UNIT-3

Basics of SQL-DDL, DML, DCL,TCL

Structured Query Language (SQL) as we all know is the database language by the use of which we can perform certain operations on the existing database and also we can use this language to create a database. SQL uses certain commands like Create, Drop, Insert etc. to carry out the required tasks.

These SQL commands are mainly categorized into four categories as discussed below:

1. **DDL(Data Definition Language) :** DDL or Data Definition Language actually consists of the SQL commands that can be used to define the database schema. It simply deals with descriptions of the database schema and is used to create and modify the structure of database objects in database.

Examples of DDL commands:

- CREATE – is used to create the database or its objects (like table, index, function, views, store procedure and triggers).
- DROP – is used to delete objects from the database.
- ALTER–is used to alter the structure of the database.
- TRUNCATE–is used to remove all records from a table, including all spaces allocated for the records are removed.
- COMMENT –is used to add comments to the data dictionary.
- RENAME –is used to rename an object existing in the database.

2. **DML(Data Manipulation Language) :** The SQL commands that deals with the manipulation of data present in database belong to DML or Data Manipulation Language and this includes most of the SQL statements.

Examples of DML:

- SELECT – is used to retrieve data from the database.
- INSERT – is used to insert data into a table.
- UPDATE – is used to update existing data within a table.
- DELETE – is used to delete records from a database table.

3. **DCL (Data Control Language) :** DCL includes commands such as GRANT and REVOKE which mainly deals with the rights, permissions and other controls of the database system.

Examples of DCL commands:

- **GRANT**–gives user's access privileges to database.
- **REVOKE**–withdraw user's access privileges given by using the GRANT command.

4. **TCL (transaction Control Language) :** TCL commands deals with the transaction within the database.

Examples of TCL commands:

- **COMMIT**– commits a Transaction.
- **ROLLBACK**– rollbacks a transaction in case of any error occurs.
- **SAVEPOINT**–sets a savepoint within a transaction.
- **SET TRANSACTION**–specify characteristics for the transaction.

DDL COMMANDS

1. **CREATE** It is used to create a new table in the database.

CREATE TABLE TABLE_NAME (COLUMN_NAME DATATYPES[DATA SIZE]);

```
CREATE TABLE EMPLOYEE(Name VARCHAR2(20), Email VARCHAR2(100), DOB DATE);
```

2. **DROP:** It is used to delete both the structure and record stored in the table.
   DROP TABLE ;

```
DROP TABLE EMPLOYEE;
```

3. **ALTER:** It is used to alter the structure of the database. This change could be either to modify the characteristics of an existing attribute or probably to add a new attribute.
   To add a new column in the table:
   ALTER TABLE table_name ADD column_name COLUMN-definition;
   To modify existing column in the table:
   ALTER TABLE MODIFY(COLUMN DEFINITION....);

```
 ALTER TABLE STU_DETAILS ADD(ADDRESS VARCHAR2(20));
ALTER TABLE STU_DETAILS MODIFY (NAME VARCHAR2(20));
```

4. **TRUNCATE:** It is used to delete all the rows from the table and free the space containing the table.
   TRUNCATE TABLE table_name;

```
TRUNCATE TABLE EMPLOYEE;
```

DML COMMANDS

1. **INSERT:** The INSERT statement is a SQL query. It is used to insert data into the row of a table.
   INSERT INTO TABLE_NAME
   (col1, col2, col3,.... col N)
   VALUES (value1, value2, value3, ... valueN);
   OR
   INSERT INTO TABLE_NAME
   VALUES (value1, value2, value3, ... valueN);

```
INSERT INTO SUB (Author, Subject) VALUES ("XYZ", "DBMS");
```

2. **UPDATE:** This command is used to update or modify the value of a column in the table.

