Management Information System Questions

Q1: Design and create a database schema for a small online bookstore. Include tables for books, authors, customers, and orders. Provide SQL commands for table creation and insert sample data.

Answer:

Tables:

- `Authors`
- `Books`
- `Customers`
- `Orders`
- `OrderDetails` (to handle many-to-many between orders and books)

SQL Commands:

```
CREATE TABLE Authors (
    author_id INT PRIMARY KEY AUTO_INCREMENT,
    name VARCHAR(100),
    bio TEXT
);

CREATE TABLE Books (
    book_id INT PRIMARY KEY AUTO_INCREMENT,
    title VARCHAR(150),
    price DECIMAL(10, 2),
    stock INT,
    author_id INT,
    FOREIGN KEY (author_id) REFERENCES Authors(author_id)
);

CREATE TABLE Customers (
    customer_id INT PRIMARY KEY AUTO_INCREMENT,
```

```
    name VARCHAR(100),
    email VARCHAR(100),
    phone VARCHAR(15)
);

CREATE TABLE Orders (
    order_id INT PRIMARY KEY AUTO_INCREMENT,
    customer_id INT,
    order_date DATE,
    FOREIGN KEY (customer_id) REFERENCES Customers(customer_id)
);

CREATE TABLE OrderDetails (
    order_id INT,
    book_id INT,
    quantity INT,
    PRIMARY KEY (order_id, book_id),
    FOREIGN KEY (order_id) REFERENCES Orders(order_id),
    FOREIGN KEY (book_id) REFERENCES Books(book_id)
);
```

Data Insert:

```
-- Authors
INSERT INTO Authors (name, bio) VALUES
('George Orwell', 'English novelist and essayist'),
('J.K. Rowling', 'British author, best known for Harry Potter');
```

```
-- Books
INSERT INTO Books (title, price, stock, author_id) VALUES
('1984', 15.99, 10, 1),
('Animal Farm', 9.99, 15, 1),
('Harry Potter and the Sorcerer''s Stone', 20.00, 8, 2);

-- Customers
INSERT INTO Customers (name, email, phone) VALUES
('Alice', 'alice@example.com', '1234567890'),
('Bob', 'bob@example.com', '0987654321');

-- Orders
INSERT INTO Orders (customer_id, order_date) VALUES
(1, CURDATE()),
(2, CURDATE() - INTERVAL 10 DAY);

-- OrderDetails
INSERT INTO OrderDetails (order_id, book_id, quantity) VALUES
(1, 1, 1),
(2, 3, 2);
```

Q2: SQL query to find customers who purchased books in the last 30 days with book titles and order dates

Answer:

```
SELECT
    c.name AS customer_name,
    b.title AS book_title,
    o.order_date
FROM
    Customers c
JOIN Orders o ON c.customer_id = o.customer_id
JOIN OrderDetails od ON o.order_id = od.order_id
JOIN Books b ON od.book_id = b.book_id
WHERE
    o.order_date >= CURDATE() - INTERVAL 30 DAY;
```

Q3: Normalize a sample unnormalized table to 3NF

Answer:

Unnormalized Table (UNF):

OrderID	CustomerName	BookTitle	AuthorName	Quantity	OrderDate

1NF (remove repeating groups)

Split into multiple rows per book:

- Make sure each field has atomic values.

2NF (remove partial dependencies)

- Separate customers, books, and authors.

Tables:

- Customers (`CustomerID`, `CustomerName`)
- Authors (`AuthorID`, `AuthorName`)
- Books (`BookID`, `BookTitle`, `AuthorID`)
- Orders (`OrderID`, `CustomerID`, `OrderDate`)
- OrderDetails (`OrderID`, `BookID`, `Quantity`)

3NF (remove transitive dependencies)

- Everything depends only on the primary key.
- No extra derivable data (e.g., no storing `AuthorName` in `Books`—it stays in `Authors`).

