SWAP

Isabelle May

CPSC 321, Fall 2024

1 Summary

The domain I selected for my project, Swap, is a college marketplace platform tailored specifically for students. The app focuses on enabling students to post and browse items for sale within their university community, offer freelance services to students or community members, message other users, and rate items and services they have received. This domain addresses a need for affordable, localized, safe, and convenient exchanges among college students and their surrounding community.

2 Use Cases

2.1 User Logs In

Action: Searching for Data
Description: A user provides their email and password to access their account. The query verifies their credentials by searching the database for a matching combination. If no matching record is found, the user is prompted to create an account or reattempt the log in.

2.2 User Signs Up

Action: Creating data
Description: To create an account, a new user enters their personal details, such as name, email, location, university, and password. The function checks if the email already exists, and if not, it inserts the user's information into the database.

2.3 Viewing User Profile

Action: Searching for data & Analyzing Data
Description: Retrieves and displays user profile information (e.g., name, email, rating) from the database and provides options for managing listings and job postings. The query also calculates the user's rating by averaging their initial rating, item ratings, and freelance work ratings.

2.4 Viewing User's Item Listings

Action: Searching for data
Description: Queries the database to retrieve and display all items listed for sale by a specific user.

2.5 Viewing User's Job Postings

Action: Searching for data
Description: Retrieves and displays all job postings created by a specific user from the database.

2.6 Create Item Listing
Action: Creating data
Description: Allows a user to create a new item listing for sale by entering details (e.g., item type, description, price) and stores the listing in the database.

2.7 Create Job Posting
Action: Creating data
Description: Allows a user to create a new freelance job posting by entering details (e.g., job description, payment rate) and stores the job posting in the database.

2.8 Delete Item Listing
Action: Removing data
Description: Allows a user to delete one of their item listings by its ID, removing the listing from the database.

2.9 Delete Job Posting
Action: Removing data
Description: Allows a user to delete one of their job postings by its ID, removing the posting from the database.

2.10 Update Item Listing
Action: Updating data
Description: Allows a user to update an existing item listing by modifying its description and/or price in the database.

2.11 Update Item Listing
Action: Updating data
Description: Allows a user to update an existing job posting by modifying its description and/or payment rate in the database.

2.12 Browsing Items
Action: Searching for data
Description: A user browses items for sale in their city, with filters to sort by category, date posted, or seller rating. The function retrieves the user's profile information to ensure the displayed listings are in their city and do not include the user's own items.

2.13 Sorting Items by Category
Action: Searching for Data
Description: The function filters and organizes items by a specific category chosen by the user from a list of possible categories. It then sorts the results by the date they were posted to show the available inventory for a particular category.

2.14 Sorting Items by Date Posted
Action: Searching for Data

Description: Items are displayed in descending order of their posting dates, ensuring users see the most recent listings first.

2.15 Sorting Items by Highest Seller Rating
Action: Searching for Data
Description: Item listings are sorted by the seller's highest rating, helping users prioritize transactions with reputable sellers.

2.16 Viewing Seller Information for Items
Action: Searching for Data
Description: The user selects an item and retrieves detailed information about the seller, including their name, email, location, and rating.

2.17 Browsing Job Listings
Action: Searching for data
Description: A user browses freelance job listings in their city, with filters to sort by category, date posted, or rating. The function retrieves the user's profile information to ensure the displayed listings are in their city and do not include the user's own job postings.

2.18 Sorting Job Listings by Skill Type
Action: Searching for Data
Description: The function filters and organizes job listings by a specific skill chosen by the user from a list of possibilities. It then sorts the results by the date they were posted to show the available jobs for a particular skill.

2.19 Sorting Job Listings by Date Posted
Action: Searching for Data
Description: Job listings are displayed in descending order of their posting dates, ensuring users see the most recent listings first.

2.20 Sorting Job Listings by Highest Rating
Action: Searching for Data
Description: Job listings are sorted by the freelance workers rating (highest to lowest), helping users prioritize transactions with reputable workers.

2.21 Viewing Freelancer Information
Action: Searching for Data
Description: The user selects a job posting and retrieves detailed information about the worker, including their name, email, location, and rating.

2.22 Recording Item Sale
Action: Creating Data and Updating Data
Description: Allows users to record item sales by listing items posted by the user, marking items as sold, and creating a corresponding transaction record. It ensures that sold items are no longer listed as available.

