1. Please list out changes in the directions of your project if the final project is different from your original proposal (based on your stage 1 proposal submission).

We successfully achieved the goals set in the first stage of our project. We built a Digital Food Management System where donors can share surplus food, and NGOs or needy individuals can book and receive it. We also implemented a Pair Matching Personalization System as a creative component that uses previous bookings and food types to match donors with recipients, along with a Ratings and Reviews feature for recipients to provide feedback. However, the notification system still remains part of the future scope as we didn't have actual emails and contact numbers.

2. Discuss what you think your application achieved or failed to achieve regarding its usefulness.

Our application successfully achieved the goals outlined in the initial proposal. Here are the key benefits the platform delivers, as we originally thought of:

- We built a platform where donors can share extra food, helping reduce waste and support those in need.
- The platform allows people in need to book surplus food, improving access and reducing hunger.
- By reducing food waste, the platform lowers the environmental impact of producing, processing, and disposing of food. This will help with better waste management and cut down the carbon footprint.
- The platform will reduce costs linked to food waste, like disposal fees and the loss of unsold goods, making the food system more efficient.

3. Discuss if you changed the schema or source of the data for your application

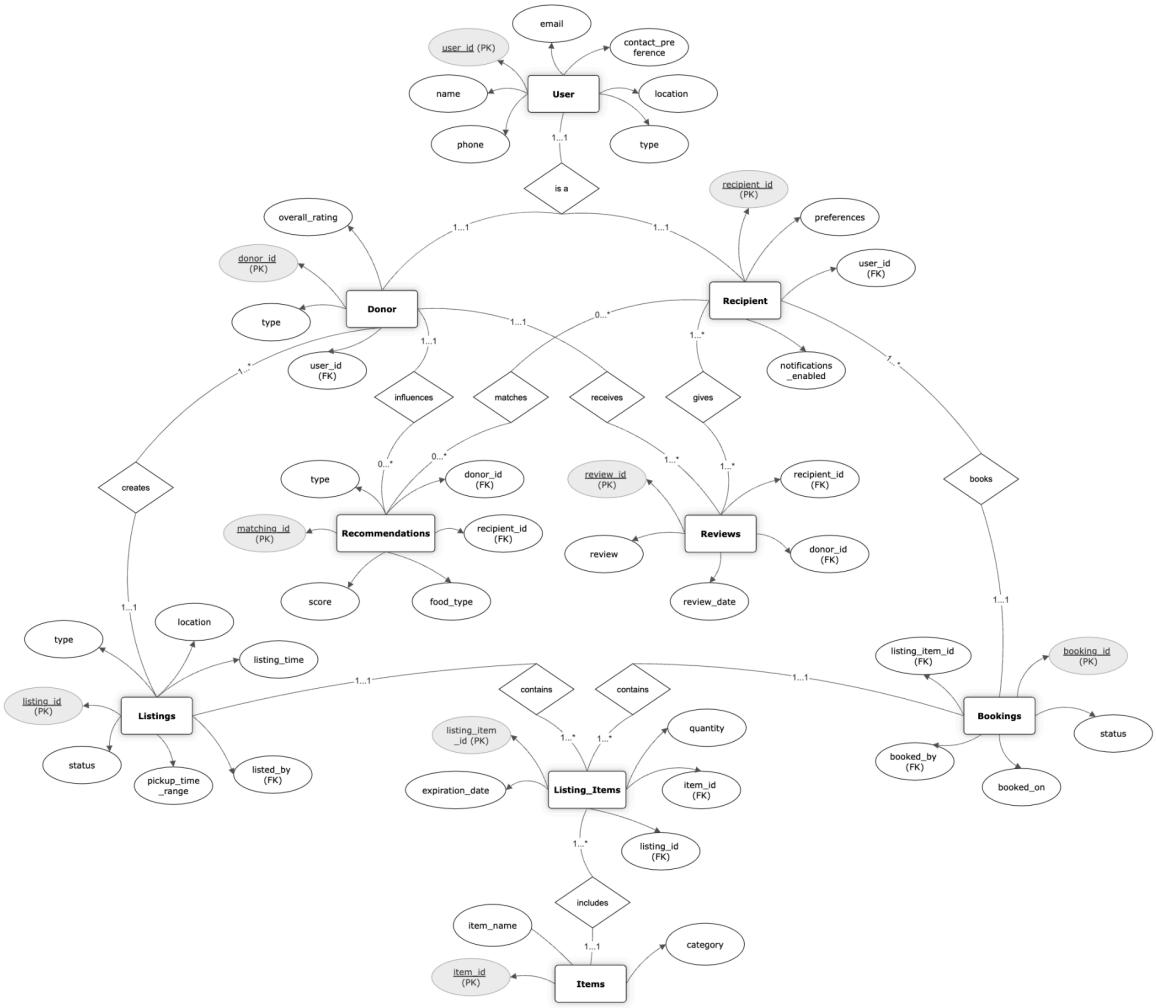
We faced the need to modify both the schema and the source of data to better align with application requirements at multiple stages during the development. Initially, our database schema lacked a dedicated table for recommendations, which was critical for implementing the donor-recipient matching algorithm. To address this, we introduced a Recommendation table that stores pre-computed matching scores based on several parameters. This allowed us to efficiently fetch matches using advanced SQL queries. Another significant change involved restructuring the ListingItem table to include a status column for tracking item availability and an additional foreign key linking it to the Booking table. This allowed us to better manage the lifecycle of donated items, from listing to booking, ensuring accurate tracking of active and completed donations. On the data source side, we initially relied on synthetic datasets for testing but later integrated real-world datasets from public sources like Kaggle and ReFED. These datasets provided insights into food donation trends and surplus metrics, enabling us to refine our matching algorithm. However, integrating these datasets required extensive data cleaning and transformation to ensure compatibility with our schema.

