '$keyword' for the *By Keyword* field.
- Do nothing until user clicks ***Search***
- When the user clicks ***Search***:
    - If the user filled in the *By Keyword* text field:
        - Find all swap-eligible items matching '$keyword' with item's 'item.name' or strings in 'item.description'

```
SELECT ID,
       CASE
         WHEN ID IN (SELECT ID
                     FROM   BoardGame) THEN "BoardGame"
         WHEN ID IN (SELECT ID
                     FROM   VideoGame) THEN "VideoGame"
         WHEN ID IN (SELECT ID
                     FROM   ComputerGame) THEN "ComputerGame"
         WHEN ID IN (SELECT ID
                     FROM   Jigsaw) THEN "Jigsaw"
         WHEN ID IN (SELECT ID
                     FROM   CardGame) THEN "CardGame"
       END AS Game_type,
       Title,
       Condition_,
       Description,
       distance
FROM   (SELECT ID,
               Title,
               Condition_,
               Description,
               i.Post_code AS pc1,
               u.Post_Code AS pc2
        FROM   (SELECT ID,
                       Title,
                       Condition_,
                       Description,
                       Owner,
                       Email,
                       Post_code
                FROM   Item
                       INNER JOIN User_
```

```
                              ON Item.Owner = User_.Email
               WHERE  ( Title LIKE '%$keyword%'
                         OR Description LIKE '%$keyword%' )
                      AND ID NOT IN (SELECT Desired_IID
                                     FROM   Pending_swap t1
                                     UNION
                                     SELECT Proposed_IID
                                     FROM   Pending_swap t2
                                     UNION
                                     SELECT Desired_IID
                                     FROM   Accepted_swap t3
                                     UNION
                                     SELECT Proposed_IID
                                     FROM   Accepted_swap t4
                                     UNION
                                     SELECT DISTINCT Desired_IID
                                     FROM   Rejected_swap
                                            JOIN Item
                                              ON Item.owner =
                                                 '$Email'
                                                 AND Proposed_IID =
Item.ID
                                     UNION
                                     SELECT DISTINCT Proposed_IID
                                     FROM   Rejected_swap
                                            JOIN Item
                                              ON Item.owner =
                                                 '$Email'
                                                 AND Desired_IID =
Item.ID)) AS
               i,
               (SELECT Post_code
                FROM   User_
                WHERE  User_.Email = '$Email') u) it
       LEFT JOIN (SELECT
                        a.post_code_1
                        AS
                        post_code_1,
                 a.post_code_2
                        AS
                        post_code_2
                        ,
                 2 * 3958.75 * atan2(Sqrt(pow(Sin(( a.lat1 - a.lat2 ) / 2),
2) + Cos(a.lat1) * Cos(a.lat2)
                 * pow(Sin( (a.long1 - a.long2 ) / 2), 2)),Sqrt(1 -
pow(Sin( (a.lat1 - a.lat2 ) /2),2) +
                 pow(Cos(a.lat2), 2) * pow(Sin((a.long1 - a.long2 ) / 2),
2)))
                 AS
                 distance
                  FROM   (SELECT a1.Latitude * Pi() / 180  AS lat1,
                                 a1.Longitude * Pi() / 180 AS long1,
                                 a1.Post_code              AS post_code_1,
                                 a2.Latitude * Pi() / 180  AS lat2,
                                 a2.Longitude * Pi() / 180 AS long2,
                                 a2.Post_code              AS post_code_2
                          FROM   ( Address AS a1
                                   CROSS JOIN Address AS a2 )) AS a) dis
              ON dis.post_code_1 = it.pc1
                 AND dis.post_code_2 = it.pc2
ORDER  BY distance,
          ID;
```

- For each item:
    - Find the user associated with the item and the location associated with the user
- If the result is empty:
    - display "Sorry, no results found". Return the user to the **Search for Item** form.
- Sort the list of items by distance, then item id, ascending.
- For each item, display item ID, Game Type, Title, Condition, Description, Distance and a link called ***Details***, with a blue highlight around the column that contains a matching key word
    - If 'item.description' has more than 100 characters, display (...) at the end of first 100 characters

- Else If the user checked the ***In My Zip Code*** button:
    - Find the user's location
    - Find all swap-eligible items with a user with the same zip code

```
SELECT ID,
       CASE
         WHEN ID IN (SELECT ID
                     FROM   BoardGame) THEN "BoardGame"
         WHEN ID IN (SELECT ID
                     FROM   VideoGame) THEN "VideoGame"
         WHEN ID IN (SELECT ID
                    FROM   ComputerGame) THEN "ComputerGame"
          WHEN ID IN (SELECT ID
                    FROM   Jigsaw) THEN "Jigsaw"
          WHEN ID IN (SELECT ID
                    FROM   CardGame) THEN "CardGame"
        END AS Game_type,
        Title,
        Condition_,
        Description,
        distance
FROM   (SELECT ID,
               Title,
               Condition_,
               Description,
               i.Post_code AS pc1,
               u.Post_Code AS pc2
         FROM   (SELECT ID,
                       Title,
                       Condition_,
                       Description,
                       Owner,
                       Email,
                       Post_code
                 FROM   Item
                       INNER JOIN User_
                               ON Item.Owner = User_.Email
                 WHERE  ( User_.post_code IN
                         (SELECT Post_code
                          FROM   User_
                          WHERE  Email = '$Email')
                         AND User_.Email != '$Email' )
                       AND ID NOT IN (SELECT Desired_IID
                                      FROM   Pending_swap t1
                                      UNION
                                      SELECT Proposed_IID
                                      FROM   Pending_swap t2
                                      UNION
                                      SELECT Desired_IID
                                      FROM   Accepted_swap t3
                                      UNION
                                      SELECT Proposed_IID
                                      FROM   Accepted_swap t4
                                      UNION
                                      SELECT DISTINCT Desired_IID
                                      FROM   Rejected_swap
                                             JOIN Item
                                               ON Item.owner =
                                                  '$Email'
                                                  AND Proposed_IID =
Item.ID
                                      UNION
                                      SELECT DISTINCT Proposed_IID
                                      FROM   Rejected_swap
                                             JOIN Item
                                               ON Item.owner =
                                                  '$Email'
                                                  AND Desired_IID =
Item.ID)) AS
               i,
               (SELECT Post_code
                FROM   User_
                WHERE  User_.Email = '$Email') u) it
       LEFT JOIN (SELECT
                        a.post_code_1
                        AS
                        post_code_1,
                 a.post_code_2
                        AS
                        post_code_2
                        ,
                 2 * 3958.75 * atan2(Sqrt(pow(Sin(( a.lat1 - a.lat2 ) / 2),
2) + Cos(a.lat1) * Cos(a.lat2)
                 * pow(Sin( (a.long1 - a.long2 ) / 2), 2)),Sqrt(1 -
pow(Sin( (a.lat1 - a.lat2 ) /2),2) +
                 pow(Cos(a.lat2), 2) * pow(Sin((a.long1 - a.long2 ) / 2),
2)))
 AS
                 distance
                  FROM   (SELECT a1.Latitude * Pi() / 180  AS lat1,
                                 a1.Longitude * Pi() / 180 AS long1,
                                 a1.Post_code              AS post_code_1,
                                 a2.Latitude * Pi() / 180  AS lat2,
                                 a2.Longitude * Pi() / 180 AS long2,
                                 a2.Post_code              AS post_code_2
                         FROM   ( Address AS a1
                                  CROSS JOIN Address AS a2 )) AS a) dis
```

```
              ON dis.post_code_1 = it.pc1
                 AND dis.post_code_2 = it.pc2
ORDER  BY distance,
          ID;
```

