

SQL

<https://www.tutorialspoint.com/sql/index.htm>

SQL (Structured Query Language) is a programming language which is used to manage data stored in relational databases like MySQL, MS Access, SQL Server, Oracle, Sybase, Informix, Postgres etc.

SQL is a database computer language designed for the retrieval and management of data in a relational databases like MySQL, MS Access, SQL Server, Oracle, Sybase, Informix, Postgres etc. SQL stands for Structured Query Language. SQL was developed in the 1970s by IBM Computer Scientists.

SQL is not a database management system, but it is a query language which is used to store and retrieve the data from a database or in simple words SQL is a language that communicates with databases.

Data Definition Language (DDL)

A Data Definition Language (DDL) is a computer language which is used to create and modify the structure of database objects which include tables, views, schemas, and indexes etc.

Command	Description
CREATE	Creates a new table, a view of a table, or other object in the database.
ALTER	Modifies an existing database object, such as a table.
DROP	Deletes an entire table, a view of a table or other objects in the database.
TRUNCATE	Truncates the entire table in a go.

Data Manipulation Language (DML)

A Data Manipulation Language (DML) is a computer programming language which is used for adding, deleting, and modifying data in a database.

Command	Description
SELECT	Retrieves certain records from one or more tables.
INSERT	Creates a record.
UPDATE	Modifies records.
DELETE	Deletes records.

Data Control Language (DCL)

Data Control Language (DCL) is a computer programming language which is used to control access to data stored in a database.

Command	Description
GRANT	Gives a privilege to user
REVOKE	Takes back privileges granted from user.

SQL stands for Structured Query Language which is a computer language for storing, manipulating and retrieving data stored in a relational database. SQL was developed in the 1970s by IBM Computer Scientists and became a standard of the American National Standards Institute (ANSI) in 1986, and the International Organization for Standardization (ISO) in 1987.

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.

What is a Table?

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

What is a Field?

Every table is broken up into smaller entities called fields. A field is a column in a table that is designed to maintain specific information about every record in the table.

For example, our CUSTOMERS table consists of different fields like ID, Name, Age, Salary, City and Country.

ID	Name	Age	Salary	City	Country
1	Ramesh	32	2000.00	Hyderabad	India
2	Mukesh	40	5000.00	New York	USA
3	Sumit	45	4500.00	Muscat	Oman
4	Kaushik	25	2500.00	Kolkata	India
5	Hardik	29	3500.00	Bhopal	India
6	Komal	38	3500.00	Saharanpur	India
7	Ayush	25	3500.00	Delhi	India
8	Javed	29	3700.00	Delhi	India

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

ID	Name	Age	Salary	City	Country
-----------	-------------	------------	---------------	-------------	----------------

1	Ramesh	32	2000.00	Hyderabad	India
---	--------	----	---------	-----------	-------

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, our CUSTOMERS table have different columns to represent ID, Name, Age, Salary, City and Country.

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 from 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. Following table has three records where the first record has a NULL value for the salary and second record has a zero value for the salary.

ID	Name	Age	Salary	City	Country
1	Ramesh	32		Hyderabad	India
2	Mukesh	40	00.00	New York	USA
3	Sumit	45	4500.00	Muscat	Oman

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.

S.No.	Constraints
1	NOT NULL Constraint Ensures that a column cannot have a NULL value.
2	DEFAULT Constraint Provides a default value for a column when none is specified.
3	UNIQUE Key Ensures that all the values in a column are different.
4	PRIMARY Key Uniquely identifies each row/record in a database table.
5	FOREIGN Key Uniquely identifies a row/record in any another database table.
6	CHECK Constraint Ensures that all values in a column satisfy certain conditions.
7	INDEX Constraint Used to create and retrieve data from the database very quickly.

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.

SQL Databases

SQL or Relational databases are used to store and manage the data objects that are related to one another, i.e. the process of handling data in a relational database is done based on a relational model.

This relational model is an approach to manage data in a structured way (using tables). A system used to manage these **relational databases** is known as Relational Database Management System (RDBMS).

SQL Database Table Structure

SQL database server stores data in table form. **Tables** are database objects used to collect data in Row and Column format. Rows represent the entities whereas columns define the attributes of each entity in a table.

Columns: Columns are vertical elements in a table. Each column in a table holds specific attribute information, and column properties such as column name and data types (Integer, Char, String, etc).

Rows: Rows are horizontal elements in a table and users can add data or retrieve by executing SQL queries.

Case Sensitivity

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.

All the SQL statements require a **semicolon (;)** at the end of each statement. Semicolon is the standard way to separate different SQL statements which allows to include multiple SQL statements in a single line.

SQL CREATE DATABASE Statement

To store data within a database, you first need to create it. This is necessary to individualize the data belonging to an organization.

```
CREATE DATABASE database_name;
```

SQL USE Statement

Once the database is created, it needs to be used in order to start storing the data accordingly.

```
USE database_name;
```

SQL DROP DATABASE Statement

If a database is no longer necessary, you can also delete it. To delete/drop a database

```
DROP DATABASE database_name;
```

SQL CREATE TABLE Statement

In an SQL driven database, the data is stored in a structured manner, i.e. in the form of tables.

```
CREATE TABLE table_name(  
  
    column1 datatype,  
  
    column2 datatype,  
  
    column3 datatype,  
  
    .....  
  
    columnN datatype,  
  
    PRIMARY KEY( one or more columns )  
  
);
```

```
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)  
  
);
```

SQL DESC Statement

Every table in a database has a structure of its own. To display the structure of database tables, we use the DESC statements.

```
DESC table_name;
```

SQL INSERT INTO Statement

The SQL INSERT INTO Statement is used to insert data into database tables.

```
INSERT INTO table_name( column1, column2....columnN)  
VALUES ( value1, value2....valueN);
```

```
INSERT INTO CUSTOMERS VALUES  
(1, 'Ramesh', 32, 'Ahmedabad', 2000.00 ),  
(2, 'Khilan', 25, 'Delhi', 1500),  
(3, 'kaushik', 23, 'Kota', 2000),  
(4, 'Chaitali', 25, 'Mumbai', 6500),  
(5, 'Hardik', 27, 'Bhopal', 8500),  
(6, 'Komal', 22, 'Hyderabad', 4500),  
(7, 'Muffy', 24, 'Indore', 10000);
```

What are SQL Data types?

