Shiksha Mandal's

G. S. College of Commerce, Wardha

DEPARTMENT OF B. COM. COMPUTER APPLICATION

BCCA Part-II SEM-III

Database Management System

Unit –III

By

Prof. Harsha N. Gangavane

Dr. Revati Bangre
Co-ordinator

Dr. Anil Ramteke
Principal (Officiating)

Unit -III

What is SQL?

SQL stands for Structured Query Language. SQL is a database computer language designed for the retrieval and management of data in a relational database. SQL is a standard language for storing, manipulating and retrieving data in databases.

What Can SQL do?

- SQL can execute queries against a database
- SQL can retrieve data from a database
- SQL can insert records in a database
- SQL can update records in a database
- SQL can delete records from a database
- SQL can create new databases
- SQL can create new tables in a database
- SQL can create stored procedures in a database
- SQL can create views in a database
- SQL can set permissions on tables, procedures, and views

Database Tables

A database most often contains one or more tables. Each table is identified by a name (e. g. "Customers" or "Orders"). Tables contain records (rows) with data.

Semicolon after SQL Statements?

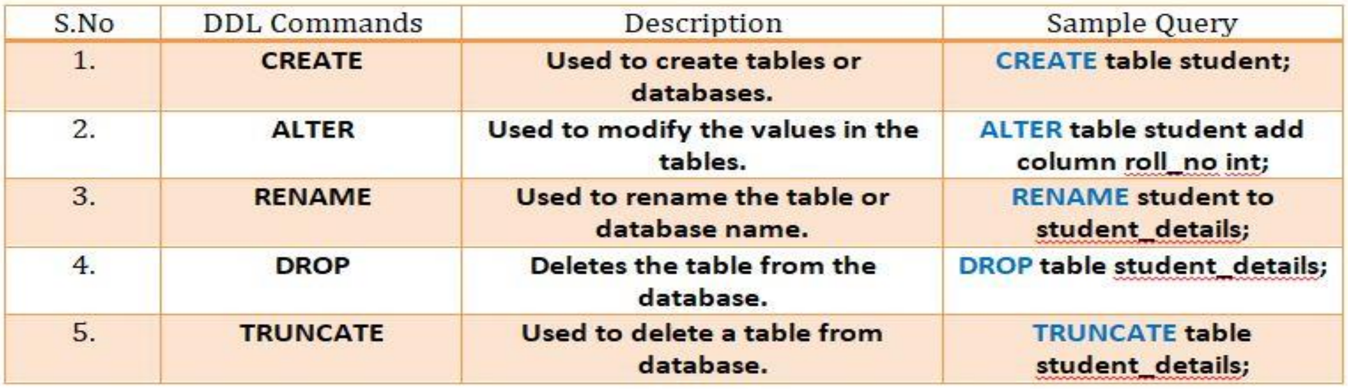
- Some database systems require a semicolon at the end of each SQL statement.
- Semicolon is the standard way to separate each SQL statement in database systems that allow more than one SQL statement to be executed in the same call to the server.

There are five types of SQL Commands which can be classified as:

- DDL (Data Definition Language).
- DML (Data Manipulation Language).
- DQL (Data Query Language).
- DCL (Data Control Language).
- TCL (Transaction Control Language).

Types of SQL Commands: Data Definition Language (DDL)

- In order to make/perform changes on the physical structure of any table residing inside a database, DDL is used. These commands when executed are auto commit in nature and all the changes in the table are reflected and saved immediately. DDL commands includes :

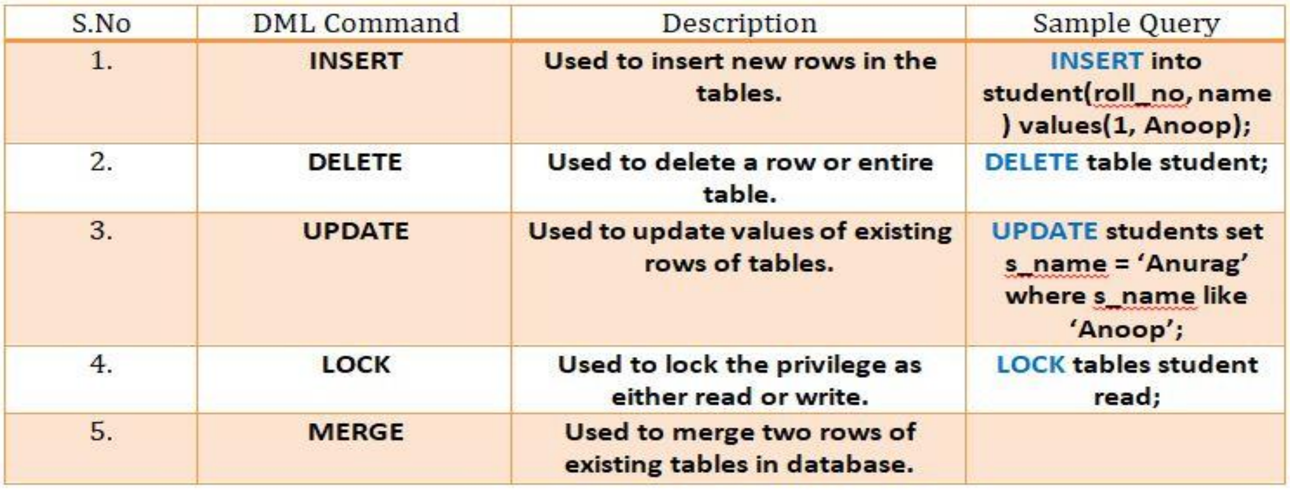


S.No	DDL Commands	Description	Sample Query
1.	CREATE	Used to create tables or databases.	CREATE table student;
2.	ALTER	Used to modify the values in the tables.	ALTER table student add column roll_no int;
3.	RENAME	Used to rename the table or database name.	RENAME student to student_details;
4.	DROP	Deletes the table from the database.	DROP table student_details;
5.	TRUNCATE	Used to delete a table from database.	TRUNCATE table student_details;

SQL DDL Commands

Types Of SQL Commands : Data Manipulation Language(DML)

- Once the tables are created and database is generated using DDL commands, manipulation inside those tables and databases is done using DML commands. The advantage of using DML commands is, if in case any wrong changes or values are made, they can be changes and rolled back easily. DML commands includes :

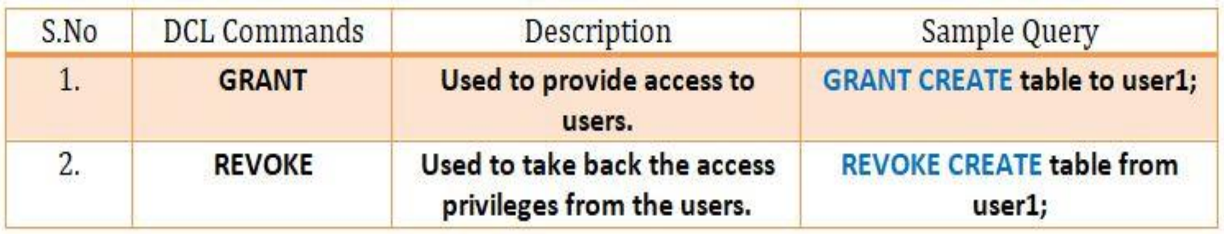


S.No	DML Command	Description	Sample Query
1.	INSERT	Used to insert new rows in the tables.	INSERT into student(roll_no, name) values(1, Anoop);
2.	DELETE	Used to delete a row or entire table.	DELETE table student;
3.	UPDATE	Used to update values of existing rows of tables.	UPDATE students set s_name = 'Anurag' where s_name like 'Anoop';
4.	LOCK	Used to lock the privilege as either read or write.	LOCK tables student read;
5.	MERGE	Used to merge two rows of existing tables in database.	

