DBMS Problem Statements - Code and Output

Problem 1: Student Course Registration System

DDL Statements:

```
-- Students table
CREATE TABLE Students (
   StudentID INT PRIMARY KEY,
   Name VARCHAR(100) NOT NULL,
   Email VARCHAR(100) UNIQUE NOT NULL
);

-- Courses table
CREATE TABLE Courses (
   CourseID INT PRIMARY KEY,
   CourseName VARCHAR(100) NOT NULL,
   Credits INT NOT NULL
);

-- Enrollments table
CREATE TABLE Enrollments (
   EnrollmentID INT PRIMARY KEY,
   StudentID INT,
   CourseID INT,
   FOREIGN KEY (StudentID) REFERENCES Students(StudentID),
   FOREIGN KEY (CourseID) REFERENCES Courses(CourseID)
);
```

Normalization to 3NF:

```
Unnormalized:
   StudentCourse(StudentID, Name, Email, CourseID, CourseName, Credits)

1NF:
   Separate into individual entities to remove multivalued fields.
   Student(StudentID, Name, Email)
   Course(CourseID, CourseName, Credits)
```

Enrollment(StudentID, CourseID)

2NF:
Ensure no partial dependencies. Already satisfied.

3NF:
Ensure no transitive dependencies. Already satisfied.

Sample Data:

```
INSERT INTO Students VALUES (1, 'Alice', 'alice@example.com');
INSERT INTO Students VALUES (2, 'Bob', 'bob@example.com');

INSERT INTO Courses VALUES (101, 'DBMS', 4);
INSERT INTO Courses VALUES (102, 'OS', 3);

INSERT INTO Enrollments VALUES (1, 1, 101);
INSERT INTO Enrollments VALUES (2, 2, 102);
```

Sample Output Query:

```
SELECT s.Name AS Student, c.CourseName AS Course
FROM Students s
JOIN Enrollments e ON s.StudentID = e.StudentID
JOIN Courses c ON e.CourseID = c.CourseID;
```

Expected Output:

```
| Student | Course |
|---------|--------|
| Alice   | DBMS   |
| Bob     | OS     |
```

Problem 2: Online Shopping System

DDL Statements:

```
CREATE TABLE Customers (
   CustomerID INT PRIMARY KEY,
   Name VARCHAR(100) NOT NULL,
   Email VARCHAR(100) UNIQUE NOT NULL
);

CREATE TABLE Products (
   ProductID INT PRIMARY KEY,
   ProductName VARCHAR(100) NOT NULL,
   Price DECIMAL(10,2) NOT NULL
);

CREATE TABLE Orders (
   OrderID INT PRIMARY KEY,
   CustomerID INT,
   ProductID INT,
   Quantity INT NOT NULL,
   FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID),
   FOREIGN KEY (ProductID) REFERENCES Products(ProductID)
);
```

Normalization to 3NF:

Unnormalized:
   OrderData(CustomerID, Name, Email, ProductID, ProductName, Price, Quantity)

1NF:
   Remove repeating groups. Break into:
   Customers(CustomerID, Name, Email)
   Products(ProductID, ProductName, Price)
   Orders(OrderID, CustomerID, ProductID, Quantity)

2NF:
   Remove partial dependencies (all non-key fields depend on full primary key)

3NF:
   Remove transitive dependencies (none remain)

Sample Data:

```
INSERT INTO Customers VALUES (1, 'John', 'john@example.com');
INSERT INTO Products VALUES (100, 'Laptop', 800.00);
INSERT INTO Orders VALUES (1, 1, 100, 2);
```

Sample Output Query:

```
SELECT c.Name AS Customer, p.ProductName, o.Quantity
FROM Customers c
JOIN Orders o ON c.CustomerID = o.CustomerID
JOIN Products p ON o.ProductID = p.ProductID;
```

Expected Output:

```
| Customer | ProductName | Quantity |
|----------|-------------|----------|
| John     | Laptop      | 2        |
```

Problem 3: Library Management System

DDL Statements:

```
CREATE TABLE Books (
  BookID INT PRIMARY KEY,
  Title VARCHAR(100),
  Author VARCHAR(100)
);

CREATE TABLE Members (
  MemberID INT PRIMARY KEY,
  Name VARCHAR(100)
);

CREATE TABLE BorrowedBooks (
  BorrowID INT PRIMARY KEY,
  MemberID INT,
  BookID INT,
  BorrowDate DATE,
  FOREIGN KEY (MemberID) REFERENCES Members(MemberID),
  FOREIGN KEY (BookID) REFERENCES Books(BookID)
);
```

Normalization to 3NF:

Unnormalized:
  BookLoans(MemberID, Name, BookID, Title, Author, BorrowDate)

1NF:
  Books(BookID, Title, Author)
  Members(MemberID, Name)
  BorrowedBooks(BorrowID, MemberID, BookID, BorrowDate)

