Ride & Pickup Database Management System

Topic 36

Shayaan Kirubakaran (Section 5)

Kirusanth Palakanthan (Section 10)

Ahmed Hasan (Section 5)

CPS510 - 07/05 Database Systems I

Dr. Glaucia Melo/ Dr. Abhari

November 26, 2025

Table of Contents

Introduction

This report presents the development and implementation of a Ride & Pickup Database Management System designed to integrate ride-sharing and item pickup services into a unified, reliable platform. Over the course of this project, our group progressed through each major phase outlined in the CPS 510 framework—beginning with application scoping, followed by ER modeling, relational schema design, SQL implementation, normalization to 3NF/BCNF, and the construction of a functional user interface. The system is built to manage key operational components such as customers, drivers, vehicles, merchants, service orders, payments, and ratings, ensuring that all ride and delivery activities are accurately recorded and maintained within a centralized Oracle database.

The objective of this solution is to provide a structured, efficient, and scalable way to handle real-world processes like driver assignment, route tracking, payment processing, and two-way feedback. By combining both transportation and delivery functionalities, the system reduces the fragmentation often found in separate service platforms and improves overall data integrity, consistency, and accessibility. This documentation summarizes the full lifecycle of the project—from conceptual analysis to final implementation—and demonstrates how each stage contributes to building a complete and operational Ride & Pickup DBMS.

Week 1 – Application Scope and System Description

In Week 1, our group began the Ride & Pickup Database Management System project by defining the application's logical scope, purpose, and key requirements. After reviewing the list of approved applications and discussing our ideas with the TA, we selected a Ride & Pickup service—similar to systems like Uber—that combines ride-sharing and item pickup into one integrated platform. This application was chosen because it provides a rich environment for database design, involving real-time transactions, multi-entity interactions, and strict integrity requirements.

Our first task was creating the application description, which outlined the main functions the DBMS must support. The system is expected to manage customers, drivers, vehicles, merchants, service orders, live locations, payments, and rating activity. A customer should be able to request either a ride or a delivery order, while the system identifies an available driver, retrieves pickup and drop-off information, and completes the service. To reflect real-world behavior, we included features such as customer balances, driver earnings, payment verification, vehicle registration, merchant participation for food deliveries, and two-way rating functionality between customers and drivers. We also identified the system's need for secure handling of payment data and proper validation to prevent conflicting or duplicated assignments.

Entities Defined in Week 1

We documented the primary entities required for the conceptual model:

- **Customer** – Stores unique customer ID, name, contact information, signup date, and customer rating. Each customer can place multiple service orders.
- **Driver** – Stores driver ID, license details, contact information, and driver rating. Drivers fulfill ride or delivery requests.
- **Vehicle** – Contains vehicle ID, license plate, make, model, color, and insurance expiry. Each vehicle is linked to a single driver.
- **Service Order** – Represents either a ride or delivery request and includes attributes such as order ID, customer, driver, timestamps, status, and fare.
- **Location** – Defines pickup and drop-off points using address details and GPS coordinates.
- **Payment** – Tracks payment ID, method, amount, timestamp, and payment status linked to a service order.
- **Merchant** – Represents stores or restaurants for delivery orders, including merchant ID and address.
- **Rating System** – Stores feedback for both customers and drivers with rating values, comments, and a reference to the related service order.

Relationships Established in Week 1

We mapped out all essential relationships between the entities to form the foundation of the ER diagram:

- A **Customer** can place many **Service Orders**, but each order belongs to one customer (1–M).
- A **Driver** can complete many **Service Orders**, but each order is fulfilled by one driver (1–M).
- A **Driver** may register multiple **Vehicles**, but only one can be active per trip (1–M).
- Each **Service Order** must be linked to at least one **Payment** (1–M).
- Each **Service Order** connects to one or more **Locations** for pickup and drop-off (1–M).

- A **Merchant** can serve many delivery orders, but each delivery order links to only one merchant (1–M).

- A **Customer** and a **Driver** can both produce multiple ratings, each tied to a specific service order (1–M).

These relationships allowed us to define the system's structure, enforcing consistency through primary keys, foreign keys, and participation constraints. We also determined early constraints such as unique IDs for all major entities, valid driver–vehicle associations, and a rule that every service order must include a payment record.

Functions and System Requirements

As part of Week 1, we outlined the system's core functions:

- **Customer Management** – Registration, personal information, account balance, and customer ratings.

- **Driver & Vehicle Management** – Registration, licensing information, registered vehicles, and driver balances.

- **Order Management** – Handling ride and food orders, storing pickup and drop-off details, and linking orders to customers and drivers.

- **Driver Assignment** – Matching customers with the nearest available driver based on location, availability, and seating capacity.

- **Location Tracking** – Real-time GPS tracking for both customers and drivers.

- **Payment Processing** – Supporting credit/debit/preloaded balance payments, generating receipts, and updating balances.

- **Rating System** – Allowing both customers and drivers to rate each other after a completed service.

Sample Queries Included in Week 1

To help guide future SQL development, we prepared example queries such as:

- Retrieving a customer's full order history

- Listing active orders assigned to drivers

- Displaying merchants involved in completed deliveries
- Tracking customer or driver balances
- Retrieving all ratings a driver has received

Week 2 – ER Diagram Design

During Week 2, our focus was to design and finalize the ER diagram for the Ride & Pickup Database Management System. Building on the system description and functional requirements established in Week 1, we translated the application features into a structured ER model that represents all major entities, attributes, and relationships in the system. This ER diagram acts as the blueprint for future steps, including schema creation, normalization, and SQL implementation.

ER Diagram Overview

Our ER diagram includes all key entities identified in Assignment 1, such as Customer, Driver, Vehicle, Service Order, Location, Payment, Merchant, and Rating. Each entity is shown with its relevant attributes—for example, Customer includes CustomerID, Name, Email, Phone Number, Address, Signup Date, and Purchase History; Driver includes DriverID, Name, License Info, Insurance Info, and Contact Details; and Vehicle includes License Plate, Make, Model, Colour, Year, and Vehicle Type.
These attribute sets are clearly visible in the diagram on page 1 of the uploaded ER model .

Primary keys are marked distinctly (e.g., Customer ID, Driver ID, Order ID, Merchant ID, Transaction ID, License Plate), and foreign-key relationships were drawn using connecting lines and crow's-foot notation to indicate relationship cardinalities. The diagram also separates composite attributes such as address into components including street address, city, province, and postal code.

Relationships Captured in the ER Model

The ER diagram formalizes the logical relationships between entities based on real functionality in a ride-sharing and delivery system:

- A customer places many service orders, but each service order is linked to one customer (1-to-M).
- A driver completes many service orders, while each service order is assigned to one driver (1-to-M).
- A driver may register multiple vehicles, but only one vehicle can be used per trip (1-to-M).

- Each service order generates one or more location records, including pickup and drop-off (1-to-M).

- Each service order must have at least one payment, establishing a 1-to-M link between Service Order and Payment.

- A merchant can be associated with multiple delivery orders, but each order references one merchant (1-to-M).

- The rating entity connects customers and drivers, allowing both sides to provide feedback on completed service orders (1-to-M relationships).

These relationships and cardinalities are visually illustrated in the uploaded diagram, where each connector specifies "1", "M", or participation constraints to reflect system logic accurately. For example, the Service Order → Location and Service Order → Payment connections clearly show multi-valued relationships, while Driver → Vehicle is indicated as a one-to-many association.

Diagram Structure and Notation

Our diagram uses clear colour-coded entities (e.g., Customer in red, Driver in green, Vehicle in yellow, Service Order in purple) and uses diamonds to represent relationships such as **MAKES, SUPPLIES, INCLUDES, RECEIVES, and CONTAINS**. This helps clarify how each component interacts within the system. All attributes—simple, composite, and multi-valued—are fully represented and directly connected to their respective entities, matching standard ER modelling conventions.

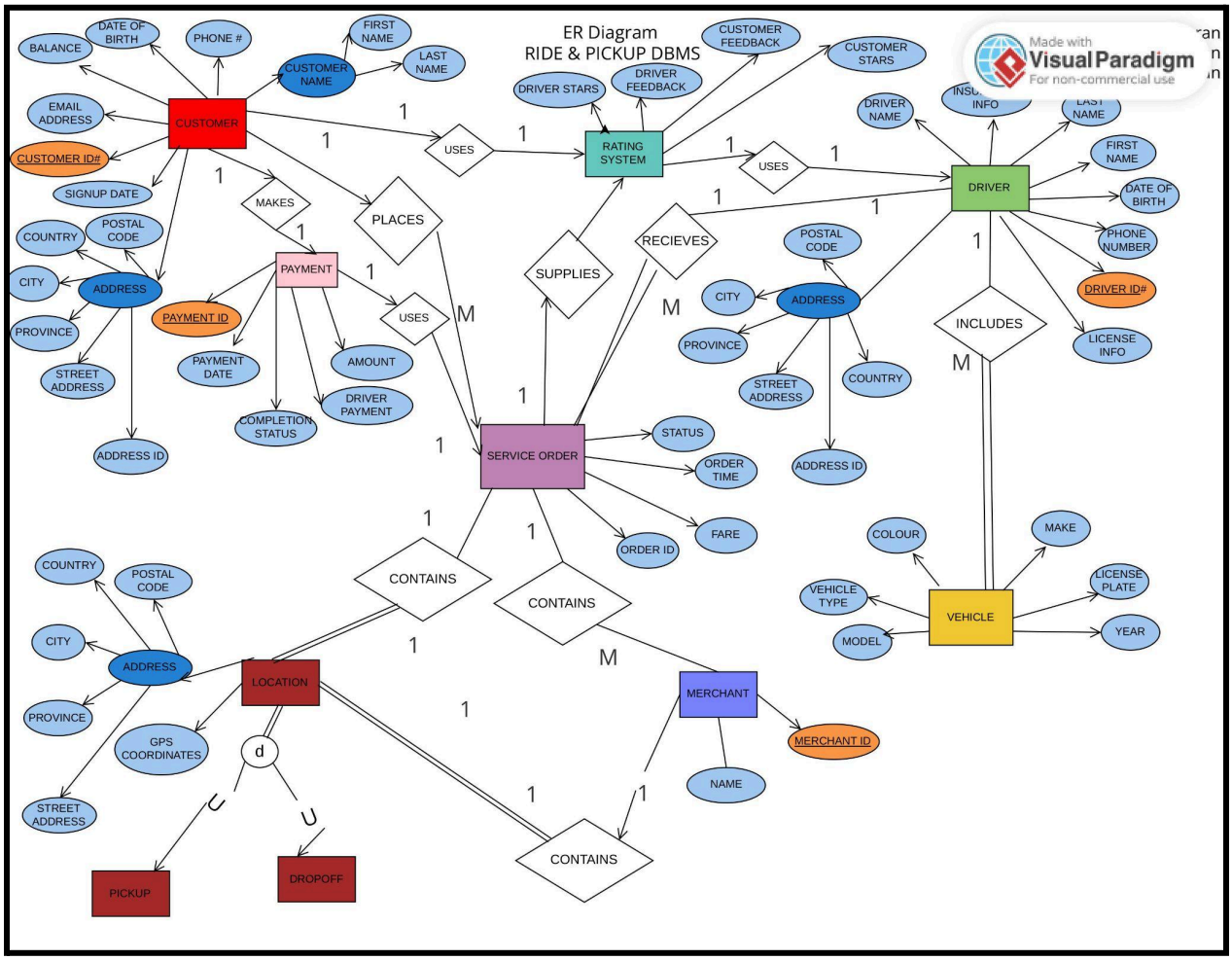
Note:

Generally speaking, most of the ER diagram remains unchanged. The most notable differences are the inclusion of the driver having a particular address in addition to the customer along with some additional attributes included for the service order. Not only that, but the foreign keys in particular for both the driver ID and customer ID indicate there were more relationships required regarding the rating system entity. Lastly, the Payment attributes were also completely changed in addition to the added relationship to service order due to that particular table also having a foreign key. These changes were made to better reflect the final design of the database.

Outcome of Week 2

This week's deliverable provides the complete conceptual foundation needed for the next phase of development. With the ER diagram finalized and approved, we are prepared to convert the model into relational schemas during Week 3. The ER diagram will continue serving as our reference point for ensuring consistency across table definitions, constraints, and relationships as the database is implemented in Oracle.

ER Diagram:



Week 3 – Schema Creation in Oracle

For Week 3, our group began implementing the Ride & Pickup DBMS by generating all required tables in Oracle. The objective of this phase was to translate the ER diagram from Week 2 into a working relational schema, create primary and foreign key constraints, and insert sample data to validate that the system behaved as expected.

The first step in our code was dropping any existing tables to prevent conflicts during development. We followed the standard Oracle pattern of using the DROP TABLE ... CASCADE CONSTRAINTS command wrapped inside an exception-handling block so the script would continue even if a table did not already exist. An example from our code is:

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE Customer CASCADE CONSTRAINTS';

```
EXCEPTION WHEN OTHERS THEN NULL;

END;

/
```

This approach was repeated for all tables in the system, including Customer, Driver, Vehicle, Merchant, Location, ServiceOrder, Rating, and Payment.

Once the environment was cleared, we created the tables that appear in our ER diagram. Each table was defined with a primary key and foreign keys where needed. For example, the ServiceOrder table linked customers, drivers, merchants, and locations:

```
CREATE TABLE ServiceOrder (

 OrderID NUMBER PRIMARY KEY,

 CustomerID NUMBER REFERENCES Customer(CustomerID),

 DriverID NUMBER REFERENCES Driver(DriverID),

 MerchantID NUMBER REFERENCES Merchant(MerchantID),

 PickupID NUMBER REFERENCES Location(LocationID),

 DropoffID NUMBER REFERENCES Location(LocationID),

 Fare NUMBER(10,2),

 OrderType VARCHAR2(20),

 Status VARCHAR2(20)

);
```

Other tables followed the same structure. Drivers were linked to vehicles, payments were linked to service orders, and locations stored pickup and drop-off addresses. Each entity's attributes matched what was defined conceptually in Week 1.

After creating the tables, we populated each one with sample data. Customers were inserted first, followed by drivers, vehicles, and merchants. We then added location entries, service orders connecting customers to drivers, and finally payment and rating records. For example, a typical insertion for a location and service order looked like:

```
INSERT INTO Location (LocationID, Street, City, Province, PostalCode)

VALUES (1, '123 King St', 'Toronto', 'ON', 'M5H2N2');
```

INSERT INTO ServiceOrder (OrderID, CustomerID, DriverID, MerchantID, PickupID, DropoffID, Fare, OrderType, Status)

VALUES (1, 101, 201, 301, 1, 2, 25.50, 'Ride', 'Completed');

Throughout the week, we tested table relationships by running SELECT queries to verify that the inserted data matched our constraints. Foreign keys prevented invalid records, such as inserting a service order without an existing customer or payment records without a valid order. This confirmed that the database enforced the relationships designed in our ER model.

By the end of Week 3, we had a fully functional relational schema with all tables created, linked, and populated. This demonstrated that our ER diagram was correctly translated into an operational structure and that the DBMS supported the constraints and relationships intended. Week 3 was a crucial milestone, as it transformed the conceptual design into a working Oracle-based implementation of the Ride & Pickup DBMS.

Our full code that was submitted during week 3:

RidePickup_Phase1.sql:

```
-----------------------------------------------------------------
-- CPS510 - Assignment 3
-- Ride & Pickup DBMS - Phase 1 Schema Design
-- Authors: Shayaan Kirubakaran, Kirusanth Palakanthan, Ahmed Hasan
-- Notes:
--   • Rerunnable script (drops old tables, then recreates them)
--   • All entities from ER diagram implemented
--   • PK, FK, NOT NULL, UNIQUE, DEFAULT, CHECK, CASCADE constraints used
-----------------------------------------------------------------


-----------------------------
-- DROP OLD TABLES IF THEY EXIST
-----------------------------
BEGIN EXECUTE IMMEDIATE 'DROP TABLE Rating_System CASCADE CONSTRAINTS';
EXCEPTION WHEN OTHERS THEN IF SQLCODE != -942 THEN RAISE; END IF; END;
/
BEGIN EXECUTE IMMEDIATE 'DROP TABLE Payment CASCADE CONSTRAINTS';
EXCEPTION WHEN OTHERS THEN IF SQLCODE != -942 THEN RAISE; END IF; END;
/
BEGIN EXECUTE IMMEDIATE 'DROP TABLE Service_Order CASCADE CONSTRAINTS';
EXCEPTION WHEN OTHERS THEN IF SQLCODE != -942 THEN RAISE; END IF; END;
```

```
/
BEGIN EXECUTE IMMEDIATE 'DROP TABLE Vehicle CASCADE CONSTRAINTS';
EXCEPTION WHEN OTHERS THEN IF SQLCODE != -942 THEN RAISE; END IF; END;
/
BEGIN EXECUTE IMMEDIATE 'DROP TABLE Merchant CASCADE CONSTRAINTS';
EXCEPTION WHEN OTHERS THEN IF SQLCODE != -942 THEN RAISE; END IF; END;
/
BEGIN EXECUTE IMMEDIATE 'DROP TABLE Customer CASCADE CONSTRAINTS';
EXCEPTION WHEN OTHERS THEN IF SQLCODE != -942 THEN RAISE; END IF; END;
/
BEGIN EXECUTE IMMEDIATE 'DROP TABLE Driver CASCADE CONSTRAINTS';
EXCEPTION WHEN OTHERS THEN IF SQLCODE != -942 THEN RAISE; END IF; END;
/
BEGIN EXECUTE IMMEDIATE 'DROP TABLE Location CASCADE CONSTRAINTS';
EXCEPTION WHEN OTHERS THEN IF SQLCODE != -942 THEN RAISE; END IF; END;
/
BEGIN EXECUTE IMMEDIATE 'DROP TABLE Address CASCADE CONSTRAINTS';
EXCEPTION WHEN OTHERS THEN IF SQLCODE != -942 THEN RAISE; END IF; END;
/


-----------------------------
-- CREATE TABLES
-----------------------------

-- ADDRESS TABLE
CREATE TABLE Address (
   Address_ID     NUMBER        PRIMARY KEY,
   Country        VARCHAR2(50)  NOT NULL,
   Province       VARCHAR2(50)  NOT NULL,
   City           VARCHAR2(50)  NOT NULL,
   Street_Address VARCHAR2(100) NOT NULL,
   Postal_Code    VARCHAR2(10)  NOT NULL
);

-- LOCATION TABLE
CREATE TABLE Location (
   Location_ID    NUMBER        PRIMARY KEY,
   Street_Address  VARCHAR2(100) NOT NULL,
   City           VARCHAR2(50)  NOT NULL,
```

```
    Province        VARCHAR2(50)  NOT NULL,
    Postal_Code     VARCHAR2(10)  NOT NULL,
    Country         VARCHAR2(50)  NOT NULL,
    GPS_Coordinates VARCHAR2(100)
);

-- CUSTOMER TABLE
CREATE TABLE Customer (
    Customer_ID   NUMBER        PRIMARY KEY,
    First_Name    VARCHAR2(50)   NOT NULL,
    Last_Name     VARCHAR2(50)   NOT NULL,
    Phone_Number  VARCHAR2(15)   NOT NULL UNIQUE,
    Email_Address VARCHAR2(100)  NOT NULL UNIQUE,
    Date_Of_Birth DATE,
    Signup_Date   DATE          DEFAULT SYSDATE,
    Balance       NUMBER(10,2)   DEFAULT 0 CHECK (Balance >= 0),
    Address_ID    NUMBER        NOT NULL,
    FOREIGN KEY (Address_ID) REFERENCES Address(Address_ID)
);

-- DRIVER TABLE
CREATE TABLE Driver (
    Driver_ID     NUMBER        PRIMARY KEY,
    First_Name     VARCHAR2(50)  NOT NULL,
    Last_Name      VARCHAR2(50)  NOT NULL,
    Phone_Number   VARCHAR2(15)  NOT NULL UNIQUE,
    Date_Of_Birth  DATE,
    License_Info   VARCHAR2(50)  NOT NULL,
    Insurance_Info VARCHAR2(100),
    Address_ID     NUMBER        NOT NULL,
    FOREIGN KEY (Address_ID) REFERENCES Address(Address_ID)
);

-- VEHICLE TABLE
CREATE TABLE Vehicle (
    Vehicle_ID    NUMBER        PRIMARY KEY,
    License_Plate VARCHAR2(20)  NOT NULL UNIQUE,
    Make          VARCHAR2(50)  NOT NULL,
    Model         VARCHAR2(50)  NOT NULL,
```

```
    Colour        VARCHAR2(20),
    Year          NUMBER(4)     CHECK (Year >= 1980),
    Vehicle_Type  VARCHAR2(50),
    Driver_ID     NUMBER        NOT NULL,
    FOREIGN KEY (Driver_ID) REFERENCES Driver(Driver_ID)
);

-- MERCHANT TABLE
CREATE TABLE Merchant (
    Merchant_ID NUMBER        PRIMARY KEY,
    Name        VARCHAR2(100) NOT NULL,
    Address_ID  NUMBER        NOT NULL,
    FOREIGN KEY (Address_ID) REFERENCES Address(Address_ID)
);

-- SERVICE_ORDER TABLE
CREATE TABLE Service_Order (
    Order_ID          NUMBER        PRIMARY KEY,
    Customer_ID       NUMBER        NOT NULL,
    Driver_ID         NUMBER        NOT NULL,
    Merchant_ID       NUMBER,
    Status            VARCHAR2(20)  DEFAULT 'Pending'
                      CHECK (Status IN ('Pending','Completed','Cancelled')),
    Order_Time        TIMESTAMP     DEFAULT CURRENT_TIMESTAMP,
    Fare              NUMBER(10,2)  CHECK (Fare >= 0),
    Pickup_Location_ID  NUMBER      NOT NULL,
    Dropoff_Location_ID NUMBER      NOT NULL,
    FOREIGN KEY (Customer_ID)       REFERENCES Customer(Customer_ID),
    FOREIGN KEY (Driver_ID)         REFERENCES Driver(Driver_ID),
    FOREIGN KEY (Merchant_ID)       REFERENCES Merchant(Merchant_ID),
    FOREIGN KEY (Pickup_Location_ID) REFERENCES Location(Location_ID),
    FOREIGN KEY (Dropoff_Location_ID)REFERENCES Location(Location_ID)
);

-- PAYMENT TABLE
CREATE TABLE Payment (
    Payment_ID   NUMBER       PRIMARY KEY,
    Order_ID     NUMBER       NOT NULL,
    Payment_Type VARCHAR2(20)  CHECK (Payment_Type IN ('Credit','Debit','Balance')),
```

```
    Amount       NUMBER(10,2)  CHECK (Amount >= 0),
    Status       VARCHAR2(20)  DEFAULT 'Pending',
    FOREIGN KEY (Order_ID) REFERENCES Service_Order(Order_ID) ON DELETE CASCADE
);

-- RATING_SYSTEM TABLE
CREATE TABLE Rating_System (
    Rating_ID        NUMBER      PRIMARY KEY,
    Order_ID         NUMBER      NOT NULL,
    Customer_ID      NUMBER      NOT NULL,
    Driver_ID        NUMBER      NOT NULL,
    Customer_Stars   NUMBER(1)   CHECK (Customer_Stars BETWEEN 1 AND 5),
    Customer_Feedback VARCHAR2(255),
    Driver_Stars     NUMBER(1)   CHECK (Driver_Stars BETWEEN 1 AND 5),
    Driver_Feedback  VARCHAR2(255),
    FOREIGN KEY (Order_ID)    REFERENCES Service_Order(Order_ID) ON DELETE CASCADE,
    FOREIGN KEY (Customer_ID) REFERENCES Customer(Customer_ID),
    FOREIGN KEY (Driver_ID)   REFERENCES Driver(Driver_ID)
);

-------------------------------------------------------------
-- END OF SCRIPT
-------------------------------------------------------------
```

RidePickup_Data.sql

```
-- 1) Address (parent for Customer, Driver, Merchant)
INSERT INTO Address (Address_ID, Country, Province, City, Street_Address, Postal_Code)
VALUES (1, 'Canada', 'Ontario', 'Toronto', '123 King St', 'M5G1X5');

-- 2) Locations (pickup/dropoff)
INSERT INTO Location (Location_ID, Street_Address, City, Province, Postal_Code, Country,
GPS_Coordinates)
VALUES (1, '456 Queen St', 'Toronto', 'Ontario', 'M5H2N2', 'Canada', NULL);
INSERT INTO Location (Location_ID, Street_Address, City, Province, Postal_Code, Country,
GPS_Coordinates)
VALUES (2, '789 Bay St', 'Toronto', 'Ontario', 'M5B2C5', 'Canada', NULL);

-- 3) Customer (needs Address_ID=1)
INSERT INTO Customer (Customer_ID, First_Name, Last_Name, Phone_Number, Email_Address,
```

```
                Date_Of_Birth, Signup_Date, Balance, Address_ID)
VALUES (1, 'John', 'Doe', '4161234567', 'john.doe@email.com',
       TO_DATE('1990-05-20','YYYY-MM-DD'), SYSDATE, 50.00, 1);

-- 4) Driver (needs Address_ID=1)
INSERT INTO Driver (Driver_ID, First_Name, Last_Name, Phone_Number, Date_Of_Birth,
                    License_Info, Insurance_Info, Address_ID)
VALUES (1, 'Jane', 'Smith', '4169876543', TO_DATE('1985-10-10','YYYY-MM-DD'),
       'LIC12345', 'InsureCo #123', 1);

-- 5) Vehicle (needs Driver_ID=1)
INSERT INTO Vehicle (Vehicle_ID, License_Plate, Make, Model, Colour, Year, Vehicle_Type,
Driver_ID)
VALUES (1, 'ABC123', 'Toyota', 'Camry', 'Black', 2020, 'Sedan', 1);

-- 6) Merchant (needs Address_ID=1)
INSERT INTO Merchant (Merchant_ID, Name, Address_ID)
VALUES (1, 'Pizza Place', 1);

-- 7) Service order (needs Customer_ID=1, Driver_ID=1, Merchant_ID=1, and two Location_IDs)
INSERT INTO Service_Order (Order_ID, Customer_ID, Driver_ID, Merchant_ID, Status,
                           Order_Time, Fare, Pickup_Location_ID, Dropoff_Location_ID)
VALUES (1, 1, 1, 1, 'Pending', CURRENT_TIMESTAMP, 25.50, 1, 2);

-- 8) Payment (child of Service_Order)
INSERT INTO Payment (Payment_ID, Order_ID, Payment_Type, Amount, Status)
VALUES (1, 1, 'Credit', 25.50, 'Paid');

-- 9) Rating (child of Service_Order, Customer, Driver)
INSERT INTO Rating_System (Rating_ID, Order_ID, Customer_ID, Driver_ID, Customer_Stars,
                           Customer_Feedback, Driver_Stars, Driver_Feedback)
VALUES (1, 1, 1, 1, 5, 'Great driver!', 5, 'Polite customer');

-- Commit the test data
COMMIT;

-- Quick checks
SELECT * FROM Address;
SELECT * FROM Location;
```

```
SELECT * FROM Customer;
SELECT * FROM Driver;
SELECT * FROM Vehicle;
SELECT * FROM Merchant;
SELECT * FROM Service_Order;
SELECT * FROM Payment;
SELECT * FROM Rating_System;
```

Week 4 – SQL Queries & Views

Part 1: SQL Query Demonstration

In Week 4, the objective was to demonstrate that our Ride & Pickup DBMS correctly stored data and supported meaningful retrieval operations. Based on the tables created in Week 3—Customer, Driver, Vehicle, Merchant, Location, Service_Order, Payment, and Rating_System—we created a series of SQL queries to display important information from the database. These queries validated that our schema was functioning properly, that our relationships were defined correctly, and that users could extract relevant operational data from the system.

The queries we implemented showcased different aspects of the Ride & Pickup DBMS, including retrieving customer information, listing available drivers and vehicles, joining service orders with associated customers and drivers, and verifying payment and rating activity. These queries reflected real use cases such as viewing order history, checking driver availability, tracking pickup and drop-off locations, and confirming completed payments.

Below are the queries used in Week 4, taken directly from our submitted SQL scripts.

RidePickUpQueries.sql

```
-------------------------------------------------------------
-- CPS510 – Assignment 4 (Part 1)
-- Ride & Pickup DBMS – Queries
-- Authors: Shayaan Kirubakaran, Kirusanth Palakanthan, Ahmed Hasan
-------------------------------------------------------------


-------------------------------------------------------------
-- Q1. Address Table: Toronto addresses
-------------------------------------------------------------
SELECT Address_ID, Street_Address AS Street, City, Province, Postal_Code
FROM   Address
WHERE  City = 'Toronto';
```

```
-------------------------------------------------------------
-- Q2. Customer Table: Balances with status
-------------------------------------------------------------
SELECT Customer_ID,
      First_Name || ' ' || Last_Name AS Customer_Name,
      Balance,
      CASE WHEN Balance > 0 THEN 'Active' ELSE 'Inactive' END AS Status
FROM   Customer
ORDER BY Balance DESC;

-------------------------------------------------------------
-- Q3. Driver Table: With insurance info
-------------------------------------------------------------
SELECT Driver_ID,
      First_Name || ' ' || Last_Name AS Driver_Name,
      License_Info,
      Insurance_Info
FROM   Driver
WHERE  Insurance_Info IS NOT NULL;

-------------------------------------------------------------
-- Q4. Vehicle Table: Manufactured after 2015
-------------------------------------------------------------
SELECT Vehicle_ID, License_Plate, Make, Model, Year, Vehicle_Type
FROM   Vehicle
WHERE  Year > 2015;

-------------------------------------------------------------
-- Q5. Merchant Table: Merchants and their city
-------------------------------------------------------------
SELECT DISTINCT M.Merchant_ID, M.Name AS Merchant_Name, A.City, A.Province
FROM   Merchant M, Address A
WHERE  M.Address_ID = A.Address_ID;

-------------------------------------------------------------
-- Q6. Location Table: Ontario locations
-------------------------------------------------------------
SELECT Location_ID, Street_Address, City, Province, Postal_Code
FROM   Location
WHERE  Province = 'Ontario';

-------------------------------------------------------------
-- Q7. Service Orders: Completed > $20
```

```
-------------------------------------------------------------
SELECT Order_ID, Customer_ID, Driver_ID, Fare, Status
FROM   Service_Order
WHERE  Status = 'Completed'
AND    Fare > 20;

-------------------------------------------------------------
-- Q8. Payment Table: Count of credit card payments
-------------------------------------------------------------
SELECT COUNT(*) AS Credit_Payments
FROM   Payment
WHERE  Payment_Type = 'Credit';

-------------------------------------------------------------
-- Q9. Ratings: Low (< 3 stars)
-------------------------------------------------------------
SELECT Rating_ID, Order_ID, Customer_Stars, Customer_Feedback
FROM   Rating_System
WHERE  Customer_Stars < 3;

-------------------------------------------------------------
-- Q10. Join: Completed orders with customer & driver names
-------------------------------------------------------------
SELECT so.Order_ID,
       c.First_Name || ' ' || c.Last_Name AS Customer_Name,
       d.First_Name || ' ' || d.Last_Name AS Driver_Name,
       so.Fare
FROM   Service_Order so, Customer c, Driver d
WHERE  so.Customer_ID = c.Customer_ID
AND    so.Driver_ID   = d.Driver_ID
AND    so.Status = 'Completed'
ORDER BY so.Fare DESC;

-------------------------------------------------------------
-- Q11. Join: Payments with order info & customer name
-------------------------------------------------------------
SELECT p.Payment_ID, p.Payment_Type, p.Amount,
       so.Order_ID,
       c.First_Name || ' ' || c.Last_Name AS Customer_Name
FROM   Payment p, Service_Order so, Customer c
WHERE  p.Order_ID = so.Order_ID
AND    so.Customer_ID = c.Customer_ID
ORDER BY p.Amount DESC;
```

```
-------------------------------------------------------------
-- Q12. Aggregation: Total orders per customer
-------------------------------------------------------------
SELECT c.Customer_ID,
       c.First_Name || ' ' || c.Last_Name AS Customer_Name,
       COUNT(so.Order_ID) AS Total_Orders
FROM   Customer c LEFT JOIN Service_Order so
       ON c.Customer_ID = so.Customer_ID
GROUP BY c.Customer_ID, c.First_Name || ' ' || c.Last_Name
ORDER BY Total_Orders DESC;

-------------------------------------------------------------
-- Q13. Aggregation: Average fare per driver
-------------------------------------------------------------
SELECT d.Driver_ID,
       d.First_Name || ' ' || d.Last_Name AS Driver_Name,
       ROUND(AVG(so.Fare),2) AS Avg_Fare
FROM   Driver d JOIN Service_Order so
       ON d.Driver_ID = so.Driver_ID
WHERE  so.Status = 'Completed'
GROUP BY d.Driver_ID, d.First_Name || ' ' || d.Last_Name
ORDER BY Avg_Fare DESC;
```

RidePickUpQueries_Data.sql

```
--------------------------------------------------------------------
-- CPS510 – Assignment 4 (Part 1)
-- Ride & Pickup DBMS – Sample Data Inserts
-- Authors: Shayaan Kirubakaran, Kirusanth Palakanthan, Ahmed Hasan
--------------------------------------------------------------------

------------------------------
-- Address Data
------------------------------
INSERT INTO Address VALUES (1, 'Canada', 'Ontario', 'Toronto', '123 King St', 'M5G1X5');
INSERT INTO Address VALUES (2, 'Canada', 'Ontario', 'Mississauga', '50 Lakeshore Rd', 'L5E2N2');
INSERT INTO Address VALUES (3, 'Canada', 'Ontario', 'Toronto', '321 Front St', 'M5V2T3');
INSERT INTO Address VALUES (4, 'Canada', 'Ontario', 'Oakville', '123 Oak St', 'L6J4L4');

------------------------------
-- Location Data
------------------------------
INSERT INTO Location VALUES (1, '456 Queen St', 'Toronto', 'Ontario', 'M5H2N2', 'Canada', NULL);
INSERT INTO Location VALUES (2, '789 Bay St', 'Toronto', 'Ontario', 'M5B2C5', 'Canada', NULL);
INSERT INTO Location VALUES (3, '200 Dundas St', 'Mississauga', 'Ontario', 'L5A1X2', 'Canada',
NULL);
INSERT INTO Location VALUES (4, '120 Lakeshore Rd', 'Mississauga', 'Ontario', 'L5G4G2', 'Canada',
NULL);

------------------------------
-- Customer Data
------------------------------
INSERT INTO Customer VALUES (1, 'John', 'Doe', '4161234567', 'john.doe@email.com',
   TO_DATE('1990-05-20','YYYY-MM-DD'), SYSDATE, 50.00, 1);
INSERT INTO Customer VALUES (2, 'Alice', 'Wong', '4165551234', 'alice.wong@email.com',
   TO_DATE('1992-07-15','YYYY-MM-DD'), SYSDATE, 100.00, 2);
INSERT INTO Customer VALUES (3, 'Bob', 'Singh', '6471112222', 'bob.singh@email.com',
   TO_DATE('1988-09-10','YYYY-MM-DD'), SYSDATE, 0.00, 3);
INSERT INTO Customer VALUES (4, 'Mia', 'Kim', '6471234567', 'mia.kim@email.com',
   TO_DATE('1995-04-25','YYYY-MM-DD'), SYSDATE, 75.00, 4);

------------------------------
-- Driver Data
------------------------------
INSERT INTO Driver VALUES (1, 'Jane', 'Smith', '4169876543',
TO_DATE('1985-10-10','YYYY-MM-DD'),
   'LIC12345', 'InsureCo #123', 1);
```

```
INSERT INTO Driver VALUES (2, 'Mark', 'Brown', '6472229999',
TO_DATE('1990-03-12','YYYY-MM-DD'),
   'LIC54321', NULL, 2);
INSERT INTO Driver VALUES (3, 'Ravi', 'Kumar', '9057778888',
TO_DATE('1982-01-25','YYYY-MM-DD'),
   'LIC98765', 'SafeDrive #456', 3);
INSERT INTO Driver VALUES (4, 'Sarah', 'Jones', '4166667777',
TO_DATE('1989-11-14','YYYY-MM-DD'),
   'LIC24680', 'BestDrive #789', 4);

------------------------------
-- Vehicle Data
------------------------------
INSERT INTO Vehicle VALUES (1, 'ABC123', 'Toyota', 'Camry', 'Black', 2020, 'Sedan', 1);
INSERT INTO Vehicle VALUES (2, 'XYZ987', 'Honda', 'Civic', 'Blue', 2018, 'Sedan', 2);
INSERT INTO Vehicle VALUES (3, 'LMN456', 'Ford', 'Escape', 'White', 2022, 'SUV', 3);
INSERT INTO Vehicle VALUES (4, 'DEF567', 'Nissan', 'Altima', 'Silver', 2021, 'Sedan', 4);

------------------------------
-- Merchant Data
------------------------------
INSERT INTO Merchant VALUES (1, 'Pizza Place', 1);
INSERT INTO Merchant VALUES (2, 'Coffee Corner', 2);
INSERT INTO Merchant VALUES (3, 'Sushi Express', 3);
INSERT INTO Merchant VALUES (4, 'Burger King', 4);

------------------------------
-- Service Orders
------------------------------
-- Completed order (Customer 1, Driver 1)
INSERT INTO Service_Order VALUES (1, 1, 1, 1, 'Completed', CURRENT_TIMESTAMP, 25.50, 1, 2);

-- Pending order (Customer 2, Driver 2)
INSERT INTO Service_Order VALUES (2, 2, 2, 2, 'Pending',   CURRENT_TIMESTAMP, 12.00, 2, 1);

-- Cancelled order (Customer 3, Driver 3)
INSERT INTO Service_Order VALUES (3, 3, 3, 3, 'Cancelled', CURRENT_TIMESTAMP, 30.00, 3, 1);

-- Completed order (Customer 2, Driver 1)
INSERT INTO Service_Order VALUES (4, 2, 1, 1, 'Completed', CURRENT_TIMESTAMP, 45.00, 1, 3);

-- Completed order (Customer 4, Driver 4)
INSERT INTO Service_Order VALUES (5, 4, 4, 4, 'Completed', CURRENT_TIMESTAMP, 40.00, 4, 2);
```

```
------------------------------
-- Payments
------------------------------
-- Payment for order 1
INSERT INTO Payment VALUES (1, 1, 'Credit', 25.50, 'Paid');
-- Payment for order 2
INSERT INTO Payment VALUES (2, 2, 'Debit',  12.00, 'Pending');
-- Payment for order 3
INSERT INTO Payment VALUES (3, 3, 'Balance',30.00, 'Failed');
-- Payment for order 4
INSERT INTO Payment VALUES (4, 4, 'Credit', 45.00, 'Paid');
-- Payment for order 5
INSERT INTO Payment VALUES (5, 5, 'Debit', 40.00, 'Paid');

------------------------------
-- Ratings
------------------------------
INSERT INTO Rating_System VALUES (1, 1, 1, 1, 5, 'Great driver!', 5, 'Polite customer');
INSERT INTO Rating_System VALUES (2, 2, 2, 2, 2, 'Driver was late', 3, 'Customer was fine');
INSERT INTO Rating_System VALUES (3, 3, 3, 3, 3, 'Trip cancelled, no feedback', 3, 'No show');
INSERT INTO Rating_System VALUES (4, 4, 2, 1, 4, 'Fast delivery!', 5, 'Great customer');
INSERT INTO Rating_System VALUES (5, 5, 4, 4, 5, 'Awesome ride!', 5, 'Friendly driver');

COMMIT;
```