SQL DML Commands

Types Of SQL Commands : Data Control Language(DCL)

- DCL commands as the name suggests manages the matters and issues related to the data control in any database. TCL commands mainly provides special privilege access to users and is also used to specify the roles of users accordingly. There are two commonly used DCL commands, these are:

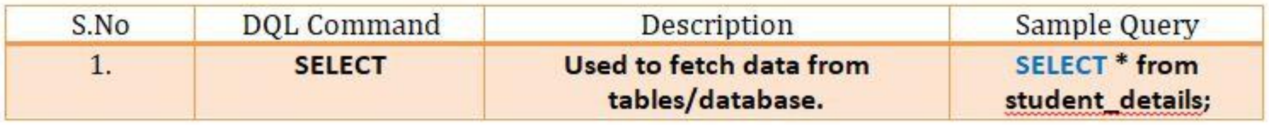


S.No	DCL Commands	Description	Sample Query
1.	GRANT	Used to provide access to users.	GRANT CREATE table to user1;
2.	REVOKE	Used to take back the access privileges from the users.	REVOKE CREATE table from user1;

SQL DCL Commands

Types Of SQL Commands : Data Query Language(DQL)

- Data query language consists of only one command over which data selection in SQL relies. SELECT command in combination with other SQL clauses is used to retrieve and fetch data from database/tables on the basis of certain conditions applied by user.

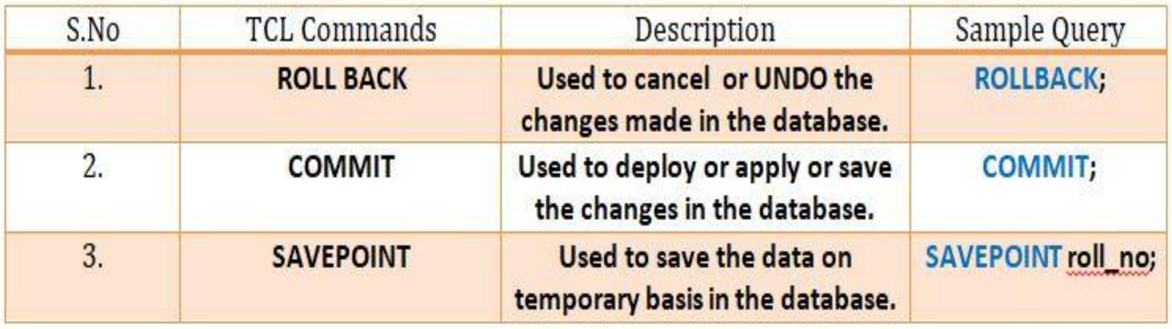


S.No	DQL Command	Description	Sample Query
1.	SELECT	Used to fetch data from tables/database.	SELECT * from student_details;

SQL DQL Commands

Types Of SQL Commands : Transaction Control Language(TCL)

- Transaction Control Language as the name suggests manages the issues and matters related to the transactions in any database. They are used to rollback or commit the changes in the database.
- Roll back means "Undo" the changes and Commit means "Applying" the changes. There are three major TCL commands.



S.No	TCL Commands	Description	Sample Query
1.	ROLL BACK	Used to cancel or UNDO the changes made in the database.	ROLLBACK;
2.	COMMIT	Used to deploy or apply or save the changes in the database.	COMMIT;
3.	SAVEPOINT	Used to save the data on temporary basis in the database.	SAVEPOINT roll_no;

SQL TCL Commands

Some of The Most Important SQL Commands

- **SELECT** - extracts data from a database
- **UPDATE** - updates data in a database
- **DELETE** - deletes data from a database
- **INSERT INTO** - inserts new data into a database

- **CREATE DATABASE** - creates a new database
- **ALTER DATABASE** - modifies a database
- **CREATE TABLE** - creates a new table
- **ALTER TABLE** - modifies a table
- **DROP TABLE** - deletes a table
- **CREATE INDEX** - creates an index (search key)
- **DROP INDEX** - deletes an index

- **What is Data?**

In simple words data can be facts related to any object in consideration.For example your name, age, height, weight, etc are some data related to you.A picture , image , file , pdf etc can also be considered data.

- **What is a Database?**

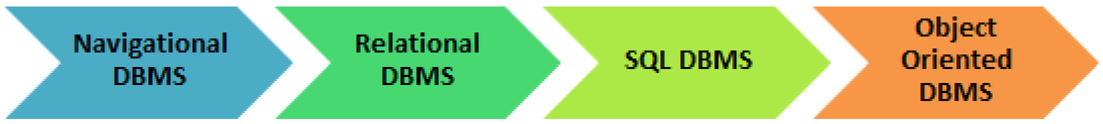
Database is a systematic collection of data. Databases support storage and manipulation of data. Databases make data management easy. Let's discuss few examples. An online telephone directory would definitely use database to store data pertaining to people, phone numbers, other contact details, etc. Your electricity service provider is obviously using a database to manage billing , client related issues, to handle fault data, etc. Let's also consider the facebook. It needs to store, manipulate and present data related to members, their friends, member activities, messages, advertisements and lot more. We can provide countless number of examples for usage of databases.

- **What is a Database Management System (DBMS)?**

Database Management System (DBMS) is a collection of programs which enables its users to access database, manipulate data, reporting / representation of data . It also helps to control access to the database. Database Management Systems are not a new concept and as such had been first implemented in 1960s. Charles Bachmen's Integrated Data Store (IDS) is said to be the first DBMS in history. With time database technologies evolved a lot while usage and expected functionalities of databases have been increased immensely.

- **Types of DBMS**

Let's see how the DBMS family got evolved with the time. Following diagram shows the evolution of DBMS categories.



There are 4 major types of DBMS.

- **Hierarchical** - this type of DBMS employs the "parent-child" relationship of storing data. This type of DBMS is rarely used nowadays. Its structure is like a tree with nodes representing records and branches representing fields. The windows registry used in Windows XP is an example of a hierarchical database. Configuration settings are stored as tree structures with nodes.

- **Network DBMS** - this type of DBMS supports many-to many relations. This usually results in complex database structures. RDM Server is an example of a database management system that implements the network model.
- **Relational DBMS** - this type of DBMS defines database relationships in form of tables, also known as relations. Unlike network DBMS, RDBMS does not support many to many relationships. Relational DBMS usually have pre-defined data types that they can support. This is the most popular DBMS type in the market. Examples of relational database management systems include MySQL, Oracle, and Microsoft SQL Server database.
- **Object Oriented Relation DBMS** - this type supports storage of new data types. The data to be stored is in form of objects. The objects to be stored in the database have attributes (i.e. gender, ager) and methods that define what to do with the data. PostgreSQL is an example of an object oriented relational DBMS.

- As the name suggests, DESCRIBE is used to describe something. Since in database we have tables, that's why we use **DESCRIBE** or **DESC**(both are same) command to describe the **structure** of a table.
  Syntax:

```
DESCRIBE one;
  OR
DESC one;
```

Note : We can use either **DESCRIBE** or **DESC**(both are **Case Insensitive**).
Suppose our table whose name is **one** has **3** columns named **FIRST_NAME**, **LAST_NAME**and **SALARY** and all are of can **contain** null values.

Output:

```
Name        Null      Type
FIRST_NAME              CHAR(25)
LAST_NAME               CHAR(25)
SALARY                NUMBER(6)
```

- Here, above on using **DESC** or either **DESCRIBE** we are able to see the **structure** of a table but **not** on the console tab, the structure of table is shown in the **describe tab** of the Database System Software.
- So **desc** or **describe** command shows the **structure** of table which include **name** of the column, **data-type** of column and the **nullability** which means, that column can contain null values or not.
- All of these features of table are described at the time of **Creation** of table.

Oracle String data types

CHAR(size)	It is used to store character data within the predefined length. It can be stored up to 2000 bytes.
NCHAR(size)	It is used to store national character data within the predefined length. It can be stored up to 2000 bytes.
VARCHAR2(size)	It is used to store variable string data within the predefined length. It can be stored up to 4000 byte.
VARCHAR(SIZE)	It is the same as VARCHAR2(size). You can also use VARCHAR(size), but it is suggested to use VARCHAR2(size)
NVARCHAR2(size)	It is used to store Unicode string data within the predefined length. We have to must specify the size of NVARCHAR2 data type. It can be stored up to 4000 bytes.

