**TASK 5 SQL 22-03-23**

**SQL DATA TYPES**

Data types are used to represent the nature of the data that can be stored in the database table. For example, in a particular column of a table, if we want to store a string type of data then we will have to declare a string data type of this column.

Data types mainly classified into three categories for every database.

* String Data types
* Numeric Data types
* Date and time Data types

**SQL SERVER DATA TYPES**

**STRING DATA TYPE**

|  |  |
| --- | --- |
| **char(n)** | It is a fixed width character string data type. Its size can be up to 8000 characters. |
| **varchar(n)** | It is a variable width character string data type. Its size can be up to 8000 characters. |
| **varchar(max)** | It is a variable width character string data types. Its size can be up to 1,073,741,824 characters. |
| **text** | It is a variable width character string data type. Its size can be up to 2GB of text data. |
| **nchar** | It is a fixed width Unicode string data type. Its size can be up to 4000 characters. |
| **nvarchar** | It is a variable width Unicode string data type. Its size can be up to 4000 characters. |
| **ntext** | It is a variable width Unicode string data type. Its size can be up to 2GB of text data. |
| **binary(n)** | It is a fixed width Binary string data type. Its size can be up to 8000 bytes. |
| **varbinary** | It is a variable width Binary string data type. Its size can be up to 8000 bytes. |
| **image** | It is also a variable width Binary string data type. Its size can be up to 2GB. |

**NUMERIC DATA TYPES**

|  |  |
| --- | --- |
| **bit** | It is an integer that can be 0, 1 or null. |
| **tinyint** | It allows whole numbers from 0 to 255. |
| **Smallint** | It allows whole numbers between -32,768 and 32,767. |
| **Int** | It allows whole numbers between -2,147,483,648 and 2,147,483,647. |
| **bigint** | It allows whole numbers between -9,223,372,036,854,775,808 and 9,223,372,036,854,775,807. |
| **float(n)** | It is used to specify floating precision number data from -1.79E+308 to 1.79E+308. The n parameter indicates whether the field should hold the 4 or 8 bytes. Default value of n is 53. |
| **real** | It is a floating precision number data from -3.40E+38 to 3.40E+38. |
| **money** | It is used to specify monetary data from -922,337,233,685,477.5808 to 922,337,203,685,477.5807. |

**DATE AND TIME DATA TYPE**

|  |  |
| --- | --- |
| **datetime** | It is used to specify date and time combination. It supports range from January 1, 1753, to December 31, 9999 with an accuracy of 3.33 milliseconds. |
| **datetime2** | It is used to specify date and time combination. It supports range from January 1, 0001 to December 31, 9999 with an accuracy of 100 nanoseconds |
| **date** | It is used to store date only. It supports range from January 1, 0001 to December 31, 9999 |
| **time** | It stores time only to an accuracy of 100 nanoseconds |
| **timestamp** | It stores a unique number when a new row gets created or modified. The time stamp value is based upon an internal clock and does not correspond to real time. Each table may contain only one-time stamp variable. |

**SQL SERVER OTHER DATA TYPES**

|  |  |
| --- | --- |
| **Sql\_variant** | It is used for various data types except for text, timestamp, and ntext. It stores up to 8000 bytes of data. |
| **XML** | It stores XML formatted data. Maximum 2GB. |
| **cursor** | It stores a reference to a cursor used for database operations. |
| **table** | It stores result set for later processing. |
| **uniqueidentifier** | It stores GUID (Globally unique identifier). |

**SQL CONSTRAINTs**

Constraints are the rules enforced on the data columns of a table. These 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 database.

Constraints could be either on a column level or a table level. The column level constraints are applied only to one column, whereas the table level constraints are applied to the whole table.

**Syntax**

CREATE TABLE *table\_name*(  
*column1 datatype* *constraint*,  
*column2 datatype* *constraint*,  
*column3 datatype* *constraint*,  
    ....  
);

Following are some of the most commonly used constraints available in SQL.

