Power Learn Project

SQL Constraints: Primary Key, foreign key and unique key week 7

Database Design

You are live with Gerald Macherechedze

Power Learn Project

SQL Constraints: Primary Key

In SQL, a primary key is a fundamental concept used to define a unique identifier for each record in a database table. It enforces data integrity and ensures that each row in the table has a distinct value in the primary key column. This constraint plays a vital role in organizing and retrieving data efficiently. Below, we'll explore the SQL primary key constraint, how to create a table with a primary key, how to add a primary key using the ALTER TABLE statement, and how to remove a primary key constraint.

SQL Primary Key Constraint

The primary key constraint is a database constraint that ensures the uniqueness and non-nullity of values in a designated column or set of columns. The primary key constraint serves two primary purposes:

Uniqueness: It ensures that each value in the primary key column(s) is unique, meaning no two rows can have the same value in that column.

Non-Null: It enforces that the values in the primary key column(s) cannot be NULL, as a primary key is intended to uniquely identify each record.

Creating a Table with a Primary Key

To create a table with a primary key, you can define the primary key constraint

when you create the table using the CREATE TABLE statement. Here's the syntax:

```
-- Syntax for creating a table with a primary key
CREATE TABLE table_name (
column1 data_type PRIMARY KEY,
column2 data_type,
...
);
```

table_name: Replace this with the name of the table you're creating.

column1: Specify the name of the column you want as the primary key.

data_type: Define the data type of the column.

Example:

Let's create a "Students" table with a primary key on the "StudentID" column:

```
-- Creating a "Students" table with a primary key

CREATE TABLE Students (

StudentID INT PRIMARY KEY,

FirstName VARCHAR(50),

LastName VARCHAR(50)

);
```

In this example, the "StudentID" column is defined as the primary key.

Adding a Primary Key with the ALTER TABLE Statement

You can also add a primary key to an existing table using the ALTER TABLE statement. Here's the syntax:

-- Syntax for adding a primary key with the ALTER TABLE statement

ALTER TABLE table_name

ADD PRIMARY KEY (column1);

table_name: The name of the table you want to modify.

column1: The column to be designated as the primary key.

Example:

Suppose you have an "Employees" table, and you want to add a primary key on the "EmployeeID" column:

-- Adding a primary key to the "Employees" table

ALTER TABLE Employees

ADD PRIMARY KEY (EmployeeID);

Removing a Primary Key Constraint

To remove a primary key constraint from a column, you can use the ALTER TABLE statement as well. Here's the syntax:

-- Syntax for removing a primary key constraint

ALTER TABLE table_name

```
DROP PRIMARY KEY;
```

table_name: The name of the table containing the primary key you want to remove.

Example:

If you decide to remove the primary key constraint from the "EmployeeID" column in the "Employees" table, you can do so with the following SQL statement:

```sql
-- Removing the primary key constraint from the "EmployeeID" column
ALTER TABLE Employees
DROP PRIMARY KEY;
```

The primary key constraint is a critical element in database design, ensuring data uniqueness and accuracy. Whether you're creating a new table or modifying an existing one, understanding how to work with primary keys is essential for maintaining data integrity in your SQL databases.

SQL Foreign Key Constraint with Examples

•A foreign key constraint is primarily used to create relationships between tables, typically to reflect

associations between data entities. Here's how it works with an example:

•Suppose you have two tables: "Orders" and "Customers." In the "Orders" table, you want to associate

each order with a customer by using the "CustomerID" column. The "CustomerID" column in the "Orders" table will reference the "CustomerID" column in the "Customers" table, which is the primary

key of the "Customers" table.

Example:

```
-- Creating a foreign key constraint
ALTER TABLE Orders
ADD CONSTRAINT FK_CustomerID
FOREIGN KEY (CustomerID)
REFERENCES Customers(CustomerID);
```

In this example, the ALTER TABLE statement is used to add a foreign key constraint to the "Orders" table. The constraint is named "FK_CustomerID," and it ensures that the "CustomerID" column in the "Orders" table must have values that exist in the "CustomerID" column of the "Customers" table. This

enforces the relationship between orders and customers.

