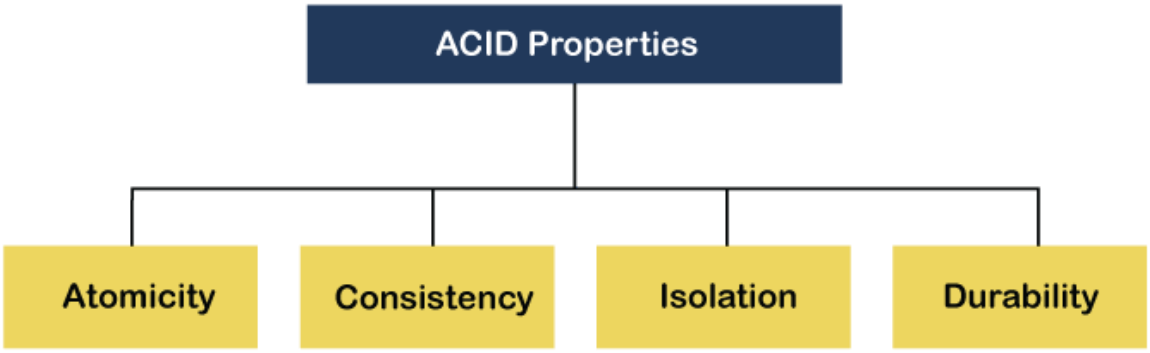
ACID Properties

The expansion of the term ACID defines for:

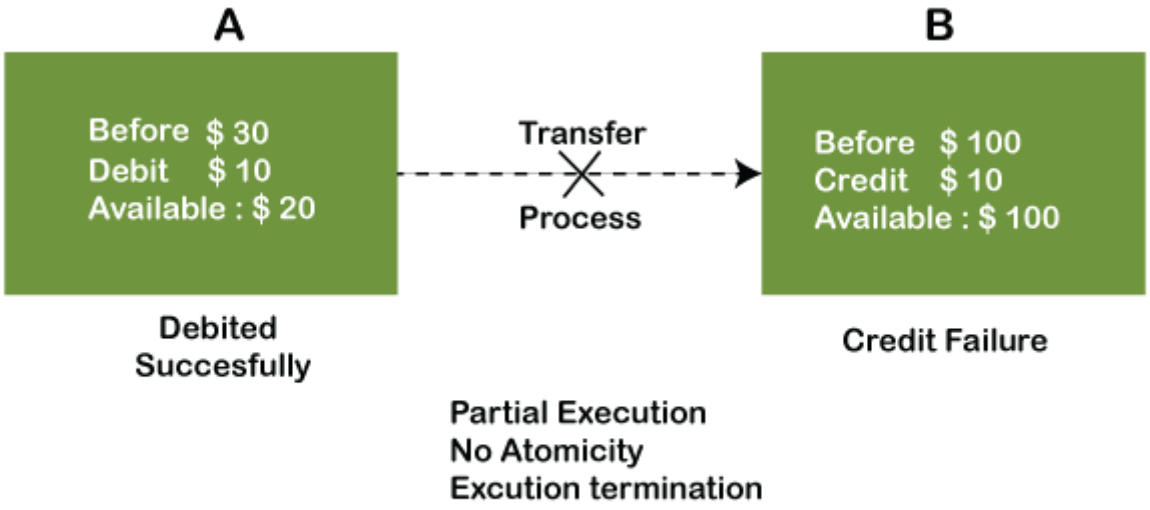


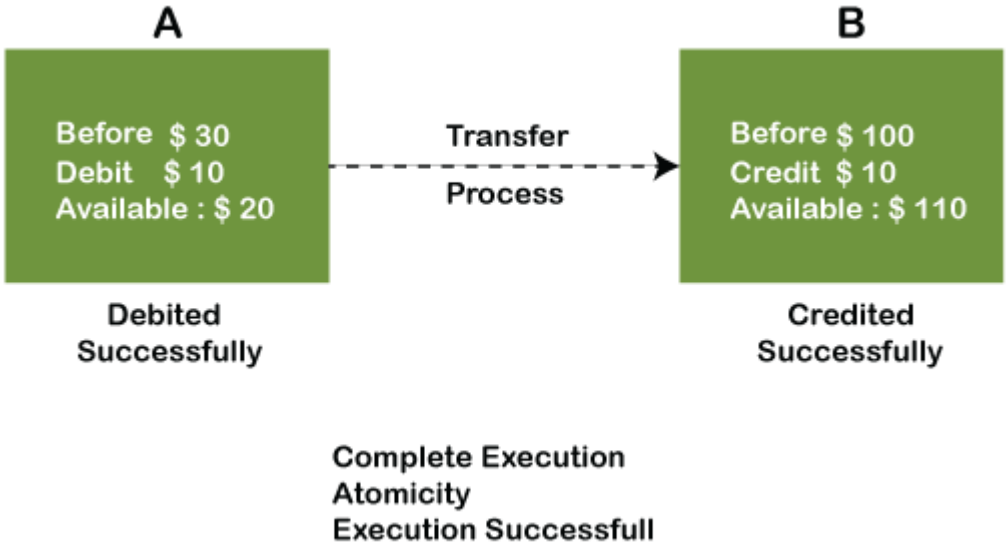
1) Atomicity

The term atomicity defines that the data remains atomic.

It means if any operation is performed on the data, either it should be performed or executed completely or should not be executed at all.

It further means that the operation should not break in between or execute partially. In the case of executing operations on the transaction, the operation should be completely executed and not partially.



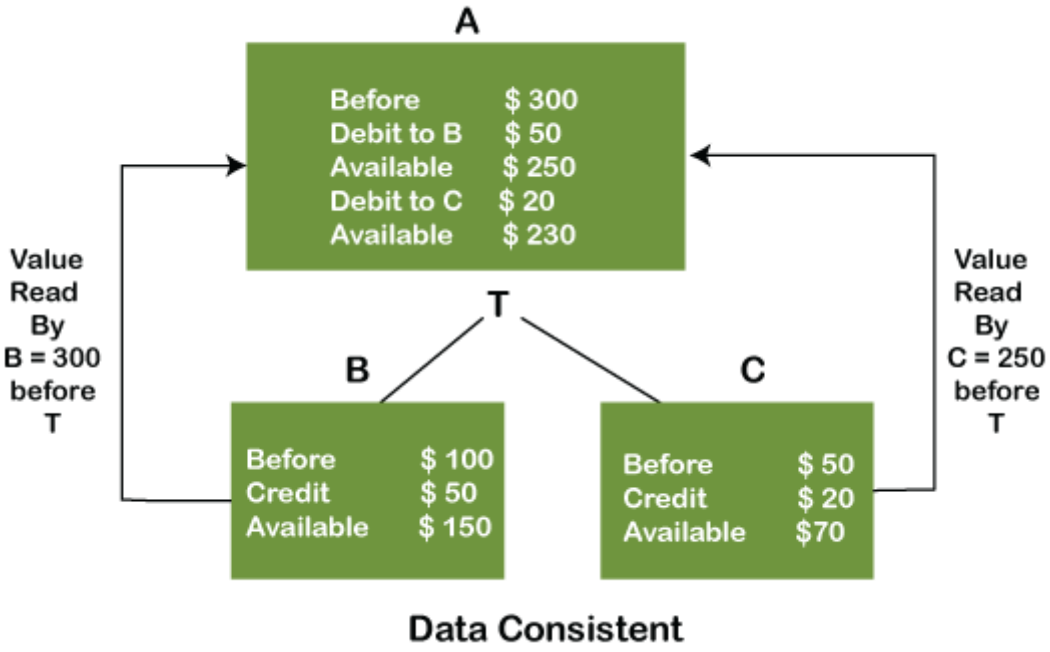


2) Consistency

The word **consistency** means that the value should remain preserved always.

In DBMS, the integrity of the data should be maintained, which means if a change in the database is made, it should remain preserved always.

