**Database Systems – Term Project 2023**

Project Name: Airline Management System

Team: Umair Ahmed(24377), Ali Khurram(25250), Ali Iqbal(24529)

**Table of Contents**

**A white paper with black text

Description automatically generated**

**2) Business Scenario**

This database schema represents an Airline Management System designed to handle various aspects of airline operations. It includes features for both customer-facing tasks like booking flights and managing reservations, as well as internal tasks like creating flights, adding aircrafts, managing employee assignments.

Customers can search for flights, book reservations, and make payments. The system tracks flight schedules, availability, and prices. It also allows passengers to manage their reservations and cancel them if needed or update them.

Internally, the system assigns pilots and crew members to flights and tracks their assignments. It also manages aircraft maintenance by assigning engineers to specific tasks and monitoring their progress. The system provides a comprehensive solution for managing all aspects of an airline's operations**.**

**2) Business Rules and Use Cases**

**Users:**

* User accounts are required to book flights and view or cancel their reservations if they are customers.
* User accounts have two types: customer and administrator(root admin with exclusive access to the entire application).
* Email addresses should follow a valid format.
* Passwords must be at least 8 characters long.
* A user can book multiple flights on his/her behalf.

**Reservations:**

* Only one flight can be booked per reservation.
* Payment is required to confirm a reservation else your reservation will be tentative and payment status will be “pending”.
* Confirmed reservations decrease available seats on the flight.

**Flights:**

* Flights have a unique identifier, aircraft assigned, departure and arrival cities, times, duration, seats, status, and price.
* Flight departure and arrival cities must be different.
* Departure time must be before arrival time.
* Available seats must be non-negative.
* Flight price must be non-negative.
* Only scheduled flights are available for booking.

**Passengers:**

* Passenger accounts are created automatically upon booking a flight.
* Passenger information includes name, phone number, email, passport number, date of birth, and gender.

**Payments:**

* Payments are linked to reservations.
* Payment status can be "Successful" or "Failed".
* Payment amount must be equal to the flight price.

**Employees:**

* There will be four types of employees:Manager, Pilot, Cabin crew and Engineer.
* An employee will be managed by single manager while a manager manages multiple employees.
* Pilots, Cabin crews and Engineers should have separate interface for them to view their assignments.
* The employees should be able to login.
* A manager is responsible to give, edit, and mark as “complete” assignments to employees, and should be able to view employees data.
* Engineer will also have his/her assignment start and completion date.

**3) Entity Attributes and Relationship**

### Entities:

1. User:

* Description: Represents a system user, either a customer or an administrator with different privileges and functionalities.
* Attributes:
  + id: (Primary Key) Unique identifier for the user.
  + name: Full name of the user.
  + isAdmin: Boolean flag indicating whether the user is an administrator (TRUE) or a customer (FALSE).
  + email: Unique email address of the user.
  + password: Hashed password for user authentication.
* Multiplicity:
  + One-to-many:
    - Has many Reservations.
    - Has many Payments.

2. Aircraft:

* Description: Represents an airplane used for airline operations.
* Attributes:
  + aircraftid: (Primary Key) Unique identifier for the aircraft.
  + aname: Aircraft model name.
  + totalseats: Total number of seats available on the aircraft.
  + astatus: Current status of the aircraft (e.g., Active, Maintenance, Retired).
* Multiplicity:
  + One-to-many:
    - Has many Flights.
    - Has many EngineerAssignments.

3. Flight:

* Description: Represents a scheduled flight with specific details like departure and arrival cities, times, duration, price, and available seats.
* Attributes:
  + flightid: (Primary Key) Unique identifier for the flight.
  + aircraftid: Foreign key referencing Aircraft.aircraftid.
  + dep: Departure city airport code.
  + arr: Arrival city airport code.
  + deptime: Departure time of the flight.
  + arrtime: Arrival time of the flight.
  + avbseats: Number of available seats remaining on the flight.
  + date: Date of the flight.
  + status: Current status of the flight (e.g., Scheduled, Delayed, Cancelled).
  + duration: Flight duration in minutes.
  + price: Price of a ticket for the flight.
* Multiplicity:
  + One-to-many:
    - Has many Reservations.
  + Many-to-one:
    - Belongs to one Aircraft.

4. Passenger:

* Description: Represents a person who has booked a reservation on a flight.
* Attributes:
  + passengerid: (Primary Key) Unique identifier for the passenger.
  + id: Foreign key referencing User.id.
  + name: Full name of the passenger.
  + phone: Phone number of the passenger.
  + email: Email address of the passenger.
  + passport: Passport number of the passenger (if applicable).
  + dob: Date of birth of the passenger.
  + gender: Gender of the passenger.
* Multiplicity:
  + One-to-one:
    - Belongs to one User.
* Many-to-one: \* Has many Reservations.

5. Payment:

* Description: Represents a payment made for a confirmed reservation.
* Attributes:
  + paymentid: (Primary Key) Unique identifier for the payment.
  + reservationid: Foreign key referencing Reservation.reservationid (optional).
  + id: Foreign key referencing User.id.
  + status: Payment status (e.g., Successful, Failed).
  + amount: Payment amount for the reservation.
  + date: Payment confirmation date.
  + time: Payment confirmation time.
* Multiplicity:
  + One-to-one:
* Belongs to one Reservation.
* Many-to-one:
  + Belongs to one User.

6. Employee:

* Description: Represents an airline employee with a specific job type like pilot, engineer, or crew member.
* Attributes:
  + employeeid: (Primary Key) Unique identifier for the employee.
  + managerid: Foreign key referencing User.id (optional).
  + name: Full name of the employee.
  + type: Employee job type (e.g., pilot, engineer, crew).
  + email: Email address of the employee.
  + password: Hashed password for employee authentication.
* Multiplicity:
  + One-to-many:
    - Has many PilotAssignments.
    - Has many CrewAssignments.
    - Has many EngineerAssignments.

– A manager manages multiple employees, and an employee is managed by one manager

**Relationships:**

1. PilotAssignment:

* Description: Represents an assignment of a pilot to a specific flight.
* Attributes:
  + AssignmentID: (Primary Key) Unique identifier for the pilot assignment.
  + employeeid: Foreign key referencing Employee.employeeid.
  + flightid: Foreign key referencing Flight.flightid.
* Multiplicity:
  + Many-to-one:
    - Belongs to one Employee.
    - Belongs to one Flight.

2. CrewAssignment:

* Description: Represents an assignment of crew members to a specific flight.
* Attributes:
  + AssignmentID: (Primary Key) Unique identifier for the crew assignment.
  + employeeid: Foreign key referencing Employee.employeeid.
  + flightid: Foreign key referencing Flight.flightid.
* Multiplicity:
  + Many-to-one:
    - Belongs to one Employee.
    - Belongs to one Flight.

