ORMEDIAN RESEARCH INSTITUTE

TOPIC:

DATABASE

SUBTOPIC: SQL FUNDAMENTALS (PART 6)

By:

ADEKOLA OLUWASEUN O.

DECEMBER 31, 2022

CONTENT COVERED

- *What SQL Constraints Are*
- *NOT NULL Constraint Implementation*
- *UNIQUE Constraint Implementation*
- *Understanding PRIMARY KEY and Its Implementation*
- *Understanding FOREIGN KEY and its implementation.*
- *What SQL CHECK is and HOW to Use it.*

UNDERSTANDING WHAT SQL CONSTRAINTS ARE

The reason behind the use of constraints in database development is to improve the accuracy, consistency and reliability of data to be stored in a database table. It is simply used to limit the datatype that can be fed into a table. Also there is limit to how constraints can be used in the database. Constraints in SQL is applicable to both the Column and Table. With the help of defined constraints, data actions will be aborted if it goes against the rule of constraints.

In SQL there are several constraints used. They are:

DEFAULT - if there is a column where value is not specified, this DEFAULT constraint is used to set default value for such column automatically.

NOT NULL - this constraint is used to ensure that a column does not accept NULL values.

CREATE INDEX -it aids the creation and insertion of data in a faster way.

PRIMARY KEY - This is a constraint that uniquely identifies any row in a table.

CHECK - For ensuring that a specified condition is satisfied.

UNIQUE - this ensures that all values that are entered into a column are entirely different.

FOREIGN KEY - if there is any action that can damage the links between tables, foreign key is use to prevent it.

IMPLEMENTATION OF NOT NULL CONSTRAINT

Ways of Implementing NOT NULL

- *During the process of creating a new table.*
- *When modifying a table later after the table has been created.*

SYNTAX: During Table Creation

Let's say we need to make all columns NOT NULL while creating a table having 3 different fields.

CREATE TABLE Grades (ID INT NOT NULL, SCORE INT(3) NOT NULL, LEVEL INT(3) NOT NULL);

If we try not to insert any value in one of these fields, the field will automatically fill itself with zero value. But this does not mean that such value cannot be updated later.

SYNTAX : During Table Modification

Assuming there was a column without such constraint in one of its fields just like:

CREATE TABLE Grades (ID INT NOT NULL, SCORE INT(3) NOT NULL, LEVEL INT(3));

TABLE MODIFICATION SYNTAX FOR NOT NULL

ALTER TABLE Tablename ALTER COLUMN Column_name datatype NOT NULL;

OR

ALTER TABLE Tablename MODIFY column_name datatype NOT NULL;

CODE EXAMPLE

ALTER TABLE Grades MODIFY LEVEL INT(3) NOT NULL;

IMPLEMENTATION OF UNIQUE CONSTRAINT

UNIQUE constraint can be used on one or more columns within a table to ensure that values in the column(s) are entirely different. For example if I set the ID field of the Grade table to be Unique, it means that I must not repeat the same value for ID when entering records for such field.

SYNTAX FOR MAKING JUST A SINGLE FIELD UNIQUE UPON TABLE CREATION

CREATE TABLE tablename (column_1 datatype NOT NULL UNIQUE, column_2 datatype, column_3 datatype,…….column_n datatype);

SCENARIO

Let's say we need to create a table such that the studentid is set to unique.

CODE EXAMPLE

CREATE TABLE student_info (Studentid int NOT NULL UNIQUE, Level int(3), Score int(3));

There are some cases where there will be need for us to set more than one column to be unique, but this works differently depending on which database we are using at that moment.

UNIQUE CONSTRAINT

Continuation…….

For example, we are required to set the studentid and the score field to be UNIQUE.

We could see that here we now have more than one field to place the UNIQUE constraint on.

CODE EXAMPLE:

CREATE TABLE student_info (Studentid int NOT NULL, Level int(3), Score int(3), CONSTRAINT UC_student_info UNIQUE (Studentid, Score));

NOTE: UC_CONSTRAINT represents UNIQUE CONSTRAINT.

ADDING UNIQUE constraint to columns during table modification

Unique constraint can also be applied to table that has already been created. It means that we can add this unique constraint to column when we want to alter a table and its content.

