CONTEXT:

SQL Tutorial

- SQL Tutorial
- SQL Syntax
- SQL Data Types
- SQL Operators

SQL Table

- What is Table
- SQL CREATE TABLE
- SQL DROP TABLE
- SQL DELETE TABLE
- SQL RENAME TABLE
- SQL TRUNCATE TABLE
- SQL COPY TABLE
- SQL TEMP TABLE
- SQL ALTER TABLE

SQL Select

- SELECT Statement
- SQL SELECT UNIQUE
- SQL SELECT DISTINCT
- SQL SELECT COUNT
- SQL SELECT TOP
- SQL SELECT FIRST
- SQL SELECT LAST
- SQL SELECT RANDOM
- SQL SELECT IN
- SQL SELECT Multiple
- SQL SELECT DATE
- SQL SELECT SUM
- SQL SELECT NULL

SQL Clause

- SQL WHERE
- SQL AND

- SQL OR
- SQL NOT
- SQL WITH
- SQL COUNT(),AVG(),SUM()
- SQL LIKE
- SQL AS
- SQL HAVING Clause
- SQL EXCEPT

SQL Order By

- ORDER BY Clause
- ORDER BY ASC
- ORDER BY DESC
- ORDER BY RANDOM
- ORDER BY LIMIT
- ORDER BY Multiple Cols
- ORDER BY DATE

SQL Insert

- INSERT Statement
- INSERT Multiple Rows

SQL Update

- UPDATE Statement
- SQL UPDATE JOIN
- SQL UPDATE DATE

SQL Delete

- DELETE Statement
- SQL DELETE TABLE
- SQL DELETE ROW
- SQL DELETE All Rows
- DELETE Duplicate Rows
- SQL DELETE DATABASE
- SQL DELETE VIEW
- SQL DELETE JOIN

SQL Join

- SQL JOIN
- SQL Outer Join
- SQL Left Join
- SQL Right Join
- SQL Full Join
- SQL Cross Join

SQL Keys

- Primary Key
- Foreign Key
- Composite Key
- Unique Key
- Alternate Key

SQL Injection

- SQL Injection

SQL String Functions

- SQL String Functions
- ASCII()
- BIT_LENGTH()
- CHAR_LENGTH()
- CHARACTER_LENGTH()
- CONCAT()
- CONCAT_WS()
- FIND_IN_SET()
- FORMAT()
- INSERT()
- INSTR()
- LCASE()
- LEFT()
- LENGTH()
- LOCATE()
- LOWER()
- LPAD()

- LTRIM()
- MID()
- NOW()
- POSITION()
- REPEAT()
- REPLACE()
- REVERSE()
- RIGHT()
- ROUND()
- RPAD()
- RTRIM()
- SPACE()
- STRCMP()
- SUBSTR()
- SUBSTRING()
- SUBSTRING_INDEX()
- UCASE()
- UPPER()

SQL MISCELLANEOUS

- CAST Function
- SQL Comments
- Joining three or more tables in SQL
- How to create functions in SQL?
- How to delete duplicate rows in SQL
- Nth highest age in table?
- 12 codd's rule
- SQL AUTO INCREMENT
- COMMIT AND ROLLBACK IN SQL
- SQL get month from date
- SAVEPOINT in SQL
- TIME datatype in SQL

- CRUD operations in SQL
- SET Operators in SQL
- SQL SUBQUERY
- SQL VIEW
- Constraints in SQL
- Pattern Matching in SQL
- SQL DATE Functions
- SQL CASE
- SQL CHECK
- SQL DEFAULT
- DELETION OF COLUMNS AND ROWS
- ADDITION OF COLUMN AND ROWS
- How to add Foreign Key in SQL
- How to add Primary Key in SQL
- SQL STORED PROCEDURE
- SQL ANY Keyword
- SQL ALL Keyword
- MAKE_SET()
- FIELD()
- CHAR()
- ELT()
- COUNT()
- SQL IS NULL

SQL Math functions

- POWER()
- ROUND()
- SUM()
- AVG()
- BIN()
- MAX()
- MIN()
- Mod()
- OCT()

- SIGN()
- SQRT()
- SQUARE()
- ABS()
- COS()
- COT()
- SIN()
- ACOS()
- ASIN()
- ATAN()
- TAN()
- CEIL()
- FLOOR()
- CEILING()
- DEGREES()
- EXP()
- RADIANS()
- RAND()
- ATN2()
- LOG()
- LOG2()
- LOG10()
- GREATEST()
- POW()
- DIV()
- LEAST()
- LN()
- UNICODE()
- REPLICATE()
- TRUNCATE()
- NCHAR()
- PATINDEX()
- PI()

SQL General functions

- NVL
- NVL2
- DECODE
- COALESCE
- NULLIF
- LNNVL
- NANVL

SQL TUTORIAL BASICS

SQL (Structured Query Language)

SQL (Structured Query Language) is used to perform operations on the records stored in the database, such as updating records, inserting records, deleting records, creating and modifying database tables, views, etc.

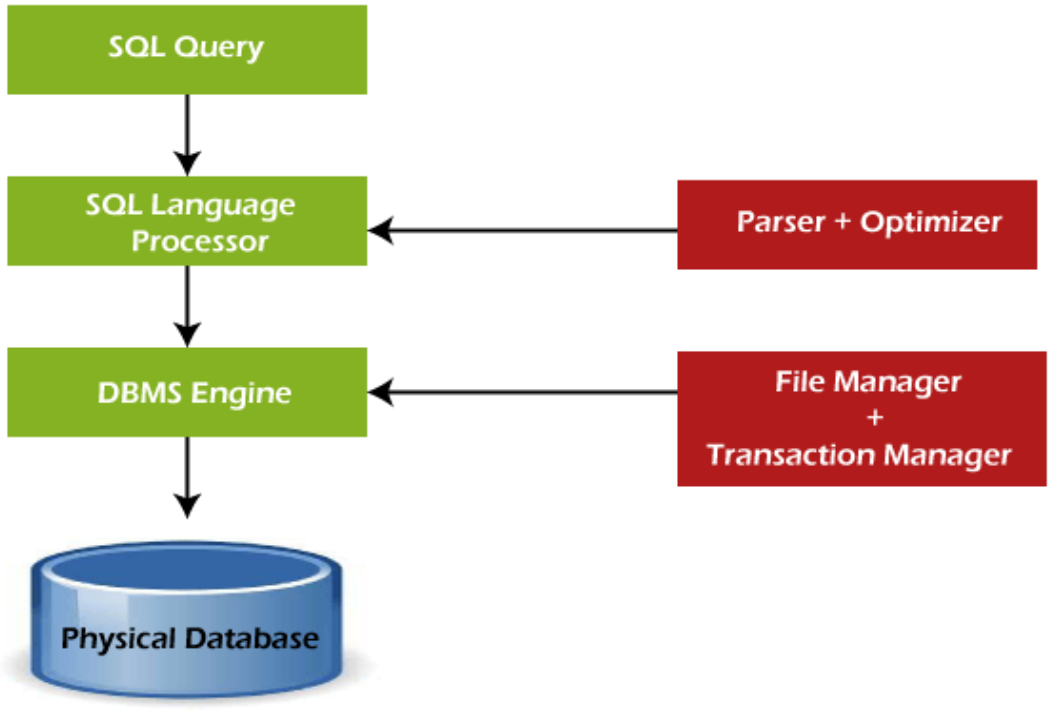
SQL is not a database system, but it is a query language.

Suppose you want to perform the queries of SQL language on the stored data in the database. You are required to install any database management system in your systems, for example, Oracle, MySQL, MongoDB, PostgreSQL, SQL Server, DB2, etc.

SQL is a short-form of the structured query language, and it is pronounced as S-Q-L or sometimes as See-Quell.

This database language is mainly designed for maintaining the data in relational database management systems. It is a special tool used by data professionals for handling structured data (data which is stored in the form of tables). It is also designed for stream processing in RDSMS.

You can easily create and manipulate the database, access and modify the table rows and columns, etc. This query language became the standard of ANSI in the year of 1986 and ISO in the year of 1987.



Some SQL Commands

The SQL commands help in creating and managing the database. The most common SQL commands which are highly used are mentioned below:

1. CREATE command
2. UPDATE command
3. DELETE command
4. SELECT command
5. DROP command
6. INSERT command

CREATE Command

This command helps in creating the new database, new table, table view, and other objects of the database.

UPDATE Command

This command helps in updating or changing the stored data in the database.

DELETE Command

This command helps in removing or erasing the saved records from the database tables. It erases single or multiple tuples from the tables of the database.

SELECT Command

This command helps in accessing the single or multiple rows from one or multiple tables of the database. We can also use this command with the WHERE clause.

DROP Command

This command helps in deleting the entire table, table view, and other objects from the database.

INSERT Command

This command helps in inserting the data or records into the database tables. We can easily insert the records in single as well as multiple rows of the table.

SQL	No-SQL
1. SQL is a relational database management system.	1. While No-SQL is a non-relational or distributed database management system.
2. The query language used in this database system is a structured query language.	2. The query language used in the No-SQL database systems is a non-declarative query language.
3. The schema of SQL databases is predefined, fixed, and static.	3. The schema of No-SQL databases is a dynamic schema for unstructured data.
4. These databases are vertically scalable.	4. These databases are horizontally scalable.
5. The database type of SQL is in the form of tables, i.e., in the form of rows and columns.	5. The database type of No-SQL is in the form of documents, key-value, and graphs.
6. It follows the ACID model.	6. It follows the BASE model.
7. Complex queries are easily managed in the SQL database.	7. NoSQL databases cannot handle complex queries.
8. This database is not the best choice for storing hierarchical data.	8. While No-SQL database is a perfect option for storing hierarchical data.
9. All SQL databases require object-relational mapping.	9. Many No-SQL databases do not require object-relational mapping.
10. Gauges, CircleCI, Hootsuite, etc., are the top enterprises that are using this query language.	10. Airbnb, Uber, and Kickstarter are the top enterprises that are using this query language.
11. SQLite, Ms-SQL, Oracle, PostgreSQL, and MySQL are examples of SQL database systems.	11. Redis, MongoDB, Hbase, BigTable, CouchDB, and Cassandra are examples of NoSQL database systems

The syntax of the structured query language is a unique set of rules and guidelines, which is not case-sensitive. Its Syntax is defined and maintained by the ISO and ANSI standards.

Following are some most important points about the SQL syntax which are to remember:

- You can write the keywords of SQL in both uppercase and lowercase, but writing the SQL keywords in uppercase improves the readability of the SQL query.
- SQL statements or syntax are dependent on text lines. We can place a single SQL statement on one or multiple text lines.
- You can perform most of the action in a database with SQL statements.
- SQL syntax depends on relational algebra and tuple relational calculus.

SQL Data Types

Data types are used to represent the nature of the data that can be stored in the database table. For example, in a particular column of a table, if we want to store a string type of data then we will have to declare a string data type of this column.

Data types mainly classified into three categories for every database.

- String Data types
- Numeric Data types
- Date and time Data type

MySQL Data Types

A list of data types used in MySQL database. This is based on MySQL 8.0.

MySQL String Data Types

CHAR(Size)	It is used to specify a fixed length string that can contain numbers, letters, and special characters. Its size can be 0 to 255 characters. Default is 1.
VARCHAR(Size)	It is used to specify a variable length string that can contain numbers, letters, and special characters. Its size can be from 0 to 65535 characters.
BINARY(Size)	It is equal to CHAR() but stores binary byte strings. Its size parameter specifies the column length in the bytes. Default is 1.

VARBINARY(Size)	It is equal to VARCHAR() but stores binary byte strings. Its size parameter specifies the maximum column length in bytes.
TEXT(Size)	It holds a string that can contain a maximum length of 255 characters.
TINYTEXT	It holds a string with a maximum length of 255 characters.
MEDIUMTEXT	It holds a string with a maximum length of 16,777,215.
LONGTEXT	It holds a string with a maximum length of 4,294,967,295 characters.
ENUM(val1, val2, val3,...)	It is used when a string object having only one value, chosen from a list of possible values. It contains 65535 values in an ENUM list. If you insert a value that is not in the list, a blank value will be inserted.
SET(val1,val2,val3,....)	It is used to specify a string that can have 0 or more values, chosen from a list of possible values. You can list up to 64 values at one time in a SET list.
BLOB(size)	It is used for BLOBs (Binary Large Objects). It can hold up to 65,535 bytes.

MySQL Numeric Data Types

BIT(Size)	It is used for a bit-value type. The number of bits per value is specified in size. Its size can be 1 to 64. The default value is 1.
INT(size)	It is used for the integer value. Its signed range varies from -2147483648 to 2147483647 and unsigned range varies from 0 to 4294967295. The size parameter specifies the max display width that is 255.
INTEGER(size)	It is equal to INT(size).
FLOAT(size, d)	It is used to specify a floating point number. Its size parameter specifies the total number of digits. The number of digits after the decimal point is specified by **d** parameter.
FLOAT(p)	It is used to specify a floating point number. MySQL used p parameter to determine whether to use FLOAT or DOUBLE. If p is between 0 to24, the data type becomes FLOAT (). If p is from 25 to 53, the data type becomes DOUBLE().

DOUBLE(size, d)	It is a normal size floating point number. Its size parameter specifies the total number of digits. The number of digits after the decimal is specified by d parameter.
DECIMAL(size, d)	It is used to specify a fixed point number. Its size parameter specifies the total number of digits. The number of digits after the decimal parameter is specified by **d** parameter. The maximum value for the size is 65, and the default value is 10. The maximum value for **d** is 30, and the default value is 0.
DEC(size, d)	It is equal to DECIMAL(size, d).
BOOL	It is used to specify Boolean values true and false. Zero is considered as false, and nonzero values are considered as true.

MySQL Date and Time Data Types

DATE	It is used to specify date format YYYY-MM-DD. Its supported range is from '1000-01-01' to '9999-12-31'.
DATETIME(fsp)	It is used to specify date and time combination. Its format is YYYY-MM-DD hh:mm:ss. Its supported range is from '1000-01-01 00:00:00' to 9999-12-31 23:59:59'.
TIMESTAMP(fsp)	It is used to specify the timestamp. Its value is stored as the number of seconds since the Unix epoch('1970-01-01 00:00:00' UTC). Its format is YYYY-MM-DD hh:mm:ss. Its supported range is from '1970-01-01 00:00:01' UTC to '2038-01-09 03:14:07' UTC.
TIME(fsp)	It is used to specify the time format. Its format is hh:mm:ss. Its supported range is from '-838:59:59' to '838:59:59'
YEAR	It is used to specify a year in four-digit format. Values allowed in four digit format from 1901 to 2155, and 0000.

Types of Operator

SQL operators are categorized in the following categories:

1. SQL Arithmetic Operators
2. SQL Comparison Operators
3. SQL Logical Operators
4. SQL Set Operators
5. SQL Bit-wise Operators
6. SQL Unary Operators

Let's discuss each operator with their types.

SQL Arithmetic Operators

The **Arithmetic Operators** perform the mathematical operation on the numerical data of the SQL tables. These operators perform addition, subtraction, multiplication, and division operations on the numerical operands.

Following are the various arithmetic operators performed on the SQL data:

1. SQL Addition Operator (+)
2. SQL Subtraction Operator (-)
3. SQL Multiplication Operator (+)
4. SQL Division Operator (-)
5. SQL Modulus Operator (+)

SQL Comparison Operators

The **Comparison Operators** in SQL compare two different data of SQL table and check whether they are the same, greater, and lesser. The SQL comparison operators are used with the WHERE clause in the SQL queries

Following are the various comparison operators which are performed on the data stored in the SQL database tables:

1. SQL Equal Operator (=)
2. SQL Not Equal Operator (!=)
3. SQL Greater Than Operator (>)
4. SQL Greater Than Equals to Operator (>=)
5. SQL Less Than Operator (<)\
6. SQL Less Than Equals to Operator (<=)

SQL Logical Operators

The **Logical Operators** in SQL perform the Boolean operations, which give two results **True and False.** These operators provide **True** value if both operands match the logical condition.

Following are the various logical operators which are performed on the data stored in the SQL database tables:

1. SQL ALL operator
2. SQL AND operator
3. SQL OR operator
4. SQL BETWEEN operator
5. SQL IN operator
6. SQL NOT operator
7. SQL ANY operator
8. SQL LIKE operator

SQL Unary Operators

The **Unary Operators** in SQL perform the unary operations on the single data of the SQL table, i.e., these operators operate only on one operand.

These types of operators can be easily operated on the numeric data value of the SQL table.

Following are the various unary operators which are performed on the numeric data stored in the SQL table:

1. SQL Unary Positive Operator
2. SQL Unary Negative Operator
3. SQL Unary Bitwise NOT Operator

SQL Set Operators

The **Set Operators** in SQL combine a similar type of data from two or more SQL database tables. It mixes the result, which is extracted from two or more SQL queries, into a single result.

Set operators combine more than one select statement in a single query and return a specific result set.

Following are the various set operators which are performed on the similar data stored in the two SQL database tables:

1. SQL Union Operator
2. SQL Union ALL Operator
3. SQL Intersect Operator
4. SQL Minus Operator

SQL TABLE

SQL Table

Table is a collection of data, organized in terms of rows and columns. In DBMS term, table is known as relation and row as tuple.

Note: A table has a specified number of columns, but can have any number of rows.

Table is the simple form of data storage. A table is also considered as a convenient representation of relations.

Let's see an example of an employee table:

SQL TABLE Variable

The **SQL Table variable** is used to create, modify, rename, copy and delete tables. Table variable was introduced by Microsoft.

It was introduced with SQL server 2000 to be an alternative of temporary tables.

Employee		
EMP_NAME	**ADDRESS**	**SALARY**
Ankit	Lucknow	15000
Raman	Allahabad	18000
Mike	New York	20000

It is a variable where we temporary store records and results. This is same like temp table but in the case of temp table we need to explicitly drop it.

Table variables are used to store a set of records. So declaration syntax generally looks like CREATE TABLE syntax.

```
create table "tablename"
```

```
("column1" "data type",
"column2" "data type",
"column3" "data type",
...
"columnN" "data type");
```

SQL CREATE TABLE

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

If you want to create a table, you should name the table and define its column and each column's data type.

Let's see the simple syntax to create the table.

```
CREATE TABLE Employee
(
EmployeeID int,
FirstName varchar(255),
LastName varchar(255),
Email varchar(255),
AddressLine varchar(255),
City varchar(255)
);
```

SQL DROP TABLE

A SQL DROP TABLE statement is used to delete a table definition and all data from a table.

This is very important to know that once a table is deleted all the information available in the table is lost forever, so we must be very careful when using this command.

Let's see the syntax to drop the table from the database.

First we verify STUDENTS table and then we would delete it from the database.

```
SQL> DESC STUDENTS;
```

This shows that STUDENTS table is available in the database, so we can drop it as follows:

```
SQL>DROP TABLE STUDENTS;
```

Now, use the following command to check whether table exists or not.

```
SQL> DESC STUDENTS;
Query OK, 0 rows affected (0.01 sec)
```

➔ **In SQL, "DESC" stands for "describe." It is a command used to retrieve metadata about a table, such as the names and data types of its columns. The syntax for using DESC in SQL varies slightly depending on the specific database management system (DBMS) you're using, but the general form is:**

SQL DELETE TABLE

The DELETE statement is used to delete rows from a table. If you want to remove a specific row from a table you should use WHERE condition.

1. **DELETE FROM table_name [WHERE condition];**

But if you do not specify the WHERE condition it will remove all the rows from the table.

1. **DELETE FROM table_name;**

Difference between DELETE and TRUNCATE statements

There is a slight difference b/w delete and truncate statement. The **DELETE statement** only deletes the rows from the table based on the condition defined by WHERE clause or delete all the rows from the table when condition is not specified.

But it does not free the space containing by the table.

The **TRUNCATE statement:** it is used to delete all the rows from the table **and free the containing space.**

Execute the following query to truncate the table:

1. **TRUNCATE TABLE** employee;

Difference b/w DROP and TRUNCATE statements

When you use the drop statement it deletes the table's row together with the table's definition so all the relationships of that table with other tables will no longer be valid.

When you drop a table:

- Table structure will be dropped
- Relationship will be dropped
- Integrity constraints will be dropped
- Access privileges will also be dropped

On the other hand, when we **TRUNCATE** a table, the table structure remains the same, so you will not face any of the above problems.

SQL RENAME TABLE

In some situations, database administrators and users want to change the name of the table in the SQL database because they want to give a more relevant name to the table.

Any database user can easily change the name by using the RENAME TABLE and ALTER TABLE statement in Structured Query Language.

The RENAME TABLE and ALTER TABLE syntax help in changing the name of the table.

Syntax of RENAME statement in SQL

1. RENAME old_table _name To new_table_name ;
   - Suppose, you want to change the above table name into "Car_2021_Details". For this, you have to type the following RENAME statement in SQL:

     RENAME Cars To Car_2021_Details ;
   - After this statement, the table "Cars" will be changed into table name "Car_2021_Details".

Syntax of ALTER TABLE statement in SQL

ALTER TABLE old_table_name RENAME TO new_table_name;

In the Syntax, we must specify the RENAME TO keyword after the old name of the table.

Suppose, you want to change the name of the above table into "Bikes_Details" using ALTER TABLE statement. For this, you have to type the following query in SQL:

ALTER TABLE Bikes RENAME TO Bikes_Details ;

After this statement, the table "Bikes" will be changed into the table name "Bikes_Details".

SQL TRUNCATE TABLE

A truncate SQL statement is used to remove all rows (complete data) from a table. It is similar to the DELETE statement with no WHERE clause.

TRUNCATE TABLE Vs DELETE TABLE

Truncate table is faster and uses lesser resources than DELETE TABLE command.

TRUNCATE TABLE Vs DROP TABLE

Drop table command can also be used to delete complete table but it deletes table structure too. TRUNCATE TABLE doesn't delete the structure of the table.

Let's see the syntax to truncate the table from the database.

TRUNCATE TABLE table_name;

SQL COPY TABLE

If you want to copy the data of one SQL table into another SQL table in the same SQL server, then it is possible by using the SELECT INTO statement in SQL.

The SELECT INTO statement in Structured Query Language copies the content from one existing table into the new table. SQL creates the new table by using the structure of the existing table.

Syntax of SELECT INTO statement in SQL

1. **SELECT * INTO New_table_name FROM old_table_name;**

SQL TEMP TABLE

The concept of temporary table is introduced by SQL server. It helps developers in many ways:

Temporary tables can be created at run-time and can do all kinds of operations that a normal table can do. These temporary tables are created inside tempdb database.

There are two types of temp tables based on the behavior and scope.

1. Local Temp Variable
2. Global Temp Variable

Local Temp Variable

Local temp tables are only available at current connection time. It is automatically deleted when user disconnects from instances. It is started with hash (#) sign.

```
CREATE TABLE #local temp table (
User id int,
Username varchar (50),
User address varchar (150)
)
```

Global Temp Variable

Global temp tables name starts with double hash (##). Once this table is created, it is like a permanent table. It is always ready for all users and not deleted until the total connection is withdrawn.

```
CREATE TABLE ##new global temp table (
User id int,
User name varchar (50),
User address varchar (150)
)
```

SQL ALTER TABLE

The ALTER TABLE statement in Structured Query Language allows you to add, modify, and delete columns of an existing table. This statement also allows database users to add and remove various SQL constraints on the existing tables.

Any user can also change the name of the table using this statement.

ALTER TABLE ADD Column statement in SQL

In many situations, you may require to add the columns in the existing table. Instead of creating a whole table or database again you can easily add single and multiple columns using the ADD keyword.

Syntax of ALTER TABLE ADD Column statement in SQL

1. **ALTER TABLE table_name ADD column_name column-definition;**

The above syntax only allows you to add a single column to the existing table. If you want to add more than one column to the table in a single SQL statement, then use the following syntax:

1. **ALTER TABLE table_name**

ADD (column_Name1 column-definition,

column_Name2 column-definition,

.....

column_NameN column-definition);

Suppose, you want to add the new column Car_Model in the above table. For this, you have to type the following query in the SQL:

ALTER TABLE Cars ADD Car_Model Varchar(20);

ALTER TABLE MODIFY Column statement in SQL

The MODIFY keyword is used for changing the column definition of the existing table.

Syntax of ALTER TABLE MODIFY Column statement in SQL

1. ALTER TABLE table_name MODIFY column_name column-definition;

This syntax only allows you to modify a single column of the existing table. If you want to modify more than one column of the table in a single SQL statement, then use the following syntax:

```
ALTER TABLE table_name
MODIFY (column_Name1 column-definition,
column_Name2 column-definition,
.....
column_NameN column-definition);
```

ALTER TABLE DROP Column statement in SQL

In many situations, you may require to delete the columns from the existing table. Instead of deleting the whole table or database you can use DROP keyword for deleting the columns.

Syntax of ALTER TABLE DROP Column statement in SQL

1. ALTER TABLE table_name DROP Column column_name ;

ALTER TABLE RENAME Column statement in SQL

The RENAME keyword is used for changing the name of columns or fields of the existing table.

Syntax of ALTER TABLE RENAME Column statement in SQL

1. ALTER TABLE table_name RENAME COLUMN old_name to new_name;

SQL SELECT STATEMENT

SQL SELECT Statement

The SELECT statement is the most used command in Structured Query Language. It is used to access the records from one or more database tables and views. It also retrieves the selected data that follow the conditions we want.

By using this command, we can also access the record from the column of the table. The table which stores the record returned by the SELECT statement is called a result-set table.

Syntax of SELECT Statement in SQL

1. **SELECT** Column_Name_1, Column_Name_2, ..., Column_Name_N **FROM** Table _Name;

In this SELECT syntax, **Column_Name_1, Column_Name_2, ..., Column_Name_N** are the name of those columns in the table whose data we want to read.

If you want to access all rows from all fields of the table, use the following SQL SELECT syntax with *

EXAMPLE:

```
CREATE TABLE Student_Records
(
Student_Id Int PRIMARY KEY,
First_Name VARCHAR (20),
Address VARCHAR (20),
Age Int NOT NULL,
Percentage Int NOT NULL,
Grade VARCHAR (10)
```

);

INSERT INTO Student_Records VALUES

(201, "Akash", "Delhi", 18, 89, "A2"),

(202, "Bhavesh", "Kanpur", 19, 93, "A1"),

(203, "Yash", "Delhi", 20, 89, "A2"),

(204, "Bhavana", "Delhi", 19, 78, "B1"),

(205, "Yatin", "Lucknow", 20, 75, "B1"),

(206, "Ishika", "Ghaziabad", 19, 51, "C1"),

(207, "Vivek", "Goa", 20, 62, "B2");

SELECT * FROM Student_Records;

OUTPUT:

```
201|Akash|Delhi|18|89|A2
202|Bhavesh|Kanpur|19|93|A1
203|Yash|Delhi|20|89|A2
204|Bhavana|Delhi|19|78|B1
205|Yatin|Lucknow|20|75|B1
206|Ishika|Ghaziabad|19|51|C1
207|Vivek|Goa|20|62|B2

[Execution complete with exit code 0]
```

SELECT First_Name,Age FROM Student_Records;

OUTPUT:

```
Akash|18
Bhavesh|19
Yash|20
Bhavana|19
Yatin|20
Ishika|19
Vivek|20

[Execution complete with exit code 0]
```

SELECT * FROM Student_Records WHERE Age = 20;

OUTPUT:

```
203|Yash|Delhi|20|89|A2
205|Yatin|Lucknow|20|75|B1
207|Vivek|Goa|20|62|B2

[Execution complete with exit code 0]
```

SELECT * FROM Student_Records Group by Age;

OUTPUT:

```
201|Akash|Delhi|18|89|A2
202|Bhavesh|Kanpur|19|93|A1
203|Yash|Delhi|20|89|A2

[Execution complete with exit code 0]
```

SELECT * FROM Student_Records Group by Grade having Percentage > 80;

OUTPUT:

```
202|Bhavesh|Kanpur|19|93|A1
201|Akash|Delhi|18|89|A2

[Execution complete with exit code 0]
```

SELECT * FROM Student_Records order by Percentage desc;

OUTPUT:

```
202|Bhavesh|Kanpur|19|93|A1
201|Akash|Delhi|18|89|A2
203|Yash|Delhi|20|89|A2
204|Bhavana|Delhi|19|78|B1
205|Yatin|Lucknow|20|75|B1
207|Vivek|Goa|20|62|B2
206|Ishika|Ghaziabad|19|51|C1

[Execution complete with exit code 0]
```

SQL SELECT UNIQUE

There is no difference between DISTINCT and UNIQUE.

SELECT UNIQUE is an old syntax which was used in oracle description but later ANSI standard defines DISTINCT as the official keyword.

After that oracle also added DISTINCT but did not withdraw the service of UNIQUE keyword for the sake of backward compatibility.

In simple words, we can say that SELECT UNIQUE statement is used to retrieve a unique or distinct element from the table.

Let's see the syntax of select unique statement.

```
SELECT UNIQUE column_name
FROM table_name;
```

SQL SELECT DISTINCT

The **SQL DISTINCT command** is used with SELECT key word to retrieve only distinct or unique data.

In a table, there may be a chance to exist a duplicate value and sometimes we want to retrieve only unique values. In such scenarios, SQL SELECT DISTINCT statement is used.

> ***Note: SQL SELECT UNIQUE and SQL SELECT DISTINCT statements are same.***

Let's see the syntax of select distinct statement.

```
SELECT DISTINCT column_name ,column_name
FROM  table_name;
```

How to use distinct in SQL?

SQL **DISTINCT** clause is used to remove the duplicates columns from the result set.

The distinct keyword is used with select keyword in conjunction. It is helpful when we avoid duplicate values present in the specific **columns/tables**. The **unique values** are fetched when we use the distinct keyword.

- SELECT DISTINCT returns only distinct (**different**) values.
- DISTINCT eliminates duplicate records from the table.

- DISTINCT can be used with aggregates: **COUNT, AVG, MAX**, etc.
- DISTINCT operates on a single column.
- Multiple columns are not supported for DISTINCT.

Syntax:

SELECT DISTINCT expressions
FROM tables
[**WHERE** conditions];

Parameters:

Expressions: The columns or calculations that we want to retrieve are called expression.

Tables: The tables that we want to retrieve the records. There is only one table in the FROM clause.

WHERE conditions: The conditions may meet for the records which are selected and it is optional.

Note:

- When one expression is provided in the **DISTINCT** clause then the query will return the unique values of the expressions.
- The query will retrieve the unique combinations for the listed expressions if more than one expression is provided in the **DISTINCT** clause here.
- In SQL, the **DISTINCT** clause cannot ignore the NULL values. So when we use the DISTINCT clause in the SQL statement, our result set will include NULL as a distinct value.

SELECT distinct Address FROM Student_Records ;

OUTPUT:

```
Delhi
Kanpur
Lucknow
Ghaziabad
Goa
```

```
[Execution complete with exit code 0]
```

SQL SELECT COUNT

The **SQL COUNT()** is a function that returns the number of records of the table in the output.

This function is used with the SQL SELECT statement.

Let's take a simple example: If you have a record of the voters in the selected area and want to count the number of voters, then it is very difficult to do it manually, but you can do it easily by using SQL SELECT COUNT query.

Syntax of Select Count Function in SQL

1. **SELECT COUNT(column_name) FROM table_name;**

In the syntax, we have to specify the column's name after the COUNT keyword and the name of the table on which the Count function is to be executed.

```
CREATE TABLE Student_Records
(
Student_Id Int PRIMARY KEY,
First_Name VARCHAR (20),
Address VARCHAR (20),
Age Int NOT NULL,
Percentage Int NOT NULL,
Grade VARCHAR (10)
);
```

```
INSERT INTO Student_Records VALUES
```

(201, "Akash", "Delhi", 18, 89, "A2"),

(202, "Bhavesh", "Kanpur", 19, 93, "A1"),

(203, "Yash", "Delhi", 20, 89, "A2"),

(204, "Bhavana", "Delhi", 19, 78, NULL),

(205, "Yatin", "Lucknow", 20, 75, "B1"),

(206, "Ishika", "Ghaziabad", 19, 51, "C1"),

(207, "Vivek", "Goa", 20, 62, NULL);

SELECT count(Grade) as total_students FROM Student_Records ;

OUTPUT:

```
5

[Execution complete with exit code 0]
```

Select Count(*) Function in SQL

The count(*) function in SQL shows all the Null and Non-Null records present in the table.

Syntax of Count (*) Function in SQL

1. **SELECT** COUNT(*) **FROM** table_name;

SELECT count(*) as total_students FROM Student_Records ;

OUTPUT:

```
7

[Execution complete with exit code 0]
```

SELECT count(First_Name) as total_students FROM Student_Records where Age != 20 ;

OUTPUT:

```
4

[Execution complete with exit code 0]
```

SELECT count(distinct Address) as total_students FROM Student_Records where Percentage > 80 ;

```
2

[Execution complete with exit code 0]
```

SELECT distinct Address as total_students FROM Student_Records where Percentage > 80 ;

```
Delhi
Kanpur

[Execution complete with exit code 0]
```

SQL SELECT TOP

The **SELECT TOP** statement in SQL shows the limited number of records or rows from the database table. The TOP clause in the statement specifies how many rows are returned.

It shows the top N number of rows from the tables in the output. This clause is used when there are thousands of records stored in the database tables.

Let's take a simple example: If a Student table has a large amount of data about students, the select TOP statement determines how much student data will be retrieved from the given table.

Syntax of TOP Clause in SQL

```
SELECT TOP number | percent column_Name1, column_Name2, ....., column_NameN  FROM table_name
```

Note: All the database systems do not support the TOP keyword for selecting the limited number of records. Oracle supports the ROWNUM keyword, and MySQL supports the LIMIT keyword.

Syntax of LIMIT Clause in MySQL

1. SELECT column_Name1,column_Name2, ..., column_NameN FROM table_name LIMIT value;

In the syntax, we have to specify the value after the LIMIT keyword. The value denotes the number of rows to be shown from the top in the output.

SELECT * from Student_Records limit 3

OUTPUT:

```
201|Akash|Delhi|18|89|A2
202|Bhavesh|Kanpur|19|93|A1
203|Yash|Delhi|20|89|A2

[Execution complete with exit code 0]
```

SQL SELECT FIRST

The SQL first() function is used to return the first value of the selected column.

Let's see the syntax of sql select first() function:

1. **SELECT FIRST**(column_name) **FROM** table_name;

Here a point is notable that first function is only supported by MS Access.

If you want to retrieve the first value of the "customer_name" column from the "customers" table, you need to write following query.

1. **SELECT FIRST**(customer_name) **AS** first_customer **FROM** customers;

> ***Note: The SELECT FIRST statement is only supported by MS Access. This statement doesn't work with other databases like Oracle, MySQL etc.***

SQL SELECT LAST

The **LAST()** function in Structured Query Language shows the last value from the specified column of the table.

Note: This SQL function is only supported in Microsoft Access database. Oracle supports ORDER BY and ROWNUM keywords, and MySQL supports the LIMIT keyword for selecting the last record.

Syntax of LAST() Function

1. **SELECT LAST** (Field_Name) **FROM** Table_Name ;

SELECT First_Name FROM Student_Records ORDER BY Student_Id DESC LIMIT 1;

OUTPUT:

```
Vivek

[Execution complete with exit code 0]
```

SQL SELECT RANDOM

The SQL SELECT RANDOM() function returns the random row. It can be used in online exam to display the random questions.

There are a lot of ways to select a random record or row from a database table. Each database server needs different SQL syntax.

If you want to select a random row with **MY SQL**:

```
SELECT column FROM table
ORDER BY RAND ( )
LIMIT 1
```

SQL SELECT IN

SQL IN is an operator used in a SQL query to help reduce the need to use multiple SQL "OR" conditions.

It is used in SELECT, INSERT, UPDATE or DELETE statement.

Advantage of SQL SELECT IN

It minimizes the use of SQL OR operator.

Let's see the syntax for SQL IN:

```
Expression IN (value 1, value 2 ... value n);
```

```
SELECT *
FROM Student_Records
WHERE Student_Id IN (201, 203, 024, 207);
```

```
201|Akash|Delhi|18|89|A2
203|Yash|Delhi|20|89|A2
207|Vivek|Goa|20|62|

[Execution complete with exit code 0]
```

SQL SELECT DATE

SQL SELECT DATE is used to retrieve a date from a database. If you want to find a particular date from a database, you can use this statement.

For example: let's see the query to get all the records after '2013-12-12'.

```
SELECT * FROM
table-name WHERE your date-column >= '2013-12-12'
```

Let's see the another query to get all the records after '2013-12-12' and before '2013-12-13' date.

```
SELECT* FROM
table-name where your date-column < '2013-12-13' and your date-column >= '2013-12-12'
```

If you want to compare the dates within the query, you should use BETWEEN operator to compare the dates.

```
SELECT * FROM
table_name WHERE yourdate BETWEEN '2012-12-12' and '2013-12-12
```

SQL SELECT SUM

It is also known as SQL SUM() function. It is used in a SQL query to return summed value of an expression.

