BOSTON
TRAINING ACADEMY

SQL – STRUCTURE QUERY LANGUAGE

© Confidentiality & Proprietary Information

This document contains information that is Proprietary and confidential ("Confidential Information") to Boston Training Academy and shall not be used or disclosed outside. Further, the Confidential Information should not be transmitted, duplicated, or used in whole or in part for any purpose other than what it is intended for herein. Any use or disclosure in whole or in part of this Confidential Information without the express written permission of Boston Training Academy is strictly prohibited.

This is a confidential document prepared by Boston Training Academy.

The illustrative formats and examples have been created solely to simulate Learning and do not purport to represent/reflect on work practices of any particular party/parties. Unauthorized possession of the material or disclosure of the Proprietary information may result in legal action.

©Boston Training Academy 2021

8. SQL – CREATE Database

The SQL **CREATE DATABASE** statement is used to create a new SQL database.

Syntax

The basic syntax of this CREATE DATABASE statement is as follows:

```
CREATE DATABASE DatabaseName;
```

Always the database name should be unique within the RDBMS.

Example

If you want to create a new database <testDB>, then the CREATE DATABASE statement would be as shown below:

```
SQL> CREATE DATABASE testDB;
```

Make sure you have the admin privilege before creating any database. Once a database is created, you can check it in the list of databases as follows:

```
SQL> SHOW DATABASES;
+--------------------------------- +
| Database                         |
+--------------------------------- +
| information_schema |
| AMROOD                           |
| d                                |
| mysql                            |
| orig                             |
| test                             |
| testDB                           |
+--------------------------------- +
7 rows in set (0.00 sec)
```

9. SQL – DROP or DELETE Database SQL

The SQL **DROP DATABASE** statement is used to drop an existing database in SQL schema.

Syntax

The basic syntax of DROP DATABASE statement is as follows:

```
DROP DATABASE DatabaseName;
```

Always the database name should be unique within the RDBMS.

Example

If you want to delete an existing database <testDB>, then the DROP DATABASE statement would be as shown below:

```
SQL> DROP DATABASE testDB;
```

NOTE: Be careful before using this operation because by deleting an existing database would result in loss of complete information stored in the database.

Make sure you have the admin privilege before dropping any database. Once a database is dropped, you can check it in the list of the databases as shown below:

```
SQL> SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| AMROOD             |
| s                  |
| mysql              |
| orig               |
| test               |
+--------------------+
6 rows in set (0.00 sec)
```

10. SQL – SELECT Database, USE Statement SQL

When you have multiple databases in your SQL Schema, then before starting your operation, you would need to select a database where all the operations would be performed.

The SQL **USE** statement is used to select any existing database in the SQL schema.

Syntax

The basic syntax of the USE statement is as shown below:

```
USE  DatabaseName;
```

Always the database name should be unique within the RDBMS.

Example

You can check the available databases as shown below:

```
SQL> SHOW DATABASES;
+--------------------------------- +
| Database                        |
+--------------------------------- +
| information_schema |
| AMROOD                          |
| d                               |
| mysql                           |
| orig                            |
| test                            |
+--------------------------------- +
6 rows in set (0.00 sec)
```

Now, if you want to work with the AMROOD database, then you can execute the following SQL command and start working with the AMROOD database.

```
SQL> USE AMROOD;
```

11. SQL – CREATE Table

Creating a basic table involves naming the table and defining its columns and each column's data type.

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

Syntax

The basic syntax of the CREATE TABLE statement is as follows:

```
CREATE TABLE table_name(
    column1 datatype,
    column2 datatype,
    column3 datatype,
    .....
    columnN datatype,
    PRIMARY KEY( one or more columns )
);
```

CREATE TABLE is the keyword telling the database system what you want to do. In this case, you want to create a new table. The unique name or identifier for the table follows the CREATE TABLE statement.

Then in brackets comes the list defining each column in the table and what sort of data type it is. The syntax becomes clearer with the following example.

A copy of an existing table can be created using a combination of the CREATE TABLE statement and the SELECT statement. You can check the complete details at Create Table Using another Table.

