# **Difference Between DBMS and RDBMS**

| **DBMS** | **RDBMS** |
| --- | --- |
| [DBMS](https://www.geeksforgeeks.org/introduction-of-dbms-database-management-system-set-1/) stores data as file. | [RDBMS](https://www.geeksforgeeks.org/rdbms-architecture/) stores data in tabular form. |
| Data elements need to access individually. | Multiple data elements can be accessed at the same time. |
| No relationship between data. | Data is stored in the form of tables which are related to each other. |
| Normalization is not present. | Normalization is present. |
| DBMS does not support distributed database. | RDBMS supports distributed database. |
| It stores data in either a navigational or hierarchical form. | It uses a tabular structure where the headers are the column names, and the rows contain corresponding values. |
| It deals with small quantity of data. | It deals with large amount of data. |
| Data redundancy is common in this model. | Keys and indexes do not allow Data redundancy. |
| It is used for small organization and deal with small data. | It is used to handle large amount of data. |
| Security is less | More security measures provided. |
| Data fetching is slower for the large amount of data. | Data fetching is fast because of relational approach. |
| The data in a DBMS is subject to low security levels with regards to data manipulation. | There exists multiple levels of data security in a RDBMS. |
| Low software and hardware necessities. | Higher software and hardware necessities. |
| Examples:[XML](https://www.geeksforgeeks.org/xml-basics/), Window Registry, Forxpro, dbaseIIIplus etc. | Examples: [MySQL](https://www.geeksforgeeks.org/architecture-of-mysql/), [PostgreSQL](https://www.geeksforgeeks.org/what-is-postgresql-introduction/), [SQL](https://www.geeksforgeeks.org/what-is-sql/) Server, Oracle, Microsoft Access etc. |

# **Difference Between SQL and NoSQL**

| **Feature** | **SQL Databases** | **NoSQL Databases** |
| --- | --- | --- |
| **Data Model** | Relational: Organizes data into tables with rows and columns, using relationships between tables (foreign keys). | Non-relational: Can be document-based, key-value pairs, wide-column stores, or graph databases, allowing flexible data models. |
| **Schema** | Fixed schema: Requires defining the structure (tables, columns, data types) before data entry, enforcing consistency. | Dynamic schema: Allows for flexible and schema-less data structures, enabling changes to data models without downtime. |
| **Scalability** | Primarily vertical scaling: Involves adding more resources (CPU, RAM) to a single server, with limited horizontal scaling capabilities. | Primarily horizontal scaling: Involves adding more servers or nodes to a distributed system, making it easier to scale out and handle large volumes of data. |
| **Query Language** | SQL (Structured Query Language): Standardized language for querying and managing relational databases, offering powerful querying capabilities. | Varies: Uses different query languages or APIs depending on the database type (e.g., MongoDB uses a JSON-like query language, Cassandra uses CQL). |
| **ACID Transactions** | Strong support for ACID (Atomicity, Consistency, Isolation, Durability) properties, ensuring reliable and consistent transactions. | Varies: Some NoSQL databases support ACID properties, but many prioritize scalability and availability, often using eventual consistency. |
| **Examples** | MySQL, PostgreSQL, Oracle, SQL Server: Widely used for applications requiring complex queries and transactions. | MongoDB, Cassandra, Redis, Couchbase: Popular for handling large-scale data, real-time analytics, and flexible data models. |
| **Use Cases** | Ideal for applications needing complex queries, transactions, and structured data (e.g., financial systems, CRM). | Suitable for big data, real-time analytics, content management, IoT, and scenarios requiring flexible or hierarchical data structures. |
| **Performance** | Optimized for complex queries, joins, and data integrity, but can face performance bottlenecks with very large datasets. | Optimized for fast read/write operations, handling high throughput and low latency, particularly for large datasets and distributed systems. |

**Explain SQL data types.**

### Numeric Data Types

1. **INT**
   * Definition: INT is used for whole numbers without decimals.
   * Example: age INT (e.g., age = 25)
2. **SMALLINT**
   * Definition: Smaller range of integers, useful for saving space.
   * Example: quantity SMALLINT (e.g., quantity = 150)
3. **BIGINT**
   * Definition: Larger range of integers, useful for big numbers.
   * Example: population BIGINT (e.g., population = 9876543210)
4. **DECIMAL(p, s)**
   * Definition: Fixed-point number with precision p and scale s.
   * Example: price DECIMAL(10, 2) (e.g., price = 1234.56)
5. **NUMERIC(p, s)**
   * Definition: Same as DECIMAL, often interchangeable.
   * Example: weight NUMERIC(5, 2) (e.g., weight = 70.50)
6. **FLOAT**
   * Definition: Approximate floating-point number.
   * Example: rating FLOAT (e.g., rating = 4.7)
7. **DOUBLE**
   * Definition: Double-precision floating-point number.
   * Example: distance DOUBLE (e.g., distance = 12345.6789)

### Character and String Data Types

1. **CHAR(n)**
   * Definition: Fixed-length string, padded with spaces if necessary.
   * Example: code CHAR(5) (e.g., code = 'A123 ')
2. **VARCHAR(n)**
   * Definition: Variable-length string.
   * Example: name VARCHAR(50) (e.g., name = 'John Doe')
3. **TEXT**
   * Definition: Large text field, for longer text data.
   * Example: description TEXT (e.g., description = 'This is a long description...')

### Date and Time Data Types

1. **DATE**
   * Definition: Stores date values (year, month, day).
   * Example: birthdate DATE (e.g., birthdate = '1990-01-01')
2. **TIME**
   * Definition: Stores time of day (hours, minutes, seconds).
   * Example: start\_time TIME (e.g., start\_time = '14:30:00')
3. **DATETIME**
   * Definition: Stores both date and time.
   * Example: created\_at DATETIME (e.g., created\_at = '2024-07-28 12:00:00')
4. **TIMESTAMP**
   * Definition: Similar to DATETIME, often used for tracking record changes.
   * Example: updated\_at TIMESTAMP (e.g., updated\_at = '2024-07-28 12:00:00')
5. **YEAR**
   * Definition: Stores year values.
   * Example: year\_published YEAR (e.g., year\_published = '2024')

### Binary Data Types

1. **BINARY(n)**
   * Definition: Fixed-length binary data.
   * Example: file\_hash BINARY(16) (e.g., file\_hash = 0x1234567890ABCDEF)
2. **VARBINARY(n)**
   * Definition: Variable-length binary data.
   * Example: image\_data VARBINARY(1024) (e.g., image\_data = 0xFFD8FFE0...)

### Boolean Data Type

* **BOOLEAN**
  + Definition: Represents true or false.
  + Example: is\_active BOOLEAN (e.g., is\_active = TRUE)

### Other Data Types

1. **ENUM**
   * Definition: Enumerated list of possible values.
   * Example: status ENUM('active', 'inactive', 'pending') (e.g., status = 'active')
2. **SET**
   * Definition: Similar to ENUM but allows multiple values.
   * Example: tags SET('news', 'tech', 'health') (e.g., tags = 'news,tech')
3. **JSON**
   * Definition: Stores JSON-formatted data.
   * Example: profile JSON (e.g., profile = '{"name": "John", "age": 30}')
4. **UUID**
   * Definition: Universally unique identifier.
   * Example: id UUID (e.g., id = '550e8400-e29b-41d4-a716-446655440000')

## ****DDL (Data Definition Language)****

[DDL](https://www.geeksforgeeks.org/features-of-structured-query-language-sql) 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.