1NF

OrderID	CustomerName	BookTitle	AuthorName	Quantity
1	Alice	1984	George Orwell	2024-04-20
1	Alice	Animal Farm	George Orwell	2024-04-20
2	Bob	1984	George Orwell	2024-04-22

2NF

Customers		CostomeID
1	Alice	1
2	Bob	2

Orders	Addreate
2024-04-20	2024-04-20
2024-04-22	2024-04-22

3NF

Customers		OrdersID
1	Alice	1
2	Bob	2

Books	AuthorID
1984	1
Animal Farm	1

3NF

Customers		OrdersID
1	Alice	Customorl
2	Bob	Customerl

Bookt	AuthorID
1984	1
George Orwell	1

Q4: Create an ERD from a list of entities

Answer:

Entities

- Represented as **blue rectangles** labeled "Entity"
- Contain placeholder text like "text", which should be replaced with actual entity names and their attributes
- Each entity can be something like:

- `Customer`
- `Book`
- `Order`

Attributes

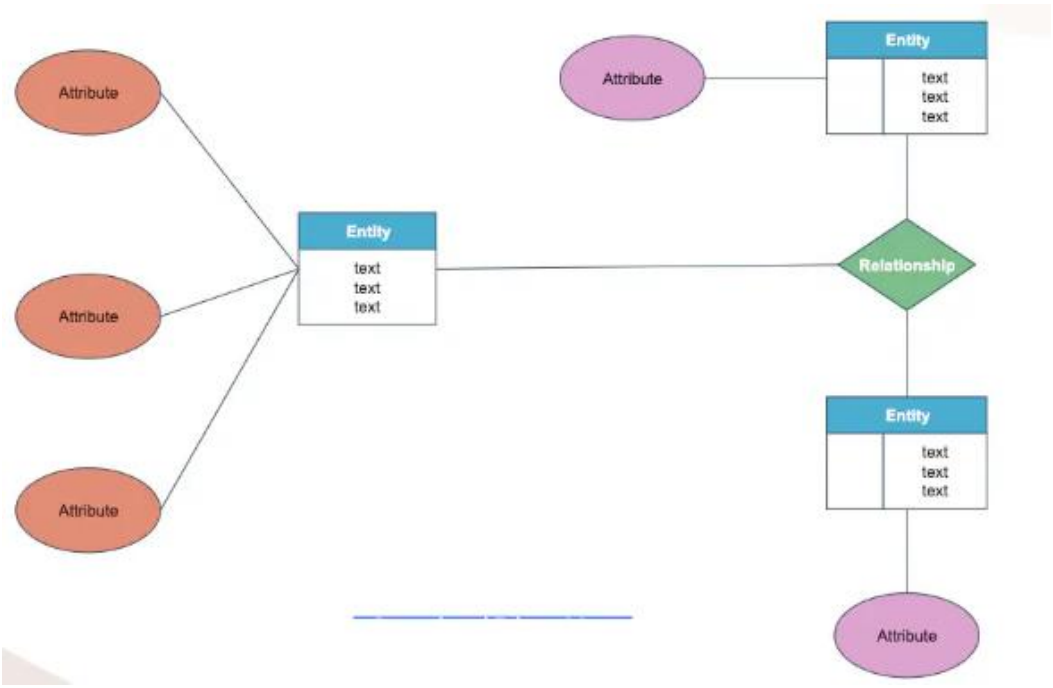
- Represented as **colored ovals** (pink/red)
- Linked to their corresponding entities
- Examples (if filled in) could be:
  - For `Customer`: `CustomerID`, `Name`, `Email`
  - For `Book`: `ISBN`, `Title`, `Price`

Relationships

- Shown as **green diamonds** labeled "Relationship"
- Connect two or more entities
- Example relationships might be:
  - `Places` between `Customer` and `Order`
  - `Contains` between `Order` and `Book`

Diagram Flow (Structure)

- Entities are connected through relationships
- Attributes branch out from each entity
- This diagram outlines how entities relate without going into specific details



Q5: INNER JOIN, LEFT JOIN, RIGHT JOIN examples

INNER JOIN:

```
SELECT Customers.name, Orders.order_id
FROM Customers
INNER JOIN Orders ON Customers.customer_id = Orders.customer_id;
```

LEFT JOIN:

```
SELECT Customers.name, Orders.order_id
FROM Customers
LEFT JOIN Orders ON Customers.customer_id = Orders.customer_id;
```

RIGHT JOIN:

```
SELECT Customers.name, Orders.order_id
FROM Customers
RIGHT JOIN Orders ON Customers.customer_id = Orders.customer_id;
```

Assumed Tables and Data:

customer_id	name
1	Alice
2	Bob
3	Carol

`Orders`

order_id	customer_id	order_date
101	1	2024-03-01
102	2	2024-03-05

INNER JOIN:

name	order_id	order_date
Alice	101	2024-03-01
Bob	102	2024-03-05

LEFT JOIN:

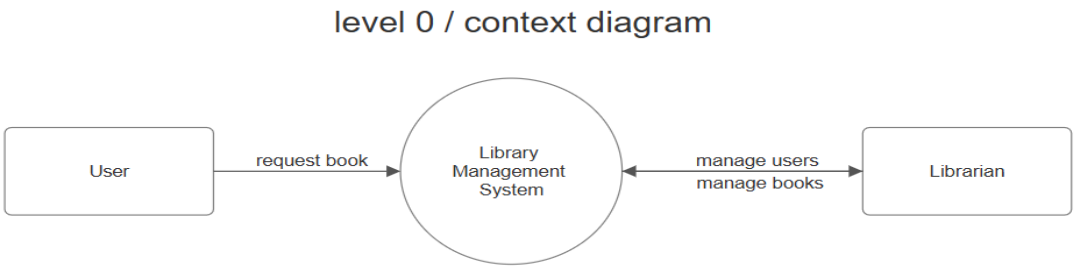
name	order_id	order_date
Alice	101	2024-03-01
Bob	102	2024-03-05
Carol	NULL	NULL

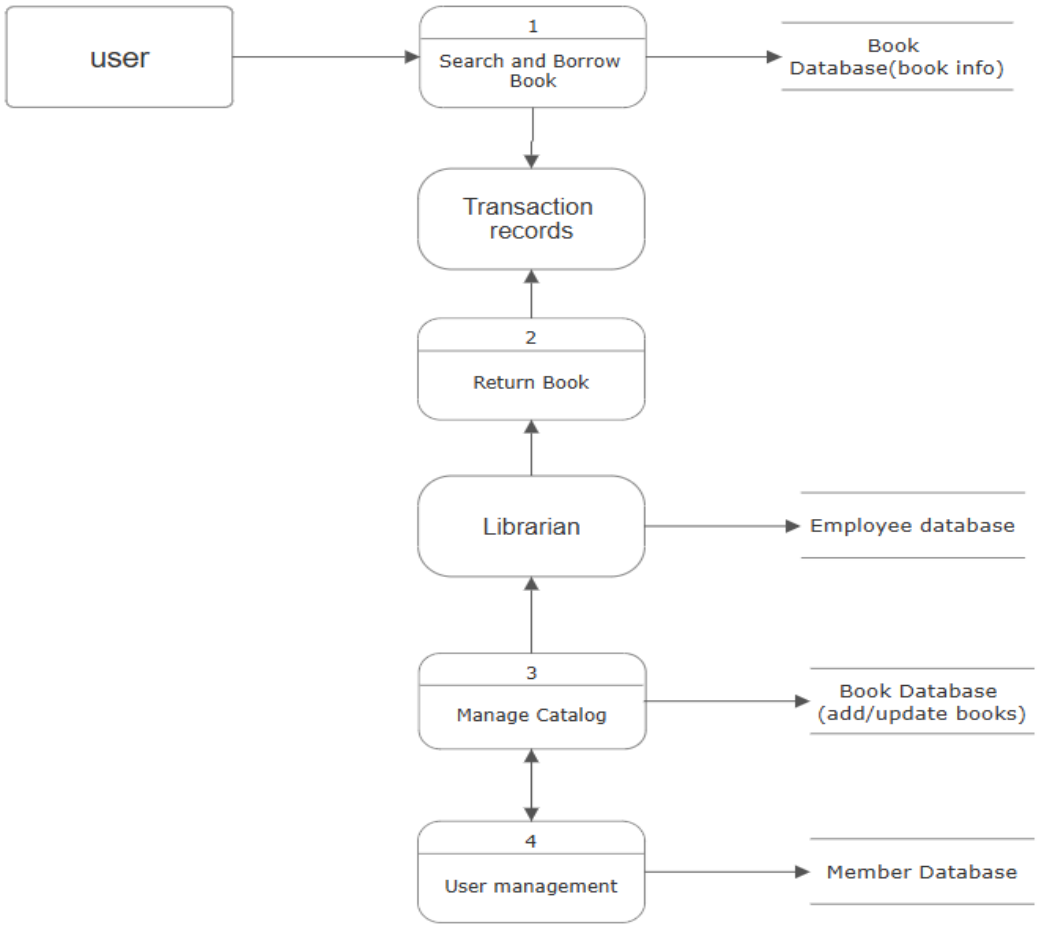
RIGHT JOIN:

name	order_id	order_date
Alice	101	2024-03-01
Bob	102	2024-03-05

6. Create a Data Flow Diagram for a library management system. Include processes like book borrowing, returning, and catalog management.

