Table of Contents

Login

Abstract Code

- When the user clicks the ***Register*** button, navigate to the **Register** form.
- User enters email and password input fields.
- If data validation is successful for both email and password input fields, then:
    - When the ***Login*** button is clicked:

      ```
      SELECT password FROM `User` WHERE email='$Email',
      ```

        - If the User record is found but the user.password != '$Password':
            - Go back to the **Login** form, with an error message.
        - Else:
            - Store login information ('$Email')
            - Go to the **View Profile** form.
- Else *email* and *password* input fields are invalid, display **Login** form, with error message

Register

Abstract Code

- In the **Register** form, the User is prompted to enter the details below :-

1. *First Name*	6. *Password*
2. *LastName*	7. *Address Type*
3. *Nickname*	8. *Phone Number* [Optional field]
4. *Email*	9. Share
5. *Postal Code*	10. Phone Type

  Other than *Phone number*, all other fields are mandatory.

  NOTE: The user will only select their postal code

- User clicked ***Register*** button from the **Login Menu,** display **Register** form
- If any of the mandatory fields are not entered, the User is prompted to enter the missing details.
- Check if email already exists:

```
SELECT email FROM `User` WHERE email = '$Email'
```

- If query is not empty display error message: Email already exists
- Else, check if phone already exists

```
SELECT phone_number FROM `Phone` WHERE
phone_number = '$PhoneNumber'
```

- If query is not empty display error message: Phone number already exists
- Else, run the **Register User** task

```
BEGIN TRAN
INSERT INTO `User` (email, password,  first_name, last_name,
nickname,postal_code, phone_number)
VALUES ('$Email', '$Password', '$FirstName', '$LastName', '$Nickname',
'$PostalCode', '$PhoneNumber');
```

- If PhoneNumber entered:

```
INSERT INTO `Phone` (phone_number, share)
VALUES ('$PhoneNumber', '$Share');
COMMIT TRAN
```

  - If '$PhoneType' = 'HomePhone'

```
INSERT INTO `HomePhone` (phone_number)
```

```
VALUES ('$PhoneNumber');
COMMIT TRAN
```

  - If '$PhoneType' = 'WorkPhone'

```
INSERT INTO `WorkPhone` (phone_number)
VALUES ('$PhoneNumber');
COMMIT TRAN
```

  - If '$PhoneType' = 'MobilePhone'

```
INSERT INTO `HomePhone` (phone_number)
VALUES ('$PhoneNumber');
COMMIT TRAN
```

- Else

```
COMMIT TRAN
```

Update User Information

Abstract Code

- User clicked ***Update User Information*** button from the **Main Menu,** display **Update User Information** form
- Check if the user has any unrated swaps or unapproved and display an error message if there is at least one. Return to **Main Menu.**

```
SELECT
        COUNT(swapped_item) AS pending_swaps
FROM
(SELECT
        proposed_item_number AS swapped_item,
FROM `Swap`
INNER JOIN `Item` AS ProposedItem ON ProposedItem.item_number =
proposed_item_number
WHERE ProposedItem.user_email = '$Email' AND counterparty_rating IS NULL OR
status = 0

UNION ALL

SELECT
        desired_item_number AS swapped_item
FROM `Swap`
INNER JOIN `Item` AS DesiredItem ON DesiredItem.item_number =
desired_item_number
WHERE DesiredItem.user_email = '$Email' AND proposer_rating IS NULL OR status
= 0) AS UserSwaps
```

- Retrieve and display user information using email as identifier and display information. Run **View User Profile** task:

```
SELECT email, password, nickname, city, first_name, last_name,
`User`.postal_code, state, city, `User`.phone_number, phone_type, share
FROM `User` WHERE email='$Email'
INNER JOIN `Address` ON `Address`.postal_cdoe = `User`.postal_code
INNER JOIN `Phone` ON `Phone`.phone_number = `User`.phone_number
INNER JOIN (
        SELECT phone_number, 'Home' as phone_type FROM `HomePhone`
        SELECT phone_number, 'Work' as phone_type FROM `WorkPhone`
        SELECT phone_number, 'Mobile' as phone_type FROM `MobilePhone`
) AS PhoneTypes ON PhoneTypes.phone_number = `User`.phone_number
```

- If needed, User should be given an option to edit and update the profile (email is not editable).
- Once the ***Update*** profile button is clicked, all the **Registration** validations will apply. (i.e. Mandatory fields and Email and Phone validation)
- Update user information to the database and refresh page.

```
BEGIN TRAN
UPDATE `User`
SET password = '$Password', nickname = '$Nickname', postal_code = '$PostalCode',
first_name = '$FirstName', last_name = '$LastName', phone_number =
'$PhoneNumber'
```

