
SQL Constraints

What are Constraints?

1. SQL Constraints are used to specify rules for the data in a table.
2. Constraints are used to limit the type of data that can go into a table.
3. This ensures the accuracy and reliability of data in the table.
4. If the data that you are trying to put in the data is not in accordance to the constraint that data is not loaded in the table.
5. Constraints can be given at two occasions in a table.

    1. At the time of creation of table.
    2. After the creation of the table by altering the table.

6. Constraints can be given at two levels.
    1. Column Level: This is when we give constraint at the time of creation of table and we write the constraint type next to the column .
    2. Table Level : This is when we give constraint at the time of creation of table and we write the constraint at the end once we have declared all the columns of the table.

Refer to the next slide to understand a column level and a table level constraint and different type of constraints.

NOT NULL

1. When a NOT NULL constraint is given on any column it means that column will never accept any NULL. Every time you insert the data you need to make sure that for that specific column you have a proper value.

```
CREATE TABLE student
        (rollNo varchar2(4) NOT NULL,
        name varchar2(20) NOT NULL,
        address varchar2(30) ,
        marks number(5,2));
```

In the above example, we are trying to create a STUDENT table. As you can see we have given two columns i.e. rollNO and name a constraint of NOT NULL.

a. This is an example of a column level constraint as we have declared the constraint in the same line where we declared the column and its data type.
b. Secondly, in this table you will not be able to insert any data where either the rollNO or name is NULL.

NOTE: NOT NULL IS ALWAYS A COLUMN LEVEL CONSTRAINT.

2. Now, let's suppose the user or the client asked you to make the column named MARKS as well as a NOT NULL column. It means you need to apply the NOT NULL constraint on the MARKS column.

3. However, the table is already created and it is existing in the database.

4. So, we can now alter the table and add the NOT NULL constraint to MARKS column.

5. We need to ALTER the table and MODIFY the column as NOT NULL is always a COLUMN LEVEL constraint.

```
ALTER TABLE STUDENT MODIFY COLUMN MARKS NUMBER(5,2) NOT NULL;
```

NOTE: WHENEVER YOU APPLY A NOT NULL CONSTRAINT TO A COLUMN WHICH ALREADY HAS DATA IN IT
MAKE SURE THAT THERE IS NO NULL VALUE IN THAT COLUMN PRE-EXISTING. IF IT IS THERE PLEASE
REMOVE OR REPLACE THOSE NULL VALUES FIRST.

AS WHEN A CONSTRAINT IS APPLIED IT IT APPLIED ON THE ENTIRE COLUMN NOT ON ROWS. AS NULL IS ALREADY PRESENT IN THAT COLUMN YOU WILL NOT BE ABLE TO APPLY THE NOT NULL CONSTRAINT.

UNIQUE

1. A Column with a UNIQUE constraint will not accept duplicate values. If you try to insert duplicate values in that column it will throw error.

Unique Key as a column constraint :

```
CREATE TABLE student
        (rollNo varchar2(4) UNIQUE ,
        name varchar2(20) ,address varchar2(30) ,
        marks number(5,2));
```

Unique Key as a table constraint:

```
CREATE TABLE student
        (rollNo varchar2(4) ,name varchar2(20) ,
        address varchar2(30) ,marks number(5,2)
        CONSTRAINT roll_key UNIQUE(rollNo));
```

2. In the above two screenshots you can see UNIQUE constraint given at column level in first screenshot and UNIQUE constraint given at table level in the second screenshot.

As you saw in the previous slide how to give a column and table level unique constraint.

Whenever, you give a UNIQUE constraint system saves that constraint with a default name in the memory. Default naming convention of a UNIQUE constraint is it keeps the same name as the column name.

However, if you would like to give a name of your choice you can give a table level UNIQUE constraint. Refer to the second screenshot in the previous slide.

You can use the below syntax to the see the constraints on a column and the name in the system as if in case you would like to remove the constraint at any point of time how would you remove the same without knowing the name by which system has saved it.

```
SELECT COLUMN_NAME, CONSTRAINT_NAME
FROM INFORMATION_SCHEMA. KEY_COLUMN_USAGE
WHERE TABLE_NAME = 'EMPLOYEES';
```

If in case you want to add a UNIQUE constraint on a column after the table is created you can ALTER the table and add it as mentioned below however, just make sure that the column on which you are adding the constraint if there is data there should be no duplicates.

```
ALTER TABLE TABLE_NAME ADD UNIQUE (COLUMN_NAME);
```

