**PROJECT DELIVERABLE**

**Database Systems Lab**

**April 04 , 2025**

**Team Number : 05**

**TEAM MEMBERS**

| **NUMBER** | **NAME** | **ROLL NO** |
| --- | --- | --- |
| **1** | **Moeez Salman** | **23L-3000** |
| **2** | **Zain Allaudin** | **23L-3036** |
| **3** | **Maaz** | **21L-5793** |

[**1.Project Idea: 3**](#_heading=h.2pgkchwces7d)

[1.1.1 Features:](#_heading=h.ldqtdhem5t24) [4](#_heading=h.gacuytyo5ryb)

[**2.Project Description: 5**](#_heading=h.hhxrm0pk6x3v)

# 

# 

# 

# 

# 

# 

# 

# 

# 1.Project Idea:

We are creating a web based project named Wheelister.This website is about a car rental website which will provide customer functionality to rent cars in an easy way and also allows the admin to add new cars to the warehouses.So this way the interaction between the customer and the admin will become very much easy and allows the user to easily book cars from his or her home.

## 1.1.1 Features: Enables users to register and log in, ensuring the security of their accounts.

## Provides users with the ability to search for various showrooms across the country.

## Allows users to book their preferred cars seamlessly.

## Enables users to view their booking history and current reservations.

## Allows users to cancel their bookings if necessary.

## Facilitates the search for different cars available nationwide.

## Provides detailed information about cars, including specifications and pricing.

## Allows users to specify the preferred time for their bookings.

## Enables administrators to add new cars to the available showroom inventory.

## Allows administrators to view all user bookings for management purposes.

## Provides administrators with secure login access to protect administrative accounts.

## Allows administrators to approve or decline user booking requests.

# 2.Project Description:

1. **User Credentials**

CREATE OR ALTER PROCEDURE CheckUserLogin

@Username NVARCHAR(100),

@Password NVARCHAR(100)

AS

BEGIN

SET NOCOUNT ON;

-- Declare variables to store the result

DECLARE @LoginSuccess BIT = 0;

DECLARE @UserID INT = NULL;

DECLARE @UserEmail NVARCHAR(30) = NULL;

DECLARE @LoyaltyPoints INT = 0;

-- Check if username and password match

SELECT

@UserID = userID,

@UserEmail = email,

@LoyaltyPoints = loyaltypoints

FROM

users

WHERE

username = @Username

AND password = @Password;

-- Set login success flag if user was found

IF @UserID IS NOT NULL

SET @LoginSuccess = 1;

-- Return the result with user information

SELECT

@LoginSuccess AS LoginSuccess,

@UserID AS UserID,

@Username AS Username,

@UserEmail AS Email,

@LoyaltyPoints AS LoyaltyPoints;

-- Optional: Update last login timestamp if needed

-- (would require adding a last\_login column to users table)

-- UPDATE users SET last\_login = GETDATE() WHERE userID = @UserID;

END;

EXEC CheckUserLogin @Username = 'testuser', @Password = 'test123';

This executes the predefined procedure that checks if the provided username and password match any user in the database.

1. **Admin Credentials**

SELECT \* FROM admins

WHERE admin\_name = 'Admin Ali'

AND adminID IN (SELECT adminID FROM admins WHERE warehouse\_no = 1);

This query verifies admin credentials and checks if they're assigned to a specific warehouse.

**`3. Searching Cars**

### Basic Car Search

SELECT car\_no, car\_name, car\_miles, car\_booking\_price

FROM cars

WHERE car\_name LIKE '%Toyota%';

Finds all cars with "Toyota" in their name using the LIKE operator.

### Advanced Car Search with Warehouse Info

SELECT c.car\_no, c.car\_name, c.car\_miles, c.car\_booking\_price,

w.warehouse\_name, w.warehouse\_number

FROM cars c

JOIN warehouse w ON c.warehouse\_no = w.warehouse\_no

WHERE c.car\_booking\_price BETWEEN 4000 AND 6000

ORDER BY c.car\_booking\_price DESC;

Joins cars with warehouse information and filters by price range, sorted by price.

**4. Searching Warehouse**

### Warehouse Search with Car Count

SELECT w.warehouse\_no, w.warehouse\_name, COUNT(c.car\_no) AS car\_count

FROM warehouse w

LEFT JOIN cars c ON w.warehouse\_no = c.warehouse\_no

GROUP BY w.warehouse\_no, w.warehouse\_name

HAVING COUNT(c.car\_no) > 0;

Shows warehouses with the count of available cars, excluding empty warehouses.

### Specific Warehouse Search

SELECT \* FROM warehouse

WHERE warehouse\_name LIKE '%Lahore%';

Finds warehouses located in Lahore using LIKE for partial matching.

**5. Add to bookings**

### New Booking Insert

INSERT INTO bookings (userID, warehouse\_no, carsbooked)

VALUES (1, 2, 1);

Adds a new booking record for a user at a specific warehouse.

### Booking with Validation

-- First check car availability

DECLARE @Available INT;

SELECT @Available = COUNT(\*) FROM cars WHERE warehouse\_no = 3 AND car\_no = 105;

IF @Available > 0

BEGIN

INSERT INTO bookings (userID, warehouse\_no, carsbooked)

VALUES (1, 3, 1);

SELECT 'Booking successful' AS Result;

END

ELSE

SELECT 'Car not available' AS Result;

Checks car availability before creating a booking.

**6. Delete from bookings**

### Remove Booking

DELETE FROM bookings

WHERE userID = 1 AND warehouse\_no = 1;

Deletes a specific booking record.