Output:

ADDRESS_ID	STREET	CITY	PROVINCE	POSTAL_COD
1	123 King St	Toronto	Ontario	M5G1X5
3	321 Front St	Toronto	Ontario	M5V2T3

CUSTOMER_ID	CUSTOMER_NAME	BALANCE	STATUS
2	Alice Wong	100	Active
4	Mia Kim	75	Active
1	John Doe	50	Active
3	Bob Singh	0	Inactive

DRIVER_ID	DRIVER_NAME	LICENSE_INFO	INSURANCE_INFO
1	Jane Smith	LIC12345	InsureCo #123
3	Ravi Kumar	LIC98765	SafeDrive #456
4	Sarah Jones	LIC24680	BestDrive #789

VEHICLE_ID	LICENSE_PLATE	MAKE	MODEL	YEAR	VEHICLE_TYPE
1	ABC123	Toyota	Camry	2020	Sedan
2	XYZ987	Honda	Civic	2018	Sedan
3	LMN456	Ford	Escape	2022	SUV
4	DEF567	Nissan	Altima	2021	Sedan

MERCHANT_ID	MERCHANT_NAME	CITY	PROVINCE
4	Burger King	Oakville	Ontario
2	Coffee Corner	Mississauga	Ontario
3	Sushi Express	Toronto	Ontario
1	Pizza Place	Toronto	Ontario

LOCATION_ID	STREET_ADDRESS	CITY	PROVINCE	POSTAL_COD
1	456 Queen St	Toronto	Ontario	M5H2N2
2	789 Bay St	Toronto	Ontario	M5B2C5
3	200 Dundas St	Mississauga	Ontario	L5A1X2
4	120 Lakeshore Rd	Mississauga	Ontario	L5G4G2

ORDER_ID	CUSTOMER_ID	DRIVER_ID	FARE	STATUS
1	1	1	25.5	Completed
4	2	1	45	Completed
5	4	4	40	Completed

CREDIT_PAYMENTS
2

RATING_ID	ORDER_ID	CUSTOMER_STARS	CUSTOMER_FEEDBACK
2	2	2	Driver was late

ORDER_ID	CUSTOMER_NAME	DRIVER_NAME	FARE
4	Alice Wong	Jane Smith	45
5	Mia Kim	Sarah Jones	40
1	John Doe	Jane Smith	25.5

PAYMENT_ID	PAYMENT_TYPE	AMOUNT	ORDER_ID	CUSTOMER_NAME
4	Credit	45	4	Alice Wong
5	Debit	40	5	Mia Kim
3	Balance	30	3	Bob Singh
1	Credit	25.5	1	John Doe
2	Debit	12	2	Alice Wong

CUSTOMER_ID	CUSTOMER_NAME	TOTAL_ORDERS
2	Alice Wong	2
4	Mia Kim	1
1	John Doe	1
3	Bob Singh	1

DRIVER_ID	DRIVER_NAME	AVG_FARE
4	Sarah Jones	40
1	Jane Smith	35.25

ADDRESS_ID	STREET	CITY	PROVINCE	POSTAL_COD
1	123 King St	Toronto	Ontario	M5G1X5
3	321 Front St	Toronto	Ontario	M5V2T3

CUSTOMER_ID	CUSTOMER_NAME	BALANCE	STATUS
2	Alice Wong	100	Active
4	Mia Kim	75	Active
1	John Doe	50	Active
3	Bob Singh	0	Inactive

DRIVER_ID	DRIVER_NAME	LICENSE_INFO	INSURANCE_INFO
1	Jane Smith	LIC12345	InsureCo #123
3	Ravi Kumar	LIC98765	SafeDrive #456
4	Sarah Jones	LIC24680	BestDrive #789

VEHICLE_ID	LICENSE_PLATE	MAKE	MODEL	YEAR	VEHICLE_TYPE
1	ABC123	Toyota	Camry	2020	Sedan
2	XYZ987	Honda	Civic	2018	Sedan
3	LMN456	Ford	Escape	2022	SUV
4	DEF567	Nissan	Altima	2021	Sedan

MERCHANT_ID	MERCHANT_NAME	CITY	PROVINCE
4	Burger King	Oakville	Ontario
2	Coffee Corner	Mississauga	Ontario
3	Sushi Express	Toronto	Ontario
1	Pizza Place	Toronto	Ontario

LOCATION_ID	STREET_ADDRESS	CITY	PROVINCE	POSTAL_COD
1	456 Queen St	Toronto	Ontario	M5H2N2
2	789 Bay St	Toronto	Ontario	M5B2C5
3	200 Dundas St	Mississauga	Ontario	L5A1X2
4	120 Lakeshore Rd	Mississauga	Ontario	L5G4G2

ORDER_ID	CUSTOMER_ID	DRIVER_ID	FARE	STATUS
1	1	1	25.5	Completed
4	2	1	45	Completed
5	4	4	40	Completed

CREDIT_PAYMENTS
2

RATING_ID	ORDER_ID	CUSTOMER_STARS	CUSTOMER_FEEDBACK
2	2	2	Driver was late

ORDER_ID	CUSTOMER_NAME	DRIVER_NAME	FARE
4	Alice Wong	Jane Smith	45
5	Mia Kim	Sarah Jones	40
1	John Doe	Jane Smith	25.5

PAYMENT_ID	PAYMENT_TYPE	AMOUNT	ORDER_ID	CUSTOMER_NAME
4	Credit	45	4	Alice Wong
5	Debit	40	5	Mia Kim
3	Balance	30	3	Bob Singh
1	Credit	25.5	1	John Doe
2	Debit	12	2	Alice Wong

CUSTOMER_ID	CUSTOMER_NAME	TOTAL_ORDERS
2	Alice Wong	2
4	Mia Kim	1
1	John Doe	1
3	Bob Singh	1

DRIVER_ID	DRIVER_NAME	AVG_FARE
4	Sarah Jones	40
1	Jane Smith	35.25

Part 2: SQL Views & Advanced Queries

In Week 4, Part 2 of the project, we expanded on our initial SQL queries by creating views and more advanced operations for the Ride & Pickup DBMS. The purpose of this phase was to demonstrate that our database design could support reporting, data summarization, and multi-table analysis using SQL features such as JOINs, DISTINCT, ORDER BY, and aggregate functions. These queries and views were designed to reflect real business use cases for a ride-sharing and delivery platform, allowing us to present organized and meaningful results from multiple related tables.

We were required to implement at least two views, each providing simplified access to frequently used information in the system. Our views focused on summarizing data from the Service_Order, Customer, Driver, Payment, and Rating_System tables. In addition to views, we developed more advanced queries that joined multiple tables and displayed combined transactional information such as order histories, payment details, and rating summaries. These outputs were used to verify that the relationships created in Week 3 functioned correctly and that the system could return detailed, multi-table results in a user-friendly format.

Below are the views and advanced queries used in our submission.

```
------------------------------------------------------------
-- CPS510 – Assignment 4 (Part 2)
-- Ride & Pickup DBMS – 2 Views, Join Queries, Advanced Queries
-- Authors: Shayaan Kirubakaran, Kirusanth Palakanthan, Ahmed Hasan
------------------------------------------------------------
-- ============================================================
-- ========================= VIEWS ============================
-- ============================================================

-- ====================================
-- VIEW 1: Active Customer Orders
-- Shows all active orders for customers with status 'Pending' or 'In Progress'
-- ====================================
CREATE OR REPLACE VIEW Active_Customer_Orders AS
SELECT DISTINCT c.Customer_ID,
      c.First_Name || ' ' || c.Last_Name AS Customer_Name,
      so.Order_ID,
      so.Status,
      so.Fare
FROM Customer c
JOIN Service_Order so ON c.Customer_ID = so.Customer_ID
WHERE so.Status IN ('Pending', 'In Progress');

-- Display view results
SELECT * FROM Active_Customer_Orders;

-- ====================================
-- VIEW 2: Driver Earnings Summary
-- Shows total earnings for each driver from completed orders
-- ====================================
CREATE OR REPLACE VIEW Driver_Earnings AS
SELECT d.Driver_ID,
      d.First_Name || ' ' || d.Last_Name AS Driver_Name,
      SUM(so.Fare) AS Total_Earnings
FROM Driver d
JOIN Service_Order so ON d.Driver_ID = so.Driver_ID
JOIN Payment p ON so.Order_ID = p.Order_ID
WHERE so.Status = 'Completed'
GROUP BY d.Driver_ID, d.First_Name, d.Last_Name
ORDER BY Total_Earnings DESC;

-- Display view results
SELECT * FROM Driver_Earnings;
```

```
------------------------------------------------------------
-- ========================= JOIN QUERIES ======================
------------------------------------------------------------

-- ====================================
-- Join Query 1: Customer Feedback on Completed Orders
-- Shows customer ratings and feedback with driver details
-- ====================================
SELECT DISTINCT c.Customer_ID,
      c.First_Name || ' ' || c.Last_Name AS Customer_Name,
      so.Order_ID,
      rs.Customer_Stars AS Customer_Rating,
      rs.Customer_Feedback,
      d.First_Name || ' ' || d.Last_Name AS Driver_Name
FROM Service_Order so
JOIN Customer c ON so.Customer_ID = c.Customer_ID
JOIN Rating_System rs ON so.Order_ID = rs.Order_ID
JOIN Driver d ON so.Driver_ID = d.Driver_ID
WHERE so.Status = 'Completed'
ORDER BY so.Order_ID;

-- ====================================
-- Join Query 2: Merchants and Their Delivery Orders
-- Shows which merchants fulfilled which orders with city and province
-- ====================================
SELECT DISTINCT m.Merchant_ID,
      m.Name AS Merchant_Name,
      so.Order_ID,
      so.Status
FROM Merchant m
JOIN Service_Order so ON m.Merchant_ID = so.Merchant_ID
WHERE so.Status IN ('Completed', 'In Progress')
ORDER BY m.Merchant_ID;

-- ====================================
-- Join Query 3: Completed Orders with Customer & Driver Info
-- ====================================
SELECT so.Order_ID,
      c.First_Name || ' ' || c.Last_Name AS Customer_Name,
      d.First_Name || ' ' || d.Last_Name AS Driver_Name,
      so.Fare
FROM Service_Order so
JOIN Customer c ON so.Customer_ID = c.Customer_ID
JOIN Driver d ON so.Driver_ID = d.Driver_ID
```

```
WHERE so.Status = 'Completed'
ORDER BY so.Fare DESC;

-------------------------------------------------------------
-- ================== ADVANCED / AGGREGATION QUERIES =========
-------------------------------------------------------------

-- ====================================
-- Advanced Query 1: Total Orders by Customer
-- ====================================
SELECT c.Customer_ID,
       c.First_Name || ' ' || c.Last_Name AS Customer_Name,
       COUNT(so.Order_ID) AS Total_Orders
FROM Customer c
LEFT JOIN Service_Order so ON c.Customer_ID = so.Customer_ID
GROUP BY c.Customer_ID, c.First_Name, c.Last_Name
ORDER BY Total_Orders DESC;

-- ====================================
-- Advanced Query 2: Average Fare per Driver
-- ====================================
SELECT d.Driver_ID,
       d.First_Name || ' ' || d.Last_Name AS Driver_Name,
       ROUND(AVG(so.Fare), 2) AS Avg_Fare
FROM Driver d
JOIN Service_Order so ON d.Driver_ID = so.Driver_ID
WHERE so.Status = 'Completed'
GROUP BY d.Driver_ID, d.First_Name, d.Last_Name
ORDER BY Avg_Fare DESC;

-- ====================================
-- Advanced Query 3: Total Earnings per Merchant
-- ====================================
SELECT m.Merchant_ID,
       m.Name AS Merchant_Name,
       SUM(so.Fare) AS Total_Earnings
FROM Merchant m
JOIN Service_Order so ON m.Merchant_ID = so.Merchant_ID
WHERE so.Status = 'Completed'
GROUP BY m.Merchant_ID, m.Name
ORDER BY Total_Earnings DESC;

-- ====================================
-- Advanced Query 4: Service Orders with Customer, Driver, Payment Info
```

```
-- ====================================
SELECT so.Order_ID,
    c.First_Name || ' ' || c.Last_Name AS Customer_Name,
    d.First_Name || ' ' || d.Last_Name AS Driver_Name,
    p.Amount AS Payment_Amount,
    so.Fare AS Service_Fare,
    so.Status AS Order_Status
FROM Service_Order so
JOIN Customer c ON so.Customer_ID = c.Customer_ID
JOIN Driver d ON so.Driver_ID = d.Driver_ID
JOIN Payment p ON so.Order_ID = p.Order_ID
WHERE so.Status = 'Completed'
ORDER BY so.Order_ID;

-- ====================================
-- Advanced Query 5: Total and Average Fare per Customer
-- ====================================
SELECT c.Customer_ID,
    c.First_Name || ' ' || c.Last_Name AS Customer_Name,
    COUNT(so.Order_ID) AS Total_Orders,
    SUM(so.Fare) AS Total_Fare,
    ROUND(AVG(so.Fare), 2) AS Avg_Fare
FROM Customer c
JOIN Service_Order so ON c.Customer_ID = so.Customer_ID
WHERE so.Status = 'Completed'
GROUP BY c.Customer_ID, c.First_Name, c.Last_Name
ORDER BY Total_Fare DESC;

-- ====================================
-- Advanced Query 6: Driver's Completed Orders and Total Earnings
-- ====================================
SELECT d.Driver_ID,
    d.First_Name || ' ' || d.Last_Name AS Driver_Name,
    COUNT(so.Order_ID) AS Total_Completed_Orders,
    SUM(so.Fare) AS Total_Earnings
FROM Driver d
JOIN Service_Order so ON d.Driver_ID = so.Driver_ID
WHERE so.Status = 'Completed'
GROUP BY d.Driver_ID, d.First_Name, d.Last_Name
ORDER BY Total_Earnings DESC;
```

Views Output:

Columns | Data | Grants | Dependencies | Details | Triggers | SQL | Errors

Actions...

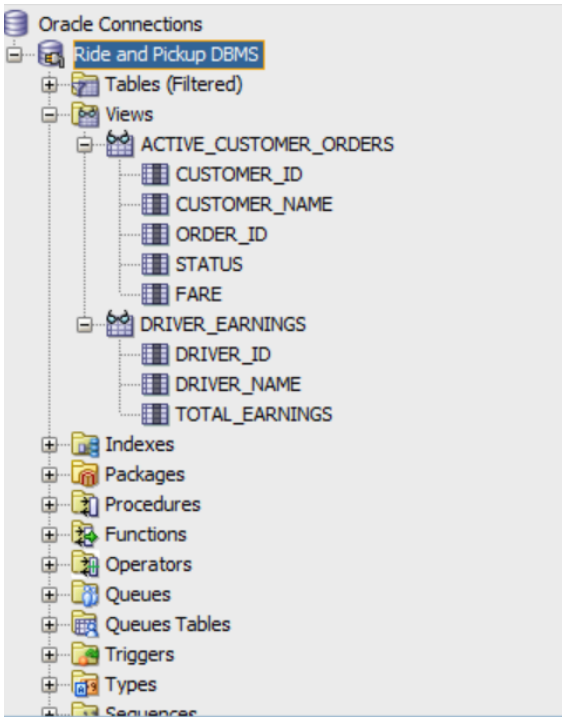
	COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS	INSERTABLE	UPDATABLE	DELETABLE
1	CUSTOMER_ID	NUMBER	No	(null)	1	(null)	NO	NO	NO
2	CUSTOMER_NAME	VARCHAR2(101)	Yes	(null)	2	(null)	NO	NO	NO
3	ORDER_ID	NUMBER	No	(null)	3	(null)	YES	YES	YES
4	STATUS	VARCHAR2(20)	Yes	(null)	4	(null)	YES	YES	YES
5	FARE	NUMBER(10,2)	Yes	(null)	5	(null)	YES	YES	YES

assignment_4_part2.sql | ridepickup_data.sql | DRIVER_EARNINGS

Columns | Data | Grants | Dependencies | Details | Triggers | SQL | Errors

Actions...

	COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS	INSERTABLE	UPDATABLE	DELETABLE
1	DRIVER_ID	NUMBER	No	(null)	1	(null)	NO	NO	NO
2	DRIVER_NAME	VARCHAR2(101)	Yes	(null)	2	(null)	NO	NO	NO
3	TOTAL_EARNINGS	NUMBER	Yes	(null)	3	(null)	NO	NO	NO



Week 5 – Unix Shell Menus & Advanced Queries

In Week 5, our group integrated Unix shell scripting with the Ride & Pickup DBMS to automate database operations and execute advanced SQL queries directly from the terminal. Using our menu-driven **menu.sh** script, users could drop tables, recreate the full schema, populate sample data, and run SQL queries without manually accessing SQL. Each component — including **create_tables.sh**, **drop_table.sh**, **populate_tables.sh**, and **queries.sh** — worked together to form a complete command-line interface for managing the system.

Our **queries.sh** script demonstrated advanced SQL operations such as multi-table joins across customers, drivers, vehicles, service orders, merchants, payments, and ratings. These queries produced organized outputs showing order histories, payment information, and driver assignments, confirming that our schema and constraints worked correctly when queried through automated scripts. Week 5 proved that the Ride & Pickup DBMS could operate effectively through Unix-based automation, supporting both database management and analytical queries via shell menus.

menu.sh

```
  GNU nano 6.2
#!/bin/sh
MainMenu() {
  while [ "$CHOICE" != "E" ]
  do
    clear
    echo "============================================================"
    echo "|           Ride & Pickup DBMS – Oracle Tool                |"
    echo "------------------------------------------------------------"
    echo " 1) Drop Tables"
    echo " 2) Create Tables"
    echo " 3) Populate Tables"
    echo " 4) Run Advanced Queries"
    echo " E) Exit"
    echo "------------------------------------------------------------"
    echo -n "Choose: "
    read CHOICE

    case $CHOICE in
      1) bash drop_table.sh;     read -p "Press Enter to continue..." ;;
      2) bash create_tables.sh;   read -p "Press Enter to continue..." ;;
      3) bash populate_tables.sh; read -p "Press Enter to continue..." ;;
      4) bash queries.sh;         read -p "Press Enter to continue..." ;;
      E) exit ;;
      *) echo "Invalid option."; sleep 1 ;;
    esac
  done
}

ProgramStart() {
  echo "Starting Ride & Pickup DBMS Menu..."
  sleep 1
  MainMenu
}

ProgramStart
```

create_tables.sh

```
  GNU nano 6.2
#!/bin/sh
echo -n "Enter Oracle password: "
read -s PASSWORD
echo
sqlplus64 "skirubak/$PASSWORD@(DESCRIPTION=
(ADDRESS=(PROTOCOL=TCP)(Host=oracle.scs.ryerson.ca)(Port=1521))
(CONNECT_DATA=(SID=orcl)))" <<EOF

-- Create all project tables
CREATE TABLE Customer (
  customerID   NUMBER PRIMARY KEY,
  name         VARCHAR2(50),
  age          NUMBER,
  address      VARCHAR2(100),
  balance      NUMBER(10,2)
);

CREATE TABLE Driver (
  driverID     NUMBER PRIMARY KEY,
  name         VARCHAR2(50),
  licenseNo    VARCHAR2(30),
  rating       NUMBER(2,1)
);

CREATE TABLE Vehicle (
  vehicleID    NUMBER PRIMARY KEY,
  driverID     NUMBER REFERENCES Driver(driverID),
  make         VARCHAR2(30),
  model        VARCHAR2(30),
  licensePlate VARCHAR2(15)
);

CREATE TABLE Merchant (
  merchantID   NUMBER PRIMARY KEY,
  name         VARCHAR2(50),
  location     VARCHAR2(100)
);

CREATE TABLE Location (
  locationID   NUMBER PRIMARY KEY,
  address      VARCHAR2(100),
  latitude     NUMBER(8,5),
  longitude    NUMBER(8,5)
);

CREATE TABLE ServiceOrder (
  orderID      NUMBER PRIMARY KEY,
  customerID   NUMBER REFERENCES Customer(customerID),
  driverID     NUMBER REFERENCES Driver(driverID),
  merchantID   NUMBER REFERENCES Merchant(merchantID),
  pickupID     NUMBER REFERENCES Location(locationID),
  dropoffID    NUMBER REFERENCES Location(locationID),
  fare         NUMBER(10,2),
  orderType    VARCHAR2(20),
  status       VARCHAR2(20)
);
```

```
CREATE TABLE Payment (
  paymentID    NUMBER PRIMARY KEY,
  orderID      NUMBER REFERENCES ServiceOrder(orderID),
  amount       NUMBER(10,2),
  method       VARCHAR2(20),
  status       VARCHAR2(20)
);

CREATE TABLE Rating (
  ratingID       NUMBER PRIMARY KEY,
  orderID        NUMBER REFERENCES ServiceOrder(orderID),
  customerRating NUMBER(2,1),
  driverRating   NUMBER(2,1),
  comments       VARCHAR2(200)
);

EXIT;
EOF
```

drop_table.sh

```
  GNU nano 6.2
#!/bin/sh
echo -n "Enter Oracle password: "
read -s PASSWORD
echo
sqlplus64 "skirubak/$PASSWORD@(DESCRIPTION=
(ADDRESS=(PROTOCOL=TCP)(Host=oracle.scs.ryerson.ca)(Port=1521))
(CONNECT_DATA=(SID=orcl)))" <<EOF

-- Drop all tables in reverse dependency order
DROP TABLE Rating CASCADE CONSTRAINTS;
DROP TABLE Payment CASCADE CONSTRAINTS;
DROP TABLE ServiceOrder CASCADE CONSTRAINTS;
DROP TABLE Vehicle CASCADE CONSTRAINTS;
DROP TABLE Driver CASCADE CONSTRAINTS;
DROP TABLE Customer CASCADE CONSTRAINTS;
DROP TABLE Merchant CASCADE CONSTRAINTS;
DROP TABLE Location CASCADE CONSTRAINTS;

EXIT;
EOF
```

populate_tables.sh

```
  GNU nano 6.2
#!/bin/sh
echo -n "Enter Oracle password: "
read -s PASSWORD
echo
sqlplus64 "skirubak/$PASSWORD@(DESCRIPTION=
(ADDRESS=(PROTOCOL=TCP)(Host=oracle.scs.ryerson.ca)(Port=1521))
(CONNECT_DATA=(SID=orcl)))" <<EOF

-- Clear any existing data before inserting new records
DELETE FROM Rating;
DELETE FROM Payment;
DELETE FROM ServiceOrder;
DELETE FROM Vehicle;
DELETE FROM Driver;
DELETE FROM Customer;
DELETE FROM Merchant;
DELETE FROM Location;

-- Customers
INSERT INTO Customer VALUES (1, 'Alice', 25, 'Toronto', 100.00);
INSERT INTO Customer VALUES (2, 'Bob', 31, 'Mississauga', 50.00);
INSERT INTO Customer VALUES (3, 'Cathy', 40, 'Scarborough', 200.00);
INSERT INTO Customer VALUES (4, 'David', 28, 'North York', 75.00);
INSERT INTO Customer VALUES (5, 'Evelyn', 33, 'Brampton', 60.00);

-- Drivers
INSERT INTO Driver VALUES (1, 'John Doe', 'ON12345', 4.8);
INSERT INTO Driver VALUES (2, 'Mary P',   'ON67890', 4.9);
INSERT INTO Driver VALUES (3, 'Kevin L',  'ON24680', 4.2);
INSERT INTO Driver VALUES (4, 'Rita Z',   'ON13579', 4.6);

-- Vehicles
INSERT INTO Vehicle VALUES (1, 1, 'Toyota', 'Camry', 'AB1234');
INSERT INTO Vehicle VALUES (2, 2, 'Honda', 'Civic', 'XY5678');
INSERT INTO Vehicle VALUES (3, 3, 'Nissan', 'Altima', 'RT4321');
INSERT INTO Vehicle VALUES (4, 4, 'Ford', 'Escape', 'QP9988');

-- Merchants
INSERT INTO Merchant VALUES (1, 'Pizza Palace', 'Toronto');
INSERT INTO Merchant VALUES (2, 'Burger Barn', 'Mississauga');
INSERT INTO Merchant VALUES (3, 'Taco Town', 'Scarborough');

-- Locations
INSERT INTO Location VALUES (1, '123 King St', 43.6532, -79.3832);
INSERT INTO Location VALUES (2, '456 Queen St', 43.6530, -79.3800);
INSERT INTO Location VALUES (3, '789 Bloor St', 43.6600, -79.4000);
INSERT INTO Location VALUES (4, '321 Dundas St', 43.6500, -79.3800);
INSERT INTO Location VALUES (5, '555 Bay St', 43.6550, -79.3805);
```

```
-- Service Orders
INSERT INTO ServiceOrder VALUES (1, 1, 1, 1, 1, 2, 18.50, 'Delivery', 'Completed');
INSERT INTO ServiceOrder VALUES (2, 2, 2, 2, 3, 4, 25.00, 'Delivery', 'Completed');
INSERT INTO ServiceOrder VALUES (3, 3, 3, 3, 2, 5, 32.75, 'Ride', 'Completed');
INSERT INTO ServiceOrder VALUES (4, 4, 4, 1, 4, 1, 20.00, 'Ride', 'Cancelled');
INSERT INTO ServiceOrder VALUES (5, 5, 1, 2, 5, 3, 15.50, 'Delivery', 'Completed');
INSERT INTO ServiceOrder VALUES (6, 3, 2, 1, 3, 1, 40.00, 'Ride', 'Completed');
INSERT INTO ServiceOrder VALUES (7, 2, 4, 3, 5, 4, 12.25, 'Delivery', 'Pending');
INSERT INTO ServiceOrder VALUES (8, 1, 3, 2, 2, 3, 21.50, 'Ride', 'Completed');

-- Payments
INSERT INTO Payment VALUES (1, 1, 18.50, 'Credit', 'Success');
INSERT INTO Payment VALUES (2, 2, 25.00, 'Credit', 'Success');
INSERT INTO Payment VALUES (3, 3, 32.75, 'Debit', 'Success');
INSERT INTO Payment VALUES (4, 4, 20.00, 'Credit', 'Failed');
INSERT INTO Payment VALUES (5, 5, 15.50, 'Debit', 'Success');
INSERT INTO Payment VALUES (6, 6, 40.00, 'Credit', 'Success');
INSERT INTO Payment VALUES (7, 7, 12.25, 'Credit', 'Pending');
INSERT INTO Payment VALUES (8, 8, 21.50, 'Credit', 'Success');

-- Ratings (customers who rated)
INSERT INTO Rating VALUES (1, 1, 5, 4.8, 'Fast delivery!');
INSERT INTO Rating VALUES (2, 2, 4, 4.9, 'Great driver!');
INSERT INTO Rating VALUES (3, 3, 5, 4.2, 'Comfortable ride');
INSERT INTO Rating VALUES (4, 5, 3, 4.7, 'Late pickup');
INSERT INTO Rating VALUES (5, 6, 5, 4.9, 'Excellent service');
-- Note: Orders 4, 7, and 8 are missing ratings to make Query #3 work

COMMIT;
EXIT;
EOF
```

queries.sh

```
  GNU nano 6.2
#!/bin/sh
echo -n "Enter Oracle password: "
read -s PASSWORD
echo
sqlplus64 "skirubak/$PASSWORD@(DESCRIPTION=
(ADDRESS=(PROTOCOL=TCP)(Host=oracle.scs.ryerson.ca)(Port=1521))
(CONNECT_DATA=(SID=orcl)))" <<EOF

-- Formatting for cleaner query outputs
SET PAGESIZE 200
SET LINESIZE 200
COLUMN Customer FORMAT A20
COLUMN Driver   FORMAT A20
COLUMN address  FORMAT A30

PROMPT
PROMPT === 1) Join: List all completed orders with customer and driver names ===
SELECT s.orderID,
       c.name AS Customer,
       d.name AS Driver,
       s.status,
       s.fare
FROM   ServiceOrder s
       JOIN Customer c ON s.customerID = c.customerID
       JOIN Driver d   ON s.driverID   = d.driverID
WHERE  s.status = 'Completed';

PROMPT
PROMPT === 2) Aggregation: Average rating per driver ===
SELECT d.name,
       ROUND(AVG(r.driverRating),2) AS AvgRating
FROM   Driver d
       JOIN ServiceOrder s ON d.driverID = s.driverID
       JOIN Rating r       ON s.orderID  = r.orderID
GROUP  BY d.name;

PROMPT
PROMPT === 3) Set Operation: Customers who paid but have not rated ===
SELECT customerID
FROM   ServiceOrder
MINUS
SELECT s.customerID
FROM   ServiceOrder s
       JOIN Rating r ON s.orderID = r.orderID;
```

```
PROMPT
PROMPT === 4) Nested Query: Drivers with earnings above average ===
SELECT name
FROM   Driver
WHERE  driverID IN (
         SELECT driverID
         FROM   ServiceOrder
         GROUP  BY driverID
         HAVING SUM(fare) > (SELECT AVG(fare) FROM ServiceOrder)
       );

PROMPT
PROMPT === 5) Statistical Query: Total earnings by drop-off address ===
SELECT l.address,
       SUM(s.fare) AS TotalEarnings
FROM   ServiceOrder s
       JOIN Location l ON s.dropoffID = l.locationID
GROUP  BY l.address
ORDER  BY TotalEarnings DESC;

EXIT;
EOF
```

User interface output:

```
============================================================
|           Ride & Pickup DBMS – Oracle Tool               |
------------------------------------------------------------
 1) Drop Tables
 2) Create Tables
 3) Populate Tables
 4) Run Advanced Queries
 E) Exit
------------------------------------------------------------
Choose:
```

```
============================================================
|           Ride & Pickup DBMS — Oracle Tool               |
------------------------------------------------------------
 1) Drop Tables
 2) Create Tables
 3) Populate Tables
 4) Run Advanced Queries
 E) Exit
------------------------------------------------------------
Choose: 1
Enter Oracle password:

SQL*Plus: Release 12.1.0.2.0 Production on Fri Oct 24 22:23:54 2025

Copyright (c) 1982, 2014, Oracle.  All rights reserved.


Connected to:
Oracle Database 11g Enterprise Edition Release 11.2.0.1.0 - 64bit Production
With the Partitioning, OLAP, Data Mining and Real Application Testing options

SQL> SQL> SQL>
Table dropped.

SQL>
Table dropped.

SQL>
Table dropped.

SQL>
Table dropped.

SQL>
Table dropped.

SQL>
Table dropped.

SQL>
Table dropped.

SQL>
Table dropped.

SQL> SQL> Disconnected from Oracle Database 11g Enterprise Edition Release 11.2.0.1.0 - 64bit Production
With the Partitioning, OLAP, Data Mining and Real Application Testing options
Press Enter to continue...
```

```
============================================================
|          Ride & Pickup DBMS – Oracle Tool                |
------------------------------------------------------------
 1) Drop Tables
 2) Create Tables
 3) Populate Tables
 4) Run Advanced Queries
 E) Exit
------------------------------------------------------------
Choose: 2
Enter Oracle password:

SQL*Plus: Release 12.1.0.2.0 Production on Fri Oct 24 22:24:26 2025

Copyright (c) 1982, 2014, Oracle.  All rights reserved.


Connected to:
Oracle Database 11g Enterprise Edition Release 11.2.0.1.0 - 64bit Production
With the Partitioning, OLAP, Data Mining and Real Application Testing options

SQL> SQL> SQL>   2    3    4    5    6    7
Table created.

SQL> SQL>   2    3    4    5    6
Table created.

SQL> SQL>   2    3    4    5    6    7
Table created.

SQL> SQL>   2    3    4    5
Table created.

SQL> SQL>   2    3    4    5    6
Table created.

SQL> SQL>   2    3    4    5    6    7    8    9   10   11
Table created.

SQL> SQL>   2    3    4    5    6    7
Table created.

SQL> SQL>   2    3    4    5    6    7
Table created.

SQL> SQL> Disconnected from Oracle Database 11g Enterprise Edition Release 11.2.0.1.0 - 64bit Production
With the Partitioning, OLAP, Data Mining and Real Application Testing options
Press Enter to continue...
```

```
============================================================
|           Ride & Pickup DBMS — Oracle Tool               |
------------------------------------------------------------
 1) Drop Tables
 2) Create Tables
 3) Populate Tables
 4) Run Advanced Queries
 E) Exit
------------------------------------------------------------
Choose: 3
Enter Oracle password:

SQL*Plus: Release 12.1.0.2.0 Production on Fri Oct 24 22:24:49 2025

Copyright (c) 1982, 2014, Oracle.  All rights reserved.


Connected to:
Oracle Database 11g Enterprise Edition Release 11.2.0.1.0 - 64bit Production
With the Partitioning, OLAP, Data Mining and Real Application Testing options

SQL> SQL> SQL>
0 rows deleted.

SQL>
0 rows deleted.

SQL>
0 rows deleted.

SQL>
0 rows deleted.

SQL>
0 rows deleted.

SQL>
0 rows deleted.

SQL>
0 rows deleted.

SQL>
0 rows deleted.

SQL> SQL> SQL>
1 row created.

SQL>
1 row created.

SQL>
1 row created.

SQL>
1 row created.

SQL>
1 row created.
```

```
SQL>
1 row created.

SQL>
1 row created.

SQL>
1 row created.

SQL>
1 row created.

SQL>
1 row created.

SQL> SQL> SQL>
1 row created.

SQL>
1 row created.

SQL>
1 row created.

SQL>
1 row created.

SQL>
1 row created.

SQL>
1 row created.

SQL>
1 row created.

SQL>
1 row created.

SQL>
1 row created.

SQL> SQL> SQL>
1 row created.

SQL>
1 row created.

SQL>
1 row created.

SQL>
1 row created.

SQL>
1 row created.

SQL>
1 row created.

SQL> SQL> SQL>
Commit complete.

SQL> Disconnected from Oracle Database 11g Enterprise Edition Release 11.2.0.1.0 - 64bit Production
With the Partitioning, OLAP, Data Mining and Real Application Testing options
Press Enter to continue...
```

```
============================================================
|           Ride & Pickup DBMS – Oracle Tool               |
------------------------------------------------------------
 1) Drop Tables
 2) Create Tables
 3) Populate Tables
 4) Run Advanced Queries
 E) Exit
------------------------------------------------------------
Choose: 4
Enter Oracle password:

SQL*Plus: Release 12.1.0.2.0 Production on Fri Oct 24 22:26:17 2025

Copyright (c) 1982, 2014, Oracle.  All rights reserved.


Connected to:
Oracle Database 11g Enterprise Edition Release 11.2.0.1.0 - 64bit Production
With the Partitioning, OLAP, Data Mining and Real Application Testing options

SQL> SQL> SQL> SQL> SQL> SQL> SQL> SQL> SQL>
SQL> === 1) Join: List all completed orders with customer and driver names ===
SQL>   2    3    4    5    6    7    8    9
   ORDERID CUSTOMER            DRIVER              STATUS                    FARE
---------- -------------------- ------------------- ------------------- ----------
         8 Alice                Kevin L             Completed                 21.5
         1 Alice                John Doe            Completed                 18.5
         2 Bob                  Mary P              Completed                   25
         6 Cathy                Mary P              Completed                   40
         3 Cathy                Kevin L             Completed                32.75
         5 Evelyn               John Doe            Completed                 15.5

6 rows selected.

SQL> SQL>
SQL> === 2) Aggregation: Average rating per driver ===
SQL>   2    3    4    5    6
NAME                                               AVGRATING
-------------------------------------------------- ----------
Kevin L                                                   4.2
Mary P                                                    4.9
John Doe                                                 4.75

SQL> SQL>
SQL> === 3) Set Operation: Customers who paid but have not rated ===
SQL>   2    3    4    5    6
CUSTOMERID
----------
         4
```

```
SQL> SQL>
SQL> === 4) Nested Query: Drivers with earnings above average ===
SQL>   2    3    4    5    6    7    8
NAME
--------------------------------------------------
John Doe
Mary P
Rita Z
Kevin L

SQL> SQL>
SQL> === 5) Statistical Query: Total earnings by drop-off address ===
SQL>   2    3    4    5    6
ADDRESS                        TOTALEARNINGS
------------------------------ -------------
123 King St                               60
321 Dundas St                          37.25
789 Bloor St                              37
555 Bay St                             32.75
456 Queen St                            18.5

SQL> SQL> Disconnected from Oracle Database 11g Enterprise Edition Release 11.2.0.1.0 - 64bit Production
With the Partitioning, OLAP, Data Mining and Real Application Testing options
Press Enter to continue...
```

Week 6 – Functional Dependencies

In Week 6, our group worked on identifying and documenting the functional dependencies (FDs) for every table in our Ride & Pickup DBMS. This marked the beginning of the normalization phase, where we verified that our schema followed the logical rules of the system and avoided redundancy. The requirement for Assignment 6 was to list the full set of FDs based strictly on the business rules and table structures, without yet performing decomposition.