2.23 Recording Freelance Work Transaction
Action: Creating Data
Description: Supports recording freelance work transactions by listing user's available jobs, capturing transaction details, and adding a new record to the freelance transaction table. It ensures a clear record of services provided and payment details.

2.24 Viewing Messages
Action: Searching for data
Description: Displays all unique conversations the user has participated in, showing interactions grouped by email. It facilitates viewing detailed message history and selecting specific conversations for further interaction.

2.25 Displaying Conversation History
Action: Searching for data
Description: Retrieves and displays all messages exchanged between the user and a selected contact, sorted by the date sent. This provides a clear timeline of communications for easy reference.

2.26 Responding to Conversation
Action: Creating data
Description: Enables users to send follow-up messages in an ongoing conversation. It enhances user engagement by maintaining the continuity of conversations directly from the history view.

2.27 Creating Message
Action: Creating data
Description: Allows users to send a new message to a specified recipient. Ensures the recipient exists and that the message content is valid before storing it in the database.

2.28 Rating Freelance Work
Action: Creating data
Description: Enables users to provide ratings and optional reviews for freelance services they have received.

2.29 Rating Item
Action: Creating data
Description: Allows users to rate and review sellers for items they've purchased contributing to the platform's credibility by fostering a transparent evaluation system for transactions.

2.30 Viewing Freelance Ratings
Action: Searching for data
Description: Displays all freelance work ratings given by the user, including ratings, optional reviews, and timestamps.

2.31 Viewing Item Ratings
Action: Searching for data
Description: Displays all item ratings given by the user, including ratings, optional reviews, and timestamps.

2.32 Viewing Freelance Ratings About User
Action: Searching for data
Description: Displays all freelance work ratings given to the user, including ratings, optional reviews, and timestamps.

2.33 Viewing Item Ratings About User
Action: Searching for data
Description: Displays all item ratings given to the user, including ratings, optional reviews, and timestamps.

2.34 Finding Active Users This Year
Action: Analyzing Data
Description: Identifies the number of active users who listed items for sale within the current year. It helps the platform gauge user engagement and activity trends over time.

2.35 FInding Total Sales by Item Category
Action: Analyzing Data
Description: This use case calculates the total revenue generated for each item category over the year. It allows the platform to determine which categories perform the best and may inform decisions about promotions or improvements in the future.

2.36 Finding Average Item Price by Seller
Action: Analyzing Data
Description: This use case computes the average price of items listed by each seller currently active on the platform. It provides insight into pricing trends and helps evaluate seller performance.

2.37 Finding Most Active Seller/Worker by Transaction
Action: Analyzing Data
Description: This use case ranks sellers by the number of transactions they have taken part in as the seller for both item and freelance sales.

2.38 Finding Most Active Item Buyers
Action: Analyzing Data
Description: This use case ranks buyers by the amount of money they have spent on items across transactions

2.39 Populating ItemTransactions

Action: Creating Data (simulates recording item transactions)
Description: Generates 45 random transactions by pairing available items for sale with users as potential buyers, ensuring the buyer is not the seller. Each transaction includes details like item ID, buyer and seller emails, price, and the current date as the completion time. It ensures randomness by shuffling rows and selecting unique buyer-seller combinations.