An SQL data type refers to the type of data which can be stored in a column of a database table. In a column, the user can store numeric, string, binary, etc by defining data types. For example integer data, character data, monetary data, date and time data, binary strings, and so on.

Defining a Data Type

SQL Data types are defined during the creation of a table in a database. While creating a table, it is required to specify its respective data type and size along with the name of the column.

```
CREATE TABLE table_name(column1 datatype, column2  
datatype....)
```

What is SQL Operator?

An SQL 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.

SQL Arithmetic Operators

SQL Arithmetic Operators are used to perform mathematical operations on the numerical values. SQL provides following operators to perform mathematical operations.

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

Operator	Description	Example
+	Addition	10 + 20 = 30
-	Subtraction	20 - 30 = -10
*	Multiplication	10 * 20 = 200
/	Division	20 / 10 = 2
%	Modulus	5 % 2 = 1

SQL Comparison Operators

SQL Comparison Operators test whether two given expressions are the same or not. These operators are used in SQL conditional statements

while comparing one expression with another and they return a **Boolean** value which can be either TRUE or FALSE. The result of an SQL comparison operation can be UNKNOWN when one or another operand has it's value as NULL.

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

Operator	Description	Example
=	Equal to	5 = 5 returns TRUE
!=	Not equal	5 != 6 returns TRUE
<>	Not equal	5 <> 4 returns TRUE
>	Greater than	4 > 5 returns FALSE
<	Less than	4 < 5 returns TRUE
>=	Greater than or equal to	4 >= 5 returns FALSE
<=	Less than or equal to	4 <= 5 returns TRUE
!<	Not less than	4 !< 5 returns FALSE
!>	Not greater than	4 !> 5 returns TRUE

SQL Logical Operators

SQL Logical Operators are very similar to comparison operators and they test for the truth of some given condition. These operators return a **Boolean** value which can be either a TRUE or FALSE. The result of an SQL logical operation can be UNKNOWN when one or another operand has it's value as NULL.

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

Operator	Description
ALL	TRUE if all of a set of comparisons are TRUE.
AND	TRUE if all the conditions separated by AND are TRUE.

ANY	TRUE if any one of a set of comparisons are TRUE.
BETWEEN	TRUE if the operand lies within the range of comparisons.
EXISTS	TRUE if the subquery returns one or more records
IN	TRUE if the operand is equal to one of a list of expressions.
LIKE	TRUE if the operand matches a pattern specially with wildcard.
NOT	Reverses the value of any other Boolean operator.
OR	TRUE if any of the conditions separated by OR is TRUE
IS NULL	TRUE if the expression value is NULL.
SOME	TRUE if some of a set of comparisons are TRUE.
UNIQUE	The UNIQUE operator searches every row of a specified table for uniqueness (no duplicates).

What is SQL Expression?

An SQL expression is a combination of one or more values, operators and SQL functions that are all evaluated to a value. These SQL EXPRESSION(s) 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.

```
SELECT * FROM CUSTOMERS WHERE SALARY = 10000;
```

SQL - Rename Database

```
ALTER DATABASE OldDatabaseName MODIFY NAME = NewDatabaseName;
```

SQL - Show Databases

```
SHOW DATABASES;
```

```
SHOW DATABASES LIKE 'test%';
```

SQL - CREATE Table

In RDBMS, Database tables are used to store the data in the form of some structures (fields and records). Here, a **field** is a column defining the type of data to be stored in a table and **record** is a row containing actual data. In simple words, we can say a Table is a combination of rows and columns.

```
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)  
);
```

```
DESC CUSTOMERS;
```

Field	Type	Null	Key	Default	Extra
ID	int(11)	NO	PRI	NULL	
NAME	varchar(20)	NO		NULL	
AGE	int(11)	NO		NULL	
ADDRESS	char(25)	YES		NULL	
SALARY	decimal(18,2)	YES		NULL	

```
CREATE TABLE IF NOT EXISTS table_name(  
    column1 datatype,  
    column2 datatype,  
    column3 datatype,  
    .....  
    columnN datatype,  
    PRIMARY KEY( one or more columns )  
);
```

Creating a Table from an Existing Table

```
CREATE TABLE NEW_TABLE_NAME AS  
SELECT [column1, column2...columnN]  
FROM EXISTING_TABLE_NAME  
WHERE Condition;
```

SQL - Show Tables (Listing Tables)

```
SHOW TABLES;
```

SQL - Rename Table

```
RENAME TABLE table_name TO new_table_name;
```

```
ALTER TABLE table_name RENAME [TO|AS] new_table_name
```

The SQL TRUNCATE TABLE Statement

The SQL **TRUNCATE TABLE** command is used to **empty** a table.

```
TRUNCATE TABLE table_name;
```

TRUNCATE vs DROP

Unlike **TRUNCATE** that resets the table structure, **DROP command** completely **freed the table space** from the memory. They are both Data Definition Language (DDL) operations as they interact with the definitions of database objects; which allows the database to **automatically commit** once these commands are executed with **no chance to roll back**.

SQL - Clone Tables

SQL **Cloning Operation** allows to create the exact copy of an existing table along with its definition.

```
CREATE TABLE new_table SELECT * FROM original_table;
```

What are Temporary Tables?

Temporary tables are pretty much what their name describes: they are the tables which are created in a database to store temporary data. We can perform SQL operations similar to the operations on permanent tables like CREATE, UPDATE, DELETE, INSERT, JOIN, etc. But these tables will be automatically deleted once the current client session is terminated. In addition to that, they can also be explicitly deleted if the users decide to drop them manually.

```
CREATE TEMPORARY TABLE table_name (
```

```
    column1 datatype,
```

```
    column2 datatype,
```

```
    column3 datatype,
```

```
    ....
```

```
    columnN datatype,
```

```
    PRIMARY KEY ( one or more columns )
```

```
) ;
```

ALTER TABLE – ADD Column

```
ALTER TABLE table_name ADD column_name datatype;
```

ALTER TABLE – DROP COLUMN

```
ALTER TABLE table_name DROP COLUMN column_name;
```

ALTER TABLE – ADD PRIMARY KEY

```
ALTER TABLE table_name  
ADD CONSTRAINT constraint_name  
PRIMARY KEY (column1, column2...);
```

The SQL DROP Table Statement

The SQL **DROP TABLE** statement is a Data Definition Language (DDL) command that is used to remove a table's definition, and its data, indexes, triggers, constraints and permission specifications (if any).

```
DROP TABLE table_name;
```

```
DROP TABLE IF EXISTS CUSTOMERS;
```

SQL DELETE TABLE Statement

The **SQL DELETE TABLE** statement is used to delete the existing records from a table in a database. If you wish to delete only the specific number of rows from the table, you can use the WHERE clause with the DELETE statement. If you omit the WHERE clause, all rows in the table will be deleted. The SQL DELETE statement operates on a single table at a time.

```
DELETE FROM table_name;
```

```
DELETE FROM table_name WHERE condition;
```

SQL Constraints

SQL Constraints are the rules applied to a data columns or the complete table to limit the type of data that can go into a table. When you try to perform any INSERT, UPDATE, or DELETE operation on the table, RDBMS will check whether that data violates any existing constraints and if there is any violation between the defined constraint and the data action, it aborts the operation and returns an error.