For future teams, we recommend designing a flexible schema that accommodates evolving requirements and planning for potential data integration challenges early in the project. Additionally, pre-computing resource-intensive operations like recommendation scores can significantly improve system

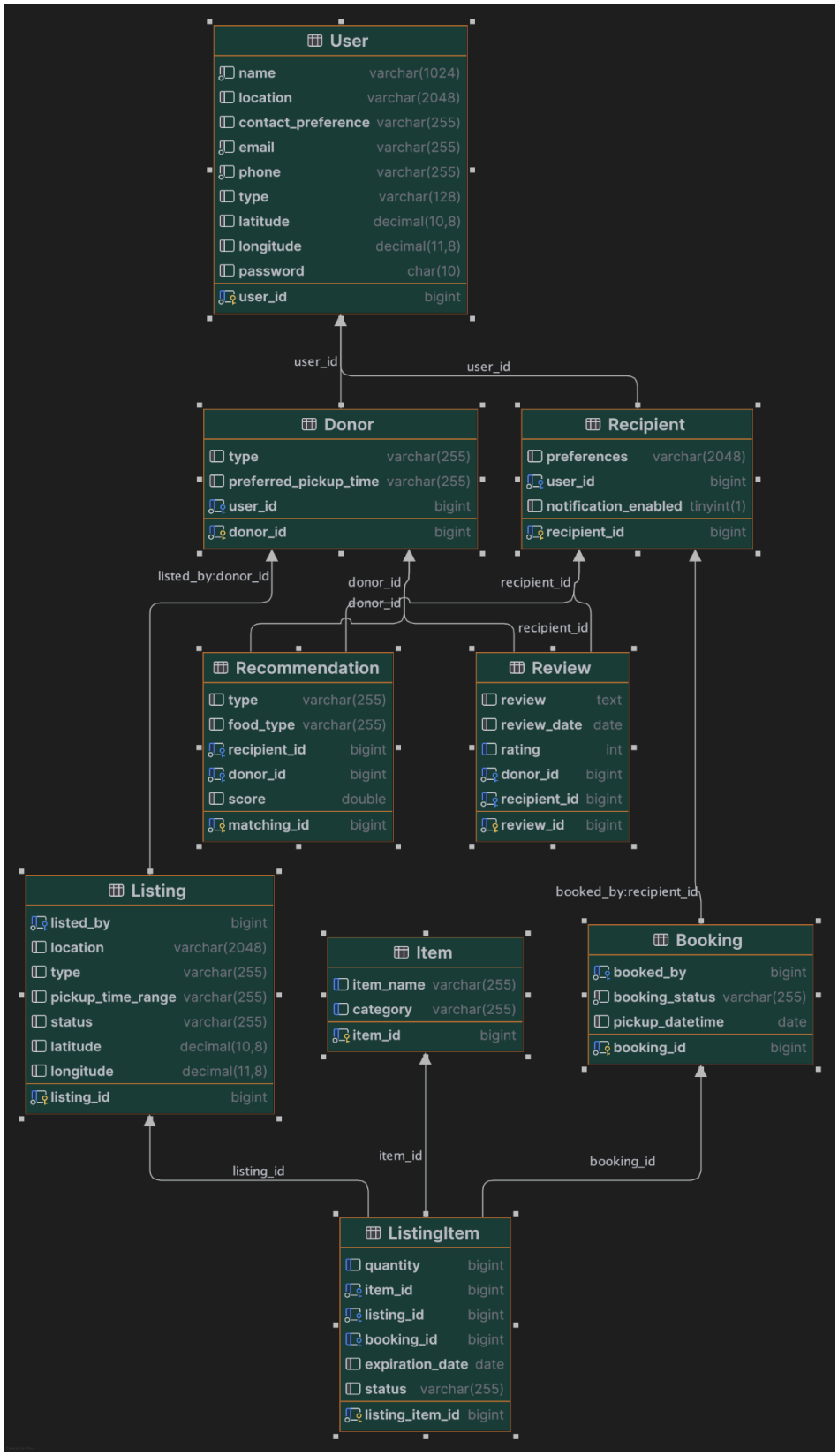
performance. These changes not only enhanced our application's functionality but would also help ensure scalability and efficiency in addressing food donation needs.

4. Discuss what you change to your ER diagram and/or your table implementations. What are some differences between the original design and the final design? Why? What do you think is a more suitable design?

Earlier Design that was reported in Stage 2



Finally implemented ERD of the Food Forward Application:



In User table, we added latitude and longitude to better filter out the listings based on the recipient's location and to calculate the distance from the recipient to the donor's location mentioned in the listing.

Added additional parameters like preferred_pickup_time to make the donor profile more customizable yet restrictive to reduce complications and also to improve the look of the dashboard. We've removed the ratings column from the donor table as its a dynamic value and ratings are specific to individual reviews rather than being a static property of the donor. This ensures that ratings can be tied to multiple reviews, and it'll become a more accurate and granular evaluation of donor performance over time.

Added additional parameter in bookings table to include pickup_datetime as well so that we can show the booking details with information on when the pickup is going to take place.

5. Discuss what functionalities you added or removed. Why?

Following are the set of functionalities that we added in our implementation:
Creation of User: This functionality allows both donors and recipients to register and create an account on the platform. During registration, users provide essential details such as their name, email address, and role (donor or recipient). This feature is fundamental because it serves as the first step for users to access the platform and interact with other users. We implemented this functionality to ensure that only authenticated users could create listings, make donations, or book donations. By adding the user creation feature, we also laid the foundation for managing user data, which can later be used for personalized interactions, such as saving preferences or managing past donations. This feature is essential for tracking user activities and maintaining a secure, organized platform where donors and recipients can trust the system.