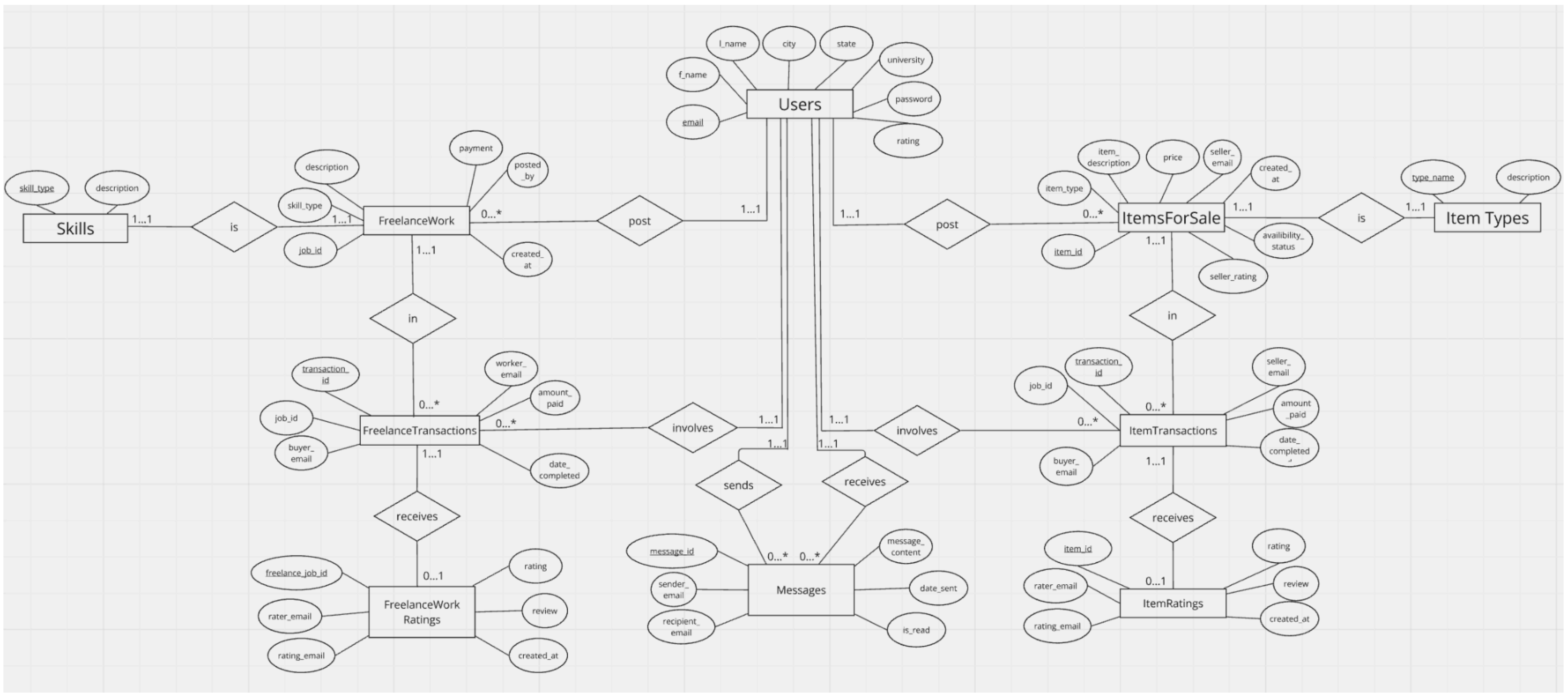
2.40 Populating FreelanceTransactions
Action: Creating Data (simulates recording freelance work transactions)
Description: Creates 45 random freelance job transactions by matching posted jobs with buyers and workers, ensuring that the buyer is not the same as the worker. Each transaction includes details like job ID, buyer and worker emails, payment amount, and the current date as the transaction date. Similar to populating the item transactions table, randomness is achieved by shuffling rows and selecting from possible buyer-worker pairs for each job.

2.41 Populating ItemRatings
Action: Creating Data (simulates ratings for items)
Description: Assigns ratings and optional reviews to items in ItemsTransactions. Ratings are randomized between 4.00 and 5.00, with reviews added 70% of the time.

2.42 Populating FreelanceWorkRatings
Action: Creating Data (simulates ratings for freelance work)
Description: Assigns ratings and optional reviews to jobs in FreelanceWorkTransactions. Ratings are randomized between 4.00 and 5.00, with reviews added 70% of the time.

3 Logical Design

3.1 Entity-Relationship Diagram



3.2 Relational Schema

Users(email, f_name, l_name, city, state, university, password, rating)
ItemTypes(type_name, description)
ItemsForSale(item_id, item_type, item_description, price, seller_email, created_at, availability_status)
Skills(skill_type, description)
FreelanceWork(job_id, skill_type, description, payment, posted_by, created_at)
ItemTransactions(transaction_id, item_id, buyer_email, seller_email, amount_paid, date_completed)
FreelanceTransactions(transaction_id, job_id, buyer_email, worker_email, amount_paid, date_completed)
ItemRatings(item_id, rater_email, rating_email, rating, review, created_at)
FreelanceWorkRatings(freelance_job_id, rater_email, rating_email, rating, review, created_at)
Messages(message_id, sender_email, recipient_email, message_content, date_sent, is_read)

