Contents

[**What is Data?** 1](#_Toc34530685)

# **What is Data?**

In simple words data can be facts related to any object in consideration.

For example your name, age, height, weight, etc are some data related to you.

A picture , image , file , pdf etc can also be considered data.

Examples: MySQL, SQL Server, MS Access, Oracle, Sybase, Informix, Postgres, and other database systems.

What is SQL?

* SQL stands for Structured Query Language
* SQL lets you access and manipulate databases
* SQL keywords are NOT case sensitive: select is the same as SELECT
* Some database systems require a semicolon at the end of each SQL statement.

What Can SQL do?

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

## RDBMS

RDBMS stands for Relational Database Management System.The data in RDBMS is stored in database objects called tables. A table is a collection of related data entries and it consists of columns and rows.

These SQL commands are mainly categorized into four categories as:

1. **DDL – Data Definition Language**
2. **DQl – Data Query Language**
3. **DML – Data Manipulation Language**
4. **DCL – Data Control Language**
5. **TCL – Transaction Control Language.**
6. **DDL(Data Definition Language) :**DDL or Data Definition Language actually consists of the SQL commands that can be used to define the database schema. It simply deals with descriptions of the database schema and is used to create and modify the structure of database objects in the database.

**Examples of DDL commands:**

* [**CREATE**](https://www.geeksforgeeks.org/sql-create/) – is used to create the database or its objects (like table, index, function, views, store procedure and triggers).
* [**DROP**](https://www.geeksforgeeks.org/sql-drop-truncate/) – is used to delete objects from the database.
* [**ALTER**](https://www.geeksforgeeks.org/sql-alter-add-drop-modify/)-is used to alter the structure of the database.
* [**TRUNCATE**](https://www.geeksforgeeks.org/sql-drop-truncate/)–is used to remove all records from a table, including all spaces allocated for the records are removed.
* [**COMMENT**](https://www.geeksforgeeks.org/sql-comments/) –is used to add comments to the data dictionary.
* [**RENAME**](https://www.geeksforgeeks.org/sql-alter-rename/)–is used to rename an object existing in the database.

1. **DQL (Data Query Language) :**

DML statements are used for performing queries on the data within schema objects. The purpose of DQL Command is to get some schema relation based on the query passed to it.

**Example of DQL:**

* + [**SELECT**](https://www.geeksforgeeks.org/sql-select-clause/) – is used to retrieve data from the a database.

1. **DML(Data Manipulation Language) :**The SQL commands that deals with the manipulation of data present in the database belong to DML or Data Manipulation Language and this includes most of the SQL statements.

**Examples of DML:**

* 1. [**INSERT**](https://www.geeksforgeeks.org/sql-insert-statement/) – is used to insert data into a table.
  2. [**UPDATE**](https://www.geeksforgeeks.org/sql-update-statement/) – is used to update existing data within a table.
  3. [**DELETE**](https://www.geeksforgeeks.org/sql-delete-statement/) – is used to delete records from a database table.

1. **DCL(Data Control Language) :**DCL includes commands such as GRANT and REVOKE which mainly deals with the rights, permissions and other controls of the database system.

**Examples of DCL commands:**

* 1. **GRANT**-gives user’s access privileges to database.
  2. **REVOKE**-withdraw user’s access privileges given by using the GRANT command.

1. **TCL(transaction Control Language) :**TCL commands deals with the [transaction within the database](https://www.geeksforgeeks.org/sql-transactions/).

**Examples of TCL commands:**

* **COMMIT**– commits a Transaction.
* [**ROLLBACK**](https://www.geeksforgeeks.org/sql-transactions/)– rollbacks a transaction in case of any error occurs.
* **SAVEPOINT**–sets a savepoint within a transaction.
* **SET TRANSACTION**–specify characteristics for the transaction.

## SELECT Statement

|  |
| --- |
| SELECT column1, column2, ... FROM table\_name;  SELECT \* FROM table\_name; |

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

|  |
| --- |
| SELECT DISTINCT column1, column2, ... FROM table\_name; |

The WHERE clause is used to extract only those records that fulfill a specified condition.

|  |
| --- |
| SELECT column1, column2, ...FROM table\_name WHERE condition;  SELECT \* FROM Customers WHERE Country='Mexico';  SELECT \* FROM Customers WHERE CustomerID=1; |

SQL requires single quotes around text values (most database systems will also allow double quotes).

However, numeric fields should not be enclosed in quotes

The following operators can be used in the WHERE clause:

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** |  |
| = | Equal |  |
| > | Greater than |  |
| < | Less than |  |
| >= | Greater than or equal |  |
| <= | Less than or equal |  |
| <> | Not equal. **Note:** In some versions of SQL this operator may be written as != |  |
| BETWEEN | Between a certain range |  |
| LIKE | Search for a pattern |  |
| IN | To specify multiple possible values for a column |  |

The WHERE clause can be combined with AND, OR, and NOT operators.

