**Recall Questions**

**1. What is Normalization?**

Normalization is the process of organizing a database to minimize redundancy and improve data integrity. It involves decomposing large tables into smaller, well-structured tables and defining relationships between them. This ensures efficient storage, prevents anomalies, and maintains data consistency.

**2. Why do we need Normalization?**

Normalization is needed to:

- **Eliminate Redundant Data**: Avoid duplicate data storage.
- **Ensure Data Integrity**: Prevent inconsistencies caused by update, insert, and delete anomalies.
- **Improve Query Performance**: Smaller tables with better indexing allow faster queries.
- **Simplify Maintenance**: Changes to one part of the database do not affect other unrelated parts.

**3. What are the main goals of Normalization?**

- **Reduce Data Redundancy** Avoid storing the same data in multiple places.
- **Ensure Data Consistency** Data should be stored in only one location to prevent mismatches.
- **Prevent Anomalies** Prevent issues when inserting, updating, or deleting data.
- **Improve Scalability** A well-structured database is easier to expand and maintain.

4. **What is Functional Dependency?**

A Functional Dependency (FD) describes a relationship where one attribute uniquely determines another.
If **A → B**, it means **A uniquely determines B**.

**Example**

StudentID → StudentName (Each StudentID corresponds to exactly one StudentName).

OrderID → OrderDate (Each OrderID has a unique OrderDate).

**Types of Functional Dependencies:**

- **Trivial FD** – If B is a subset of A (e.g., {A, B} → A).
- **Non-Trivial FD** – If B is not a subset of A (e.g., A → B).
- **Transitive FD** – If A → B and B → C, then A → C.

**5. What is Partial Dependency?**

A Partial Dependency occurs when a non-key attribute is dependent on part of a composite primary key instead of the whole key.

**Occurs in: 2NF Violation**

**Example:**

| StudentID | CourseID | CourseName |
|---|---|---|
| 101 | C001 | Math |
| 101 | C002 | Science |

CourseName depends only on CourseID, not StudentID → **Partial Dependency**.

**6. What is Transitive Dependency?**

A Transitive Dependency occurs when a non-key attribute depends on another non-key attribute instead of directly on the primary key.

**Occurs in: 3NF Violation**

**Example:**

| StudentID | CourseID | Instructor | InstructorPhone |
|---|---|---|---|
| 101 | C001 | Prof. A | 123456789 |
| 101 | C002 | Prof. B | 987654321 |

InstructorPhone depends on Instructor, not CourseID → **Transitive Dependency**.

**7. What is 1NF? What rule does it follow?**

**First Normal Form (1NF) Rules:**

- Each column should have atomic values (no multiple values in one cell).
- Each row should be unique (has a primary key).

Example (Unnormalized Table):

| StudentID | Name | Courses |
|---|---|---|
| 101 | John | Math, Science |
| 102 | Alice | English |

Courses column has **multiple values → Not in 1NF**.

Let's  Convert into 1NF

| StudentID | Name | Course |
|---|---|---|
| 101 | John | Math |
| 101 | John | Science |
| 102 | Alice | English |

**8. What is 2NF? What rule does it follow?**

- **Second Normal Form (2NF) Rules:**
- Must be in 1NF.
- Remove Partial Dependencies (every non-key attribute should depend on the whole primary key).

Example (1NF Table - Not in 2NF)

| StudentID | CourseID | CourseName | Instructor |
|---|---|---|---|
| 101 | C001 | Math | Prof. A |
| 101 | C002 | Science | Prof. B |

Here we observed that Primary key is the combination of both { **StudentID** , **CourseID** }

But CourseName depends only on CourseID, not on StudentID → **Partial Dependency**.

Let's Convert to 2NF by creating separate tables:

**Students Table:**

| StudentID | Name |
|---|---|
| 101 | John |
| 102 | Alice |

**Courses Table:**

| CourseID | CourseName | Instructor |
|---|---|---|
| **C001** | **Math** | **Prof. A** |
| **C002** | **Science** | **Prof. B** |

**Enrollments Table:**

| StudentID | CourseID |
|---|---|
| 101 | C001 |
| 101 | C002 |

**9. What is 3NF? What rule does it follow?**

**Third Normal Form (3NF) Rules:**

- Must be in 2NF.
- Remove Transitive Dependencies (every non-key attribute should depend only on the primary key).