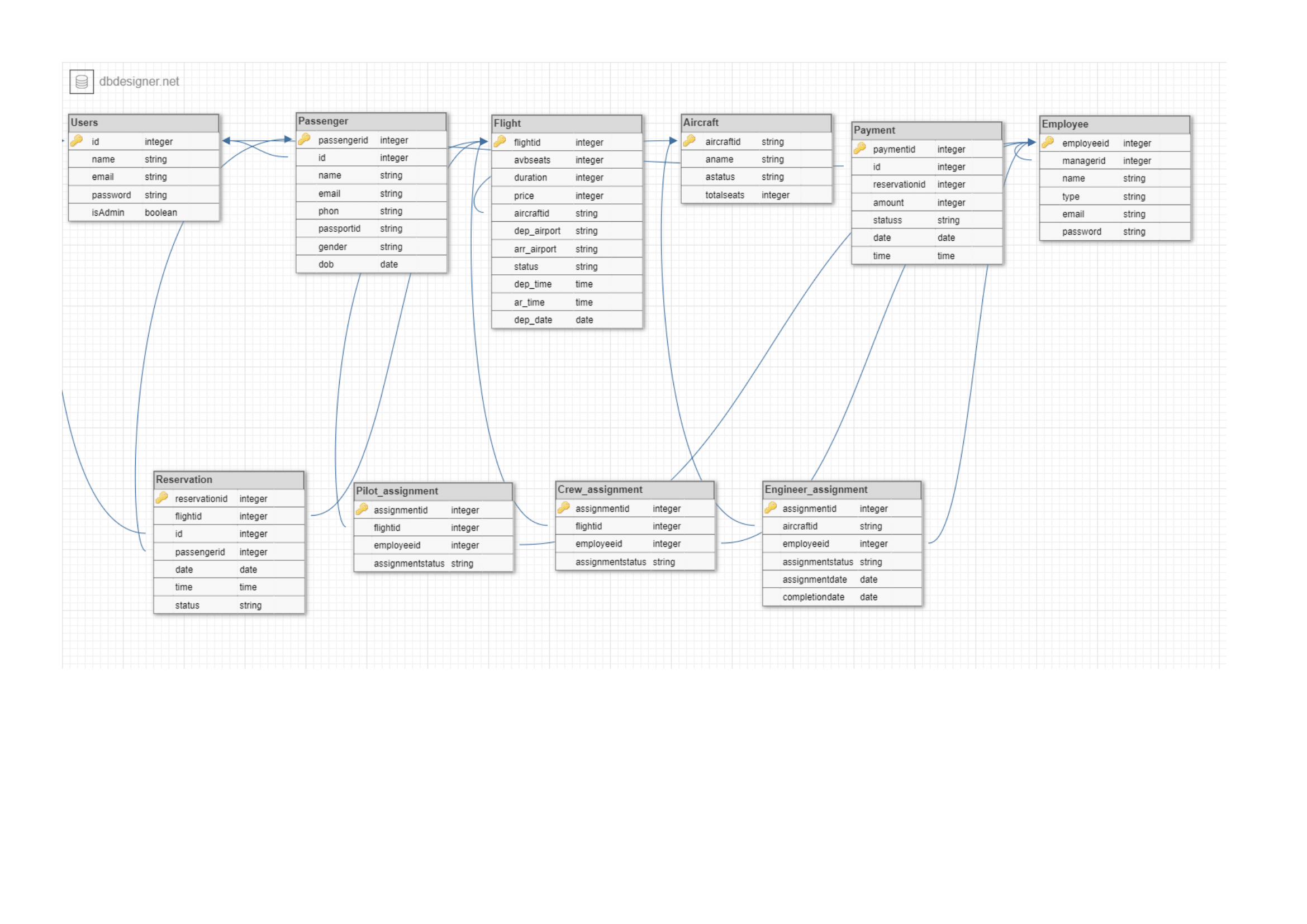
3. EngineerAssignment:

* Description: Represents an assignment of an engineer to an aircraft maintenance task.
* Attributes:
  + AssignmentID: (Primary Key) Unique identifier for the engineer assignment.
  + employeeid: Foreign key referencing Employee.employeeid.
  + aircraftid: Foreign key referencing Aircraft.aircraftid.
  + AssignmentDate: Date the assignment was created.
  + CompletionDate: Date the assignment was completed (optional).
  + AssignmentStatus: Current status of the assignment (e.g., Open, In Progress, Completed).
* Multiplicity:
  + Many-to-one:
    - Belongs to one Employee.
    - Belongs to one Aircraft.

4. Reservation:

* Description: Represents a booking made by a user for a specific flight.
* Attributes:
  + reservationid: (Primary Key) Unique identifier for the reservation.
  + passengerid: Foreign key referencing Passenger.passengerid.
  + flightid: Foreign key referencing Flight.flightid.
  + id: Foreign key referencing User.id.
  + date: Reservation confirmation date.
  + time: Reservation confirmation time.
  + status: Current status of the reservation (e.g., Waiting, Confirmed, Cancelled).
* Multiplicity:
  + One-to-one:
    - Has one Passenger.
    - Has one Payment (optional).
  + Many-to-one:
    - Belongs to one Flight.
    - Belongs to one User.
  + A screenshot of a computer

    Description automatically generated

**5) Relational Schema**

**Normalization Steps:**

**Tables:**

1. USERS
2. Aircraft
3. Flight
4. Passenger
5. Reservation
6. Payment
7. Employee
8. PilotAssignments
9. CrewAssignments
10. EngineerAssignments

**First Normal Form (1NF):**

* Atomic Values: All columns should contain atomic (indivisible) values.
* For all the tables, the columns seem to contain atomic values, so they already meet 1NF requirements.

**Second Normal Form (2NF):**

* Remove Partial Dependencies: No non-prime attribute should be functionally dependent on a part of a composite primary key.
* In this schema, the tables have simple primary keys and don't exhibit composite keys, so there are no partial dependencies to remove.

**Third Normal Form (3NF):**

Remove Transitive Dependencies: No non-prime attribute should be transitively dependent on the primary key.

Let's check each table:

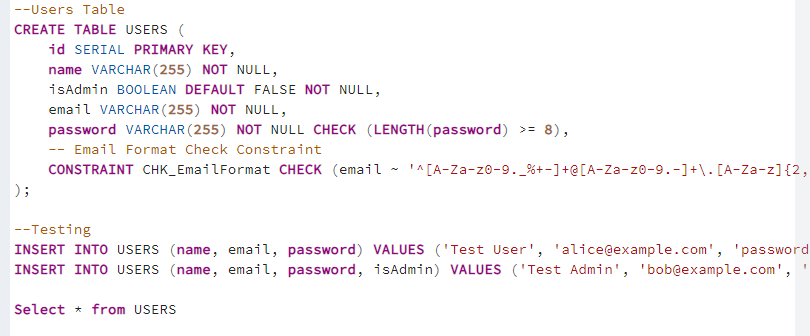
* Users: No transitive dependencies.
* Aircraft: No transitive dependencies.
* Flight: No transitive dependencies.
* Passenger: No transitive dependencies.
* Reservation: No transitive dependencies.
* Payment: No transitive dependencies.
* Employee: No transitive dependencies.
* PilotAssignments, CrewAssignments, EngineerAssignments: These tables exhibit redundancy, but no evident transitive dependencies.

**Conclusion:**

Our schema is in 3NF. Each table represents a unique entity without significant transitive dependencies or other violations of normalization principles.

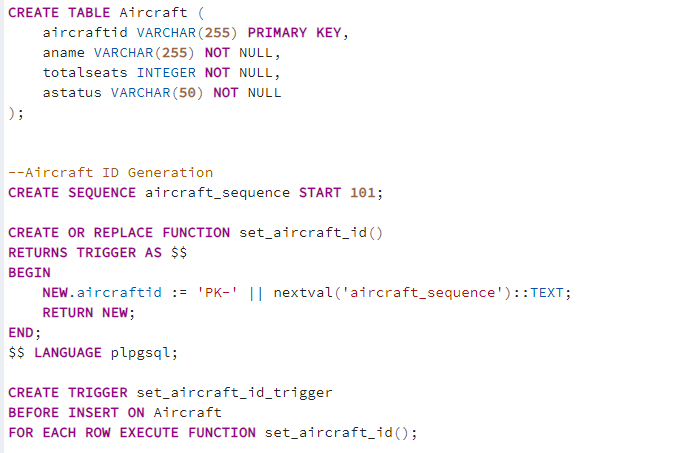
**DDL Script**

1. **Users Table:**



* id: SERIAL data type is used to create an auto-incrementing column for the primary key.
* name: VARCHAR(255) data type is used for a variable-length string with a maximum length of 255 characters. It cannot contain NULL values.
* isAdmin: BOOLEAN data type with a default value of FALSE and cannot contain NULL values.
* email: VARCHAR(255) data type is used for storing email addresses. It cannot contain NULL values, and it is also constrained by the regular expression specified in the CHK\_EmailFormat constraint.
* password: VARCHAR(255) data type is used for storing passwords with a minimum length check of 8 characters.