- [NOT NULL](#) - Ensures that a column cannot have a NULL value
- [UNIQUE](#) - Ensures that all values in a column are different
- [PRIMARY KEY](#) - A combination of a [NOT NULL](#) and [UNIQUE](#). Uniquely identifies each row in a table
- [FOREIGN KEY](#) - Prevents actions that would destroy links between tables
- [CHECK](#) - Ensures that the values in a column satisfies a specific condition
- [DEFAULT](#) - Sets a default value for a column if no value is specified
- [CREATE INDEX](#) - Used to create and retrieve data from the database very quickly

The SQL INSERT INTO Statement

The SQL **INSERT INTO Statement** is used to add new rows of data into a table in the database. Almost all the RDBMS provide this SQL query to add the records in database tables.

Each value in the records we are inserting in a table using this statement should be of the same datatype as the respective column and satisfy the constraints of the column (if any). The values passed using an insert statement should match the number of columns in the table or, the number of columns mentioned in the current query. If any of these conditions are not satisfied, this statement generates an error.

For specific columns/fields

```
INSERT INTO TABLE_NAME (column1, column2...columnN)
VALUES (value1, value2...valueN);
```


For all the columns/fields in the same order of the table columns

```
INSERT INTO CUSTOMERS  
VALUES (7, 'Muffy', 24, 'Indore', 10000.00 );]
```

Inserting into the table multiple columns at a time.

```
INSERT INTO CUSTOMERS VALUES  
(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, 'Hyderabad', 4500.00 ),  
(7, 'Muffy', 24, 'Indore', 10000.00 );
```

The SQL SELECT Statement

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

```
SELECT column1, column2, columnN FROM table_name;
```

```
SELECT * FROM table_name;
```

```
SELECT 56*65;
```

The SQL Select Into Statement

The **SQL SELECT INTO** Statement creates a new table and inserts data from an existing table into the newly created table. The new table is automatically created based on the structure of the columns in the SELECT statement and can be created in the same database or in a different database

```
SELECT * INTO new_table_name FROM existing_table_name
```

```
SELECT * INTO CUSTOMER_BACKUP FROM CUSTOMERS;
```

The Insert Into... Select Statement

The SQL **INSERT INTO... SELECT** statement is used to add/insert one or more new rows from an existing table to another table. This statement is a combination of two different statements: INSERT INTO and SELECT.

```
INSERT INTO table_new
```

```
SELECT (column1, column2, ...columnN)
```

```
FROM table_old;
```

```
INSERT INTO BUYERS SELECT * FROM CUSTOMERS;
```

The SQL UPDATE Statement

The SQL **UPDATE** Statement is used to modify the existing records in a table. This statement is a part of Data Manipulation Language (DML), as it only modifies the data present in a table without affecting the table's structure.

Since it only interacts with the data of a table, the **SQL UPDATE** statement needs to be used **cautiously**. If the rows to be modified aren't selected properly, all the rows in the table will be **affected** and the correct table data is either **lost or needs to be reinserted**.

```
UPDATE table_name
```

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

```
WHERE [condition];
```

```
UPDATE CUSTOMERS SET ADDRESS = 'Pune' WHERE ID = 6
```

```
UPDATE CUSTOMERS SET AGE = AGE+5, SALARY = SALARY+3000;
```

The SQL DELETE Statement

The SQL **DELETE** Statement is used to delete the records from an existing table. In order to filter the records to be deleted (or, delete particular records), we need to use the **WHERE** clause along with the DELETE statement.

If you execute a **DELETE** statement **without** a **WHERE** clause, it will **delete all the records from the table**.

Using the DELETE statement, we can delete one or more rows of a single table and records across multiple tables.

```
DELETE FROM CUSTOMERS WHERE ID = 6;
```

```
DELETE FROM CUSTOMERS WHERE AGE > 25;
```

SQL - SORTING Results

The SQL **ORDER BY** clause is used to sort the data in ascending or descending order, based on one or more columns. By default, some databases sort the query results in an ascending order.

```
SELECT * FROM CUSTOMERS ORDER BY NAME;
```

```
SELECT * FROM CUSTOMERS ORDER BY NAME DESC;
```

What is SQL View

A view in SQL is a virtual table that is stored in the database with an associated name. It is actually a composition of a table in the form of a predefined SQL query. A view can contain rows from an existing table (all or selected). A view can be created from one or many tables. Unless indexed, a view does not exist in a database.

```
CREATE VIEW view_name AS
```

```
SELECT column1, column2....
```

```
FROM table_name
```

```
WHERE [condition];
```

```
CREATE VIEW CUSTOMERS_VIEW AS SELECT * FROM CUSTOMERS;
```

SQL UPDATE View Statement

A view is a database object that can contain rows (all or selected) from an existing table. It can be created from one or many tables which depends on the provided SQL query to create a view.

```
UPDATE CUSTOMERS_VIEW
```

```
SET AGE = 35 WHERE name = 'Ramesh';
```

SQL - DROP or DELETE View

```
DROP VIEW view_name;
```

```
DROP VIEW IF EXISTS DEMO_VIEW;
```

```
DELETE FROM CUSTOMERS_VIEW3 WHERE AGE = 22;
```

SQL - Rename View

The **RENAME TABLE** statement in MySQL database is used to rename views. You just have to make sure that the new name of the view does not overlap with the name of any existing views.

```
RENAME TABLE old_view_name To new_view_name;
```

```
RENAME TABLE CUSTOMERS_VIEW TO VIEW_CUSTOMERS;
```

