

---

# Citywheels Ride Booking Database

---

## TABLE OF CONTENTS

<b>1. Introduction</b>	<b>2</b>
<b>2. Executive Summary</b>	<b>4</b>
<b>3. Database Design</b>	<b>5</b>
<b>4. Sample Data</b>	<b>8</b>
<b>5. Business Queries</b>	<b>10</b>
<b>6. Conclusion</b>	<b>20</b>

# 1. INTRODUCTION

## 1.1 Project Background

CityWheels is a burgeoning ride-hailing service operating in Pakistan, with ambitious plans for global expansion into GCC countries and beyond. In today's digital economy, data-driven decision-making and efficient management of transportation services require a robust, scalable, and secure database infrastructure. This documentation presents the complete database management system designed specifically for CityWheels's operational needs.

## 1.2 Purpose and Scope

The CityWheels Ride-Hailing Database System serves as the technological backbone for managing all aspects of the ride-hailing platform, including:

- Driver and passenger profile management
- Ride booking and history tracking
- Payment processing and financial records
- Customer ratings and feedback systems
- Promotional offer management
- Customer support ticketing

## 1.3 Objectives

This project aims to achieve the following objectives:

- Design a normalized, efficient database structure adhering to 3NF principles
- Implement complex business queries for data analysis and decision-making
- Ensure data integrity through proper constraints and relationships
- Address legal and ethical considerations for cross-border data sharing
- Provide a scalable architecture capable of supporting future growth

## 1.4 Document Structure

This documentation is organized into the following sections:

- Database Design - ER diagrams and relational schemas
- Schema Documentation - Detailed table structures
- Sample Data - Data population and statistics
- Business Queries - Analytical SQL implementations
- Legal Research - Cross-border data compliance analysis
- Implementation Guide - Installation and usage instructions

## 1.5 Target Audience






This documentation is intended for:

- Database administrators managing the CityWheels platform
  - Developers integrating applications with the database
  - Business analysts extracting insights for decision-making
  - Legal and compliance teams reviewing data protection measures
  - Academic evaluators assessing the database design
-

## 2. EXECUTIVE SUMMARY

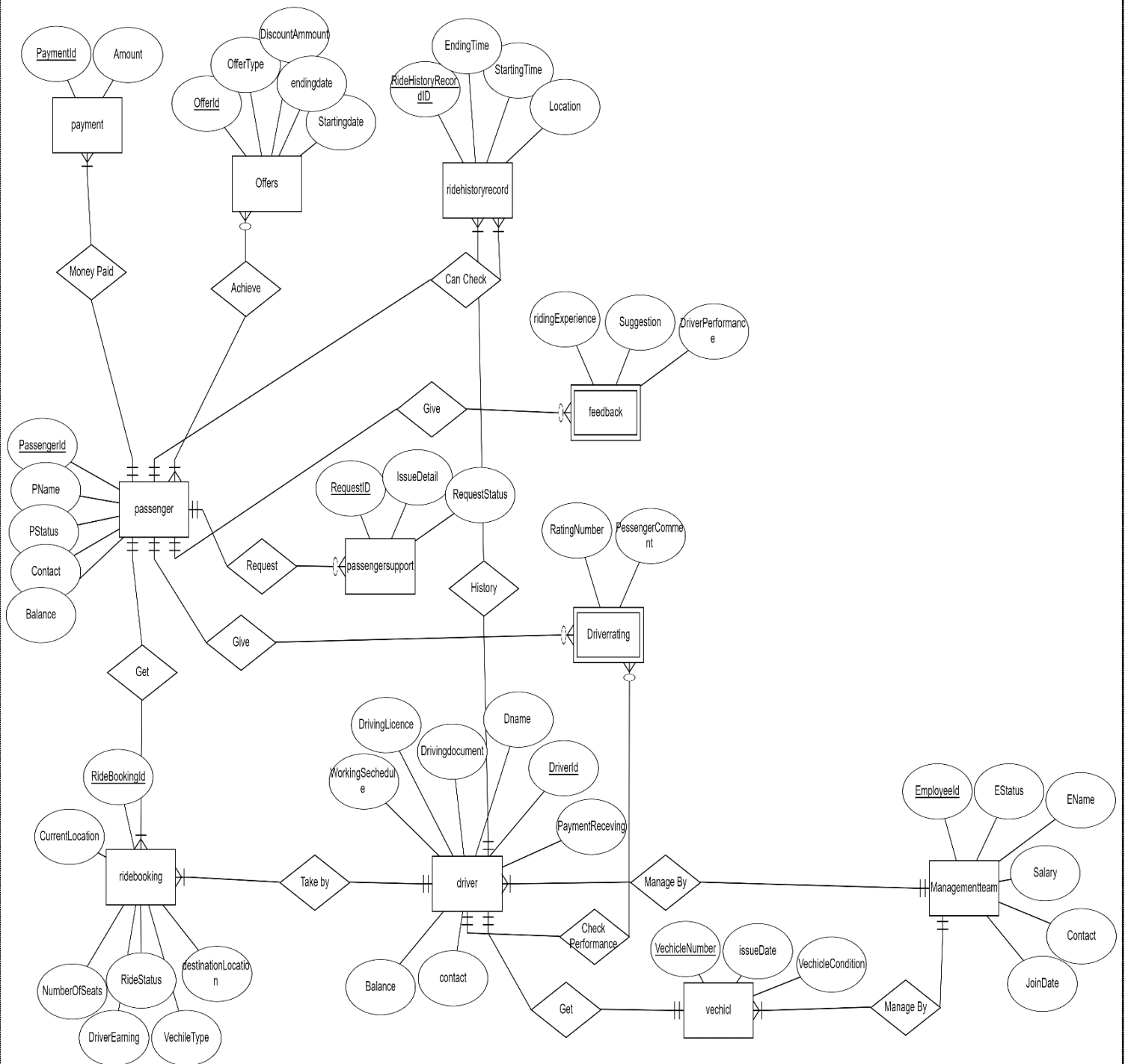
The CityWheels Ride-Hailing Database System is a comprehensive MySQL-based solution designed to manage all operational aspects of a modern ride-hailing service. Developed as part of the Database Systems course, this project demonstrates professional database design principles, complex query implementation, and real-world business logic application.

