Table of Contents:

Login Form

Abstract Code:

- If user has an account, then:
    - User enters *email* or *phone_number*, *password* input fields.
    - Upon:
        - Click ***Enter*** button

```
SELECT password FROM user WHERE user.email = 'email';
```

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

User Registration Form

Abstract Code:

- User enters *email*, *nickname*, *password*, *city*, *first_name*, *last_name*, *state* , *postal_code* in required input fields.
- If user inputs *phone_number*, then:
    - User selects *phone_type* of *phone_number* in dropdown
- If user selects checkbox
    - This user's phone.*disclosure_choice* == true upon write
- Else
    - This user's phone.*disclosure_choice* == false upon write
- Upon:
    - Click ***Register*** button

- If *postal_code* is not on the list of valid postal codes, then show the error message “Postal code invalid.”

```
SELECT postal_code FROM address WHERE postal_code = ‘postal_code’;
```

- Else if any *email* == user.*email* or *phone_number* == phone.*phone_number* , then show error message “User email or phone number is already registered”

```
SELECT email FROM user WHERE email = ‘email’;
SELECT phone_number FROM phone WHERE phone_number = ‘phone_number’;
```

- Else write user’s input into user and phone table

```
INSERT INTO user VALUES (‘email’, ‘password’, ‘nickname’, ‘first_name’,
‘last_name’, ‘postal_code’);
INSERT INTO phone VALUES (‘phone_number’, ‘email’, ‘phone_type’, ‘is_shared’);
```

- Jump to **Login Form**

Main Menu Form

Abstract Code:

- Query user’s *first_name* and *last_name* from user table and display a welcome message

```
SELECT first_name, last_name FROM user WHERE email = ‘email’;
```

- Display the following statistics:
  - “My Rating” using average of all ratings associated with the current user from accepted_swap table:
- Display “None” if no ratings have been made for the user’s items.

```
SELECT (SUM(proposer_rating) + SUM(counterparty_rating)) /
(COUNT(proposer_rating) + COUNT(counterparty_rating)) AS total_rating
FROM accepted_swap AS acc
LEFT JOIN swap AS a ON acc.item_number_pro = a.item_number_pro
LEFT JOIN item AS b ON b.item_number = a.item_number_pro
LEFT JOIN user AS p ON p.email = b.email
LEFT JOIN swap AS c ON acc.item_number_counter = c.item_number_counter
LEFT JOIN item AS d ON d.item_number = c.item_number_counter
LEFT JOIN user AS w ON w.email = d.email
WHERE p.email = 'email' OR w.email = 'email';
```

- “Unaccepted Swaps” using swap table:
  - If the number of “Unaccepted Swaps” greater than zero, create a clickable link can jump to **Accept/Reject Swaps Form.**

```
SELECT COUNT(swap_status)
FROM user
LEFT JOIN item ON item.email = user.email
LEFT JOIN swap ON swap.item_number_counter = item.item_number
WHERE user.email = 'email' AND swap_status = 'pending';
```

  - If any swaps are more than five days old, or the user has more than five “Unaccepted Swaps”, print the number in bold and in red.

```
SELECT COUNT(swap_status)
FROM user
LEFT JOIN item ON item.email = user.email
LEFT JOIN swap ON swap.item_number_counter = item.item_number
WHERE user.email = 'email’ AND swap_status = 'pending'  AND CURDATE() -
propose_date > 5;
SELECT COUNT(swap_status)
FROM user
LEFT JOIN item ON item.email = user.email
LEFT JOIN swap ON swap.item_number_counter = item.item_number
WHERE user.email = 'email' AND swap_status = 'pending';
```

- “Unrated Swaps” using accepted_swap table*:*
  - If the number of “Unrated Swaps” greater than zero, create a clickable link can jump to **Rate Swaps Form.**

```
SELECT (COUNT(proposer_rating) + COUNT(counterparty_rating)) AS
unrated_swaps
FROM accepted_swap AS acc
LEFT JOIN swap AS a ON acc.item_number_pro = a.item_number_pro
LEFT JOIN item AS b ON b.item_number = a.item_number_pro
LEFT JOIN user AS p ON p.email = b.email
LEFT JOIN swap AS c ON acc.item_number_counter = c.item_number_counter
LEFT JOIN item AS d ON d.item_number = c.item_number_counter
LEFT JOIN user AS w ON w.email = d.email
WHERE proposer_rating IS NULL OR counterparty_rating IS NULL AND (p.email =
'email' OR w.email = 'email');
```

- If the number of "Unrated Swaps" greater than 2, print the number in bold and red.

```
SELECT (COUNT(proposer_rating) + COUNT(counterparty_rating)) > 2 AS
unrated_swaps_above_two
FROM accepted_swap AS acc
LEFT JOIN swap AS a ON acc.item_number_pro = a.item_number_pro
LEFT JOIN item AS b ON b.item_number = a.item_number_pro
LEFT JOIN user AS p ON p.email = b.email
LEFT JOIN swap AS c ON acc.item_number_counter = c.item_number_counter
LEFT JOIN item AS d ON d.item_number = c.item_number_counter
LEFT JOIN user AS w ON w.email = d.email
WHERE proposer_rating IS NULL OR counterparty_rating IS NULL AND (p.email =
'email' OR w.email = 'email');
```

- Show "***List Item***", "***My items***", "***Search items***", "***Swap history***", "***Update my info***", and "***Logout***" tabs.
- Upon:
  - Click ***List Item*** button- Jump to **Listing an Item Form.**
  - Click ***My items*** button- Jump to **Display User's Available Items** task.
  - Click ***Search items*** button- Jump to **Search Items Form**.
  - Click ***Swap history*** button- Jump to **Swap History Form.**
  - Click ***Update my info*** button- Jump to **Update User Information Form**.
  - Click ***Logout*** button- Invalidate login session and jump again to the **Login** Form.