Using our finalized ER diagram and Oracle table definitions, we identified the primary key for each relation and documented which attributes were determined by that key. Just like in many transactional systems, each table in our Ride & Pickup DBMS is built around a single-attribute primary key. As a result, each FD follows the standard logical structure:

Primary Key → All Non-Key Attributes

This matches the behavior of customers, drivers, vehicles, locations, service orders, payments, merchants, and ratings in real ride-sharing and delivery platforms. Below is the complete list of functional dependencies we produced, consistent with our Assignment 6 submission.

Address Table:

	COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
1	ADDRESS_ID	NUMBER	No	(null)	1	(null)
2	COUNTRY	VARCHAR2(50 BYTE)	No	(null)	2	(null)
3	PROVINCE	VARCHAR2(50 BYTE)	No	(null)	3	(null)
4	CITY	VARCHAR2(50 BYTE)	No	(null)	4	(null)
5	STREET_ADDRESS	VARCHAR2(100 BYTE)	No	(null)	5	(null)
6	POSTAL_CODE	VARCHAR2(10 BYTE)	No	(null)	6	(null)

Customer Table:

	COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
1	CUSTOMERID	NUMBER	No	(null)	1	(null)
2	NAME	VARCHAR2(50 BYTE)	Yes	(null)	2	(null)
3	AGE	NUMBER	Yes	(null)	3	(null)
4	ADDRESS	VARCHAR2(100 BYTE)	Yes	(null)	4	(null)
5	BALANCE	NUMBER(10,2)	Yes	(null)	5	(null)

Driver Table:

	COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
1	DRIVERID	NUMBER	No	(null)	1	(null)
2	NAME	VARCHAR2(50 BYTE)	Yes	(null)	2	(null)
3	LICENSENO	VARCHAR2(30 BYTE)	Yes	(null)	3	(null)
4	RATING	NUMBER(2,1)	Yes	(null)	4	(null)

Location Table:

	COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
1	LOCATIONID	NUMBER	No	(null)	1	(null)
2	ADDRESS	VARCHAR2(100 BYTE)	Yes	(null)	2	(null)
3	LATITUDE	NUMBER(8,5)	Yes	(null)	3	(null)
4	LONGITUDE	NUMBER(8,5)	Yes	(null)	4	(null)

Merchant Table

	COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
1	MERCHANTID	NUMBER	No	(null)	1	(null)
2	NAME	VARCHAR2(50 BYTE)	Yes	(null)	2	(null)
3	LOCATION	VARCHAR2(100 BYTE)	Yes	(null)	3	(null)

Payment Table:

	COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
1	PAYMENTID	NUMBER	No	(null)	1	(null)
2	ORDERID	NUMBER	Yes	(null)	2	(null)
3	AMOUNT	NUMBER(10,2)	Yes	(null)	3	(null)
4	METHOD	VARCHAR2(20 BYTE)	Yes	(null)	4	(null)
5	STATUS	VARCHAR2(20 BYTE)	Yes	(null)	5	(null)

Ratings Table:

	COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
1	RATINGID	NUMBER	No	(null)	1	(null)
2	ORDERID	NUMBER	Yes	(null)	2	(null)
3	CUSTOMERRATING	NUMBER(2,1)	Yes	(null)	3	(null)
4	DRIVERRATING	NUMBER(2,1)	Yes	(null)	4	(null)
5	COMMENTS	VARCHAR2(200 BYTE)	Yes	(null)	5	(null)

Rating System Table:

	COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
1	RATING_ID	NUMBER	No	(null)	1	(null)
2	ORDER_ID	NUMBER	No	(null)	2	(null)
3	CUSTOMER_ID	NUMBER	No	(null)	3	(null)
4	DRIVER_ID	NUMBER	No	(null)	4	(null)
5	CUSTOMER_STARS	NUMBER(1,0)	Yes	(null)	5	(null)
6	CUSTOMER_FEEDBACK	VARCHAR2(255 BYTE)	Yes	(null)	6	(null)
7	DRIVER_STARS	NUMBER(1,0)	Yes	(null)	7	(null)
8	DRIVER_FEEDBACK	VARCHAR2(255 BYTE)	Yes	(null)	8	(null)

Service_Order Table:

	COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
1	ORDER_ID	NUMBER	No	(null)	1	(null)
2	CUSTOMER_ID	NUMBER	No	(null)	2	(null)
3	DRIVER_ID	NUMBER	No	(null)	3	(null)
4	MERCHANT_ID	NUMBER	Yes	(null)	4	(null)
5	STATUS	VARCHAR2(20 BYTE)	Yes	'Pending'	5	(null)
6	ORDER_TIME	TIMESTAMP(6)	Yes	CURRENT_TIMESTAMP	6	(null)
7	FARE	NUMBER(10,2)	Yes	(null)	7	(null)
8	PICKUP_LOCATION_ID	NUMBER	No	(null)	8	(null)
9	DROPOFF_LOCATION_ID	NUMBER	No	(null)	9	(null)

Vehicle Table:

	COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
1	VEHICLEID	NUMBER	No	(null)	1	(null)
2	DRIVERID	NUMBER	Yes	(null)	2	(null)
3	MAKE	VARCHAR2(30 BYTE)	Yes	(null)	3	(null)
4	MODEL	VARCHAR2(30 BYTE)	Yes	(null)	4	(null)
5	LICENSEPLATE	VARCHAR2(15 BYTE)	Yes	(null)	5	(null)

Functional Dependencies

ADDRESS:
{Address_ID} → Country, Province, City, Street_Address, Postal_Code

CUSTOMER:
{CustomerID} → Name, Age, Address, Balance

DRIVER:
{DriverID} → Name, LicenseNo, Rating

VEHICLE:
{VehicleID} → DriverID, Make, Model, LicensePlate

LOCATION:
{LocationID} → Address, Latitude, Longitude

MERCHANT:
{MerchantID} → Name, Location

PAYMENT:
{PaymentID} → OrderID, Amount, Method, Status

RATING:
{RatingID} → OrderID, CustomerRating, DriverRating, Comments

SERVICE_ORDER:
{OrderID} → CustomerID, DriverID, MerchantID, PickupLocationID, DropoffLocationID, Fare, Status, OrderType, OrderTime

Explanation

In this database, each table has a primary key (PK) that uniquely identifies every record. The concept of a functional dependency (FD) means that the primary key *functionally determines* all the non-key attributes

within the same table.
This ensures that for every unique primary-key value, there is exactly one corresponding set of non-key attribute values.

For example:

- In the Customer table, the CustomerID uniquely determines the customer's Name, Age, Address, and Balance. No two customers share the same ID, and knowing the CustomerID allows the system to retrieve all that customer's information.

- In the Driver table, the DriverID determines the Name, LicenseNo, and Rating. This guarantees that each driver's data is stored once and referenced consistently across related tables.

- The Vehicle table links to the driver through DriverID, so VehicleID determines which driver the vehicle belongs to, as well as its make, model, and license plate.

- In the Service_Order table, the OrderID determines all details of that specific order, including which customer, driver, and merchant are involved, pickup and drop-off locations, fare, order status, and time. This single identifier ties together every element of a transaction.

These dependencies prove that each non-key attribute relies *only* on its table's primary key and not on any other non-key field. This eliminates redundancy, avoids update or deletion anomalies, and preserves data integrity. In summary, the primary key determines all non-key attributes in every table. This structure ensures that each table stores one type of information in the most efficient way possible, forming a strong foundation for the normalization and optimization steps that follow in Assignments 7 and 8.

Week 7 – 3NF Verification

In Week 7, our focus shifted from listing functional dependencies (FDs) to verifying whether each relation in our Ride & Pickup Database Management System satisfies Third Normal Form (3NF). Since our database is built around single-attribute primary keys and straightforward business rules for rides, deliveries, locations, payments, and ratings, the goal of this step was to ensure that no table contained partial or transitive dependencies that could introduce redundancy or update anomalies.

To verify 3NF, we followed the procedure discussed in the lecture and applied it to every relation in our schema:

1. Identify the primary key for each table.

All tables in our design use a single-attribute primary key (e.g., CustomerID, DriverID, VehicleID, LocationID, OrderID, PaymentID, RatingID).
Because there are no composite keys, partial dependencies cannot exist.

2. Check for partial dependencies.

Partial dependencies occur only when a relation has a composite key.
Since none of our tables use composite keys, **no partial dependencies are present.**

3. Check for transitive dependencies.

A transitive dependency exists when a non-key attribute depends on another non-key attribute rather than directly on the primary key.
Based on the FDs we derived in Week 6, each table follows the structure:
PrimaryKey → All Non-Key Attributes.
Because no non-key attribute determines any other non-key attribute, **no transitive dependencies were found** in any relation.

4. Decompose if violations exist.

After analyzing all relations, no decompositions were needed—every table already met the requirements of 3NF.
This outcome aligns with our Week 6 functional dependencies and confirms that the Ride & Pickup DBMS is structurally sound, free of redundancy, and logically consistent.

Bernstein's 3NF Algorithm with Example

To demonstrate the method used, we applied Bernstein's 3NF synthesis algorithm to one relation in our system. The same steps were executed for every table, but the following uses the **Customer** relation as an example.

Relation:

Customer(CustomerID, Name, Age, Address, Balance)

Functional Dependency From Business Rules:

CustomerID → Name, Age, Address, Balance

Step 1: Split Right-Hand Side Into Single-Attribute FDs

- CustomerID → Name
- CustomerID → Age
- CustomerID → Address

- CustomerID → Balance

Step 2: Minimal Cover (Redundancy Check)

For each FD, we check whether removing it still allows the right-hand side attribute to appear in the closure.

Example: Test if **CustomerID → Name** is redundant.

1. Remove CustomerID → Name
2. Compute $CustomerID^+$ using remaining FDs:

Start: $CustomerID^+ = \{CustomerID\}$
Apply CustomerID → Age → add Age
Apply CustomerID → Address → add Address
Apply CustomerID → Balance → add Balance

Result: {CustomerID, Age, Address, Balance}
Name is not in the closure → FD is **not** redundant.

Repeating for the other FDs yields the same result:
None of the dependencies are redundant.

Thus, the minimal cover is:

- CustomerID → Name
- CustomerID → Age
- CustomerID → Address
- CustomerID → Balance

Step 3: Candidate Key Identification (Using Attribute Closure)

Compute $CustomerID^+$ using the minimal cover:

Start: {CustomerID}
Add Name
Add Age
Add Address
Add Balance

Final closure:
CustomerID$^+$ = {CustomerID, Name, Age, Address, Balance}

Since this contains all attributes, **CustomerID is a key**.

Step 4: 3NF Synthesis – Create Relations for Each FD

From the minimal cover:

- CustomerID → Name → R_1(CustomerID, Name)
- CustomerID → Age → R_2(CustomerID, Age)
- CustomerID → Address → R_3(CustomerID, Address)
- CustomerID → Balance → R_4(CustomerID, Balance)

Step 5: Merge Relations With the Same Determinant

All relations share the same determinant, CustomerID.
Merging them produces:

R^c(CustomerID, Name, Age, Address, Balance)

This is identical to the original Customer table.

Therefore, the Customer relation is already in 3NF, lossless, and dependency-preserving.

Other Relations in the Ride & Pickup DBMS

We applied the same algorithm to all other tables, including:

- Driver(DriverID, Name, LicenseNo, Rating)
- Vehicle(VehicleID, DriverID, Make, Model, LicensePlate)
- Location(LocationID, Latitude, Longitude, Address)
- Merchant(MerchantID, Name, Location)

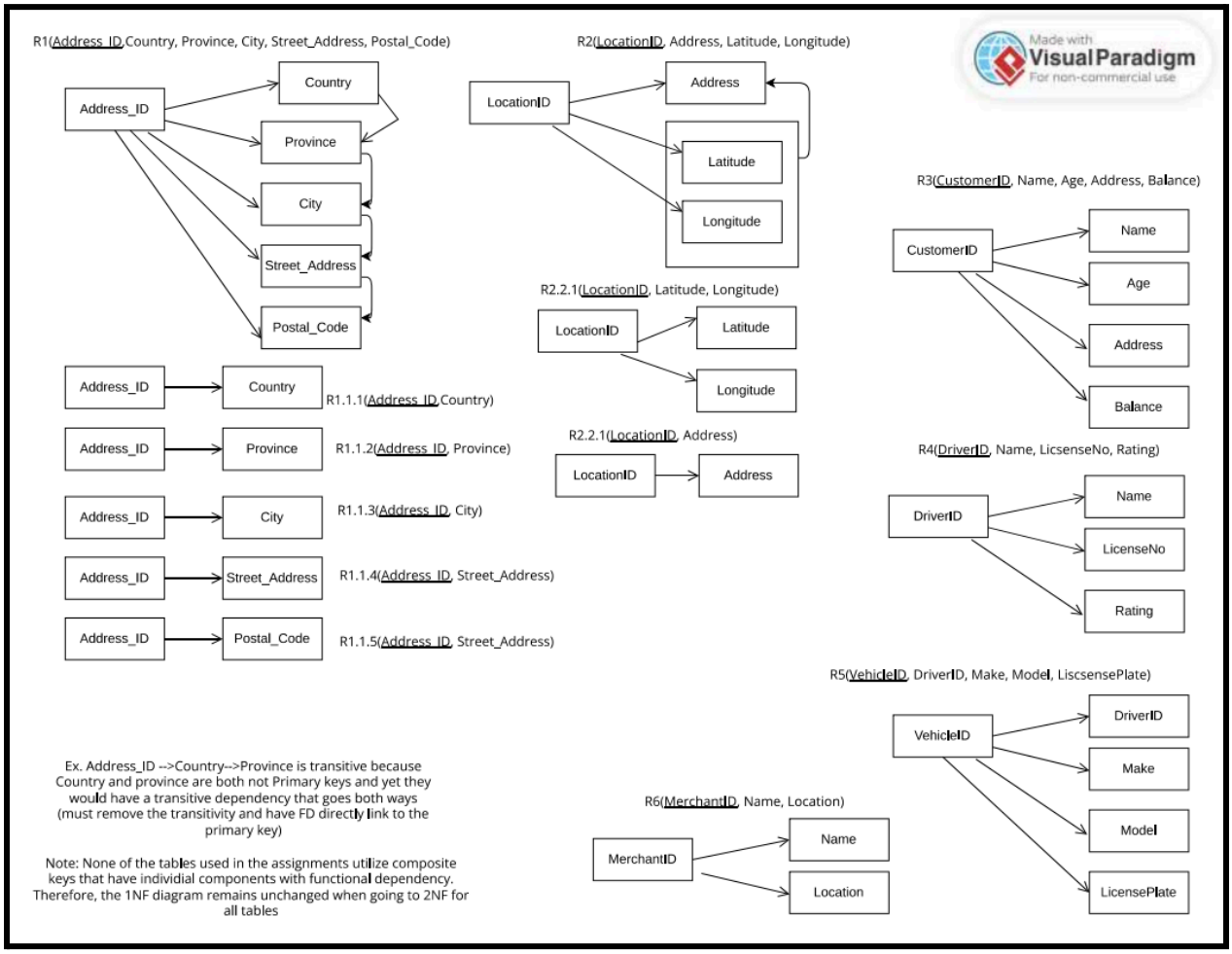
- ServiceOrder(OrderID, CustomerID, DriverID, MerchantID, PickupLocationID, DropoffLocationID, Status, OrderType, OrderTime, Fare)

