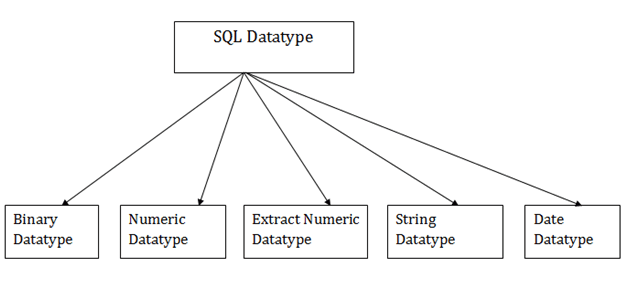
**Unit 4**

**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 Commands**

* 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 Commands**

There are five types of SQL commands: DDL, DML, DCL, TCL, and DQL.



**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
2. (col1, col2, col3,.... col N)
3. VALUES (value1, value2, value3, .... valueN);

Or

1. INSERT INTO TABLE\_NAME
2. 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 CONDITION]

**For example:**

1. UPDATE students
2. SET User\_Name = 'Sonoo'
3. 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
2. 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
2. WHERE AGE = 25;
3. 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
2. WHERE AGE = 25;
3. 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:**

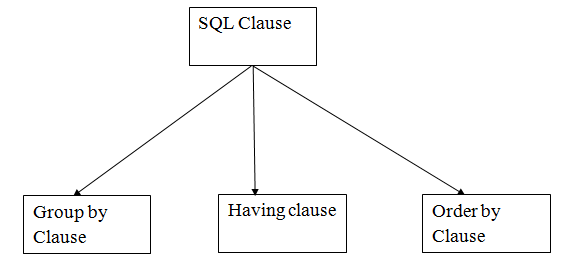
1. SELECT expressions
2. FROM TABLES
3. WHERE conditions;

**For example:**

1. SELECT emp\_name
2. FROM employee
3. WHERE age > 20;

**4.3 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:**

1. SELECT column1, column2
2. FROM table\_name
3. WHERE conditions
4. GROUP BY column1, column2
5. HAVING conditions
6. ORDER BY column1, column2;

**Example:**

1. SELECT COMPANY, COUNT(\*)
2. FROM PRODUCT\_MAST
3. GROUP BY COMPANY
4. 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:**

1. SELECT column1, column2
2. FROM table\_name
3. WHERE condition
4. 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;

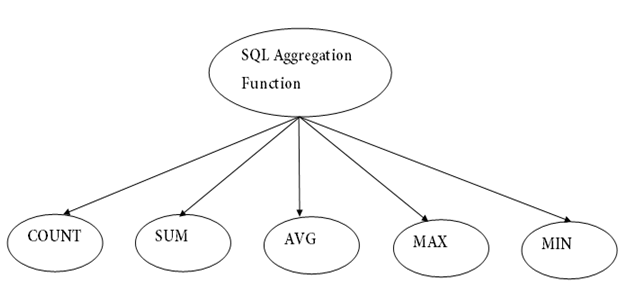
**Output:**

|  |  |  |
| --- | --- | --- |
| **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**

1. SELECT table1.column1, table1.column2, table2.column1,....
2. FROM table1
3. INNER JOIN table2
4. ON table1.matching\_column = table2.matching\_column;

**Query**

1. SELECT EMPLOYEE.EMP\_NAME, PROJECT.DEPARTMENT
2. FROM EMPLOYEE
3. INNER JOIN PROJECT
4. 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**

1. 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**

1. SELECT column\_name FROM table1
2. INTERSECT
3. SELECT column\_name FROM table2;

**Example:**

**Using the above First and Second table.**

Intersect query will be:

1. SELECT \* FROM First
2. INTERSECT
3. 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:**

1. SELECT column\_name FROM table1
2. MINUS
3. SELECT column\_name FROM table2;

**Example**

**Using the above First and Second table.**

Minus query will be:

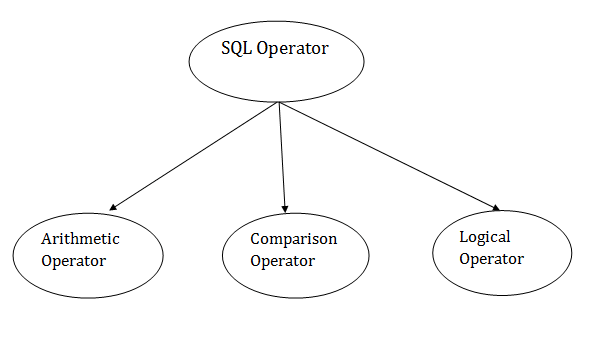
1. SELECT \* FROM First
2. MINUS
3. SELECT \* FROM Second;

The resultset table will look like:

|  |  |
| --- | --- |
| **ID** | **NAME** |
| 1 | Jack |
| 2 | Harry |

**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. |

**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;

**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 |