FKs: ItemsForSale.seller_email → Users.email
ItemsForSale.item_type → ItemTypes.type_name
FreelanceWork.posted_by → Users.email
FreelanceWork.skill_type → Skills.skill_type
ItemTransactions.item_id → ItemsForSale.item_id
ItemTransactions.buyer_email → Users.email
ItemTransactions.seller_email → Users.email
FreelanceTransactions.job_id → FreelanceWork.job_id
FreelanceTransactions.buyer_email → Users.email
FreelanceTransactions.worker_email → Users.email
ItemRatings.item_id → ItemsForSale.item_id
ItemRatings.rater_email → Users.email
ItemRatings.rating_email → Users.email
FreelanceWorkRatings.freelance_job_id → FreelanceWork.job_id
FreelanceWorkRatings.rater_email → Users.email
FreelanceWorkRatings.rating_email → Users.email
Messages.sender_email → Users.email
Messages.recipient_email → Users.email

4 Use-Case SQL Statements

For each use case, give the corresponding SQL statements that you developed. Each SQL statement should support the use case. Be sure to describe each query (one sentence) and state the corresponding parameters (which you can denote as ? or %s, depending on how you implemented them in your applications). Organize this section like section 3 (each use case as a separate subsection).

4.1 User Logs In
Description: This query checks if a user exists in the Users table with a matching email and password.
Parameters (%s): user_email, password

```
SELECT f_name FROM Users WHERE email = %s AND password = %s
```

4.2 User Signs Up
Description: This query inserts a new user's details into the Users table.
Parameters (%s): f_name, l_name, city, state, email, university, password

```
INSERT INTO Users (f_name, l_name, city, state, email, university, password)
VALUES (%s, %s, %s, %s, %s, %s, %s)
```

4.3 Viewing User Profile
Description: Retrieves all items listed for sale by the user, including item details and availability status.
Parameters (%s): user_email

```
SELECT
        u.f_name, u.l_name, u.email, u.city, u.state, u.university,
        ROUND(
            (u.rating +
             CASE WHEN COUNT(ir.rating) > 0 THEN AVG(ir.rating) ELSE 5 END +
             CASE WHEN COUNT(fr.rating) > 0 THEN AVG(fr.rating) ELSE 5 END) / 3,
            2
        ) AS calculated_rating
    FROM
        Users u
    LEFT JOIN
        ItemRatings ir ON ir.rating_email = u.email
    LEFT JOIN
        FreelanceWorkRatings fr ON fr.rating_email = u.email
    WHERE
        u.email = %s
    GROUP BY
        u.f_name, u.l_name, u.email, u.city, u.state, u.university, u.rating;
```

4.4 Viewing User's Item Listings
Description: Retrieves all items listed for sale by the user, including item details and availability status.
Parameters (%s): user_email

```
SELECT item_id, item_type, item_description, price, availability_status
   FROM ItemsForSale
   WHERE seller_email = (SELECT email FROM Users WHERE email = %s);
```

4.5 Viewing User's Job Postings
Description: Retrieves all job postings created by the user, including job details and payment rates.
Parameters (%s): user_email

```
SELECT job_id, skill_type, description, payment
FROM FreelanceWork
WHERE posted_by = (SELECT email FROM Users WHERE email = %s);
```

4.6 Create Item Listing
Description: Inserts a new item listing into the database for sale by the user.
Parameters:
Parameters (%s): user_email, item_type, item_description, price, availability_status

```
INSERT INTO ItemsForSale (seller_email, item_type, item_description, price,
availability_status)
VALUES (%s, %s, %s, %s, %s) RETURNING item_id;
```

4.7 Create Job Posting
Description: Inserts a new freelance job posting into the database.
Parameters (%s): user_email, skill_type, description, payment

```
INSERT INTO FreelanceWork (skill_type, description, payment, posted_by)
VALUES (%s, %s, %s, %s) RETURNING job_id;
```

4.8 Delete Item Listing
Description: Deletes a specific item listing owned by the user.
Parameters (%s): item_id, user_email

```
DELETE FROM ItemsForSale
WHERE item_id = %s AND seller_email = %s;
```

4.9 Delete Job Posting
Description: Deletes a specific job posting created by the user.
Parameters (%s): job_id, user_email

```
DELETE FROM FreelanceWork
WHERE job_id = %s AND posted_by = %s;
```

4.10 Update Item Listing
Description: Updates the description and price of an item listing owned by the user.

Parameters (%s): item_description, price, item_id

```
UPDATE ItemsForSale
SET item_description = %s, price = %s
WHERE item_id = %s;
```

4.11 Update Item Listing

Description: Updates the description and rate of a job posting created by the user.
Parameters (%s): description, rate, job_id

```
UPDATE FreelanceWork
SET description = %s, rate = %s
WHERE job_id = %s;
```