DDL is a set of SQL commands used to create, modify, and delete database structures but not data. These commands are normally not used by a general user, who should be accessing the database via an application.

**List of DDL Commands**

Some DDL commands and their syntax are:

| **Command** | **Description** | **Syntax** |
| --- | --- | --- |
| [CREATE](https://www.geeksforgeeks.org/sql-create) | Create database or its objects (table, index, function, views, store procedure, and triggers) | CREATE TABLE table\_name (column1 data\_type, column2 data\_type, ...); |
| [DROP](https://www.geeksforgeeks.org/sql-drop-truncate) | Delete objects from the database | DROP TABLE table\_name; |
| [ALTER](https://www.geeksforgeeks.org/sql-alter-add-drop-modify) | Alter the structure of the database | ALTER TABLE table\_name ADD COLUMN column\_name data\_type; |
| [TRUNCATE](https://www.geeksforgeeks.org/sql-drop-truncate) | Remove all records from a table, including all spaces allocated for the records are removed | TRUNCATE TABLE table\_name; |
| [COMMENT](https://www.geeksforgeeks.org/sql-comments) | Add comments to the data dictionary | COMMENT 'comment\_text' ON TABLE table\_name; |
| [RENAME](https://www.geeksforgeeks.org/sql-alter-rename) | Rename an object existing in the database | RENAME TABLE old\_table\_name TO new\_table\_name; |

**DQL (Data Query Language)**

**DQL**statements are used for performing queries on the data within schema objects. The purpose of the DQL Command is to get some schema relation based on the query passed to it. We can define DQL as follows it is a component of SQL statement that allows getting data from the database and imposing order upon it. It includes the SELECT statement.

This command allows getting the data out of the database to perform operations with it. When a SELECT is fired against a table or tables the result is compiled into a further temporary table, which is displayed or perhaps received by the program i.e. a front-end.

**DQL Command**

There is only one DQL command in SQL i.e.

| **Command** | **Description** | **Syntax** |
| --- | --- | --- |
| [**SELECT**](https://www.geeksforgeeks.org/sql-select-clause) | It is used to retrieve data from the database | SELECT column1, column2, ...FROM table\_name<br>WHERE condition; |

**DML(Data Manipulation Language)**

The SQL commands that deal with the manipulation of data present in the database belong to DML or Data Manipulation Language and this includes most of the SQL statements.

It is the component of the SQL statement that controls access to data and to the database. Basically, DCL statements are grouped with DML statements.

**List of DML commands**

Some DML commands and their syntax are:

| **Command** | **Description** | **Syntax** |
| --- | --- | --- |
| [INSERT](https://www.geeksforgeeks.org/sql-insert-statement) | Insert data into a table | INSERT INTO table\_name (column1, column2, ...) VALUES (value1, value2, ...); |
| [UPDATE](https://www.geeksforgeeks.org/sql-update-statement) | Update existing data within a table | UPDATE table\_name SET column1 = value1, column2 = value2 WHERE condition; |
| [DELETE](https://www.geeksforgeeks.org/sql-delete-statement) | Delete records from a database table | DELETE FROM table\_name WHERE condition; |
| [LOCK](https://www.geeksforgeeks.org/sql-lock-table) | Table control concurrency | LOCK TABLE table\_name IN lock\_mode; |
| CALL | Call a PL/SQL or JAVA subprogram | CALL procedure\_name(arguments); |
| EXPLAIN PLAN | Describe the access path to data | EXPLAIN PLAN FOR SELECT \* FROM table\_name; |

**DCL (Data Control Language)**

DCL includes commands such as GRANT and REVOKE which mainly deal with the rights, permissions, and other controls of the database system.

**List of  DCL commands:**

|  |  |  |
| --- | --- | --- |
| Command | Description | Syntax |
| [GRANT](https://www.geeksforgeeks.org/mysql-grant-revoke-privileges) | Assigns new privileges to a user account, allowing access to specific database objects, actions, or functions. | GRANT privilege\_type [(column\_list)] ON [object\_type] object\_name TO user [WITH GRANT OPTION]; |
| [REVOKE](https://www.geeksforgeeks.org/difference-between-grant-and-revoke) | Removes previously granted privileges from a user account, taking away their access to certain database objects or actions. | REVOKE [GRANT OPTION FOR] privilege\_type [(column\_list)] ON [object\_type] object\_name FROM user [CASCADE]; |

**TCL (Transaction Control Language)**

Transactions group a set of tasks into a single execution unit. Each transaction begins with a specific task and ends when all the tasks in the group are successfully completed. If any of the tasks fail, the transaction fails.

Therefore, a transaction has only two results: success or failure. You can explore more about transactions[***here***](https://www.geeksforgeeks.org/sql-transactions). Hence, the following TCL commands are used to control the execution of a transaction:

**List of TCL Commands**

Some TCL commands and their syntax are:

| **Command** | **Description** | **Syntax** |
| --- | --- | --- |
| [BEGIN TRANSACTION](https://www.geeksforgeeks.org/sql-transactions/#:~:text=) | Starts a new transaction | BEGIN TRANSACTION [transaction\_name]; |
| [COMMIT](https://www.geeksforgeeks.org/sql-transactions/#:~:text=) | Saves all changes made during the transaction | COMMIT; |
| [ROLLBACK](https://www.geeksforgeeks.org/sql-transactions/#:~:text=) | Undoes all changes made during the transaction | ROLLBACK; |
| [SAVEPOINT](https://www.geeksforgeeks.org/sql-transactions/#:~:text=) | Creates a savepoint within the current transaction | SAVEPOINT savepoint\_name; |