Answer:





7. **Draw an ERD for a student management system. Include entities such as students, courses, instructors, and enrollments.**

Answer:

Entities and Attributes:

1. Student

- *student_id* (PK)
- first_name
- last_name
- s_email
- date_of_birth

2. Instructor

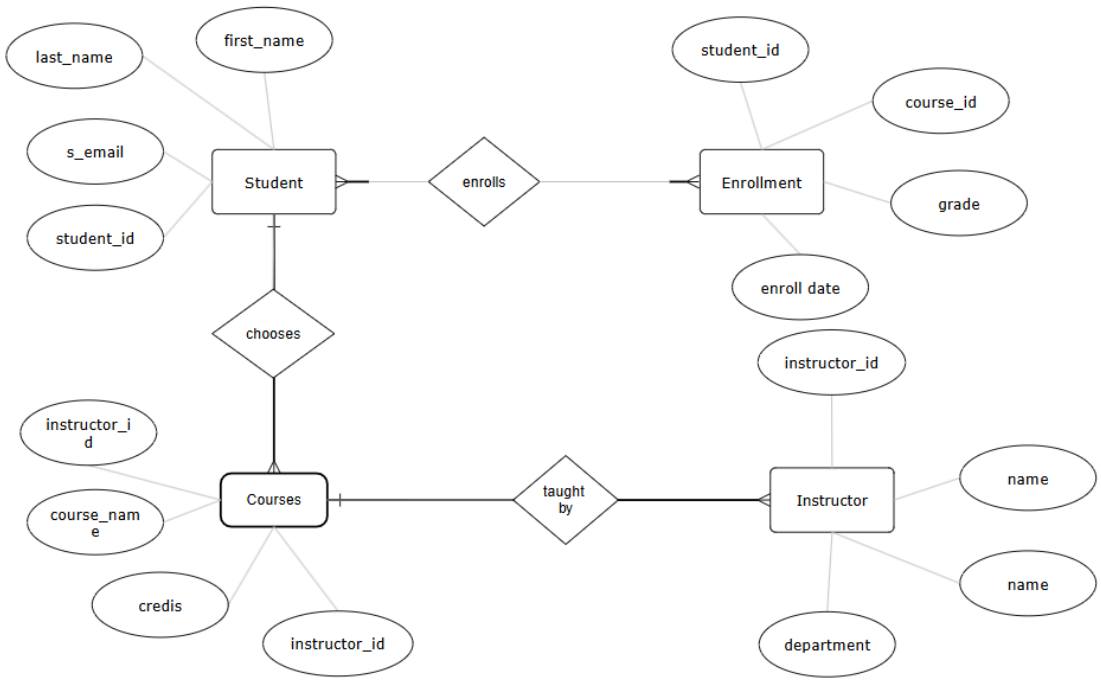
- *instructor_id* (PK)
- name
- email
- department

3. Course

- *course_id* (PK)
- course_name
- credits
- instructor_id (FK → Instructor)

4. Enrollment

- *enrollment_id* (PK)
- student_id (FK → Student)
- course_id (FK → Course)
- enrollment_date
- grade

