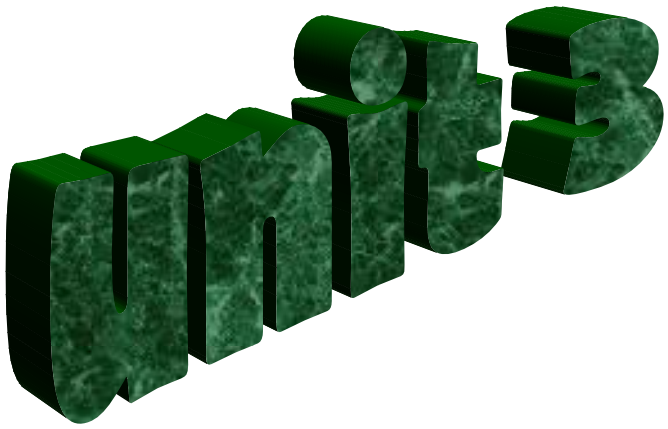


SQL

- SQL stands for Structured Query Language. It is used for storing and managing data in relational database management system (RDMS).
- It is a standard language for Relational Database System. It enables a user to create, read, update and delete relational databases and tables.
- All the RDBMS like MySQL, Informix, Oracle, MS Access and SQL Server use SQL as their standard database language.
- SQL allows users to query the database in a number of ways, using English-like statements.

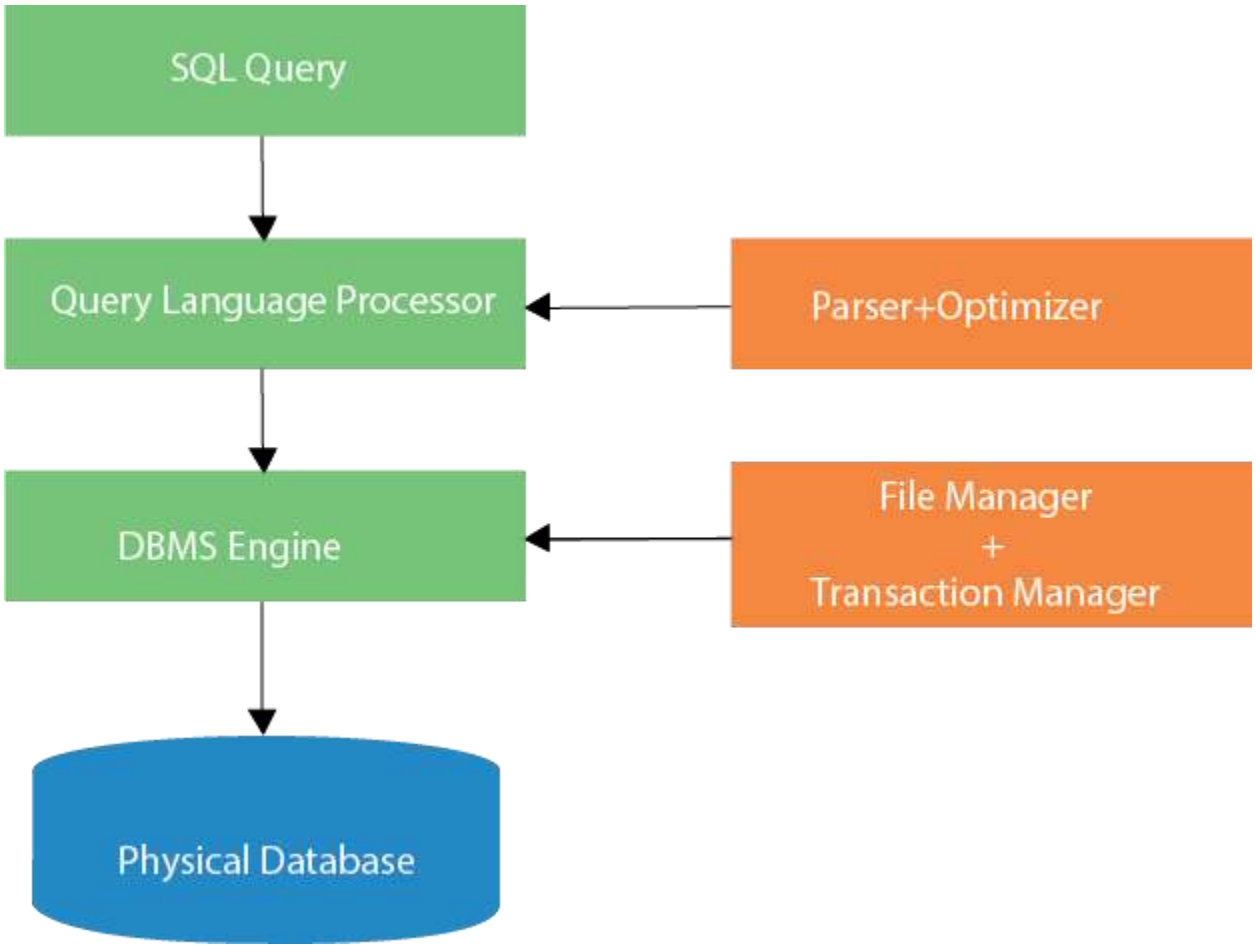
Rules:

SQL follows the following rules:

- Structure query language is not case sensitive. Generally, keywords of SQL are written in uppercase.
- Statements of SQL are dependent on text lines. We can use a single SQL statement on one or multiple text line.
- Using the SQL statements, you can perform most of the actions in a database.
- SQL depends on tuple relational calculus and relational algebra.

SQL process:

- When an SQL command is executing for any RDBMS, then the system figure out the best way to carry out the request and the SQL engine determines that how to interpret the task.
- In the process, various components are included. These components can be optimization Engine, Query engine, Query dispatcher, classic, etc.
- All the non-SQL queries are handled by the classic query engine, but SQL query engine won't handle logical files.



Characteristics of SQL

- SQL is easy to learn.
- SQL is used to access data from relational database management systems.
- SQL can execute queries against the database.
- SQL is used to describe the data.
- SQL is used to define the data in the database and manipulate it when needed.
- SQL is used to create and drop the database and table.
- SQL is used to create a view, stored procedure, function in a database.
- SQL allows users to set permissions on tables, procedures, and views.

Advantages of SQL

There are the following advantages of SQL:

High speed

Using the SQL queries, the user can quickly and efficiently retrieve a large amount of records from a database.

No coding needed

In the standard SQL, it is very easy to manage the database system. It doesn't require a substantial amount of code to manage the database system.

Well defined standards

Long established are used by the SQL databases that are being used by ISO and ANSI.

Portability

SQL can be used in laptop, PCs, server and even some mobile phones.

Interactive language

SQL is a domain language used to communicate with the database. It is also used to receive answers to the complex questions in seconds.

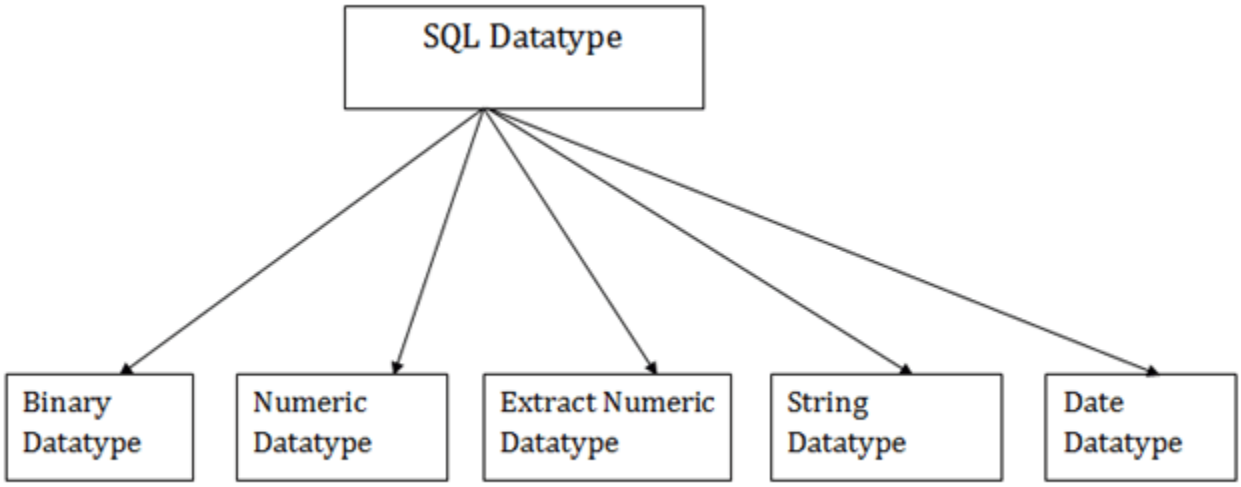
Multiple data view

Using the SQL language, the users can make different views of the database structure.

SQL Datatype

- SQL Datatype is used to define the values that a column can contain.
- Every column is required to have a name and data type in the database table.

Datatype of SQL:



1. Binary Datatypes

There are Three types of binary Datatypes which are given below:

Data Type	Description
binary	It has a maximum length of 8000 bytes. It contains fixed-length binary data.
varbinary	It has a maximum length of 8000 bytes. It contains variable-length binary data.
image	It has a maximum length of 2,147,483,647 bytes. It contains variable-length binary data.