**30 commonly used SQL commands with examples:**

| **SQL Command** | **Example** |
| --- | --- |
| SELECT | SELECT name, age FROM users; |
| INSERT INTO | INSERT INTO users (name, age) VALUES ('Alice', 30); |
| UPDATE | UPDATE users SET age = 31 WHERE name = 'Alice'; |
| DELETE | DELETE FROM users WHERE name = 'Alice'; |
| CREATE TABLE | CREATE TABLE users (id INT, name VARCHAR(50)); |
| ALTER TABLE | ALTER TABLE users ADD email VARCHAR(100); |
| DROP TABLE | DROP TABLE users; |
| CREATE DATABASE | CREATE DATABASE company\_db; |
| DROP DATABASE | DROP DATABASE company\_db; |
| CREATE INDEX | CREATE INDEX idx\_name ON users (name); |
| DROP INDEX | DROP INDEX idx\_name; |
| JOIN | SELECT users.name, orders.total FROM users JOIN orders ON users.id = orders.user\_id; |
| INNER JOIN | SELECT \* FROM users INNER JOIN orders ON users.id = orders.user\_id; |
| LEFT JOIN | SELECT \* FROM users LEFT JOIN orders ON users.id = orders.user\_id; |
| RIGHT JOIN | SELECT \* FROM users RIGHT JOIN orders ON users.id = orders.user\_id; |
| FULL JOIN | SELECT \* FROM users FULL JOIN orders ON users.id = orders.user\_id; |
| GROUP BY | SELECT COUNT(\*), department FROM employees GROUP BY department; |
| ORDER BY | SELECT \* FROM users ORDER BY name ASC; |
| HAVING | SELECT department, COUNT(\*) FROM employees GROUP BY department HAVING COUNT(\*) > 10; |
| WHERE | SELECT \* FROM users WHERE age > 25; |
| DISTINCT | SELECT DISTINCT department FROM employees; |
| LIMIT | SELECT \* FROM users LIMIT 10; |
| OFFSET | SELECT \* FROM users LIMIT 10 OFFSET 5; |
| UNION | SELECT name FROM employees UNION SELECT name FROM managers; |
| UNION ALL | SELECT name FROM employees UNION ALL SELECT name FROM managers; |
| LIKE | SELECT \* FROM users WHERE name LIKE 'A%'; |
| IN | SELECT \* FROM users WHERE department IN ('HR', 'IT'); |
| BETWEEN | SELECT \* FROM orders WHERE order\_date BETWEEN '2023-01-01' AND '2023-12-31'; |
| IS NULL | SELECT \* FROM users WHERE email IS NULL; |
| IS NOT NULL | SELECT \* FROM users WHERE email IS NOT NULL; |

**What is SQL?**

[**SQL**](https://www.geeksforgeeks.org/sql-tutorial) stands for Structured Query Language. It is a language used to interact with the database, i.e to create a database, to create a table in the database, to retrieve data or update a table in the database, etc. SQL is an ANSI(American National Standards Institute) standard. Using SQL, we can do many things. For example – we can execute queries, we can insert records into a table, can update records, can create a database, can create a table, can delete a table, etc.

**Does SQL support programming language features?**

It is true that SQL is a language, but it does not support programming as it is not a programming language, it is a command language. We do not have conditional statements in SQL like for loops or if..else, we only have commands which we can use to query, update, delete, etc. data in the database. SQL allows us to manipulate data in a database.

**What is the difference between CHAR and VARCHAR datatype in SQL?**

Both of these data types are used for characters, but varchar is used for character strings of variable length, whereas char is used for character strings of fixed length. **For example**, if we specify the type as char(5) then we will not be allowed to store a string of any other length in this variable, but if we specify the type of this variable as varchar(5) then we will be allowed to store strings of variable length. We can store a string of length 3 or 4 or 2 in this variable.

**What do you mean by data definition language?**

**Data definition language** or DDL allows to execution of queries like CREATE, DROP, and ALTER. That is those queries that define the data.

**What do you mean by data manipulation language?**

Data manipulation Language or DML is used to access or manipulate data in the database. It allows us to perform the below-listed functions:

Insert data or rows in a database

Delete data from the database

Retrieve or fetch data

Update data in a database.

**What is the view in SQL?**

A view in SQL is a virtual table that is based on the result set of an SQL query. It provides a way to simplify complex queries, encapsulate complex logic, and enhance security by restricting access to certain data. Views do not store data themselves; instead, they display data stored in other tables. Here are the key aspects of a view:

**View Definition**: Defined using the CREATE VIEW statement with a SELECT query.

**Virtual Nature**: Views do not store data themselves; they display data from the underlying tables.

**Query Simplification**: Simplifies complex queries by encapsulating them within a view.

**Security and Access Control**: Restricts user access to certain parts of the data.

**Updatability**: Some views can be updated directly, but this depends on the complexity and the SQL database system.

**Example :-**

CREATE VIEW EmployeeDepartments AS

SELECT e.EmployeeID, e.Name, d.DepartmentName

FROM Employees e

JOIN Departments d ON e.DepartmentID = d.DepartmentID;

SELECT \* FROM EmployeeDepartments;

**What do you mean by foreign key?**

A foreign key is a column or a set of columns in one table that establishes a link between the data in two tables. It acts as a cross-reference between tables by referencing the primary key of another table. Foreign keys ensure referential integrity, meaning that a value in the foreign key column must match a value in the primary key column of the referenced table or be null.

**Syntax:**

CREATE TABLE Orders  
(  
O\_ID int NOT NULL,  
ORDER\_NO int NOT NULL,  
C\_ID int,  
PRIMARY KEY (O\_ID),  
FOREIGN KEY (C\_ID) REFERENCES Customers(C\_ID)  
)

**What are table and Field?**

**Table:**A table has a combination of rows and columns. Rows are called records and columns are called fields. In MS SQL Server, the tables are being designated within the database and schema names.

**Field:**In DBMS, a database field can be defined as – a single piece of information from a record.

**What is the primary key?**

A [**Primary Key**](https://www.geeksforgeeks.org/difference-between-primary-and-candidate-key)is one of the candidate keys. One of the candidate keys is selected as the most important and becomes the primary key. There cannot be more than one primary key in a table.

**Explain the various types of keys in a database.**

Ans. In a database, there are several types of keys:

**Candidate Key**: A set of attributes that uniquely identifies each record in a table. A table can have multiple candidate keys, but each must uniquely identify the records.

**Super Key**: A superset of a candidate key, meaning it includes the candidate key and possibly additional attributes. It still uniquely identifies each record in the table.

**Primary Key**: A specific candidate key chosen to uniquely identify each record in the table. A primary key cannot contain NULL values and must be unique.

**Unique Key**: Similar to a primary key in that it must contain unique values across records, but unlike a primary key, it can contain NULL values. A table can have multiple unique keys.

**Alternate Key**: Candidate keys that are not chosen as the primary key. These are still capable of uniquely identifying records.

**Foreign Key**: An attribute or a set of attributes in one table that references the primary key of another table, establishing a relationship between the two tables.

**What is a Default constraint?**

The [**DEFAULT**](https://www.geeksforgeeks.org/sql-default-constraint)constraint is used to fill a column with default and fixed values. The value will be added to all new records when no other value is provided.

**What is RANK() and DENSE\_RANK() in sql**

In SQL, RANK and DENSE\_RANK are window functions that assign a rank to each row within a partition of a result set. They are often used in analytical queries to provide ranking based on the values of one or more columns.

**RANK**

The RANK function assigns a unique rank to each distinct value within the partition. The ranks are not consecutive if there are ties (i.e., duplicate values). Instead, gaps are left in the ranking sequence.

**Key Features:**

* **Gaps in Ranks**: If two or more rows have the same rank, the next rank(s) will have gaps.
* **Order By Clause**: RANK is typically used with the ORDER BY clause to specify how the rows should be ranked.