Let's see the Syntax for the select sum function:

```
SELECT SUM (expression)
FROM tables
WHERE conditions;
```

expression may be numeric field or formula.

SQL SUM EXAMPLE with single field:

If you want to know how the combined total salary of all employee whose salary is above 20000 per month.

```
SELECT SUM (salary) AS "Total Salary"
FROM employees
WHERE salary > 20000;
```

In this example, you will find the expression as "Total Salary" when the result set is returned.

SQL SUM EXAMPLE with SQL DISTINCT:

You can also use SQL DISTINCT clause with SQL SUM function.

```
SELECT SUM (DISTINCT salary) AS "Total Salary"
FROM employees
WHERE salary > 20000;
```

SQL SUM EXAMPLE with SQL GROUP BY:

Sometimes there is a need to use the SQL GROUP BY statement with the SQL SUM function.

For example, we could also use the SQL SUM function to return the name of department and the total sales related to department.

```
SELECT department, SUM (sales) AS "Total Sales"
FROM order_details
GROUP BY department;
```

```
CREATE TABLE Student_Records
(
Student_Id Int PRIMARY KEY,
First_Name VARCHAR (20),
Address VARCHAR (20),
Age Int NOT NULL,
Percentage Int NOT NULL,
Grade VARCHAR (10)
);
```

```
INSERT INTO Student_Records VALUES
(201, "Akash", "Delhi", 18, 89, "A2"),
(202, "Bhavesh", "Kanpur", 19, 93, "A1"),
(203, "Yash", "Delhi", 20, 89, "A2"),
(204, "Bhavana", "Delhi", 19, 78, NULL),
(205, "Yatin", "Lucknow", 20, 75, "B1"),
(206, "Ishika", "Ghaziabad", 19, 51, "C1"),
(207, "Vivek", "Goa", 20, 62, NULL);
```

```
SELECT SUM (Percentage) AS "Total"
FROM Student_Records
WHERE Percentage > 80;
```

```
271

[Execution complete with exit code 0]
```

```
SELECT SUM (distinct Percentage) AS "Total"
FROM Student_Records
WHERE Percentage > 80;
```

```
182

[Execution complete with exit code 0]
```

```
SELECT SUM (Percentage),Address AS "Total"
FROM Student_Records
group by Address;
```

```
256|Delhi
51|Ghaziabad
62|Goa
93|Kanpur
75|Lucknow

[Execution complete with exit code 0]
```

SQL SELECT NULL

First of all we should know that what null value is? Null values are used to represent missing unknown data.

There can be two conditions:

1. Where SQL is NULL
2. Where SQL is NOT NULL

If in a table, a column is optional, it is very easy to insert data in column or update an existing record without adding a value in this column. This means that field has null value.

> ***Note: we should not compare null value with 0. They are not equivalent.***

SELECT Student_Id,First_Name,Address

FROM Student_Records

where Grade is NULL

```
204|Bhavana|Delhi
207|Vivek|Goa

[Execution complete with exit code 0]
```

SELECT Student_Id,First_Name,Address

FROM Student_Records

where Grade is NOT NULL

```
201|Akash|Delhi
202|Bhavesh|Kanpur
203|Yash|Delhi
205|Yatin|Lucknow
206|Ishika|Ghaziabad

[Execution complete with exit code 0]
```

SQL CLAUSE

SQL WHERE

A **WHERE clause** in SQL is a data manipulation language statement.

WHERE clauses are not mandatory clauses of SQL DML statements. But it can be used to limit the number of rows affected by a SQL DML statement or returned by a query.

Actually. it filters the records. It returns only those queries which fulfill the specific conditions.

WHERE clause is used in SELECT, UPDATE, DELETE statement etc.

Let's see the syntax for sql where:

```
SELECT column1, column 2, ... column n
FROM   table_name
WHERE [conditions]
```

WHERE clause uses some conditional selection

=	equal
>	greater than
<	less than
>=	greater than or equal
<=	less than or equal
< >	not equal to

SQL AND

- The SQL **AND** condition is used in SQL query to create two or more conditions to be met.
- It is used in SQL **SELECT, INSERT, UPDATE** and **DELETE**
- Let's see the syntax for SQL AND:

- SELECT columns FROM tables WHERE condition 1 AND condition 2;
- The SQL AND condition require that both conditions should be met.
- The SQL AND condition also can be used to join multiple tables in a SQL statement.
- To understand this concept practically, let us see some examples.

SELECT Student_Id,First_Name,Address

FROM Student_Records

where Grade is NOT NULL AND Student_Id in (201,204,205);

```
201|Akash|Delhi
205|Yatin|Lucknow

[Execution complete with exit code 0]
```

UPDATE Student_Records SET Address = "Delhi" WHERE Student_Id = 206 AND First_Name = "Ishika";

SELECT* from Student_Records;

```
201|Akash|Delhi|18|89|A2
202|Bhavesh|Kanpur|19|93|A1
203|Yash|Delhi|20|89|A2
204|Bhavana|Delhi|19|78|
205|Yatin|Lucknow|20|75|B1
206|Ishika|Delhi|19|51|C1
207|Vivek|Goa|20|62|

[Execution complete with exit code 0]
```

DELETE FROM Student_Records WHERE Student_Id = 206 AND First_Name = "Ishika";

SELECT* from Student_Records;

```
201|Akash|Delhi|18|89|A2
202|Bhavesh|Kanpur|19|93|A1
203|Yash|Delhi|20|89|A2
204|Bhavana|Delhi|19|78|
205|Yatin|Lucknow|20|75|B1
207|Vivek|Goa|20|62|

[Execution complete with exit code 0]
```

SQL OR

The SQL **OR** condition is used in SQL query to create a SQL statement where records are returned when any one condition met. It can be used in a **SELECT** statement, **INSERT** statement, **UPDATE** statement or **DELETE** statement.

Let's see the syntax for the OR condition:

1. **SELECT** columns **FROM** tables **WHERE** condition 1 OR condition 2;

SELECT * FROM Student_Records WHERE Student_Id = 206 OR First_Name = "Yash";

```
203|Yash|Delhi|20|89|A2
206|Ishika|Ghaziabad|19|51|C1

[Execution complete with exit code 0]
```

DELETE FROM Student_Records WHERE Student_Id = 206 OR First_Name = "Yash";

SELECT* from Student_Records;

```
201|Akash|Delhi|18|89|A2
202|Bhavesh|Kanpur|19|93|A1
204|Bhavana|Delhi|19|78|
205|Yatin|Lucknow|20|75|B1
207|Vivek|Goa|20|62|

[Execution complete with exit code 0]
```

UPDATE Student_Records SET Address = "Delhi" WHERE Student_Id = 206 OR First_Name = "Vivek";

SELECT* from Student_Records;

```
201|Akash|Delhi|18|89|A2
202|Bhavesh|Kanpur|19|93|A1
203|Yash|Delhi|20|89|A2
204|Bhavana|Delhi|19|78|
205|Yatin|Lucknow|20|75|B1
206|Ishika|Delhi|19|51|C1
207|Vivek|Delhi|20|62|

[Execution complete with exit code 0]
```

SELECT * FROM Student_Records WHERE Student_Id = 206 or First_Name = NULL;

```
206|Ishika|Ghaziabad|19|51|C1

[Execution complete with exit code 0]
```

SQL NOT

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

NOT Syntax

```
SELECT column1, column2, ...
FROM table_name
WHERE NOT condition;
```

CREATE TABLE Student_records

(

Student_Id Int PRIMARY KEY,

First_Name VARCHAR (20),

Address VARCHAR (20),

Age Int NOT NULL,

Percentage Int NOT NULL,

Grade VARCHAR (10)

);

INSERT INTO Student_records VALUES

(201, "Akash", "Delhi", 18, 89, "A2"),

(302, "Bhavesh", "Kanpur", 19, 93, "A1"),

(303, "Yash", "Delhi", 20, 89, "A2"),

(204, "Bhavana", "Delhi", 19, 78, NULL),

(205, "Yatin", "Lucknow", 20, 75, "B1"),

(206, "Ishika", "Ghaziabad", 19, 51, "C1"),

(307, "Vivek", "Goa", 20, 62, NULL);

SELECT * FROM Student_records

WHERE NOT Address = "Delhi";

```
302|Bhavesh|Kanpur|19|93|A1
205|Yatin|Lucknow|20|75|B1
206|Ishika|Ghaziabad|19|51|C1
307|Vivek|Goa|20|62|

[Execution complete with exit code 0]
```

SQL WITH CLAUSE

The SQL WITH clause is used to provide a sub-query block which can be referenced in several places within the main SQL query. It was introduced by oracle in oracle 9i release2 database.

There is an example of employee table:

Syntax for the SQL WITH clause -

This syntax is for SQL WITH clause using a single sub-query alias.

```
WITH <alias_name> AS (sql_sub-query_statement)
SELECT column_list FROM <alias_name> [table name]
[WHERE <join_condition>]
```

When you use multiple sub-query aliases, the syntax will be as follows.

```
WITH <alias_name_A>  AS (sql_sub-query_statement)
<alias_name_B> AS (sql_sub-query_statement_from_alias_name_A
Or sql_sub-query_statement)
SELECT <column_list>
FROM <alias_name_A >,< alias_name_B >, [tablenames]
[WHERE < join_condition>]
```

SQL COUNT(),AVG() and SUM()

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

COUNT() Syntax

SELECT COUNT(*column_name*)
FROM *table_name*
WHERE *condition*;

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

AVG() Syntax

SELECT AVG(*column_name*)
FROM *table_name*
WHERE *condition*;

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

SUM() Syntax

SELECT SUM(*column_name*)
FROM *table_name*

SELECT 'sum' AS Message;

SELECT SUM(Age) FROM Student_records;

SELECT 'count' AS Message;

SELECT COUNT(Age) FROM Student_records;

SELECT 'average' AS Message;

SELECT AVG(Age) FROM Student_records ;

```
sum
135
count
7
average
19.2857142857143

[Execution complete with exit code 0]
```

SQL LIKE

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

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

- The percent sign (%) represents zero, one, or multiple characters
- The underscore sign (_) represents one, single character

The percent sign and the underscore can also be used in combinations!

LIKE Syntax

SELECT *column1, column2, ...*
FROM *table_name*
WHERE *columnN* LIKE *pattern*;

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

Here are some examples showing different LIKE operators with '%' and '_' wildcards:

```
SELECT * FROM Student_records

WHERE Address LIKE  "%anpu%";
```

```
302|Bhavesh|Kanpur|19|93|A1

[Execution complete with exit code 0]
```

```
CREATE TABLE animals (

    id INT NOT NULL AUTO_INCREMENT,

    name CHAR(30) NOT NULL,

    PRIMARY KEY (id)

);

INSERT INTO animals (name) VALUES

   ('dog'),('cat'),('penguin'),
```

('lax'),('whale'),('ostrich');

UPDATE animals SET name = 'dolphin' WHERE name LIKE 'l%';

SELECT * FROM animals;

```
id      name
1       dog
2       cat
3       penguin
4       dolphin
5       whale
6       ostrich

[Execution complete with exit code 0]
```

DELETE FROM animals WHERE name LIKE 'l%';

SELECT * FROM animals;

```
id      name
1       dog
2       cat
3       penguin
5       whale
6       ostrich

[Execution complete with exit code 0]
```

LIKE Operator	**Description**
WHERE CustomerName LIKE 'a%'	Finds any values that start with "a"
WHERE CustomerName LIKE '%a'	Finds any values that end with "a"

WHERE CustomerName LIKE '%or%'	Finds any values that have "or" in any position
WHERE CustomerName LIKE '_r%'	Finds any values that have "r" in the second position
WHERE CustomerName LIKE 'a_%'	Finds any values that start with "a" and are at least 2 characters in length
WHERE CustomerName LIKE 'a__%'	Finds any values that start with "a" and are at least 3 characters in length
WHERE ContactName LIKE 'a%o'	Finds any values that start with "a" and ends with "o"

SQL BETWEEN

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

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

BETWEEN Syntax

```
SELECT column_name(s)
FROM table_name
WHERE column_name BETWEEN value1 AND value2;
```

SELECT * FROM Student_records

WHERE Age BETWEEN 15 AND 19;

```
201|Akash|Delhi|18|89|A2
302|Bhavesh|Kanpur|19|93|A1
204|Bhavana|Delhi|19|78|
206|Ishika|Ghaziabad|19|51|C1

[Execution complete with exit code 0]
```

SELECT * FROM Student_records

WHERE Age NOT BETWEEN 15 AND 19;

```
303|Yash|Delhi|20|89|A2
205|Yatin|Lucknow|20|75|B1
307|Vivek|Goa|20|62|

[Execution complete with exit code 0]
```

UPDATE Student_records SET Percentage = 98 WHERE Age BETWEEN 15 and 19;

Select * from Student_records;

```
Student_Id      First_Name      Address Age     Percentage      Grade
201     Akash   Delhi   18      98      A2
204     Bhavana Delhi   19      98      NULL
205     Yatin   Lucknow 20      75      B1
206     Ishika  Ghaziabad       19      98      C1
302     Bhavesh Kanpur  19      98      A1
303     Yash    Delhi   20      89      A2
307     Vivek   Goa     20      62      NULL

[Execution complete with exit code 0]
```

DELETE FROM Student_records WHERE Age BETWEEN 15 and 19;

Select * from Student_records;

```
Student_Id      First_Name      Address Age     Percentage      Grade
205     Yatin   Lucknow 20      75      B1
303     Yash    Delhi   20      89      A2
307     Vivek   Goa     20      62      NULL

[Execution complete with exit code 0]
```

SQL SELECT AS

- SQL '**AS**' is used to assign a new name temporarily to a table column or even a table.
- It makes an easy presentation of query results and allows the developer to label results more accurately without permanently renaming table columns or even the table itself.
- Let's see the syntax of select as:

1. **SELECT** Column_Name1 **AS** New_Column_Name, Column_Name2 **As** New_Column_Name **FROM** Table_Name;

Here, the Column_Name is the name of a column in the original table, and the New_Column_Name is the name assigned to a particular column only for that specific query. This means that New_Column_Name is a temporary name that will be assigned to a query.

```
CREATE TABLE Student_Records
(
Student_Id Int PRIMARY KEY,
First_Name VARCHAR (20),
Address VARCHAR (20),
Age Int NOT NULL,
Percentage Int NOT NULL,
Grade VARCHAR (10)
);
```

```
INSERT INTO Student_Records VALUES
(201, "Akash", "Delhi", 18, 89, "A2"),
(202, "Bhavesh", "Kanpur", 19, 93, "A1"),
(203, "Yash", "Delhi", 20, 89, "A2"),
(204, "Bhavana", "Delhi", 19, 78, NULL),
(205, "Yatin", "Lucknow", 20, 75, "B1"),
(206, "Ishika", "Ghaziabad", 19, 51, "C1"),
(207, "Vivek", "Goa", 20, 62, NULL);
```

```
SELECT Student_Id AS 'id', First_Name FROM Student_Records;
```

```
201|Akash
202|Bhavesh
```

```
203|Yash
204|Bhavana
205|Yatin
206|Ishika
207|Vivek

[Execution complete with exit code 0]
```

Note: SQL AS is the same as SQL ALIAS.

SELECT Student_Name AS Student, AVG (Student_Percentage) AS Average_Percentage FROM students;

```
201|Akash|19.2857142857143

[Execution complete with exit code 0]
```

SELECT Student_Id AS Stu, CONCAT(First_Name, ', ', Age) AS Student_Info FROM Student_Records;

```
// Error: near line 21: in prepare, no such function: CONCAT (1)

[Execution complete with exit code 1]
```

HAVING Clause in SQL

The HAVING clause places the condition in the groups defined by the GROUP BY clause in the SELECT statement.

This SQL clause is implemented after the 'GROUP BY' clause in the 'SELECT' statement.

This clause is used in SQL because we cannot use the WHERE clause with the SQL aggregate functions. Both WHERE and HAVING clauses are used for filtering the records in SQL queries.

Difference between HAVING and WHERE Clause

The difference between the WHERE and HAVING clauses in the database is the most important question asked during an IT interview.

The following table shows the comparisons between these two clauses, but the main difference is that the WHERE clause uses condition for filtering records before any groupings are made, while HAVING clause uses condition for filtering values from a group.

HAVING	WHERE
1. The HAVING clause is used in database systems to fetch the data/values from the groups according to the given condition.	1. The WHERE clause is used in database systems to fetch the data/values from the tables according to the given condition.
2. The HAVING clause is always executed with the GROUP BY clause.	2. The WHERE clause can be executed without the GROUP BY clause.
3. The HAVING clause can include SQL aggregate functions in a query or statement.	3. We cannot use the SQL aggregate function with WHERE clause in statements.
4. We can only use SELECT statement with HAVING clause for filtering the records.	4. Whereas, we can easily use WHERE clause with UPDATE, DELETE, and SELECT statements.
5. The HAVING clause is used in SQL queries after the GROUP BY clause.	5. The WHERE clause is always used before the GROUP BY clause in SQL queries.
6. We can implements this SQL clause in column operations.	6. We can implements this SQL clause in row operations.
7. It is a post-filter.	7. It is a pre-filter.
8. It is used to filter groups.	8. It is used to filter the single record of the table.

Syntax of HAVING clause in SQL

1. **SELECT** column_Name1, column_Name2, ..., column_NameN aggregate_function_name(column_Name)
   FROM table_name **GROUP BY** column_Name1 **HAVING** condition;

MIN Function with HAVING Clause:

If you want to show each department and the minimum salary in each department, you must write the following query:

1. **SELECT MIN**(Emp_Salary), Emp_Dept **FROM** Employee **GROUP BY** Emp_Dept;

```
CREATE TABLE Student_Records
(
Student_Id Int PRIMARY KEY,
First_Name VARCHAR (20),
Address VARCHAR (20),
Age Int NOT NULL,
Percentage Int NOT NULL,
Grade VARCHAR (10)
);
```

```
INSERT INTO Student_Records VALUES
(201, "Akash", "Delhi", 18, 89, "A2"),
```

(202, "Bhavesh", "Kanpur", 19, 93, "A1"),

(203, "Yash", "Delhi", 20, 89, "A2"),

(204, "Bhavana", "Delhi", 19, 78, NULL),

(205, "Yatin", "Lucknow", 20, 75, "B1"),

(206, "Ishika", "Ghaziabad", 19, 51, "C1"),

(207, "Vivek", "Goa", 20, 62, NULL);

SELECT MIN(Percentage), First_Name FROM Student_Records GROUP BY Address;

```
78|Bhavana
51|Ishika
62|Vivek
93|Bhavesh
75|Yatin

[Execution complete with exit code 0]
```

SELECT max(Percentage), First_Name FROM Student_Records GROUP BY Address;

```
89|Akash
51|Ishika
62|Vivek
93|Bhavesh
75|Yatin

[Execution complete with exit code 0]
```

SQL EXCEPT

Usually, we use a JOIN clause to get the combined result from more than one table. Sometimes, we need a result set that contains records from one table but not available in the other table. In that case, SQL provides an EXCEPT clause/operator.

The EXCEPT clause in SQL is widely used to filter records from more than one table. This statement first combines the two SELECT statements and returns records from the first SELECT query, which aren't present in the second SELECT query's result. In other words, it retrieves all rows from the first SELECT query while deleting redundant rows from the second.

This statement behaves the same as the minus operator does in mathematics. This article will illustrate how to use the SQL EXCEPT clause with the help of basic examples.

Rules for SQL EXCEPT

We should consider the following rules before using the EXCEPT statement in SQL:

- In all SELECT statements, the number of columns and orders in the tables must be the same.
- The corresponding column's data types should be either the same or compatible.
- The fields in the respective columns of two SELECT statements cannot be the same.

SQL EXCEPT Syntax

The following syntax illustrates the use of EXCEPT clause:

1. **SELECT** column_lists **from** table_name1
2. **EXCEPT**
3. **SELECT** column_lists **from** table_name2;

NOTE: It is to note that MySQL does not support EXCEPT clause. So here we are going to use the PostgreSQL database to explain SQL EXCEPT examples.

The below image explains the working of EXCEPT operation in the two tables T1 and T2:

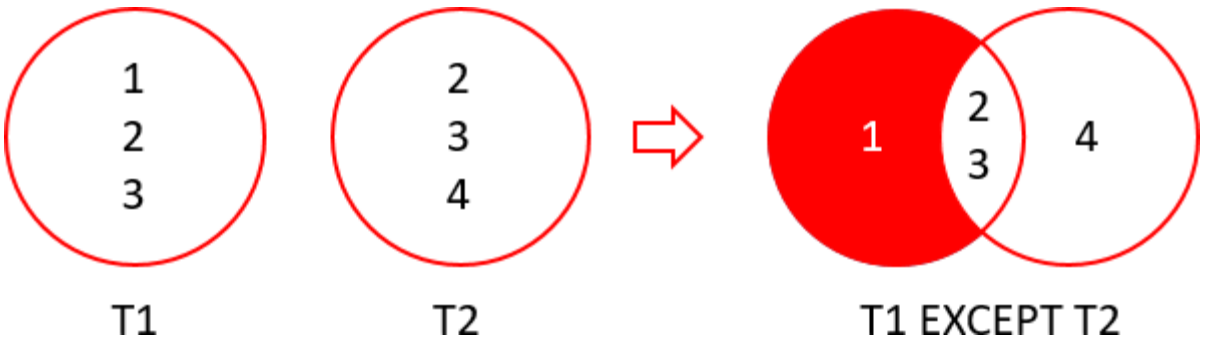


Illustration:

- Table T1 includes data 1, 2, and 3.
- Table T2 includes data 2, 3, and 4.

When we execute the EXCEPT query on these tables, we will get 1, which is unique data from the T1, and it will not found in the T2.

```
  CREATE TABLE customer
(
   id integer NOT NULL primary key,
   name text  NOT NULL,
   age integer NOT NULL,
   salary real NOT NULL
);

CREATE TABLE orders
(
   order_id integer NOT NULL,
   date NOT NULL,
   cust_id integer NOT NULL,
   amount real NOT NULL
);

INSERT INTO customer(id, name, age, salary)
   VALUES (101, 'John', 24, 20000),
   (102, 'Mike', 22, 25000),
   (103, 'Emily', 24, 22000),
```

```
    (104, 'James', 20, 30000),
    (105, 'Sophia', 21, 35000);

INSERT INTO orders( order_id, date, cust_id, amount)
    VALUES (1, '2009-10-08', 103, 1500),
    (2, '2009-11-06', 103, 1000),
    (3, '2009-12-05', 102, 2500),
    (4, '2009-09-08', 101, 1800);

SELECT  id, name, amount, date
   FROM customer
   LEFT JOIN orders
   ON customer.id = orders.cust_id
EXCEPT
   SELECT  id, name, amount, date
   FROM customer
   RIGHT JOIN orders
   ON customer.id = orders.cust_id;
```

```
id      name    amount  date
104     James   NULL    NULL
105     Sophia  NULL    NULL

[Execution complete with exit code 0]
```

EXCEPT statements in a single table

Generally, we use the EXCEPT statements in two tables, but we can also use them to filter records from a single table.

```
CREATE TABLE contacts (
id INT PRIMARY KEY AUTO_INCREMENT,
first_name VARCHAR(30) NOT NULL,
last_name VARCHAR(25) NOT NULL,
    email VARCHAR(210) NOT NULL,
    age VARCHAR(22) NOT NULL
);


INSERT INTO contacts (first_name,last_name,email,age)
VALUES ('Kavin','Peterson','kavin.peterson@verizon.net','21'),
      ('Nick','Jonas','nick.jonas@me.com','18'),
      ('Peter','Heaven','peter.heaven@google.com','23'),
      ('Michal','Jackson','michal.jackson@aol.com','22'),
      ('Sean','Bean','sean.bean@yahoo.com','23'),
      ('Tom ','Baker','tom.baker@aol.com','20'),
      ('Ben','Barnes','ben.barnes@comcast.net','17'),
      ('Mischa ','Barton','mischa.barton@att.net','18'),
      ('Sean','Bean','sean.bean@yahoo.com','16'),
      ('Eliza','Bennett','eliza.bennett@yahoo.com','25'),
      ('Michal','Krane','michal.Krane@me.com','25'),
      ('Peter','Heaven','peter.heaven@google.com','20'),
      ('Brian','Blessed','brian.blessed@yahoo.com','20'),
      ('Kavin','Peterson','kavin.peterson@verizon.net','30');
```

SELECT id, first_name,age FROM contacts

EXCEPT

SELECT id, first_name, age FROM contacts WHERE age > 21;

```
id      first_name      age
8       Mischa  18
9       Sean    16
7       Ben     17
2       Nick    18
13      Brian   20
1       Kavin   21
6       Tom     20
12      Peter   20

[Execution complete with exit code 0]
```

SQL ORDER BY CLAUSE

SQL ORDER BY Clause

- Whenever we want to sort the records based on the columns stored in the tables of the SQL database, then we consider using the ORDER BY clause in SQL.
- The ORDER BY clause in SQL will help us to sort the records based on the specific column of a table. This means that all the values stored in the column on which we are applying ORDER BY clause will be sorted, and the corresponding column values will be displayed in the sequence in which we have obtained the values in the earlier step.
- Using the ORDER BY clause, we can sort the records in ascending or descending order as per our requirement. The records will be sorted in ascending order whenever the **ASC keyword** is used with ORDER by clause. **DESC keyword** will sort the records in descending order.
- ***If no keyword is specified after the column based on which we have to sort the records, in that case, the sorting will be done by default in the ascending order.***

Before writing the queries for sorting the records, let us understand the syntax.

Syntax to sort the records in ascending order:

1. **SELECT** ColumnName1,...,ColumnNameN **FROM** TableName **ORDER BY** ColumnName **ASC**;

Syntax to sort the records in descending order:

1. **SELECT** ColumnName1,...,ColumnNameN **FROM** TableName **ORDER BY** ColumnNameDESC;

Syntax to sort the records in ascending order without using ASC keyword:

1. **SELECT** ColumnName1,...,ColumnNameN **FROM** TableName **ORDER BY** ColumnName;

```
CREATE TABLE Student_Records
(
```

```
Student_Id Int PRIMARY KEY,
First_Name VARCHAR (20),
Address VARCHAR (20),
Age Int NOT NULL,
Percentage Int NOT NULL,
Grade VARCHAR (10)
);
```

```
INSERT INTO Student_Records VALUES
(201, "Akash", "Delhi", 18, 89, "A2"),
(202, "Bhavesh", "Kanpur", 19, 93, "A1"),
(203, "Yash", "Delhi", 20, 89, "A2"),
(204, "Bhavana", "Delhi", 19, 78, NULL),
(205, "Yatin", "Lucknow", 20, 75, "B1"),
(206, "Ishika", "Ghaziabad", 19, 51, "C1"),
(207, "Vivek", "Goa", 20, 62, NULL);
```

```
SELECT * FROM Student_Records  ORDER BY Age;
```

```
201|Akash|Delhi|18|89|A2
202|Bhavesh|Kanpur|19|93|A1
204|Bhavana|Delhi|19|78|
206|Ishika|Ghaziabad|19|51|C1
203|Yash|Delhi|20|89|A2
205|Yatin|Lucknow|20|75|B1
207|Vivek|Goa|20|62|

[Execution complete with exit code 0]
```

```
SELECT * FROM Student_Records  ORDER BY First_Name;
```

```
201|Akash|Delhi|18|89|A2
204|Bhavana|Delhi|19|78|
202|Bhavesh|Kanpur|19|93|A1
206|Ishika|Ghaziabad|19|51|C1
207|Vivek|Goa|20|62|
203|Yash|Delhi|20|89|A2
205|Yatin|Lucknow|20|75|B1

[Execution complete with exit code 0]
```

SELECT * FROM Student_Records ORDER BY First_Name desc;

```
205|Yatin|Lucknow|20|75|B1
203|Yash|Delhi|20|89|A2
207|Vivek|Goa|20|62|
206|Ishika|Ghaziabad|19|51|C1
202|Bhavesh|Kanpur|19|93|A1
204|Bhavana|Delhi|19|78|
201|Akash|Delhi|18|89|A2

[Execution complete with exit code 0]
```

SQL ORDER BY CLAUSE WITH ASCENDING ORDER

- Whenever we want to sort the records based on the columns stored in the tables of the SQL database, then we consider using the ORDER BY clause in SQL.
- The ***ORDER BY clause in SQL helps us sort the records based on a table's specific column.*** This means that initially, all the values stored in the column on which we are applying the ORDER BY clause will be sorted. Then the corresponding column values will be displayed in the same sequence in which the values we have obtained in the earlier step.
- Using the ORDER BY clause, we can sort the records in ascending or descending order as per our requirement. The records will be sorted in ascending order whenever the **ASC keyword** is used with the ORDER by clause. Whereas, **DESC keyword** will sort the records in descending order. If no keyword is specified after the column based on which we have to sort the records, then in that case, the sorting will be done by default in the ascending order.

Before writing the queries for sorting the records, let us understand the syntax.

Syntax to sort the records in ascending order:

1. **SELECT** ColumnName1,...,ColumnNameN **FROM** TableName **ORDER BY** Colu mnName **ASC**;

SQL ORDER BY CLAUSE WITH DESCENDING ORDER

- Whenever we want to sort the records based on the columns stored in the tables of the SQL database, then we consider using the ORDER BY clause in SQL.
- The ORDER BY clause in SQL helps us to sort the records based on the specific column of a table. This means that initially, all the values stored in the column on which we are applying the ORDER BY clause will be sorted. Then the corresponding column values will be displayed in the same sequence in which the values we have obtained in the earlier step.
- Using the ORDER BY clause, we can sort the records in ascending or descending order as per our requirement. The records will be sorted in ascending order whenever the ASC keyword is used with the ORDER by clause. ***DESC keyword will sort the records in descending order***. If no keyword is specified after the column based on which we have to sort the records, then, in that case, the sorting will be done by default in the ascending order.

Before writing the queries for sorting the records, let us understand the syntax.

Syntax to sort the records in descending order:

1. **SELECT** ColumnName1,...,ColumnNameN **FROM** TableName **ORDER BY** Colu mnNameDESC;

SQL ORDER BY RANDOM

If you want the resulting record to be **ordered randomly**, you should use the following codes according to several databases.

Here is a question: what is the need to fetch a random record or a row from a database?

Sometimes you may want to display random information like *articles, links, pages*, etc., to your user.

If you want to fetch random rows from any of the databases, you have to use some altered queries according to the databases.

- **Select a random row with Postgre SQL:**

1. **SELECT column FROM table ORDER BY** RANDOM () LIMIT 1;

SELECT * FROM Student_Records ORDER BY RANDOM ();

```
201|Akash|Delhi|18|89|A2
203|Yash|Delhi|20|89|A2
204|Bhavana|Delhi|19|78|
205|Yatin|Lucknow|20|75|B1
207|Vivek|Goa|20|62|
202|Bhavesh|Kanpur|19|93|A1
206|Ishika|Ghaziabad|19|51|C1

[Execution complete with exit code 0]
```

SELECT * FROM Student_Records ORDER BY RANDOM () LIMIT 1;

Every time you run you will get random data

```
201|Akash|Delhi|18|89|A2

[Execution complete with exit code 0]
204|Bhavana|Delhi|19|78|

[Execution complete with exit code 0]
205|Yatin|Lucknow|20|75|B1

[Execution complete with exit code 0]
```

SQL ORDER BY LIMIT

We can retrieve limited rows from the database. I can be used in pagination where are forced to show only limited records like 10, 50, 100 etc.

SELECT * FROM Student_Records ORDER BY RANDOM () LIMIT 3;

```
204|Bhavana|Delhi|19|78|
203|Yash|Delhi|20|89|A2
202|Bhavesh|Kanpur|19|93|A1

[Execution complete with exit code 0]
```

SQL SORTING ON MULTIPLE COLUMNS

Let's take an example of customer table which has many columns, the following SQL statement selects all customers from the table named "customer", stored by the "country" and "Customer-Name" columns:

```
SELECT * FROM customers
ORDER BY country, Customer-Name;
```

SELECT * FROM Student_Records ORDER BY Age,Grade

```
201|Akash|Delhi|18|89|A2
204|Bhavana|Delhi|19|78|
202|Bhavesh|Kanpur|19|93|A1
206|Ishika|Ghaziabad|19|51|C1
207|Vivek|Goa|20|62|
203|Yash|Delhi|20|89|A2
205|Yatin|Lucknow|20|75|B1

[Execution complete with exit code 0]
```

SQL ORDER BY DATE

- ORDER BY is a clause in SQL which is used with SELECT query to fetch the records in ascending or descending order from a table.
- Just like we sort the integer and the string values stored in the column of the tables, similarly, we can sort the dates stored in the SQL table's column.
- All the records will be, by default, sorted in the ascending order. To sort the records in descending order, the DESC keyword is used.

CREATE TABLE student(

ID INT PRIMARY KEY,

Name VARCHAR(20),

Percentage INT,

```
    Location VARCHAR(20),
    DateOfBirth DATE);

INSERT INTO student(ID, Name, Percentage, Location, DateOfBirth)
    VALUES (1, "Manthan Koli", 79, "Delhi", "2003-08-20"),
    (2, "Dev Dixit", 75, "Pune", "1999-06-17"),
    (3, "Aakash Deshmukh", 87, "Mumbai", "1997-09-12"),
    (4, "Aaryan Jaiswal", 90, "Chennai", "2005-10-02"),
    (5, "Rahul Khanna", 92, "Ambala", "1996-03-04");
```

SELECT * FROM student Order by DateOfBirth;

```
ID      Name    Percentage      Location        DateOfBirth
5       Rahul Khanna    92      Ambala  1996-03-04
3       Aakash Deshmukh 87      Mumbai  1997-09-12
2       Dev Dixit       75      Pune    1999-06-17
1       Manthan Koli    79      Delhi   2003-08-20
4       Aaryan Jaiswal  90      Chennai 2005-10-02

[Execution complete with exit code 0]
```

SELECT * FROM student Order by DateOfBirth desc;

```
ID      Name    Percentage      Location        DateOfBirth
4       Aaryan Jaiswal  90      Chennai 2005-10-02
1       Manthan Koli    79      Delhi   2003-08-20
2       Dev Dixit       75      Pune    1999-06-17
3       Aakash Deshmukh 87      Mumbai  1997-09-12
5       Rahul Khanna    92      Ambala  1996-03-04

[Execution complete with exit code 0]
```

SELECT Name,DATE_FORMAT(DateOfBirth,'%Y') FROM student ORDER BY DATE_FORMAT(DateOfBirth,'%Y') ;

```
Name    DATE_FORMAT(DateOfBirth,'%Y')
Rahul Khanna    1996
Aakash Deshmukh 1997
```

```
Dev Dixit       1999
Manthan Koli    2003
Aaryan Jaiswal  2005

[Execution complete with exit code 0]
```

SELECT Name,DAY(DateOfBirth) as day FROM student ORDER BY DAY(DateOfBirth) ;

```
Name    day
Aaryan Jaiswal  2
Rahul Khanna    4
Aakash Deshmukh 12
Dev Dixit       17
Manthan Koli    20

[Execution complete with exit code 0]
```

SQL INSERT

SQL INSERT STATEMENT

SQL INSERT statement is a SQL query. It is used to insert a single or multiple records in a table.

There are two ways to insert data in a table:

1. By SQL insert into statement
    1. By specifying column names
    2. Without specifying column names
2. By SQL insert into select statement

1) Inserting data directly into a table

You can insert a row in the table by using SQL INSERT INTO command.

There are two ways to insert values in a table.

In the first method there is no need to specify the column name where the data will be inserted, you need only their values.

```
INSERT INTO table_name
VALUES (value1, value2, value3....);
```

The second method specifies both the column name and values which you want to insert.

```
INSERT INTO table_name (column1, column2, column3....)
VALUES (value1, value2, value3.....);
```

Let's take an example of table which has five records within it.

```
INSERT INTO STUDENTS (ROLL_NO, NAME, AGE, CITY)
VALUES (1, ABHIRAM, 22, ALLAHABAD);
INSERT INTO STUDENTS (ROLL_NO, NAME, AGE, CITY)
VALUES (2, ALKA, 20, GHAZIABAD);
INSERT INTO STUDENTS (ROLL_NO, NAME, AGE, CITY)
VALUES (3, DISHA, 21, VARANASI);
INSERT INTO STUDENTS (ROLL_NO, NAME, AGE, CITY)
VALUES (4, ESHA, 21, DELHI);
```

```
INSERT INTO STUDENTS (ROLL_NO, NAME, AGE, CITY)
VALUES (5, MANMEET, 23, JALANDHAR);
```

It will show the following table as the final result.

ROLL_NO	NAME	AGE	CITY
1	ABHIRAM	22	ALLAHABAD
2	ALKA	20	GHAZIABAD
3	DISHA	21	VARANASI
4	ESHA	21	DELHI
5	MANMEET	23	JALANDHAR

You can create a record in CUSTOMERS table by using this syntax also.

```
INSERT INTO CUSTOMERS
VALUES (6, PRATIK, 24, KANPUR);
```

2) Inserting data through SELECT Statement

SQL INSERT INTO SELECT Syntax

```
INSERT INTO table_name
[(column1, column2, .... column)]
SELECT column1, column2, .... Column N
FROM table_name [WHERE condition];
```

> Note: when you add a new row, you should make sure that data type of the value and the column should be matched.