SYNTAX

ALTER TABLE tablename ADD UNIQUE (column1, column2, column3,…..columnn);

UNIQUE CONSTRAINTS

Continuation……

SCENARIO

Assuming none of the fields of the student_info table had UNIQUE constraint, and we intend to add UNIQUE constraint to the Score column . We can simply modify our table in the format shown below:

CODE EXAMPLE

ALTER TABLE student_info ADD UNIQUE (Score);

DELETING UNIQUE CONSTRAINT FROM THE TABLE

There might be need for us to remove some constraints from the database table when we feel they are not needed again in the table.

SYNTAX

ALTER TABLE Tablename DROP UNIQUE (Column_name1, Column_name2, Column_name3,'…..Column_name_n);

SCENARIO

Let's assume that we need to change remove the UNIQUE constraint place that was placed on the on the studentid, and column fields.

CODE EXAMPLE:

ALTER TABLE student_info DROP UNIQUE(studentid, Score);

WHAT PRIMARY KEYS ARE IN DATABASES

Primary keys are just the unique identifiers of each row in a database table. One or more column can have primary keys.

Primary keys can be applied when creating a table or when altering already created tables.

SYNTAX FOR ADDING PRIMARY KEYS WHEN CREATING A TABLE

Assuming we want to create a table called evaluator which will hold 3 fields: ID, SCORE, AGG. If we intend to make the ID field the primary key, we can implement it in either of the following ways:

CREATE TABLE evaluator (ID int NOT NULL, FirstName Varchar (200), SCORE float, AGG float, PRIMARY KEY (ID));

PRIMARY KEY CONSTRAINT FOR MULTIPLE COLUMN

As we have known that primary keys can be applied to more than one field in a single table.

If our motive is to create a table where we want to specify more than one column as primary keys, let's take the table evaluator table again as an example. Since we have four different fields, and we wish to place primary key on ID and FirstName. The syntax to achieve this is given below:

CREATE TABLE evaluator (ID int NOT NULL, FirstName Varchar (200) NOT NULL, SCORE float, AGG float, CONSTRAINT PK_Evaluator PRIMARY KEY (ID, FirstName));

HOW TO ADD PRIMARY KEY DURING TABLE ALTERATION

There is possibility of adding a primary key to a table even after such table has been created. Assuming the evaluator table created previously had no primary key constraint placed on the ID field, we can simply achieve this through the use of ALTER TABLE statement together with some other functionalities.

Since in this case scenario, we are only interested in adding primary key to the ID field, then the syntax becomes:

ALTER TABLE evaluator ADD PRIMARY KEY (ID);

If our interest is to add primary key to multiple columns during table alteration, then the syntax will become:

ALTER TABLE evaluator ADD CONSTRAINT PK_evaluator PRIMARY KEY (ID, FirstName);

DELETION OF PRIMARY KEY

We can always delete any primary key field permanently if we think such field is not needed again.

To achieve this aim, the DROP statement is used together with the ALTER TABLE statement.

SCENARIO

Let's say we need to drop the primary key constraint initially assigned to the ID field, the syntax to get this done is shown in the CODE EXAMPLE below.

CODE EXAMPLE

```
ALTER TABLE evaluator DROP PRIMARY KEY;
OR
ALTER TABLE evaluator DROP CONSTRAINT PK_evaluator;
```

SQL FOREIGN KEY CONSTRAINT

Linking of two or more tables could be needed at some points in our database if these tables are related. ***FOREIGN KEY*** *and* ***PRIMARY KEY*** *work hand in hand. To prevent actions that could damage the links between related database tables, FOREIGN KEY is employed.*

FOREIGN KEY is used on a column_name of another table when such column_name exists in already created table as a primary key.

PRIMARY KEY is used in the parent table while FOREIGN KEY is used in the child table.

Whenever we want to include foreign key in a table, we have to make reference to the original table where the such column serves as primary key. This is just to show the link between tables.

For example, let us examine two different tables named Customers and Orders respectively.

FOREIGN KEY

Continuation…….

SCENARIO

If the table named "Customers" has the following columns CustomerID, CustomerName, CustomerAddress with customerID being the PRIMARY KEY while the table named Orders has the following fields CustomerID, OrderID, OrderNo with OrderID being the PRIMARY KEY.

We can see that CustomerID is common to both tables named Customers and Orders. Hence, we can say that the CustomerID in the Orders table points to the PRIMARY KEY in the Customers table and hence, we can refer to this field as the FOREIGN KEY.