- If PhoneNumber entered:

```
UPDATE `Phone`
SET phone_number = '$PhoneNumber', share = '$Share'
```

  - If '$PhoneType' changed. Delete previous entry from the respective phone type and insert again:
  - E.g.

```
DELETE FROM `HomePhone` WHERE phone_number =  '$PhoneNumber'
```

```
INSERT INTO `WorkPhone` (phone_number)
VALUES ('$PhoneNumber');
```

```
COMMIT TRAN
```

- Else

```
COMMIT TRAN
```

Main Menu

Abstract Code

- In the **MainMenu** Page, the User's *first name* and *last name* need to be displayed.

```
SELECT
CONCAT(first_name, '  ', last_name ) AS name,
FROM `User` WHERE email='$Email''
```

- Run the **Get Unrated Swaps** task for the logged user: query information about the unrated swaps and display number.

```
SELECT
        COUNT(swapped_item) AS unrated_swaps
FROM
(SELECT
        proposed_item_number AS swapped_item,
FROM `Swap`
INNER JOIN `Item` AS ProposedItem ON ProposedItem.item_number =
proposed_item_number
WHERE ProposedItem.user_email = '$Email' AND counterparty_rating IS NULL

UNION ALL

SELECT
        desired_item_number AS swapped_item
FROM `Swap`
INNER JOIN `Item` AS DesiredItem ON DesiredItem.item_number =
desired_item_number
WHERE DesiredItem.user_email = '$Email' AND proposer_rating IS NULL) AS
USwaps
```

- Run the **Get Unaccepted Swaps** task for the logged user and display number.

```
SELECT
        COUNT(item_number) AS unaccepted_swaps
FROM `Swap`
INNER JOIN `Item` ON item_number = desired_item_number
WHERE user_email = '$Email' AND status IS NULL
```

- Users have an option to logout by clicking the ***logout*** button.

- Users have an option to view ***unrated swaps***, if any, by clicking the link under unrated swaps.
- Similarly, ***unaccepted swaps*** can be also viewed by clicking the link under the unrated swaps.
- My ratings will be displayed based on the calculations. Running the **Calculate Rating** task for the logged user:

```
SELECT
        ROUND(AVG(rating),2) AS rating,
        user_email
FROM
(SELECT
        proposer_rating AS rating,
        user_email
FROM `Swap`
INNER JOIN `Item` AS ProposedItem ON ProposedItem.item_number =
proposed_item_number
WHERE ProposedItem.user_email = '$Email' AND `proposer_rating` IS NOT NULL
UNION ALL
SELECT
        counterparty_rating AS rating,
        user_email
FROM `Swap`
INNER JOIN `Item` AS DesiredItem ON DesiredItem.item_number =
desired_item_number
WHERE DesiredItem.user_email = '$Email' AND `counterparty_rating` IS NOT NULL)
AS Ratings
```

- If the ***List Item*** button is clicked, User will have an option to list a new item.
- To look for all the items listed by the User, ***My Items*** button needs to be clicked.
- Users have the option to search items by clicking the ***Search Items*** button.
- To view swap history, the ***Swap History*** button needs to be clicked.
- The user may navigate to the **Update User Information** form by clicking ***Update my Info***.

Listing An Item

Abstract Code

- User clicked ***List Item*** button from the **MainMenu**
- Run the **tasks Get Unrated Swaps** and **Get Unaccepted Swaps** in the **MainMenu**
    - If Unaccepted swaps > 5 or Unrated swaps > 2, return error message
    - Else: display **New Item Listing** form, run the **List Item** task: User selected *Game type ('$Game_type'), Condition ('$Condition')*, inputted *Title ('$Title'), description ('$Description'),* '$Email' is the email of current user
        - First insert the new item into \`Item\`. Lock the database until subclass item is added:

```
BEGIN TRAN
INSERT INTO `Item` (user_email, name, condition, description)
VALUES ('$Email', '$Title', '$Condition', '$Description');
```

        - If '$Game_type' = 'Board Game':

```
INSERT INTO `BoardItem` (item_number)
VALUES (SELECT MAX(item_number) FROM `Item`)
COMMIT TRAN;
```

        - Elif '$Game_type' = 'Card Game':

```
INSERT INTO `CardItem` (item_number)
VALUES (SELECT MAX(item_number) FROM `Item`)
COMMIT TRAN;
```

