# SQL Tutorial

**SQL** is a database computer language designed for the retrieval and management of data in a relational database. **SQL** stands for **Structured Query Language**. This tutorial will give you a quick start to SQL. It covers most of the topics required for a basic understanding of SQL and to get a feel of how it works.

## Why to Learn SQL?

SQL is Structured Query Language, which is a computer language for storing, manipulating and retrieving data stored in a relational database.

SQL is the standard language for Relational Database System. All the Relational Database Management Systems (RDMS) like MySQL, MS Access, Oracle, Sybase, Informix, Postgres and SQL Server use SQL as their standard database language.

Also, they are using different dialects, such as −

* MS SQL Server using T-SQL,
* Oracle using PL/SQL,
* MS Access version of SQL is called JET SQL (native format) etc.

## Applications of SQL

As mentioned before, SQL is one of the most widely used query language over the databases. I'm going to list few of them here:

* Allows users to access data in the relational database management systems.
* Allows users to describe the data.
* Allows users to define the data in a database and manipulate that data.
* Allows to embed within other languages using SQL modules, libraries & pre-compilers.
* Allows users to create and drop databases and tables.
* Allows users to create view, stored procedure, functions in a database.
* Allows users to set permissions on tables, procedures and views.

## Audience

This **SQL tutorial** is prepared for beginners to help them understand the basic as well as the advanced concepts related to SQL languages. This tutorial will give you enough understanding on the various components of SQL along with suitable examples.

## Prerequisites

Before you start practicing with various types of examples given in this tutorial, I am assuming that you are already aware about what a database is, especially the RDBMS and what is a computer programming language.

# SQL - Overview

SQL is a language to operate databases; it includes database creation, deletion, fetching rows, modifying rows, etc. SQL is an **ANSI** (American National Standards Institute) standard language, but there are many different versions of the SQL language.

## What is SQL?

SQL is Structured Query Language, which is a computer language for storing, manipulating and retrieving data stored in a relational database.

SQL is the standard language for Relational Database System. All the Relational Database Management Systems (RDMS) like MySQL, MS Access, Oracle, Sybase, Informix, Postgres and SQL Server use SQL as their standard database language.

Also, they are using different dialects, such as −

* MS SQL Server using T-SQL,
* Oracle using PL/SQL,
* MS Access version of SQL is called JET SQL (native format) etc.

## Why SQL?

SQL is widely popular because it offers the following advantages −

* Allows users to access data in the relational database management systems.
* Allows users to describe the data.
* Allows users to define the data in a database and manipulate that data.
* Allows to embed within other languages using SQL modules, libraries & pre-compilers.
* Allows users to create and drop databases and tables.
* Allows users to create view, stored procedure, functions in a database.
* Allows users to set permissions on tables, procedures and views.

## A Brief History of SQL

* **1970** − Dr. Edgar F. "Ted" Codd of IBM is known as the father of relational databases. He described a relational model for databases.
* **1974** − Structured Query Language appeared.
* **1978** − IBM worked to develop Codd's ideas and released a product named System/R.
* **1986** − IBM developed the first prototype of relational database and standardized by ANSI. The first relational database was released by Relational Software which later came to be known as Oracle.

## SQL Process

When you are executing an SQL command for any RDBMS, the system determines the best way to carry out your request and SQL engine figures out how to interpret the task.

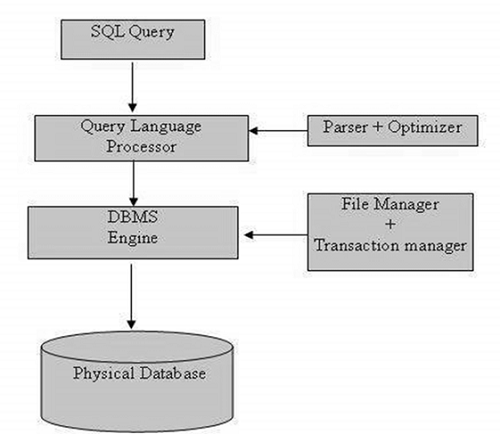
There are various components included in this process.

These components are −

* Query Dispatcher
* Optimization Engines
* Classic Query Engine
* SQL Query Engine, etc.

A classic query engine handles all the non-SQL queries, but a SQL query engine won't handle logical files.

Following is a simple diagram showing the SQL Architecture −



## SQL Commands

The standard SQL commands to interact with relational databases are CREATE, SELECT, INSERT, UPDATE, DELETE and DROP. These commands can be classified into the following groups based on their nature −

### DDL - Data Definition Language

|  |  |
| --- | --- |
| **Sr.No.** | **Command & Description** |
| 1 | **CREATE**  Creates a new table, a view of a table, or other object in the database. |
| 2 | **ALTER**  Modifies an existing database object, such as a table. |
| 3 | **DROP**  Deletes an entire table, a view of a table or other objects in the database. |

### DML - Data Manipulation Language

|  |  |
| --- | --- |
| **Sr.No.** | **Command & Description** |
| 1 | **SELECT**  Retrieves certain records from one or more tables. |
| 2 | **INSERT**  Creates a record. |
| 3 | **UPDATE**  Modifies records. |
| 4 | **DELETE**  Deletes records. |

### DCL - Data Control Language

|  |  |
| --- | --- |
| **Sr.No.** | **Command & Description** |
| 1 | **GRANT**  Gives a privilege to user. |
| 2 | **REVOKE**  Takes back privileges granted from user. |

SQL - RDBMS Concepts

## What is RDBMS?

RDBMS stands for **R**elational **D**atabase **M**anagement **S**ystem. RDBMS is the basis for SQL, and for all modern database systems like MS SQL Server, IBM DB2, Oracle, MySQL, and Microsoft Access.

A Relational database management system (RDBMS) is a database management system (DBMS) that is based on the relational model as introduced by E. F. Codd.

## What is a table?

The data in an RDBMS is stored in database objects which are called as **tables**. This table is basically a collection of related data entries and it consists of numerous columns and rows.