Donor Side Functionalities:

1. **View Donor Profile:**
   This functionality allows donors to view and manage their personal profile, including contact information and past donations. We added this feature to maintain up-to-date donor information, ensuring the system remains transparent and trustworthy, especially in cases where donors need to be contacted about their listings.
2. **Create and Update Listing and Items:**
   Donors can create a new listing with details about the food they want to donate, such as the item type, quantity, expiration date, and any other relevant details. When a donor starts searching for an item, the item names that best match the donor input query, will be suggested to the donor. When a donor adds an item from the search list, the food category is automatically detected for the donor. Additionally, donors can update these listings if they need to adjust the food availability or change other details. This functionality is crucial because it provides donors with the flexibility to manage their donations easily, ensuring that the available listings are accurate and up-to-date for recipients to find.

3. **View Donations:**
   Donors can view all their active donation listings, showing a summary of what items are available for donation and their current status. This allows donors to track their contributions and manage the availability of different food items. By adding this feature, we wanted to give donors a centralized place where they could monitor and manage their listings, ensuring that no donation is overlooked.
4. **View Previous Bookings:**
   This feature allows donors to view a history of previous donations, including any bookings made by recipients. It helps donors see which items were donated, to whom, and when the donations were made. We added this feature to enhance transparency, allowing donors to track past transactions and build trust within the platform.
5. **View Reviews:**
   Donors can see reviews left by recipients who have booked and received donations from them. The donor can see the name of the reviewer, ratings given by the reviewer and the actual review. These reviews provide valuable feedback on the donation process and help donors understand how their contributions are impacting recipients. This functionality was added to foster trust and accountability within the platform, encouraging better donation practices.

Recipient Side Functionalities:

1. **View Recipient Profile:**
   Recipients can view their profile created during account creation, which includes their preferences, contact details, and location information. This allows them to customize their profile according to their needs and make it easier for donors to find them based on their preferences. By including this feature, we aimed to create a personalized experience for recipients and ensure that their preferences are clearly communicated to potential donors.
2. **Search and Filter Listings:**
   This functionality allows recipients to search for available donation listings based on specific criteria such as food type, expiration date, and location. The best suitable donor is recommended to the Recipient based on the recipient's applied filters. This will help recipients find the most relevant donations, ensuring that they receive the food they need in a timely manner. This feature was added to streamline the process of finding food listings, saving time and making the platform more efficient for recipients.
3. **Recommend Donors:**
   The recommendation system suggests donors to recipients based on their preferences and past activity. This feature helps recipients discover donors who are likely to offer donations that match their needs. The reason for adding this feature was to enhance the user experience, making it easier for recipients to connect with the right donors, improving the likelihood of successful donations.
4. **View and Book Donor Listings:**
   Recipients can browse detailed donation listings and select the ones that best suit their needs. Once they find a matching listing, they can book the donation, which triggers the donor to

prepare the food for pickup. This functionality was essential for allowing recipients to take action on the listings they find suitable, ensuring a clear and actionable path for receiving donations.

5. **Add Review to Booking:**
   After receiving a donation, recipients can leave a review for the donor, sharing their experience with the donation process. Reviews help future recipients make informed decisions and contribute to building a community based on trust. This feature was added to provide a feedback loop that encourages transparency and helps maintain a quality experience for all users on the platform.

Additionally, We implemented advanced queries including - Transactions, Procedure, Trigger and Constraint to improve the overall database system.