4.12 Browsing Items

Description: This query retrieves available listings from users in the same city as the logged-in user, excluding the user's own items.
Parameters (%s): user_city, user_email

```
SELECT i.item_type, i.item_description, i.price, i.seller_email, i.created_at
FROM ItemsForSale i
JOIN Users u ON i.seller_email = u.email
WHERE i.availability_status = TRUE AND u.city = %s AND i.seller_email != %s
ORDER BY i.created_at DESC;
```

4.13 Sorting Items by Category

Description: This query retrieves items for sale, filtered by category and city, excluding the user's own items.
Parameters (%s): selected_category, user_city, user_email

```
SELECT item_id, item_description, price, seller_email, created_at
FROM ItemsForSale i
JOIN Users u ON i.seller_email = u.email
WHERE i.item_type = %s AND u.city = %s AND i.seller_email != %s AND i.availability_status =
TRUE
ORDER BY created_at DESC;
```

4.14 Sorting Items by Date Posted

Description: This query retrieves items for sale, sorted by the date posted, filtered by city and excluding the user's own items.
Parameters (%s): user_city, user_email

```
SELECT i.item_type, i.item_description, i.price, i.seller_email, i.created_at
FROM ItemsForSale i
JOIN Users u ON i.seller_email = u.email
```

```
WHERE i.availability_status = TRUE AND u.city = %s AND i.seller_email != %s
ORDER BY i.created_at DESC;
```

4.15 Sorting Items by Highest Seller Rating

Description: This query retrieves items for sale, sorted by the highest seller rating, filtered by city and excluding the user's own items.
Parameters (%s): user_city, user_email

```
SELECT i.item_type, i.item_description, i.price, i.seller_email, u.seller_rating
FROM ItemsForSale i
JOIN Users u ON i.seller_email = u.email
WHERE i.availability_status = TRUE AND u.city = %s AND i.seller_email != %s
ORDER BY u.seller_rating DESC;
```

4.16 Viewing Seller Information for Items

Description: This query retrieves the seller's profile information (name, email, city, state, and rating) based on their email.
Parameters (%s): seller_email

```
SELECT f_name, l_name, email, city, state, seller_rating
FROM Users WHERE email = %s;
```

4.17 Browsing Job Listings

Description: This query retrieves all active job listings from the FreelanceWork table, excluding the user's own postings, and orders the results by the job creation date in descending order.
Parameters (%s): user_email

```
SELECT job_id, skill_type, description, payment, posted_by
FROM FreelanceWork
WHERE posted_by != %s AND city = %s
ORDER BY created_at DESC;
```

4.18 Sorting Job Listings by Skill Type

Description: This query retrieves job listings from the FreelanceWork table, filtered by the selected skill type, excluding the user's own postings, and ensuring the jobs are posted by users in the same city as the current user.

Parameters (%s): selected_skill_type, user_email, user_city

```
SELECT job_id, skill_type, description, payment, created_at
FROM FreelanceWork
WHERE posted_by != %s AND posted_by IN (
   SELECT email FROM Users WHERE city = %s
)
```

```
ORDER BY created_at DESC;
```

4.19 Sorting Job Listings by Date Posted

Description: This query retrieves job listings from the FreelanceWork table, excluding the user's own postings, for users in the same city, and orders the results by the job creation date in descending order.
Parameters (%s): user_email, user_city

```
SELECT job_id, skill_type, description, payment, created_at
FROM FreelanceWork
WHERE posted_by != %s AND posted_by IN (
   SELECT email FROM Users WHERE city = %s
)
ORDER BY created_at DESC;
```

4.20 Sorting Job Listings by Highest Rating

Description: This query retrieves job listings from the FreelanceWork table, excluding the user's own postings, for users in the same city, and sorts the results by the freelancer's seller rating in descending order.
Parameters (%s): user_email, user_city

```
SELECT FW.job_id, FW.skill_type, FW.description, FW.payment, U.f_name, U.l_name,
U.seller_rating
FROM FreelanceWork FW
JOIN Users U ON FW.posted_by = U.email
WHERE FW.posted_by != %s AND U.city = %s
ORDER BY U.seller_rating DESC;
```

4.21 Viewing Freelancer Information
Description: This query retrieves the profile information of a freelancer, including their first name, last name, email, city, state, and seller rating, based on the freelancer's email.
Parameters (%s): freelancer_email

```
SELECT f_name, l_name, email, city, state, seller_rating
FROM Users
WHERE email = %s;
```

4.22 Recording Item Sale
Description: After finding the items posted by the user and allowing the user to select which item they want to add to the transaction, this query records the transaction for the sale of the item, including the item ID, buyer’s email, seller’s email, and the amount paid. Another query then updates the availability status of an item, marking it as sold.