2. Approximate Numeric Datatype :

The subtypes are given below:

Data type	From	To	Description
float	-1.79E + 308	1.79E + 308	It is used to specify a floating-point value e.g. 6.2, 2.9 etc.

real	-3.40e + 38	3.40E + 38	It specifies a single precision floating point number

3. Exact Numeric Datatype

The subtypes are given below:

Data type	Description
int	It is used to specify an integer value.
smallint	It is used to specify small integer value.
bit	It has the number of bits to store.
decimal	It specifies a numeric value that can have a decimal number.
numeric	It is used to specify a numeric value.

4. Character String Datatype

The subtypes are given below:

Data type	Description
char	It has a maximum length of 8000 characters. It contains Fixed-length non-unicode characters.
varchar	It has a maximum length of 8000 characters. It contains variable-length non-unicode characters.
text	It has a maximum length of 2,147,483,647 characters. It contains variable-length non-unicode characters.

5. Date and time Datatypes

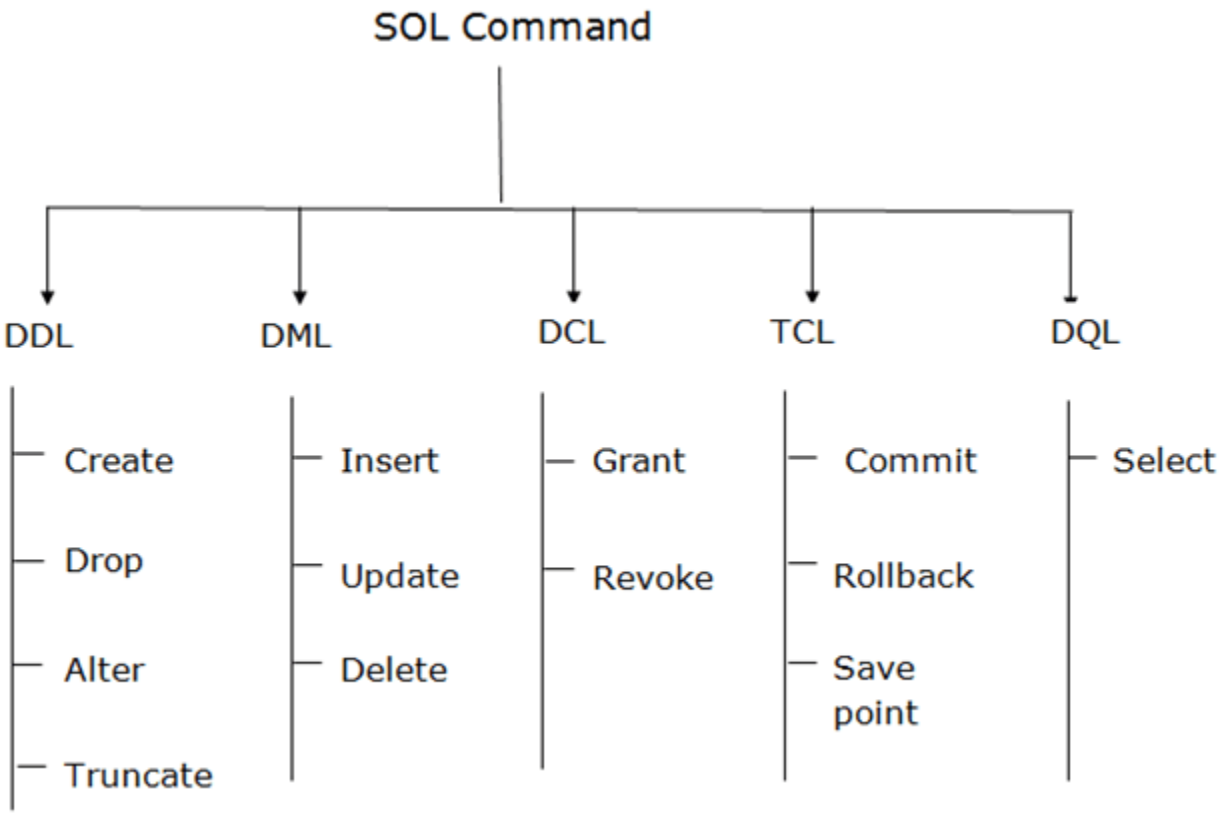
The subtypes are given below:

Datatype	Description
date	It is used to store the year, month, and days value.
time	It is used to store the hour, minute, and second values.
timestamp	It stores the year, month, day, hour, minute, and the second value.

SQL command

- SQL commands are instructions. It is used to communicate with the database. It is also used to perform specific tasks, functions, and queries of data.
- SQL can perform various tasks like create a table, add data to tables, drop the table, modify the table, set permission for users.

Types of SQL Command:



1. Data definition language (DDL)

- DDL changes the structure of the table like creating a table, deleting a table, altering a table, etc.
- All the command of DDL are auto-committed that means it permanently save all the changes in the database.

Here are some commands that come under DDL:

- CREATE
- ALTER
- DROP
- TRUNCATE

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

Syntax:

1. CREATE TABLE TABLE_NAME (COLUMN_NAME DATATYPES[,....]);

Example:

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

b. DROP: It is used to delete both the structure and record stored in the table.

Syntax

```
1. DROP TABLE ;
```

Example

```
1. DROP TABLE EMPLOYEE;
```

c. 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.

Syntax:

To add a new column in the table

```
1. ALTER TABLE table_name ADD column_name COLUMN-definition;
```

To modify existing column in the table:

```
1. ALTER TABLE MODIFY(COLUMN DEFINITION....);
```

EXAMPLE

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

d. TRUNCATE: It is used to delete all the rows from the table and free the space containing the table.

Syntax:

```
1. TRUNCATE TABLE table_name;
```

Example:

```
1. TRUNCATE TABLE EMPLOYEE;
```

2. Data Manipulation Language

- DML commands are used to modify the database. It is responsible for all form of changes in the database.
- The command of DML is not auto-committed that means it can't permanently save all the changes in the database. They can be rollback.

Here are some commands that come under DML:

- INSERT
- UPDATE
- DELETE

a. INSERT: The INSERT statement is a SQL query. It is used to insert data into the row of a table.

Syntax:

1. INSERT INTO TABLE_NAME
   (col1, col2, col3,.... col N) VALUES (value1, value2, value3, ... valueN);

   Or

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

For example:

1. INSERT INTO javatpoint (Author, Subject) VALUES ("Sonoo", "DBMS");

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

Syntax:

1. UPDATE table_name SET [column_name1= value1,...column_nameN = valueN] [WHERE CO
   NDITION]

For example:

1. UPDATE students SET User_Name = 'Sonoo' WHERE Student_Id = '3'

c. DELETE: It is used to remove one or more row from a table.

Syntax:

1. DELETE FROM table_name [WHERE condition];

For example:

1. DELETE FROM javatpoint WHERE Author="Sonoo";

3. Data Control Language

DCL commands are used to grant and take back authority from any database user.

Here are some commands that come under DCL:

- Grant
- Revoke

a. Grant: It is used to give user access privileges to a database.

Example

1. GRANT SELECT, UPDATE ON MY_TABLE TO SOME_USER, ANOTHER_USER;

b. Revoke: It is used to take back permissions from the user.

Example

1. REVOKE SELECT, UPDATE ON MY_TABLE FROM USER1, USER2;

4. Transaction Control Language

TCL commands can only use with DML commands like INSERT, DELETE and UPDATE only.

These operations are automatically committed in the database that's why they cannot be used while creating tables or dropping them.

Here are some commands that come under TCL:

- COMMIT
- ROLLBACK
- SAVEPOINT

a. Commit: Commit command is used to save all the transactions to the database.

Syntax:

```
1. COMMIT;
```

Example:

```
1. DELETE FROM CUSTOMERS WHERE AGE = 25;
2. COMMIT;
```

b. Rollback: Rollback command is used to undo transactions that have not already been saved to the database.

Syntax:

```
1. ROLLBACK;
```

Example:

```
1. DELETE FROM CUSTOMERS WHERE AGE = 25;
2. ROLLBACK;
```

c. SAVEPOINT: It is used to roll the transaction back to a certain point without rolling back the entire transaction.

Syntax:

```
1. SAVEPOINT SAVEPOINT_NAME;
```

5. Data Query Language

DQL is used to fetch the data from the database.