Example

The following code block is an example, which creates a CUSTOMERS table with an ID as a primary key and NOT NULL are the constraints showing that these fields cannot be NULL while creating records in this table:

```
SQL> CREATE TABLE CUSTOMERS(
    ID      INT              NOT NULL,
    NAME VARCHAR (20)        NOT NULL,
    AGE   INT                NOT NULL,
    ADDRESS   CHAR (25) ,
    SALARY    DECIMAL (18, 2),
    PRIMARY KEY (ID)
);
```

You can verify if your table has been created successfully by looking at the message displayed by the SQL server, otherwise you can use the **DESC** command as follows:

```
SQL> DESC CUSTOMERS;
+---------+---------------+------+-----+---------+ ----------+
| Field   | Type          | Null | Key | Default | Extra |
+---------+---------------+------+-----+---------+ ----------+
| ID      | int(11)       | NO   |PRI|           |           |
| NAME    | varchar(20)   | NO   |     |         |           |
| AGE     | int(11)       | NO   |     |         |           |
| ADDRESS | char(25)      | YES  |     | NULL    |           |
| SALARY  | decimal(18,2) | YES  |     | NULL    |           |
+---------+---------------+------+-----+---------+ ----------+
5 rows in set (0.00 sec)
```

Now, you have CUSTOMERS table available in your database which you can use to store the required information related to customers.

SQL - Creating a Table from an Existing Table

A copy of an existing table can be created using a combination of the CREATE TABLE statement and the SELECT statement. The new table has the same column definitions. All columns or specific columns can be selected. When you will create a new table using the existing table, the new table would be populated using the existing values in the old table.

Syntax

The basic syntax for creating a table from another table is as follows:

```
CREATE TABLE NEW_TABLE_NAME AS
    SELECT [ column1, column2...columnN ]
    FROM EXISTING_TABLE_NAME
    [ WHERE ]
```

Here, column1, column2... are the fields of the existing table and the same would be used to create fields of the new table.

Example

Following is an example which would create a table SALARY using the CUSTOMERS table and having the fields – customer ID and customer SALARY:

```
SQL> CREATE TABLE SALARY AS
    SELECT ID, SALARY
    FROM CUSTOMERS;
```

This would create a new table SALARY which will have the following records.

```
+--------+----------+
| ID | SALARY     |
+--------+----------+
|    1 |   2000.00 |
|    2 |   1500.00 |
|    3 |   2000.00 |
|    4 |   6500.00 |
|    5 |   8500.00 |
|    6 |   4500.00 |
|    7 |  10000.00 |
+--------+----------+
```

12. SQL – DROP or DELETE Table

The SQL **DROP TABLE** statement is used to remove a table definition and all the data, indexes, triggers, constraints and permission specifications for that table.

NOTE: You should be very careful while using this command because once a table is deleted then all the information available in that table will also be lost forever.

Syntax

The basic syntax of this DROP TABLE statement is as follows:

```
DROP TABLE  table_name;
```

Example

Let us first verify the CUSTOMERS table and then we will delete it from the database as shown below.

```
SQL> DESC CUSTOMERS;
+---------+---------------+------+-----+-----------------+ -------------+
| Field   | Type          | Null | Key | Default         | Extra |
+---------+---------------+------+-----+---------+ -------------+
| ID      | int(11)       | NO   |PRI|           |          |
| NAME    | varchar(20)   | NO   |     |           |          |
| AGE     | int(11)       | NO   |     |           |          |
| ADDRESS | char(25)      | YES  |     | NULL      |          |
| SALARY  | decimal(18,2) | YES  |     | NULL      |          |
+---------+---------------+------+-----+-----------------+ ----------+
5 rows in set (0.00 sec)
```

This means that the CUSTOMERS table is available in the database, so let us now drop it as shown below.

```
SQL> DROP TABLE CUSTOMERS;
Query OK, 0 rows affected (0.01 sec)
```

Now, if you would try the DESC command, then you will get the following error:

```
SQL> DESC CUSTOMERS;
ERROR 1146 (42S02): Table 'TEST.CUSTOMERS" doesn't exist
```

Here, TEST is the database name which we are using for our examples.

13. SQL – INSERT Query

The SQL **INSERT INTO** Statement is used to add new rows of data to a table in the database.

