



SQL (Structured Query Language)

Module 1 (Basics of SQL)

Contents

Week 2

1.6 SQL	Comments	2
1.7 SQL	Constraints	3
1.8 SQL	Creating Roles	6
1.9 SQL	Indexes	7
2.0 SQL	Sequences	9

1.6 Comments

As in any programming languages comments matter a lot in SQL also. In this set we will learn about writing comments in any SQL snippet. A comment is a programmer-readable explanation or annotation in the source code of a computer program. They are added with the purpose of making the source code easier for humans to understand, and are generally ignored by compilers and interpreters. It is also used to explain sections of SQL statements, or to prevent execution of SQL statements.

Note: The examples in this chapter will not work in Firefox and Microsoft Edge! Comments are not supported in Microsoft Access databases. Firefox and Microsoft Edge are using Microsoft Access database in our examples.

Comments can be written in the following three formats:

1. Single line comments.
2. Multi line comments
3. In line comments

Single line comments: Comments starting and ending in a single line are considered as single line comments. Line starting with ‘--’ is a comment and will not be executed.

Syntax:

```
-- single line comment
-- another comment
SELECT * FROM Customers;
```

Multi line comments: Comments starting in one line and ending in different line are considered as multi line comments. Line starting with ‘/*’ is considered as starting point of comment and are terminated when ‘*/’ is encountered.

Syntax:

```
/* multi line comment
another comment */
SELECT * FROM Customers;
```

Multi line comment ->

```
/* SELECT * FROM Students;
SELECT * FROM STUDENT_DETAILS;
SELECT * FROM Orders; */
SELECT * FROM Articles;
```

In line comments: In line comments are an extension of multi-line comments, comments can be stated in between the statements and are enclosed in between ‘/*’ and ‘*/’.

Syntax:

```
SELECT * FROM /* Customers; */
```

In line comment ->

```
SELECT * FROM Students;
SELECT * FROM /* STUDENT_DETAILS;
SELECT * FROM Orders;
SELECT * FROM /* Articles;
```

1.7 Constraint

Constraints are the rules that we can apply on the type of data in a table. That is, we can specify the limit on the type of data that can be stored in a particular column in a table using constraints. 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 available constraints in SQL are:

- ❖ **NOT NULL:** This constraint tells that we cannot store a null value in a column. That is, if a column is specified as NOT NULL then we will not be able to store null in this particular column any more.
Ensures that a column cannot have a NULL value
For example, the below query creates a table Student with the fields ID and NAME as NOT NULL. That is, we are bound to specify values for these two fields every time we wish to insert a new row.

```
CREATE TABLE Student
(
  ID int(6) NOT NULL,
  NAME varchar(10) NOT NULL,
  ADDRESS varchar(20)
);
```

- ❖ **UNIQUE:** This constraint when specified with a column, tells that all the values in the column must be unique. That is, the values in any row of a column must not be repeated.
Ensures that all values in a column are different
For example, the below query creates a table Student where the field ID is specified as UNIQUE. i.e, no two students can have the same ID. Unique constraint in detail.

```
CREATE TABLE Student
(
  ID int(6) NOT NULL UNIQUE,
  NAME varchar(10),
  ADDRESS varchar(20)
);
```

- ❖ **PRIMARY KEY:** A primary key is a field which can uniquely identify each row in a table. And this constraint is used to specify a field in a table as primary key.
Uniquely identifies a row/record in another table
A table can have only one field as primary key. Below query will create a table named Student and specifies the field ID as primary key.

```
CREATE TABLE Student
(
  ID int(6) NOT NULL UNIQUE,
  NAME varchar(10),
  ADDRESS varchar(20),
  PRIMARY KEY(ID)
);
```

- ❖ **FOREIGN KEY:** A Foreign key is a field which can uniquely identify each row in a another table. And this constraint is used to specify a field as Foreign key.

Uniquely identifies a row/record in another table

O_ID ORDER_NO C_ID

1	2253	3
2	3325	3
3	4521	2
4	8532	1

C_ID NAME ADDRESS

1	RAMESH	DELHI
2	SURESH	NOIDA
3	DHARMESH	GURGAON

what we have here is Order table and Customer table respectively.