It uses only one command:

- SELECT

a. 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:

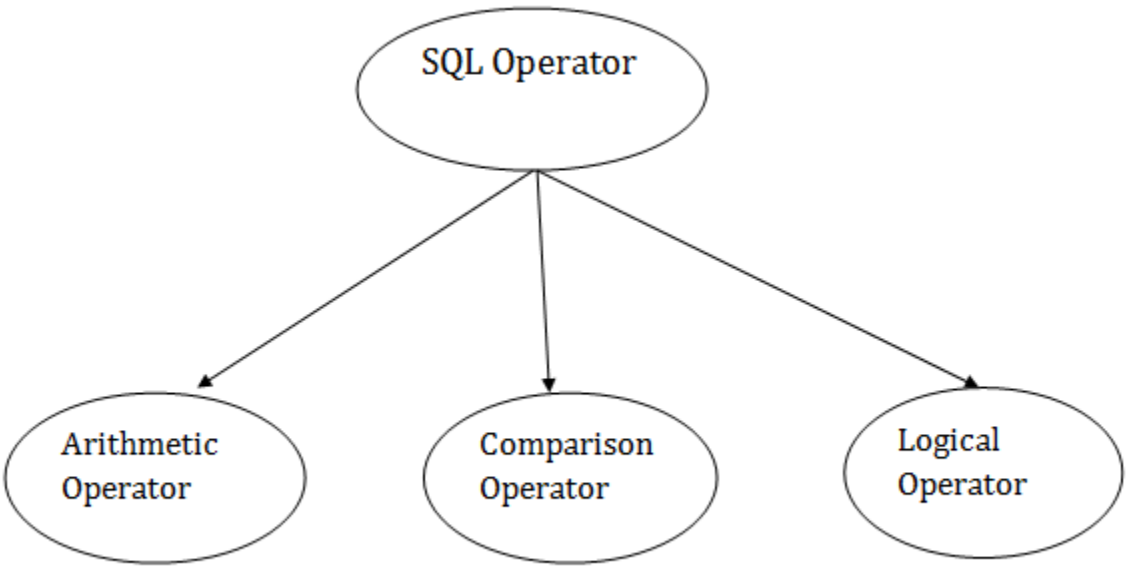
```
1. SELECT expressions FROM TABLES WHERE conditions;
```

For example:

```
1. SELECT emp_name  FROM employee  WHERE age > 20;
```

SQL Operator

There are various types of SQL operator:



SQL Arithmetic Operators

Let's assume 'variable a' and 'variable b'. Here, 'a' contains 20 and 'b' contains 10.

Operator	Description	Example
+	It adds the value of both operands.	a+b will give 30
-	It is used to subtract the right-hand operand from the left-hand operand.	a-b will give 10
*	It is used to multiply the value of both operands.	a*b will give 200
/	It is used to divide the left-hand operand by the right-hand operand.	a/b will give 2

%	It is used to divide the left-hand operand by the right-hand operand and returns reminder.	a%b will give 0

SQL Comparison Operators:

Let's assume 'variable a' and 'variable b'. Here, 'a' contains 20 and 'b' contains 10.

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

SQL Logical Operators

There is the list of logical operator used in SQL:

Operator	Description
ALL	It compares a value to all values in another value set.
AND	It allows the existence of multiple conditions in an SQL statement.
ANY	It compares the values in the list according to the condition.
BETWEEN	It is used to search for values that are within a set of values.
IN	It compares a value to that specified list value.
NOT	It reverses the meaning of any logical operator.
OR	It combines multiple conditions in SQL statements.
EXISTS	It is used to search for the presence of a row in a specified table.
LIKE	It compares a value to similar values using wildcard operator.

SQL Table

- SQL Table is a collection of data which is organized in terms of rows and columns. In DBMS, the table is known as relation and row as a tuple.
- Table is a simple form of data storage. A table is also considered as a convenient representation of relations.

Let's see an example of the **EMPLOYEE** table:

EMP_ID	EMP_NAME	CITY	PHONE_NO
1	Kristen	Washington	7289201223
2	Anna	Franklin	9378282882
3	Jackson	Bristol	9264783838
4	Kellan	California	7254728346
5	Ashley	Hawaii	9638482678

In the above table, "EMPLOYEE" is the table name, "EMP_ID", "EMP_NAME", "CITY", "PHONE_NO" are the column names. The combination of data of multiple columns forms a row, e.g., 1, "Kristen", "Washington" and 7289201223 are the data of one row.

Operation on Table

1. Create table
2. Drop table
3. Delete table
4. Rename table

SQL Create Table

SQL create table is used to create a table in the database. To define the table, you should define the name of the table and also define its columns and column's data type.

Syntax

```
1. create table "table_name"("column1" "data type","column2" "data type","column3" "data ty
   pe", ... "columnN" "data type");
```

Example

```
1. SQL> CREATE TABLE EMPLOYEE (
2. EMP_ID INT                    NOT NULL,
3. EMP_NAME VARCHAR (25) NOT NULL,
4. PHONE_NO INT                   NOT NULL,
5. ADDRESS CHAR (30),
6. PRIMARY KEY (ID)
7. );
```

If you create the table successfully, you can verify the table by looking at the message by the SQL server. Else you can use DESC command as follows:

SQL> DESC EMPLOYEE;

Field	Type	Null	Key	Default	Extra
EMP_ID	int(11)	NO	PRI	NULL	
EMP_NAME	varchar(25)	NO		NULL	

PHONE_NO	NO	int(11)		NULL	
ADDRESS	YES			NULL	char(30)

- 4 rows in set (0.35 sec)

Now you have an EMPLOYEE table in the database, and you can use the stored information related to the employees.

Drop table

A SQL drop table is used to delete a table definition and all the data from a table. When this command is executed, all the information available in the table is lost forever, so you have to very careful while using this command.

Syntax

1. DROP TABLE "table_name";

Firstly, you need to verify the **EMPLOYEE** table using the following command:

1. SQL> DESC EMPLOYEE;

Field	Type	Null	Key	Default	Extra
EMP_ID	int(11)	NO	PRI	NULL	
EMP_NAME	varchar(25)	NO		NULL	
PHONE_NO	NO	int(11)		NULL	
ADDRESS	YES			NULL	char(30)

- 4 rows in set (0.35 sec)

This table shows that EMPLOYEE table is available in the database, so we can drop it as follows:

1. SQL>DROP TABLE EMPLOYEE;

Now, we can check whether the table exists or not using the following command:

1. Query OK, 0 rows affected (0.01 sec)

   As this shows that the table is dropped, so it doesn't display it.

SQL DELETE table

In SQL, DELETE statement is used to delete rows from a table. We can use WHERE condition to delete a specific row from a table. If you want to delete all the records from the table, then you don't need to use the WHERE clause.

Syntax

1. DELETE FROM table_name WHERE condition;

Example

Suppose, the EMPLOYEE table having the following records:

EMP_ID	EMP_NAME	CITY	PHONE_NO	SALARY
1	Kristen	Chicago	9737287378	150000
2	Russell	Austin	9262738271	200000
3	Denzel	Boston	7353662627	100000
4	Angelina	Denver	9232673822	600000
5	Robert	Washington	9367238263	350000
6	Christian	Los angels	7253847382	260000

The following query will DELETE an employee whose ID is 2.

1. SQL> DELETE FROM EMPLOYEE WHERE EMP_ID = 3;

Now, the EMPLOYEE table would have the following records.

EMP_ID	EMP_NAME	CITY	PHONE_NO	SALARY

1	Kristen	Chicago	9737287378	150000
2	Russell	Austin	9262738271	200000
4	Angelina	Denver	9232673822	600000
5	Robert	Washington	9367238263	350000
6	Christian	Los angels	7253847382	260000

If you don't specify the WHERE condition, it will remove all the rows from the table.

1. DELETE FROM EMPLOYEE;

Now, the EMPLOYEE table would not have any records.

SQL SELECT Statement

In SQL, the SELECT statement is used to query or retrieve data from a table in the database. The returns data is stored in a table, and the result table is known as result-set.

Syntax

1. SELECT column1, column2, ... FROM table_name;

Here, the expression is the field name of the table that you want to select data from.

Use the following syntax to select all the fields available in the table:

1. SELECT * FROM table_name;

Example:

EMPLOYEE

EMP_ID	EMP_NAME	CITY	PHONE_NO	SALARY
1	Kristen	Chicago	9737287378	150000
2	Russell	Austin	9262738271	200000

3	Angelina	Denver	9232673822	600000
4	Robert	Washington	9367238263	350000
5	Christian	Los angels	7253847382	260000

To fetch the EMP_ID of all the employees, use the following query:

1. SELECT EMP_ID FROM EMPLOYEE;

Output

EMP_ID
1
2
3
4
5

To fetch the EMP_NAME and SALARY, use the following query:

1. SELECT EMP_NAME, SALARY FROM EMPLOYEE;

EMP_NAME	SALARY
Kristen	150000
Russell	200000
Angelina	600000
Robert	350000
Christian	260000

To fetch all the fields from the EMPLOYEE table, use the following query:

1. SELECT * FROM EMPLOYEE

Output

EMP_ID	EMP_NAME	CITY	PHONE_NO	SALARY
1	Kristen	Chicago	9737287378	150000
2	Russell	Austin	9262738271	200000
3	Angelina	Denver	9232673822	600000
4	Robert	Washington	9367238263	350000
5	Christian	Los angels	7253847382	260000

SQL INSERT Statement

The SQL INSERT statement is used to insert a single or multiple data in a table. In SQL, You can insert the data in two ways:

1. Without specifying column name
2. By specifying column name

Sample Table

EMPLOYEE

EMP_ID	EMP_NAME	CITY	SALARY	AGE
1	Angelina	Chicago	200000	30
2	Robert	Austin	300000	26
3	Christian	Denver	100000	42
4	Kristen	Washington	500000	29
5	Russell	Los angels	200000	36

1. Without specifying column name

If you want to specify all column values, you can specify or ignore the column values.