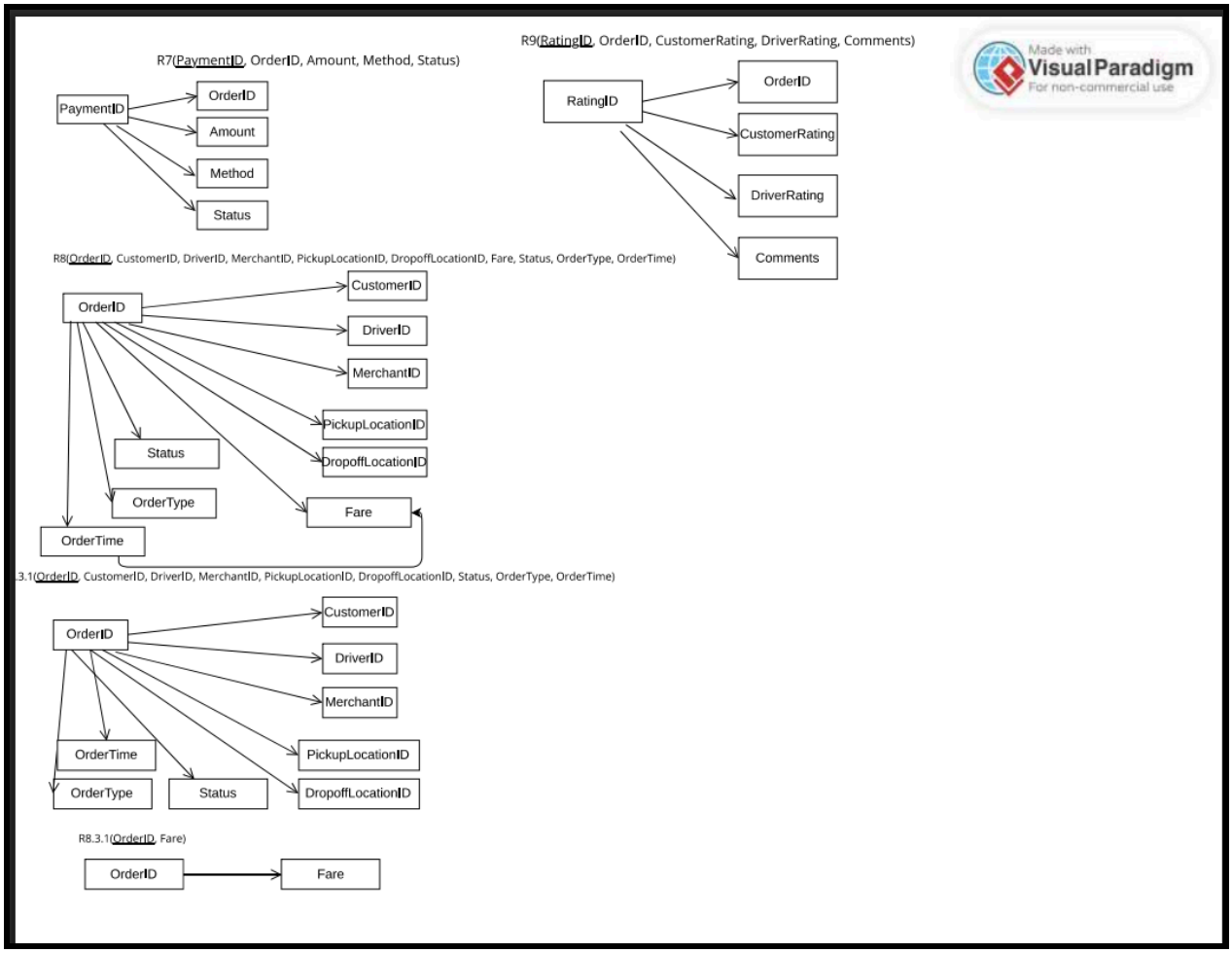
- Payment(PaymentID, OrderID, Amount, Method, Status)

- Rating(RatingID, OrderID, CustomerRating, DriverRating, Comments)

In every case, the functional dependencies were of the form:
PrimaryKey → all non-key attributes

Because of this, every relation is already in Third Normal Form, and the decomposition produced by Bernstein's algorithm results in the same original tables.





Week 8 – BCNF Verification

In Week 8, we extended our normalization work by checking whether each relation in our Ride & Pickup DBMS satisfies Boyce–Codd Normal Form (BCNF). BCNF applies a stricter rule than 3NF by requiring that, for every dependency X → Y, the determinant X must be a superkey of the relation.

Using the functional dependencies from Week 6 and the 3NF results from Week 7, we evaluated every table in our schema. Since all relations in our design use single-attribute primary keys (e.g., CustomerID, DriverID, VehicleID, OrderID, PaymentID), and every dependency follows the form **Primary Key → All Non-Key Attributes**, there are no cases where a non-key attribute determines another attribute.

Because no FD violates the BCNF condition, all relations in our Ride & Pickup DBMS are already in BCNF, and no decomposition was required.

R_1 (CustomerID, Name, Age, Address, Balance, Rating)
FD: CustomerID → Name, Age, Address, Balance, Rating

R_2 (DriverID, Name, LicenseNo, Contact, Rating)
FD: DriverID → Name, LicenseNo, Contact, Rating

R_3 (VehicleID, DriverID, Make, Model, Color, LicensePlate, InsuranceExpiry)
FD: VehicleID → DriverID, Make, Model, Color, LicensePlate, InsuranceExpiry
Notes: LicensePlate → VehicleID was avoided by using VehicleID as the identifier and enforcing uniqueness on LicensePlate.

R_4 (OrderID, CustomerID, DriverID, VehicleID, MerchantID, PickupLocationID, DropoffLocationID, Fare, Status, Timestamp)
FD: OrderID → CustomerID, DriverID, VehicleID, MerchantID, PickupLocationID, DropoffLocationID, Fare, Status, Timestamp

R_5 (LocationID, Address, Latitude, Longitude)
FD: LocationID → Address, Latitude, Longitude

R_6 (PaymentID, OrderID, Method, Amount, PaymentDate, CompletionStatus)
FD: PaymentID → OrderID, Method, Amount, PaymentDate, CompletionStatus

R_7 (MerchantID, Name, Category, Address, ContactInfo)
FD: MerchantID → Name, Category, Address, ContactInfo
Notes: Name → Category was treated as a possible business rule but not enforced as an FD since different merchants may share a name.

R_8 (RatingID, OrderID, CustomerID, DriverID, Stars, Comments)
FD: RatingID → OrderID, CustomerID, DriverID, Stars, Comments
Notes: (OrderID, CustomerID) → Stars was avoided by using a surrogate key RatingID to allow multiple ratings and updates without anomalies.

Each determinant is a key → every table already satisfies BCNF.

R_6 (Address_ID, Country, Province, City, street_address, postalCode)
FD: Address → Country, Province, City, street_address, postalCode

We apply the BCNF algorithm to one representative relation (R_4). The same logic applies to all others.

Input: R_4 (OrderID, CustomerID, DriverID, VehicleID, MerchantID, PickupLocationID, DropoffLocationID, Fare, Status, Timestamp)
F = { OrderID → CustomerID, DriverID, VehicleID, MerchantID, PickupLocationID, DropoffLocationID, Fare, Status, Timestamp }

Step 1 – Find Candidate Keys: $(OrderID)^+$ = all attributes ⇒ OrderID is a key.

Step 2 – Test BCNF Condition: For each FD X → Y, is X a (super)key? Yes. X = OrderID (a key) → passes.

Step 3 – Decompose if Violated: No violations → no split required.

Step 4 – Verify All Relations: Every relation's determinant is its key → all are BCNF.

If future business rules add new FDs (for example, LicensePlate → VehicleID or MerchantName → Category), those specific tables would require decomposition. Under the current design, however, all relations pass the BCNF test.

Note: a majority of the tables that were unchanged in 3NF when in 1NF originally are also unchanged for BCNF. The reason is because there are no redundant FDs and there is already a superkey shown that is able to represent all relationships when a closure is formed. Ex. The customer table (FD: CustomerID → Name, Age, Address, Balance, Rating) has no transitivity seen, CustomerID+={Name, Age, Address, Balance, Rating}, and no subset can be made that still makes this closure hold. More complicated examples that did have transitivity are more likely to have change when going to BCNF

R_4 (OrderID, CustomerID, DriverID, VehicleID, MerchantID, PickupLocationID, DropoffLocationID, Fare, Status, Timestamp)
FD: [OrderID → CustomerID, DriverID, VehicleID, MerchantID, PickupLocationID, DropoffLocationID, Fare, Status, Timestamp], [Timestamp → Fare]

*Timestamp+={Timestamp, Fare} violates BCNF because it doesn't include all attributes. This relation must be decomposed

R_4.1 (Timestamp, Fare)
R_4.2 (OrderID, CustomerID, DriverID, VehicleID, MerchantID, PickupLocationID, DropoffLocationID, Status, Timestamp)

R_5 (LocationID, Address, Latitude, Longitude)
FD: [LocationID → Address, Latitude, Longitude], [Latitude, Longitude → Address]

*(Latitude, Longitude)+={Latitude, Longitude, Address} violates BCNF because they are non key attributes and Location is not included in the closure. This means the relationship must be decomposed into two separate ones.

$R_{5.1}$ (Latitude, Longitude,Address)

$R_{5.2}$ (Location, Latitude, Longitude)

R_6 (Address_ID, Country, Province, City, street_address, postalCode)

FD: [Address → Country, Province, City, street_address, postalCode], [Country→ Province], [Province→ City], [City→ street_address], [street_address→ PostalCode]

*Country+={Country, Province, City, street_address, PostalCode} violates BCNF due to Address_ID not being included. Each of the following FDs must be broken down

$R_{6.1}$ (Country,Province)

$R_{6.2}$ (Province, City)

$R_{6.3}$ (City, street_address)

$R_{6.4}$ (street_address, PostalCode)

$R_{6.5}$ (Address_ID, street_address)

Week 9 – Web UI Demonstration (PHP, HTML + Oracle)

In Week 9, our group completed the final implementation of the Ride & Pickup DBMS by developing and demonstrating a fully functioning web-based user interface connected to Oracle. This assignment required us to integrate all earlier stages—entity design, schema creation, SQL queries, views, constraints, sample data, and normalization—into a complete application that allowed real-time interaction with the database. Using PHP hosted on the TMU web server, we created a working interface capable of executing operations such as creating tables, populating data, running queries, and viewing system information related to customers, drivers, vehicles, merchants, service orders, locations, payments, and ratings.

System Overview

Our web application was built with PHP and connected to the Oracle database using the oci_connect() function. A structured menu provided clear navigation and allowed the TA to access each core component of the system. The available menu options included:

- Dropping all Ride & Pickup DBMS tables
- Creating all tables
- Populating the database with sample ride and delivery data

- Running advanced SQL queries
- Viewing each table individually (Customer, Driver, Vehicle, Merchant, Location, Service Order, Payment, Rating)
- Performing search, update, and delete operations through form inputs

The interface demonstrated that our database supported all required operations and that the design followed proper DBMS principles, including constraint enforcement, relationship handling, and query execution.

Website Link

The application was deployed on the TMU Oracle web server:

https://webdev.scs.ryerson.ca/~kpalakan/assignment9.php

Access to the UI required the TMU VPN for authentication, preventing unauthorized external access. Database credentials were embedded directly into the PHP file, ensuring that only authenticated TMU accounts could connect to Oracle and run the system.

Back-End Implementation

The backend PHP script included the following key components:

- A persistent Oracle connection via oci_connect()
- A reusable SQL execution function for drop, create, and populate scripts
- A dynamic query function capable of accepting bind variables
- A shared table-formatting function to neatly display query results
- Conditional routing using URL parameters (e.g., ?action=drop, ?action=viewCustomers)

This structure provided a consistent workflow and ensured that every UI action directly executed SQL statements on the Oracle server, illustrating how a web frontend interacts with a relational backend.

User Interface Demonstration

During the in-lab demonstration, we showcased the full workflow of the application. The submitted screenshots included:

- The main home menu
- Viewing entries for Customer, Driver, Vehicle, Merchant, Location, Service Order, Payment, and Rating
- Running advanced SQL queries such as multi-table joins and revenue calculations
- Performing search operations (e.g., find a customer by ID, locate a completed order)
- Updating attributes such as driver ratings or order status
- Deleting unwanted records
- Error-handling messages for invalid input or constraint violations

These outputs confirmed that the interface correctly connected to the Oracle database and accurately reflected the relationships and constraints defined in earlier assignments.

Advanced Queries Integration

All advanced queries created in Assignments 4 and 5 were integrated into the GUI. Examples included:

- Joins across customers, drivers, service orders, and payments
- Aggregation queries such as total platform revenue, average fare, or driver earnings
- Views created in Week 4 for order summaries and driver performance
- Queries involving merchant activity and delivery statistics

Running these queries through the UI demonstrated that the backend logic was functional, normalized, and capable of supporting complex reporting requirements for a ride-sharing and delivery system.

The code for this can be found below:

```
<?php
// CPS510 – Assignment 9 (Ride & Pickup DBMS)
// SINGLE PHP FILE — Works with SQL files in same folder

// ============================================
// 1. ORACLE DATABASE CONNECTION
// ============================================
```

```php
$conn = oci_connect(
    'kpalakan',
    '02102510',
    '(DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(Host=oracle.scs.ryerson.ca)(Port=1521))
    (CONNECT_DATA=(SID=orcl)))'
);

if (!$conn) {
    $e = oci_error();
    die("<h2>DB Connection Failed</h2><pre>".$e['message']."</pre>");
}

// ================================================
// 2. START SESSION + CSRF
// ================================================
ini_set('session.cookie_httponly', 1);
ini_set('session.use_strict_mode', 1);
if (session_status() === PHP_SESSION_NONE) session_start();

if (empty($_SESSION['csrf_token'])) {
    $_SESSION['csrf_token'] = bin2hex(random_bytes(24));
}
$CSRF = $_SESSION['csrf_token'];

// ================================================
// 3. HELPERS
// ================================================
function h($s) { return htmlspecialchars((string)$s, ENT_QUOTES | ENT_SUBSTITUTE, 'UTF-8'); }
function is_post() { return $_SERVER['REQUEST_METHOD'] === 'POST'; }
function check_csrf($token) {
    return isset($_SESSION['csrf_token']) && hash_equals($_SESSION['csrf_token'], $token);
}

// Whitelist our known table names here (adjust to match our schema)
$ALLOWED_TABLES = [
    'Address','Customer','Driver','Vehicle','Merchant','Location',
    'Service_Order','Payment','Rating_System'
];
```

```php
function validate_identifier($name, $allowed) {
    if (!in_array($name, $allowed, true)) return false;
    return preg_match('/^[A-Za-z0-9_]+$/', $name);
}

// Basic client-side friendly validation helper (server-side fallback)
function validate_input_value($val) {
    // trim and limit length to avoid extremely long inputs
    $v = trim((string)$val);
    if (strlen($v) > 2000) return false;
    return $v;
}

// ================================================
// 4. SQL FUNCTIONS (safer handling)
// ================================================
function runSqlFile($conn, $filename) {
    if (!file_exists($filename)) {
        echo "<div class='error'>ERROR: File not found: " . h($filename) . "</div>";
        return;
    }

    $sql = file_get_contents($filename);
    // split statements on semicolon + newline (keeps simple PL/SQL mostly intact)
    $statements = preg_split('/;[\\s]*\r?\n/', $sql);
    foreach ($statements as $stmt) {
        $trim = trim($stmt);
        if ($trim === "") continue;
        $stid = oci_parse($conn, $trim);
        if (!@oci_execute($stid)) {
            $err = oci_error($stid);
            echo "<div class='error'>SQL execution error: " . h($err['message']) . "</div>";
        }
    }
}

function runQuery($conn, $query, $bindings = []) {
    $stid = oci_parse($conn, $query);
```

```
    // Bind variables safely (oci_bind_by_name binds by reference)
    $refs = [];
    $i = 0;
    foreach ($bindings as $name => $val) {
        $i++;
        $var = "bv_" . $i;
        $$var = $val;
        $bindName = (strpos($name, ':') === 0) ? $name : ':' . $name;
        oci_bind_by_name($stid, $bindName, $$var);
        $refs[] = &$$var; // hold references to avoid garbage collection
    }

    if (!@oci_execute($stid)) {
        $err = oci_error($stid);
        throw new RuntimeException("Query error: " . $err['message']);
    }
    return $stid;
}

function printTable($stid) {
    echo "<div class='table-wrap'><table class='result'>";
    echo "<thead><tr>";
    $ncols = oci_num_fields($stid);
    for ($i = 1; $i <= $ncols; $i++) {
        echo "<th>" . h(oci_field_name($stid, $i)) . "</th>";
    }
    echo "</tr></thead><tbody>";
    while ($row = oci_fetch_array($stid, OCI_ASSOC+OCI_RETURN_NULLS)) {
        echo "<tr>";
        foreach ($row as $col) {
            echo "<td>" . h($col) . "</td>";
        }
        echo "</tr>";
    }
    echo "</tbody></table></div>";
}

// ============================================
// 5. HANDLE ACTIONS (view, search, update, delete, run SQL files, run queries)
```

```php
// ================================================
$errors = [];
$messages = [];

if (isset($_GET['action'])) {
    $action = $_GET['action'];

    if (in_array($action, ['drop','create','populate','queries'], true)) {
        // these run SQL files or queries.sql
        $token_ok = isset($_GET['token']) && hash_equals($_SESSION['csrf_token'], $_GET['token']);
        if (!$token_ok) {
            $errors[] = "Missing or invalid token for action: " . h($action);
        } else {
            try {
                if ($action === "drop") {
                    runSqlFile($conn, "drop_tables.sql");
                    $messages[] = "Tables dropped successfully.";
                } elseif ($action === "create") {
                    runSqlFile($conn, "create_tables.sql");
                    $messages[] = "Tables created successfully.";
                } elseif ($action === "populate") {
                    runSqlFile($conn, "populate_tables.sql");
                    $messages[] = "Tables populated successfully.";
                } elseif ($action === "queries") {
                    $messages[] = "Executing queries from queries.sql";
                    $content = file_get_contents("queries.sql");
                    $queries = preg_split('/;[\\s]*\r?\n/', $content);
                    foreach ($queries as $q) {
                        $t = trim($q);
                        if ($t === "") continue;
                        $messages[] = $t;
                        try {
                            $stid = runQuery($conn, $t);
                            ob_start();
                            printTable($stid);
                            echo ob_get_clean();
                        } catch (Exception $e) {
                            $errors[] = $e->getMessage();
                        }
```

```
                }
            }
        } catch (Exception $e) {
            $errors[] = $e->getMessage();
        }
    }
}

// view table (GET)
if ($action === 'view' && isset($_GET['table'])) {
    $table = $_GET['table'];
    if (!validate_identifier($table, $ALLOWED_TABLES)) {
        $errors[] = "Invalid table name.";
    } else {
        try {
            $stid = runQuery($conn, "SELECT * FROM " . $table);
            $messages[] = "Viewing table: " . h($table);
            ob_start();
            printTable($stid);
            echo ob_get_clean();
        } catch (Exception $e) {
            $errors[] = $e->getMessage();
        }
    }
}
}

// POST actions: search, update, delete
if (is_post()) {
    $act = $_POST['action'] ?? '';
    $token = $_POST['csrf_token'] ?? '';
    if (!check_csrf($token)) {
        $errors[] = "Invalid CSRF token.";
    } else {
        if ($act === 'search') {
            $t = trim($_POST['table'] ?? '');
            $field = trim($_POST['field'] ?? '');
            $value = $_POST['value'] ?? '';
```

```
    if (!validate_identifier($t, $ALLOWED_TABLES) || !preg_match('/^[A-Za-z0-9_]+$/', $field)) {
        $errors[] = "Invalid table or field name.";
    } else {
        $val = validate_input_value($value);
        if ($val === false) { $errors[] = "Invalid search value."; }
        else {
            try {
                $sql = "SELECT * FROM $t WHERE $field = :val";
                $stid = runQuery($conn, $sql, [':val' => $val]);
                $messages[] = "Search results in " . h($t) . " for " . h($field) . " = " . h($val);
                ob_start(); printTable($stid); echo ob_get_clean();
            } catch (Exception $e) {
                $errors[] = $e->getMessage();
            }
        }
    }
}

if ($act === 'update') {
    $t = trim($_POST['table'] ?? '');
    $field = trim($_POST['field'] ?? '');
    $value = $_POST['value'] ?? '';
    $pk = trim($_POST['pk'] ?? '');
    $pkval = $_POST['pkval'] ?? '';

    if (!validate_identifier($t, $ALLOWED_TABLES) ||
        !preg_match('/^[A-Za-z0-9_]+$/', $field) ||
        !preg_match('/^[A-Za-z0-9_]+$/', $pk)) {
        $errors[] = "Invalid identifiers.";
    } else {
        $val = validate_input_value($value);
        $pkv = validate_input_value($pkval);
        if ($val === false || $pkv === false) $errors[] = "Invalid values.";
        else {
            try {
                $sql = "UPDATE $t SET $field = :val WHERE $pk = :pkval";
                runQuery($conn, $sql, [':val' => $val, ':pkval' => $pkv]);
                $messages[] = "Record updated in " . h($t);
            } catch (Exception $e) {
```

```
                    $errors[] = $e->getMessage();
                }
            }
        }
    }

    if ($act === 'delete') {
        $t = trim($_POST['table'] ?? '');
        $pk = trim($_POST['pk'] ?? '');
        $val = $_POST['val'] ?? '';

        if (!validate_identifier($t, $ALLOWED_TABLES) || !preg_match('/^[A-Za-z0-9_]+$/', $pk)) {
            $errors[] = "Invalid table or primary key name.";
        } else {
            $v = validate_input_value($val);
            if ($v === false) $errors[] = "Invalid value.";
            else {
                try {
                    $sql = "DELETE FROM $t WHERE $pk = :val";
                    runQuery($conn, $sql, [':val' => $v]);
                    $messages[] = "Record deleted from " . h($t);
                } catch (Exception $e) {
                    $errors[] = $e->getMessage();
                }
            }
        }
    }
  }
}
?>

<!doctype html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Ride & Pickup DBMS – CPS510</title>
<meta name="viewport" content="width=device-width,initial-scale=1">
<style>
:root{
```

```
  --bg:#f5f8ff; --card:#ffffff; --accent:#3b82f6; --accent-2:#06b6d4; --muted:#53627a;
  --success:#16a34a; --danger:#ef4444;
}
*{box-sizing:border-box}
body{margin:0;font-family:Inter,Arial,Helvetica,sans-serif;background:linear-gradient(135deg,#e6eefc
0%,#f7fbff 100%);color:#0f172a;padding:22px}
.container{max-width:1100px;margin:0 auto}
.header{display:flex;justify-content:space-between;align-items:center;padding:14px
18px;border-radius:10px;background:linear-gradient(180deg,#ffffff,#f1f7ff);box-shadow:0 6px 18px
rgba(15,23,42,0.06)}
.title{font-size:20px;color:var(--accent);margin:0}
.subtitle{font-size:13px;color:var(--muted)}
.top-actions a{background:linear-gradient(90deg,var(--accent),var(--accent-2));color:#fff;padding:8px
12px;border-radius:8px;text-decoration:none;font-weight:600;margin-left:8px}
.layout{display:flex;gap:18px;margin-top:18px;align-items:flex-start}
.left{width:340px;flex-shrink:0}
.card{background:var(--card);padding:14px;border-radius:12px;box-shadow:0 6px 18px rgba(15,23,42,0.06)}
.btn{display:block;text-decoration:none;color:#fff;background:linear-gradient(90deg,var(--accent),var(--accent
-2));padding:10px;border-radius:8px;text-align:center;margin:8px 0;font-weight:700}
.small-btn{display:inline-block;padding:8px
10px;border-radius:8px;background:#eef7ff;color:var(--accent);text-decoration:none;border:1px solid
rgba(59,130,246,0.12);margin:6px 4px}
.form-input{width:100%;padding:8px;border-radius:8px;border:1px solid #e6eef6;margin-top:6px}
.label{font-size:13px;color:var(--muted);margin-top:10px;display:block}
.result{width:100%;border-collapse:collapse;margin-top:12px}
.result th, .result td{padding:8px;border:1px solid #f0f4f8;text-align:left;font-size:14px}
.note{font-size:13px;color:var(--muted);margin-top:8px}
.msg{padding:10px;border-radius:8px;margin-bottom:10px;border-left:4px solid
var(--success);background:#f0fdf4;color:var(--success)}
.err{padding:10px;border-radius:8px;margin-bottom:10px;border-left:4px solid
var(--danger);background:#fff5f5;color:var(--danger)}
.footer{margin-top:18px;text-align:center;color:var(--muted);font-size:13px}
@media(max-width:980px){.layout{flex-direction:column}.left{width:100%}}
</style>
</head>
<body>
<div class="container">
  <div class="header">
    <div>
```