**Example (2NF Table - Not in 3NF):**

| CourseID | CourseName | Instructor | InstructorPhone |
|---|---|---|---|
| C001 | Math | Prof. A | 123456789 |
| C002 | Science | Prof. B | 987654321 |

In the above table InstructorPhone depends on Instructor, not CourseID → **Transitive Dependency**.

**Courses Table:**

| CourseID | CourseName | InstructorID |
|---|---|---|
| C001 | Math | I001 |
| C002 | Science | I002 |

**Instructors Table:**

| InstructorID | Instructor | InstructorPhone |
|---|---|---|
| I001 | Prof. A | 123456789 |
| I002 | Prof. B | 987654321 |

**10. What is BCNF? How is it different from 3NF?**

**Boyce-Codd Normal Form (BCNF) Rules:**

- Must be in 3NF.
- Every determinant (attribute that determines another attribute) should be a candidate key.

**Difference between 3NF and BCNF:**

- 3NF allows a non-prime attribute to determine another non-prime attribute as long as it depends on the primary key.
- BCNF is stricter – if any determinant is not a candidate key, further decomposition is needed.

**11. What are the Advantages and Disadvantages of Normalization?**

**Advantages of Normalization:**

1. **Eliminates Data Redundancy:** Prevents storing the same data in multiple places, reducing storage usage.

2. **Improves Data Integrity:** Updates are performed in a single location, ensuring consistency.
3. **Prevents Anomalies:** Avoids **Insert, Update, and Delete Anomalies** that can cause inconsistencies.
4. **Better Data Organization:** Structured tables make database management easier.
5. **Efficient Query Performance:** Properly indexed and normalized tables improve query speed.

**Disadvantages of Normalization:**

1. **Complex Queries:** Joins are required to retrieve data from multiple tables, making queries slower.
2. **Increased Number of Tables:** Breaking down data leads to more tables, which may be harder to manage.
3. **Higher Processing Time:** Multiple joins in complex queries may slow down performance.
4. **Difficult for Beginners:** Understanding relationships and foreign key constraints requires expertise.

**12. What is the Difference Between 1NF and 2NF?**

| Aspect | 1NF (First Normal Form) | 2NF (Second Normal Form) |
|---|---|---|
| **Definition** | Ensures atomicity (no repeating or multi-valued columns). | Removes partial dependencies on a composite key. |
| **Key Rule** | Each column contains atomic values (single data per cell). | All non-key columns must depend on the whole primary key. |
| **Requirement** | The table should have a primary key. | Table must be in 1NF before converting to 2NF. |
| **Example** (Before Normalization) | Courses column contains multiple values. | A non-key attribute depends only on part of a composite key. |
| **Solution** | Split multi-valued attributes into separate rows. | Create a separate table to eliminate partial dependencies. |

**13. What is the Difference Between 2NF and 3NF?**

| **Aspect** | 2NF (Second Normal Form) | 3NF (Third Normal Form) |
|---|---|---|
| **Definition** | Removes partial dependencies. | Removes transitive dependencies. |

| Aspect | 2NF (Second Normal Form) | 3NF (Third Normal Form) |
|---|---|---|
| **Key Rule** | Every non-key attribute must depend on the whole primary key. | Every non-key attribute must depend only on the primary key. |
| **Requirement** | Must be in 1NF first. | Must be in 2NF first. |
| **Problem Solved** | Fixes partial dependencies. | Fixes transitive dependencies. |
| **Example of Violation** | CourseName depends on CourseID, not StudentID. | InstructorPhone depends on Instructor, not CourseID. |