If you would like to give the constraint a name of your choice then you can use the below query.

```
ALTER TABLE TABLE_NAME ADD CONSTRAINT CONSTRAINT_NAME UNIQUE (COLUMN_NAME);
```

DEFAULT

1. If you have given a DEFAULT constraint to any column in a table it means that while loading the data or while inserting the data in the table at any point of time you do not give the value for that specific column it will automatically fill in the default value that you have provided.

```
CREATE TABLE student
        (rollNo varchar2(4) ,name varchar2(20) ,
        address varchar2(30) ,
        marks number(5,2) DEFAULT 0);
```

As you can see in the above screenshot, at any point of time while loading the data in the data in student table if no value if provided for MARKS it will put a zero as that is the default value given for that column.

Let's suppose after the table creation later you decided to give a default value for name column as well. So, you can do that by ALTERING the table in the below manner.

ALTER TABLE STUDENT ALTER NAME SET DEFAULT 'UNKNOWN';

CHECK

1. It is used when we need to enforce some rules on a particular column. In other words, we need to make sure that the value entered in that column should satisfy the condition that we have provided in CHECK.

```
CREATE TABLE student
        (rollNo varchar2(4)  CHECK(rollNo like 'C%'),
        name varchar2(20)  CONSTRAINT chk_nm
        CHECK(name = upper(name)),
        address varchar2(30) ,
        marks number(5,2) CHECK(marks > 40));
```

2. Give an attempt and make your trainer understand what kind of integrity checks will happen on the columns as per the abve screenshot.
3. If in case you want to add a check constraint to a pre-existing column in a table you can use the below query.

ALTER TABLE TABLE_NAME ADD CHECK (COLUMN NAME AND CONDITION)

4. If in case you want to add a check constraint to a pre-existing column in a table with a name of your choice you can use the below query.

ALTER TABLE TABLE_NAME ADD CONSTRAINT CONSTRAINT_NAME CHECK (COLNAME&COND)

PRIMARY KEY

1. PRIMARY KEY constraint uniquely identifies each record in a table.
2. Any column which is a PRIMARY KEY cannot hold duplicate values and cannot have NULLs.
3. You can only have ONE PRIMARY KEY in a table in any scenario.
4. That one PRIMARY KEY can be of one column or multiple columns.

Primary Key as a column constraint –

```
CREATE TABLE student
        (rollNo varchar2(4)  PRIMARY KEY,
        name varchar2(20) , address varchar2(30) ,
        marks number(5,2));
```

Primary Key as a table constraint –

```
CREATE TABLE student
        (rollNo varchar2(4)  ,name varchar2(20) ,
        address varchar2(30) ,marks number(5,2)
        PRIMARY KEY(rollNo));
```

If in case you would like to add a PRIMARY KEY constraint to a pre-existing column in a table first you need to make sure of two things that the column that you are trying to create as a PRIMARY KEY that column should not have duplicate values and there should be no NULLs. If both the condition are satisfied you can make use of the below query.

ALTER TABLE TABLE_NAME ADD PRIMARY KEY (COLUMN_NAME);

NOTE: YOU CANNOT GIVE A NAME OF YOUR CHOICE TO A PRIMARY KEY IN MYSQL AS THE DEFAULT NAME THAT THE SYSTEM WILL GIVE TO THE PRIMARY KEY IS PRIMARY.

So, even if you write the below query it will act same like the above one. System will ignore the name given by you.

ALTER TABLE TABLE_NAME ADD CONSTRAINT CONSTRAINT_NAME PRIMARY KEY (COLUMN NAME)

FOREIGN KEY

1. A foreign key is a referential constraint.
2. A foreign key helps to establish relationship between two tables. (you need to have at least one common column between two tables).
3. A foreign key is a field (or collection of fields) in one table that refers to the PRIMARY KEY in another table.
4. The foreign key is called the CHILD table and the table with the PRIMARY KEY is called as the referenced or PARENT table.

Persons Table

PersonID	LastName	FirstName	Age
1	Hansen	Ola	30
2	Svendson	Tove	23
3	Pettersen	Kari	20

Orders Table

OrderID	OrderNumber	PersonID
1	77895	3
2	44678	3
3	22456	2
4	24562	1

This is how the relationship was established between Orders and Person Table.
Persons (Parent Table)
Orders (Child Table)

```
CREATE TABLE Orders (
   OrderID int NOT NULL,
   OrderNumber int NOT NULL,
   PersonID int,
   PRIMARY KEY (OrderID),
   FOREIGN KEY (PersonID) REFERENCES
Persons(PersonID)
);
```

Let's suppose Orders and Persons Table already existed in the database and you would like to add a FOREIGN KEY constraint later so you can use the ALTER statement.