If any integrity constraints are defined for the table, you must follow them.

SQL INSERT Multiple Rows

Many times developers ask that is it possible to insert multiple rows into a single table in a single statement. Currently, developers have to write multiple insert statements when they insert values in a table. It is not only boring but also time-consuming.

```
CREATE TABLE Student_Records
(
Student_Id Int PRIMARY KEY,
First_Name VARCHAR (20),
Address VARCHAR (20),
Age Int NOT NULL,
Percentage Int NOT NULL,
Grade VARCHAR (10)
);


INSERT INTO Student_Records VALUES
(201, "Akash", "Delhi", 18, 89, "A2"),
(202, "Bhavesh", "Kanpur", 19, 93, "A1"),
(203, "Yash", "Delhi", 20, 89, "A2"),
(204, "Bhavana", "Delhi", 19, 78, NULL),
(205, "Yatin", "Lucknow", 20, 75, "B1"),
(206, "Ishika", "Ghaziabad", 19, 51, "C1"),
(207, "Vivek", "Goa", 20, 62, NULL);

SELECT * FROM Student_Records ORDER BY Age,Grade
```

SQL UPDATE

SQL UPDATE

The SQL commands (*UPDATE* and *DELETE*) are used to modify the data that is already in the database. The SQL DELETE command uses a WHERE clause.

SQL UPDATE statement is used to change the data of the records held by tables. Which rows is to be update, it is decided by a condition. To specify condition, we use WHERE clause.

The UPDATE statement can be written in following form:

UPDATE table_name **SET** [column_name1= value1,... column_nameN = valueN] [**WHERE** condition]

Or

UPDATE table_name
SET column_name = expression
WHERE conditions

```
CREATE TABLE Student_Records
(
Student_Id Int PRIMARY KEY,
First_Name VARCHAR (20),
Address VARCHAR (20),
Age Int NOT NULL,
Percentage Int NOT NULL,
Grade VARCHAR (10)
);
```

```
INSERT INTO Student_Records VALUES
```

(201, "Akash", "Delhi", 18, 89, "A2"),

(202, "Bhavesh", "Kanpur", 19, 93, "A1"),

(203, "Yash", "Delhi", 20, 89, "A2"),

(204, "Bhavana", "Delhi", 19, 78, NULL),

(205, "Yatin", "Lucknow", 20, 75, "B1"),

(206, "Ishika", "Ghaziabad", 19, 51, "C1"),

(207, "Vivek", "Goa", 20, 62, NULL);

UPDATE Student_Records

SET First_Name = "lochu"

WHERE Student_Id = 206;

SELECT * FROM Student_Records;

```
201|Akash|Delhi|18|89|A2
202|Bhavesh|Kanpur|19|93|A1
203|Yash|Delhi|20|89|A2
204|Bhavana|Delhi|19|78|
205|Yatin|Lucknow|20|75|B1
206|lochu|Ghaziabad|19|51|C1
207|Vivek|Goa|20|62|

[Execution complete with exit code 0]
```

Updating Multiple Fields:

If you are going to update multiple fields, you should separate each field assignment with a comma.

SQL UPDATE statement for multiple fields:

MYSQL SYNTAX FOR UPDATING TABLE:

```
UPDATE table_name
```

```
SET field1 = new-value1, field2 = new-value2,
[WHERE CLAUSE]
```

```
UPDATE Student_Records
SET First_Name = "lochu",Age = 22
WHERE Student_Id = 206;
```

```
SELECT * FROM Student_Records;
```

```
201|Akash|Delhi|18|89|A2
202|Bhavesh|Kanpur|19|93|A1
203|Yash|Delhi|20|89|A2
204|Bhavana|Delhi|19|78|
205|Yatin|Lucknow|20|75|B1
206|lochu|Ghaziabad|22|51|C1
207|Vivek|Goa|20|62|

[Execution complete with exit code 0]
```

SQL UPDATE with JOIN

SQL UPDATE JOIN means we will update one table using another table and join condition.

Let us take an example of a customer table. I have updated customer table that contains latest customer details from another source system. I want to update the customer table with latest data. In such case, I will perform join between target table and source table using join on customer ID.

Let's see the *syntax* of SQL UPDATE query with JOIN statement.

```
UPDATE customer_table
INNER JOIN
Customer_table
ON customer_table.rel_cust_name = customer_table.cust_id
SET customer_table.rel_cust_name = customer_table.cust_name
```

SQL UPDATE DATE

How to update a date and time field in SQL?

If you want to update a date & time field in SQL, you should use the following query.

let's see the syntax of sql update date.

```
UPDATE table
SET Column_Name = 'YYYY-MM-DD HH:MM:SS'
WHERE Id = value
```

SQL DELETE

SQL DELETE

The **SQL DELETE statement** is used to delete rows from a table. Generally DELETE statement removes one or more records from a table.

SQL DELETE Syntax

Let's see the Syntax for the SQL DELETE statement:

1. **DELETE FROM** table_name [**WHERE** condition];

   The WHERE clause in the SQL DELETE statement is optional and it identifies the rows in the column that gets deleted.

   WHERE clause is used to prevent the deletion of all the rows in the table, If you don't use the WHERE clause you might loss all the rows.

Another example of delete statement is given below

1. **DELETE FROM** EMPLOYEE;

It will delete all the records of EMPLOYEE table.

```
CREATE TABLE Student_Records
(
Student_Id Int PRIMARY KEY,
First_Name VARCHAR (20),
Address VARCHAR (20),
Age Int NOT NULL,
Percentage Int NOT NULL,
Grade VARCHAR (10)
);
```

```
INSERT INTO Student_Records VALUES
```

(201, "Akash", "Delhi", 18, 89, "A2"),

(202, "Bhavesh", "Kanpur", 19, 93, "A1"),

(203, "Yash", "Delhi", 20, 89, "A2"),

(204, "Bhavana", "Delhi", 19, 78, NULL),

(205, "Yatin", "Lucknow", 20, 75, "B1"),

(206, "Ishika", "Ghaziabad", 19, 51, "C1"),

(207, "Vivek", "Goa", 20, 62, NULL);

DELETE From Student_Records Where Student_Id = 205;

SELECT * FROM Student_Records;

```
201|Akash|Delhi|18|89|A2
202|Bhavesh|Kanpur|19|93|A1
203|Yash|Delhi|20|89|A2
204|Bhavana|Delhi|19|78|
206|Ishika|Ghaziabad|19|51|C1
207|Vivek|Goa|20|62|

[Execution complete with exit code 0]
```

DELETE From Student_Records;

SELECT * FROM Student_Records;

```
[Execution complete with exit code 0]
```

SQL DELETE TABLE

The DELETE statement is used to delete rows from a table. If you want to remove a specific row from a table you should use WHERE condition.

1. **DELETE FROM** table_name [**WHERE** condition];

But if you do not specify the WHERE condition it will remove all the rows from the table.

1. **DELETE FROM** table_name;

There are some more terms similar to DELETE statement like as DROP statement and TRUNCATE statement but they are not exactly same there are some differences between them.

Difference between DELETE and TRUNCATE statements

There is a slight difference b/w delete and truncate statement. The **DELETE statement** only deletes the rows from the table based on the condition defined by WHERE clause or delete all the rows from the table when condition is not specified.

But it does not free the space containing by the table.

The **TRUNCATE statement:** it is used to delete all the rows from the table **and free the containing space.**

1. **TRUNCATE TABLE** employee;

Difference b/w DROP and TRUNCATE statements

When you use the drop statement it deletes the table's row together with the table's definition so all the relationships of that table with other tables will no longer be valid.

When you drop a table:

- Table structure will be dropped
- Relationship will be dropped
- Integrity constraints will be dropped
- Access privileges will also be dropped

On the other hand when we **TRUNCATE** a table, the table structure remains the same, so you will not face any of the above problems

SQL DELETE ROW

Let us take an example of student.

If you want to delete a student with id 003 from the student_name table, then the SQL DELETE query should be like this:

DELETE FROM student_name

```
WHERE id = 003;
```

SQL DELETE ALL ROWS

The statement SQL DELETE ALL ROWS is used to delete all rows from the table. If you want to delete all the rows from student table the query would be like,

1. `DELETE FROM STUDENT_NAME;`

SQL DELETE DUPLICATE ROWS

If you have got a situation that you have multiple duplicate records in a table, so at the time of fetching records from the table you should be more careful. You make sure that you are fetching unique records instead of fetching duplicate records.

To overcome with this problem we use DISTINCT keyword.

It is used along with SELECT statement to eliminate all duplicate records and fetching only unique records.

SYNTAX:

The basic syntax to eliminate duplicate records from a table is:

```
SELECT DISTINCT column1, column2,....columnN
FROM table _name
WHERE [conditions]
```

SELECT DISTINCT First_Name,Address FROM Student_Records ORDER BY Student_Id;

```
Akash|Delhi
Bhavesh|Kanpur
Yash|Delhi
Bhavana|Delhi
Yatin|Lucknow
Ishika|Ghaziabad
Vivek|Goa
Yash|hello

[Execution complete with exit code 0]
```

SQL DELETE DATABASE

You can easily remove or delete indexes, tables and databases with the DROP statement.

The DROP index statement is:

Used to delete index in the table

DROP INDEX syntax for MySQL:

```
1. ALTER TABLE table_name DROP INDEX index_name
```

SQL DELETE VIEW

Before knowing about what is SQL delete view, it is important to know -

What is SQL view?

A view is a result set of a stored query on the data.

The SQL view is a table which does not physically exist. It is only a virtual table.

SQL VIEW can be created by a SQL query by joining one or more table.

Syntax for SQL create view -

```
CREATE VIEW view_name AS
SELECT columns
FROM tables
WHERE conditions;
```

If you want to delete a SQL view, It is done by SQL DROP command you should use the following syntax:

SQL DROP VIEW syntax:

```
1. DROP VIEW view_name
```

CREATE VIEW view_name AS

SELECT Student_Id,First_Name

FROM Student_Records

WHERE Address != "Delhi";

SELECT * FROM view_name;

```
202|Bhavesh
208|Yash
205|Yatin
206|Ishika
207|Vivek

[Execution complete with exit code 0]
```

CREATE VIEW view_name AS

SELECT Student_Id,First_Name

FROM Student_Records

WHERE Address != "Delhi";

DROP VIEW view_name;

SELECT * FROM view_name;

```
Error: near line 29: in prepare, no such table: view_name (1)

[Execution complete with exit code 1]
```

SQL DELETE JOIN

This is very commonly asked question that how to delete or update rows using join clause

It is not a very easy process, sometimes, we need to update or delete records on the basis of complex WHERE clauses.

There are three tables which we use to operate on SQL syntax for DELETE JOIN.

These tables are table1, table2 and target table.

SQL Syntax for delete JOIN

```
DELETE [target table]
FROM    [table1]
        INNER JOIN [table2]
ON [table1.[joining column] = [table2].[joining column]
WHERE   [condition]
```

Syntax for update

```
UPDATE [target table]
SET [target column] = [new value]
FROM    [table1]
        INNER JOIN [table2]
ON [table1.[joining column] = [table2].[joining column]
WHERE   [condition]
```

SQL JOIN

SQL JOIN

As the name shows, JOIN means *to combine something*. In case of SQL, JOIN means **"to combine two or more tables"**.

The SQL JOIN clause takes records from two or more tables in a database and combines it together.

ANSI standard SQL defines five types of JOIN :

1. inner join,
2. left outer join,
3. right outer join,
4. full outer join, and
5. cross join.

In the process of joining, rows of both tables are combined in a single table.

Why SQL JOIN is used?

If you want to access more than one table through a select statement.

If you want to combine two or more table then SQL JOIN statement is used .it combines rows of that tables in one table and one can retrieve the information by a SELECT statement.

The joining of two or more tables is based on common field between them.

SQL INNER JOIN also known as simple join is the most common type of join.

```
CREATE TABLE table1
(
Student_Id Int PRIMARY KEY,
First_Name VARCHAR (20),
Address VARCHAR (20),
Age Int NOT NULL,
```

```
Percentage Int NOT NULL,
Grade VARCHAR (10)
);


INSERT INTO table1 VALUES
(201, "Akash", "Delhi", 18, 89, "A2"),
(202, "Bhavesh", "Kanpur", 19, 93, "A1"),
(203, "Yash", "Delhi", 20, 89, "A2"),
(204, "Bhavana", "Delhi", 19, 78, NULL),
(205, "Yatin", "Lucknow", 20, 75, "B1"),
(206, "Ishika", "Ghaziabad", 19, 51, "C1"),
(207, "Vivek", "Goa", 20, 62, NULL);



CREATE TABLE table2
(
    Id Int PRIMARY KEY,
Last_Name VARCHAR (20),
Hobby VARCHAR (20),
```

```
Attendance Int NOT NULL,
Batch VARCHAR (10)
);


INSERT INTO table2 VALUES
(201,"kumar","drawing",45,"red"),
(202,"gowd","singling",55,"blue"),
(203,"shekar","dancing",65,"yellow"),
(204,"vilehya","painting",45,"green"),
(205,"krishna","travelling",85,"red"),
(206,"koushal","gardening",75,"yellow"),
(207,"shankar","drawing",55,"blue");


SELECT Student_Id, First_Name, Last_Name, Hobby
  FROM table1 t1, table2 t2
  where t1.Student_Id = t2.Id
```

```
201|Akash|kumar|drawing
202|Bhavesh|gowd|singling
203|Yash|shekar|dancing
204|Bhavana|vilehya|painting
205|Yatin|krishna|travelling
206|Ishika|koushal|gardening
207|Vivek|shankar|drawing

[Execution complete with exit code 0]
```

SELECT Student_Id, First_Name, Last_Name, Hobby

FROM table1 t1, table2 t2

```
201|Akash|kumar|drawing
201|Akash|gowd|singling
201|Akash|shekar|dancing
201|Akash|vilehya|painting
201|Akash|krishna|travelling
201|Akash|koushal|gardening
201|Akash|shankar|drawing
202|Bhavesh|kumar|drawing
202|Bhavesh|gowd|singling
202|Bhavesh|shekar|dancing
202|Bhavesh|vilehya|painting
202|Bhavesh|krishna|travelling
202|Bhavesh|koushal|gardening
202|Bhavesh|shankar|drawing
203|Yash|kumar|drawing
203|Yash|gowd|singling
203|Yash|shekar|dancing
203|Yash|vilehya|painting
203|Yash|krishna|travelling
203|Yash|koushal|gardening
203|Yash|shankar|drawing
204|Bhavana|kumar|drawing
204|Bhavana|gowd|singling
204|Bhavana|shekar|dancing
204|Bhavana|vilehya|painting
204|Bhavana|krishna|travelling
204|Bhavana|koushal|gardening
204|Bhavana|shankar|drawing
205|Yatin|kumar|drawing
205|Yatin|gowd|singling
205|Yatin|shekar|dancing
205|Yatin|vilehya|painting
205|Yatin|krishna|travelling
205|Yatin|koushal|gardening
205|Yatin|shankar|drawing
206|Ishika|kumar|drawing
206|Ishika|gowd|singling
206|Ishika|shekar|dancing
206|Ishika|vilehya|painting
206|Ishika|krishna|travelling
206|Ishika|koushal|gardening
206|Ishika|shankar|drawing
207|Vivek|kumar|drawing
207|Vivek|gowd|singling
207|Vivek|shekar|dancing
207|Vivek|vilehya|painting
207|Vivek|krishna|travelling
207|Vivek|koushal|gardening
207|Vivek|shankar|drawing

[Execution complete with exit code 0]
```

SQL OUTER JOIN

- In the SQL outer JOIN, ***all the content from both the tables is integrated together.***
- Even though the records from both the tables are matched or not, the matching and non-matching records from both the tables will be considered an output of the outer join in SQL.
- There are three different types of outer join in SQL:
    1. **Left Outer Join**
    2. **Right Outer Join**
    3. **Full Outer Join**

1. Left Outer Join:

- If we use the left outer join to combine two different tables, then we will get all the records from the left table. But we will get only those records from the right table, which have the corresponding key in the left table.
- Syntax of writing a query to perform left outer join:

```
SELECT TableName1.columnName1, TableName2.columnName2 FROM TableName1 LEFT OUTER JOIN TableName2 ON TableName1.ColumnName = TableName2.ColumnName;
```

```
CREATE TABLE table1
(
Student_Id Int PRIMARY KEY,
First_Name VARCHAR (20),
Address VARCHAR (20),
Age Int NOT NULL,
Percentage Int NOT NULL,
Grade VARCHAR (10)
);
```

```
INSERT INTO table1 VALUES
(201, "Akash", "Delhi", 18, 89, "A2"),
(302, "Bhavesh", "Kanpur", 19, 93, "A1"),
(303, "Yash", "Delhi", 20, 89, "A2"),
(204, "Bhavana", "Delhi", 19, 78, NULL),
(205, "Yatin", "Lucknow", 20, 75, "B1"),
(206, "Ishika", "Ghaziabad", 19, 51, "C1"),
(307, "Vivek", "Goa", 20, 62, NULL);
```

```
CREATE TABLE table2
(
    Id Int PRIMARY KEY,
Last_Name VARCHAR (20),
Hobby VARCHAR (20),
Attendance Int NOT NULL,
Batch VARCHAR (10)
);
```

INSERT INTO table2 VALUES

(201,"kumar","drawing",45,"red"),

(302,"gowd","singling",55,"blue"),

(203,"shekar","dancing",65,"yellow"),

(304,"vilehya","painting",45,"green"),

(205,"krishna","travelling",85,"red"),

(306,"koushal","gardening",75,"yellow"),

(207,"shankar","drawing",55,"blue");

SELECT t1.Student_Id, t1.First_Name, t2.Last_Name, t2.Batch
FROM table1 t1

LEFT OUTER JOIN table2 t2 ON t1.Student_Id = t2.Id;

```
Student_Id      First_Name      Last_Name       Batch
201     Akash   kumar   red
204     Bhavana NULL    NULL
205     Yatin   krishna red
206     Ishika  NULL    NULL
302     Bhavesh gowd    blue
303     Yash    NULL    NULL
307     Vivek   NULL    NULL

[Execution complete with exit code 0]
```

2. Right Outer Join:

- Right outer join is the reverse of left outer join. If we use the right outer join to combine two different tables, then we will get all the records from the right

table. But we will get only those records from the left table, which have the corresponding key in the right table.

- Syntax of writing a query to perform right outer join:

1. **SELECT** TableName1.columnName1, TableName2.columnName2 **FROM** Table Name1 RIGHT OUTER JOIN TableName2 **ON** TableName1.ColumnName = TableN ame2.ColumnName;

SELECT t2.Id, t1.First_Name, t2.Last_Name, t1.Grade FROM table1 t1

RIGHT OUTER JOIN table2 t2 ON t1.Student_Id = t2.Id;

```
Id      First_Name      Last_Name       Grade
201     Akash   kumar   A2
203     NULL    shekar  NULL
205     Yatin   krishna B1
207     NULL    shankar NULL
302     Bhavesh gowd    A1
304     NULL    vilehya NULL
306     NULL    koushal NULL

[Execution complete with exit code 0]
```

3. Full Outer Join:

- If we use a full outer join to combine two different tables, ***then we will get all the records from both the table,*** e., we will get all the records from the left table as well as the right table.
- **MySQL doesn't support FULL OUTER JOIN directly**. So to implement full outer join in MySQL, we will execute two queries in a single query. The first query will be of LEFT OUTER JOIN, and the second query will be of RIGHT OUTER JOIN. We will combine the first and second query with the UNION operator to see the results of FULL OUTER JOIN.
- Syntax of writing a query to perform full outer join:

1. **SELECT** TableName1.columnName1, TableName2.columnName2 **FROM** Table Name1 LEFT OUTER JOIN TableName2 **ON** TableName1.ColumnName = TableNa me2.ColumnName **UNION SELECT** TableName1.columnName1, TableName2.column

```
Name2 FROM TableName1  RIGHT OUTER JOIN TableName2  ON TableName1.Colu
mnName = TableName2.ColumnName;
```

```
CREATE TABLE table1
(
Student_Id Int PRIMARY KEY,
First_Name VARCHAR (20),
Address VARCHAR (20),
Age Int NOT NULL,
Percentage Int NOT NULL,
Grade VARCHAR (10)
);
```

```
INSERT INTO table1 VALUES
(201, "Akash", "Delhi", 18, 89, "A2"),
(302, "Bhavesh", "Kanpur", 19, 93, "A1"),
(303, "Yash", "Delhi", 20, 89, "A2"),
(204, "Bhavana", "Delhi", 19, 78, NULL),
(205, "Yatin", "Lucknow", 20, 75, "B1"),
(206, "Ishika", "Ghaziabad", 19, 51, "C1"),
(307, "Vivek", "Goa", 20, 62, NULL);
```

```
CREATE TABLE table2
(
Id Int PRIMARY KEY,
Last_Name VARCHAR (20),
Hobby VARCHAR (20),
```

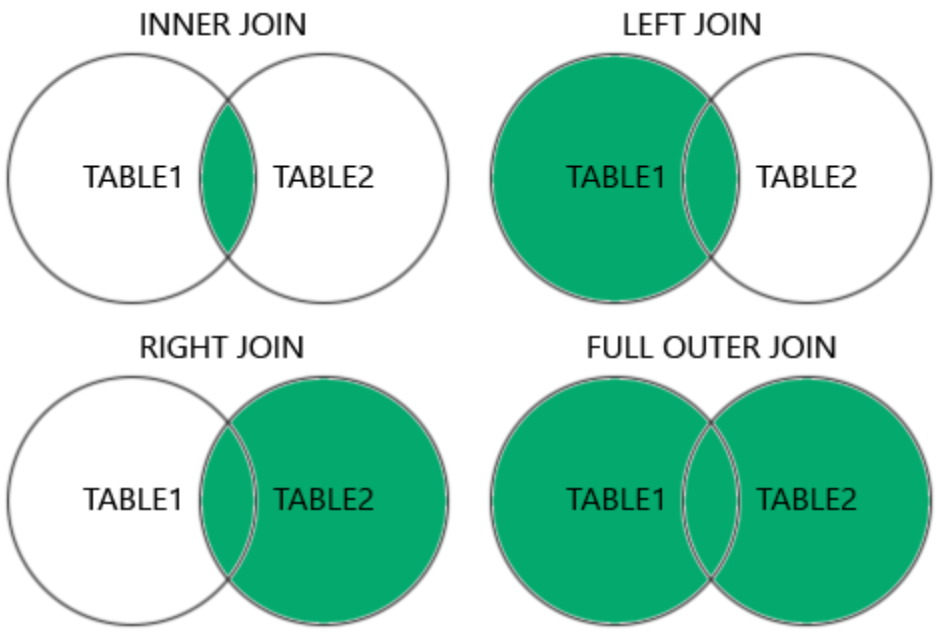
```
Attendance Int NOT NULL,
Batch VARCHAR (10)
);


INSERT INTO table2 VALUES
(201,"kumar","drawing",45,"red"),
(302,"gowd","singling",55,"blue"),
(203,"shekar","dancing",65,"yellow"),
(304,"vilehya","painting",45,"green"),
(205,"krishna","travelling",85,"red"),
(306,"koushal","gardening",75,"yellow"),
(207,"shankar","drawing",55,"blue");


SELECT t2.Id, t1.First_Name, t2.Last_Name, t1.Grade FROM table1 t1
RIGHT OUTER JOIN table2 t2 ON t1.Student_Id = t2.Id  UNION
SELECT t2.Id, t1.First_Name, t2.Last_Name, t1.Grade FROM table1 t1
LEFT OUTER JOIN table2 t2 ON t1.Student_Id = t2.Id
```

```
Id      First_Name      Last_Name       Grade
201     Akash   kumar   A2
203     NULL    shekar  NULL
205     Yatin   krishna B1
207     NULL    shankar NULL
302     Bhavesh gowd    A1
304     NULL    vilehya NULL
306     NULL    koushal NULL
NULL    Bhavana NULL    NULL
NULL    Ishika  NULL    C1
NULL    Yash    NULL    A2
NULL    Vivek   NULL    NULL

[Execution complete with exit code 0]
```

Different Types of SQL JOINs

Here are the different types of the JOINs in SQL:

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



SQL Left Join

- Join operation in SQL is used to **combine multiple tables together into a single table.**
- If we use ***left join to combine two different tables, then we will get all the records from the left table***. But we will get only those records from the right table, which have the corresponding key in the left table. Rest other records in the right table for which the common column value doesn't match with the common column value of the left table; then, it is displayed as NULL.

```
SELECT t2.Id, t1.First_Name, t2.Last_Name, t1.Grade FROM table1 t1
LEFT JOIN table2 t2 ON t1.Student_Id = t2.Id
```

```
Id      First_Name      Last_Name       Grade
201     Akash   kumar   A2
NULL    Bhavana NULL    NULL
205     Yatin   krishna B1
NULL    Ishika  NULL    C1
302     Bhavesh gowd    A1
NULL    Yash    NULL    A2
NULL    Vivek   NULL    NULL

[Execution complete with exit code 0]
```

SQL RIGHT JOIN

- Join operation in SQL is used to combine multiple tables together into a single table.
- ***If we use the right join to combine two different tables, then we will get all the records from the right table***. But we will get only those records from the left table, which have the corresponding key in the right table. Rest other records in the left table for which the common column value doesn't match with the common column value of the right table; displayed as NULL.
- Let us look at the syntax of writing a query to perform the right join operation in SQL.

```
SELECT TableName1.columnName1, TableName2.columnName2 FROM TableNa
me1
RIGHT JOIN TableName2 ON TableName1.ColumnName = TableName2.ColumnName;
```

```
SELECT t2.Id, t1.First_Name, t2.Last_Name, t1.Grade FROM table1
t1
Right JOIN table2 t2 ON t1.Student_Id = t2.Id
```

```
Id      First_Name      Last_Name       Grade
201     Akash   kumar   A2
203     NULL    shekar  NULL
205     Yatin   krishna B1
207     NULL    shankar NULL
302     Bhavesh gowd    A1
304     NULL    vilehya NULL
306     NULL    koushal NULL

[Execution complete with exit code 0]
```

SQL FULL JOIN

The SQL full join is the result of combination of both left and right outer join and the join tables have all the records from both tables. It puts NULL on the place of matches not found.

SQL full outer join and SQL join are same. generally it is known as SQL FULL JOIN.

Syntax for full outer join:

```
SELECT *
FROM table1
FULL OUTER JOIN table2
ON table1.column_name = table2.column_name;
```

SQL Cross Join

- Join operation in SQL is used to combine multiple tables together into a single table.
- If we use the cross join to combine two different tables, then we will get the Cartesian product of the sets of rows from the joined table. When each row of the first table is combined with each row from the second table, it is known as Cartesian join or cross join.
- After performing the cross join operation, the total number of rows present in the final table will be equal to the product of the number of rows present in table 1 and the number of rows present in table 2.
- If there are two records in table 1 and three records in table 2, then after performing cross join operation, we will get six records in the final table.
- Let us look at the syntax of writing a query to perform the cross join operation in SQL.

1. **SELECT** TableName1.columnName1, TableName2.columnName2 **FROM** Table Name1 CROSS JOIN TableName2 **ON** TableName1.ColumnName = TableNam e2.ColumnName;

SELECT t2.Id, t1.First_Name, t2.Last_Name, t1.Grade FROM table1 t1

CROSS JOIN table2 t2

```
Id      First_Name      Last_Name       Grade
201     Vivek   kumar   NULL
201     Yash    kumar   A2
201     Bhavesh kumar   A1
201     Ishika  kumar   C1
201     Yatin   kumar   B1
201     Bhavana kumar   NULL
201     Akash   kumar   A2
203     Vivek   shekar  NULL
203     Yash    shekar  A2
203     Bhavesh shekar  A1
203     Ishika  shekar  C1
203     Yatin   shekar  B1
203     Bhavana shekar  NULL
203     Akash   shekar  A2
205     Vivek   krishna NULL
205     Yash    krishna A2
205     Bhavesh krishna A1
205     Ishika  krishna C1
205     Yatin   krishna B1
205     Bhavana krishna NULL
205     Akash   krishna A2
207     Vivek   shankar NULL
207     Yash    shankar A2
207     Bhavesh shankar A1
207     Ishika  shankar C1
207     Yatin   shankar B1
207     Bhavana shankar NULL
207     Akash   shankar A2
302     Vivek   gowd    NULL
302     Yash    gowd    A2
302     Bhavesh gowd    A1
302     Ishika  gowd    C1
302     Yatin   gowd    B1
302     Bhavana gowd    NULL
302     Akash   gowd    A2
304     Vivek   vilehya NULL
304     Yash    vilehya A2
304     Bhavesh vilehya A1
304     Ishika  vilehya C1
304     Yatin   vilehya B1
304     Bhavana vilehya NULL
304     Akash   vilehya A2
306     Vivek   koushal NULL
306     Yash    koushal A2
306     Bhavesh koushal A1
306     Ishika  koushal C1
306     Yatin   koushal B1
306     Bhavana koushal NULL
306     Akash   koushal A2

[Execution complete with exit code 0]
```

SELECT t2.Id, t1.First_Name, t2.Last_Name, t1.Grade FROM table1 t1

CROSS JOIN table2 t2 ON t1.Student_Id = t2.Id

```
Id      First_Name      Last_Name       Grade
201     Akash   kumar   A2
205     Yatin   krishna B1
302     Bhavesh gowd    A1

[Execution complete with exit code 0]
```

SQL KEYS

SQL PRIMARY KEY

A column or columns is called **primary key (PK)** that *uniquely identifies each row in the table*.

If you want to create a primary key, you should define a PRIMARY KEY constraint when you create or modify a table.

When multiple columns are used as a primary key, it is known as **composite primary key**.

In designing the composite primary key, you should use as few columns as possible. It is good for storage and performance both, the more columns you use for primary key the more storage space you require.

Inn terms of performance, less data means the database can process faster.

Points to remember for primary key:

- Primary key enforces the entity integrity of the table.
- Primary key always has unique data.
- A primary key length cannot be exceeded than 900 bytes.
- A primary key cannot have null value.
- There can be no duplicate value for a primary key.
- A table can contain only one primary key constraint.

> When we specify a primary key constraint for a table, database engine automatically creates a unique index for the primary key column.

Main advantage of primary key:

The main advantage of this uniqueness is that we get **fast access**.

> In oracle, it is not allowed for a primary key to contain more than 32 columns.

MySQL:

```
CREATE TABLE students
(
S_Id int NOT NULL,
```

```
LastName varchar (255) NOT NULL,
FirstName varchar (255),
Address varchar (255),
City varchar (255),
PRIMARY KEY (S_Id)
)
```

SQL primary key for multiple columns:

MySQL, SQL Server, Oracle, MS Access:

```
CREATE TABLE students
(
S_Id int NOT NULL,
LastName varchar (255) NOT NULL,
FirstName varchar (255),
Address varchar (255),
City varchar (255),
CONSTRAINT pk_StudentID PRIMARY KEY (S_Id, LastName)
)
```

Note:you should note that in the above example there is only one PRIMARY KEY (pk_StudentID). However it is made up of two columns (S_Id and LastName).

SQL primary key on ALTER TABLE

When table is already created and you want to create a PRIMARY KEY constraint on the "S_Id" column you should use the following SQL:

Primary key on one column:

```
ALTER TABLE students
ADD PRIMARY KEY (S_Id)
```

Primary key on multiple column:

```
ALTER TABLE students
ADD CONSTRAINT pk_StudentID PRIMARY KEY (S_Id,LastName)
```

How to DROP a PRIMARY KEY constraint?

If you want to DROP (remove) a primary key constraint, you should use following syntax:

MySQL:

```
ALTER TABLE students
DROP PRIMARY KEY
```

SQL FOREIGN KEY

In the relational databases, a foreign key is a field or a column that is used to establish a link between two tables.

In simple words you can say that, a foreign key in one table used to point primary key in another table.

Here orders are given by students.

First table:

S_Id	LastName	FirstName	CITY
1	MAURYA	AJEET	ALLAHABAD
2	JAISWAL	RATAN	GHAZIABAD
3	ARORA	SAUMYA	MODINAGAR

Second table:

O_Id	OrderNo	S_Id
1	99586465	2
2	78466588	2
3	22354846	3
4	57698656	1

Here you see that "S_Id" column in the "Orders" table points to the "S_Id" column in "Students" table.

- The "S_Id" column in the "Students" table is the PRIMARY KEY in the "Students" table.
- The "S_Id" column in the "Orders" table is a FOREIGN KEY in the "Orders" table

SQL FOREIGN KEY constraint ON CREATE TABLE:

(Defining a foreign key constraint on single column)

To create a foreign key on the "S_Id" column when the "Orders" table is created:

MySQL:

```
CREATE TABLE orders
(
O_Id int NOT NULL,
Order_No  int NOT NULL,
S_Id int,
PRIMARY KEY (O_Id),
FOREIGN KEY (S_Id) REFERENCES Persons (S_Id)
)
```

SQL FOREIGN KEY constraint for ALTER TABLE:

If the Order table is already created and you want to create a FOREIGN KEY constraint on the "S_Id" column, you should write the following syntax:

Defining a foreign key constraint on single column:

MySQL / SQL Server / Oracle / MS Access:

```
ALTER TABLE Orders
ADD CONSTRAINT fk_PerOrders
FOREIGN KEY(S_Id)
REFERENCES Students (S_Id)
```

DROP SYNTAX for FOREIGN KEY COSTRAINT:

If you want to drop a FOREIGN KEY constraint, use the following syntax:

MySQL:

```
ALTER TABLE Orders
ROP FOREIGN KEY fk_PerOrders
```

Difference between primary key and foreign key in SQL:

- These are some important differences between primary key and foreign key in SQL-
- Primary key cannot be null on the other hand foreign key can be null.
- Primary key is always unique while foreign key can be duplicated.
- Primary key uniquely identify a record in a table while foreign key is a field in a table that is primary key in another table.
- There is only one primary key in the table on the other hand we can have more than one foreign key in the table.
- By default primary key adds a clustered index on the other hand foreign key does not automatically create an index, clustered or non-clustered. You must manually create an index for foreign key.

SQL Composite Key

A composite key is a combination of two or more columns in a table that can be used to uniquely identify each row in the table when the columns are combined uniqueness is guaranteed, but when it taken individually it does not guarantee uniqueness.

Sometimes more than one attributes are needed to uniquely identify an entity. A primary key that is made by the combination of more than one attribute is known as a composite key.

In other words we can say that:

Composite key is a key which is the combination of more than one field or column of a given table. It may be a candidate key or primary key.

MySQL:

```
CREATE TABLE SAMPLE_TABLE
(COL1 integer,
COL2 varchar(30),
COL3 varchar(50),
PRIMARY KEY (COL1, COL2));
```

Unique Key in SQL

- A unique key is a set of one or more than one fields/columns of a table that uniquely identify a record in a database table.
- You can say that it is little like primary key but it can accept only one null value and it cannot have duplicate values.
- The unique key and primary key both provide a guarantee for uniqueness for a column or a set of columns.
- There is an automatically defined unique key constraint within a primary key constraint.
- There may be many unique key constraints for one table, but only one PRIMARY KEY constraint for one table.

MySQL:

```
CREATE TABLE students
CREATE TABLE students
(
S_Id int NOT NULL,
LastName varchar (255) NOT NULL,
FirstName varchar (255),
City varchar (255),
UNIQUE (S_Id)
)
```

(Defining a unique key constraint on multiple columns):

MySQL / SQL Server / Oracle / MS Access:

```
 CREATE TABLE students
(
S_Id int NOT NULL,
LastName varchar (255) NOT NULL,
FirstName varchar (255),
City varchar (255),
CONSTRAINT uc_studentId UNIQUE (S_Id, LastName)
)
```

SQL UNIQUE KEY constraint on ALTER TABLE:

If you want to create a unique constraint on "S_Id" column when the table is already created, you should use the following SQL syntax:

(Defining a unique key constraint on single column):

MySQL / SQL Server / Oracle / MS Access:

```
ALTER TABLE students
ADD UNIQUE (S_Id)
```

(Defining a unique key constraint on multiple columns):

MySQL / SQL Server / Oracle / MS Access:

```
ALTER TABLE students
ADD CONSTRAINT uc_StudentId UNIQUE  (S_Id, LastName)
```

DROP SYNTAX FOR A FOREIGN KEY constraint:

If you want to drop a UNIQUE constraint, use the following SQL syntax:

MySQL:

```
ALTER TABLE students
DROP INDEX uc_studentID
```

Alternate Key in SQL

Alternate key is a secondary key it can be simple to understand by an example:

Let's take an example of student it can contain NAME, ROLL NO., ID and CLASS.

Here ROLL NO. is primary key and rest of all columns like NAME, ID and CLASS are alternate keys.

If a table has more than one candidate key, one of them will become the primary key and rest of all are called alternate keys.