SELECT

employee\_id,

salary,

RANK() OVER (ORDER BY salary DESC) AS rank

FROM

employees;

**Example Result:**

| **employee\_id** | **salary** | **rank** |
| --- | --- | --- |
| 1 | 10000 | 1 |
| 2 | 9000 | 2 |
| 3 | 9000 | 2 |
| 4 | 8000 | 4 |

In this example, employees with the same salary (9000) share the same rank, and the next rank after the tie skips a value (from 2 to 4).

**DENSE\_RANK**

The DENSE\_RANK function also assigns a rank to each distinct value within the partition, but without gaps in the ranking sequence. Ties are assigned the same rank, and the next rank continues sequentially.

**Key Features:**

* **No Gaps in Ranks**: If two or more rows have the same rank, the next rank is consecutive without any gaps.
* **Order By Clause**: DENSE\_RANK is used with the ORDER BY clause to determine the ranking order.

**Example:**

SELECT

employee\_id,

salary,

DENSE\_RANK() OVER (ORDER BY salary DESC) AS dense\_rank

FROM

employees;

**Example Result:**

| **employee\_id** | **salary** | **dense\_rank** |
| --- | --- | --- |
| 1 | 10000 | 1 |
| 2 | 9000 | 2 |
| 3 | 9000 | 2 |
| 4 | 8000 | 3 |

In this example, employees with the same salary (9000) share the same rank, and the next rank continues sequentially (from 2 to 3).

**What is normalization?**

* Normalization is used to minimize the redundancy from a relation or set of relations. It is also used to eliminate undesirable characteristics like Insertion, Update, and Deletion Anomalies.
* Normalization is the process of organizing the data in the database.
* Normalization divides the larger table into smaller and links them using relationships.
* The normal form is used to reduce redundancy from the database table.

**What is Denormalization?**

[Denormalization](https://www.geeksforgeeks.org/denormalization-in-databases)is a database optimization technique in which we add redundant data to one or more tables. This can help us avoid costly joins in a relational database. Note that denormalization does not mean not doing normalization. It is an optimization technique that is applied after normalization.

In a traditional normalized database, we store data in separate logical tables and attempt to minimize redundant data. We may strive to have only one copy of each piece of data in the database.

**What is a query?**

An **SQL** query is used to retrieve the required data from the database. However, there may be multiple SQL queries that yield the same results but with different levels of efficiency. An inefficient query can drain the database resources, reduce the database speed or result in a loss of service for other users. So it is very important to optimize the query to obtain the best database performance.

**What is a subquery?**

In SQL, a [Subquery](https://www.geeksforgeeks.org/sql-subquery)can be simply defined as a query within another query. In other words, we can say that a Subquery is a query that is embedded in the WHERE clause of another SQL query.

**What are the different operators available in SQL?**

There are three operators available in SQL namely:

Arithmetic Operators

Logical Operators

Comparison Operators

**What is a Constraint?**

Constraints are the rules that we can apply to the type of data in a table. That is, we can specify the limit on the type of data that can be stored in a particular column in a table using constraints. For more details please refer to [SQL|Constraints](https://www.geeksforgeeks.org/sql-constraints) article.

### 1. ****PRIMARY KEY****

A primary key uniquely identifies each record in a table. A table can have only one primary key, which can consist of single or multiple columns.

CREATE TABLE employees (id INT PRIMARY KEY, name VARCHAR(50), department VARCHAR(50));

### 2. ****FOREIGN KEY****

A foreign key is a column (or group of columns) that creates a link between data in two tables. It references the primary key of another table.

### CREATE TABLE orders (order\_id INT PRIMARY KEY, employee\_id INT, FOREIGN KEY (employee\_id) REFERENCES employees(id));

### 3. ****UNIQUE****

The UNIQUE constraint ensures that all values in a column are different. Unlike the primary key, a table can have multiple UNIQUE constraints.

### CREATE TABLE products (product\_id INT PRIMARY KEY, product\_name VARCHAR(100) UNIQUE);

### 4. ****NOT NULL****

The NOT NULL constraint ensures that a column cannot have a NULL value. It is used when the field must always contain a value.

### CREATE TABLE customers (customer\_id INT PRIMARY KEY, customer\_name VARCHAR(100) NOT NULL, email VARCHAR(100));

### 5. ****CHECK****

The CHECK constraint limits the values that can be placed in a column. It ensures that all values in a column satisfy a specific condition.

CREATE TABLE employees (id INT PRIMARY KEY, name VARCHAR(50), age INT CHECK (age >= 18));

### 6. ****DEFAULT****

The DEFAULT constraint provides a default value for a column when none is specified. This ensures that the column is populated even if no value is provided during insertion.

CREATE TABLE orders (order\_id INT PRIMARY KEY, product\_name VARCHAR(100), order\_date DATE DEFAULT CURRENT\_DATE);

### 7. ****INDEX****

An INDEX constraint is not technically a constraint but is often used in conjunction with constraints to speed up searches and queries by creating an index on columns.

CREATE INDEX idx\_employee\_name ON employees (name);

**What is Auto Increment?**

Sometimes, while creating a table, we do not have a unique identifier within the table, hence we face difficulty in choosing Primary Key. So as to resolve such an issue, we’ve to manually provide unique keys to every record, but this is often also a tedious task. So we can use the  Auto-Increment feature that automatically generates a numerical Primary key value for every new record inserted. The Auto Increment feature is supported by all the Databases. For more details please refer [SQL Auto Increment](https://www.geeksforgeeks.org/sql-auto-increment)article.

**What is a stored procedure?**

A stored procedure is a named set of SQL statements that are stored in a database and can be executed multiple times. It is a precompiled collection of one or more SQL statements and procedural logic, which is stored as a schema object in a relational database management system (RDBMS). Stored procedures allow you to group and execute multiple SQL statements as a single unit, which provides several benefits:

Here's a simple example of a stored procedure in SQL Server that retrieves the list of employees from an Employees table:

CREATE PROCEDURE GetEmployees

AS

BEGIN

SELECT EmployeeID, Name, DepartmentID

FROM Employees;

END;

Once created, the stored procedure GetEmployees can be executed by simply calling its name:

EXEC GetEmployees;

**What are aggregate and scalar functions?**

For doing operations on data SQL has many built-in functions, they are categorized into two categories and further sub-categorized into seven different functions under each category. The categories are:

**Aggregate functions:**These functions are used to do operations from the values of the column and a single value is returned.

[**SQL Aggregate Functions**](https://www.geeksforgeeks.org/aggregate-functions-in-sql/) operate on a data group and return a singular output. They are mostly used with the [**GROUP BY**](https://www.geeksforgeeks.org/sql-group-by/)clause to summarize data.

| **Aggregate Function** | **Description** | **Syntax** |
| --- | --- | --- |
| **AVG()** | Calculates the average value | SELECT AVG(column\_name) FROM table\_name; |
| **COUNT()** | Counts the number of rows | SELECT COUNT(column\_name) FROM table\_name |
| **FIRST()** | Returns the first value in an ordered set of values | SELECT FIRST(column\_name) FROM table\_name; |
| **LAST()** | Returns the last value in an ordered set of values | SELECT LAST(column\_name) FROM table\_name; |
| **MAX()** | Retrieves the maximum value from a column | SELECT MAX(column\_name) FROM table\_name; |
| **MIN()** | Retrieves the minimum value from a column | SELECT MIN(column\_name) FROM table\_name; |
| **SUM()** | Calculates the total sum of values in a numeric column | SELECT SUM(column\_name) FROM table\_name; |

**SQL Scalar functions**

**SQL Scalar Functions**are built-in functions that operate on a single value and return a single value.

Scalar functions in SQL helps in efficient data manipulation and simplification of complex calculations in SQL queries.

