

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.

Example

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

SQL UNIQUE Constraint

The **UNIQUE** constraint ensures that all values in a column are different.

Both the **UNIQUE** and **PRIMARY KEY** constraints provide a guarantee for uniqueness for a column or set of columns.

A **PRIMARY KEY** constraint automatically has a **UNIQUE** constraint.

However, you can have many **UNIQUE** constraints per table, but only one **PRIMARY KEY** constraint per table.

MySQL:

```
CREATE TABLE Persons (  
    ID int NOT NULL,  
    LastName varchar(255) NOT NULL,  
    FirstName varchar(255),  
    Age int,  
    UNIQUE (ID)  
);
```

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 column it will allow 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)  
);
```

SQL DEFAULT Constraint

The **DEFAULT** constraint is used to set 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:

My SQL / SQL Server / Oracle / MS Access:

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

L PRIMARY KEY Constraint

The **PRIMARY KEY** constraint uniquely identifies each record in a 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).

SQL PRIMARY KEY on CREATE TABLE

The following SQL creates a **PRIMARY KEY** on the "ID" column when the "Persons" table is created:

MySQL:

```
CREATE TABLE Persons (  
  ID int NOT NULL,  
  LastName varchar(255) NOT NULL,  
  FirstName varchar(255),  
  Age int,  
  PRIMARY KEY (ID)  
);
```

QL FOREIGN KEY Constraint

The **FOREIGN KEY** constraint is used to prevent actions that would destroy links between tables.

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

The table with the foreign key is called the child table, and the table with the primary key is called the referenced or parent table.

Persons Table

PersonID	LastName
1	Hansen
2	Svendson
3	Pettersen

Orders Table

OrderID	OrderNumber	PersonID
1	77895	3

2	44678	3
3	22456	2
4	24562	1

Notice that the "PersonID" column in the "Orders" table points to the "PersonID" column in the "Persons" table.

The "PersonID" column in the "Persons" table is the **PRIMARY KEY** in the "Persons" table.

The "PersonID" column in the "Orders" table is a **FOREIGN KEY** in the "Orders" table.

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

The SQL CREATE TABLE Statement

The **CREATE TABLE** statement is used to create a new table in a database.

Syntax

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

SQL ALTER TABLE Statement

The **ALTER TABLE** statement is used to add, delete, or modify columns in an existing table.

The **ALTER TABLE** statement is also used to add and drop various constraints on an existing table.

ALTER TABLE - ADD Column

To add a column in a table, use the following syntax:

```
ALTER TABLE table_name  
ADD column_name datatype;
```

Example

```
ALTER TABLE Customers  
ADD Email varchar(255);
```

The SQL DELETE Statement



The **DELETE** statement is used to delete existing records in a table.

DELETE Syntax

```
DELETE FROM table_name WHERE condition;
```

Example

```
DELETE FROM Customers WHERE CustomerName='Alfreds Futterkiste';
```

SQLQuery1.sql - I...hammad Hasan (70)) *  

```
PRIMARY KEY (ID)
);
--Altering Table
ALTER TABLE Persons
ADD Email varchar(255);

select * from Persons;

INSERT INTO Persons (ID, LastName, FirstName, Age, Email)
VALUES ('001', 'Hasan', 'Muhammad', '28', 'mhasan@test.com');


INSERT INTO Persons (ID, LastName, FirstName, Age, Email)
VALUES ('002', 'Ali', 'Muhammad', '25', 'ali@test.com');



INSERT INTO Persons (ID, LastName, FirstName, Age, Email)
VALUES ('003', 'Sajid', 'Muhammad', '22', 'sajid@test.com');

select * from Persons;

--Altering Table
DELETE FROM Persons WHERE ID='003';

select * from Persons;
```

100 % 

 Results  Messages

	ID	LastName	FirstName	Age	Email
1	1	Hasan	Muhammad	28	mhasan@test.com
2	2	Ali	Muhammad	25	ali@test.com
3	3	Sajid	Muhammad	22	sajid@test.com