6. Explain how you think your advanced database programs complement your application.

In our project, the advanced database programs go hand-in-hand with the application by addressing key challenges in data handling, matching, and optimization. For example, the donor-recipient matching algorithm relies on advanced SQL queries to match donors and recipients based on parameters like proximity, food type, and urgency. This is achieved using a combination of joins, nested subqueries, and aggregations that leverage the Recommendation table to pre-compute matching scores. Additionally, our database programs include complex queries like Listing Status Summary, which provides donors with insights into their active and completed listings by aggregating data from multiple tables. This helps donors track their contributions effectively. Similarly, Fetch Available Listings by Recipient Preferences uses recipient preferences to filter listings dynamically while ensuring expired or unavailable items are excluded. These queries utilize indexing strategies to optimize performance. The integration of stored procedures further complements the application by automating repetitive tasks like fetching detailed donor profiles or managing booking statuses. These procedures ensure consistency and reduce the likelihood of errors in data retrieval or updates. By hosting the database on GCP, we ensure scalability and reliability for handling large datasets. The cloud-based infrastructure supports real-time updates, enabling features like notifications for new matches or urgent pickups. To conclude, our advanced database programs form the backbone of the application by enabling efficient data management, personalized recommendations, and seamless user interactions. Future teams can build upon this foundation by further optimizing queries and exploring additional indexing strategies to enhance scalability.

7. Each team member should describe one technical challenge that the team encountered. This should be sufficiently detailed such that another future team could use this as helpful advice if they were to start a similar project or where to maintain your project.

Sai Prakash Gorti -

We underestimated the number of endpoints that would be required to build all the screens we had thought of during the mockup stage. Keeping track of potentially what all endpoints would be required to build a particular screen before hand would definitely help in the later stages. It'll also help keep track of the progress. While building the procedure, it was crucial to follow the flow of the application's requirements and look out for potential edge cases. For example, in a database procedure like GetBookingDetails, ensuring that all relevant data is fetched in one go while maintaining performance can be challenging. We had complex joins, filtering conditions, and aggregations that can make the procedure difficult to debug and optimize.

Sai Krishna Rohith Kattamuri -

Another technical challenge was designing and optimizing the donor-recipient matching algorithm to ensure efficient food distribution. The algorithm needed to consider multiple parameters, such as proximity, food category, etc, while maintaining fairness and scalability. Initially, the complexity of handling diverse data points from multiple database tables (e.g., Listing, Item, Recipient) resulted in slow query performance due to inefficient joins and filtering. Additionally, ensuring equitable distribution avoiding repeated matches for the same recipients added another layer of complexity. To address these challenges, we implemented a periodic bulk update to a Recommendations table that kept track of scores and prioritized matches based on recipient needs and donor availability. This helped minimize real-time computation overhead. To improve performance, we optimized database queries by creating indexes on frequently queried fields like item_name and status.

Omkar Dhekane

During the development of creating Listing from the donor side, our team faced a technical challenge when database constraint violations were not properly handled, leading to vague error messages that confused users. The database had a **CHECK** constraint to validate item quantities, ensuring they were within a valid range. However, when users attempted to create listings with invalid quantities (e.g., negative or zero), the backend threw a generic SQL error with an unclear message. This lack of detailed feedback made it difficult to provide users with meaningful error messages. To address this, we enhanced error handling on the front end by checking the backend's error message for specific keywords chk_valid_quantity**,** allowing us to alert the user with a more informative message about the quantity constraint violation. We also recommended backend improvements to provide more specific error codes or messages related to constraints. Additionally, we implemented frontend validation to prevent users from submitting invalid data in the first place. The lessons learned were clear: always ensure both frontend and backend provide specific, actionable error messages, and perform input validation early to minimize user frustration. For future teams, it's crucial to standardize error handling and validation across the system to ensure a smooth user experience and avoid unnecessary confusion.

Nikunj Agarwal

One of the challenges we encountered early on was implementing triggers and procedures directly through our code. In class, we primarily worked with triggers and procedures by writing SQL statements directly in the database, which was straightforward but different from how we needed to approach it in

our application. For our project, we had to first determine what procedures and triggers were appropriate for our specific requirements. This involved carefully analyzing the application's functionality and identifying where triggers would automate tasks or maintain data consistency. Once we decided on the necessary triggers and procedures, the next challenge was implementing them in Java. Since we chose not to use an Object-Relational Mapping (ORM) framework like Hibernate, we relied on the JDBC Template for database interactions. This meant manually writing SQL queries and ensuring proper integration between the Java code and the database. It required extra effort to manage connections, handle exceptions, and ensure our triggers and procedures worked seamlessly within the application.