**Scalar functions:**These functions are based on user input, these too return a single value.

| **Scalar function** | **Description** | **Syntax** |
| --- | --- | --- |
| **UCASE()** | Converts a string to uppercase | SELECT UCASE(column\_name) FROM table\_name; |
| **LCASE()** | Converts a string to lowercase | SELECT LCASE(column\_name) FROM table\_name; |
| **MID()** | Extracts a substring from a string | SELECT MID(column\_name, start, length) FROM table\_name; |
| **LEN()** | Returns the length of a string | SELECT LEN(column\_name) FROM table\_name; |
| **ROUND()** | Rounds a number to a specified number of decimals | SELECT ROUND(column\_name, decimals) FROM table\_name; |
| **NOW()** | Returns the current date and time | SELECT NOW(); |
| **FORMAT()** | Formats a value with the specified format | SELECT FORMAT(column\_name, format) FROM table\_name; |

**What is an ALIAS command?**

Aliases are the temporary names given to a table or column for the purpose of a particular SQL query. It is used when the name of a column or table is used other than its original name, but the modified name is only temporary.

Aliases are created to make table or column names more readable.

The renaming is just a temporary change and the table name does not change in the original database.

Aliases are useful when table or column names are big or not very readable.

These are preferred when there is more than one table involved in a query.

For more details, please read the [SQL | Aliases](https://www.geeksforgeeks.org/sql-aliases)article.

**What are Union, minus, and Interact commands?**

Set Operations in SQL eliminate duplicate tuples and can be applied only to the relations which are union compatible. Set Operations available in SQL are :

Set Union

Set Intersection

Set Difference

**UNION Operation:** This operation includes all the tuples which are present in either of the relations. For example: To find all the customers who have a loan or an account or both in a bank.

SELECT CustomerName FROM Depositor   
 UNION   
 SELECT CustomerName FROM Borrower ;

The union operation automatically eliminates duplicates. If all the duplicates are supposed to be retained, UNION ALL is used in place of UNION.

**INTERSECT Operation:**This operation includes the tuples which are present in both of the relations. For example: To find the customers who have a loan as well as an account in the bank:

SELECT CustomerName FROM Depositor   
 INTERSECT  
 SELECT CustomerName FROM Borrower ;

The Intersect operation automatically eliminates duplicates. If all the duplicates are supposed to be retained, INTERSECT ALL is used in place of INTERSECT.

**EXCEPT for Operation:**This operation includes tuples that are present in one relationship but should not be present in another relationship. For example: To find customers who have an account but no loan at the bank:

SELECT CustomerName FROM Depositor   
 EXCEPT  
 SELECT CustomerName FROM Borrower ;

The Except operation automatically eliminates the duplicates. If all the duplicates are supposed to be retained, EXCEPT ALL is used in place of EXCEPT.

**What is ETL in SQL?**

ETL is a process in Data Warehousing and it stands for **Extract**, **Transform,**and **Load**. It is a process in which an ETL tool extracts the data from various data source systems, transforms it in the staging area, and then finally, loads it into the Data Warehouse system. These are three database functions that are incorporated into one tool to pull data out from one database and put data into another database.

**How to copy tables in SQL?**

Sometimes, in SQL, we need to create an exact copy of an already defined (or created) table. [MySQL](https://www.geeksforgeeks.org/sql-tutorial/#mysql) enables you to perform this operation. Because we may need such duplicate tables for testing the data without having any impact on the original table and the data stored in it.

**1. Copying Table Structure Only**

To create a copy of the table structure without the data, use the following syntax:

CREATE TABLE new\_table AS

SELECT \* FROM original\_table WHERE 1=0;

* **Explanation**: WHERE 1=0 ensures that no rows are copied, just the table structure.

**2. Copying Table Structure and Data**

To copy both the structure and data from an existing table:

CREATE TABLE new\_table AS

SELECT \* FROM original\_table;

**3. Copying Table with a New Structure and Data**

If you want to copy data into a new table with potentially different columns or types:

CREATE TABLE new\_table (id INT, name VARCHAR(50));

INSERT INTO new\_table (id, name)

SELECT id, name FROM original\_table;

* **Explanation**: This allows you to modify the structure or data type in the new table while copying.

**4. Using INSERT INTO ... SELECT for Existing Tables**

If the new table already exists and you just want to copy data:

INSERT INTO new\_table (column1, column2, ...)

SELECT column1, column2, ...

FROM original\_table;

1. **Using Clone method**

The CLONE statement is a feature available in some SQL database systems, like MySQL, that allows you to create a copy of a table, including its structure, data, indexes, and constraints.

CREATE TABLE new\_table CLONE original\_table;

In this example, new\_table is created

**What is SQL injection?**

SQL injection is a technique used to exploit user data through web page inputs by injecting SQL commands as statements. Basically, these statements can be used to manipulate the application’s web server by malicious users.

SQL injection is a code injection technique that might destroy your database.

SQL injection is one of the most common web hacking techniques.

SQL injection is the placement of malicious code in SQL statements, via web page input.

For more details, please read the [SQL | Injection](https://www.geeksforgeeks.org/sql-injection-2)article.

**What are the differences between SQL and PL/SQL?**

Some common differences between SQL and PL/SQL are as shown below:

| **SQL** | **PL/SQL** |
| --- | --- |
| SQL is a query execution or commanding language | PL/SQL is a complete programming language |
| SQL is a data-oriented language. | PL/SQL is a procedural language |
| SQL is very declarative in nature. | PL/SQL has a procedural nature. |
| It is used for manipulating data. | It is used for creating applications. |
| We can execute one statement at a time in SQL | We can execute blocks of statements in PL/SQL |
| SQL tells databases, what to do? | PL/SQL tells databases how to do. |
| We can embed SQL in PL/SQL | We can not embed PL/SQL in SQL |

**What is the difference between BETWEEN and IN operators in SQL?**

**BETWEEN:**The **BETWEEN** operator is used to fetch rows based on a range of values.   
For example,

SELECT \* FROM Students   
WHERE ROLL\_NO BETWEEN 20 AND 30;

This query will select all those rows from the table. Students where the value of the field ROLL\_NO lies between 20 and 30.    
**IN:** The **IN**operator is used to fetch values contained in specific sets.   
For example,

SELECT \* FROM Students   
WHERE ROLL\_NO IN (20,21,23);

This query will select all those rows from the table Students where the value of the field ROLL\_NO is either 20 or 21 or 23.

**Write an SQL query to find the names of employees starting with ‘A’.**

The LIKE operator of SQL is used for this purpose. It is used to fetch filtered data by searching for a particular pattern in the where clause.

