In SQL, normalization refers to the process of organizing a relational database to minimize redundancy & dependency by dividing large tables into smaller, more manageable ones.

Key Concepts of SQL Normalization:

1st Normal Form (1NF):- A table is in 1NF if it has a primary key & each column contains atomic (indivisible) values.
In other words, there should be no repeating groups or arrays within a single column.

Not in 1NF:-

ID	Name	Phone Numbers
1	Alice	123-456, 987-432

Convert to 1NF

ID	Name	Phone Numbers
1	Alice	123-456
1	Alice	987-432

2nd Normal Form (2NF):- A table is in 2NF if is in 1NF & all non-key attributes are fully dependent on the primary key.
This removes partial dependency (i.e., when a non-key column is dependent only on part of a composite primary key).

OrderID	ProductID	P. Name	Quantity
1	101	Widget	2
1	102	Gadget	5

Not in 2NF

Product Name (P. Name) is only dependent on ProductID, not the composite primary key (OrderID, ProductID).

to convert to 2NF: Create a separate Products table:

Orders Table:

OrderID	ProductID	Quality
1	101	2
1	102	5

Product Table:-

ProductID	Product Name
101	Widget
102	Gadget

→ 3rd Normal Form (3NF): A table is in 3NF if it in 2NF & there is no transitive dependency.
In other-words, non-key attributes shoud not depend on other non-key attributes.

- Not in 3NF:-

S_ID	C_ID	Instructor Name	Instructor Phone
101	MATH101	Dr. Smith	123-4567
102	MATH 101	Dr. Smith	123-4567

Here, Instructor Name depends on Instructor Name, which is a non-key attribute.

- To convert to 3NF: Split into two tables:-

Student Courses Table.

S_ID	C_ID	Instructor Name
101	MATH101	Dr. Smith
102	MATH101	Dr. Smith

Instructors Tables:-

Instructor Name	Instructor Phone
Dr. Smith	123-4567

4) Boyce-Codd Normal Form (BCNF): A table is in BCNF if it is in 3NF & every determinant is a candidate key. This addresses situations where a non-primary attributes (not-part of the primary key) can determine a part of the primary key.

Not in BCNF:-

CourseID	Instructor	Department
MATH101	Dr. Smith	Math
MATH101	Dr. White	Physics

Department depends on Instructor, but Instructor is not a primary key, leading to a violation of BCNF.

Courses Tables:-

CourseID	Instructor
MATH101	Dr. Smith
MATH101	Dr. White

Instructors Tables:

Instructor	Department
Dr. Smith	Math
Dr. White	Physics

- **4th Normal Form (4NF):** A table is in 4NF if it is in BCNF & contains no multi-valued dependencies. This applies to situations where one attribute is dependent on multiple values of another attributes.

Not in 4NF :-

Student ID	Course	Hobby
101	Math	Basketball
101	Math	Painting

The table has two values for the Hobby attribute for the same student, creating a multivalued dependency.

To convert to 4NF :- Split into two tables :-

Student Courses Table :-

Student ID	Course
101	Math

Student Hobbies Table :-

Student ID	Hobby
101	Basketball
101	Painting

Benefits of SQL Normalization :-

Reduced Redundancy :- By breaking down large tables, we minimize duplication of data (e.g. a student's name doesn't need to be repeated for every course they take).

Data Integrity : It enforces consistency & avoids anomalies during insert, update, & delete operations

Easier Maintenance : It is easier to modify & extend a normalized database (e.g., adding new features or changing structures).

Trade-offs :

Performance : A highly normalized database might require more complex queries involving multiple joins, which can impact performance.

Over-Normalization : Going too far with normalization (especially to 5NF or higher) can lead to excessive fragmentation of data, making queries more complex than nessary.

ACID stands for (Atomicity Consistency Isolation Durability).

ACID properties are essential for ensuring that dabase transaction are processed reliably & that the database remains in a valid state, even in the event of system crashes, power failures or other issues.

Atomicity:- A transaction is atomic, meaning it is an indivisible unit of work. A transaction either completes in full (commit) or has no effect at all (rollback).

Example:- Suppose a bank transaction involves transfering money from one ~~at~~ account to another. If part of the ~~tran~~ transaction fails (e.g. the withdrawal successeds but the deposit fails), the entire transaction will be rolled back, ensuring that no money is lost.

Atomicity ensures that if an error occurs during a transaction, the database is returned to its previous state, maintaining data integrity.

Consistency:- A transaction takes the database frome one consistent state to another consistent state. It ensures that the data in the database follows all the predefined rules, constraints & relationships (e.g. primary key, foreign key, etc.)

Example:- If a database has a rule that the balance is an account cannot be negative, consistency ensures that this rule is ~~mained~~ maintaied. If a transaction violates this rule (e.g. ~~attemp~~ attempting to withdraw more money than the balance). the transaction will fail, & the database will remain consistent.

Usage:- Consistency ensures that no data is left in a invalid state. After a transaction, the database must always comply with all integrity constraints, such as datatypes, referential integrity & more.

Isolation:- Isolation ensures that transactions are executed in isolation from one another. Even through multiple transactions may be executed concurrently, the result will be the same as if they were executed one after another, in some serial order.

Example:- Suppose 2 people are transferring money from different accounts at the same time. Isolation ensures that their ~~tran~~ transactions do not interface with each other, such as one transaction reading data that is halfway through the other transaction.

Usage:- Isolation prevents problems such as dirty reads (reading uncommitted data), non-repeatable reads (data changes b/w reads within a transaction). & phantom reads (new records appearing during a transaction). The level of isolation can be configured using isolation levels such as Read Uncommitted, Read Committed, Repeatable-Read, and Serializable.

Durability:- Durability ensures that once a transaction has been committed, it is permanent & survives & any system failures, such as power outages or crashes.

Example:- After a bank transaction is completed (e.g. transferring funds from one account to another), the changes are saved to the database. Even if the system crashes immediately after the transaction, the data will be not be lost. When the system recovers, the changes are still intact.

Usage:- Durability garantees that committed transactions are written to non-volatile storage, ensuring that data is not lost, even in case of failure. This property is achieved through techniques like database logs & write-ahead logging (WAL).

Why ACID:-

Data Integrity, Reliability, Concurrency, Recovery.

Correlated Subquery in MySQL.

A correlated subquery is a type of subquery where are the subquery depends on the outer query for its values.

Unlike a regular (non-correlated) subquery, which is independent can be executed on its own, a correlated subquery references columns from the outer query. This means the subquery is reevaluated for each row of the outer query.

Key Characteristics of Correlated Subqueries:-

1. The subquery depends on the outer query.
2. The subquery is executed once for each row processed by the outer query.
3. It refeernce columns from the outer query in the subquery's WHERE or SELECT clause.

Syntax of Correlated Subquery.

```
Select column1, column2
From table1 AS t1
WHERE ColumnX = (SELECT columnY
                 FROM table2 AS t2
                 WHERE t2.columnZ = t1.columnZ);
```

Sol

```
SELECT e.name, e.salary, e.department-id
FROM employees e
WHERE e.salary > (
        Select AVG(e2.salary)
        FROM employees e2
        WHERE e2.department_id = e.department_id
      );
```

id	name	department_id	Salary
1	Alice	101	5000
2	Bob	102	6000
3	Charolle	101	7000
4	David	103	5500

department_id	department_name
101	Sales
102	Marketing
103	HR