Syntax

1. INSERT INTO TABLE_NAME VALUES (value1, value2, value 3, ... Value N);

Query

1. INSERT INTO EMPLOYEE VALUES (6, 'Marry', 'Canada', 600000, 48);

Output: After executing this query, the EMPLOYEE table will look like:

EMP_ID	EMP_NAME	CITY	SALARY	AGE
1	Angelina	Chicago	200000	30
2	Robert	Austin	300000	26
3	Christian	Denver	100000	42
4	Kristen	Washington	500000	29
5	Russell	Los angels	200000	36
6	Marry	Canada	600000	48

2. By specifying column name

To insert partial column values, you must have to specify the column names.

Syntax

1. INSERT INTO TABLE_NAME [(col1, col2, col3,.... col N)] VALUES (value1, value2, value 3, .. Value N);

Query

1. INSERT INTO EMPLOYEE (EMP_ID, EMP_NAME, AGE) VALUES (7, 'Jack', 40);

Output: After executing this query, the table will look like:

EMP_ID	EMP_NAME	CITY	SALARY	AGE
1	Angelina	Chicago	200000	30
2	Robert	Austin	300000	26
3	Christian	Denver	100000	42
4	Kristen	Washington	500000	29
5	Russell	Los angels	200000	36
6	Marry	Canada	600000	48
7	Jack	null	null	40

Note: In SQL INSERT query, if you add values for all columns then there is no need to specify the column name. But, you must be sure that you are entering the values in the same order as the column exists.

SQL Update Statement

The SQL UPDATE statement is used to modify the data that is already in the database. The condition in the WHERE clause decides that which row is to be updated.

Syntax

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

Sample Table

EMPLOYEE

EMP_ID	EMP_NAME	CITY	SALARY	AGE
1	Angelina	Chicago	200000	30
2	Robert	Austin	300000	26

3	Christian	Denver	100000	42
4	Kristen	Washington	500000	29
5	Russell	Los angels	200000	36
6	Marry	Canada	600000	48

Updating single record

Update the column EMP_NAME and set the value to 'Emma' in the row where SALARY is 500000.

Syntax

```
1. UPDATE table_name
2. SET column_name = value
3. WHERE condition;
```

Query

```
1. UPDATE EMPLOYEE
2. SET EMP_NAME = 'Emma'
3. WHERE SALARY = 500000;
```

Output: After executing this query, the EMPLOYEE table will look like:

EMP_ID	EMP_NAME	CITY	SALARY	AGE
1	Angelina	Chicago	200000	30
2	Robert	Austin	300000	26
3	Christian	Denver	100000	42
4	Emma	Washington	500000	29
5	Russell	Los angels	200000	36
6	Marry	Canada	600000	48

Updating multiple records

If you want to update multiple columns, you should separate each field assigned with a comma. In the EMPLOYEE table, update the column EMP_NAME to 'Kevin' and CITY to 'Boston' where EMP_ID is 5.

Syntax

```
1. UPDATE table_name
2. SET column_name = value1, column_name2 = value2
3. WHERE condition;
```

Query

```
1. UPDATE EMPLOYEE
2. SET EMP_NAME = 'Kevin', City = 'Boston'
3. WHERE EMP_ID = 5;
```

Output

EMP_ID	EMP_NAME	CITY	SALARY	AGE
1	Angelina	Chicago	200000	30
2	Robert	Austin	300000	26
3	Christian	Denver	100000	42
4	Kristen	Washington	500000	29
5	Kevin	Boston	200000	36
6	Marry	Canada	600000	48

Without use of WHERE clause

If you want to update all row from a table, then you don't need to use the WHERE clause. In the EMPLOYEE table, update the column EMP_NAME as 'Harry'.

Syntax

```
1. UPDATE table_name
```

2. SET column_name = value1;

Query

```
1. UPDATE EMPLOYEE
2. SET EMP_NAME = 'Harry';
```

Output

EMP_ID	EMP_NAME	CITY	SALARY	AGE
1	Harry	Chicago	200000	30
2	Harry	Austin	300000	26
3	Harry	Denver	100000	42
4	Harry	Washington	500000	29
5	Harry	Los angels	200000	36
6	Harry	Canada	600000	48

SQL DELETE Statement

The SQL DELETE statement is used to delete rows from a table. Generally, DELETE statement removes one or more records form a table.

Syntax

```
1. DELETE FROM table_name WHERE some_condition;
```

Sample Table

EMPLOYEE

EMP_ID	EMP_NAME	CITY	SALARY	AGE
1	Angelina	Chicago	200000	30
2	Robert	Austin	300000	26

3	Christian	Denver	100000	42
4	Kristen	Washington	500000	29
5	Russell	Los angels	200000	36
6	Marry	Canada	600000	48

Deleting Single Record

Delete the row from the table EMPLOYEE where EMP_NAME = 'Kristen'. This will delete only the fourth row.

Query

```
1. DELETE FROM EMPLOYEE
2. WHERE EMP_NAME = 'Kristen';
```

Output: After executing this query, the EMPLOYEE table will look like:

EMP_ID	EMP_NAME	CITY	SALARY	AGE
1	Angelina	Chicago	200000	30
2	Robert	Austin	300000	26
3	Christian	Denver	100000	42
5	Russell	Los angels	200000	36
6	Marry	Canada	600000	48

Deleting Multiple Record

Delete the row from the EMPLOYEE table where AGE is 30. This will delete two rows(first and third row).

Query

```
1. DELETE FROM EMPLOYEE WHERE AGE= 30;
```

Output: After executing this query, the EMPLOYEE table will look like:

EMP_ID	EMP_NAME	CITY	SALARY	AGE
2	Robert	Austin	300000	26
3	Christian	Denver	100000	42
5	Russell	Los angels	200000	36
6	Marry	Canada	600000	48

Delete all of the records

Delete all the row from the EMPLOYEE table. After this, no records left to display. The EMPLOYEE table will become empty.

Syntax

1. DELETE * FROM table_name;
2. or
3. DELETE FROM table_name;

Query

1. DELETE FROM EMPLOYEE;

Output: After executing this query, the EMPLOYEE table will look like:

EMP_ID	EMP_NAME	CITY	SALARY

Note: Using the condition in the WHERE clause, we can delete single as well as multiple records. If you want to delete all the records from the table, then you don't need to use the WHERE clause.

SQL Sub Query

A Subquery is a query within another SQL query and embedded within the WHERE clause.

Important Rule:

- A subquery can be placed in a number of SQL clauses like WHERE clause, FROM clause, HAVING clause.
- You can use Subquery with SELECT, UPDATE, INSERT, DELETE statements along with the operators like =, <, >, >=, <=, IN, BETWEEN, etc.
- A subquery is a query within another query. The outer query is known as the main query, and the inner query is known as a subquery.
- Subqueries are on the right side of the comparison operator.
- A subquery is enclosed in parentheses.
- In the Subquery, ORDER BY command cannot be used. But GROUP BY command can be used to perform the same function as ORDER BY command.

1. Subqueries with the Select Statement

SQL subqueries are most frequently used with the Select statement.

Syntax

```
1. SELECT column_name
2. FROM table_name
3. WHERE column_name expression operator
4. ( SELECT column_name  from table_name WHERE ... );
```

Example

Consider the EMPLOYEE table have the following records:

ID	NAME	AGE	ADDRESS	SALARY
1	John	20	US	2000.00
2	Stephan	26	Dubai	1500.00
3	David	27	Bangkok	2000.00
4	Alina	29	UK	6500.00
5	Kathrin	34	Bangalore	8500.00
6	Harry	42	China	4500.00

7	Jackson	25	Mizoram	10000.00

The subquery with a SELECT statement will be:

```
1. SELECT *
2.     FROM EMPLOYEE
3.     WHERE ID IN (SELECT ID
4.     FROM EMPLOYEE
5.     WHERE SALARY > 4500);
```

This would produce the following result:

ID	NAME	AGE	ADDRESS	SALARY
4	Alina	29	UK	6500.00
5	Kathrin	34	Bangalore	8500.00
7	Jackson	25	Mizoram	10000.00

2. Subqueries with the INSERT Statement

- SQL subquery can also be used with the Insert statement. In the insert statement, data returned from the subquery is used to insert into another table.
- In the subquery, the selected data can be modified with any of the character, date functions.

Syntax:

```
1. INSERT INTO table_name (column1, column2, column3....)
2. SELECT *
3. FROM table_name
4. WHERE VALUE OPERATOR
```

Example

Consider a table EMPLOYEE_BKP with similar as EMPLOYEE.

Now use the following syntax to copy the complete EMPLOYEE table into the EMPLOYEE_BKP table.

```
1. INSERT INTO EMPLOYEE_BKP
2.    SELECT * FROM EMPLOYEE
3.    WHERE ID IN (SELECT ID
4.    FROM EMPLOYEE);
```

3. Subqueries with the UPDATE Statement

The subquery of SQL can be used in conjunction with the Update statement. When a subquery is used with the Update statement, then either single or multiple columns in a table can be updated.