The required query is:

SELECT \* FROM Employees WHERE EmpName like 'A%' ;

You may refer to this article [WHERE clause](https://www.geeksforgeeks.org/sql-where-clause) for more details on the LIKE operator.

**LIKE operator and Wildcards**

In SQL, the LIKE operator is used for pattern matching, and it utilizes various wildcard characters to define patterns. Here are the wildcards commonly used with the LIKE operator`, along with examples for each:

**1. Percent Sign (%)**

**The percent sign represents zero, one, or multiple characters.**

**Examples:**

* **Finding values starting with a specific letter:**

SELECT \* FROM employees WHERE name LIKE 'J%';

* + Matches: 'John', 'Jane', 'Jack', etc.
* **Finding values ending with a specific string:**

SELECT \* FROM products WHERE product\_name LIKE '%shoe';

* + Matches: 'Running Shoe', 'Casual Shoe', etc.
* **Finding values containing a specific substring:**

SELECT \* FROM emails WHERE address LIKE '%@company.com%';

* + Matches: 'john@company.com', 'jane@company.com', etc.

**2. Underscore (\_)**

**The underscore represents a single character.**

**Examples:**

* **Finding values with a specific character in a specific position:**

SELECT \* FROM items WHERE item\_code LIKE 'A\_3%';

* + Matches: 'A13', 'AB3', 'AX3', etc.
* **Finding three-letter words starting with 'H' and ending with 't':**

SELECT \* FROM words WHERE word LIKE 'H\_t';

* + Matches: 'Hat', 'Hit', 'Hot', etc.

**3. Square Brackets ([])**

**Square brackets are used to specify a set or range of characters to match.**

**Examples:**

* **Finding values starting with 'a', 'b', or 'c':**

SELECT \* FROM names WHERE name LIKE '[abc]%';

* + Matches: 'Alice', 'Bob', 'Charlie', etc.
* **Finding values where the second character is a digit from 1 to 3:**

SELECT \* FROM codes WHERE code LIKE '\_[1-3]%';

* + Matches: 'A1B', 'B2C', 'C3D', etc.

**4. Caret (^) or Exclamation (!) (used within square brackets)**

**The caret (or exclamation mark) negates the set of characters, matching any character not in the set.**

**Examples:**

* **Finding values not starting with 'a', 'b', or 'c':**

SELECT \* FROM names WHERE name LIKE '[^abc]%';

* + Matches: 'David', 'Edward', 'Frank', etc.
* **Finding values where the second character is not a digit from 1 to 3:**

SELECT \* FROM codes WHERE code LIKE '\_[^1-3]%';

* + Matches: 'A4B', 'B5C', 'C6D', etc.

**5. Hyphen (-)**

**The hyphen is used within square brackets to specify a range of characters.**

**Examples:**

* **Finding values with letters in the range 'A' to 'Z':**

SELECT \* FROM items WHERE item\_code LIKE '[A-Z]%';

* + Matches: 'A123', 'B456', 'C789', etc.
* **Finding values where the second character is a digit from 1 to 5:**

SELECT \* FROM codes WHERE code LIKE '\_[1-5]%';

* + Matches: 'A1B', 'B2C', 'C3D', 'D4E', 'E5F', etc.

**Combined Example**

**Combining multiple wildcards for complex patterns:**

**Example:**

* **Finding values starting with 'a' or 'b', followed by any character, then '3', and ending with zero or more characters:**

SELECT \* FROM items WHERE item\_code LIKE '[ab]\_3%';

* + Matches: 'aX3', 'bY3', 'a13', 'b23', etc.

These wildcards provide powerful tools for pattern matching in SQL, allowing flexible and complex queries on textual data.

**What is a join in SQL? What are the types of joins?**

An SQL Join statement is used to combine data or rows from two or more tables based on a common field between them. Different types of Joins are:

**INNER JOIN**: It returns only the rows that have matching values in both tables. The INNER JOIN keyword selects all rows from both tables as long as the condition is satisfied. This keyword will create the result set by combining all rows from both the tables where the condition satisfies i.e. the value of the common field will be the same.

SELECT columns

FROM table1

INNER JOIN table2

ON table1.column = table2.column;

**LEFT JOIN**: This join returns all the rows of the table on the left side of the join and matching rows for the table on the right side of the join. For the rows for which there is no matching row on the right side, the result set will be null. LEFT JOIN is also known as LEFT OUTER JOIN

SELECT columns

FROM table1

LEFT JOIN table2

ON table1.column = table2.column;

**RIGHT JOIN**: RIGHT JOIN is similar to LEFT JOIN. This join returns all the rows of the table on the right side of the join and matching rows for the table on the left side of the join. For the rows for which there is no matching row on the left side, the result set will contain null. RIGHT JOIN is also known as RIGHT OUTER JOIN.

SELECT columns

FROM table1

RIGHT JOIN table2

ON table1.column = table2.column;

**FULL JOIN**: FULL JOIN creates the result set by combining the results of both LEFT JOIN and RIGHT JOIN. The result set will contain all the rows from both tables. For the rows for which there is no matching, the result set will contain NULL values.

SELECT columns

FROM table1

FULL JOIN table2

ON table1.column = table2.column;

**CROSS JOIN**: Returns the Cartesian product of two tables, meaning it returns all possible combinations of rows from the two tables.

SELECT columns

FROM table1

CROSS JOIN table2;

**SELF JOIN**: A join in which a table is joined with itself. It's useful for comparing rows within the same table.

SELECT a.column, b.column

FROM table a, table b

WHERE a.common\_field = b.common\_field;

**What is an index?**

In SQL, an index is a database object that improves the speed of data retrieval operations on a table. Indexes are created on columns that are frequently used in query conditions, such as the WHERE clause, to make data access more efficient. Here’s a detailed explanation of indexes in SQL:

* **Definition**: An index is a data structure that allows the database to find and retrieve specific rows much faster than it could do without an index. It works similarly to an index in a book, which allows you to quickly find the page number for a specific topic.
* **Purpose**: The main purpose of an index is to improve the performance of data retrieval operations. It does this by creating a quick lookup path to access the data.

**Types of Indexes**

1. **Primary Index**:
   * Automatically created when a primary key is defined.
   * Ensures uniqueness of the primary key column(s).
2. **Unique Index**:
   * Ensures that the indexed columns contain unique values.
   * Can be created on columns that should not have duplicate values.
3. **Clustered Index**:
   * Determines the physical order of data in a table.
   * There can be only one clustered index per table.
   * Often created automatically on primary key columns.
4. **Non-Clustered Index**:
   * Does not alter the physical order of data.
   * There can be multiple non-clustered indexes on a table.
   * Stores a pointer to the physical location of the data.
5. **Composite Index**:
   * An index on multiple columns.
   * Useful for queries that filter by multiple columns.