Listing an Item Form

Abstract Code:

- If user has more than two unrated swaps or more than five unaccepted swaps, then:
  - Show a message that they cannot list a new item.
- Else:
  - User selects the item type from the dropdown.

```
-- compute the number of unaccepted and unrated swaps to determine the eligibility of
the user to list a new item

-- unaccepted swaps
WITH number_of_unaccepted_swaps AS (
SELECT i.item_number,i.email, s.swap_status
```

```
FROM item i
LEFT JOIN swap s
ON i.item_number =s.item_number_counter OR i.item_number = s.item_number_pro
WHERE s.swap_status LIKE 'pending' AND i.item_number = s.item_number_counter
AND i.email='email'
),
-- unrated swaps if previously a proposer
number_of_unrated_swaps_pro AS (
SELECT i.item_number, i.email, a_s.proposer_rating
FROM item i
LEFT JOIN swap s
ON i.item_number =s.item_number_counter OR i.item_number = s.item_number_pro
LEFT JOIN accepted_swap a_s
ON  i.item_number = a_s.item_number_pro or  i.item_number =
a_s.item_number_counter
WHERE s.swap_status like 'accepted'
AND  i.item_number = a_s.item_number_pro
AND i.email='email'
-- unrated swaps if previously a counterparty
number_of_unrated_swaps_counter AS (
SELECT i.item_number, i.email, a_s.counterparty_rating
FROM item i
LEFT JOIN swap s
ON  i.item_number =s.item_number_counter OR  i.item_number =
s.item_number_pro
LEFT JOIN accepted_swap a_s
ON i.item_number = a_s.item_number_pro OR  i.item_number =
a_s.item_number_counter
WHERE s.swap_status like 'accepted'
AND  i.item_number = a_s.item_number_counter
AND i.email='email'
)
-- check the listing eligibility for the user
SELECT
IF(IF(SUM(CASE WHEN rate_and_swap_status.rating_or_swap_status  IS NULL
THEN 1 ELSE 0 END) IS NOT NULL, SUM(CASE WHEN
rate_and_swap_status.rating_or_swap_status  IS NULL THEN 1 ELSE 0 END), 0) > 2
OR COUNT(rate_and_swap_status.rating_or_swap_status like 'pending') > 5, 'YOU
CAN NOT LIST A NEW ITEM DUE TO PREVIOUS UNRATED SWAPS OR PENDING
SWAPS','YOU CAN LIST YOUR ITEM USING THE DROP DOWN MENU ') AS
listing_item_eligibility
FROM (
SELECT pro.item_number, pro.email, pro.proposer_rating AS rating_or_swap_status
FROM number_of_unrated_swaps_pro AS pro
UNION ALL
SELECT counter.item_number, counter.email, counter.counterparty_rating
from number_of_unrated_swaps_counter AS counter
UNION ALL
SELECT swap.item_number,swap.email, swap.swap_status
FROM number_of_unaccepted_swaps AS swap) AS rate_and_swap_status
```

```
SELECT DISTINCT item_type
FROM
(
SELECT item_number, 'computer_game' AS item_type
FROM computer_game
UNION
SELECT item_number, 'video_game' AS item_type
FROM video_game
UNION
SELECT item_number, 'card_game' AS item_type
FROM card_game
UNION
SELECT item_number, 'board_game' AS item_type
from board_game
UNION
SELECT item_number, 'jigsaw' AS item_type
FROM jigsaw
) AS Game_type
```

- If item type is a "Computer Game", then add additional text field for *os*

```
INSERT INTO computer_game(os) VALUES ('os');
```

- Else if the item type is a "Video Game", then add additional text fields for *platform* and *media*.

```
INSERT INTO video_game(platform,media) VALUES ('platform', 'media');
```

- Else if the item type is a "Jigsaw", then add additional text field for *piece_count*

```
INSERT INTO jigsaw(piece_count) VALUES ('piece_count');
```

- User fills out appropriate additional text fields
- User enters *title* and *description* (optional) in text fields and selects *item_condition* from drop down.

```
INSERT INTO item(title) VALUES (‘title’);
INSERT INTO item(description) VALUES (‘description’);
SELECT item_condition FROM item WHERE item_condition = ‘mint’ OR
item_condition = ‘Like New’ OR item_condition = ‘Lightly Used’ OR item_condition =
‘Moderately Used’ OR item_condition = ‘Heavily Used’ OR item_condition =
‘Damaged/Missing Parts’;
```

- Upon:
    - Click ***List Item*** button:
        - If there is an error, then:
            - List appropriate error message
        - Else:
            - Save item into item table
            - Assign index to item by the system
            - Show success message with item.*item_number*

My Items Form

Abstract Code:

- Run **Display User’s Available Items** sub-task:
    - Query about the users and their items using *email* as an identifier from item table:
        - For all the item types (games), query and display the number of items owned by the user.
        - Display the total number of items in the user’s possession.

```
SELECT
COUNT(board_game.item_number) AS ‘Board games’,
COUNT(card_game.item_number) AS ‘Card games’,
COUNT(computer_game.item_number) AS ‘Computer games’,
COUNT(jigsaw.item_number) AS ‘Jigsaw puzzles’,
COUNT(video_game.item_number) AS ‘Video Games’,
COUNT(item.item_number) AS Total
FROM
user JOIN item USING(email)
LEFT JOIN card_game USING(item_number)
LEFT JOIN board_game USING(item_number)
LEFT JOIN computer_game USING(item_number)
LEFT JOIN video_game USING(item_number)
LEFT JOIN jigsaw USING(item_number)
WHERE email = 'email'
```

- Sort the items by item number in an ascending order.
- For each item, query and display its title, item_condition, and description from item table
    - Show the first 100 characters of the description and if the number of characters for the description is more than 100, place an ellipse […].
    - Include details link on each line related to the item number.

```
SELECT item_number AS ‘item #’, item_type AS ‘Game type’, title AS ‘Title’,
item_condition AS ‘Condition’, item_description AS ‘Description’
FROM item WHERE email = ‘email’
ORDER BY item_number ASC
```

- After clicking on details link, run **Display Item Details** sub-task**:**
    - Set $selected_item to be the selected item’s item.*item_number* and jump to **View Item Form**

Searching Items Form

Abstract Code:

- Generate four radio buttons for the user to select.
- Generate text input fields for search by *keywords* option and search by *postal code* option.
- Generate integer input field for search by *miles* option.
- Generate ***search*** button at the lower right corner of the form.
- When user choose one of four search options:
    - If the user chooses search by keyword option, and input *keywords,* then click on ***search*** button, jump to **Search By Keyword** sub-task:
        - Query user input keywords against item.*title* and then item.item_*description* from item table, when there is a match in either attribute, run **Show Proposable Item** sub-task

```
SELECT Table_3.item_number AS "Item #", game_types.item_type AS 'Game type',
Title, Table_3.Item_Condition AS 'Condition', Description, Table_3.Distance
FROM
(
SELECT  item_number,Title,Item_Condition, Description,
3961 * (2 * ATAN2(SQRT(POW(SIN(RADIANS(counterparty_latitude -
proposer_latitude)/2),2) + (COS(RADIANS(counterparty_latitude)) *
COS(RADIANS(proposer_latitude))*
POW(SIN(RADIANS(counterparty_longitude - proposer_longitude)/2),2))),
SQRT(1-POW(SIN(RADIANS(counterparty_latitude - proposer_latitude)/2),2) +
(COS(RADIANS(counterparty_latitude)) *
 COS(RADIANS(proposer_latitude)) * POW(SIN(RADIANS(counterparty_longitude -
proposer_longitude)/2),2))))) AS 'Distance'
FROM (
WITH pro_address AS(
SELECT user.email, address.longitude AS pro_long, address.latitude AS pro_lat
FROM user
LEFT JOIN address USING (postal_code)
WHERE user.email LIKE ‘email’
), counter_addresses AS (
SELECT  table_1.item_number, user.email, title, item_description,item_condition,
address.longitude  AS counter_long, address.latitude AS counter_lat FROM (
SELECT  i.item_number, i.email, i.title, i.item_condition, i.item_description
FROM item i LEFT JOIN swap s ON i.item_number = s.item_number_counter OR
i.item_number = s.item_number_pro
WHERE (i.title LIKE ‘%Keyword%’ OR i.item_description LIKE ‘%Keyword%’) AND
s.swap_status IS NULL AND i.email NOT LIKE  ‘email’) AS table_1
LEFT JOIN user  ON table_1.email = user.email
LEFT JOIN address USING (postal_code)
)
SELECT counter_addresses.item_number AS item_number, counter_addresses.title
AS Title, counter_addresses.item_condition AS Item_Condition,
counter_addresses.item_description AS Description, pro_address.pro_long AS
proposer_longitude, pro_address.pro_lat AS proposer_latitude,
counter_addresses.counter_long AS counterparty_longitude,
counter_addresses.counter_lat AS counterparty_latitude
FROM pro_address, counter_addresses) AS table_2 ORDER BY item_number ASC
)
AS Table_3 LEFT JOIN
(
SELECT item_number, 'computer_game' AS item_type FROM computer_game
UNION SELECT item_number, 'video_game' AS item_type FROM video_game
UNION
SELECT item_number, 'card_game' AS item_type FROM card_game
UNION
SELECT item_number, 'board_game' AS item_type from board_game
UNION
SELECT item_number, 'jigsaw' AS item_type FROM jigsaw
) AS game_types USING (item_number);
```

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

```
SELECT Table_1.item_number AS 'Item #', game_types.item_type AS 'Game type',
Table_1.title AS Title, Table_1.item_condition AS 'Condition', Table_1.item_description
AS Description, 0 AS 'Distance'
FROM
(
SELECT i.item_number,  i.title, i.item_condition, i.item_description FROM item i
LEFT JOIN user ON i.email = user.email
LEFT JOIN swap s ON i.item_number = s.item_number_counter
WHERE s.swap_status IS NULL
AND user.postal_code = (SELECT postal_code FROM user WHERE user.email LIKE
‘email’) AND user.email NOT LIKE ‘email’ ORDER BY item_number ASC
) As Table_1
LEFT JOIN ( SELECT item_number, 'computer_game' AS item_type
FROM computer_game UNION SELECT item_number, 'video_game' AS item_type
FROM video_game UNION SELECT item_number, 'card_game' AS item_type
FROM card_game UNION
SELECT item_number, 'board_game' AS item_type FROM board_game
UNION SELECT item_number, 'jigsaw' AS item_type
FROM jigsaw) AS game_types USING (item_number);
```

- If the user chooses with X miles of me search options, user input *miles*, then click on ***search***, jump to **Search Within Miles** sub-task.
  - With user input *miles,* query address table and run **Distance Calculating** sub-task**,** flag postal code where results from distance calculation is less or equal to *miles.*
  - Then query user table where users live in these flagged postal code
  - Match all these user’s items and run **Show Proposable Item** sub-task**.**

```
SELECT Table_3.item_number AS 'Item #', game_types.item_type AS 'Game type',
Title, Table_3.Item_Condition AS 'Condition', Description, Table_3.Distance
FROM
(
SELECT  item_number,Title,Item_Condition, Description,
3961 * (2 * ATAN2(SQRT(POW(SIN(RADIANS(counterparty_latitude -
proposer_latitude)/2),2) + (COS(RADIANS(counterparty_latitude)) *
COS(RADIANS(proposer_latitude))*
POW(SIN(RADIANS(counterparty_longitude - proposer_longitude)/2),2))),
SQRT(1-POW(SIN(RADIANS(counterparty_latitude - proposer_latitude)/2),2) +
(COS(RADIANS(counterparty_latitude)) *
 COS(RADIANS(proposer_latitude)) * POW(SIN(RADIANS(counterparty_longitude -
proposer_longitude)/2),2))))) AS 'Distance'
FROM ( WITH pro_address AS(
SELECT user.email, address.longitude AS pro_long, address.latitude AS pro_lat
FROM user LEFT JOIN address USING (postal_code)
WHERE user.email LIKE  ‘email’), counter_addresses AS (SELECT
table_1.item_number, user.email, title, item_description,item_condition,
address.longitude  AS counter_long, address.latitude AS counter_lat
FROM (SELECT  i.item_number, i.email, i.title, i.item_condition, i.item_description
FROM item i
LEFT JOIN swap s ON i.item_number = s.item_number_counter OR i.item_number =
s.item_number_pro WHERE s.swap_status IS NULL
AND i.email NOT LIKE  ‘email’) AS table_1
LEFT JOIN user  ON table_1.email = user.email
LEFT JOIN address USING (postal_code)
)
SELECT counter_addresses.item_number AS item_number, counter_addresses.title
AS Title, counter_addresses.item_condition AS Item_Condition,
counter_addresses.item_description AS Description, pro_address.pro_long AS
proposer_longitude, pro_address.pro_lat AS proposer_latitude,
counter_addresses.counter_long AS counterparty_longitude,
counter_addresses.counter_lat AS counterparty_latitude
FROM pro_address, counter_addresses) AS table_2
ORDER BY item_number ASC
)
AS Table_3 LEFT JOIN
(SELECT item_number, 'computer_game' AS item_type
FROM computer_game UNION
SELECT item_number, 'video_game' AS item_type FROM video_game
UNION SELECT item_number, 'card_game' AS item_type
FROM card_game UNION
SELECT item_number, 'board_game' AS item_type
FROM board_game
UNION
SELECT item_number, 'jigsaw' AS item_type
FROM jigsaw) AS game_types
USING (item_number)
WHERE Table_3.Distance <  ‘Within X Miles’;
```