Remember, a table is the most common and simplest form of data storage in a relational database. The following program is an example of a CUSTOMERS table −

+----+----------+-----+-----------+----------+

| ID | NAME | AGE | ADDRESS | SALARY |

+----+----------+-----+-----------+----------+

| 1 | Ramesh | 32 | Ahmedabad | 2000.00 |

| 2 | Khilan | 25 | Delhi | 1500.00 |

| 3 | kaushik | 23 | Kota | 2000.00 |

| 4 | Chaitali | 25 | Mumbai | 6500.00 |

| 5 | Hardik | 27 | Bhopal | 8500.00 |

| 6 | Komal | 22 | MP | 4500.00 |

| 7 | Muffy | 24 | Indore | 10000.00 |

+----+----------+-----+-----------+----------+

## What is a field?

Every table is broken up into smaller entities called fields. The fields in the CUSTOMERS table consist of ID, NAME, AGE, ADDRESS and SALARY.

A field is a column in a table that is designed to maintain specific information about every record in the table.

## What is a Record or a Row?

A record is also called as a row of data is each individual entry that exists in a table. For example, there are 7 records in the above CUSTOMERS table. Following is a single row of data or record in the CUSTOMERS table −

+----+----------+-----+-----------+----------+

| 1 | Ramesh | 32 | Ahmedabad | 2000.00 |

+----+----------+-----+-----------+----------+

A record is a horizontal entity in a table.

## What is a column?

A column is a vertical entity in a table that contains all information associated with a specific field in a table.

For example, a column in the CUSTOMERS table is ADDRESS, which represents location description and would be as shown below −

+-----------+

| ADDRESS |

+-----------+

| Ahmedabad |

| Delhi |

| Kota |

| Mumbai |

| Bhopal |

| MP |

| Indore |

+----+------+

## What is a NULL value?

A NULL value in a table is a value in a field that appears to be blank, which means a field with a NULL value is a field with no value.

It is very important to understand that a NULL value is different than a zero value or a field that contains spaces. A field with a NULL value is the one that has been left blank during a record creation.

## SQL Constraints

Constraints are the rules enforced on data columns on a table. These 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 database.

Constraints can either be column level or table level. Column level constraints are applied only to one column whereas, table level constraints are applied to the entire table.

Following are some of the most commonly used constraints available in SQL −

* [NOT NULL Constraint](https://www.tutorialspoint.com/sql/sql-not-null.htm) − Ensures that a column cannot have a NULL value.
* [DEFAULT Constraint](https://www.tutorialspoint.com/sql/sql-default.htm) − Provides a default value for a column when none is specified.
* [UNIQUE Constraint](https://www.tutorialspoint.com/sql/sql-unique.htm) − Ensures that all the values in a column are different.
* [PRIMARY Key](https://www.tutorialspoint.com/sql/sql-primary-key.htm) − Uniquely identifies each row/record in a database table.
* [FOREIGN Key](https://www.tutorialspoint.com/sql/sql-foreign-key.htm) − Uniquely identifies a row/record in any another database table.
* [CHECK Constraint](https://www.tutorialspoint.com/sql/sql-check.htm) − The CHECK constraint ensures that all values in a column satisfy certain conditions.
* [INDEX](https://www.tutorialspoint.com/sql/sql-index.htm) − Used to create and retrieve data from the database very quickly.

## Data Integrity

The following categories of data integrity exist with each RDBMS −

* **Entity Integrity −** There are no duplicate rows in a table.
* **Domain Integrity −** Enforces valid entries for a given column by restricting the type, the format, or the range of values.
* **Referential integrity −** Rows cannot be deleted, which are used by other records.
* **User-Defined Integrity −** Enforces some specific business rules that do not fall into entity, domain or referential integrity.

## Database Normalization

Database normalization is the process of efficiently organizing data in a database. There are two reasons of this normalization process −

* Eliminating redundant data, for example, storing the same data in more than one table.
* Ensuring data dependencies make sense.

Both these reasons are worthy goals as they reduce the amount of space a database consumes and ensures that data is logically stored. Normalization consists of a series of guidelines that help guide you in creating a good database structure.

Normalization guidelines are divided into normal forms; think of a form as the format or the way a database structure is laid out. The aim of normal forms is to organize the database structure, so that it complies with the rules of first normal form, then second normal form and finally the third normal form.

It is your choice to take it further and go to the fourth normal form, fifth normal form and so on, but in general, the third normal form is more than enough.

* [First Normal Form (1NF)](https://www.tutorialspoint.com/sql/first-normal-form.htm)
* [Second Normal Form (2NF)](https://www.tutorialspoint.com/sql/second-normal-form.htm)
* [Third Normal Form (3NF)](https://www.tutorialspoint.com/sql/third-normal-form.htm)

SQL - Syntax

SQL is followed by a unique set of rules and guidelines called Syntax. This tutorial gives you a quick start with SQL by listing all the basic SQL Syntax.

All the SQL statements start with any of the keywords like SELECT, INSERT, UPDATE, DELETE, ALTER, DROP, CREATE, USE, SHOW and all the statements end with a semicolon (;).

The most important point to be noted here is that SQL is case insensitive, which means SELECT and select have same meaning in SQL statements. Whereas, MySQL makes difference in table names. So, if you are working with MySQL, then you need to give table names as they exist in the database.

## Various Syntax in SQL

All the examples given in this tutorial have been tested with a MySQL server.

### SQL SELECT Statement

SELECT column1, column2....columnN

FROM table\_name;

### SQL DISTINCT Clause

SELECT DISTINCT column1, column2....columnN

FROM table\_name;

### SQL WHERE Clause

SELECT column1, column2....columnN

FROM table\_name

WHERE CONDITION;

### SQL AND/OR Clause

SELECT column1, column2....columnN

FROM table\_name

WHERE CONDITION-1 {AND|OR} CONDITION-2;

### SQL IN Clause

SELECT column1, column2....columnN

FROM table\_name

WHERE column\_name IN (val-1, val-2,...val-N);

### SQL BETWEEN Clause

SELECT column1, column2....columnN

FROM table\_name

WHERE column\_name BETWEEN val-1 AND val-2;

### SQL LIKE Clause

SELECT column1, column2....columnN

FROM table\_name

WHERE column\_name LIKE { PATTERN };

### SQL ORDER BY Clause

SELECT column1, column2....columnN

FROM table\_name

WHERE CONDITION

ORDER BY column\_name {ASC|DESC};

### SQL GROUP BY Clause

SELECT SUM(column\_name)

FROM table\_name

WHERE CONDITION

GROUP BY column\_name;