1. **Aircraft Table:**

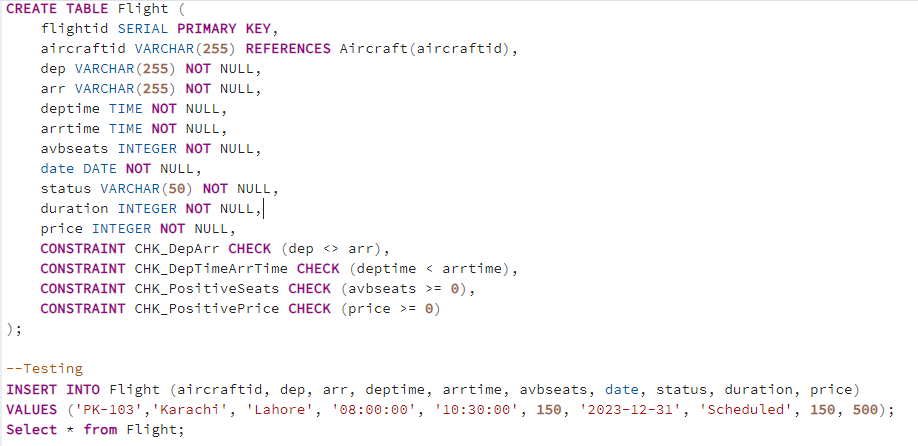
****

**A close up of words

Description automatically generated**

* aircraftid: VARCHAR(255) data type is used for a variable-length string with a maximum length of 255 characters. It is the primary key for the table.
* aname: VARCHAR(255) data type is used for a variable-length string with a maximum length of 255 characters. It cannot contain NULL values.
* totalseats: INTEGER data type is used for storing the total number of seats in the aircraft. It cannot contain NULL values.
* astatus: VARCHAR(50) data type is used for a variable-length string with a maximum length of 50 characters. It cannot contain NULL values.

1. **Flight Table:**

****

* flightid: SERIAL data type is used to create an auto-incrementing column for the primary key.
* aircraftid: VARCHAR(255) data type is a foreign key that references the aircraftid column in the Aircraft table.
* dep and arr: VARCHAR(255) data types are used for departure and arrival locations. They cannot contain NULL values.
* deptime and arrtime: TIME data types are used for departure and arrival times. They cannot contain NULL values.
* avbseats: INTEGER data type is used for storing the available seats. It cannot contain NULL values, and there is a constraint to ensure it's not negative.
* date: DATE data type is used for the flight date. It cannot contain NULL values.
* status: VARCHAR(50) data type is used for a variable-length string with a maximum length of 50 characters. It cannot contain NULL values.
* duration, and price: INTEGER data types are used for storing the flight duration and price, respectively. They cannot contain NULL values, and there are constraints to ensure they are not negative.
* Additional constraints (CHK\_DepArr, CHK\_DepTimeArrTime, CHK\_PositiveSeats, CHK\_PositivePrice) are applied to ensure specific conditions are met.

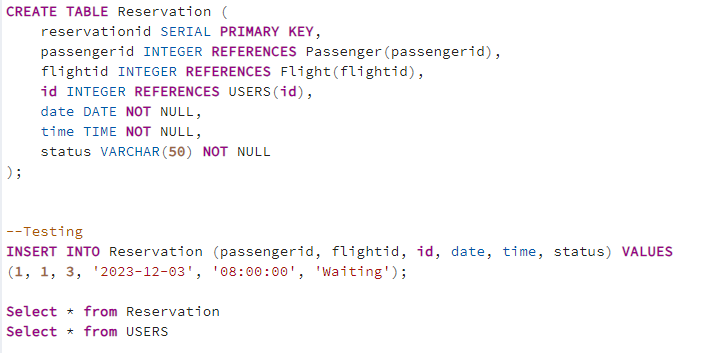
1. **Passenger Table:**

**A computer code with text

Description automatically generated**

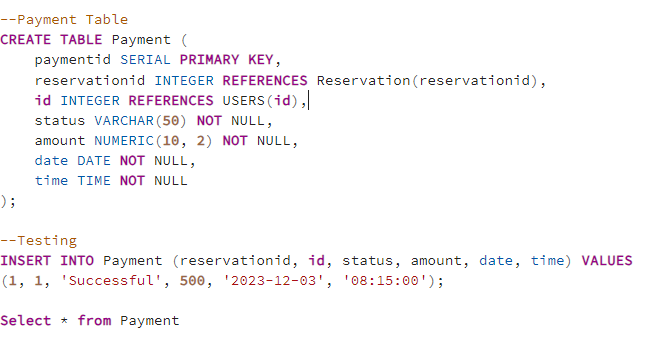
* passengerid: SERIAL data type is used to create an auto-incrementing column for the primary key.
* id: INTEGER data type is a foreign key that references the id column in the USERS table.
* name, phone, email, passport, dob, and gender: VARCHAR and INTEGER data types are used for various attributes of a passenger. They cannot contain NULL values.

1. **Reservation Table:** (Kind of a bridge table between passenger and flight)

****

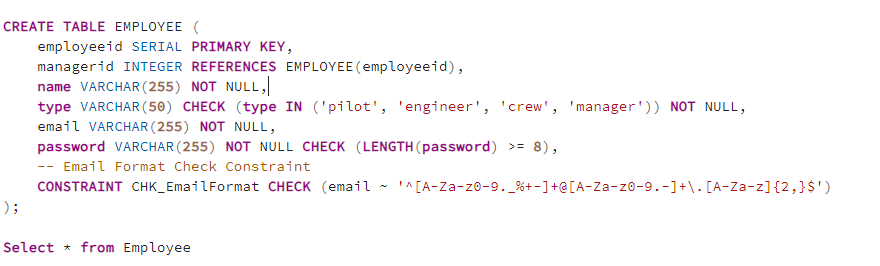
* reservationid: SERIAL data type is used to create an auto-incrementing column for the primary key.
* passengerid: INTEGER data type is a foreign key that references the passengerid column in the Passenger table.
* flightid: INTEGER data type is a foreign key that references the flightid column in the Flight table.
* id: INTEGER data type is a foreign key that references the id column in the USERS table.
* date and time: DATE and TIME data types are used for reservation date and time. They cannot contain NULL values.
* status: VARCHAR(50) data type is used for a variable-length string with a maximum length of 50 characters. It cannot contain NULL values.

1. **Payment Table:**

****

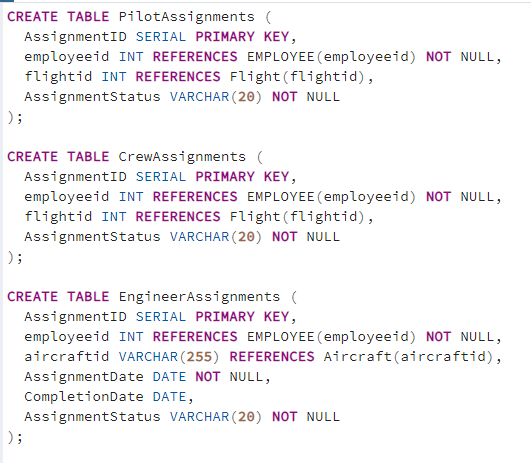
* paymentid: SERIAL data type is used to create an auto-incrementing column for the primary key.
* reservationid: INTEGER data type is a foreign key that references the reservationid column in the Reservation table.
* id: INTEGER data type is a foreign key that references the id column in the USERS table.
* status: VARCHAR(50) data type is used for a variable-length string with a maximum length of 50 characters. It cannot contain NULL values.
* amount: NUMERIC(10, 2) data type is used for storing the payment amount with two decimal places. It cannot contain NULL values.
* date and time: DATE and TIME data types are used for payment date and time. They cannot contain NULL values.