8. Are there other things that changed comparing the final application with the original proposal?

In our original proposal, we included a cloud-based notification system to enhance real-time communication and ensure seamless updates for both donors and recipients. However, we ultimately decided to omit this feature due to time constraints, the lack of real world available test numbers and emails, and the complexity of implementing a multi-channel alert system within our development timeline. Instead, we focused on essential functionalities such as donor and recipient dashboards, booking systems, and search capabilities. While not including this notification system simplified the implementation process, we may consider it for future iterations to improve scalability and user experience.

9. Describe future work that you think, other than the interface, that the application can improve on

1. We can implement composite indexes for frequently queried columns, such as (listing_id, status) in ListingItem to improve the performance of the query when filtering based on search criteria.
2. We can introduce caching to store frequently accessed data like active listings or donor profiles to reduce database load and improve response times.
3. Improve our ML model by taking feedback and retraining based on the ratings and reviews periodically.
4. Using geolocation instead of entering latitude and longitude
5. We can refactor some complex queries by using UNIONs instead of piling up OR operators
6. Use message queues like Amazon SNS for asynchronous tasks like sending notifications or updating recommendation scores
7. Improve the authentication and authorization using SAML SSO
8. Create admin profile and Implement RBAC to restrict access to sensitive features based on user roles (Admin, Donor, Recipient)

10. Describe the final division of labor and how well you managed teamwork
    1. Documentation and records: Sai Krishna Rohith Kattamuri , Omkar Dhekane
        a. This work involves creating design documents for feature modeling, documentation of the code written for future readability, project reports, presentation reports

2. Frontend Development: Omkar Dhekane , Sai Prakash Gorti
    a. This work includes the creation of frontend UI for the application
    b. API handling to link frontend UI to backend endpoints
3. Database Design and Advanced Query Creation:
    a. Creation of databases - deciding entities, fields, relationships: Nikunj Agarwal , Sai Prakash Gorti
    b. Creation of ER Diagrams: Sai Prakash Gorti , Omkar Dhekane
    c. Hosting of databases on the cloud: Sai Prakash Gorti
    d. Record creation and insertion: Sai Krishna Rohith Kattamuri
    e. Advanced queries including triggers, procedures, indexing, optimizations, constraints: Nikunj Agarwal , Sai Prakash Gorti , Omkar Dhekane , Sai Krishna Rohith Kattamuri
4. Backend and Frontend systems:
    a. User sign-in, registration, and profile update functionality, Review: Sai Prakash Gorti , Omkar Dhekane
    b. Food Items Upload and Food Items Display Based on Search: Nikunj Agarwal , Sai Prakash Gorti , Omkar Dhekane
    c. Donor - Recipient Match Recommendation Algorithm: Sai Krishna Rohith Kattamuri

At every stage, work was divided equally, and interlinked work was handed over to people who worked on those items in earlier stages to maintain continuity. Stayed in constant touch asynchronously over text messages and organized meetings periodically to keep tabs on the progress for timely delivery.

TRANSACTION QUERIES

1)

```
START TRANSACTION
INSERT INTO Listing (listed_by, location, latitude, longitude, type,
pickup_time_range, status)
VALUES
   (<donorId>,
    '<location>',
    <latitude>,
    <longitude>,
    '<type>',
    '<pickup_time_range>',
    '<status>');
```

```
SET @listingId = LAST_INSERT_ID();

INSERT INTO ListingItem (item_id, listing_id, quantity, expiration_date, status)
VALUES
   (<itemId1>, @listingId, <quantity1>, '<expiration_date1>', '<status1>'),
   (<itemId2>, @listingId, <quantity2>, '<expiration_date2>', '<status2>'),
   (<itemId3>, @listingId, <quantity3>, '<expiration_date3>', '<status3>');

COMMIT;
```

