**What is normalization in sql**

Database normalization is an important process used to organize and structure relational databases.

This process ensures that data is stored in a way that minimizes redundancy, simplifies querying, and improves data integrity.

Normalization is the process of organizing data within a database to eliminate redundancy.

Normalization involves breaking down a large, complex table into smaller and simpler tables while maintaining data relationships.

Normalization is commonly used when dealing with large datasets.

A major problem with redundant data is that it occupies unnecessary storage space. If we store the same product details in every order record, it leads to duplication. With normalization, we can eliminate redundancy by splitting data into separate tables.

**Why is Normalization in SQL Important?**

Normalization basically plays very important role in database design.

Reasons why normalization is essential:

1. **It Reduces redundancy:** when the same information is stored multiple times, it creates duplicates and a good way of avoiding this is to split the data into smaller tables.
2. **Improves performance:**You can perform faster query execution on smaller tables that have undergone normalization.
3. **It Minimizes update anomalies:** We can easily update data without affecting other records.
4. **Enhanced data integrity:**  Normalization ensures that data remains consistent and accurate.

**Types of Database Normalization**

**First Normal Form(1NF)**

This normalization level ensures that each column in your data contains only atomic values. Atomic values in this context means that each entry in a column is indivisible.

It is like saying that each cell in a spreadsheet should hold just one piece of information. 1NF ensures atomicity of data, with each column cell containing only a single value and each column having unique names.

### **Second Normal Form (2NF)**

It eliminates partial dependencies by ensuring that non-key attributes depend only on the primary key.

There should be a direct relationship between each column and the primary key column and not between other columns.

**Third Normal Form (3NF)**

It removes transitive dependencies by ensuring that non-key attributes depend only on the primary key. This level of normalization builds on 2NF.

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

This is a strict version of 3NF that addresses additional anomalies. At this normalization level, every determinant is a candidate key.

**Fourth Normal Form (4NF)**

This is a normalization level that builds on BCNF by dealing with multi-valued dependencies.

**Fifth Normal Form (5NF)**

5NF is the highest normalization level that addresses join dependencies. It is used in specific scenarios to further minimize redundancy by breaking a table into smaller tables.

**Database Normalization With Real-World Examples**

**First Normal Form (1NF) Normalization**

1NF ensures that each column cell contains only atomic values. Imagine a library database with a table storing book information (title, author, genre, and borrowed\_by). If the table is not normalized, borrowed\_by could contain a list of borrower names separated by commas. This violates 1NF, as a single cell holds multiple values. The table below is a good representation of a table that violates 1NF, as described earlier.

|  |  |  |  |
| --- | --- | --- | --- |
| **title** | **author** |  | **borrowed\_by** |
| To Kill a Mockingbird | Harper Lee |  | John Doe, Jane Doe, James Brown |
| The Lord of the Rings | J. R. R. Tolkien | Fantasy | Emily Garcia, David Lee |
| Harry Potter and the Sorcerer’s Stone | J.K. Rowling | Fantasy | Michael Chen |

**The solution?**

In 1NF, we create a separate table for borrowers and link them to the book table. These tables can either be linked using the foreign key in the borrower table or a separate linking table. The foreign key in the borrowers table approach involves adding a foreign key column to the borrowers table that references the primary key of the books table. This will enforce a relationship between the tables, ensuring data consistency.

You can find a representation of this below:

**Books table**

|  |  |  |  |
| --- | --- | --- | --- |
| **book\_id (PK)** | **title** | **author** | **genre** |
| 1 | To Kill a Mockingbird | Harper Lee | Fiction |
| 2 | The Lord of the Rings | J. R. R. Tolkien | Fantasy |
| 3 | Harry Potter and the Sorcerer’s Stone | J.K. Rowling | Fantasy |

**Borrowers table**

|  |  |  |
| --- | --- | --- |
| **borrower\_id (PK)** | **name** | **book\_id (FK)** |
| 1 | John Doe | 1 |
| 2 | Jane Doe | 1 |
| 3 | James Brown | 1 |
| 4 | Emily Garcia | 2 |
| 5 | David Lee | 2 |
| 6 | Michael Chen | 3 |

**Second Normal Form (2NF)**

This level of normalization, as already described, builds upon 1NF by ensuring there are no partial dependencies on the primary key. In simpler terms, all non-key attributes must depend on the entire primary key and not just part of it.

From the 1NF that was implemented, we already have two separate tables (you can check the 1NF section).

Now, let’s say we want to link these tables to record borrowings. The initial approach might be to simply add a borrower\_id column to the **books table**, as shown below:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **book\_id (PK)** | **title** | **author** | **genre** | **borrower\_id (FK)** |
| 1 | To Kill a Mockingbird | Harper Lee | Fiction | 1 |
| 2 | The Lord of the Rings | J. R. R. Tolkien | Fantasy | NULL |
| 3 | Harry Potter and the Sorcerer’s Stone | J.K. Rowling | Fantasy | 6 |

This might look like a solution, but it **violates** 2NF simply because the borrower\_id only partially depends on the book\_id. A book can have multiple borrowers, but a single borrower\_id can only be linked to one book in this structure. This creates a partial dependency.

**The solution?**

We need to achieve the many-to-many relationship between books and borrowers to achieve 2NF. This can be done by introducing a separate table:

**Book\_borrowings table**

| **borrowing\_id (PK)** | **book\_id (FK)** | **borrower\_id (FK)** | **borrowed\_date** |
| --- | --- | --- | --- |
| **1** | **1** | **1** | **2024-05-04** |
| **2** | **2** | **4** | **2024-05-04** |
| **3** | **3** | **6** | **2024-05-04** |

This table establishes a clear relationship between books and borrowers. The book\_id and borrower\_id act as foreign keys, referencing the primary keys in their respective tables. This approach ensures that borrower\_id depends on the entire primary key (book\_id) of the books table, complying with 2NF.