### Key Achievements:

-  12 normalized tables with proper relationships
  -  500+ sample records across all entities
  -  10 complex analytical business queries
  -  Complete ERD and relational schema diagrams
  -  Legal research on cross-border data compliance (PDPL & GDPR)
-

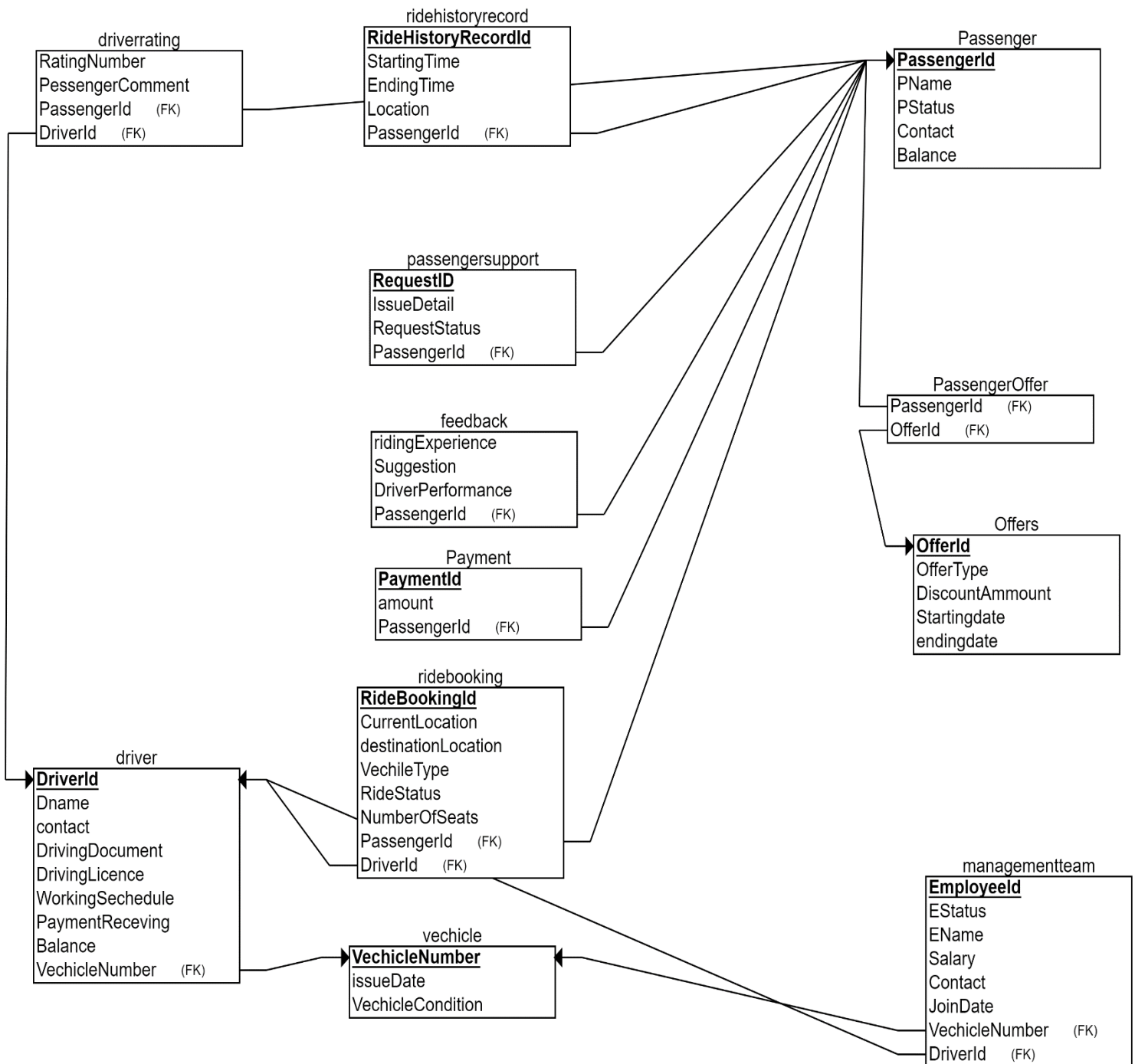
### 3. DATABASE DESIGN

#### 3.1 Entity Relationship Diagram (ERD)



**Figure 1:** Physical ERD showing all tables, attributes, and relationships

## 3.2 Relational Schema



**Figure 2:** Relational schema with primary and foreign key relationships

### 3.3 Design Methodology

The database follows these design principles:

- **Normalization:** 3NF compliance to eliminate redundancy
- **Referential Integrity:** Foreign key constraints across all relationships
- **Data Types:** Appropriate data types for efficient storage
- **Naming Conventions:** Consistent PascalCase for tables and columns

### 3.4 Relationships Overview

Relationship Type	Tables Involved	Description
One-to-Many	Driver → RideBooking	One driver can have many ride bookings
One-to-Many	Passenger → RideBooking	One passenger can have many ride bookings
One-to-Many	Driver → DriverRating	One driver can receive many ratings
One-to-Many	Passenger → PassengerSupport	One passenger can create many support tickets
Many-to-Many	Passenger ↔ Offers	Passengers can avail multiple offers, offers can be availed by multiple passengers (resolved through PassengerOffer junction table)

## 4. SAMPLE DATA

### 4.1 Data Population Statistics

Table	Record Count
Vehicle	50
Driver	50
Passenger	50
DriverRating	50
ManagementTeam	50
RideBooking	54
Feedback	50
RideHistoryRecord	50
PassengerSupport	50
Offers	50
PassengerOffer	50
Payment	50

**Total Records:** 600+ records

## 4.2 Sample Data Screenshots

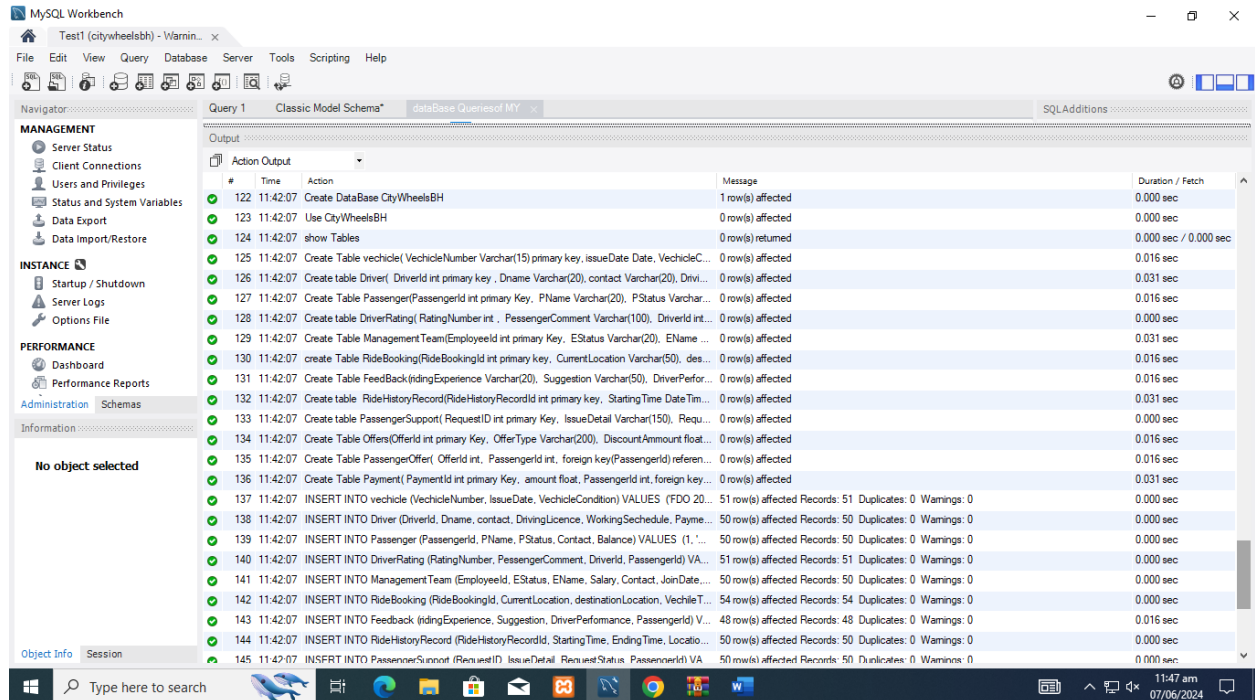


Figure 3: data insertion screenshots

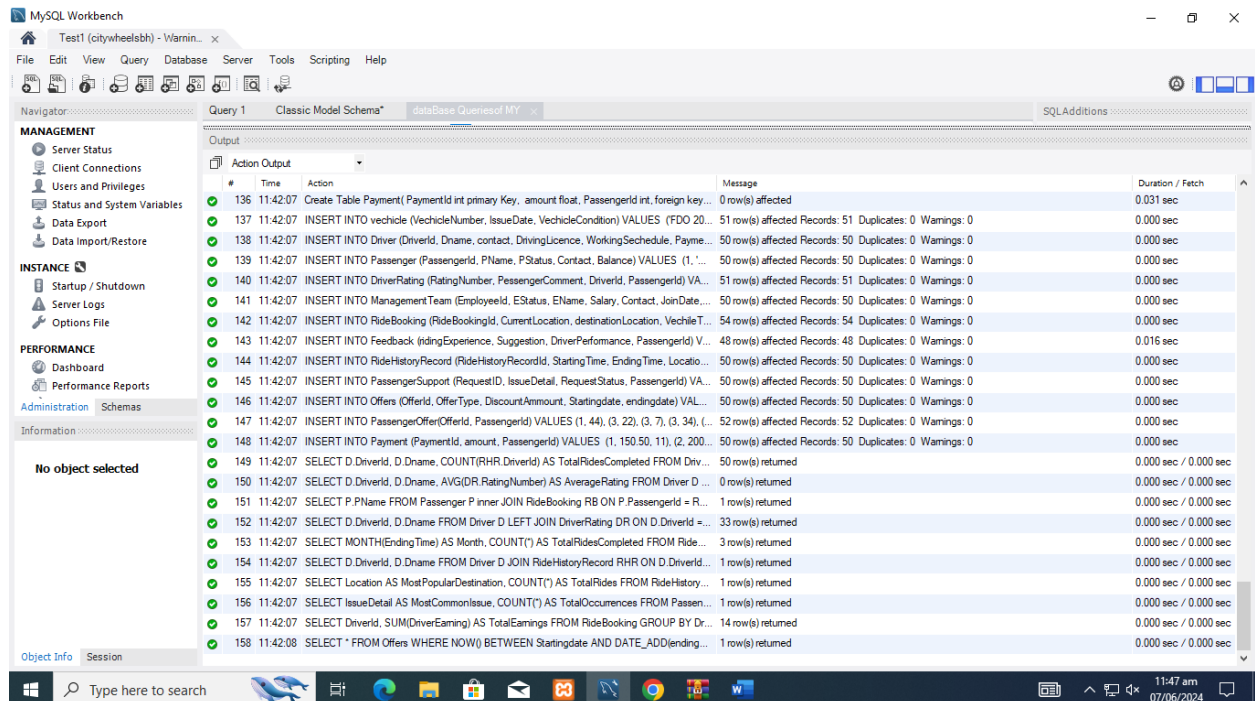


