# SQL Constraints

SQL constraints are used to specify rules for the data in a table.

Constraints are used to limit the type of data that can go into a table. This ensures the accuracy and reliability of the data in the table. If there is any violation between the constraint and the data action, the action is aborted.

Constraints can be column level or table level. Column level constraints apply to a column, and table level constraints apply to the whole table.

The following constraints are commonly used in SQL:

| Constraints | Description |
|---|---|
| NOT NULL | Ensures that a column can't have a NULL value. |
| UNIQUE | Ensures that values in all column are different. |
| PRIMARY KEY | A combination of NOT NULL and Unique. It is used to identify a tuple. |
| FOREIGN KEY | Uniquely identifies a row/record in another table |
| CHECK | Ensures that all values in a column satisfies a specific condition |
| DEFAULT | Sets a default value for a column when no value is specified |
| INDEX | Used to create and retrieve data from the database very quickly |

## SQL Create Constraints

Constraints can be specified when the table is created with the CREATE TABLE statement, or after the table is created with the ALTER TABLE statement.

### Syntax

```
CREATE TABLE table_name (
    column1 datatype constraint,
    column2 datatype constraint,
    column3 datatype constraint,
    ....
);
```

## SQL NOT NULL Constraint

By default, a column can hold NULL values.

The NOT NULL constraint enforces a column to NOT accept NULL values.

This enforces a field to always contain a value, which means that you cannot insert a new record, or update a record without adding a value to this field.

### SQL NOT NULL on CREATE TABLE

The following SQL ensures that the "Reg_No", "Name","Branch"  and "Email" columns will NOT accept NULL values when the "Students" table is created:

```
CREATE TABLE STUDENTS(
Reg_No int NOT NULL,
Name varchar(255) NOT NULL,
Branch varchar(255) NOT NULL,
Email varchar(255)
);
```

### SQL NOT NULL on ALTER TABLE

To create a NOT NULL constraint on the "Email" column when the "Students" table is already created, use the following SQL:

```
ALTER TABLE Students
MODIFY Email int NOT NULL;
```

## SQL UNIQUE Constraint

A UNIQUE constraint ensures that all the value in that column are different.

The UNIQUE & PRIMARY KEY are similar but only there is one PRIMARY KEY constraint per table & UNIQUE can be as much as required.

You can say that it is little like primary key but it can accept only one null value .

The unique key and primary key both provide a guarantee for uniqueness for a column or a set of columns.

### SQL UNIQUE Constraint on CREATE TABLE

The following SQL creates a UNIQUE constraint on the "Reg_No" column when the "Students" table is created:

**MySQL**

```
CREATE TABLE STUDENTS(
Reg_No int NOT NULL,
Name varchar(255) NOT NULL,
Branch varchar(255) NOT NULL,
Email varchar(255),
UNIQUE (Reg_No)
);
```

**Oracle**

```
CREATE TABLE STUDENTS(
Reg_No int NOT NULL UNIQUE,
Name varchar(255) NOT NULL,
Branch varchar(255) NOT NULL,
Email varchar(255)
);
```

To name a UNIQUE constraint, and to define a UNIQUE constraint on multiple columns, use the following SQL syntax:

**MySQL / Oracle :**

```
CREATE TABLE STUDENTS(
Reg_No int NOT NULL,
Name varchar(255) NOT NULL,
Branch varchar(255) NOT NULL,
Email varchar(255),
CONSTRAINT UC_STUDENTS UNIQUE (Reg_No,Email)
);
```

## SQL UNIQUE Constraint on ALTER TABLE

To create a UNIQUE constraint on the "Reg_No" column when the table is already created, use the following SQL:

```
ALTER TABLE STUDENTS
ADD UNIQUE (Reg_No);
```

To name a UNIQUE constraint, and to define a UNIQUE constraint on multiple columns, use the following SQL syntax:

```
ALTER TABLE STUDENTS
ADD CONSTRAINT UC_STUDENTS UNIQUE (Reg_No,Email);
```

## DROP a UNIQUE Constraint

To drop a UNIQUE constraint, use the following SQL:
**MySQL:**

```
ALTER TABLE Students
DROP INDEX UC_Students;
```

> **NOTE :** SQL isn't Case sensitive language.