ALTER Table Orders add foreign key (personid) references Persons (PersonId);

This will create a foreign key constraint with a default name in the system. If you want to give a name of your choice to the constraint you can use the below query

ALTER Table Orders add constraint constraint_name foreign key (personid) references Persons (PersonId);

Now, once you have established the relationship you need to keep in mind the below things.

1. You will not be able to insert a value directly in the child table as the tables are now connected so the value has to go to the parent table first and then the child table.

1. You will not be able to update or delete the information from the child table directly.

Note: If you want to do anything in the child table directly, you need to first switch off the foreign key concept between the tables by writing the below query.

```
Set foreign_key_checks = 0;
```

Whenever, you create a foreign key relationship there are three more important keywords that comes into existence.

Without mentioning these keyword system will not allow you to update or delete anything in the parent table as system does not know what to do with such rows in the child table.

1. ON UPDATE CASCADE : It means if the value is updated for the column (based on which tables are connected) in the Parent Table system will update the values in the Child Table as well for that column.

2. ON DELETE CASCADE: It means if the value is deleted for the column (based on which tables are connected) in the Parent Table system will delete the row in the Child Table as well.

3. ON DELETE SET NULL : It means if the value is deleted for the column (based on which tables are connected) in the Parent Table system will delete the value for that column in the CHILD table and will putNULL.

See the example on the next slide.

Persons Table

PersonID	LastName	FirstName	Age
1	Hansen	Ola	30
2	Svendson	Tove	23
3	Pettersen	Kari	20

Orders Table

OrderID	OrderNumber	PersonID
1	77895	3
2	44678	3
3	22456	2
4	24562	1

```
CREATE TABLE Orders (
   OrderID int NOT NULL,
   OrderNumber int NOT NULL,
   PersonID int,
   PRIMARY KEY (OrderID),
   FOREIGN KEY (PersonID) REFERENCES
Persons(PersonID)
) on update cascade;
```

Now, as Orders and Persons are connected and as you can see in the query Persons is the Parent Table and Orders is Child as Orders is referencing to Parents. I have used the keyword as on update cascade while establishing the relationship.

If I go to Persons table and update the PersonID 1 to 11, the same change will reflect in the Orders Table it means in the order table wherever the personid is 1 It will also change to 11.

Note: If I do not write on update cascade and just try to update in persons table system will throw an error as system will not have any cluse as to what to do with person id 1 in the child table.

Persons Table

PersonID	LastName	FirstName	Age
1	Hansen	Ola	30
2	Svendson	Tove	23
3	Pettersen	Kari	20

Orders Table

OrderID	OrderNumber	PersonID
1	77895	3
2	44678	3
3	22456	2
4	24562	1

```
CREATE TABLE Orders (
   OrderID int NOT NULL,
   OrderNumber int NOT NULL,
   PersonID int,
   PRIMARY KEY (OrderID),
   FOREIGN KEY (PersonID) REFERENCES
Persons(PersonID)
) on delete cascade;
```

Now, as Orders and Persons are connected and as you can see in the query Persons is the Parent Table and Orders is Child as Orders is referencing to Parents. I have used the keyword as on update cascade while establishing the relationship.

If I go to Persons table and delete the PersonID 1. the same change will reflect in the Orders Table it means all the rows where the personid is 1 those complete row will be deleted.

Note: If I do not write on delete cascade and system will not allow to delete the data from the parent table.

Persons Table

PersonID	LastName	FirstName	Age
1	Hansen	Ola	30
2	Svendson	Tove	23
3	Pettersen	Kari	20

Orders Table

OrderID	OrderNumber	PersonID
1	77895	3
2	44678	3
3	22456	2
4	24562	1

```
CREATE TABLE Orders (
   OrderID int NOT NULL,
   OrderNumber int NOT NULL,
   PersonID int,
   PRIMARY KEY (OrderID),
   FOREIGN KEY (PersonID) REFERENCES
Persons(PersonID)
) on delete set null;
```

Now, as Orders and Persons are connected and as you can see in the query Persons is the Parent Table and Orders is Child as Orders is referencing to Parents. I have used the keyword as on update cascade while establishing the relationship.

If I go to Persons table and delete the PersonID 1. the same change will reflect in the Orders Table it means all the rows where the personid is 1 in the person id column for that row it will be NULL.

Note: If I do not write on delete set null then system will not allow to delete the data from the parent table.