As we can see clearly that the field C_ID in Orders table is the primary key in Customers table, i.e. it uniquely identifies each row in the Customers table. Therefore, it is a Foreign Key in Orders table.

Syntax:


```
CREATE TABLE Orders
(
O_ID int NOT NULL,
ORDER_NO int NOT NULL,
C_ID int,
PRIMARY KEY (O_ID),
FOREIGN KEY (C_ID) REFERENCES Customers(C_ID)
)
```

- ❖ **CHECK:** This constraint helps to validate the values of a column to meet a particular condition. That is, it helps to ensure that the value stored in a column meets a specific condition.

Ensures that all values in a column satisfies a specific condition

For example, the below query creates a table Student and specifies the condition for the field AGE as (**AGE >= 18**). That is, the user will not be allowed to enter any record in the table with AGE < 18.

```
CREATE TABLE Student
(
ID int(6) NOT NULL,
NAME varchar(10) NOT NULL,
AGE int NOT NULL CHECK (AGE >= 18)
);
```

- 
- ❖ **DEFAULT:** This constraint specifies a default value for the column when no value is specified by the user.
Sets a default value for a column when no value is specified
For example, the below query will create a table named Student and specify the default value for the field AGE as 18.

```
CREATE TABLE Student  
(  
    ID int(6) NOT NULL,  
    NAME varchar(10) NOT NULL,  
    AGE int DEFAULT 18  
);
```
 - ❖ **Index :** Used to create and retrieve data from the database very quickly

How to specify constraints?

We can specify constraints at the time of creating the table using CREATE TABLE statement. We can also specify the constraints after creating a table using ALTER TABLE statement.

Syntax:

```
CREATE TABLE sample_table
```

```
(
```

```
column1 data_type(size) constraint_name,
```

```
column2 data_type(size) constraint_name,
```

```
column3 data_type(size) constraint_name,
```

```
....
```

```
);
```

sample_table: Name of the table to be created.

data_type: Type of data that can be stored in the field.

constraint_name: Name of the constraint. for example- NOT NULL, UNIQUE, PRIMARY KEY etc.

1.8 Creating Roles

A role is created to ease setup and maintenance of the security model. It is a named group of related privileges that can be granted to the user. When there are many users in a database it becomes difficult to grant or revoke privileges to users. Creates a set of privileges which may be assigned to users of a database. Once a role is assigned to a user, (s)he gets all the Privileges of that role. By creating and granting roles, best means of database security can be practiced. Therefore, if you define roles:

- ❖ You can grant or revoke privileges to users, thereby automatically granting or revoking privileges.
- ❖ You can either create Roles or use the system roles pre-defined.

Here are the said privileges:

System Roles Privileges granted to the Role

- ❖ **Connect**
Create table, Create view, Create synonym, Create sequence, Create session etc.
- ❖ **Resource**
Create Procedure, Create Sequence, Create Table, Create Trigger etc. The primary usage of the Resource role is to restrict access to database objects.
- ❖ **DBA**
All system privileges

Creating and Assigning a Role –

First, the (Database Administrator)DBA must create the role. Then the DBA can assign privileges to the role and users to the role.

Syntax –

```
CREATE ROLE manager;  
Role created
```

- ✓ Now that the role is created, the DBA can use the GRANT statement to assign users to the role as well as assign privileges to the role.
- ✓ It's easier to GRANT or REVOKE privileges to the users through a role rather than assigning a privilege directly to every user.
- ✓ If a role is identified by a password, then GRANT or REVOKE privileges have to be identified by the password.

Grant privileges to a role –

```
GRANT create table, create view  
TO manager;  
Grant succeeded.
```

Grant a role to users

```
GRANT manager TO SAM, STARK;  
Grant succeeded.
```

Revoke privilege from a Role :

```
REVOKE create table FROM manager;
```

Drop a Role :

```
DROP ROLE manager;
```

Explanation – Firstly it creates a manager role and then allows managers to create tables and views. It then grants Sam and Stark the role of managers. Now Sam and Stark can create tables and views. If users have multiple roles granted to them, they receive all of the privileges associated with all of the roles. Then create table privilege is removed from role 'manager' using Revoke. The role is dropped from the database using drop.