        - Elif '$Game_type' = 'Video Game':
            - User can input '$Platform', which can only be selected from 'Nintendo', 'PlayStation', 'Xbox' in UI dropdown;
            - User can input '$Media, which can only be selected from 'Linux', 'macOS', 'Windows' in UI dropdown;

```
INSERT INTO `VideoItem` (item_number, platform, media)
VALUES ((SELECT MAX(item_number) FROM `Item`), '$Platform', '$Media')
COMMIT TRAN;
```

        - Elif '$Game_type' = Computer Game':
            - User can input '$Platform', which can only be selected from 'Linux', 'macOS', 'Windows' in UI dropdown;

```
INSERT INTO `ComputerItem` (item_number, platform)
VALUES ((SELECT MAX(item_number) FROM `Item`), '$Platform')
COMMIT TRAN;
```

        - Elif '$Game_type' = 'Jigsaw Puzzle':
            - User can input '$Piece_count'

```
INSERT INTO `JigsawItem` (item_number, piece_count)
```

```
VALUES ((SELECT MAX(item_number) FROM `Item`), '$Piece_count')
COMMIT TRAN;
```

- Display a window showing Listing success with:

```
SELECT MAX(item_number) FROM `Item`
```

- User can continue to list items, or return to **Main Menu** by clicking ***Exit*** button

My items

Abstract Code

- User clicked ***My Items*** button from the **Main Menu,** display **My Items** form
- Get my item list table and display, add a new column to view item details:

```
SELECT item_number, game_type, name, condition, LEFT(description, 100)
FROM
(
SELECT item_number, 'Board game' AS game_type FROM `BoardItem`
UNION ALL
SELECT item_number, 'Card game' AS game_type FROM `CardItem`
UNION ALL
SELECT item_number, 'Video game' AS game_type FROM `VideoItem`
UNION ALL
SELECT item_number, 'Computer game' AS game_type FROM `ComputerItem`
UNION ALL
SELECT item_number, 'Jigsaw puzzle' AS game_type FROM `JigsawItem`
)
INNER JOIN `Item` ON item_number = `Item`.item_number
Where user_email = '$Email'
ORDER BY item_number ASC;
```

- Display count of game type from my item list:

```
SELECT game_type, COUNT(*) as type_counts FROM
(
SELECT item_number, 'Board game' AS game_type FROM `BoardItem`
UNION ALL
SELECT item_number, 'Card game' AS game_type FROM `CardItem`
UNION ALL
SELECT item_number, 'Video game' AS game_type FROM `VideoItem`
UNION ALL
SELECT item_number, 'Computer game' AS game_type FROM `ComputerItem`
UNION ALL
SELECT item_number, 'Jigsaw puzzle' AS game_type FROM `JigsawItem`
)
INNER JOIN `Item` ON item_number = `Item`.item_number
Where user_email = '$Email'
GROUP BY game_type;
```

- Display total counts and display:

```
SELECT COUNT(item_number) FROM `Item ` WHERE user_email = '$Email';
```

Search For Items

Abstract Code

- User clicked ***Search Items*** button from the **Main Menu,** display **Search Items** form
- If user clicked ***By keyword*** option, then enter a keyword string ($Keyword):

```
SELECT
        3958.75 * (
        2 * ATN2(
                SQRT(
                        POWER( SIN ((lat2-lat1)/2) ,2) * POWER( COS(lat2) ,2) *
                        POWER( SIN ((lon2-lon1)/2) ,2)
                        ) ,
                SQRT( 1 -
                        (POWER( SIN ((lat2-lat1)/2) ,2) * POWER( COS(lat2) ,2) *
                        POWER( SIN ((lon2-lon1)/2) ,2))
                )
        )
        ) AS distance,
        Item_number,
        name,
        game_type,
        condition,
        LEFT(description, 100)
FROM (
SELECT
        RADIANS(ProposerAddress.latitude) AS lat1,
        RADIANS(ProposerAddress.longitude) AS lon1,
        RADIANS(CounterpartyAddress.latitude) AS lat2,
        RADIANS(CounterpartyAddress.longitude) AS lon2,
FROM `Item`
INNER JOIN(
SELECT item_number, 'Board game' AS game_type FROM `BoardItem`
UNION ALL
SELECT item_number, 'Card game' AS game_type FROM `CardItem`
UNION ALL
SELECT item_number, 'Video game' AS game_type FROM `VideoItem`
UNION ALL
SELECT item_number, 'Computer game' AS game_type FROM `ComputerItem`
UNION ALL
SELECT item_number, 'Jigsaw puzzle' AS game_type FROM `JigsawItem`
) ON item_number = `Item`.item_number
INNER JOIN `User` ON user_email = email
INNER JOIN `Address` AS ProposerAddress ON `User`.postal_code =
ProposerAddress.postal_code
CROSS JOIN (
        SELECT latitude, longitude FROM `Address`
        INNER JOIN `User` ON `User`.postal_code = `Address`.postal_code AND
        email = '$Email'
) AS CounterpartyAddress
```

```
WHERE LOWER(name) LIKE '%$Keyword%' OR  LOWER(description) LIKE
'%$Keyword%';
```