- If the result is empty,
    - display “Sorry, no results found”. Return the user to the **<u>Search for Item</u>** form.
- Sort the list of items item id, ascending.
- For each item, display item ID, Game Type, Title, Condition, Description, Distance and a link called ***Details***
    - ***If ‘item.description’ has more than 100 characters, display (...) at the end of first 100 characters***

- Else If the user selected a value in the ***Within X Miles*** dropdown:
    - Find the user’s location
    - Find all swap-eligible items with a user with a distance less than X miles

```
SELECT ID,
       CASE
         WHEN ID IN (SELECT ID
                     FROM   BoardGame) THEN "BoardGame"
         WHEN ID IN (SELECT ID
                     FROM   VideoGame) THEN "VideoGame"
         WHEN ID IN (SELECT ID
                     FROM   ComputerGame) THEN "ComputerGame"
         WHEN ID IN (SELECT ID
                     FROM   Jigsaw) THEN "Jigsaw"
         WHEN ID IN (SELECT ID
                     FROM   CardGame) THEN "CardGame"
       END AS Game_type,
       Title,
       Condition_,
       Description,
       distance
FROM   (SELECT ID,
               Title,
               Condition_,
               Description,
               i.Post_code AS pc1,
               u.Post_Code AS pc2
        FROM   (SELECT ID,
                       Title,
                       Condition_,
                        Description,
                        Owner,
                        Email,
                        Post_code
                 FROM   Item
                        INNER JOIN User_
                                ON Item.Owner = User_.Email
                 WHERE ID NOT IN (SELECT Desired_IID
                                      FROM   Pending_swap t1
                                      UNION
                                      SELECT Proposed_IID
                                      FROM   Pending_swap t2
                                      UNION
                                      SELECT Desired_IID
                                      FROM   Accepted_swap t3
                                      UNION
                                      SELECT Proposed_IID
                                      FROM   Accepted_swap t4
                                      UNION
                                      SELECT DISTINCT Desired_IID
                                      FROM   Rejected_swap
                                             JOIN Item
                                               ON Item.owner =
                                                  '$Email'
                                                  AND Proposed_IID =
Item.ID
                                      UNION
                                      SELECT DISTINCT Proposed_IID
                                      FROM   Rejected_swap
                                             JOIN Item
                                               ON Item.owner =
                                                  '$Email'
                                                  AND Desired_IID =
Item.ID)) AS
                i,
                (SELECT Post_code
                 FROM   User_
                 WHERE  User_.Email = '$Email') u) it
        LEFT JOIN (SELECT
                         a.post_code_1
                         AS
                         post_code_1,
                  a.post_code_2
```

```
                        AS
                        post_code_2
                        ,
                2 * 3958.75 * atan2(Sqrt(pow(Sin(( a.lat1 - a.lat2 ) / 2),
2) + Cos(a.lat1) * Cos(a.lat2)
                 *   pow(Sin( (a.long1 - a.long2 ) / 2), 2)),Sqrt(1 -
pow(Sin( (a.lat1 - a.lat2 ) /2),2) +
                 pow(Cos(a.lat2), 2) * pow(Sin((a.long1 - a.long2 ) / 2),
2)))
AS
                 distance
                  FROM   (SELECT a1.Latitude * Pi() / 180  AS lat1,
                                 a1.Longitude * Pi() / 180 AS long1,
                                 a1.Post_code              AS post_code_1,
                                 a2.Latitude * Pi() / 180  AS lat2,
                                 a2.Longitude * Pi() / 180 AS long2,
                                 a2.Post_code              AS post_code_2
                          FROM   ( Address AS a1
                                   CROSS JOIN Address AS a2 )) AS a) dis
              ON dis.post_code_1 = it.pc1
                 AND dis.post_code_2 = it.pc2
                 WHERE distance <= $Distance
ORDER  BY distance,
          ID;
```

- For each item:
    - Find the user associated with the item and the location associated with the user
- Sort the list of items by distance, then item id, ascending.
- Display item ID, Game Type, Title, Condition, Description, Distance and a link called ***Details***
    - If 'item.description' has more than 100 characters, display (...) at the end of first 100 characters