The AND and OR operators are used to filter records based on more than one condition:

* The AND operator displays a record if all the conditions separated by AND are TRUE.
* The OR operator displays a record if any of the conditions separated by OR is TRUE.

The NOT operator displays a record if the condition(s) is NOT TRUE.

|  |
| --- |
| SELECT column1, column2, ... FROM table\_name WHERE condition1 AND condition2 AND condition3 ...; |
| SELECT column1, column2, ... FROM table\_name WHERE condition1 OR condition2 OR condition3 ...; |
| SELECT column1, column2, ... FROM table\_name WHERE NOT condition; |

The ORDER BY keyword is used to sort the result-set in ascending or descending order.

The ORDER BY keyword sorts the records in ascending order by default. To sort the records in descending order, use the DESC keyword.

|  |
| --- |
| SELECT column1, column2, ... FROM table\_name ORDER BY column1, column2, ... ASC|DESC;  SELECT \* FROM Customers ORDER BY Country DESC;  SELECT \* FROM Customers ORDER BY Country, CustomerName; |

SQL statement selects all customers from the "Customers" table, sorted by the "Country" and the "CustomerName" column. This means that it orders by Country, but if some rows have the same Country, it orders them by CustomerName:

following SQL statement selects all customers from the "Customers" table, sorted ascending by the "Country" and descending by the "CustomerName" column:

|  |
| --- |
| SELECT \* FROM Customers ORDER BY Country ASC, CustomerName DESC; |

## The SQL INSERT INTO Statement

The INSERT INTO statement is used to insert new records in a table.

It is possible to write the INSERT INTO statement in two ways.

The first way specifies both the column names and the values to be inserted:

|  |
| --- |
| INSERT INTO table\_name (column1, column2, column3, ...) VALUES (value1, value2, value3, ...); |

Columns which are not specified above will have NULL by default

If you are adding values for all the columns of the table, you do not need to specify the column names in the SQL query. However, make sure the order of the values is in the same order as the columns in the table.

|  |
| --- |
| INSERT INTO table\_name VALUES (value1, value2, value3, ...); |

## What is a NULL Value?

A field with a NULL value is a field with no value.

If a field in a table is optional, it is possible to insert a new record or update a record without adding a value to this field. Then, the field will be saved with a NULL value.

**Note:** A NULL value is different from a zero value or a field that contains spaces. A field with a NULL value is one that has been left blank during record creation!

It is not possible to test for NULL values with comparison operators, such as =, <, or <>.

We will have to use the IS NULL and IS NOT NULL operators instead.

|  |
| --- |
| SELECT column\_namesFROM table\_name WHERE column\_name IS NULL; |
| SELECT column\_namesFROM table\_name WHERE column\_name IS NOT NULL; |

## The SQL UPDATE Statement

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

|  |
| --- |
| UPDATE table\_name SET column1 = value1, column2 = value2, ... WHERE condition; |

**Note:** Be careful when updating records in a table! Notice the WHERE clause in the UPDATE statement. The WHERE clause specifies which record(s) that should be updated. If you omit the WHERE clause, all records in the table will be updated!

|  |
| --- |
| UPDATE Customers SET ContactName = 'Alfred Schmidt', City= 'Frankfurt' WHERE CustomerID = 1; |

## The SQL DELETE Statement

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

|  |
| --- |
| DELETE FROM table\_name WHERE condition; |

**Note:** Be careful when deleting records in a table! Notice the WHERE clause in the DELETE statement. The WHERE clause specifies which record(s) should be deleted. If you omit the WHERE clause, all records in the table will be deleted!

## The SQL SELECT TOP Clause

The SELECT TOP clause is used to specify the number of records to return.

The SELECT TOP clause is useful on large tables with thousands of records. Returning a large number of records can impact performance.

**Note:** Not all database systems support the SELECT TOP clause. MySQL supports the LIMIT clause to select a limited number of records, while Oracle uses ROWNUM.

|  |
| --- |
| **SQL Server / MS Access Syntax:** SELECT TOP number|*percent* column\_name(s) FROM table\_nameWHERE condition; |
| **MySQL Syntax:**  SELECT column\_name(s) FROM table\_nameWHERE condition LIMIT number; |
| **Oracle Syntax:**  SELECT column\_name(s) FROM table\_name WHERE ROWNUM <= number; |

## The SQL MIN() and MAX() Functions

The MIN() function returns the smallest value of the selected column.

The MAX() function returns the largest value of the selected column.

|  |
| --- |
| SELECT MIN(column\_name) FROM table\_name WHERE condition; |
| SELECT MAX(column\_name) FROM table\_name WHERE condition; |

## The SQL COUNT(), AVG() and SUM() Functions

The COUNT() function returns the number of rows that matches a specified criterion.

The AVG() function returns the average value of a numeric column.