In simple words, you can say that any of the candidate key which is not part of primary key is called an alternate key. So when we talk about alternate key, the column may not be primary key but still it is a unique key in the column.

> ***An alternate key is just a candidate key that has not been selected as the primary key.***

SQL Injection

The SQL Injection is a code penetration technique that might cause loss to our database. It is one of the most practiced web hacking techniques to place malicious code in SQL statements, via webpage input. SQL injection can be used to manipulate the application's web server by malicious users.

SQL injection generally occurs when we ask a user to input their username/userID. Instead of a name or ID, the user gives us an SQL statement that we will unknowingly run on our database. For Example - we create a SELECT statement by adding a variable "demoUserID" to select a string. The variable will be fetched from user input (getRequestString).

```
1. demoUserI = getrequestString("UserId");
2. demoSQL = "SELECT * FROM users WHERE UserId =" +demoUserId;
```

Types of SQL injection attacks

SQL injections can do more harm other than passing the login algorithms. Some of the SQL injection attacks include:

- Updating, deleting, and inserting the data: An attack can modify the cookies to poison a web application's database query.
- It is executing commands on the server that can download and install malicious programs such as Trojans.
- We are exporting valuable data such as credit card details, email, and passwords to the attacker's remote server.
- Getting user login details: It is the simplest form of SQL injection. Web application typically accepts user input through a form, and the front end passes the user input to the back end database for processing.

Example of SQL Injection

We have an application based on employee records. Any employee can view only their own records by entering a unique and private employee ID. We have a field like an Employee ID. And the employee enters the following in the input field:

236893238 or 1=1

It will translate to:

1. **SELECT** * **from** EMPLOYEE **where** EMPLOYEE_ID == 236893238 or 1=1

The SQL code above is valid and will return EMPLOYEE_ID row from the EMPLOYEE table. The 1=1 will return all records for which this holds true. All the employee data is compromised; now, the malicious user can also similarly delete the employee records.

Example:

1. **SELECT** * **from** Employee **where** (Username == "" or 1=1) AND (**Password**="" or 1=1).

Now the malicious user can use the '=' operator sensibly to retrieve private and secure user information. So instead of the query mentioned above, the following query, when exhausted, retrieve protected data, not intended to be shown to users.

1. **SELECT** * **from** EMPLOYEE **where** (Employee_name =" " or 1=1) AND (**Password**=" " or 1=1)

SQL STRING FUNCTIONS

SQL String Functions

In this article, you will learn about the various string functions of Structured Query Language in detail with examples.

What are String Functions in SQL?

SQL String functions are the predefined functions that allow the database users for string manipulation. These functions only accept, process, and give results of the string data type.

Following are the most important string functions in Structured Query Language:

1. ASCII()
2. CHAR_LENGTH()
3. CHARACTER_LENGTH()
4. CONCAT()
5. CONCAT_WS()
6. FIND_IN_SET()
7. FORMAT()
8. INSERT()
9. INSTR()
10. LCASE()
11. LEFT()
12. LOCATE()
13. LOWER()
14. LPAD()
15. LTRIM()
16. MID()
17. POSITION()
18. REPEAT()
19. REPLACE()
20. REVERSE()
21. RIGHT()
22. RPAD()

23. RTRIM()
24. SPACE()
25. STRCMP()
26. SUBSTR()
27. SUBSTRING()
28. SUBSTRING_INDEX()
29. UCASE()
30. UPPER()

ASCII String Function

This function in SQL returns the ASCII value of the character in the output. It gives the ASCII value of the left-most character of the string.

```
CREATE TABLE Student_records
(
Student_Id Int PRIMARY KEY,
First_Name VARCHAR (20),
Address VARCHAR (20),
Age Int NOT NULL,
Percentage Int NOT NULL,
Grade VARCHAR (10)
);



INSERT INTO Student_records  VALUES
(201, "Akash", "Delhi", 18, 89, "A2"),
(302, "Bhavesh", "Kanpur", 19, 93, "A1"),
(303, "Yash", "Delhi", 20, 89, "A2"),
```

(204, "Bhavana", "Delhi", 19, 78, NULL),

(205, "Yatin", "Lucknow", 20, 75, "B1"),

(206, "Ishika", "Ghaziabad", 19, 51, "C1"),

(307, "Vivek", "Goa", 20, 62, NULL);

SELECT First_Name,ASCII(First_Name) FROM Student_records;

```
First_Name      ASCII(First_Name)
Akash   65
Bhavana 66
Yatin   89
Ishika  73
Bhavesh 66
Yash    89
Vivek   86

[Execution complete with exit code 0]
```

BIT_LENGTH Function in SQL

The BIT_LENGTH string function of Structured Query Language returns the length of the string in bits.

Syntax of BIT_LENGTH String Function

Syntax1: This syntax uses the BIT_LENGTH function with the column name of the SQL table:

1. **SELECT** BIT_LENGTH(Column_Name) **AS** Alias_Name **FROM** Table_Name;

In the syntax, we have to specify the name of that column on which we want to perform the BIT_LENGTH string function for finding the length of the string in bits.

Syntax2: This syntax uses the BIT_LENGTH function with the string:

1. **SELECT** BIT_LENGTH(Original_String);

SELECT BIT_LENGTH('JAVATPOINT') AS BIT_LENGTH_word;

```
BIT_LENGTH_word
80

[Execution complete with exit code 0]
```

CHAR_LENGTH String Function

This string function returns the length of the specified word. It shows the number of characters from the word.

SELECT First_Name,CHAR_LENGTH(First_Name) FROM Student_records;

```
First_Name      CHAR_LENGTH(First_Name)
Akash   5
Bhavana 7
Yatin   5
Ishika  6
Bhavesh 7
Yash    4
Vivek   5

[Execution complete with exit code 0]
```

CHARACTER_LENGTH String Function

This string function returns the length of the given string. It shows the number of all characters and spaces from the sentence

SELECT CHARACTER_LENGTH("hello my name is lochani") ;

```
CHARACTER_LENGTH("hello my name is lochani")
24

[Execution complete with exit code 0]
```

CONCAT String Function

This string function concatenates two strings or words and forms a new string in the result.

Syntax of CONCAT String Function:

1. **SELECT** CONCAT(Column_Name1, Column_Name2, ... column_NameN) **AS** Alias_Name **FROM** Table_Name;
2. **SELECT** CONCAT(String_1, String_2, String_3, ..., String_N);

SELECT CONCAT(First_Name,"-",Grade) FROM Student_records;

```
CONCAT(First_Name,"-",Grade)
Akash-A2
NULL
Yatin-B1
Ishika-C1
Bhavesh-A1
Yash-A2
NULL

[Execution complete with exit code 0]
```

CONCAT_WS String Function

This string function concatenates multiple strings or words with the help of concatenating symbol. This function uses another parameter that denotes the concatenate symbol.

Syntax of CONCAT_WS String Function:

Syntax1: This syntax uses CONCAT_WS() with table columns:

1. **SELECT** CONCAT_WS(Concatenate_symbol, Column_Name1, Column_Name2, ... column_NameN) **AS** Alias_Name **FROM** Table_Name;

Syntax2: This syntax uses CONCAT_WS() with multiple strings:

1. **SELECT** CONCAT_WS(Concatenate_symbol, String_1, String_2, String_3, ..., String_N);

SELECT CONCAT_WS("::",First_Name,Age) FROM Student_records;

```
CONCAT_WS("::",First_Name,Age)
Akash::18
Bhavana::19
Yatin::20
Ishika::19
Bhavesh::19
Yash::20
Vivek::20

[Execution complete with exit code 0]
```

FIND_IN_SET String Function

This string function allows you to find the position of the searched_string in the set of strings.

Syntax of FIND_IN_SET String Function:

1. **SELECT** FIND_IN_SET(Concatenate_symbol, String_1, String_2, String_3, ..., String_N);

SELECT FIND_IN_SET('Goa', 'Mumbai, Goa, Banglore, Delhi, Kolkata, Chennai');

```
FIND_IN_SET('Goa', 'Mumbai, Goa, Banglore, Delhi, Kolkata, Chennai')
2
```

FORMAT String Function

This String function allows you to display the given string in the specified format.

Syntax of FORMAT String Function:

Syntax1: This syntax uses FORMAT() with table column:

1. **SELECT** FORMAT(Column_Name1, Format_String) **AS** Alias_Name **FROM** Table_Name;

Syntax2: This syntax uses FORMAT() with the string:

1. **SELECT** FORMAT(String_1, Format_String);

SELECT FORMAT(123456.789, '#,##0.00 "dollars"') AS FormattedValue;

FormattedValue

123,456.79 dollars

INSERT String Function

This string function allows the database users to insert the sub-string in the original string at the given index position.

Syntax of INSERT String Function:

Syntax1: This syntax uses INSERT() with the column of the SQL:

1. **SELECT INSERT**(Column_Name, Position, Number, String) **AS** Alias_Name **FRO M** Table_Name;

Syntax2: This syntax uses INSERT() with the string:

1. **SELECT INSERT**(String_1, Position, Number, String_2);

SELECT Address, INSERT(Address, 3, 4, 'Agra') AS Insert_Agra FROM Student_records;

```
Address Insert_Agra
Delhi   DeAgra
Delhi   DeAgra
Lucknow LuAgraw
Ghaziabad       GhAgrabad
Kanpur  KaAgra
Delhi   DeAgra
Goa     GoAgra

[Execution complete with exit code 0]
```

INSTR String Function

This string function returns the index value of the first occurrence of the given character in the string.

Syntax of INSTR String Function:

Syntax1: This syntax uses INSTR() with the column of the SQL:

1. **SELECT** INSTR(Column_Name, **character**) **AS** Alias_Name **FROM** Table_Name;

Syntax2: This syntax uses INSTR() with the string:

1. **SELECT** INSTR(String, **character**);

SELECT First_Name, INSTR(First_Name, 'a') AS FN FROM Student_records;

```
First_Name      FN
Akash   1
```

```
Bhavana 3
Yatin   2
Ishika  6
Bhavesh 3
Yash    2
Vivek   0

[Execution complete with exit code 0]
```

LCASE String Function

This string function allows users to convert the specified string into lower case letters.

Syntax of LCASE String Function:

Syntax1: This syntax uses LCASE() with the column of the SQL table:

1. **SELECT** LCASE(Column_Name) **AS** Alias_Name **FROM** Table_Name;

Syntax2: This syntax uses LCASE() with the string:

1. **SELECT** LCASE(String);

SELECT First_Name, LCASE(First_Name) AS LOW FROM Student_records;

```
First_Name      LOW
Akash   akash
Bhavana bhavana
Yatin   yatin
Ishika  ishika
Bhavesh bhavesh
Yash    yash
Vivek   vivek

[Execution complete with exit code 0]
```

LEFT String Function

This string function shows the leftmost characters from the given string. It reads the characters to the given index position.

Syntax of LEFT String Function:

Syntax1: This syntax uses LEFT() with the column of the SQL table:

1. **SELECT** LEFT(Column_Name, Index_position) **AS** Alias_Name **FROM** Table_Name;

Syntax2: This syntax uses LEFT() with the string:

1. **SELECT** LEFT(String, Index_position);

SELECT First_Name, LEFT(First_Name,3) AS Left_char FROM Student_records;

```
First_Name      Left_char
Akash   Aka
Bhavana Bha
Yatin   Yat
Ishika  Ish
Bhavesh Bha
Yash    Yas
Vivek   Viv

[Execution complete with exit code 0]
```

LENGTH Function in SQL

The LENGTH string function of Structured Query Language returns the number of characters of the given string or word.

Syntax of LENGTH String Function

Syntax1: This syntax uses the LENGTH function with the column name of the SQL table:

1. **SELECT** LENGTH(Column_Name) **AS** Alias_Name **FROM** Table_Name;

In the syntax, we have to specify the name of that column on which we want to perform the LENGTH string function for finding the number of characters of each value.

Syntax2: This syntax uses the LENGTH function with the string:

1. **SELECT** LENGTH(Original_String);

SELECT LENGTH(' JAVATPOINT') AS LENGTH_word;

```
LENGTH_word
11
```

```
[Execution complete with exit code 0]
```

LOCATE String Function

This string function shows the index value of the first occurrence of the word in the given string.

Syntax of LOCATE String Function:

Syntax1: This syntax uses LOCATE() with the column of the SQL table:

1. **SELECT** LOCATE(Search_string, Column_Name, Search_position) **AS** Alias_Name **FROM** Table_Name;

Syntax2: This syntax uses LOCATE() with the string:

1. **SELECT** LOCATE(Search_string, String Search_position);

SELECT First_Name, LOCATE('a', First_Name, 2) AS LOCATE_r FROM Student_records;

```
First_Name      LOCATE_r
Akash   3
Bhavana 3
Yatin   2
Ishika  6
Bhavesh 3
Yash    2
Vivek   0

[Execution complete with exit code 0]
```

LOWER String Function

This string function allows users to convert the specified string into lower case letters. This function is also the same as the LCASE() string function.

Syntax of LOWER String Function:

Syntax1: This syntax uses LOWER() with the column of the SQL table:

1. **SELECT** LOWER(Column_Name) **AS** Alias_Name **FROM** Table_Name;

Syntax2: This syntax uses LOWER() with the string:

1. **SELECT** LOWER(String);

LPAD String Function

This string function adds the given symbol to the left of the given string.

Syntax of LPAD String Function:

Syntax1: This syntax uses LPAD() with the column of the SQL table:

1. **SELECT** LPAD(Column_Name, **size**, symbol) **AS** Alias_Name **FROM** Table_Nam e;

Syntax2: This syntax uses LPAD() with the string:

1. **SELECT** LPAD(String, **size**, symbol);

SELECT First_Name, LPAD(First_Name,10,'*') AS star FROM Student_records;

```
First_Name      star
Akash   *****Akash
Bhavana ***Bhavana
Yatin   *****Yatin
Ishika  ****Ishika
Bhavesh ***Bhavesh
Yash    ******Yash
Vivek   *****Vivek

[Execution complete with exit code 0]
```

LTRIM String Function

This string function cuts the given character or string from the left of the given original string. It also removes the space from the left of the specified string.

Syntax of LTRIM String Function:

Syntax1: This syntax uses LTRIM() with the column of the SQL table:

1. **SELECT** LTRIM(Column_Name, string) **AS** Alias_Name **FROM** Table_Name;

Syntax2: This syntax uses LTRIM() with the string:

1. **SELECT** LTRIM(Original_String, trimmed_string);

SELECT LTRIM('NEW DELHI IS THE CAPITAL OF INDIA', 'NEW DELHI');

```
S THE CAPITAL OF INDIA

[Execution complete with exit code 0]
```

SELECT LTRIM('####98221545', '#');

```
98221545

[Execution complete with exit code 0]
```

MID String Function

This string function extracts the sub-string from the given position of the original string.

Syntax of MID String Function:

Syntax1: This syntax uses MID() with the column of the SQL table:

1. **SELECT** MID(Column_Name, Starting_Position, Length) **AS** Alias_Name **FROM** Table_Name;

Syntax2: This syntax uses MID() with the string:

1. **SELECT** MID(Original_String, Starting_Position, Length);

SELECT First_Name, MID(First_Name, 3, 8) AS MID_name FROM Student_records;

```
First_Name      MID_name
Akash   ash
```

```
Bhavana avana
Yatin   tin
Ishika  hika
Bhavesh avesh
Yash    sh
Vivek   vek

[Execution complete with exit code 0]
```

NOW()

- NOW () function returns the current system' date and time.

Syntax to find the current date and time:

1. **SELECT** NOW ();

OR

Syntax to find the current date and time from a table's column:

1. **SELECT** NOW () **FROM** TableName;

SELECT NOW () AS CurrentDatenTime;

2021-09-13 10:38:24

POSITION String Function

This string function finds the position of the first occurrence of the given string in the main string.

Syntax of POSITION String Function:

Syntax1: This syntax uses POSITION() with the column of the SQL table:

1. **SELECT** POSITION(String IN Column_Name) **AS** Alias_Name **FROM** Table_Name;

Syntax2: This syntax uses POSITION() with the string:

1. **SELECT** POSITION(String IN Original_String);

SELECT First_Name, POSITION('a' IN First_Name) AS pos FROM Student_records;

```
First_Name      pos
Akash   1
Bhavana 3
Yatin   2
Ishika  6
Bhavesh 3
Yash    2
Vivek   0

[Execution complete with exit code 0]
```

REPEAT String Function

This string function writes the given string or character till the given number of times.

Syntax of REPEAT String Function:

Syntax1: This syntax uses REPEAT() with the column of the SQL table:

1. **SELECT** REPEAT(Column_Name, Repetation_Number) **AS** Alias_Name **FROM** Table_Name;

Syntax2: This syntax uses REPEAT() with the string:

1. **SELECT** REPEAT(String, Repetation_Number);

SELECT First_Name, REPEAT(Age,3) AS rep FROM Student_records;

```
First_Name      rep
Akash   181818
Bhavana 191919
Yatin   202020
Ishika  191919
Bhavesh 191919
Yash    202020
Vivek   202020

[Execution complete with exit code 0]
```

REPLACE String Function

This string function cuts the given string by removing the given sub-string.

Syntax of REPLACE String Function:

Syntax1: This syntax uses REPLACE() with the column of the SQL table:

1. **SELECT** REPLACE(Column_Name, sub_string) **AS** Alias_Name **FROM** Table_Name;

Syntax2: This syntax uses REPLACE() with the string:

1. **SELECT** REPLACE(Original_String, sub_string);

SELECT Address, REPLACE(Address, 'a') AS REPLACE_a_Address FROM Student_records;

REVERSE String Function

This string function of Structured query Language reverses all the characters of the string.

Syntax of REVERSE String Function:

Syntax1: This syntax uses REVERSE() with the column of the SQL table:

1. **SELECT** REVERSE(Column_Name) **AS** Alias_Name **FROM** Table_Name;

Syntax2: This syntax uses REVERSE() with the string:

1. **SELECT** REVERSE(String);

SELECT Address, REVERSE(Address) AS rev FROM Student_records;

```
Address rev
Delhi   ihleD
Delhi   ihleD
Lucknow wonkcuL
Ghaziabad       dabaizahG
Kanpur  rupnaK
Delhi   ihleD
```

```
Goa     aoG

[Execution complete with exit code 0]
```

RIGHT String Function

This string function shows the right-most characters from the given string. It reads the characters from the right side to the given index position.

Syntax of RIGHT String Function:

Syntax1: This syntax uses RIGHT() with the column of the SQL table:

1. **SELECT** RIGHT(Column_Name, Index_position) **AS** Alias_Name **FROM** Table_Name;

Syntax2: This syntax uses RIGHT() with the string:

1. **SELECT** RIGHT(String, Index_position);

SELECT First_Name, RIGHT(First_Name,3) AS reigt FROM Student_records;

```
First_Name      reigt
Akash   ash
Bhavana ana
Yatin   tin
Ishika  ika
Bhavesh esh
Yash    ash
Vivek   vek

[Execution complete with exit code 0]
```

ROUND()

The ROUND () function is used to round a numeric column to the number of decimals specified.

Syntax to round the numeric values:

1. **SELECT** ROUND (NumericValue, Decimals);

where,

Decimal represents the number of decimals to be fetched.

OR

Syntax to round the numeric values from the table's column:

1. **SELECT** ROUND (ColumnName, Decimals) **FROM** TableName;

SELECT ROUND (18000.44444, 0) AS RoundedValue;

```
RoundedValue
18000

[Execution complete with exit code 0]
```

RPAD String Function

This string function adds the given symbol to the right of the given string.

Syntax of RPAD String Function:

Syntax1: This syntax uses RPAD() with the column of the SQL table:

1. **SELECT** RPAD(Column_Name, **size**, symbol) **AS** Alias_Name **FROM** Table_Nam e;

Syntax2: This syntax uses RPAD() with the string:

1. **SELECT** RPAD(String, **size**, symbol);

SELECT First_Name, RPAD(First_Name,10,'*') AS rpd FROM Student_records;

```
First_Name      rpd
Akash   Akash*****
Bhavana Bhavana***
Yatin   Yatin*****
Ishika  Ishika****
Bhavesh Bhavesh***
```

```
Yash    Yash******
Vivek   Vivek*****

[Execution complete with exit code 0]
```

RTRIM String Function

This string function cuts the given character or string from the right of the given original string. It also removes the space from the right of the specified string.

Syntax of RTRIM String Function:

Syntax1: This syntax uses RTRIM() with the column of the SQL table:

1. **SELECT** RTRIM(Column_Name) **AS** Alias_Name **FROM** Table_Name;

Syntax2: This syntax uses RTRIM() with the string:

1. **SELECT** RTRIM(Original_String);

SELECT RTRIM('NEW DELHI IS THE CAPITAL OF INDIA', 'CAPITAL OF INDIA');

```
NEW DELHI IS THE

[Execution complete with exit code 0]
```

SPACE String Function

This string function adds the specified number of spaces.

Syntax of SPACE String Function:

1. **SELECT** SPACE(Number);

Example of SPACE String function:

The following SELECT query adds the 11 spaces:

1. **SELECT** SPACE(11);

SUBSTR String Function

This string function extracts the sub-string from the given position of the original string.

Syntax of SUBSTR String Function:

Syntax1: This syntax uses SUBSTR() with the column of the SQL table:

1. **SELECT** SUBSTR(Column_Name, Starting_Position, Length) **AS** Alias_Name **FR OM** Table_Name;

Syntax2: This syntax uses SUBSTR() with the string:

1. **SELECT** SUBSTR(Original_String, Starting_Position, Length);

SELECT Address, SUBSTR(Address, 3, 8) AS Address FROM Student_records;

```
Address Address
Delhi   lhi
Delhi   lhi
Lucknow cknow
Ghaziabad       aziabad
Kanpur  npur
Delhi   lhi
Goa     a

[Execution complete with exit code 0]
```

STRCMP String Function

This string function compares the two specified strings with each other. This function returns 0 if both strings in SQL are similar, returns -1 if the first string is smaller than the second string, and returns 1 if the first string is bigger than the second string.

Syntax of STRCMP String Function:

Syntax1: This syntax uses STRCMP() with the columns of the SQL table:

1. **SELECT** STRCMP(Column_Name1, Column_Name2) **AS** Alias_Name **FROM** Tab le_Name;

Syntax2: This syntax uses STRCMP() with the two strings:

1. **SELECT** STRCMP(String1, String2);

SELECT First_Name,Address, STRCMP(First_Name,Address) AS cmp FROM Student_records;

```
First_Name      Address cmp
Akash   Delhi   -1
Bhavana Delhi   -1
Yatin   Lucknow 1
Ishika  Ghaziabad       1
Bhavesh Kanpur  -1
Yash    Delhi   1
Vivek   Goa     1

[Execution complete with exit code 0]
```

SUBSTRING String Function

This string function shows the character of the given index value in the original string.

Syntax of SUBSTRING String Function:

Syntax1: This syntax uses SUBSTRING() with the column of the SQL table:

1. **SELECT** SUBSTRING(Column_Name, Index_Position, Starting_Position) **AS** Alias _Name **FROM** Table_Name;

Syntax2: This syntax uses SUBSTRING() with the string:

1. **SELECT** SUBSTRING(Original_String, Index_Position, Starting_

SELECT First_Name,Address, SUBSTRING(First_Name,3,1) AS cmp FROM Student_records;

```
First_Name      Address cmp
Akash   Delhi   a
Bhavana Delhi   a
Yatin   Lucknow t
Ishika  Ghaziabad       h
Bhavesh Kanpur  a
Yash    Delhi   s
Vivek   Goa     v
```

```
[Execution complete with exit code 0]
```

UCASE String Function

This string function allows users to convert the specified string into upper case letters or capital letters.

Syntax of UCASE String Function:

Syntax1: This syntax uses UCASE() with the column of the SQL table:

1. **SELECT** UCASE(Column_Name) **AS** Alias_Name **FROM** Table_Name;

Syntax2: This syntax uses UCASE() with the string:

1. **SELECT** UCASE(String);

SELECT First_Name,UCASE(First_Name) AS uc FROM Student_records;

```
First_Name      uc
Akash   AKASH
Bhavana BHAVANA
Yatin   YATIN
Ishika  ISHIKA
Bhavesh BHAVESH
Yash    YASH
Vivek   VIVEK

[Execution complete with exit code 0]
```

UPPER String Function

This string function allows users to convert the specified string into the UPPER case letters. This function is also the same as the UCASE() string function.

Syntax of UPPER String Function:

Syntax1: This syntax uses UPPER() with the column of the SQL table:

1. **SELECT** UPPER(Column_Name) **AS** Alias_Name **FROM** Table_Name;

Syntax2: This syntax uses UPPER() with the string:

1. **SELECT** UPPER(String);

SQL MISCELLENOUS

SQL CAST Function

The SQL CAST function is mainly used to convert the expression from one data type to another data type. If the SQL Server CAST function is unable to convert a declaration to the desired data type, this function returns an error. We use the CAST function to convert numeric data into character or string data.

Syntax:

```
CAST (expression AS [data type])
```

Here, the [data type] is a type of valid data types in RDBMS.

The syntax is:

```
CAST (EXPRESSION AS Data_ Type[(Length)]
_ _ CAST in the SQL example
SELECT CAST (123 AS VARCHAR (20)) [result_name]
FROM [Source]
```

SELECT First_Name,CAST(Percentage AS integer) FROM Student_records;

```
First_Name      CAST(Percentage AS integer)
Akash   89
Bhavana 79
Yatin   75
Ishika  51
Bhavesh 93
Yash    89
Vivek   62

[Execution complete with exit code 0]
```

SQL Comments

SQL Comments are used to explain the sections of the SQL statements, and used to prevent the statements of SQL. In many programming languages, comments matter a lot.

A Microsoft Access database does not support the comments. So, **Mozilla Firefox** and **Microsoft Edge** use the Microsoft Access database in the examples.

There are three types of comments, which are given below:

1. **Single line comments.**
2. **Multi-line comments**
3. **Inline comments**

Single Line Comment

Comments starting and ending with a single line are said as individual line comments. The line which starts with '–' is a single line comment, and that particular line is not executed.

The text between **--** and end of the line is ignored and cannot be executed.

Syntax:

```
-- single-line comment

-- another comment

SELECT * FROM Customers;
```

The following example uses a single-line comment:

Example 1

```
--Select all:
SELECT * FROM Employees;
```

Multi-line Comments

Comments that start in one line and end in different front are said as multi-line comments. The text between **/*** and ***/** is ignored in the code part.

The line starting with '/*' is considered as a starting point of comment and terminated when '*/' lies at the end.

Syntax:

```
/* multi-line comment
another comment */
```

```
SELECT * FROM Customers;
```

Example 1

```
/*Select all the columns
of all the records
in the Customers table:*/
SELECT * FROM Employees;
```

Inline comments:

Inline comments are an extension of multi-line comments, and comments can be stated between the statements and are enclosed in between '/*' and '*/.'

Syntax:

SELECT * FROM /*Employees; */

Examples:

```
Multi line comment ->
/* SELECT * FROM Teachers;
SELECT * FROM Teacher_DETAILS;
SELECT * FROM Orders; */
SELECT * FROM Course;
```

SQL Comment Indicators

SQL Comment Indicator is indicated according to the given examples

It includes the double hyphen (—), braces ({ }), and C-style (/* . . . */) comment delimiters. It also includes the comments after the statement.

```
SELECT * FROM customer; -- Selects all rows and columns
SELECT * FROM employee; {Selects all rows and columns}
SELECT * FROM employee; /*Selects all columns and rows*/copy to the clipboar
d
```

Joining Three or More Tables in SQL

Joining multiple tables in SQL is some tricky task. It can be more difficult if you need to ***join*** more than two tables in single SQL query, we will analyze how to retrieve data from multiple tables using **INNER JOINs**. In this section, we have used two approaches to **join three or more tables in SQL.**

Example:

We are creating three tables, as follows:

student

marks

details

There are two approaches to join three or more tables in SQL:

1. Using JOINS in SQL:

The same logic is applied here which is used to join **two tables** i.e., the **minimum** number of join statements to join **n** tables are **(n-1)**.

```
create table student(
    s_id int primary key,
    s_name varchar(17));
insert into student values(1, 'Jack');
insert into student values(2, 'Rithvik');
insert into student values(3, 'Jaspreet');
insert into student values(4, 'Praveen');
insert into student values(5, 'Bisa');
insert into student values(6, 'Suraj');
```

```
create table marks(
    school_id int primary key,
    s_id int,
    score int,
    status varchar(20));
insert into marks values(1004, 1, 23, 'fail');
insert into marks values(1008, 6, 95, 'pass');
insert into marks values(1012, 2, 97, 'pass');
insert into marks values(1016, 7, 67, 'pass');
insert into marks values(1020, 3, 100, 'pass');
insert into marks values(1025, 8, 73, 'pass');
insert into marks values(1030, 4, 88, 'pass');
insert into marks values(1035, 9,  13, 'fail');
insert into marks values(1040, 5,  16, 'fail');
insert into marks values(1050, 10, 53, 'pass');


create table details(
    address_city varchar(20),
    email_ID varchar(20),
    school_id int,
    accomplishments varchar(50));
```

insert into details values('Bangalore', 'jsingh@jtp.com', 1020, 'ACM ICPC selected');

insert into details values('Hyderabad', 'praveen@jtp.com', 1030, 'Geek of the month');

insert into details values('Delhi', 'rithvik@jtp.com', 1012, 'IOI finalist');

insert into details values('Chennai', 'om@jtp.com', 1111, 'Geek of the year');

insert into details values('Banglore',' suraj@jtp.com', 1008, 'IMO finalist');

insert into details values('Mumbai', 'sasukeh@jtp.com', 2211, 'Made a robot');

insert into details values('Ahmedabad', 'itachi@jtp.com', 1172, 'Code Jam finalist');

insert into details values('Jaipur', 'kumar@jtp.com', 1972, 'KVPY finalist');

select s_name, score, status, address_city, email_id,

accomplishments from student s inner join mark m on

s.s_id = m.s_id inner join details d on

d.school_id = m.school_id;

```
s_name  score   status  address_city    email_id        accomplishments
Jaspreet        100     pass    Bangalore       jsingh@jtp.com  ACM ICPC
selected
Praveen 88      pass    Hyderabad       praveen@jtp.com Geek of the month
Rithvik 97      pass    Delhi   rithvik@jtp.com IOI finalist
Suraj   95      pass    Banglore         suraj@jtp.com  IMO finalist

[Execution complete with exit code 0]
```

2. Using the Parent-child Relationship:

In the parent-child relationship, we use **where** clause to join two or more tables. Create column **X** as a primary key in one table and a foreign key in another table

Look at the tables which are created:
s_id is the **primary key** in the student table and **foreign key** in the marks table. **(student (parent) - marks(child))**.
school_id is the **primary key** in the marks table and **foreign key** in the student table. **(marks(parent) - details(child))**.

```
select s_name, score, status, address_city,
email_id, accomplishments from student s,
marks m, details d where s.s_id = m.s_id and
m.school_id = d.school_id;
```

```
s_name  score   status  address_city    email_id        accomplishments
Jaspreet        100     pass    Bangalore       jsingh@jtp.com  ACM ICPC
selected
Praveen 88      pass    Hyderabad       praveen@jtp.com Geek of the month
Rithvik 97      pass    Delhi   rithvik@jtp.com IOI finalist
Suraj   95      pass    Banglore         suraj@jtp.com  IMO finalist

[Execution complete with exit code 0]
```

How to create functions in SQL?

SQL has many **built-in functions** for performing the calculation of data. **SQL** provides **built-in** functions to perform the **operations**. Some useful functions of SQL are performing the **mathematical calculations, string concatenation** and **sub-string** etc.

SQL functions are **divided** into two parts:

1. Aggregate Functions
2. Scalar Functions

SQL Aggregate Functions

SQL **Aggregate** functions return a single value which is calculated from the values.

- **AVG():** It returns the average value of the column.

- **COUNT():** It returns the number of rows in the table.
- **FIRST():** It returns the first value of the column.
- **LAST():** It returns the last value
- **MAX():** It returns the largest value of the column.
- **MIN():** It returns the smallest value of the column.
- **SUM():** It returns the sum of rows of the table.

SQL Scalar functions

SQL Scalar functions returns the single value according to the input value.

Scalar functions:

- **UCASE():** It converts the database field to uppercase.
- **LCASE():** It converts a field to lowercase.
- **MID():** It extracts characters from the text field.
- **LEN():** It returns the length of a text field.
- **ROUND():** It rounds a numeric field to the number of decimals.
- **NOW():** It returns the current date and time.
- **FORMAT():** It formats how a field is to be displayed.

How to Delete Duplicate Rows in SQL?