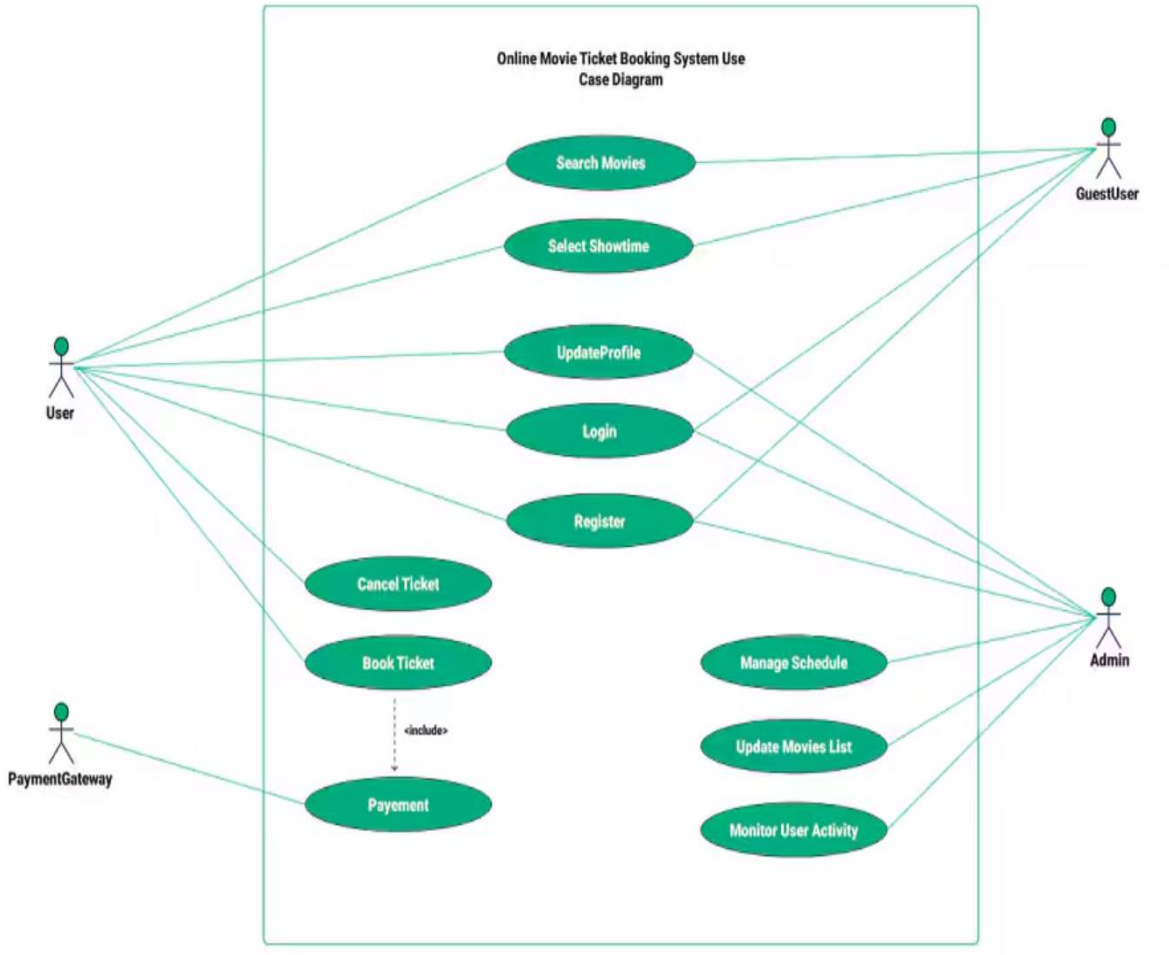


🔗 Relationships:

- A **student** can enroll in **many courses** (many-to-many via Enrollment).
- A **course** can have **many students** (many-to-many via Enrollment).
- An **instructor** can teach **many courses** (one-to-many).

8. Develop a Use Case Diagram for an online ticket booking system. Identify the actors and use cases.

Answer:



Actors:

1. User
    - A registered user of the system.
2. GuestUser
    - A user who has not logged in or registered.
3. Admin
    - System administrator managing movies and schedules.
4. PaymentGateway
    - External system handling payment transactions.

Use Cases:

Accessible by User and/or GuestUser:

1. Search Movies
2. Select Showtime
3. Login
4. Register
5. Update Profile

Accessible by Registered User only:

6. Book Ticket
7. Cancel Ticket
8. Payment *(included in "Book Ticket" use case)*

Accessible by Admin:

9. Manage Schedule
10. Update Movies List
11. Monitor User Activity

9. Describe and draw the high-level architecture of a Customer Relationship Management (CRM) system. Include components like the user interface, application server, and database server.

Answer:

1. Presentation Layer (User Interface):

- Who uses it: Sales reps, marketing teams, customer support, managers.
- What it does: Provides web/mobile interfaces for users to view, enter, and manage customer data.
- Technology used: Web browsers, mobile apps, HTML/CSS/JavaScript frontends.

2. Application Layer (Application Server):

- What it does: Contains business logic — handles rules like assigning leads, generating reports, tracking customer interactions.
- Technology used: Backend frameworks like Node.js, Java, .NET, or Python (Django/Flask).

