xDt: 12.03.2024

# **SQL Training Session – 3**

***Topics Covered:***

1. SQL Data Types
2. SQL Constraints

**SQL Data Types:**

SQL data types define the type of value that can be stored in a table column. For example, if you want a column to store only integer values, you can define its data type as INT.

SQL data types can be broadly divided into following categories:

1. Numeric data types such as: INT, TINYINT, BIGINT, FLOAT, REAL, etc.
2. Date and Time data types such as: DATE, TIME, DATETIME, etc
3. Character and String data types such as: CHAR, VARCHAR, TEXT, etc.
4. Unicode character string data types such as: NCHAR, NVARCHAR, NTEXT, etc.
5. Binary data type such as: BINARY, VARBINARY, etc.
6. Miscellaneous data types such as: CLOB, BLOB, XML, CURSOR, TABLE, etc.

**SQL Numeric Data Types:**

|  |  |  |
| --- | --- | --- |
| **Data Type** | **From** | **To** |
| BIT | 1 | 0 |
| TINYINT | 0 | 255 |
| SMALLINT | -32,768 | 32,767 |
| INT | -2,147,483,648 | 2,147,483,647 |
| BIGINT | -9,223,372,036,854,775,808 | 9,223,372,036,854,775,807 |
| DECIMAL | -10^38 + 1 | 10^38 - 1 |
| NUMERIC | -10^38 + 1 | 10^38 - 1 |
| FLOAT | -1.79E+308 | 1.79E+308 |
| REAL | -3.40E+38 | 3.40E+38 |

**SQL Date & Time Data Types:**

|  |  |
| --- | --- |
| **Data Type** | **Description** |
| DATE | Stores date in the format YYYY-MM-DD |
| TIME | Stores time in the format HH:MI: SS |
| DATETIME | Stores date and time information in the format YYYY-MM-DD HH:MI: SS |
| TIMESTAMP | Stores number of seconds passed since the Unix epoch ('1970-01-01 00:00:00' UTC) |
| YEAR | Stores year in a 2-digit or 4-digit format. Range 1901 to 2155 in 4-digit format. Range 70 to 69, representing 1970 to 2069. |

**SQL Character & String Data Types:**

|  |  |
| --- | --- |
| **Data Type** | **Description** |
| CHAR | Fixed length with a maximum length of 8,000 characters |
| VARCHAR | Variable-length storage with a maximum length of 8,000 characters |
| VARCHAR (max) | Variable-length storage with provided max characters, not supported in MySQL |
| TEXT | Variable-length storage with a maximum size of 2GB data |

**SQL Unicode Character & String Data Types:**

|  |  |
| --- | --- |
| **Data Type** | **Description** |
| NCHAR | Fixed length with a maximum length of 4,000 characters |
| NVARCHAR | Variable-length storage with a maximum length of 4,000 characters |
| NVARCHAR (max) | Variable-length storage with provided max characters |
| NTEXT | Variable-length storage with a maximum size of 1GB data |

**SQL Binary Data Types:**

|  |  |
| --- | --- |
| **Data Type** | **Description** |
| BINARY | Fixed length with a maximum length of 8,000 bytes |
| VARBINARY | Variable-length storage with a maximum length of 8,000 bytes |
| VARBINARY (max) | Variable-length storage with provided max bytes |
| IMAGE | Variable-length storage with a maximum size of 2GB binary data |

**SQL Binary Data Types:**

|  |  |
| --- | --- |
| **Data Type** | **Description** |
| CLOB | Character large objects that can hold up to 2 GB |
| BLOB | For large binary objects |
| XML | For storing XML data |
| JSON | For storing JSON data |

*Ref:* [*https://www.digitalocean.com/community/tutorials/sql-data-types*](https://www.digitalocean.com/community/tutorials/sql-data-types)

**SQL Constraints:**

In SQL, a constraint is any rule applied to a column or table that limits what data can be entered into it. Any time you attempt to perform an operation that changes that data held in a table — such as an INSERT, UPDATE, or DELETE statement — the RDBMS will test whether that data violates any existing constraints and, if so, return an error.

Users define constraints when they first create a table, or they can add them later on with an ALTER TABLE statement as long as it doesn’t conflict with any data already in the table. When you create a constraint, the database system will generate a name for it automatically, but in most SQL implementations you can add a custom name for any constraint. These names are used to refer to constraints in ALTER TABLE statements when changing or removing them.

The SQL standard formally defines just five constraints:

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

**PRIMARY KEY:**

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).

**EX:** The following SQL creates a PRIMARY KEY on the ‘ID’ column when the “Persons” table is created.

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

**FOREIGN KEY:**

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.

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

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

**UNIQUE KEY:**

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.

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

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

**CHECK:**

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.

**EX:** 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:

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

**NOT NULL:**

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.

**EX:** The following SQL ensures that the "ID", "LastName", and "FirstName" columns will NOT accept NULL values when the "Persons" table is created:

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

**DEFAULT:**

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.

**EX:** 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 'Sandnes');

*Ref:* *https://www.digitalocean.com/community/conceptual-articles/understanding-sql-constraints*