Figure 4: data insertion screenshots

## 5. BUSINESS QUERIES

### 5.1 Query 1: Driver Performance Analysis

**Purpose:** List drivers and the total number of rides they have completed.

SQL Query:

```
SELECT D.DriverId, D.Dname, COUNT(RHR.DriverId) AS TotalRidesCompleted
```

```
FROM Driver D
```

```
LEFT JOIN RideHistoryRecord RHR ON D.DriverId = RHR.DriverId
```

```
GROUP BY D.DriverId, D.Dname;
```

**Business Value:** Identifies top-performing drivers and those with low ride counts.

The screenshot displays the MySQL Workbench interface. The 'Query 1' tab is active, showing the following SQL query:

```
-- Task 1
SELECT D.DriverId, D.Dname, COUNT(RHR.DriverId) AS TotalRidesCompleted
FROM Driver D
LEFT JOIN RideHistoryRecord RHR ON D.DriverId = RHR.DriverId
GROUP BY D.DriverId, D.Dname;
```

The 'Result Grid' shows the output of the query, listing 13 drivers and their total rides completed:

DriverId	Dname	TotalRidesCompleted
1	M ali	1
2	Hussan	1
3	Wajid Ali	1
4	Noor Hassan	7
5	Bilal	1
6	Khalid	1
7	Ahmed	2
8	Saeed	1
9	Nadeem	1
10	Imran	1
11	Rizwan	1
12	Sohail	1
13	Kamran	1

The interface also includes a sidebar with 'MANAGEMENT' and 'PERFORMANCE' sections, and a bottom status bar showing the time as 11:48 am on 07/06/2024.

