SQL

NORMALIZATION

Normalization in SQL

Normalization is the process of organizing data in a database to **reduce redundancy** and **improve data integrity.** The goal is to ensure that data is stored logically, making it easier to maintain, update, and avoid anomalies like **update**, **insert**, and **delete** anomalies.

Normalization involves breaking down a large, complex table into **smaller**, manageable ones and ensuring that relationships between the tables are maintained. This is done through **normal forms.**

Levels of Normalization

There are several levels of normalization, each building upon the previous one.

The most common levels of normalization are :

- **First Normal Form (1NF)**
- **Second Normal Form (2NF)**
- **Third Normal Form (3NF)**
- **Boyce-Codd Normal Form (BCNF)**
- **Fourth Normal Form (4NF)**
- **Fifth Normal Form (5NF).**

First Normal Form (1NF)

A table is in **First Normal Form (1NF)** if:

- All columns contain atomic (indivisible) values.
- Each column contains values of a single type.
- Each column has a unique name.
- The order in which data is stored does not matter.

Example:

Unnormalized Table (Before 1NF):

Student_ID	Name	Courses
1	John Doe	Math, English
2	Jane Smith	History, Math

Normalized Table (1NF):

Student_ID	Name	Course
1	John Doe	Math
1	John Doe	English
2	Jane Smith	History
2	Jane Smith	Math

Explanation:

- The Courses column in the unnormalized table contains multiple values (comma-separated).
- To bring it into **1NF**, we split the data into multiple rows, ensuring that each column contains atomic values.

Second Normal Form (2NF)

A table is in **Second Normal Form (2NF)** if:

- It is in **1NF.**
- There is no partial dependency, meaning non-key attributes are fully dependent on the primary key.

Partial dependency occurs when a non-key column is dependent on only part of the primary key (in the case of a composite primary key).

Example:

Unnormalized Table (Before 2NF):

Student_ID	Course_ID	Instructor	Grade
1	101	Mr. Smith	A
1	102	Mrs. Johnson	B
2	101	Mr. Smith	A

Normalized Table (2NF):

1. **Student_Courses Table (Primary Key: Student_ID, Course_ID):**

Student_ID	Course_ID	Grade
1	101	A
1	102	B
2	101	A

1. **Courses Table (Primary Key: Course_ID):**

Course_ID	Instructor
101	Mr. Smith
102	Mrs. Johnson

Explanation:

- The Instructor is dependent on the **Course_ID**, not the whole composite primary key **(Student_ID, Course_ID).**
- So, we separate the instructor information into a different table, ensuring that the non-key attribute Instructor is fully dependent on the entire primary key **(Course_ID).**

Third Normal Form (3NF)

A table is in **Third Normal Form (3NF)** if:

- It is in **2NF.**
- There is no transitive dependency, meaning non-key attributes are not dependent on other non-key attributes.

Transitive dependency occurs when a non-key attribute depends on another non-key attribute, which in turn depends on the primary key.

Example:

Unnormalized Table (Before 3NF):

Employee_ID	Name	Department	Department_Location
1	John Doe	HR	Building A
2	Jane Smith	IT	Building B

Normalized Table (3NF):

1. **Employees Table (Primary Key: Employee_ID):**

Employee_ID	Name	Department_ID
1	John Doe	1
2	Jane Smith	2

1. **Departments Table (Primary Key: Department_ID):**

Department_ID	Department	Department_Location
1	HR	Building A
2	IT	Building B

Explanation:

- The **Department_Location** depends on the Department, which is a non-key attribute in the **Employees** table.
- To remove this transitive dependency, we create a separate **Departments** table and link it with the Employees table using the **Department_ID**.

Boyce-Codd Normal Form (BCNF)

A table is in **Boyce-Codd Normal Form (BCNF)** if:

- It is in **3NF.**
- For every non-trivial functional dependency, the left-hand side (the determinant) is a **superkey.**

In simpler terms, a table is in **BCNF** if each non-key attribute is dependent on the entire primary key, and there are no exceptions.

Example:

Unnormalized Table (Before BCNF):

Student_ID	Course_ID	Instructor
1	101	Mr. Smith
1	102	Mrs. Johnson
2	101	Mr. Smith

In this example, Instructor is functionally dependent on **Course_ID**, not on the entire composite key (Student_ID, Course_ID).

Normalized Table (BCNF):

1. **Student_Courses Table (Primary Key: Student_ID, Course_ID):**

Student_ID	Course_ID
1	101
1	102
2	101

1. **Courses Table (Primary Key: Course_ID):**

Course_ID	Instructor
101	Mr. Smith
102	Mrs. Johnson

Explanation:

- The original table violated BCNF because Instructor depended only on **Course_ID**, not on the entire primary key.
- We separated the Instructor into a **Courses** table, ensuring that **every non-key** attribute is fully dependent on a **superkey**.

Fourth Normal Form (4NF)

A table is in Fourth Normal Form (4NF) if:

- It is in **BCNF.**
- It has no multi-valued dependencies, meaning no column can contain multiple independent values for the same row.

Example:

Unnormalized Table (Before 4NF):

Student_ID	Course	Hobby
1	Math	Painting
1	English	Reading

Normalized Table (4NF):

1. **Student_Courses Table:**

Student_ID	Course
1	Math
1	English

1. **Student_Hobbies Table:**

Student_ID	Hobby
1	Painting
1	Reading

Explanation:

- In the original table, a student could have multiple hobbies and courses, leading to redundancy.
- We separate hobbies and courses into different tables to avoid multi-valued dependencies.

Fifth Normal Form (5NF)

A table is in Fifth Normal Form (5NF) if:

- It is in **4NF**.
- It does not contain any join dependency, i.e., data can only be reconstructed using joins between smaller tables.

Summary of Normal Forms:

Normal Form	Criteria
1NF	Eliminate repeating groups; ensure atomicity of data in columns.
2NF	Eliminate partial dependencies (i.e., non-key attributes depend on the full primary key).
3NF	Eliminate transitive dependencies (i.e., non-key attributes depend on other non-key attributes).
BCNF	Ensure that every determinant is a candidate key.
4NF	Eliminate multi-valued dependencies.
5NF	Ensure that no join dependency exists.

Benefits of Normalization:

1. **Reduces Data Redundancy:** Prevents the same data from being stored in multiple places.
2. **Improves Data Integrity:** Ensures that data is consistent across the database.
3. **Efficient Updates:** Changes to data are easier since there is no duplication of data.
4. **Faster Queries:** Smaller, more specific tables can speed up query performance.

Drawbacks of Normalization:

- **Complex Queries:** More joins might be required, making queries more complex.
- **Performance Impact:** Excessive normalization might lead to performance degradation in certain cases, especially when dealing with large datasets.

Conclusion:

Normalization helps to structure databases efficiently by minimizing redundancy and dependency, but it must be balanced with performance considerations.