Data Modelling

TASK ONE

Designing a Relational Database

1.1 Introduction:

The main aspect of this task is data modelling for the Kangaroo Delivery Company using the Entity Relationship Diagram (ERD) technique with Crow Foot Notation techniques. In this task, we will design a relational database for the more compatible business of Kangaroo Delivery Company using MySQL workbench. Our special focus is to develop a potential database that serves the needs of the business growth through its entities, relationships, and participation constraints.

1.2 Background:

In response to the dynamic landscape of online delivery services, Kangaroo, an emerging player in the field, is aspiring to develop an effective Relational Database Management System (RDBMS). The desire of the company is to meet the increasing demands of its operations with the driving force behind this strategic decision. This entails creating an extensive database system with careful consideration for payroll details that includes essential data categories like Customers, Items, Restaurants, Orders, Drivers, and Vehicles. Our goal is to develop a dynamic Relational Database Management System R(DBMS) that supports Kangaroo's rising business model and improves its capacity to offer top-notch online delivery services with ERD model based on Crow Foot notations.

1.3 Relationship and Cardinality/Multiplicity: In the context of the Kangaroo database, we will consider the following relationships and cardinalities:

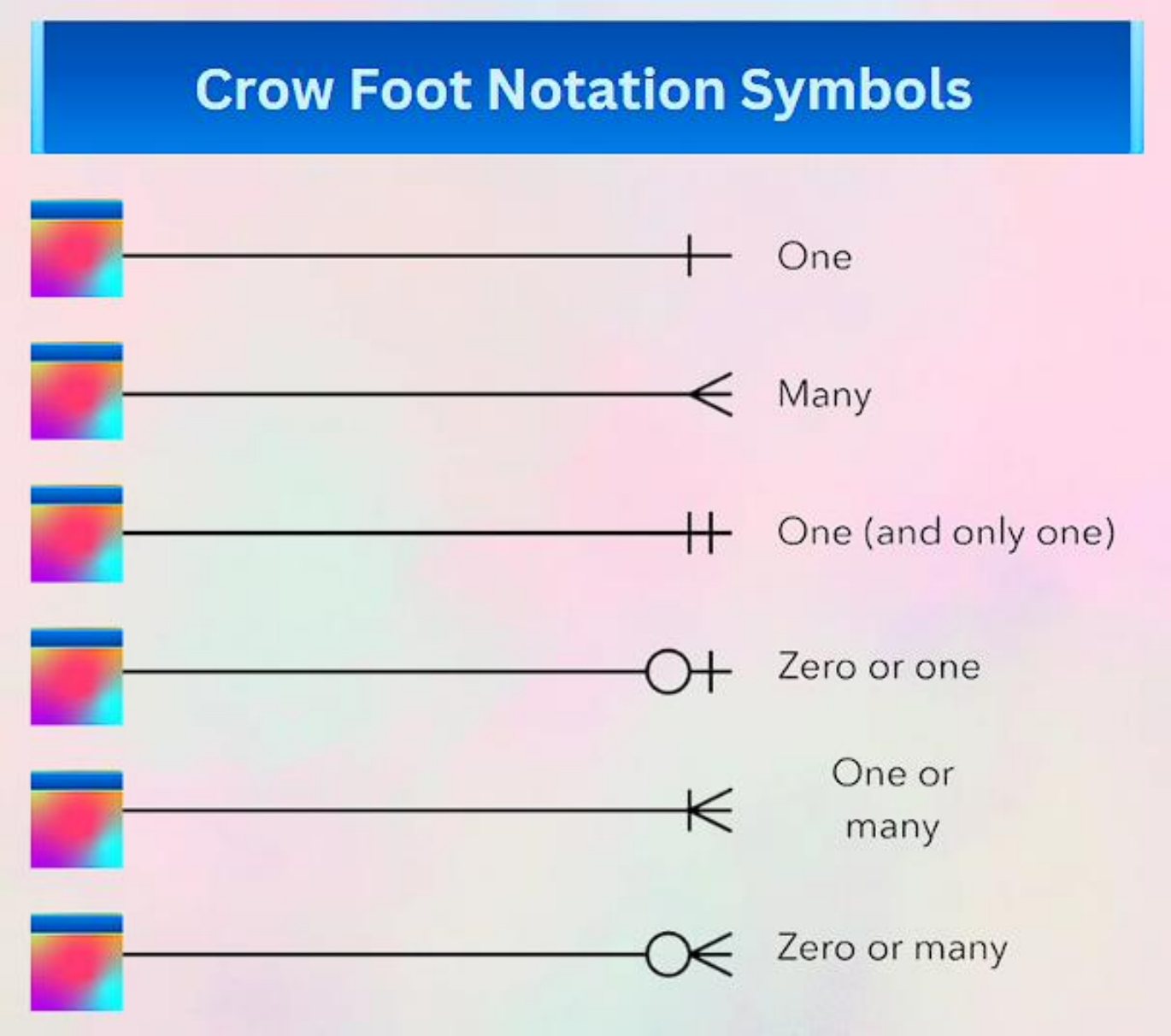
1. One to One: A driver is assigned a single motorbike.
2. One to Many: A customer can place multiple orders. A manager manages multiple drivers.
3. Many to One: A driver is associated with one restaurant.
4. Many to Many: A driver can deliver multiple orders, and an order can be delivered by multiple drivers.

The concept of cardinality/multiplicity defines the number of instances a table can have in a relationship. We will utilize cardinality terms such as One and Only One, One or Many, Zero or One or Many, and Zero or One, as applicable.

1.4 Crow Foot Notation Symbols:

To visually represent the relationships and cardinalities in the Kangaroo database, we will employ Crow Foot Notation symbols:

- A straight line indicates a "One" relationship.
- A crow's foot symbol (three lines) indicates a "Many" relationships.
- Combining these symbols with entities represents different cardinalities, such as "One to Many" or "Many to One."

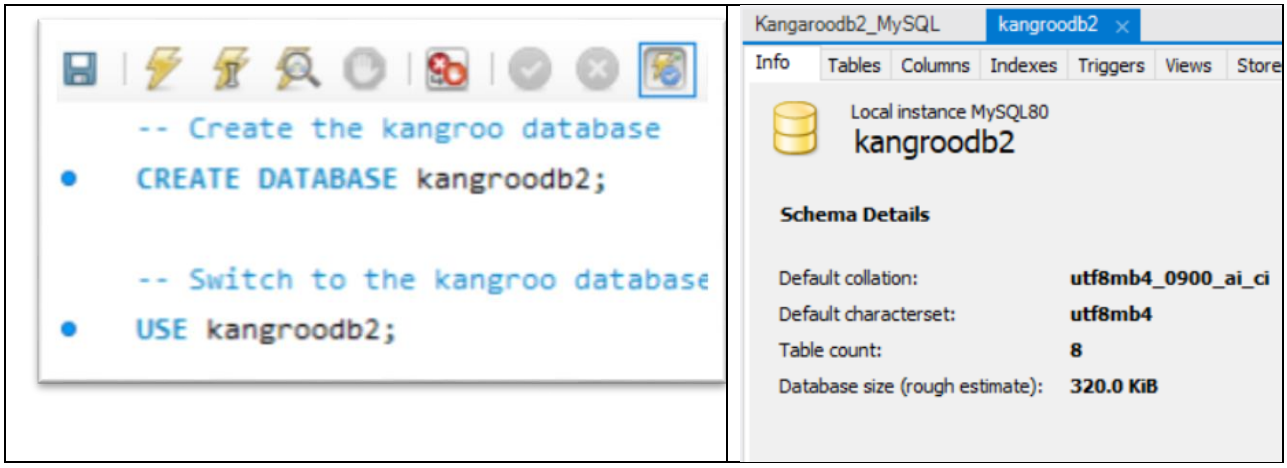


1.5 Entities of Kangaroo Database:


EIGHT TABLES OF KANGAROO DATABASE
Customers
Drivers
Vehicles
Managers
Restaurants
Items
Orders
OrderItems

1.6. Database creation.

It is crucial to build a database before building tables since it acts as the primary container for storing and grouping relevant data tables. By logically arranging tables within schemas, databases offer an organized environment for effective table management. A logical and well-organized structure for the entire system is ensured by first building the "kangaroo" database, which creates a specialized field where all the interconnected tables for customers, items, restaurants, orders, drivers, and more can exist.