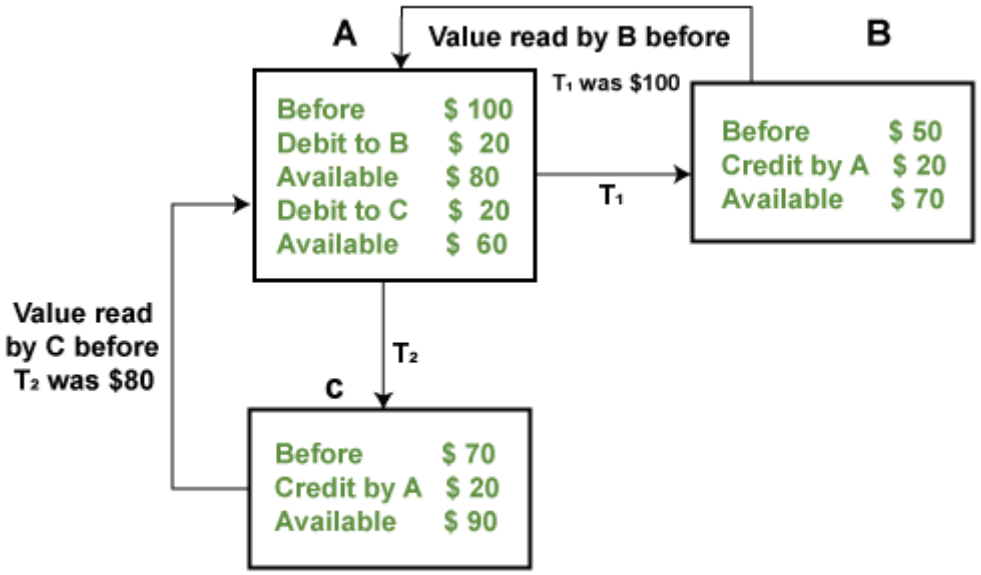
In the case of transactions, the integrity of the data is very essential so that the database remains consistent before and after the transaction. The data should always be correct.



3) Isolation

The term 'isolation' means separation. In DBMS, Isolation is the property of a database where no data should affect the other one and may occur concurrently. In short, the operation on one database should begin when the operation on the first database gets complete. It means if two operations are being performed on two different databases, they may not affect the value of one another. In the case of transactions, when two or more transactions occur simultaneously, the consistency should remain maintained. Any changes that occur in any particular transaction will not be seen by other transactions until the change is not committed in the memory.

Example: If two operations are concurrently running on two different accounts, then the value of both accounts should not get affected. The value should remain persistent. As you can see in the below diagram, account A is making T1 and T2 transactions to account B and C, but both are executing independently without affecting each other. It is known as Isolation.