Parameters (%s): item_id, buyer_email, seller_email, amount_paid

```
INSERT INTO ItemTransactions (item_id, buyer_email, seller_email, amount_paid)
VALUES (%s, %s, %s, %s);

UPDATE ItemsForSale
SET availability_status = FALSE
WHERE item_id = %s;
```

4.23 Recording Freelance Work Transaction
Description: After finding the job listings posted by the user and allowing the user to select which job they want to add to the transaction, This query records the transaction for the sale of freelance work, including the job ID, buyer's email, worker's email, and the amount paid.
Parameters (%s): item_id, buyer_email, seller_email, amount_paid

```
INSERT INTO FreelanceTransaction (job_id, buyer_email, worker_email, amount_paid)
VALUES (%s, %s, %s, %s);
```

4.24 Viewing Messages
Description: This query retrieves the distinct emails with whom the user has had a conversation, by checking the sender and recipient fields in the Messages table.
Parameters (%s): user_email

```
SELECT DISTINCT
   CASE
      WHEN sender_email = %s THEN recipient_email
      ELSE sender_email
   END AS other_email
FROM Messages
WHERE sender_email = %s OR recipient_email = %s;
```

4.25 Displaying Conversation History
Description: This query retrieves the messages between the user and the selected email address, ordered by the date the message was sent.
Parameters (%s): user_email x2, selected_email x2

```
SELECT sender_email, message_content, date_sent
FROM Messages
WHERE (sender_email = %s AND recipient_email = %s)
OR (sender_email = %s AND recipient_email = %s)
ORDER BY date_sent ASC;
```

4.26 Responding to Conversation

Description: This query allows a user to respond to an existing conversation by inserting a new message into the Messages table with the sender's email, recipient's email, and message content.
Parameters (%s): sender_email, recipient_email, message_content

```
INSERT INTO Messages (sender_email, recipient_email, message_content)
VALUES (%s, %s, %s);
```

4.27 Creating Message

Description: This query (same as above) allows a user to write a message to a new user who they haven’t previously messaged. It does this by inserting a new message into the Messages table with the sender's email, recipient's email, and message content.
Parameters (%s): sender_email, recipient_email, message_content

```
INSERT INTO Messages (sender_email, recipient_email, message_content)
VALUES (%s, %s, %s);
```

4.28 Rating Freelance Work

Description: This query retrieves freelance transactions where the user is the buyer, showing the job ID, description, and worker's email for each transaction. Another query then inserts the new rating into the FreelanceWorkRatings table.
Parameters (%s): buyer_email and freelance_job_id, rater_email, rating_email, rating, review

```
SELECT FT.job_id, FW.description, FT.worker_email
FROM FreelanceTransactions FT
JOIN FreelanceWork FW ON FT.job_id = FW.job_id
WHERE FT.buyer_email = %s;

INSERT INTO FreelanceWorkRatings (freelance_job_id, rater_email, rating_email, rating,
review)
VALUES (%s, %s, %s, %s, %s);
```

4.29 Rating Item

Description: This query retrieves item transactions where the user is the buyer, showing the item ID, description, and seller's email for each transaction. Another query then inserts the new rating into the ItemRatings table.
Parameters (%s): user_email and item_id, rater_email, rating_email, rating, review

```
SELECT IT.item_id, I.item_description, IT.seller_email
FROM ItemTransactions IT
JOIN ItemsForSale I ON IT.item_id = I.item_id
WHERE IT.buyer_email = %s;

INSERT INTO ItemRatings (item_id, rater_email, rating_email, rating, review)
VALUES (%s, %s, %s, %s, %s);
```

4.30 Viewing Freelance Ratings

Description: This query retrieves the user's ratings for freelance work they have rated, including the rating, review, and the date it was created.
Parameters (%s): user_email

```
SELECT FWR.rating, FWR.review, FWR.created_at
FROM FreelanceWorkRatings FWR
WHERE FWR.rater_email = %s;
```