### SQL COUNT Clause

SELECT COUNT(column\_name)

FROM table\_name

WHERE CONDITION;

### SQL HAVING Clause

SELECT SUM(column\_name)

FROM table\_name

WHERE CONDITION

GROUP BY column\_name

HAVING (arithematic function condition);

### SQL CREATE TABLE Statement

CREATE TABLE table\_name(

column1 datatype,

column2 datatype,

column3 datatype,

.....

columnN datatype,

PRIMARY KEY( one or more columns )

);

### SQL DROP TABLE Statement

DROP TABLE table\_name;

### SQL CREATE INDEX Statement

CREATE UNIQUE INDEX index\_name

ON table\_name ( column1, column2,...columnN);

### SQL DROP INDEX Statement

ALTER TABLE table\_name

DROP INDEX index\_name;

### SQL DESC Statement

DESC table\_name;

### SQL TRUNCATE TABLE Statement

TRUNCATE TABLE table\_name;

### SQL ALTER TABLE Statement

ALTER TABLE table\_name {ADD|DROP|MODIFY} column\_name {data\_ype};

### SQL ALTER TABLE Statement (Rename)

ALTER TABLE table\_name RENAME TO new\_table\_name;

### SQL INSERT INTO Statement

INSERT INTO table\_name( column1, column2....columnN)

VALUES ( value1, value2....valueN);

### SQL UPDATE Statement

UPDATE table\_name

SET column1 = value1, column2 = value2....columnN=valueN

[ WHERE CONDITION ];

### SQL DELETE Statement

DELETE FROM table\_name

WHERE {CONDITION};

### SQL CREATE DATABASE Statement

CREATE DATABASE database\_name;

### SQL DROP DATABASE Statement

DROP DATABASE database\_name;

### SQL USE Statement

USE database\_name;

### SQL COMMIT Statement

COMMIT;

### SQL ROLLBACK Statement

ROLLBACK;

SQL - Data Types

SQL Data Type is an attribute that specifies the type of data of any object. Each column, variable and expression has a related data type in SQL. You can use these data types while creating your tables. You can choose a data type for a table column based on your requirement.

SQL Server offers six categories of data types for your use which are listed below −

## Exact Numeric Data Types

|  |  |  |
| --- | --- | --- |
| **DATA TYPE** | **FROM** | **TO** |
| bigint | -9,223,372,036,854,775,808 | 9,223,372,036,854,775,807 |
| int | -2,147,483,648 | 2,147,483,647 |
| smallint | -32,768 | 32,767 |
| tinyint | 0 | 255 |
| bit | 0 | 1 |
| decimal | -10^38 +1 | 10^38 -1 |
| numeric | -10^38 +1 | 10^38 -1 |
| money | -922,337,203,685,477.5808 | +922,337,203,685,477.5807 |
| smallmoney | -214,748.3648 | +214,748.3647 |

## Approximate Numeric Data Types

|  |  |  |
| --- | --- | --- |
| **DATA TYPE** | **FROM** | **TO** |
| float | -1.79E + 308 | 1.79E + 308 |
| real | -3.40E + 38 | 3.40E + 38 |

## Date and Time Data Types

|  |  |  |
| --- | --- | --- |
| **DATA TYPE** | **FROM** | **TO** |
| datetime | Jan 1, 1753 | Dec 31, 9999 |
| smalldatetime | Jan 1, 1900 | Jun 6, 2079 |
| date | Stores a date like June 30, 1991 | |
| time | Stores a time of day like 12:30 P.M. | |

**Note** − Here, datetime has 3.33 milliseconds accuracy where as smalldatetime has 1 minute accuracy.

## Character Strings Data Types

|  |  |
| --- | --- |
| **Sr.No.** | **DATA TYPE & Description** |
| 1 | **char**  Maximum length of 8,000 characters.( Fixed length non-Unicode characters) |
| 2 | **varchar**  Maximum of 8,000 characters.(Variable-length non-Unicode data). |
| 3 | **varchar(max)**  Maximum length of 2E + 31 characters, Variable-length non-Unicode data (SQL Server 2005 only). |
| 4 | **text**  Variable-length non-Unicode data with a maximum length of 2,147,483,647 characters. |

## Unicode Character Strings Data Types

|  |  |
| --- | --- |
| **Sr.No.** | **DATA TYPE & Description** |
| 1 | **nchar**  Maximum length of 4,000 characters.( Fixed length Unicode) |
| 2 | **nvarchar**  Maximum length of 4,000 characters.(Variable length Unicode) |
| 3 | **nvarchar(max)**  Maximum length of 2E + 31 characters (SQL Server 2005 only).( Variable length Unicode) |
| 4 | **ntext**  Maximum length of 1,073,741,823 characters. ( Variable length Unicode ) |

## Binary Data Types

|  |  |
| --- | --- |
| **Sr.No.** | **DATA TYPE & Description** |
| 1 | **binary**  Maximum length of 8,000 bytes(Fixed-length binary data ) |
| 2 | **varbinary**  Maximum length of 8,000 bytes.(Variable length binary data) |
| 3 | **varbinary(max)**  Maximum length of 2E + 31 bytes (SQL Server 2005 only). ( Variable length Binary data) |
| 4 | **image**  Maximum length of 2,147,483,647 bytes. ( Variable length Binary Data) |

## Misc Data Types

|  |  |
| --- | --- |
| **Sr.No.** | **DATA TYPE & Description** |
| 1 | **sql\_variant**  Stores values of various SQL Server-supported data types, except text, ntext, and timestamp. |
| 2 | **timestamp**  Stores a database-wide unique number that gets updated every time a row gets updated |
| 3 | **uniqueidentifier**  Stores a globally unique identifier (GUID) |
| 4 | **xml**  Stores XML data. You can store xml instances in a column or a variable (SQL Server 2005 only). |
| 5 | **cursor**  Reference to a cursor object |
| 6 | **table**  Stores a result set for later processing |

SQL - Operators

## What is an Operator in SQL?

An operator is a reserved word or a character used primarily in an SQL statement's WHERE clause to perform operation(s), such as comparisons and arithmetic operations. These Operators are used to specify conditions in an SQL statement and to serve as conjunctions for multiple conditions in a statement.