- Else if the user entered a postal code:
    - If the postal code is invalid:
        - Display an error message to the user. Return to the **Search for Item** form
    - Find all swap-eligible items with a user with the same zip code

```
SELECT ID,
       CASE
         WHEN ID IN (SELECT ID
                     FROM   BoardGame) THEN "BoardGame"
         WHEN ID IN (SELECT ID
                     FROM   VideoGame) THEN "VideoGame"
         WHEN ID IN (SELECT ID
                     FROM   ComputerGame) THEN "ComputerGame"
         WHEN ID IN (SELECT ID
                     FROM   Jigsaw) THEN "Jigsaw"
         WHEN ID IN (SELECT ID
                     FROM   CardGame) THEN "CardGame"
       END AS Game_type,
       Title,
       Condition_,
       Description,
       distance
FROM   (SELECT ID,
               Title,
               Condition_,
               Description,
               i.Post_code AS pc1,
               u.Post_Code AS pc2
        FROM   (SELECT ID,
                       Title,
                       Condition_,
                       Description,
                       Owner,
                       Email,
                       Post_code
                FROM   Item
                       INNER JOIN User_
                               ON Item.Owner = User_.Email
                WHERE  ( User_.post_code = $Postcode
                         AND User_.Email != '$Email' )
                       AND ID NOT IN (SELECT Desired_IID
                                      FROM   Pending_swap t1
                                      UNION
                                      SELECT Proposed_IID
                                      FROM   Pending_swap t2
                                      UNION
                                      SELECT Desired_IID
                                      FROM   Accepted_swap t3
                                      UNION
                                      SELECT Proposed_IID
                                      FROM   Accepted_swap t4
                                      UNION
                                      SELECT DISTINCT Desired_IID
                                      FROM   Rejected_swap
                                             JOIN Item
                                               ON Item.owner =
                                                  '$Email'
                                                  AND Proposed_IID = Item.ID
                                      UNION
                                      SELECT DISTINCT Proposed_IID
                                      FROM   Rejected_swap
                                             JOIN Item
                                               ON Item.owner =
                                                  '$Email'
                                                  AND Desired_IID = Item.ID)) AS
               i,
               (SELECT Post_code
                FROM   User_
                WHERE  User_.Email = '$Email') u) it
       LEFT JOIN (SELECT
                        a.post_code_1
                        AS
                        post_code_1,
                 a.post_code_2
                        AS
                        post_code_2
                        ,
                2 * 3958.75 * atan2(Sqrt(pow(Sin(( a.lat1 - a.lat2 ) / 2), 2) + Cos(a.lat1) *
Cos(a.lat2)
                * pow(Sin( (a.long1 - a.long2 ) / 2), 2)),Sqrt(1 - pow(Sin( (a.lat1 - a.lat2 ) /2),2) +
                pow(Cos(a.lat2), 2) * pow(Sin((a.long1 - a.long2 ) / 2), 2))) AS
                 distance
                  FROM   (SELECT a1.Latitude * Pi() / 180  AS lat1,
                                 a1.Longitude * Pi() / 180 AS long1,
                                 a1.Post_code              AS post_code_1,
                                 a2.Latitude * Pi() / 180  AS lat2,
                                 a2.Longitude * Pi() / 180 AS long2,
                                 a2.Post_code              AS post_code_2
                          FROM   ( Address AS a1
                                   CROSS JOIN Address AS a2 )) AS a) dis
              ON dis.post_code_1 = it.pc1
                 AND dis.post_code_2 = it.pc2
ORDER  BY distance,
          ID;
```

- If the result is empty,
    - display “Sorry, no results found”. Return the user to the **Search for Item** form.
- Sort the list of items item id, ascending.
- For each item, display item ID, Game Type, Title, Condition, Description, Distance and a link called ***Details***
    - If ‘item.description’ has more than 100 characters, display (...) at the end of first 100 characters

- Then, do nothing until the user clicks something
- When the user clicks ***Details*** on a specific item, run the **View Items Details** task for that item.

View Item Details

Abstract Code
(assume the $Item_subclass_table and $ID are known before viewing)
When the user clicks on **Details:**

- Display all the item's attributes: item ID, title/name, type-specific attribute, and description (if applicable)

```
Select Title, $Item_subclass_table.*,
CASE
         WHEN Item.ID IN (SELECT ID
                     FROM   BoardGame) THEN "BoardGame"
         WHEN Item.ID IN (SELECT ID
                     FROM   VideoGame) THEN "VideoGame"
         WHEN Item.ID IN (SELECT ID
                     FROM   ComputerGame) THEN "ComputerGame"
         WHEN Item.ID IN (SELECT ID
                     FROM   Jigsaw) THEN "Jigsaw"
         WHEN Item.ID IN (SELECT ID
                     FROM   CardGame) THEN "CardGame"
       END AS Game_type, Condition_ from Item LEFT JOIN
$Item_subclass_table on $Item_subclass_table.ID=Item.ID Where Item.ID=$id};
```

- IF the item belongs to another user:
    - In addition to the above, display the item's owner information {user.$nickname}, {user.$city}, {user.$state}, {user.$postalcode} and calculate the distance between the proposer and the counterparty.

```
SELECT Nick_name,
       City,
       State,
       distance
FROM   (SELECT Nick_name,
               City,
               State,
               User_.Post_code
        FROM   Item
               LEFT JOIN User_
                      ON Item.Owner = User_.Email
               LEFT JOIN Address
                      ON User_.Post_code = Address.Post_code
        WHERE  ID = $id) t1,
       (SELECT Post_code
        FROM   User_
        WHERE  Email = '$Email')t2,
       (SELECT a.post_code_1
               AS
               post_code_1,
               a.post_code_2
               AS post_code_2,
                2 * 3958.75 * atan2(Sqrt(pow(Sin(( a.lat1 - a.lat2 ) / 2),
2) + Cos(a.lat1) * Cos(a.lat2) *   pow(Sin( (a.long1 - a.long2 ) / 2),
2)),Sqrt(1 - pow(Sin( (a.lat1 - a.lat2 ) /2),2) +
pow(Cos(a.lat2), 2) * pow(Sin((a.long1 - a.long2 ) / 2), 2)))
               AS
               distance
        FROM   (SELECT a1.Latitude * Pi() / 180  AS lat1,
                       a1.Longitude * Pi() / 180 AS long1,
                       a1.Post_code              AS post_code_1,
                       a2.Latitude * Pi() / 180  AS lat2,
                       a2.Longitude * Pi() / 180 AS long2,
                       a2.Post_code              AS post_code_2
                FROM   ( Address AS a1
                         CROSS JOIN Address AS a2 )) AS a) dis
WHERE  t1.Post_code = dis.post_code_1
       AND t2.Post_code = dis.post_code_2;
```

- Owner’s rating need to be calculated based on the records in the swap table (method is also mentioned in **view main menu**)

```
SELECT AVG(rating) AS My_Rating FROM (
  SELECT Item.ID, Proposer_rating AS rating FROM (
    Accepted_swap
    INNER JOIN Item
    ON Item.ID = Accepted_swap.Proposed_IID AND "$Email" = Item.Owner
  )
  UNION
  SELECT Item.ID, Counterparty_rating AS rating FROM (
    Accepted_swap
    INNER JOIN Item
    ON Item.ID = Accepted_swap.Desired_IID AND "$Email" = Item.Owner
  )
) AS ratings
WHERE ratings.rating IS NOT NULL;
```