1. **Employee Table:**

****

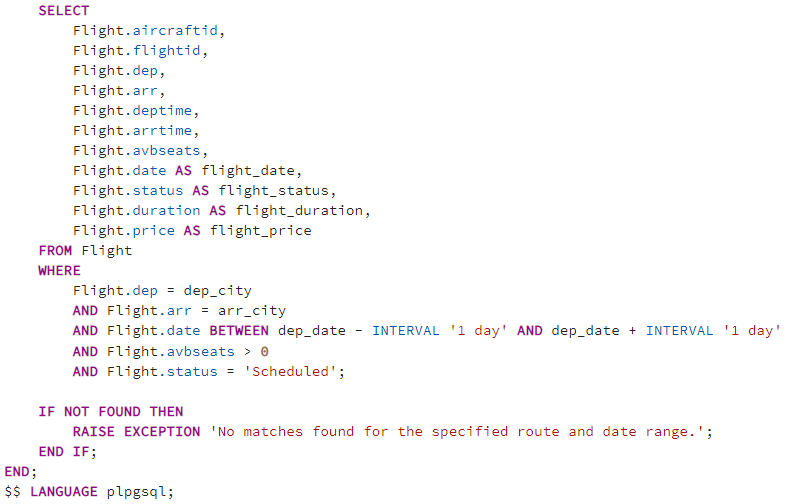
* employeeid: Auto-incrementing primary key.
* managerid: Foreign key referencing employeeid in the same table.
* name: Variable-length string (max 255), cannot be NULL.
* type: Specifies employee type, limited to 'pilot', 'engineer', 'crew', 'manager', cannot be NULL.
* email: Stores email addresses, must adhere to a specified format, cannot be NULL.
* password: Stores passwords, minimum length constraint: 8 characters, cannot be NULL.

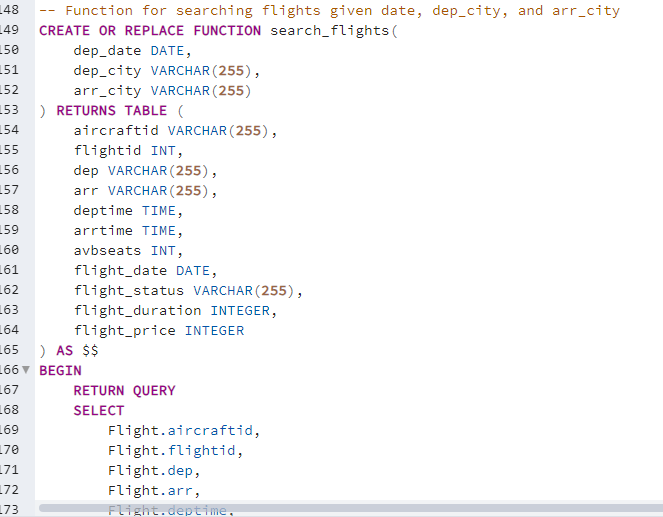
1. **Assignments Tables**

****

* AssignmentID: SERIAL data type is used to create an auto-incrementing column for the primary key.
* employeeid: INT data type, and it is a foreign key that references the employeeid column in the EMPLOYEE table. The NOT NULL constraint ensures that this field cannot contain NULL values.

**Functions**

**1) search\_flights**



Purpose:

Searches for flights based on departure date, departure city, and arrival city.

Parameters:

dep\_date (DATE): Departure date.

dep\_city (VARCHAR(255)): Departure city.

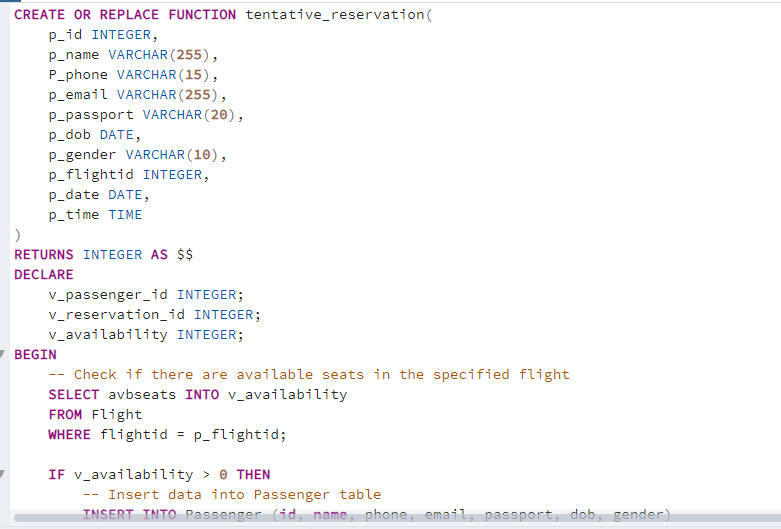
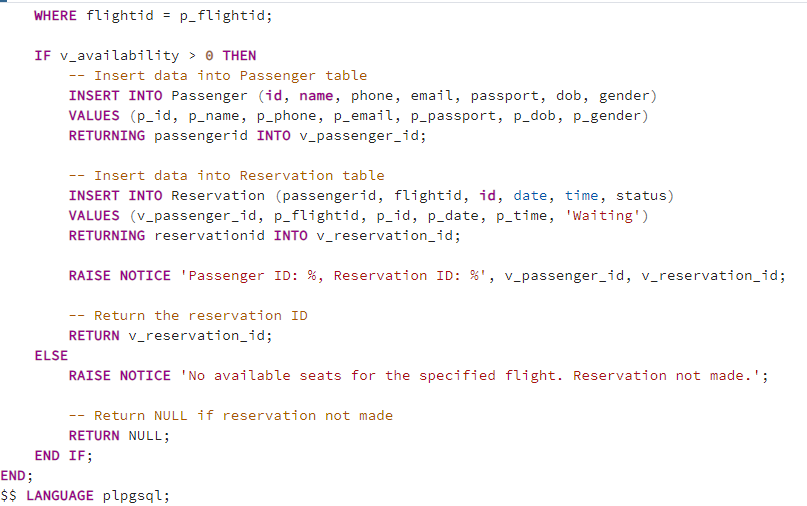
arr\_city (VARCHAR(255)): Arrival city.

Returns:

Table with flight details (aircraftid, flightid, dep, arr, deptime, arrtime, avbseats, flight\_date, flight\_status, flight\_duration, flight\_price).

Constraints:

Filters flights by departure and arrival cities, date range (within 1 day of the given date), available seats, and status 'Scheduled'.

**2) tentative\_reservation**

Purpose:

Attempts to make a tentative reservation with reservation status waiting and also an entry in the passenger for a passenger on a specified flight.

Parameters:

Various passenger details (p\_id, p\_name, p\_phone, etc.).

p\_flightid (INTEGER): Flight ID for the reservation.

p\_date (DATE): Reservation date.

p\_time (TIME): Reservation time.

Returns:

Returns the reservation ID if successful, otherwise NULL.

Actions:

Checks seat availability in the specified flight.

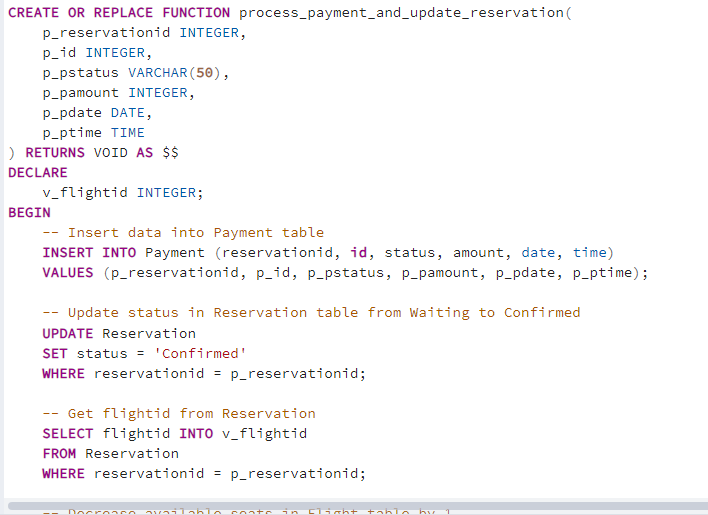
Inserts passenger details into the Passenger table.