- Else if user clicked ***In my postal code*** button:

```
SELECT
        3958.75 * (
        2 * ATN2(
                SQRT(
                        POWER( SIN ((lat2-lat1)/2) ,2) * POWER( COS(lat2) ,2) *
                        POWER( SIN ((lon2-lon1)/2) ,2)
                        ) ,
                SQRT( 1 -
                        (POWER( SIN ((lat2-lat1)/2) ,2) * POWER( COS(lat2) ,2) *
                        POWER( SIN ((lon2-lon1)/2) ,2))
                )
        )
        ) AS distance,
        Item_number,
        name,
        game_type,
        condition,
        LEFT(description, 100)
FROM (
SELECT
        RADIANS(ProposerAddress.latitude) AS lat1,
        RADIANS(ProposerAddress.longitude) AS lon1,
        RADIANS(CounterpartyAddress.latitude) AS lat2,
        RADIANS(CounterpartyAddress.longitude) AS lon2,
FROM `Item`
INNER JOIN(
SELECT item_number, 'Board game' AS game_type FROM `BoardItem`
UNION ALL
SELECT item_number, 'Card game' AS game_type FROM `CardItem`
UNION ALL
SELECT item_number, 'Video game' AS game_type FROM `VideoItem`
UNION ALL
SELECT item_number, 'Computer game' AS game_type FROM `ComputerItem`
UNION ALL
SELECT item_number, 'Jigsaw puzzle' AS game_type FROM `JigsawItem`
) ON item_number = `Item`.item_number
INNER JOIN `User` ON user_email = email
INNER JOIN `Address` AS ProposerAddress ON `User`.postal_code =
ProposerAddress.postal_code
CROSS JOIN (
        SELECT latitude, longitude FROM `Address`
        INNER JOIN `User` ON `User`.postal_code = `Address`.postal_code AND
        email = '$Email'
) AS CounterpartyAddress
WHERE ProposerAddress.postal_code = (SELECT postal_code FROM `User`
WHERE email = '$Email')
```

- Else if user clicked ***In postal code*** button, then enter a ($Postalcode):

```
SELECT
        3958.75 * (
        2 * ATN2(
                SQRT(
                        POWER( SIN ((lat2-lat1)/2) ,2) * POWER( COS(lat2) ,2) *
                        POWER( SIN ((lon2-lon1)/2) ,2)
                        ) ,
                SQRT( 1 -
                        (POWER( SIN ((lat2-lat1)/2) ,2) * POWER( COS(lat2) ,2) *
                        POWER( SIN ((lon2-lon1)/2) ,2))
                )
        )
        ) AS distance,
        Item_number,
        name,
        game_type,
        condition,
        LEFT(description, 100)
FROM (
SELECT
        RADIANS(ProposerAddress.latitude) AS lat1,
        RADIANS(ProposerAddress.longitude) AS lon1,
        RADIANS(CounterpartyAddress.latitude) AS lat2,
        RADIANS(CounterpartyAddress.longitude) AS lon2,
FROM `Item`
INNER JOIN(
SELECT item_number, 'Board game' AS game_type FROM `BoardItem`
UNION ALL
SELECT item_number, 'Card game' AS game_type FROM `CardItem`
UNION ALL
SELECT item_number, 'Video game' AS game_type FROM `VideoItem`
UNION ALL
SELECT item_number, 'Computer game' AS game_type FROM `ComputerItem`
UNION ALL
SELECT item_number, 'Jigsaw puzzle' AS game_type FROM `JigsawItem`
) ON item_number = `Item`.item_number
INNER JOIN `User` ON user_email = email
INNER JOIN `Address` AS ProposerAddress ON `User`.postal_code =
ProposerAddress.postal_code
CROSS JOIN (
        SELECT latitude, longitude FROM `Address`
        INNER JOIN `User` ON `User`.postal_code = `Address`.postal_code AND
        email = '$Email'
) AS CounterpartyAddress
WHERE ProposerAddress.postal_code = '$Postalcode'
```