* Arithmetic operators
* Comparison operators
* Logical operators
* Operators used to negate conditions

## SQL Arithmetic Operators

Assume **'variable a'** holds 10 and **'variable b'** holds 20, then −

[Show Examples](https://www.tutorialspoint.com/sql/sql-arithmetic-operators.htm)

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| + (Addition) | Adds values on either side of the operator. | a + b will give 30 |
| - (Subtraction) | Subtracts right hand operand from left hand operand. | a - b will give -10 |
| \* (Multiplication) | Multiplies values on either side of the operator. | a \* b will give 200 |
| / (Division) | Divides left hand operand by right hand operand. | b / a will give 2 |
| % (Modulus) | Divides left hand operand by right hand operand and returns remainder. | b % a will give 0 |

## SQL Comparison Operators

Assume **'variable a'** holds 10 and **'variable b'** holds 20, then −

[Show Examples](https://www.tutorialspoint.com/sql/sql-comparison-operators.htm)

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

## SQL Logical Operators

Here is a list of all the logical operators available in SQL.

[Show Examples](https://www.tutorialspoint.com/sql/sql-logical-operators.htm)

|  |  |
| --- | --- |
| **Sr.No.** | **Operator & Description** |
| 1 | **ALL**  The ALL operator is used to compare a value to all values in another value set. |
| 2 | **AND**  The AND operator allows the existence of multiple conditions in an SQL statement's WHERE clause. |
| 3 | **ANY**  The ANY operator is used to compare a value to any applicable value in the list as per the condition. |
| 4 | **BETWEEN**  The BETWEEN operator is used to search for values that are within a set of values, given the minimum value and the maximum value. |
| 5 | **EXISTS**  The EXISTS operator is used to search for the presence of a row in a specified table that meets a certain criterion. |
| 6 | **IN**  The IN operator is used to compare a value to a list of literal values that have been specified. |
| 7 | **LIKE**  The LIKE operator is used to compare a value to similar values using wildcard operators. |
| 8 | **NOT**  The NOT operator reverses the meaning of the logical operator with which it is used. Eg: NOT EXISTS, NOT BETWEEN, NOT IN, etc. **This is a negate operator.** |
| 9 | **OR**  The OR operator is used to combine multiple conditions in an SQL statement's WHERE clause. |
| 10 | **IS NULL**  The NULL operator is used to compare a value with a NULL value. |
| 11 | **UNIQUE**  The UNIQUE operator searches every row of a specified table for uniqueness (no duplicates). |

SQL - Expressions

An expression is a combination of one or more values, operators and SQL functions that evaluate to a value. These SQL EXPRESSIONs are like formulae and they are written in query language. You can also use them to query the database for a specific set of data.

### Syntax

Consider the basic syntax of the SELECT statement as follows −

SELECT column1, column2, columnN

FROM table\_name

WHERE [CONDITION|EXPRESSION];

There are different types of SQL expressions, which are mentioned below −

* Boolean
* Numeric
* Date

Let us now discuss each of these in detail.

## Boolean Expressions

SQL Boolean Expressions fetch the data based on matching a single value. Following is the syntax −

SELECT column1, column2, columnN

FROM table\_name

WHERE SINGLE VALUE MATCHING EXPRESSION;

Consider the CUSTOMERS table having the following records −

SQL> SELECT \* FROM CUSTOMERS;

+----+----------+-----+-----------+----------+

| ID | NAME | AGE | ADDRESS | SALARY |

+----+----------+-----+-----------+----------+

| 1 | Ramesh | 32 | Ahmedabad | 2000.00 |

| 2 | Khilan | 25 | Delhi | 1500.00 |

| 3 | kaushik | 23 | Kota | 2000.00 |

| 4 | Chaitali | 25 | Mumbai | 6500.00 |

| 5 | Hardik | 27 | Bhopal | 8500.00 |

| 6 | Komal | 22 | MP | 4500.00 |

| 7 | Muffy | 24 | Indore | 10000.00 |

+----+----------+-----+-----------+----------+

7 rows in set (0.00 sec)

The following table is a simple example showing the usage of various SQL Boolean Expressions −

SQL> SELECT \* FROM CUSTOMERS WHERE SALARY = 10000;

+----+-------+-----+---------+----------+

| ID | NAME | AGE | ADDRESS | SALARY |

+----+-------+-----+---------+----------+

| 7 | Muffy | 24 | Indore | 10000.00 |

+----+-------+-----+---------+----------+

1 row in set (0.00 sec)

## Numeric Expression

These expressions are used to perform any mathematical operation in any query. Following is the syntax −

SELECT numerical\_expression as OPERATION\_NAME

[FROM table\_name

WHERE CONDITION] ;

Here, the numerical\_expression is used for a mathematical expression or any formula. Following is a simple example showing the usage of SQL Numeric Expressions −

SQL> SELECT (15 + 6) AS ADDITION

+----------+

| ADDITION |

+----------+

| 21 |

+----------+

1 row in set (0.00 sec)

There are several built-in functions like avg(), sum(), count(), etc., to perform what is known as the aggregate data calculations against a table or a specific table column.

SQL> SELECT COUNT(\*) AS "RECORDS" FROM CUSTOMERS;

+---------+

| RECORDS |

+---------+

| 7 |

+---------+

1 row in set (0.00 sec)

## Date Expressions

Date Expressions return current system date and time values −

SQL> SELECT CURRENT\_TIMESTAMP;

+---------------------+

| Current\_Timestamp |

+---------------------+

| 2009-11-12 06:40:23 |

+---------------------+

1 row in set (0.00 sec)

Another date expression is as shown below −

SQL> SELECT GETDATE();;

+-------------------------+

| GETDATE |

+-------------------------+

| 2009-10-22 12:07:18.140 |

+-------------------------+

1 row in set (0.00 sec)

SQL - CREATE Database

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

## Syntax

The basic syntax of this CREATE DATABASE statement is as follows −

CREATE DATABASE DatabaseName;

Always the database name should be unique within the RDBMS.

## Example

If you want to create a new database <testDB>, then the CREATE DATABASE statement would be as shown below −

SQL> CREATE DATABASE testDB;

Make sure you have the admin privilege before creating any database. Once a database is created, you can check it in the list of databases as follows −

SQL> SHOW DATABASES;

+--------------------+

| Database |

+--------------------+

| information\_schema |

| AMROOD |

| TUTORIALSPOINT |

| mysql |

| orig |

| test |

| testDB |

+--------------------+

7 rows in set (0.00 sec)

SQL - DROP or DELETE Database

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

## Syntax

The basic syntax of DROP DATABASE statement is as follows −

DROP DATABASE DatabaseName;

Always the database name should be unique within the RDBMS.