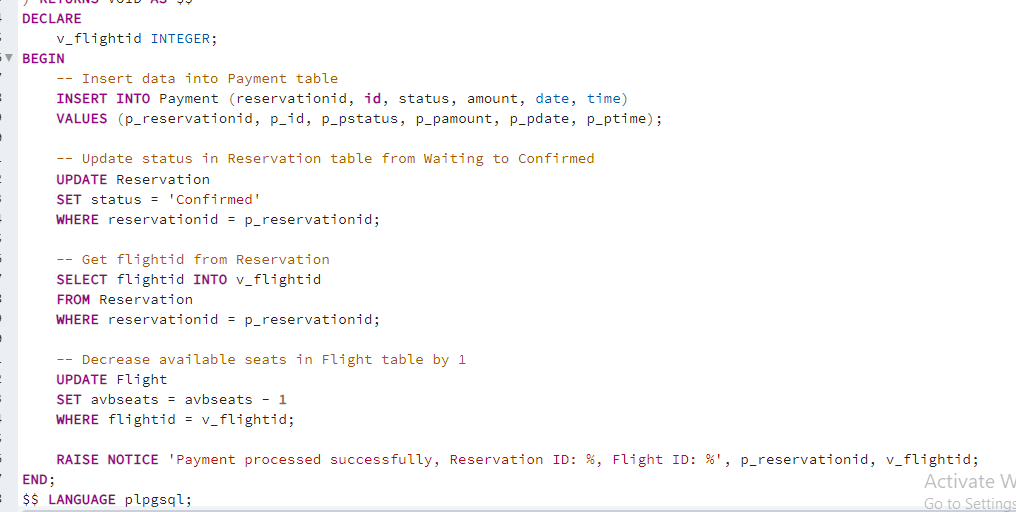
Inserts reservation details into the Reservation table.

Updates available seats in the Flight table.

Outputs Passenger ID and Reservation ID if successful.

**Procedures**

 **1)process\_payment\_and\_update\_reservation**

****

Purpose:

Processes payment and updates the reservation status to 'Confirmed'.

Parameters:

p\_reservationid (INTEGER): Reservation ID.

p\_id (INTEGER): User ID making the payment.

p\_pstatus (VARCHAR(50)): Payment status ('Paid' or other).

p\_pamount (INTEGER): Payment amount.

p\_pdate (DATE): Payment date.

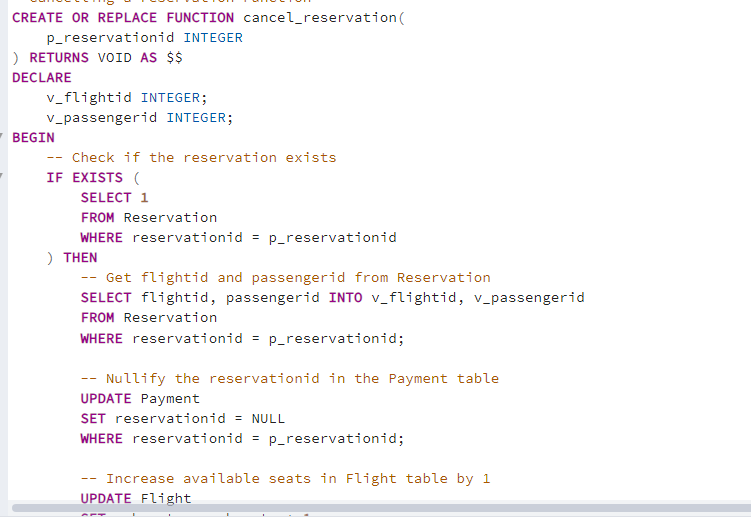
p\_ptime (TIME): Payment time.

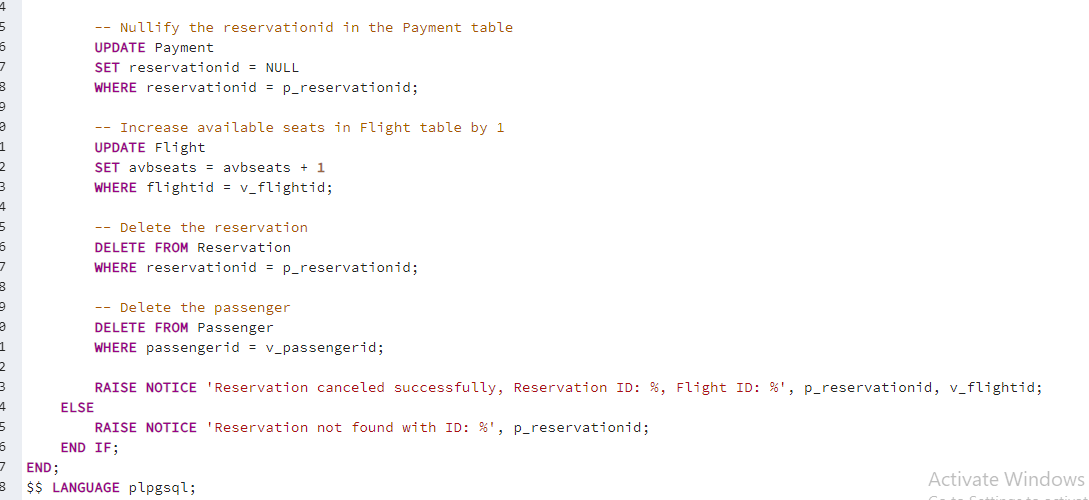
Actions:

Inserts payment details into the Payment table.

Updates reservation status to 'Confirmed'.

Decreases available seats in the corresponding flight by 1.

**2) cancel\_reservation**



Purpose:

Cancels a reservation, updating related tables.

Parameters:

p\_reservationid (INTEGER): Reservation ID to be canceled.

Actions:

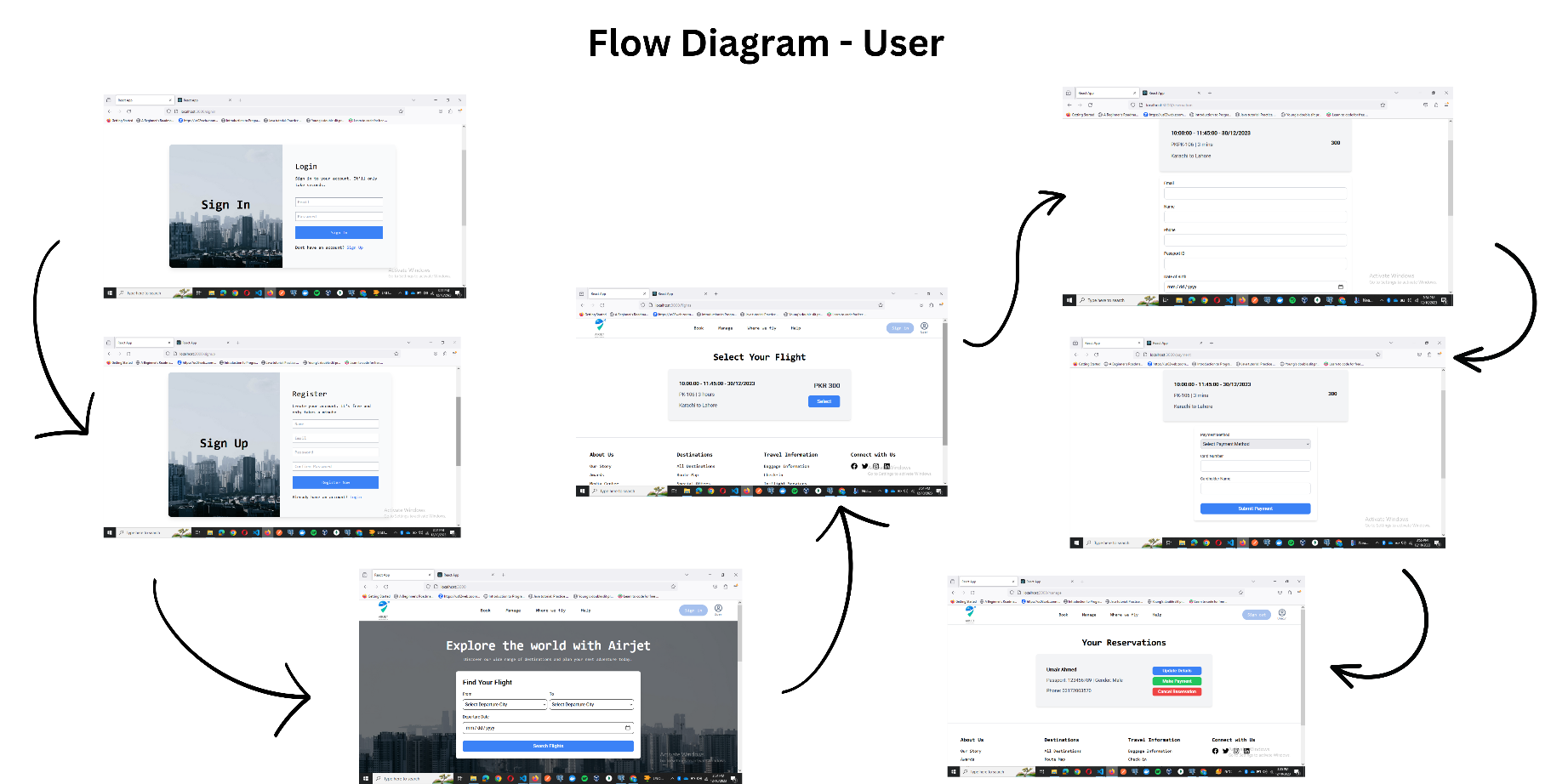
Checks if the reservation exists.