A) Delete duplicate rows with the DELETE JOIN statement

```
CREATE TABLE contacts (
id INT PRIMARY KEY AUTO_INCREMENT,
first_name VARCHAR(30) NOT NULL,
last_name VARCHAR(25) NOT NULL,
    email VARCHAR(210) NOT NULL,
    age VARCHAR(22) NOT NULL
```

);

INSERT INTO contacts (first_name,last_name,email,age)
VALUES ('Kavin','Peterson','kavin.peterson@verizon.net','21'),
('Nick','Jonas','nick.jonas@me.com','18'),
('Peter','Heaven','peter.heaven@google.com','23'),
('Michal','Jackson','michal.jackson@aol.com','22'),
('Sean','Bean','sean.bean@yahoo.com','23'),
('Tom ','Baker','tom.baker@aol.com','20'),
('Ben','Barnes','ben.barnes@comcast.net','17'),
('Mischa ','Barton','mischa.barton@att.net','18'),
('Sean','Bean','sean.bean@yahoo.com','16'),
('Eliza','Bennett','eliza.bennett@yahoo.com','25'),
('Michal','Krane','michal.Krane@me.com','25'),
('Peter','Heaven','peter.heaven@google.com','20'),
('Brian','Blessed','brian.blessed@yahoo.com','20'),
('Kavin','Peterson','kavin.peterson@verizon.net','30');

SELECT email,count(email) FROM contacts
group by email

```
email   count(email)
kavin.peterson@verizon.net      2
nick.jonas@me.com       1
peter.heaven@google.com 2
michal.jackson@aol.com  1
sean.bean@yahoo.com     2
tom.baker@aol.com       1
ben.barnes@comcast.net  1
mischa.barton@att.net   1
```

```
eliza.bennett@yahoo.com 1
michal.Krane@me.com     1
brian.blessed@yahoo.com 1

[Execution complete with exit code 0]
```

```
SELECT  email,count(email) FROM  contacts
group by email
having count(email) > 1;
```

```
email   count(email)
kavin.peterson@verizon.net      2
peter.heaven@google.com 2
sean.bean@yahoo.com     2

[Execution complete with exit code 0]
```

```
DELETE t1 FROM contacts t1
INNER JOIN contacts t2
WHERE
    t1.id > t2.id AND
    t1.email = t2.email;
```

```
select * from contacts;
```

```
id      first_name      last_name       email   age
1       Kavin   Peterson        kavin.peterson@verizon.net      21
2       Nick    Jonas   nick.jonas@me.com       18
3       Peter   Heaven  peter.heaven@google.com 23
4       Michal  Jackson michal.jackson@aol.com  22
5       Sean    Bean    sean.bean@yahoo.com     23
6       Tom     Baker   tom.baker@aol.com       20
7       Ben     Barnes  ben.barnes@comcast.net  17
8       Mischa  Barton  mischa.barton@att.net   18
10      Eliza   Bennett eliza.bennett@yahoo.com 25
11      Michal  Krane   michal.Krane@me.com     25
13      Brian   Blessed brian.blessed@yahoo.com 20

[Execution complete with exit code 0]
```

Nth Highest age

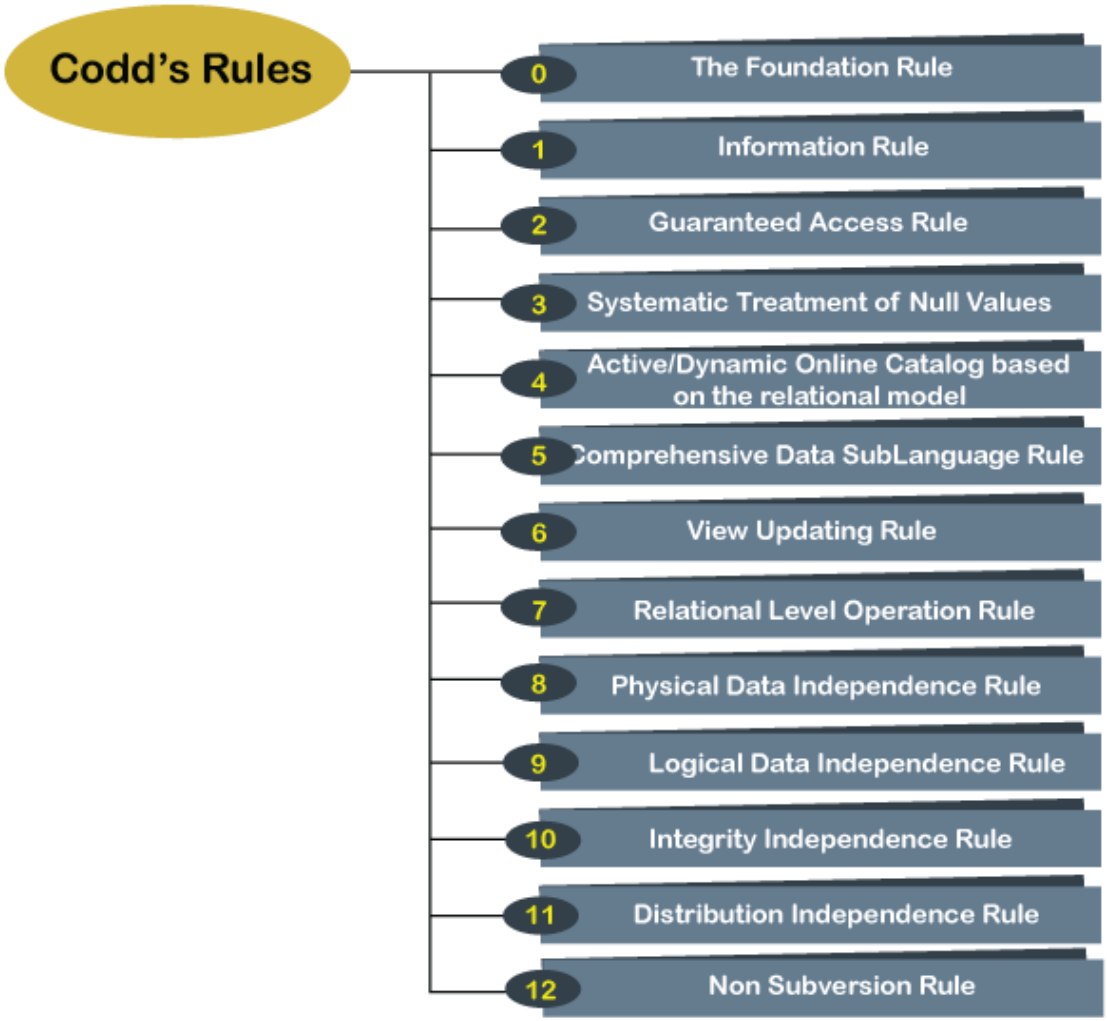
```
CREATE TABLE contacts (
id INT PRIMARY KEY AUTO_INCREMENT,
first_name VARCHAR(30) NOT NULL,
last_name VARCHAR(25) NOT NULL,
    email VARCHAR(210) NOT NULL,
    age VARCHAR(22) NOT NULL
);


INSERT INTO contacts (first_name,last_name,email,age)
VALUES ('Kavin','Peterson','kavin.peterson@verizon.net','21'),
       ('Nick','Jonas','nick.jonas@me.com','18'),
       ('Peter','Heaven','peter.heaven@google.com','23'),
       ('Michal','Jackson','michal.jackson@aol.com','22'),
       ('Sean','Bean','sean.bean@yahoo.com','23'),
       ('Tom ','Baker','tom.baker@aol.com','20'),
       ('Ben','Barnes','ben.barnes@comcast.net','17'),
       ('Mischa ','Barton','mischa.barton@att.net','18'),
       ('Sean','Bean','sean.bean@yahoo.com','16'),
       ('Eliza','Bennett','eliza.bennett@yahoo.com','25'),
       ('Michal','Krane','michal.Krane@me.com','25'),
       ('Peter','Heaven','peter.heaven@google.com','20'),
```

```
('Brian','Blessed','brian.blessed@yahoo.com','20'),
('Kavin','Peterson','kavin.peterson@verizon.net','30');

select first_name,age from contacts order by age desc;

SELECT min(age)
FROM (
    SELECT DISTINCT age
    FROM contacts
    ORDER BY age DESC
    LIMIT 2
) AS subquery;
```

```
first_name      age
Kavin   30
Eliza   25
Michal  25
Peter   23
Sean    23
Michal  22
Kavin   21
Tom     20
Peter   20
Brian   20
Nick    18
Mischa  18
Ben     17
Sean    16
min(age)
25

[Execution complete with exit code 0]
```

12 Codd's Rules

Every database has tables, and constraints cannot be referred to as a rational database system. And if any database has only relational data model, it cannot be a **Relational Database System (RDBMS)**. So, some rules define a database to be the correct RDBMS. These rules were developed by **Dr. Edgar F. Codd (E.F. Codd)** in **1985**, who has vast research knowledge on the Relational Model of database Systems. Codd presents his 13 rules for a database to test the concept of DBMS against his relational model, and if a database follows the rule, it is called a **true relational database (RDBMS)**. These 13 rules are popular in RDBMS, known as **Codd's 12 rules**.



Rule 0: The Foundation Rule

The database must be in relational form. So that the system can handle the database through its relational capabilities.

Rule 1: Information Rule

A database contains various information, and this information must be stored in each cell of a table in the form of rows and columns.

Rule 2: Guaranteed Access Rule

Every single or precise data (atomic value) may be accessed logically from a relational database using the combination of primary key value, table name, and column name.

Rule 3: Systematic Treatment of Null Values

This rule defines the systematic treatment of Null values in database records. The null value has various meanings in the database, like missing the data, no value in a cell, inappropriate information, unknown data and the primary key should not be null.

Rule 4: Active/Dynamic Online Catalog based on the relational model

It represents the entire logical structure of the descriptive database that must be stored online and is known as a database dictionary. It authorizes users to access the database and implement a similar query language to access the database.

Rule 5: Comprehensive Data SubLanguage Rule

The relational database supports various languages, and if we want to access the database, the language must be the explicit, linear or well-defined syntax, character strings and supports the comprehensive: data definition, view definition, data manipulation, integrity constraints, and limit transaction management operations. If the database allows access to the data without any language, it is considered a violation of the database.

Rule 6: View Updating Rule

All views table can be theoretically updated and must be practically updated by the database systems.

Rule 7: Relational Level Operation (High-Level Insert, Update and delete) Rule

A database system should follow high-level relational operations such as insert, update, and delete in each level or a single row. It also supports union, intersection and minus operation in the database system.

Rule 8: Physical Data Independence Rule

All stored data in a database or an application must be physically independent to access the database. Each data should not depend on other data or an application. If data is updated or the physical structure of the database is changed, it will not show any effect on external applications that are accessing the data from the database.

Rule 9: Logical Data Independence Rule

It is like physical data independence. It means, if any changes occurred to the logical level (table structures), it should not affect the user's view (application). For example, suppose a table either split into two tables, or two table joins to create a single table, these changes should not be impacted on the user view application.

Rule 10: Integrity Independence Rule

A database must maintain integrity independence when inserting data into table's cells using the SQL query language. All entered values should not be changed or rely on any external factor or application to maintain integrity. It is also helpful in making the database-independent for each front-end application.

Rule 11: Distribution Independence Rule

The distribution independence rule represents a database that must work properly, even if it is stored in different locations and used by different end-users. Suppose a user accesses the database through an application; in that case, they should not be aware that another user uses particular data, and the data they always get is only located on one site. The end users can access the database, and these access data should be independent for every user to perform the SQL queries.

Rule 12: Non Subversion Rule

The non-submersion rule defines RDBMS as a SQL language to store and manipulate the data in the database. If a system has a low-level or separate language other than SQL to access the database system, it should not subvert or bypass integrity to transform data

SQL Auto Increment

Databases are used to store the humongous amount of data logically. You might have come across various instances wherein you face difficulty mentioning a unique number for every record present in the table. This scenario is practically impossible because the manual entry is restricted. Thus, there's no scope for **incrementing the values**. In this kind of situation, you drop in other methods and choose auto increment in SQL.

The auto-increment in SQL entering a unique number increases the count to every new record present in the table automatically. To get familiar with auto-increment in SQL, let's look at some justifications to understand better.

What is an auto-increment in SQL?

As the name suggests, auto-increment is defined as a field that is mainly used to generate a unique number for each record being added to a table at any instance. In general terms, it is used for a primary key column because a primary key in SQL is supposed to be unique and not null. Thus, auto-increment makes it easy for the developers to automatically generate a unique number for all the records.

The data that you store in the database will not always have unique properties, so that some records are picked and manipulated uniquely. To ensure such unique identification of each record, you create an attribute for unique identification that can specify each row. Thus, after creating such a unique identity, you need to set it with auto increment so that values once incremented can be conveniently stored in the database.

Auto increment in SQL adds up an advantage by easily identifying each record. It is important to keep in mind that you need to set the column as auto-increment in the Create statement. Else, some problems may arise. Thus, to maintain integrity, you need to assign a primary key to the auto-increment, ensuring that the data's identity is maintained in the database.

Now, since you are quite familiar with auto-increment in SQL, let's proceed with discussing this field in different Database Management Systems.

Auto Increment features in SQL.

Some of the features of auto-increment are enlisted below:

1. It helps you to create Primary Key which might not have any unique identification attribute in data.
2. The value in auto-increment can be explicitly initialized and modified at any point in time.
3. Unique identification of records can be easily created.
4. With Auto Increment, flexibility to process the gap between each record can be handled easily.
5. Writing queries in SQL for auto-increment is syntactically easy.

```
CREATE TABLE animals (
    id INT NOT NULL AUTO_INCREMENT,
    name CHAR(30) NOT NULL,
    PRIMARY KEY (id)
);
INSERT INTO animals (name) VALUES
   ('dog'),('cat'),('penguin'),
   ('lax'),('whale'),('ostrich');
SELECT * FROM animals;
```

```
id      name
1       dog
2       cat
3       penguin
4       lax
5       whale
6       ostrich

[Execution complete with exit code 0]
```

Commit and Rollback in SQL

- Commit and rollback are the ***transaction control commands*** in SQL.
- All the commands that are executed consecutively, treated as a single unit of work and termed as a **transaction.**
- If you want to save all the commands which are executed in a transaction, then just after completing the transaction, you have to execute the ***commit*** command. This command will save all the commands which are executed on a table. All these changes made to the table will be saved to the disk permanently.
- Whenever the commit command is executed in SQL, all the updations which we have carried on the table will be uploaded to the server, and hence our work will be saved.
- The ***rollback*** command is used to get back to the previous permanent status of the table, which is saved by the commit command.
- Suppose, we have started editing a table and later thought that the changes that we have recently carried out on a table are not required. Then, in that case, we can roll

back our transaction, which simply means to get back to the previous permanent status of the table, which is saved by the commit command.

Note: One thing to note about the rollback command is that if you have already committed your recent changes, you cannot rollback your transaction. In that case, you can only roll to the last permanent change.

```
CREATE TABLE student(
    ID INT PRIMARY KEY,
    Name VARCHAR(20),
    Percentage INT,
    Location VARCHAR(20),
    DateOfBirth DATE);
START TRANSACTION;
INSERT INTO student(ID, Name, Percentage, Location,
DateOfBirth)
    VALUES (1, "Manthan Koli", 79, "Delhi", "2003-08-20"),
    (2, "Dev Dixit", 75, "Pune", "1999-06-17"),
    (3, "Aakash Deshmukh", 87, "Mumbai", "1997-09-12"),
    (4, "Aaryan Jaiswal", 90, "Chennai", "2005-10-02"),
    (5, "Rahul Khanna", 92, "Ambala", "1996-03-04");

SELECT * FROM student;
COMMIT;
SET autocommit = 0;

DELETE FROM student WHERE ID = 5;
SELECT *FROM student;
```

ROLLBACK;

SELECT * FROM student;

UPDATE student SET Percentage = 80 WHERE ID = 1;

SELECT * FROM student;

ROLLBACK;

SELECT * FROM student;

```
ID      Name    Percentage      Location        DateOfBirth
1       Manthan Koli    79      Delhi   2003-08-20
2       Dev Dixit       75      Pune    1999-06-17
3       Aakash Deshmukh 87      Mumbai  1997-09-12
4       Aaryan Jaiswal  90      Chennai 2005-10-02
5       Rahul Khanna    92      Ambala  1996-03-04
ID      Name    Percentage      Location        DateOfBirth
1       Manthan Koli    79      Delhi   2003-08-20
2       Dev Dixit       75      Pune    1999-06-17
3       Aakash Deshmukh 87      Mumbai  1997-09-12
4       Aaryan Jaiswal  90      Chennai 2005-10-02
ID      Name    Percentage      Location        DateOfBirth
1       Manthan Koli    79      Delhi   2003-08-20
2       Dev Dixit       75      Pune    1999-06-17
3       Aakash Deshmukh 87      Mumbai  1997-09-12
4       Aaryan Jaiswal  90      Chennai 2005-10-02
5       Rahul Khanna    92      Ambala  1996-03-04
ID      Name    Percentage      Location        DateOfBirth
1       Manthan Koli    80      Delhi   2003-08-20
2       Dev Dixit       75      Pune    1999-06-17
3       Aakash Deshmukh 87      Mumbai  1997-09-12
4       Aaryan Jaiswal  90      Chennai 2005-10-02
5       Rahul Khanna    92      Ambala  1996-03-04
ID      Name    Percentage      Location        DateOfBirth
1       Manthan Koli    79      Delhi   2003-08-20
2       Dev Dixit       75      Pune    1999-06-17
3       Aakash Deshmukh 87      Mumbai  1997-09-12
4       Aaryan Jaiswal  90      Chennai 2005-10-02
5       Rahul Khanna    92      Ambala  1996-03-04

[Execution complete with exit code 0]
```

SQL get month from the date

- To remember important dates, we can store them in the database tables of SQL.
- There may arise a situation in which instead of retrieving an entire date from the column of an SQL table, you only want the month of the date to be fetched from the table's columns.
- There are different dates functions available in SQL, using which we can fetch different parts of date from the columns as per our requirements.
- The **MONTH ()** function in SQL is used to get the month from an entire date stored in the table's column.
- Along with retrieving the date in the default format in which it is stored, there is a DATE_FORMAT () function in SQL using which the date values can be retrieved in a more readable format.

```
CREATE TABLE student(
    ID INT PRIMARY KEY,
    Name VARCHAR(20),
    Percentage INT,
    Location VARCHAR(20),
    DateOfBirth DATE);


INSERT INTO student(ID, Name, Percentage, Location,
DateOfBirth)
    VALUES (1, "Manthan Koli", 79, "Delhi", "2003-08-20"),
    (2, "Dev Dixit", 75, "Pune", "1999-06-17"),
    (3, "Aakash Deshmukh", 87, "Mumbai", "1997-09-12"),
    (4, "Aaryan Jaiswal", 90, "Chennai", "2005-10-02"),
    (5, "Rahul Khanna", 92, "Ambala", "1996-03-04");
```

SELECT Name,MONTH(DateOfBirth) FROM student;

```
Name    MONTH(DateOfBirth)
Manthan Koli    8
Dev Dixit       6
Aakash Deshmukh 9
Aaryan Jaiswal  10
Rahul Khanna    3

[Execution complete with exit code 0]
```

SELECT Name,DATE_FORMAT(DateOfBirth,'%M') as month FROM student;

```
Name    month
Manthan Koli    August
Dev Dixit       June
Aakash Deshmukh September
Aaryan Jaiswal  October
Rahul Khanna    March

[Execution complete with exit code 0]
```

Savepoint in SQL

- Savepoint is a command in SQL that is used with the rollback command.
- It is a command in Transaction Control Language that is used to mark the transaction in a table.
- Consider you are making a very long table, and you want to roll back only to a certain position in a table then; this can be achieved using the savepoint.
- If you made a transaction in a table, you could mark the transaction as a certain name, and later on, if you want to roll back to that point, you can do it easily by using the transaction's name.
- Savepoint is helpful when we want to roll back only a small part of a table and not the whole table. In simple words, we can say savepoint is a bookmark in SQL.

CREATE TABLE student(

ID INT PRIMARY KEY,

Name VARCHAR(20),

```
    Percentage INT,
    Location VARCHAR(20),
    DateOfBirth DATE);

INSERT INTO student(ID, Name, Percentage, Location,
DateOfBirth)
    VALUES (1, "Manthan Koli", 79, "Delhi", "2003-08-20"),
    (2, "Dev Dixit", 75, "Pune", "1999-06-17"),
    (3, "Aakash Deshmukh", 87, "Mumbai", "1997-09-12"),
    (4, "Aaryan Jaiswal", 90, "Chennai", "2005-10-02"),
    (5, "Rahul Khanna", 92, "Ambala", "1996-03-04");

START TRANSACTION;
INSERT INTO student VALUES (10, "Saurabh Singh", 54,
"Kashmir", "1989-01-06");
SAVEPOINT ini;
DELETE FROM student WHERE ID = 5;
SAVEPOINT del;
UPDATE student SET Percentage = 80 WHERE ID = 1;
SAVEPOINT upd;
ROLLBACK TO upd;
SELECT * FROM student;
ROLLBACK TO del;
SELECT * FROM student;
ROLLBACK TO ini;
```

SELECT * FROM student;

```
ID      Name    Percentage      Location        DateOfBirth
1       Manthan Koli    80      Delhi   2003-08-20
2       Dev Dixit       75      Pune    1999-06-17
3       Aakash Deshmukh 87      Mumbai  1997-09-12
4       Aaryan Jaiswal  90      Chennai 2005-10-02
10      Saurabh Singh   54      Kashmir 1989-01-06
ID      Name    Percentage      Location        DateOfBirth
1       Manthan Koli    79      Delhi   2003-08-20
2       Dev Dixit       75      Pune    1999-06-17
3       Aakash Deshmukh 87      Mumbai  1997-09-12
4       Aaryan Jaiswal  90      Chennai 2005-10-02
10      Saurabh Singh   54      Kashmir 1989-01-06
ID      Name    Percentage      Location        DateOfBirth
1       Manthan Koli    79      Delhi   2003-08-20
2       Dev Dixit       75      Pune    1999-06-17
3       Aakash Deshmukh 87      Mumbai  1997-09-12
4       Aaryan Jaiswal  90      Chennai 2005-10-02
5       Rahul Khanna    92      Ambala  1996-03-04
10      Saurabh Singh   54      Kashmir 1989-01-06

[Execution complete with exit code 0]
```

TIME Datatype in SQL

- There are many scenarios in SQL when you need to store the time in the SQL tables of your database.
- To store the time in your SQL tables, your first step should be to create a column in your table which is capable of storing the time.
- If you want the time to be stored in the column of your table, you need to create a column with the TIME data type.
- The TIME data type by default stores the time in "HH:MM:SS" format.
- Using the SELECT statement in SQL, you can retrieve the time from the column of the SQL tables.
- Along with retrieving the time in the default format in which it is stored, there is a TIME_FORMAT () function in SQL using which the time can be retrieved in a more readable format.
- You can retrieve the time in the default format in which it is stored in the table, i.e., "HH:MM:SS," or you also have the option of retrieving the specific parts of the time such as hour, minute, and seconds by choosing an appropriate parameter according to our requirement and passing it to the TIME_FORMAT() function. The time can also be retrieved in a 12 hour and a 24-hour format.

- We can also print the time followed by AM/ PM.

```
CREATE TABLE student(

ID INT PRIMARY KEY,

Name VARCHAR(20),

Percentage INT,

Location VARCHAR(20),

Item_OrderDate DATE,

Item_OrderTime TIME);
```

```
INSERT INTO student(ID, Name, Percentage, Location, Item_OrderDate,Item_OrderTime)

VALUES (1, "Manthan Koli", 79, "Delhi", "2003-08-20", "19:10:00"),

(2, "Dev Dixit", 75, "Pune", "1999-06-17", "13:10:00"),

(3, "Aakash Deshmukh", 87, "Mumbai", "1997-09-12", "09:16:00"),

(4, "Aaryan Jaiswal", 90, "Chennai", "2005-10-02", "16:10:00"),

(5, "Rahul Khanna", 92, "Ambala", "1996-03-04", "18:40:00");
```

```
SELECT Name,TIME_FORMAT (Item_OrderTime, "%r") AS Formatted_Time FROM student
```

```
Name    Formatted_Time
Manthan Koli    07:10:00 PM
Dev Dixit       01:10:00 PM
Aakash Deshmukh 09:16:00 AM
Aaryan Jaiswal  04:10:00 PM
Rahul Khanna    06:40:00 PM

[Execution complete with exit code 0]
```

SELECT Name,TIME_FORMAT (Item_OrderTime, "%T %p") AS Formatted_Time FROM student ;

```
Name    Formatted_Time
Manthan Koli    19:10:00 PM
Dev Dixit       13:10:00 PM
Aakash Deshmukh 09:16:00 AM
Aaryan Jaiswal  16:10:00 PM
Rahul Khanna    18:40:00 PM

[Execution complete with exit code 0]
```

;

SELECT Name,TIME_FORMAT (Item_OrderTime, "%H") AS Formatted_Time FROM student ;

```
Name    Formatted_Time
Manthan Koli    19
Dev Dixit       13
Aakash Deshmukh 09
Aaryan Jaiswal  16
Rahul Khanna    18

[Execution complete with exit code 0]
```

SELECT Name, TIME_FORMAT (Item_OrderTime, "%i") AS Formatted_Time FROM student ;

```
Name    Formatted_Time
Manthan Koli    10
Dev Dixit       10
Aakash Deshmukh 16
Aaryan Jaiswal  10
Rahul Khanna    40

[Execution complete with exit code 0]
```

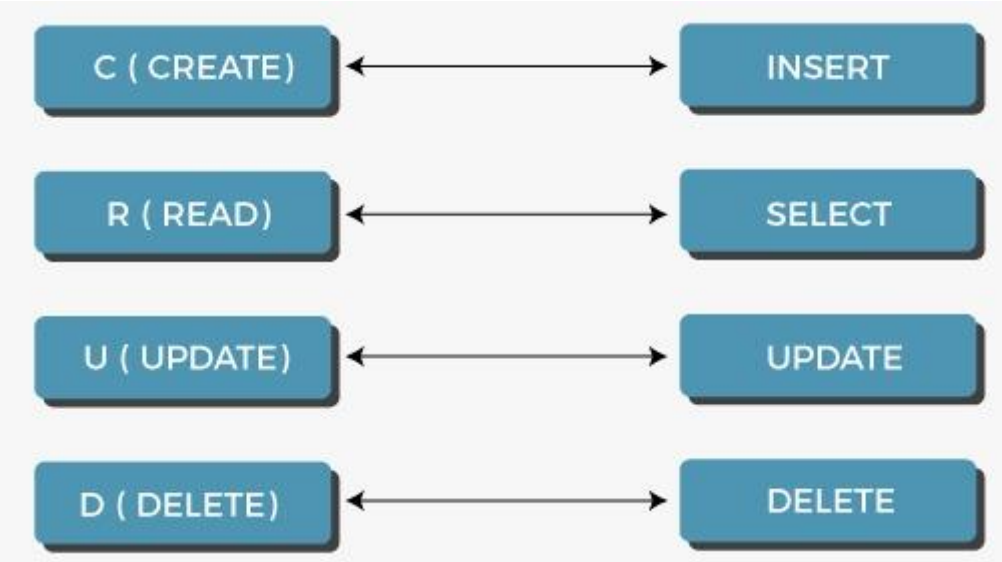
SELECT Name, TIME_FORMAT (Item_OrderTime, "%S") AS Item_OrderSeconds FROM student ;

```
Name    Item_OrderSeconds
Manthan Koli    00
Dev Dixit       00
Aakash Deshmukh 00
Aaryan Jaiswal  00
Rahul Khanna    00

[Execution complete with exit code 0]
```

CRUD Operations in SQL

As we know, CRUD operations act as the foundation of any computer programming language or technology. So before taking a deeper dive into any programming language or technology, one must be proficient in working on its CRUD operations. This same rule applies to databases as well.

Let us start with the understanding of CRUD operations in SQL with the help of examples. We will be writing all the queries in the supporting examples using the MySQL database.



1. Create:

In CRUD operations, 'C' is an acronym for **create,** which *means to add or insert data into the SQL table*. So, firstly we will create a table using CREATE command and then we will use the INSERT INTO command to insert rows in the created table.

Syntax for table creation:

1. **CREATE TABLE** Table_Name (ColumnName1 Datatype, ColumnName2 Datatype,..., ColumnNameN Datatype);

where,

- Table_Name is the name that we want to assign to the table.
- Column_Name is the attributes under which we want to store data of the table.

- Datatype is assigned to each column. Datatype decides the type of data that will be stored in the respective column.

Syntax for insertion of data in table:

1. **INSERT INTO** Table_Name (ColumnName1,...., ColumnNameN) **VALUES** (Valu e 1,....,Value N),....., (Value 1,.

2. Read:

In CRUD operations, 'R' is an acronym for **read,** which means ***retrieving or fetching the data from the SQL table.*** So, we will use the SELECT command to fetch the inserted records from the SQL table. We can retrieve all the records from a table using an asterisk (*) in a SELECT query. There is also an option of retrieving only those records which satisfy a particular condition by using the WHERE clause in a SELECT query.

Syntax to fetch all the records:

1. **SELECT** ***FROM** TableName;

Syntax to fetch records according to the condition:

1. **SELECT** ***FROM** TableName **WHERE** CONDITION;

3. Update:

In CRUD operations, 'U' is an acronym for the **update,** which ***means making updates to the records present in the SQL tables.*** So, we will use the UPDATE command to make changes in the data present in tables.

Syntax:

1. **UPDATE** Table_Name **SET** ColumnName = Value **WHERE** CONDITION;

4. Delete:

In CRUD operations, 'D' is an acronym for **delete,** which means ***removing or deleting the records from the SQL tables.*** We can delete all the rows from the SQL tables using the DELETE query. There is also an option to remove only the specific records that satisfy a particular condition by using the WHERE clause in a DELETE query.

Syntax to delete all the records:

1. **DELETE FROM** TableName;

Syntax to delete records according to the condition:

1. **DELETE FROM** TableName **WHERE** CONDITION;

SQL INDEX

The Index in SQL is a special table used to speed up the searching of the data in the database tables. It also retrieves a vast amount of data from the tables frequently. The INDEX requires its own space in the hard disk.

The index concept in SQL is same as the index concept in the novel or a book.

It is the best SQL technique for improving the performance of queries. The drawback of using indexes is that they slow down the execution time of UPDATE and INSERT statements. But they have one advantage also as they speed up the execution time of SELECT and WHERE statements.

In SQL, an Index is created on the fields of the tables. We can easily build one or more indexes on a table. The creation and deletion of the Index do not affect the data of the database.

In this article, you will learn how to create, alter, and remove an index in the SQL database.

Why SQL Index?

The following reasons tell why Index is necessary in SQL:

- SQL Indexes can search the information of the large database quickly.
- This concept is a quick process for those columns, including different values.
- This data structure sorts the data values of columns (fields) either in ascending or descending order. And then, it assigns the entry for each value.
- Each Index table contains only two columns. The first column is row_id, and the other is indexed-column.
- When indexes are used with smaller tables, the performance of the index may not be recognized.

Create an INDEX

In SQL, we can easily create the Index using the following CREATE Statement:

1. **CREATE INDEX** Index_Name **ON** Table_Name (Column_Name);

Here, **Index_Name** is the name of that index that we want to create, and **Table_Name** is the name of the table on which the index is to be created. The **Column_Name** represents the name of the column on which index is to be applied.

If we want to create an index on the combination of two or more columns, then the following syntax can be used in SQL:

1. **CREATE INDEX** Index_Name **ON** Table_Name (column_name1, column_name 2, ..., column_nameN);

Create UNIQUE INDEX

Unique Index is the same as the Primary key in SQL. The unique index does not allow selecting those columns which contain duplicate values.

This index is the best way to maintain the data integrity of the SQL tables.

Syntax for creating the Unique Index is as follows:

1. **CREATE UNIQUE INDEX** Index_Name **ON** Table_Name (Column_Name);

Example for creating a Unique Index in SQL:

Let's take the above Employee table. The following SQL query creates the unique index i**ndex_salary** on the **Emp_Salary** column of the **Employee** table.

1. **CREATE UNIQUE INDEX** index_salary **ON** Employee (Emp_Salary);

Rename an INDEX

We can easily rename the index of the table in the relational database using the ALTER command.

Syntax:

ALTER INDEX old_Index_Name RENAME **TO** new_Index_Name;

Example for Renaming the Index in SQL:

The following SQL query renames the index **'index_Salary'** to **'index_Employee_Salary'** of the above Employee table:

```
1. ALTER INDEX index_Salary RENAME TO index_Employee_Salary;
```

Remove an INDEX

An Index of the table can be easily removed from the SQL database using the DROP command. If you want to delete an index from the data dictionary, you must be the owner of the database or have the privileges for removing it.

Syntaxes for Removing an Index in relational databases are as follows:

In Oracle database:

```
1. DROP INDEX Index_Name;
```

In MySQL database:

```
1. ALTER TABLE Table_Name DROP INDEX Index_Name;
```

In Ms-Access database:

```
1. DROP INDEX Index_Name ON Table_Name;
```

In SQL Server Database:

```
1. DROP INDEX Table_Name.Index_Name;
```

Example for removing an Index in SQL:

Suppose we want to remove the above **'index_Salary'** from the SQL database. For this, we have to use the following SQL query:

```
1. DROP INDEX index_salary;
```

Alter an INDEX

An index of the table can be easily modified in the relational database using the ALTER command.

The basic syntax for modifying the Index in SQL is as follows:

1. **ALTER INDEX** Index_Name **ON** Table_Name REBUILD;

When should INDEXES not be used in SQL?

The Indexes should not be used in SQL in the following cases or situations:

- SQL Indexes can be avoided when the size of the table is small.
- When the table needs to be updated frequently.
- Indexed should not be used on those cases when the column of a table contains a large number of NULL values.

```
CREATE TABLE Student_records
(
Student_Id Int PRIMARY KEY,
First_Name VARCHAR (20),
Address VARCHAR (20),
Age Int NOT NULL,
Percentage Int NOT NULL,
Grade VARCHAR (10)
);
```

```
INSERT INTO Student_records  VALUES
(201, "Akash", "Delhi", 18, 89, "A2"),
(302, "Bhavesh", "Kanpur", 19, 93, "A1"),
(303, "Yash", "Delhi", 20, 89, "A2"),
(204, "Bhavana", "Delhi", 19, 78, NULL),
(205, "Yatin", "Lucknow", 20, 75, "B1"),
(206, "Ishika", "Ghaziabad", 19, 51, "C1"),
```

(307, "Vivek", "Goa", 20, 62, NULL);

CREATE INDEX Grade_sys ON Student_records(First_Name, Grade);

SELECT *

FROM Student_records

WHERE First_Name = 'John' AND Grade = 'A';

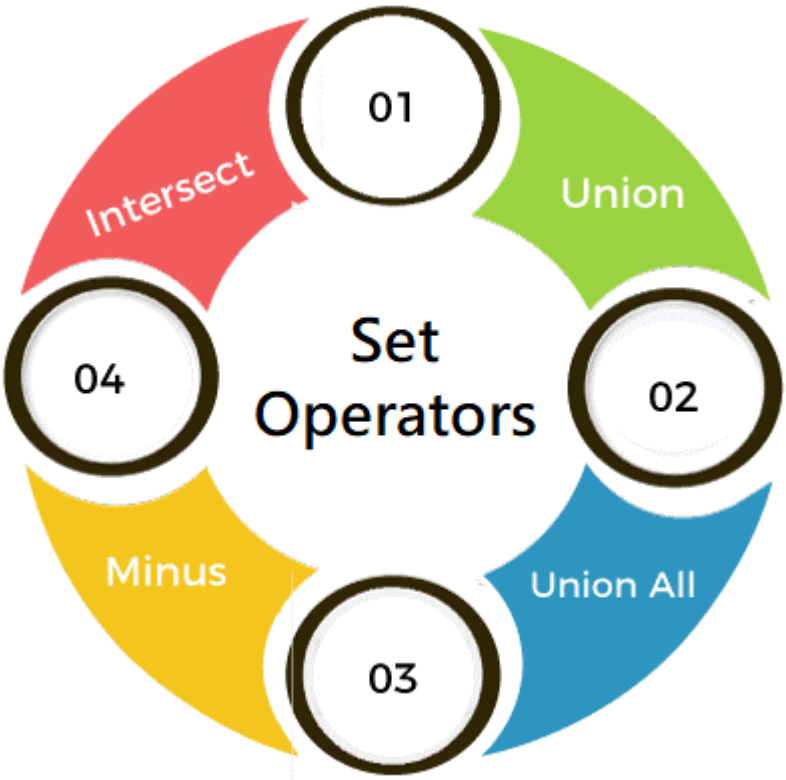
```
ERROR 1142 (42000) at line 21: INDEX command denied to user
'mycompiler'@'localhost' for table 'Student_records'

[Execution complete with exit code 1]
```

SET Operators in SQL

SET operators are special type of operators which are used to *combine the result of two queries.*

Operators covered under SET operators are:

1. **UNION**
2. **UNION ALL**
3. **INTERSECT**
4. **MINUS**



There are certain rules which must be followed to perform operations using SET operators in SQL. Rules are as follows:

1. **The number and order of columns must be the same.**
2. **Data types must be compatible**

1. UNION:

- UNION will be used to combine the result of two select statements.
- Duplicate rows will be eliminated from the results obtained after performing the UNION operation.

```
CREATE TABLE table1
(
Student_Id Int PRIMARY KEY,
First_Name VARCHAR (20),
Address VARCHAR (20),
Age Int NOT NULL,
```

```
Percentage Int NOT NULL,
Grade VARCHAR (10)
);

INSERT INTO table1 VALUES
(201, "Akash", "Delhi", 18, 89, "A2"),
(202, "Bhavesh", "Kanpur", 19, 93, "A1"),
(203, "Yash", "Delhi", 20, 89, "A2"),
(204, "Bhavana", "Delhi", 19, 78, NULL),
(205, "Yatin", "Lucknow", 20, 75, "B1"),
(206, "Ishika", "Ghaziabad", 19, 51, "C1"),
(207, "Vivek", "Goa", 20, 62, NULL);

CREATE TABLE table2
(
Student_Id Int PRIMARY KEY,
First_Name VARCHAR (20),
Address VARCHAR (20),
Age Int NOT NULL,
Percentage Int NOT NULL,
Grade VARCHAR (10)
);
INSERT INTO table2 VALUES
(201, "Akash", "Delhi", 18, 89, "A2"),
(102, "Bhavesh", "Kanpur", 19, 93, "A1"),
```

(203, "Yash", "Delhi", 20, 89, "A2"),

(104, "Bhavana", "Delhi", 19, 78, NULL),

(205, "Yatin", "Lucknow", 20, 75, "B1"),

(106, "Ishika", "Ghaziabad", 19, 51, "C1"),

(207, "Vivek", "Goa", 20, 62, NULL);

SELECT *FROM table1 UNION SELECT *FROM table2;

```
Student_Id      First_Name      Address Age     Percentage      Grade
201     Akash   Delhi   18      89      A2
202     Bhavesh Kanpur  19      93      A1
203     Yash    Delhi   20      89      A2
204     Bhavana Delhi   19      78      NULL
205     Yatin   Lucknow 20      75      B1
206     Ishika  Ghaziabad       19      51      C1
207     Vivek   Goa     20      62      NULL
102     Bhavesh Kanpur  19      93      A1
104     Bhavana Delhi   19      78      NULL
106     Ishika  Ghaziabad       19      51      C1

[Execution complete with exit code 0]
```

2. UNION ALL

- This operator combines all the records from both the queries.
- Duplicate rows will be not be eliminated from the results obtained after performing the UNION ALL operation.

SELECT *FROM table1 UNION ALL SELECT *FROM table2;

```
Student_Id      First_Name      Address Age     Percentage      Grade
201     Akash   Delhi   18      89      A2
202     Bhavesh Kanpur  19      93      A1
203     Yash    Delhi   20      89      A2
204     Bhavana Delhi   19      78      NULL
205     Yatin   Lucknow 20      75      B1
206     Ishika  Ghaziabad       19      51      C1
207     Vivek   Goa     20      62      NULL
102     Bhavesh Kanpur  19      93      A1
104     Bhavana Delhi   19      78      NULL
106     Ishika  Ghaziabad       19      51      C1
201     Akash   Delhi   18      89      A2
203     Yash    Delhi   20      89      A2
205     Yatin   Lucknow 20      75      B1
```

```
207     Vivek   Goa     20      62      NULL

[Execution complete with exit code 0]
```

3. INTERSECT:

- It is used to combine two SELECT statements, but it only returns the records which are common from both SELECT statements.