Isolation - Independent execution of T_1 & T_2 by A

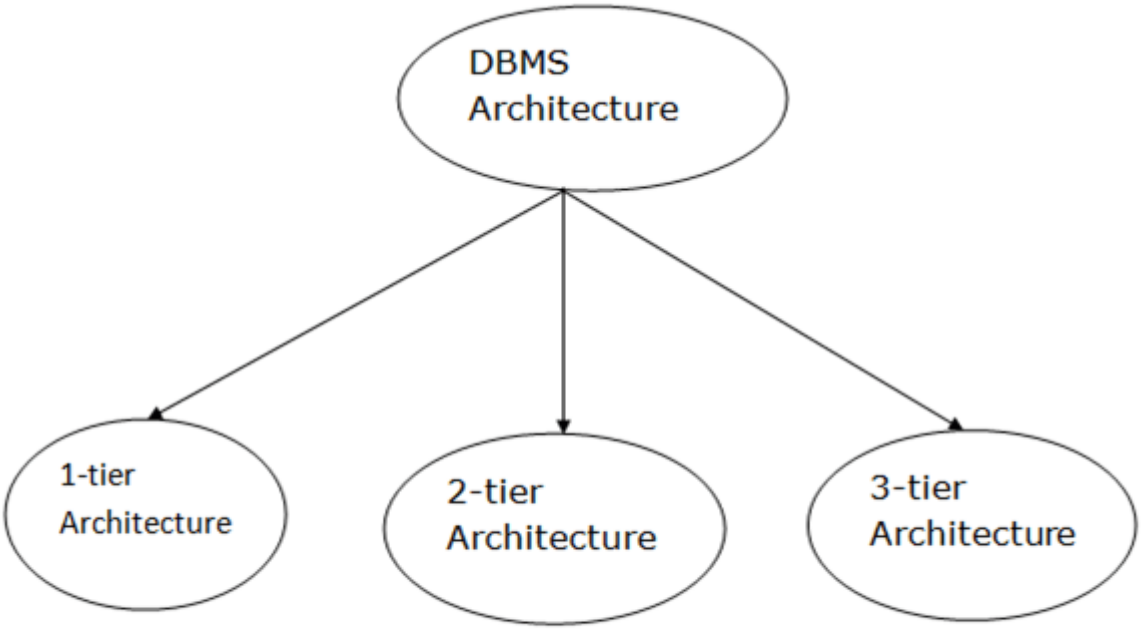
4) Durability

Durability ensures the permanency of something. In DBMS, the term durability ensures that the data after the successful execution of the operation becomes permanent in the database. The durability of the data should be so perfect that even if the system fails or leads to a crash, the database still survives. However, if gets lost, it becomes the responsibility of the recovery manager for ensuring the durability of the database. For committing the values, the COMMIT command must be used every time we make changes.

DBMS Architecture

- The DBMS design depends upon its architecture. The basic client/server architecture is used to deal with a large number of PCs, web servers, database servers and other components that are connected with networks.
- The client/server architecture consists of many PCs and a workstation which are connected via the network.
- DBMS architecture depends upon how users are connected to the database to get their request done.

Types of DBMS Architecture



Database architecture can be seen as a single tier or multi-tier. But logically, database architecture is of two types like: **2-tier architecture** and **3-tier architecture**.

1-Tier Architecture

- In this architecture, the database is directly available to the user. It means the user can directly sit on the DBMS and uses it.
- Any changes done here will directly be done on the database itself. It doesn't provide a handy tool for end users.
- The 1-Tier architecture is used for development of the local application, where programmers can directly communicate with the database for the quick response.

2-Tier Architecture

- The 2-Tier architecture is same as basic client-server. In the two-tier architecture, applications on the client end can directly communicate with the database at the server side. For this interaction, API's like: **ODBC**, **JDBC** are used.
- The user interfaces and application programs are run on the client-side.
- The server side is responsible to provide the functionalities like: query processing and transaction management.
- To communicate with the DBMS, client-side application establishes a connection with the server side.

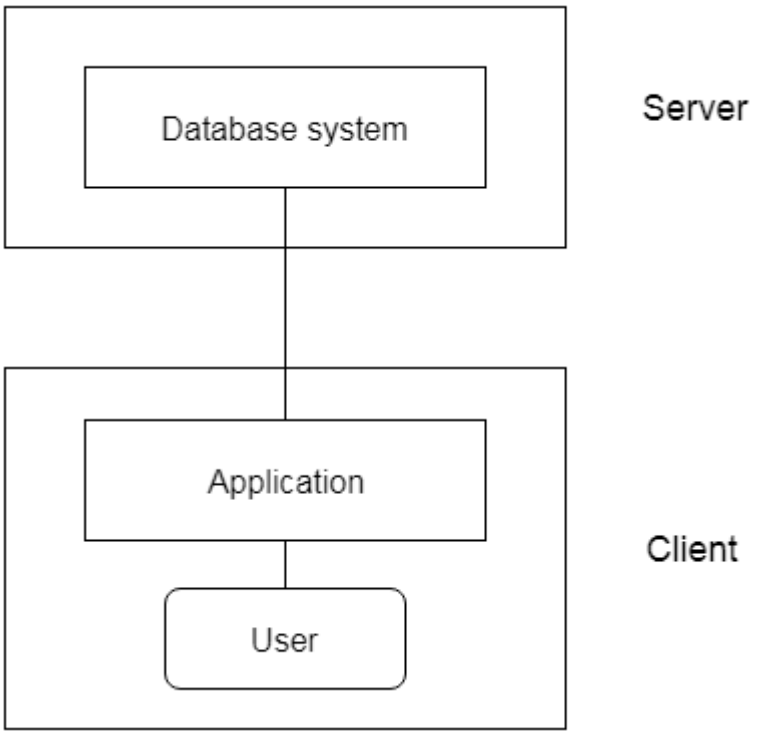


Fig: 2-tier Architecture

3-Tier Architecture

- The 3-Tier architecture contains another layer between the client and server. In this architecture, client can't directly communicate with the server.
- The application on the client-end interacts with an application server which further communicates with the database system.
- End user has no idea about the existence of the database beyond the application server. The database also has no idea about any other user beyond the application.