The SUM() function returns the total sum of a numeric column.

|  |
| --- |
| SELECT COUNT(column\_name) FROM table\_name WHERE condition; |
| SELECT AVG(column\_name) FROM table\_name WHERE condition; |
| SELECT SUM(column\_name) FROM table\_name WHERE condition; |

**Note:** NULL values are ignored.

## The SQL LIKE Operator

The LIKE operator is used in a WHERE clause to search for a specified pattern in a column.

There are two wildcards often used in conjunction with the LIKE operator:

* % - The percent sign represents zero, one, or multiple characters
* \_ - The underscore represents a single character

|  |
| --- |
| SELECT column1, column2, ... FROM table\_name WHERE columnN LIKE pattern; |

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

|  |  |
| --- | --- |
| **LIKE Operator** | **Description** |
| WHERE CustomerName LIKE 'a%' | Finds any values that start with "a" |
| WHERE CustomerName LIKE '%a' | Finds any values that end with "a" |
| WHERE CustomerName LIKE '%or%' | Finds any values that have "or" in any position |
| WHERE CustomerName LIKE '\_r%' | Finds any values that have "r" in the second position |
| WHERE CustomerName LIKE 'a\_%' | Finds any values that start with "a" and are at least 2 characters in length |
| WHERE CustomerName LIKE 'a\_\_%' | Finds any values that start with "a" and are at least 3 characters in length |
| WHERE ContactName LIKE 'a%o' | Finds any values that start with "a" and ends with "o" |

### **Wildcard Characters in SQL Server**

|  |  |  |
| --- | --- | --- |
| **Symbol** | **Description** | **Example** |
| % | Represents zero or more characters | bl% finds bl, black, blue, and blob |
| \_ | Represents a single character | h\_t finds hot, hat, and hit |
| [] | Represents any single character within the brackets | h[oa]t finds hot and hat, but not hit |
| ^ | Represents any character not in the brackets | h[^oa]t finds hit, but not hot and hat |
| - | Represents a range of characters | c[a-b]t finds cat and cbt |

## Using the [charlist] Wildcard

The following SQL statement selects all customers with a City starting with "b", "s", or "p":

|  |
| --- |
| SELECT \* FROM Customers WHERE City LIKE '[bsp]%'; |

The following SQL statement selects all customers with a City starting with "a", "b", or "c":

|  |
| --- |
| SELECT \* FROM Customers WHERE City LIKE '[a-c]%'; |

 following SQL statements select all customers with a City NOT starting with "b", "s", or "p":

|  |
| --- |
| SELECT \* FROM Customers WHERE City LIKE '[!bsp]%'; |
| SELECT \* FROM Customers WHERE City NOT LIKE '[bsp]%'; |

## The SQL IN Operator

The IN operator allows you to specify multiple values in a WHERE clause.

|  |
| --- |
| SELECT column\_name(s) FROM table\_name WHERE column\_name IN (value1, value2, ...); |
| SELECT column\_name(s) FROM table\_name WHERE column\_name IN (*SELECT* STATEMENT); |

Examples

|  |
| --- |
| SELECT \* FROM Customers WHERE Country NOT IN ('Germany', 'France', 'UK'); |
| SELECT \* FROM Customers WHERE Country IN (SELECT Country FROM Suppliers); |

## The SQL BETWEEN Operator

The BETWEEN operator selects values within a given range. The values can be numbers, text, or dates.

The BETWEEN operator is inclusive: begin and end values are included.

|  |
| --- |
| SELECT column\_name(s) FROM table\_name WHERE column\_name BETWEEN value1 AND value2; |

To display the products outside the range of the previous example, use NOT BETWEEN:

|  |
| --- |
| SELECT \* FROM Products WHERE Price NOT BETWEEN 10 AND 20; |

following SQL statement selects all products with a price BETWEEN 10 and 20. In addition; do not show products with a CategoryID of 1,2, or 3:

|  |
| --- |
| SELECT \* FROM Products WHERE Price BETWEEN 10 AND 20 AND CategoryID NOT IN (1,2,3); |

following SQL statement selects all products with a ProductName BETWEEN Carnarvon Tigers and Mozzarella di Giovanni:

|  |
| --- |
| SELECT \* FROM Products WHERE ProductName BETWEEN 'Carnarvon Tigers' AND 'Mozzarella di Giovanni' ORDER BY ProductName; |

following SQL statement selects all orders with an OrderDate BETWEEN '01-July-1996' and '31-July-1996':

|  |
| --- |
| SELECT \* FROM Orders WHERE OrderDate BETWEEN #01/07/1996# AND #31/07/1996#; |
| SELECT \* FROM Orders WHERE OrderDate BETWEEN '1996-07-01' AND '1996-07-31'; |

## SQL Aliases

SQL aliases are used to give a table, or a column in a table, a temporary name.

Aliases are often used to make column names more readable.