## Example

If you want to delete an existing database <testDB>, then the DROP DATABASE statement would be as shown below −

SQL> DROP DATABASE testDB;

**NOTE** − Be careful before using this operation because by deleting an existing database would result in loss of complete information stored in the database.

Make sure you have the admin privilege before dropping any database. Once a database is dropped, you can check it in the list of the databases as shown below −

SQL> SHOW DATABASES;

+--------------------+

| Database |

+--------------------+

| information\_schema |

| AMROOD |

| TUTORIALSPOINT |

| mysql |

| orig |

| test |

+--------------------+

6 rows in set (0.00 sec)

SQL - SELECT Database, USE Statement

When you have multiple databases in your SQL Schema, then before starting your operation, you would need to select a database where all the operations would be performed.

The SQL **USE** statement is used to select any existing database in the SQL schema.

## Syntax

The basic syntax of the USE statement is as shown below −

USE DatabaseName;

Always the database name should be unique within the RDBMS.

## Example

You can check the available databases as shown below −

SQL> SHOW DATABASES;

+--------------------+

| Database |

+--------------------+

| information\_schema |

| AMROOD |

| TUTORIALSPOINT |

| mysql |

| orig |

| test |

+--------------------+

6 rows in set (0.00 sec)

Now, if you want to work with the AMROOD database, then you can execute the following SQL command and start working with the AMROOD database.

SQL> USE AMROOD;

SQL - CREATE Table

Creating a basic table involves naming the table and defining its columns and each column's data type.

The SQL **CREATE TABLE** statement is used to create a new table.

## Syntax

The basic syntax of the CREATE TABLE statement is as follows −

CREATE TABLE table\_name(

column1 datatype,

column2 datatype,

column3 datatype,

.....

columnN datatype,

PRIMARY KEY( one or more columns )

);

CREATE TABLE is the keyword telling the database system what you want to do. In this case, you want to create a new table. The unique name or identifier for the table follows the CREATE TABLE statement.

Then in brackets comes the list defining each column in the table and what sort of data type it is. The syntax becomes clearer with the following example.

A copy of an existing table can be created using a combination of the CREATE TABLE statement and the SELECT statement. You can check the complete details at [Create Table Using another Table.](https://www.tutorialspoint.com/sql/sql-create-table-using-tables.htm)

## Example

The following code block is an example, which creates a CUSTOMERS table with an ID as a primary key and NOT NULL are the constraints showing that these fields cannot be NULL while creating records in this table −

SQL> CREATE TABLE CUSTOMERS(

ID INT NOT NULL,

NAME VARCHAR (20) NOT NULL,

AGE INT NOT NULL,

ADDRESS CHAR (25) ,

SALARY DECIMAL (18, 2),

PRIMARY KEY (ID)

);

You can verify if your table has been created successfully by looking at the message displayed by the SQL server, otherwise you can use the **DESC** command as follows −

SQL> DESC CUSTOMERS;

+---------+---------------+------+-----+---------+-------+

| Field | Type | Null | Key | Default | Extra |

+---------+---------------+------+-----+---------+-------+

| ID | int(11) | NO | PRI | | |

| NAME | varchar(20) | NO | | | |

| AGE | int(11) | NO | | | |

| ADDRESS | char(25) | YES | | NULL | |

| SALARY | decimal(18,2) | YES | | NULL | |

+---------+---------------+------+-----+---------+-------+

5 rows in set (0.00 sec)

Now, you have CUSTOMERS table available in your database which you can use to store the required information related to customers.

SQL - DROP or DELETE Table

The SQL **DROP TABLE** statement is used to remove a table definition and all the data, indexes, triggers, constraints and permission specifications for that table.

**NOTE** − You should be very careful while using this command because once a table is deleted then all the information available in that table will also be lost forever.

## Syntax

The basic syntax of this DROP TABLE statement is as follows −

DROP TABLE table\_name;

## Example

Let us first verify the CUSTOMERS table and then we will delete it from the database as shown below −

SQL> DESC CUSTOMERS;

+---------+---------------+------+-----+---------+-------+

| Field | Type | Null | Key | Default | Extra |

+---------+---------------+------+-----+---------+-------+

| ID | int(11) | NO | PRI | | |

| NAME | varchar(20) | NO | | | |

| AGE | int(11) | NO | | | |

| ADDRESS | char(25) | YES | | NULL | |

| SALARY | decimal(18,2) | YES | | NULL | |

+---------+---------------+------+-----+---------+-------+

5 rows in set (0.00 sec)

This means that the CUSTOMERS table is available in the database, so let us now drop it as shown below.

SQL> DROP TABLE CUSTOMERS;

Query OK, 0 rows affected (0.01 sec)

Now, if you would try the DESC command, then you will get the following error −

SQL> DESC CUSTOMERS;

ERROR 1146 (42S02): Table 'TEST.CUSTOMERS' doesn't exist

Here, TEST is the database name which we are using for our examples.

SQL - INSERT Query

he SQL **INSERT INTO** Statement is used to add new rows of data to a table in the database.

### Syntax

There are two basic syntaxes of the INSERT INTO statement which are shown below.

INSERT INTO TABLE\_NAME (column1, column2, column3,...columnN)

VALUES (value1, value2, value3,...valueN);

Here, column1, column2, column3,...columnN are the names of the columns in the table into which you want to insert the data.

You may not need to specify the column(s) name in the SQL query if you are adding values for all the columns of the table. But make sure the order of the values is in the same order as the columns in the table.

The **SQL INSERT INTO** syntax will be as follows −

INSERT INTO TABLE\_NAME VALUES (value1,value2,value3,...valueN);

### Example

The following statements would create six records in the CUSTOMERS table.

INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)

VALUES (1, 'Ramesh', 32, 'Ahmedabad', 2000.00 );

INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)

VALUES (2, 'Khilan', 25, 'Delhi', 1500.00 );

INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)

VALUES (3, 'kaushik', 23, 'Kota', 2000.00 );

INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)

VALUES (4, 'Chaitali', 25, 'Mumbai', 6500.00 );

INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)

VALUES (5, 'Hardik', 27, 'Bhopal', 8500.00 );

INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)

VALUES (6, 'Komal', 22, 'MP', 4500.00 );

You can create a record in the CUSTOMERS table by using the second syntax as shown below.