**Creating an Index**

* **Primary Index**:

CREATE TABLE employees (

id INT PRIMARY KEY,

name VARCHAR(100),

position VARCHAR(50)

);

* **Unique Index**:

CREATE UNIQUE INDEX idx\_employee\_email ON employees(email);

* **Clustered Index**:

CREATE CLUSTERED INDEX idx\_employee\_id ON employees(id);

* **Non-Clustered Index**:

CREATE INDEX idx\_employee\_name ON employees(name);

* **Composite Index**:

CREATE INDEX idx\_employee\_name\_position ON employees(name, position);

**Using an Index**

Indexes are used automatically by the SQL database engine when performing queries. For example, if you have an index on the name column of the employees table, a query like this will benefit from the index:

SELECT \* FROM employees WHERE name = 'John';

**Dropping an Index**

If an index is no longer needed, it can be dropped:

DROP INDEX idx\_employee\_name ON employees;

**Pros and Cons of Indexes**

* **Pros**:
  + Faster data retrieval.
  + Improved query performance, especially for large datasets.
  + Can enforce uniqueness (unique indexes).
* **Cons**:
  + Slower write operations (INSERT, UPDATE, DELETE) due to the overhead of maintaining the index.
  + Consumes additional disk space.
  + Complexity in index management and optimization.

**Best Practices**

* Index columns that are frequently used in WHERE, JOIN, ORDER BY, and GROUP BY clauses.
* Avoid over-indexing, as too many indexes can degrade write performance and consume more disk space.
* Regularly monitor and analyze index usage and performance to ensure they are effective.

**What is the On Delete cascade constraint?**

An ‘ON DELETE CASCADE’ constraint is used in MySQL to delete the rows from the child table automatically when the rows from the parent table are deleted. For more details, please read [MySQL – On Delete Cascade constraint](https://www.geeksforgeeks.org/mysql-on-delete-cascade-constraint)article.

**What is a trigger?**

Ans. A trigger is a piece of code associated with insert, update, or delete operations on a table. It is automatically executed when the associated query is run. A trigger is a special type of stored procedure in a database that automatically executes or fires when a specified event occurs in the database. Triggers are used to enforce data integrity, automate system tasks, and maintain business rules within a database. They are commonly used to monitor and respond to changes in data within a table.

CREATE TABLE EmployeeChanges (

ChangeID INT IDENTITY(1,1) PRIMARY KEY,

EmployeeID INT,

ChangeDate DATETIME DEFAULT GETDATE(),

OldName VARCHAR(50),

NewName VARCHAR(50)

);

CREATE TRIGGER trgAfterUpdate

ON Employees

AFTER UPDATE

AS

BEGIN

INSERT INTO EmployeeChanges (EmployeeID, OldName, NewName)

SELECT

d.EmployeeID,

d.Name AS OldName,

i.Name AS NewName

FROM

deleted d

INNER JOIN

inserted i ON d.EmployeeID = i.EmployeeID;

END;

**What is the difference between Cluster and Non-Cluster Index?**

| **CLUSTERED INDEX** | **NON-CLUSTERED INDEX** |
| --- | --- |
| The clustered index is faster. | The non-clustered index is slower. |
| The clustered index requires less memory for operations. | The non-Clustered index requires more memory for operations. |
| In a clustered index, the index is the main data. | In the Non-Clustered index, the index is a copy of data. |
| A table can have only one clustered index. | A table can have multiple non-clustered indexes. |
| The clustered index has an inherent ability to store data on the disk. | The non-Clustered index does not have the inherent ability to store data on the disk. |
| Clustered indexes store pointers to block not data. | The non-Clustered index store both value and a pointer to the the the actual row that holds data. |
| In Clustered index leaf nodes are actual data itself. | In a Non-Clustered index, leaf nodes are not the actual data itself rather they only contain included columns. |
| In the Clustered index, the Clustered key defines the order of data within the table. | In the Non-Clustered index, the index key defines the order of data within the index. |
| A Clustered index is a type of index in which table records are physically reordered to match the index. | A Non-Clustered index is a special type of index in which the logical order of index does not match the physical stored order of the rows on the disk. |

**What is Case WHEN in SQL?**

Control statements form an important part of most languages since they control the execution of other sets of statements. These are found in SQL too and should be exploited for uses such as query filtering and query optimization through careful selection of tuples that match our requirements. In this post, we explore the Case-Switch statement in SQL. The CASE statement is SQL’s way of handling if/then logic.

**Syntax 1:**

*CASE case\_value WHEN when\_value THEN statement\_list [WHEN when\_value THEN statement\_list] … [ELSE statement\_list]END CASE*

**Syntax 2:**

*CASE WHEN search\_condition THEN statement\_list [WHEN search\_condition THEN statement\_list] … [ELSE statement\_list]END CASE*

For more details, please read the [SQL | Case Statement](https://www.geeksforgeeks.org/sql-case-statement) article.

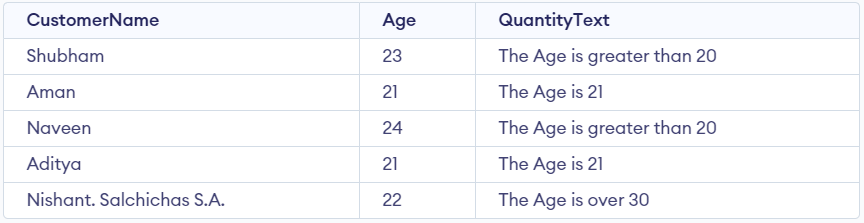
Exmple:

| **ustomerID** | **CustomerName** | **LastName** | **Country** | **Age** |
| --- | --- | --- | --- | --- |
| 1 | Shubham | Thakur | India | 23 |
| 2 | Aman | Chopra | Australia | 21 |
| 3 | Naveen | Tulasi | Sri Lanka | 24 |
| 4 | Aditya | Arpan | Austria | 21 |
| 5 | Nishant. Salchichas S.A. | Jain | Spain | 22 |

**Query:**

**SELECT** CustomerName, Age,  
**CASE**  
 **WHEN** Age> 22 **THEN** 'The Age is greater than 22'  
 **WHEN** Age = 21 **THEN** 'The Age is 21'  
 **ELSE** 'The Age is over 30'  
**END AS** QuantityText  
**FROM** Customer;

**Output:**



**Name different types of case manipulation functions available in SQL.**

There are three types of case manipulation functions available in SQL. They are,

**LOWER**: The purpose of this function is to return the string in lowercase. It takes a string as an argument and returns the string by converting it into lower case.   
**Syntax:**

*LOWER(‘string’)*

**UPPER**: The purpose of this function is to return the string in uppercase. It takes a string as an argument and returns the string by converting it into uppercase.   
**Syntax:**

*UPPER(‘string’)*

**INITCAP**: The purpose of this function is to return the string with the first letter in uppercase and the rest of the letters in lowercase.   
**Syntax:**

*INITCAP(‘string’)*

**What are local and global variables and their differences?**

**Name the function which is used to remove spaces at the end of a string?**

In SQL, the spaces at the end of the string are removed by a trim function.