- Else if user clicked ***Within xxx miles of me*** button:
    - User inputted a $Distance

```
SELECT
        3958.75 * (
        2 * ATN2(
                SQRT(
                        POWER( SIN ((lat2-lat1)/2) ,2) * POWER( COS(lat2) ,2) *
                        POWER( SIN ((lon2-lon1)/2) ,2)
                        ) ,
                SQRT( 1 -
                        (POWER( SIN ((lat2-lat1)/2) ,2) * POWER( COS(lat2) ,2) *
                        POWER( SIN ((lon2-lon1)/2) ,2))
                )
        )
        ) AS distance,
        Item_number,
        name,
        game_type,
        condition,
        LEFT(description, 100)
FROM (
SELECT
        RADIANS(ProposerAddress.latitude) AS lat1,
        RADIANS(ProposerAddress.longitude) AS lon1,
        RADIANS(CounterpartyAddress.latitude) AS lat2,
        RADIANS(CounterpartyAddress.longitude) AS lon2,
FROM `Item`
INNER JOIN(
SELECT item_number, 'Board game' AS game_type FROM `BoardItem`
UNION ALL
SELECT item_number, 'Card game' AS game_type FROM `CardItem`
UNION ALL
SELECT item_number, 'Video game' AS game_type FROM `VideoItem`
UNION ALL
SELECT item_number, 'Computer game' AS game_type FROM `ComputerItem`
UNION ALL
SELECT item_number, 'Jigsaw puzzle' AS game_type FROM `JigsawItem`
) ON item_number = `Item`.item_number
INNER JOIN `User` ON user_email = email
INNER JOIN `Address` AS ProposerAddress ON `User`.postal_code =
ProposerAddress.postal_code
CROSS JOIN (
        SELECT latitude, longitude FROM `Address`
        INNER JOIN `User` ON `User`.postal_code = `Address`.postal_code AND
        email = '$Email'
) AS CounterpartyAddress
WHERE distance < '$Distance'
```

View Item

Abstract Code

- In **My Items**, **Search Items Accept/Reject Swap, Swap History** forms, user clicked ***Detail*** button along with each item with $ItemNumber:

```
SELECT item_number, game_type, user_email, name, condition, description, distance
FROM `Item`
INNER JOIN(
SELECT item_number, 'Board game' AS game_type FROM `BoardItem`
UNION ALL
SELECT item_number, 'Card game' AS game_type FROM `CardItem`
UNION ALL
SELECT item_number, 'Video game' AS game_type FROM `VideoItem`
UNION ALL
SELECT item_number, 'Computer game' AS game_type FROM `ComputerItem`
UNION ALL
SELECT item_number, 'Jigsaw puzzle' AS game_type FROM `JigsawItem`
) ON item_number = `Item`.item_number
WHERE item_number = '$ItemNumber';
```

- Get $GameType from above query
  - If '$Game_type' = 'Video Game':

```
SELECT platform, media FROM `VideoItem` WHERE item_number = '$ItemNumber';
```

  - If '$Game_type' = Computer Game':

```
SELECT platform FROM `ComputerItem` WHERE item_number = '$ItemNumber';
```

  - If '$Game_type' = 'Jigsaw Puzzle':

```
SELECT piece_count FROM `JigsawItem` WHERE item_number = '$ItemNumber';
```

- Display above information first.
- Get $UserEmail from above query
  - If $UserEmail != '$Email' (Which is current User's email):
  - Get address of that user and display:

```
SELECT city, state, postal_code
FROM `User` INNER JOIN `Address` ON `User`.postal_code = `Address`.postal_code
WHERE `User`.email = '$UserEmail'
```

  - Get rating of that user and display:

```
SELECT
        ROUND(AVG(rating),2) AS rating
        user_email
FROM
(SELECT
        proposer_rating AS rating,
          user_email
FROM `Swap`
INNER JOIN `Item` AS ProposedItem ON ProposedItem.item_number =
proposed_item_number
WHERE ProposedItem.user_email = '$UserEmail' AND `proposer_rating` IS NOT
NULL
UNION ALL
SELECT
          counterparty_rating AS rating,
          user_email
FROM `Swap`
INNER JOIN `Item` AS DesiredItem ON DesiredItem.item_number =
desired_item_number
WHERE DesiredItem.user_email = '$UserEmail' AND `counterparty_rating` IS NOT
NULL) AS Ratings
```