Syntax

There are two basic syntaxes of the INSERT INTO statement which are shown below.

```
INSERT INTO TABLE_NAME (column1, column2, column3,...columnN)]
VALUES (value1, value2, value3,...valueN);
```

Here, column1, column2, column3,...columnN are the names of the columns in the table into which you want to insert the data.

You may not need to specify the column(s) name in the SQL query if you are adding values for all the columns of the table. But make sure the order of the values is in the same order as the columns in the table.

The **SQL INSERT INTO** syntax will be as follows:

```
INSERT INTO TABLE_NAME VALUES (value1,value2,value3,...valueN);
```

Example

The following statements would create six records in the CUSTOMERS table.

```
INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)
VALUES (1, 'Ramesh', 32, 'Ahmedabad', 2000.00 );

INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)
VALUES (2, 'Khilan', 25, 'Delhi', 1500.00 );

INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)
VALUES (3, 'kaushik', 23, 'Kota', 2000.00 );

INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)
VALUES (4, 'Chaitali', 25, 'Mumbai', 6500.00 );

INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)
VALUES (5, 'Hardik', 27, 'Bhopal', 8500.00 );
```

```
INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)
VALUES (6, 'Komal', 22, 'MP', 4500.00 );
```

You can create a record in the CUSTOMERS table by using the second syntax as shown below.

```
INSERT INTO CUSTOMERS
VALUES (7, 'Muffy', 24, 'Indore', 10000.00 );
```

All the above statements would produce the following records in the CUSTOMERS table as shown below.

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

Populate one table using another table

You can populate the data into a table through the select statement over another table; provided the other table has a set of fields, which are required to populate the first table.

Here is the syntax:

```
INSERT INTO first_table_name [(column1, column2, ... columnN)]
   SELECT column1, column2, ...columnN
   FROM second_table_name
   [WHERE condition];
```

14. SQL – SELECT Query

The SQL **SELECT** statement is used to fetch the data from a database table which returns this data in the form of a result table. These result tables are called result-sets.

Syntax

The basic syntax of the SELECT statement is as follows.:

```
SELECT column1, column2, columnN FROM table_name;
```

Here, column1, column2... are the fields of a table whose values you want to fetch. If you want to fetch all the fields available in the field, then you can use the following syntax.

```
SELECT * FROM table_name;
```

Example

Consider the CUSTOMERS table having the following records:

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

The following code is an example, which would fetch the ID, Name and Salary fields of the customers available in CUSTOMERS table.

```
SQL> SELECT ID, NAME, SALARY FROM CUSTOMERS;
```

This would produce the following result:

```
+----+----------+----------+
| ID | NAME     | SALARY   |
+----+----------+----------+
|  1 | Ramesh   |  2000.00 |
|  2 | Khilan   |  1500.00 |
|  3 | kaushik  |  2000.00 |
|  4 | Chaitali |  6500.00 |
|  5 | Hardik   |  8500.00 |
|  6 | Komal    |  4500.00 |
|  7 | Muffy    | 10000.00 |
+----+----------+----------+
```

If you want to fetch all the fields of the CUSTOMERS table, then you should use the following query.

```
SQL> SELECT * FROM CUSTOMERS;
```

This would produce the result as shown below.

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

15. SQL – WHERE Clause

The SQL **WHERE** clause is used to specify a condition while fetching the data from a single table or by joining with multiple tables. If the given condition is satisfied, then only it returns a specific value from the table. You should use the WHERE clause to filter the records and fetching only the necessary records.

The WHERE clause is not only used in the SELECT statement, but it is also used in the UPDATE, DELETE statement, etc., which we would examine in the subsequent chapters.

Syntax

The basic syntax of the SELECT statement with the WHERE clause is as shown below.

```
SELECT column1, column2, columnN
FROM table_name
WHERE [condition]
```

You can specify a condition using the comparison or logical operators like **>**, **<**, **=**, **LIKE**, **NOT**, etc. The following examples would make this concept clear.