- If the user chooses the search by postal code option, input *postal code*, then click on ***search***, jump to **Search By Postal Codes** sub-task.
    - User input *postal code,* find all other users who live in *postal code*, match all these user’s items and run **Show Proposable Item** sub-task

```
SELECT Table_3.item_number AS 'Item #', game_types.item_type AS 'Game type',
Title, Table_3.Item_Condition AS 'Condition', Description, Table_3.Distance
FROM
(
SELECT  item_number,Title,Item_Condition, Description,
3961 * (2 * ATAN2(SQRT(POW(SIN(RADIANS(counterparty_latitude -
proposer_latitude)/2),2) + (COS(RADIANS(counterparty_latitude)) *
COS(RADIANS(proposer_latitude))*
POW(SIN(RADIANS(counterparty_longitude - proposer_longitude)/2),2))),
SQRT(1-POW(SIN(RADIANS(counterparty_latitude - proposer_latitude)/2),2) +
(COS(RADIANS(counterparty_latitude)) *
 COS(RADIANS(proposer_latitude)) * POW(SIN(RADIANS(counterparty_longitude -
proposer_longitude)/2),2))))) AS 'Distance'
FROM (WITH pro_address AS(
SELECT user.email, address.longitude AS pro_long, address.latitude AS pro_lat
FROM user LEFT JOIN address USING (postal_code)
WHERE user.email LIKE ‘email’
), counter_addresses AS (
SELECT  table_1.item_number, user.email, title,
item_description,item_condition,address.postal_code, address.longitude  AS
counter_long, address.latitude AS counter_lat
FROM (
SELECT  i.item_number, i.email, i.title, i.item_condition, i.item_description FROM item
i
LEFT JOIN swap s ON i.item_number = s.item_number_counter OR i.item_number =
s.item_number_pro
WHERE s.swap_status IS NULL AND i.email NOT LIKE  ‘email’
)
AS table_1
LEFT JOIN user ON table_1.email = user.email
LEFT JOIN address USING (postal_code)
WHERE address.postal_code LIKE ‘Postal_Code’
)
SELECT counter_addresses.item_number AS item_number, counter_addresses.title
AS Title,
counter_addresses.item_condition AS Item_Condition,
counter_addresses.item_description AS Description,
pro_address.pro_long AS proposer_longitude, pro_address.pro_lat AS
proposer_latitude,
counter_addresses.counter_long AS counterparty_longitude,
counter_addresses.counter_lat AS counterparty_latitude
FROM pro_address, counter_addresses) AS table_2
ORDER BY item_number ASC
)
AS Table_3
LEFT JOIN
(
SELECT item_number, 'computer_game' AS item_type
FROM computer_game
UNION
SELECT item_number, 'video_game' AS item_type
FROM video_game
UNION
SELECT item_number, 'card_game' AS item_type
```

```
FROM card_game
UNION
SELECT item_number, 'board_game' AS item_type
FROM board_game
UNION
SELECT item_number, 'jigsaw' AS item_type
FROM jigsaw
) AS game_types
USING (item_number)
```

View Items Form

Abstract Code:

- When the user enters the **View Items Form** from another form, it will pass the $selected_item session variable.
  - Look up the selected item in the item table such that item.*item_number* is equal to $selected_item. Using this record, display:
    - item.*title*, item.*description*, item.*condition*, item.item_*swap_status*, and item.*item_type*
    - If item.*item_type* is ‘computer_game’: display computer_game.*os*
    - If item.*item_type* is ‘video_game’: display video_game.*platform* and video_game.*media*
    - If item.*item_type* is ‘jigsaw’: display jigsaw.*piece_count*
    - Use the item table to look up the item’s owner; if user.*user_email* is equal to $email, it belongs to the current user. If it does not belong to the current user:

```
SELECT table_1.item_number,table_1.title AS Title, Game_type.item_type AS 'Game
type', video_game.platform_type AS platform, video_game.media AS Media,
table_1.item_condition AS 'Condition', jigsaw.piece_count AS 'Number of pieces',
computer_game.os AS 'OS'
FROM (
SELECT *
FROM item i
LEFT JOIN swap ON i.item_number = swap.item_number_pro
OR i.item_number = swap.item_number_counter
WHERE i.item_number = 19
AND swap.swap_status IS NULL
AND i.email = ‘email’) AS table_1
LEFT JOIN
(
SELECT item_number, 'computer_game' AS item_type
FROM computer_game
UNION
SELECT item_number, 'video_game' AS item_type
FROM video_game
UNION
SELECT item_number, 'card_game' AS item_type
FROM card_game
UNION
SELECT item_number, 'board_game' AS item_type
FROM board_game
UNION
SELECT item_number, 'jigsaw' AS item_type
FROM jigsaw
) AS Game_type
USING (item_number)
LEFT JOIN video_game
USING (item_number)
LEFT JOIN computer_game
USING (item_number)
LEFT JOIN card_game
USING (item_number)
LEFT JOIN board_game
USING (item_number)
LEFT JOIN jigsaw
USING (item_number);
```

- Calculate and display distance to the current user by looking up:

```
SELECT email, table_2.item_number AS 'Item #',table_2.title AS Title,item_type AS
'Game type',
platform_type AS Platform, media AS media, item_condition AS 'Condition',
piece_count AS 'Number of Pieces', os AS OS, city AS City, state AS State,
postal_code AS 'Postal code',
3961 * (2 * ATAN2(SQRT(POW(SIN(RADIANS(counterparty_latitude -
proposer_latitude)/2),2) + (COS(RADIANS(counterparty_latitude)) *
COS(RADIANS(proposer_latitude))*
POW(SIN(RADIANS(counterparty_longitude - proposer_longitude)/2),2))),
SQRT(1-POW(SIN(RADIANS(counterparty_latitude - proposer_latitude)/2),2) +
(COS(RADIANS(counterparty_latitude)) *
 COS(RADIANS(proposer_latitude)) * POW(SIN(RADIANS(counterparty_longitude -
proposer_longitude)/2),2))))) AS 'Distance'
-- counterparty_longitude, counterparty_latitude,proposer_longitude,
proposer_latitude
FROM
(
WITH proposer AS
(
SELECT address.longitude AS proposer_longitude, address.latitude AS
proposer_latitude
FROM user
LEFT JOIN address USING (postal_code)
WHERE user.email = ‘email’
),
counter AS
(
SELECT *
FROM (
SELECT *
-- i.item_number,i.title
FROM item i
LEFT JOIN swap ON i.item_number = swap.item_number_pro
OR i.item_number = swap.item_number_counter
LEFT JOIN user USING (email)
LEFT JOIN address USING (postal_code)
WHERE i.item_number = 19
AND swap.swap_status IS NULL
AND i.email = ‘email’
) AS table_1
LEFT JOIN
(
SELECT item_number, 'computer_game' AS item_type
FROM computer_game
UNION
SELECT item_number, 'video_game' AS item_type
FROM video_game
UNION
SELECT item_number, 'card_game' AS item_type
FROM card_game
UNION
SELECT item_number, 'board_game' AS item_type
FROM board_game
UNION
SELECT item_number, 'jigsaw' AS item_type
FROM jigsaw
) AS Game_type
USING (item_number)
LEFT JOIN video_game
USING (item_number)
LEFT JOIN computer_game
USING (item_number)
LEFT JOIN card_game
USING (item_number)
LEFT JOIN board_game
USING (item_number)
LEFT JOIN jigsaw
USING (item_number)
)
SELECT counter.email, counter.item_number,counter.title, counter.item_type,
counter.platform_type, counter.media, counter.item_condition, counter.piece_count,
counter.os, counter.city, counter.state, counter.postal_code,
counter.longitude AS counterparty_longitude, counter.latitude AS
counterparty_latitude, proposer.proposer_longitude AS proposer_longitude,
proposer.proposer_latitude AS proposer_latitude
FROM counter, proposer
) AS table_2;
-- compute rating
SELECT w.email,(SUM(proposer_rating) + SUM(counterparty_rating)) /
(COUNT(proposer_rating) + COUNT(counterparty_rating)) AS total_rating
FROM accepted_swap AS acc
LEFT JOIN swap AS a ON acc.item_number_pro = a.item_number_pro
LEFT JOIN item AS b ON b.item_number = a.item_number_pro
LEFT JOIN user AS p ON p.email = b.email
LEFT JOIN swap AS c ON acc.item_number_counter = c.item_number_counter
LEFT JOIN item AS d ON d.item_number = c.item_number_counter
LEFT JOIN user AS w ON w.email = d.email
WHERE p.email = ‘email’’ OR w.email = ‘email’;
```