- If the user's postal_code != '$User'.postal_code, then run the task of **GetDistanceBetweenTwoUsers**:

```
SELECT
          3958.75 * (
          2 * ATN2(
                    SQRT(
                              POWER( SIN ((lat2-lat1)/2) ,2) * POWER( COS(lat2) ,2) *
                              POWER( SIN ((lon2-lon1)/2) ,2)
                              ) ,
                    SQRT( 1 -
                              (POWER( SIN ((lat2-lat1)/2) ,2) * POWER( COS(lat2) ,2) *
                              POWER( SIN ((lon2-lon1)/2) ,2))
                    )
          )
          ) AS distance,
FROM (
SELECT
          RADIANS(OtherAddress.latitude) AS lat1,
          RADIANS(OtherAddress.longitude) AS lon1,
          RADIANS(MyAddress.latitude) AS lat2,
          RADIANS(MyAddress.longitude) AS lon2,
FROM `User`
INNER JOIN `Address` AS OthersAddress ON `User`.postal_code =
OtherAddress.postal_code AND email = '$UserEmail'
CROSS JOIN (
          SELECT latitude, longitude FROM `Address`
          INNER JOIN `User` ON `User`.postal_code = `Address`.postal_code AND
          email = '$Email'
) AS MyAddress);
```

Accept/Reject Swap

Abstract Code

- User clicked the number in the ***Unaccepted Swaps*** panel of the **Main Menu:**
- Run the **Read Swaps** Task: Find associated swaps using the logged user $Email with swap status NULL.

```
SELECT proposed_date, DesiredItem.name,  Proposer.nickname,
ProposedItem.name, Proposer.email
FROM `Swap`
INNER JOIN `Item` AS DesiredItem ON DesiredItem.item_numer =
desired_item_number
INNER JOIN `Item` AS ProposedItem ON ProposedItem.item_number =
proposed_item_number
INNER JOIN `User` AS Proposer ON Proposer.email = ProposedItem.user_email
INNER JOIN `User` AS CounterParty ON CounterParty.email =
DesiredItem.user_email
WHERE DesiredItem.user_email = '$Email' AND `status` IS NULL;
```

- Store the Proposer.email in $ProposerEmails
- Calculate the ratings of the proposers:

```
SELECT
        ROUND(AVG(rating),2) AS rating
        user_email as proposer_email,
FROM
(SELECT
        proposer_rating AS rating,
        user_email
FROM `Swap`
INNER JOIN `Item` AS ProposedItem ON ProposedItem.item_number =
proposed_item_number
WHERE ProposedItem.user_email IN ('$ProposerEmails') AND `proposer_rating` IS
NOT NULL
UNION ALL
SELECT
        counterparty_rating AS rating,
        user_email
FROM `Swap`
INNER JOIN `Item` AS DesiredItem ON DesiredItem.item_number =
desired_item_number
WHERE DesiredItem.user_email IN ('$ProposerEmails') AND `counterparty_rating` IS
NOT NULL) AS Ratings
GROUP BY user_email;
```

Calculate distance of the proposers:

```
SELECT
        3958.75 * (
        2 * ATN2(
                SQRT(
                        POWER( SIN ((lat2-lat1)/2) ,2) * POWER( COS(lat2) ,2) *
                        POWER( SIN ((lon2-lon1)/2) ,2)
                        ) ,
                SQRT( 1 -
                        (POWER( SIN ((lat2-lat1)/2) ,2) * POWER( COS(lat2) ,2) *
                        POWER( SIN ((lon2-lon1)/2) ,2))
                )
        )
        ) AS distance,
        email AS proposer_email
FROM (
SELECT
        RADIANS(ProposerAddress.latitude) AS lat1,
        RADIANS(ProposerAddress.longitude) AS lon1,
        RADIANS(CounterpartyAddress.latitude) AS lat2,
        RADIANS(CounterpartyAddress.longitude) AS lon2,
FROM `User`
INNER JOIN `Address` AS ProposerAddress ON `User`.postal_code =
ProposerAddress.postal_code AND email IN ('$ProposerEmail')
CROSS JOIN (
        SELECT latitude, longitude FROM `Address`
        INNER JOIN `User` ON `User`.postal_code = `Address`.postal_code AND
        email = '$Email'
) AS CounterpartyAddress
```

- Display pending swaps using the above query results. (Date, DesiredItem.name, Proposer.nickname, rating and distance)

- When the ***Accept*** button is pressed:
    - Confirm consistency of items: run the same queries as above and confirm that no changes have been made (e.g. change in proposer location/distance)
    - Run **Accept Swap** task**:**

```
UPDATE `Swap` SET accept_reject_date = GETDATE(), status = 1 WHERE
proposed_item_number = '$ProposedItemNumber' AND desired_item_number
= '$DesiredItemNumber'
```