Oracle Numeric Data Types

NUMBER(p, s)	It contains precision p and scale s. The precision p can range from 1 to 38, and the scale s can range from -84 to 127.
FLOAT(p)	It is a subtype of the NUMBER data type. The precision p can range from 1 to 126.
BINARY_FLOAT	It is used for binary precision(32-bit). It requires 5 bytes, including length byte.
BINARY_DOUBLE	It is used for double binary precision (64-bit). It requires 9 bytes, including length byte.

Oracle Date and Time Data Types

DATE	It is used to store a valid date-time format with a fixed length. Its range varies from January 1, 4712 BC to December 31, 9999 AD.
TIMESTAMP	It is used to store the valid date in YYYY-MM-DD with time hh:mm:ss format.

Oracle Large Object Data Types (LOB Types)

BLOB	It is used to specify unstructured binary data. Its range goes up to 2^{32}-1 bytes or 4 GB.
BFILE	It is used to store binary data in an external file. Its range goes up to 2^{32}-1 bytes or 4 GB.
CLOB	It is used for single-byte character data. Its range goes up to 2^{32}-1 bytes or 4 GB.
NCLOB	It is used to specify single byte or fixed length multibyte national character set (NCHAR) data. Its range is up to 2^{32}-1 bytes or 4 GB.
RAW(size)	It is used to specify variable length raw binary data. Its range is up to 2000 bytes per row. Its maximum size must be specified.
LONG RAW	It is used to specify variable length raw binary data. Its range up to 2^{31}-1 bytes or 2 GB, per row.

Creating table or defining the structure of a table

Syntax:-

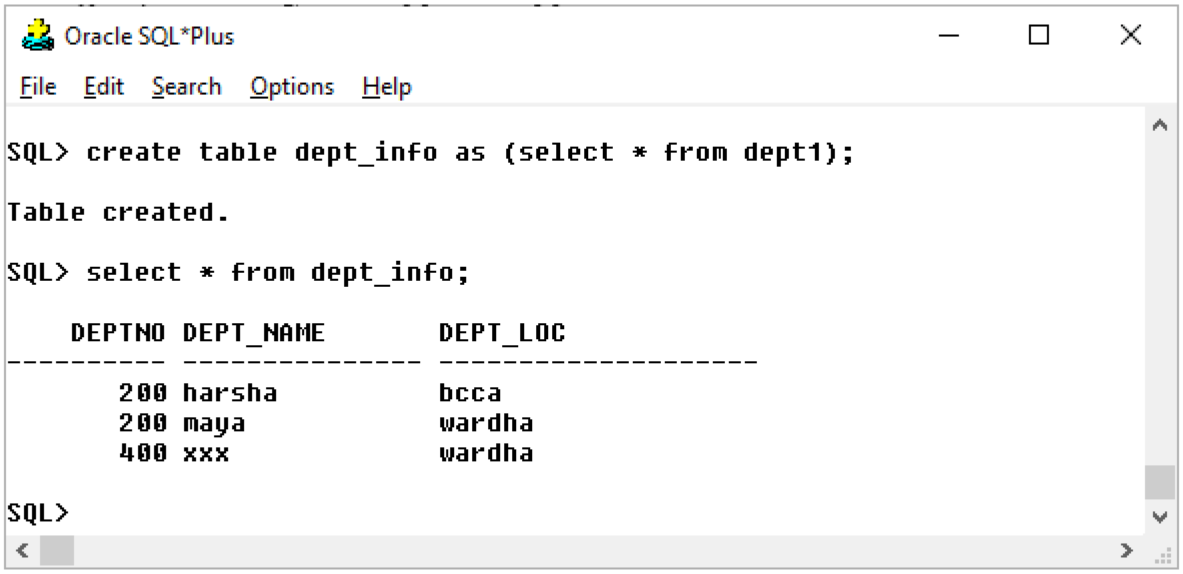
```
CREATE TABLE table_name
(
   column1 datatype,
   column2 datatype,
   column3 datatype,
  ....
);
```

Creating table or defining the structure of a table

```
create table one
(
id int not null,
name char(25)
)
```

Creating a table from a table

Syntax: - **Create table newtablename as (select Column1, column2, column3….,columnN);**

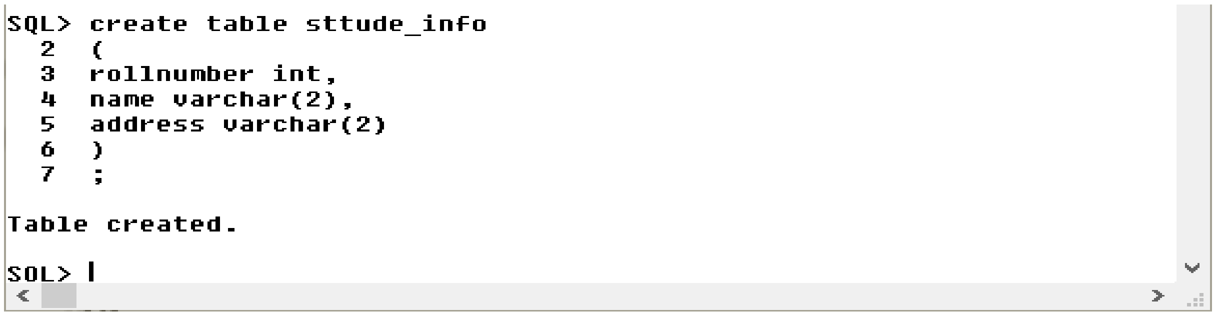


```
SQL> create table dept_info as (select * from dept1);

Table created.

SQL> select * from dept_info;

    DEPTNO DEPT_NAME       DEPT_LOC
---------- --------------- --------------------
       200 harsha          bcca
       200 maya            wardha
       400 xxx             wardha

SQL>
```

Here, we created a table whose name is **one** and its columns are **ID**, **NAME** and the **id** is of **not null** type i.e, we **can't** put null values in the **ID** column but we **can** put null values in the **NAME** column.

Example to demonstrate DESC:

Step 1: Defining structure of table i.e, Creating a table:



```
SQL> create table sttude_info
  2  (
  3  rollnumber int,
  4  name varchar(2),
  5  address varchar(2)
  6  )
  7  ;

Table created.

SQL>
```

- **ALTER TABLE**

This command is used to add, modify, drop the desired table columns in the table. It is basically used to make any changes in desired table structure.

- The basic syntax of an ALTER TABLE command to add a **New Column** in an existing table is as follows.

```
ALTER TABLE table_name ADD column_name datatype;
```

- The basic syntax of an ALTER TABLE command to **DROP COLUMN** in an existing table is as follows.

`

```
ALTER TABLE table_name DROP COLUMN column_name;
```

- The basic syntax of an ALTER TABLE command to change the **DATA TYPE** of a column in a table is as follows.

```
ALTER TABLE table_name MODIFY  column_name datatype;
```

- The basic syntax of an ALTER TABLE command to add a **NOT NULL** constraint to a column in a table is as follows.

```
ALTER TABLE table_name MODIFY column_name datatype NOT NULL;
```

- The basic syntax of ALTER TABLE to **ADD UNIQUE CONSTRAINT** to a table is as follows.

```
ALTER TABLE table_name
ADD CONSTRAINT MyUniqueConstraint UNIQUE(column1, column2...);
```

- The basic syntax of an ALTER TABLE command to **ADD CHECK CONSTRAINT** to a table is as follows.

```
ALTER TABLE table_name
ADD CONSTRAINT MyUniqueConstraint CHECK (CONDITION);
```

- The basic syntax of an ALTER TABLE command to **ADD PRIMARY KEY** constraint to a table is as follows.

```
ALTER TABLE table_name
ADD CONSTRAINT MyPrimaryKey PRIMARY KEY (column1, column2...);
```

- The basic syntax of an ALTER TABLE command to **DROP CONSTRAINT** from a table is as follows.

```
ALTER TABLE table_name
DROP CONSTRAINT MyUniqueConstraint;
```