The SQL Where Clause

The SQL **WHERE** clause is used to filter the results obtained by the DML statements such as SELECT, UPDATE and DELETE etc. We can retrieve the data from a single table or multiple tables(after join operation) using the WHERE clause.

```
SELECT column1, column2, ...
```

```
FROM table_name
```

```
WHERE condition;
```

```
SELECT ID, NAME, SALARY FROM CUSTOMERS WHERE SALARY > 2000;
```

The SQL TOP Clause

While we are retrieving data from an SQL table, the SQL **TOP clause** is used to restrict the number of rows returned by a SELECT query in SQL server. In addition, we can also use it with UPDATE and DELETE statements to limit (restrict) the resultant records.

```
SELECT TOP 4 * FROM CUSTOMERS ORDER BY SALARY DESC;
```

FETCH FIRST | NEXT n ROW[S] ONLY

SELECT employeeid, firstname, revenue

FROM sales.employee

ORDER BY revenue desc

OFFSET 2 ROWS

FETCH NEXT 4 ROWS ONLY;

The SQL DISTINCT Keyword

The SQL **DISTINCT** keyword is used in conjunction with the SELECT statement to fetch unique records from a table.

We use DISTINCT keyword with the SELECT statement when there is a need to avoid duplicate values present in any specific columns/tables. When we use DISTINCT keyword, SELECT statement returns only the unique records available in the table.

```
SELECT DISTINCT column1, column2, .....columnN
```

```
FROM table_name;
```

The SQL ORDER BY Clause

The SQL **ORDER BY** clause is used to sort the data in either ascending or descending order, based on one or more columns. This clause can sort data by a single column or by multiple columns. Sorting by multiple columns can be helpful when you need to sort data hierarchically, such as sorting by state, city, and then by the person's name.

ORDER BY is used with the SQL SELECT statement and is usually specified after the WHERE, HAVING, and GROUP BY clauses.

The SQL **ORDER BY** clause is used to sort the data in ascending or descending order, based on one or more columns. By default, some databases sort the query results in an ascending order.

```
SELECT * FROM CUSTOMERS ORDER BY NAME;
```

```
SELECT * FROM CUSTOMERS ORDER BY NAME DESC;
```

The SQL GROUP BY Clause

The SQL **GROUP BY** clause is used in conjunction with the SELECT statement to arrange identical data into groups. This clause follows the WHERE clause in a SELECT statement and precedes the ORDER BY and HAVING clauses (if they exist).

The main purpose of grouping the records of a table based on particular columns is to perform calculations on these groups. Therefore, The GROUP BY clause is typically used with **aggregate functions** such as SUM(), COUNT(), AVG(), MAX(), or MIN() etc.

The SQL HAVING Clause

The SQL **HAVING clause** is similar to the WHERE clause; both are used to filter rows in a table based on specified criteria. However, the HAVING clause is used to filter grouped rows instead of single rows. These rows are grouped together by the GROUP BY clause, so, the HAVING clause must always be followed by the GROUP BY clause.

Moreover, the HAVING clause can be used with aggregate functions such as COUNT(), SUM(), AVG(), etc., whereas the WHERE clause cannot be used with them.

```
SELECT column1, column2, aggregate_function(column)
FROM table_name
GROUP BY column1, column2
HAVING condition;
```

```
SELECT
FROM
WHERE
GROUP BY
HAVING
ORDER BY
```

SQL - AND and OR Conjunctive Operators

The SQL AND Operator

The SQL **AND** returns **true** or **1**, if both its operands evaluates to true. We can use it to combine two conditions in the WHERE clause of an SQL statement.

```
SELECT ID, NAME, SALARY FROM CUSTOMERS
WHERE SALARY > 2000 AND AGE < 25;
```

The SQL OR Operator

The **OR** operator returns true if at least one its operands evaluates to true, and false otherwise. We can combine two conditions in an SQL statement's WHERE clause using the OR operator.

```
SELECT ID, NAME, SALARY FROM CUSTOMERS  
WHERE SALARY > 2000 OR AGE < 25;
```

The SQL LIKE Operator

The **SQL LIKE** operator is used to retrieve the data in a column of a table, based on a specified pattern.

It is used along with the WHERE clause of the UPDATE, DELETE and SELECT statements, to filter the rows based on the given pattern. These patterns are specified using **Wildcards**.

```
SELECT * FROM CUSTOMERS WHERE SALARY LIKE '200%';
```

S.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.

The SQL IN Operator

The SQL **IN Operator** is used to specify multiple values or sub query in the **WHERE** clause. It returns all rows in which the specified column matches one of the values in the list. The list of values or sub query must be specified in the parenthesis e.g. IN (**select query**) or IN (**Value1, Value2, Value3, ...**).

```
SELECT * FROM CUSTOMERS
WHERE NAME IN ('Khilan', 'Hardik', 'Muffy');
```

The **SQL ANY and ALL** operators are used to perform a comparison between a single value and a range of values returned by the subquery.

```
SELECT * FROM CUSTOMERS
WHERE SALARY > ANY (SELECT SALARY FROM CUSTOMERS WHERE AGE =
32);
```

```
SELECT * FROM CUSTOMERS
WHERE SALARY <>
ALL (SELECT SALARY FROM CUSTOMERS WHERE AGE = 25);
```

The SQL EXISTS Operator

The **SQL EXISTS** operator is used to verify whether a particular record exists in a MySQL table. While using this operator we need to specify the record (for which you have to check the existence) using a subquery.

The EXISTS operator is used in the **WHERE clause** of a **SELECT** statement to filter records based on the existence of related records in another table.