## 5.2 Query 2: High-Performance Driver Ratings

**Purpose:** Find the average rating of drivers who have completed more than 50 rides.

SQL Query:

```
SELECT D.DriverId, D.Dname, AVG(DR.RatingNumber) AS AverageRating
```

```
FROM Driver D
```

```
JOIN DriverRating DR ON D.DriverId = DR.DriverId
```

```
GROUP BY D.DriverId, D.Dname
```

```
HAVING COUNT(DR.DriverId) > 50;
```

**Note:** Returns no rows as no driver has completed 50+ rides in sample data.

The screenshot displays the MySQL Workbench interface. The 'Query 1' tab shows the following SQL code:

```
776 Left JOIN RideHistoryRecord RHR ON D.DriverId = RHR.DriverId
777 GROUP BY D.DriverId, D.Dname;
778
779
780 -- Task 2
781 Select D.DriverId, D.Dname, AVG(DR.RatingNumber) AS AverageRating FROM Driver D inner join DriverRating DR ON D.DriverId =
782 GROUP by D.DriverId, D.Dname Having COUNT(DR.DriverId) > 50;
```

The 'Result Grid' shows the following columns: DriverId, Dname, AverageRating. The 'Output' pane shows the execution results:

#	Time	Action	Message	Duration / Fetch
37	18:52:14	SELECT DriverId, SUM(DriverEarning) AS TotalEarnings from RideBooking GROUP BY DriverId...	14 row(s) returned	0.000 sec / 0.000 sec
38	18:52:14	SELECT * FROM Offers WHERE NOW() BETWEEN Startingdate AND DATE_ADD(endingdat...	1 row(s) returned	0.437 sec / 0.000 sec
39	19:55:45	Select D.DriverId, D.Dname, COUNT(RHR.DriverId) AS TotalRidesCompleted from Driver D Le...	50 row(s) returned	0.656 sec / 0.000 sec
40	19:59:32	Select D.DriverId, D.Dname, AVG(DR.RatingNumber) AS AverageRating FROM Driver D inne...	0 row(s) returned	0.140 sec / 0.000 sec

### 5.3 Query 3: Frequent Passengers

**Purpose:** List passengers who have used CityWheels more than 20 times.

SQL Query:

```
SELECT P.PName
```

```
FROM Passenger P
```

```
INNER JOIN RideBooking RB ON P.PassengerId = RB.PassengerId
```

```
GROUP BY P.PassengerId, P.PName
```

```
HAVING COUNT(RB.PassengerId) > 20;
```

**Business Value:** Identifies loyal customers for targeted marketing.

The screenshot displays the MySQL Workbench interface. The 'Query Editor' window shows a SQL query for finding frequent passengers. The 'Result Grid' shows one result: PassengerId 1, PName Naeem. The 'Output' window shows the execution log with four messages.

**Query Editor:**

```
782 GROUP by D.DriverId, D.DName Having COUNT(DR.DriverId) > 50;
783
784
785 -- Task 3
786 Select P.PassengerId,P.PName from Passenger P inner JOIN RideBooking RB on P.PassengerId = RB.PassengerId
787 GROUP BY P.PassengerId, P.PName Having COUNT(RB.PassengerId) > 20;
788
789 -- task 4
```

**Result Grid:**

PassengerId	PName
1	Naeem

**Output:**

#	Time	Action	Message	Duration / Fetch
38	18:52:14	SELECT * FROM Offers WHERE NOW() BETWEEN Startingdate AND DATE_ADD(endingdat...	1 row(s) returned	0.437 sec / 0.000 sec
39	19:55:45	Select D.DriverId, D.DName, COUNT(RHR.DriverId) AS TotalRidesCompleted from Driver D Le...	50 row(s) returned	0.656 sec / 0.000 sec
40	19:59:32	Select D.DriverId, D.DName, AVG(DR.RatingNumber) AS AverageRating FROM Driver D inne...	0 row(s) returned	0.140 sec / 0.000 sec
41	20:02:08	Select P.PassengerId,P.PName from Passenger P inner JOIN RideBooking RB on P.Passenger...	1 row(s) returned	0.187 sec / 0.000 sec

## 5.4 Query 4: Consistently High-Rated Drivers

**Purpose:** Identify drivers who have never received a rating below 4.

SQL Query:

```
SELECT D.DriverId, D.DName
```

```
FROM Driver D
```

```
LEFT JOIN DriverRating DR ON D.DriverId = DR.DriverId
```

```
GROUP BY D.DriverId, D.DName
```

```
HAVING COALESCE(MIN(DR.RatingNumber), 5) >= 4;
```

**Business Value:** Recognizes excellence for rewards and incentives.

The screenshot displays the MySQL Workbench interface. The 'Query Editor' window shows a SQL query for identifying drivers with a rating of 4 or higher. The query is as follows:

```
-- task 4
798 • Select D.DriverId, D.DName from Driver D left join DriverRating DR ON D.DriverId = DR.DriverId
799 • GROUP BY D.DriverId, D.DName
800 • Having COALESCE(MIN(DR.RatingNumber), 5) >= 4;
801 -- Task 4
802 • SELECT D.DriverId, D.DName from Driver D where D.DriverId NOT IN
803 • (SELECT DR.DriverId FROM DriverRating DR WHERE DR.RatingNumber < 4 );
```

The 'Result Grid' shows the output of the query, listing DriverId and DName for 10 drivers:

DriverId	DName
2	Hussan
10	Imran
12	Sohail
16	Asif
18	Adnan
20	Majid
22	Rashid
24	Saad
26	Enam
28	Enam

The 'Output' window shows the execution log, indicating that the query was executed successfully and returned 33 rows.

Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help.

