INTRODUCTION TO DATABASES

Class Assesment

X23388978
ADRIÁN TARÍN MARTÍNEZ

Contents

Part 1: Conceptual Design

1. Suggest a situation where you can use a database to manage and record daily transactions.

Situation: Online Food Delivery System

In an online food delivery system, a robust database is crucial to efficiently manage and record daily transactions. It ensures that customer orders are processed seamlessly, deliveries are managed effectively, and restaurants can track their menu and sales performance.

Customer Management:

The Customer table stores essential customer information, including contact details, addresses, and dates of birth. By analyzing customer demographics, the business can personalize its marketing strategies. For example, offering discounts for birthdays or creating loyalty programs to increase customer retention.

Order Management:

The Order table captures transaction details such as the order date, customer information, and the assigned courier. This ensures traceability and allows the system to monitor trends like peak ordering times or preferred delivery hours.

Menu and Sales Tracking:

The Menu table records details about the food items offered by restaurants, including prices and availability. Combined with the OrderDetails table, which tracks quantities and calculates totals for each order item, this setup allows for detailed sales analytics, such as identifying best-selling items.

Courier and Delivery Operations:

The Courier and Vehicle tables manage delivery logistics. Each courier is linked to a vehicle, ensuring proper resource allocation. This setup can be used to track delivery times and optimize routes, enhancing overall customer satisfaction.

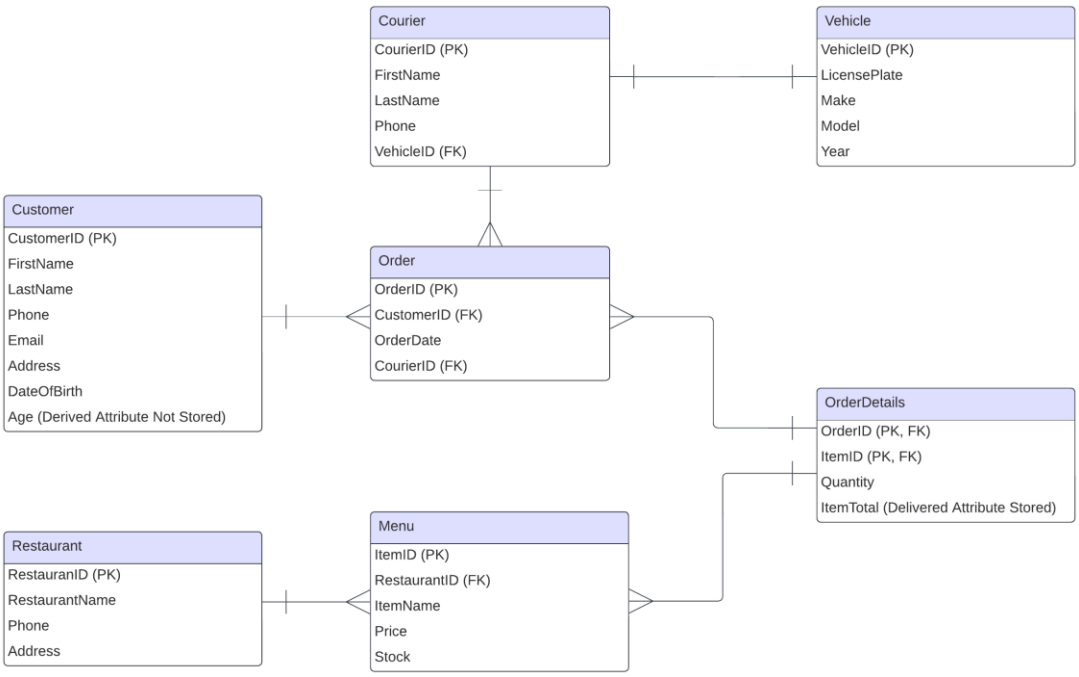
Business Insights:

With a structure like this, the database can generate comprehensive reports as:

- Monthly Sales Analysis: Using Order and OrderDetails, the total sales and items sold for each restaurant can be calculated.
- Top Couriers: Using order assignments from the Courier table, performance metrics like the number of deliveries completed by each courier can be derived.