SELECT *FROM table1 INTERSECT SELECT *FROM table2;

```
Student_Id      First_Name      Address Age     Percentage      Grade
205     Yatin   Lucknow 20      75      B1
207     Vivek   Goa     20      62      NULL
203     Yash    Delhi   20      89      A2
201     Akash   Delhi   18      89      A2

[Execution complete with exit code 0]
```

4.MINUS:

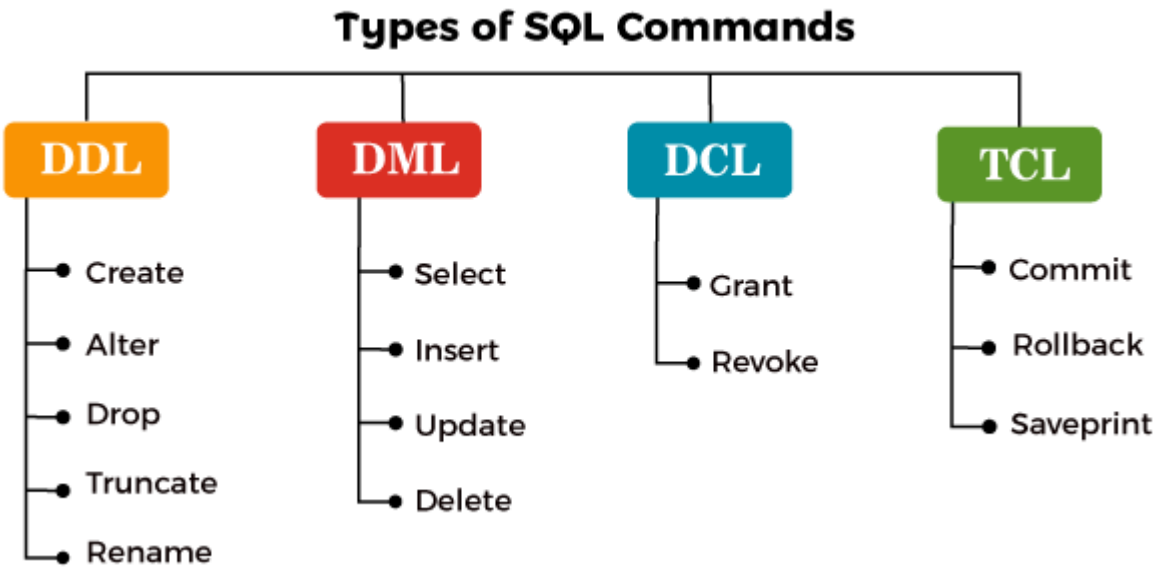
- It displays the rows which are present in the first query but absent in the second query with no duplicates.

SELECT * FROM table1 MINUS SELECT * FROM table2;

Types of SQL Commands

SQL is a structured query language, which is used to deal with structured data. Structured data is data that is generally stored in the form of relations or tables.

Whenever we store the data in tables or relations, we need SQL commands. Moreover, these commands are also required to retrieve the data which is stored in tables.



Let us take a deeper dive into the classification of SQL commands with the help of practical examples. We will use the MySQL database for writing all the queries.

(A) DDL

- DDL stands for **data definition language**. DDL Commands deal with the schema, i.e., the table in which our data is stored.
- All the structural changes such as creation, deletion and alteration on the table can be carried with the DDL commands in SQL.
- Commands covered under DDL are:
    1. **CREATE**
    2. **ALTER**
    3. **DROP**
    4. **TRUNCATE**
    5. **RENAME**

Let us see each of the commands in the DDL category with more details.

1. CREATE:

In SQL, whenever we wish to *create a new database or a table in a database*, we use **CREATE** command.

Syntax to create a new database:

1. **CREATE DATABASE** DatabaseName;

Syntax to create a table:

1. **CREATE TABLE** TableName (ColumnName1 datatype, ColumnName2 datatyp e,....., ColumnName3 datat

2. ALTER

In SQL, whenever we wish to alter the table structure, we will use the ALTER command. Using *alter command, we can make structural changes to the table, such* ***as adding a new column, removing or deleting an existing column from the table, changing the datatype of an existing column and renaming an existing column***.

Let's look at the syntax before writing the queries using ALTER command.

Syntax of ALTER command to add a new column:

1. **ALTER TABLE** table_name **ADD** column_name datatype(**size**);

Syntax of ALTER command to delete an existing column:

1. **ALTER TABLE** table_name **DROP COLUMN** column_name;

Syntax of ALTER command to rename the existing table's column:

1. **ALTER TABLE** table_name RENAME **COLUMN** old_column_name **TO** new_col umn_name;

Syntax of ALTER command to change the datatype of an existing column:

1. **ALTER TABLE** table_name **MODIFY** column_name datatype(**size**);

3. DROP

DROP command is **used to remove or delete the table's records and the table's structure from the database.**

Syntax:

1. **DROP TABLE** table_name;

. TRUNCATE

A TRUNCATE command is used to delete the table's records, *but the table's structure will remain unaffected in the database.*

Syntax:

```
1. TRUNCATE TABLE table_name;
```

RENAME

Rename COMMAND is used to give a new name to an existing table.

Syntax to rename a table:

```
1. RENAME TABLE old_table_name TO new_table_name;
```

(B) DML

- DML stands for *Data Manipulation Language*. Using DML commands in SQL, we can make changes in the data present in tables.
- Whenever we wish to manipulate the data or fetch the data present in SQL tables, we can use DML commands in SQL.
- DML commands in SQL will change the data, such as **inserting new records, deleting or updating existing records from the SQL tables**. We can also retrieve all the data from SQL tables according to our requirements.
- Commands covered under DDL are:
    1. **INSERT**
    2. **SELECT**
    3. **UPDATE**
    4. **DELETE**

Let us see each of the commands in the DML category with more details.

1. INSERT

INSERT command is ***used to insert records in a table***. We can insert a single as well as multiple records for a single table at the same time.

Syntax:

1. **INSERT INTO** table_name **VALUES** (Column1_Value, Column2_Value, …., ColumnN_Value);

SELECT

A ***SELECT command is used to retrieve the records from the table***. According to our requirements, we can retrieve all the records or some specific records from the table. Whenever we want to retrieve some specific records from the table, then we have to specify the WHERE clause in a SELECT query. WHERE clause will contain a condition, any record that matches the condition will be considered as a part of the output.

Syntax to retrieve all the records:

1. **SELECT** ***FROM** table_name;

Syntax to retrieve some specific records:

1. **SELECT** ***FROM** table_name **WHERE** condition;

UPDATE

UPDATE command works for the values present in the table. Whenever we wish to update a value for any record present in a table, we will use the UPDATE command in SQL.

Syntax:

1. **UPDATE** table_name **SET** column_name = value **WHERE** condition;

DELETE

DELETE command is used to remove records from a table.

Syntax:

1. **DELETE FROM** table_name;

© DCL

- DCL stands for **Data Control Language.**

- Whenever we want to control the access to the data present in SQL tables, we will use DCL commands in SQL. Only the authorized users can access the data stored in the tables.
- Every user will have some pre-defined privileges; accordingly, the data can be accessed by that particular user. Using the DCL commands in SQL, we can give privileges to the user on the SQL database and tables, or we can also revoke the given privileges from the user.
- Commands covered under DCL are:

1. GRANT

Access privileges can be assigned to a user for the databases and tables using the GRANT command.

2. REVOKE

All the access privileges which are already assigned to the user can be revoked by using the REVOKE command.

(D) TCL:

- TCL stands for **Transaction Control Language**. TCL commands are generally used in transactions.
- Using TCL commands in SQL, we can save our transactions to the database and roll them back to a specific point in our transaction. We can also save a particular portion of our transaction using the SAVEPOINT command.
- Commands covered under TCL are:

1. COMMIT:

To save all the operations executed in a particular transaction, we need to execute a commit command just after the transaction completion.

2. ROLLBACK

Using the rollback command in SQL, you can roll to the last saved state of a transaction.

3. SAVEPOINT

Using the SAVEPOINT command, you can assign a name to a specific part of the transaction.

SQL Subquery

The Subquery or Inner query is an SQL query placed inside another SQL query. It is embedded in the HAVING or WHERE clause of the SQL statements.

Following are the important rules which must be followed by the SQL Subquery:

1. The SQL subqueries can be used with the following statements along with the SQL expression operators:

- SELECT statement,
- UPDATE statement,
- INSERT statement, and
- DELETE statement.

2. The subqueries in SQL are always enclosed in the parenthesis and placed on the right side of the SQL operators.

3. We cannot use the ORDER BY clause in the subquery. But, we can use the GROUP BY clause, which performs the same function as the ORDER BY clause.

4. If the subquery returns more than one record, we have to use the multiple value operators before the Subquery.

5. We can use the BETWEEN operator within the subquery but not with the subquery.

Subquery with SELECT statement

In SQL, inner queries or nested queries are used most frequently with the SELECT statement. The syntax of Subquery with the SELECT statement is described in the following block:

```
SELECT Column_Name1, Column_Name2, ...., Column_NameN
FROM Table_Name WHERE Column_Name Comparison_Operator
( SELECT Column_Name1, Column_Name2, ...., Column_NameN
FROM Table_Name WHERE condition;
```

```
CREATE TABLE table1
(
Student_Id Int PRIMARY KEY,
First_Name VARCHAR (20),
Address VARCHAR (20),
Age Int NOT NULL,
Percentage Int NOT NULL,
Grade VARCHAR (10)
);


INSERT INTO table1 VALUES
(201, "Akash", "Delhi", 18, 89, "A2"),
(202, "Bhavesh", "Kanpur", 19, 93, "A1"),
(203, "Yash", "Delhi", 20, 89, "A2"),
(204, "Bhavana", "Delhi", 19, 78, NULL),
(205, "Yatin", "Lucknow", 20, 75, "B1"),
(206, "Ishika", "Ghaziabad", 19, 51, "C1"),
(207, "Vivek", "Goa", 20, 62, NULL);



SELECT AVG(Percentage) FROM table1;
SELECT * FROM table1 WHERE Percentage > ( SELECT
AVG(Percentage) FROM table1);
```

```
76.7142857142857
201|Akash|Delhi|18|89|A2
202|Bhavesh|Kanpur|19|93|A1
203|Yash|Delhi|20|89|A2
204|Bhavana|Delhi|19|78|

[Execution complete with exit code 0]
```

UPDATE table1 SET Age = Age + 5 WHERE Student_Id IN (SELECT Student_Id FROM table1 WHERE Address = "Delhi");

SELECT * FROM table1;

```
201|Akash|Delhi|23|89|A2
202|Bhavesh|Kanpur|19|93|A1
203|Yash|Delhi|25|89|A2
204|Bhavana|Delhi|24|78|
205|Yatin|Lucknow|20|75|B1
206|Ishika|Ghaziabad|19|51|C1
207|Vivek|Goa|20|62|

[Execution complete with exit code 0]
```

DELETE FROM table1 WHERE Student_Id IN (SELECT Student_Id FROM table1 WHERE Grade = 'C1') ;

SELECT * From table1;

```
201|Akash|Delhi|18|89|A2
202|Bhavesh|Kanpur|19|93|A1
203|Yash|Delhi|20|89|A2
204|Bhavana|Delhi|19|78|
205|Yatin|Lucknow|20|75|B1
207|Vivek|Goa|20|62|

[Execution complete with exit code 0]
```

SQL View

SQL provides the concept of VIEW, which hides the complexity of the data and restricts unnecessary access to the database. It permits the users to access only a particular column rather than the whole data of the table.

The **View** in the Structured Query Language is considered as the virtual table, which depends on the result-set of the predefined SQL statement.

Like the SQL tables, Views also store data in rows and columns, but the rows do not have any physical existence in the database.

Any database administrator and user can easily create the View by selecting the columns from one or more database tables. They can also delete and update the views according to their needs.

A view can store either all the records of the table or a particular record from the table using the WHERE clause.

Create a SQL View

You can easily create a View in Structured Query Language by using the CREATE VIEW statement. You can create the View from a single table or multiple tables.

Syntax to Create View from Single Table

```
CREATE VIEW View_Name AS
SELECT Column_Name1, Column_Name2, ....., Column_NameN
FROM Table_Name
WHERE condition;
```

In the syntax, View_Name is the name of View you want to create in SQL. The SELECT command specifies the rows and columns of the table, and the WHERE clause is optional, which is used to select the particular record from the table.

Syntax to Create View from Multiple Tables

You can create a View from multiple tables by including the tables in the SELECT statement.

```
REATE VIEW View_Name AS
SELECT Table_Name1.Column_Name1, Table_Name1.Column_Name2, Table_Name2.Colu
mn_Name2, ....., Table_NameN.Column_NameN
FROM Table_Name1, Table_Name2, ....., Table_NameN
WHERE condition;
```

```
CREATE TABLE table1
(
Student_Id Int PRIMARY KEY,
First_Name VARCHAR (20),
Address VARCHAR (20),
Age Int NOT NULL,
Percentage Int NOT NULL,
Grade VARCHAR (10)
);

INSERT INTO table1 VALUES
(201, "Akash", "Delhi", 18, 89, "A2"),
(202, "Bhavesh", "Kanpur", 19, 93, "A1"),
(203, "Yash", "Delhi", 20, 89, "A2"),
(204, "Bhavana", "Delhi", 19, 78, NULL),
(205, "Yatin", "Lucknow", 20, 75, "B1"),
(206, "Ishika", "Ghaziabad", 19, 51, "C1"),
(207, "Vivek", "Goa", 20, 62, NULL);

CREATE VIEW Student_View AS
SELECT Student_Id, First_Name, Grade
FROM table1
WHERE Percentage > 80;
Select * FROM Student_View;
```

```
201|Akash|A2
202|Bhavesh|A1
203|Yash|A2

[Execution complete with exit code 0]
```

Example to Create a View from Multiple tables

```
CREATE TABLE table1
(
Student_Id Int PRIMARY KEY,
First_Name VARCHAR (20),
Address VARCHAR (20),
Age Int NOT NULL,
Percentage Int NOT NULL,
Grade VARCHAR (10)
);



INSERT INTO table1 VALUES
(201, "Akash", "Delhi", 18, 89, "A2"),
(202, "Bhavesh", "Kanpur", 19, 93, "A1"),
(203, "Yash", "Delhi", 20, 89, "A2"),
(204, "Bhavana", "Delhi", 19, 78, NULL),
(205, "Yatin", "Lucknow", 20, 75, "B1"),
(206, "Ishika", "Ghaziabad", 19, 51, "C1"),
(207, "Vivek", "Goa", 20, 62, NULL);
```

```
CREATE TABLE table2
(
    Id Int PRIMARY KEY,
Last_Name VARCHAR (20),
Hobby VARCHAR (20),
Attendance Int NOT NULL,
Batch VARCHAR (10)
);


INSERT INTO table2 VALUES
(201,"kumar","drawing",45,"red"),
(202,"gowd","singling",55,"blue"),
(203,"shekar","dancing",65,"yellow"),
(204,"vilehya","painting",45,"green"),
(205,"krishna","travelling",85,"red"),
(206,"koushal","gardening",75,"yellow"),
(207,"shankar","drawing",55,"blue");

CREATE VIEW Student_Teacher_View AS
SELECT table1.Student_Id, table1.First_Name, table2.Last_Name,
table2.Hobby
FROM table1,table2
```

WHERE table1.Student_Id=table2.Id;

Select * from Student_Teacher_View;

```
201|Akash|kumar|drawing
202|Bhavesh|gowd|singling
203|Yash|shekar|dancing
204|Bhavana|vilehya|painting
205|Yatin|krishna|travelling
206|Ishika|koushal|gardening
207|Vivek|shankar|drawing

[Execution complete with exit code 0]
```

Constraints in SQL

Constraints in SQL means we are applying certain conditions or restrictions on the database. This further means that before inserting data into the database, we are checking for some conditions. If the condition we have applied to the database holds true for the data which is to be inserted, then only the data will be inserted into the database tables.

Constraints in SQL can be categorized into two types:

1. **Column Level Constraint:**
   Column Level Constraint is used to apply a constraint on a single column.
2. **Table Level Constraint:**
   Table Level Constraint is used to apply a constraint on multiple columns.

Some of the real-life examples of constraints are as follows:

1. Every person has a unique email id. This is because while creating an email account for any user, the email providing services such as Gmail, Yahoo or any other email providing service will always check for the availability of the email id that the user wants for himself. If some other user already takes the email id that the user wants, then that id cannot be assigned to another user. This simply means that no two users can have the same email ids on the same email providing service. So, here the email id is the constraint on the database of email providing services.
2. Whenever we set a password for any system, there are certain constraints that are to be followed. These constraints may include the following:

- There must be one uppercase character in the password.
- Password must be of at least eight characters in length.
- Password must contain at least one special symbol.

Constraints available in SQL are:

1. NOT NULL
2. UNIQUE
3. PRIMARY KEY
4. FOREIGN KEY
5. CHECK
6. DEFAULT
7. CREATE INDEX

Now let us try to understand the different constraints available in SQL in more detail with the help of examples. We will use MySQL database for writing all the queries.

1. NOT NULL

- NULL means empty, i.e., the value is not available.
- Whenever a table's column is declared as NOT NULL, then the value for that column cannot be empty for any of the table's records.
- There must exist a value in the column to which the NOT NULL constraint is applied.

> ***NOTE: NULL does not mean zero. NULL means empty column, not even zero.***

Syntax to apply the NOT NULL constraint during table creation:

1. **CREATE TABLE** TableName (ColumnName1 datatype NOT NULL, ColumnNam e2 datatype,...., ColumnNameN datatype);

2. UNIQUE

- Duplicate values are not allowed in the columns to which the UNIQUE constraint is applied.
- The column with the unique constraint will always contain a unique value.

- This constraint can be applied to one or more than one column of a table, which means more than one unique constraint can exist on a single table.
- Using the UNIQUE constraint, you can also modify the already created tables.

Syntax to apply the UNIQUE constraint on a single column:

1. **CREATE TABLE** TableName (ColumnName1 datatype **UNIQUE**, ColumnName 2 datatype,...., ColumnNameN datatype);

3. PRIMARY KEY

- PRIMARY KEY Constraint is a combination of NOT NULL and Unique constraints.
- NOT NULL constraint and a UNIQUE constraint together forms a PRIMARY constraint.
- The column to which we have applied the primary constraint will always contain a unique value and will not allow null values.

Syntax of primary key constraint during table creation:

1. **CREATE TABLE** TableName (ColumnName1 datatype **PRIMARY KEY**, Column Name2 datatype,...., ColumnNameN datatype);

4. FOREIGN KEY

- A foreign key is used for referential integrity.
- When we have two tables, and one table takes reference from another table, i.e., the same column is present in both the tables and that column acts as a primary key in one table. That particular column will act as a foreign key in another table.

Syntax to apply a foreign key constraint during table creation:

1. **CREATE TABLE** tablename(ColumnName1 Datatype(**SIZE**) **PRIMARY KEY**, Col umnNameN Datatype(**SIZE**), **FOREIGN KEY**(ColumnName) **REFERENCES** PARENT _TABLE_NAME(Primary_Key_ColumnName));

5.CHECK

- Whenever a check constraint is applied to the table's column, and the user wants to insert the value in it, then the value will first be checked for certain conditions before inserting the value into that column.
- **For example:** if we have an age column in a table, then the user will insert any value of his choice. The user will also enter even a negative value or any other invalid value. But, if the user has applied check constraint on the age column with the condition age greater than 18. Then in such cases, even if a user tries to insert an invalid value such as zero or any other value less than 18, then the age column will not accept that value and will not allow the user to insert it due to the application of check constraint on the age column.

Syntax to apply check constraint on a single column:

1. **CREATE TABLE** TableName (ColumnName1 datatype **CHECK** (ColumnName1 Condition), ColumnName2 datatype,...., ColumnNameN datatype);

6.DEFAULT

Whenever a default constraint is applied to the table's column, and the user has not specified the value to be inserted in it, then the default value which was specified while applying the default constraint will be inserted into that particular column.

Syntax to apply default constraint during table creation:

1. **CREATE TABLE** TableName (ColumnName1 datatype **DEFAULT** Value, Colum nName2 datatype,...., ColumnNameN datatype);

7.CREATE INDEX

CREATE INDEX constraint is used to create an index on the table. Indexes are not visible to the user, but they help the user to speed up the searching speed or retrieval of data from the database.

Syntax to create an index on single column:

1. **CREATE INDEX** IndexName **ON** TableName (ColumnName 1);

Pattern Matching in SQL

- LIKE clause is used to perform the pattern matching task in SQL.
- A WHERE clause is generally preceded by a LIKE clause in an SQL query.
- LIKE clause searches for a match between the patterns in a query with the pattern in the values present in an SQL table. If the match is successful, then that particular value will be retrieved from the SQL table.
- LIKE clause can work with strings and numbers.

The LIKE clause uses the following symbols known as wildcard operators in SQL to perform this pattern-matching task in SQL.

1. To represent zero, one or more than one character, % (percentage) is used.
2. To represent a single character _ (underscore) is used.

(A) Using LIKE clause with % (percentage)

Example 1: Write a query to display employee details in which name starts with 'Pr'.

Query:

```
mysql> SELECT * FROM employee_details WHERE Name LIKE 'Pr%';
```

We have used the SELECT query with the WHERE clause applied on the Name column followed by the LIKE clause. We have specified the expression value as 'Pr' in the LIKE clause, followed by the wildcard operator percent (%). So, according to the query, all the records that have names starting with the string 'Pr' followed by any other character will be considered a part of the output.

Example 2: Write a query to display employee details in which 'ya' is a substring in a name.

Query:

```
mysql> SELECT * FROM employee_details WHERE Name LIKE '%ya%';
```

We have used the SELECT query with the WHERE clause applied on the Name column followed by the LIKE clause. In the LIKE clause, we have specified the expression value as 'ya' preceded and followed by the wildcard operator percent(%). So, according to

the query, all the records which have names containing 'ya' as the substring, followed and preceded by any other character, will be considered as a part of the output.

Example 3: Write a query to display employee details in which city name ends with 'i'.

Query:

```
mysql> SELECT * FROM employee_details WHERE City LIKE '%i';
```

We have used the SELECT query with the WHERE clause applied on the City column followed by the LIKE clause. In the LIKE clause, we have specified the expression value as 'i' preceded by the wildcard operator percent (%). So according to the query, all the records with city names ending with 'i' preceded by any other character will be considered a part of the output.

(B) Using LIKE clause with _ (underscore)

Example 1:

Write a query to display employee details in which city name starts with 'Na', ends with 'ik', and contains any single character between 'Na' and 'ik'.

Query:

```
mysql> SELECT * FROM employee_details WHERE City LIKE 'Na_ik';
```

We have used the SELECT query with the WHERE clause applied on the City column followed by the LIKE clause. In the LIKE clause, we have specified the expression value as 'Na' followed by the wildcard operator underscore (_) with the string 'ik'. So, according to the query, all the records that have city names starting with 'Na' followed by any single character and ending with 'ik' will be considered as a part of the output.

(C) Using LIKE clause with % and _ operator in a single query

Example 1: Write a query to display employee details in which employee name contains 'a' at fifth position.

Query:

```
mysql> SELECT * FROM employee_details WHERE Name LIKE '____a%';
```

We have used the SELECT query with the WHERE clause applied on the Name column followed by the LIKE clause. In the LIKE clause, we have specified the expression value as the wildcard operator underscore (_) five times, followed by 'a' with another wildcard

operator percentage (%). So according to the query, all the records which has names starting with any five alphabets followed by an alphabet 'a' and ending with any other alphabet will be considered as a part of the output.

Example 2:

Write a query to display employee details in which salary contains a substring '00' starting from the 5th position.

Query:

```
mysql> SELECT * FROM employee_details WHERE Salary LIKE '__00%';
```

We have used the SELECT query with the WHERE clause applied on the Salary column followed by the LIKE clause. In the LIKE clause, we have specified the expression value as the wildcard operator underscore (_) twice, followed by double zero ending with another wildcard operator percentage (%). So, according to the query, all the records that have salary starting with any two digits followed by a double zero and ending with any number will be considered as a part of the output.

D) Using LIKE clause with NOT operator

Example 1:

Write a query to display employee details in which employee name is not like 'Priya'.

Query:

```
mysql> SELECT * FROM employee_details WHERE Name NOT LIKE 'Priya%';
```

We have used the SELECT query with the WHERE clause applied on the Name column followed by the LIKE clause preceded by NOT. In the LIKE clause, we have specified the expression value as 'Priya' followed by the wildcard operator percentage (%). So, according to the query, all the records that have names not starting with 'Priya' followed by any alphabet will be considered as a part of the output.

SQL Date Functions

In this tutorial, we will learn about some of the important date functions which are in-built in SQL.

Date functions in SQL:

1. NOW()
2. CURDATE()
3. CURTIME()
4. DATE()
5. EXTRACT()
6. DATE_ADD()
7. DATE_SUB()
8. DATEDIFF()
9. DATE_FORMAT()

NOW():

NOW () function in SQL will give the current system's date and time.

Syntax:

1. **SELECT** NOW ();

SELECT NOW() as now_time;

```
now_time
2024-03-28 05:29:05

[Execution complete with exit code 0]
```

CURDATE()

CURDATE () function in SQL will give the current system's date.

Syntax:

1. **SELECT** CURDATE ();

SELECT CURDATE() as date;

```
date
2024-03-28

[Execution complete with exit code 0]
```

CURTIME()

CURTIME () function in SQL will give the current system time.

Syntax:

1. **SELECT** CURTIME ();

SELECT CURTIME() as time;

```
time
05:36:00

[Execution complete with exit code 0]
```

DATE()

Using the DATE () function in SQL, you can specifically extract the date from the DATETIME datatype column.

Syntax:

1. **SELECT DATE** (DateTimeValue);

SELECT DATE ("2021-10-24 18:28:44") AS SHOW_DATE;

```
SHOW_DATE
2021-10-24

[Execution complete with exit code 0]
```

EXTRACT()

Using the EXTRACT() function in SQL, we can extract a specific part of date and time according to our requirements: day, month, year, day, hour, minute, etc.

Syntax:

1. **SELECT** EXTRACT (PART **FROM DATE** / **TIME**);

SELECT EXTRACT(YEAR FROM "2021-10-24") AS SHOW_YEAR;

```
SHOW_YEAR
2021

[Execution complete with exit code 0]
```

SELECT EXTRACT(MONTH FROM "2021-10-24") AS SHOW_MONTH;

```
SHOW_MONTH
10

[Execution complete with exit code 0]
```

SELECT EXTRACT(DAY FROM "2021-10-24") AS SHOW_DAY;

```
SHOW_DAY
24

[Execution complete with exit code 0]
```

SELECT EXTRACT(HOUR FROM "19:10:43") AS SHOW_HOUR;

```
SHOW_HOUR
19

[Execution complete with exit code 0]
```

SELECT EXTRACT(MINUTE FROM "19:10:43") AS SHOW_MINUTE;

```
SHOW_MINUTE
10

[Execution complete with exit code 0]
```

SELECT EXTRACT(SECOND FROM "19:10:43") AS SHOW_SECOND;

```
SHOW_SECOND
43

[Execution complete with exit code 0]
```

DATE_ADD()

Using the DATE_ADD () function in SQL, we can add a specific time interval to the given date.

Syntax:

1. **SELECT** DATE_ADD (**DATE**, INTERVAL VALUE Unit_to_be_added);

SELECT DATE_ADD("2021-10-24", INTERVAL 15 DAY) AS NEW_DATE;

```
NEW_DATE
2021-11-08

[Execution complete with exit code 0]
```

SELECT DATE_ADD("2021-10-24", INTERVAL 5 MONTH) AS NEW_DATE;

```
NEW_DATE
2022-03-24

[Execution complete with exit code 0]
```

DATE_SUB()

Using the DATE_SUB () function in SQL, we can remove a specific time interval from the given date.

Syntax:

1. **SELECT** DATE_SUB (**DATE**, INTERVAL VALUE Unit_to_be_subtracted);

SELECT DATE_SUB("2021-10-24", INTERVAL 25 YEAR) AS NEW_DATE;

```
NEW_DATE
1996-10-24

[Execution complete with exit code 0]
```

DATEDIFF()

Using the DATEDIFF() function in SQL will give us the number of days that fall between the two given dates.

Syntax:

1. **SELECT** DATEDIFF(Date1, Date2);

SELECT DATEDIFF("2021-10-24", "2021-10-09") AS NEW_DATE;

```
NEW_DATE
15

[Execution complete with exit code 0]
```

DATE_FORMAT()

Using the DATE_FORMAT () function in SQL, we can display the date or time-related information in a well-formatted manner.

Syntax of using a DATE_FORMAT function on a table's column:

1. **SELECT** DATE_FORMAT (ColumnName, Expression) **FROM Table Name**;

SELECT DATE_FORMAT("2021-10-24", "%W %D %M %Y") AS Formatted_Date;

```
Formatted_Date
Sunday 24th October 2021

[Execution complete with exit code 0]
```

SQL CASE

The **CASE** is a statement that operates if-then-else type of logical queries. This statement returns the value when the specified condition evaluates to True. When no condition evaluates to True, it returns the value of the ELSE part.

When there is no ELSE part and no condition evaluates to True, it returns a NULL value.

In Structured Query Language, CASE statement is used in SELECT, INSERT, and DELETE statements with the following three clauses:

1. WHERE Clause
2. ORDER BY Clause
3. GROUP BY Clause

This statement in SQL is always followed by at least one pair of WHEN and THEN statements and always finished with the END keyword.

The CASE statement is of two types in relational databases:

1. Simple CASE statement
2. Searched CASE statement

Syntax of CASE statement in SQL

```
CASE <expression>

WHEN condition_1 THEN statement_1
WHEN condition_2 THEN statement_2 .....
WHEN condition_N THEN statement_N

ELSE result

END;
```

Here, the CASE statement evaluates each condition one by one.

If the expression matches the condition of the first WHEN clause, it skips all the further WHEN and THEN conditions and returns the statement_1 in the result.

If the expression does not match the first WHEN condition, it compares with the seconds WHEN condition. This process of matching will continue until the expression is matched with any WHEN condition.

If no condition is matched with the expression, the control automatically goes to the ELSE part and returns its result. In the CASE syntax, the ELSE part is optional.

In Syntax, CASE and END are the most important keywords which show the beginning and closing of the CASE statement

```
CREATE TABLE table1
(
Student_Id Int PRIMARY KEY,
First_Name VARCHAR (20),
Address VARCHAR (20),
Age Int NOT NULL,
Percentage Int NOT NULL,
Grade VARCHAR (10)
);


INSERT INTO table1 VALUES
(201, "Akash", "Delhi", 18, 89, "A2"),
(202, "Bhavesh", "Kanpur", 19, 93, "A1"),
(203, "Yash", "Delhi", 20, 59, "A2"),
(204, "Bhavana", "Delhi", 19, 38, NULL),
(205, "Yatin", "Lucknow", 20, 75, "B1"),
(206, "Ishika", "Ghaziabad", 19, 41, "C1"),
(207, "Vivek", "Goa", 20, 62, NULL);
```

```
SELECT Student_Id, First_Name, Percentage,
CASE
WHEN Percentage >= 90 THEN 'Outstanding'
WHEN Percentage >= 80 AND Percentage < 90 THEN 'Excellent'
WHEN Percentage >= 70 AND Percentage < 80 THEN 'Good'
WHEN Percentage >= 60 AND Percentage < 70 THEN 'Average'
WHEN Percentage >= 50 AND Percentage < 60 THEN 'Bad'
WHEN Percentage < 50 THEN 'Failed'
END AS Stu_Remarks
FROM table1;
```

```
Student_Id      First_Name      Percentage      Stu_Remarks
201     Akash   89      Excellent
202     Bhavesh 93      Outstanding
203     Yash    59      Bad
204     Bhavana 38      Failed
205     Yatin   75      Good
206     Ishika  41      Failed
207     Vivek   62      Average

[Execution complete with exit code 0]
```

How to use CHECK in SQL

In this article, you will learn how to use the CHECK keyword to the column in SQL queries.

What is CHECK in SQL?

CHECK is a SQL constraint that allows database users to enter only those values which fulfill the specified condition. If any column is defined as a CHECK constraint, then that column holds only TRUE values.

The following syntax adds the CHECK constraint to the column at the time of table creation:

```
1. CREATE TABLE Table_Name
2. (
3. Column_Name_1 DataType (character_size of the column_1) CHECK (Boolean_
   Expression),
4. Column_Name_2 DataType (character_size of the column_2) CHECK (Boolean_Express
   ion),
5. Column_Name_3 DataType (character_size of the column_3) CHECK (Boolean_
   Expression),
6. ........,
7. Column_Name_N DataType (character_size of the column_N) CHECK (Boolean
   _Expression)
8. ) ;
```

We can easily use the CHECK constraint to one or more columns in one SQL table.

The following syntax adds the CHECK constraint to the column when the table already exists:

```
1. ALTER TABLE Table_Name ALTER COLUMN Column_Name datatype CHECK;
```

If you want to use the CHECK constraint at the time of table creation, you have to follow the steps given below:

1. Create the new database
2. Create a new table with the CHECK constraint
3. Insert the values
4. View the records of Table

```
CREATE TABLE table1
(
Student_Id Int PRIMARY KEY,
First_Name VARCHAR (20),
Address VARCHAR (20),
Age Int NOT NULL CHECK (Age >=18),
Percentage Int NOT NULL,
Grade VARCHAR (10)
);

INSERT INTO table1 VALUES
(201, "Akash", "Delhi", 18, 89, "A2"),
(202, "Bhavesh", "Kanpur", 19, 93, "A1"),
(203, "Yash", "Delhi", 20, 59, "A2"),
(204, "Bhavana", "Delhi", 19, 38, NULL),
(205, "Yatin", "Lucknow", 20, 75, "B1"),
(206, "Ishika", "Ghaziabad", 19, 41, "C1"),
(207, "Vivek", "Goa", 10, 62, NULL);

SELECT Student_Id, First_Name, Percentage
FROM table1;
```

```
ERROR 3819 (HY000) at line 12: Check constraint 'table1_chk_1' is violated.

[Execution complete with exit code 1]
```

Add CHECK constraint to Existing table

Any database user can easily add a CHECK constraint to the existing table by using the ADD keyword in the SQL ALTER query.

Syntax to specify CHECK constraint to the Existing table:

1. **ALTER TABLE** Table_Name **ADD CONSTRAINT** Constraint_Name **CHECK**(Boolean_Expression);

ALTER TABLE table1 ADD CONSTRAINT chk_age CHECK(Age >= 18);

SELECT Student_Id, First_Name, Percentage

FROM table1;

```
ERROR 3819 (HY000) at line 21: Check constraint 'chk_age' is violated.

[Execution complete with exit code 1]
```

ALTER TABLE table1 DROP CONSTRAINT chk_age;

DEFAULT in SQL

In this SQL article, you will learn how to use DEFAULT on the columns of the table in Structured Query Language.

What is a DEFAULT constraint?

The DEFAULT is a constraint in SQL which allows users to fill a column with the default or fixed value.

If no value is specified to the column at the time of insertion, then the default value will be added automatically to that column.

The following syntax adds the DEFAULT constraint to the column at the time of table creation:

```
CREATE TABLE Table_Name
(
Column_Name_1 DataType (character_size of the column_1) DEFAULT Value,
Column_Name_2 DataType (character_size of the column_2) DEFAULT Value,
Column_Name_3 DataType (character_size of the column_3) DEFAULT Value,
.....,
Column_Name_N DataType (character_size of the column_N) DEFAULT Value,
) ;
```

In the SQL DEFAULT syntax, we have to define the value with the DEFAULT constraint. The database users can easily specify the DEFAULT constraint to one or more columns in one SQL table.

The following syntax adds the DEFAULT constraint to the column when the table already exists:

1. **ALTER TABLE** Table_Name **ALTER COLUMN** Column_Name datatype **DEFAULT**;

If you want to use the DEFAULT constraint at the time of table creation, you have to follow the steps given below:

1. Create the new database
2. Create a new table and add DEFAULT
3. Insert the records
4. View the table's data

```
CREATE TABLE Doctor_Info
(
Doctor_ID INT NOT NULL PRIMARY KEY,
Doctor_Name VARCHAR(100),
Doctor_Specialist VARCHAR(80) DEFAULT 'Heart',
Doctor_Gender Varchar(20) DEFAULT 'Male',
```

```
Doctor_Country Varchar(80) DEFAULT 'Russia'

) ;


INSERT INTO Doctor_Info  VALUES

 (1035, 'Jones',DEFAULT,DEFAULT,'U. K.'),

 (1015, 'Moris',DEFAULT,DEFAULT,DEFAULT),

 (1003, 'Harry','Fever',DEFAULT,'U. K.'),

 (1044,'Bella',DEFAULT,'Female', 'U. K.'),

 (1025, 'Moria',DEFAULT,DEFAULT,DEFAULT);


SELECT * FROM Doctor_Info;
```

```
Doctor_ID       Doctor_Name     Doctor_Specialist       Doctor_Gender
        Doctor_Country
1003    Harry   Fever   Male    U. K.
1015    Moris   Heart   Male    Russia
1025    Moria   Heart   Male    Russia
1035    Jones   Heart   Male    U. K.
1044    Bella   Heart   Female  U. K.

[Execution complete with exit code 0]
```

DELETIONS OF COLUMNS AND ROWS

How to Delete Column from Table in SQL

This article describes how to delete one or more columns from the table in Structured Query Language.

The ALTER command in SQL deletes the single and multiple columns from the SQL table. It allows the database users to modify the structure of the created table in the database.

The syntax for deleting a Single Column from the table is given below:

1. **ALTER TABLE** Table_Name **DROP** Column_Name;

The syntax for deleting Multiple Columns from the table is given below:

1. **ALTER TABLE** Table_Name **DROP** Column_Name1, Column_Name2, ..., Colu mn_NameN;

We have to use the DROP keyword in the ALTER command for deleting one or more columns from the table.