If you're using MySQL, the code is as follows −

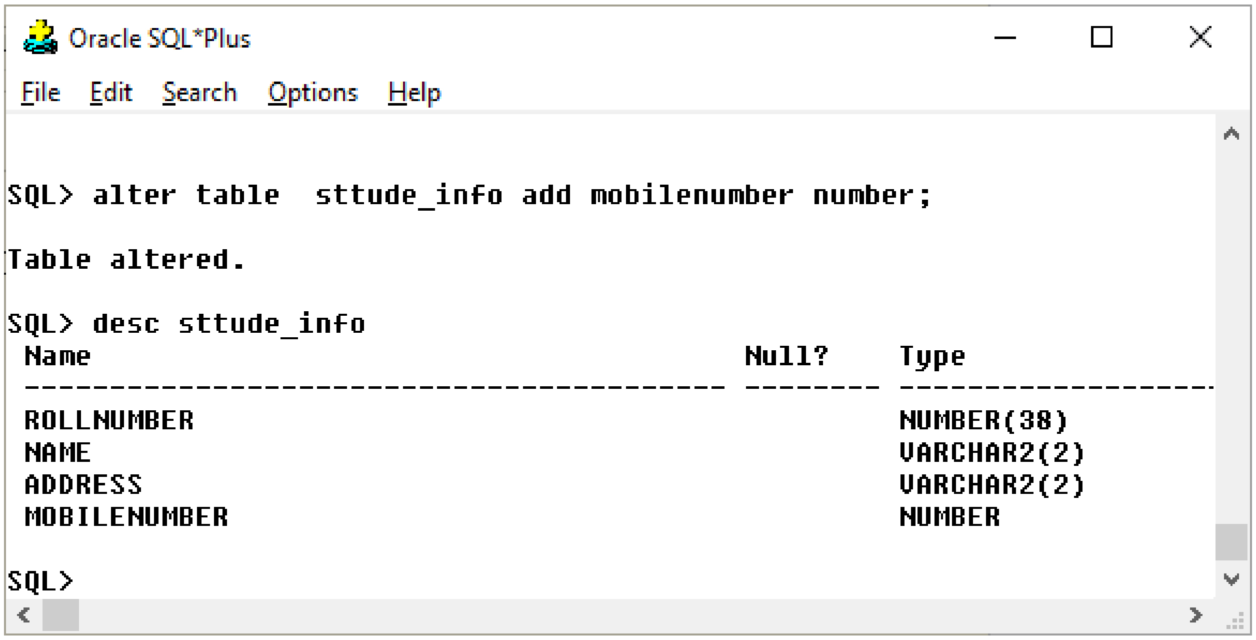
```
ALTER TABLE table_name
DROP INDEX MyUniqueConstraint;
```

The basic syntax of an ALTER TABLE command to **DROP PRIMARY KEY** constraint from a table is as follows.

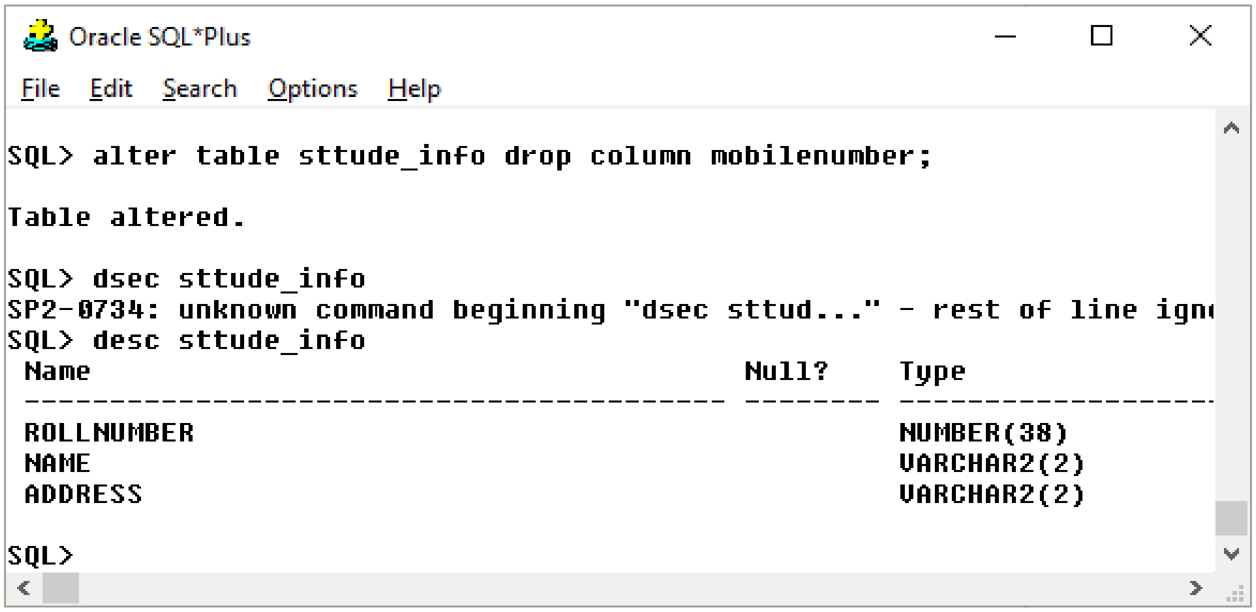
```
ALTER TABLE table_name
DROP CONSTRAINT MyPrimaryKey;
```

If you're using MySQL, the code is as follows −

```
ALTER TABLE table_name
DROP PRIMARY KEY;
```



```
SQL> alter table  sttude_info add mobilenumber number;

Table altered.

SQL> desc sttude_info
 Name                                      Null?    Type
 ----------------------------------------- -------- --------------------
 ROLLNUMBER                                         NUMBER(38)
 NAME                                               VARCHAR2(2)
 ADDRESS                                            VARCHAR2(2)
 MOBILENUMBER                                       NUMBER

SQL>
```



```
SQL> alter table sttude_info drop column mobilenumber;

Table altered.

SQL> dsec sttude_info
SP2-0734: unknown command beginning "dsec stud..." - rest of line ign
SQL> desc sttude_info
 Name                                      Null?    Type
 ----------------------------------------- -------- --------------------
 ROLLNUMBER                                         NUMBER(38)
 NAME                                               VARCHAR2(2)
 ADDRESS                                            VARCHAR2(2)

SQL>
```