Syntax

```
1. UPDATE table
2. SET column_name = new_value
3. WHERE VALUE OPERATOR
4.    (SELECT COLUMN_NAME
5.    FROM TABLE_NAME
6.    WHERE condition);
```

Example

Let's assume we have an EMPLOYEE_BKP table available which is backup of EMPLOYEE table. The given example updates the SALARY by .25 times in the EMPLOYEE table for all employee whose AGE is greater than or equal to 29.

```
1. UPDATE EMPLOYEE
2.    SET SALARY = SALARY * 0.25
3.    WHERE AGE IN (SELECT AGE FROM CUSTOMERS_BKP
4.       WHERE AGE >= 29);
```

This would impact three rows, and finally, the EMPLOYEE table would have the following records.

ID	NAME	AGE	ADDRESS	SALARY
1	John	20	US	2000.00
2	Stephan	26	Dubai	1500.00
3	David	27	Bangkok	2000.00

4	Alina	29	UK	1625.00
5	Kathrin	34	Bangalore	2125.00
6	Harry	42	China	1125.00
7	Jackson	25	Mizoram	10000.00

4. Subqueries with the DELETE Statement

The subquery of SQL can be used in conjunction with the Delete statement just like any other statements mentioned above.

Syntax

```
1. DELETE FROM TABLE_NAME
2. WHERE VALUE OPERATOR
3.    (SELECT COLUMN_NAME
4.    FROM TABLE_NAME
5.    WHERE condition);
```

Example

Let's assume we have an EMPLOYEE_BKP table available which is backup of EMPLOYEE table. The given example deletes the records from the EMPLOYEE table for all EMPLOYEE whose AGE is greater than or equal to 29.

```
1. DELETE FROM EMPLOYEE
2.    WHERE AGE IN (SELECT AGE FROM EMPLOYEE_BKP
3.       WHERE AGE >= 29 );
```

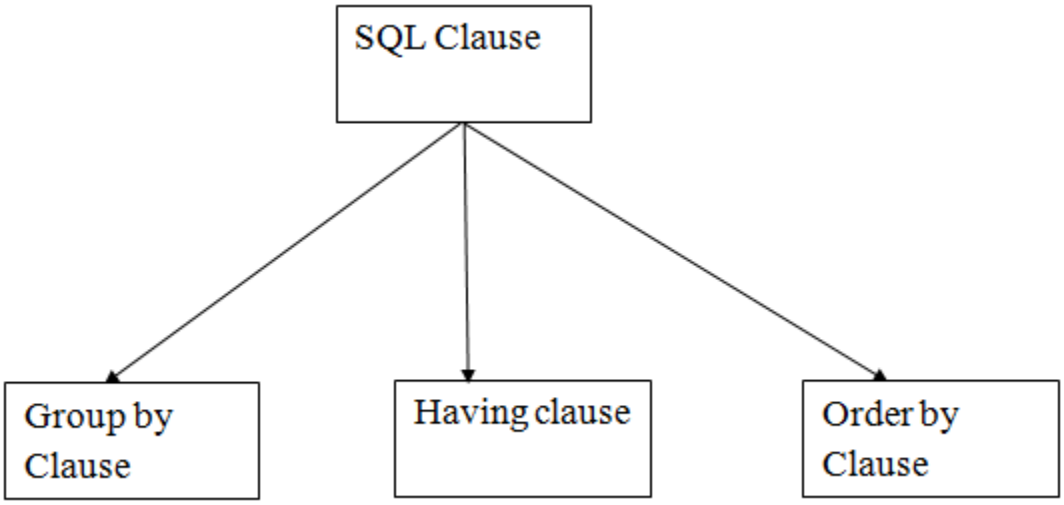
This would impact three rows, and finally, the EMPLOYEE table would have the following records.

ID	NAME	AGE	ADDRESS	SALARY
1	John	20	US	2000.00

2	Stephan	26	Dubai	1500.00
3	David	27	Bangkok	2000.00
7	Jackson	25	Mizoram	10000.00

SQL Clauses

The following are the various SQL clauses:



1. GROUP BY

- SQL GROUP BY statement is used to arrange identical data into groups. The GROUP BY statement is used with the SQL SELECT statement.
- The GROUP BY statement follows the WHERE clause in a SELECT statement and precedes the ORDER BY clause.
- The GROUP BY statement is used with aggregation function.

Syntax

1. SELECT column
2. FROM table_name
3. WHERE conditions

4. GROUP BY column
5. ORDER BY column

Sample table:

PRODUCT_MAST

PRODUCT	COMPANY	QTY	RATE	COST
Item1	Com1	2	10	20
Item2	Com2	3	25	75
Item3	Com1	2	30	60
Item4	Com3	5	10	50
Item5	Com2	2	20	40
Item6	Cpm1	3	25	75
Item7	Com1	5	30	150
Item8	Com1	3	10	30
Item9	Com2	2	25	50
Item10	Com3	4	30	120

Example:

```
1. SELECT COMPANY, COUNT(*)
2. FROM PRODUCT_MAST
3. GROUP BY COMPANY;
```

Output:

```
Com1    5
Com2    3
Com3    2
```

2. HAVING

- HAVING clause is used to specify a search condition for a group or an aggregate.
- Having is used in a GROUP BY clause. If you are not using GROUP BY clause then you can use HAVING function like a WHERE clause.

Syntax:

```
SELECT column1, column2
FROM table_name
WHERE conditions
GROUP BY column1, column2
HAVING conditions
ORDER BY column1, column2;
```

Example:

```
SELECT COMPANY, COUNT(*)
FROM PRODUCT_MAST
GROUP BY COMPANY
HAVING COUNT(*)>2;
```

Output:

```
Com1    5
Com2    3
```

3. ORDER BY

- The ORDER BY clause sorts the result-set in ascending or descending order.
- It sorts the records in ascending order by default. DESC keyword is used to sort the records in descending order.

Syntax:

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

Where

ASC: It is used to sort the result set in ascending order by expression.

DESC: It sorts the result set in descending order by expression.

Example: Sorting Results in Ascending Order

Table:

CUSTOMER

CUSTOMER_ID	NAME	ADDRESS
12	Kathrin	US
23	David	Bangkok
34	Alina	Dubai
45	John	UK
56	Harry	US

Enter the following SQL statement:

```
1. SELECT *
2. FROM CUSTOMER
3. ORDER BY NAME;
```

Output:

CUSTOMER_ID	NAME	ADDRESS
34	Alina	Dubai
23	David	Bangkok
56	Harry	US
45	John	UK
12	Kathrin	US

Example: Sorting Results in Descending Order

Using the above CUSTOMER table

```
1. SELECT *
2. FROM CUSTOMER
3. ORDER BY NAME DESC;
```

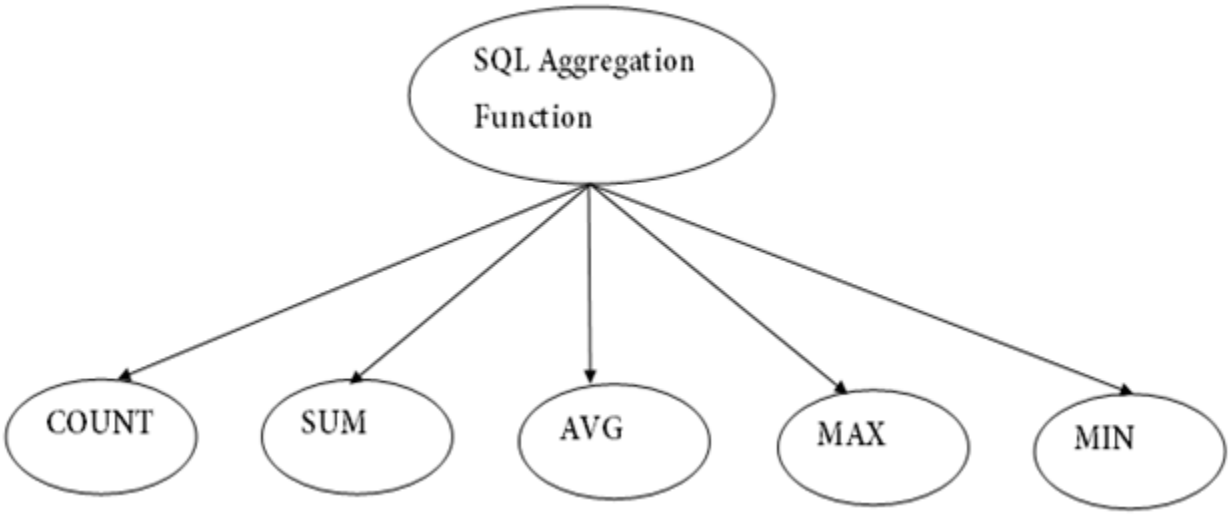
Output:

CUSTOMER_ID	NAME	ADDRESS
12	Kathrin	US
45	John	UK
56	Harry	US
23	David	Bangkok
34	Alina	Dubai

SQL Aggregate Functions

- SQL aggregation function is used to perform the calculations on multiple rows of a single column of a table. It returns a single value.
- It is also used to summarize the data.