By integrating all these tables, the system ensures data consistency and accuracy, making it a powerful tool for day-to-day operations and long-term strategic planning. The database not only records transactions but also provides actionable insights, making it invaluable for managing an online food delivery business.

2. Draw a suitable ER diagram for your database.



ER Diagram.

The ER Diagram for the Online Food Delivery System illustrates the core entities and their relationships:

Entities:

- Customer, Restaurant, Menu, Courier, Vehicle, Order, and OrderDetails.

Relationships:

- One-to-Many:
  - Customers place multiple orders.
  - Restaurants have multiple menu items.
  - Couriers deliver multiple orders.
- Many-to-Many:
  - Orders and Menu items connect via OrderDetails.
- One-to-One:
  - Each courier is assigned a unique vehicle and vice versa.

Derived Attributes:

- Age is a derived attribute in the Customer entity, is not stored in the table to avoid redundancy and resource consumption, but can be calculated dynamically when needed from DateOfBirth using the formula:

  `TIMESTAMPDIFF(YEAR, DateOfBirth, CURDATE()).`

- ItemTotal is a derived attribute in the OrderDetails entity, computed as Quantity * Price and stored to preserve historical data. This calculation is automated through a trigger.

This diagram provides a conceptual overview of the database's structure and relationships, setting the foundation for the relational model and ensuring data consistency and efficient operations.

3. Convert the conceptual design into a relational model. Make sure that the tables are in a 3rd normal form.

Relational Model for OnlineFoodDeliverySystem Database

Customer Table:

- Attributes:
    - CustomerID (Primary Key, INT, Auto Increment)
    - FirstName (VARCHAR(100), Not Null)
    - LastName (VARCHAR(100), Not Null)
    - Phone (VARCHAR(15), Not Null)
    - Email (VARCHAR(100), Not Null)
    - Address (TEXT, Not Null)
    - DateOfBirth (DATE, Not Null)
    - Derived Attribute:

Age could be calculated dynamically as:

```
TIMESTAMPDIFF(YEAR, DateOfBirth, CURDATE())
```

Restaurant Table:

- Attributes:
    - RestaurantID (Primary Key, INT, Auto Increment)
    - RestaurantName (VARCHAR(100), Not Null)
    - Phone (VARCHAR(15), Not Null)
    - Address (TEXT, Not Null)

Menu Table:

- Attributes:
    - ItemID (Primary Key, INT, Auto Increment)
    - RestaurantID (Foreign Key, INT, References Restaurant(RestaurantID) ON DELETE CASCADE)
    - ItemName (VARCHAR(100), Not Null)
    - Price (DECIMAL(10, 2), Not Null)

Vehicle Table:

- Attributes:
    - VehicleID (Primary Key, INT, Auto Increment)
    - LicensePlate (VARCHAR(20), Not Null)

- Make (VARCHAR(50), Not Null)
- Model (VARCHAR(50), Not Null)
- Year (INT, Not Null)

Courier Table:

- Attributes:
  - CourierID (Primary Key, INT, Auto Increment)
  - FirstName (VARCHAR(100), Not Null)
  - LastName (VARCHAR(100), Not Null)
  - Phone (VARCHAR(15), Not Null)
  - VehicleID (Unique, Foreign Key, INT, References Vehicle(VehicleID) ON DELETE CASCADE)

Order Table:

Attributes:

- OrderID (Primary Key, INT, Auto Increment)
  - CustomerID (Foreign Key, INT, References Customer(CustomerID) ON DELETE CASCADE)
  - CourierID (Foreign Key, INT, References Courier(CourierID) ON DELETE CASCADE)
  - OrderDate (DATETIME, Not Null)

OrderDetails Table:

- Attributes:
  - OrderID (Composite Primary Key, Foreign Key, INT, References Order(OrderID) ON DELETE CASCADE)
  - ItemID (Composite Primary Key, Foreign Key, INT, References Menu(ItemID) ON DELETE CASCADE)
  - Quantity (INT, Not Null)
  - ItemTotal (DECIMAL(10, 2), Derived from Quantity * Menu.Price)
- Trigger:

Populates ItemTotal using the following trigger logic:

```
DELIMITER $$

CREATE TRIGGER CalculateItemTotal

BEFORE INSERT ON OrderDetails

FOR EACH ROW

BEGIN

    DECLARE itemPrice DECIMAL(10, 2);

    SELECT Price INTO itemPrice

    FROM Menu

    WHERE ItemID = NEW.ItemID;

    SET NEW.ItemTotal = NEW.Quantity * itemPrice;
```

```
END $$
DELIMITER ;
```

Relationships:

- One-to-Many Relationships:
  - Restaurant ↔ Menu (A restaurant has many menu items).
  - Customer ↔ Order (A customer can place many orders).
  - Courier ↔ Order (A courier can deliver many orders).
- Many-to-Many Relationship:
  - Order ↔ Menu via OrderDetails (An order can include many items, and an item can belong to multiple orders).
- One-to-One Relationship:
  - Courier ↔ Vehicle (Each courier is assigned one unique vehicle).

This model adheres to 3rd Normal Form (3NF):

- Each table has a Primary Key.
- Non-key attributes are functionally dependent on the Primary Key.
- No transitive dependencies exist between attributes.

Part 2: Physical Design

1. Create the corresponding database using DDL

```
CREATE DATABASE OnlineFoodDeliverySystem;
USE OnlineFoodDeliverySystem;
```

2. Create all the necessary tables identified above using DDL

```
-- Create the Customer table
CREATE TABLE Customer (
    CustomerID INT AUTO_INCREMENT PRIMARY KEY,
    FirstName VARCHAR(100) NOT NULL,
    LastName VARCHAR(100) NOT NULL,
    Phone VARCHAR(15) NOT NULL,
    Email VARCHAR(100) NOT NULL,
    Address TEXT NOT NULL,
    DateOfBirth DATE NOT NULL
```

```
);

-- To calculate the Age as derived attribute in queries when needed,
eliminating the need to store it explicitly.
SELECT
    CustomerID,
    FirstName,
    LastName,
    TIMESTAMPDIFF(YEAR, DateOfBirth, CURDATE()) AS Age
FROM Customer;

-- Create the Restaurant table
CREATE TABLE Restaurant (
    RestaurantID INT AUTO_INCREMENT PRIMARY KEY,
    RestaurantName VARCHAR(100) NOT NULL,
    Phone VARCHAR(15) NOT NULL,
    Address TEXT NOT NULL
);

-- Create the Menu table
CREATE TABLE Menu (
    ItemID INT AUTO_INCREMENT PRIMARY KEY,
    RestaurantID INT NOT NULL,
    ItemName VARCHAR(100) NOT NULL,
    Price DECIMAL(10, 2) NOT NULL,
    FOREIGN KEY (RestaurantID) REFERENCES Restaurant(RestaurantID) ON
DELETE CASCADE
);

-- Create the Vehicle table
CREATE TABLE Vehicle (
    VehicleID INT AUTO_INCREMENT PRIMARY KEY,
    LicensePlate VARCHAR(20) NOT NULL,
    Make VARCHAR(50) NOT NULL,
    Model VARCHAR(50) NOT NULL,
```

```
    Year INT NOT NULL
);

-- Create the Courier table
CREATE TABLE Courier (
    CourierID INT AUTO_INCREMENT PRIMARY KEY,
    FirstName VARCHAR(100) NOT NULL,
    LastName VARCHAR(100) NOT NULL,
    Phone VARCHAR(15) NOT NULL,
    VehicleID INT UNIQUE NOT NULL,
    FOREIGN KEY (VehicleID) REFERENCES Vehicle(VehicleID) ON DELETE
CASCADE
);

-- Create the Order table
CREATE TABLE `Order` (
    OrderID INT AUTO_INCREMENT PRIMARY KEY,
    CustomerID INT NOT NULL,
    CourierID INT NOT NULL,
    OrderDate DATETIME NOT NULL,
    FOREIGN KEY (CustomerID) REFERENCES Customer(CustomerID) ON DELETE
CASCADE,
    FOREIGN KEY (CourierID) REFERENCES Courier(CourierID) ON DELETE
CASCADE
);

-- Create the OrderDetails table
CREATE TABLE OrderDetails (
    OrderID INT NOT NULL,
    ItemID INT NOT NULL,
    Quantity INT NOT NULL,
      ItemTotal DECIMAL(10, 2),
    PRIMARY KEY (OrderID, ItemID),
    FOREIGN KEY (OrderID) REFERENCES `Order`(OrderID) ON DELETE
CASCADE,
    FOREIGN KEY (ItemID) REFERENCES Menu(ItemID) ON DELETE CASCADE
```

```
);


-- Trigger to populate OrderDetails.ItemTotal
DELIMITER $$
CREATE TRIGGER CalculateItemTotal
BEFORE INSERT ON OrderDetails
FOR EACH ROW
BEGIN
    DECLARE itemPrice DECIMAL(10, 2);
    SELECT Price INTO itemPrice
    FROM Menu
    WHERE ItemID = NEW.ItemID;
    SET NEW.ItemTotal = NEW.Quantity * itemPrice;
END $$
DELIMITER ;
```