- The item’s related user and that user’s user.*latitude* and user.*longitude*
- Look up the current user’s user.*latitude* and user.*longitude* using the $email session variable.
- Highlight the calculated distance based on the following rules:
  - 0-25 miles: green
  - 25-50 miles: yellow
  - 50-100 miles: orange
  - 100+ miles: red

- Display the item owner’s user.*nickname*
- Looking at the current user’s swaps, if they have less than or equal to 2 unrated swaps or less than or equal to5 unaccepted swaps and the current item is available for swapping:

```
-- compute the number of unaccepted and unrated swaps to determine the eligibility of
```

```
the user to list a new item

-- unaccepted swaps
WITH number_of_unaccepted_swaps AS (
SELECT i.item_number,i.email, s.swap_status
FROM item i
LEFT JOIN swap s
ON i.item_number =s.item_number_counter OR i.item_number = s.item_number_pro
WHERE s.swap_status LIKE 'pending' AND i.item_number = s.item_number_counter
AND i.email='email'
),
-- unrated swaps if previously a proposer
number_of_unrated_swaps_pro AS (
SELECT i.item_number, i.email, a_s.proposer_rating
FROM item i
LEFT JOIN swap s
ON i.item_number =s.item_number_counter OR i.item_number = s.item_number_pro
LEFT JOIN accepted_swap a_s
ON  i.item_number = a_s.item_number_pro or  i.item_number =
a_s.item_number_counter
WHERE s.swap_status like 'accepted'
AND  i.item_number = a_s.item_number_pro
AND i.email='email'
-- unrated swaps if previously a counterparty
number_of_unrated_swaps_counter AS (
SELECT i.item_number, i.email, a_s.counterparty_rating
FROM item i
LEFT JOIN swap s
ON  i.item_number =s.item_number_counter OR  i.item_number =
s.item_number_pro
LEFT JOIN accepted_swap a_s
ON i.item_number = a_s.item_number_pro OR  i.item_number =
a_s.item_number_counter
WHERE s.swap_status like 'accepted'
AND  i.item_number = a_s.item_number_counter
AND i.email='email'
)
-- check the listing eligibility for the user
SELECT
IF(IF(SUM(CASE WHENrate_and_swap_status.rating_or_swap_status  IS NULL
THEN 1 ELSE 0 END) IS NOT NULL, SUM(CASE WHEN
rate_and_swap_status.rating_or_swap_status  IS NULL THEN 1 ELSE 0 END), 0) > 2
OR COUNT(rate_and_swap_status.rating_or_swap_status like 'pending') > 5, 'YOU
CAN NOT LIST A NEW ITEM DUE TO PREVIOUS UNRATED SWAPS OR PENDING
SWAPS','YOU CAN LIST YOUR ITEM USING THE DROP DOWN MENU ') AS
listing_item_eligibility
```

```
FROM (
SELECT pro.item_number, pro.email, pro.proposer_rating AS rating_or_swap_status
FROM number_of_unrated_swaps_pro AS pro
UNION ALL
SELECT counter.item_number, counter.email, counter.counterparty_rating
FROM number_of_unrated_swaps_counter AS counter
UNION ALL
SELECT swap.item_number,swap.email, swap.swap_status
FROM number_of_unaccepted_swaps AS swap) AS rate_and_swap_status
```

- If the user clicks on ***Propose Swap*** button, run the **Propose a Swap Form**
- If the user clicks ***Exit*** button, clear the $selected_item variable and return to the previous for

Propose a Swap Form

Abstract Code:

- Run **Display Desired and Proposed Items** sub-task:
    - Read desired item from swap table and display on form.

```
SELECT desired_item.item_number, proposed_item.item_number,
desired_item.item_title , proposed_item.item_title
FROM swap
INNER JOIN item proposed_item ON swap.item_number_pro =
proposed_item.item_number
LEFT JOIN item desired_item ON swap.item_number_counter =
desired_item.item_number
```

-
    - Query to get Proposer's and Counterparty's address.*latitude* and address.*longitude* values.

```
WITH users_cte AS (
SELECT
desired.email AS COUNTERPARTY
,proposed.email AS PROPOSER
FROM swap
LEFT JOIN item desired ON swap.item_number_counter = item.item_number
LEFT JOIN item proposed ON swap.item_number_pro = item.item_number
)
, latlong AS ( SELECT users_cte.COUNTERPARTY, users_cte.PROPOSER,
counterparty_address.latitude AS counterparty_latitude,
counterparty_address.longitude AS counterparty_longitude,
proposer_address.latitude AS proposer_latitude, proposer_address.longitude AS
proposer_longitude
FROM users_cte
LEFT JOIN user counterparty_user ON counterparty_user.email =
users_cte.COUNTERPARTY
LEFT JOIN user proposer_user ON proposer_user.email = users_cte.PROPOSER
LEFT JOIN address counterparty_address ON counterparty_user.postal_code =
counterparty_address.postal_code
LEFT JOIN address proposer_address ON proposer_user.postal_code =
proposer_address.postal_code
)
SELECT
3961 * (2 * ATAN2(SQRT(POW(SIN(RADIANS(counterparty_latitude -
proposer_latitude)/2),2) + (COS(RADIANS(counterparty_latitude)) *
COS(RADIANS(proposer_latitude)) * POW(SIN(RADIANS(counterparty_longitude -
proposer_longitude)/2),2))), SQRT(1-POW(SIN(RADIANS(counterparty_latitude -
proposer_latitude)/2),2) + (COS(RADIANS(counterparty_latitude)) *
COS(RADIANS(proposer_latitude)) * POW(SIN(RADIANS(counterparty_longitude -
proposer_longitude)/2),2))))) AS DISTANCE
FROM latlong
```