IMPLEMENTATION OF FOREIGN KEY

--CUSUSTOMERS TABLE

```
CREATE TABLE Customers(
CustomerID int NOT NULL,
CustomerName varchar(100),
CustomerAddress varchar (100),
PRIMARY KEY (CustomerID));
```

OR

```
CREATE TABLE Customers (
CustomerID int NOT NULL PRIMARY KEY,
 CustomerName varchar(100),
CustomerAddress varchar (100)
);
```

--ORDERS TABLE

```
CREATE TABLE Orders (
OrderID int NOT NULL,
 OrderNo int NOT NULL,
CustomerID int,
PRIMARY KEY (OrderID),
CONSTRAINT FK_CustomerOrder
 FOREIGN KEY (CustomerID)
REFERENCES Customers (CustomerID)
);
```

OR

```
CREATE TABLE Orders (
OrderID int NOT NULL PRIMARY KEY ,
 OrderNo int NOT NULL,
CustomerID int FOREIGN KEY
REFERENCES Customers (CustomerID)
);
```

ADDING FOREIGN KEY AFTER TABLE HAS BEEN CREATED

Foreign key constraint can be added to a table during alteration if we forgot to specify foreign key constraint.

Let's assume that we did not specify the foreign key during the table creation. If we intend to include it during the table alteration.

SYNTAX

ALTER TABLE Orders ADD FOREIGN KEY (CustomerID) REFERENCES Customers (CustomerID);

OR

ALTER TABLE Orders ADD CONSTRAINT FK_CustomerOrder FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID);

DROPPING FOREIGN KEY FROM DATABASE TABLE

If in the long run we realize that a given foreign key is no longer needed in a table, we can always find a way to remove it from the table.

For example, if we intend to drop the foreign key (CustomerID) from the Orders Table.

The SYNTAX will be:

ALTER TABLE Orders DROP FOREIGN KEY FK_CustomerOrder;

OR

ALTER TABLE Orders DROP CONSTRAINT FK_CustomerOrder;

SQL CHECK

It is used to give restrictions to the range of value that can be passed into a column. For example, let's consider an recruitment form in which only universities graduates that are 25year and above are eligible to fill the form. This CHECK constraint could be implemented in such database to ensure that the condition is not violated by applicants. If we input value that does not comply with this rule, error message will pop up.

NOTE:

SYNTAX

CREATE TABLE Recruits (ID int NOT NULL, FullName varchar(200), Country varchar(15), Age int, CHECK (Age>=25));

OR

To define CHECK for Multiple columns (say column City and Age)

CREATE TABLE Recruits (ID int NOT NULL, FullName varchar(200), Country varchar(15), Age int, CONSTRAINT CHK_Recruits CHECK (Age>=25 AND Country= "Nigeria"));

SQL CHECK

Continuation…..

We can apply the CHECK constraint on a table that has already been created. This requires us to use the ALTER TABLE statement with the CHECK constraint to achieve that.

This can be performed on a single column or multiple column

SINGLE COLUMN CHECK SYNTAX

ALTER TABLE Recruits ADD CHECK (Age>=25);

MULTIPLE COLUMN CHECK SYNTAX

ALTER TABLE Recruits ADD CONSTRAINT CHK_RecruitAge CHECK (Age>=25 AND Country="Nigeria");

DROPPING SQL CHECK CONSTRAINT FROM A TABLE

Just like every other constraint in SQL, we can delete the CHECK constraint if we feel that it is no longer needed in structuring our table.

If we implement this DROP on the CHECK constraint, then it leave such column neutral, i.e. it doesn't verify whether there is limit to the value or range of value that can be passed into such column.

SCENARIO

Let's remove the check constraint placed on the Recruits Table.

SYNTAX FOR DROPPING THE CONSTRAINTS

```
ALTER TABLE Recruits DROP CHECK CHK_RecruitAge;
```

OR

```
ALTER TABLE Recruits DROP CONSTRAINT CHK_RecruitAge;
```

THANKS FOR VIEWING

NOTE: ***Subsequent topics under SQL will be discussed in Part 7***
Also, all Implementation as far as this presentation is concerned are in line with MySQL Syntax. All have been tested on XAMPP Server.

THE NEXT PART WILL REVOLVE AROUND THE CONTINUATION OF HOW TO MANIPULATE SQL DATABASES.