INSERT INTO CUSTOMERS

VALUES (7, 'Muffy', 24, 'Indore', 10000.00 );

All the above statements would produce the following records in the CUSTOMERS table as shown below.

+----+----------+-----+-----------+----------+

| ID | NAME | AGE | ADDRESS | SALARY |

+----+----------+-----+-----------+----------+

| 1 | Ramesh | 32 | Ahmedabad | 2000.00 |

| 2 | Khilan | 25 | Delhi | 1500.00 |

| 3 | kaushik | 23 | Kota | 2000.00 |

| 4 | Chaitali | 25 | Mumbai | 6500.00 |

| 5 | Hardik | 27 | Bhopal | 8500.00 |

| 6 | Komal | 22 | MP | 4500.00 |

| 7 | Muffy | 24 | Indore | 10000.00 |

+----+----------+-----+-----------+----------+

## Populate one table using another table

You can populate the data into a table through the select statement over another table; provided the other table has a set of fields, which are required to populate the first table.

Here is the syntax −

INSERT INTO first\_table\_name [(column1, column2, ... columnN)]

SELECT column1, column2, ...columnN

FROM second\_table\_name

[WHERE condition];

SQL - SELECT Query

The SQL **SELECT** statement is used to fetch the data from a database table which returns this data in the form of a result table. These result tables are called result-sets.

## Syntax

The basic syntax of the SELECT statement is as follows −

SELECT column1, column2, columnN FROM table\_name;

Here, column1, column2... are the fields of a table whose values you want to fetch. If you want to fetch all the fields available in the field, then you can use the following syntax.

SELECT \* FROM table\_name;

## Example

Consider the CUSTOMERS table having the following records −

+----+----------+-----+-----------+----------+

| ID | NAME | AGE | ADDRESS | SALARY |

+----+----------+-----+-----------+----------+

| 1 | Ramesh | 32 | Ahmedabad | 2000.00 |

| 2 | Khilan | 25 | Delhi | 1500.00 |

| 3 | kaushik | 23 | Kota | 2000.00 |

| 4 | Chaitali | 25 | Mumbai | 6500.00 |

| 5 | Hardik | 27 | Bhopal | 8500.00 |

| 6 | Komal | 22 | MP | 4500.00 |

| 7 | Muffy | 24 | Indore | 10000.00 |

+----+----------+-----+-----------+----------+

The following code is an example, which would fetch the ID, Name and Salary fields of the customers available in CUSTOMERS table.

SQL> SELECT ID, NAME, SALARY FROM CUSTOMERS;

This would produce the following result −

+----+----------+----------+

| ID | NAME | SALARY |

+----+----------+----------+

| 1 | Ramesh | 2000.00 |

| 2 | Khilan | 1500.00 |

| 3 | kaushik | 2000.00 |

| 4 | Chaitali | 6500.00 |

| 5 | Hardik | 8500.00 |

| 6 | Komal | 4500.00 |

| 7 | Muffy | 10000.00 |

+----+----------+----------+

If you want to fetch all the fields of the CUSTOMERS table, then you should use the following query.

SQL> SELECT \* FROM CUSTOMERS;

This would produce the result as shown below.

+----+----------+-----+-----------+----------+

| ID | NAME | AGE | ADDRESS | SALARY |

+----+----------+-----+-----------+----------+

| 1 | Ramesh | 32 | Ahmedabad | 2000.00 |

| 2 | Khilan | 25 | Delhi | 1500.00 |

| 3 | kaushik | 23 | Kota | 2000.00 |

| 4 | Chaitali | 25 | Mumbai | 6500.00 |

| 5 | Hardik | 27 | Bhopal | 8500.00 |

| 6 | Komal | 22 | MP | 4500.00 |

| 7 | Muffy | 24 | Indore | 10000.00 |

+----+----------+-----+-----------+----------+

[**Previous Page**](https://www.tutorialspoint.com/sql/sql-insert-query.htm)

SQL - WHERE Clause

The SQL **WHERE** clause is used to specify a condition while fetching the data from a single table or by joining with multiple tables. If the given condition is satisfied, then only it returns a specific value from the table. You should use the WHERE clause to filter the records and fetching only the necessary records.

The WHERE clause is not only used in the SELECT statement, but it is also used in the UPDATE, DELETE statement, etc., which we would examine in the subsequent chapters.

## Syntax

The basic syntax of the SELECT statement with the WHERE clause is as shown below.

SELECT column1, column2, columnN

FROM table\_name

WHERE [condition]

You can specify a condition using the [comparison or logical operators](https://www.tutorialspoint.com/sql/sql-operators.htm) like >, <, =, **LIKE, NOT**, etc. The following examples would make this concept clear.

## Example

Consider the CUSTOMERS table having the following records −

+----+----------+-----+-----------+----------+

| ID | NAME | AGE | ADDRESS | SALARY |

+----+----------+-----+-----------+----------+

| 1 | Ramesh | 32 | Ahmedabad | 2000.00 |

| 2 | Khilan | 25 | Delhi | 1500.00 |

| 3 | kaushik | 23 | Kota | 2000.00 |

| 4 | Chaitali | 25 | Mumbai | 6500.00 |

| 5 | Hardik | 27 | Bhopal | 8500.00 |

| 6 | Komal | 22 | MP | 4500.00 |

| 7 | Muffy | 24 | Indore | 10000.00 |

+----+----------+-----+-----------+----------+

The following code is an example which would fetch the ID, Name and Salary fields from the CUSTOMERS table, where the salary is greater than 2000 −

SQL> SELECT ID, NAME, SALARY

FROM CUSTOMERS

WHERE SALARY > 2000;

This would produce the following result −

+----+----------+----------+

| ID | NAME | SALARY |

+----+----------+----------+

| 4 | Chaitali | 6500.00 |

| 5 | Hardik | 8500.00 |

| 6 | Komal | 4500.00 |

| 7 | Muffy | 10000.00 |

+----+----------+----------+

The following query is an example, which would fetch the ID, Name and Salary fields from the CUSTOMERS table for a customer with the name **Hardik**.