An alias only exists for the duration of the query.

|  |
| --- |
| SELECT column\_name AS alias\_name FROM table\_name; |
| SELECT column\_name(s) FROM table\_name AS alias\_name; |

## SQL JOIN

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

      

## SQL INNER JOIN Keyword

The INNER JOIN keyword selects records that have matching values in both tables.

|  |
| --- |
| SELECT column\_name(s) FROM table1 INNER JOIN table2ON table1.column\_name = table2.column\_name; |

## JOIN Three Tables

The following SQL statement selects all orders with customer and shipper information:

|  |
| --- |
| SELECT Orders.OrderID, Customers.CustomerName, Shippers.ShipperName FROM ((Orders INNER JOIN Customers ON Orders.CustomerID = Customers.CustomerID) INNER JOIN Shippers ON Orders.ShipperID = Shippers.ShipperID); |

## SQL LEFT JOIN Keyword

The LEFT JOIN keyword returns all records from the left table (table1), and the matched records from the right table (table2). The result is NULL from the right side, if there is no match.

|  |
| --- |
| SELECT column\_name(s) FROM table1 LEFT JOIN table2ON table1.column\_name = table2.column\_name; |

## SQL RIGHT JOIN Keyword

The RIGHT JOIN keyword returns all records from the right table (table2), and the matched records from the left table (table1). The result is NULL from the left side, when there is no match.

|  |
| --- |
| SELECT column\_name(s) FROM table1 RIGHT JOIN table2ON table1.column\_name = table2.column\_name; |

## SQL FULL OUTER JOIN Keyword

The FULL OUTER JOIN keyword returns all records when there is a match in left (table1) or right (table2) table records.

**Note:** FULL OUTER JOIN can potentially return very large result-sets!

**Tip:** FULL OUTER JOIN and FULL JOIN are the same.

|  |
| --- |
| SELECT column\_name(s) FROM table1 FULL OUTER JOIN table2ON table1.column\_name = table2.column\_nameWHERE condition; |

## SQL Self JOIN

A self JOIN is a regular join, but the table is joined with itself.

|  |
| --- |
| SELECT column\_name(s) FROM table1 T1, table1 T2 WHERE condition; |

T1 and T2 are different table aliases for the same table.

## The SQL UNION Operator

The UNION operator is used to combine the result-set of two or more SELECT statements.

* Each SELECT statement within UNION must have the same number of columns
* The columns must also have similar data types
* The columns in each SELECT statement must also be in the same order

|  |
| --- |
| SELECT column\_name(s) FROM table1 UNION SELECT column\_name(s) FROM table2; |

The UNION operator selects only distinct values by default. To allow duplicate values, use UNION ALL:

|  |
| --- |
| SELECT column\_name(s) FROM table1 UNION ALL SELECT column\_name(s) FROM table2; |

## The SQL GROUP BY Statement

The GROUP BY statement groups rows that have the same values into summary rows, like "find the number of customers in each country".

The GROUP BY statement is often used with aggregate functions (COUNT, MAX, MIN, SUM, AVG) to group the result-set by one or more columns.

|  |
| --- |
| SELECT column\_name(s) FROM table\_name WHERE condition GROUP BY column\_name(s)ORDER BY column\_name(s); |

following SQL statement lists the number of customers in each country:

|  |
| --- |
| SELECT COUNT(CustomerID), Country FROM Customers GROUP BY Country; |

following SQL statement lists the number of orders sent by each shipper:

|  |
| --- |
| SELECT Shippers.ShipperName, COUNT(Orders.OrderID) AS NumberOfOrders FROM Orders LEFT JOIN Shippers ON Orders.ShipperID = Shippers.ShipperID GROUP BY ShipperName; |

## The SQL HAVING Clause

The HAVING clause was added to SQL because the WHERE keyword could not be used with aggregate functions.

|  |
| --- |
| SELECT column\_name(s) FROM table\_name WHERE condition GROUP BY column\_name(s)HAVING conditionORDER BY column\_name(s); |

 following SQL statement lists the number of customers in each country, sorted high to low (Only include countries with more than 5 customers)

|  |
| --- |
| SELECT COUNT(CustomerID), Country FROM Customers GROUP BY Country HAVING COUNT(CustomerID) > 5 ORDER BY COUNT(CustomerID) DESC; |

following SQL statement lists if the employees "Davolio" or "Fuller" have registered more than 25 orders:

|  |
| --- |
| SELECT Employees.LastName, COUNT(Orders.OrderID) AS NumberOfOrders FROM Orders INNER JOIN Employees ON Orders.EmployeeID = Employees.EmployeeID WHERE LastName = 'Davolio' OR LastName = 'Fuller' GROUP BY LastName HAVING COUNT(Orders.OrderID) > 25; |

## The SQL EXISTS Operator

The EXISTS operator is used to test for the existence of any record in a subquery.