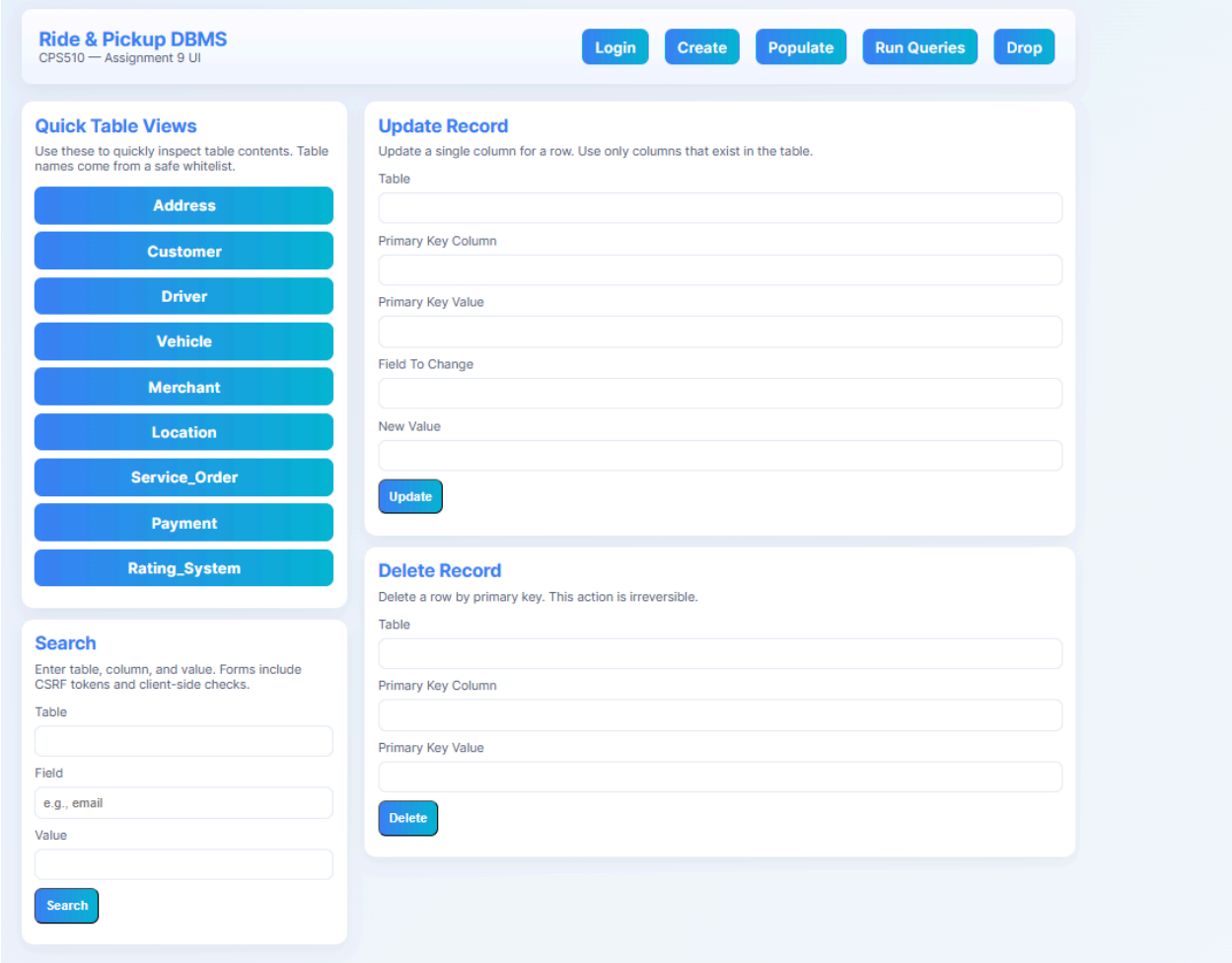
```
    <h1 class="title">Ride & Pickup DBMS</h1>
    <div class="subtitle">CPS510 — Assignment 9 UI</div>
   </div>
   <div class="top-actions">
    <a href="login.php" class="small-btn">Login</a>
    <!-- CSRF-protected links for sensitive SQL file actions (token appended) -->
    <a href="?action=create&token=<?= h($CSRF) ?>" class="small-btn">Create</a>
    <a href="?action=populate&token=<?= h($CSRF) ?>" class="small-btn">Populate</a>
    <a href="?action=queries&token=<?= h($CSRF) ?>" class="small-btn">Run Queries</a>
    <a href="?action=drop&token=<?= h($CSRF) ?>" class="small-btn" onclick="return confirm('Drop
tables? This is irreversible.');">Drop</a>
   </div>
  </div>

  <div class="layout">
   <div class="left">
    <div class="card">
     <h3 style="margin:0;color:var(--accent)">Quick Table Views</h3>
     <p class="note">Use these to quickly inspect table contents. Table names come from a safe whitelist.</p>
     <?php foreach ($ALLOWED_TABLES as $tbl): ?>
      <a class="btn" href="?action=view&table=<?= h($tbl) ?>"><?= h($tbl) ?></a>
     <?php endforeach; ?>
    </div>

    <div class="card" style="margin-top:12px">
     <h3 style="margin:0;color:var(--accent)">Search</h3>
     <p class="note">Enter table, column, and value. Forms include CSRF tokens and client-side checks.</p>
     <form method="post" onsubmit="return validateSearch(this);" style="margin-top:8px">
      <input type="hidden" name="action" value="search">
      <input type="hidden" name="csrf_token" value="<?= h($CSRF) ?>">
      <label class="label">Table</label>
      <input name="table" class="form-input" list="tables" required>
      <datalist id="tables">
       <?php foreach ($ALLOWED_TABLES as $t): ?><option value="<?= h($t) ?>"></option><?php
endforeach; ?>
      </datalist>
      <label class="label">Field</label>
      <input name="field" class="form-input" placeholder="e.g., email" required>
      <label class="label">Value</label>
```

```
      <input name="value" class="form-input" required>
      <button class="btn" type="submit">Search</button>
    </form>
  </div>

</div>

<div class="right" style="flex:1">
  <?php foreach ($messages as $m): ?><div class="msg"><?= h($m) ?></div><?php endforeach; ?>
  <?php foreach ($errors as $e): ?><div class="err"><?= h($e) ?></div><?php endforeach; ?>

  <div class="card">
    <h3 style="margin:0;color:var(--accent)">Update Record</h3>
    <p class="note">Update a single column for a row. Use only columns that exist in the table.</p>
    <form method="post" onsubmit="return validateUpdate(this);" style="margin-top:8px">
      <input type="hidden" name="action" value="update">
      <input type="hidden" name="csrf_token" value="<?= h($CSRF) ?>">
      <label class="label">Table</label><input name="table" class="form-input" list="tables" required>
      <label class="label">Primary Key Column</label><input name="pk" class="form-input" required>
      <label class="label">Primary Key Value</label><input name="pkval" class="form-input" required>
      <label class="label">Field To Change</label><input name="field" class="form-input" required>
      <label class="label">New Value</label><input name="value" class="form-input" required>
      <button class="btn" type="submit">Update</button>
    </form>
  </div>

  <div class="card" style="margin-top:12px">
    <h3 style="margin:0;color:var(--accent)">Delete Record</h3>
    <p class="note">Delete a row by primary key. This action is irreversible.</p>
    <form method="post" onsubmit="return confirm('Delete record? This cannot be undone.') &&
validateDelete(this);" style="margin-top:8px">
      <input type="hidden" name="action" value="delete">
      <input type="hidden" name="csrf_token" value="<?= h($CSRF) ?>">
      <label class="label">Table</label><input name="table" class="form-input" list="tables" required>
      <label class="label">Primary Key Column</label><input name="pk" class="form-input" required>
      <label class="label">Primary Key Value</label><input name="val" class="form-input" required>
      <button class="btn" type="submit">Delete</button>
    </form>
  </div>
```

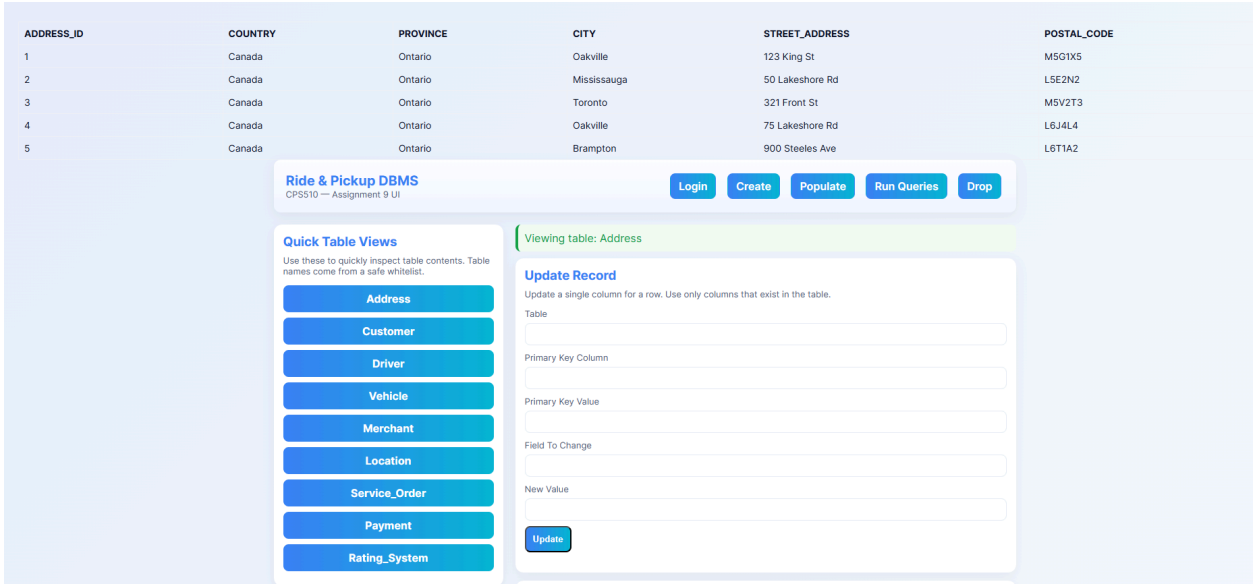
```
    <!-- If any table output was printed above (from actions), it will show here because printTable echoes
HTML -->
   </div>
  </div>

  <div class="footer">
   <div>a</div>
  </div>
</div>

<script>
// Client-side validations for UX only (server checks remain authoritative)
function identOK(s) { return /^[A-Za-z0-9_]+$/.test(s); }

function validateSearch(f) {
  const table = f.table.value.trim(), field = f.field.value.trim();
  if (!identOK(table)) { alert('Invalid table name. Use alphanumeric and underscores only.'); return false; }
  if (!identOK(field)) { alert('Invalid field name. Use alphanumeric and underscores only.'); return false; }
  if (f.value.value.length > 2000) { alert('Value too long'); return false; }
  return true;
}

function validateUpdate(f) {
  if (!identOK(f.table.value.trim()) || !identOK(f.pk.value.trim()) || !identOK(f.field.value.trim())) {
    alert('Table, PK column and field must be alphanumeric/underscore only.'); return false;
  }
  return true;
}

function validateDelete(f) {
  if (!identOK(f.table.value.trim()) || !identOK(f.pk.value.trim())) {
    alert('Invalid identifiers.'); return false;
  }
  return true;
}
</script>
</body>
</html>
```

Screenshots:

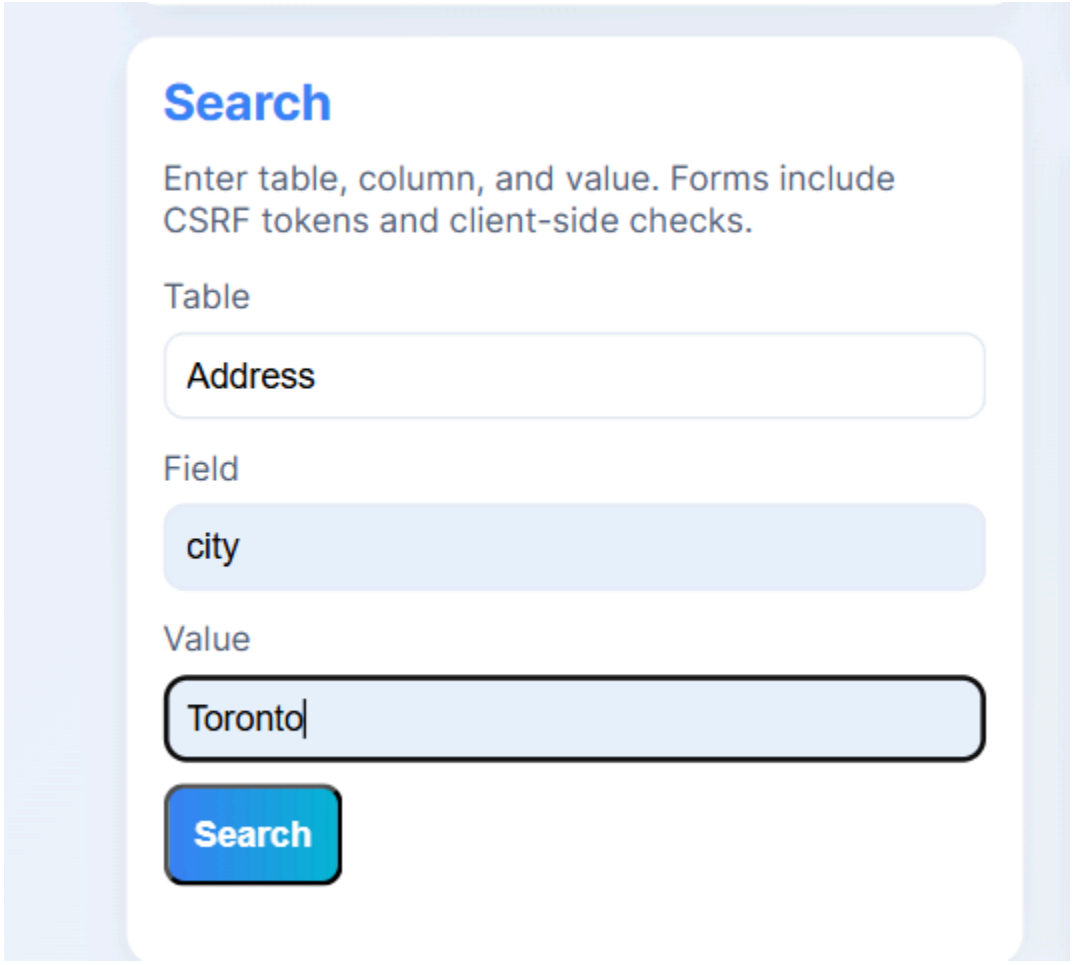
Home Screen



Viewing Table:



Search function:



ADDRESS_ID	COUNTRY	PROVINCE	CITY	STREET_ADDRESS	POSTAL_CODE
1	Canada	Ontario	Oakville	123 King St	M5G1X5
2	Canada	Ontario	Mississauga	50 Lakeshore Rd	L5E2N2
3	Canada	Ontario	Toronto	321 Front St	M5V2T3
4	Canada	Ontario	Oakville	75 Lakeshore Rd	L6J4L4
5	Canada	Ontario	Brampton	900 Steeles Ave	L6T1A2

ADDRESS_ID	COUNTRY	PROVINCE	CITY	STREET_ADDRESS	POSTAL_CODE
3	Canada	Ontario	Toronto	321 Front St	M5V2T3

Ride & Pickup DBMS
CPS510 — Assignment 9 UI

Login | Create | Populate | Run Queries | Drop

Update function:

Update Record

Update a single column for a row. Use only columns that exist in the table.

Table

Address

Primary Key Column

ADDRESS_ID

Primary Key Value

1

Field To Change

city

New Value

Brampton

Update

ADDRESS_ID	COUNTRY	PROVINCE	CITY	STREET_ADDRESS	POSTAL_CODE
1	Canada	Ontario	Brampton	123 King St	M5G1X5
2	Canada	Ontario	Mississauga	50 Lakeshore Rd	L5E2N2
3	Canada	Ontario	Toronto	321 Front St	M5V2T3
4	Canada	Ontario	Oakville	75 Lakeshore Rd	L6J4L4
5	Canada	Ontario	Brampton	900 Steeles Ave	L6T1A2

Ride & Pickup DBMS
CPS510 — Assignment 9 UI

Login | Create | Populate | Run Queries | Drop

Quick Table Views

Viewing table: Address

Example of error handling:

```
Query error: ORA-00904: "CUSTOMER_ID": invalid identifier
```

```
Query error: ORA-02292: integrity constraint (KPALAKAN.SYS_C002469624) violated - child record found
```

```
Query error: ORA-00942: table or view does not exist
```

Advanced Queries

```
Executing queries from queries.sql
```

```
---------------------------------------------------------------- -- CPS510 – Assignment 4 (Part 1) -- Ride & Pickup DBMS – Queries -- FIXED FOR ASSIGNMENT 9 (Payment column names updated) ---------------------------------------------------------------- -- Q1. Toronto addresses SELECT Address_ID, Street_Address AS Street, City, Province, Postal_Code FROM Address WHERE City = 'Toronto'
```

```
-- Q2. Customer balances with active/inactive status SELECT Customer_ID, First_Name || ' ' || Last_Name AS Customer_Name, Balance, CASE WHEN Balance > 0 THEN 'Active' ELSE 'Inactive' END AS Status FROM Customer ORDER BY Balance DESC
```

```
-- Q3. Drivers with insurance info SELECT Driver_ID, First_Name || ' ' || Last_Name AS Driver_Name, License_Info, Insurance_Info FROM Driver WHERE Insurance_Info IS NOT NULL
```

```
-- Q4. Vehicles manufactured after 2015 SELECT Vehicle_ID, License_Plate, Make, Model, Year, Vehicle_Type FROM Vehicle WHERE Year > 2015
```

```
-- Q5. Merchants and their cities SELECT DISTINCT M.Merchant_ID, M.Name AS Merchant_Name, A.City, A.Province FROM Merchant M, Address A WHERE M.Address_ID = A.Address_ID
```

```
-- Q6. Ontario locations SELECT Location_ID, Street_Address, City, Province, Postal_Code FROM Location WHERE Province = 'Ontario'
```

```
-- Q7. Completed service orders > $20 SELECT Order_ID, Customer_ID, Driver_ID, Fare, Status FROM Service_Order WHERE Status = 'Completed' AND Fare > 20
```

```
-- Q8. Count credit payments (FIXED) SELECT COUNT(*) AS Credit_Payments FROM Payment WHERE Method = 'Credit'
```

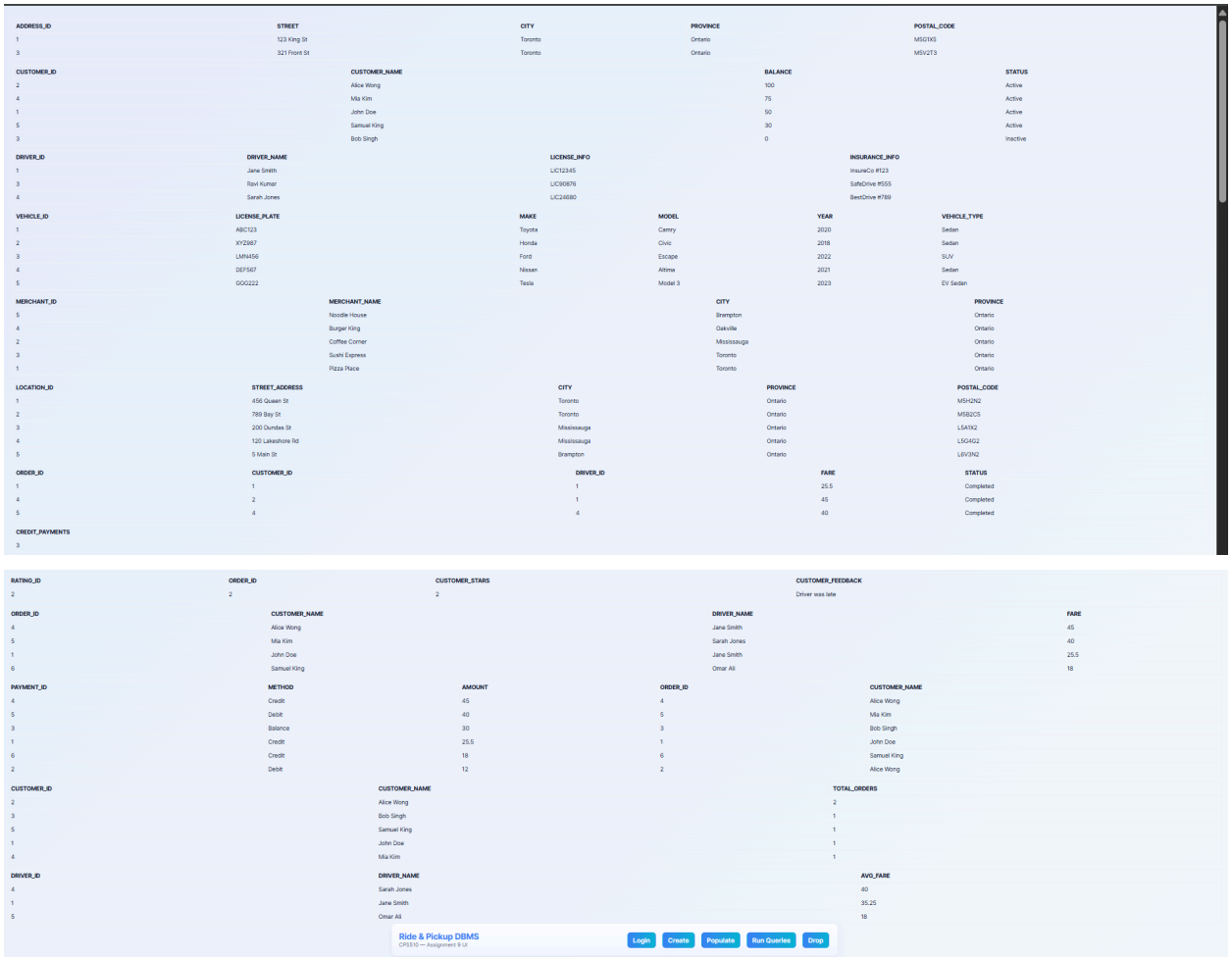
```
-- Q9. Low ratings (< 3 stars) SELECT Rating_ID, Order_ID, Customer_Stars, Customer_Feedback FROM Rating_System WHERE Customer_Stars < 3
```

```
-- Q10. Completed orders with customer & driver SELECT so.Order_ID, c.First_Name || ' ' || c.Last_Name AS Customer_Name, d.First_Name || ' ' || d.Last_Name AS Driver_Name, so.Fare FROM Service_Order so, Customer c, Driver d WHERE so.Customer_ID = c.Customer_ID AND so.Driver_ID = d.Driver_ID AND so.Status = 'Completed' ORDER BY so.Fare DESC
```

```
-- Q11. Payment + Order + Customer (FIXED) SELECT p.Payment_ID, p.Method, p.Amount, so.Order_ID, c.First_Name || ' ' || c.Last_Name AS Customer_Name FROM Payment p, Service_Order so, Customer c WHERE p.Order_ID = so.Order_ID AND so.Customer_ID = c.Customer_ID ORDER BY p.Amount DESC
```

```
-- Q12. Total orders per customer SELECT c.Customer_ID, c.First_Name || ' ' || c.Last_Name AS Customer_Name, COUNT(so.Order_ID) AS Total_Orders FROM Customer c LEFT JOIN Service_Order so ON c.Customer_ID = so.Customer_ID GROUP BY c.Customer_ID, c.First_Name || ' ' || c.Last_Name ORDER BY Total_Orders DESC
```

```
-- Q13. Average fare per driver SELECT d.Driver_ID, d.First_Name || ' ' || d.Last_Name AS Driver_Name, ROUND(AVG(so.Fare),2) AS Avg_Fare FROM Driver d JOIN Service_Order so ON d.Driver_ID = so.Driver_ID WHERE so.Status = 'Completed' GROUP BY d.Driver_ID, d.First_Name || ' ' || d.Last_Name ORDER BY Avg_Fare DESC
```



ADDRESS_ID | STREET | CITY | PROVINCE | POSTAL_CODE
1 | 123 King St | Toronto | Ontario | M5G1X5
3 | 321 Front St | Toronto | Ontario | M5V2T3

CUSTOMER_ID | CUSTOMER_NAME | BALANCE | STATUS
2 | Alice Wong | 100 | Active
4 | Mia Kim | 75 | Active
1 | John Doe | 50 | Active
5 | Samuel King | 30 | Active
3 | Bob Singh | 0 | Inactive

DRIVER_ID | DRIVER_NAME | LICENSE_INFO | INSURANCE_INFO
1 | Jane Smith | LIC12345 | InsureCo #123
3 | Ravi Kumar | LIC90876 | SafeDrive #555
4 | Sarah Jones | LIC24680 | BestDrive #789

VEHICLE_ID | LICENSE_PLATE | MAKE | MODEL | YEAR | VEHICLE_TYPE
1 | ABC123 | Toyota | Camry | 2020 | Sedan
2 | XYZ987 | Honda | Civic | 2018 | Sedan
3 | LMN456 | Ford | Escape | 2022 | SUV
4 | DEF567 | Nissan | Altima | 2021 | Sedan
5 | GGG222 | Tesla | Model 3 | 2023 | EV Sedan

MERCHANT_ID | MERCHANT_NAME | CITY | PROVINCE
5 | Noodle House | Brampton | Ontario
4 | Burger King | Oakville | Ontario
2 | Coffee Corner | Mississauga | Ontario
3 | Sushi Express | Toronto | Ontario
1 | Pizza Place | Toronto | Ontario

LOCATION_ID | STREET_ADDRESS | CITY | PROVINCE | POSTAL_CODE
1 | 456 Queen St | Toronto | Ontario | M5H2N2
2 | 789 Bay St | Toronto | Ontario | M5B2C5
3 | 200 Dundas St | Mississauga | Ontario | L5A1X2
4 | 120 Lakeshore Rd | Mississauga | Ontario | L5G4G2
5 | 5 Main St | Brampton | Ontario | L6V3N2

ORDER_ID | CUSTOMER_ID | DRIVER_ID | FARE | STATUS
1 | 1 | 1 | 25.5 | Completed
4 | 2 | 1 | 45 | Completed
5 | 4 | 4 | 40 | Completed

CREDIT_PAYMENTS
3

RATING_ID | ORDER_ID | CUSTOMER_STARS | CUSTOMER_FEEDBACK
2 | 2 | 2 | Driver was late

ORDER_ID | CUSTOMER_NAME | DRIVER_NAME | FARE
4 | Alice Wong | Jane Smith | 45
5 | Mia Kim | Sarah Jones | 40
1 | John Doe | Jane Smith | 25.5
6 | Samuel King | Omar Ali | 18

PAYMENT_ID | METHOD | AMOUNT | ORDER_ID | CUSTOMER_NAME
4 | Credit | 45 | 4 | Alice Wong
5 | Debit | 40 | 5 | Mia Kim
3 | Balance | 30 | 3 | Bob Singh
1 | Credit | 25.5 | 1 | John Doe
6 | Credit | 18 | 6 | Samuel King
2 | Debit | 12 | 2 | Alice Wong

CUSTOMER_ID | CUSTOMER_NAME | TOTAL_ORDERS
2 | Alice Wong | 2
3 | Bob Singh | 1
5 | Samuel King | 1
1 | John Doe | 1
4 | Mia Kim | 1

DRIVER_ID | DRIVER_NAME | AVG_FARE
4 | Sarah Jones | 40
1 | Jane Smith | 35.25
5 | Omar Ali | 18

Ride & Pickup DBMS
CPS510 — Assignment 9 UI
Login | Create | Populate | Run Queries | Drop

Week 10 – Final Documentation & Relational Algebra

In Week 10, we finalized the documentation for our Ride & Pickup Database Management System by compiling all work completed throughout Weeks 1–9 into a single cohesive report. This included revisiting our ER diagram, schema creation, SQL scripts, functional dependencies, normalization (3NF and BCNF verification), Unix command-line queries, and the complete web-based PHP interface. Any sections that evolved during implementation were updated to ensure consistency with the final version of our database.

A core requirement of this week was to provide Relational Algebra (RA) notation for the SQL queries demonstrated in Assignments 4, 5, and 9. Since the SQL components were already complete, this step involved translating each query into its corresponding RA expression using operators such as selection (σ), projection (Π), join (⋈), grouping/aggregation (F), union (∪), and intersection (∩). These RA statements show the theoretical foundation of each query and demonstrate how our DBMS operations can be expressed mathematically.
Below is the list of RA expressions for our Ride & Pickup queries:

Relational Algebra Notation for Ride & Pickup DBMS Queries

1.

ρ(Addresss_ID, Street, City, Province, Postal_Code)Π(Address_ID, Street_Address, City, Province, Postal_Code)(σ(city='Toronto')(Customer))

2.

Π(Customer_ID, First_Name, Last_Name, Balance)(Customer)

3.

Π(Driver_ID, First_Name, Last_Name, License_Info, Insurance_Info)(σ(Insurance_Info IS NOT NULL)(Driver))

4.

Π(Vehicle_ID, License_Plate, Make, Model, Year, Vehicle_Type)(σ(Year>2015)(Vehicle))

5.

Π(M.Merchant_ID, M.Name, A.City, A.Province)(Merchant ⋈ Address)

6.

Π(Location_ID, Street_Address, City, Province, Postal_Code)(σ(Province='Ontario')(Location))

7.

Π(Order_ID, Customer_ID, Driver_ID, Fare, Status)(σ(so.Status='Completed' AND Fare>20)(Service_Order))

8.

F(Count *)(σ(Method='Credit')(Payment))

9.

Π(Rating_ID, Order_ID, Customer_Stars, Customer_Feedback)(σ(Customer_Stars<3)(Rating_System))

10.

Π(c.First_Name, c.Last_Name, d.First_Name, d.Last_Name, so.Order_ID)(σ(so.Status='Completed')(Customer ⋈ Driver ⋈ Service_Order))

11.

Π(p.Payment_ID, p.Method, p.Amount, so.Order_ID, c.First_Name, c.Last_Name)(σ(so.Status='Completed')(Payment ⋈ Customer ⋈ Service_Order))

12.

Π(Merchant_ID,Name)(Merchant)−Π(Merchant_ID,Name)(Merchant⋉Service_Order)

13.

Π(d.Driver_ID, d.First_Name, d.Last_Name)F(Round(AVERAGE so.Fare))(σ(so.Status='Completed')(Driver ⋈ Service_Order))

Conclusion

In conclusion, the Ride & Pickup DBMS project successfully delivered a fully functional system capable of managing customers, drivers, vehicles, merchants, service orders, payments, and ratings in an integrated platform. Throughout the development process, we applied core database concepts such as ER modeling, schema creation, functional dependency analysis, normalization to 3NF and BCNF, SQL query design, Unix-based automation, and PHP–Oracle integration. The resulting system demonstrates strong data accuracy, referential integrity, and reliable query performance, reflecting real-world challenges found in ride-sharing and delivery applications. Overall, this project provided valuable experience in designing, implementing, and deploying a complete database application from start to finish.