**Oracle:**

```
ALTER TABLE Students
DROP CONSTRAINT UC_Students;
```

# SQL PRIMARY KEY Constraint

A column is called PRIMARY KEY (PK) that uniquely identifies each row in the table.
Primary keys must contain UNIQUE values, and cannot contain NULL values.

A table can have only ONE primary key; and in the table, this primary key can consist of single or multiple columns (fields).

### Points to remember for primary key:

- Primary key enforces the entity integrity of the table.
- Primary key always has unique data.
- A primary key length cannot be exceeded than 900 bytes.
- A primary key cannot have null value.
- There can be no duplicate value for a primary key.
- A table can contain only one primary key constraint.

## SQL PRIMARY KEY on CREATE TABLE

The following SQL creates a PRIMARY KEY on the "Reg_No" column when the "Students" table is created:

**MySQL:**

```
CREATE TABLE Students (
    Reg_No int NOT NULL,
    Name varchar(255) NOT NULL,
    Branch varchar(255),
    Email varchar(255),
    PRIMARY KEY (Reg_No)
);
```

**Oracle:**

```
CREATE TABLE Students (
    Reg_No int NOT NULL PRIMARY KEY,
    Name varchar(255) NOT NULL,
    Branch varchar(255),
    Email varchar(255)
);
```

To allow naming of a PRIMARY KEY constraint, and for defining a PRIMARY KEY constraint on multiple columns, use the following SQL syntax:

**MySQL / Oracle :**

```
CREATE TABLE Students (
    Reg_No int NOT NULL,
    Name varchar(255) NOT NULL,
    Branch varchar(255),
    Email varchar(255),
    CONSTRAINT PK_Students PRIMARY KEY(Reg_No,Name)
);
```

> **NOTE:** In the example above there is only ONE PRIMARY KEY (PK_Students).
>
> However, the VALUE of the primary key is made up of TWO COLUMNS (Reg_No + Name).

In designing the composite primary key, you should use as few columns as possible. It is good for storage and performance both, the more columns you use for primary key the more storage space you require.

In terms of performance, less data means the database can process faster.

## SQL PRIMARY KEY on ALTER TABLE

To create a PRIMARY KEY constraint on the "Reg_No" column when the table is already created, use the following SQL:

```
ALTER TABLE Students
ADD PRIMARY KEY(Reg_No);
```

To allow naming of a PRIMARY KEY constraint, and for defining a PRIMARY KEY constraint on multiple columns, use the following SQL syntax

```
ALTER TABLE Students
ADD CONSTRAINT PK_Students PRIMARY KEY (Reg_No,Name);
```

**Note:** If you use the ALTER TABLE statement to add a primary key, the primary key column(s) must already have been declared to not contain NULL values (when the table was first created).

## DROP a PRIMARY KEY Constraint

To drop a PRIMARY KEY constraint, use the following SQL:

**MySQL:**

```
ALTER TABLE Students
DROP PRIMARY KEY;
```

**Oracle:**

```
ALTER TABLE Students
DROP CONSTRAINT PK_Students;
```

# SQL FOREIGN KEY Constraint

A FOREIGN KEY is a field or a column that is used to establish a link between two tables.

A FOREIGN KEY is a field (or collection of fields) in one table that refers to the PRIMARY KEY in another table.

The FOREIGN KEY constraint also prevents invalid data from being inserted into the foreign key column, because it has to be one of the values contained in the table it points to.

## SQL FOREIGN KEY on CREATE TABLE

The following SQL creates a FOREIGN KEY on the "Reg_No" column when the "Marks" table is created:

**MySQL:**

```
CREATE TABLE Marks(
    Reg_No int NOT NULL,
    Subj_Code varChar(255),
    Marks int NOT NULL,
    PRIMARY KEY(Subj_Code),
    FOREIGN KEY(Reg_No) REFERENCES Students(Reg_No)
);
```

**Oracle:**

```
CREATE TABLE Marks(
    Marks int NOT NULL PRIMARY KEY,
    Subj_Code varChar(255),
    Reg_No int FOREIGN KEY REFERENCES Students(Reg_No)
);
```

## SQL FOREIGN KEY on ALTER TABLE

To create a FOREIGN KEY constraint on the "Reg_No" column when the "Marks" table is already created, use the following SQL:

```
ALTER TABLE Marks
ADD FOREIGN KEY (Reg_No) REFERENCES Students(Reg_No);
```

## DROP a FOREIGN KEY Constraint

To drop a FOREIGN KEY constraint, use the following SQL:

**MySQL:**

```
ALTER TABLE Marks
DROP FOREIGN KEY FK_StuMark
```

### Difference between primary key and foreign key in SQL:

These are some important difference between primary key and foreign key in SQL-

Primary key cannot be null on the other hand foreign key can be null.