SQL INSERT INTO

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

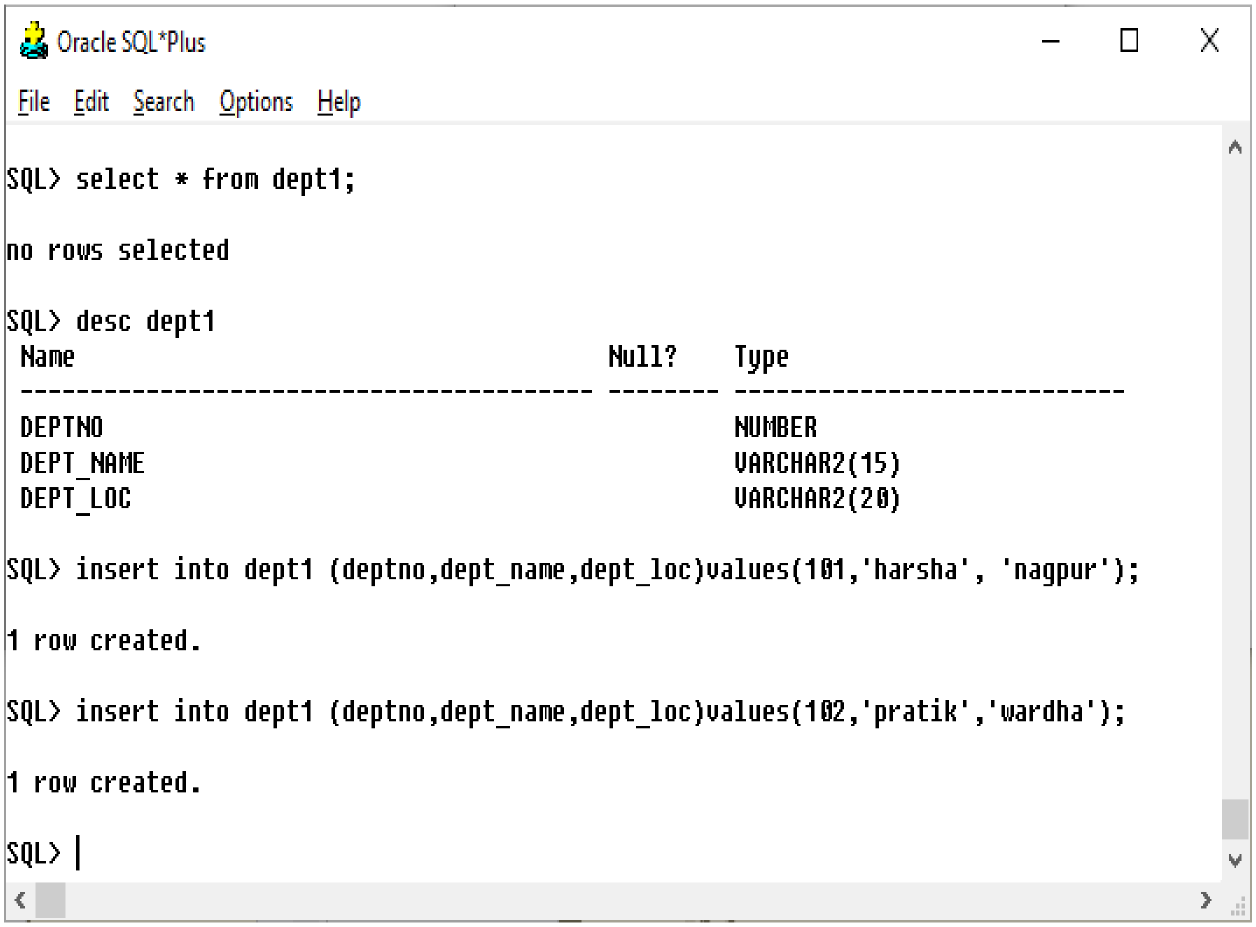
Syntax

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);

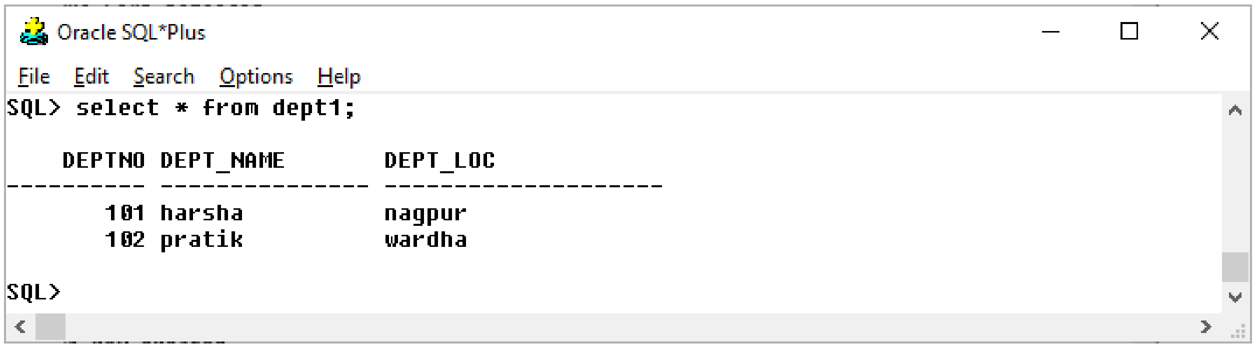
Here, column1, column2, column3,...columnN are the names of the columns in the table into which you want to insert the data.

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.



```
SQL> select * from dept1;

no rows selected

SQL> desc dept1
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 DEPTNO                                             NUMBER
 DEPT_NAME                                          VARCHAR2(15)
 DEPT_LOC                                           VARCHAR2(20)

SQL> insert into dept1 (deptno,dept_name,dept_loc)values(101,'harsha', 'nagpur');

1 row created.

SQL> insert into dept1 (deptno,dept_name,dept_loc)values(102,'pratik','wardha');

1 row created.

SQL> |
```

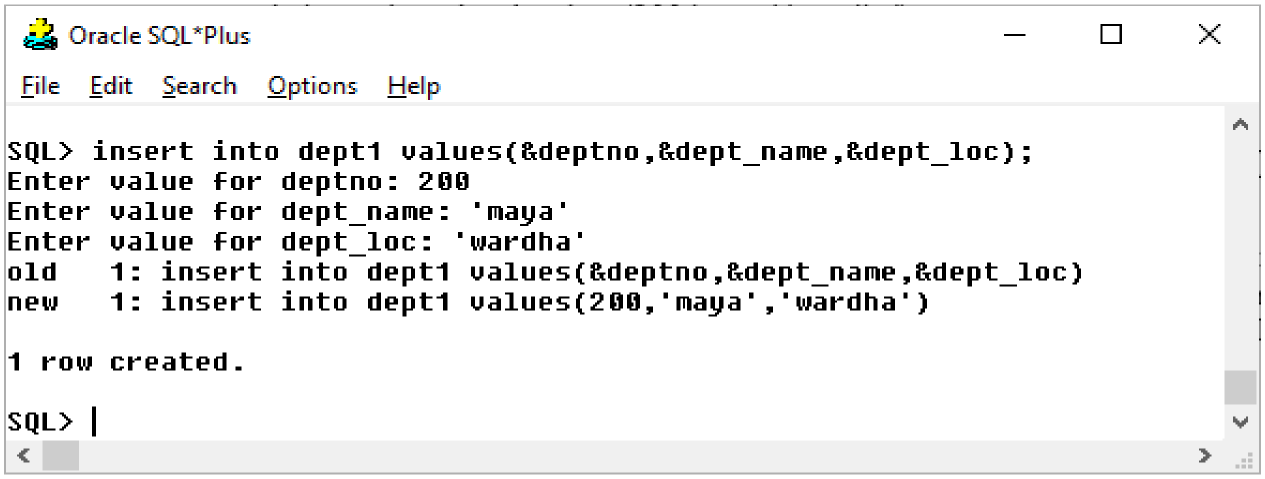
After Inserting



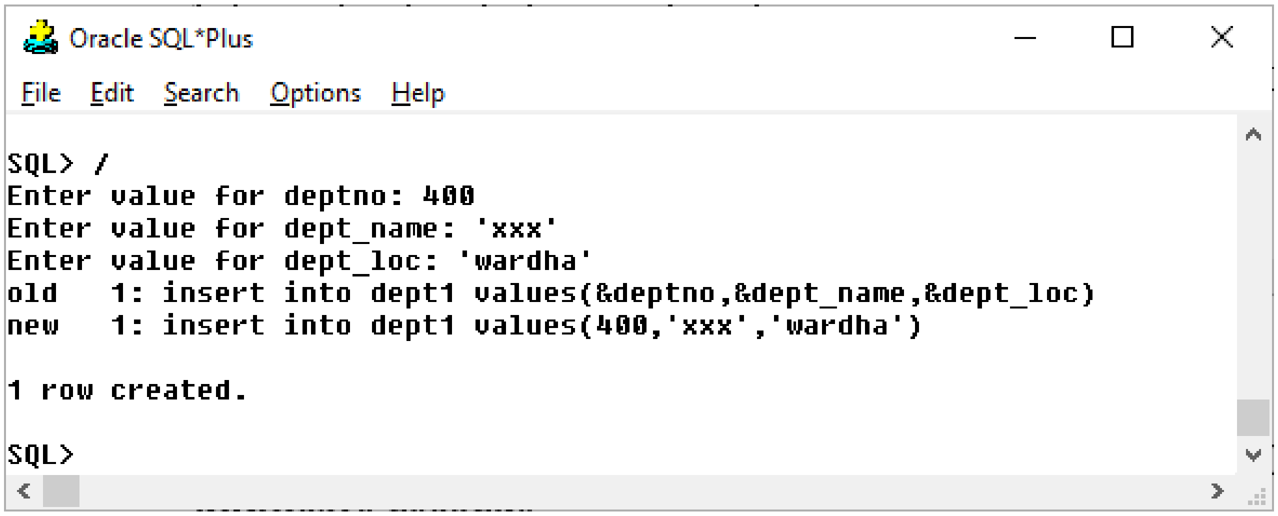
```
SQL> select * from dept1;

    DEPTNO DEPT_NAME       DEPT_LOC
---------- --------------- --------------------
       101 harsha          nagpur
       102 pratik          wardha

SQL>
```