- IF the distance is:
    - between 0.0 and 25.0 miles:
        - Display distance with highlighted green background
    - Between 25.0 and 50.0 miles:
        - Display distance with highlighted yellow background
    - Between 50.0 and 100.0 miles
        - Display distance with highlighted orange background
    - Over 100.0 miles:
        - Display distance with highlighted red background
- IF the current user does not have more than 2 unrated swaps or more than 5 unaccepted swaps and the item is available for swapping:

```
//unrated_proposed_item
SELECT Count(*) AS unrated_proposed_item
FROM   Accepted_swap
       LEFT JOIN Item I1
              ON Accepted_swap.Proposed_IID = I1.ID
       LEFT JOIN Item I2
              ON Accepted_swap.Desired_IID = I2.ID
WHERE  ( I1.Owner = '$Email'
         AND Proposer_rating IS NULL )
        OR ( I2.Owner = '$Email'
             AND Counterparty_rating IS NULL );

//number_of_unaccepted_swap
SELECT Count(*) AS number_of_unaccepted_swap
FROM   Pending_swap
       LEFT JOIN Item I1
              ON Accepted_swap.Proposed_IID = I1.ID
       LEFT JOIN Item I2
              ON Accepted_swap.Desired_IID = I2.ID
WHERE  I1.Owner = '$Email' OR I2.Owner = '$Email';
```

    - IF the user clicks the ***Propose Swap*** button, go to **Proposing a Swap** form

View “My Items” Section

Abstract Code

- Define variable $Email as user email from URL / HTTP request
- Create a table labeled “Item Counts”. For each game type, create a column header named after the type. Populate with the counts of user’s swappable items by type, in the appropriate column, from the query below. The variable $Email will represent the current user’s email.

```
SELECT (SELECT Count(*)
        FROM   Item i,
               VideoGame v
        WHERE  i.ID = v.ID
               AND i.Owner = '$Email'
               AND i.ID NOT IN (SELECT Desired_IID
                                FROM   Pending_swap)
               AND i.ID NOT IN (SELECT Proposed_IID
                                FROM   Pending_swap)
               AND i.ID NOT IN (SELECT Desired_IID
                                FROM   Accepted_swap)
               AND i.ID NOT IN (SELECT Proposed_IID
                                FROM   Accepted_swap)) AS CountVideoGame,
       (SELECT Count(*)
        FROM   Item i,
               BoardGame b
        WHERE  i.ID = b.ID
               AND i.Owner = '$Email'
               AND i.ID NOT IN (SELECT Desired_IID
                                FROM   Pending_swap)
               AND i.ID NOT IN (SELECT Proposed_IID
                                FROM   Pending_swap)
               AND i.ID NOT IN (SELECT Desired_IID
                                FROM   Accepted_swap)
               AND i.ID NOT IN (SELECT Proposed_IID
                                FROM   Accepted_swap)) AS CountBoardGame,
       (SELECT Count(*)
        FROM   Item i,
               Jigsaw j
        WHERE  i.ID = j.ID
               AND i.Owner = '$Email'
               AND i.ID NOT IN (SELECT Desired_IID
                                FROM   Pending_swap)
               AND i.ID NOT IN (SELECT Proposed_IID
                                FROM   Pending_swap)
               AND i.ID NOT IN (SELECT Desired_IID
                                FROM   Accepted_swap)
               AND i.ID NOT IN (SELECT Proposed_IID
                                FROM   Accepted_swap)) AS CountJigsaw,
       (SELECT Count(*)
        FROM   Item i,
               ComputerGame comp
        WHERE  i.ID = comp.ID
               AND i.Owner = '$Email'
               AND i.ID NOT IN (SELECT Desired_IID
                                FROM   Pending_swap)
               AND i.ID NOT IN (SELECT Proposed_IID
                                FROM   Pending_swap)
               AND i.ID NOT IN (SELECT Desired_IID
                                FROM   Accepted_swap)
               AND i.ID NOT IN (SELECT Proposed_IID
                                FROM   Accepted_swap)) AS
CountComputerGame,
       (SELECT Count(*)
        FROM   Item i,
               CardGame card
        WHERE  i.ID = card.ID
               AND i.Owner = '$Email'
               AND i.ID NOT IN (SELECT Desired_IID
                                FROM   Pending_swap)
               AND i.ID NOT IN (SELECT Proposed_IID
                                FROM   Pending_swap)
               AND i.ID NOT IN (SELECT Desired_IID
                                FROM   Accepted_swap)
               AND i.ID NOT IN (SELECT Proposed_IID
                                FROM   Accepted_swap)) AS CountCardGame,
       (SELECT Count(*)
        FROM   Item i
        WHERE  i.Owner = '$Email'
               AND i.ID NOT IN (SELECT Desired_IID
                                FROM   Pending_swap)
               AND i.ID NOT IN (SELECT Proposed_IID
                                FROM   Pending_swap)
               AND i.ID NOT IN (SELECT Desired_IID
                                FROM   Accepted_swap)
               AND i.ID NOT IN (SELECT Proposed_IID
                                FROM   Accepted_swap)) AS
       TotalSwappableItemCount;
```

- Beneath the “Item Counts” table, display another table called “My Items” with a column for Item Number, Game Type, Title, Condition, Description, and an untitled column to use for links to the item’s details.
- Find ITEMs including ID, Title, Condition, Description, and Game_type, where $Email is owner and item is not associated with a pending or complete swap using the query below. Save results to variable $user_items. The variable $Email will represent the current user’s email.

```
SELECT ID, Title, Condition_, Description ,
      CASE
      WHEN i.id IN (SELECT id
                        FROM   BoardGame) THEN "boardgame"
      WHEN i.id IN (SELECT id
                        FROM   VideoGame) THEN "videogame"
      WHEN i.id IN (SELECT id
                        FROM   ComputerGame) THEN "computergame"
      WHEN i.id IN (SELECT id
                        FROM   Jigsaw) THEN "jigsaw"
      WHEN i.id IN (SELECT id
                        FROM   CardGame) THEN "cardgame"
      END AS Game_type
FROM   Item i
WHERE  i.owner = '$Email'
      AND i.id NOT IN (SELECT desired_iid
                        FROM   Pending_swap)
      AND i.id NOT IN (SELECT proposed_iid
                        FROM   Pending_swap)
      AND i.id NOT IN (SELECT desired_iid
                        FROM   Accepted_swap)
      AND i.id NOT IN (SELECT proposed_iid
                        FROM   Accepted_swap);
```

- Do nothing
- If the ***Details*** link is clicked for an item
    - Do the **View Details** task for that item.