   UPDATE table_name SET [column_name1= value1,...column_nameN = valueN] [
   WHERE CONDITION]

```
UPDATE students
SET User_Name = 'ABC'
WHERE Student_Id = '3'
```

3. **DELETE:** It is used to remove one or more row from a table.
   DELETE FROM table_name [WHERE condition];

```
 DELETE FROM SUB
WHERE Author="Sonoo";
```

DCL COMMANDS

1. **Grant:** It is used to give user access privileges to a database.
   GRANT SELECT, UPDATE ON MY_TABLE TO SOME_USER, ANOTHER_USER;
2. **Revoke:** It is used to take back permissions from the user.
   REVOKE SELECT, UPDATE ON MY_TABLE FROM USER1, USER2;

TCL COMMANDS

1. **Commit:** Commit command is used to save all the transactions to the database.
   COMMIT;

```
DELETE FROM CUSTOMERS
WHERE AGE = 25;
COMMIT;
```

2. **Rollback:** Rollback command is used to undo transactions that have not already been saved to the database.
   ROLLBACK;

```
DELETE FROM CUSTOMERS
WHERE AGE = 25;
ROLLBACK;
```

3. **SAVEPOINT:** It is used to roll the transaction back to a certain point without rolling back the entire transaction.
   SAVEPOINT SAVEPOINT_NAME;

Data Query Language

DQL is used to fetch the data from the database.

It uses only one command:

- SELECT

1. **SELECT:** This is the same as the projection operation of relational algebra. It is used to select the attribute based on the condition described by WHERE clause.

Syntax:

```
SELECT expressions
FROM TABLES
WHERE conditions;
```

SQL Constraints

SQL Constraints are rules used to limit the type of data that can go into a table, to maintain the accuracy and integrity of the data inside table.

Constraints can be divided into the following two types,

1. **Column level constraints:** Limits only column data.
2. **Table level constraints:** Limits whole table data.

Constraints are used to make sure that the integrity of data is maintained in the database. Following are the most used constraints that can be applied to a table.

- NOT NULL

- UNIQUE
- PRIMARY KEY
- FOREIGN KEY
- CHECK
- DEFAULT

NOT NULL Constraint

By default, a column can hold NULL values. If you do not want a column to have a NULL value, use the NOT NULL constraint.

- It restricts a column from having a NULL value.
- We use ALTER statement and MODIFY statement to specify this constraint.

One important point to note about this constraint is that it cannot be defined at table level.

```
CREATE TABLE Student

(      s_id int NOT NULL,

       name varchar(60),

       age  int

);
```

```
ALTER TABLE Student

MODIFY s_id int NOT NULL;
```

UNIQUE Constraint

It ensures that a column will only have unique values. A UNIQUE constraint field cannot have any duplicate data.

- It prevents two records from having identical values in a column
- We use ALTER statement and MODIFY statement to specify this constraint.

```
CREATE TABLE Student

(       s_id int  NOT NULL,

        name varchar(60),

        age int  NOT NULL UNIQUE);
```

Primary Key Constraint

Primary key constraint uniquely identifies each record in a database. A Primary Key must contain unique value and it must not contain null value. Usually Primary Key is used to index the data inside the table.

PRIMARY KEY constraint at Table Level

```
CREATE table Student

(      s_id int PRIMARY KEY,

Name varchar(60) NOT NULL,

       Age int);
```

PRIMARY KEY constraint at Column Level

```
ALTER table Student

ADD PRIMARY KEY (s_id);
```

Foreign Key Constraint

Foreign Key is used to relate two tables. The relationship between the two tables matches the Primary Key in one of the tables with a Foreign Key in the second table.

- This is also called a referencing key.
- We use ALTER statement and ADD statement to specify this constraint.
- **Customer_Detail** Table

c_id	Customer_Name	address
101	Adam	Noida
102	Alex	Delhi

103	Stuart	Rohtak

- **Order_Detail** Table

Order_id	Order_Name	c_id
10	Order1	101
11	Order2	103
12	Order3	102

- In **Customer_Detail** table, **c_id** is the primary key which is set as foreign key in **Order_Detail** table. The value that is entered in **c_id** which is set as foreign key in **Order_Detail** table must be present in **Customer_Detail** table where it is set as primary key. This prevents invalid data to be inserted into **c_id** column of **Order_Detail** table.
- If you try to insert any incorrect data, DBMS will return error and will not allow you to insert the data.