        - If distance between Proposer and Counterparty is >= 100.00 miles, display a warning message containing the distance at the top of the form in red.
    - Query item table to get all of the User's associated available items and display the following information within the form: item.*item_number*, item.*item_type*, item.*title*, item.item_*condition*.
    - Order list of items by ascending item.*item_number*

```
SELECT item_number, item_type, title ,item_condition FROM item WHERE email =
'email'
ORDER BY item_number
```

- Run **Proposed Item Choice and Confirmation** sub-task:
    - Show ***Select*** radio button for each item displayed.
        - If ***Select*** button is clicked, populate the radio button and show ***Confirm*** button.
            - If ***Confirm*** button is clicked, generate a date stamp and insert into the swap table.

```
INSERT INTO swap(item_number_pro, item_number_counter, swap_status,
propose_date) VALUES (proposed_item, desired_item, 'PENDING',
CURRENT_DATE);
```

        - Show a message letting the user know swap has started and generate an ***OK*** button.
        - If the user clicks on the button, take the user back to **Main Menu Form**.
    - Else, if no items are chosen OR all previously selected items are unselected by the user, do not show the ***Confirm*** button

Accept/Reject Swaps Form

Abstract Code:

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

```
SELECT swap.propose_date ,swap.item_number_pro ,swap.item_number_counter
,user.nickname
FROM item
LEFT JOIN swap ON item.item_number = swap.item_number_pro
LEFT JOIN user ON item.email = user.email
WHERE email = ‘email’
```

        - Calculate average of all ratings associated with proposer from accepted_swap table

```
SELECT
AVG(accepted_swap.counterparty_rating) AS AVG_COUNTERPARTY_RATING
FROM item
LEFT JOIN accepted_swap ON item.item_number =
accepted_swap.item_number_pro
WHERE email = ‘email’
```

        - Distance
            - Read proposer’s address.*longitude* and address.*latitude.*

- Read counterparty's address.*longitude* and address.*latitude.*
- Calculate the distance between using above info.
- Return and display the distance

```
WITH users_cte AS (
SELECT
desired.email AS COUNTERPARTY
,proposed.email AS PROPOSER
FROM swap
LEFT JOIN item desired ON swap.item_number_counter = item.item_number
LEFT JOIN item proposed ON swap.item_number_pro = item.item_number
)
, latlong AS (
SELECT users_cte.COUNTERPARTY, users_cte.PROPOSER ,
counterparty_address.latitude AS counterparty_latitude,
counterparty_address.longitude AS counterparty_longitude,
proposer_address.latitude AS proposer_latitude, proposer_address.longitude AS
proposer_longitude
FROM users_cte
LEFT JOIN user counterparty_user ON counterparty_user.email =
users_cte.COUNTERPARTY
LEFT JOIN user proposer_user ON proposer_user.email = users_cte.PROPOSER
LEFT JOIN address counterparty_address ON counterparty_user.postal_code =
counterparty_address.postal_code
LEFT JOIN address proposer_address ON proposer_user.postal_code =
proposer_address.postal_code
)
SELECT
3961 * (2 * ATAN2(SQRT(POW(SIN(RADIANS(counterparty_latitude -
proposer_latitude)/2),2) + (COS(RADIANS(counterparty_latitude)) *
COS(RADIANS(proposer_latitude)) * POW(SIN(RADIANS(counterparty_longitude -
proposer_longitude)/2),2))), SQRT(1-POW(SIN(RADIANS(counterparty_latitude -
proposer_latitude)/2),2) + (COS(RADIANS(counterparty_latitude)) *
COS(RADIANS(proposer_latitude)) * POW(SIN(RADIANS(counterparty_longitude -
proposer_longitude)/2),2))))) AS DISTANCE
FROM latlong
```

- ■ ***Proposed Item*** link
  - If ***Proposed Item*** link is clicked – Jump to **View Items Form**
  - Display ***Accept*** and ***Reject*** button for each available swap.

- Run **Accept/Reject** sub-task:
  - ○ If an ***Accept*** button is clicked:
  - ○ Get User's user.*email* and user.*first_name* from user table and display within form or separate dialog box

```
SELECT email, first_name FROM user WHERE email = 'email'
```

- Check User's phone.*phone_number* value from phone table:
    - If null, display message that phone is unavailable.
    - Else, get Sharing Option:
        - If Sharing Option = "Allowed", display phone.*phone_number* and phone.*phone_type* type on form.

```
SELECT phone_number, is_shared, phone_type FROM phone WHERE email =
'email'
```

- Generate a date stamp upon clicking ***Accept*** button and insert accepted_swap.*accepted_date* into the accepted_swap table.

```
INSERT INTO
accepted_swap(item_number_pro,item_number_counter,accepted_date) VALUES
(item_number_pro,item_number_counter,CURRENT_DATE);
```

- Update swap.*swap_status* to "accepted."

```
UPDATE swap SET swap_status = 'accepted' WHERE item_number_pro =
item_number_pro
AND item_number_counter = item_number_counter
```

- Remove accepted_swap from display list of proposed swaps:
- If the number of items in swaps list == 0, display **Main Menu Form**.

- If ***Reject*** button is clicked:
    - Generate a date stamp to reflect swap rejected_swap.*rejected_date* and insert into the rejected_swap table.

```
INSERT INTO rejected_swap(item_number_pro,item_number_counter,rejected_date)
VALUES (item_number_pro,item_number_counter,CURRENT_DATE);
```

- Update swap.*swap_status* to "rejected."

```
UPDATE swap SET swap_status = 'rejected' WHERE item_number_pro =
item_number_pro
AND item_number_counter = item_number_counter;
```

        - Remove rejected_swap from display list of proposed swaps.
        - If the number of items swaps list == 0, display **Main Menu Form**.

Rate Swaps Form

Abstract Code:

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

Show Unrated Swap

- For current user session, query item table where item's owner email is current user email where item.*item_swap_status* is "accepted" AND corresponding rating attributes is null
    - Then query swap where these *item_number* either show up in swap.*being_proposed* OR *swap.being_desired*
    - Then determine user's role and find counter party

```
WITH temp_tb AS (select *
FROM user u1 LEFT JOIN item i1 USING(email)
LEFT JOIN accepted_swap a1 on item_number = a1.item_number_pro
WHERE email = 'email'
UNION

SELECT *
FROM user u2 LEFT JOIN item i2 using(email)
LEFT JOIN accepted_swap a2 on item_number = a2.item_number_counter
WHERE email = 'email )

SELECT accepted_date,pro.title,con.title
FROM temp_tb, item pro, item con,user other_user
WHERE pro.item_number = temp_tb.item_number_pro and con.item_number =
temp_tb.item_number_counter
```

- If the item_number shows up in a swap as swap.*being_offered*
    - the role for this user is proposer

