What is Normalization?

Normalization is the process of **organizing data** in a database to reduce **redundancy** and improve **data integrity**.

A properly normalized database should have:

- Scalar (atomic) values in each field
- Minimal redundancy
- Minimal null values
- Minimal information loss

It's a **multi-step process** that transforms an unnormalized relation into a set of smaller, related tables.

Anomalies in Unnormalized Databases

1. Insertion Anomaly

Occurs when new data **cannot be inserted** without adding redundant or unrelated data.

> Example: You can't add a new module unless there's a student enrolled in it.

2. Deletion Anomaly

Occurs when deleting some data **accidentally removes** valuable related data.

> Example: If a student is deleted, the lecturer info might also be lost.

3. Update (Modification) Anomaly

Occurs when the same data appears multiple times and needs to be updated everywhere.

> Example: Changing a lecturer's email in one record but forgetting to change others causes inconsistency.

Solution — Multiple Tables

Normalization separates data into related tables:

- Removes redundancy
- Maintains data consistency
- Ensures logical data dependencies

Example:

Student Table	Module Table
Student No, Name, Module	Module, Lecturer, Email

Now, if a lecturer's email changes, it's updated in one place.

Functional Dependencies (FDs)

Definition:

A set of attributes **X functionally determines** another set **Y** if each X value uniquely determines a Y value.

Written as:

> X → Y

Example FDs:

- NIN → EmployeeName
- PNumber → {PName, PLocation}
- {NIN, PNumber} → HoursWorked

Formal Definition:

If two tuples have the same value for X, they must have the same value for Y.

> If $t_1[X] = t_2[X]$, then $t_1[Y] = t_2[Y]$

Functional dependencies come from **real-world rules** about data relationships.

Example (Conference Database)

- **Author Table:**

  AuthNo → AuthName, AuthEmail, AuthAddress

- **Paper Table:**

  PaperNo → PrimaryAuthNo, Title, Abstract, Status

- **Reviewer Table:**

  RevNo → RevName, RevEmail, RevAddress

- **Review Table:**

  {RevNo, PaperNo} → AuthComment, ProgComment, Date, Ratings

Normal Forms Used: 1NF, 2NF, 3NF

Normal Form	Requirement	Dependency Type
1NF	Atomic values only	Functional dependency of non-key attributes on key
2NF	No partial dependency	Full functional dependency on the primary key
3NF	No transitive dependency	Non-key attributes depend only on key

---

Unnormalized Relations

Contain **repeating groups or lists** within a single field.

Example:

Student No	Name	Module
1	Kehinde	CO2102, CO2103
2	Karim	CO3201, CO3001

First Normal Form (1NF)

A table is in 1NF if:

- All attributes have **atomic (indivisible)** values.
- No repeating groups.

Convert repeating values into **separate rows**:

Student No	Name	Module
1	Kehinde	CO2102
1	Kehinde	CO2103
2	Karim	CO3201

Second Normal Form (2NF)

A table is in 2NF if:

- It's already in **1NF**
- Every **non-key attribute** is **fully functionally dependent** on the **whole primary key**

> 2NF removes partial dependencies (where an attribute depends on only part of a composite key).

If there's no composite primary key, the table is **automatically in 2NF**.

Example:

Score Table:

Student No	Module ID	Marks	Convenor
6	1	64	Kehinde
6	2	70	ABC

Here, *Convenor* depends only on *Module ID* — not the whole composite key.

Move *Convenor* to the **Module Table**.

Now:

- **Score Table:** (StudentNo, ModuleID, Marks)
- **Module Table:** (ModuleID, ModuleName, Convenor)

Third Normal Form (3NF)

A table is in 3NF if:

- It's already in **2NF**
- There are **no transitive dependencies** between non-key attributes

> A transitive dependency occurs when a non-key attribute depends on another non-key attribute.

Example:

OrderNo (PK)	Customer	ContactPerson	Total
1	Acme	Dave	20

Here:

- *Customer* → *ContactPerson* (transitive dependency)

Split into:

- **Orders Table:** OrderNo, Customer, Total
- **Customer Table:** Customer, ContactPerson

Summary Table

Normal Form	Condition	Removes
1NF	No repeating groups	Repetition within a column
2NF	No partial dependencies	Dependency on part of composite key
3NF	No transitive dependencies	Dependency among non-key attributes

SQL Components

SQL has three main sub-languages:

- **DDL (Data Definition Language)** – defines database structure.

  Commands: `CREATE`, `ALTER`, `DROP`, `TRUNCATE`.

- **DML (Data Manipulation Language)** – manages data inside tables.