The EXISTS operator returns true if the subquery returns one or more records.

|  |
| --- |
| SELECT column\_name(s) FROM table\_name WHERE EXISTS (SELECT column\_name FROM table\_name WHERE condition); |

The following SQL statement returns TRUE and lists the suppliers with a product price less than 20:

|  |
| --- |
| SELECT SupplierName FROM Suppliers WHERE EXISTS (SELECT ProductName FROM Products WHERE Products.SupplierID = Suppliers.supplierID AND Price < 20); |

## The SQL ANY and ALL Operators

The ANY and ALL operators are used with a WHERE or HAVING clause.

The ANY operator returns true if any of the subquery values meet the condition.

The ALL operator returns true if all of the subquery values meet the condition.

|  |
| --- |
| SELECT column\_name(s) FROM table\_name WHERE column\_name operator ANY (SELECT column\_name FROM table\_name WHERE condition); |

**Note:** The operator must be a standard comparison operator (=, <>, !=, >, >=, <, or <=).

|  |
| --- |
| SELECT column\_name(s) FROM table\_name WHERE column\_name operator ALL (SELECT column\_name FROM table\_name WHERE condition); |

The following SQL statement returns TRUE and lists the product names if it finds ANY records in the OrderDetails table that quantity = 10:

|  |
| --- |
| SELECT ProductName FROM Products WHERE ProductID = ANY (SELECT ProductID FROM OrderDetails WHERE Quantity = 10); |

## The SQL SELECT INTO Statement

The SELECT INTO statement copies data from one table into a new table.

|  |
| --- |
| SELECT \* INTO newtable [IN externaldb] FROM oldtableWHERE condition; |

Copy only some columns into a new table:

|  |
| --- |
| SELECT column1, column2, column3, ... INTO newtable [IN externaldb] FROM oldtableWHERE condition; |

The new table will be created with the column-names and types as defined in the old table. You can create new column names using the AS clause.

The following SQL statement creates a backup copy of Customers:

|  |
| --- |
| SELECT \* INTO CustomersBackup2017 FROM Customers; |

The following SQL statement uses the IN clause to copy the table into a new table in another database:

|  |
| --- |
| SELECT \* INTO CustomersBackup2017 IN 'Backup.mdb' FROM Customers; |

The following SQL statement copies data from more than one table into a new table:

|  |
| --- |
| SELECT Customers.CustomerName, Orders.OrderID INTO CustomersOrderBackup2017 FROM Customers LEFT JOIN Orders ON Customers.CustomerID = Orders.CustomerID; |

**Tip:** SELECT INTO can also be used to create a new, empty table using the schema of another. Just add a WHERE clause that causes the query to return no data:

|  |
| --- |
| SELECT \* INTO newtable FROM oldtable WHERE 1 = 0; |

## The SQL INSERT INTO SELECT Statement

The INSERT INTO SELECT statement copies data from one table and inserts it into another table.

* INSERT INTO SELECT requires that data types in source and target tables match
* The existing records in the target table are unaffected

|  |
| --- |
| INSERT INTO table2 SELECT \* FROM table1WHERE condition; |

Copy only some columns from one table into another table:

|  |
| --- |
| INSERT INTO table2 (column1, column2, column3, ...) SELECT column1, column2, column3, ... FROM table1 WHERE condition; |

The following SQL statement copies only the German suppliers into "Customers":

|  |
| --- |
| INSERT INTO Customers (CustomerName, City, Country) SELECT SupplierName, City, Country FROM Suppliers WHERE Country='Germany'; |

## The SQL CASE Statement

The CASE statement goes through conditions and returns a value when the first condition is met (like an IF-THEN-ELSE statement). So, once a condition is true, it will stop reading and return the result. If no conditions are true, it returns the value in the ELSE clause.

If there is no ELSE part and no conditions are true, it returns NULL.

|  |
| --- |
| CASE     WHEN condition1 THEN result1     WHEN condition2 THEN result2     WHEN conditionN THEN resultN     ELSE result END; |

The following SQL goes through conditions and returns a value when the first condition is met:

|  |
| --- |
| SELECT OrderID, Quantity, CASE     WHEN Quantity > 30 THEN 'The quantity is greater than 30'     WHEN Quantity = 30 THEN 'The quantity is 30'     ELSE 'The quantity is under 30' END AS QuantityText FROM OrderDetails; |

The following SQL will order the customers by City. However, if City is NULL, then order by Country:

|  |
| --- |
| SELECT CustomerName, City, Country FROM Customers ORDER BY (CASE     WHEN City IS NULL THEN Country     ELSE City END); |