**14. What is the Difference Between 3NF and BCNF?**

| Aspect | 3NF (Third Normal Form) | BCNF (Boyce-Codd Normal Form) |
|---|---|---|
| **Definition** | Removes transitive dependencies. | Ensures that every determinant is a candidate key. |
| **Key Rule** | Every non-key attribute must depend only on the primary key. | Every determinant must be a candidate key. |
| **Requirement** | Must be in 2NF first. | Must be in 3NF first. |
| **Problem Solved** | Fixes transitive dependencies. | Fixes cases where 3NF still allows redundancy. |
| **Complexity** | Simpler, used in most databases. | Stricter, used in highly normalized databases. |

**15. What Happens If We Do Not Normalize a Database?**

**Data Redundancy (Repetition of Data)**

- **Example:** Storing the same customer address in every order leads to duplicate data.
- **Problem:** Wastes space and increases chances of inconsistency.

**Insert Anomalies (Problems Adding Data)**

- **Example:** If a new student joins but hasn't enrolled in any course yet, we may not be able to store their data in a table that requires a CourseID.

**Update Anomalies (Problems Modifying Data)**

- **Example:** If an Instructor's phone number changes, we need to update it in multiple places instead of a single record.
- **Problem:** Risk of inconsistent data across records.

**Delete Anomalies (Unintended Loss of Data)**

- **Example:** If the last course of an instructor is removed, the instructor's details might also get lost.

**16. Convert the following unnormalized table into 1NF, 2NF, and 3NF:**

**| OrderID | Customer | Address | Items Ordered |**

**|---|---|---|---|**

**| 001 | John | NY, USA | Laptop, Mouse|**

**| 002 | Alice | CA, USA | Phone |**

Let's normalize the given Unnormalized Table step by step into 1NF, 2NF, and 3NF.

**Step 1: Unnormalized Table (UNF):** Items Ordered contains multiple values, violating 1NF.

| OrderID | Customer | Address | Items Ordered |
|---|---|---|---|
| 001 | John | NY, USA | Laptop, Mouse |
| 002 | Alice | CA, USA | Phone |

**Step 2: Convert to 1NF (First Normal Form)**

**Rules of 1NF:**

- Each column must contain atomic (single) values.
- No repeating groups or multiple values in a single column.

**Let's Devide this UNF table into Two different tables {Order_Details} and {Item_Details}.**

**Order_Details**

| OrderID | Customer | Address |
|---|---|---|
| 001 | John | NY, USA |
| 001 | John | NY, USA |
| 002 | Alice | CA, USA |

**Item_Details**

| OrderID | Item Ordered |
|---|---|
| 001 | Laptop |
| 001 | Mouse |
| 002 | Phone |

**Step 3: Convert to 2NF (Second Normal Form)**

**Rules of 2NF:**

- Must be in 1NF.
- Remove partial dependencies (i.e., every non-key attribute should depend on the whole primary key).

**Issue in 1NF Table:**

- Customer and Address depend only on OrderID, **not** on Item Ordered.
- **Partial Dependency** exists.

**Let's Split the table into separate Orders and OrderDetails tables.**

**Orders Table (Stores unique orders)**

| OrderID | Customer | Address |
|---|---|---|
| 001 | John | NY, USA |
| 002 | Alice | CA, USA |

**OrderDetails Table (Handles ordered items separately)**

| OrderID | Item Ordered |
|---|---|
| 001 | Laptop |
| 001 | Mouse |
| 002 | Phone |

**Now the table follows 2NF rules.**

**Step 4: Convert to 3NF (Third Normal Form)**

**Rules of 3NF:**

- Must be in 2NF.
- Remove transitive dependencies (i.e., non-key attributes must depend only on the primary key).

**Issue in 2NF Tables:**

- Address depends on Customer, **not** directly on OrderID.
- **Transitive Dependency** exists.

Solution: Move Customer and Address to a separate Customers table.