3. Populate at least three of your tables with some data using DML (insert into statement)

```
-- Insert data into the Customer table
INSERT INTO Customer (FirstName, LastName, Phone, Email, Address,
DateOfBirth)
VALUES
('Walter', 'White', '1234567890', 'heisenberg10@gmail.com', '123 Elm
Street', '1961-1-1'),
('Jesse', 'Pinkman', '0987654321', 'kaptain_cook@hotmail.com', '456
Maple Avenue', '1986-1-1'),
('Gus', 'Fring', '1122334455', 'pollos.hermanos@yahoo.com', '789 Oak
Lane', '1968-1-1');


-- Insert data into the Restaurant table
INSERT INTO Restaurant (RestaurantName, Phone, Address)
VALUES
('Pizza Paradise', '9876543210', '321 Main Street'),
('Burger Haven', '8765432109', '654 Second Avenue'),
('Sushi World', '7654321098', '987 Third Boulevard');
```

```
-- Insert data into the Menu table
INSERT INTO Menu (RestaurantID, ItemName, Price)
VALUES
(1, 'Pepperoni Pizza', 12.99),
(1, 'Margherita Pizza', 10.99),
(2, 'Classic Burger', 8.99),
(2, 'Cheese Fries', 4.99),
(3, 'California Roll', 9.99),
(3, 'Tempura', 7.99);

-- Insert data into the Vehicle table
INSERT INTO Vehicle (LicensePlate, Make, Model, Year)
VALUES
('ABC123', 'Toyota', 'Corolla', 2020),
('XYZ789', 'Honda', 'Civic', 2021),
('LMN456', 'Yamaha', 'R3', 2019);

-- Insert data into the Courier table
INSERT INTO Courier (FirstName, LastName, Phone, VehicleID)
VALUES
('Sam', 'Brown', '5551234567', 1),
('Ella', 'Green', '5559876543', 2),
('Tom', 'White', '5555678910', 3);

-- Insert data into the Order table
INSERT INTO `Order` (CustomerID, CourierID, OrderDate)
VALUES
(1, 1, '2024-11-25'),
(2, 2, '2024-11-26'),
(3, 3, '2024-11-27');

-- Insert data into the OrderDetails table
INSERT INTO OrderDetails (OrderID, ItemID, Quantity)
VALUES
(1, 1, 2), -- 2 Pepperoni Pizzas
```

```
(1, 2, 1), -- 1 Margherita Pizza
(2, 3, 3), -- 3 Classic Burgers
(2, 4, 2), -- 2 Cheese Fries
(3, 5, 4); -- 4 California Rolls
```