## SQL IFNULL(), ISNULL(), COALESCE(), and NVL() Functions

**MySQL**

The MySQL [IFNULL()](https://www.w3schools.com/sql/func_mysql_ifnull.asp) function lets you return an alternative value if an expression is NULL:

|  |
| --- |
| SELECT ProductName, UnitPrice \* (UnitsInStock + IFNULL(UnitsOnOrder, 0)) FROM Products; |
| SELECT ProductName, UnitPrice \* (UnitsInStock + COALESCE(UnitsOnOrder, 0)) FROM Products; |

**SQL Server**

The SQL Server [ISNULL()](https://www.w3schools.com/sql/func_sqlserver_isnull.asp) function lets you return an alternative value when an expression is NULL:

|  |
| --- |
| SELECT ProductName, UnitPrice \* (UnitsInStock + ISNULL(UnitsOnOrder, 0)) FROM Products; |

**Oracle**

The Oracle NVL() function achieves the same result:

|  |
| --- |
| SELECT ProductName, UnitPrice \* (UnitsInStock + NVL(UnitsOnOrder, 0)) FROM Products; |

## SQL Comments

Comments are used to explain sections of SQL statements, or to prevent execution of SQL statements.

## Single Line Comments

Single line comments start with --.

Any text between -- and the end of the line will be ignored (will not be executed).

|  |
| --- |
| --SELECT \* FROM Customers; SELECT \* FROM Products; |

## Multi-line Comments

Multi-line comments start with /\* and end with \*/.

Any text between /\* and \*/ will be ignored.

|  |
| --- |
| /\*Select all the columns of all the records in the Customers table:\*/ SELECT \* FROM Customers; |

## The SQL CREATE DATABASE Statement

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

|  |
| --- |
| CREATE DATABASE databasename; |

**Tip:** Make sure you have admin privilege before creating any database. Once a database is created, you can check it in the list of databases with the following SQL command: SHOW DATABASES;

## The SQL DROP DATABASE Statement

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

|  |
| --- |
| DROP DATABASE databasename; |

**Note:** Be careful before dropping a database. Deleting a database will result in loss of complete information stored in the database!

## The SQL CREATE TABLE Statement

The CREATE TABLE statement is used to create a new table in a database.

|  |
| --- |
| CREATE TABLE table\_name (     column1 datatype,     column2 datatype,     column3 datatype,    .... ); |

## Create Table Using Another Table

A copy of an existing table can also be created using CREATE TABLE.

The new table gets the same column definitions. All columns or specific columns can be selected.

If you create a new table using an existing table, the new table will be filled with the existing values from the old table.

|  |
| --- |
| CREATE TABLE new\_table\_name AS     SELECT column1, column2,...     FROM existing\_table\_name     WHERE ....; |

## The SQL DROP TABLE Statement

The DROP TABLE statement is used to drop an existing table in a database.

|  |
| --- |
| DROP TABLE table\_name; |

## SQL ALTER TABLE Statement

The ALTER TABLE statement is used to add, delete, or modify columns in an existing table.

The ALTER TABLE statement is also used to add and drop various constraints on an existing table.

## ALTER TABLE - ADD Column

|  |
| --- |
| ALTER TABLE table\_name ADD column\_name datatype; |

## ALTER TABLE - DROP COLUMN

|  |
| --- |
| ALTER TABLE table\_name DROP COLUMN column\_name; |

## ALTER TABLE - ALTER/MODIFY COLUMN

|  |
| --- |
| ALTER TABLE table\_name ALTER COLUMN column\_name datatype; |

## Oracle 10G and later:

|  |
| --- |
| ALTER TABLE table\_name MODIFY column\_name datatype; |

# **SQL Constraints**

## SQL constraints are used to specify rules for data in a table.

## Constraints can be specified when the table is created with the CREATE TABLE statement, or after the table is created with the ALTER TABLE statement.

|  |
| --- |
| CREATE TABLE table\_name (     column1 datatype *constraint*,     column2 datatype *constraint*,     column3 datatype *constraint*,     .... ); |

SQL constraints are used to specify rules for the data in a table.

Constraints are used to limit the type of data that can go into a table. This ensures the accuracy and reliability of the data in the table. If there is any violation between the constraint and the data action, the action is aborted.

Constraints can be column level or table level. Column level constraints apply to a column, and table level constraints apply to the whole table.

The following constraints are commonly used in SQL:

* [**NOT NULL**](https://www.w3schools.com/sql/sql_notnull.asp) - Ensures that a column cannot have a NULL value
* [**UNIQUE**](https://www.w3schools.com/sql/sql_unique.asp) - Ensures that all values in a column are different
* [**PRIMARY KEY**](https://www.w3schools.com/sql/sql_primarykey.asp) - A combination of a NOT NULL and UNIQUE. Uniquely identifies each row in a table
* [**FOREIGN KEY**](https://www.w3schools.com/sql/sql_foreignkey.asp) - Uniquely identifies a row/record in another table
* [**CHECK**](https://www.w3schools.com/sql/sql_check.asp) - Ensures that all values in a column satisfies a specific condition
* [**DEFAULT**](https://www.w3schools.com/sql/sql_default.asp) - Sets a default value for a column when no value is specified
* [**INDEX**](https://www.w3schools.com/sql/sql_create_index.asp) - Used to create and retrieve data from the database very quickly

## SQL NOT NULL Constraint

By default, a column can hold NULL values.