## 5.5 Query 5: Monthly Ride Trends

**Purpose:** Find the total number of rides completed in each month of the current year.

SQL Query:

```
SELECT MONTH(EndingTime) AS Month, COUNT(*) AS TotalRidesCompleted
```

```
FROM RideHistoryRecord
```

```
WHERE YEAR(EndingTime) = YEAR(CURRENT_DATE())
```

```
GROUP BY MONTH(EndingTime);
```

**Business Value:** Seasonal trend analysis for resource planning.

The screenshot displays the MySQL Workbench interface. The 'Query Editor' window shows a SQL query for 'Query 1' in the 'dataBase Queries of MY\*' database. The query is as follows:

```
795 (SELECT DR.DriverId FROM DriverRating DR WHERE DR.RatingNumber < 4 );
796
797 -- task 5
798 • select* from RideHistoryRecord;
799 • Select MONTH(EndingTime) AS Month, COUNT(*) AS TotalRidesCompleted From RideHistoryRecord
800 where YEAR(EndingTime) = YEAR(CURRENT_DATE())
801 GROUP BY MONTH(EndingTime);
```

The 'Result Grid' shows the output of the query:

Month	TotalRidesCompleted
5	16
6	30
7	4

The 'Output' window shows the execution log with the following entries:

#	Time	Action	Message	Duration / Fetch
47	20:15:49	select* from RideHistoryRecord LIMIT 0, 1000	50 row(s) returned	0.000 sec / 0.000 sec
48	20:16:22	select* from RideHistoryRecord LIMIT 0, 1000	50 row(s) returned	0.000 sec / 0.000 sec
49	20:17:30	Select MONTH(EndingTime) AS Month, COUNT(*) AS TotalRidesCompleted From RideHistory...	3 row(s) returned	0.000 sec / 0.000 sec
50	20:17:40	Select MONTH(EndingTime) AS Month, COUNT(*) AS TotalRidesCompleted From RideHistory...	3 row(s) returned	0.000 sec / 0.000 sec

## 5.6 Query 6: Driver-Passenger Diversity

**Purpose:** Show drivers who have completed rides with at least three different passengers.

SQL Query:

```
SELECT D.DriverId, D.Dname
```

```
FROM Driver D
```

```
JOIN RideHistoryRecord RHR ON D.DriverId = RHR.DriverId
```

```
GROUP BY D.DriverId, D.Dname
```

```
HAVING COUNT(DISTINCT RHR.PassengerId) >= 3;
```

**Business Value:** Identifies drivers with broad customer reach.

The screenshot displays the MySQL Workbench interface. The 'Query Editor' window shows a SQL query for 'Query 1' with the following content:

```
-- task 6
Select D.DriverId, D.Dname From Driver D inner Join RideHistoryRecord RHR ON D.DriverId = RHR.DriverId
GROUP BY D.DriverId, D.Dname Having COUNT(DISTINCT RHR.PassengerId) >= 3;
```

The 'Result Grid' shows the results of the query, displaying a single row with DriverId 4 and Dname Noor Hassan.

DriverId	Dname
4	Noor Hassan

The 'Output' window shows the execution log with the following entries:

#	Time	Action	Message	Duration / Fetch
48	20:16:22	select* from RideHistoryRecord LIMIT 0, 1000	50 row(s) returned	0.000 sec / 0.000 sec
49	20:17:30	Select MONTH(EndingTime) AS Month, COUNT(*) AS TotalRidesCompleted From RideHistory...	3 row(s) returned	0.000 sec / 0.000 sec
50	20:17:40	Select MONTH(EndingTime) AS Month, COUNT(*) AS TotalRidesCompleted From RideHistory...	3 row(s) returned	0.000 sec / 0.000 sec
51	20:19:39	Select D.DriverId, D.Dname From Driver D inner Join RideHistoryRecord RHR ON D.DriverId = ...	1 row(s) returned	0.000 sec / 0.000 sec

## 5.7 Query 7: Popular Destinations

**Purpose:** Determine the most popular destination among all rides.

SQL Query:

```
SELECT Location AS MostPopularDestination, COUNT(*) AS TotalRides
```

```
FROM RideHistoryRecord
```

```
GROUP BY Location
```

```
ORDER BY TotalRides DESC
```

```
LIMIT 1;
```

**Business Value:** Strategic planning for driver positioning and marketing.

The screenshot displays the MySQL Workbench interface. The 'Query' tab is active, showing a SQL query for finding the most popular destination. The 'Result Grid' shows the output of the query, which is a single row for 'UCP Lahore' with a total of 3 rides. The 'Output' tab at the bottom shows the execution log with timestamps and messages for each step of the query execution.