1.6. Table Creation in MySQL workbench:

We write the following code to create the tables and did the query to see the table structure:

1.6.1. Customers Table:

Code

```
-- Create Customers table
CREATE TABLE Customers (
    CustomerID INT PRIMARY KEY,
    LastName VARCHAR(50),
    FirstName VARCHAR(50),
    Email VARCHAR(100),
    PhoneNumber VARCHAR(20)
);
```

Table Structure

```
11 • select * from kangaroo.customers;
12
```

Result Grid | Filter Rows: | Edit:

CustomerID	LastName	FirstName	Email	PhoneNumber
NULL	NULL	NULL	NULL	NULL

1.6.2 Drivers Table:

```sql
CREATE TABLE Drivers (
    DriverID INT PRIMARY KEY AUTO_INCREMENT,
    Name VARCHAR(100) NOT NULL,
    Salary DECIMAL(10, 2) DEFAULT 0.00,
    Email VARCHAR(100) UNIQUE,
    ManagerID INT,
    LicenseNumber VARCHAR(50) UNIQUE,
    IssueDate DATE,
    CountryOfIssue VARCHAR(50),
    ExpiryDate DATE,
    FOREIGN KEY (ManagerID) REFERENCES Drivers(DriverID) ON DELETE SET NULL
);
```

```sql
select * from kangaroo.drivers;
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

DriverID	Name	Salary	Email	ManagerID	LicenseNumber	IssueDate	CountryOfIssue	ExpiryDate
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

1.6.3 Restaurants Table:

```sql
-- Create the Restaurants table
CREATE TABLE Restaurants (
    RestaurantID INT PRIMARY KEY AUTO_INCREMENT,
    RestaurantName VARCHAR(100) NOT NULL,
    Address VARCHAR(200)
);
```

1.6.4 Vehicles Table:

```
114 • ⊖ CREATE TABLE Vehicles (
115         VehicleID INT PRIMARY KEY AUTO_INCREMENT,
116         RegistrationNumber VARCHAR(50) UNIQUE,
117         Color VARCHAR(50),
118         PurchaseDate DATE,
119         EngineSize VARCHAR(20),
120         DriverID INT,
121         FOREIGN KEY (DriverID) REFERENCES Drivers(DriverID) ON DELETE SET NULL
```

```
11 •    select * from kangaroo.vehicles;
12
```

Result Grid | Filter Rows: | Edit: | Export/I

VehicleID	RegistrationNumber	Color	PurchaseDate	EngineSize	DriverID
NULL	NULL	NULL	NULL	NULL	NULL

1.6.5 Managers Table:

```
36      -- Create Managers table
37 • ⊖ CREATE TABLE Managers (
38          ManagerID INT PRIMARY KEY,
39          Name VARCHAR(100),
90          DriverID INT,
91          FOREIGN KEY (DriverID) REFERENCES Drivers(DriverID)
92      );
```

1.6.6 Orders Table:

```
171 • CREATE TABLE Orders (
172       OrderID INT PRIMARY KEY AUTO_INCREMENT,
173       OrderDate DATE,
174       CustomerID INT,
175       DriverID INT,
176       RestaurantID INT,
177       FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID) ON DELETE CASCADE,
178       FOREIGN KEY (DriverID) REFERENCES Drivers(DriverID) ON DELETE SET NULL,
179       FOREIGN KEY (RestaurantID) REFERENCES Restaurants(RestaurantID) ON DELETE SET NULL
180   );
```

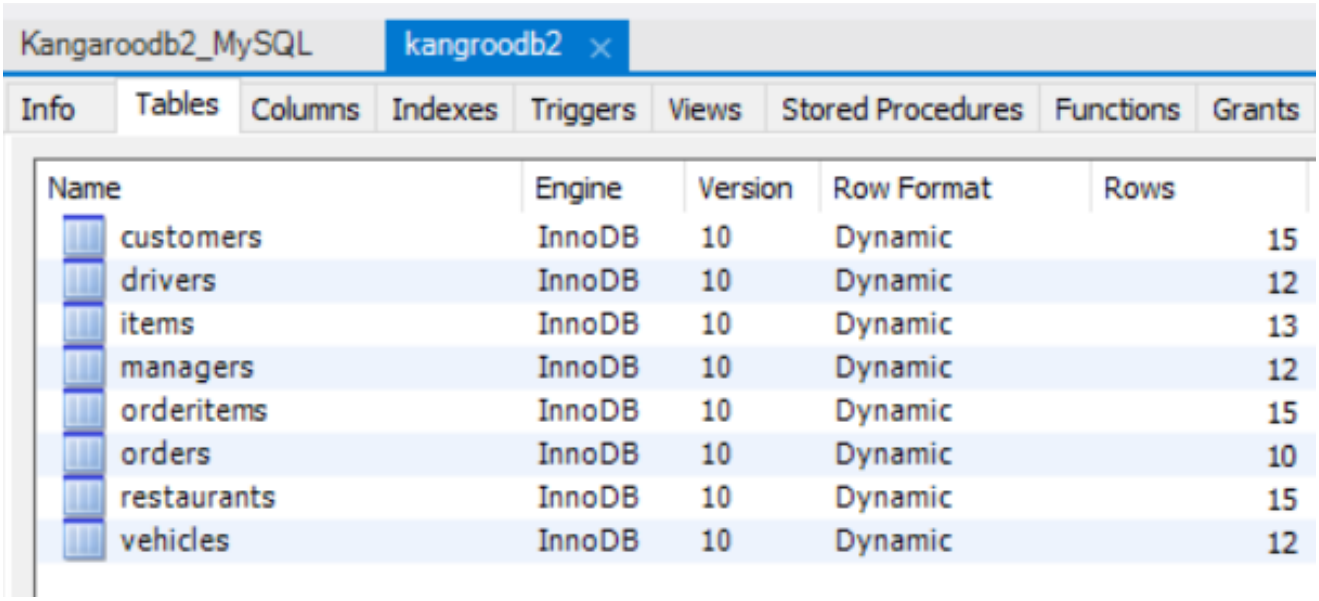
1.6.7 Items Table:

```
CREATE TABLE Items (
    ItemID INT PRIMARY KEY AUTO_INCREMENT,
    ItemName VARCHAR(100) NOT NULL,
    ItemPrice DECIMAL(10, 2) DEFAULT 0.00,
    Category VARCHAR(50)
);
```

1.6.8 OrderItems Table:

```
169     -- Create the OrderItems table for the many-to-many relationship between Orders and Items
170 • CREATE TABLE OrderItems (
171       OrderID INT,
172       ItemID INT,
173       PRIMARY KEY (OrderID, ItemID),
174       FOREIGN KEY (OrderID) REFERENCES Orders(OrderID) ON DELETE CASCADE,
175       FOREIGN KEY (ItemID) REFERENCES Items(ItemID) ON DELETE CASCADE
176   );
177
```

1.6.9 total tables in kangaroodb2 database:



Kangaroodb2_MySQL | kangroodb2

Info | Tables | Columns | Indexes | Triggers | Views | Stored Procedures | Functions | Grants

Name	Engine	Version	Row Format	Rows
customers	InnoDB	10	Dynamic	15
drivers	InnoDB	10	Dynamic	12
items	InnoDB	10	Dynamic	13
managers	InnoDB	10	Dynamic	12
orderitems	InnoDB	10	Dynamic	15
orders	InnoDB	10	Dynamic	10
restaurants	InnoDB	10	Dynamic	15
vehicles	InnoDB	10	Dynamic	12

1.6.9 Screen Shot of Indexes:

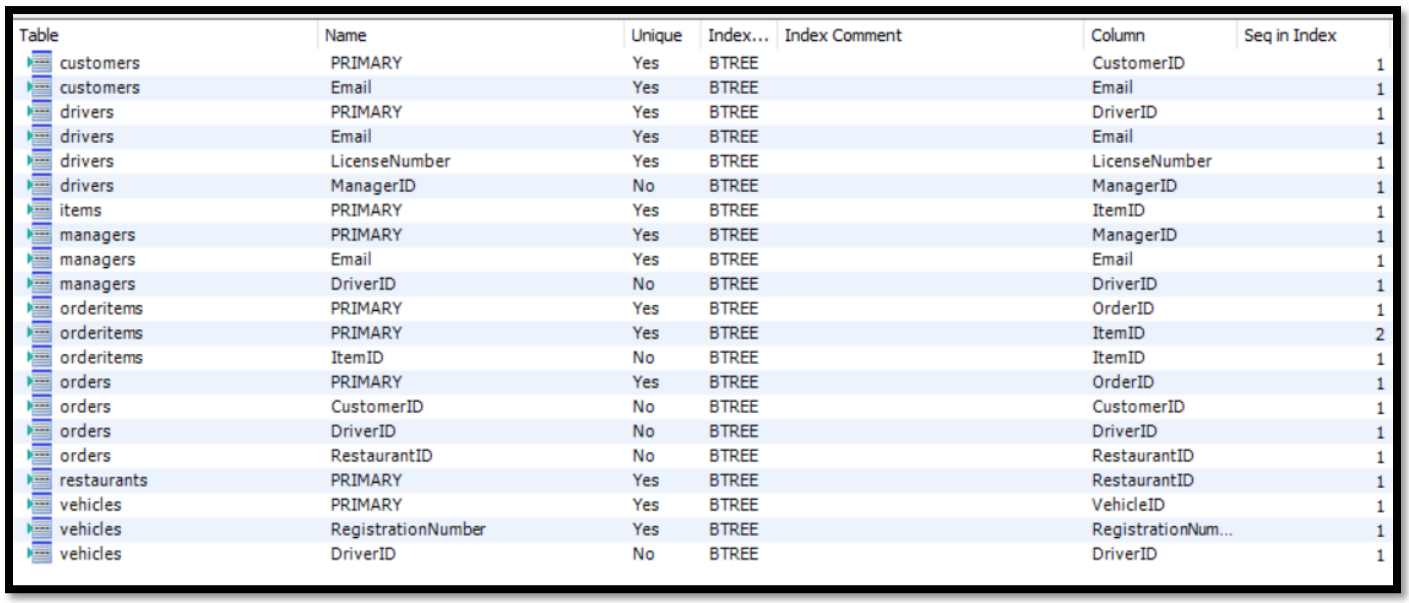


Table	Name	Unique	Index...	Index Comment	Column	Seq in Index
customers	PRIMARY	Yes	BTREE		CustomerID	1
customers	Email	Yes	BTREE		Email	1
drivers	PRIMARY	Yes	BTREE		DriverID	1
drivers	Email	Yes	BTREE		Email	1
drivers	LicenseNumber	Yes	BTREE		LicenseNumber	1
drivers	ManagerID	No	BTREE		ManagerID	1
items	PRIMARY	Yes	BTREE		ItemID	1
managers	PRIMARY	Yes	BTREE		ManagerID	1
managers	Email	Yes	BTREE		Email	1
managers	DriverID	No	BTREE		DriverID	1
orderitems	PRIMARY	Yes	BTREE		OrderID	1
orderitems	PRIMARY	Yes	BTREE		ItemID	2
orderitems	ItemID	No	BTREE		ItemID	1
orders	PRIMARY	Yes	BTREE		OrderID	1
orders	CustomerID	No	BTREE		CustomerID	1
orders	DriverID	No	BTREE		DriverID	1
orders	RestaurantID	No	BTREE		RestaurantID	1
restaurants	PRIMARY	Yes	BTREE		RestaurantID	1
vehicles	PRIMARY	Yes	BTREE		VehicleID	1
vehicles	RegistrationNumber	Yes	BTREE		RegistrationNum...	1
vehicles	DriverID	No	BTREE		DriverID	1

1.7 Relationships and Participation Constraints Matrix:

We can present the table relationship with the following table:

Relationship	Participating Entities	Relationship Type	Participation Constraint Explanation
Customers to Orders	Customer - Order	One to Many	Each customer places one or more orders (Mandatory), an order is optional.
Drivers to Motorbikes	Driver - Motorbike	One to One	Each driver is assigned one motorbike (Mandatory), and vice versa.
Managers to Drivers	Manager - Driver	One to Many	Each manager supervises one or more drivers (Mandatory), a driver is optional.
Drivers to Restaurants	Driver - Restaurant	Many to One	Each driver is associated with one restaurant (Mandatory), a restaurant is mandatory.
Orders to OrderItems	Order - OrderItem	One to Many	Each order contains one or more items (Mandatory), an item in an order is optional.
Items to OrderItems	Item - OrderItem	Many to One	Each item can be part of multiple orders (Mandatory), an item in an order is mandatory.
Drivers to Deliveries	Driver - Delivery	One to Many	Each driver makes one or more deliveries (Mandatory), a delivery is optional.
Orders to Deliveries	Order - Delivery	One to Many	Each order can have one or more deliveries (Mandatory), a delivery is optional.

In terms of participation constraints:

- **Mandatory Participation (1):** In a mandatory participation, each entity instance in one entity set must be associated with at least one entity instance in another entity set. For example, each **Customer** must have at least one associated **Order**.
- **Optional Participation (0 or 1):** In an optional participation, an entity instance in one entity set may or may not be associated with an entity instance in another entity set. For example, a **Delivery** may or may not be associated with an **Order.**

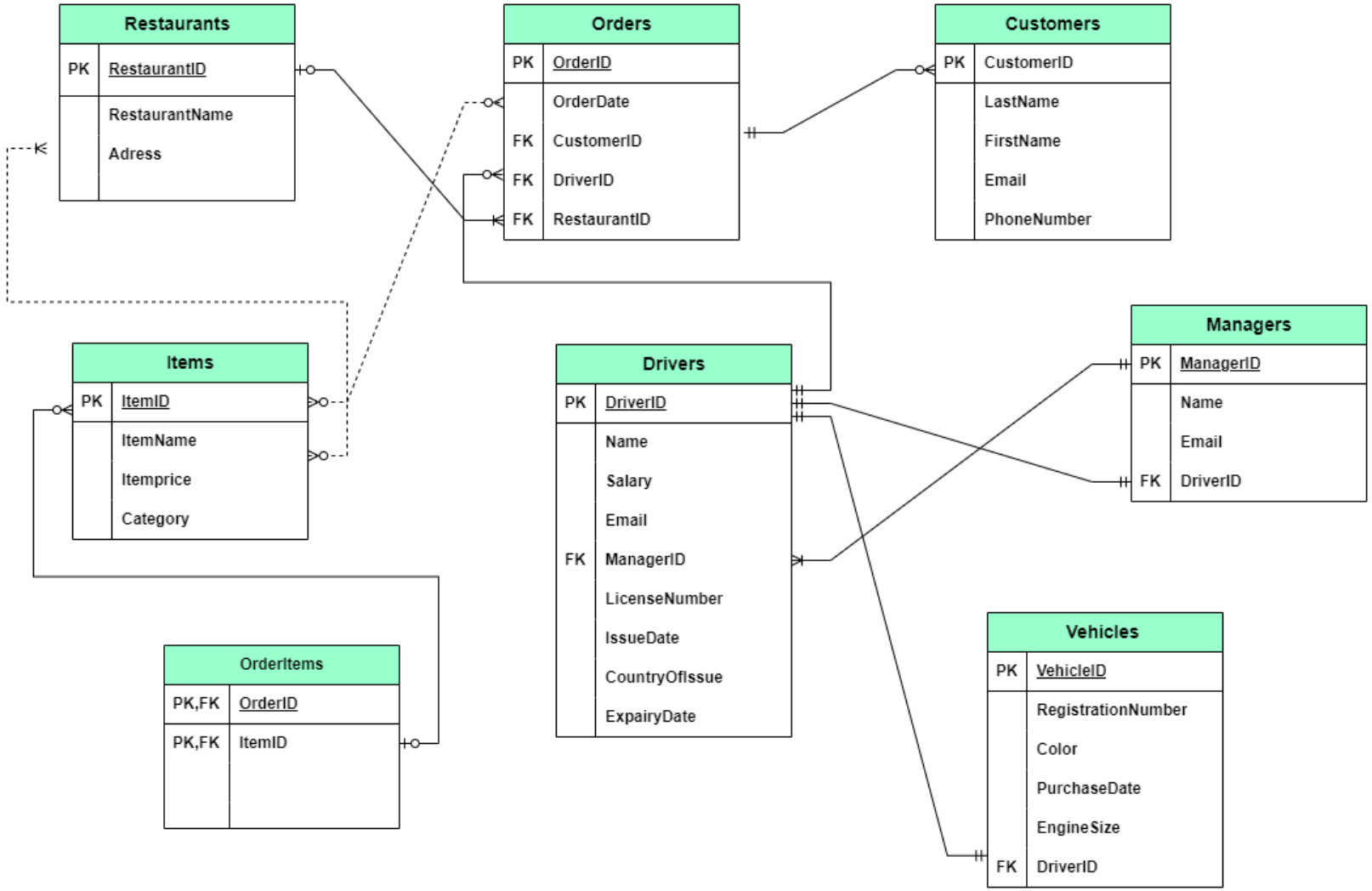
1.8 Cardinality and Relationships Overview between the Tables:

Relationship	Parent Table	Child Table	Parent PK	Child FK	Cardinality	Participation Constraint
Customers - Orders	Customers	Orders	CustomerID	CustomerID	1:N (One-to-Many)	Mandatory on Customers
Drivers - Vehicles	Drivers	Vehicles	DriverID	DriverID	1:1 (One-to-One)	Optional on Drivers
Drivers - Managers	Managers	Drivers	ManagerID	ManagerID	1:N (One-to-Many)	Optional on Managers
Drivers - Orders	Drivers	Orders	DriverID	DriverID	1:N (One-to-Many)	Optional on Drivers
Drivers - Managers	Drivers	Managers	DriverID	DriverID	1:1 (One-to-One)	Optional on Drivers
Items - OrderItems	Items	OrderItems	ItemID	ItemID	1:N (One-to-Many)	Mandatory on Items
Orders - Order Items	Orders	OrderItems	OrderID	OrderID	1:N (One-to-Many)	Mandatory on Orders
Orders - Restaurants	Orders	Restaurants	RestaurantID	RestaurantID	1:N (One-to-Many)	Optional on Orders
Restaurant-Managers	Managers	Restaurants	ManagerID	ManagerID	1:N (One-to-Many)	Optional on Managers

1.9 Data Modelling:

In this stage we are going to create the Logical ERD. We can do that by using Wizard in MySQL workbench but as using any wizard is prohibited, we will draw with draw.io (online drawing tools) to establish the entity relationship.

Logical ERD of Kangaroo Food Delivery Company :



TASK TWO

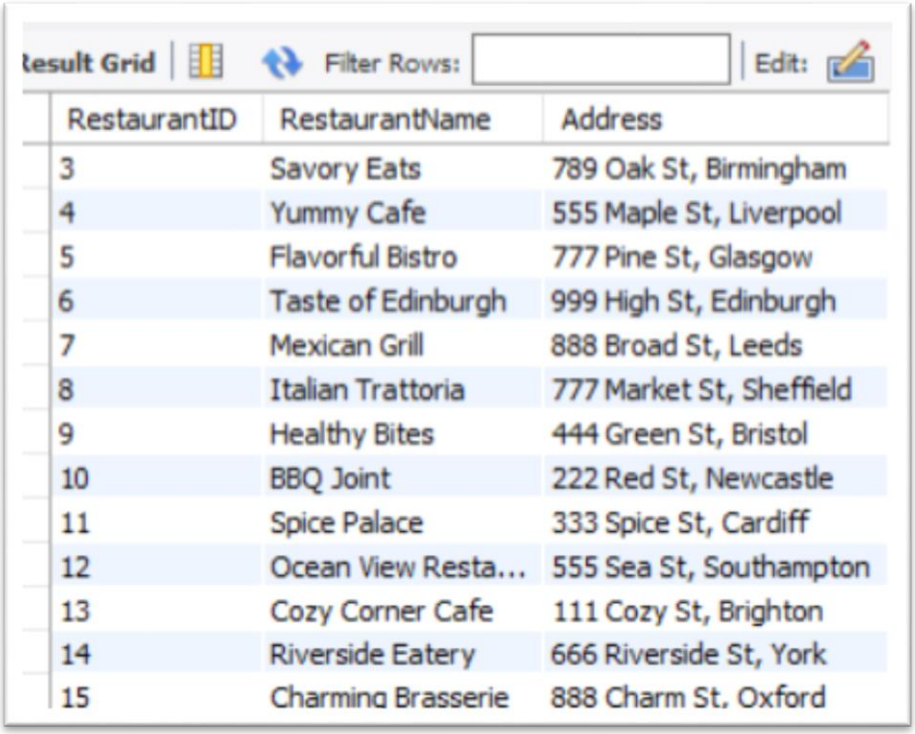
Data Population and Implementation of Database Design

2.Data Populations and Perform Queries:

2.1 Restaurants Table:

Following is the code to populate the restaurant’s table.

