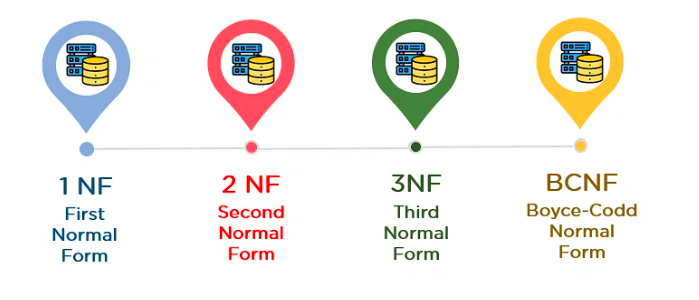
**CHAPTER 2 – Normalization & Denormalization**

**Normalization**

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.

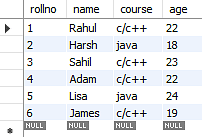
Normalization organizes the columns and tables of a database to ensure that database integrity constraints properly execute their dependencies. It is a systematic technique of decomposing tables to eliminate data redundancy (repetition) and undesirable characteristics like Insertion, Update, and Deletion anomalies.

**1st Normal Form (1NF)**

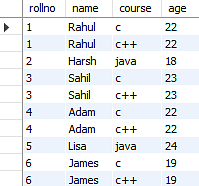
* A table is referred to as being in its First Normal Form if atomicity of the table is 1.
* Here, atomicity states that a single cell cannot hold multiple values. It must hold only a single-valued attribute.
* The First normal form disallows the multi-valued attribute, composite attribute, and their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



In the students record 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.



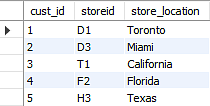
By applying the First Normal Form, you achieve atomicity, and also every column has unique values.

**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.

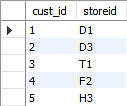
Now understand the Second Normal Form with the help of an example.

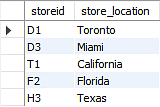
Consider the table Location:



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:





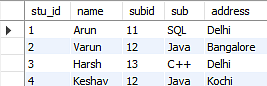
As you have removed the partial functional dependency from the location table, the column store\_location entirely depends on the primary key of that table, storeid.

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)**

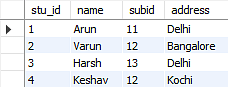
* The first condition for the table to be in Third Normal Form is that the table should be in the Second Normal Form.
* The second condition is that there should be no transitive dependency for non-prime attributes, which indicates that non-prime attributes (which are not a part of the candidate key) 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.

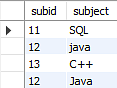
Below is a student table that has student id, student name, subject id, subject name, and address of the student as its columns.



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:





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.

**Boyce CoddNormal Form (BCNF)**

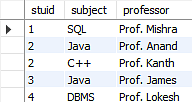
Boyce Codd Normal Form is also known as 3.5 NF. It is the superior version of 3NF and was developed by Raymond F. Boyce and Edgar F. Codd to tackle certain types of anomalies which were not resolved with 3NF.

The first condition for the table to be in Boyce Codd Normal Form is that the table should be in the third normal form. Secondly, every Right-Hand Side (RHS) attribute of the functional dependencies should depend on the super key of that particular table.

For example:

You have a functional dependency X → Y. In the particular functional dependency, X has to be the part of the super key of the provided table.

Consider the below subject table:



The subject table follows these conditions:

* Each student can enroll in multiple subjects.
* Multiple professors can teach a particular subject.
* For each subject, it assigns a professor to the student.

In the above table, student\_id and subject together form the primary key because using student\_id and subject; you can determine all the table columns.

Another important point to be noted here is that one professor teaches only one subject, but one subject may have two professors.

Which exhibit there is a dependency between subject and professor, i.e., subject depends on the professor's name.

The table is in 1st Normal form as all the column names are unique, all values are atomic, and all the values stored in a particular column are of the same domain.

The table also satisfies the 2nd Normal Form, as there is no Partial Dependency.

And, there is no Transitive Dependency; hence, the table also satisfies the 3rd Normal Form.

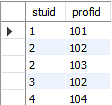
This table follows all the Normal forms except the Boyce Codd Normal Form.

As you can see stuid, and subject forms the primary key, which means the subject attribute is a prime attribute.

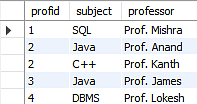
However, there exists yet another dependency - professor → subject.