Propose Swaps

Abstract Code

Subtask 1: Proposal Dialogue

- Calculate distance from current/proposer USER to the counterparty USER by identifying each ADDRESS via the USER
    - IF the distance between current/proposer ADDRESS and counterparty ADDRESS is greater than or equal to 100 miles, display a warning with distance to counterparty in miles

```
SELECT a.distance,
       a.post_code_1,
       a.post_code_2
FROM
  (SELECT a.post_code_1 AS post_code_1,
          a.post_code_2 AS post_code_2,
          2 *  3958.75 * atan2(sqrt(pow(sin((a.lat1 - a.lat2) / 2), 2) +
cos(a.lat1) * cos(a.lat2) * pow(sin((a.long1 - a.long2) / 2), 2)), sqrt(1 -
pow(sin((a.lat1 - a.lat2) / 2), 2) + pow(cos(a.lat2), 2) * pow(sin((a.long1 -
a.long2) / 2), 2))) AS distance
   FROM
     (SELECT a1.Latitude * PI() / 180 AS lat1,
             a1.Longitude * PI() / 180 AS long1,
             a1.Post_code AS post_code_1,
             a2.Latitude * PI() / 180 AS lat2,
             a2.Longitude * PI() / 180 AS long2,
             a2.Post_code AS post_code_2
      FROM (Address AS a1
            CROSS JOIN Address AS a2)) AS a) AS a
WHERE a.post_code_1 =
    (SELECT Post_code
     FROM User_
     WHERE Email = '$Email_1' )
  AND a.post_code_2 =
    (SELECT Post_code
     FROM User_
     WHERE Email = '$Email_2' );
```

- Define variable $eligible_items
- Find ITEMS where owner is the current USER and item is not associated with a pending or complete SWAP and save the list of result items to the $eligible_items variable

- Display a table with column headers “Item #”, “Game Type”, “Title”, and “Condition”. An additional empty column will be shown.

```
SELECT ID,
       Game_type,
       Title,
       Condition_,
       Owner
FROM Item
WHERE (Owner='$Email'
       AND id NOT IN
         (SELECT desired_iid AS item_id
          FROM Pending_swap t1
          UNION SELECT proposed_iid AS item_id
          FROM Pending_swap t2
          UNION SELECT desired_iid AS item_id
          FROM Accepted_swap t3
          UNION SELECT proposed_iid AS item_id
          FROM Accepted_swap t4));
```

- For each item in $eligible_items, create a row in the table and populate the item ID, game type, title, and condition in the matching column. In the final column with the empty header, display a ticker box labeled “Select”
- Do nothing.
- When a user clicks on one of their item’s “Select” box, record that item’s ID in the active session as the proposed item, and create a button ***Confirm***.
- Do nothing.
- WHEN the user clicks on another item
    - Record that item’s ID in the active session as the proposed item
    - Deselect all other tickers associated with other items.
- WHEN the user clicks ***Confirm***
    - Execute **Propose a Swap** subtask with the current session’s proposed item ID

Subtask 2: **Propose a Swap**

- Create an instance of the SWAP entity representing the two items involved in the swap.

```
INSERT INTO Pending_swap (proposed_iid, desired_iid, proposed_date)
VALUES ('$proposed_iid', '$desired_iid', '$proposed_date');
```

- Display confirmation message with a hyperlink button to the **View Main menu** task

Accept/Reject Swaps

Abstract Code

- Define variable $Email as user email from URL / HTTP request
- Find SWAPs where counterparty ITEM is associated with a USER which matches the $Email and status is not ***Accept*** or ***Reject***, and save to variable $proposed_swaps
- Sort $proposed_swaps by “date” proposed (in ascending order)
- Render a table with headers “Date”, “Desired Item”, “Proposer”, “Rating”, “Distance”, “Proposed Item”, and a final empty column
- For each swap in $proposed_swaps, render a row and populate each column with the appropriate value. Round number rating to hundredths. Round Distance to tenths. In empty column show ***Accept*** and ***Reject*** buttons

```
SELECT t2.email AS proposer_email,
       t2.proposed_date,
       t2.proposed_iid,
       t2.desired_iid,
       t2.desired_item,
       t2.nick_name,
       t2.proposed_item,
       Avg(rating) AS rating
FROM
  (SELECT t1.proposed_date,
          t1.proposed_iid,
          t1.desired_iid,
          t1.desired_item,
          i.title AS proposed_item,
          u.Nick_name,
          u.email
   FROM
     (SELECT ps.proposed_date,
             ps.proposed_iid,
             ps.desired_iid,
             i.title AS desired_item,
             u.post_code AS my_post_code
      FROM Pending_swap ps
      INNER JOIN Item i ON ps.desired_iid=i.id
      INNER JOIN User_ u ON i.owner=u.email
      INNER JOIN Address a ON u.post_code = a.post_code
      WHERE i.owner='$Email') t1
   INNER JOIN Item i ON t1.proposed_iid=i.id
   INNER JOIN User_ u ON i.owner=u.email
   INNER JOIN
     (SELECT a.post_code_1 AS post_code_1,
             a.post_code_2 AS post_code_2,
             2 * 3958.75 * atan2(sqrt(pow(sin((a.lat1 - a.lat2) / 2), 2) +
cos(a.lat1) * cos(a.lat2) * pow(sin((a.long1 - a.long2) / 2), 2)), sqrt(1 -
pow(sin((a.lat1 - a.lat2) / 2), 2) + pow(cos(a.lat2), 2) * pow(sin((a.long1
- a.long2) / 2), 2))) AS distance
      FROM
        (SELECT a1.Latitude * PI() / 180 AS lat1,
                a1.Longitude * PI() / 180 AS long1,
                a1.Post_code AS post_code_1,
                a2.Latitude * PI() / 180 AS lat2,
                a2.Longitude * PI() / 180 AS long2,
                a2.Post_code AS post_code_2
         FROM (Address AS a1
               CROSS JOIN Address AS a2)) AS a) AS d ON u.post_code =
d.post_code_1
   AND t1.my_post_code = d.post_code_2) t2
LEFT JOIN (
             (SELECT u.email,
                     a_s.proposer_rating AS rating
              FROM Accepted_swap a_s
              INNER JOIN Item i ON a_s.proposed_iid =i.id
              INNER JOIN User_ u ON i.owner=u.email)
           UNION ALL
             (SELECT u.email,
                     a_s.counterparty_rating AS rating
              FROM Accepted_swap a_s
              INNER JOIN Item i ON a_s.desired_iid =i.id
              INNER JOIN User_ u ON i.owner=u.email)) t3 ON
t2.email=t3.email
GROUP BY t2.proposed_date,
         t2.proposed_iid,
         t2.desired_iid,
         t2.desired_item,
         t2.nick_name,
         t2.proposed_item;
```