Primary key is always unique while foreign key can be duplicated.

Primary key uniquely identify a record in a table while foreign key is a field in a table that is primary key in another table.

There is only one primary key in the table on the other hand we can have more than one foreign key in the table.

By default primary key adds a clustered index on the other hand foreign key does not automatically create an index, clustered or non-clustered. You must manually create an index for foreign key.

## SQL CHECK Constraint

The CHECK constraint is used to limit the value range that can be placed in a column.

If you define a CHECK constraint on a single column it allows only certain values for this column.

If you define a CHECK constraint on a table it can limit the values in certain columns based on values in other columns in the row.

## SQL CHECK on CREATE TABLE

The following SQL creates a CHECK constraint on the "Age" column when the "Persons" table is created. The CHECK constraint ensures that the age of a person must be 18, or older:

**MySQL:**

```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    CHECK (Age>=18)
);
```

**Oracle:**

```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int CHECK (Age>=18)
);
```

To allow naming of a CHECK constraint, and for defining a CHECK constraint on multiple columns, use the following SQL syntax:

```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    City varchar(255),
    CONSTRAINT CHK_Person CHECK (Age>=18 AND City='Allahabad')
);
```

## SQL CHECK on ALTER TABLE

To create a CHECK constraint on the "Age" column when the table is already created, use the following SQL:

**MySQL / SQL Server / Oracle / MS Access:**

```
ALTER TABLE Persons
ADD CHECK (Age>=18);
```

To allow naming of a CHECK constraint, and for defining a CHECK constraint on multiple columns, use the following SQL syntax:

**MySQL / SQL Server / Oracle / MS Access:**

```
ALTER TABLE Persons
ADD CONSTRAINT CHK_PersonAge CHECK (Age>=18 AND City='Allahabad');
```

### DROP a CHECK Constraint

To drop a CHECK constraint, use the following SQL:

**SQL Server / Oracle / MS Access:**

```
ALTER TABLE Persons
DROP CONSTRAINT CHK_PersonAge;
```

**MySQL:**

```
ALTER TABLE Persons
DROP CHECK CHK_PersonAge;
```

## SQL DEFAULT Constraint

The DEFAULT constraint is used to provide a default value for a column.

The default value will be added to all new records IF no other value is specified.

### SQL DEFAULT on CREATE TABLE

The following SQL sets a DEFAULT value for the "City" column when the "Persons" table is created:

```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    City varchar(255) DEFAULT 'PrayagRaj'
);
```

### SQL DEFAULT on ALTER TABLE

To create a DEFAULT constraint on the "City" column when the table is already created, use the following SQL:

**MySQL:**

```
ALTER TABLE Persons
ALTER City SET DEFAULT 'Allahabad';
```

**Oracle:**

```
ALTER TABLE Persons
MODIFY City DEFAULT 'Allahabad';
```

### DROP a DEFAULT Constraint

To drop a DEFAULT constraint, use the following SQL:

**MySQL:**

```
ALTER TABLE Persons
ALTER City DROP DEFAULT;
```

**Oracle:**

```
ALTER TABLE Persons
ALTER COLUMN City DROP DEFAULT;
```

## SQL CREATE INDEX Statement

The CREATE INDEX statement is used to create indexes in tables.

Indexes are used to retrieve data from the database more quickly than otherwise. The users cannot see the indexes, they are just used to speed up searches/queries.

> **Note:** Updating a table with indexes takes more time than updating a table without (because the indexes also need an update). So, only create indexes on columns that will be frequently searched against.

## CREATE INDEX Example

The SQL statement below creates an index named "idx_lastname" on the "LastName" column in the "Persons" table:

```
CREATE INDEX idx_lastname
ON Persons (LastName);
```

If you want to create an index on a combination of columns, you can list the column names within the parentheses, separated by commas:

```
CREATE INDEX idx_pname
ON Persons (LastName, FirstName);
```

## DROP INDEX Statement

The DROP INDEX statement is used to delete an index in a table.

```
DROP INDEX index_name;
```

### MYSQL

```
ALTER TABLE table_name
DROP INDEX index_name;
```