NEED FOR DATABASE:-

1. **Organized Data Storage**: Manage large data efficiently.
2. **Scalability**: Handle growing data needs.
3. **Data Retrieval**: Query data quickly with SQL.
4. **Data Integrity**: Use constraints and relationships to maintain accurate data.
5. **Data Sharing and Collaboration**:
   Facilitates easy sharing of data among users, applications, or systems in a secure and controlled manner.

Normalization:-

Normalization is the process of structuring a relational database to reduce data redundancy and dependency. It divides larger tables into smaller ones while maintaining relationships between them. This ensures efficient data organization, eliminates anomalies, and enhances data integrity.

Types of Normalization

1. First Normal Form (1NF)

A table is in 1NF if:

- Each column contains atomic (indivisible) values.
- There are no repeating groups or arrays in the columns.
- Each row is uniquely identified by a primary key.

2. Second Normal Form (2NF)

A table is in 2NF if:

- It is in 1NF.
- All non-prime attributes (columns not part of the primary key) are fully functionally dependent on the entire primary key.

This removes partial dependency, where a non-prime attribute depends on only part of a composite primary key.

3. Third Normal Form (3NF)

A table is in 3NF if:

- It is in 2NF.
- There is no transitive dependency, meaning non-prime attributes are only dependent on the primary key and not on other non-prime attributes.

4. Boyce-Codd Normal Form (BCNF)

A table is in BCNF if:

- It is in 3NF.
- Every determinant (a column that determines another column) is a candidate key.

BCNF resolves issues that arise when a table has overlapping candidate keys.

5. Fourth Normal Form (4NF)

A table is in 4NF if:

- It is in BCNF.
- There are no multi-valued dependencies, meaning no column should depend on multiple values of another column.

Queries:-

Select Queries:-

Copy code

-- Select all students with safe aliases

SELECT s.StudentID, s.Name AS StudentName, s.Email, c.CourseName

FROM Students AS s

JOIN Courses AS c ON s.CourseID = c.CourseID;

-- Order students by enrollment date in descending order

SELECT *

FROM Students

ORDER BY EnrollmentDate DESC;

-- Find students enrolled in specific courses using IN

SELECT Name, Email

FROM Students

WHERE CourseID IN (SELECT CourseID FROM Courses WHERE CourseName IN ('Java Programming', 'Python

Insert Queries

```
-- Insert a new student
INSERT INTO Students (Name, Email, Age, Gender, EnrollmentDate, CourseID)
VALUES ('John Doe', 'john.doe@example.com', 22, 'Male', '2024-12-01', 1);
```

Update Queries

```
-- Update fee for a course
UPDATE Courses
SET Fee = 299.99
WHERE CourseName = 'Data Science';
```

Delete Queries

```
-- Delete students who have not enrolled in any course
DELETE FROM Students
WHERE CourseID NOT IN (SELECT CourseID FROM Courses);
```

3. Joins

Inner Join

```
-- Fetch student and their respective course details
SELECT s.Name AS StudentName, c.CourseName, c.InstructorName
FROM Students AS s
INNER JOIN Courses AS c
ON s.CourseID = c.CourseID;
```

Left Join

```
-- Fetch all students with course details (if enrolled)
SELECT s.Name AS StudentName, c.CourseName
FROM Students AS s
LEFT JOIN Courses AS c
ON s.CourseID = c.CourseID;
```

Union

-- Combine student names and instructor names

SELECT Name AS Participant

FROM Students

UNION

SELECT InstructorName AS Participant

FROM Courses;

4. Table Alteration and Constraints

Add Foreign Key and Constraints

-- Add a foreign key constraint (if not already defined)

ALTER TABLE Students

ADD CONSTRAINT FK_Course FOREIGN KEY (CourseID) REFERENCES Courses(CourseID) ON DELETE CASCADE;

Add Unique or Default Constraints

-- Add default value for Gender if not already added

ALTER TABLE Students

MODIFY COLUMN Gender VARCHAR(10) DEFAULT 'Not Specified';

Index Creation

-- Create index on CourseID for optimized joins

CREATE INDEX idx_course ON Students (CourseID);

8. Advanced Queries

Group By and Having

```
-- Group students by CourseID and count them
SELECT CourseID, COUNT(*) AS TotalStudents
FROM Students
GROUP BY CourseID;

-- Only show courses with more than 5 students
SELECT CourseID, COUNT(*) AS TotalStudents
FROM Students
GROUP BY CourseID
HAVING COUNT(*) > 5;
```