It can be used in SELECT, UPDATE, DELETE or INSERT statements.

```
SELECT * FROM CUSTOMERS WHERE  
EXISTS (  
    SELECT PRICE FROM CARS  
    WHERE CARS.ID = CUSTOMERS.ID AND PRICE > 2000000  
);
```

The SQL CASE Statement

The **SQL CASE** statement is a conditional statement that helps us to make decisions based on a set of conditions. It evaluates the set of conditions and returns the respective values when a condition is satisfied.

```
SELECT NAME, AGE,  
CASE  
    WHEN AGE > 30 THEN 'Gen X'  
    WHEN AGE > 25 THEN 'Gen Y'  
    WHEN AGE > 22 THEN 'Gen Z'  
    ELSE 'Gen Alpha'  
END AS Generation  
FROM CUSTOMERS;
```

SQL - NOT Operator

AND – Operator

OR – Operator

NOT – Operator

SQL NOT is a logical operator/connective used to negate a condition or Boolean expression in a WHERE clause. That is, TRUE becomes FALSE and vice versa.

```
SELECT * FROM CUSTOMERS WHERE NOT (SALARY > 2000.00);
```

The SQL NOT EQUAL Operator

The SQL **NOT EQUAL** operator is used to compare two values and return true if they are not equal. It is represented by "<>" and "!=". The difference between these two is that <> follows the ISO standard, but != doesn't. So, it is recommended to use the <> operator.

```
SELECT * FROM CUSTOMERS WHERE NAME <> 'Ramesh';
```

The SQL IS NULL Operator

The SQL **IS NULL** operator is used to check whether a value in a column is NULL. It returns true if the column value is NULL; otherwise false.

The NULL is a value that represents missing or unknown data, and the IS NULL operator allows us to filter for records that contain NULL values in a particular column.

```
SELECT * FROM CUSTOMERS WHERE ADDRESS IS NULL;
```

The SQL IS NOT NULL Operator

The SQL **IS NOT NULL** operator is used to filter data by verifying whether a particular column has a not-null values. This operator can be used with SQL statements such as SELECT, UPDATE, and DELETE.

```
SELECT * FROM CUSTOMERS WHERE ADDRESS IS NOT NULL;
```

The SQL NOT NULL Constraint

The **NOT NULL** constraint in SQL is used to ensure that a column in a table doesn't contain NULL (empty) values, and prevent any attempts to insert or update rows with NULL values.

```
CREATE TABLE CUSTOMERS (  
    ID INT NOT NULL,  
    NAME VARCHAR (20) NOT NULL,
```

```
AGE INT NOT NULL,  
ADDRESS CHAR (25) ,  
SALARY DECIMAL (20, 2) ,  
PRIMARY KEY (ID)  
);
```

The SQL BETWEEN Operator

The **BETWEEN** operator is a logical operator in SQL, that is used to retrieve the data within a specified range. The retrieved values can be integers, characters, or dates.

You can use the **BETWEEN** operator to replace a combination of "greater than equal AND less than equal" conditions.

```
SELECT * FROM CUSTOMERS WHERE AGE BETWEEN 20 AND 25;
```

The SQL UNION Operator

The SQL UNION operator is used to combine data from multiple tables by eliminating duplicate rows (if any).

To use the UNION operator on multiple tables, all these tables must be union compatible. And they are said to be union compatible if and only if they meet the following criteria –

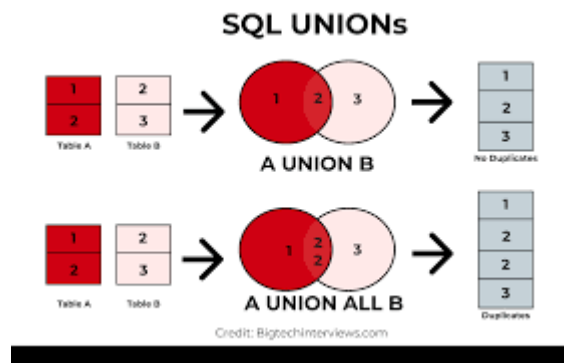
```
SELECT SALARY FROM CUSTOMERS UNION SELECT AMOUNT FROM ORDERS;
```

UNION and **UNION ALL** operators are just the SQL implementation of algebraic set operators. Both of them are used to retrieve the rows from multiple tables and return them as one single table. The difference between these two operators is that **UNION** only returns **distinct** rows while **UNION ALL** returns **all the rows** present in the tables.

What is UNION?

UNION is a type of operator/clause in SQL, that works similar to the union operator in relational algebra. It just combines the information from multiple tables that are union compatible.

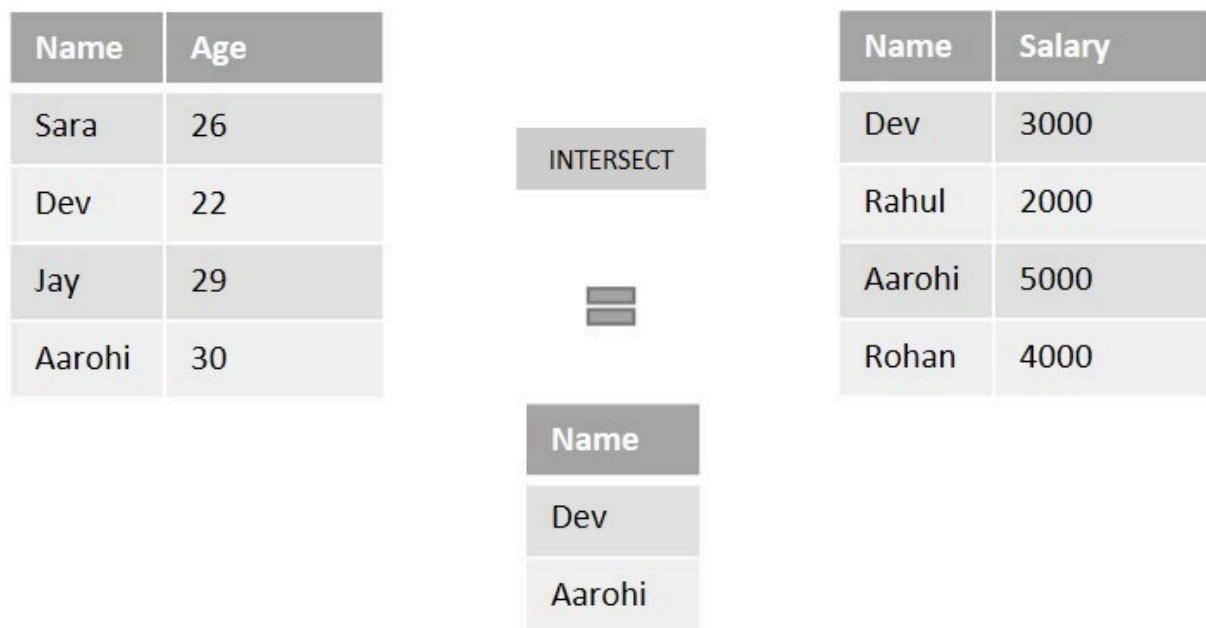
Only distinct rows from the tables are added to the resultant table, as UNION automatically eliminates all the duplicate records.