- Do nothing until the user clicks either ***Accept*** or ***Reject***.
- If the user clicks ***Accept*** on a proposed swap

- IF phone information is present and phone is shared
    - Display dialog with proposer's $email, $first_name, $phone_number/type (if available) and sharing option (if set).

```
SELECT User_.email,
   User_.first_name,
   CASE Phone.shareable
     WHEN NULL THEN NULL
     WHEN false THEN NULL
     WHEN true THEN Phone.number_
   end Phone_number,
   CASE Phone.shareable
     WHEN NULL THEN NULL
     WHEN false THEN NULL
     WHEN true THEN Phone.type_
   end Phone_type
FROM   User_ left join Phone on User_.email = Phone.owner
WHERE  User_.email = '$proposer_email_from_task1';
```

- IF phone number is available, but not shareable
    - display as not available
- Record a value of "***Accept***" for the status of the swap in the SWAP table

- Record a value of current time in UTC for the $accept_reject_date of the swap in the SWAP table

```
START TRANSACTION;


INSERT INTO Accepted_swap
(
    proposed_iid ,
    desired_iid ,
    proposed_date ,
    accept_reject_date
)  select proposed_iid,
    desired_iid,
    proposed_date,
    NOW()
from Pending_swap
WHERE proposed_iid = $proposed_iid
AND    desired_iid = $desired_iid;
```

```
    DELETE
    FROM   Pending_swap
    WHERE  proposed_iid = $proposed_iid
    AND    desired_iid = $desired_iid;

    COMMIT;
```

- Jump to **Rate Swaps** task

- Else If the user clicks ***Reject*** on a proposed swap
    - Record a value of "***Reject***" for the status of the swap in the SWAP table
    - Record a value of current time in UTC for the accept_reject_date of the swap in the SWAP entity

```
START TRANSACTION;


    INSERT INTO Rejected_swap
    (
        proposed_iid ,
        desired_iid ,
        proposed_date ,
        accept_reject_date
    )  select proposed_iid,
        desired_iid,
        proposed_date,
        NOW()
    from Pending_swap
    WHERE proposed_iid = $proposed_iid
    AND    desired_iid = $desired_iid;


    DELETE
    FROM   Pending_swap
    WHERE  proposed_iid = $proposed_iid
    AND    desired_iid = $desired_iid;

    COMMIT;
```

- If the number of proposed swaps is zero
    - User will be returned to <u>**Main Menu**</u>

View Swap History

Abstract Code

- Calculate Rejected % for proposer USER and counterparty USER by dividing the rejected swaps by total swaps
- Render summary metrics: summary, logged in user's total swaps proposed, total received, subtotals for accepted and rejected, and % rejected
    - If % rejected (rounded to tenths) >=50.0%
        - highlight % rejected in red
    - Else
        - Do nothing

```
SELECT
    COUNT(*) as total,
    SUM(case when status = "Accepted" then 1 else 0 end) / COUNT(*) as
accept_pct,
    SUM(case when status = "Rejected" then 1 else 0 end) / COUNT(*) as
reject_pct,
    swap_roles.My_role

FROM (
    SELECT Proposed_IID, Desired_IID,  "Accepted" AS status, "Proposer" AS
My_role
    FROM Accepted_swap
    INNER JOIN Item
    ON ID = Proposed_IID AND Owner = '$Email'
    UNION
    SELECT Proposed_IID, Desired_IID, "Rejected" AS status, "Proposer" AS
My_role
    FROM Rejected_swap
    INNER JOIN Item
    ON ID = Proposed_IID AND Owner = '$Email'
    UNION
    SELECT Proposed_IID, Desired_IID, "Accepted" AS status, "Counterparty"
AS My_role
    FROM Accepted_swap
    INNER JOIN Item
    ON ID = desired_IID AND Owner = '$Email'
    UNION
    SELECT Proposed_IID, Desired_IID,  "Rejected" AS status, "Counterparty"
AS My_role
    FROM Rejected_swap
    INNER JOIN Item
    ON ID = desired_IID AND Owner =  '$Email'
    ) as swap_roles
GROUP BY
swap_roles.My_role;
```

- For each swap for current user, render a row with acceptance/rejection date, swap proposed date, proposed item title, desired item title, other user’s nickname, rating
    - If swap has not been rated
        - Render dropdown list with rating options as integers 0 through 5
        - When rating is selected update the associated SWAP entity’s rating and refresh the page
    - If User clicks ***Detail*** button
        - Jump to **View Swap Details** task

```
SELECT
    a.Proposed_date,
    a.Accept_reject_date,
    Swap_status,
    My_role,
    Proposed_item,
    Title AS Desired_item,
    nick_name AS Other_User,
    Rating
FROM (
    SELECT
        Proposed_IID,
        Desired_IID,
        Proposed_date,
        Accept_reject_date,
        Title AS Proposed_item,
        "Accepted" AS Swap_status,
        "Proposer" AS My_role,
        Proposer_rating AS Rating,
        Desired_IID as other_item
    FROM Accepted_swap
    INNER JOIN Item
    ON ID = Proposed_IID AND Owner = '$Email'
    UNION
    SELECT
        Proposed_IID,
        Desired_IID,
        Proposed_date,
        Accept_reject_date,
        Title AS Proposed_item,
        "Rejected" AS Swap_status,
        "Proposer" AS My_role,
        NULL as Rating,
        Desired_IID as other_item
    FROM Rejected_swap
    INNER JOIN Item
    ON ID = Proposed_IID AND Owner = '$Email'
    UNION
    SELECT
        Proposed_IID,
        Desired_IID,
        Proposed_date,
        Accept_reject_date,
        Title AS Proposed_item,
        "Accepted" AS Swap_status,
        "Counterparty" AS My_role,
        Counterparty_rating AS Rating,
        Proposed_IID as other_item
    FROM Accepted_swap
    INNER JOIN Item
    ON ID = Desired_IID AND Owner = '$Email'
    UNION
    SELECT
        Proposed_IID,
        Desired_IID,
        Proposed_date,
        Accept_reject_date,
        Title AS Proposed_item,
        "Rejected" AS Swap_status,
        "Counterparty" AS My_role,
        NULL as Rating,
        Proposed_IID as other_item
    FROM Rejected_swap
    INNER JOIN Item
    ON ID = Desired_IID AND Owner = '$Email'
) as a
    INNER JOIN Item
    ON Item.ID = a.other_item
    INNER JOIN User_
    ON User_.email = Item.Owner;
```