**Syntax:**

***Trim(s) ,****Where s is a any string.*

**Difference between delete drop and truncate in tabular**

|  | **DELETE** | **TRUNCATE** | **DROP** |
| --- | --- | --- | --- |
| **Type** | DML | DDL | DDL |
| **Uses a lock** | Row lock | Table lock | Table lock |
| **Works in WHERE** | Yes | No | No |
| **Removes ...** | One, some, or all rows in a table. | All rows in a table. | Entire table structure: data, privileges, indexes, constraints, triggers. |
| **Resets ID auto-increment** | No | MySQL: Yes Oracle: No PostgreSQL: User decides SQL Server: Yes | Doesn’t apply |
| **Rollback** | Yes | MySQL: No Oracle: No PostgreSQL: Yes SQL Server: Yes | MySQL: No Oracle: Yes PostgreSQL: Yes SQL Server : Yes |
| **Transaction logging** | Each row | Whole table (minimal) | Whole table (minimal) |
| **Works with indexed views** | Yes | No | No |
| **Space requirements** | More space | Less space | More space |
| **Fires triggers** | Yes | No | No |
| **Speed** | Slow | Fastest | Faster |

**Define SQL Order by the statement?**

The ORDER BY statement in SQL is used to sort the fetched data in either ascending or descending according to one or more columns.

By default ORDER BY sorts the data in **ascending order.**

We can use the keyword DESC to sort the data in descending order and the keyword ASC to sort in ascending order.

**SELECT** \* **FROM** students **ORDER BY** ROLL\_NO **DESC**;

**Explain SQL Having statement?**

HAVING is used to specify a condition for a group or an aggregate function used in the select statement. The WHERE clause selects before grouping. The HAVING clause selects rows after grouping. Unlike the HAVING clause, the WHERE clause cannot contain aggregate functions. See [Having vs Where Clause?](https://www.geeksforgeeks.org/having-vs-where-clause-in-sql)

**Difference between having and where clause**

The WHERE and HAVING clauses in SQL are both used to filter records, but they serve different purposes and are used in different contexts.

### ****WHERE Clause****

* **Purpose**: The WHERE clause is used to filter records before any grouping operations are performed. It applies to rows in the database and restricts the number of rows returned by the query based on specified conditions.
* **Usage**: The WHERE clause is typically used with SELECT, UPDATE, DELETE, etc.
* **Conditions**: The WHERE clause cannot be used with aggregate functions like COUNT(), SUM(), AVG(), etc.

**Example:**

SELECT \* FROM employees WHERE department\_id = 1;

### ****HAVING Clause****

* **Purpose**: The HAVING clause is used to filter groups of rows after the GROUP BY operation has been performed. It is often used to filter aggregated data.
* **Usage**: The HAVING clause is used with SELECT queries that include GROUP BY.
* **Conditions**: The HAVING clause can include aggregate functions.

**Example:**

SELECT d\_id, COUNT(\*) AS employee\_count FROM employees GROUP BY d\_id HAVING COUNT(\*) > 10;

* This query selects departments with more than 10 employees, showing the department ID and the count of employees.

**Explain SQL AND & OR statement with an example?**

In SQL, the AND & OR operators are used for filtering the data and getting precise results based on conditions. The AND and OR operators are used with the WHERE clause.

These **two operators** are called **conjunctive operators**.

**AND Operator:**This operator displays only those records where both conditions **condition 1 and condition 2 evaluate to True.**

**OR Operator:**This operator displays the records where either one of the conditions condition 1 and condition 2 evaluates to True. That is,**either condition1 is True or condition2 is True.**

For more details please read the [SQL | AND and OR](https://www.geeksforgeeks.org/sql-and-and-or-operators)operators article.

**Define BETWEEN statements in SQL?**

The SQL BETWEEN condition allows you to easily test if an expression is within a range of values (inclusive). The values can be text, date, or numbers. It can be used in a SELECT, INSERT, UPDATE, or DELETE statement. The SQL BETWEEN Condition will return the records where the expression is within the range of value1 and value2.

For more details please read [SQL | Between & I operator](https://www.geeksforgeeks.org/sql-between-in-operator) article.

**Why do we use  Commit and Rollback commands?**

| **COMMIT** | **ROLLBACK** |
| --- | --- |
| COMMIT permanently saves the changes made by the current transaction. | ROLLBACK undo the changes made by the current transaction. |
| The transaction can not undo changes after COMMIT execution. | Transaction reaches its previous state after ROLLBACK. |
| When the transaction is successful, COMMIT is applied. | When the transaction is aborted, ROLLBACK occurs. |

**What is the need for group functions in SQL?**

In database management, group functions, also known as aggregate functions,  is a function where the values of multiple rows are grouped together as input on certain criteria to form a single value of more significant meaning.

**Various Group Functions**

1)Count()  
2) Sum()  
3) Avg()  
4) Min()  
5) Max()

For more details please read the [Aggregate functions in the SQL](https://www.geeksforgeeks.org/aggregate-functions-in-sql) article.

**How can you fetch common records from two tables? (Inner Join)**

The below statement could be used to get data from multiple tables, so, we need to use join to get data from multiple tables.

**Syntax :**

*SELECT tablenmae1.colunmname, tablename2.columnnmae*

*FROM tablenmae1*

*JOIN tablename2*

*ON tablenmae1.colunmnam = tablename2.columnnmae*

*ORDER BY columnname;*

**What are the advantages of PL/SQL functions?**

The advantages of PL / SQL functions are as follows:

We can make a single call to the database to run a block of statements. Thus, it improves the performance against running SQL multiple times. This will reduce the number of calls between the database and the application.

We can divide the overall work into small modules which becomes quite manageable, also enhancing the readability of the code.

It promotes reusability.

It is secure since the code stays inside the database, thus hiding internal database details from the application(user). The user only makes a call to the PL/SQL functions. Hence, security and data hiding is ensured.

**What is the SQL query to display the current date?**

CURRENT\_DATE returns to the current date. This function returns the same value if it is executed more than once in a single statement, which means that the value is fixed, even if there is a long delay between fetching rows in a cursor.

Syntax:

*CURRENT\_DATE*

*or*

*CURRENT DATE*

**What are Nested Triggers?**

A trigger can also contain INSERT, UPDATE, and DELETE logic within itself, so when the trigger is fired because of data modification it can also cause another data modification, thereby firing another trigger. A trigger that contains data modification logic within itself is called a nested trigger.

**What is the difference between COALESCE() & ISNULL()?**

**COALESCE():**COALESCE function in SQL returns the first non-NULL expression among its arguments. If all the expressions evaluate to null, then the COALESCE function will return null.  
**Syntax:**

*SELECT column(s), CAOLESCE(expression\_1,….,expression\_n)FROM table\_name;*

**ISNULL():**The ISNULL function has different uses in SQL Server and MySQL. In SQL Server, ISNULL() function is used to replace NULL values.  
**Syntax:**

*SELECT column(s), ISNULL(column\_name, value\_to\_replace)FROM table\_name;*

**SQL queries**

<https://www.edureka.co/blog/interview-questions/sql-query-interview-questions>

<https://artoftesting.com/sql-queries-for-interview>

<https://www.interviewbit.com/sql-query-interview-questions/>