```
INSERT INTO Restaurants (RestaurantID, RestaurantName, Address)
VALUES
    (1, 'Delicious Diner', '123 Main St, London'),
    (2, 'Tasty Tavern', '456 Elm St, Manchester'),
    (3, 'Savory Eats', '789 Oak St, Birmingham'),
    (4, 'Yummy Cafe', '555 Maple St, Liverpool'),
    (5, 'Flavorful Bistro', '777 Pine St, Glasgow'),
    (6, 'Taste of Edinburgh', '999 High St, Edinburgh'),
    (7, 'Mexican Grill', '888 Broad St, Leeds'),
    (8, 'Italian Trattoria', '777 Market St, Sheffield'),
    (9, 'Healthy Bites', '444 Green St, Bristol'),
    (10, 'BBQ Joint', '222 Red St, Newcastle'),
```



Result Grid | Filter Rows: | Edit:

RestaurantID	RestaurantName	Address
3	Savory Eats	789 Oak St, Birmingham
4	Yummy Cafe	555 Maple St, Liverpool
5	Flavorful Bistro	777 Pine St, Glasgow
6	Taste of Edinburgh	999 High St, Edinburgh
7	Mexican Grill	888 Broad St, Leeds
8	Italian Trattoria	777 Market St, Sheffield
9	Healthy Bites	444 Green St, Bristol
10	BBQ Joint	222 Red St, Newcastle
11	Spice Palace	333 Spice St, Cardiff
12	Ocean View Resta...	555 Sea St, Southampton
13	Cozy Corner Cafe	111 Cozy St, Brighton
14	Riverside Eatery	666 Riverside St, York
15	Charming Brasserie	888 Charm St, Oxford

2.2 Customers Table:

```
-- Populating Customers table with sample data
INSERT INTO customers (CustomerID, LastName, FirstName, Email, PhoneNumber)
VALUES
  (1, 'Anderson', 'John', 'john.anderson@example.com', '07555-1234'),
  (2, 'Sainani', 'Roshika', 'roshika.sainani@example.com', '07555-5678'),
  (3, 'Williams', 'Michael', 'michael.williams@outlook.com', '07555-9876'),
  (4, 'Brown', 'Jessica', 'jessica.brown@example.com', '07555-4321'),
  (5, 'Jones', 'David', 'david.jones@yahoo.com', '07555-8765'),
  (6, 'Davis', 'Sophia', 'sophia.davis@examail.com', '07555-3456'),
  (7, 'Miller', 'Ethan', 'ethan.miller@example.com', '07555-7890'),
  (8, 'Wilson', 'Olivia', 'olivia.wilson@example.com', '07555-6543'),
  (9, 'Moore', 'Liam', 'liam.moore@example.com', '07555-2345'),
  (10, 'Taylor', 'Ava', 'ava.taylor@example.com', '07555-9087'),
  (11, 'Anderson', 'Noah', 'noah.anderson@example.com', '07555-5671'),
  (12, 'Thomas', 'Emma', 'emma.thomas@example.com', '07555-1238'),
  (13, 'Jackson', 'William', 'william.jackson@example.com', '07555-8762'),
  (14, 'White', 'Sofia', 'sofia.white@example.com', '07555-3412'),
  (15, 'Harris', 'James', 'james.harris@example.com', '07555-9081');
```

```
71 •    SELECT * FROM Customers
72      WHERE CustomerID IN (1, 3, 8, 11);
```

Result Grid | Filter Rows: | Edit: | Export/Imp

CustomerID	LastName	FirstName	Email	PhoneNumbe
1	Anderson	John	john.anderson@example.com	07555-1234
3	Williams	Michael	michael.williams@outlook.com	07555-9876
8	Wilson	Olivia	olivia.wilson@example.com	07555-6543
11	Anderson	Noah	noah.anderson@example.com	07555-5671

2.3 Drivers Table:

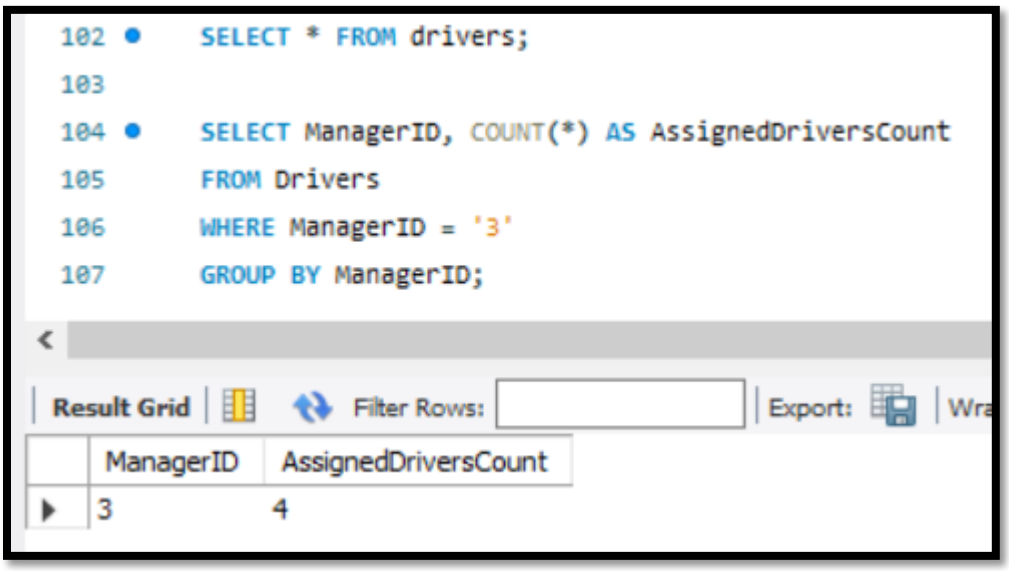
```
INSERT INTO Drivers (Name, Salary, Email, ManagerID, LicenseNumber, IssueDate, CountryOfIssue, ExpiryDate)
VALUES
    ('Michael Anderson', 2500.00, 'michael.anderson@example.com', 1, 'DL12345', '2020-01-15', 'UK', '2025-01-15'),
    ('Sophia Williams', 2300.00, 'sophia.williams@example.com', 1, 'DL67890', '2019-05-20', 'US', '2024-05-20'),
    ('James Smith', 2200.00, 'james.smith@example.com', 3, 'DL98765', '2018-08-10', 'UK', '2023-08-10'),
    ('Emma Johnson', 2400.00, 'emma.johnson@example.com', 2, 'DL54321', '2022-02-25', 'CA', '2027-02-25'),
    ('David Brown', 2100.00, 'david.brown@example.com', 3, 'DL45678', '2017-07-05', 'AU', '2022-07-05'),
    ('Olivia Jones', 2600.00, 'olivia.jones@example.com', 5, 'DL87654', '2021-04-18', 'NZ', '2026-04-18'),
    ('William Davis', 2000.00, 'william.davis@example.com', 5, 'DL23456', '2016-03-12', 'UK', '2021-03-12'),
    ('Ava Wilson', 2300.00, 'ava.wilson@example.com', 4, 'DL76543', '2015-11-30', 'US', '2020-11-30'),
    ('Liam Garcia', 2400.00, 'liam.garcia@example.com', 3, 'DL78901', '2019-09-23', 'CA', '2024-09-23'),
    ('Sophie Martinez', 2100.00, 'sophie.martinez@example.com', 2, 'DL01234', '2018-06-14', 'AU', '2023-06-14'),
    ('Noah Lee', 2500.00, 'noah.lee@example.com', 1, 'DL89012', '2020-12-08', 'NZ', '2025-12-08'),
    ('Isabella Clark', 2200.00, 'isabella.clark@example.com', 3, 'DL34567', '2017-10-03', 'UK', '2022-10-03');
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