2)

```
START TRANSACTION;

INSERT INTO User (name, location, email, phone, type, password)
VALUES ('<name>', '<location>', '<email>', '<phone>', '<type>', '<password>');

SET @userId = LAST_INSERT_ID();

INSERT INTO Donor (type, user_id)
VALUES ('<type>', @userId);

COMMIT;
```

3)

```
START TRANSACTION;

INSERT INTO User (name, location, email, phone, type, password)
VALUES ('<name>', '<location>', '<email>', '<phone>', '<type>', '<password>');

SET @userId = LAST_INSERT_ID();

INSERT INTO Recipient (notification_enabled, user_id)
VALUES (<notification_enabled>, @userId);

COMMIT;
```

PROCEDURES

1)

```
create
    definer = root@`%` procedure GetBookingDetails(IN bookingId bigint)
BEGIN
    -- start a transaction to ensure atomicity
    START TRANSACTION;

    -- check if the booking exists
    IF NOT EXISTS (SELECT 1 FROM Booking WHERE booking_id = bookingId) THEN
        -- if the booking does not exist, return a message and rollback
        SELECT 'booking does not exist' AS message;
        ROLLBACK;

        -- check if the booking is canceled
    ELSEIF EXISTS (SELECT 1 FROM Booking WHERE booking_id = bookingId AND
booking_status = 'CANCELLED') THEN
        -- if the booking is canceled, return a message and rollback
        SELECT 'booking has been canceled' AS message;
        ROLLBACK;

    ELSE
        -- retrieve booking details along with recipient information
        SELECT
            b.booking_id AS bookingId,
            b.booking_status AS status,
            b.pickup_datetime AS pickupDate,
            l.location AS pickupLocation,
            l.pickup_time_range AS pickupTime,
            li.quantity AS quantity,
            li.expiration_date AS expirationDate,
            i.item_name AS itemName,
            u.name AS recipientName,
            u.email AS email,
            u.phone AS phoneNumber
        FROM
            Booking b
                JOIN Listing l ON b.booking_id = l.listing_id
                JOIN ListingItem li ON l.listing_id = li.listing_id
                JOIN Item i ON li.item_id = i.item_id
                JOIN User u ON b.booked_by = u.user_id
        WHERE
            b.booking_id = bookingId;
```

```
        -- retrieve a summary of items in the booking
        SELECT
            COUNT(li.listing_item_id) AS totalItems,
            SUM(li.quantity) AS totalQuantity,
            MAX(li.expiration_date) AS latestExpirationDate
        FROM
            ListingItem li
        WHERE
            li.booking_id = bookingId;

        -- retrieve donor information along with aggregated reviews and ratings
        SELECT
            d.donor_id AS donorId,
            u.name AS donorName,
            u.email AS donorEmail,
            u.phone AS donorPhone,
            AVG(r.rating) AS averageRating,
            COUNT(r.review_id) AS totalReviews
        FROM
            Donor d
                JOIN User u ON d.user_id = u.user_id
                LEFT JOIN Review r ON r.donor_id = d.donor_id
                JOIN Listing l ON l.listed_by = d.donor_id
                JOIN Booking b ON b.booking_id = l.listing_id
        WHERE
            b.booking_id = bookingId
        GROUP BY
            d.donor_id;

        -- update the last accessed timestamp in the booking table
        UPDATE Booking
        SET last_accessed = NOW()
        WHERE booking_id = bookingId;

        -- commit the transaction if everything succeeds
        COMMIT;
    END IF;
END;
```

