DBMS Mini Project Deliverables

Title: Car Sales Management System

TEAM:

SRN: PES1UG22CS319

NAME: MADHUSUDAN M

SRN: PES1UG22CS359

NAME: MOHITH M JOGI

1. Project Overview

An **Online Car Sales Management System** for buying, selling, and managing cars, with features for car listings, orders, test drive bookings, and user profile analytics. The application facilitates a seamless interaction between car owners and potential buyers.

---

2. Core Features

Frontend:

1. **Car Listings:**
    - Users can:
        - Add a car listing with details such as make, model, year, price, and description.
        - Upload multiple images for each car.
        - Edit or delete their listings.
    - Display listings with a search and filter option (e.g., by price, brand, year).
2. **Orders:**
    - Users can place orders for cars.
    - A confirmation message or order summary is displayed upon order placement.
3. **Test Drive Booking:**

- Users can:
    - Request a test drive by filling out a form (name, email, phone number, date, time).
    - Receive confirmation or cancellation from the car owner.
- Car owners can:
    - View all test drive requests for their cars.
    - Accept or cancel requests.

4. **User Profile (Dashboard):**
    - Displays:
        - **Pie Chart** showing:
            - Number of listings.
            - Number of orders.
            - Total income generated.
        - List of active test drive requests with their statuses (pending/accepted/cancelled).
    - Actions:
        - Update account details (e.g., email, password).

Backend:

1. **Database Design:**
    - **Tables:**
        - users: Handles user details and authentication.
        - listings: Stores car details (make, model, year, price, description, user_id, etc.).
        - listing_images: Stores paths to car images.
        - orders: Tracks orders (user_id, listing_id, order_status, total_amount, etc.).
        - test_drives: Manages test drive requests (user_id, listing_id, date, time, status).
    - **Constraints:**
        - Auto-increment IDs.

        - Foreign keys linking users with their listings, orders, and test drive requests.

2. **APIs:**
    - **Listings:**
        - POST /listings: Add a new car listing.
        - GET /listings: Fetch all listings.
        - PUT /listings/{id}: Update a car listing.
        - DELETE /listings/{id}: Delete a car listing.
    - **Orders:**
        - POST /orders: Place an order.
        - GET /orders: Fetch all orders (filtered by user or owner).
    - **Test Drives:**
        - POST /test-drives: Book a test drive.
        - GET /test-drives: Fetch test drive requests (filtered by user or owner).
        - PUT /test-drives/{id}: Update test drive status (accept/cancel).
    - **Dashboard:**
        - GET /dashboard/{user_id}: Fetch statistics for the user dashboard.

Queries:

The /orders route uses SQL queries to handle the creation, retrieval, and management of car sales orders. Key functionalities include:

1. **Order Creation:**
    - Validates listing availability (status = "on sale") and prevents duplicate bookings for the same buyer and listing.
    - Inserts order details into the database and updates the listing's status to "booked."
2. **Order Retrieval:**
    - Fetches a buyer's order history, including listing details and order statuses.

- Retrieves incoming orders for sellers, providing buyer details and order statuses.

3. **Order Management:**
    - Accepts an order by updating its status to "accepted."
    - Cancels an order by deleting it from the database, ensuring clean data management.

Search Listings with Filters

```
SELECT l.*, GROUP_CONCAT(li.imagePath) AS images
FROM listings l
LEFT JOIN listing_images li ON l.id = li.listing_id
WHERE l.condition = ? AND l.make = ? AND l.sellingPrice <= ?
GROUP BY l.id
LIMIT ?;
```

Get Listings by User

```
SELECT l.*, GROUP_CONCAT(li.imagePath) AS images
FROM listings l
LEFT JOIN listing_images li ON l.id = li.listing_id
WHERE l.email = ?
GROUP BY l.id;
```

Calculate Total Earnings

```
SELECT SUM(sellingPrice) AS totalEarnings
FROM listings
WHERE email = ?;
```

Insert a New Listing

```
INSERT INTO listings (
  listingTitle, tagline, originalPrice, sellingPrice, category, condition, make, model, year,
  driveType, transmission, fuelType, mileage, engineSize, cylinder, color, door, vin,
  listingDescription, features, email
) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?);
```

Insert Images for a Listing

```
INSERT INTO listing_images (listing_id, imagePath)
VALUES ?;
```

Count Listings by User

```
SELECT COUNT(*) AS listingCount
FROM listings
WHERE email = ?;
```

Count Orders Made by User

```
SELECT COUNT(*) AS orderCount
FROM orders
WHERE buyer_email = ?;
```

Count Test Drive Bookings by User

```
SELECT COUNT(*) AS bookingCount
FROM test_drive_bookings
WHERE user_email = ?;
```

Calculate Total Earnings for User

```
SELECT SUM(sellingPrice) AS totalEarnings
FROM listings
WHERE email = ?;
```