Inserting multiple rows in one insert command

Syntax: - Insert into Table-Name values (&column1, &column2, &column3….,columnN);



```
SQL> insert into dept1 values(&deptno,&dept_name,&dept_loc);
Enter value for deptno: 200
Enter value for dept_name: 'maya'
Enter value for dept_loc: 'wardha'
old   1: insert into dept1 values(&deptno,&dept_name,&dept_loc)
new   1: insert into dept1 values(200,'maya','wardha')

1 row created.

SQL> |
```

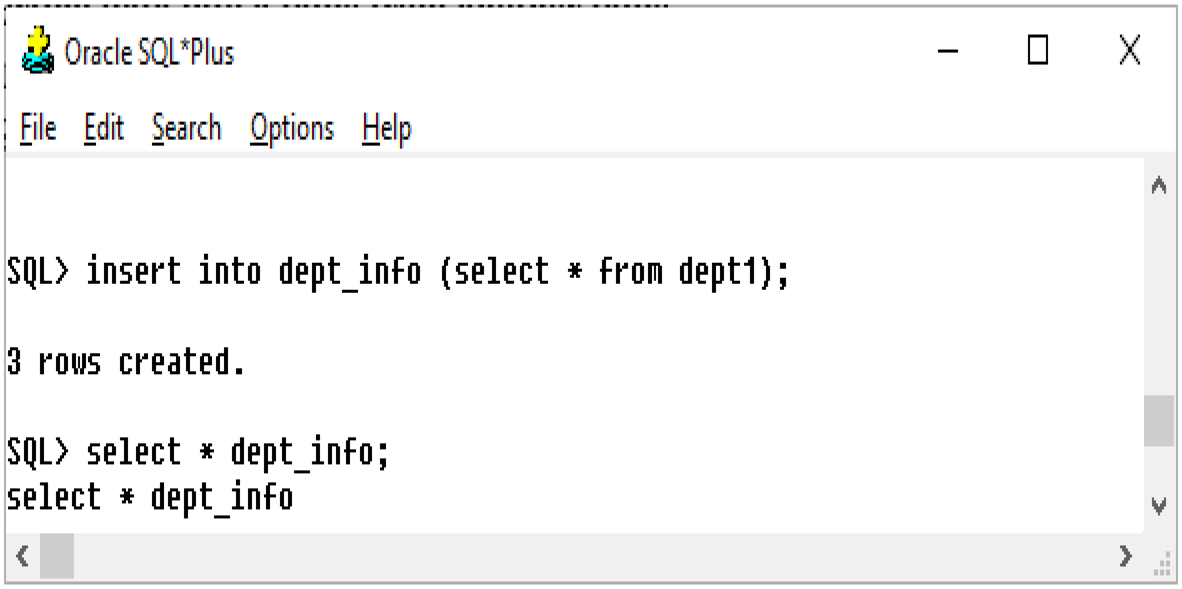


```
SQL> /
Enter value for deptno: 400
Enter value for dept_name: 'xxx'
Enter value for dept_loc: 'wardha'
old   1: insert into dept1 values(&deptno,&dept_name,&dept_loc)
new   1: insert into dept1 values(400,'xxx','wardha')

1 row created.

SQL>
```

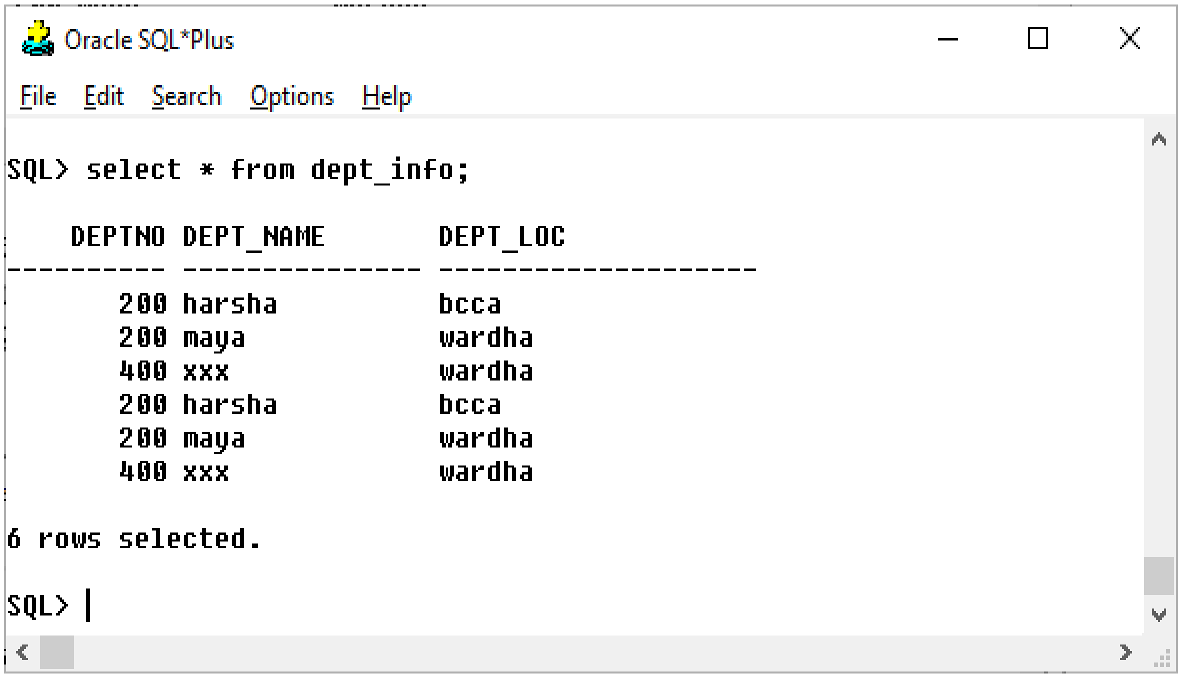
- **Inserting data into a table from another table,**

Syntax: Insert into insert-table-name
(Select * from Table-name)

Example:-



```
SQL> insert into dept_info (select * from dept1);

3 rows created.

SQL> select * dept_info;
select * dept_info
```



```
SQL> select * from dept_info;

    DEPTNO DEPT_NAME       DEPT_LOC
---------- --------------- --------------------
       200 harsha          bcca
       200 maya            wardha
       400 xxx             wardha
       200 harsha          bcca
       200 maya            wardha
       400 xxx             wardha

6 rows selected.

SQL> 
```