2NF:
  All fields depend on entire key

3NF:
  No transitive dependencies remain

Sample Data:

```
INSERT INTO Books VALUES (1, 'DBMS Concepts', 'Korth');
INSERT INTO Members VALUES (1, 'Alice');
INSERT INTO BorrowedBooks VALUES (1, 1, 1, '2024-01-01');
```

Sample Output Query:

```
SELECT m.Name, b.Title, bb.BorrowDate
FROM Members m
JOIN BorrowedBooks bb ON m.MemberID = bb.MemberID
JOIN Books b ON bb.BookID = b.BookID;
```

Expected Output:

```
| Name  | Title         | BorrowDate |
|-------|---------------|------------|
| Alice | DBMS Concepts | 2024-01-01 |
```

Problem 4: Employee Payroll System

DDL Statements:

```
CREATE TABLE Employees (
   EmployeeID INT PRIMARY KEY,
   Name VARCHAR(100),
   Department VARCHAR(100)
);

CREATE TABLE Salaries (
   SalaryID INT PRIMARY KEY,
   EmployeeID INT,
   BasicPay DECIMAL(10, 2),
   Deductions DECIMAL(10, 2),
   NetPay DECIMAL(10, 2),
   FOREIGN KEY (EmployeeID) REFERENCES Employees(EmployeeID)
);
```

Normalization to 3NF:

Unnormalized:
   EmployeeSalary(EmployeeID, Name, Department, BasicPay, Deductions, NetPay)

1NF:
   Employees(EmployeeID, Name, Department)
   Salaries(SalaryID, EmployeeID, BasicPay, Deductions, NetPay)

2NF:
   All fields depend on full key

3NF:
   No transitive dependencies

Sample Data:

```
INSERT INTO Employees VALUES (1, 'Bob', 'IT');
INSERT INTO Salaries VALUES (1, 1, 50000, 5000, 45000);
```

Sample Output Query:

```
SELECT e.Name, s.NetPay
FROM Employees e
JOIN Salaries s ON e.EmployeeID = s.EmployeeID;
```

Expected Output:

```
| Name | NetPay |
|------|--------|
| Bob  | 45000  |
```

Problem 5: Inventory Management System

DDL Statements:

```
CREATE TABLE Items (
  ItemID INT PRIMARY KEY,
  ItemName VARCHAR(100),
  Quantity INT
);

CREATE TABLE Suppliers (
  SupplierID INT PRIMARY KEY,
  SupplierName VARCHAR(100)
);

CREATE TABLE Supplies (
  SupplyID INT PRIMARY KEY,
  ItemID INT,
  SupplierID INT,
  SupplyDate DATE,
  FOREIGN KEY (ItemID) REFERENCES Items(ItemID),
  FOREIGN KEY (SupplierID) REFERENCES Suppliers(SupplierID)
);
```

Normalization to 3NF:

Unnormalized:
  SupplyRecord(ItemID, ItemName, Quantity, SupplierID, SupplierName, SupplyDate)

1NF:
  Items(ItemID, ItemName, Quantity)
  Suppliers(SupplierID, SupplierName)
  Supplies(SupplyID, ItemID, SupplierID, SupplyDate)

2NF:
  All attributes depend on the entire primary key

3NF:
  No transitive dependencies

Sample Data:

```
INSERT INTO Items VALUES (1, 'Monitor', 50);
INSERT INTO Suppliers VALUES (1, 'ABC Electronics');
INSERT INTO Supplies VALUES (1, 1, 1, '2024-02-01');
```

Sample Output Query:

```
SELECT i.ItemName, s.SupplierName, sp.SupplyDate
FROM Items i
JOIN Supplies sp ON i.ItemID = sp.ItemID
JOIN Suppliers s ON sp.SupplierID = s.SupplierID;
```

Expected Output:

```
| ItemName | SupplierName   | SupplyDate |
|----------|----------------|------------|
| Monitor  | ABC Electronics | 2024-02-01 |
```

Problem 6:

```
-- Problem 6: Course Registration System
CREATE TABLE Students (
    StudentID INT PRIMARY KEY,
    Name VARCHAR(100),
    Email VARCHAR(100)
);

CREATE TABLE Courses (
    CourseID INT PRIMARY KEY,
    CourseName VARCHAR(100),
    Credits INT
);

CREATE TABLE Registrations (
    RegistrationID INT PRIMARY KEY,
    StudentID INT,
    CourseID INT,
    FOREIGN KEY (StudentID) REFERENCES Students(StudentID),
    FOREIGN KEY (CourseID) REFERENCES Courses(CourseID)
);

ALTER TABLE Registrations ADD RegistrationDate DATE;
ALTER TABLE Courses MODIFY Credits INTEGER;
DROP TABLE Registrations;
```