4.31 Viewing Item Ratings

Description: This query retrieves the user's ratings for items they have rated, including the rating, review, and the date it was created.
Parameters (%s): user_email

```
SELECT IR.rating, IR.review, IR.created_at
FROM ItemRatings IR
WHERE IR.rater_email = %s;
```

4.32 Viewing Freelance Ratings About User

Description: This query retrieves ratings received by the user for freelance work, including the rating, review, creation date, and the email of the rater, ordered by the creation date in descending order.
Parameters (%s): user_email

```
SELECT rating, review, created_at, rater_email
FROM FreelanceWorkRatings
WHERE rating_email = %s
ORDER BY created_at DESC;
```

4.33 Viewing Item Ratings About User

Description: his query retrieves ratings received by the user for items they have sold, including the rating, review, creation date, and the email of the rater, ordered by the creation date in descending order.
Parameters (%s): user_email

```
SELECT rating, review, created_at, rater_email
FROM ItemRatings
WHERE rating_email = %s
ORDER BY created_at DESC;
```

4.34 Finding Active Users This Year

Description: This query counts the number of distinct users who have items listed for sale in the year 2024.
Parameters (%s): None

```
SELECT COUNT(DISTINCT u.email) AS active_users
FROM Users u
WHERE u.email IN (
   SELECT DISTINCT i.seller_email
   FROM ItemsForSale i
   WHERE i.created_at BETWEEN '2024-01-01' AND '2024-12-31'
);
```

4.35 FInding Total Sales by Item Category
Description: This query calculates the total sales for each item category, summing the amount paid for items in the year 2024.
Parameters (%s): None

```
SELECT i.item_type, SUM(it.amount_paid) AS total_sales
FROM ItemTransactions it
JOIN ItemsForSale i ON it.item_id = i.item_id
WHERE it.date_completed BETWEEN '2024-01-01' AND '2024-12-31'
GROUP BY i.item_type
ORDER BY total_sales DESC;
```

4.36 Finding Average Item Price by Seller
Description: This query calculates the average price of items listed for sale by each seller, considering only items that are available.
Parameters (%s): None

```
SELECT i.seller_email, AVG(i.price) AS avg_item_price
FROM ItemsForSale i
WHERE i.availability_status = TRUE
GROUP BY i.seller_email
ORDER BY avg_item_price DESC;
```

4.37 Finding Most Active Sellers by Transaction
Description: This query returns the top 10 users who have completed the most freelance and item transactions as the “seller”.
Parameters (%s): None

```
WITH ItemSellerTransactions AS (
         SELECT it.seller_email AS email, COUNT(it.transaction_id) AS transactions_count
         FROM ItemTransactions it
         GROUP BY it.seller_email
      ),
      FreelanceWorkerTransactions AS (
         SELECT ft.worker_email AS email, COUNT(ft.transaction_id) AS transactions_count
         FROM FreelanceTransactions ft
         GROUP BY ft.worker_email
```

```
    ),
    CombinedTransactions AS (
      SELECT email, SUM(transactions_count) AS total_transactions
      FROM (
        SELECT * FROM ItemSellerTransactions
        UNION ALL
        SELECT * FROM FreelanceWorkerTransactions
      ) subquery
      GROUP BY email
    )
    SELECT email, total_transactions
    FROM CombinedTransactions
    ORDER BY total_transactions DESC
    LIMIT 10;
```

4.38 Finding Most Active Buyers
Description: This query returns the top 10 buyers who have spent the most money across item transactions.
Parameters (%s): None

```
WITH BuyerSpending AS (
      SELECT
        it.buyer_email,
        COUNT(it.transaction_id) AS transaction_count,
        SUM(it.amount_paid) AS total_spent,
        AVG(it.amount_paid) AS avg_spent_per_transaction,
        RANK() OVER (ORDER BY SUM(it.amount_paid) DESC) AS rank
      FROM ItemTransactions it
      WHERE it.date_completed BETWEEN '2024-01-01' AND '2024-12-31'
      GROUP BY it.buyer_email
    )
    SELECT
      buyer_email,
      transaction_count,
      total_spent,
      avg_spent_per_transaction
    FROM BuyerSpending
    WHERE rank <= 10
    ORDER BY rank;
```