The SQL INTERSECT Operator

The **INTERSECT** operator in SQL is used to retrieve the records that are identical/common between the result sets of two or more tables.

Let us consider the below tables as an example to get a better understanding –



```
SELECT NAME, AGE, HOBBY FROM STUDENTS_HOBBY
```

```
INTERSECT
```

```
SELECT NAME, AGE, HOBBY FROM STUDENTS;
```

The SQL EXCEPT Operator

The **EXCEPT** operator in SQL is used to retrieve all the unique records from the left operand (query), except the records that are present in the result set of the right operand (query).

NAME	AGE	EXCEPT	NAME	SALARY
Sara	26		DEV	3000
Dev	22	=	Kalyan	2000
Jay	29		Aarohi	5000
Sara	26	NAME		
		Sara		
		Jay		

```
SELECT NAME, HOBBY, AGE FROM STUDENTS
```

```
EXCEPT
```

```
SELECT NAME, HOBBY, AGE FROM STUDENTS_HOBBY;
```

The SQL Aliasing

Aliases are used to address database tables with a shorter or more meaningful name within an SQL query. The basic syntax of a table alias is as follows.

-- Alias with single quotes (Not recommended for most DBs)

```
SELECT name AS 'student_name' FROM students;
```

-- Alias with double quotes (Standard for case sensitivity)

```
SELECT name AS "student_name" FROM students;
```

-- Alias without quotes (Most common and portable)

```
SELECT name AS student_name FROM students;
```

The SQL Join Clause

The SQL **Join** clause is used to combine data from two or more tables in a database. When the related data is stored across multiple tables, joins help you to retrieve records combining the fields from these tables using their foreign keys.

```
SELECT ID, NAME, AGE, AMOUNT
```

```
FROM CUSTOMERS
```

```
JOIN ORDERS
```

```
ON CUSTOMERS.ID = ORDERS.CUSTOMER_ID;
```

LEFT JOIN – returns **all** rows from the **left** table, even if there are no **matches** in the **right** table.

RIGHT JOIN – returns **all** rows from the **right** table, even if there are no **matches** in the **left** table.

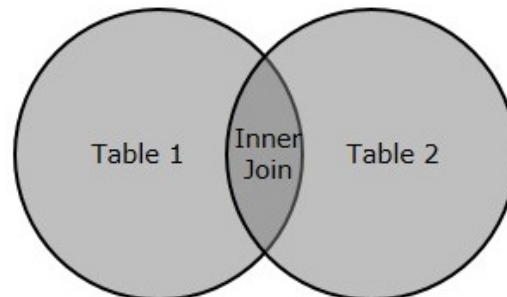
FULL JOIN – returns rows when there is a **match** in **one** of the tables.

SELF JOIN – is used to join a table to **itself** as if the table were two tables, temporarily renaming at least one table in the SQL statement.

CROSS Join – returns the **Cartesian product** of the sets of records from the two or more joined tables.

The SQL Inner Join

The **SQL Inner Join** is a type of join that combines multiple tables by retrieving records that have matching values in both tables (in the common column).



EmpDetails			MaritalStatus		
ID	Name	Salary	ID	Name	Status
1	John	40000	1	John	Married
2	Alex	25000	3	Simon	Married
3	Simon	43000	4	Stella	Unmarried

+
=

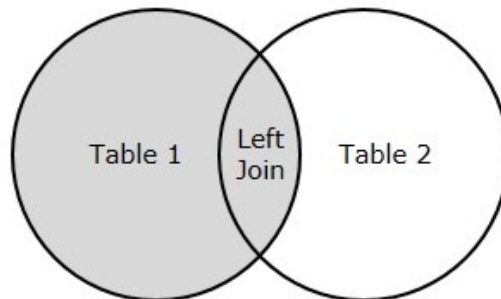
ID	Name	Salary	Status
1	John	40000	Married
3	Simon	43000	Married

```
SELECT ID, NAME, AMOUNT, DATE
FROM CUSTOMERS
INNER JOIN ORDERS
ON CUSTOMERS.ID = ORDERS.CUSTOMER_ID;
```

The SQL Left Join

Left Join or Left Outer Join in SQL combines two or more tables, where the first table is returned wholly; but, only the matching record(s) are retrieved from the consequent tables. If zero (0) records are matched in

the consequent tables, the join will still return a row in the result, but with NULL in each column from the right table.



```
SELECT ID, NAME, DATE, AMOUNT FROM CUSTOMERS
```

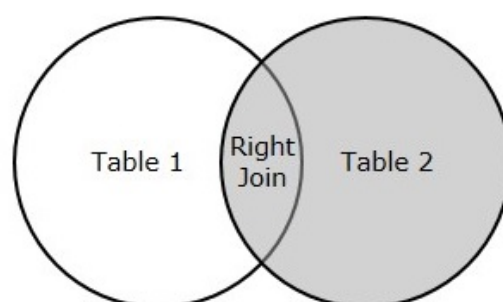
```
LEFT JOIN ORDERS
```

```
ON CUSTOMERS.ID = ORDERS.CUSTOMER_ID
```

```
WHERE ORDERS.AMOUNT > 2000.00;
```

The SQL Right Join

The Right Join or Right Outer Join query in SQL returns all rows from the right table, even if there are no matches in the left table. In short, a right join returns all the values from the right table, plus matched values from the left table or NULL in case of no matching join predicate.



```
SELECT ID, NAME, AMOUNT, DATE
```