**Query 1**

```
804 • Select D.DriverId, D.Dname From Driver D inner Join RideHistoryRecord RHR ON D.DriverId = RHR.DriverId
805 GROUP BY D.DriverId, D.Dname Having COUNT(DISTINCT RHR.PassengerId) >= 3;
806
807
808 -- task 7
809 • Select Location AS MostPopularDestination, COUNT(*) as TotalRides FROM RideHistoryRecord
810 GROUP BY Location order by TotalRides Desc LIMIT 1;
811
```

**Result Grid**

MostPopularDestination	TotalRides
UCP Lahore	3

**Output**

#	Time	Action	Message	Duration / Fetch
49	20:17:30	Select MONTH(EndingTime) AS Month, COUNT(*) AS TotalRidesCompleted From RideHistory...	3 row(s) returned	0.000 sec / 0.000 sec
50	20:17:40	Select MONTH(EndingTime) AS Month, COUNT(*) AS TotalRidesCompleted From RideHistory...	3 row(s) returned	0.000 sec / 0.000 sec
51	20:19:39	Select D.DriverId, D.Dname From Driver D inner Join RideHistoryRecord RHR ON D.DriverId = ...	1 row(s) returned	0.000 sec / 0.000 sec
52	20:21:28	Select Location AS MostPopularDestination, COUNT(*) as TotalRides FROM RideHistoryReco...	1 row(s) returned	0.000 sec / 0.000 sec

## 5.8 Query 8: Common Support Issues

**Purpose:** Identify the most common issue reported in support tickets.

SQL Query:

```
SELECT IssueDetail AS MostCommonIssue, COUNT(*) AS TotalOccurrences
```

```
FROM PassengerSupport
```

```
GROUP BY IssueDetail
```

```
ORDER BY TotalOccurrences DESC
```

```
LIMIT 1;
```

**Business Value:** Focus training and process improvements on frequent issues.

The screenshot displays the MySQL Workbench interface. The 'Query' tab is active, showing a SQL query for 'task 8'. The query is as follows:

```
-- task 8
814 • Select IssueDetail AS MostCommonIssue, COUNT(*) AS TotalOccurrences from PassengerSupport
815 group by IssueDetail ORDER by TotalOccurrences Desc
816 limit 1;
```

The 'Result Grid' shows the output of the query:

MostCommonIssue	TotalOccurrences
My driver did not arrive on time.	12

The 'Output' tab is also visible, showing a list of actions and their results:

#	Time	Action	Message	Duration / Fetch
50	20:17:40	Select MONTH(EndingTime) AS Month, COUNT(*) AS TotalRidesCompleted From RideHistory...	3 row(s) returned	0.000 sec / 0.000 sec
51	20:19:39	Select D.DriverId, D.DName From Driver D inner Join RideHistoryRecord RHR ON D.DriverId = ...	1 row(s) returned	0.000 sec / 0.000 sec
52	20:21:28	Select Location AS MostPopularDestination, COUNT(*) as TotalRides FROM RideHistoryReco...	1 row(s) returned	0.000 sec / 0.000 sec
53	20:23:05	Select IssueDetail AS MostCommonIssue, COUNT(*) AS TotalOccurrences from PassengerSup...	1 row(s) returned	0.000 sec / 0.000 sec

## 5.9 Query 9: Driver Earnings

**Purpose:** Show the total earnings for each driver in the current year.

SQL Query:

```
select D.DriverId,SUM(Rb.DriverEarning) AS TotalEarnings ,
```

```
(Rh.StartingTime) from Driver D Left join RideBooking Rb
```

```
on D.DriverId=RB.DriverId
```

```
Left join RideHistoryRecord RH on D.DriverId=Rh.DriverId
```

```
WHERE YEAR(RH.StartingTime) = YEAR(current_date())
```

```
GROUP BY DriverId;
```

**Business Value:** Financial reporting and driver compensation tracking.

The screenshot displays the MySQL Workbench interface. The 'Query' tab is active, showing a SQL query for 'task 9'. The query is as follows:

```
-- task 9
820 • Select* from RideBooking;
821 • Select* from RideHistoryRecord;
822 • select D.DriverId,SUM(Rb.DriverEarning) AS TotalEarnings ,
823 (Rh.StartingTime) from Driver D Left join RideBooking Rb
824 on D.DriverId=RB.DriverId
825 Left join RideHistoryRecord RH on D.DriverId=Rh.DriverId
826 WHERE YEAR(RH.StartingTime) = YEAR(current_date())
827 GROUP BY DriverId;
828
```

The 'Result Grid' shows the output of the query, which includes columns for DriverId, TotalEarnings, and StartingTime. The data is as follows:

DriverId	TotalEarnings	StartingTime
1	1025	2024-07-03 14:00:00
2	1065	2024-06-25 13:00:00
3	985	2024-07-01 10:00:00
4	9695	2024-05-23 12:00:00
5	1110	2024-06-26 15:00:00
6	NULL	2024-07-02 12:00:00
7	NULL	2024-05-27 10:00:00

The 'Output' tab shows the execution log, including the time taken for each step and the number of rows returned. The log indicates that the query was executed successfully and returned 43 rows.