### Conditional Booking Deletion

DELETE FROM bookings

WHERE userID IN (SELECT userID FROM users WHERE username = 'testuser')

AND warehouse\_no = 2;

Deletes bookings for a specific user at a specific warehouse using a subquery.

**7. Admin accept or decline bookings**

### View Pending Bookings

SELECT b.userID, u.username, b.warehouse\_no, w.warehouse\_name, b.carsbooked

FROM bookings b

JOIN users u ON b.userID = u.userID

JOIN warehouse w ON b.warehouse\_no = w.warehouse\_no

WHERE b.warehouse\_no IN (SELECT warehouse\_no FROM admins WHERE admin\_name = 'Admin Ali');

Shows bookings for warehouses managed by a specific admin.

### Update Booking Status (Accept/Decline)

-- This would typically update a status column (would need to be added to schema)

-- For demonstration, we'll delete declined bookings

**DELETE FROM bookings**

WHERE userID = 1 AND warehouse\_no = 1

AND carsbooked = (SELECT MIN(carsbooked) FROM bookings WHERE userID = 1);

Simulates declining a booking by removing it (in a real system, you'd update a status field).

**8. Change username (USER)**

### Update Username

UPDATE users

SET username = 'newusername'

WHERE userID = 1 AND password = '123456789’';

Changes username with password verification.

**9. Change password (USER)**

### Update Password

UPDATE users

SET password = '12345678’'

WHERE username = 'testuser' AND password = '12345’';

Changes password after verifying old password.

**10. Change username (ADMIN)**

### Admin Username Update

UPDATE admins

SET admin\_name = 'NewAdminName'

WHERE adminID = 1 AND warehouse\_no = 1;

Updates admin username with warehouse verification.

**11. Change password (ADMIN)**

### Admin Password Update

-- This would require schema modification

-- For demonstration, we'll assume it exists

UPDATE admins

SET password = 'newadminpass'

WHERE adminID = 1;

Hypothetical query to update admin password (schema would need modification).

**12. Add new Carrs: (ADMIN):**

### Insert New Car

INSERT INTO cars (car\_no, warehouse\_no, car\_name, car\_miles, car\_booking\_price)

VALUES (108, 3, 'MG HS', 12000, 7500);

Adds a new car to a warehouse.

### Bulk Car Insert

INSERT INTO cars (car\_no, warehouse\_no, car\_name, car\_miles, car\_booking\_price)

VALUES

(109, 4, 'Toyota Fortuner', 18000, 8500),

(110, 4, 'Suzuki Swift', 22000, 4000);

Adds multiple cars at once.

**13. Add new warehouse: (ADMIN):**

### Insert New Warehouse

INSERT INTO warehouse (warehouse\_name, warehouse\_number)

VALUES ('Faisalabad Auto Hub', '+923411234567');

Creates a new warehouse record.

### Add Warehouse with Admin Assignment

BEGIN TRANSACTION;

DECLARE @NewWarehouseID INT;

INSERT INTO warehouse (warehouse\_name, warehouse\_number)

VALUES ('Quetta Storage', '+923811234567');

SET @NewWarehouseID = SCOPE\_IDENTITY();

INSERT INTO admins (admin\_name, warehouse\_no)

VALUES ('Admin Khan', @NewWarehouseID);

COMMIT;

Adds a new warehouse and assigns an admin in a transaction.

## Additional Functionalities

### View for User Bookings

CREATE VIEW UserBookingsView AS

SELECT u.username, w.warehouse\_name, c.car\_name, b.carsbooked

FROM bookings b

JOIN users u ON b.userID = u.userID

JOIN warehouse w ON b.warehouse\_no = w.warehouse\_no

LEFT JOIN cars c ON b.warehouse\_no = c.warehouse\_no;

Creates a view to easily see user bookings with related information.

### Car Statistics Procedure

CREATE PROCEDURE GetCarStatistics

AS

BEGIN

SELECT

AVG(car\_booking\_price) AS avg\_price,

MIN(car\_booking\_price) AS min\_price,

MAX(car\_booking\_price) AS max\_price,

SUM(car\_booking\_price) AS total\_value

FROM cars;

END;

Provides statistical information about cars in the system.

### Union Query for All Locations

SELECT location FROM users

UNION

SELECT warehouse\_location FROM warehouse;

Combines user locations and warehouse locations (note: warehouse\_location would need to be added).

### Intersection Query

SELECT userID FROM users

INTERSECT

SELECT userID FROM bookings;

Finds users who have made bookings.

### Nested Query for Popular Cars

SELECT car\_name, car\_booking\_price

FROM cars

WHERE warehouse\_no IN (

SELECT warehouse\_no

FROM bookings

GROUP BY warehouse\_no

HAVING COUNT(\*) > 1

);

Shows cars in warehouses with more than 1 booking.

### Right Join Example

SELECT w.warehouse\_name, c.car\_name

FROM cars c

RIGHT JOIN warehouse w ON c.warehouse\_no = w.warehouse\_no;

Shows all warehouses, even those without cars.

### Group By with Having

SELECT warehouse\_no, COUNT(\*) AS car\_count

FROM cars

GROUP BY warehouse\_no

HAVING COUNT(\*) > 1;

Finds warehouses with more than one car.

### Outer Join Example

SELECT u.username, b.carsbooked

FROM users u

FULL OUTER JOIN bookings b ON u.userID = b.userID;

Shows all users and all bookings, matching where possible.