**Customers Table (To Remove the Transitive Dependency)**

| CustomerID | Customer | Address |
|---|---|---|
| C001 | John | NY, USA |
| C002 | Alice | CA, USA |

**Orders Table (References Customer ID)**

| OrderID | CustomerID |
|---|---|
| 001 | C001 |
| 002 | C002 |

**OrderDetails Table{OrderID, Item Ordered}**

**# Remains Same as 2NF Table**

**Now the database is in 3NF.**

**17. Design a 3NF database for an Online Bookstore.**

**Unnormalized Table (UNF)**

The unnormalized table contains repeating groups and multi-valued attributes in the BooksOrdered column.

| OrderID | CustomerName | Email | Address | BooksOrdered | Price | Author |
|---|---|---|---|---|---|---|
| 001 | John Doe | john@example.com | NY, USA | "Book A, Book B" | "10, 15" | "Author X, Y" |
| 002 | Alice Smith | alice@example.com | CA, USA | "Book C" | "20" | "Author Z" |

**Issues in UNF:**

- BooksOrdered, Price, and Author have multiple values in a single cell (not atomic).
- Repeating groups (one order contains multiple books)

**First Normal Form (1NF)**

**Rules of 1NF:**

- Each column must have atomic values (no multiple values in a single cell).
- There should be a primary key.

| OrderID | CustomerName | Email | Address | BookOrdered | Price | Author |
|---|---|---|---|---|---|---|
| 001 | John Doe | john@example.com | NY, USA | Book A | 10 | Author X |
| 001 | John Doe | john@example.com | NY, USA | Book B | 15 | Author Y |
| 002 | Alice Smith | alice@example.com | CA, USA | Book C | 20 | Author Z |

**Now, every column contains atomic values (no multi-valued attributes).**

**Second Normal Form (2NF)**

**Rules of 2NF:**

- Must be in 1NF.
- Remove partial dependencies (every non-key attribute must depend on the whole primary key).
- Identify the composite key.

**Issue in 1NF:**

- CustomerName, Email, and Address depend only on OrderID, not on BookOrdered.
- Author depends only on BookOrdered, not on OrderID.

**Solution: Create Separate Tables**

**Customers Table (Removes Partial Dependency)**

| CustomerID | CustomerName | Email | Address |
|---|---|---|---|
| **1** | **John Doe** | **john@example.com** | **NY, USA** |
| **2** | **Alice Smith** | **alice@example.com** | **CA, USA** |

**Orders Table**

| OrderID | CustomerID | OrderDate |
|---|---|---|
| **001** | **1** | **2024-03-10** |
| **002** | **2** | **2024-03-11** |

**Books Table (Separates Book Information)**

| BookID | Title | AuthorID | Price |
|---|---|---|---|
| **B001** | **Book A** | **A001** | **10** |
| **B002** | **Book B** | **A002** | **15** |
| **B003** | **Book C** | **A003** | **20** |

**OrderDetails Table (Many-to-Many Between Orders & Books)**

| OrderID | BookID | Quantity |
|---|---|---|
| 001 | B001 | 1 |
| 001 | B002 | 1 |
| 002 | B003 | 1 |

**Authors Table (Removes Partial Dependency)**

| AuthorID | AuthorName |
|---|---|
| A001 | Author X |
| A002 | Author Y |
| A003 | Author Z |

**Now, every non-key column depends only on the whole primary key.**

**Third Normal Form (3NF)**

**Rules of 3NF:**

- Must be in 2NF.
- Remove transitive dependencies (non-key attributes must depend only on the primary key).

**Issue in 2NF:**

- AuthorName depends on AuthorID, not BookID (Transitive Dependency).
- Address depends on CustomerID, not OrderID (Transitive Dependency).

**Solution: Create Separate Tables**

**Customers Table (No Transitive Dependencies)**

| CustomerID | CustomerName | Email | Address |
|---|---|---|---|
| 1 | John Doe | john@example.com | NY, USA |
| 2 | Alice Smith | alice@example.com | CA, USA |

**Orders Table**

| OrderID | CustomerID | OrderDate |
|---|---|---|
| 001 | 1 | 2024-03-10 |
| 002 | 2 | 2024-03-11 |

**Books Table**

| BookID | Title | AuthorID | Price |
|---|---|---|---|
| B001 | Book A | A001 | 10 |
| B002 | Book B | A002 | 15 |
| B003 | Book C | A003 | 20 |