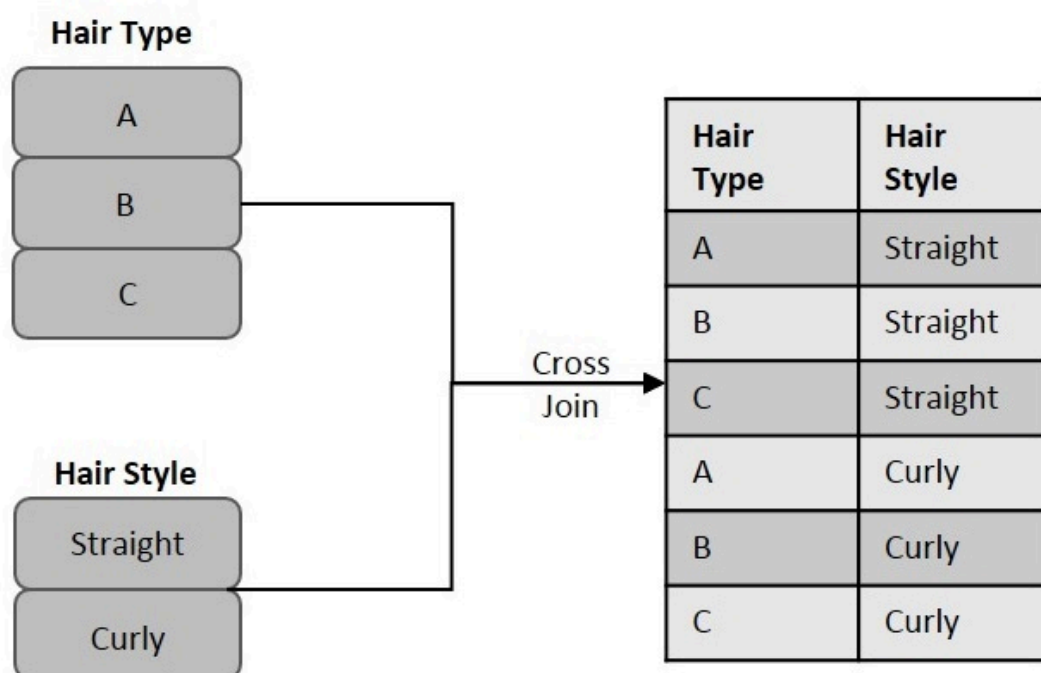
```
FROM CUSTOMERS
```

```
RIGHT JOIN ORDERS
```

```
ON CUSTOMERS.ID = ORDERS.CUSTOMER_ID;
```

The SQL Cross Join

An **SQL Cross Join** is a basic type of inner join that is used to retrieve the Cartesian product (or cross product) of two individual tables. That means, this join will combine each row of the first table with each row of second table (i.e. permutations).



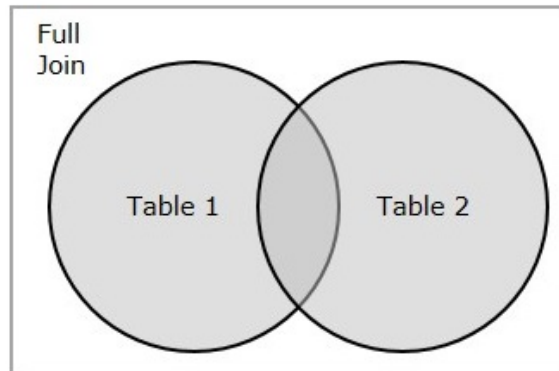
```
SELECT ID, NAME, AMOUNT, DATE
```

```
FROM CUSTOMERS
```

```
CROSS JOIN ORDERS;
```

The SQL Full Join

SQL Full Join creates a new table by joining two tables as a whole. The joined table contains all records from both the tables and fills NULL values for missing matches on either side. In short, full join is a type of outer join that combines the result-sets of both left and right joins.



```
SELECT ID, NAME, AMOUNT, DATE  
FROM CUSTOMERS  
FULL JOIN ORDERS  
ON CUSTOMERS.ID = ORDERS.CUSTOMER_ID;
```

The SQL Self Join

The **SQL Self Join** is used to join a table to itself as if the table were two tables. To carry this out, alias of the tables should be used at least once.

Self Join is a type of inner join, which is performed in cases where the comparison between two columns of a same table is required; probably to establish a relationship between them. In other words, a table is joined with itself when it contains both **Foreign Key** and **Primary Key** in it.

```
SELECT a.ID, b.NAME as EARNS_HIGHER, a.NAME  
as EARNS_LESS, a.SALARY as LOWER_SALARY  
FROM CUSTOMERS a, CUSTOMERS b  
WHERE a.SALARY < b.SALARY
```

```
ORDER BY a.SALARY;
```

The SQL DELETE... JOIN Clause

The purpose of Joins in SQL is to combine records of two or more tables based on common columns/fields. Once the tables are joined, performing the deletion operation on the obtained result-set will delete records from all the original tables at a time.

```
DELETE a  
FROM CUSTOMERS AS a INNER JOIN ORDERS AS b  
ON a.ID = b.CUSTOMER_ID  
WHERE a.SALARY < 2000.00;
```

The SQL UPDATE... JOIN Clause

The **UPDATE** statement only modifies the data in a single table and **JOINS** in SQL are used to fetch the combination of rows from multiple tables, with respect to a matching field.

```
UPDATE CUSTOMERS  
JOIN ORDERS  
ON CUSTOMERS.ID = ORDERS.CUSTOMER_ID  
SET CUSTOMERS.SALARY = CUSTOMERS.SALARY + 1000,  
ORDERS.AMOUNT = ORDERS.AMOUNT + 500;
```

The SQL Unique Key

The **SQL Unique Key** (or, Unique constraint) does not allow duplicate values in a column of a table. It prevents two records from having same values in a column.

The SQL Primary Key

The SQL **Primary Key** is a column (or combination of columns) that uniquely identifies each record in a database table. The Primary Key also speeds up data access and is used to establish a relationship between tables.

The SQL Foreign Key

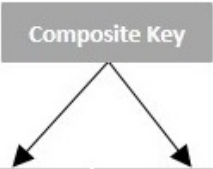
In SQL, a **Foreign Key** is a column in one table that matches a Primary Key in another table, allowing the two tables to be connected together.

A foreign key also maintains referential integrity between two tables, making it impossible to drop the table containing the primary key (preserving the connection between the tables).

The SQL Composite Key

An SQL **Composite Key** is a key that can be defined on two or more columns in a table to uniquely identify any record. It can also be described as a Primary Key created on multiple columns.

Composite Keys are necessary in scenarios where a database table does not have a single column that can uniquely identify each row from the table. In such cases, we might need to use the combination of columns to ensure that each record in the table is distinct and identifiable.