BCNF does not follow in the table as a subject is a prime attribute, the professor is a non-prime attribute.

 To transform the table into the BCNF, you will divide the table into two parts. One table will hold stuid which already exists and the second table will hold a newly created column profid.



And in the second table will have the columns profid, subject, and professor, which satisfies the BCNF.



With this, you have reached the conclusion of the ‘Normalization in SQL’ tutorial.

# 2. DeNormalization

The intentional introduction of redundancy in a table in order to improve performance is called “Denormalization”. Denormalization is a technique to move from higher to lower normal forms of database modelling in order to speed up database access. De-normalization is the reverse process of Normalization i.e, to combine two or more tables into a single table.

**Example:**

Consider the following tables which maintains the details of the products and orders placed.

**Orders**  **Products**

|  |  |  |
| --- | --- | --- |
| **OrderNo.** | **ProductId** | **Qty** |
| 101 | P1 | 2 |
| 102 | P3 | 1 |
| 103 | P1 | 1 |
| 104 | P2 | 3 |
| 105 | P2 | 2 |

|  |  |  |
| --- | --- | --- |
| **ProductId** | **Desc** | **Cost** |
| P1 | XXX | 20 |
| P2 | YYY | 10 |
| P3 | ZZZ | 12 |

If you have calculated the total cost of each order placed as the cost of the product plus a tax of 10% of the product cost, the query to calculate the total cost sales as follows:

***select sum((cost\*qty)+(0.10\*cost\*qty)) from orders join products on orders.ProductId =products.ProductId***

If there are thousands of rows, the server will take a lot of time to process the query and return the results as there is a join and computation involved. Hence, to speed up the processing of the query, you may want to store the cost of each order along with the tax as follows in the Orders table,

**Orders**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **OrderNo.** | **ProductId** | **Qty** | **ProductCost** | **Tax** | **Ordercost** |
| 101 | P1 | 2 | 40 | 4 | 44 |
| 102 | P3 | 1 | 12 | 1.2 | 13.2 |
| 103 | P1 | 1 | 20 | 2 | 22 |
| 104 | P2 | 3 | 30 | 3 | 33 |
| 105 | P2 | 2 | 20 | 2 | 22 |

Now, to query on the total sales made, you would have to issue a simple query as below,

***Select SUM(Ordercost) from Orders***

This table structure has simplified the query and speeded up the processing of the query. By storing extra columns, you are introducing redundancy in the table but improving the performance of queries.

The decision to denormalize will obviously result in trade-off between performance and data integrity. Denormalization also increases disk space utilization.

Comparison of Normalization and Denormalization

|  |  |  |
| --- | --- | --- |
| Topics | Normalization | Denormalization |
| Definition | Process of organizing data in a database to minimize redundancy and dependency | Process of intentionally adding redundancy to a database to increase performance |
| Goal | Eliminate data redundancy, minimize update anomalies, and maintain data integrity | Increase query performance by reducing the number of joins required to retrieve data |
| Approach | Breaking down a table into smaller, more focused tables (higher normal forms) | Combining related tables or adding redundant data to a table |
| Benefits | Better data integrity, smaller table sizes, easier data maintenance | Faster data retrieval, reduced query complexity |
| Drawbacks | Higher number of tables, increased complexity of queries, may require more storage space | Increased risk of data inconsistencies, more complex data maintenance, increased storage requirements for redundant data |

**What is Data Redundancy?**

Data redundancy refers to the duplication of data within a database or information system.

In other words, it occurs when the same data is stored multiple times in different places, potentially leading to inconsistencies and inaccuracies if the data is not properly synchronized. This can occur for a variety of reasons, such as poor database design, data import/export processes, or application development practices.

Data redundancy can have several negative consequences, including:

1. Increased storage requirements.
2. Slower data retrieval times.
3. Increased risk of data inconsistencies or errors.

To avoid data redundancy, it is important to implement good data management practices, such as normalizing data structures, ensuring data integrity, and eliminating duplicate records.

**What is Data Integrity?**

Data integrity refers to the accuracy, consistency, and reliability of data throughout its entire life cycle.

This includes the process of data creation, storage, retrieval, and modification. Maintaining data integrity is essential for ensuring that data is trustworthy and can be used effectively for decision-making.

Data integrity can be compromised in several ways, such as human error, system failures, software bugs, hardware malfunctions, or intentional tampering. When data integrity is compromised, the data may become inaccurate, incomplete, or inconsistent, which can lead to errors, misunderstandings, and incorrect decisions.