If you want to delete the column from the table, you have to follow the following steps one by one in the given order:

1. Create a Database in your system.
2. Create a Table in the database and Insert the data into the table.
3. Show the table before column deletion.
4. Delete a single column from the table.
5. Show the table after deletion.

```
CREATE TABLE Doctor_Info

(

Doctor_ID INT NOT NULL PRIMARY KEY,

Doctor_Name VARCHAR(100),

Doctor_Specialist VARCHAR(80) DEFAULT 'Heart',

Doctor_Gender Varchar(20) DEFAULT 'Male',

Doctor_Country Varchar(80) DEFAULT 'Russia'

) ;


INSERT INTO Doctor_Info  VALUES
```

(1035, 'Jones',DEFAULT,DEFAULT,'U. K.'),

(1015, 'Moris',DEFAULT,DEFAULT,DEFAULT),

(1003, 'Harry','Fever',DEFAULT,'U. K.'),

(1044,'Bella',DEFAULT,'Female', 'U. K.'),

(1025, 'Moria',DEFAULT,DEFAULT,DEFAULT);

ALTER TABLE Doctor_Info DROP Doctor_Gender;

SELECT * FROM Doctor_Info;

```
Doctor_ID       Doctor_Name     Doctor_Specialist       Doctor_Country
1003    Harry   Fever   U. K.
1015    Moris   Heart   Russia
1025    Moria   Heart   Russia
1035    Jones   Heart   U. K.
1044    Bella   Heart   U. K.

[Execution complete with exit code 0]
```

ALTER TABLE Doctor_Info

DROP Doctor_Gender,

DROP Doctor_Specialist;

SELECT * FROM Doctor_Info;

```
Doctor_ID       Doctor_Name     Doctor_Country
1003    Harry   U. K.
1015    Moris   Russia
1025    Moria   Russia
1035    Jones   U. K.
1044    Bella   U. K.

[Execution complete with exit code 0]
```

How to Delete one row in SQL

Here, you will learn how to delete one row or record from a table in Structured Query Language.

We can easily delete one record using the SQL DELETE statement. This statement also removes all the existing rows from the database tables. It also helps in removing the data from the SQL views.

Once a row has been deleted from the table, that row cannot be recovered.

The SQL syntax for deleting a specific row is given below:

1. **DELETE FROM** Table_Name **WHERE** condition;

In this syntax, the WHERE clause specifies that record which you want to remove from the table. If you run the DELETE command without WHERE clause, the query will remove all the rows from the SQL table.

If you want to remove the records permanently from the table, you have to follow the following steps one by one in the given order:

1. Create a Database.
2. Create a Table and Insert the data into the table.
3. Show the table before deletion.
4. Delete one record from the table.
5. Show the table after deletion.

DELETE FROM Doctor_Info WHERE Doctor_ID = 1015;

SELECT * FROM Doctor_Info;

```
Doctor_ID       Doctor_Name     Doctor_Specialist       Doctor_Gender
        Doctor_Country
1003    Harry   Fever   Male    U. K.
1025    Moria   Heart   Male    Russia
1035    Jones   Heart   Male    U. K.
1044    Bella   Heart   Female  U. K.

[Execution complete with exit code 0]
```

Delete Multiple Records from the table

DELETE FROM Doctor_Info WHERE Doctor_ID <= 1015;

SELECT * FROM Doctor_Info;

```
Doctor_ID       Doctor_Name     Doctor_Specialist       Doctor_Gender
        Doctor_Country
1025    Moria   Heart   Male    Russia
```

```
1035    Jones   Heart   Male    U. K.
1044    Bella   Heart   Female  U. K.

[Execution complete with exit code 0]
```

ADDITION OF ROWS AND COLUMNS

How to Add Column in the Table in SQL

In this section, we shall learn how to add a column in the table in Structured Query Language.

The ALTER command in SQL allows the database users to add one or more columns in the SQL table. It allows the database users to modify the structure of the existing table in the database.

The syntax for adding a Single Column from the table is given below:

1. **ALTER** TABLETable_Name **ADD** Column_Name datatype;

The syntax for deleting Multiple Columns from the table is given below:

1. **ALTER** TABLETable_Name **ADD** Column_Name1 Column1_datatype, Column_Name2 Column2_datatype, ..., Column_NameN Columnn_datatype;

We have to use the ADD keyword in the ALTER command for adding one or more columns in the table.

If you want to add a column in the table, you have to follow the following steps one by one in a given order:

1. Create a Database.
2. Create a Table in the database.
3. View the Table structure before column addition.
4. Add a single column to the table.
5. View the Table structure after column addition.

ALTER TABLE Doctor_Info ADD Average INT NOT NULL DEFAULT 0;

SELECT * FROM Doctor_Info;

```
Doctor_ID       Doctor_Name     Doctor_Specialist       Doctor_Gender
        Doctor_Country  Average
1003    Harry   Fever   Male    U. K.   0
1015    Moris   Heart   Male    Russia  0
1025    Moria   Heart   Male    Russia  0
1035    Jones   Heart   Male    U. K.   0
1044    Bella   Heart   Female  U. K.   0

[Execution complete with exit code 0]
```

ALTER TABLE Doctor_Info ADD Engine_Number Varchar(50),

ADD Car_Number Varchar(45) ;

SELECT * FROM Doctor_Info;

```
Doctor_ID       Doctor_Name     Doctor_Specialist       Doctor_Gender
        Doctor_Country  Engine_Number   Car_Number
1003    Harry   Fever   Male    U. K.   NULL    NULL
1015    Moris   Heart   Male    Russia  NULL    NULL
1025    Moria   Heart   Male    Russia  NULL    NULL
1035    Jones   Heart   Male    U. K.   NULL    NULL
1044    Bella   Heart   Female  U. K.   NULL    NULL

[Execution complete with exit code 0]
```

How to Add Row in the Table in SQL

INSERT INTO Doctor_Info VALUES

(1035, 'Jones',DEFAULT,DEFAULT,'U. K.'),

(1015, 'Moris',DEFAULT,DEFAULT,DEFAULT),

(1003, 'Harry','Fever',DEFAULT,'U. K.'),

(1044,'Bella',DEFAULT,'Female', 'U. K.'),

(1025, 'Moria',DEFAULT,DEFAULT,DEFAULT);

How to Add Foreign Key in SQL

In this article, we will learn how to add a Foreign Key to the column in the table of our SQL database.

The **FOREIGN KEY** in SQL is used to join the record of two tables in the database. The column defined as the FOREIGN KEY in one table must be the PRIMARY KEY in another table in the same database.

We can easily add foreign key to the column in the following two ways:

1. Add foreign key using Create table statement
2. Add foreign key using Alter Table statement

If you want to add a FOREIGN KEY to the column into the SQL table, you have to follow the below steps in the given sequence:

1. Create the database in the system.
2. Create two tables in the same database.
3. View Table structure before foreign key addition.
4. Add a foreign key to the table.
5. View the table structure.

```
CREATE TABLE Cars_Price_Details

(

Model INT NOT NULL PRIMARY KEY,

Cars_Price INT NOT NULL

);


CREATE TABLE Cars_Details

(

    Car_Number INT AUTO_INCREMENT PRIMARY KEY,
```

Model INT, -- Define Model column without the FOREIGN KEY constraint here

Cars_Name VARCHAR(20),

Color VARCHAR(20) UNIQUE,

Price INT NOT NULL,

FOREIGN KEY (Model) REFERENCES Cars_Price_Details(Model) -- Define FOREIGN KEY constraint separately

);

DESC Cars_Details;

```
Field   Type    Null    Key     Default Extra
Car_Number      int     NO      PRI     NULL    auto_increment
Model   int     YES     MUL     NULL
Cars_Name       varchar(20)     YES             NULL
Color   varchar(20)     YES     UNI     NULL
Price   int     NO              NULL

[Execution complete with exit code 0]
```

Add Foreign key to the Existing Table

If you want to add the foreign key to the existing table, you have to use the following ALTER syntax in SQL:

1. **ALTER TABLE** Table_Name1 **ADD CONSTRAINT** ForeignKey_Name **FOREIGN KEY** (Column_Name) **REFERENCES** Table_Name2 (Column_Name);

The following query adds a FOREIGN KEY on the Model column when the Cars_Details table already exists in the database system:

1. **ALTER TABLE** Cars_Details **ADD CONSTRAINT** FK_Cars_Details **FOREIGN KEY** (Model) **REFERENCES** Cars_Price_Details (Car_Model);

Delete foreign key from the table

If you want to delete the foreign key from the column of the table, you have to use the following **ALTER** syntax in SQL:

1. **ALTER TABLE** Table_Name **DROP FOREIGN KEY** Foreign_Key_Name;

The following query deletes the created FOREIGN KEY from the Model column of the Cars_Details table:

1. **ALTER TABLE** Cars **DROP FOREIGN KEY** FK_Cars_Details;

How to Add a Primary Key in SQL

In this article, we will learn how to add Primary Key to the column in the table of our SQL database.

The **PRIMARY KEY** is used to retrieve each record of the SQL table. The field defined as the PRIMARY KEY must contain different and NOT NULL values. You can easily add a primary key to the column in the following two ways:

1. Add Primary key using Create table statement
2. Add primary key using Alter Table statement

If you want to add primary key to a column in the table, you have to follow the below steps in the given sequence:

1. Create a database in the system.
2. Create the table in the SQL database.
3. View the table structure.
4. Add the primary key to column in the table.
5. View the table structure.

```
CREATE TABLE Cars_Details
(
    Car_Number INT AUTO_INCREMENT PRIMARY KEY,
    Model INT,
```

```
    Cars_Name VARCHAR(20),
    Color VARCHAR(20) UNIQUE,
    Price INT NOT NULL
);
```

```
DESC Cars_Details;
```

```
Field   Type    Null    Key     Default Extra
Car_Number      int     NO      PRI     NULL    auto_increment
Model   int     YES             NULL
Cars_Name       varchar(20)     YES             NULL
Color   varchar(20)     YES     UNI     NULL
Price   int     NO              NULL

[Execution complete with exit code 0]
```

Delete Primary key from the table

If you want to delete the Primary key from the column of the table, then you have to use the following **ALTER** syntax in SQL:

1. **ALTER TABLE** Table_Name **DROP PRIMARY KEY**;

The following query deletes the PRIMARY KEY from the Model column of the Cars table:

1. **ALTER TABLE** Cars **DROP PRIMARY KEY**;

Add Primary key to the Existing Table

If you want to add a primary key in the existing table, you have to use the following ALTER syntax in SQL:

1. **ALTER TABLE** Table_Name **ADD CONSTRAINT** Constraint_Name **PRIMARY KEY** (Column_Name);

The following query adds a PRIMARY KEY constraint on the Color column when the Cars table already exists in the database system:

1. **ALTER TABLE** Cars **ADD CONSTRAINT** clr_prmrykey **PRIMARY KEY** (Color);

SQL Stored Procedure

A stored procedure in Structured Query Language is a group of logical statements stored in the database for performing a particular task.

It is a subprogram consisting of a name, a list of parameters, and Transact-SQL statements.

Any user can store the stored procedure as a named object in the SQL database and can call it by using triggers, other procedures, and other programming applications such as Java, PHP, R, C#, Python, etc.

SQL database creates an execution plan and stores it in the cache memory when the stored procedure is called for the first time. The plan is reused by SQL Server, which executes the stored procedure quickly with reliable performance.

Types of Stored Procedure

Following are the two types of Stored Procedures in SQL:

- User-defined Stored Procedures
- System Stored Procedures

User-defined Stored Procedures

User-defined Stored Procedures are created by the database developers and administrators and stored in the current database.

This type of stored procedure provides one or more SQL statements for retrieving, updating, and deleting values from database tables.

User-Defined stored procedure is further categorized into the following two types:

1. T-SQL Stored Procedure
2. CLR Stored Procedure

T-SQL Stored Procedure

The Transact-SQL procedure accepts the parameters and returns them. This stored procedure manages INSERT, UPDATE, and DELETE statements with or without parameters and gives the row data in the output.

CLR Stored Procedure

CLR stored procedure is that stored procedure which is created by the combination of CLR (Common Language Runtime) and another stored procedure written in a CLR-based language like C# and VB.NET.

CLR procedures are the objects of .Net Framework, which execute in the memory of the SQL database server.

System Stored Procedures

SQL database server creates and executes the system stored procedures for administrative activities. SQL database server does not allow developers to interfere with system stored procedures.

Syntax of Stored Procedure in SQL

The following syntax is used to create the simple stored procedure in Structured Query Language:

```
CREATE PROCEDURE Procedure_Name
AS
/*    SQL Statements */
GO;
```

The following syntax is used to execute the stored procedure in Structured Query Language:

1. **EXEC** Procedure_Name ;

```
CREATE TABLE Student_Stored_Procedure

(

Student_ID INT NOT NULL,

Student_Name varchar(100),

Student_Course varchar(50),

Student_Age INT,

Student_Marks INT

);
```

```
INSERT INTO Student_Stored_Procedure VALUES
(101, 'Anuj', 'B.tech', 20, 88),
(102, 'Raman', 'MCA', 24, 98),
(104, 'Shyam', 'BBA', 19, 92),
(107, 'Vikash', 'B.tech', 20, 78),
(111, 'Monu', 'MBA', 21, 65),
(114, 'Jones', 'B.tech', 18, 93),
(121, 'Parul', 'BCA', 20, 97),
(123, 'Divya', 'B.tech', 21, 89),
(128, 'Hemant', 'MBA', 23, 90),
(130, 'Nidhi', 'BBA', 20, 88),
(132, 'Priya', 'MBA', 22, 99),
(138, 'Mohit', 'MCA', 21, 92);

CREATE PROCEDURE Show_All_Students
AS
BEGIN
    SELECT * FROM Student_Stored_Procedure;
END;
GO;
```

EXEC Show_All_Students;

Trigger in SQL

In this article, you will learn about the trigger and its implementation with examples.

A **Trigger** in Structured Query Language is a set of procedural statements which are executed automatically when there is any response to certain events on the particular table in the database. Triggers are used to protect the data integrity in the database.

In SQL, this concept is the same as the trigger in real life. For example, when we pull the gun trigger, the bullet is fired.

To understand the concept of trigger in SQL, let's take the below hypothetical situation:

Suppose Rishabh is the human resource manager in a multinational company. When the record of a new employee is entered into the database, he has to send the 'Congrats' message to each new employee. If there are four or five employees, Rishabh can do it manually, but if the number of new Employees is more than the thousand, then in such condition, he has to use the trigger in the database.

Thus, now Rishabh has to create the trigger in the table, which will automatically send a 'Congrats' message to the new employees once their record is inserted into the database.

The trigger is always executed with the specific table in the database. If we remove the table, all the triggers associated with that table are also deleted automatically.

In Structured Query Language, triggers are called only either before or after the below events:

1. **INSERT Event:** This event is called when the new row is entered in the table.
2. **UPDATE Event:** This event is called when the existing record is changed or modified in the table.
3. **DELETE Event:** This event is called when the existing record is removed from the table.

Types of Triggers in SQL

Following are the six types of triggers in SQL:

1. **AFTER INSERT Trigger**
   This trigger is invoked after the insertion of data in the table.
2. **AFTER UPDATE Trigger**
   This trigger is invoked in SQL after the modification of the data in the table.
3. **AFTER DELETE Trigger**
   This trigger is invoked after deleting the data from the table.
4. **BEFORE INSERT Trigger**
   This trigger is invoked before the inserting the record in the table.
5. **BEFORE UPDATE Trigger**
   This trigger is invoked before the updating the record in the table.
6. **BEFORE DELETE Trigger**
   This trigger is invoked before deleting the record from the table.

Syntax of Trigger in SQL

```
CREATE TRIGGER Trigger_Name
[ BEFORE | AFTER ]  [ Insert | Update | Delete]
ON [Table_Name]
[ FOR EACH ROW | FOR EACH COLUMN ]
AS
Set of SQL Statement
```

In the trigger syntax, firstly, we have to define the name of the trigger after the CREATE TRIGGER keyword. After that, we have to define the BEFORE or AFTER keyword with anyone event.

Then, we define the name of that table on which trigger is to occur.

After the table name, we have to define the row-level or statement-level trigger.

And, at last, we have to write the SQL statements which perform actions on the occurring of event

```
CREATE TABLE Student_Trigger
(
Student_RollNo INT NOT NULL PRIMARY KEY,
```

```
Student_FirstName Varchar (100),

Student_EnglishMarks INT,

Student_PhysicsMarks INT,

Student_ChemistryMarks INT,

Student_MathsMarks INT,

Student_TotalMarks INT,

Student_Percentage INT );


CREATE TRIGGER Student_Table_Marks

BEFORE INSERT

ON

Student_Trigger

FOR EACH ROW

SET new.Student_TotalMarks = new.Student_EnglishMarks +
new.Student_PhysicsMarks + new.Student_ChemistryMarks +
new.Student_MathsMarks,

new.Student_Percentage = ( new.Student_TotalMarks / 400) * 100;


INSERT INTO Student_Trigger (Student_RollNo,
Student_FirstName, Student_EnglishMarks, Student_PhysicsMarks,
Student_ChemistryMarks, Student_MathsMarks,
Student_TotalMarks, Student_Percentage) VALUES

    ( 201, Sorya, 88, 75, 69, 92, 0, 0);
```

SELECT * FROM Student_Trigger;

SQL ANY Keyword

What is Any in SQL?

The ANY is an operator in SQL. This operator compares the given value to each subquery value and returns those values that satisfy the condition.

ANY operator is mainly used in the HAVING or WHERE clause with the INSERT, UPDATE, DELETE and UPDATE SQL statements.

It always evaluates to TRUE if at least one subquery value matches according to the given condition.

The syntax for using ANY operator in Structured Query Language:

1. **SELECT** Column_Name_1, Column_Name_2, Column_Name_3, ..., Column_Name_N **FROM** Table_Name **WHERE** Column_Name Comparison_Operator ANY (**SELECT** Column_Name **FROM** Table_Name **WHERE** [condition]);

In the syntax, the ANY operator is followed by the SQL comparison operator, which helps compare the column value with the subquery.

Following are the SQL comparison operators used with the ANY operator in queries:

1. Equal operator (=)

The equal comparison operator with ANY operator evaluates to TRUE when the column's value is equal to any value of the subquery.

Syntax:

1. Column_Name = ANY (subquery);

2. Not Equal operator (!=)

This comparison operator with ANY operator evaluates to TRUE when the column's value is not equal to any value of the subquery.

Syntax:

```
1. Column_Name != ANY (subquery);
```

3. Greater Than operator (>)

This comparison operator with ANY operator evaluates to TRUE when the column's value is greater than the smallest value of the subquery.

Syntax:

```
1. Column_Name > ANY (subquery);
```

4. Less Than operator (<)

This comparison operator with ANY operator evaluates to TRUE when the column's value is less than the largest value of the subquery.

Syntax:

```
1. Column_Name < ANY (subquery);
```

5. Greater Than Equal To operator (>=)

This comparison operator with ANY operator evaluates to TRUE when the column's value is greater than or equals the smallest value of the subquery.

Syntax:

```
1. Column_Name >= ANY (subquery);
```

6. Less Than Equals To operator (<=)

This comparison operator with ANY operator evaluates to TRUE when the column's value is less than and equals to the largest value of the subquery.

Syntax:

```
1. Column_Name <= ANY (subquery);
```

If you want to use the SQL ANY operator in the tables for performing operations, you have to follow the given steps in the same manner:

1. Create a database in the system.

2. Create two new tables.
3. Insert the data in both tables
4. View the Inserted data of both tables
5. Use the ANY operator to view the data in different ways.

```
CREATE TABLE Teacher_Info
(
Teacher_ID INT NOT NULL PRIMARY KEY,
Teacher_First_Name VARCHAR (100),
Teacher_Last_Name VARCHAR (100),
Teacher_Dept_Id INT NOT NULL,
Teacher_Address Varchar (80),
Teacher_City Varchar (80),
Teacher_Salary INT
);

CREATE TABLE Department_Info
(
Dept_Id INT NOT NULL,
Dept_Name Varchar(100),
Head_Id INT
);
```

```
INSERT INTO Teacher_Info (Teacher_ID, Teacher_First_Name,
Teacher_Last_Name, Teacher_Dept_Id, Teacher_Address,
Teacher_City, Teacher_Salary) VALUES

    (1001, 'Arush', 'Sharma', 4001, '22 street', 'New Delhi', 20000),

    (1002, 'Bulbul', 'Roy', 4002, '120 street', 'New Delhi', 38000),

    (1004, 'Saurabh', 'Sharma', 4001, '221 street', 'Mumbai', 45000),

    (1005, 'Shivani', 'Singhania', 4001, '501 street', 'Kolkata', 42000),

    (1006, 'Avinash', 'Sharma', 4002, '12 street', 'Delhi', 28000),

    (1007, 'Shyam', 'Besas', 4003, '202 street', 'Lucknow', 35000);


INSERT INTO Department_Info (Dept_ID, Dept_Name, Head_Id)
VALUES (4001, 'Arun', 1005);

INSERT INTO Department_Info (Dept_ID, Dept_Name, Head_Id)
VALUES (4002, 'Zayant', 1009);

INSERT INTO Department_Info (Dept_ID, Dept_Name, Head_Id)
VALUES (4003, 'Manish', 1007);


SELECT * FROM Teacher_Info WHERE Teacher_Id = ANY
(SELECT Head_Id from Department_Info);
```

```
Teacher_ID      Teacher_First_Name      Teacher_Last_Name
        Teacher_Dept_Id Teacher_Address Teacher_City    Teacher_Salary
1005    Shivani Singhania       4001    501 street      Kolkata 42000
1007    Shyam   Besas   4003    202 street      Lucknow 35000

[Execution complete with exit code 0]
```

SELECT AVG (Teacher_Salary) from Teacher_Info GROUP BY Teacher_Dept_Id ;

SELECT * FROM Teacher_Info WHERE Teacher_Salary < ANY (SELECT AVG (Teacher_Salary) from Teacher_Info GROUP BY Teacher_Dept_Id);

```
AVG ( Teacher_Salary )
35666.6667
33000.0000
35000.0000
Teacher_ID      Teacher_First_Name      Teacher_Last_Name
        Teacher_Dept_Id Teacher_Address Teacher_City    Teacher_Salary
1001    Arush   Sharma  4001    22 street       New Delhi       20000
1006    Avinash Sharma  4002    12 street       Delhi   28000
1007    Shyam   Besas   4003    202 street      Lucknow 35000

[Execution complete with exit code 0]
```

SQL ANY Keyword

What is Any in SQL?

The ALL is an operator in SQL. This operator compares the single record to every record of the list returned by the sub-query. This operator is always used with the SQL comparison operator, which is followed by the inner query.

The syntax for using ALL operator in Structured Query Language:

1. **SELECT** Column_Name_1, Column_Name_2, Column_Name_3, ..., Column_Name_N **FROM** Table_Name **WHERE** Column_Name Comparison_Operator ALL (**SELECT** Column_Name **FROM** Table_Name **WHERE** [condition]);

In the ALL syntax, the ALL operator is followed by the SQL comparison operator, which helps compare the column value with the sub-query.

We can use the following comparison operators with the ALL operator in the statements of SQL:

1. Equal operator (=)

This comparison operator with ALL operator evaluates to TRUE when the value of specified column is equal to any value in the returned list.

Syntax:

```
1. Column_Name = ALL (subquery);
```

2. Not Equal operator (!=)

This comparison operator with the ALL operator evaluates to TRUE when the value of the specified column is not equal to any value of the returned list.

Syntax:

```
1. Column_Name != ALL (subquery);
```

3. Greater Than operator (>)

This comparison operator with the ALL operator evaluates to TRUE when the value of the specified column is greater than the biggest value of the returned list.

Syntax:

```
1. Column_Name > ALL (subquery);
```

4. Less Than operator (<)

This comparison operator with the ALL operator evaluates to TRUE when the value of the specified column is less than the smallest value of the returned list.

Syntax:

```
1. Column_Name < ALL (subquery);
```

5. Greater Than Equal To operator (>=)

This comparison operator with the ALL operator evaluates to TRUE when the value of the specified column is greater than or equals the biggest value of the returned list.

Syntax:

1. Column_Name >= ALL (subquery);

6. Less Than Equals To operator (<=)

This comparison operator with the ALL operator evaluates to TRUE when the value of the specified column is less than or equals the smallest value of the returned list.

Syntax:

1. Column_Name <= ALL (subquery);

If you want to perform the 'ALL' operator in the tables of SQL, then you have to follow the below points one by one in the given manner:

1. Create a database in the system.
2. Create two new tables.
3. Insert the data in both tables
4. View the Inserted data of both tables
5. Use the ALL operator to view the data in different ways.

```
CREATE TABLE Department_Info

(

Dept_Id INT NOT NULL,

Dept_Name Varchar(100),

Head_Id INT

);
```

```
INSERT INTO Teacher_Info (Teacher_ID, Teacher_First_Name,
Teacher_Last_Name, Teacher_Dept_Id, Teacher_Address,
Teacher_City, Teacher_Salary) VALUES
```

(1001, 'Arush', 'Sharma', 4001, '22 street', 'New Delhi', 20000),

(1002, 'Bulbul', 'Roy', 4002, '120 street', 'New Delhi', 38000),

(1004, 'Saurabh', 'Sharma', 4001, '221 street', 'Mumbai', 45000),

(1005, 'Shivani', 'Singhania', 4001, '501 street', 'Kolkata', 42000),

(1006, 'Avinash', 'Sharma', 4002, '12 street', 'Delhi', 28000),

(1007, 'Shyam', 'Besas', 4003, '202 street', 'Lucknow', 30000);

INSERT INTO Department_Info (Dept_ID, Dept_Name, Head_Id) VALUES (4001, 'Arun', 1005);

INSERT INTO Department_Info (Dept_ID, Dept_Name, Head_Id) VALUES (4002, 'Zayant', 1009);

INSERT INTO Department_Info (Dept_ID, Dept_Name, Head_Id) VALUES (4003, 'Manish', 1007);

SELECT AVG (Teacher_Salary) from Teacher_Info GROUP BY Teacher_Dept_Id;

SELECT * FROM Teacher_Info WHERE Teacher_Salary < ALL (SELECT AVG (Teacher_Salary) from Teacher_Info GROUP BY Teacher_Dept_Id);

```
AVG ( Teacher_Salary )
35666.6667
33000.0000
30000.0000
Teacher_ID      Teacher_First_Name      Teacher_Last_Name
        Teacher_Dept_Id Teacher_Address Teacher_City    Teacher_Salary
1001    Arush   Sharma  4001    22 street       New Delhi       20000
1006    Avinash Sharma  4002    12 street       Delhi   28000
```

```
[Execution complete with exit code 0]
```

MAKE_SET Function in SQL

The MAKE_SET string function in Structured Query Language returns the value of the given bit from the set of multiple values.

Syntax of MAKE_SET String Function

In SQL, we can use the MAKE_SET function with the columns of the table, strings, and characters.

Syntax 1:

```
SELECT MAKE_SET(bits, Column_Name1, column_Name2, Column_Name3,.....
 Column_NameN) AS Alias_Name FROM Table_Name;
```

In this syntax, we used the MAKE_SET function with the existing table of SQL. Here, we have to define the name and columns of that table on which we want to perform MAKE_SET function.

Syntax 2:

```
SELECT MAKE_SET(bits, "String1", "String2", "String3", ..... "StringN") AS Alias_
Name;
```

In this syntax, we used the MAKE_SET function with list of strings.

Syntax 3:

```
SELECT MAKE_SET(bits, "Character_1", "character_2", "Character_3", ......., "Char
acter_N") AS Alias_Name;
```

In this syntax, we used the MAKE_SET function with list of characters.

Examples of MAKE_SET String function

Example 1: The following query uses the MAKE_SET function with the list of strings:

```
SELECT MAKE_SET( 1, ''H'', ''I'', ''A'', ''P'', ''Q'', ''S'', ''R'', ''T'', ''V'', ''M'' ) AS Value
_at1st_bit;
```

```
CREATE TABLE Teacher_Info
(
Teacher_ID INT NOT NULL PRIMARY KEY,
Teacher_First_Name VARCHAR (100),
Teacher_Last_Name VARCHAR (100),
Teacher_Dept_Id INT NOT NULL,
Teacher_Address Varchar (80),
Teacher_City Varchar (80),
Teacher_Salary INT
);
```

```
INSERT INTO Teacher_Info (Teacher_ID, Teacher_First_Name,
Teacher_Last_Name, Teacher_Dept_Id, Teacher_Address,
Teacher_City, Teacher_Salary) VALUES
    (1001, 'Arush', 'Sharma', 4001, '22 street', 'New Delhi', 20000),
    (1002, 'Bulbul', 'Roy', 4002, '120 street', 'New Delhi', 38000),
    (1004, 'Saurabh', 'Sharma', 4001, '221 street', 'Mumbai', 45000),
    (1005, 'Shivani', 'Singhania', 4001, '501 street', 'Kolkata', 42000),
    (1006, 'Avinash', 'Sharma', 4002, '12 street', 'Delhi', 28000),
    (1007, 'Shyam', 'Besas', 4003, '202 street', 'Lucknow', 30000);
```

SELECT Teacher_First_Name, Teacher_Last_Name,Teacher_Address,MAKE_SET(2, Teacher_First_Name, Teacher_Last_Name,Teacher_Address) AS Value_at_2bit FROM Teacher_Info;

```
Teacher_First_Name      Teacher_Last_Name       Teacher_Address Value_at_2bit
Arush   Sharma  22 street       Sharma
Bulbul  Roy     120 street      Roy
Saurabh Sharma  221 street      Sharma
Shivani Singhania       501 street      Singhania
Avinash Sharma  12 street       Sharma
Shyam   Besas   202 street      Besas

[Execution complete with exit code 0]
```

FIELD Function in SQL

The FIELD string function in Structured Query Language returns the position of the given string from the list of strings. If the given string is not found in the list of string, then FIELD function returns 0 in result.

Syntax of FIELD String Function

In SQL, we can use the FIELD function with the columns of the table, strings, and characters.

Syntax 1:

1. **SELECT** FIELD(Searched_value, Column_Name1, column_Name2, Column_Name3,……. Column_NameN) **AS** Alias_Name **FROM** Table_Name;

In this syntax, we used the FIELD function with existing table of SQL. Here, we have to define the name and columns of that table on which we want to perform FIELD function.

Syntax 2:

1. **SELECT** FIELD("Searched_String", "String1", "String2", "String3", …… "StringN") **AS** Alias_Name;

In this syntax, we used the FIELD function with list of strings.

Syntax 3:

```
SELECT FIELD("Searched_character", "Character_1", "character_2", "Character_3", ....., "Character_N") AS Alias_Name;
```

In this syntax, we used the FIELD function with list of characters.

Examples of FIELD String function

Example 1: The following query uses the FIELD function with the list of strings:

```
SELECT FIELD( ''S'', ''H'', ''I'', ''A'', ''P'', ''Q'', ''S'', ''R'', ''T'', ''V'', ''M'' ) AS Position_of_S;
```

SELECT Teacher_First_Name, Teacher_Last_Name,Teacher_Address,FIELD('Sharma', Teacher_First_Name, Teacher_Last_Name,Teacher_Address) AS Value_at_2bit FROM Teacher_Info;

```
Teacher_First_Name      Teacher_Last_Name       Teacher_Address Value_at_2bit
Arush   Sharma  22 street       2
Bulbul  Roy     120 street      0
Saurabh Sharma  221 street      2
Shivani Singhania       501 street      0
Avinash Sharma  12 street       2
Shyam   Besas   202 street      0

[Execution complete with exit code 0]
```

CHAR Function in SQL

The CHAR string function shows the ASCII value of the integer passed in the function. This function takes only one argument. If we pass the integer value which exceeds the given range then it shows NULL value.

Syntax of CHAR String Function

In SQL, we can use the CHAR function with the columns of the table, strings, and characters.

Syntax 1:

1. **SELECT CHAR**(Integer_Column_Name) **AS** Alias_Name **FROM** Table_Name;

In this syntax, we used the CHAR function with existing table of SQL. Here, we have to define the name and integer column of that table on which we want to perform CHAR function.

Syntax 2:

1. **SELECT CHAR**(Integer_Value) **AS** Alias_Name;

In this syntax, we used the CHAR function with the integer value.

Examples of CHAR String function

Example 1: The following SELECT query shows the ASCII value of 20:

1. **SELECT CHAR**(20)**AS** ASCII_of_20;

SELECT Teacher_First_Name, Teacher_Last_Name,Teacher_ID,CHAR(Teacher_ID) AS char_value FROM Teacher_Info;

```
Teacher_First_Name      Teacher_Last_Name       Teacher_ID      char_value
Arush   Sharma  21
Bulbul  Roy     62      >
Saurabh Sharma  64      @
Shivani Singhania       75      K
Avinash Sharma  76      L
Shyam   Besas   77      M

[Execution complete with exit code 0]
```

ELT Function in SQL

The ELT string function in Structured Query Language returns the string from the list of string according to the given index number. This function returns NULL, if no string is found at the given index position.

Syntax of ELT String Function

In SQL, we can use the ELT function with the columns of the table, strings, and characters.

Syntax 1:

```
SELECT ELT(Index_Value, Column_Name1, column_Name2, Column_Name3,.... Column_NameN) AS Alias_Name FROM Table_Name;
```

In this syntax, we used the ELT function with existing table of SQL. Here, we have to define the name and columns of that table on which we want to perform ELT function.

Syntax 2:

```
SELECT ELT(Index_Value, "String1", "String2", "String3", ..... "StringN") AS Alias_Name;
```

In this syntax, we used the ELT function with list of strings.

Syntax 3:

```
SELECT ELT(Index_Value, "Character_1", "character_2", "Character_3", ......., "Character_N") AS Alias_Name;
```

In this syntax, we used the ELT function with list of characters.

Examples of ELT String function

Example 1: The following query uses the ELT function with the list of strings:

```
SELECT ELT( 5, ''H'', ''I'', ''A'', ''P'', ''Q'', ''S'', ''R'', ''T'', ''V'', ''M'' ) AS String_at_1st Position;
```

```
SELECT Teacher_First_Name, Teacher_Last_Name,Teacher_ID,ELT(1, Teacher_First_Name, Teacher_Last_Name,Teacher_Address) AS Value_at_2bit FROM Teacher_Info;
```

```
Teacher_First_Name      Teacher_Last_Name       Teacher_ID      Value_at_2bit
Arush   Sharma  1001    Arush
Bulbul  Roy     1002    Bulbul
Saurabh Sharma  1004    Saurabh
Shivani Singhania       1005    Shivani
Avinash Sharma  1006    Avinash
Shyam   Besas   1007    Shyam
```

```
[Execution complete with exit code 0]
```

COUNT Function in SQL

The COUNT is an aggregate function in SQL which returns the total number of rows from the table.

Syntax of COUNT Function

In the Structured Query Language, we use the COUNT function with the columns of the table as shown in the following block:

1. **SELECT** COUNT(column_Name) **AS** Alias_Name **FROM** Table_Name;

In this syntax, we have to define the name and column of that table on which we want to perform the COUNT function.

SELECT COUNT(Teacher_Dept_Id) AS countuuu FROM Teacher_Info;

```
countuuu
6

[Execution complete with exit code 0]
```

SQL ISNULL

This tutorial will teach us to implement the IS NULL condition and IsNull function in SQL and SQL servers.

IS NULL Condition in SQL

The user can use the IS NULL condition to verify whether a data value is NULL. If the value is NULL, the condition will return TRUE, or it will return False. The user can implement the IS NULL condition in SQL's SELECT, INSERT, DELETE, or UPDATE clauses.

Syntax of IS NULL Condition

The syntax to implement the IS NULL condition is as follows:

Expr IS NULL

Parameters Or Arguments in IS NULL Condition

Expr: It specifies the value or statement checked for the NULL value.

```
CREATE TABLE Teacher_Info
(
Teacher_ID INT ,
Teacher_First_Name VARCHAR (100),
Teacher_Last_Name VARCHAR (100),
Teacher_Dept_Id INT ,
Teacher_marks INT,
Teacher_Address Varchar (80),
Teacher_City Varchar (80),
Teacher_Salary FLOAT
);


INSERT INTO Teacher_Info (Teacher_ID,
Teacher_First_Name, Teacher_Last_Name,
Teacher_Dept_Id,Teacher_marks, Teacher_Address,
Teacher_City, Teacher_Salary) VALUES
    (1001, 'Arush', 'Sharma', 4001,0, '22 street', 'New Delhi',
NULL),
```

(NULL, 'Bulbul', 'Roy', 4002,45, '120 street', 'New Delhi', 3800.560),

(1004, 'Saurabh', 'Sharma', NULL,-78, '221 street', 'Mumbai', 4500.760),

(NULL, 'Shivani', 'Singhania', 4001,-56, '501 street', 'Kolkata', 4200.650),

(NULL, 'Avinash', 'Sharma', NULL,0, '12 street', 'Delhi', 3800.560),

(1007, 'Shyam', 'Besas', NULL,54, '202 street', 'Lucknow', 3000.340);

select Teacher_ID,Teacher_First_Name from Teacher_Info where Teacher_Dept_Id IS NULL;

```
Teacher_ID      Teacher_First_Name
1004    Saurabh
NULL    Avinash
1007    Shyam

[Execution complete with exit code 0]
```

SQL MATH FUNCTIONS

POWER Function in SQL

The POWER is a mathematical function in SQL which returns the value of a number raised to the power of another number. In the power function, we have to pass the two number as the argument in which one number acts as exponent and other acts as the base.

