**DATA DEFINITION IN SQL**

**Data definition language (DDL)** refers to the set of SQL commands that can create and manipulate the structures of a database. DDL statements are used to create, change, and remove objects including indexes, triggers, tables, and views. Common DDL statements include:

* CREATE (generates a new table)
* ALTER (alters table)
* DROP (removes a table from the database)

**BASIC STRUCTURE OF SQL QUERIES**

The basic structure of an SQL query consists of three clauses: **select**, **from**, and **where**. The query takes as its input the relations listed in the from clause, operates on them as specified in the where and select clauses, and then produces a relation as the result.

* The **SELECT** statement is used to query the database and retrieve the fields that you specify. You can select as many fields (column names) as you want, or use the asterisk symbol "\*" to select all fields.
* The **FROM** statement specifies the table names that will be queried to retrieve the desired data.
* The **WHERE** clause (optional) specifies which data values or rows will be returned or displayed, based on the criteria you specify.
* The **GROUP BY** clause (optional) organizes data into groups.
* The **ORDER BY** clause (optional) sorts the data by the field specified.

**LIKE condition**

The LIKE operator can be used in the conditional selection of the where clause. Like is a very powerful operator that allows you to select only rows that are "like" what you specify. The percent sign "%" can be used as a wild card to match any possible character that might appear before or after the characters specified. For example:

**SELECT name**

**FROM members**

**WHERE name LIKE 'Mar%'**

Will select all names starting with "Mar" such as "Mark, Mary and Margaret" .

**GROUP BY clause**

The GROUP BY clause will gather all of the rows together that contain data in the field(s) and will allow aggregate functions to be performed on the one or more columns.

SELECT max(age), city, name, address

FROM members

GROUP BY city

This query will select the maximum age for the members in each unique city. Basically, the age for the person who is oldest in each city will be displayed. Their name, address and city will be returned.

**ORDER BY clause**

ORDER BY is an optional clause which will allow you to display the results of your query in a sorted order -- either ascending (ASC - Default) or descending (DESC) based on the fields that you specify to order by. If you would like to order based on multiple columns, you must seperate the columns with commas.

SELECT name, city, age

FROM members

ORDER by city, age DESC

SET Operations in SQL

SQL supports few Set operations which can be performed on the table data. These are used to get meaningful results from data stored in the table, under different special conditions.

* UNION
* UNION ALL
* INTERSECT
* MINUS

## UNION Operation

**UNION** is used to combine the results of two or more SELECT statements. However it will eliminate duplicate rows from its resultset. In case of union, number of columns and datatype must be same in both the tables, on which UNION operation is being applied.



SELECT \* FROM table1

UNION

SELECT \* FROM table2;

## UNION ALL

This operation is similar to Union. But it also shows the duplicate rows.



SELECT \* FROM table1

UNION ALL

SELECT \* FROM table2;

## INTERSECT

Intersect operation is used to combine two SELECT statements, but it only retuns the records which are common from both SELECT statements. In case of **Intersect** the number of columns and datatype must be same.



SELECT \* FROM First

INTERSECT

SELECT \* FROM Second;

## MINUS

The Minus operation combines results of two SELECT statements and return only those in the final result, which belongs to the first set of the result.



SELECT \* FROM First

MINUS

SELECT \* FROM Second;

## SQL aggregate functions

An aggregate function allows you to perform a calculation on a set of values to return a single scalar value.

The following are the most commonly used SQL aggregate functions:

* [AVG](http://www.sqltutorial.org/sql-avg.aspx)– calculates the average of a set of values.
* [COUNT](http://www.sqltutorial.org/sql-count.aspx)– counts rows in a specified table or view.
* [MIN](http://www.sqltutorial.org/sql-min-max.aspx)– gets the minimum value in a set of values.
* [MAX](https://zentut.com/sql-tutorial/sql-max/)– gets the maximum value in a set of values.
* [SUM](http://www.sqltutorial.org/sql-sum.aspx)– calculates the sum of values.

Nested Subqueries

A subquery, also known as a nested query or subselect, is a [SELECT](https://www.tutorialrepublic.com/sql-tutorial/sql-select-statement.php) query embedded within the [WHERE](https://www.tutorialrepublic.com/sql-tutorial/sql-where-clause.php)or [HAVING](https://www.tutorialrepublic.com/sql-tutorial/sql-having-clause.php) clause of another SQL query. The data returned by the subquery is used by the outer statement in the same way a literal value would be used.

Subqueries provide an easy and efficient way to handle the queries that depend on the results from another query. They are almost identical to the normal SELECT statements, but there are few restrictions. The most important ones are listed below:

* A subquery must always appear within parentheses.
* A subquery must return only one column. This means you cannot use SELECT \* in a subquery unless the table you are referring has only one column. You may use a subquery that returns multiple columns, if the purpose is row comparison.
* You can only use subqueries that return more than one row with multiple value operators, such as the [IN](https://www.tutorialrepublic.com/sql-tutorial/sql-in-between-operators.php) or [NOT IN](https://www.tutorialrepublic.com/sql-tutorial/sql-in-between-operators.php) operator.
* A subquery cannot be a [UNION](https://www.tutorialrepublic.com/sql-tutorial/sql-union-operation.php). Only a single SELECT statement is allowed.

Subqueries are most frequently used with the [SELECT](https://www.tutorialrepublic.com/sql-tutorial/sql-select-statement.php) statement, however you can use them within a [INSERT](https://www.tutorialrepublic.com/sql-tutorial/sql-insert-statement.php), [UPDATE](https://www.tutorialrepublic.com/sql-tutorial/sql-update-statement.php), or [DELETE](https://www.tutorialrepublic.com/sql-tutorial/sql-delete-statement.php) statement as well, or inside another subquery.

## Subqueries with the SELECT Statement

The following statement will return the details of only those customers whose order value in the *orders*table is more than 5000 dollar. Also note that we've used the keyword [DISTINCT](https://www.tutorialrepublic.com/sql-tutorial/sql-distinct-clause.php) in our subquery to eliminate the duplicate *cust\_id* values from the result set.

**SELECT \* FROM customers**

**WHERE cust\_id IN (SELECT DISTINCT cust\_id FROM orders**

 **WHERE order\_value > 5000);**

## Subqueries with the INSERT Statement

Subqueries can also be used with INSERT statements. Here's an example:

INSERT INTO premium\_customers

SELECT \* FROM customers

WHERE cust\_id IN (SELECT DISTINCT cust\_id FROM orders

 WHERE order\_value > 5000);

## Subqueries with the UPDATE Statement

You can also use the subqueries in conjunction with the UPDATE statement to update the single or multiple columns in a table, as follow:

UPDATE orders

SET order\_value = order\_value + 10

WHERE cust\_id IN (SELECT cust\_id FROM customers

 WHERE postal\_code = 75016);

## Subqueries with the DELETE Statement

Similarly, you can use the subqueries in conjunction with the DELETE statement to delete the single or multiple rows in a table, as follow:

**DELETE FROM orders**

**WHERE order\_id IN (SELECT order\_id FROM order\_details**

 **WHERE product\_id = 5);**

**JOIN RELATIONS**

A JOIN clause is used to combine rows from two or more tables, based on a related column between them.

## Different Types of SQL JOINs

Here are the different types of the JOINs in SQL:

* **(INNER) JOIN**: Returns records that have matching values in both tables
* **LEFT (OUTER) JOIN**: Returns all records from the left table, and the matched records from the right table
* **RIGHT (OUTER) JOIN**: Returns all records from the right table, and the matched records from the left table
* **FULL (OUTER) JOIN**: Returns all records when there is a match in either left or right table

  

 