**OrderDetails Table**

| OrderID | BookID | Quantity |
|---|---|---|
| 001 | B001 | 1 |
| 001 | B002 | 1 |
| 002 | B003 | 1 |

**Authors Table**

| AuthorID | AuthorName |
|---|---|
| A001 | Author X |
| A002 | Author Y |
| A003 | Author Z |

**Payments Table (Handles Payment Information)**

| PaymentID | OrderID | PaymentDate | AmountPaid | PaymentMethod |
|---|---|---|---|---|
| P001 | 001 | 2024-03-10 | 25.00 | Credit Card |
| P002 | 002 | 2024-03-11 | 20.00 | PayPal |

**Now, there are no transitive dependencies.**

**18. Explain the Process of Denormalization and When It Is Needed.**

**What is Denormalization?**

Denormalization is the process of combining tables in a database to improve read performance by reducing the need for joins. It is the opposite of normalization, where data redundancy is intentionally introduced for better query efficiency.

**Process of Denormalization**

1. **Identify performance bottlenecks** – Analyze queries that involve multiple joins and take too long.
2. **Merge frequently joined tables** – Reduce joins by adding redundant data from related tables.

3. **Add derived columns** – Precompute and store aggregated values instead of calculating them on the fly.
4. **Use indexing strategies** – Implement indexes to optimize access to denormalized tables.
5. **Balance normalization and performance** – Maintain a trade-off between redundancy and data integrity.

**When is Denormalization Needed?**

- For Read-Heavy Applications: Faster retrieval in reporting systems and data warehouses.
- For Performance Optimization : Reduces costly joins in high-traffic applications.
- For Caching & Aggregation : Precomputed totals (e.g., total sales) reduce recalculations.
- For NoSQL & Big Data Use Cases : Denormalized data helps in distributed databases for faster access.

**19. What is 1NF Violation? How Do You Fix It?**

**What is 1NF Violation?**

A 1NF violation occurs when a table contains repeating groups or multi-valued attributes (i.e., a column has multiple values in a single cell).

**Example of 1NF Violation**

| OrderID | Customer | Items Ordered |
|---|---|---|
| 001 | John | Laptop, Mouse |
| 002 | Alice | Phone |

The "**Items Ordered**" column contains multiple values (Laptop, Mouse), violating 1NF.

**Convert multi-valued attributes into separate rows.**

| OrderID | Customer | Item Ordered |
|---|---|---|
| **001** | **John** | **Laptop** |
| **001** | **John** | **Mouse** |
| **002** | **Alice** | **Phone** |

Now, each column has atomic values, ensuring 1NF compliance.

**20. Explain Database Anomalies and How Normalization Prevents Them.**

Database anomalies are inconsistencies or errors in a poorly designed database. These anomalies occur due to redundancy and poor organization of data.

**Types of Database Anomalies**

**Insertion Anomaly** : Occurs when inserting data is difficult due to missing related information.
**Update Anomaly** : Occurs when changing one value requires multiple updates.
**Deletion Anomaly** : Occurs when deleting data unintentionally removes essential information.

**How Normalization Prevents Anomalies?**

- **1NF** – Removes multi-valued attributes (fixes redundancy & insertion anomalies).
- **2NF** – Eliminates **partial dependencies** (fixes update anomalies).
- **3NF** – Removes **transitive dependencies** (fixes deletion anomalies).
- **BCNF** – Ensures candidate keys properly determine attributes. This way Normalization improves data integrity, prevents inconsistencies, and enhances database efficiency.

**21. Normalize a Hotel Booking Database to 3NF.**

**Step 1. Unnormalized Table (UNF) :** This table contains repeating groups and multi-valued attributes.

| **BookingID** | **CustomerName** | **Phone** | **RoomNo** | **RoomType** | **CheckIn** | **CheckOut** | **ServicesUsed** | **TotalAmount** |
|---|---|---|---|---|---|---|---|---|
| B001 | John Doe | 9876543210 | 101 | Deluxe | 2024-03-10 | 2024-03-12 | "Breakfast, Laundry" | 5000 |
| B002 | Alice Smith | 8765432109 | 102 | Standard | 2024-03-11 | 2024-03-13 | "Gym" | 3000 |

**Issues:**

- ServicesUsed contains multiple values (not atomic).
- Repeating groups (same customer may use multiple services).