3. Data Layer (Database Server):

- What it does: Stores customer records, communication history, purchase data, support tickets, etc.
- Technology used: Relational DBs like MySQL, PostgreSQL or NoSQL DBs like MongoDB (for flexible data).

Optional Components:

- APIs: For integration with email systems, social media, ERP, or payment gateways.
- Analytics Engine: For reporting, forecasting, and insights.
- Security Module: Ensures user roles, encryption, and audit trails.

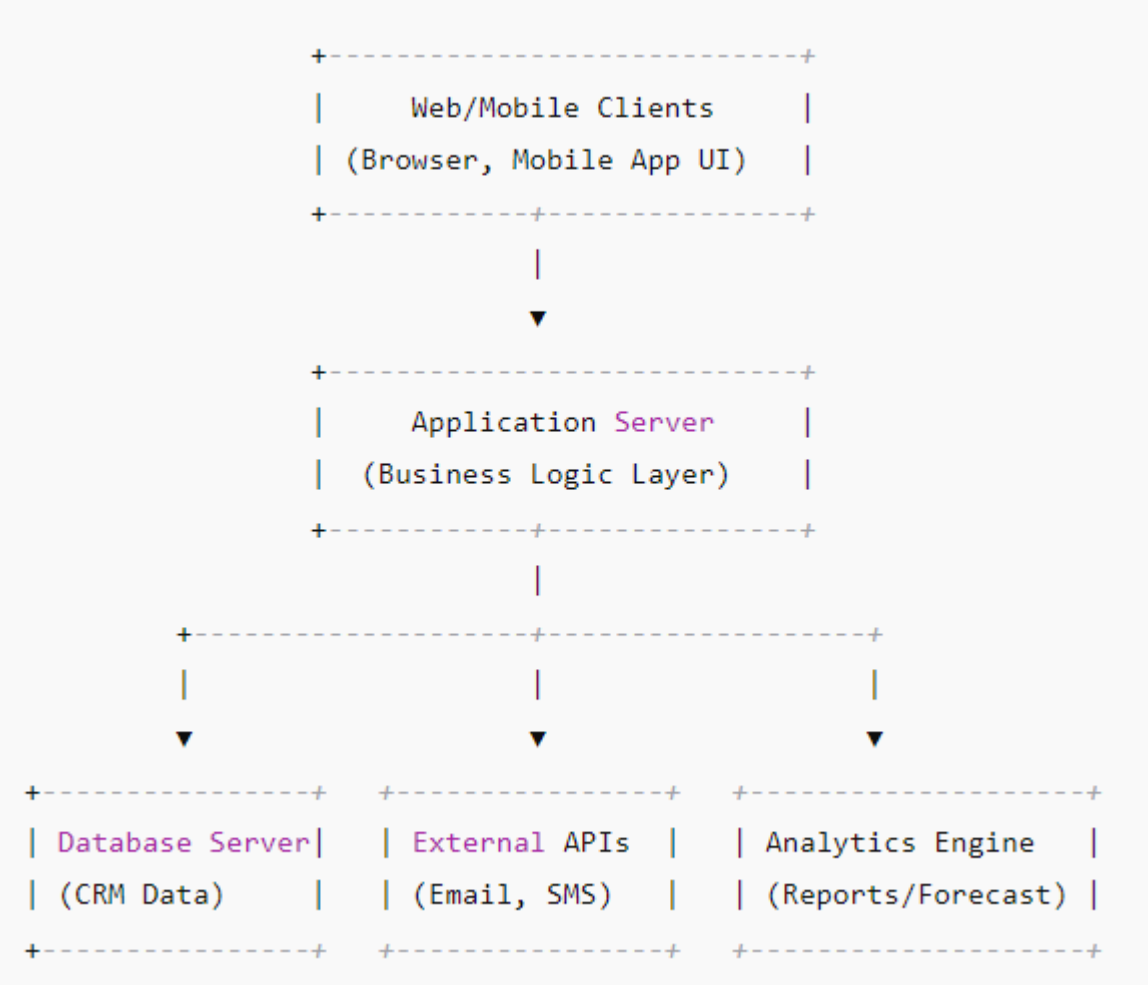


Fig: high level architecture of CRM system

10. Perform CRUD operations in SQL

Answer:

Create

```
CREATE TABLE Customers (
    CustomerID INT PRIMARY KEY,
    Name VARCHAR(100),
    Email VARCHAR(100)
);
INSERT INTO Customers (CustomerID, Name, Email)
VALUES (1, 'Alice Johnson', 'alice@example.com');
```

Explanation:
This adds a new row to the `Customers` table with a unique `CustomerID`, name, and email.