Problem 7:

```
-- Problem 7: Online Order Management System
CREATE TABLE Customers (
    CustomerID INT PRIMARY KEY,
    Name VARCHAR(100),
    Email VARCHAR(100)
);

CREATE TABLE Orders (
    OrderID INT PRIMARY KEY,
    CustomerID INT,
    OrderDate DATE,
    FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)
);

CREATE TABLE Products (
    ProductID INT PRIMARY KEY,
    ProductName VARCHAR(100),
    Price DECIMAL(10, 2)
);

ALTER TABLE Orders ADD ShippingAddress VARCHAR(255);
ALTER TABLE Products RENAME COLUMN ProductName TO ProductDescription;
DROP TABLE Products;
```

Problem 8:

```
-- Problem 8: University Management System
CREATE TABLE Students (
    StudentID INT PRIMARY KEY,
    Name VARCHAR(100),
    DOB DATE
);

CREATE TABLE Courses (
    CourseID INT PRIMARY KEY,
    CourseName VARCHAR(100),
    Credits INT
);

CREATE TABLE Enrollments (
    EnrollmentID INT PRIMARY KEY,
    StudentID INT,
    CourseID INT,
    FOREIGN KEY (StudentID) REFERENCES Students(StudentID),
    FOREIGN KEY (CourseID) REFERENCES Courses(CourseID)
);

ALTER TABLE Enrollments ADD Grade CHAR(1);
ALTER TABLE Students MODIFY DOB DATE;
DROP TABLE Students;
```

Problem 9:

```
-- Problem 9: University Management Queries
SELECT * FROM Students WHERE StudentID > 1000;
SELECT * FROM Courses WHERE Credits >= 3;
SELECT * FROM Enrollments WHERE CourseID <> 101 AND StudentID < 50;
```

Problem 10:

```
-- Problem 10: Library Database Queries
SELECT * FROM Members WHERE Name LIKE '%John%';
SELECT * FROM Books WHERE Title LIKE 'Java%' OR Title LIKE '%Programming';
SELECT * FROM Members WHERE ActiveStatus = TRUE;
```

Problem 11:

```
-- Problem 11: Online Shopping System Queries
SELECT OrderID, Quantity * Price AS OrderAmount FROM Orders;
SELECT AVG(Price) AS AveragePrice FROM Products;
SELECT SUM(Quantity) AS TotalQuantity FROM Orders;
```

Problem 12:

```
-- Problem 12: Sales Data Queries
SELECT ProductID, SUM(SalesAmount) AS TotalSales FROM Sales GROUP BY ProductID;
SELECT ProductID, MAX(SalesAmount), MIN(SalesAmount), AVG(SalesAmount) FROM Sales GROUP BY ProductID;
SELECT ProductID, COUNT(*) AS SaleCount FROM Sales GROUP BY ProductID;
```

Problem 13:

```
-- Problem 13: Library Borrowing Queries
SELECT * FROM Borrowing WHERE BorrowDate < '2021-01-01';
SELECT BorrowID, DATEDIFF(ReturnDate, BorrowDate) AS Duration FROM Borrowing;
SELECT CURRENT_DATE(), CURRENT_TIME();
```

Problem 14:

```
-- Problem 14: Hospital Management System Complex Queries
SELECT DoctorID, COUNT(*) AS TotalAppointments FROM Appointments GROUP BY DoctorID HAVING COUNT(*) > 5;
SELECT PatientID FROM Appointments GROUP BY PatientID HAVING COUNT(DISTINCT DoctorID) > 1;
SELECT * FROM Patients WHERE Age > 30 AND EXISTS (
      SELECT 1 FROM Appointments WHERE Patients.PatientID = Appointments.PatientID AND AppointmentDate >=
DATE_SUB(CURDATE(), INTERVAL 6 MONTH)
);
```

Problem 15:

```
-- Problem 15: Set Operations on Customers
(SELECT * FROM Customers)
UNION
(SELECT * FROM VIP_Customers);

(SELECT * FROM Customers)
INTERSECT
(SELECT * FROM VIP_Customers);

(SELECT * FROM Customers)
EXCEPT
(SELECT * FROM VIP_Customers);
```

Problem 16:

```
-- Problem 16: Create View and Update
CREATE VIEW StudentCourseView AS
SELECT s.Name AS StudentName, c.CourseName
FROM Students s
JOIN Enrollments e ON s.StudentID = e.StudentID
JOIN Courses c ON c.CourseID = e.CourseID;

UPDATE Students SET Name = 'New Name' WHERE StudentID = 1;
```

Problem 17:

```
-- Problem 17: Sales View and Update Attempt
CREATE VIEW ProductSalesView AS
SELECT ProductID, SUM(SaleAmount) AS TotalSales FROM Sales GROUP BY ProductID;

-- Attempt to update (may fail if view is not updatable)
UPDATE ProductSalesView SET TotalSales = 2000 WHERE ProductID = 101;
```