4. Populate your database with a large data set representing a one-year transaction (01/01/2022 - 31/12/2022) on each table. (Use online data generators such as Mockaroo or generate data to generate synthetic data.)

```
-- Load data from CSV to table Customer
LOAD DATA INFILE 'C:\\ProgramData\\MySQL\\MySQL Server
9.0\\Uploads\\Customer.csv'
INTO TABLE Customer
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n'
IGNORE 1 ROWS
(CustomerID, FirstName, LastName, Phone, Email, Address, DateOfBirth);

-- Load data from CSV to Restaurant table
LOAD DATA INFILE 'C:\\ProgramData\\MySQL\\MySQL Server
9.0\\Uploads\\Restaurant.csv'
INTO TABLE Restaurant
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n'
IGNORE 1 ROWS
(RestaurantName, Phone, Address);

-- Load data from CSV to Menu table
LOAD DATA INFILE 'C:\\ProgramData\\MySQL\\MySQL Server
9.0\\Uploads\\Menu.csv'
INTO TABLE Menu
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n'
IGNORE 1 ROWS
```

```
(RestaurantID, ItemName, Price);


-- Load data from CSV to Vehicle table
LOAD DATA INFILE 'C:\\ProgramData\\MySQL\\MySQL Server
9.0\\Uploads\\Vehicle.csv'
INTO TABLE Vehicle
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n'
IGNORE 1 ROWS
(VehicleID, LicensePlate, Make, Model, Year);


-- Load data from CSV to Courier table
LOAD DATA INFILE 'C:/ProgramData/MySQL/MySQL Server
9.0/Uploads/Courier.csv'
INTO TABLE Courier
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n'
IGNORE 1 ROWS
(CourierID, FirstName, LastName, Phone, VehicleID);


-- Load data from CSV to Order table
LOAD DATA INFILE 'C:/ProgramData/MySQL/MySQL Server
9.0/Uploads/Order.csv'
INTO TABLE `Order`
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n'
IGNORE 1 ROWS
(OrderID, CustomerID, CourierID, OrderDate);


-- Load data from CSV to OrderDetails table
LOAD DATA INFILE 'C:/ProgramData/MySQL/MySQL Server
9.0/Uploads/OrderDetails.csv'
INTO TABLE OrderDetails
```

```
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n'
IGNORE 1 ROWS
(OrderID, ItemID, Quantity);
```

Part 3: SQL Statements

1. Update a column in a table of your choice (for example, increase the salary of employees by 5%).

```
-- Increase the price of all the items by 1€.
UPDATE Menu
SET Price = Price + 1;
```

2. Delete a column in a table of your choice.

```
ALTER TABLE Vehicle
DROP COLUMN Model;
```

3. Show the total number of transactions your database is storing and, depending on your database, the most sold/listed item or customer with the highest number of purchases.

```
-- Total number of transactions:
SELECT COUNT(*) AS TotalTransactions
FROM `Order`;


-- Customer with more orders:
SELECT c.CustomerID, c.FirstName, c.LastName, COUNT(o.OrderID) AS
NumberOfOrders
FROM `Order` o
JOIN Customer c ON o.CustomerID = c.CustomerID
GROUP BY c.CustomerID
ORDER BY NumberOfOrders DESC
LIMIT 1;
```

4. Write a query statement that includes “Order by” and “Group by”.

-- Top 10 Costumers with more Total Amount Spent:

```
SELECT c.CustomerID, c.FirstName, c.LastName, SUM(od.ItemTotal) AS
TotalAmountSpent

FROM `Order` o

JOIN OrderDetails od ON o.OrderID = od.OrderID

JOIN Customer c ON o.CustomerID = c.CustomerID

GROUP BY c.CustomerID

ORDER BY TotalAmountSpent DESC

LIMIT 10;
```