The NOT NULL constraint enforces a column to NOT accept NULL values.

This enforces a field to always contain a value, which means that you cannot insert a new record, or update a record without adding a value to this field.

The following SQL ensures that the "ID", "LastName", and "FirstName" columns will NOT accept NULL values when the "Persons" table is created:

|  |
| --- |
| CREATE TABLE Persons (     ID int NOT NULL,     LastName varchar(255) NOT NULL,     FirstName varchar(255) NOT NULL,     Age int ); |

To create a NOT NULL constraint on the "Age" column when the "Persons" table is already created, use the following SQL:

|  |
| --- |
| ALTER TABLE Persons MODIFY Age int NOT NULL; |

## SQL UNIQUE Constraint

The UNIQUE constraint ensures that all values in a column are different.

Both the UNIQUE and PRIMARY KEY constraints provide a guarantee for uniqueness for a column or set of columns.

A PRIMARY KEY constraint automatically has a UNIQUE constraint.

However, you can have many UNIQUE constraints per table, but only one PRIMARY KEY constraint per table.

|  |
| --- |
| CREATE TABLE Persons (     ID int NOT NULL UNIQUE,     LastName varchar(255) NOT NULL,     FirstName varchar(255),     Age int ); |

To name a UNIQUE constraint, and to define a UNIQUE constraint on multiple columns, use the following SQL syntax:

|  |
| --- |
| CREATE TABLE Persons (     ID int NOT NULL,     LastName varchar(255) NOT NULL,     FirstName varchar(255),     Age int,     CONSTRAINT UC\_Person UNIQUE (ID,LastName) ); |

To create a UNIQUE constraint on the "ID" column when the table is already created, use the following SQL:

|  |
| --- |
| ALTER TABLE Persons ADD UNIQUE (ID); |
| ALTER TABLE Persons ADD CONSTRAINT UC\_Person UNIQUE (ID,LastName); |

To drop a UNIQUE constraint, use the following SQL:

|  |
| --- |
| ALTER TABLE Persons DROP CONSTRAINT constraint\_name; |

## SQL PRIMARY KEY Constraint

The PRIMARY KEY constraint uniquely identifies each record in a table.

Primary keys must contain UNIQUE values, and cannot contain NULL values.

A table can have only ONE primary key; and in the table, this primary key can consist of single or multiple columns (fields).

|  |
| --- |
| CREATE TABLE Persons (     ID int NOT NULL PRIMARY KEY,     LastName varchar(255) NOT NULL,     FirstName varchar(255),     Age int ); |

To allow naming of a PRIMARY KEY constraint, and for defining a PRIMARY KEY constraint on multiple columns, use the following SQL syntax:

|  |
| --- |
| CREATE TABLE Persons (     ID int NOT NULL,     LastName varchar(255) NOT NULL,     FirstName varchar(255),     Age int,     CONSTRAINT PK\_Person PRIMARY KEY (ID,LastName) ); |

To create a PRIMARY KEY constraint on the "ID" column when the table is already created, use the following SQL:

|  |
| --- |
| ALTER TABLE Persons ADD CONSTRAINT PK\_Person PRIMARY KEY (ID,LastName); |

To drop a PRIMARY KEY constraint, use the following SQL:

|  |
| --- |
| AALTER TABLE Persons DROP CONSTRAINT PK\_Person; |

## SQL FOREIGN KEY Constraint

A FOREIGN KEY is a key used to link two tables together.

A FOREIGN KEY is a field (or collection of fields) in one table that refers to the PRIMARY KEY in another table.

The table containing the foreign key is called the child table, and the table containing the candidate key is called the referenced or parent table.

|  |
| --- |
| CREATE TABLE Orders (     OrderID int NOT NULL PRIMARY KEY,     OrderNumber int NOT NULL,     PersonID int FOREIGN KEY REFERENCES Persons(PersonID) ); |

To create a FOREIGN KEY constraint on the "PersonID" column when the "Orders" table is already created, use the following SQL:

|  |
| --- |
| ALTER TABLE Orders ADD CONSTRAINT FK\_PersonOrder FOREIGN KEY (PersonID) REFERENCES Persons(PersonID); |

To drop a FOREIGN KEY constraint, use the following SQL:

|  |
| --- |
| ALTER TABLE Orders DROP CONSTRAINT FK\_PersonOrder; |

## SQL CHECK Constraint

The CHECK constraint is used to limit the value range that can be placed in a column.

If you define a CHECK constraint on a single column it allows only certain values for this column.

If you define a CHECK constraint on a table it can limit the values in certain columns based on values in other columns in the row.