Here, it is important to note that all the strings should be given inside single quotes (''). Whereas, numeric values should be given without any quote as in the above example.

SQL> SELECT ID, NAME, SALARY

FROM CUSTOMERS

WHERE NAME = 'Hardik';

This would produce the following result −

+----+----------+----------+

| ID | NAME | SALARY |

+----+----------+----------+

| 5 | Hardik | 8500.00 |

+----+----------+----------+

SQL - AND and OR Conjunctive Operators

The SQL **AND** & **OR** operators are used to combine multiple conditions to narrow data in an SQL statement. These two operators are called as the conjunctive operators.

These operators provide a means to make multiple comparisons with different operators in the same SQL statement.

## The AND Operator

The **AND** operator allows the existence of multiple conditions in an SQL statement's WHERE clause.

### Syntax

The basic syntax of the AND operator with a WHERE clause is as follows −

SELECT column1, column2, columnN

FROM table\_name

WHERE [condition1] AND [condition2]...AND [conditionN];

You can combine N number of conditions using the AND operator. For an action to be taken by the SQL statement, whether it be a transaction or a query, all conditions separated by the AND must be TRUE.

### Example

Consider the CUSTOMERS table having the following records −

+----+----------+-----+-----------+----------+

| ID | NAME | AGE | ADDRESS | SALARY |

+----+----------+-----+-----------+----------+

| 1 | Ramesh | 32 | Ahmedabad | 2000.00 |

| 2 | Khilan | 25 | Delhi | 1500.00 |

| 3 | kaushik | 23 | Kota | 2000.00 |

| 4 | Chaitali | 25 | Mumbai | 6500.00 |

| 5 | Hardik | 27 | Bhopal | 8500.00 |

| 6 | Komal | 22 | MP | 4500.00 |

| 7 | Muffy | 24 | Indore | 10000.00 |

+----+----------+-----+-----------+----------+

Following is an example, which would fetch the ID, Name and Salary fields from the CUSTOMERS table, where the salary is greater than 2000 and the age is less than 25 years −

SQL> SELECT ID, NAME, SALARY

FROM CUSTOMERS

WHERE SALARY > 2000 AND age < 25;

This would produce the following result −

+----+-------+----------+

| ID | NAME | SALARY |

+----+-------+----------+

| 6 | Komal | 4500.00 |

| 7 | Muffy | 10000.00 |

+----+-------+----------+

## The OR Operator

The OR operator is used to combine multiple conditions in an SQL statement's WHERE clause.

### Syntax

The basic syntax of the OR operator with a WHERE clause is as follows −

SELECT column1, column2, columnN

FROM table\_name

WHERE [condition1] OR [condition2]...OR [conditionN]

You can combine N number of conditions using the OR operator. For an action to be taken by the SQL statement, whether it be a transaction or query, the only any ONE of the conditions separated by the OR must be TRUE.

### Example

Consider the CUSTOMERS table having the following records −

+----+----------+-----+-----------+----------+

| ID | NAME | AGE | ADDRESS | SALARY |

+----+----------+-----+-----------+----------+

| 1 | Ramesh | 32 | Ahmedabad | 2000.00 |

| 2 | Khilan | 25 | Delhi | 1500.00 |

| 3 | kaushik | 23 | Kota | 2000.00 |

| 4 | Chaitali | 25 | Mumbai | 6500.00 |

| 5 | Hardik | 27 | Bhopal | 8500.00 |

| 6 | Komal | 22 | MP | 4500.00 |

| 7 | Muffy | 24 | Indore | 10000.00 |

+----+----------+-----+-----------+----------+

The following code block hasa query, which would fetch the ID, Name and Salary fields from the CUSTOMERS table, where the salary is greater than 2000 OR the age is less than 25 years.

SQL> SELECT ID, NAME, SALARY

FROM CUSTOMERS

WHERE SALARY > 2000 OR age < 25;

This would produce the following result −

+----+----------+----------+

| ID | NAME | SALARY |

+----+----------+----------+

| 3 | kaushik | 2000.00 |

| 4 | Chaitali | 6500.00 |

| 5 | Hardik | 8500.00 |

| 6 | Komal | 4500.00 |

| 7 | Muffy | 10000.00 |

+----+----------+----------+

SQL - UPDATE Query

The SQL **UPDATE** Query is used to modify the existing records in a table. You can use the WHERE clause with the UPDATE query to update the selected rows, otherwise all the rows would be affected.

## Syntax

The basic syntax of the UPDATE query with a WHERE clause is as follows −

UPDATE table\_name

SET column1 = value1, column2 = value2...., columnN = valueN

WHERE [condition];

You can combine N number of conditions using the AND or the OR operators.

## Example

Consider the CUSTOMERS table having the following records −

+----+----------+-----+-----------+----------+

| ID | NAME | AGE | ADDRESS | SALARY |

+----+----------+-----+-----------+----------+

| 1 | Ramesh | 32 | Ahmedabad | 2000.00 |

| 2 | Khilan | 25 | Delhi | 1500.00 |

| 3 | kaushik | 23 | Kota | 2000.00 |

| 4 | Chaitali | 25 | Mumbai | 6500.00 |

| 5 | Hardik | 27 | Bhopal | 8500.00 |

| 6 | Komal | 22 | MP | 4500.00 |

| 7 | Muffy | 24 | Indore | 10000.00 |

+----+----------+-----+-----------+----------+

The following query will update the ADDRESS for a customer whose ID number is 6 in the table.

SQL> UPDATE CUSTOMERS

SET ADDRESS = 'Pune'

WHERE ID = 6;

Now, the CUSTOMERS table would have the following records −

+----+----------+-----+-----------+----------+

| ID | NAME | AGE | ADDRESS | SALARY |

+----+----------+-----+-----------+----------+

| 1 | Ramesh | 32 | Ahmedabad | 2000.00 |

| 2 | Khilan | 25 | Delhi | 1500.00 |

| 3 | kaushik | 23 | Kota | 2000.00 |

| 4 | Chaitali | 25 | Mumbai | 6500.00 |

| 5 | Hardik | 27 | Bhopal | 8500.00 |

| 6 | Komal | 22 | Pune | 4500.00 |

| 7 | Muffy | 24 | Indore | 10000.00 |

+----+----------+-----+-----------+----------+

If you want to modify all the ADDRESS and the SALARY column values in the CUSTOMERS table, you do not need to use the WHERE clause as the UPDATE query would be enough as shown in the following code block.