Adding a Foreign Key Constraint to Existing Tables:

To add a foreign key constraint to an existing table, you can use the ALTER TABLE statement. Here's the syntax:

```
-- Syntax for adding a foreign key constraint with the ALTER TABLE statement
ALTER TABLE table_name
ADD CONSTRAINT constraint_name

FOREIGN KEY (column1)
```

REFERENCES referenced_table(referenced_column);

table_name: The name of the table you want to modify.

constraint_name: Give the foreign key constraint a meaningful name.

column1: The column in the current table that you want to define as the foreign key.

referenced_table: The name of the table you're referencing.

referenced_column: The column in the referenced table, typically the primary key.

Example:

Suppose you have an existing "Orders" table, and you want to add a foreign key constraint to

associate

orders with customers using the "CustomerID" column:

```sql
-- Adding a foreign key constraint to an existing table
ALTER TABLE Orders
ADD CONSTRAINT FK_CustomerID
FOREIGN KEY (CustomerID)
REFERENCES Customers(CustomerID);
```

This SQL statement adds the "FK_CustomerID" foreign key constraint to the "Orders" table, ensuring that values in the "CustomerID" column must match values in the "CustomerID" column of the "Customers" table.

REFERENCES referenced_table(referenced_column);

To remove a foreign key constraint from a table, you can use the ALTER TABLE statement with the DROP CONSTRAINT clause. Here's the syntax:

```
-- Syntax for removing a foreign key constraint
ALTER TABLE table_name
DROP CONSTRAINT constraint_name;
```

table_name: The name of the table containing the foreign key constraint.

constraint_name: The name of the foreign key constraint to be removed.

Example:

If you want to remove the "FK_CustomerID" foreign key constraint from the "Orders" table, you can do so with the following SQL statement:

```
-- Removing a foreign key constraint
ALTER TABLE Orders
DROP CONSTRAINT FK_CustomerID;
```

Removing a foreign key constraint is useful when you no longer require the enforced relationship between tables or when you need to modify the structure of your database.

SQL Unique Constraint

In SQL, a unique constraint is a database feature that ensures that the values in a specific column or set of columns are unique among all the rows in a table.

It guarantees that no duplicate values are allowed in the designated column(s), which is vital for maintaining data integrity.

Here, we'll explore the SQL unique constraint, compare it to the primary key constraint, learn how to create a unique constraint, add it to an existing table, and remove it when necessary.

Unique vs. Primary Key Constraint

While both unique and primary key constraints serve to enforce data integrity and uniqueness, they have distinct characteristics:

Unique Constraint:

Allows one unique value in the column(s).

Allows NULL values, meaning a unique constraint can have multiple NULL values.

Can be applied to multiple columns.

Often used when you want to ensure the uniqueness of data, but not necessarily as an identifier for records.

Primary Key Constraint:

Implies a unique constraint, but with the added condition that the column(s) cannot contain NULL values.

Serves as a unique identifier for each record in a table.

Usually applied to a single column (though composite primary keys, using multiple columns, are also possible).

Creating a Unique Constraint:

To create a unique constraint in SQL, you can define it when creating a table using the

CREATE TABLE

statement. Here's the syntax:

```
-- Syntax for creating a table with a unique constraint
CREATE TABLE table_name (
column1 data_type UNIQUE,
column2 data_type,
...
);
```

table_name: The name of the table you're creating.

column1: Specify the name of the column you want to have a unique constraint.

data_type: Define the data type of the column.

Example:

Let's create a "Students" table with a unique constraint on the "StudentID" column:

```
-- Creating a "Students" table with a unique constraint
CREATE TABLE Students (
StudentID INT UNIQUE,
FirstName VARCHAR(50),
LastName VARCHAR(50)
);
```

In this example, the "StudentID" column is defined as unique.

Adding a Unique Constraint to an Existing Table

To add a unique constraint to an existing table, you can use the ALTER TABLE statement. Here's the

syntax:

```
-- Syntax for adding a unique constraint with the ALTER TABLE statement
);
```

```
ALTER TABLE table_name
ADD CONSTRAINT constraint_name UNIQUE (column1);
```

table_name: The name of the table you want to modify.

constraint_name: Give the unique constraint a meaningful name.

column1: The column in the current table that you want to designate as unique.

Example:

Suppose you have an existing "Emails" table, and you want to ensure that the "EmailAddress" column

contains only unique values:

```
-- Adding a unique constraint to an existing table
ALTER TABLE Emails
ADD CONSTRAINT UC_EmailAddress UNIQUE (EmailAddress);
```

This SQL statement adds a unique constraint named "UC_EmailAddress" to the "Emails" table,

ensuring

that the "EmailAddress" column contains only unique values.