Types of SQL Aggregation Function



1. COUNT FUNCTION

- COUNT function is used to Count the number of rows in a database table. It can work on both numeric and non-numeric data types.
- COUNT function uses the COUNT(*) that returns the count of all the rows in a specified table. COUNT(*) considers duplicate and Null.

Syntax

1. COUNT(*)
2. or
3. COUNT([ALL|DISTINCT] expression)

Sample table:

PRODUCT_MAST

PRODUCT	COMPANY	QTY	RATE	COST
Item1	Com1	2	10	20
Item2	Com2	3	25	75
Item3	Com1	2	30	60

Item4	Com3	5	10	50
Item5	Com2	2	20	40
Item6	Cpm1	3	25	75
Item7	Com1	5	30	150
Item8	Com1	3	10	30
Item9	Com2	2	25	50
Item10	Com3	4	30	120

Example: COUNT()

```
1. SELECT COUNT(*)
2. FROM PRODUCT_MAST;
```

Output:

```
10
```

Example: COUNT with WHERE

```
1. SELECT COUNT(*)
2. FROM PRODUCT_MAST;
3. WHERE RATE>=20;
```

Output:

```
7
```

Example: COUNT() with DISTINCT

```
1. SELECT COUNT(DISTINCT COMPANY)
2. FROM PRODUCT_MAST;
```

Output:

```
3
```

Example: COUNT() with GROUP BY

```
1. SELECT COMPANY, COUNT(*)
2. FROM PRODUCT_MAST
3. GROUP BY COMPANY;
```

Output:

```
Com1     5
Com2     3
Com3     2
```

Example: COUNT() with HAVING

```
1. SELECT COMPANY, COUNT(*)
2. FROM PRODUCT_MAST
3. GROUP BY COMPANY
4. HAVING COUNT(*)>2;
```

Output:

```
Com1     5
Com2     3
```

2. SUM Function

Sum function is used to calculate the sum of all selected columns. It works on numeric fields only.

Syntax

```
1. SUM()
2. or
3. SUM( [ALL|DISTINCT] expression )
```

Example: SUM()

```
1. SELECT SUM(COST)
2. FROM PRODUCT_MAST;
```

Output:

```
670
```

Example: SUM() with WHERE

```
1. SELECT SUM(COST)
```

```
2. FROM PRODUCT_MAST
3. WHERE QTY>3;
```

Output:

```
320
```

Example: SUM() with GROUP BY

```
1. SELECT SUM(COST)
2. FROM PRODUCT_MAST
3. WHERE QTY>3
4. GROUP BY COMPANY;
```

Output:

```
Com1    150
Com2    170
```

Example: SUM() with HAVING

```
1. SELECT COMPANY, SUM(COST)
2. FROM PRODUCT_MAST
3. GROUP BY COMPANY
4. HAVING SUM(COST)>=170;
```

Output:

```
Com1    335
Com3    170
```

3. AVG function

The AVG function is used to calculate the average value of the numeric type. AVG function returns the average of all non-Null values.

Syntax

```
1. AVG()
2. or
3. AVG( [ALL|DISTINCT] expression )
```

Example:

```
1. SELECT AVG(COST)
```

2. FROM PRODUCT_MAST;

Output:

```
67.00
```

4. MAX Function

MAX function is used to find the maximum value of a certain column. This function determines the largest value of all selected values of a column.

Syntax

1. MAX()
2. or
3. MAX([ALL|DISTINCT] expression)

Example:

1. SELECT MAX(RATE)
2. FROM PRODUCT_MAST;

```
30
```

5. MIN Function

MIN function is used to find the minimum value of a certain column. This function determines the smallest value of all selected values of a column.

Syntax

1. MIN()
2. or
3. MIN([ALL|DISTINCT] expression)

Example:

1. SELECT MIN(RATE)
2. FROM PRODUCT_MAST;

Output:

```
10
```

SQL JOIN

As the name shows, JOIN means to combine something. In case of SQL, JOIN means "to combine two or more tables".

In SQL, JOIN clause is used to combine the records from two or more tables in a database.

Types of SQL JOIN

1. INNER JOIN
2. LEFT JOIN
3. RIGHT JOIN
4. FULL JOIN

Sample Table

EMPLOYEE

EMP_ID	EMP_NAME	CITY	SALARY	AGE
1	Angelina	Chicago	200000	30
2	Robert	Austin	300000	26
3	Christian	Denver	100000	42
4	Kristen	Washington	500000	29
5	Russell	Los angels	200000	36
6	Marry	Canada	600000	48

PROJECT

PROJECT_NO	EMP_ID	DEPARTMENT
101	1	Testing
102	2	Development
103	3	Designing

104	4	Development

1. INNER JOIN

In SQL, INNER JOIN selects records that have matching values in both tables as long as the condition is satisfied. It returns the combination of all rows from both the tables where the condition satisfies.

Syntax

```
SELECT table1.column1, table1.column2, table2.column1,....
FROM table1
INNER JOIN table2
ON table1.matching_column = table2.matching_column;
```

Query

```
SELECT EMPLOYEE.EMP_NAME, PROJECT.DEPARTMENT
FROM EMPLOYEE
INNER JOIN PROJECT
ON PROJECT.EMP_ID = EMPLOYEE.EMP_ID;
```

Output

EMP_NAME	DEPARTMENT
Angelina	Testing
Robert	Development
Christian	Designing
Kristen	Development

2. LEFT JOIN

The SQL left join returns all the values from left table and the matching values from the right table. If there is no matching join value, it will return NULL.

Syntax

```
SELECT table1.column1, table1.column2, table2.column1,....
```

```
2. FROM table1
3. LEFT JOIN table2
4. ON table1.matching_column = table2.matching_column;
```

Query

```
1. SELECT EMPLOYEE.EMP_NAME, PROJECT.DEPARTMENT
2. FROM EMPLOYEE
3. LEFT JOIN PROJECT
4. ON PROJECT.EMP_ID = EMPLOYEE.EMP_ID;
```

Output

EMP_NAME	DEPARTMENT
Angelina	Testing
Robert	Development
Christian	Designing
Kristen	Development
Russell	NULL
Marry	NULL

3. RIGHT JOIN

In SQL, RIGHT JOIN returns all the values from the values from the rows of right table and the matched values from the left table. If there is no matching in both tables, it will return NULL.

Syntax

```
1. SELECT table1.column1, table1.column2, table2.column1,....
2. FROM table1
3. RIGHT JOIN table2
4. ON table1.matching_column = table2.matching_column;
```

Query

```
1. SELECT EMPLOYEE.EMP_NAME, PROJECT.DEPARTMENT
2. FROM EMPLOYEE
3. RIGHT JOIN PROJECT
4. ON PROJECT.EMP_ID = EMPLOYEE.EMP_ID;
```

Output

EMP_NAME	DEPARTMENT
Angelina	Testing
Robert	Development
Christian	Designing
Kristen	Development

4. FULL JOIN

In SQL, FULL JOIN is the result of a combination of both left and right outer join. Join tables have all the records from both tables. It puts NULL on the place of matches not found.

Syntax

```
1. SELECT table1.column1, table1.column2, table2.column1,....
2. FROM table1
3. FULL JOIN table2
4. ON table1.matching_column = table2.matching_column;
```

Query

```
1. SELECT EMPLOYEE.EMP_NAME, PROJECT.DEPARTMENT
2. FROM EMPLOYEE
3. FULL JOIN PROJECT
4. ON PROJECT.EMP_ID = EMPLOYEE.EMP_ID;
```

Output

EMP_NAME	DEPARTMENT
Angelina	Testing

Robert	Development
Christian	Designing
Kristen	Development
Russell	NULL
Marry	NULL

SQL Set Operation

The SQL Set operation is used to combine the two or more SQL SELECT statements.

Types of Set Operation

1. Union
2. UnionAll
3. Intersect
4. Minus

1. Union

- The SQL Union operation is used to combine the result of two or more SQL SELECT queries.
- In the union operation, all the number of datatype and columns must be same in both the tables on which UNION operation is being applied.
- The union operation eliminates the duplicate rows from its resultset.

Syntax

1. SELECT column_name FROM table1
2. UNION

3. SELECT column_name FROM table2;

Example:

The First table

ID	NAME
1	Jack
2	Harry
3	Jackson

The Second table

ID	NAME
3	Jackson
4	Stephan
5	David

Union SQL query will be:

1. SELECT * FROM First
2. UNION
3. SELECT * FROM Second;

The resultset table will look like:

ID	NAME
1	Jack
2	Harry
3	Jackson

4	Stephan
5	David

2. Union All

Union All operation is equal to the Union operation. It returns the set without removing duplication and sorting the data.

Syntax:

1. SELECT column_name FROM table1
2. UNION ALL
3. SELECT column_name FROM table2;

Example: Using the above First and Second table.

Union All query will be like:

1. SELECT * FROM First
2. UNION ALL
3. SELECT * FROM Second;

The resultset table will look like:

ID	NAME
1	Jack
2	Harry
3	Jackson
3	Jackson
4	Stephan
5	David

3. Intersect

- It is used to combine two SELECT statements. The Intersect operation returns the common rows from both the SELECT statements.
- In the Intersect operation, the number of datatype and columns must be the same.
- It has no duplicates and it arranges the data in ascending order by default.