- The 3-Tier architecture is used in case of large web application.

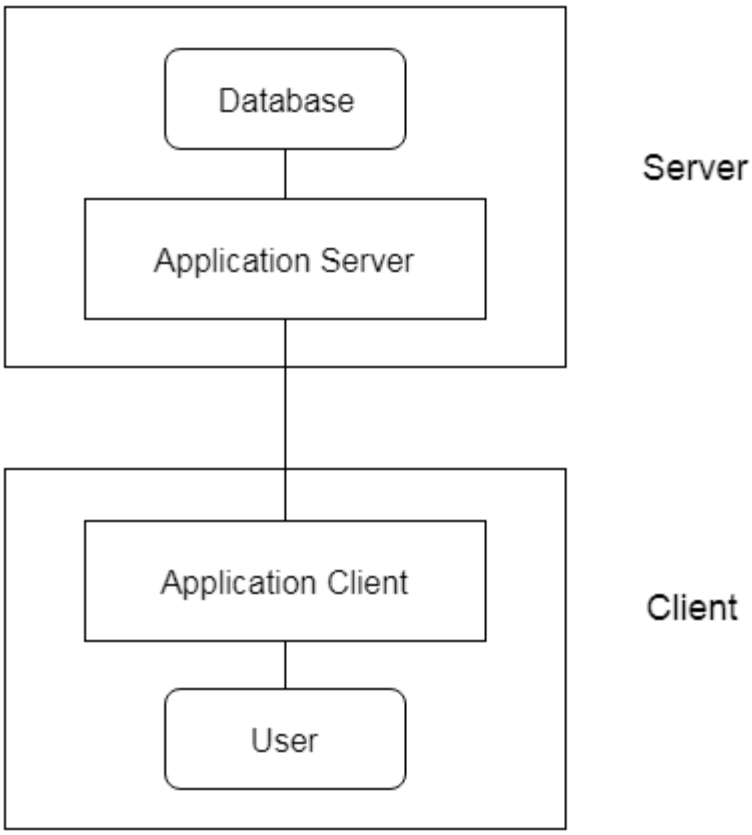


Fig: 3-tier Architecture

De-normalization in SQL

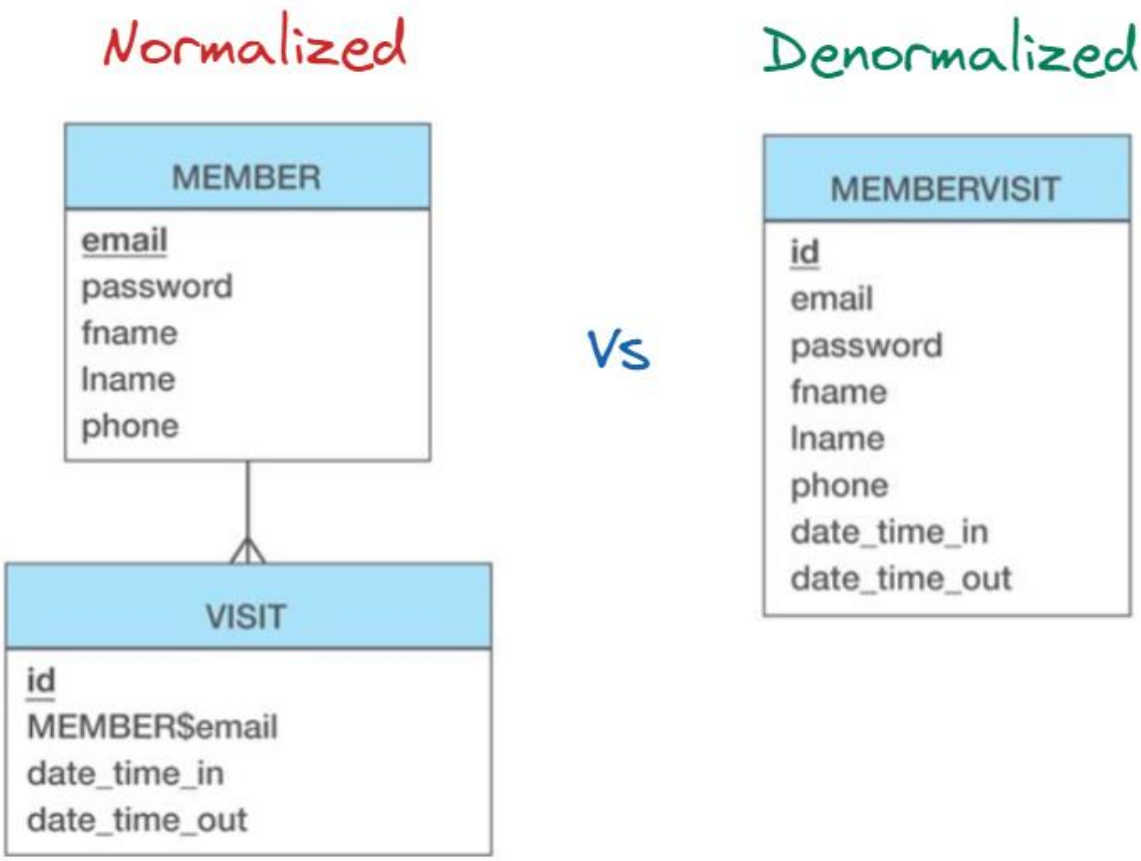
De-normalization is the process of optimizing the read performance of a database by adding redundant data or combining tables that were previously normalized. It involves reversing the normalization process to reduce the complexity of queries and improve performance, particularly in systems where read operations are more frequent than write operations.