SQL> UPDATE CUSTOMERS

SET ADDRESS = 'Pune', SALARY = 1000.00;

Now, CUSTOMERS table would have the following records −

+----+----------+-----+---------+---------+

| ID | NAME | AGE | ADDRESS | SALARY |

+----+----------+-----+---------+---------+

| 1 | Ramesh | 32 | Pune | 1000.00 |

| 2 | Khilan | 25 | Pune | 1000.00 |

| 3 | kaushik | 23 | Pune | 1000.00 |

| 4 | Chaitali | 25 | Pune | 1000.00 |

| 5 | Hardik | 27 | Pune | 1000.00 |

| 6 | Komal | 22 | Pune | 1000.00 |

| 7 | Muffy | 24 | Pune | 1000.00 |

+----+----------+-----+---------+---------+

SQL - DELETE Query

The SQL DELETE Query is used to delete the existing records from a table.

You can use the WHERE clause with a DELETE query to delete the selected rows, otherwise all the records would be deleted.

## Syntax

The basic syntax of the DELETE query with the WHERE clause is as follows −

DELETE FROM table\_name

WHERE [condition];

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

## Example

Consider the CUSTOMERS table having the following records −

+----+----------+-----+-----------+----------+

| ID | NAME | AGE | ADDRESS | SALARY |

+----+----------+-----+-----------+----------+

| 1 | Ramesh | 32 | Ahmedabad | 2000.00 |

| 2 | Khilan | 25 | Delhi | 1500.00 |

| 3 | kaushik | 23 | Kota | 2000.00 |

| 4 | Chaitali | 25 | Mumbai | 6500.00 |

| 5 | Hardik | 27 | Bhopal | 8500.00 |

| 6 | Komal | 22 | MP | 4500.00 |

| 7 | Muffy | 24 | Indore | 10000.00 |

+----+----------+-----+-----------+----------+

The following code has a query, which will DELETE a customer, whose ID is 6.

SQL> DELETE FROM CUSTOMERS

WHERE ID = 6;

Now, the CUSTOMERS table would have the following records.

+----+----------+-----+-----------+----------+

| ID | NAME | AGE | ADDRESS | SALARY |

+----+----------+-----+-----------+----------+

| 1 | Ramesh | 32 | Ahmedabad | 2000.00 |

| 2 | Khilan | 25 | Delhi | 1500.00 |

| 3 | kaushik | 23 | Kota | 2000.00 |

| 4 | Chaitali | 25 | Mumbai | 6500.00 |

| 5 | Hardik | 27 | Bhopal | 8500.00 |

| 7 | Muffy | 24 | Indore | 10000.00 |

+----+----------+-----+-----------+----------+

If you want to DELETE all the records from the CUSTOMERS table, you do not need to use the WHERE clause and the DELETE query would be as follows −

SQL> DELETE FROM CUSTOMERS;

Now, the CUSTOMERS table would not have any record.

SQL - LIKE Clause

The SQL **LIKE** clause is used to compare a value to similar values using wildcard operators. There are two wildcards used in conjunction with the LIKE operator.

* The percent sign (%)
* The underscore (\_)

The percent sign represents zero, one or multiple characters. The underscore represents a single number or character. These symbols can be used in combinations.

## Syntax

The basic syntax of % and \_ is as follows −

SELECT FROM table\_name

WHERE column LIKE 'XXXX%'

or

SELECT FROM table\_name

WHERE column LIKE '%XXXX%'

or

SELECT FROM table\_name

WHERE column LIKE 'XXXX\_'

or

SELECT FROM table\_name

WHERE column LIKE '\_XXXX'

or

SELECT FROM table\_name

WHERE column LIKE '\_XXXX\_'

You can combine N number of conditions using AND or OR operators. Here, XXXX could be any numeric or string value.

## Example

The following table has a few examples showing the WHERE part having different LIKE clause with '%' and '\_' operators −

|  |  |
| --- | --- |
| **Sr.No.** | **Statement & Description** |
| 1 | **WHERE SALARY LIKE '200%'**  Finds any values that start with 200. |
| 2 | **WHERE SALARY LIKE '%200%'**  Finds any values that have 200 in any position. |
| 3 | **WHERE SALARY LIKE '\_00%'**  Finds any values that have 00 in the second and third positions. |
| 4 | **WHERE SALARY LIKE '2\_%\_%'**  Finds any values that start with 2 and are at least 3 characters in length. |
| 5 | **WHERE SALARY LIKE '%2'**  Finds any values that end with 2. |
| 6 | **WHERE SALARY LIKE '\_2%3'**  Finds any values that have a 2 in the second position and end with a 3. |
| 7 | **WHERE SALARY LIKE '2\_\_\_3'**  Finds any values in a five-digit number that start with 2 and end with 3. |

Let us take a real example, consider the CUSTOMERS table having the records as shown below.

+----+----------+-----+-----------+----------+

| ID | NAME | AGE | ADDRESS | SALARY |

+----+----------+-----+-----------+----------+

| 1 | Ramesh | 32 | Ahmedabad | 2000.00 |

| 2 | Khilan | 25 | Delhi | 1500.00 |

| 3 | kaushik | 23 | Kota | 2000.00 |

| 4 | Chaitali | 25 | Mumbai | 6500.00 |

| 5 | Hardik | 27 | Bhopal | 8500.00 |

| 6 | Komal | 22 | MP | 4500.00 |

| 7 | Muffy | 24 | Indore | 10000.00 |

+----+----------+-----+-----------+----------+

Following is an example, which would display all the records from the CUSTOMERS table, where the SALARY starts with 200.

SQL> SELECT \* FROM CUSTOMERS

WHERE SALARY LIKE '200%';

This would produce the following result −

+----+----------+-----+-----------+----------+

| ID | NAME | AGE | ADDRESS | SALARY |

+----+----------+-----+-----------+----------+

| 1 | Ramesh | 32 | Ahmedabad | 2000.00 |

| 3 | kaushik | 23 | Kota | 2000.00 |

+----+----------+-----+-----------+----------+

SQL - TOP, LIMIT or ROWNUM Clause