- **DISTINCT or DISTINCTROW identifier to eliminate duplicate records.**

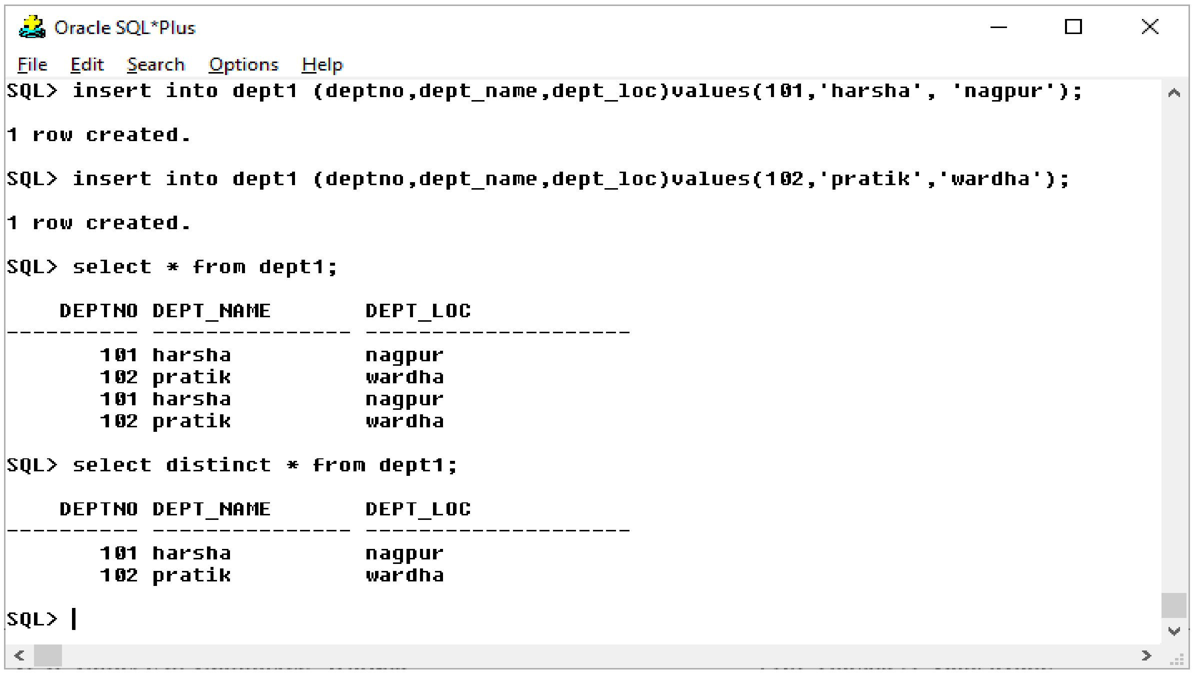
When the result set from a SELECT statement contains duplicate rows, you may want to remove them and keep every row data to be unique for a column or combination of columns. You can use the DISTINCT or DISTINCTROW identifier to eliminate duplicate records. DISTINCT and DISTINCTROW are synonyms and specify removal of duplicate rows from the result set.

- **The SQL SELECT DISTINCT Statement**

The SELECT DISTINCT statement is used to return only distinct (different) values.

Inside a table, a column often contains many duplicate values; and sometimes you only want to list the different (distinct) values.

SELECT DISTINCT Syntax **SELECT DISTINCT *column1*, *column2*, ….. FROM *table_name*;**



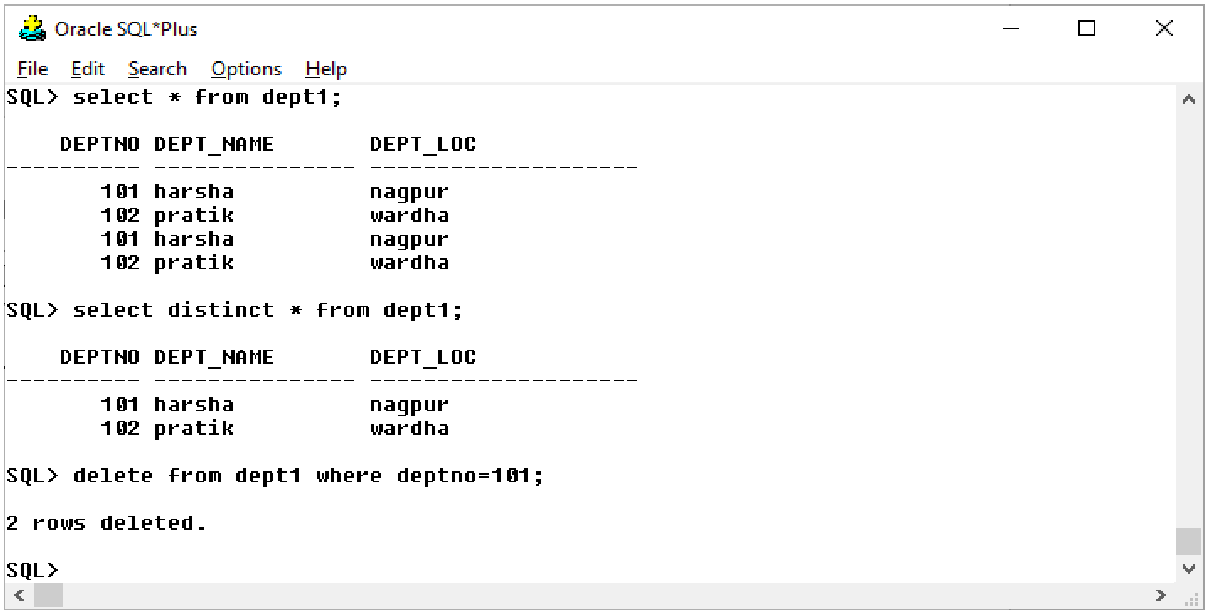
```
SQL> insert into dept1 (deptno,dept_name,dept_loc)values(101,'harsha', 'nagpur');

1 row created.

SQL> insert into dept1 (deptno,dept_name,dept_loc)values(102,'pratik','wardha');

1 row created.

SQL> select * from dept1;

    DEPTNO DEPT_NAME       DEPT_LOC
---------- --------------- --------------------
       101 harsha          nagpur
       102 pratik          wardha
       101 harsha          nagpur
       102 pratik          wardha

SQL> select distinct * from dept1;

    DEPTNO DEPT_NAME       DEPT_LOC
---------- --------------- --------------------
       101 harsha          nagpur
       102 pratik          wardha

SQL> |
```

- **The SQL DELETE Statement**

The DELETE statement is used to delete existing records in a table.

Syntax: -

DELETE FROM *table-name* WHERE *condition*;



```
SQL> select * from dept1;

    DEPTNO DEPT_NAME       DEPT_LOC
---------- --------------- --------------------
       101 harsha          nagpur
       102 pratik          wardha
       101 harsha          nagpur
       102 pratik          wardha

SQL> select distinct * from dept1;

    DEPTNO DEPT_NAME       DEPT_LOC
---------- --------------- --------------------
       101 harsha          nagpur
       102 pratik          wardha

SQL> delete from dept1 where deptno=101;

2 rows deleted.

SQL>
```

Example: -

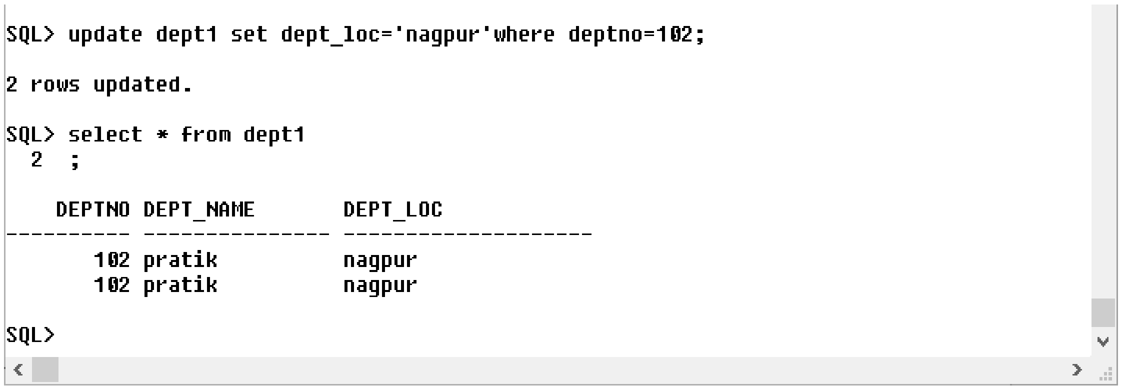
delete from dept1 where deptno=101;

- ## The SQL UPDATE Statement

The UPDATE statement is used to modify the existing records in a table.

Syntax

UPDATE *table_name*
SET *column1* = *value1*, *column2* = *value2*, ...
WHERE *condition*;



```
SQL> update dept1 set dept_loc='nagpur'where deptno=102;

2 rows updated.

SQL> select * from dept1
  2  ;

    DEPTNO DEPT_NAME       DEPT_LOC
---------- --------------- --------------------
       102 pratik          nagpur
       102 pratik          nagpur

SQL>
```

- **What you can do with SELECT statement**

With a SELECT statement, you can retrieve information from a database table in the format you want. Basically, you can control what rows and columns for the data retrieval.