CustomerID	Name	Email
1	Alice Johnson	alice@example.com

Read

```
SELECT * FROM Customers;
```

Explanation:
`SELECT *` fetches all records and columns from the table.

CustomerID	Name	Email
1	Alice Johnson	alice@example.com

Update

```
UPDATE Customers
SET Email = 'alice.johnson@newmail.com'
WHERE CustomerID = 1;
```

Explanation:
Changes the email of the customer where `CustomerID = 1`.

CustomerID	Name	Email
1	Alice Johnson	alice.johnson@newmail.com

DELETE

DELETE FROM Customers

WHERE **CustomerID** = 1;

Explanation:

Removes the row with `CustomerID = 1` from the table.

CustomerID	Name	Email

No rows returned

11. Stored Procedure to Calculate Total Sales

Answer:

```
CREATE PROCEDURE CalculateMonthlySales (@Month INT, @Year INT)
AS
BEGIN
   SELECT SUM(TotalAmount) AS TotalSales
   FROM Sales
    WHERE MONTH(SaleDate) = @Month AND YEAR(SaleDate) = @Year;
END;
```

Execute the Procedure

```
EXEC CalculateMonthlySales 3, 2025;
```

12. Trigger to Update Stock After Order

Answer:

```
CREATE TRIGGER UpdateStock

AFTER INSERT ON Orders

FOR EACH ROW

BEGIN

   UPDATE Products

   SET StockQuantity = StockQuantity - NEW.Quantity

   WHERE ProductID = NEW.ProductID;

END;
```

13. Simple Form and Report

Answer:

Form Concept (In DBMS GUI, like MS Access)

Create form to input:

- Name
- Email
- Phone

Sample Table

```
CREATE TABLE Customers (

   CustomerID INT PRIMARY KEY,

   Name VARCHAR(100),

   Contact VARCHAR(100)

);
```

Report Query

```
SELECT Name, Contact FROM Customers;
```

14. Encryption in SQL

Answer:

Encryption Methods:

- **Transparent Data Encryption (TDE)**
- **Column-Level Encryption**
- **Hashing with Salt**

Example: Column Encryption (SQL Server)

```
CREATE SYMMETRIC KEY SSN_Key

WITH ALGORITHM = AES_256

ENCRYPTION BY PASSWORD = 'StrongPassword@123';


OPEN SYMMETRIC KEY SSN_Key

DECRYPTION BY PASSWORD = 'StrongPassword@123';


INSERT INTO Employees (Name, SSN)

VALUES ('John Doe', EncryptByKey(Key_GUID('SSN_Key'), '123-45-6789'));


CLOSE SYMMETRIC KEY SSN_Key;
```

15. User Roles and Permissions

Answer:

Create Roles

```
CREATE ROLE SalesRole;

GRANT SELECT, INSERT ON Sales TO SalesRole;
```

Assign Role to User

EXEC sp_addrolemember 'SalesRole', 'username';

Access Levels Example

Access Level	Permission Type	Example Use Case
Read-Only	`SELECT` only	View reports
Read/Write	`SELECT`, `INSERT`, `UPDATE`	Data entry users
Admin	All privileges (`SELECT`, `INSERT`, `UPDATE`, `DELETE`, DDL)	DB management
Custom	Specific permissions based on role needs	Only update certain columns or tables

16. Backup and Restore a Database

Answer:

Backup

BACKUP DATABASE MyDatabase

TO DISK = 'D:\Backups\MyDatabase.bak';

Restore

RESTORE DATABASE MyDatabase

FROM DISK = 'D:\Backups\MyDatabase.bak';

Why Regular Backups Are Important

- Prevent data loss from system failures
- Restore system quickly after corruption
- Protect against accidental deletion or ransomware attacks

17. Explain how indexing can improve the performance of SQL queries. Provide an example of creating an index for a specific query.

Answer:

Indexing improves SQL query performance by allowing the database engine to find rows faster, similar to how an index in a book helps you locate a topic quickly without reading the entire book.

How it works:

- Without an index, SQL must scan every row in the table (full table scan).
- With an index, SQL can quickly locate the required rows using a data structure like a **B-tree** or **hash table**.