DriverID	Name	Salary	Email	ManagerID	LicenseNumber	IssueDate	CountryOfIssue	ExpiryDate
1	Michael Anderson	2500.00	michael.anderson@example.com	1	DL12345	2020-01-15	UK	2025-01-15
2	Sophia Williams	2300.00	sophia.williams@example.com	1	DL67890	2019-05-20	US	2024-05-20
3	James Smith	2200.00	james.smith@example.com	3	DL98765	2018-08-10	UK	2023-08-10
4	Emma Johnson	2400.00	emma.johnson@example.com	2	DL54321	2022-02-25	CA	2027-02-25
5	David Brown	2100.00	david.brown@example.com	3	DL45678	2017-07-05	AU	2022-07-05
6	Olivia Jones	2600.00	olivia.jones@example.com	5	DL87654	2021-04-18	NZ	2026-04-18
7	William Davis	2000.00	william.davis@example.com	5	DL23456	2016-03-12	UK	2021-03-12
8	Ava Wilson	2300.00	ava.wilson@example.com	4	DL76543	2015-11-30	US	2020-11-30
9	Liam Garcia	2400.00	liam.garcia@example.com	3	DL78901	2019-09-23	CA	2024-09-23
10	Sophie Martinez	2100.00	sophie.martinez@example.com	2	DL01234	2018-06-14	AU	2023-06-14
11	Noah Lee	2500.00	noah.lee@example.com	1	DL89012	2020-12-08	NZ	2025-12-08
12	Isabella Clark	2200.00	isabella.clark@example.com	3	DL34567	2017-10-03	UK	2022-10-03

Query 1:

To check the table structure and check constraints we have performed the following queries:



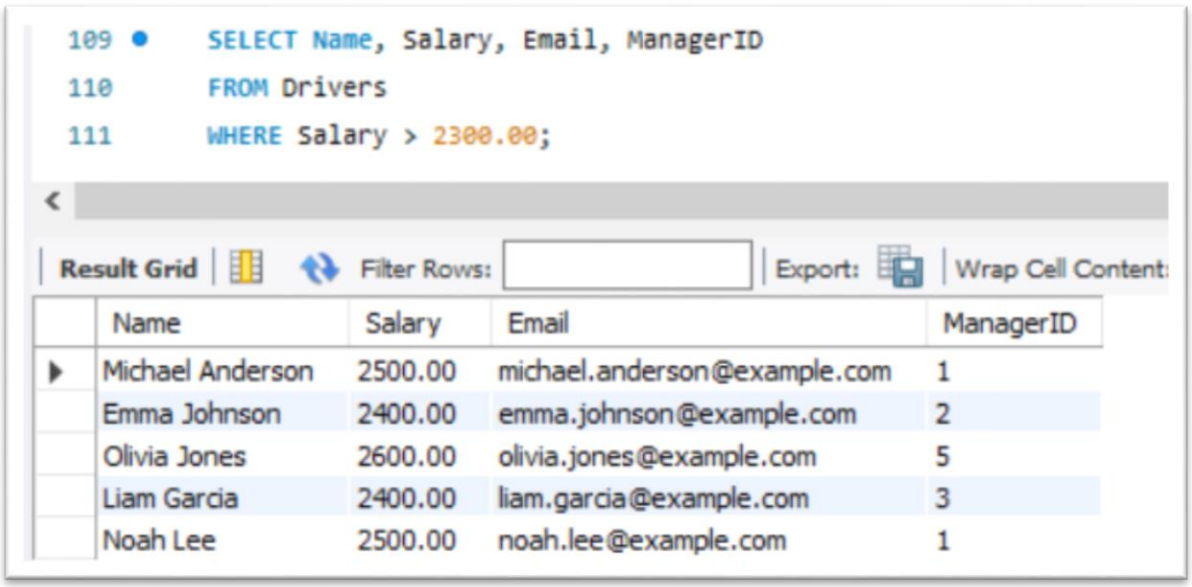
```
102    SELECT * FROM drivers;
103
104    SELECT ManagerID, COUNT(*) AS AssignedDriversCount
105    FROM Drivers
106    WHERE ManagerID = '3'
107    GROUP BY ManagerID;
```

Result Grid | Filter Rows: | Export: | Wra

ManagerID	AssignedDriversCount
3	4

We used the **COUNT** function to provide counts based on different conditions and groupings.

Query 2:



```
109    SELECT Name, Salary, Email, ManagerID
110    FROM Drivers
111    WHERE Salary > 2300.00;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Name	Salary	Email	ManagerID
Michael Anderson	2500.00	michael.anderson@example.com	1
Emma Johnson	2400.00	emma.johnson@example.com	2
Olivia Jones	2600.00	olivia.jones@example.com	5
Liam Garcia	2400.00	liam.garcia@example.com	3
Noah Lee	2500.00	noah.lee@example.com	1

This query returned the details (Name, Salary, Email, and ManagerID) of drivers whose salary is greater than $2300.

2.4 Vehicles Table:

```
INSERT INTO Vehicles (RegistrationNumber, Color, PurchaseDate, EngineSize, DriverID)
VALUES
    ('ABC123', 'Blue', '2022-03-10', '250cc', 1),
    ('XYZ789', 'Red', '2021-08-20', '200cc', 2),
    ('DEF456', 'Green', '2020-05-15', '150cc', 3),
    ('GHI789', 'Black', '2019-10-08', '180cc', 4),
    ('JKL012', 'Silver', '2022-01-05', '220cc', 5),
    ('MNO345', 'White', '2021-06-30', '190cc', 6),
    ('PQR678', 'Yellow', '2020-11-25', '210cc', 7),
    ('STU901', 'Purple', '2018-04-14', '170cc', 8),
    ('VWX234', 'Orange', '2017-09-09', '160cc', 9),
    ('YZA567', 'Gray', '2019-02-28', '200cc', 10),
    ('BCD890', 'Brown', '2022-07-17', '180cc', 11),
    ('EFG123', 'Pink', '2018-12-03', '140cc', 12);
```

Result Grid | Filter Rows: | Edit: | Export/Im

VehicleID	RegistrationNumber	Color	PurchaseDate	EngineSize	DriverID
1	ABC123	Blue	2022-03-10	250cc	1
2	XYZ789	Red	2021-08-20	200cc	2
3	DEF456	Green	2020-05-15	150cc	3
4	GHI789	Black	2019-10-08	180cc	4
5	JKL012	Silver	2022-01-05	220cc	5
6	MNO345	White	2021-06-30	190cc	6
7	PQR678	Yellow	2020-11-25	210cc	7
8	STU901	Purple	2018-04-14	170cc	8
9	VWX234	Orange	2017-09-09	160cc	9
10	YZA567	Gray	2019-02-28	200cc	10
11	BCD890	Brown	2022-07-17	180cc	11
12	EFG123	Pink	2018-12-03	140cc	12

2.5 Items Table:

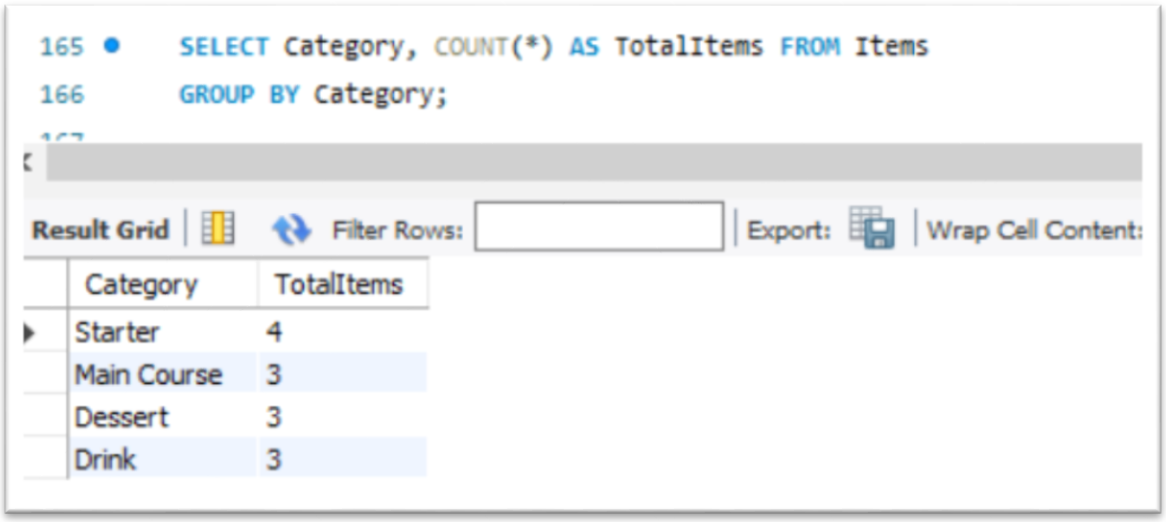
```
INSERT INTO Items (ItemID, ItemName, ItemPrice, Category)
VALUES
    (1, 'Garlic Bread', 3.50, 'Starter'),
    (2, 'Pizza Margherita', 10.99, 'Main Course'),
    (3, 'Chocolate Cake', 5.99, 'Dessert'),
    (4, 'Coke', 1.99, 'Drink'),
    (5, 'Chicken Wings', 7.50, 'Starter'),
    (6, 'Pasta Carbonara', 12.99, 'Main Course'),
    (7, 'Ice Cream Sundae', 6.49, 'Dessert'),
    (8, 'Orange Juice', 2.49, 'Drink'),
    (9, 'Bruschetta', 4.75, 'Starter'),
    (10, 'Steak with Fries', 16.99, 'Main Course'),
    (11, 'Tiramisu', 7.25, 'Dessert'),
    (12, 'Lemonade', 1.75, 'Drink'),
    (13, 'Caprese Salad', 6.25, 'Starter');
```

Result Grid | Filter Rows: | Edit:

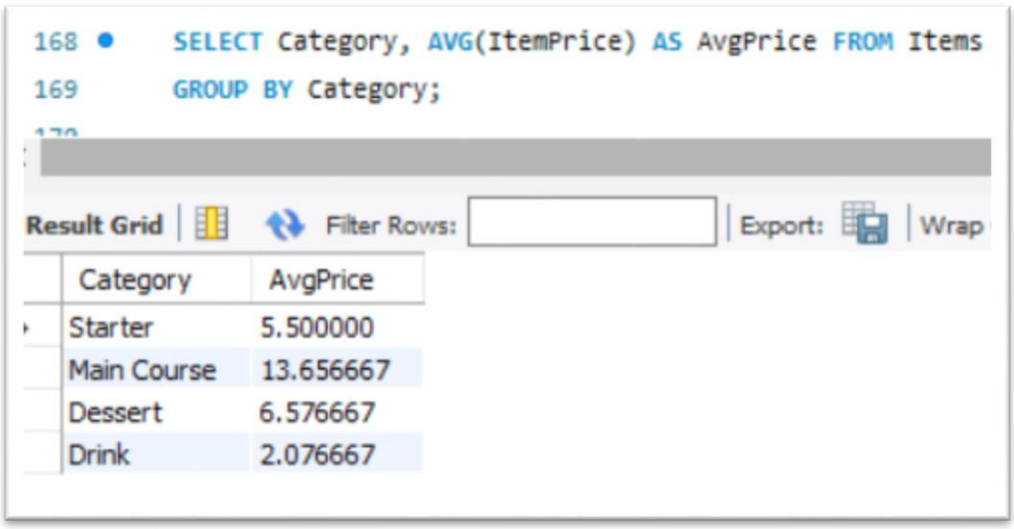
ItemID	ItemName	ItemPrice	Category
1	Garlic Bread	3.50	Starter
2	Pizza Margherita	10.99	Main Course
3	Chocolate Cake	5.99	Dessert
4	Coke	1.99	Drink
5	Chicken Wings	7.50	Starter
6	Pasta Carbonara	12.99	Main Course
7	Ice Cream Sundae	6.49	Dessert
8	Orange Juice	2.49	Drink
9	Bruschetta	4.75	Starter
10	Steak with Fries	16.99	Main Course
11	Tiramisu	7.25	Dessert
12	Lemonade	1.75	Drink
13	Caprese Salad	6.25	Starter

Query 3:

To Retrieve the total number of items in each category we wrote the following code in our kangaroobd2 database:



Query 4: To Retrieve the average price of items in each category:



By writing this query we can check the table functionality.

2.6 Orders Table:

```
182 •   INSERT INTO Orders (OrderID, OrderDate, CustomerID, DriverID, RestaurantID)
183     VALUES
184         (1, '2023-08-01', 1, 1, 1),
185         (2, '2023-08-02', 2, 2, 2),
186         (3, '2023-08-03', 3, 3, 3),
187         (4, '2023-08-04', 4, 4, 4),
188         (5, '2023-08-05', 5, 5, 5),
189         (6, '2023-08-06', 6, 6, 6),
190         (7, '2023-08-07', 7, 7, 7),
191         (8, '2023-08-08', 8, 8, 8),
192         (9, '2023-08-09', 9, 9, 9),
193         (10, '2023-08-10', 10, 10, 10);
```

Query 5: Retrieve Order Information with Customer and Driver Details

This query fetches order information along with details of the associated customer and driver.

```
195 •   SELECT
196         o.OrderID,
197         o.OrderDate,
198         c.FirstName AS CustomerFirstName,
199         c.LastName AS CustomerLastName,
200         d.Name AS DriverName,
201         r.RestaurantName
202     FROM
203         Orders o
204     JOIN
205         Customers c ON o.CustomerID = c.CustomerID
206     LEFT JOIN
207         Drivers d ON o.DriverID = d.DriverID
208     JOIN
209         Restaurants r ON o.RestaurantID = r.RestaurantID;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

OrderID	OrderDate	CustomerFirstName	CustomerLastName	DriverName	RestaurantName
1	2023-08-01	John	Anderson	Michael Anderson	Delicious Diner
2	2023-08-02	Roshika	Sainani	Sophia Williams	Tasty Tavern
3	2023-08-03	Michael	Williams	James Smith	Savory Eats
4	2023-08-04	Jessica	Brown	Emma Johnson	Yummy Cafe
5	2023-08-05	David	Jones	David Brown	Flavorful Bistro
6	2023-08-06	Sophia	Davis	Olivia Jones	Taste of Edinburgh
7	2023-08-07	Ethan	Miller	William Davis	Mexican Grill
8	2023-08-08	Olivia	Wilson	Ava Wilson	Italian Trattoria
9	2023-08-09	Liam	Moore	Liam Garcia	Healthy Bites
10	2023-08-10	Ava	Taylor	Sophie Martinez	BBQ Joint

Explanation:

- The query uses **JOIN** to combine information from multiple tables.
- It fetches the **OrderID**, **OrderDate**, customer's first and last names, driver's name, and restaurant's name.
- **JOIN** clauses are used to link the **Orders** table with the **Customers**, **Drivers**, and **Restaurants** tables based on their respective IDs.
- The use of **LEFT JOIN** with the **Drivers** table ensures that even if a driver is not assigned to an order (DriverID is NULL), the order details will still be retrieved.
- The result provides a comprehensive view of each order along with the related customer and driver information.

2.7 OrderItems Table:

Code for inserting Values

```
INSERT INTO OrderItems (OrderID, ItemID)
VALUES
    (1, 1),
    (1, 2),
    (2, 2),
    (2, 3),
    (3, 3),
    (3, 4),
    (4, 4),
    (4, 1),
    (5, 1),
    (5, 2),
    (6, 3),
    (6, 4),
    (7, 2),
    (7, 3),
    (8, 4);
```

Table Data view

```
select * from orderitem
```

Result Grid | Filter Rows:

OrderID	ItemID
1	2
2	2
5	2
7	2
2	3
3	3
6	3
7	3
3	4
4	4

Query 6: Calculate Total Revenue per Restaurant:

This query calculates the total revenue generated by each restaurant based on the sum of item prices from the associated orders.

```
SELECT
    r.RestaurantID,
    r.RestaurantName,
    SUM(i.ItemPrice) AS TotalRevenue
FROM
    Orders o
JOIN
    Restaurants r ON o.RestaurantID = r.RestaurantID
JOIN
    OrderItems oi ON o.OrderID = oi.OrderID
JOIN
    Items i ON oi.ItemID = i.ItemID
GROUP BY
    r.RestaurantID, r.RestaurantName;
```

Result Grid | Filter Rows: | Export:

RestaurantID	RestaurantName	TotalRevenue

Explanation:

- The query uses multiple **JOIN** clauses to link the **Orders**, **Restaurants**, **OrderItems**, and **Items** tables.
- It calculates the total revenue for each restaurant by summing the item prices from the associated orders.
- The **GROUP BY** clause is used to group the results by **RestaurantID** and **RestaurantName**.
- Each restaurant's total revenue is then displayed in the result set.

Query 7: Retrieve Customers and Their Most Recent Orders

This query retrieves customer information along with details of their most recent order.

```
253 •       SELECT
254         c.CustomerID,
255         c.FirstName,
256         c.LastName,
257         o.OrderID AS RecentOrderID,
258         o.OrderDate AS RecentOrderDate,
259         r.RestaurantName AS RecentRestaurant
260     FROM
261         Customers c
262     LEFT JOIN
263         (
264             SELECT
265                 OrderID,
266                 CustomerID,
267                 OrderDate,
268                 RestaurantID
269             FROM
270                 Orders o1
271             WHERE
272                 OrderDate = (
273                     SELECT MAX(OrderDate)
274                     FROM Orders o2
275                     WHERE o1.CustomerID = o2.CustomerID
276                 )
277     ) o ON c.CustomerID = o.CustomerID
278     LEFT JOIN
279         Restaurants r ON o.RestaurantID = r.RestaurantID;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

CustomerID	FirstName	LastName	RecentOrderID	RecentOrderDate	RecentRestaurant
1	John	Anderson	1	2023-08-01	Delicious Diner
2	Roshika	Sainani	2	2023-08-02	Tasty Tavern
3	Michael	Williams	3	2023-08-03	Savory Eats
4	Jessica	Brown	4	2023-08-04	Yummy Cafe
5	David	Jones	5	2023-08-05	Flavorful Bistro
6	Sophia	Davis	6	2023-08-06	Taste of Edinburgh
7	Ethan	Miller	7	2023-08-07	Mexican Grill
8	Olivia	Wilson	8	2023-08-08	Italian Trattoria
9	Liam	Moore	9	2023-08-09	Healthy Bites
10	Ava	Taylor	10	2023-08-10	BBQ Joint
11	Noah	Anderson	NULL	NULL	NULL
12	Emma	Thomas	NULL	NULL	NULL
13	William	Jackson	NULL	NULL	NULL
14	Sofia	White	NULL	NULL	NULL
15	James	Harris	NULL	NULL	NULL

Explanation:

- The query retrieves customer information (**CustomerID**, **FirstName**, **LastName**) along with details of their most recent order.
- It uses a subquery to select the most recent order for each customer by comparing the **OrderDate** using a correlated subquery.
- The subquery is aliased as **o** and is left-joined with the **Customers** table on the **CustomerID**.
- An additional left join is performed with the **Restaurants** table to include the restaurant name of the most recent order.
- This query provides a result set showing each customer's information and the details of their most recent order, including order ID, order date, and restaurant name.

2.8 Manager Table:

```
290 •   INSERT INTO Managers (Name, Email, DriverID)
291     VALUES
292         ('John Manager', 'john.manager@example.com', 1),
293         ('Emily Manager', 'emily.manager@example.com', 2),
294         ('Sarah Manager', 'sarah.manager@example.com', 3),
295         ('Daniel Manager', 'daniel.manager@example.com', 4),
296         ('Alice Manager', 'alice.manager@example.com', 5),
297         ('Liam Manager', 'liam.manager@example.com', 6),
298         ('Sophie Manager', 'sophie.manager@example.com', 7),
299         ('Ava Manager', 'ava.manager@example.com', 8),
300         ('Noah Manager', 'noah.manager@example.com', 9),
301         ('Isabella Manager', 'isabella.manager@example.com', 10),
302         ('Ethan Manager', 'ethan.manager@example.com', 11),
303         ('Mia Manager', 'mia.manager@example.com', 12);
```

```
463 •   select * from managers;
```

Result Grid | Filter Rows: | Edit:

ManagerID	Name	Email	DriverID
1	John Manager	john.manager@example.com	1
2	Emily Manager	emily.manager@example.com	2
3	Sarah Manager	sarah.manager@example.com	3
4	Daniel Manager	daniel.manager@example.com	4
5	Alice Manager	alice.manager@example.com	5
6	Liam Manager	liam.manager@example.com	6
7	Sophie Manager	sophie.manager@example.com	7
8	Ava Manager	ava.manager@example.com	8
9	Noah Manager	noah.manager@example.com	9
10	Isabella Manager	isabella.manager@example.com	10
11	Ethan Manager	ethan.manager@example.com	11

Query 8: Retrieve Managers with Their Total Drivers

```
305 •   SELECT
306         m.ManagerID,
307         m.Name AS ManagerName,
308         COUNT(d.DriverID) AS TotalDrivers
309     FROM
310         Managers m
311     LEFT JOIN
312         Drivers d ON m.ManagerID = d.ManagerID
313     GROUP BY
314         m.ManagerID, m.Name
315     ORDER BY
316         TotalDrivers DESC;
```

Result Grid | Filter Rows:

ManagerID	ManagerName	TotalDrivers
3	Sarah Manager	4
1	John Manager	3
2	Emily Manager	2
5	Alice Manager	2
4	Daniel Manager	1

This query retrieves a list of managers along with the total number of drivers each manager supervises. It uses a LEFT JOIN to connect the Managers table with the Drivers table based on the ManagerID column. The COUNT function is then used to calculate the total number of drivers for each manager. The GROUP BY clause groups the results by ManagerID and Name, and the ORDER BY clause sorts the results in descending order of total drivers.

Query 9: Calculate Average Salary per Manager

```
318 •   SELECT
319         m.ManagerID,
320         m.Name AS ManagerName,
321         AVG(d.Salary) AS AvgSalary
322     FROM
323         Managers m
324     LEFT JOIN
325         Drivers d ON m.ManagerID = d.ManagerID
326     GROUP BY
327         m.ManagerID, m.Name
328     ORDER BY
329         AvgSalary DESC;
```

This query calculates the average salary of drivers managed by each manager and presents the results in descending order of average salary. It selects the ManagerID and Name columns from the Managers table and uses the AVG function to calculate the average salary (Salary) for drivers managed by each manager. The GROUP BY clause groups the results by ManagerID and Name, and the ORDER BY clause orders the results by average salary in descending order.

Query 10: Retrieve Drivers with Their Total Orders

```
331 •       SELECT
332         d.DriverID,
333         d.Name AS DriverName,
334         COUNT(o.OrderID) AS TotalOrders
335     FROM
336         Drivers d
337     LEFT JOIN
338         Orders o ON d.DriverID = o.DriverID
339     GROUP BY
340         d.DriverID, d.Name
341     ORDER BY
342         TotalOrders DESC;
```

Result Grid | Filter Rows:

DriverID	DriverName	TotalOrders
3	James Smith	1
4	Emma Johnson	1
5	David Brown	1
6	Olivia Jones	1
7	William Davis	1
8	Ava Wilson	1
9	Liam Garcia	1
10	Sophie Martinez	1
11	Noah Lee	0
12	Isabella Clark	0

This query retrieves a list of drivers along with the total number of orders each driver has delivered. It uses a LEFT JOIN to connect the Drivers table with the Orders table based on the DriverID column. The COUNT function is then used to calculate the total number of orders for each driver. The GROUP BY clause groups the results by DriverID and Name, and the ORDER BY clause sorts the results in descending order of total orders. This query provides valuable information about the drivers' performance in terms of order deliveries.

TASK THREE

“Optimizing Kangaroo’s Business Operations Through Strategic Database Design”

3.1 Introduction:

For the food delivery industries to become more operationally efficient, effective database architecture is essential. To develop effective operations and individualised customer services, this database design digs into the world of the Kangaroo food delivery service and examines how proper database structures can effortlessly connect with its unique needs. In this task, we will illustrate the benefits of database design adapted for Kangaroo's business performance by a thorough analysis of the relationships among several tables, including customers, drivers, restaurants, and orders.

The Kangaroo can maximize the value of their data, acquire a competitive edge, and prosper in today's data-centric digital environment by giving smart database design priority.

3.3 Harnessing the Power of Databases for Kangaroo's Business Growth:

The elaborate explanation of how the database design supports the business scenario by considering each table and other relevant aspects of Kangaroo’s business aspects.

3.3.1 Data Integrity and Relationships:

By breaking down the information into separate tables, each with a specific purpose, our database design ensures data integrity. The primary keys and foreign keys of the tables established a relationship between these tables, to protect data duplication and other inconsistencies.

For example, the "Customers" table is linked to the "Orders" table through the "CustomerID" foreign key, which ensures that orders are associated with valid customers. Therefore, performs a reliable representation of the real-world relationships between customers and their orders.

3.3.2 Centralized the Monitoring Systems:

As Kangaroo experiences growth in its business, the demand for personnel to facilitate operations increases. Managing the expanding volume of data can be challenging. Effective database systems offer a solution by enabling secure, centralized data administration to increase the possibilities of success.

3.3.3 Enhance Human Resource Management:

Utilizing an SQL database for drivers, managers record management brings about time and cost savings. The use of a SQL database facilitates the HR duties, automates time-consuming tasks, and speeds up data processing for staff hours, leave, payroll, benefits, and other topics.

3.3.4 Efficient Queries:

The ERD structure of the database supports efficient querying. Queries involving multiple tables can be executed with speed and accuracy due to the well-defined relationships. For instance, a query to retrieve details of orders

along with customer information and items ordered can be easily accomplished through JOIN operations between the "Customers," "Orders," and "OrderItems" tables. Efficient Query can be benefited for Kangaroo by the following ways:

I. Real-Time Tracking and Monitoring:

Efficient queries enable real-time tracking and monitoring of orders, drivers, and delivery status. Business managers can quickly retrieve and analyse data to see where drivers are, which orders are pending, and estimated delivery times. With the help of this function, the company can reach customers accurately and make the necessary adjustments in time to guarantee on-time delivery.

```
431 •   SELECT
432         o.OrderID,
433         c.FirstName AS CustomerFirstName,
434         c.LastName AS CustomerLastName,
435         d.Name AS DriverName,
436         r.RestaurantName,
437         o.OrderDate,
438         o.EstimatedDeliveryTime
439     FROM
440         Orders o
441     JOIN
442         Customers c ON o.CustomerID = c.CustomerID
442         Customers c ON o.CustomerID = c.CustomerID
443     JOIN
444         Drivers d ON o.DriverID = d.DriverID
445     JOIN
446         Restaurants r ON o.RestaurantID = r.RestaurantID
447     WHERE
448         o.DeliveryStatus = 'Pending';
```

This query enables real-time monitoring of pending orders, offering accurate information to ensure on-time deliveries.

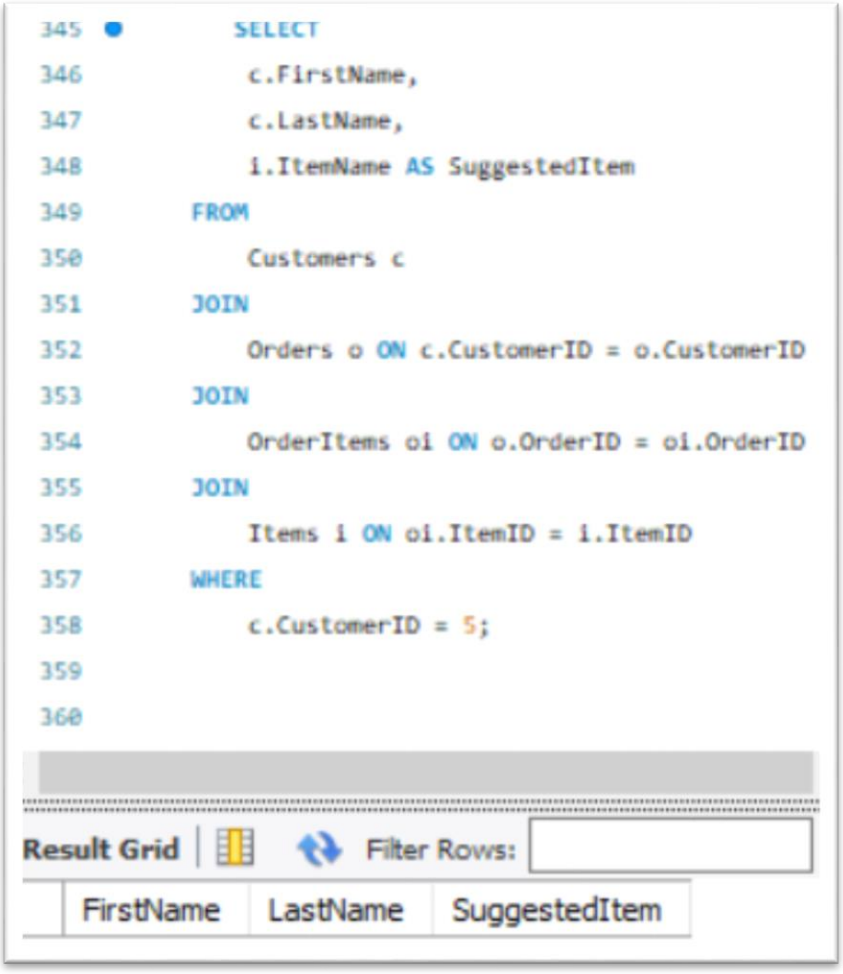
II. **Optimal Resource Allocation:** With effective inquiries, the company can make the best use of its resources. Managers, for instance, can determine the busiest order times and allocate additional drivers during certain periods. They can forecast demand patterns and change staffing levels accordingly by analyzing past data, which minimizes delivery delays and consumer displeasing.