For rows, you can:

1. Limit the number of rows returned in the result set.
2. Retrieve the data that meets the criteria you have specified.
3. Retrieve distinctive data.
4. Sort the table data in the result set in descending or ascending order.
5. Group the table data in the result set in a way that meets your application requirements.
6. Summarize the table data in the result set to get statistical information.
7. Retrieve data from two or more tables for related information.

For columns, you can:

1. Specify which column(s) to retrieve.
2. Rename a table column to a different name.
3. Perform arithmetic operations or conversion operations on a column data.
4. Retrieve data in a column based on conditions in one or more other columns in the same table.
5. Retrieve data in a column based on conditions in one or more other columns in another table.

Using a simple SELECT statement

Here is what a simple SELECT statement looks like:

```
SELECT column1, column2, column3... FROM tablename
```

1. SELECT clause identifies which column's data to retrieve in the table.
2. FROM clause identifies which table to retrieve data from.

Note that SELECT and FROM can be called keyword or clause. In other words, keyword and clause can be used interchangeably. Putting two or more clauses together in a SQL forms a SQL statement.

select all columns, all rows

```
SELECT *
FROM categories;
```

```
SELECT CategoryID, CategoryName, Description, Picture
FROM categories;
SELECT CategoryID, CategoryName, Description, Picture
FROM `categories`;
```

Four select statements are listed above. They retrieve all rows and all columns from categories table. Asterisk (*) means all columns. We can also explicitly list all column names (CategoryID, CategoryName, Description, Picture) in the statement. It's equivalent to using Asterisk. Back quotes are used to enclose a table name. It's not necessary to use back quotes around a table name unless the table name has space(s) in it.

- ## 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

```
SQL> SELECT ID, NAME, SALARY

FROM CUSTOMERS

WHERE SALARY > 2000;
```

- **What is an Operator in SQL?**

An operator is a reserved word or a character used primarily in an SQL statement's WHERE clause to perform operation(s), such as comparisons and arithmetic operations. These Operators are used to specify conditions in an SQL statement and to serve as conjunctions for multiple conditions in a statement.

- Arithmetic operators
- Comparison operators
- Logical operators
- Operators used to negate conditions

SQL Arithmetic Operators

Assume **'variable a'** holds 10 and **'variable b'** holds 20, then −

Show Examples

Operator	Description	Example
+ (Addition)	Adds values on either side of the operator.	a + b will give 30
- (Subtraction)	Subtracts right hand operand from left hand operand.	a - b will give -10
* (Multiplication)	Multiplies values on either side of the operator.	a * b will give 200
/ (Division)	Divides left hand operand by right hand operand.	b / a will give 2
% (Modulus)	Divides left hand operand by right hand operand and returns remainder.	b % a will give 0

SQL Comparison Operators

Assume **'variable a'** holds 10 and **'variable b'** holds 20, then −

Show Examples

Operator	Description	Example
=	Checks if the values of two operands are equal or not, if yes then condition becomes true.	(a = b) is not true.
!=	Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.	(a != b) is true.
<>	Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.	(a <> b) is true.
>	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	(a > b) is not true.
<	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	(a < b) is true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	(a >= b) is not true.
<=	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	(a <= b) is true.
!<	Checks if the value of left operand is not less than the value of right operand, if yes then condition becomes true.	(a !< b) is false.
!>	Checks if the value of left operand is not greater than the value of right operand, if yes then condition becomes true.	(a !> b) is true.

SQL Logical Operators

Here is a list of all the logical operators available in SQL.

Show Examples

Sr.No.	Operator & Description
1	**ALL** The ALL operator is used to compare a value to all values in another value set.
2	**AND** The AND operator allows the existence of multiple conditions in an SQL statement's WHERE clause.
3	**ANY** The ANY operator is used to compare a value to any applicable value in the list as per the condition.
4	**BETWEEN** The BETWEEN operator is used to search for values that are within a set of values, given the minimum value and the maximum value.
5	**EXISTS** The EXISTS operator is used to search for the presence of a row in a specified table that meets a certain criterion.
6	**IN**

	The IN operator is used to compare a value to a list of literal values that have been specified.
7	**LIKE** The LIKE operator is used to compare a value to similar values using wildcard operators.
8	**NOT** The NOT operator reverses the meaning of the logical operator with which it is used. Eg: NOT EXISTS, NOT BETWEEN, NOT IN, etc. **This is a negate operator.**
9	**OR** The OR operator is used to combine multiple conditions in an SQL statement's WHERE clause.
10	**IS NULL** The NULL operator is used to compare a value with a NULL value.
11	**UNIQUE** The UNIQUE operator searches every row of a specified table for uniqueness (no duplicates).

SQL - Expressions

An expression is a combination of one or more values, operators and SQL functions that evaluate to a value. These SQL EXPRESSIONs are like formulae and they are written in query language. You can also use them to query the database for a specific set of data.

Syntax

Consider the basic syntax of the SELECT statement as follows −

```
SELECT column1, column2, columnN
FROM table_name
WHERE [CONDITION|EXPRESSION];
```

There are different types of SQL expressions, which are mentioned below −

- Boolean
- Numeric
- Date

Let us now discuss each of these in detail.

Boolean Expressions

SQL Boolean Expressions fetch the data based on matching a single value. Following is the syntax −

```
SELECT column1, column2, columnN
FROM table_name
WHERE SINGLE VALUE MATCHING EXPRESSION;
```

The following table is a simple example showing the usage of various SQL Boolean Expressions

```
SQL> SELECT * FROM CUSTOMERS WHERE SALARY = 10000;
+----+-------+-----+---------+----------+
| ID | NAME  | AGE | ADDRESS | SALARY   |
+----+-------+-----+---------+----------+
|  7 | Muffy |  24 | Indore  | 10000.00 |
+----+-------+-----+---------+----------+
1 row in set (0.00 sec)
```

Numeric Expression

These expressions are used to perform any mathematical operation in any query. Following is the syntax −

```
SELECT numerical_expression as  OPERATION_NAME
[FROM table_name
WHERE CONDITION] ;
```

Here, the numerical_expression is used for a mathematical expression or any formula. Following is a simple example showing the usage of SQL Numeric Expressions −

```
SQL> SELECT (15 + 6) AS ADDITION
+----------+
| ADDITION |
+----------+
|       21 |
+----------+
1 row in set (0.00 sec)
```

Aggregate Function

SQL aggregation is the **task of collecting a set of values to return a single value**. It is done with the help of aggregate functions, such as SUM, COUNT, and AVG. For example, in a database of products, you might want to calculate the average price of the whole inventory.

There are several built-in functions like avg(), sum(), count(), etc., to perform what is known as the aggregate data calculations against a table or a specific table column also.

```
SQL> SELECT COUNT(*) AS "RECORDS" FROM CUSTOMERS;
+---------+
| RECORDS |
+---------+
|       7 |
+---------+
1 row in set (0.00 sec)
```

Date Expressions

Date Expressions return current system date and time values −

```
SQL>  SELECT CURRENT_TIMESTAMP;
+---------------------+
| Current_Timestamp   |
+---------------------+
| 2009-11-12 06:40:23 |
+---------------------+
1 row in set (0.00 sec)
```

Another date expression is as shown below −

```
SQL>  SELECT  GETDATE();;
+-------------------------+
| GETDATE                 |
+-------------------------+
| 2009-10-22 12:07:18.140 |
+-------------------------+
1 row in set (0.00 sec)
```

AND & **OR** operators

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 |
+----+----------+----------+
```

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.

Sr.No.	Statement & Description
1	**WHERE SALARY LIKE '200%'** Finds any values that start with 200.
2	**WHERE SALARY LIKE '%200%'** Finds any values that have 200 in any position.
3	**WHERE SALARY LIKE '_00%'** Finds any values that have 00 in the second and third positions.

4	**WHERE SALARY LIKE '2_%_%'** Finds any values that start with 2 and are at least 3 characters in length.
5	**WHERE SALARY LIKE '%2'** Finds any values that end with 2.
6	**WHERE SALARY LIKE '_2%3'** Finds any values that have a 2 in the second position and end with a 3.
7	**WHERE SALARY LIKE '2___3'** Finds any values in a five-digit number that start with 2 and end with 3.