Problem 18:

```
-- Problem 18: Appointments View
CREATE VIEW AppointmentDetails AS
SELECT a.AppointmentDate, d.Name AS DoctorName, p.Name AS PatientName
FROM Appointments a
JOIN Doctors d ON a.DoctorID = d.DoctorID
JOIN Patients p ON a.PatientID = p.PatientID;
```

Problem 19:

```
-- Problem 19: EmployeeSalaryView and Update
CREATE VIEW EmployeeSalaryView AS
SELECT EmployeeName, Salary FROM Employees;

UPDATE EmployeeSalaryView SET Salary = 50000 WHERE EmployeeName = 'John Doe';
```

Problem 20:

```
-- Problem 20: SaleAmount View and Update
CREATE VIEW HighValueSales AS
SELECT * FROM Sales WHERE SaleAmount > 1000;

UPDATE HighValueSales SET SaleAmount = 900 WHERE ProductID = 1;
```

Problem 21:

```
-- Problem 21: Total Sales per Product View
CREATE VIEW TotalSalesPerProduct AS
SELECT ProductID, SUM(SaleAmount) AS TotalSales FROM Sales GROUP BY ProductID;

-- Attempt to update
UPDATE TotalSalesPerProduct SET TotalSales = 10000 WHERE ProductID = 1;

DROP VIEW EmployeeSalaryView;
```

Problem 22:

```
-- Problem 22: E-commerce Order View
CREATE VIEW OrderDetails AS
SELECT c.Name AS CustomerName, p.ProductName, o.OrderDate
FROM Orders o
JOIN Customers c ON o.CustomerID = c.CustomerID
JOIN Products p ON o.ProductID = p.ProductID;

UPDATE OrderDetails SET OrderDate = '2023-01-01' WHERE CustomerName = 'Alice';
```

Problem 23:

```
-- Problem 23: Insert Employee Procedure
CREATE OR REPLACE PROCEDURE InsertEmployee (
    p_EmpID IN INT,
    p_Name IN VARCHAR,
    p_Salary IN DECIMAL
) AS
BEGIN
```

```
    INSERT INTO Employees (EmployeeID, Name, Salary)
    VALUES (p_EmpID, p_Name, p_Salary);
END;
```

Problem 24:

```
-- Problem 24: Update Salary Procedure
CREATE OR REPLACE PROCEDURE UpdateSalary (
    p_EmpID IN INT,
    p_NewSalary IN DECIMAL
) AS
BEGIN
    UPDATE Employees SET Salary = p_NewSalary WHERE EmployeeID = p_EmpID;
END;
```

Problem 25:

```
-- Problem 25: Delete Employee Procedure
CREATE OR REPLACE PROCEDURE DeleteEmployee (
    p_EmpID IN INT
) AS
BEGIN
    DELETE FROM Employees WHERE EmployeeID = p_EmpID;
END;
```

Problem 26:

```
-- Problem 26: Statement-level Trigger for Insert Logging
CREATE OR REPLACE TRIGGER LogInsertEmployee
AFTER INSERT ON Employees
BEGIN
    INSERT INTO LogTable (LogDate, Message)
    VALUES (SYSDATE, 'New employee inserted. Total count: ' || (SELECT COUNT(*) FROM Employees));
END;
```

Problem 27:

```
-- Problem 27: Statement-level Trigger for Deletion Logging
CREATE OR REPLACE TRIGGER LogDeleteEmployee
AFTER DELETE ON Employees
BEGIN
    INSERT INTO DeletionLog (LogDate, DeletedCount)
    VALUES (SYSDATE, SQL%ROWCOUNT);
END;
```

Problem 28:

```
-- Problem 28: Row-level Trigger for Salary Updates
CREATE OR REPLACE TRIGGER SalaryUpdateLog
AFTER UPDATE OF Salary ON Employees
FOR EACH ROW
BEGIN
    INSERT INTO SalaryHistory (EmployeeID, OldSalary, NewSalary, ChangeDate)
    VALUES (:OLD.EmployeeID, :OLD.Salary, :NEW.Salary, SYSDATE);
```

```
END;
```

Problem 29:

```
-- Problem 29: PL/SQL Block with Implicit Cursor
DECLARE
    v_EmpID Employees.EmployeeID%TYPE := 101;
    v_Name Employees.Name%TYPE;
    v_Salary Employees.Salary%TYPE;
BEGIN
    SELECT Name, Salary INTO v_Name, v_Salary FROM Employees WHERE EmployeeID = v_EmpID;
    DBMS_OUTPUT.PUT_LINE('ID: ' || v_EmpID || ', Name: ' || v_Name || ', Salary: ' || v_Salary);
END;
```