4.39 Populating ItemTransactions
Description: This query generates 45 random item transactions by selecting random items and buyers, ensuring the buyer is not the seller, and using the price from the ItemsForSale table.
Parameters (%s): None

```
INSERT INTO ItemTransactions (item_id, buyer_email, seller_email, amount_paid,
date_completed)
SELECT
   random_transactions.item_id,
   random_transactions.buyer_email,
   random_transactions.seller_email,
   random_transactions.price AS amount_paid,
   NOW()
FROM (
   SELECT
      i.item_id,
      u1.email AS buyer_email,
      u2.email AS seller_email,
      i.price
   FROM
      ItemsForSale i
   CROSS JOIN
      Users u1
   INNER JOIN
      Users u2 ON u2.email = i.seller_email
   WHERE
      u1.email != u2.email
   ORDER BY
      RANDOM()
   LIMIT 45
) AS random_transactions;
```

4.40 Populating FreelanceTransactions

Description: This query generates 45 random freelance transactions by selecting random freelance jobs and buyers, ensuring the buyer is not the worker, and using the payment from the FreelanceWork table.
Parameters (%s): None

```
INSERT INTO FreelanceTransactions (job_id, buyer_email, worker_email, amount_paid,
date_completed)
SELECT
   random_jobs.job_id,
   random_jobs.buyer_email,
   random_jobs.worker_email,
   random_jobs.payment AS amount_paid,
   NOW() AS date_completed
FROM (
   SELECT
      fw.job_id,
        u1.email AS buyer_email,
        u2.email AS worker_email,
        fw.payment
    FROM
        FreelanceWork fw
    CROSS JOIN
        Users u1
    INNER JOIN
        Users u2 ON u2.email = fw.posted_by
    WHERE
        u1.email != u2.email
    ORDER BY
        RANDOM()
    LIMIT 45
) AS random_jobs;
```

4.41 Populating ItemRatings
Description: This query generates item ratings based on completed transactions, with random ratings between 4.00 and 5.00 and a 50% chance of no review.
Parameters (%s): None

```
INSERT INTO ItemRatings (item_id, rater_email, rating_email, rating, review, created_at)
SELECT DISTINCT
    it.item_id,
    it.buyer_email AS rater_email,
    it.seller_email AS rating_email,
    ROUND((4 + RANDOM())::NUMERIC, 2) AS rating,
    CASE
        WHEN RANDOM() < 0.5 THEN NULL
        ELSE 'Great transaction! Would recommend.'
    END AS review,
    it.date_completed + INTERVAL '1 day'
FROM
    ItemTransactions it
ON CONFLICT DO NOTHING;
```

4.42 Populating FreelanceWorkRatings
Description: This query generates freelance work ratings based on completed freelance transactions, with random ratings between 4.00 and 5.00 and a 70% chance of review.
Parameters (%s): None

```
INSERT INTO FreelanceWorkRatings (freelance_job_id, rater_email, rating_email, rating,
review, created_at)
SELECT DISTINCT
```

```
    ft.job_id,
    ft.buyer_email AS rater_email,
    ft.worker_email AS rating_email,
    ROUND((4 + RANDOM())::NUMERIC, 2) AS rating,
    CASE
        WHEN RANDOM() < 0.7 THEN 'Great job! Would hire again.'
        ELSE NULL
    END AS review,
    ft.date_completed + INTERVAL '2 days'
FROM
    FreelanceTransactions ft;
```

5 Applications

The application to support my use cases is a platform called "SWAP" that connects users to buy and sell items as well as post and find freelance services. The application allows users to log in, manage their profiles, browse available items and listings, record transactions, communicate through messages, rate items and users, and access platform analytics. Integrating multiple modules for each feature and utilizing the PostgreSQL database allowed me to build an easy to use platform with efficient data management and real-time transaction processing.

6 Conclusions

For this project, I successfully implemented a platform that allows users to log in, browse items and listings, manage transactions, send and receive messages, and view platform analytics, all while interacting with a PostgreSQL database for real-time data handling.

One of the main challenges I faced was artificially populating the databases for transactions and ratings. This was especially difficult since in these tables, foreign keys reference serial attributes in ItemsForSale and FreelanceWork that change each time items and job listings are artificially entered into the database upon creation of the database. To complete this I made sure to carefully track and manage those keys and performed some research into how to do this the most efficiently. Additionally, organizing the application and structuring the queries in a user-friendly way proved to be tricky, especially making sure that I was passing all the necessary attributes to functions for proper query execution. Through this, I gained valuable experience in database design and query management. If given more time, I would build out the platform’s features even further by adding a verification aspect for joining university communities and allowing users to view listings in nearby cities within a certain radius. I would also transfer the project into a GUI for a more intuitive user experience.