FOREIGN KEY constraint at Table Level

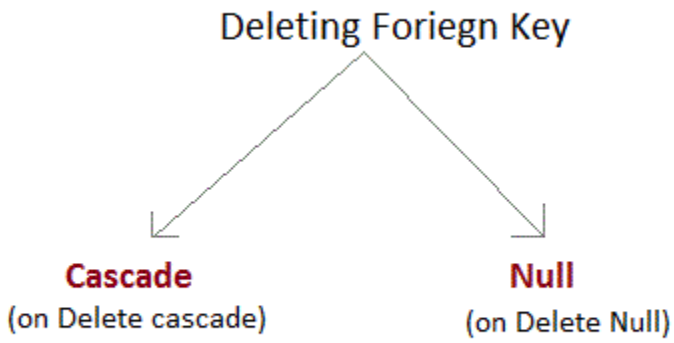
```
CREATE table Order_Detail(
    order_id int PRIMARY KEY,
    order_name varchar(60) NOT NULL,
    c_id int FOREIGN KEY REFERENCES Customer_Detail(c_id)
);
```

FOREIGN KEY constraint at Column Level

```
ALTER table Order_Detail
ADD FOREIGN KEY (c_id) REFERENCES Customer_Detail(c_id);
```

Behaviour of Foriegn Key Column on Delete

There are two ways to maintin the integrity of data in Child table, when a particular record is deleted in the main table. When two tables are connected with Foriegn key, and certain data in the main table is deleted, for which a record exits in the child table, then we must have some mechanism to save the integrity of data in the child table.



1. **On Delete Cascade:** This will remove the record from child table, if that value of foreign key is deleted from the main table.
2. **On Delete Null:** This will set all the values in that record of child table as NULL, for which the value of foreign key is deleted from the main table.

CHECK Constraint

CHECK constraint is used to restrict the value of a column between a range. It performs check on the values, before storing them into the database. Its like condition checking before saving data into a column.

Using CHECK constraint at Table Level

```
CREATE table Student(

    s_id int NOT NULL CHECK(s_id > 0),

    Name varchar(60) NOT NULL,

    Age int

);
```

Using `CHECK` constraint at Column Level

```
ALTER table Student ADD CHECK(s_id > 0);
```

The SQL IN Operator

The IN operator allows you to specify multiple values in a WHERE clause.

The IN operator is a shorthand for multiple OR conditions.

IN Syntax

```
SELECT column_name(s)
FROM table_name
WHERE column_name IN (value1, value2, ...);
```

OR

```
SELECT column_name(s)
FROM table_name
WHERE column_name IN (SELECT STATEMENT);
```

SAME AS NOT IN COMMAND

Aggregate Functions

These functions **return a single value** after performing calculations on a group of values. Following are some of the frequently used Aggregrate functions.

1. AVG
2. SUM
3. COUNT
4. MIN
5. MAX

`AVG()` Function

Average returns average value after calculating it from values in a numeric column.

Its general **syntax** is,

```
SELECT AVG(column_name) FROM table_name
```

Using AVG() function

Consider the following **Emp** table

Eid	name	age	salary
401	Anu	22	9000
402	Shane	29	8000
403	Rohan	34	6000
404	Scott	44	10000
405	Tiger	35	8000

SQL query to find average salary will be,

```
SELECT avg(salary) from Emp;
```

Result of the above query will be,

avg(salary)
8200

`COUNT()` Function

Count returns the number of rows present in the table either based on some condition or without condition.