    - display a dialog with the proposer's email, first name, and phone number/type, if available and if sharing option is set.

```
SELECT
        email,
        first_name,
        IIF(share_phone AND phone_number IS NOT NULL,
                CONCAT(phone_number, '-',phone_number_type),
                'No phone number available') AS phone
FROM `User` WHERE email = '$ProposerEmail'
```

    - Run the **Read Swaps** task to repopulate the list (removing the accepted)

- When the ***Reject*** button is pressed:
    - Run **Reject Swap** task**:**
        - Record the Rejected Date
        - Update Swap Status to “rejected”

```
UPDATE `Swap` SET accept_reject_date = GETDATE(), status = 0 WHERE
proposed_item_number = '$ProposedItemNumber' AND desired_item_number
= '$DesiredItemNumber'
```

    - Run the **Read Swaps** task again to repopulate the list (removing the rejected)

- If no more swaps are pending return to the **Main Menu.** That is, if the query returns 0 entries.

Propose Swap

Abstract Code

- User clicked ***Propose Swap*** from the **Item Details** view.
- Run the **Get Unrated Swaps** task for the logged user: query information about the unrated swaps.
- Run the **Get Unaccepted Swaps** task for the logged user:
- If the number of unrated swaps is greater than 2 OR the number of unaccepted swaps is greater than 5, then return to the previous screen and alert the user.
- Run the **View Item** task for the counterparty using the '$DesiredItemNumber' of the desired item and **Calculate Distance** of the counterparty.
    - If the counterparty is >= 100.0 miles away from the user, display a warning message containing that distance.

- Run the **My Items** Task for the currently logged user using the '$Email'
    - Remove items that are already associated with a swap (proposed or desired)
- When the Confirm button is pressed AND an item is selected ('$ProposedItemNumber'), then:
    - Verify that no other pending swaps are associated with the two item:

```
SELECT
        COUNT(*)
FROM `Swap`
WHERE (proposed_item_number = '$ProposedItemNumber' OR
desired_item_number = '$DesiredItemNumber') AND (status  IS NULL OR
status = 1)
```

    - If there is already a pending swap, show an error message and return to the previous screen.
    - Verify that the two items have never been in a swap before:

```
SELECT
        COUNT(*)
FROM `Swap`
WHERE (proposed_item_number = '$ProposedItemNumber' AND
desired_item_number = '$DesiredItemNumber')
```

    - If there is already an associated swap, show an error message and return to the previous screen.
    - Run the **Write Swap** task:

```
INSERT INTO `Swap` (proposed_item_number, desired_item_number,
proposed_date) VALUES ( '$ProposedItemNumber',
'$DesiredItemNumber',GETDATE())
```

- Display confirmation message
- Return to **Main Menu**.

Rate Swap

Abstract Code

- User clicks ***Unrated Swap*** number in the **Main Menu** view
- Run the **View Unrated Swaps** task: query information about accepted swaps (and the related items) that are pending a rating from the logged user.

```
SELECT
        accept_reject_date,
        'Proposer' AS role,
        ProposedItem.name AS proposed_item_name,
        DesiredItem.name AS desired_item_name,
        OtherUser.nickname AS other_user
FROM `Swap`
INNER JOIN `Item` AS ProposedItem ON ProposedItem.item_number =
proposed_item_number
INNER JOIN `Item` AS DesiredItem ON DesiredItem.item_number =
desired_item_number
INNER JOIN `User` AS OtherUser ON DesiredItem.user_email = OtherUser.email
WHERE ProposedItem.user_email = '$Email' AND counterparty_rating IS NULL AND
status IS NOT NULL
UNION ALL
SELECT
        accept_reject_date,
        'Counterparty' AS role,
        ProposedItem.name AS proposed_item_name,
        DesiredItem.name AS desired_item_name,
        OtherUser.nickname AS other_user
FROM `Swap`
INNER JOIN `Item` AS ProposedItem ON ProposedItem.item_number =
proposed_item_number
INNER JOIN `Item` AS DesiredItem ON DesiredItem.item_number =
desired_item_number
INNER JOIN `User` AS OtherUser ON ProposedItem.user_email = OtherUser.email
WHERE DesiredItem.user_email = '$Email' AND proposer_rating IS NULL AND status IS
NOT NULL
ORDER BY accept_reject_date DESC
```

- When the user inputs ***0-5 rating*** for any swaps run the **Rate Swap** task
    - If the user is the counterparty:

```
UPDATE `Swap` SET proposer_rating =  '$Rating' WHERE proposed_item_number =
'$ProposedItemNumber AND desired_item_number = '$DesiredItemNumber
```