Purpose of De-normalization

1. **Improved Query Performance**: Fewer joins mean faster query execution.
2. **Simpler Queries**: Easier to write and understand as data is pre-joined.
3. **Enhanced Scalability**: Reduces the computational overhead of complex joins.
4. **Reduced latency** : way to improve the speed of data transfer by reducing the time it takes for data to travel.

Differences Between Normalization and De-normalization

Aspect	Normalization	De-normalization
Focus	Minimizing redundancy and dependency	Improving read performance
Data Structure	Divides data into multiple tables	Combines data into fewer tables
Redundancy	Eliminates redundancy	Introduces controlled redundancy
Use Case	Transactional systems	Analytical systems or reporting databases
Performance Impact	Optimized for write operations	Optimized for read operations



Techniques of De-normalization

1. **Merging Tables**:
   - Combine two or more tables into one to reduce the need for joins.
2. **Adding Redundant Columns**:
   - Add columns from related tables to avoid frequent lookups or joins.
   - Example: Storing the customer name directly in the `Orders` table.
3. **Storing Aggregated Data**:
   - Pre-calculate and store summary or aggregated data, such as total sales or average ratings.
4. **Using Derived Tables**:
   - Create tables that are the result of pre-processed queries for frequently accessed reports.
5. **Replication of Data**:
   - Duplicate data across tables to avoid joins for specific use cases.

Advantages of De-normalization

- Faster query execution due to reduced joins.
- Simplified data retrieval process.
- Suitable for read-heavy applications like reporting tools and dashboards.

- Enhances scalability in distributed database systems.

Disadvantages of De-normalization

- Increased storage requirements due to redundancy.
- Greater risk of data anomalies and inconsistencies.
- Complexity in maintaining the data integrity during updates or deletions.
- Potentially slower write operations due to redundant data updates.

Example

Normalized Design:

1. **Customers** Table:

```
CustomerID   |   CustomerName
-----------------------------
1            |   Alice
2            |   Bob
```

6. **Orders** Table:

```
OrderID   |   CustomerID   |   Amount
-------------------------------------
101       |   1            |   500
102       |   2            |   300
```

De-normalized Design:

1. **Orders** Table:

```
OrderID   |   CustomerName   |   Amount
-----------------------------------------
101       |   Alice          |   500
102       |   Bob            |   300
```

Here, the `CustomerName` column is directly added to the `Orders` table, avoiding the need for a join.

When to Use De-normalization

- In analytical systems where read speed is crucial.
- When the cost of joins outweighs the storage concerns.
- In distributed systems where minimizing network latency is critical.
- For pre-aggregated data in reporting or dashboards.

De-normalization should be applied cautiously to ensure it aligns with the specific use case and system requirements.

What Is Normalization in SQL?

Normalization is the process to eliminate data redundancy and enhance data integrity in the table. Normalization also helps to organize the data in the database.