Its general **syntax** is,

```
SELECT COUNT(column_name) FROM table-name
```

Using COUNT() function

Consider the following **Emp** table

eid	name	age	salary
401	Anu	22	9000
402	Shane	29	8000
403	Rohan	34	6000
404	Scott	44	10000
405	Tiger	35	8000

SQL query to count employees, satisfying specified condition is,

```
SELECT COUNT(name) FROM Emp WHERE salary = 8000;
```

Result of the above query will be,

count(name)

2

Example of COUNT(distinct)

Consider the following **Emp** table

Eid	name	age	salary
401	Anu	22	9000
402	Shane	29	8000
403	Rohan	34	6000
404	Scott	44	10000
405	Tiger	35	8000

SQL query is,

```
SELECT COUNT(DISTINCT salary) FROM emp;
```

Result of the above query will be,

count(distinct salary)
4

`FIRST()` Function

First function returns first value of a selected column

Syntax for FIRST function is,

```
SELECT FIRST(column_name) FROM table-name;
```

Using FIRST() function

Consider the following **Emp** table

Eid	name	age	salary
401	Anu	22	9000
402	Shane	29	8000
403	Rohan	34	6000
404	Scott	44	10000
405	Tiger	35	8000

SQL query will be,

```
SELECT FIRST(salary) FROM Emp;
```

and the result will be,

first(salary)
9000

`LAST()` Function

LAST function returns the return last value of the selected column.

Syntax of LAST function is,

```
SELECT LAST(column_name) FROM table-name;
```

Using LAST() function

Consider the following **Emp** table

Eid	name	age	salary
401	Anu	22	9000
402	Shane	29	8000
403	Rohan	34	6000
404	Scott	44	10000

405	Tiger	35	8000

SQL query will be,

```
SELECT LAST(salary) FROM emp;
```

Result of the above query will be,

last(salary)
8000

`MAX()` Function

MAX function returns maximum value from selected column of the table.

Syntax of MAX function is,

```
SELECT MAX(column_name) from table-name;
```

Using MAX() function

Consider the following **Emp** table

Eid	name	age	salary
401	Anu	22	9000
402	Shane	29	8000

403	Rohan	34	6000
404	Scott	44	10000
405	Tiger	35	8000

SQL query to find the Maximum salary will be,

```
SELECT MAX(salary) FROM emp;
```

Result of the above query will be,

MAX(salary)
10000

`MIN()` Function

MIN function returns minimum value from a selected column of the table.

Syntax for MIN function is,

```
SELECT MIN(column_name) from table-name;
```

Using MIN() function

Consider the following **Emp** table,

Eid	**name**	**age**	**salary**

401	Anu	22	9000
402	Shane	29	8000
403	Rohan	34	6000
404	Scott	44	10000
405	Tiger	35	8000

SQL query to find minimum salary is,

```
SELECT MIN(salary) FROM emp;
```

Result will be,

MIN(salary)
6000

`SUM()` Function

SUM function returns total sum of a selected columns numeric values.

Syntax for SUM is,

```
SELECT SUM(column_name) from table-name;
```

Using SUM() function

Consider the following **Emp** table

Eid	name	age	salary
401	Anu	22	9000
402	Shane	29	8000
403	Rohan	34	6000
404	Scott	44	10000
405	Tiger	35	8000

SQL query to find sum of salaries will be,

```
SELECT SUM(salary) FROM emp;
```

Result of above query is,

SUM(salary)
41000

Scalar Functions

Scalar functions return a single value from an input value. Following are some frequently used Scalar Functions in SQL.

`UCASE()` Function

UCASE function is used to convert value of string column to Uppercase characters.

Syntax of UCASE,

```
SELECT UCASE(column_name) from table-name;
```

Using UCASE() function

Consider the following **Emp** table

Eid	name	age	salary
401	anu	22	9000
402	shane	29	8000
403	rohan	34	6000
404	scott	44	10000
405	Tiger	35	8000

SQL query for using UCASE is,

```
SELECT UCASE(name) FROM emp;
```

Result is,

UCASE(name)
ANU
SHANE
ROHAN
SCOTT
TIGER

`LCASE()` Function

LCASE function is used to convert value of string columns to Lowecase characters.

