This "**Complete Guide to SQL Constraints**" helps you:

- Keep your data correct and consistent.
- Stop duplicate or missing data.
- Connect tables the right way.
- Save time with default values.
- Make your database faster.

It’s easy to understand and great for beginners or anyone working with databases.

A Complete Guide to SQL Constraints

SQLPost #4 By: JK

A Complete Guide to SQL Constraints

Introduction

SQL constraints are rules or conditions that are applied to table columns to enforce data integrity. These constraints ensure that the data stored in the database adheres to specific standards or requirements. They are essential for maintaining accurate, reliable, and consistent data. SQL constraints are crucial when defining or altering tables to ensure the accuracy and validity of data.

This guide will explain the different types of SQL constraints and provide examples of how they work in real-world database scenarios.

SQL Constraints Overview

SQL constraints can be categorized into the following types:

1. NOT NULL Constraint
2. UNIQUE Constraint
3. PRIMARY KEY Constraint
4. FOREIGN KEY Constraint
5. CHECK Constraint
6. DEFAULT Constraint
7. INDEX Constraint

Each constraint has its own purpose and is used to enforce specific rules for data integrity.

1. NOT NULL Constraint

Description: The NOT NULL constraint ensures that a column cannot have a NULL value. It is used when you want to ensure that a particular column must always have a value (i.e., cannot be left empty).

Use Case:

- Ensuring that required information, such as a customer's name or product ID, is always provided.

Example:

```
CREATE TABLE employees (
   employee_id INT NOT NULL,
   employee_name VARCHAR(100) NOT NULL
);
```

In this example, both employee_id and employee_name cannot be NULL when inserting data.

2. UNIQUE Constraint

Description: The UNIQUE constraint ensures that all values in a column are different from one another. Unlike the primary key, a column with a UNIQUE constraint can accept NULL values (only one NULL is allowed).

Use Case:

- Ensuring that each email address or username in a table is unique.

Example:

```
CREATE TABLE users (
   user_id INT NOT NULL,
   email VARCHAR(100) UNIQUE
);
```

Here, the email column is required to contain unique values, preventing duplicate email addresses.

3. PRIMARY KEY Constraint

Description: The PRIMARY KEY constraint is used to uniquely identify each record in a table. It is a combination of the NOT NULL and UNIQUE constraints, meaning that the primary key column(s) cannot have NULL values, and every record must have a unique value.

Use Case:

- Defining the main identifier of a table, such as an employee ID or order ID.

Example:

```
CREATE TABLE products (
   product_id INT PRIMARY KEY,
   product_name VARCHAR(100)
);
```

In this case, product_id is the primary key and must have unique, non-null values.

4. FOREIGN KEY Constraint

Description: The FOREIGN KEY constraint is used to link one table to another. It establishes a relationship between columns in two different tables. The foreign key in one table points to a primary key in another table, ensuring referential integrity.

Use Case:

- Enforcing the relationship between two tables, such as linking an order to a customer.

Example:

```
CREATE TABLE orders (
   order_id INT PRIMARY KEY,
   customer_id INT,
   FOREIGN KEY (customer_id) REFERENCES customers(customer_id)
);
```

In this case, the customer_id in the orders table must exist in the customers table, ensuring that each order is associated with a valid customer.

5. CHECK Constraint

Description: The CHECK constraint ensures that all values in a column meet a specific condition. It is used to limit the range of values that can be stored in a column.

Use Case:

- Ensuring that a salary column only allows values greater than a specific number or a column only accepts values within a valid range.

Example:

```
CREATE TABLE employees (
   employee_id INT PRIMARY KEY,
   employee_name VARCHAR(100),
   salary DECIMAL(10, 2),
   CHECK (salary > 0)
);
```

In this example, the salary column can only accept positive values, ensuring that no negative salaries are entered.

6. DEFAULT Constraint

Description: The DEFAULT constraint assigns a default value to a column when no value is specified during an insert operation. If no value is provided, the default value is automatically assigned.

Use Case:

- Providing default values for columns like date, status, or country.

Example:

```
CREATE TABLE orders (
   order_id INT PRIMARY KEY,
   order_date DATE DEFAULT CURRENT_DATE
);
```

Here, if no order date is provided during insertion, the current date will be used as the default.

7. INDEX Constraint

Description: An INDEX is not technically a constraint but is used to improve the speed of data retrieval operations on a table. It creates a data structure that allows the database to find rows more quickly based on the indexed column.

Use Case:

- Speeding up query performance, particularly for columns frequently used in WHERE clauses or joins.

Example:

```
CREATE INDEX idx_employee_name ON employees (employee_name);
```

This creates an index on the employee_name column, making queries that filter by employee name more efficient.

Summary:

SQL constraints are like rules that help make sure the information in a database is correct and follows certain patterns. Just like school rules keep things organized,

1. **NOT NULL Constraint**: This rule says that some columns (like names or prices) must always have information in them. No empty spaces (NULL) are allowed.
2. **UNIQUE Constraint**: This rule says that each entry in a column must be different from the others. For example, each student ID should be unique to avoid confusion.
3. **PRIMARY KEY Constraint**: This is a special rule that combines both the "NOT NULL" and "UNIQUE" rules. It is used for things like student IDs, making sure that each student has their own unique ID that can never be empty.
4. **FOREIGN KEY Constraint**: This rule is used to connect two tables, like a student table and a grades table. It makes sure that the grade belongs to a real student by linking the two tables together.
5. **CHECK Constraint**: This rule is like a check on the data to make sure it's reasonable. For example, it makes sure that someone's age isn't negative.
6. **DEFAULT Constraint**: This rule automatically adds a default value when no information is provided. For example, if someone doesn't specify a grade, the system might automatically add a default grade of "A".
7. **INDEX Constraint**: This rule helps the database work faster by making it easier to search for information, like finding all students from a particular city quickly.

In simple terms, SQL constraints are rules that help keep the data in a database neat, correct, and easy to find.