Nullifies the reservation ID in the Payment table.

Increases available seats in the corresponding flight by 1.

Deletes the reservation from the Reservation table.

Deletes the corresponding passenger from the Passenger table.

**6) Application Flow**

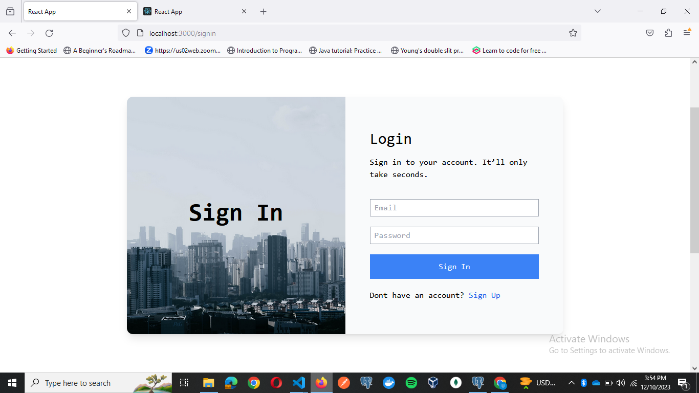
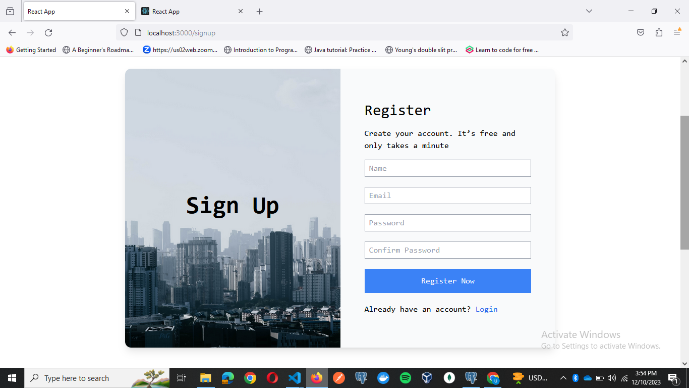
**A screenshot of a computer

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

**7) Page-by-Page Navigation and SQL Queries**

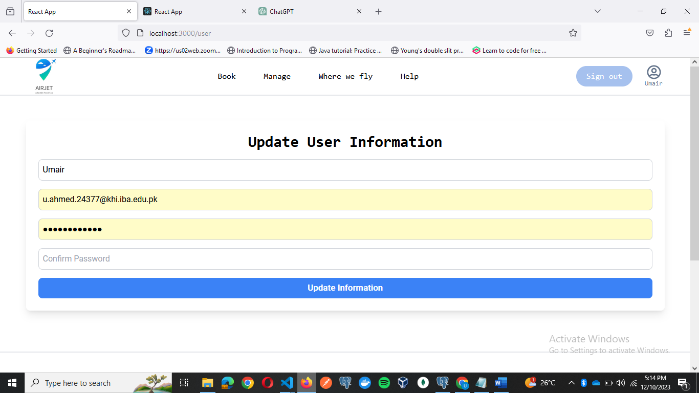
**Login Page/Register Page:**

Main Queries used:

For Register: INSERT INTO users (name, email, password, isadmin) VALUES ($1, $2, $3, $4) RETURNING id, name, email, isadmin

For Login: SELECT \* FROM users WHERE email = $1

We have a Users table which contains the attributes name, email, password and isadmin (Boolean). Users which will have isadmin = true will have global access of the website. In case of register we are inserting into the users table the values taken from the user and in case of log in we are just finding if a user exists by email. If it does then we move on to comparing the password entered by the user.

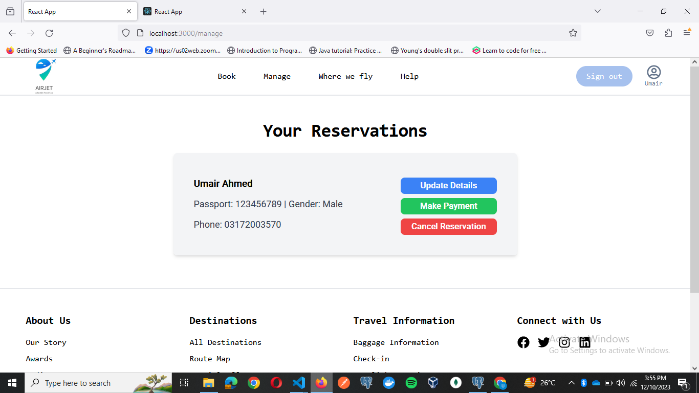
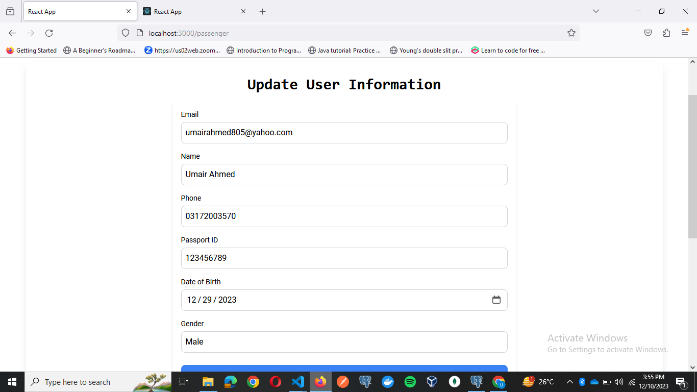
**Update User Details Page:**

Main Queries used:

Used the process\_payment\_and\_update\_reservation function mentioned above in the document.

SELECT process\_payment\_and\_update\_reservation($1, $2, $3, $4, $5, $6)

What it does is taken in the reservationid and first creates an insert in the payments table, then it updates the reservationid in the reservation to mark the status as Confirmed. Then it selects the flightid from that reservationid row and updates the seats in that flight, decrements it.

**Manage Page :**

Main Queries used:

Used the cancel\_reservation function for cancel part:

SELECT cancel\_reservation($1)

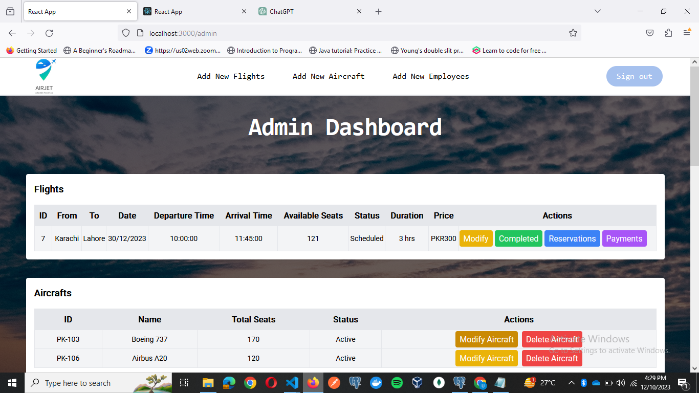
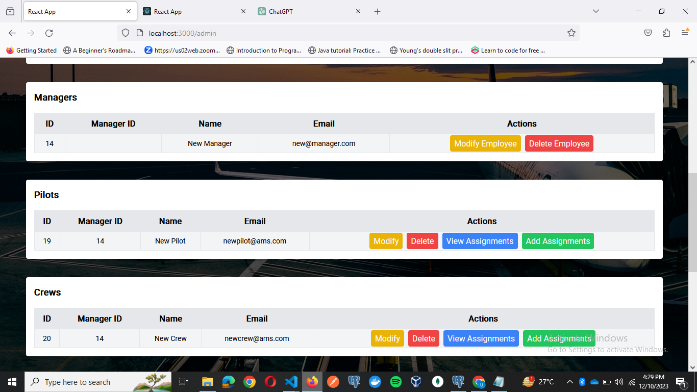
Used this for updating the passenger details:

UPDATE Passenger SET id = $1, name = $2, phone = $3, email = $4, passport = $5, dob = $6, gender = $7 WHERE passengerid = $8 RETURNING \*

The cancel reservation function just takes in the reservationid and cancels the reservation updates the available seats in flight and delete from reservation and payments table.