    - find other user's nick name by looking up the item_number in swap.*being_desired*, find the user.*nickname* of that item's owner.
- Else if item_number shows up in a swap as swap.*being_desired*
    - the role of the user is counterparty
    - find other user's nick name by looking up the item_number in swap.*being_offered*, find the user.*nickname* of that item's owner.

```
SELECT nickname FROM user LEFT JOIN item USING(email) WHERE user.email =
'email'
```

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

Rate Unrated Swap

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

```
WITH original_tb AS (select proposer_rating,counterparty_rating FROM
accepted_swap)
UPDATE accepted_swap
SET proposer_rating = CASE WHEN (counter_party IS NULL then) 'user_input'
else original_tb.propose_rating END,
counterparty_rating = CASE WHEN(counter_party IS NULL) then 'user_input'
else original_tb.counterparty END
```

Swap History Form

Abstract Code:

- Run **View Swap History** sub-task:
    - Run query on user and swap tables to derive the following and display on form for both Proposer role and Counterparty role, listed separately:
        - Total swaps proposed

        - Total received
        - Sub-totals for accepted and rejected swaps
        - % rejected
    - If % rejected >= 50.0%, highlight percentage in red.

```
- - Proposer
SELECT
SUM(CASE WHEN swap.propose_date IS NOT NULL THEN 1 ELSE 0 END) AS
SWAP_COUNT,
SUM(CASE WHEN accepted_swap.accepted_date IS NOT NULL THEN 1 ELSE 0 END)
AS ACCEPTED_COUNT,
SUM(CASE WHEN rejected_swap.rejected_Date IS NOT NULL THEN 1 ELSE 0 END)
AS REJECTED_COUNT,
SUM(CASE WHEN rejected_swap.rejected_Date IS NOT NULL THEN 1 ELSE 0 END)
/SUM(CASE WHEN swap.propose_date IS NOT NULL THEN 1 ELSE 0 END) AS
REJECTED_PCT
FROM item
INNER JOIN swap ON item.item_number = swap.item_number_pro
LEFT JOIN accepted_swap ON swap.item_number_pro =
accepted_swap.item_number_pro AND swap.item_number_counter =
accepted_swap.item_number_counter
LEFT JOIN rejected_swap ON swap.item_number_pro =
rejected_swap.item_number_pro AND swap.item_number_counter =
rejected_swap.item_number_counter
WHERE item.email = email;

- - Counterparty
SELECT
SUM(CASE WHEN swap.propose_date IS NOT NULL THEN 1 ELSE 0 END) AS
SWAP_COUNT,
SUM(CASE WHEN accepted_swap.accepted_date IS NOT NULL THEN 1 ELSE 0 END)
AS ACCEPTED_COUNT,
SUM(CASE WHEN rejected_swap.rejected_Date IS NOT NULL THEN 1 ELSE 0 END)
AS REJECTED_COUNT,
SUM(CASE WHEN rejected_swap.rejected_Date IS NOT NULL THEN 1 ELSE 0 END)
/SUM(CASE WHEN swap.propose_date IS NOT NULL THEN 1 ELSE 0 END) AS
REJECTED_PCT
FROM item
INNER JOIN swap ON item.item_number = swap.item_number_counter
LEFT JOIN accepted_swap on swap.item_number_pro =
accepted_swap.item_number_pro AND swap.item_number_counter =
accepted_swap.item_number_counter
LEFT JOIN rejected_swap ON swap.item_number_pro =
rejected_swap.item_number_pro AND swap.item_number_counter =
rejected_swap.item_number_counter
```

```
WHERE item.email = 'email';
```

- Query swap, item, user, accepted_swap, and rejected_swap table to get the User's completed swaps; For each completed swap, display the following information in a table format:
  - Swap.*propose_date*
  - accepted_swap.*accepted_date* or rejected_swap.*rejected_date*
  - Swap.*swap_status*
  - User's role
  - Proposed item title
  - Desired item title
  - Other User's nickname
  - Swap rating:

```
SELECT proposed_swap.propose_date ,accepted_swap.accepted_date ,
rejected_swap.rejected_date ,'Proposer' AS user_role , item.title , desired.title,
other_user.nickname, accepted_swap.counterparty_rating
FROM item
LEFT JOIN swap proposed_swap ON item.item_number =
proposed_swap.item_number_pro
LEFT JOIN accepted_swap
ON proposed_swap.item_number_pro = accepted_swap.item_number_pro
AND  proposed_swap.item_number_counter = accepted_swap.item_number_counter
LEFT JOIN rejected_swap
ON proposed_swap.item_number_pro = rejected_swap.item_number_pro
AND  proposed_swap.item_number_counter = rejected_swap.item_number_counter
LEFT JOIN item desired ON proposed_swap.item_number_counter =
desired.item_number
LEFT JOIN user other_user ON desired.email = other_user.email
WHERE item.email = 'email'

UNION ALL

SELECT proposed_swap.propose_date ,accepted_swap.accepted_date,
rejected_swap.rejected_date, 'Proposer' AS user_role , item.title ,desired.title,
other_user.nickname, accepted_swap.counterparty_rating
FROM item
LEFT JOIN swap proposed_swap ON item.item_number =
proposed_swap.item_number_pro
LEFT JOIN accepted_swap
ON proposed_swap.item_number_pro = accepted_swap.item_number_pro
AND  proposed_swap.item_number_counter = accepted_swap.item_number_counter
LEFT JOIN rejected_swap
```

```
ON proposed_swap.item_number_pro = rejected_swap.item_number_pro
AND  proposed_swap.item_number_counter = rejected_swap.item_number_counter
LEFT JOIN item desired ON proposed_swap.item_number_counter =
desired.item_number
LEFT JOIN user other_user ON desired.email = other_user.email
WHERE item.email = ‘email’
```

- If Swap rating is NULL, display rating mechanism.
    - Run **Rate Unrated Swaps** sub-task:
        - Once rating has been chosen, insert rating into accepted_swap table and refresh page.
        - Once refreshed, run a query for proposer_rating again and display on form.
        - Rate unrated swap by allowing the User to choose the rating they would like to give to the other user:
        - If myrole == proposer, write to accepted_swap.*counterparty_rating*
        - Else if myrole == counterparty, write to accepted_swap.*proposer_rating*
- For each swap listed in history, display a ***Detail*** link:
    - When ***Detail*** link is clicked, display **Swap Details Form** for the swap associated with the link in the table.