Problem 30:

```
-- Problem 30: PL/SQL Block with Explicit Cursor
DECLARE
    CURSOR emp_cursor IS SELECT EmployeeID, Name, Salary FROM Employees;
    v_EmpID Employees.EmployeeID%TYPE;
    v_Name Employees.Name%TYPE;
    v_Salary Employees.Salary%TYPE;
BEGIN
    OPEN emp_cursor;
    LOOP
        FETCH emp_cursor INTO v_EmpID, v_Name, v_Salary;
        EXIT WHEN emp_cursor%NOTFOUND;
        DBMS_OUTPUT.PUT_LINE('ID: ' || v_EmpID || ', Name: ' || v_Name || ', Salary: ' || v_Salary);
    END LOOP;
    CLOSE emp_cursor;
END;
```

1. Design and implement a database for a student course registration system. The system should manage Students, Courses, and Enrollments. • Use DDL statements to create the tables. • Apply normalization (up to 3NF) and explain each step. • Include constraints like primary keys, foreign keys, and NOT NULL where applicable.

```
create database Student_course_registration_system;

use Student_course_registration_system;

mysql> create table student_details ( Id int Not Null , Name Varchar(30), Course Varchar(30));

Query OK, 0 rows affected (0.08 sec)


mysql> create table Course_details ( C_Id int Not Null , Course_name Varchar(30));

Query OK, 0 rows affected (0.03 sec)


mysql> create table Enrollment_details ( Id int Not Null , C_Id int Not Null );

Query OK, 0 rows affected (0.07 sec)


mysql> desc student_details;

+--------+-------------+------+-----+---------+-------+
| Field  | Type        | Null | Key | Default | Extra |
+--------+-------------+------+-----+---------+-------+
| Id     | int         | NO   |     | NULL    |       |
| Name   | varchar(30) | YES  |     | NULL    |       |
| Course | varchar(30) | YES  |     | NULL    |       |
+--------+-------------+------+-----+---------+-------+

3 rows in set (0.00 sec)


mysql> desc Course_details;

+-------------+-------------+------+-----+---------+-------+
| Field       | Type        | Null | Key | Default | Extra |
+-------------+-------------+------+-----+---------+-------+
| C_Id        | int         | NO   |     | NULL    |       |
| Course_name | varchar(30) | YES  |     | NULL    |       |
+-------------+-------------+------+-----+---------+-------+
2 rows in set (0.00 sec)

desc Enrollment_details;

+-------+------+------+-----+---------+-------+

| Field | Type | Null | Key | Default | Extra |

+-------+------+------+-----+---------+-------+

| Id    | int  | NO   |     | NULL    |       |

| C_Id  | int  | NO   |     | NULL    |       |

+-------+------+------+-----+---------+-------+

2 rows in set (0.00 sec)


mysql> alter table student_details modify column Id int Primary key;

Query OK, 0 rows affected (0.08 sec)

Records: 0  Duplicates: 0  Warnings: 0


mysql> alter table Enrollment_details modify column Id int Primary key;

Query OK, 0 rows affected (0.04 sec)

Records: 0  Duplicates: 0  Warnings: 0

alter table Enrollment_details add Foreign key(S_Id) references Student_details(Id);

Query OK, 0 rows affected (0.08 sec)

Records: 0  Duplicates: 0  Warnings: 0


mysql> desc Enrollment_details;

+-------+------+------+-----+---------+-------+

| Field | Type | Null | Key | Default | Extra |

+-------+------+------+-----+---------+-------+

| Id    | int  | NO   | PRI | NULL    |       |

| C_Id  | int  | NO   |     | NULL    |       |

| S_Id  | int  | NO   | MUL | NULL    |       |

+-------+------+------+-----+---------+-------+

3 rows in set (0.00 sec)

insert into Student_details(Id,Name,Course) values('1','Vaidehi','It');
```

```
Query OK, 1 row affected (0.01 sec)


mysql> insert into Student_details(Id,Name,Course) values('2','Ashwini', 'Cs');

Query OK, 1 row affected (0.00 sec)

Select Id,Name,Course from Student_details;

+----+---------+--------+

| Id | Name    | Course |

+----+---------+--------+

|  1 | Vaidehi | It     |

|  2 | Ashwini | Cs     |

+----+---------+--------+

2 rows in set (0.00 sec)

mysql> insert into Course_details(C_Id,Course_name) values('1', 'Cs');

Query OK, 1 row affected (0.01 sec)


mysql> insert into Course_details(C_Id,Course_name) values('2', 'It');

Query OK, 1 row affected (0.01 sec)

mysql> Select C_Id,Course_name from Course_details;

+------+-------------+

| C_Id | Course_name |

+------+-------------+

|    1 | Cs          |

|    2 | It          |

+------+-------------+

2 rows in set (0.00 sec)

mysql> insert into Enrollment_details(Id,C_Id,S_Id) values('1', '2','1');

Query OK, 1 row affected (0.01 sec)


mysql> insert into Enrollment_details(Id,C_Id,S_Id) values('2', '1','2');

Query OK, 1 row affected (0.01 sec)
```

```
mysql> Select Id,C_Id,S_Id from Enrollment_details;

+----+------+------+

| Id | C_Id | S_Id |

+----+------+------+

|  1 |    2 |    1 |

|  2 |    1 |    2 |

+----+------+------+

2 rows in set (0.00 sec)
```

2. Design and implement a database for an online shopping system with at least the following entities: Customers, Products, and Orders. • Write DDL statements to create the required tables. • Normalize the schema from 1NF to 3NF. • Highlight and explain how redundant data is eliminated during normalization.