```
450 •       SELECT
451         HOUR(OrderDate) AS OrderHour,
452         COUNT(OrderID) AS TotalOrders
453     FROM
454         Orders
455     WHERE
456         DeliveryStatus = 'Pending'
457     GROUP BY
458         OrderHour
459     ORDER BY
460         TotalOrders DESC
461     LIMIT 1;
```

This query analyzes the busiest hour for pending orders by extracting the hour from the **OrderDate** and counting the total number of pending orders within that hour. The result provides insights into the peak order time, aiding operational planning for efficient resource allocation and delivery management.

III. Personalized Customer Experience:

Effective inquiries give Kangaroo the ability to tailor their consumers experience. Through the Query they can make menu recommendations that are in line with specific consumer preferences by looking at previous orders and preferences. Increased client satisfaction and the chance of repeat business result from this.



By this way we can generate the recommended menu for the customers.

IV. Inventory Management and Demand Forecasting:

Accurate inventory management and demand forecasting are made possible by efficient queries. The company can determine which products are likely to be in demand on a certain day by running a query against the database to look at item sales trends.

```
360    SELECT
361        i.ItemName,
362        COUNT(oi.ItemID) AS TotalOrders
363    FROM
364        OrderItems oi
365    JOIN
366        Items i ON oi.ItemID = i.ItemID
367    JOIN
368        Orders o ON oi.OrderID = o.OrderID
369    WHERE
370        DATE(o.OrderDate) = CURDATE()
371    GROUP BY
372        i.ItemName
373    ORDER BY
```

Result Grid | Filter Rows:

ItemName	TotalOrders
Garlic Bread	1
Chicken Wings	1
Coke	1

V. Performance Evaluation and Incentives

Effective inquiries assist in evaluating and rewarding driver performance. Managers can perform queries to evaluate elements like order accuracy, delivery timings, and customer reviews. To motivate drivers and improve service quality, performance-based incentives, rewards, and training programs are decided upon using this data.

```sql
SELECT
    d.Name AS DriverName,
    AVG(o.DeliveryTime) AS AverageDeliveryTime,
    AVG(o.CustomerRating) AS AverageCustomerRating
FROM
    Drivers d
LEFT JOIN
    Orders o ON d.DriverID = o.DriverID
WHERE
    o.DeliveryStatus = 'Delivered'
GROUP BY
    d.Name;
```

This SQL code retrieves the average delivery time and average customer rating for each driver who has completed deliveries. It utilizes a LEFT JOIN between the "Drivers" and "Orders" tables to gather performance metrics, helping the business evaluate driver efficiency and customer satisfaction to optimize delivery operations and enhance service quality.

VI. Enhanced Marketing Strategies:

Effective queries support the creation of focused marketing strategies. Kangaroo can target specific customers market with marketing campaigns by looking at customers feedback, order history, and demographic data.

VII. Data-Driven Business Growth:

Data-driven company growth strategies are driven by effective SQL queries. Data insights might reveal underperforming cities, and business expansion possibilities.

```
374 •   SELECT
375         r.City,
376         COUNT(o.OrderID) AS TotalOrders
377     FROM
378         Restaurants r
379     LEFT JOIN
380         Orders o ON r.RestaurantID = o.RestaurantID
381     WHERE
382         DATE(o.OrderDate) BETWEEN '2023-08-23' AND '2023-09-15'
383     GROUP BY
384         r.City;
```

This SQL code provides insights into the distribution of orders among different cities based on the specified date range. By joining the "Restaurants" and "Orders" tables and grouping the results by city, the business can identify trends in order volume, optimize resource allocation, and make informed decisions to cater to varying demand across cities.

VIII. Reduced Workload and Improved Productivity:

Employees have less manual work to do because of efficient queries. Staff members don't have to spend time manually digging through spreadsheets or documents because they can instantly access the necessary data from the database. Employees can concentrate on tasks that provide value which results to the increase in total productivity.

```
387 •   SELECT
388         d.Name AS DriverName,
389         d.Email,
390         COUNT(o.OrderID) AS TotalOrders
391     FROM
392         Drivers d
393     LEFT JOIN
394         Orders o ON d.DriverID = o.DriverID
395     GROUP BY
396         d.DriverID;
```

Result Grid | Filter Rows: | Export:

DriverName	Email	TotalOrder
Michael Anderson	michael.anderson@example.com	1
Sophia Williams	sophia.williams@example.com	1
James Smith	james.smith@example.com	1
Emma Johnson	emma.johnson@example.com	1
David Brown	david.brown@example.com	1
Olivia Jones	olivia.jones@example.com	1
William Davis	william.davis@example.com	1

By joining the "Drivers" and "Orders" tables, it retrieves the driver's name, email, and the total number of orders they have completed. This information helps management assess each driver's contribution, allocate resources effectively, and identify high-performing drivers for recognition or incentives.

3.3.5 Data Accessibility:

Data accessibility is improved by the database's well-organized ERD. The division of data into separate tables, such as "Restaurants," "Drivers," and "Items," makes it simple to get particular data. This helps locating delivery drivers who are available, verifying restaurant information, and maintaining inventory.

```
386     -- Checking restaurant details
387 •   SELECT RestaurantName, Address
388     FROM Restaurants
389     WHERE RestaurantID = 9;
```

Result Grid | Filter Rows:

RestaurantName	Address
Healthy Bites	444 Green St, Bristol

3.3.6 Scalability and Flexibility: Scalability and flexibility are anticipated by the database design. The existing framework can be expanded with new eateries, goods, and drivers. The normalization of the architecture guarantees that updates may be done quickly, lowering the possibility of data abnormalities.

```
391     -- Adding a new restaurant
392 •   INSERT INTO Restaurants (RestaurantName, Address)
393     VALUES ('New Restaurant', '123 New Street');
394
395     -- Adding a new item to the menu
396 •   INSERT INTO Items (ItemName, ItemPrice, Category)
397     VALUES ('New Item', 8.99, 'Main Course');
```

3.3.7 Reporting and Analytics: Business management can monitor a number of KPIs, including the quantity of orders placed in each restaurant, the typical delivery time per driver, and the top products. This data-driven insight helps to find areas for improvement, optimize operations, and make wise decisions.

```
405     -- Number of orders per restaurant
406 •   SELECT r.RestaurantName, COUNT(o.OrderID) AS TotalOrders
407     FROM Restaurants r
408     LEFT JOIN Orders o ON r.RestaurantID = o.RestaurantID
409     GROUP BY r.RestaurantName;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

RestaurantName	TotalOrders
Delicious Diner	1
Tasty Tavern	1
Savory Eats	1
Yummy Cafe	1
Flavorful Bistro	1
Taste of Edinburgh	1
Mexican Grill	1

3.3.8 Security and Access Control: The database design includes security measures to protect sensitive data. Access controls can be enforced to ensure that only authorized personnel have access to certain tables or information. For example, access to financial data or customer personal information can be restricted to specific roles, safeguarding against data breaches and unauthorized usage.

```
-- Restricting access to financial data
SELECT DriverID, Name, Salary
FROM Drivers
WHERE DriverID = [AuthorizedManagerID];

-- Restricting access to financial data
SELECT DriverID, Name, Salary
FROM Drivers
WHERE DriverID = [AuthorizedManagerID];
```

The database design of Kangaroo food delivery company effortlessly integrates with the business environment by converting innate connections into organized tables, fostering data integrity, and facilitating effective querying and reporting.

Conclusion

The meticulous database design and implementation undertaken in this assignment are integral to the efficient functioning and growth of the Kangaroo Food Delivery Company. Task 1 established a solid foundation by creating interconnected tables for customers, drivers, restaurants, and orders. Task 2 further enriched this framework with relevant data. Task 3, however, proved pivotal, showcasing how optimized queries can transform raw data into actionable insights. Real-time tracking, personalized customer experiences, resource allocation optimization, and data-driven growth strategies are just a few benefits realized. Moreover, the structured design facilitates easy scalability, informed decision-making through reporting, and robust security. Ultimately, this well-crafted database positions Kangaroo for enhanced operational efficiency, customer satisfaction, and strategic competitiveness in the food delivery industry.