2)

```
create procedure FindListingsWithFilters(IN recipientId int, IN foodType
varchar(255),
                                                    IN quantityNeeded
bigint, IN expiryDate date,
```

```
                                                        IN pickupTimeStart
varchar(255), IN pickupTimeEnd varchar(255),
                                                        IN location
varchar(255), IN distance double,
                                                        IN userLongitude
double, IN userLatitude double)
BEGIN
   SELECT
       l.listing_id AS listingId,
       l.location AS location,
       l.pickup_time_range AS pickupTimeRange,
       l.status AS listingStatus,
       li.listing_item_id AS listingItemId,
       li.quantity AS quantity,
       li.expiration_date AS itemExpirationDate,
       li.status AS itemStatus,
       i.item_id AS itemId,
       i.item_name AS itemName,
       i.category AS category,
       d.donor_id AS donorId,
       u.name AS donorName,
       u.email AS donorEmail,
       u.phone AS donorPhone,
       1 AS priority -- recommendations have higher priority
   FROM Listing l
            JOIN ListingItem li ON l.listing_id = li.listing_id
            JOIN Item i ON li.item_id = i.item_id
            JOIN Donor d ON l.listed_by = d.donor_id
            JOIN User u ON d.user_id = u.user_id
            JOIN Recommendation r ON r.donor_id = d.donor_id AND r.recipient_id
= recipientId
   WHERE
       (foodType IS NULL OR i.category = foodType)
     AND (quantityNeeded IS NULL OR li.quantity >= quantityNeeded)
     AND (expiryDate IS NULL OR li.expiration_date <= expiryDate)
     AND (pickupTimeStart IS NULL OR l.pickup_time_range >= pickupTimeStart)
     AND (pickupTimeEnd IS NULL OR l.pickup_time_range <= pickupTimeEnd)
     AND (location IS NULL OR l.location LIKE CONCAT('%', location, '%'))
     AND (distance IS NULL OR (
       userLongitude IS NOT NULL AND
       userLatitude IS NOT NULL AND
       ST_Distance_Sphere(POINT(l.longitude, l.latitude), POINT(userLongitude,
userLatitude)) <= distance))
     AND l.status = 'ACTIVE'
     AND li.status = 'AVAILABLE'
     AND (li.expiration_date IS NULL OR li.expiration_date > CURRENT_DATE)

   UNION ALL
```

```
   SELECT
       l.listing_id AS listingId,
       l.location AS location,
       l.pickup_time_range AS pickupTimeRange,
       l.status AS listingStatus,
       li.listing_item_id AS listingItemId,
       li.quantity AS quantity,
       li.expiration_date AS itemExpirationDate,
       li.status AS itemStatus,
       i.item_id AS itemId,
       i.item_name AS itemName,
       i.category AS category,
       d.donor_id AS donorId,
       u.name AS donorName,
       u.email AS donorEmail,
       u.phone AS donorPhone,
       2 AS priority -- all other listings with lower priority
   FROM Listing l
            JOIN ListingItem li ON l.listing_id = li.listing_id
            JOIN Item i ON li.item_id = i.item_id
            JOIN Donor d ON l.listed_by = d.donor_id
            JOIN User u ON d.user_id = u.user_id
   WHERE
       (foodType IS NULL OR i.category = foodType)
     AND (quantityNeeded IS NULL OR li.quantity >= quantityNeeded)
     AND (expiryDate IS NULL OR li.expiration_date <= expiryDate)
     AND (pickupTimeStart IS NULL OR l.pickup_time_range >= pickupTimeStart)
     AND (pickupTimeEnd IS NULL OR l.pickup_time_range <= pickupTimeEnd)
     AND (location IS NULL OR l.location LIKE CONCAT('%', location, '%'))
     AND (distance IS NULL OR (
       userLongitude IS NOT NULL AND
       userLatitude IS NOT NULL AND
       ST_Distance_Sphere(POINT(l.longitude, l.latitude), POINT(userLongitude,
userLatitude)) <= distance))
     AND l.status = 'ACTIVE'
     AND li.status = 'AVAILABLE'
     AND (li.expiration_date IS NULL OR li.expiration_date > CURRENT_DATE)

   ORDER BY priority, listingId;
END;
```

-