Rate Swaps

Abstract Code

- User clicked on the ***Unrated Swaps*** panel in the **Main Menu**
- Display all accepted swaps for user, where user has not rated.
    - Populate a table with the swap acceptance date, the user's role in the proposal (proposer or counterparty), proposed item title, desired item title, other user's nickname, and a drop-down rating list (0 – 5), ordered by acceptance date descending
    - Data for the above table will be retrieved using the query below. $Email will represent the current user's email.

```
SELECT Accepted_swap.Proposed_iid,
       Accepted_swap.Desired_iid,
       Accepted_swap.Accept_reject_date,
       CASE
         WHEN i1.owner = '$Email' THEN 'Proposer'
         ELSE 'Counterparty'
       END      AS my_role,
       i1.Title AS proposed_item,
       i2.Title AS desired_item,
       CASE
         WHEN i1.Owner = '$Email' THEN u2. Nick_name
         ELSE u1.Nick_name
       END      AS other_user
FROM   Accepted_swap
       LEFT JOIN Item i1
              ON Accepted_swap.proposed_iid = i1.ID
       LEFT JOIN Item i2
              ON Accepted_swap.desired_iid = i2.ID
       LEFT JOIN User_ u1
              ON i1.owner = u1.email
       LEFT JOIN User_ u2
              ON i2.owner = u2.email
WHERE  ( i1.owner = '$Email'
         AND Proposer_rating IS NULL )
        OR ( i2.owner = '$Email'
             AND Counterparty_rating IS NULL );
```

- If current user chooses a rating for an unrated swap
    - Record the rating in the database using the appropriate query below. "my_role" from the previous query can be used to determine which update method to use. The rating the user

provided will be represented with the variable $Rating in the query. Variables $Desired_id and $Proposed_id will be used to identify the swap.

```
//WHEN USER IS PROPOSER
```

```
UPDATE Accepted_swap
SET   Proposer_rating = '$Rating'
WHERE  Desired_iid = '$Desired_id'
      AND proposed_iid = '$Proposed_id';
```

```
//WHEN USER IS COUNTERPARTY
```

```
UPDATE Accepted_swap
SET    Counterparty_rating = '$Rating'
WHERE  Desired_iid = '$Desired_id'
       AND proposed_iid = '$Proposed_id';
```

- IF additional swaps are unrated (get this info using the same query used to display unrated swaps)
    - refresh the **Rate Swaps** form
- IF no additional swaps need rating
    - return the user to the **Main Menu**

View Swap Details

Abstract Code

- User clicked on the ***Detail*** button in the **<u>Swap History</u>** form
- Render Swap Detail Page
    - When the user clicks “Details” from the **<u>View Swap History</u>** form, Swap Status (Accepted/Rejected), swap ID (Proposed_IID, Desired_IID), and current user Email are provided inputs to this task.
    - In upper left-hand corner of the page, display the selected swap’s details (proposed date, accepted/rejected date, swap status, current user’s role, and rating current user left (if available).
    - IF the input from the **<u>View Swap History</u>** form for Swap Status is Accepted, use the query below to find values for each of the above fields. The current user’s email will be interpolated into the query with the variable $Email. The swap’s identifying desired ID and proposed ID will also be interpolated into the query as the variables $Desired_ID and $Proposed_ID.

```
SELECT Proposed_date,
      Accept_reject_date,
      "Accepted" AS Status,
      CASE
      WHEN i1.Owner = '$Email' THEN "Proposer"
      WHEN i2.Owner = '$Email' THEN "Counterparty"
      END         AS MyRole,
      CASE
      WHEN i1.Owner = '$Email' THEN Proposer_rating
      WHEN i2.Owner = '$Email' THEN Counterparty_rating
      END         AS RatingLeft
FROM   Accepted_swap
      LEFT JOIN Item i1
            ON i1.ID = Proposed_IID
      LEFT JOIN Item i2
            ON i2.ID = Desired_IID
WHERE  Proposed_IID = '$Proposed_id'
      AND Desired_IID = '$Desired_id';
```

    - IF the input from the **<u>View Swap History</u>** form for Swap Status is Rejected, use the query below to find values for each of the fields in the “Swap Details” panel in the upper left-hand corner of the page.

```
 SELECT Proposed_date,
      Accept_reject_date,
      "Rejected"        AS Status,
      CASE
      WHEN i1.Owner = '$Email' THEN "Proposer"
      WHEN i2.Owner = '$Email' THEN "Counterparty"
      END               AS MyRole,
      "Not Applicable" AS RatingLeft
FROM   Rejected_swap
      LEFT JOIN Item i1
            ON i1.ID = Proposed_IID
      LEFT JOIN Item i2
            ON i2.ID = Desired_IID
WHERE  Proposed_IID = '$Proposed_id'
      AND Desired_IID = '$Desired_id';
```

- IF the selected swap is accepted and not rated:
    - Allow a rating to be submitted with rating dropdown (0-5)
    - IF user submits rating
        - refresh/update the form after submission
    - Use the “my_role” value from the “Swap Details” panel query to determine which statement to use to update the rating.
        - IF user is the proposer, update rating as below. The variable $Rating will be used to interpolate the value provided by the user into the statement. The variables $Desired_id and $Proposed_id will be set to the values provided when this task was linked to from the **View Swap History** page and interpolated into the SQL statement.

```
UPDATE Accepted_swap
SET   Proposer_rating = '$Rating'
WHERE  Desired_iid = '$Desired_id'
      AND proposed_iid = '$Proposed_id';
```

        - IF user is the counterparty, update rating as below

```
UPDATE Accepted_swap
SET   Counterparty_rating = '$Rating'
WHERE  Desired_iid = '$Desired_id'
      AND proposed_iid = '$Proposed_id';
```

    - In the upper right-hand corner of the page, for both rejected and accepted swaps, show the other user’s nickname and distance.
    - IF the swap is accepted, also show the user’s contact details (first name and email)
    - IF the swap is accepted and the other user’s phone is available and shared, display the phone number and type.