Syntax for LCASE is,

```
SELECT LCASE(column_name) FROM table-name;
```

Using LCASE() function

Consider the following **Emp** table

Eid	name	age	salary

401	Anu	22	9000
402	Shane	29	8000
403	Rohan	34	6000
404	SCOTT	44	10000
405	Tiger	35	8000

SQL query for converting string value to Lower case is,

```
SELECT LCASE(name) FROM emp;
```

Result will be,

LCASE(name)
Anu
Shane
Rohan
Scott

Tiger

`MID()` Function

MID function is used to extract substrings from column values of string type in a table.

Syntax for MID function is,

```
SELECT MID(column_name, start, length) from table-name;
```

Using MID() function

Consider the following **Emp** table

Eid	name	age	salary
401	anu	22	9000
402	shane	29	8000
403	rohan	34	6000
404	scott	44	10000
405	Tiger	35	8000

SQL query will be,

```
SELECT MID(name,2,2) FROM emp;
```

Result will come out to be,

MID(name,2,2)
Nu
Ha
Oh
Co
Ig

`ROUND()` Function

ROUND function is used to round a numeric field to number of nearest integer. It is used on Decimal point values.

Syntax of Round function is,

```
SELECT ROUND(column_name, decimals) from table-name;
```

Using ROUND() function

Consider the following **Emp** table

Eid	name	age	salary

401	anu	22	9000.67
402	shane	29	8000.98
403	rohan	34	6000.45
404	scott	44	10000
405	Tiger	35	8000.01

SQL query is,

```
SELECT ROUND(salary) from emp;
```

Result will be,

ROUND(salary)
9001
8001
6000
10000

8000

The SQL AND, OR and NOT Operators

The WHERE clause can be combined with AND, OR, and NOT operators.

The AND and OR operators are used to filter records based on more than one condition:

- The AND operator displays a record if all the conditions separated by AND are TRUE.
- The OR operator displays a record if any of the conditions separated by OR is TRUE.

The NOT operator displays a record if the condition(s) is NOT TRUE.

AND Syntax

```
SELECT column1, column2, ...
FROM table_name
WHERE condition1 AND condition2 AND condition3 ...;
```

OR Syntax

```
SELECT column1, column2, ...
FROM table_name
WHERE condition1 OR condition2 OR condition3 ...;
```

NOT Syntax

```
SELECT column1, column2, ...
FROM table_name
WHERE NOT condition;
```

ORDER BY Keyword

The ORDER BY keyword is used to sort the result-set in ascending or descending order.

The ORDER BY keyword sorts the records in ascending order by default. To sort the records in descending order, use the DESC keyword.

```
SELECT column1, column2, ...
FROM table_name
ORDER BY column1, column2, ... ASC|DESC;
```

LIKE Operator

The LIKE operator is used in a WHERE clause to search for a specified pattern in a column.

There are two wildcards often used in conjunction with the LIKE operator:

- % - The percent sign represents zero, one, or multiple characters
- _ - The underscore represents a single character

The percent sign and the underscore can also be used in combinations.

```
SELECT column1, column2, ...
FROM table_name
WHERE columnN LIKE pattern;
```

LIKE Operator	Description
WHERE CustomerName LIKE 'a%'	Finds any values that start with "a"
WHERE CustomerName LIKE '%a'	Finds any values that end with "a"
WHERE CustomerName LIKE '%or%'	Finds any values that have "or" in any position
WHERE CustomerName LIKE '_r%'	Finds any values that have "r" in the second position

WHERE CustomerName LIKE 'a_%'	Finds any values that start with "a" and are at least 2 c length
WHERE CustomerName LIKE 'a__%'	Finds any values that start with "a" and are at least 3 c length
WHERE ContactName LIKE 'a%o'	Finds any values that start with "a" and ends with "o"

BETWEEN Operator

The BETWEEN operator selects values within a given range. The values can be numbers, text, or dates.

The BETWEEN operator is inclusive: begin and end values are included.

```
SELECT column_name(s)
FROM table_name
WHERE column_name BETWEEN value1 AND value2;
```

Example: Find the names of all instructors with salary between 90,000 and 100,000 (90,000 and £ 100,000)

```
select name
  from instructor
  where salary between 90000 and 100000;
```