Syntax

```
SELECT column_name FROM table1
INTERSECT
SELECT column_name FROM table2;
```

Example:

Using the above First and Second table.

Intersect query will be:

```
SELECT * FROM First
INTERSECT
SELECT * FROM Second;
```

The resultset table will look like:

ID	NAME
3	Jackson

4. Minus

- It combines the result of two SELECT statements. Minus operator is used to display the rows which are present in the first query but absent in the second query.
- It has no duplicates and data arranged in ascending order by default.

Syntax:

```
SELECT column_name FROM table1
MINUS
SELECT column_name FROM table2;
```

Example

Using the above First and Second table.

Minus query will be:

```
1. SELECT * FROM First
2. MINUS
3. SELECT * FROM Second;
```

The resultset table will look like:

ID	NAME
1	Jack
2	Harry

PL/SQL Trigger

Trigger is invoked by Oracle engine automatically whenever a specified event occurs.Trigger is stored into database and invoked repeatedly, when specific condition match.

Triggers are stored programs, which are automatically executed or fired when some event occurs.

Triggers are written to be executed in response to any of the following events.

- A database manipulation (DML) statement (DELETE, INSERT, or UPDATE).
- A database definition (DDL) statement (CREATE, ALTER, or DROP).
- A database operation (SERVERERROR, LOGON, LOGOFF, STARTUP, or SHUTDOWN).

Triggers could be defined on the table, view, schema, or database with which the event is associated.

Advantages of Triggers

These are the following advantages of Triggers:

- Trigger generates some derived column values automatically
- Enforces referential integrity
- Event logging and storing information on table access
- Auditing

- Synchronous replication of tables
- Imposing security authorizations
- Preventing invalid transactions

Creating a trigger:

Syntax for creating trigger:

```
1. CREATE [OR REPLACE ] TRIGGER trigger_name
2. {BEFORE | AFTER | INSTEAD OF }
3. {INSERT [OR] | UPDATE [OR] | DELETE}
4. [OF col_name]
5. ON table_name
6. [REFERENCING OLD AS o NEW AS n]
7. [FOR EACH ROW]
8. WHEN (condition)
9. DECLARE
10.    Declaration-statements
11. BEGIN
12.    Executable-statements
13. EXCEPTION
14.    Exception-handling-statements
15. END;
```

Here,

- CREATE [OR REPLACE] TRIGGER trigger_name: It creates or replaces an existing trigger with the trigger_name.
- {BEFORE | AFTER | INSTEAD OF} : This specifies when the trigger would be executed. The INSTEAD OF clause is used for creating trigger on a view.
- {INSERT [OR] | UPDATE [OR] | DELETE}: This specifies the DML operation.
- [OF col_name]: This specifies the column name that would be updated.
- [ON table_name]: This specifies the name of the table associated with the trigger.
- [REFERENCING OLD AS o NEW AS n]: This allows you to refer new and old values for various DML statements, like INSERT, UPDATE, and DELETE.
- [FOR EACH ROW]: This specifies a row level trigger, i.e., the trigger would be executed for each row being affected. Otherwise the trigger will execute just once when the SQL statement is executed, which is called a table level trigger.
- WHEN (condition): This provides a condition for rows for which the trigger would fire. This clause is valid only for row level triggers.

PL/SQL Trigger Example

Let's take a simple example to demonstrate the trigger. In this example, we are using the following CUSTOMERS table:

Create table and have records:

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	23	Allahabad	20000
2	Suresh	22	Kanpur	22000
3	Mahesh	24	Ghaziabad	24000
4	Chandan	25	Noida	26000
5	Alex	21	Paris	28000
6	Sunita	20	Delhi	30000

Create trigger:

Let's take a program to create a row level trigger for the CUSTOMERS table that would fire for INSERT or UPDATE or DELETE operations performed on the CUSTOMERS table. This trigger will display the salary difference between the old values and new values:

```
1.  CREATE OR REPLACE TRIGGER display_salary_changes
2.  BEFORE DELETE OR INSERT OR UPDATE ON customers
3.  FOR EACH ROW
4.  WHEN (NEW.ID > 0)
5.  DECLARE
6.    sal_diff number;
7.  BEGIN
8.    sal_diff := :NEW.salary  - :OLD.salary;
9.    dbms_output.put_line('Old salary: ' || :OLD.salary);
10.   dbms_output.put_line('New salary: ' || :NEW.salary);
11.   dbms_output.put_line('Salary difference: ' || sal_diff);
12. END;
13. /
```

After the execution of the above code at SQL Prompt, it produces the following result.

```
Trigger created.
```

Check the salary difference by procedure:

Use the following code to get the old salary, new salary and salary difference after the trigger created.

```
1.  DECLARE
2.    total_rows number(2);
3.  BEGIN
4.    UPDATE  customers
5.    SET salary = salary + 5000;
6.    IF sql%notfound THEN
7.       dbms_output.put_line('no customers updated');
8.    ELSIF sql%found THEN
9.       total_rows := sql%rowcount;
10.      dbms_output.put_line( total_rows || ' customers updated ');
11.   END IF;
12. END;
13. /
```

Output:

```
Old salary: 20000
New salary: 25000
Salary difference: 5000
Old salary: 22000
New salary: 27000
Salary difference: 5000
Old salary: 24000
New salary: 29000
Salary difference: 5000
Old salary: 26000
New salary: 31000
Salary difference: 5000
Old salary: 28000
New salary: 33000
Salary difference: 5000
Old salary: 30000
New salary: 35000
Salary difference: 5000
6 customers updated
```

Note: As many times you executed this code, the old and new both salary is incremented by 5000 and hence the salary difference is always 5000.

After the execution of above code again, you will get the following result.

```
Old salary: 25000
```

```
New salary: 30000
Salary difference: 5000
Old salary: 27000
New salary: 32000
Salary difference: 5000
Old salary: 29000
New salary: 34000
Salary difference: 5000
Old salary: 31000
New salary: 36000
Salary difference: 5000
Old salary: 33000
New salary: 38000
Salary difference: 5000
Old salary: 35000
New salary: 40000
Salary difference: 5000
6 customers updated
```

Important Points

Following are the two very important point and should be noted carefully.

- OLD and NEW references are used for record level triggers these are not avialable for table level triggers.
- If you want to query the table in the same trigger, then you should use the AFTER keyword, because triggers can query the table or change it again only after the initial changes are applied and the table is back in a consistent state.

Integrity Constraints

- Integrity constraints are a set of rules. It is used to maintain the quality of information.
- Integrity constraints ensure that the data insertion, updating, and other processes have to be performed in such a way that data integrity is not affected.
- Thus, integrity constraint is used to guard against accidental damage to the database.

SQL Integrity Constraints

Integrity Constraints are used to apply business rules for the database tables.

The constraints available in SQL are **Foreign Key**, **Not Null**, **Unique**, **Check**.
Constraints can be defined in two ways
1) The constraints can be specified immediately after the column definition. This is

called column-level definition.
2) The constraints can be specified after all the columns are defined. This is called table-level definition.

1) SQL Primary key:

This constraint defines a column or combination of columns which uniquely identifies each row in the table.

Syntax to define a Primary key at column level:

```
column name datatype [CONSTRAINT constraint_name] PRIMARY KEY
```

Syntax to define a Primary key at table level:

```
[CONSTRAINT constraint_name] PRIMARY KEY (column_name1,column_name2,..)
```

- **column_name1, column_name2** are the names of the columns which define the primary Key.
- The syntax within the bracket i.e. [CONSTRAINT constraint_name] is optional.

For Example: To create an employee table with Primary Key constraint, the query would be like.

Primary Key at column level:

```
CREATE TABLE employee
( id number(5) PRIMARY KEY,
name char(20),
dept char(10),
age number(2),
salary number(10),
location char(10)
);
```

or

```
CREATE TABLE employee
( id number(5) CONSTRAINT emp_id_pk PRIMARY KEY,
name char(20),
dept char(10),
age number(2),
salary number(10),
location char(10)
);
```

Primary Key at column level:

```
CREATE TABLE employee
( id number(5),
name char(20),
dept char(10),
age number(2),
salary number(10),
location char(10),
CONSTRAINT emp_id_pk PRIMARY KEY (id)
);
```

Primary Key at table level:

```
CREATE TABLE employee
( id number(5), NOT NULL,
name char(20),
dept char(10),
age number(2),
salary number(10),
```

```
location                                                    char(10),
ALTER TABLE employee ADD CONSTRAINT PK_EMPLOYEE_ID PRIMARY KEY
(id)
);
```

2) SQL Foreign key or Referential Integrity :

This constraint identifies any column referencing the PRIMARY KEY in another table. It establishes a relationship between two columns in the same table or between different tables. For a column to be defined as a Foreign Key, it should be a defined as a Primary Key in the table which it is referring. One or more columns can be defined as Foreign key.