Commands: `SELECT`, `INSERT`, `UPDATE`, `DELETE`.

- **DCL (Data Control Language)** – controls access and permissions.

  Commands: `GRANT`, `REVOKE`, `COMMIT`, `ROLLBACK`.

SQL Database Definition (DDL)

Major CREATE statements:

- `CREATE SCHEMA` – defines a user's part of the DB.
- `CREATE TABLE` – defines a new table and its columns.
- `CREATE VIEW` – creates a virtual (logical) table.

Other statements: `CREATE CHARACTER SET`, `COLLATION`, `DOMAIN`, `ASSERTION`.

Domain Types in SQL

Steps in Table Creation

Type	Description
`char(n)`	Fixed-length string
`varchar(n)`	Variable-length string
`int`, `smallint`	Integer types
`numeric(p,d)`	Fixed-point with precision `p`, `d` digits after decimal
`float(n)`	Floating-point with precision `n`

---

1. Identify data types.
2. Decide NULL / NOT NULL.
3. Define unique & candidate keys.
4. Set **primary** and **foreign keys**.
5. Assign default values.
6. Apply constraints.
7. Create table & indexes.

CREATE TABLE Syntax

```
CREATE TABLE table_name (
  column1 datatype,
  column2 datatype,
  ...
);
```

Example:

```
CREATE TABLE instructor (
  ID CHAR(5),
  name VARCHAR(20),
  dept_name VARCHAR(20),
  salary NUMERIC(8,2)
);
```

SQL Syntax Rules

- Keywords: Uppercase (not case-sensitive).
- Text in `'single quotes'`.
- Use `,` to separate fields and `;` to end statements.
- Parentheses `()` for grouping.

Tables Explained

Schema: Table name + attributes.

Example: `Product(PId, Price, Category, Manufacturer)`

Key: Attribute(s) with unique values.

Check structure:

```
DESCRIBE Product;
```

Remove table:

```
DROP TABLE Product;
```

Inserting Data

```
INSERT INTO Product VALUES (1001, 19.99, 'Gadgets', 'GizmoWorks');
```

If some values are missing:

```
INSERT INTO Product (Price, Category) VALUES (49.99, 'Phone');
```

→ Missing values become `NULL`.

SELECT Statement

Used to retrieve data.

Syntax:

```
SELECT columns
FROM table
WHERE condition
GROUP BY columns
HAVING condition
ORDER BY columns;
```

Processing order: FROM → WHERE → GROUP BY → HAVING → SELECT → ORDER BY.

SELECT Examples

- **All columns:** `SELECT * FROM Product;`
- **Specific columns:** `SELECT Manufacturer FROM Product;`
- **Distinct values:** `SELECT DISTINCT Category FROM Product;`

- **Conditional:** `SELECT * FROM Product WHERE Price > 100;`
- **Range:** `SELECT * FROM Product WHERE Price BETWEEN 0 AND 30;`
- **Pattern matching:**
  - `%` = any characters
  - `_` = one character

```
SELECT * FROM Product WHERE Category LIKE '%Phone%';
```

Aggregate Functions

Used to summarize data:

Function	Meaning
`AVG()`	Average value
`MIN()`	Minimum value
`MAX()`	Maximum value
`SUM()`	Total sum
`COUNT()`	Count of values

Example:

```
SELECT COUNT(Manufacturer) AS Num, MAX(Price), MIN(Price)
FROM Product;
```

With expressions:

```
SELECT SUM(price * quantity)
FROM Purchase
WHERE product = 'Apple';
```

GROUP BY & HAVING

- **GROUP BY:** groups rows for aggregation.
- **HAVING:** filters groups (like WHERE filters rows).

Example:

```
SELECT product, SUM(price*quantity) AS TotalSales
FROM Purchase
WHERE date > '10/1/2005'
GROUP BY product
HAVING SUM(price*quantity) > 20;
```

ORDER BY

Sorts results (default ASC, use DESC for descending).

```
SELECT PId, Price, Manufacturer
FROM Product
WHERE Category = 'Gadgets'
ORDER BY Price DESC;
```

UPDATE Statement

```
UPDATE Product
SET Manufacturer = 'LG'
WHERE PId = 1001;
```

Omit WHERE → updates all rows.

DELETE Statement

```
DELETE FROM Product WHERE Manufacturer = 'LG'; -- specific rows
DELETE FROM Product;                           -- all rows
```

```
DROP TABLE Product;                    -- removes entire table
```

NULL Values

Used when value is **unknown**, **missing**, or **not applicable**.

- Any arithmetic or comparison with `NULL` → `NULL` or `UNKNOWN`.
- Attributes can be declared nullable or not.