The following SQL creates a CHECK constraint on the "Age" column when the "Persons" table is created. The CHECK constraint ensures that the age of a person must be 18, or older:

|  |
| --- |
| CREATE TABLE Persons (     ID int NOT NULL,     LastName varchar(255) NOT NULL,     FirstName varchar(255),     Age int CHECK (Age>=18) ); |

To allow naming of a CHECK constraint, and for defining a CHECK constraint on multiple columns, use the following SQL syntax:

|  |
| --- |
| CREATE TABLE Persons (     ID int NOT NULL,     LastName varchar(255) NOT NULL,     FirstName varchar(255),     Age int,     City varchar(255),     CONSTRAINT CHK\_Person CHECK (Age>=18 AND City='Sandnes') ); |

To create a CHECK constraint on the "Age" column when the table is already created, use the following SQL:

|  |
| --- |
| ALTER TABLE Persons ADD CONSTRAINT CHK\_PersonAge CHECK (Age>=18 AND City='Sandnes'); |

To drop a CHECK constraint, use the following SQL:

|  |
| --- |
| ALTER TABLE Persons DROP CONSTRAINT CHK\_PersonAge; |

## SQL DEFAULT Constraint

The DEFAULT constraint is used to provide a default value for a column.

The default value will be added to all new records IF no other value is specified.

|  |
| --- |
| CREATE TABLE Orders (     ID int NOT NULL,     OrderNumber int NOT NULL,     OrderDate date DEFAULT GETDATE() ); |

To create a DEFAULT constraint on the "City" column when the table is already created, use the following SQL:

|  |
| --- |
| ALTER TABLE Persons MODIFY City DEFAULT 'Sandnes'; |

To drop a DEFAULT constraint, use the following SQL:

|  |
| --- |
| ALTER TABLE Persons ALTER COLUMN City DROP DEFAULT; |

## SQL CREATE INDEX Statement

The CREATE INDEX statement is used to create indexes in tables.

Indexes are used to retrieve data from the database more quickly than otherwise. The users cannot see the indexes, they are just used to speed up searches/queries.

**Note:** Updating a table with indexes takes more time than updating a table without (because the indexes also need an update). So, only create indexes on columns that will be frequently searched against.

Creates an index on a table. Duplicate values are allowed:

|  |
| --- |
| CREATE INDEX index\_name ON table\_name (column1, column2, ...); |

Creates a unique index on a table. Duplicate values are not allowed:

|  |
| --- |
| CREATE UNIQUE INDEX index\_name ON table\_name (column1, column2, ...); |

The DROP INDEX statement is used to delete an index in a table.

|  |
| --- |
| DROP INDEX index\_name; |

## AUTO INCREMENT Field

Auto-increment allows a unique number to be generated automatically when a new record is inserted into a table.

Often this is the primary key field that we would like to be created automatically every time a new record is inserted.

|  |
| --- |
| CREATE SEQUENCE seq\_person MINVALUE 1 START WITH 1 INCREMENT BY 1 CACHE 10; |

The code above creates a sequence object called seq\_person, that starts with 1 and will increment by 1. It will also cache up to 10 values for performance. The cache option specifies how many sequence values will be stored in memory for faster access.

To insert a new record into the "Persons" table, we will have to use the nextval function (this function retrieves the next value from seq\_person sequence):

|  |
| --- |
| INSERT INTO Persons (Personid,FirstName,LastName) VALUES (seq\_person.nextval,'Lars','Monsen'); |

## SQL Dates

The most difficult part when working with dates is to be sure that the format of the date you are trying to insert, matches the format of the date column in the database.

**MySQL** comes with the following data types for storing a date or a date/time value in the database:

* DATE - format YYYY-MM-DD
* DATETIME - format: YYYY-MM-DD HH:MI:SS
* TIMESTAMP - format: YYYY-MM-DD HH:MI:SS
* YEAR - format YYYY or YY

## SQL CREATE VIEW Statement

In SQL, a view is a virtual table based on the result-set of an SQL statement.

A view contains rows and columns, just like a real table. The fields in a view are fields from one or more real tables in the database.

You can add SQL functions, WHERE, and JOIN statements to a view and present the data as if the data were coming from one single table.

|  |
| --- |
| CREATE VIEW view\_name AS SELECT column1, column2, ... FROM table\_name WHERE condition; |

**Note:** A view always shows up-to-date data! The database engine recreates the data, using the view's SQL statement, every time a user queries a view.

We can query the view above as follows:

|  |
| --- |
| SELECT \* FROM view\_name ; |

A view can be updated with the CREATE OR REPLACE VIEW command.

|  |
| --- |
| CREATE OR REPLACE VIEW view\_name AS SELECT column1, column2, ... FROM table\_name WHERE condition; |

A view is deleted with the DROP VIEW command.

|  |
| --- |
| DROP VIEW view\_name; |