## 5.10 Query 10: Active Promotions

**Purpose:** Get a list of all promotions that are currently active and expire within the next 30 days.

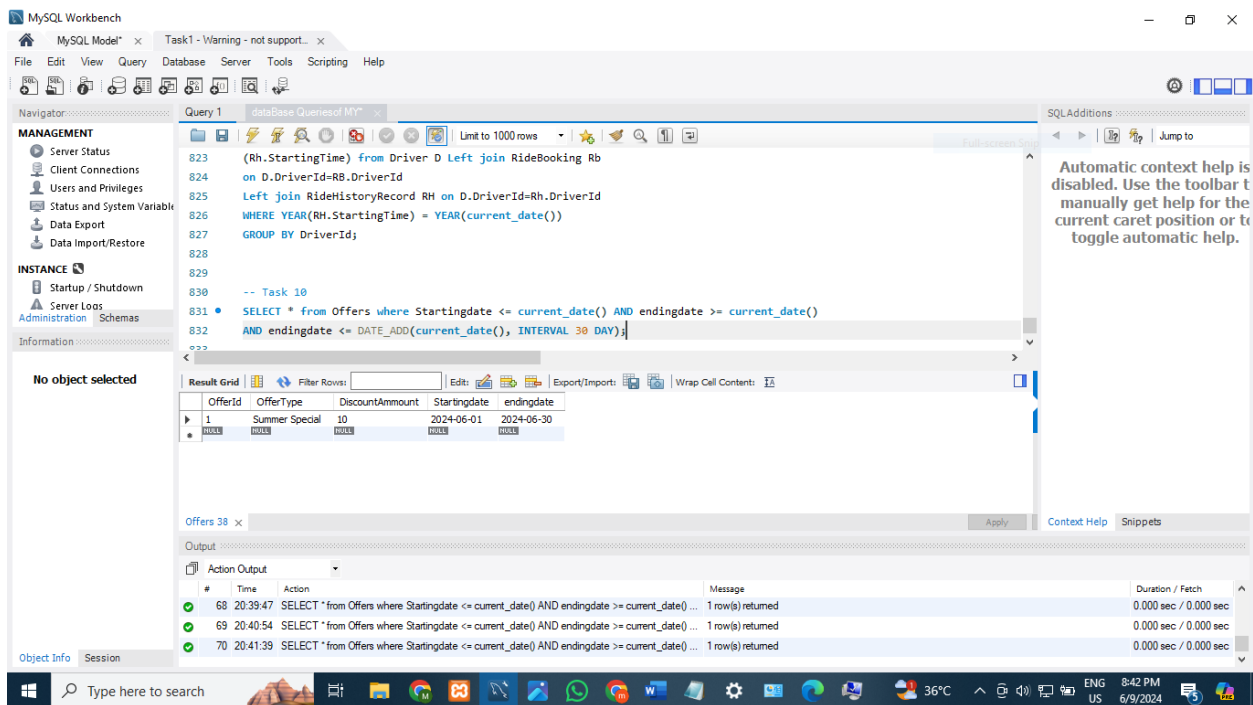
SQL Query:

SELECT \*

FROM Offers

WHERE NOW() BETWEEN Startingdate AND DATE\_ADD(endingdate, INTERVAL 30 DAY);

**Business Value:** Marketing campaign management and promotion tracking.



The screenshot displays the MySQL Workbench interface. The 'Query' tab is active, showing a SQL query that filters for active promotions. The query is as follows:

```
(Rh.StartingTime) from Driver D Left join RideBooking Rb
on D.DriverId=Rb.DriverId
Left join RideHistoryRecord RH on D.DriverId=Rh.DriverId
WHERE YEAR(RH.StartingTime) = YEAR(current_date())
GROUP BY DriverId;

-- Task 10
SELECT * from Offers where Startingdate <= current_date() AND endingdate >= current_date()
AND endingdate <= DATE_ADD(current_date(), INTERVAL 30 DAY);
```

The 'Result Grid' shows the output of the query, which includes a table with columns: OfferId, OfferType, DiscountAmount, Startingdate, and endingdate. The first row shows an offer with OfferId 1, OfferType 'Summer Special', DiscountAmount 10, Startingdate 2024-06-01, and endingdate 2024-06-30. The 'Output' pane at the bottom shows the execution of the query, indicating that 1 row(s) were returned.

OfferId	OfferType	DiscountAmount	Startingdate	endingdate
1	Summer Special	10	2024-06-01	2024-06-30

## 6. CONCLUSION

In conclusion, the CityWheels database project represents a successful application of database design principles to a real-world business scenario. The system effectively manages all core operations of a ride-hailing service while maintaining data integrity through proper normalization and relationship constraints.

The project's dual focus on technical implementation and legal compliance demonstrates understanding that modern database systems must operate within complex regulatory frameworks. The research on cross-border data sharing under PDPL and GDPR provides valuable insights for CityWheels's global expansion strategy.

This documentation, complete with ER diagrams, schema details, sample data, and analytical queries, provides all necessary information for deployment and maintenance. The system's scalability ensures it can grow with the business, supporting expansion from Pakistan to international markets.

Overall, this project successfully meets all course requirements while delivering practical value for a real-world business scenario, demonstrating both technical competence and awareness of broader legal and ethical considerations in database management.

---