The make payment button simple does the same which happens in the payment page.

The update passenger button simply updates the passenger details in the passenger table.

**Admin Dashboard Page :**

Main Queries used:

SELECT \* FROM Reservation WHERE flightid = $1 Get All Reservation for a specific flight.

SELECT \* FROM Flight WHERE status = $1 ORDER BY date ASC',['Scheduled'] Displays all flights where status is scheduled

SELECT \* FROM Aircraft Get All Aircrafts in the system currently.

SELECT \* FROM EMPLOYEE where type = \'crew\(gets any of the four types) Get All (Crew/Pilot/Engineer/Manager) Employees

DELETE FROM EMPLOYEE WHERE employeeId = $1 RETURNING Delete an employee

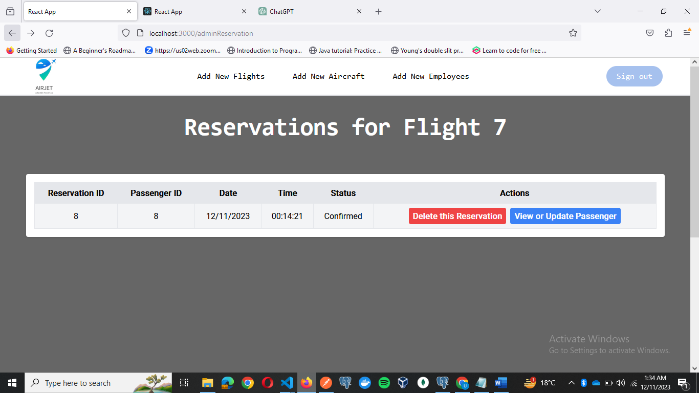
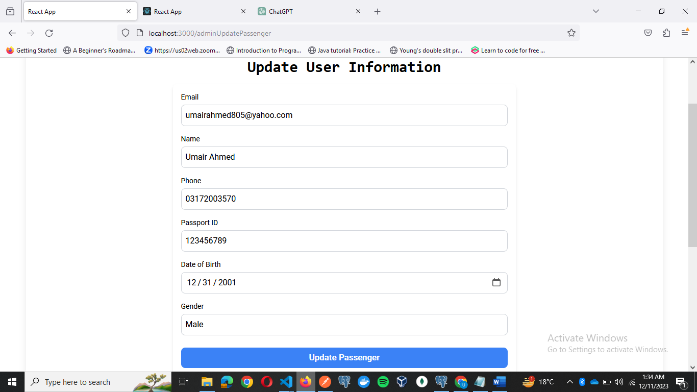
UPDATE Aircraft SET aname = $1, totalseats = $2, astatus = $3 WHERE aircraftid = $4 RETURNING \* Modify an aircraft, can change status from active to non active or vice versa.

DELETE FROM Aircraft WHERE aircraftid = $1 RETURNING \* Delete an aircraft

UPDATE EMPLOYEE SET name = $1, email = $2, password = $3, managerId = $4 WHERE employeeId = $5 RETURNING employeeId, name, email, managerId Modify the employee details such as name email password

UPDATE Flight SET status = $1 WHERE flightid = $2 RETURNING \*',['Completed', flightid] Mark the status of the flight as completed which will remove it from the table

SELECT \* FROM Payment WHERE reservationid IN (SELECT reservationid FROM Reservation WHERE flightid = $1) Get all the payments for a specific flight.

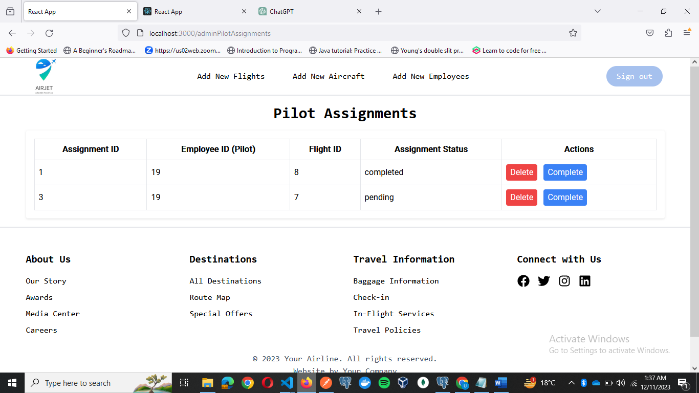
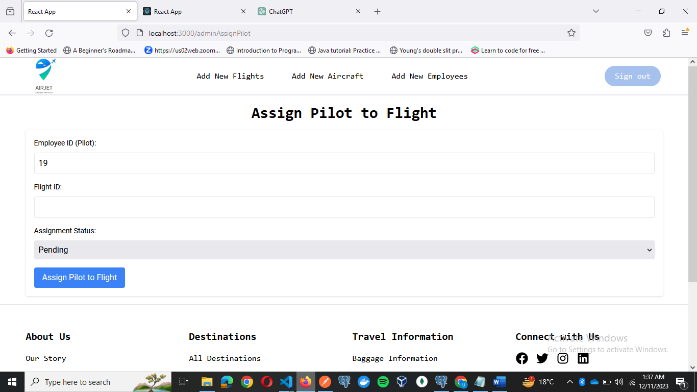
**View Reservations for a flight and view or edit the passenger for that reservation :**

Main Queries used:

SELECT \* FROM Reservation WHERE flightid = $1 Used to get all reservations for a flight

UPDATE Passenger SET id = $1, name = $2, phone = $3, email = $4, passport = $5, dob = $6, gender = $7 WHERE passengerid = $8 RETURNING \* Update the details of a passenger by passengerid

SELECT \* FROM Passenger WHERE passengerid = $1', [passengerid] Viewing the details of a passenger

**View Assignments for an employee and Add assignments for an employee :**

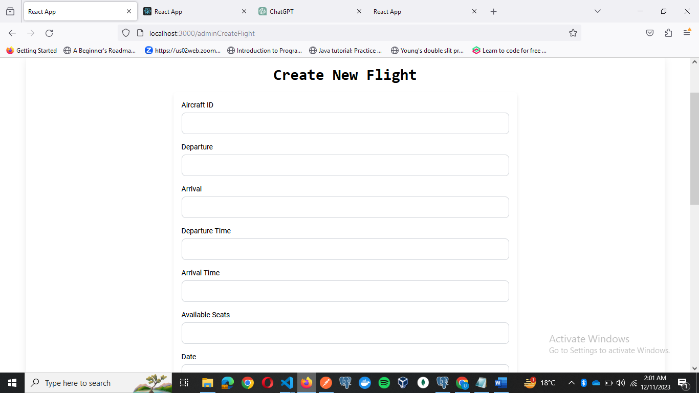
Main Queries used:

SELECT \* FROM PilotAssignments where employeeid = $1', [id] Used to get the assignments for a specific employee(pilot in this case)

INSERT INTO PilotAssignments (employeeid, flightid, AssignmentStatus) VALUES ($1, $2, $3) RETURNING \* Used to create an assignment for an employee (pilot in this case)

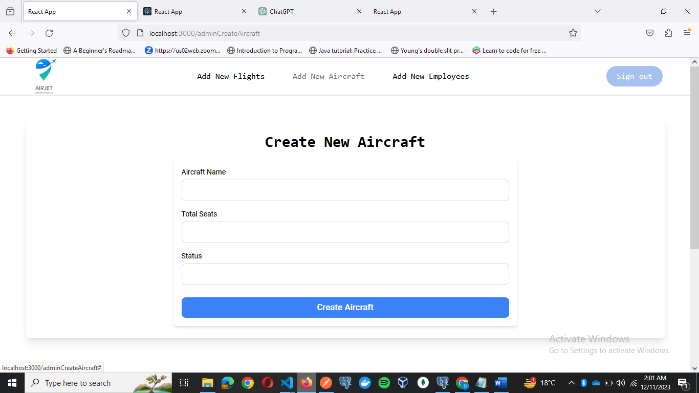
UPDATE PilotAssignments SET AssignmentStatus = $1 WHERE AssignmentID = $2' Used to mark an assignment as completed