Insert New Booking Using Stored Procedure

```
CALL book_test_drive(?, ?, ?, ?, ?, ?);
```

Retrieve User's Test Drive Bookings

```
SELECT test_drive_bookings.id,
    listings.listingTitle,
    test_drive_bookings.test_drive_date,
    test_drive_bookings.test_drive_time,
    test_drive_bookings.status
```

```
FROM test_drive_bookings
JOIN listings ON test_drive_bookings.listing_id = listings.id
WHERE test_drive_bookings.user_email = ?;
```

Accept Test Drive Booking

```
UPDATE test_drive_bookings
SET status = "accepted"
WHERE id = ?;
```

Delete a Booking

```
DELETE FROM test_drive_bookings
WHERE id = ?;
```

Insert a new user into the users table if the username is unique.

```
INSERT INTO users (username, email)
VALUES (?, ?)
ON DUPLICATE KEY UPDATE email = VALUES(email);
```

Triggers:

1.1. after_order_delete Trigger

- **Event**: After a row is deleted from the orders table.
- **Purpose**:
  - Logs the cancellation of an order into the cancellation_logs table.
- **Logic**:
  - When an order is deleted, the buyer_email, item_id (order ID), and the reason ("Cancelled by user") are recorded in the cancellation_logs table.

```
CREATE TRIGGER after_order_delete
AFTER DELETE ON orders
FOR EACH ROW
```

```
BEGIN
  INSERT INTO cancellation_logs (user_email, item_type, item_id, reason)
  VALUES (OLD.buyer_email, 'order', OLD.id, 'Cancelled by user');
END;
```

1.2. after_test_drive_booking_delete Trigger

- **Event**: After a row is deleted from the test_drive_bookings table.
- **Purpose**:
  - Logs the cancellation of a test drive booking into the cancellation_logs table.
- **Logic**:
  - When a booking is deleted, the user_email, item_id (booking ID), and the reason ("Cancelled by user") are recorded in the cancellation_logs table.

```
CREATE TRIGGER after_test_drive_booking_delete
AFTER DELETE ON test_drive_bookings
FOR EACH ROW
BEGIN
  INSERT INTO cancellation_logs (user_email, item_type, item_id, reason)
  VALUES (OLD.user_email, 'test_drive_booking', OLD.id, 'Cancelled by user');
END;
```

1.3. after_order_delete_update_listing_status Trigger

- **Event**: After a row is deleted from the orders table.
- **Purpose**:
  - Updates the status of the associated listing in the listings table back to "on sale".
- **Logic**:
  - When an order is deleted, the status column of the corresponding listing (based on listing_id) is updated to "on sale".

```
CREATE TRIGGER after_order_delete_update_listing_status
AFTER DELETE ON orders
FOR EACH ROW
```

```
BEGIN
   UPDATE listings
   SET status = 'on sale'
   WHERE id = OLD.listing_id;
END;
```

2. Stored Procedure

Stored procedures are reusable SQL blocks that perform a specific task. Here's the procedure defined in your database:

2.1. Procedure: book_test_drive

- **Purpose**:
    - Handles the booking of a test drive while ensuring no duplicate bookings are made for the same listing on the same date.
- **Parameters**:
    - p_listing_id: ID of the listing for which the test drive is being booked.
    - p_user_email: Email of the user booking the test drive.
    - p_phone_number: Phone number of the user.
    - p_test_drive_date: Date for the test drive.
    - p_test_drive_time: Time for the test drive.
    - p_additional_info: Additional details provided by the user.
- **Logic**:

1. Checks if a booking already exists for the given listing and date.
2. If a booking exists, raises an error.
3. If no booking exists, inserts a new record into the test_drive_bookings table.

```
DELIMITER //

CREATE PROCEDURE book_test_drive (
  IN p_listing_id INT,
  IN p_user_email VARCHAR(255),
  IN p_phone_number VARCHAR(20),
```

```
  IN p_test_drive_date DATE,
  IN p_test_drive_time TIME,
  IN p_additional_info TEXT
)
BEGIN
  DECLARE existing_booking INT;

  -- Check for existing booking
  SELECT COUNT(*) INTO existing_booking
  FROM test_drive_bookings
  WHERE listing_id = p_listing_id AND test_drive_date = p_test_drive_date;

  IF existing_booking > 0 THEN
    SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'A test drive for this listing is already booked on the selected date.';
  ELSE
    -- Insert the new booking
    INSERT INTO test_drive_bookings (
      listing_id, user_email, phone_number, test_drive_date, test_drive_time, additional_info
    ) VALUES (
      p_listing_id, p_user_email, p_phone_number, p_test_drive_date, p_test_drive_time, p_additional_info
    );
  END IF;
END //

DELIMITER ;
```

ER DIAGRAM

Car sale management system