Removing a Unique Constraint

To remove a unique constraint from a column, you can use the ALTER TABLE statement with the DROP CONSTRAINT clause. Here's the syntax:

```
-- Syntax for removing a unique constraint
ALTER TABLE table_name
DROP CONSTRAINT constraint_name;
```

table_name: The name of the table containing the unique constraint.

constraint_name: The name of the unique constraint to be removed.

Example:

If you decide to remove the unique constraint named "UC_EmailAddress" from the "Emails" table, you can do so with the following SQL statement:

Removing a Unique Constraint

To remove a unique constraint from a column, you can use the ALTER TABLE statement with the DROP CONSTRAINT clause. Here's the syntax:

```
-- Syntax for removing a unique constraint
ALTER TABLE table_name
DROP CONSTRAINT constraint_name;
```

table_name: The name of the table containing the unique constraint.

constraint_name: The name of the unique constraint to be removed.

Example:

If you decide to remove the unique constraint named "UC_EmailAddress" from the "Emails" table, you can do so with the following SQL statement:

```
-- Removing a unique constraint

ALTER TABLE Emails
DROP CONSTRAINT UC_EmailAddress;
```

Removing a unique constraint can be useful when your data requirements change or when you need to

adjust your database schema.

Introduction to SQL NOT NULL Constraint:

-The NOT NULL Constraint in SQL

The NOT NULL constraint is a type of integrity constraint that can be applied to a column in a databasetable.

When this constraint is defined on a column, it signifies that every row (record) in that table must contain a non-null value for that particular column. In other words, the column cannot be empty, and each entry must have a meaningful value.

The NOT NULL constraint is used to ensure that critical data is always present in a table, which is essential for maintaining the integrity and reliability of the data stored in the database.

It is commonly employed in scenarios where certain attributes, like email addresses, phone numbers, or primary keys, should always have a value associated with them.

ALTER TABLE with NOT NULL Statement:

To add or remove the NOT NULL constraint from an existing table, you can use the ALTER TABLE statement. This statement allows you to make structural changes to an existing table, including modifying the constraints applied to its columns. Here's how you can use the ALTER TABLE statement to

set a column as NOT NULL or to remove the constraint:

Adding the NOT NULL Constraint:

sql

Copy code

```sql
-- Syntax for adding the NOT NULL constraint
ALTER TABLE table_name
ALTER TABLE table_name
MODIFY column_name data_type NOT NULL;
```

table_name: The name of the table containing the column you want to modify.

column_name: The name of the column to which you want to add the NOT NULL constraint.

data_type: The data type of the column.

Example:

Suppose you have a "Customers" table, and you want to ensure that the "Email" column always contains

a value. You can add the NOT NULL constraint to the "Email" column as follows:

Introduction to SQL NOT NULL Constraint:

-- Removing a unique constraint

In SQL, the NOT NULL constraint is a crucial feature used to ensure data integrity by specifying that a

column must contain a value, and it cannot be left empty or filled with a null value. This constraint is

typically used when you want to enforce the presence of data in a specific column, preventing the

insertion of records with missing or undefined values. Here's an introduction to the SQL NOT NULL

constraint and how it is employed to maintain data consistency in database tables.

sql

Copy code

```sql
-- Adding the NOT NULL constraint to the "Email" column
ALTER TABLE Customers
MODIFY Email VARCHAR(255) NOT NULL;
```

Removing the NOT NULL Constraint:

To remove the NOT NULL constraint from a column, use a similar ALTER TABLE statement:

sql

Copy code

```sql
-- Syntax for removing the NOT NULL constraint
ALTER TABLE table_name
MODIFY column_name data_type;
```

table_name: The name of the table containing the column with the NOT NULL constraint.

column_name: The name of the column from which you want to remove the NOT NULL constraint.

data_type: The data type of the column.

Example:

If you decide to allow the "Email" column in the "Customers" table to have NULL values, you can

remove

the NOT NULL constraint as follows:

sql

Copy code

```
-- Removing the NOT NULL constraint from the "Email" column
ALTER TABLE Customers
MODIFY Email VARCHAR(255);
```

The NOT NULL constraint is a valuable tool in SQL for maintaining data accuracy and preventing the

insertion of incomplete or inconsistent information into your database tables. It helps enforce data

quality standards and enhances the reliability of your database systems.