Syntax to define a Foreign key at column level:

```
[CONSTRAINT              constraint_name]              REFERENCES
Referenced_Table_name(column_name)
```

Syntax to define a Foreign key at table level:

```
[CONSTRAINT constraint_name] FOREIGN KEY(column_name) REFERENCES
referenced_table_name(column_name);
```

For Example:

1) Lets use the "product" table and "order_items".

Foreign Key at column level:

```
CREATE                          TABLE                          product
(  product_id  number(5)  CONSTRAINT  pd_id_pk  PRIMARY  KEY,
product_name                                                  char(20),
supplier_name                                                 char(20),
unit_price                                                   number(10)
);
CREATE                        TABLE                         order_items
(  order_id  number(5)  CONSTRAINT  od_id_pk  PRIMARY  KEY,
product_id   number(5)   CONSTRAINT   pd_id_fk   REFERENCES,
product(product_id),
product_name                                                  char(20),
supplier_name                                                 char(20),
unit_price                                                   number(10)
);
```

Foreign Key at table level:

```
CREATE                        TABLE                         order_items
(              order_id                number(5)                    ,
product_id                                                   number(5),
product_name                                                  char(20),
supplier_name                                                 char(20),
unit_price                                                   number(10)
CONSTRAINT         od_id_pk         PRIMARY         KEY(order_id),
CONSTRAINT   pd_id_fk   FOREIGN   KEY(product_id)   REFERENCES
product(product_id)
);
```

2) If the employee table has a 'mgr_id' i.e, manager id as a foreign key which references primary key 'id' within the same table, the query would be like,

```
CREATE                          TABLE                           employee
(          id          number(5)          PRIMARY          KEY,
name                                                          char(20),
dept                                                          char(10),
```

```
age                                                    number(2),
mgr_id          number(5)          REFERENCES          employee(id),
salary                                                 number(10),
location                                               char(10)
);
```

3) SQL Not Null Constraint :

This constraint ensures all rows in the table contain a definite value for the column which is specified as not null. Which means a null value is not allowed.

Syntax to define a Not Null constraint:

```
[CONSTRAINT constraint name] NOT NULL
```

For Example: To create a employee table with Null value, the query would be like

```
CREATE                          TABLE                          employee
(                            id                             number(5),
name       char(20)       CONSTRAINT       nm_nn       NOT       NULL,
dept                                                        char(10),
age                                                         number(2),
salary                                                      number(10),
location                                                    char(10)
);
```

4) SQL Unique Key:

This constraint ensures that a column or a group of columns in each row have a distinct value. A column(s) can have a null value but the values cannot be duplicated.

Syntax to define a Unique key at column level:

```
[CONSTRAINT constraint_name] UNIQUE
```

Syntax to define a Unique key at table level:

```
[CONSTRAINT constraint_name] UNIQUE(column_name)
```

For Example: To create an employee table with Unique key, the query would be like,

Unique Key at column level:

```
CREATE                          TABLE                          employee
(           id           number(5)           PRIMARY           KEY,
name                                                        char(20),
dept                                                        char(10),
age                                                         number(2),
salary                                                      number(10),
location                    char(10)                        UNIQUE
);
```

or

```
CREATE                          TABLE                          employee
(           id           number(5)           PRIMARY           KEY,
name                                                        char(20),
dept                                                        char(10),
age                                                         number(2),
salary                                                      number(10),
location       char(10)       CONSTRAINT       loc_un       UNIQUE
);
```

Unique Key at table level:

```
CREATE TABLE employee
(  id  number(5)  PRIMARY KEY,
name  char(20),
dept  char(10),
age  number(2),
salary  number(10),
location  char(10),
CONSTRAINT  loc_un  UNIQUE(location)
);
```

5) SQL Check Constraint :

This constraint defines a business rule on a column. All the rows must satisfy this rule. The constraint can be applied for a single column or a group of columns.

Syntax to define a Check constraint:
`[CONSTRAINT constraint_name] CHECK (condition)`
For Example: In the employee table to select the gender of a person, the query would be like
Check Constraint at column level:

```
CREATE TABLE employee
(  id  number(5)  PRIMARY KEY,
name  char(20),
dept  char(10),
age  number(2),
gender  char(1)  CHECK  (gender  in  ('M','F')),
salary  number(10),
location  char(10)
);
```

Check Constraint at table level:

```
CREATE TABLE employee
(  id  number(5)  PRIMARY KEY,
name  char(20),
dept  char(10),
age  number(2),
gender  char(1),
salary  number(10),
location  char(10),
CONSTRAINT  gender_ck  CHECK  (gender  in  ('M','F'))
);
```

-

SQL SELECT NULL

First of all we should know that what null value is? Null values are used to represent missing unknown data.

There can be two conditions:

1. Where SQL is NULL
2. Where SQL is NOT NULL

If in a table, a column is optional, it is very easy to insert data in column or update an existing record without adding a value in this column. This means that field has null value.

> *Note: we should not compare null value with 0. They are not equivalent.*

Where SQL is NULL:

How to select records with null values only? (in the marks column)

There is an example of student table:

SIR_NAME	NAME	MARKS
TYAGI	SEEMA	
SINGH	RAMAN	5.5
SHARMA	AMAR	
JAISWAL	VICKY	6.2

Let's see the query to get all the records where marks is NULL:

```
1. SELECT SIR_NAME, NAME, MARKS FROM STUDENTS
2. WHERE MARKS IS NULL
```

It will return the following records:

SIR_NAME	NAME	MARKS
SHARMA	AMAR	
TYAGI	SEEMA	

Where SQL is NOT NULL:

How to select records with no null values(in marks column)? Let's see the query to get all the records where marks is NOT NULL

```
1. SELECT SIR_NAME, FIRSTNAME, MARKS FROM STUDENTS
```

2. **WHERE** MARKS **IS** NOT NULL

SIR_NAME	NAME	MARKS
SINGH	RAMAN	5.5
JAISWAL	VICKY	6.2

Views in SQL

- Views in SQL are considered as a virtual table. A view also contains rows and columns.
- To create the view, we can select the fields from one or more tables present in the database.
- A view can either have specific rows based on certain condition or all the rows of a table.

Sample table:

Student_Detail

STU_ID	NAME	ADDRESS
1	Stephan	Delhi
2	Kathrin	Noida
3	David	Ghaziabad
4	Alina	Gurugram

Student_Marks

STU_ID	NAME	MARKS	AGE
1	Stephan	97	19
2	Kathrin	86	21
3	David	74	18
4	Alina	90	20
5	John	96	18

1. Creating view

A view can be created using the **CREATE VIEW** statement. We can create a view from a single table or multiple tables.

Syntax:

```
1. CREATE VIEW view_name AS
2. SELECT column1, column2.....
3. FROM table_name
4. WHERE condition;
```

2. Creating View from a single table

In this example, we create a View named DetailsView from the table Student_Detail.

Query:

```
1. CREATE VIEW DetailsView AS
2. SELECT NAME, ADDRESS
3. FROM Student_Details
4. WHERE STU_ID < 4;
```

Just like table query, we can query the view to view the data.

```
1. SELECT * FROM DetailsView;
```

Output:

NAME	ADDRESS
Stephan	Delhi
Kathrin	Noida
David	Ghaziabad

3. Creating View from multiple tables

View from multiple tables can be created by simply include multiple tables in the SELECT statement.

In the given example, a view is created named MarksView from two tables Student_Detail and Student_Marks.

Query:

```
1. CREATE VIEW MarksView AS
2. SELECT Student_Detail.NAME, Student_Detail.ADDRESS, Student_Marks.MARKS
3. FROM Student_Detail, Student_Mark
4. WHERE Student_Detail.NAME = Student_Marks.NAME;
```

To display data of View MarksView:

```
1. SELECT * FROM MarksView;
```

NAME	ADDRESS	MARKS
Stephan	Delhi	97
Kathrin	Noida	86
David	Ghaziabad	74
Alina	Gurugram	90

4. Deleting View

A view can be deleted using the Drop View statement.

Syntax

1. DROP VIEW view_name;

Example:

If we want to delete the View **MarksView**, we can do this as:

1. DROP VIEW MarksView

SQL Index

- Indexes are special lookup tables. It is used to retrieve data from the database very fast.
- An Index is used to speed up select queries and where clauses. But it shows down the data input with insert and update statements. Indexes can be created or dropped without affecting the data.
- An index in a database is just like an index in the back of a book.
- **For example:** When you reference all pages in a book that discusses a certain topic, you first have to refer to the index, which alphabetically lists all the topics and then referred to one or more specific page numbers.

1. Create Index statement

It is used to create an index on a table. It allows duplicate value.

Syntax

```
1. CREATE INDEX index_name
2. ON table_name (column1, column2, ...);
```

Example

```
1. CREATE INDEX idx_name
2. ON Persons (LastName, FirstName);
```

2. Unique Index statement

It is used to create a unique index on a table. It does not allow duplicate value.

Syntax

```
1. CREATE UNIQUE INDEX index_name
2. ON table_name (column1, column2, ...);
```

Example

```
1. CREATE UNIQUE INDEX websites_idx
2. ON websites (site_name);
```

3. Drop Index Statement

It is used to delete an index in a table.

Syntax

```
1. DROP INDEX index_name;
```

Example

```
1. DROP INDEX websites_idx;
```