It is a multi-step process that sets the data into tabular form and removes the duplicated data from the relational tables.

Normalization organizes the columns and tables of a database to ensure that database integrity constraints properly execute their dependencies.

It is a systematic technique of decomposing tables to eliminate data redundancy (repetition) and undesirable characteristics like Insertion, Update, and Deletion anomalies.

.

1st Normal Form (1NF)

- **A table is referred to as being in its First Normal Form if atomicity of the table is 1.**
- **Here, atomicity states that a single cell cannot hold multiple values. It must hold only a single-valued attribute.**
- The First normal form disallows the multi-valued attribute, composite attribute, and their combinations.

Now you will understand the First Normal Form with the help of an example.

Below is a students' record table that has information about student roll number, student name, student course, and age of the student.

rollno	name	course	age
1	Rahul	c/c++	22
2	Harsh	java	18
3	Sahil	c/c++	23
4	Adam	c/c++	22
5	Lisa	java	24
6	James	c/c++	19
NULL	NULL	NULL	NULL

In the studentsrecord table, you can see that the course column has two values. Thus it does not follow the First Normal Form. Now, if you use the First Normal Form to the above table, you get the below table as a result.

rollno	name	course	age
1	Rahul	c	22
1	Rahul	c++	22
2	Harsh	java	18
3	Sahil	c	23
3	Sahil	c++	23
4	Adam	c	22
4	Adam	c++	22
5	Lisa	java	24
6	James	c	19
6	James	c++	19

By applying the First Normal Form, you achieve atomicity, and also every column has unique values.

2.Second Normal Form (2NF)

The first condition for the table to be in Second Normal Form is that the table has to **be in First Normal Form**. **The table should not possess partial dependency**. The partial dependency here means the proper subset of the candidate key should give a non-prime attribute.

Now understand the Second Normal Form with the help of an example.

Consider the table Location:

	cust_id	storeid	store_location
▶	1	D1	Toronto
	2	D3	Miami
	3	T1	California
	4	F2	Florida
	5	H3	Texas

The Location table possesses a composite primary key cust_id, storeid. The non-key attribute is store_location. In this case, store_location only depends on storeid, which is a part of the primary key. Hence, this table does not fulfill the second normal form.

To bring the table to Second Normal Form, you need to split the table into two parts. This will give you the below tables:

	cust_id	storeid
▶	1	D1
	2	D3
	3	T1
	4	F2
	5	H3

	storeid	store_location
▶	D1	Toronto
	D3	Miami
	T1	California
	F2	Florida
	H3	Texas

As you have removed the partial functional dependency from the location table, the column store_location entirely depends on the primary key of that table, storeid.

Now that you understood the 1st and 2nd Normal forms, you will look at the next part of this Normalization in SQL tutorial.

Third Normal Form (3NF)

- The first condition for the table to be in Third Normal Form is that the table should be in the Second Normal Form.
- The second condition is that there should be **no transitive dependency** for non-prime attributes, which indicates that non-prime attributes (which are not a part of the candidate key) should not depend on other non-prime attributes in a table. Therefore, a transitive dependency is a functional dependency in which A → C (A determines C) indirectly, because of A → B and B → C (where it is not the case that B → A).

- The third Normal Form ensures the reduction of data duplication. It is also used to achieve data integrity.

Below is a student table that has student id, student name, subject id, subject name, and address of the student as its columns.

stu_id	name	subid	sub	address
1	Arun	11	SQL	Delhi
2	Varun	12	Java	Bangalore
3	Harsh	13	C++	Delhi
4	Keshav	12	Java	Kochi

In the above student table, stu_id determines subid, and subid determines sub. Therefore, stu_id determines sub via subid. This implies that the table possesses a transitive functional dependency, and it does not fulfill the third normal form criteria.

Now to change the table to the third normal form, you need to divide the table as shown below:

stu_id	name	subid	address
1	Arun	11	Delhi
2	Varun	12	Bangalore
3	Harsh	13	Delhi
4	Keshav	12	Kochi

subid	subject
11	SQL
12	java
13	C++
12	Java

As you can see in both the tables, all the non-key attributes are now fully functional, dependent only on the primary key. In the first table, columns name, subid, and addresses only depend on stu_id. In the second table, the sub only depends on subid.