    - If the user is the proposer:

```
UPDATE `Swap` SET counterparty_rating =  '$Rating' WHERE proposed_item_number =
'$ProposedItemNumber AND desired_item_number = '$DesiredItemNumber
```

- Run the **View Unrated Swaps** again to update the list and remove the rated swap.
- If no additional swaps need rating, return the user to the **main menu**

View Swap History

Abstract Code

- User clicks ***Swap History*** from the **Main Menu** view
- Run the **View Swaps Summary** task: query total swaps proposed, total received, etc.

```
SELECT
        my_role,
        COUNT(*) AS  total,
        SUM(status) AS accepted,
        COUNT(*)  - SUM(status) AS rejected,
        (1  - SUM(status) / COUNT(*) ) * 100  rejected_percent
FROM (
SELECT
        'Proposer' AS my_role,
        status
FROM `Swap`
INNER JOIN `Item` AS ProposedItem ON ProposedItem.item_number =
proposed_item_number
WHERE ProposedItem.user_email = '$Email' AND status IS NOT NULL
UNION ALL
SELECT
        'Counterparty' AS role,
        status
FROM `Swap`
INNER JOIN `Item` AS DesiredItem ON DesiredItem.item_number =
desired_item_number
WHERE DesiredItem.user_email = '$Email' AND status IS NOT NULL
) AS UserSwaps
GROUP BY my_role
```

- Display the swaps summary table at the top

- Run the **View Swap History** task: query all the swaps associated with the user

```
SELECT
        proposed_date,
        accept_reject_date,
        IIF(status IS NULL,'', IIF(status = 1, 'Accepted', 'Rejected')) AS swap_status
        'Proposer' AS my_role,
        ProposedItem.name AS proposed_item_name,
        DesiredItem.name AS desired_item_name,
        OtherUser.nickname AS other_user
        counterparty_rating AS rating
FROM `Swap`
INNER JOIN `Item` AS ProposedItem ON ProposedItem.item_number =
proposed_item_number
INNER JOIN `Item` AS DesiredItem ON DesiredItem.item_number =
desired_item_number
INNER JOIN `User` AS OtherUser ON DesiredItem.user_email = OtherUser.email
WHERE ProposedItem.user_email = '$Email'

UNION ALL

SELECT
        proposed_date
        accept_reject_date,
        IIF(status IS NULL,'', IIF(status = 1, 'Accepted', 'Rejected')) AS swap_status
        'Counterparty' AS my_role,
        ProposedItem.name AS proposed_item_name,
        DesiredItem.name AS desired_item_name,
        OtherUser.nickname AS other_user,
        proposer_rating AS rating
FROM `Swap`
INNER JOIN `Item` AS ProposedItem ON ProposedItem.item_number =
proposed_item_number
INNER JOIN `Item` AS DesiredItem ON DesiredItem.item_number =
desired_item_number
INNER JOIN `User` AS OtherUser ON ProposedItem.user_email = OtherUser.email
WHERE DesiredItem.user_email = '$Email'
ORDER BY accept_reject_date DESC, proposed_date ASC
```

- Display the all swaps table with the query information.
- When the user inputs ***0-5 rating*** for any unrated swaps run the **Rate Swap** task

View Swap Details

Abstract Code

- User clicks ***Details*** in the **Swap History** page
- Run the **View Swap Details** task: query information about the swap

```
SELECT
        proposed_date,
        accepted_rejected_date,
        IIF(status=1,'Accepted','Rejected') AS status,
        IIF(ProposedItem.user_email = '$Email', 'Proposer', 'Counterparty') AS my_role,
        IIF(ProposedItem.user_email = '$Email', counterparty_rating, proposer_rating)
        AS rating_left
FROM `Swap`
INNER JOIN `Item` AS ProposedItem ON ProposedItem.item_number =
proposed_item_number
INNER JOIN `Item` AS DesiredItem ON DesiredItem.item_number =
desired_item_number
```

- Run the **View Profile** task using the '$OtherUserEmail'

```
SELECT
        nickname,
        CONCAT(first_name, ' ', last_name ) AS name,
        Email,
        IIF(share_phone AND phone_number IS NOT NULL,
                CONCAT(phone_number, '-',phone_number_type),
                'No phone number available') AS phone
FROM `User`
WHERE email = '$OtherUserEmail'
```

- Run the **Calculate Distance** task to get the distance between the logged user and the OtherUser.
- If the rating left is empty and the user inputs a ***0-5 rating*** run the **Rate Swap** task