**SQL Constraints**

SQL Constraints are rules used to limit the type of data that can go into a table, to maintain the accuracy and integrity of the data inside table.

Constraints can be divided into the following two types,

1. **Column level constraints:** Limits only column data.
2. **Table level constraints:** Limits whole table data.

Constraints are used to make sure that the integrity of data is maintained in the database. Following are the most used constraints that can be applied to a table.

* NOT NULL
* UNIQUE
* PRIMARY KEY
* FOREIGN KEY
* CHECK
* DEFAULT

## NOT NULL Constraint

**NOT NULL** constraint restricts a column from having a NULL value. Once **NOT NULL** constraint is applied to a column, you cannot pass a null value to that column. It enforces a column to contain a proper value.

One important point to note about this constraint is that it cannot be defined at table level.

### Example using NOT NULL constraint

CREATE TABLE Student(s\_id int NOT NULL, Name varchar(60), Age int);

The above query will declare that the **s\_id** field of **Student** table will not take NULL value.

## UNIQUE Constraint

**UNIQUE** constraint ensures that a field or column will only have unique values. A **UNIQUE** constraint field will not have duplicate data. This constraint can be applied at column level or table level.

### Using UNIQUE constraint when creating a Table (Table Level)

Here we have a simple CREATE query to create a table, which will have a column **s\_id** with unique values.

CREATE TABLE Student(s\_id int NOT NULL UNIQUE, Name varchar(60), Age int);

The above query will declare that the **s\_id** field of **Student** table will only have unique values and wont take NULL value.

### Using UNIQUE constraint after Table is created (Column Level)

ALTER TABLE Student ADD UNIQUE(s\_id);

The above query specifies that **s\_id** field of **Student** table will only have unique value.

## Primary Key Constraint

Primary key constraint uniquely identifies each record in a database. A Primary Key must contain unique value and it must not contain null value. Usually Primary Key is used to index the data inside the table.

### Using PRIMARY KEY constraint at Table Level

CREATE table Student (s\_id int PRIMARY KEY, Name varchar(60) NOT NULL, Age int);

The above command will creates a PRIMARY KEY on the s\_id.

### Using PRIMARY KEY constraint at Column Level

ALTER table Student ADD PRIMARY KEY (s\_id);

The above command will creates a PRIMARY KEY on the s\_id.

## Foreign Key Constraint

FOREIGN KEY is used to relate two tables. FOREIGN KEY constraint is also used to restrict actions that would destroy links between tables. To understand FOREIGN KEY, let's see its use, with help of the below tables:

**Customer\_Detail** Table

|  |  |  |
| --- | --- | --- |
| **c\_id** | **Customer\_Name** | **address** |
| 101 | Adam | Noida |
| 102 | Alex | Delhi |
| 103 | Stuart | Rohtak |

**Order\_Detail** Table

|  |  |  |
| --- | --- | --- |
| **Order\_id** | **Order\_Name** | **c\_id** |
| 10 | Order1 | 101 |
| 11 | Order2 | 103 |
| 12 | Order3 | 102 |

In **Customer\_Detail** table, **c\_id** is the primary key which is set as foreign key in **Order\_Detail** table. The value that is entered in **c\_id** which is set as foreign key in **Order\_Detail** table must be present in **Customer\_Detail** table where it is set as primary key. This prevents invalid data to be inserted into **c\_id** column of **Order\_Detail** table.

If you try to insert any incorrect data, DBMS will return error and will not allow you to insert the data.

### Using FOREIGN KEY constraint at Table Level

CREATE table Order\_Detail(

order\_id int PRIMARY KEY,

order\_name varchar(60) NOT NULL,

c\_id int, FOREIGN KEY(c\_id)REFERENCES Customer\_Detail(c\_id)

);

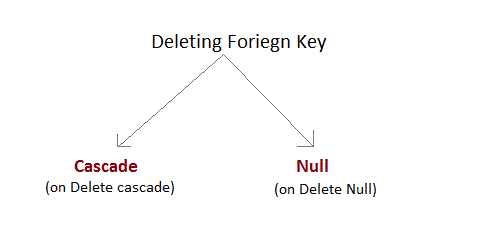
In this query, **c\_id** in table Order\_Detail is made as foriegn key, which is a reference of **c\_id** column in Customer\_Detail table.

### Using FOREIGN KEY constraint at Column Level

ALTER table Order\_Detail ADD FOREIGN KEY (c\_id) REFERENCES Customer\_Detail(c\_id);

### Behaviour of Foriegn Key Column on Delete

There are two ways to maintin the integrity of data in Child table, when a particular record is deleted in the main table. When two tables are connected with Foriegn key, and certain data in the main table is deleted, for which a record exits in the child table, then we must have some mechanism to save the integrity of data in the child table.



1. **On Delete Cascade :** This will remove the record from child table, if that value of foriegn key is deleted from the main table.
2. **On Delete Null :** This will set all the values in that record of child table as NULL, for which the value of foriegn key is deleted from the main table.
3. If we don't use any of the above, then we cannot delete data from the main table for which data in child table exists. We will get an error if we try to do so.

ERROR : Record in child table exist

## CHECK Constraint

**CHECK** constraint is used to restrict the value of a column between a range. It performs check on the values, before storing them into the database. Its like condition checking before saving data into a column.

### Using CHECK constraint at Table Level

CREATE table Student(

s\_id int NOT NULL CHECK(s\_id > 0),

Name varchar(60) NOT NULL,

Age int

);

The above query will restrict the **s\_id** value to be greater than zero.

### Using CHECK constraint at Column Level

ALTER table Student ADD CHECK(s\_id > 0);