1. Design and implement a database for a student course registration system. The system should manage Students, Courses, and Enrollments.

- Use DDL statements to create the tables.
- Apply normalization (up to 3NF) and explain each step.
- Include constraints like primary keys, foreign keys, and NOT NULL where applicable.

---

2. Design and implement a database for an online shopping system with at least the following entities: Customers, Products, and Orders.

- Write DDL statements to create the required tables.
- Normalize the schema from 1NF to 3NF.
- Highlight and explain how redundant data is eliminated during normalization.

---

3. Create a database for a library management system with entities such as Books, Members, and Loans.

- Write the DDL statements to define tables with constraints.
- Show the process of normalization to 3NF with examples of functional dependencies.
- Explain the final schema and why it satisfies 3NF.

---

4. Design a database for a hospital system managing Patients, Doctors, and Appointments.

- Implement the database schema using DDL SQL statements.
- Apply normalization on your initial unnormalized data model.
- Provide justification for normalization decisions.

---

5. Design and implement a database for an employee management system. The system should include two tables: Employees and Departments.

- Employees: Contains employee details like ID, Name, Salary, and Department ID (foreign key).
- Departments: Contains department details like Department ID and Department Name.
- Create the Employees and Departments tables with primary key and foreign key constraints.
- Write SQL DDL statements for creating these tables.
- Alter the Employees table to add a new column for the employee’s PhoneNumber.
- Modify the Departments table to rename the DepartmentName column to DeptName.
- Drop the Employees table after completing the tasks.

---

6. Design and implement a course registration system with the following tables: Students, Courses, and Registrations.

- Students: Contains StudentID, Name, and Email.
- Courses: Contains CourseID, CourseName, and Credits.
- Registrations: Contains RegistrationID, StudentID (foreign key), and CourseID (foreign key).
- Create the Students, Courses, and Registrations tables with primary key and foreign key constraints.
- Write SQL DDL statements for creating these tables.
- Alter the Registrations table to add a new column called RegistrationDate.
- Modify the Courses table to change the column Credits data type to INTEGER.
- Drop the Registrations table after completing the tasks.

---

7. Create a database for an online order management system that includes the following tables: Customers, Orders, and Products.

- Customers: Contains CustomerID, Name, and Email.
- Orders: Contains OrderID, CustomerID (foreign key), and OrderDate.
- Products: Contains ProductID, ProductName, and Price.
- Create the Customers, Orders, and Products tables with primary key and foreign key constraints.
- Alter the Orders table to add a new column ShippingAddress.
- Modify the Products table to rename the column ProductName to ProductDescription.
- Drop the Products table after completing the tasks.

---

8. Design and create a university management system with the following tables: Students, Courses, and Enrollments.

- Students: Contains StudentID, Name, and DOB.
- Courses: Contains CourseID, CourseName, and Credits.
- Enrollments: Contains EnrollmentID, StudentID (foreign key), and CourseID (foreign key).
- Create the Students, Courses, and Enrollments tables with primary key and foreign key constraints.
- Alter the Enrollments table to add a column Grade.
- Modify the Students table to change the DOB column's data type to DATE instead of VARCHAR.
- Drop the Students table after completing the tasks.

---

9. Assume you have the following tables for a University Management System: Students, Courses, and Enrollments. Perform the following queries:

- Retrieve all Students whose StudentID is greater than 1000.
- Get all Courses that have a Credits value equal to or greater than 3.

- List the Enrollments where the CourseID is not 101 and StudentID is less than 50.

---

10. Given the Books and Members tables in a Library Database, perform the following:

- Retrieve all Members whose name contains the substring "John" using pattern matching.
- Retrieve Books whose Title starts with "Java" or ends with "Programming".
- Get all Members who are Active (where the ActiveStatus column is TRUE).

---

11. In an Online Shopping System, given Orders and Products tables, perform the following:

- Calculate the total OrderAmount by multiplying Quantity and Price for each order.
- Compute the Average Price of all Products in the catalog.
- Retrieve the total quantity of products ordered (i.e., the sum of the Quantity column in the Orders table).

---

12. In the Sales table, where SalesID, ProductID, and SalesAmount are present, perform the following:

- Get the total sales for each ProductID.
- Retrieve the maximum, minimum, and average sales amount for each ProductID.
- Group the data by ProductID and show the count of sales per product.

---

13. Using a Library Management System, assume you have a table Borrowing with BorrowDate and ReturnDate. Perform the following:

- Retrieve all Books that were borrowed before January 1st, 2021.
- Find the duration (in days) for which each book was borrowed (i.e., ReturnDate - BorrowDate).
- Show the current date and the current time.