Swap Details Form

Abstract Code:

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

        - Look up the current user's user.*latitude* and user.*longitude* using the $email session variable.
        - Display the distance calculated using the above variables.
            - Display user.*user_email* of user who owns item that is being_desired
    - Display item.*title* and item.*description* of item being_desired
    - If user is not the owner of item being_desired:
        - The being_desired item's related user and that user's user.*latitude* and user.*longitude*
        - Look up the current user's user.*latitude* and user.*longitude* using the $email session variable.
        - Display the distance calculated using the above variables.
    - Display swap.*swap_status* for the current swap
    - If swap.*swap_status* is 'Accepted':
        - display accepted_swap.*accepted_date*
    - else if swap.*swap_status* 'Rejected':
        - display rejected_swap.*rejected_date*
    - else display swap.*propose_date*
    - If swap.*swap_status* == 'Accepted'
        - If user is the owner of item being_offered:
            - Look up accepted_swap.*proposer_rating*.
        - If value is not null, display.
        - Else if value is null:
            - Take user's input for *swap_rating*. In swap table, update accepted_swap.*proposer_rating* per the user input.
            - Return to **Swap Details Form** (refresh).
        - Else if user is the owner of item.*being_desired*:
            - If swap.*swap_status* == 'Accepted'
                - If user is the owner of item being_offered:
                - Look up accepted_swap.*counterparty_rating*. If value is not null, display. If value is null:
                - Take user's input for *swap_rating*. In swap table, update accepted_swap.*counterparty_rating* per the user input.
            - Return to **Swap Details Form** (refresh).

```
WITH users_cte AS (
SELECT desired.email AS COUNTERPARTY, proposed.email AS PROPOSER
FROM swap
LEFT JOIN item desired ON swap.item_number_counter = desired.item_number
LEFT JOIN item proposed ON swap.item_number_pro = proposed.item_number
WHERE swap.item_number_pro = swap_proposed_item
AND  swap.item_number_counter = swap_counter_item
)
, latlong AS (
SELECT users_cte.COUNTERPARTY, users_cte.PROPOSER,
counterparty_address.latitude AS counterparty_latitude,
counterparty_address.longitude AS counterparty_longitude,
proposer_address.latitude AS proposer_latitude, proposer_address.longitude AS
proposer_longitude

FROM users_cte
LEFT JOIN user counterparty_user ON counterparty_user.email =
users_cte.COUNTERPARTY
LEFT JOIN user proposer_user ON proposer_user.email = users_cte.PROPOSER
LEFT JOIN address counterparty_address ON counterparty_user.postal_code =
counterparty_address.postal_code
LEFT JOIN address proposer_address ON proposer_user.postal_code =
proposer_address.postal_code
)
,distance AS (
SELECT
3961 * (2 * ATAN2(SQRT(POW(SIN(RADIANS(counterparty_latitude -
proposer_latitude)/2),2) + (COS(RADIANS(counterparty_latitude)) *
COS(RADIANS(proposer_latitude)) * POW(SIN(RADIANS(counterparty_longitude -
proposer_longitude)/2),2))), SQRT(1-POW(SIN(RADIANS(counterparty_latitude -
proposer_latitude)/2),2) + (COS(RADIANS(counterparty_latitude)) *
COS(RADIANS(proposer_latitude)) * POW(SIN(RADIANS(counterparty_longitude -
proposer_longitude)/2),2)))))
AS DISTANCE
FROM latlong
)
SELECT item.email, item.title, item.item_description, swap.swap_status ,
swap.propose_date, accepted_swap.accepted_date,
accepted_swap.proposer_rating, accepted_swap.counterparty_rating,
distance.distance
FROM swap
INNER JOIN item ON item.item_number = swap.item_number_pro
LEFT JOIN accepted_swap ON swap.item_number_pro =
accepted_swap.item_number_pro AND swap.item_number_counter =
accepted_swap.item_number_counter
LEFT JOIN distance ON 1=1
WHERE swap.item_number_pro = swap_proposed_item
AND  swap.item_number_counter = swap_counter_item
```

```
WITH original_tb AS (select proposer_rating,counterparty_rating FROM
accepted_swap)
UPDATE accepted_swap
SET counterparty_rating = CASE WHEN counterparty_rating IS NULL then
'user_input'
else original_tb.counterparty END
```

- If the user has no items involved in swaps, display "No swaps found." Return to **Main Menu Form**

Update User Information Form

Abstract Code:

When user clicks the ***Update my info*** button from the **Main Menu Form**, run:

- **Display User Information** sub-task:
    - Query the user table such that email is equal to user.*user_email* and display the user's user.*first_name*, user.*last_name*, user.*nickname*, and user.*email*

```
SELECT first_name, last_name, nickname, email FROM user WHERE email =
'email';
```

- If user has_a phone find the phone using the user.*email* (equal to 'email') in the phone table.
    - Display the phone.*phone_number*, phone.*phone_type*, and phone.*is_shared*

```
SELECT phone_number, phone_type, is_shared FROM phone WHERE email =
'email';
```

- If ***Delete User Profile*** button is pushed*:*
    - Find and delete row for this user based on user.*email* being equal to 'email' in the user table.
    - Clear 'email' session variable.
    - Return to **Login Form**

```
DELETE FROM user WHERE email = 'email';
```

- Display the ***Change User Information*** button. If clicked, run the **Updating User Information** sub-task:
  - While no buttons are pressed, do nothing.
  - If ***Update Email button*** is pressed:
    - take the user's input
    - replace user.*email* with the input ('new_email') in the user table.
    - Update the 'email' session variable to reflect this change as well ('email' == 'new_email').

```
UPDATE user SET email = 'new_email' WHERE email = 'email';
```

    - Run **Display User Information** sub-task**.**
  - If ***Update Name Preferences*** button is pressed*:*
    - take the user's input for the three name fields.
    - Find the user in the user table based on user.*email* and update user.*first_name*, user.*last_name*, and user.*nickname* in user table with the user's input ('new_first_name', 'new_last_name', and 'new_nickname', respectively).

```
UPDATE user SET first_name = 'new_first_name' WHERE email = 'email';
UPDATE user SET last_name = 'new_last_name' WHERE email = 'email';
UPDATE user SET nickname = 'new_nickname' WHERE email = 'email';
```

    - Run **Display User Information** sub-task**.**
  - If ***Update Phone Preferences*** button is pressed***:***
    - Take the user's input for phone.*phone_number*, phone.*phone_type*, and phone.*is_shared*.
    - Find the user's phone in the phone table using user.*user_email* and update phone.*phone_number*, phone.*phone_type*, phone.*is_shared* in the phone table with the user's input ('new_phone_number', 'new_phone_type', and 'new_is_shared', respectively).

```
UPDATE phone SET phone_number = ‘new_phone_number’ WHERE email = ‘email’;
UPDATE phone SET phone_type = ‘new_phone_type’ WHERE email = ‘email’;
UPDATE phone SET is_shared = ‘new_is_shared’ WHERE email = ‘email’;
```

        - Run **Display User Information** sub-task**.**
- If ***Exit*** button is clicked, return to **Main Menu Form**.