DELETE FROM PilotAssignments WHERE AssignmentID = $1 To delete an assignment

**Add a Flight :**

Main Queries used:

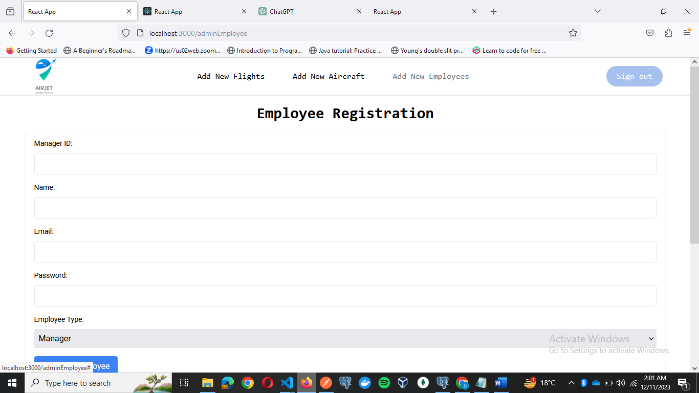
INSERT INTO Flight (aircraftid, dep, arr, deptime, arrtime, avbseats, date, status, duration, price) VALUES ($1, $2, $3, $4, $5, $6, $7, $8, $9, $10) RETURNING \* To create a new flight

**Add an Aircraft :**

Main Queries used:

INSERT INTO Aircraft (aname, totalseats, astatus) VALUES ($1, $2, $3) RETURNING \*

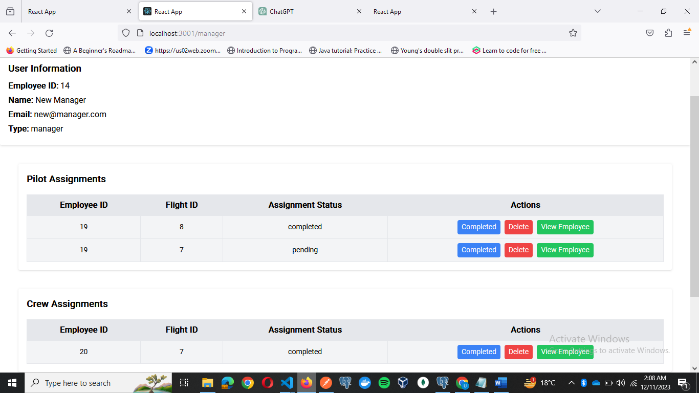
To create a new aircraft

**Add an Employee :**

Main Queries used:

INSERT INTO EMPLOYEE (managerid, name, email, password, type) VALUES ($1, $2, $3, $4, $5) RETURNING employeeId, name, email, type

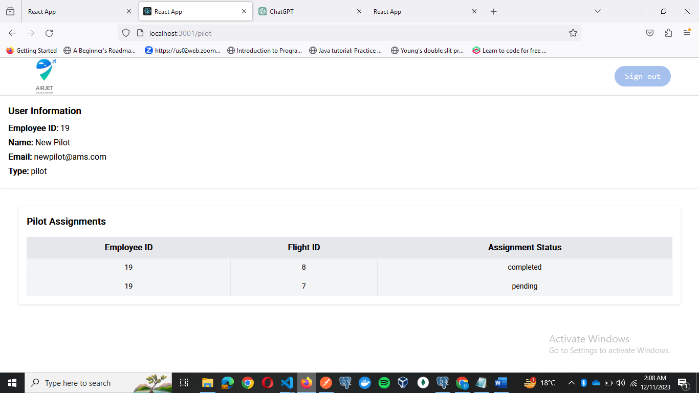
To create a new employee with the relevant type.

**Manager Screen :**

UPDATE PilotAssignments SET AssignmentStatus = $1 WHERE AssignmentID = $2' Used to mark an assignment as completed

DELETE FROM PilotAssignments WHERE AssignmentID = $1 To delete an assignment

UPDATE EMPLOYEE SET name = $1, email = $2, password = $3, managerId = $4 WHERE employeeId = $5 RETURNING employeeId, name, email, managerId View and Modify the employee details such as name email password

**Employee Screen :**

Main Queries used:

SELECT \* FROM PilotAssignments where employeeid = $1', [id]Get assignments for an employee and display it for the employee to view.

**8)Work Breakdown among team Members**

**Ali Iqbal :**

* Designed the Initial Database Schema on OracleDB.
* Backend API Logic Building

**Ali Khurram:**

* Backend API creation and testing.
* Report and Powerpoint Creation

**Umair Ahmed:**

* Converting the database to postgres and fine tuning it according to backend api needs.
* Frontend Design and Backend connection.

1. **Technical Stack Used and Why?**

**PERN Stack Overview:**

The Airline Management System will be developed using the PERN stack, a comprehensive technology stack using PostgreSQL, Express.js, React.js, Node.js, and Redux. Each component of the stack is carefully selected to provide a solid foundation for building a scalable, efficient, and user-friendly application.

**PostgreSQL (Relational Database Management System):**

PostgreSQL is chosen as the relational database management system (RDBMS) for the Airline Management System. Key features that make PostgreSQL a suitable choice include:

* ACID Compliance: PostgreSQL ensures data integrity and reliability through its adherence to ACID (Atomicity, Consistency, Isolation, Durability) properties, making it suitable for transactional applications such as our airline management system.
* Extensibility and Customization: PostgreSQL supports custom data types, functions, and extensions, allowing for the implementation of specific requirements tailored to the airline industry.

**Express.js (Back-End Framework):**

Express.js is chosen as the back-end framework for the Airline Management System. Key attributes include:

* Routing and Middleware: Express.js simplifies the creation of robust APIs and handles middleware functions efficiently, facilitating the implementation of authentication, logging, and other crucial functionalities.
* Modularity: The modular structure of Express.js allows for the organization of code into separate, manageable components, enhancing code maintainability and scalability.

**React.js (Front-End Framework):**

React.js, a JavaScript library for building user interfaces, serves as the front-end framework for the Airline Management System. Key advantages include:

* Component-Based Architecture: React's component-based structure enables the creation of reusable UI components, promoting code reusability and maintainability.
* Virtual DOM: React's virtual DOM optimizes rendering performance, ensuring a smooth and responsive user interface for the airline management application.
* Declarative Syntax: The declarative nature of React simplifies the development process and enhances code readability, making it easier to understand and maintain.

**Redux (State Management):**

Redux is integrated into the front-end architecture as the state management solution. Key features and advantages of using Redux in the Airline Management System include:

* Predictable State Management: Redux follows a predictable state management pattern, facilitating a clear understanding of how the application state changes over time.
* Centralized State:Redux provides a centralized store to manage the state of the application, enabling efficient data flow and reducing the complexity of state management.
* State Immutability:By encouraging the use of immutable data, Redux helps prevent unintended side effects and makes it easier to track and understand state changes.
* Enhanced Debugging:
* The Redux DevTools extension offers powerful debugging capabilities, allowing real-time inspection of state changes and actions for easier troubleshooting.

**Node.js (Runtime Environment):**

Node.js is chosen as the runtime environment for the server side. Key features include:

* Non-blocking I/O: Node.js operates on a non-blocking, event-driven architecture, making it suitable for handling concurrent connections and enhancing the performance of real-time applications like the Airline Management System.
* Package Management: Node Package Manager (NPM) provides a vast ecosystem of packages and modules, streamlining the integration of third-party libraries and tools.

The selection of the PERN stack with Redux for the Airline Management System represents a strategic choice to create a robust, scalable, and user-friendly application. This technology stack combines the strengths of a reliable relational database, a flexible back-end framework, a dynamic front-end library, and a sophisticated state management solution.