**Step 2. Convert to First Normal Form (1NF)**

**Rules of 1NF:**

- Ensure atomicity (no multiple values in a single cell).
- Each row must be unique.

**Convert Multi-Valued Attributes into Separate Rows**

| BookingID | CustomerName | Phone | RoomNo | RoomType | CheckIn | CheckOut | ServiceUsed | TotalAmount |
|---|---|---|---|---|---|---|---|---|
| B001 | John Doe | 9876543210 | 101 | Deluxe | 2024-03-10 | 2024-03-12 | Breakfast | 5000 |
| B001 | John Doe | 9876543210 | 101 | Deluxe | 2024-03-10 | 2024-03-12 | Laundry | 5000 |
| B002 | Alice Smith | 8765432109 | 102 | Standard | 2024-03-11 | 2024-03-13 | Gym | 3000 |

**Now, each cell contains atomic values.**

**SQL Query for 1NF :**

```
CREATE TABLE Customers (
    CustomerID INT PRIMARY KEY AUTO_INCREMENT,
    CustomerName VARCHAR(100),
    Phone VARCHAR(15) UNIQUE
);
CREATE TABLE Rooms (
    RoomID INT PRIMARY KEY AUTO_INCREMENT,
    RoomNo INT UNIQUE,
    RoomType VARCHAR(50)
);
CREATE TABLE Bookings (
    BookingID INT PRIMARY KEY AUTO_INCREMENT,
    CustomerID INT,
    RoomID INT,
    CheckIn DATE,
```

```
    CheckOut DATE,
    TotalAmount DECIMAL(10,2),
    FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID),
    FOREIGN KEY (RoomID) REFERENCES Rooms(RoomID)
);
CREATE TABLE Services (
    ServiceID INT PRIMARY KEY AUTO_INCREMENT,
    ServiceName VARCHAR(100)
);
CREATE TABLE BookingServices (
    BookingServiceID INT PRIMARY KEY AUTO_INCREMENT,
    BookingID INT,
    ServiceID INT,
    FOREIGN KEY (BookingID) REFERENCES Bookings(BookingID),
    FOREIGN KEY (ServiceID) REFERENCES Services(ServiceID)
);
```

**Step 3. Convert to Second Normal Form (2NF)**

**Rules of 2NF:**

- Must be in **1NF**.
- Remove **partial dependencies** (every non-key attribute must depend on the **whole primary key**).

**Issue in 1NF:**

- CustomerName and Phone depend only on **BookingID**, not on **RoomNo**.
- RoomType depends only on **RoomNo**, not on **BookingID**.
- **Partial Dependency exists** → Needs separate tables.

**Solution: Create Separate Tables**

- Move **Customers** to a new table.

- Move **Rooms** to a new table.
- Create a **Bookings Table** to link CustomerID & RoomID.

Now, **Customers**, **Rooms**, and **Services** are separate tables, and no partial dependencies exist.

**Step 4. Convert to Third Normal Form (3NF)**

**Rules of 3NF:**

- Must be in 2NF.
- Remove transitive dependencies (a non-key column must depend only on the primary key).

**Issue in 2NF:**

- RoomType depends on RoomNo, not on BookingID (Transitive Dependency).
- TotalAmount depends on BookingID, but it's derived from Room Price & Services Used.

**Solution:**

- Move RoomType to Rooms Table.
- Create a Payments Table to track payments separately.

**SQL Query for 3NF**

```
CREATE TABLE Payments (
   PaymentID INT PRIMARY KEY AUTO_INCREMENT,
   BookingID INT,
   PaymentDate DATE,
   PaymentMethod VARCHAR(50),
   AmountPaid DECIMAL(10,2),
   FOREIGN KEY (BookingID) REFERENCES Bookings(BookingID)
);
```

**Now, all transitive dependencies are removed. Now, the database is fully normalized (3NF) and efficient.**