* **NOT NULL** - Ensures that a column cannot have a NULL value
* **UNIQUE** - Ensures that all values in a column are different
* **PRIMARY KEY** - A combination of a NOT NULL and UNIQUE. Uniquely identifies each row in a table
* **FOREIGN KEY** - Prevents actions that would destroy links between tables
* **CHECK** - Ensures that the values in a column satisfies a specific condition
* **DEFAULT** - Sets a default value for a column if no value is specified
* **CREATE INDEX** - Used to create and retrieve data from the database very quickly

## **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 "ID", "LastName", and "FirstName" columns will NOT accept NULL values when the "Persons" table is created:

### Example

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

**SQL NOT NULL on ALTER TABLE**

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

**SQL Server / MS Access:**

ALTER TABLE Persons  
ALTER COLUMN Age int NOT NULL;

## **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.

**SQL UNIQUE CONSTRAINT ON CREATE TABLE**

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

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

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

## **SQL UNIQUE CONSTRAINT ON ALTER TABLE**

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

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

ALTER TABLE Persons  
ADD UNIQUE (ID);

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

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

ALTER TABLE Persons  
ADD CONSTRAINT UC\_Person UNIQUE (ID,LastName);

## **DROP A UNIQUE CONSTRAINT**

To drop a UNIQUE constraint, use the following SQL:

**MySQL:**

ALTER TABLE Persons  
DROP INDEX UC\_Person;

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

ALTER TABLE Persons  
DROP CONSTRAINT UC\_Person;

## **SQL 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)  
);

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

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

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

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

CREATE TABLE Persons (  
    ID int NOT NULL,  
    LastName varchar(255) NOT NULL,  
    FirstName varchar(255),  
    Age int,  
    CONSTRAINT PK\_Person PRIMARY KEY (ID,LastName)  
);

**Note:** In the example above there is only ONE PRIMARY KEY (PK\_Person). However, the VALUE of the primary key is made up of TWO COLUMNS (ID + LastName).

## **SQL PRIMARY KEY ON ALTER TABLE**

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

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

ALTER TABLE Persons  
ADD PRIMARY KEY (ID);

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

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

ALTER TABLE Persons  
ADD CONSTRAINT PK\_Person PRIMARY KEY (ID,LastName);

**Note:** If you use ALTER TABLE to add a primary key, the primary key column(s) must 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 Persons  
DROP PRIMARY KEY;

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

ALTER TABLE Persons  
DROP CONSTRAINT PK\_Person;

## **SQL 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](https://www.w3schools.com/sql/sql_primarykey.asp) 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.

Look at the following two tables:

### 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 |

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.

**SQL FOREIGN KEY ON CREATE TABLE**

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

**MySQL:**

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

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

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

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

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

CREATE TABLE Orders (  
    OrderID int NOT NULL,  
    OrderNumber int NOT NULL,  
    PersonID int,  
    PRIMARY KEY (OrderID),  
    CONSTRAINT FK\_PersonOrder FOREIGN KEY (PersonID)  
    REFERENCES Persons(PersonID)  
);

## **SQL FOREIGN KEY ON ALTER TABLE**

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

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

ALTER TABLE Orders  
ADD FOREIGN KEY (PersonID) REFERENCES Persons(PersonID);

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

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

ALTER TABLE Orders  
ADD CONSTRAINT FK\_PersonOrder  
FOREIGN KEY (PersonID) REFERENCES Persons(PersonID);

## **DROP A FOREIGN KEY CONSTRAINT**

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

**MySQL:**

ALTER TABLE Orders  
DROP FOREIGN KEY FK\_PersonOrder;

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

ALTER TABLE Orders  
DROP CONSTRAINT FK\_PersonOrder;

## **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 Server / Oracle / MS Access:**

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:

**MySQL / 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),  
    CONSTRAINT CHK\_Person CHECK (Age>=18 AND City='Sandnes')  
);

## **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'  
);

The DEFAULT constraint can also be used to insert system values, by using functions like [GETDATE()](https://www.w3schools.com/sql/func_sqlserver_getdate.asp):

CREATE TABLE Orders (  
    ID int NOT NULL,  
    OrderNumber int NOT NULL,  
    OrderDate date DEFAULT GETDATE()  
);