Example

Consider the CUSTOMERS table having the following records:

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

The following code is an example which would fetch the ID, Name and Salary fields from the CUSTOMERS table, where the salary is greater than 2000:

```
SQL> SELECT ID, NAME, SALARY
FROM CUSTOMERS
WHERE SALARY > 2000;
```

This would produce the following result:

```
+----+----------+----------+
| ID | NAME     | SALARY   |
+----+----------+----------+
|  4 | Chaitali |  6500.00 |
|  5 | Hardik   |  8500.00 |
|  6 | Komal    |  4500.00 |
|  7 | Muffy    | 10000.00 |
+----+----------+----------+
```

The following query is an example, which would fetch the ID, Name and Salary fields from the CUSTOMERS table for a customer with the name **Hardik**.

Here, it is important to note that all the strings should be given inside single quotes (''). Whereas, numeric values should be given without any quote as in the above example.

```
SQL> SELECT ID, NAME, SALARY
FROM CUSTOMERS
WHERE NAME = 'Hardik';
```

This would produce the following result:

```
+----+----------+----------+
| ID | NAME     | SALARY   |
+----+----------+----------+
|  5 | Hardik   |  8500.00 |
+----+----------+----------+
```

16. SQL – AND & OR Conjunctive Operators SQL

The SQL **AND** & **OR** operators are used to combine multiple conditions to narrow data in an SQL statement. These two operators are called as the conjunctive operators.

These operators provide a means to make multiple comparisons with different operators in the same SQL statement.

The AND Operator

The **AND** operator allows the existence of multiple conditions in an SQL statement's WHERE clause.

Syntax

The basic syntax of the AND operator with a WHERE clause is as follows:

```
SELECT column1, column2, columnN
FROM table_name
WHERE [condition1] AND [condition2]...AND [conditionN];
```

You can combine N number of conditions using the AND operator. For an action to be taken by the SQL statement, whether it be a transaction or a query, all conditions separated by the AND must be TRUE.

Example

Consider the CUSTOMERS table having the following records:

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

Following is an example, which would fetch the ID, Name and Salary fields from the CUSTOMERS table, where the salary is greater than 2000 and the age is less than 25 years.

```
SQL> SELECT ID, NAME, SALARY
FROM CUSTOMERS
WHERE SALARY > 2000 AND age < 25;
```

This would produce the following result:

```
+----+----------+----------+
| ID | NAME     | SALARY   |
+----+----------+----------+
|  6 | Komal    |  4500.00 |
|  7 | Muffy    | 10000.00 |
+----+----------+----------+
```

The OR Operator

The OR operator is used to combine multiple conditions in an SQL statement's WHERE clause.

Syntax

The basic syntax of the OR operator with a WHERE clause is as follows:

```
SELECT column1, column2, columnN
FROM table_name
WHERE [condition1] OR [condition2]...OR [conditionN]
```

You can combine N number of conditions using the OR operator. For an action to be taken by the SQL statement, whether it be a transaction or query, the only any ONE of the conditions separated by the OR must be TRUE.

Example

Consider the CUSTOMERS table having the following records:

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
```

```
|    5| Hardik    |    27| Bhopal    |   8500.00|
|    6| Komal     |    22| MP        |   4500.00|
|    7| Muffy     |    24| Indore    |  10000.00|
+-------+----------+-----+-----------+----------+
```

The following code block has a query, which would fetch the ID, Name and Salary fields from the CUSTOMERS table, where the salary is greater than 2000 OR the age is less than 25 years.

```
SQL> SELECT ID, NAME, SALARY
FROM CUSTOMERS
WHERE SALARY > 2000 OR age < 25;
```

This would produce the following result:

```
+-------+----------+----------+
| ID | NAME     | SALARY   |
+-------+----------+----------+
|    3| kaushik  |   2000.00|
|    4| Chaitali |   6500.00|
|    5| Hardik   |   8500.00|
|    6| Komal    |   4500.00|
|    7| Muffy    |  10000.00|
+-------+----------+----------+
```

17. SQL – UPDATE Query

The SQL **UPDATE** Query is used to modify the existing records in a table. You can use the WHERE clause with the UPDATE query to update the selected rows, otherwise all the rows would be affected.

Syntax

The basic syntax of the UPDATE query with a WHERE clause is as follows:

```
UPDATE table_name
SET column1 = value1, column2 = value2...., columnN = valueN
WHERE [condition];
```

You can combine N number of conditions using the AND or the OR operators.

Example

Consider the CUSTOMERS table having the following records:

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

The following query will update the ADDRESS for a customer whose ID number is 6 in the table.