---

14. In the Hospital Management System with Patients, Doctors, and Appointments tables, perform the following complex query:

- Find the DoctorID and the total number of appointments for each doctor, but only include doctors who have more than 5 appointments in total.
- List the patients who have had appointments with more than one doctor.
- Find all patients who are older than 30 and have had an appointment with a doctor in the last 6 months.

---

15. Given two tables, Customers and VIP_Customers, perform the following set operations:

- Retrieve a list of all customers who are either in the Customers table or in the VIP_Customers table (using UNION).

- Retrieve a list of customers who appear in both the Customers and VIP_Customers tables (using INTERSECT).
- Retrieve a list of Customers who are not in the VIP_Customers table (using EXCEPT).

---

16. Create a view called StudentCourseView that combines Students and Courses tables in the University Management System. The Students table contains StudentID, Name, and Age. The Courses table contains CourseID, CourseName, and Credits. Create the view to list the student names along with the courses they are enrolled in.

After creating the view, perform an update on the Students table using the StudentCourseView view by changing a student's Name.

---

17. Assume you have a Sales table with SaleID, ProductID, and SaleAmount. Create a view that calculates the total sales for each ProductID. Then, attempt to update the base table (Sales) using the view.

- Sales Table:
    - SaleID (Primary Key)
    - ProductID
    - SaleAmount

---

18. In the Hospital Management System, assume you have three tables: Doctors, Patients, and Appointments. The Appointments table has AppointmentID, DoctorID, PatientID, and AppointmentDate. Create a view that joins these three tables to show all appointments, including Doctor Name, Patient Name, and Appointment Date.

---

19. In the Employee Management System, create a view called EmployeeSalaryView that shows employee names and their salary. The Employees table contains EmployeeID, EmployeeName, and Salary. After creating the view, attempt to update the Salary for an employee using the view.

---

20. Consider the Sales table with SaleID, ProductID, SaleAmount, and SaleDate. Create a view that shows only sales with a SaleAmount greater than $1000. Then attempt to update the SaleAmount for a specific product through the view.

---

21. Create a view that calculates the total sales per product in the Sales table. Then, attempt to update the Sales table using this view.

After working with views, it is often necessary to drop them when they are no longer needed.

- Drop the view EmployeeSalaryView you created in a previous example.

---

22. In the E-commerce System, you have the following tables: Customers, Orders, and Products. Create a view that shows CustomerName, ProductName, and OrderDate for each order. After creating this view, attempt to update the OrderDate via the view.

23. Write a PL/SQL stored procedure that inserts a new employee into the Employees table. The procedure should accept the EmployeeID, Name, and Salary as input parameters. After executing the procedure, retrieve the details of the newly inserted employee.

---

24. Write a PL/SQL stored procedure that updates the Salary of an employee. The procedure should accept EmployeeID and NewSalary as input parameters.

---

25. Write a PL/SQL stored procedure that deletes an employee from the Employees table. The procedure should take EmployeeID as an input parameter.

---

26. Write a statement-level trigger that logs the total number of employees in the Employees table to a LogTable whenever a new employee is added.

---

27. Write a statement-level trigger that logs each deletion from the Employees table into a DeletionLog table. The trigger should log the number of records deleted and the date of deletion.

---

28. Write a row-level trigger that logs any update to the Salary field in the Employees table. Whenever an employee's salary is updated, the trigger should insert a record into a SalaryHistory table to keep track of the previous salary and the new salary.

---

29. Write a PL/SQL block that uses an implicit cursor to fetch a single employee's details (e.g., EmployeeID, Name, and Salary) from the Employees table, where the EmployeeID is passed as a parameter. The block should display the details using DBMS_OUTPUT.

---

30. Write a PL/SQL block that uses an explicit cursor to fetch all employees' details (e.g., EmployeeID, Name, and Salary) from the Employees table and displays them using DBMS_OUTPUT.

Problem Statements 1 to 22 are compulsory. Also 23,26,28,29,30 are also compulsory.

23. Write a PL/SQL stored procedure that inserts a new employee into the Employees table. The procedure should accept the EmployeeID, Name, and Salary as input parameters. After executing the procedure, retrieve the details of the newly inserted employee.

Solution:

✅ **Step 1: Assume a table structure**

```
CREATE TABLE Employees (
   EmployeeID INT PRIMARY KEY,
   Name VARCHAR(100),
   Salary DECIMAL(10,2)
);
```

✅ **Step 2: Create the stored procedure**

```
DELIMITER //
CREATE PROCEDURE InsertAndGetEmployee(
   IN p_EmployeeID INT,
   IN p_Name VARCHAR(100),
   IN p_Salary DECIMAL(10,2)
)
BEGIN
   -- Insert the new employee
   INSERT INTO Employees (EmployeeID, Name, Salary)
   VALUES (p_EmployeeID, p_Name, p_Salary);

   -- Return the inserted employee details
   SELECT * FROM Employees WHERE EmployeeID = p_EmployeeID;
END //
DELIMITER ;
```

✅ **Step 3: Call the procedure**

```
CALL InsertAndGetEmployee(101, 'Alice Johnson', 55000.00);
```

---

28. Write a row-level trigger that logs any update to the Salary field in the Employees table. Whenever an employee's salary is updated, the trigger should insert a record into a SalaryHistory table to keep track of the previous salary and the new salary.