1.9 Indexes

An index is a schema object. It is used by the server to speed up the retrieval of rows by using a pointer. It can reduce disk I/O(input/output) by using a rapid path access method to locate data quickly. An index helps to speed up select queries and where clauses, but it slows down data input, with the update and the insert statements. Indexes can be created or dropped with no effect on the data. Indexes are special lookup tables that the database search engine can use to speed up data retrieval. Simply put, an index is a pointer to data in a table. An index in a database is very similar to an index in the back of a book.

For example, if you want to reference all pages in a book that discusses a certain topic, you first refer to the index, which lists all the topics alphabetically and are then referred to one or more specific page numbers. An index helps to speed up SELECT queries and WHERE clauses, but it slows down data input, with the UPDATE and the INSERT statements. Indexes can be created or dropped with no effect on the data.

Creating an Index –

It's syntax is:

```
CREATE INDEX index  
ON TABLE column;
```

Where the index is the name given to that index and TABLE is the name of the table on which that index is created and column is the name of that column for which it is applied.

For multiple columns –

```
CREATE INDEX index  
ON TABLE (column1, column2,.....);
```

Unique Indexes –

```
CREATE UNIQUE INDEX index  
ON TABLE column;
```

Used for the maintenance of the integrity of the data present in the table as well as for the fast performance, it does not allow multiple values to enter into the table.

When should indexes be created –

- A column contains a wide range of values
- A column does not contain a large number of null values
- One or more columns are frequently used together in a where clause or a join condition

When should indexes be avoided –

- The table is small
- The columns are not often used as a condition in the query
- The column is updated frequently

Removing an Index –

```
DROP INDEX index;
```

Must be the owner of the index or have the DROP ANY INDEX privilege.

Altering an Index:

- ALTER INDEX IndexName
- ON TableName REBUILD;
-

Confirming Indexes

```
select * from USER_INDEXES;
```

It will show you all the indexes present in the server, in which you can locate your own tables too.

Renaming an index

```
EXEC sp_rename  
index_name,  
new_index_name,  
N'INDEX';
```


2.0 Sequences

A sequence is a user defined schema bound object that generates a sequence of numeric values. Are frequently used in many databases because many applications require each row in a table to contain a unique value and sequences provides an easy way to generate them. The sequence of numeric values is generated in an ascending or descending order at defined intervals and can be configured to restart when exceeds max_value.

AUTO_INCREMENT is applied on columns, it automatically increments the column value by 1 each time a new record is inserted into the table. Sequence is also somewhat similar to AUTO_INCREMENT but it has some additional features too.

Syntax:

```
CREATE SEQUENCE sequence_name  
START WITH initial_value  
INCREMENT BY increment_value  
MINVALUE minimum value  
MAXVALUE maximum value  
CYCLE|NOCYCLE ;
```

sequence_name: Name of the sequence.

initial_value: starting value from where the sequence starts.

Initial_value should be greater than or equal to minimum value and less than equal to maximum value.

increment_value: Value by which sequence will increment itself.
Increment_value can be positive or negative.

minimum_value: Minimum value of the sequence.
maximum_value: Maximum value of the sequence.

cycle: When sequence reaches its set_limit it starts from beginning.

nocycle: An exception will be thrown if sequence exceeds its max_value.

Creating sequence in ascending order.

```
CREATE SEQUENCE sequence_1  
start with 1  
increment by 1  
minvalue 0  
maxvalue 100  
cycle;
```

Above query will create a sequence named sequence_1. Sequence will start from 1 and will be incremented by 1 having maximum value 100. Sequence will repeat itself from start value after exceeding 100.

Creating sequence in descending order.

```
CREATE SEQUENCE sequence_2
start with 100
increment by -1
minvalue 1
maxvalue 100
cycle;
```

Above query will create a sequence named sequence_2. Sequence will start from 100 and should be less than or equal to maximum value and will be incremented by -1 having minimum value 1.

Example to use sequence : create a table named students with columns as id and name.

```
CREATE TABLE students
(
ID number(10),
NAME char(20)
);
```

Now insert values into table

```
INSERT into students VALUES(sequence_1.nextval,'Ramesh');
INSERT into students VALUES(sequence_1.nextval,'Suresh');
```

where sequence_1.nextval will insert id's in id column in a sequence as defined in sequence_1.

Output:

ID	NAME
1	Ramesh
2	Suresh