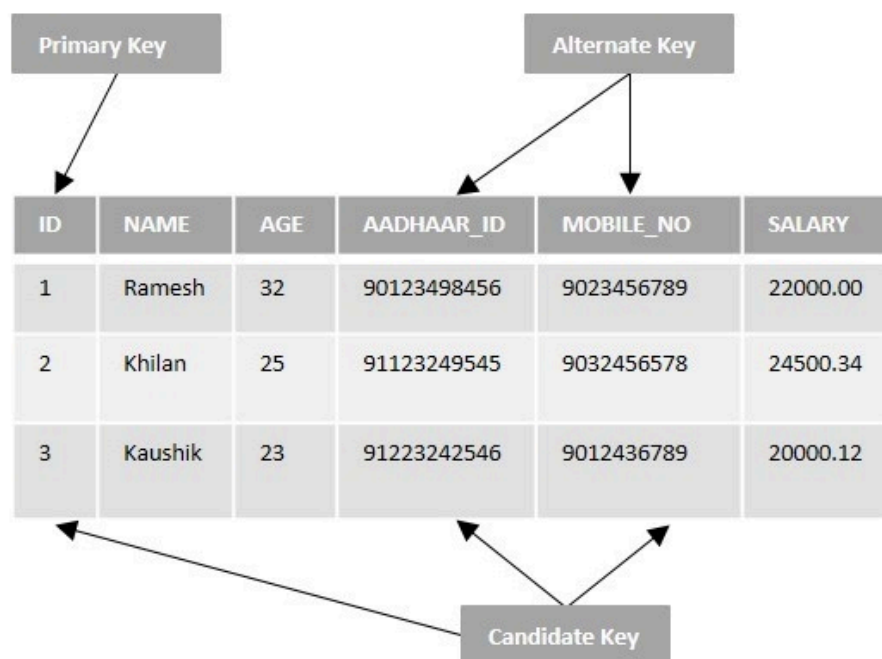
The diagram shows a box labeled "Composite Key" with two arrows pointing down to the "AADHAAR_ID" and "MOBILE_NO" columns of the table below.

ID	NAME	AGE	AADHAAR_ID	MOBILE_NO	SALARY
1	Ramesh	32	90123498456	9023456789	22000.00
2	Khilan	25	91123249545	9032456578	24500.34

The SQL Alternate Key

SQL **Alternate Keys** in a database table are candidate keys that are not currently selected as a primary key. They can be used to uniquely identify a tuple(or a record) in a table.

There is no specific query or syntax to set the alternate key in a table. It is just a column that is a close second candidate which could be selected as a primary key. Hence, they are also called secondary candidate keys.



SQL Injection

SQL Injection is a type of security attack that exploits a vulnerability in a database by executing malicious queries. This will allow attackers to access sensitive data, tamper it and also delete it permanently.

Injection usually occurs when you ask a user for input, like their name and instead of a name they give you a SQL statement that you will unknowingly run on your database. Never trust user provided data, process this data only after validation; as a rule, this is done by **Pattern Matching**.

SQL Stored Procedures

An SQL **stored procedure** is a group of pre-compiled SQL statements (prepared SQL code) that can be reused by simply calling it whenever needed.

It can be used to perform a wide range of database operations such as inserting, updating, or deleting data, generating reports, and performing complex calculations. Stored procedures are very useful because they allow you to encapsulate (bundle) a set of SQL statements as a single unit and execute them repeatedly with different parameters, making it easy to manage and reuse the code.

```
DELIMITER // CREATE PROCEDURE GetEmployeeInfo(IN emp_id INT)
BEGIN SELECT name, department FROM employees WHERE id = emp_id;
END // DELIMITER ;
```

SQL Transactions

A transaction is a unit or sequence of work that is performed on a database. Transactions are accomplished in a logical order, whether in a manual fashion by a user or automatically by some sort of a database program.

A transaction is the propagation of one or more changes to the database. For example, if you are creating, updating or deleting a record from the table, then you are performing a transaction on that table. It is important to control these transactions to ensure the data integrity and to handle database errors.

Sequences in SQL are database objects that generate a sequence of unique integer values. They are frequently used in databases because many applications require that each row in a table must contain unique values and sequences provide an easy way to generate them.

Sequences are a feature of many SQL database management systems, such as Oracle, PostgreSQL, SQL server, and IBM DB2.

```
CREATE TABLE BUYERS (
  ID INT AUTO_INCREMENT,
  NAME VARCHAR(20) NOT NULL,
```

```

    AGE INT NOT NULL,

    ADDRESS CHAR (25),

    SALARY DECIMAL (18, 2),

    PRIMARY KEY (ID)

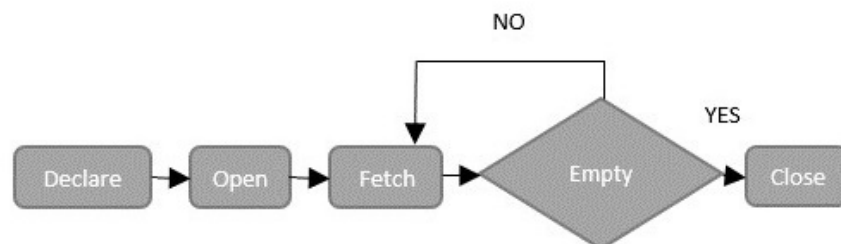
);

ALTER TABLE BUYERS AUTO_INCREMENT=100;

```

Cursors in SQL

A **Cursor** is a temporary memory that is allocated by the database server at the time of performing the **Data Manipulation Language** operations on a table, such as INSERT, UPDATE and DELETE etc. It is used to retrieve and manipulate data stored in the SQL table.



DECLARE

```
-- Declare cursor
```

```
CURSOR employee_cursor IS
```

```
    SELECT employee_id, employee_name
```

```
    FROM employees
```

```
    WHERE department_id = 10;
```



```
-- Declare variables to hold data fetched from the cursor

v_employee_id employees.employee_id%TYPE;

v_employee_name employees.employee_name%TYPE;


BEGIN

-- Open the cursor (implicitly done when the cursor is used)

OPEN employee_cursor;


-- Loop through the cursor rows

LOOP

    -- Fetch a row into the variables

    FETCH employee_cursor INTO v_employee_id, v_employee_name;


    -- Exit the loop if no more rows

    EXIT WHEN employee_cursor%NOTFOUND;


    -- Process the fetched data (e.g., print it)

    DBMS_OUTPUT.PUT_LINE('Employee ID: ' || v_employee_id || ',
Name: ' || v_employee_name);

END LOOP;


-- Close the cursor

CLOSE employee_cursor;


END;
```