- IF the swap is accepted, retrieve the information for the user info panel using the query below. This query includes phone and contact info if available. Variables used previously to interpolate the current user's email and the swap's identifying attributes will be used in this query.

```
SELECT Email,
       Nick_name,
       First_name,
       CASE
              WHEN p.Number_ IS NOT NULL
              AND    p.Shareable = 1 THEN p.Number_
              ELSE 'Not Available'
       END AS Number_,
       CASE
              WHEN p.Number_ IS NOT NULL
              AND    p.Shareable = 1 THEN p.Type_
              ELSE 'Not Available'
       END AS Phone_type,
       (
              SELECT distance
              FROM   (
                            SELECT distance
                            FROM   (
                                          SELECT a.post_code_1
AS post_code_1,
                                                 a.post_code_2
AS post_code_2,
                                                 2 * 3958.75 *
Atan2(Sqrt(Pow(Sin((a.lat1 - a.lat2) / 2), 2) + Cos(a.lat1) * Cos(a.lat2) *
Pow(Sin((a.long1 - a.long2) / 2), 2)), Sqrt(1 - Pow(Sin((a.lat1 - a.lat2) /
2), 2) + Pow(Cos(a.lat2), 2) * Pow(Sin((a.long1 - a.long2) / 2), 2))) AS
distance
                                          FROM   (
                                                        SELECT
a1.Latitude  * Pi() / 180 AS lat1,

a1.Longitude * Pi() / 180 AS long1,

a1.Post_code              AS post_code_1,

a2.Latitude  * Pi() / 180  AS lat2,
a2.Longitude * Pi() / 180  AS long2,

a2.Post_code               AS post_code_2
                                                  FROM
(Address AS a1
                                                  CROSS JOIN
Address  AS a2)) AS a) AS D_table
                            WHERE  D_table.post_code_1 = u2.Post_code
                            AND    D_table.Post_code_2 IN
                                   (
                                          SELECT u1.Post_code
                                          FROM   User_ u1
                                          WHERE  u1.Email = '$Email' ) ) AS
Distance
              FROM   User_ u2,
                     Item i,
                     Accepted_swap a,
                     Phone p
              WHERE  u2.Email = i.Owner
              AND    u2.Email != '$Email'
              AND    p.Owner = i.Owner
              AND    (
                            i.ID = a.Proposed_IID
                     OR     i.ID = a.Desired_IID )
              AND    a.Proposed_IID = '$Proposed_id'
              AND    a.Desired_IID = '$Desired_id' ;
```

- IF the swap is rejected, use the query below to retrieve the user info. This will not display contact information. Only nickname and distance will be displayed.

```
SELECT Nick_name,
      (SELECT distance
      FROM   (SELECT a.post_code_1
                           AS
                           post_code_1,
                     a.post_code_2
                           AS post_code_2,
                     2 * 3958.75 * Atan2(Sqrt(Pow(Sin(( a.lat1 - a.lat2
) / 2)
                                                   , 2) +
                                                Cos(a.lat1) * Cos(a.lat2) *
                                                              Pow(Sin(
                                                ( a.long1 - a.long2 ) / 2),
                                                2)),
                                                Sqrt(
                                                              1 -
Pow(Sin(
                                                              (
                                                a.lat1 - a.lat2 ) /
                                                              2),
                                                              2) +
Pow(Cos(a.lat2), 2) *
Pow(Sin(
(
a.long1 - a.long2 ) /
2), 2)))
AS
distance
FROM   (SELECT a1.Latitude * Pi() / 180  AS lat1,
a1.Longitude * Pi() / 180 AS long1,
a1.Post_code            AS post_code_1,
a2.Latitude * Pi() / 180  AS lat2,
a2.Longitude * Pi() / 180 AS long2,
a2.Post_code            AS post_code_2
FROM   (Address AS a1
CROSS JOIN Address AS a2)) AS a) AS D_table
WHERE  D_table.post_code_1 = u2.Post_code
AND D_table.Post_code_2 IN (SELECT u1.Post_code
FROM   User_ u1
WHERE  u1.Email = '$Email')) AS Distance
FROM   User_ u2,
      Item i,
      Rejected_swap s
WHERE  u2.Email = i.Owner
      AND u2.Email != '$Email'
      AND ( i.ID = s.Proposed_IID
            OR i.ID = s.Desired_IID )
      AND s.Proposed_IID = '$Proposed_id'
      AND s.Desired_IID = '$Desired_id';
```

- For both accepted and rejected swaps, display the proposed item's details in the bottom left-hand corner. Retrieve the information using the query below. The variable $Swap_table will interpolate the table into the query, $Proposed_id and $Desired_id will be interpolated to identify the Swap

```
 SELECT ID,
      Title,
      Condition_,
      Description,
      CASE
      WHEN i.ID IN (SELECT ID
                    FROM   BoardGame) THEN "BoardGame"
      WHEN i.ID IN (SELECT ID
                    FROM   VideoGame) THEN "VideoGame"
      WHEN i.ID IN (SELECT ID
                    FROM   ComputerGame) THEN "ComputerGame"
      WHEN i.ID IN (SELECT ID
                    FROM   Jigsaw) THEN "Jigsaw"
      WHEN i.ID IN (SELECT ID
                    FROM   CardGame) THEN "CardGame"
      END AS Game_type
FROM   Item i
      JOIN $Swap_table as s
      ON i.ID = s.Proposed_IID
WHERE  s.Proposed_IID = '$Proposed_id'
      AND s.Desired_IID = '$Desired_id'
      AND i.ID = '$Proposed_id';
```

- For both accepted and rejected swaps, display the desired item's details in the bottom right-hand corner. Retrieve the information using the query below. The variable $Swap_table will interpolate the table (Accepted_swap or Rejected_swap) into the query, $Proposed_id and $Desired_id will be interpolated to identify the swap.

```
 SELECT ID,
      Title,
      Condition_,
      Description,
      CASE
      WHEN i.ID IN (SELECT ID
                    FROM   BoardGame) THEN "BoardGame"
      WHEN i.ID IN (SELECT ID
                    FROM   VideoGame) THEN "VideoGame"
      WHEN i.ID IN (SELECT ID
                    FROM   ComputerGame) THEN "ComputerGame"
      WHEN i.ID IN (SELECT ID
                    FROM   Jigsaw) THEN "Jigsaw"
      WHEN i.ID IN (SELECT ID
                    FROM   CardGame) THEN "CardGame"
      END AS Game_type
FROM   Item i
      JOIN $Swap_table as s
      ON i.ID = s.Proposed_IID
WHERE  s.Proposed_IID = '$Proposed_id}'
      AND s.Desired_IID = 'Desired_id'
      AND i.ID = 'Proposed_id';
```