5. Write a query statement that uses pattern matching (example: customer living in a given street, number of Johns, people with today’s birthday…).

```
-- Find all customers with the first name "Matias".

SELECT COUNT(*) AS NumberOfMatias

FROM Customer

WHERE FirstName LIKE 'Matias';
```

6. Show information from three tables based on criteria of your choice (hint: join).

```
-- Retrieve the top 10 orders by total spent in 2022, showing data
from 4 tables (Customer, Order, OrderDetails, Menu).

SELECT

    c.CustomerID,

    c.FirstName AS CustomerFirstName,

    c.LastName AS CustomerLastName,

    o.OrderID,

    o.OrderDate,

    SUM(od.ItemTotal) AS TotalSpent,

    SUM(od.Quantity) AS NumberOfProducts,

    GROUP_CONCAT(m.ItemName) AS ProductsPurchased

FROM

    Customer c

JOIN
```

```
    `Order` o ON c.CustomerID = o.CustomerID
JOIN
    OrderDetails od ON o.OrderID = od.OrderID
JOIN
    Menu m ON od.ItemID = m.ItemID
WHERE
    YEAR(o.OrderDate) = 2022
GROUP BY
    c.CustomerID, o.OrderID
ORDER BY
    TotalSpent DESC
LIMIT 10;
```

7. Create a view that includes information from the most frequent seven transactions (customer names or most sold items …).

```
-- View for Most Frequent Customers (Top 7 customers who placed the
most orders)
CREATE VIEW MostFrequentCustomers AS
SELECT
    c.FirstName,
    c.LastName,
    COUNT(o.OrderID) AS TotalOrders
FROM
    Customer c
JOIN
    `Order` o ON c.CustomerID = o.CustomerID
GROUP BY
    c.CustomerID
ORDER BY
    TotalOrders DESC
LIMIT 7;

-- View for Most Frequent Items.
CREATE VIEW MostFrequentItems AS
SELECT
```

```
      m.ItemID,
    m.ItemName,
    COUNT(od.ItemID) AS TotalSold
FROM
    Menu m
JOIN
    OrderDetails od ON m.ItemID = od.ItemID
GROUP BY
    m.ItemID
ORDER BY
    TotalSold DESC
LIMIT 7;
```

8. Create a set of queries that summarises the annual transactions. For example, if your transaction table is about selling product, you can create queries that:

• Shows the total number of transactions with corresponding details every month,

• Shows customer purchase value per month,

• Shows name of product and number sold each month

```
-- Total Number of Transactions per Month
SELECT
    MONTH(OrderDate) AS Month,
    YEAR(OrderDate) AS Year,
    COUNT(OrderID) AS TotalTransactions
FROM `Order`
WHERE YEAR(OrderDate) = 2022
GROUP BY YEAR(OrderDate), MONTH(OrderDate)
ORDER BY Year, Month;

-- Customer Purchase Value Per Month
SELECT
    c.CustomerID,
    CONCAT(c.FirstName, ' ', c.LastName) AS CustomerName,
    MONTH(o.OrderDate) AS Month,
```

```
    YEAR(o.OrderDate) AS Year,
    SUM(od.ItemTotal) AS TotalPurchase
FROM `Order` o
JOIN Customer c ON o.CustomerID = c.CustomerID
JOIN OrderDetails od ON o.OrderID = od.OrderID
GROUP BY c.CustomerID, YEAR(o.OrderDate), MONTH(o.OrderDate)
ORDER BY Year, Month, CustomerName;

-- Name of Product and Number Sold Each Month
SELECT
      MONTH(o.OrderDate) AS Month,
    YEAR(o.OrderDate) AS Year,
      m.ItemID,
    m.ItemName,
    SUM(od.Quantity) AS TotalQuantitySold
FROM OrderDetails od
JOIN Menu m ON od.ItemID = m.ItemID
JOIN `Order` o ON od.OrderID = o.OrderID
GROUP BY m.ItemID, YEAR(o.OrderDate), MONTH(o.OrderDate)
ORDER BY Year, Month, TotalQuantitySold DESC;
```