Example:

table called `employees exists`:

SELECT * FROM employees WHERE last_name = 'Smith';

If `last_name` is not indexed, this query will scan all rows. By adding an index:

CREATE INDEX idx_lastname ON employees(last_name);

Now, the database can use the index `idx_lastname` to directly locate all employees with the last name "Smith", reducing query time significantly.

18. Describe techniques for tuning a database to improve performance. Discuss how to identify and address performance bottlenecks.

Answer:

Techniques for Database Tuning:

1. Indexing:
   - Add indexes to columns used in `WHERE`, `JOIN`, and `ORDER BY` clauses.
2. Query Optimization:
   - Rewrite slow queries using efficient joins, subqueries, or avoiding `SELECT *`.
   - Use EXPLAIN or QUERY PLAN to analyze query execution paths.
3. Partitioning:
   - Divide large tables into smaller, manageable pieces (horizontal or vertical partitioning).
4. Connection Pooling:
   - Reuse database connections to reduce overhead.
5. Caching:

- Store frequently accessed data in memory (e.g., Redis or Memcached).

6. Database Configuration Tuning:
    - Adjust parameters like buffer pool size, cache size, and parallel processing limits.

Identifying and Addressing Bottlenecks:

1. Use Monitoring Tools:
    - Tools like MySQL Workbench, SQL Server Profiler, or pgAdmin help identify slow queries, high CPU usage, and I/O waits.
2. Analyze Slow Query Logs:
    - Find queries with long execution times.
3. Check Locking and Deadlocks:
    - Identify and resolve issues where processes wait on each other.
4. Normalize/Denormalize Tables:
    - Normalize to reduce redundancy or denormalize to reduce joins based on access patterns.

19. Analyze a case study where a company implemented a new MIS. Identify the key challenges and benefits experienced by the company.

Answer:

Case Study: XYZ Retail Chain Implementing an MIS

Background: XYZ Retail Chain implemented a Management Information System (MIS) to improve inventory control, sales reporting, and decision-making processes.

Challenges Faced:

1. Resistance to Change:
    - Employees were used to manual systems and hesitant to adopt new technology.
2. Data Migration Issues:
    - Converting legacy data to the new system caused inconsistencies.
3. Training Requirements:
    - Staff needed extensive training to use the new MIS effectively.
4. Cost Overruns:
    - Implementation costs exceeded initial estimates due to customization needs.

Benefits Achieved:

1. Real-time Reporting:
    - Managers accessed sales and inventory data instantly, leading to faster decisions.
2. Improved Efficiency:

    - Automation of routine tasks reduced manual errors and saved time.
3. Better Inventory Management:
    - Stock levels were optimized, reducing overstock and stockouts.
4. Data-driven Decisions:
    - The system provided analytics that helped plan marketing and logistics better.

20. MIS Implementation Plan:

Question: Develop a basic implementation plan for a new MIS in a small business. Include phases such as planning, design, development, testing, and deployment.

Answer:

1. Planning Phase:

- Objective Definition: Clearly define what the MIS should achieve (e.g., streamline sales, improve inventory control).
- Needs Assessment: Analyze current business processes and identify gaps.
- Feasibility Study: Assess budget, technical capacity, and timeline.
- Resource Allocation: Assign project team roles, identify hardware/software needs.

2. Design Phase:

- System Architecture: Choose system type (web-based, desktop, cloud).
- Database Design: Outline how data will be structured and stored.
- Interface Design: Plan user-friendly dashboards, input forms, and reports.
- Security Planning: Decide on access controls, data encryption, and backups.

3. Development Phase:

- Software Development: Begin coding based on design specs (may use off-the-shelf software or build custom).
- Database Setup: Create tables, relationships, and indexes.
- Integrations: Connect with existing systems (e.g., accounting software).
- Documentation: Prepare user manuals and technical documentation.

4. Testing Phase:

- Unit Testing: Test individual components or modules.
- System Testing: Check how modules work together.
- User Acceptance Testing (UAT): Let staff test and give feedback.
- Bug Fixing: Fix issues identified during testing.

5. Deployment Phase:

- Data Migration: Import data from old systems.

- Go Live: Launch the system in a real business environment.
- Training: Conduct staff training sessions for system use.
- Monitoring: Track performance, usage, and resolve early-stage issues.
- Maintenance Plan: Set a schedule for updates and support.