```
SQL> UPDATE CUSTOMERS
SET ADDRESS = 'Pune'
WHERE ID = 6;
```

Now, the CUSTOMERS table would have the following records:

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | Pune      |  4500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

If you want to modify all the ADDRESS and the SALARY column values in the CUSTOMERS table, you do not need to use the WHERE clause as the UPDATE query would be enough as shown in the following code block.

```
SQL> UPDATE CUSTOMERS
SET ADDRESS = "Pune", SALARY = 1000.00;
```

Now, CUSTOMERS table would have the following records:

```
+----+----------+-----+---------+----------+
| ID | NAME     | AGE | ADDRESS | SALARY   |
+----+----------+-----+---------+----------+
|  1 | Ramesh   |  32 | Pune    |  1000.00 |
|  2 | Khilan   |  25 | Pune    |  1000.00 |
|  3 | kaushik  |  23 | Pune    |  1000.00 |
|  4 | Chaitali |  25 | Pune    |  1000.00 |
|  5 | Hardik   |  27 | Pune    |  1000.00 |
|  6 | Komal    |  22 | Pune    |  1000.00 |
|  7 | Muffy    |  24 | Pune    |  1000.00 |
+----+----------+-----+---------+----------+
```

18. SQL – DELETE Query

The SQL **DELETE** Query is used to delete the existing records from a table.

You can use the WHERE clause with a DELETE query to delete the selected rows, otherwise all the records would be deleted.

Syntax

The basic syntax of the DELETE query with the WHERE clause is as follows:

```
DELETE FROM table_name
WHERE [condition];
```

You can combine N number of conditions using AND or OR operators.

Example

Consider the CUSTOMERS table having the following records:

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

The following code has a query, which will DELETE a customer, whose ID is 6.

```
SQL> DELETE FROM CUSTOMERS
WHERE ID = 6;
```

Now, the CUSTOMERS table would have the following records.

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

If you want to DELETE all the records from the CUSTOMERS table, you do not need to use the WHERE clause and the DELETE query would be as follows:

```
SQL> DELETE FROM CUSTOMERS;
```

Now, the CUSTOMERS table would not have any record.

19. SQL – LIKE Clause

The SQL **LIKE** clause is used to compare a value to similar values using wildcard operators. There are two wildcards used in conjunction with the LIKE operator.

- The percent sign (%)
- The underscore (_)

The percent sign represents zero, one or multiple characters. The underscore represents a single number or character. These symbols can be used in combinations.

Syntax

The basic syntax of % and _ is as follows:

```
SELECT FROM table_name
WHERE column LIKE 'XXXX%'

or

SELECT FROM table_name
WHERE column LIKE '%XXXX%'

or

SELECT FROM table_name
WHERE column LIKE 'XXXX_'

or

SELECT FROM table_name
WHERE column LIKE '_XXXX'

or

SELECT FROM table_name
WHERE column LIKE '_XXXX_'
```

You can combine N number of conditions using AND or OR operators. Here, XXXX could be any numeric or string value.

Example

The following table has a few examples showing the WHERE part having different LIKE clause with **'%'** and **'_'** operators:

Statement	Description
WHERE SALARY LIKE '200%'	Finds any values that start with 200.
WHERE SALARY LIKE '%200%'	Finds any values that have 200 in any position.
WHERE SALARY LIKE '_00%'	Finds any values that have 00 in the second and third positions.
WHERE SALARY LIKE '2_%_%'	Finds any values that start with 2 and are at least 3 characters in length.
WHERE SALARY LIKE '%2'	Finds any values that end with 2.
WHERE SALARY LIKE '_2%3'	Finds any values that have a 2 in the second position and end with a 3.
WHERE SALARY LIKE '2___3'	Finds any values in a five-digit number that start with 2 and end with 3.

Let us take a real example, consider the CUSTOMERS table having the records as shown below.

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00

Following is an example, which would display all the records from the CUSTOMERS table, where the SALARY starts with 200.

```
SQL> SELECT * FROM CUSTOMERS
WHERE SALARY LIKE '200%';
```

This would produce the following result:

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad | 2000.00  |
|  3 | kaushik  |  23 | Kota      | 2000.00  |
+----+----------+-----+-----------+----------+
```