Solution:

✅ **Step 1: Assume the existing Employees table**

```
CREATE TABLE Employees (
  EmployeeID INT PRIMARY KEY,
  Name VARCHAR(100),
  Salary DECIMAL(10,2)
);
```

✅ **Step 2: Create the SalaryHistory table**

```
CREATE TABLE SalaryHistory (
  HistoryID INT AUTO_INCREMENT PRIMARY KEY,
  EmployeeID INT,
  OldSalary DECIMAL(10,2),
  NewSalary DECIMAL(10,2),
  ChangeDate TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

✅ **Step 3: Create the trigger**

This trigger activates **after any update** to the Salary field **only if the salary actually changes**.

```
DELIMITER //
CREATE TRIGGER trg_SalaryUpdate
AFTER UPDATE ON Employees
FOR EACH ROW
BEGIN
  IF OLD.Salary != NEW.Salary THEN
    INSERT INTO SalaryHistory (EmployeeID, OldSalary, NewSalary)
    VALUES (OLD.EmployeeID, OLD.Salary, NEW.Salary);
  END IF;
END //
DELIMITER ;
```

✅ **Usage Example**

```
-- Update salary
```

```
UPDATE Employees
SET Salary = 60000.00
WHERE EmployeeID = 101;

-- Check log
SELECT * FROM SalaryHistory WHERE EmployeeID = 101;
```

---

26. Write a statement-level trigger that logs the total number of employees in the Employees table to a LogTable whenever a new employee is added.

Solution:

✅ **Step 1: Create Employees table**

```
CREATE TABLE Employees (
   EmployeeID INT PRIMARY KEY,
   Name VARCHAR(100),
   Salary DECIMAL(10,2)
);
```

✅ **Step 2: Create LogTable**

```
CREATE TABLE LogTable (
   LogID INT AUTO_INCREMENT PRIMARY KEY,
   LogTime TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
   TotalEmployees INT
);
```

✅ **Step 3: Create the trigger (row-level, simulating statement-level)**

```
DELIMITER //
CREATE TRIGGER trg_LogEmployeeCount
AFTER INSERT ON Employees
FOR EACH ROW
BEGIN
   INSERT INTO LogTable (TotalEmployees)
   SELECT COUNT(*) FROM Employees;
END //
DELIMITER ;
```

---

29. Write a PL/SQL block that uses an implicit cursor to fetch a single employee's details (e.g., EmployeeID, Name, and Salary) from the Employees table, where the EmployeeID is passed as a parameter. The block should display the details using DBMS_OUTPUT.

Solution:

```
DELIMITER //
CREATE PROCEDURE GetEmployeeDetails(IN p_EmployeeID INT)
BEGIN
  DECLARE v_Name VARCHAR(100);
  DECLARE v_Salary DECIMAL(10,2);

  SELECT Name, Salary
  INTO v_Name, v_Salary
  FROM Employees
  WHERE EmployeeID = p_EmployeeID;

  SELECT
    p_EmployeeID AS EmployeeID,
    v_Name AS Name,
    v_Salary AS Salary;
END //
DELIMITER ;
```

Call the procedure with:

```
CALL GetEmployeeDetails(101);
```

---

30. Write a PL/SQL block that uses an explicit cursor to fetch all employees' details (e.g., EmployeeID, Name, and Salary) from the Employees table and displays them using DBMS_OUTPUT.

Solution:

```
DELIMITER //
CREATE PROCEDURE ShowEmployees()
BEGIN
```

```
   DECLARE done INT DEFAULT FALSE;
   DECLARE v_EmployeeID INT;
   DECLARE v_Name VARCHAR(100);
   DECLARE v_Salary DECIMAL(10,2);

   DECLARE emp_cursor CURSOR FOR
      SELECT EmployeeID, Name, Salary FROM Employees;

   DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

   OPEN emp_cursor;

   read_loop: LOOP
      FETCH emp_cursor INTO v_EmployeeID, v_Name, v_Salary;
      IF done THEN
         LEAVE read_loop;
      END IF;

      -- In MySQL, SELECT outputs to client
      SELECT CONCAT('ID: ', v_EmployeeID, ', Name: ', v_Name, ', Salary: ', v_Salary) AS
EmployeeDetails;
   END LOOP;

   CLOSE emp_cursor;
END //
DELIMITER ;
```

You can run it with:

```
CALL ShowEmployees();
```