Syntax of POWER String Function

```
1. SELECT POWER(Number1, Number2) AS Alias_Name;
```

In this POWER function, following are the two arguments:

1. **Number1:** It acts as the base in the power function
2. **Number2:** And, it acts as the exponent.

In Structured Query Language, we can also use the POWER function with the integer columns of the table as shown in the following block:

```
1. SELECT POWER(column_Name1, column_Name2) AS Alias_Name FROM Table
   _Name;
```

```
CREATE TABLE Teacher_Info

(

Teacher_ID INT NOT NULL PRIMARY KEY,

Teacher_First_Name VARCHAR (100),

Teacher_Last_Name VARCHAR (100),

Teacher_Dept_Id INT NOT NULL,

Teacher_Address Varchar (80),

Teacher_City Varchar (80),

Teacher_Salary INT

);
```

```
INSERT INTO Teacher_Info (Teacher_ID, Teacher_First_Name,
Teacher_Last_Name, Teacher_Dept_Id, Teacher_Address,
Teacher_City, Teacher_Salary) VALUES

    (1001, 'Arush', 'Sharma', 4001, '22 street', 'New Delhi', 20000),

    (1002, 'Bulbul', 'Roy', 4002, '120 street', 'New Delhi', 38000),

    (1004, 'Saurabh', 'Sharma', 4001, '221 street', 'Mumbai', 45000),

    (1005, 'Shivani', 'Singhania', 4001, '501 street', 'Kolkata', 42000),

    (1006, 'Avinash', 'Sharma', 4002, '12 street', 'Delhi', 28000),

    (1007, 'Shyam', 'Besas', 4003, '202 street', 'Lucknow', 30000);
```

```
SELECT
Teacher_Dept_Id,Teacher_First_Name,POWER(Teacher_ID,3) AS
expo FROM Teacher_Info;
```

```
Teacher_Dept_Id Teacher_First_Name      expo
4001    Arush   1003003001
4002    Bulbul  1006012008
4001    Saurabh 1012048064
4001    Shivani 1015075125
4002    Avinash 1018108216
4003    Shyam   1021147343

[Execution complete with exit code 0]
```

ROUND Function in SQL

The SQL ROUND function rounds the specified number till the particular decimal places.

Syntax of ROUND String Function

1. **SELECT** ROUND(Number, Decimal_places, Operation) **AS** Alias_Name;

In this ROUND function, following are the three arguments:

1. **Number:** It is that decimal number which is to be rounded. Decia
2. **Decimal_places:** It can be positive or negative integer which shows the number of decimal places to be round.
3. **Operation:** It is optional.

In Structured Query Language, we can also use the ROUND function with the integer columns of the table as shown in the following block:

1. **SELECT** ROUND(Column_Name, Decimal_places, Operation) **AS** Alias_Name **FR OM** Table_Name;

In this syntax, we used the ROUND function with existing table of SQL. Here, we have to define the name and columns of that table on which we want to perform the ROUND function.

```
CREATE TABLE Teacher_Info

(

Teacher_ID INT NOT NULL PRIMARY KEY,

Teacher_First_Name VARCHAR (100),

Teacher_Last_Name VARCHAR (100),

Teacher_Dept_Id INT NOT NULL,

Teacher_Address Varchar (80),

Teacher_City Varchar (80),

Teacher_Salary INT
```

);

INSERT INTO Teacher_Info (Teacher_ID, Teacher_First_Name, Teacher_Last_Name, Teacher_Dept_Id, Teacher_Address, Teacher_City, Teacher_Salary) VALUES

(1001, 'Arush', 'Sharma', 4001, '22 street', 'New Delhi', 2000.9080),

(1002, 'Bulbul', 'Roy', 4002, '120 street', 'New Delhi', 3800.560),

(1004, 'Saurabh', 'Sharma', 4001, '221 street', 'Mumbai', 4500.760),

(1005, 'Shivani', 'Singhania', 4001, '501 street', 'Kolkata', 4200.650),

(1006, 'Avinash', 'Sharma', 4002, '12 street', 'Delhi', 2800.780),

(1007, 'Shyam', 'Besas', 4003, '202 street', 'Lucknow', 3000.340);

SELECT Teacher_Dept_Id,Teacher_First_Name,ROUND(Teacher_Salary) AS salary FROM Teacher_Info;

```
Teacher_Dept_Id Teacher_First_Name      salary
4001    Arush   2001
4002    Bulbul  3801
4001    Saurabh 4501
4001    Shivani 4201
4002    Avinash 2801
4003    Shyam   3000

[Execution complete with exit code 0]
```

SUM Function in SQL

The SUM is an aggregate function in SQL which returns the sum of integer column of the table.

Syntax of SUM Function

In the Structured Query Language, we use the SUM function with the columns of the table as shown in the following block:

1. **SELECT** SUM(column_Name) **AS** Alias_Name **FROM** Table_Name;

In this syntax, we have to define the name and column of that table on which we want to perform the SUM function.

SUM Function with WHERE clause

We can also use the WHERE clause with the SUM function which adds the values of filtered rows.

The syntax for using the SUM function with the WHERE clause is as follows:

1. **SELECT** SUM(column_Name) **AS** Alias_Name **FROM** Table_Name **WHERE** Condition;

SUM Function with DISTINCT clause

We can also use the DISTINCT clause with the SUM function which adds the distinct values of the column from the table.

The syntax for using the SUM function with the DISTINCT clause is as follows:

1. **SELECT** SUM(**DISTINCT** (column_Name)) **AS** Alias_Name **FROM** Table_Name **WHERE** Condition;

SUM Function with GROUP BY clause

We can also use the GROUP BY clause with the SUM function which adds the values which exist in the same group.

The syntax for using the SUM function with the GROUP BY clause is as follows:

1. **SELECT** Column_Name1, SUM(column_Name)) **AS** Alias_Name **FROM** Table_Name **GROUP BY Column;**

SELECT SUM(Teacher_Salary) AS salary FROM Teacher_Info;

```
salary
20305

[Execution complete with exit code 0]
```

SELECT SUM(Teacher_Salary) AS salary FROM Teacher_Info WHERE Teacher_Last_Name ='Sharma';

```
salary
9303

[Execution complete with exit code 0]
```

CREATE TABLE Teacher_Info

(

Teacher_ID INT NOT NULL PRIMARY KEY,

Teacher_First_Name VARCHAR (100),

Teacher_Last_Name VARCHAR (100),

Teacher_Dept_Id INT NOT NULL,

Teacher_Address Varchar (80),

Teacher_City Varchar (80),

Teacher_Salary INT

);

INSERT INTO Teacher_Info (Teacher_ID, Teacher_First_Name, Teacher_Last_Name, Teacher_Dept_Id, Teacher_Address, Teacher_City, Teacher_Salary) VALUES

(1001, 'Arush', 'Sharma', 4001, '22 street', 'New Delhi', 2000.9080),

(1002, 'Bulbul', 'Roy', 4002, '120 street', 'New Delhi', 3800.560),

(1004, 'Saurabh', 'Sharma', 4001, '221 street', 'Mumbai', 4500.760),

(1005, 'Shivani', 'Singhania', 4001, '501 street', 'Kolkata', 4200.650),

(1006, 'Avinash', 'Sharma', 4002, '12 street', 'Delhi', 3800.560),

(1007, 'Shyam', 'Besas', 4003, '202 street', 'Lucknow', 3000.340);

SELECT SUM(DISTINCT(Teacher_Salary)) AS salary FROM Teacher_Info ;

```
salary
17504

[Execution complete with exit code 0]
```

SELECT Teacher_Last_Name,SUM(Teacher_Salary) AS salary FROM Teacher_Info GROUP BY Teacher_Last_Name ;

```
Teacher_Last_Name       salary
Sharma  10303
Roy     3801
Singhania       4201
Besas   3000

[Execution complete with exit code 0]
```

AVG Function in SQL

The AVG is an aggregate function in SQL which returns the average of values of the integer column of the table.

Syntax of AVG Function

In the Structured Query Language, we use the AVG function with the columns of the table as shown in the following block:

1. **SELECT** AVG(column_Name) **AS** Alias_Name **FROM** Table_Name;

In this syntax, we have to define the name and column of that table on which we want to perform the AVG function.

AVG Function with WHERE clause

We can also use the WHERE clause with the AVG function which adds the values of filtered rows.

The syntax for using the AVG function with the WHERE clause is as follows:

1. **SELECT** AVG(column_Name) **AS** Alias_Name **FROM** Table_Name **WHERE** Condition;

AVG Function with DISTINCT clause

We can also use the DISTINCT clause with the AVG function which adds the distinct values of the column from the table.

The syntax for using the AVG function with the DISTINCT clause is as follows:

1. **SELECT** AVG(**DISTINCT** (column_Name)) **AS** Alias_Name **FROM** Table_Name **WHERE** Condition;

```
CREATE TABLE Teacher_Info
(
Teacher_ID INT NOT NULL PRIMARY KEY,
Teacher_First_Name VARCHAR (100),
Teacher_Last_Name VARCHAR (100),
Teacher_Dept_Id INT NOT NULL,
Teacher_Address Varchar (80),
Teacher_City Varchar (80),
Teacher_Salary INT
);
```

INSERT INTO Teacher_Info (Teacher_ID, Teacher_First_Name, Teacher_Last_Name, Teacher_Dept_Id, Teacher_Address, Teacher_City, Teacher_Salary) VALUES

(1001, 'Arush', 'Sharma', 4001, '22 street', 'New Delhi', 2000.9080),

(1002, 'Bulbul', 'Roy', 4002, '120 street', 'New Delhi', 3800.560),

(1004, 'Saurabh', 'Sharma', 4001, '221 street', 'Mumbai', 4500.760),

(1005, 'Shivani', 'Singhania', 4001, '501 street', 'Kolkata', 4200.650),

(1006, 'Avinash', 'Sharma', 4002, '12 street', 'Delhi', 3800.560),

(1007, 'Shyam', 'Besas', 4003, '202 street', 'Lucknow', 3000.340);

SELECT Teacher_Last_Name,AVG(Teacher_Salary) AS salary FROM Teacher_Info GROUP BY Teacher_Last_Name ;

```
Teacher_Last_Name       salary
Sharma  3434.3333
Roy     3801.0000
Singhania       4201.0000
Besas   3000.0000

[Execution complete with exit code 0]
```

SELECT Teacher_Last_Name,AVG(Teacher_Salary) AS salary FROM Teacher_Info WHERE Teacher_Last_Name='Sharma' ;

```
Teacher_Last_Name       salary
Sharma  3434.3333

[Execution complete with exit code 0]
```

SELECT AVG(DISTINCT(Teacher_Salary)) AS salary FROM Teacher_Info ;

```
salary
3500.8000

[Execution complete with exit code 0]
```

SELECT AVG(Teacher_Salary) AS salary FROM Teacher_Info ;

```
salary
3550.8333

[Execution complete with exit code 0]
```

BIN Function in SQL

The BIN is a SQL function which converts the given decimal number to its binary equivalent. This function return NULL, if the NULL is passed in the function.

Syntax of BIN Function

1. **SELECT** BIN(Decimal_Number) **AS** Alias_Name;

In the BIN syntax, we have to pass that decimal number whose binary equivalent we want to find.

In the Structured Query Language, we can also use the BIN function with the column of the table as shown in the following block:

1. **SELECT** BIN(column_Name) **AS** Alias_Name **FROM** Table_Name;

SELECT Teacher_ID,Teacher_First_Name,BIN(Teacher_ID) AS b FROM Teacher_Info ;

```
Teacher_ID      Teacher_First_Name      b
1001    Arush   1111101001
1002    Bulbul  1111101010
1004    Saurabh 1111101100
1005    Shivani 1111101101
1006    Avinash 1111101110
1007    Shyam   1111101111

[Execution complete with exit code 0]
```

MAX Function in SQL

The MAX is an aggregate function in SQL which returns the maximum or largest value from the specified column of the table.

Syntax of MAX Function

In the Structured Query Language, we use the MAX function with the columns of the table as shown in the following block:

1. **SELECT MAX**(column_Name) **AS** Alias_Name **FROM** Table_Name;

In this syntax, we have to define the name and column of that table on which we want to perform the MAX function.

MAX Function with WHERE clause

We can also use the WHERE clause with the MAX function which returns the largest value from the filtere rows.

The syntax for using the MAX function with the WHERE clause is as follows:

1. **SELECT MAX**(column_Name) **AS** Alias_Name **FROM** Table_Name **WHERE** Condition;

MAX Function with GROUP BY clause

We can also use the GROUP BY clause with the MAX function which returns the maximum value from the same group.

The syntax for using the MAX function with the GROUP BY clause is as follows:

1. **SELECT** Column_Name1, **MAX**(column_Name)) **AS** Alias_Name **FROM** Table_Name **GROUP BY Column;**

SELECT MAX(Teacher_ID) AS m FROM Teacher_Info ;

```
m
1007

[Execution complete with exit code 0]
```

SELECT Teacher_Last_Name,MAX(Teacher_ID) AS m FROM Teacher_Info GROUP BY Teacher_Last_Name;

```
Teacher_Last_Name       m
Sharma  1006
Roy     1002
Singhania       1005
Besas   1007

[Execution complete with exit code 0]
```

SELECT Teacher_Last_Name,MAX(Teacher_ID) AS m FROM Teacher_Info WHERE Teacher_Last_Name='Sharma';

```
Teacher_Last_Name       m
Sharma  1006

[Execution complete with exit code 0]
```

MIN Function in SQL

The MIN is an aggregate function in SQL which returns the minimum or smallest value from the specified column of the table.

Syntax of MIN Function

In the Structured Query Language, we use the MIN function with the columns of the table as shown in the following block:

1. **SELECT MIN**(column_Name) **AS** Alias_Name **FROM** Table_Name;

MIN Function with WHERE clause

We can also use the WHERE clause with the MIN function which returns the smallest value from the filtere rows.

The syntax for using the MIN function with the WHERE clause is as follows:

1. **SELECT MIN**(column_Name) **AS** Alias_Name **FROM** Table_Name **WHERE** Con dition;

MIN Function with GROUP BY clause

We can also use the GROUP BY clause with the MIN function which returns the minimum value from the same group.

The syntax for using the MIN function with the GROUP BY clause is as follows:

1. **SELECT** Column_Name1, **MIN**(column_Name)) **AS** Alias_Name **FROM** Table_Name **GROUP BY** column;

SELECT Teacher_Last_Name,MIN(Teacher_ID) AS m FROM Teacher_Info WHERE Teacher_Last_Name='Sharma';

```
Teacher_Last_Name       m
Sharma  1001

[Execution complete with exit code 0]
```

SELECT Teacher_Last_Name,MIN(Teacher_ID) AS m FROM Teacher_Info GROUP BY Teacher_Last_Name;

```
Teacher_Last_Name       m
Sharma  1001
Roy     1002
Singhania       1005
Besas   1007

[Execution complete with exit code 0]
```

SELECT MIN(Teacher_ID) AS m FROM Teacher_Info;

```
m
1001

[Execution complete with exit code 0]
```

MOD Function in SQL

The MOD is a string function in SQL which returns the remainder from the division of first number by second number.

Syntax of MOD Function

1. **SELECT** MOD(Number1, Number2) **AS** Alias_Name;

In the MOD syntax, Number1 is the dividend and Number2 is the divisor.

In the Structured Query Language, we can also use the MOD function with the columns of the table as shown in the following block:

1. **SELECT** MOD(Column_Name1, Column_Name2) **AS** Alias_Name **FROM** Table_Name;

SELECT Teacher_First_Name,MOD(Teacher_Dept_Id,Teacher_ID) AS modd FROM Teacher_Info;

```
Teacher_First_Name      modd
Arush   998
Bulbul  996
Saurabh 989
Shivani 986
Avinash 984
Shyam   982

[Execution complete with exit code 0]
```

OCT Function in SQL

The OCT is a SQL string function which converts the given decimal number to its octal equivalent.

Syntax of OCT Function

1. **SELECT** OCT(Decimal_Number) **AS** Alias_Name;

In the OCT syntax, we have to pass that decimal number whose octal equivalent we want to find.

In the Structured Query Language, we can also use the OCT function with the column of the table as shown in the following block:

1. **SELECT** OCT(column_Name) **AS** Alias_Name **FROM** Table_Name;

In this syntax, we have to define the name and column of that table on which we want to perform the OCT function.

SELECT Teacher_First_Name,OCT(Teacher_ID) AS num FROM Teacher_Info;

```
Teacher_First_Name      num
Arush   1751
Bulbul  1752
Saurabh 1754
Shivani 1755
Avinash 1756
Shyam   1757

[Execution complete with exit code 0]
```

SIGN Function in SQL

The SIGN string function in Structured Query Language returns the specified number with the positive or negative sign. If the number is greater than zero, the function returns 1 otherwise -1. If the number specified as zero, the function returns zero in result.

Syntax of SIGN String Function

Syntax1: This syntax uses the SIGN function with the column name of the SQL table:

1. **SELECT** SIGN(Column_Name) **AS** Alias_Name **FROM** Table_Name;

In the syntax, we have to specify the name of that column on which we have to use SIGN function

Syntax2: We can also use the SIGN function with any number:

1. **SELECT** SIGN(Number) **AS** Alias_Name;

Examples of SIGN String function

Example 1: The following SELECT query uses the SIGN function 0

1. **SELECT** SIGN(0)**AS** SIGN_zero;

Output:

SIGN_zero
0

Example 2: The following SELECT query shows the SIGN of the number

```
1. SELECT SIGN(+10) AS SIGN_positive;
```

Output:

SIGN_positive
1

Example 3: following SELECT query shows the SIGN of the negative number:

```
1. SELECT SIGN(-0.5) AS SIGN_negative;
```

Output:

SIGN_negative
-1

```
CREATE TABLE Teacher_Info
(
Teacher_ID INT NOT NULL PRIMARY KEY,
Teacher_First_Name VARCHAR (100),
Teacher_Last_Name VARCHAR (100),
Teacher_Dept_Id INT NOT NULL,
Teacher_marks INT,
Teacher_Address Varchar (80),
Teacher_City Varchar (80),
Teacher_Salary INT
);
```

INSERT INTO Teacher_Info (Teacher_ID, Teacher_First_Name, Teacher_Last_Name, Teacher_Dept_Id,Teacher_marks, Teacher_Address, Teacher_City, Teacher_Salary) VALUES

(1001, 'Arush', 'Sharma', 4001,0, '22 street', 'New Delhi', 2000.9080),

(1002, 'Bulbul', 'Roy', 4002,45, '120 street', 'New Delhi', 3800.560),

(1004, 'Saurabh', 'Sharma', 4001,-78, '221 street', 'Mumbai', 4500.760),

(1005, 'Shivani', 'Singhania', 4001,-56, '501 street', 'Kolkata', 4200.650),

(1006, 'Avinash', 'Sharma', 4002,0, '12 street', 'Delhi', 3800.560),

(1007, 'Shyam', 'Besas', 4003,54, '202 street', 'Lucknow', 3000.340);

SELECT Teacher_First_Name,Teacher_marks,SIGN(Teacher_marks) AS num FROM Teacher_Info;

```
Teacher_First_Name      Teacher_marks   num
Arush   0       0
Bulbul  45      1
Saurabh -78     -1
```

```
Shivani -56     -1
Avinash 0       0
Shyam   54      1

[Execution complete with exit code 0]
```

SQRT Function in SQL

The SQRT is a SQL function of mathematics which gives the square root of the given number. Suppose, the number is 25, then this function returns 5.

Syntax of SQRT Function

1. **SELECT** SQRT(Number) **AS** Alias_Name;

In the SQRT syntax, we have to pass that number whose square root we want to find.

In the Structured Query Language, we can also use the SQRT function with the fields of the table as shown in the following block:

1. **SELECT** SQRT(column_Name) **AS** Alias_Name **FROM** Table_Name;

SELECT Teacher_First_Name,Teacher_ID,SQRT(Teacher_ID) AS num FROM Teacher_Info;

```
Teacher_First_Name      Teacher_ID      num
Arush   1001    31.63858403911275
Bulbul  1002    31.654383582688826
Saurabh 1004    31.68595903550972
Shivani 1005    31.701734968294716
Avinash 1006    31.71750305430741
Shyam   1007    31.73326330524486

[Execution complete with exit code 0]
```

SQUARE Function in SQL

The SQUARE is a mathematical function in Structured Query Language which returns the square of any specified number. This function shows the result in floating value.

We can specify both positive or negative number in the SQUARE function, but it always returns positive value.

Syntax of SQUARE String Function

Syntax1: This syntax uses the SQUARE function with the column name of the SQL table:

```
SELECT SQUARE(Column_Name) AS Alias_Name FROM Table_Name;
```

In the syntax, we have to specify the name of that column on which we want to perform the SQUARE string function. We can also use multiple square functions in single query.

Syntax2: This syntax uses the SQUARE function with the number:

```
SELECT SQUARE(Number);
```

Examples of SQUARE String function

Example 1: The following SELECT query shows the square of 4:

```
SELECT SQUARE(4) AS SQUARE_of_4;
```

```
SELECT Teacher_First_Name,Teacher_marks,SQUARE(Teacher_marks) AS num FROM Teacher_Info;
```

```
ERROR 1370 (42000) at line 23: execute command denied to user
'mycompiler'@'localhost' for routine 'mycompiler.SQUARE'

[Execution complete with exit code 1]
```

ABS Function in SQL

The ABS is a mathematical function in the Structured Query Language which returns the absolute value of the particular number.

In simple words, this function finds the distance of a given number on the number line from zero.

This function accepts single numeric or any non-numeric data and returns the data type same as the argument type.

Syntax of ABS String Function

Syntax1: This syntax uses the ABS function with the column name of the SQL table:

1. **SELECT** ABS(Numeric_Column_Name) **AS** Alias_Name **FROM** Table_Name;

In this SELECT syntax, we have to specify the name of that numeric column on which we want to use ABS function.

Syntax2: We can also use the ABS function with the particular number as like the below syntax:

1. **SELECT** ABS(**Numeric** value) **AS** Alias_Name;

SELECT Teacher_First_Name,Teacher_marks,ABS(Teacher_marks) AS num FROM Teacher_Info;

```
Teacher_First_Name      Teacher_marks   num
Arush   0       0
Bulbul  45      45
Saurabh -78     78
Shivani -56     56
Avinash 0       0
Shyam   54      54

[Execution complete with exit code 0]
```

COS Function in SQL

The COS is a SQL function of mathematics which returns the cosine value of the specified number.

Syntax of COS Function

1. **SELECT** COS(Number) **AS** Alias_Name;

In the COS syntax, we have to pass that decimal number whose cosine value we want to return.

In the Structured Query Language, we can also use the COS function with the field of the table as shown in the following block:

1. **SELECT** COS(column_Name) **AS** Alias_Name **FROM** Table_Name;

In this syntax, we have to define the name and column of that table on which we want to perform the COS function.

SELECT Teacher_First_Name,Teacher_marks,COS(Teacher_marks) AS num FROM Teacher_Info;

```
Teacher_First_Name      Teacher_marks   num
Arush   0       1
Bulbul  45      0.5253219888177297
Saurabh -78     -0.8578030932449878
Shivani -56     0.8532201077225842
Avinash 0       1
Shyam   54      -0.8293098328631502

[Execution complete with exit code 0]
```

COT Function in SQL

The COT is mathematics function which returns the cotangent value of the given number in the Structured Query Language.

Syntax of COT Function

1. **SELECT** COT(Number) **AS** Alias_Name;

In the COT syntax, we have to pass that numeric number in the function whose cot value we want to calculate.

In the Structured Query Language, we can also use the COT function in the SELECT query with the table fields:

1. **SELECT** COT(Column_Name) **AS** Alias_Name **FROM** Table_Name;

In this SELECT query, we have to define the name and field of that table on which we want to perform the COT function.

SELECT Teacher_First_Name,Teacher_ID,COT(Teacher_ID) AS num FROM Teacher_Info;

```
Teacher_First_Name      Teacher_ID      num
Arush   1001    -0.42602656008158657
```

```
Bulbul  1002    -5.894255362759312
Saurabh 1004    -0.26725027818316094
Shivani 1005    -3.1255792428803546
Avinash 1006    1.2107620470937386
Shyam   1007    -0.12012740899160543

[Execution complete with exit code 0]
```

SIN Function in SQL

The SIN is a SQL function of mathematics which returns the sine value of the specified number.

Syntax of SIN Function

1. **SELECT** SIN(Number) **AS** Alias_Name;

In the SIN syntax, we have to pass that decimal number whose sine value we want to return.

In the Structured Query Language, we can also use the SIN function with the field of the table as shown in the following block:

1. **SELECT** SIN(column_Name) **AS** Alias_Name **FROM** Table_Name;

In this syntax, we have to define the name and column of that table on which we want to perform the SIN function

SELECT Teacher_First_Name,Teacher_ID,SIN(Teacher_ID) AS num FROM Teacher_Info;

```
Teacher_First_Name      Teacher_ID      num
Arush   1001    0.9199905975863218
Bulbul  1002    0.1672665419737925
Saurabh 1004    -0.9660944251371497
Shivani 1005    -0.30472449023565196
Avinash 1006    0.6368077356795173
Shyam   1007    0.992861866200276

[Execution complete with exit code 0]
```

ACOS Function in SQL

The ACOS() is a mathematics function which returns the arc cosine value of the given number in the Structured Query Language. We have to specify the number between -1 to 1, otherwise this function returns NULL value in output.

Syntax of ACOS Function

1. **SELECT** ACOS(Number) **AS** Alias_Name;

In this SELECT syntax, we have to pass that numeric number in the function whose arc cosine value we want to find.

In the Structured Query Language, we can also use the ACOS function in SELECT query with the table field:

1. **SELECT** ACOS(column_Name) **AS** Alias_Name **FROM** Table_Name;

SELECT Teacher_First_Name,Teacher_marks,ACOS(Teacher_marks) AS num FROM Teacher_Info;

```
Teacher_First_Name      Teacher_marks   num
Arush   0       1.5707963267948966
Bulbul  45      NULL
Saurabh -78     NULL
Shivani -56     NULL
Avinash 0       1.5707963267948966
Shyam   54      NULL

[Execution complete with exit code 0]
```

ASIN Function in SQL

The ASIN is a SQL mathematics function which returns the arc sin of the specified number. This function returns NULL value, if the specified number is not between -1 to 1.

Syntax of ASIN Function

1. **SELECT** ASIN(Number) **AS** Alias_Name;

In the ASIN syntax, we have to pass that number whose arc sin value we want to calculate.

In the Structured Query Language, we can also use the ASIN function with the column of the table as shown in the following block:

```
1. SELECT ASIN(column_Name) AS Alias_Name FROM Table_Name;
```

```
SELECT Teacher_First_Name,Teacher_marks,ASIN(Teacher_marks) AS num FROM Teacher_Info;
```

```
Teacher_First_Name      Teacher_marks   num
Arush   0       0
Bulbul  45      NULL
Saurabh -78     NULL
Shivani -56     NULL
Avinash 0       0
Shyam   54      NULL

[Execution complete with exit code 0]
```

ATAN Function in SQL

The ATAN is a SQL function of mathematics which returns the arc tangent value of the specified number.

Syntax of ATAN Function

```
1. SELECT ATAN(Number) AS Alias_Name;
```

In the ATAN syntax, we have to pass that decimal number whose arc tangent value we want to find.

In the Structured Query Language, we can also use the ATAN function with the column of the table as shown in the following block:

```
1. SELECT ATAN(column_Name) AS Alias_Name FROM Table_Name;
```

```
SELECT Teacher_First_Name,Teacher_marks,ATAN(Teacher_marks) AS num FROM Teacher_Info;
```

```
Teacher_First_Name      Teacher_marks   num
Arush   0       0
Bulbul  45      1.5485777614681775
Saurabh -78     -1.557976516321996
Shivani -56     -1.5529410816553442
Avinash 0       0
Shyam   54      1.5522799247268875

[Execution complete with exit code 0]
```

TAN Function in SQL

The TAN() is a mathematics function which returns the tangent value of the given number in the Structured Query Language.

Syntax of TAN Function

1. **SELECT** TAN(Number) **AS** Alias_Name;

In this SELECT syntax, we have to pass that numeric number in the function whose tan value we want to find.

In the Structured Query Language, we can also use the TAN function in SELECT query with the table field:

1. **SELECT** TAN(Numeric_Column_Name) **AS** Alias_Name **FROM** Table_Name;

SELECT Teacher_First_Name,Teacher_marks,TAN(Teacher_marks) AS num FROM Teacher_Info;

```
Teacher_First_Name      Teacher_marks   num
Arush   0       0
Bulbul  45      1.6197751905438615
Saurabh -78     0.5991799983411151
Shivani -56     0.6112736881917098
Avinash 0       0
Shyam   54      0.6738001006480598

[Execution complete with exit code 0]
```

CEIL Function in SQL

The CEIL function in Structured Query Language returns the smallest integer value which is greater than or equal to the given number.

Syntax of CEIL Function

Syntax1: This syntax uses the CEIL function with the column name of the SQL table:

```
1. SELECT CEIL(Integer_Column_Name) AS Alias_Name FROM Table_Name;
```

In this first syntax, we have to specify the name of that integer column on which we want to execute the CEIL numeric function.

Syntax2: This syntax uses the CEIL function with the integer or decimal value:

```
1. SELECT CEIL(Decimal_Number);
```

SELECT Teacher_First_Name,Teacher_Salary,CEIL(Teacher_Salary) AS num FROM Teacher_Info;

```
Teacher_First_Name      Teacher_Salary  num
Arush   2000.91 2001
Bulbul  3800.56 3801
Saurabh 4500.76 4501
Shivani 4200.65 4201
Avinash 3800.56 3801
Shyam   3000.34 3001

[Execution complete with exit code 0]
```

FLOOR Function in SQL

The FLOOR numeric function in Structured Query Language returns the largest integer value which is smaller than or equal to the given number.

Syntax of FLOOR Function

Syntax1: This syntax uses the FLOOR function with the column name of the SQL table:

```
1. SELECT FLOOR(Integer_Column_Name) AS Alias_Name FROM Table_Name;
```

In this first syntax, we have to specify the name of that integer column on which we want to execute the FLOOR numeric function.

Syntax2: This syntax uses the FLOOR function with the integer or decimal value:

1. **SELECT** FLOOR(Decimal_Number);

SELECT Teacher_First_Name,Teacher_Salary,FLOOR(Teacher_Salary) AS num FROM Teacher_Info;

```
Teacher_First_Name      Teacher_Salary  num
Arush   2000.91 2000
Bulbul  3800.56 3800
Saurabh 4500.76 4500
Shivani 4200.65 4200
Avinash 3800.56 3800
Shyam   3000.34 3000

[Execution complete with exit code 0]
```

LIMIT Function in SQL

The LIMIT function in Structured Query Language returns the records from the table according to the specified limit value.

All the SQL version does not support the LIMIT function. It is important to note that the value of LIMIT must be a non-negative integer.

Syntax of LIMIT Function

In SQL, we can use the LIMIT function with the columns of the string as well as integers.

1. **SELECT** Column_Name1, Column_Name2, Column_Name3, ……., Column_Name N **FROM** Table_Name LIMIT Value;

In this syntax, we have to the specify the LIMIT keyword with its value after the name of the table.

SELECT Teacher_First_Name,Teacher_Salary FROM Teacher_Info LIMIT 3;

```
Teacher_First_Name      Teacher_Salary
Arush   2000.91
Bulbul  3800.56
```

```
Saurabh 4500.76

[Execution complete with exit code 0]
```

SELECT Teacher_First_Name,Teacher_Salary,Teacher_marks FROM Teacher_Info ORDER BY Teacher_marks DESC LIMIT 3;

```
Teacher_First_Name      Teacher_Salary  Teacher_marks
Shyam   3000.34 54
Bulbul  3800.56 45
Arush   2000.91 0

[Execution complete with exit code 0]
```

SELECT Teacher_First_Name,Teacher_Salary,Teacher_marks FROM Teacher_Info ORDER BY Teacher_marks LIMIT 3;

```
Teacher_First_Name      Teacher_Salary  Teacher_marks
Saurabh 4500.76 -78
Shivani 4200.65 -56
Arush   2000.91 0

[Execution complete with exit code 0]
```

CEILING Function in SQL

The CEILING function in Structured Query Language returns the smallest integer value which is greater than or equal to the given number.

Syntax of CEILING Function

Syntax1: This syntax uses the CEILING function with the column name of the SQL table:

1. **SELECT** CEILING(Integer_Column_Name) **AS** Alias_Name **FROM** Table_Name;

In this syntax, we have to specify the name of that integer column on which we want to execute the CEILING numeric function.

Syntax2: This syntax uses the CEILING function with the integer or decimal value:

1. **SELECT** CEILING(Decimal_Number);

SELECT Teacher_First_Name,Teacher_Salary,CEILING(Teacher_Salary) FROM Teacher_Info;

```
Teacher_First_Name      Teacher_Salary  CEILING(Teacher_Salary)
Arush   2000.91 2001
Bulbul  3800.56 3801
Saurabh 4500.76 4501
Shivani 4200.65 4201
Avinash 3800.56 3801
Shyam   3000.34 3001

[Execution complete with exit code 0]
```

DEGREES Function in SQL

The DEGREES is a mathematical function in the Structured Query Language which converts the specified value of radians into degree.

This function accepts the angle of radian as argument and returns the equivalent angle in degree. The return type of function is same as the argument type.

Syntax of DEGREES Function

Syntax1: This syntax uses the DEGREES function with the column name of the SQL table:

1. **SELECT** DEGREES(Numeric_Column_Name) **AS** Alias_Name **FROM** Table_Name;

In this SELECT syntax, we have to specify the name of that numeric column on which we want to use DEGREES function.

SELECT Teacher_First_Name,Teacher_Salary,DEGREES(Teacher_Salary) FROM Teacher_Info;

```
Teacher_First_Name      Teacher_Salary  DEGREES(Teacher_Salary)
Arush   2000.91 114643.58124394032
Bulbul  3800.56 217756.05114341475
Saurabh 4500.76 257874.53917260206
Shivani 4200.65 240679.5106163383
Avinash 3800.56 217756.05114341475
Shyam   3000.34 171906.8241400433
```

```
[Execution complete with exit code 0]
```

EXP Function in SQL

The EXP is a SQL function of mathematics which returns e raised to the power of given number.

Syntax of EXP Function

1. **SELECT** EXP(Number) **AS** Alias_Name;

In the EXP syntax, we have to pass that decimal number whose e raised value we want to return.

In the Structured Query Language, we can also use the EXP function with the field of the table as shown in the following block:

1. **SELECT** EXP(column_Name) **AS** Alias_Name **FROM** Table_Name;

In this syntax, we have to define the name and column of that table on which we want to perform the EXP function.

SELECT Teacher_First_Name,Teacher_marks,EXP(Teacher_marks) FROM Teacher_Info;

```
Teacher_First_Name      Teacher_marks   EXP(Teacher_marks)
Arush   0       1
Bulbul  45      3.4934271057485095e19
Saurabh -78     1.3336148155022614e-34
Shivani -56     4.780892883885469e-25
Avinash 0       1
Shyam   54      2.830753303274694e23

[Execution complete with exit code 0]
```

RADIANS Function in SQL

The RADIANS numeric function of Structured Query Language returns the radians value of the specified degree.

Syntax of RADIANS Function

Syntax1: This syntax uses the RADIANS function with the column name of the SQL table:

1. **SELECT** RADIANS(Numeric_Column_Name) **AS** Alias_Name **FROM** Table_Nam e;

In this first syntax, we have to specify the name of that column on which we want to execute the RADIANS function for converting the degree value into radians.

Syntax2: This syntax uses the RADIANS function with the number:

1. **SELECT** RADIANS(Number);

SELECT Teacher_First_Name,Teacher_marks,RADIANS(Teacher_marks) FROM Teacher_Info;

```
Teacher_First_Name      Teacher_marks   RADIANS(Teacher_marks)
Arush   0       0
Bulbul  45      0.7853981633974483
Saurabh -78     -1.361356816555577
Shivani -56     -0.9773843811168246
Avinash 0       0
Shyam   54      0.9424777960769379

[Execution complete with exit code 0]
```

RAND Function in SQL

The RAND() is a mathematical function in Structured Query Language which returns the random number between 0 and 1. 0 and 1 may also be returned by the function in the output.

Syntax of RAND Function

1. **SELECT** RAND(Number) **AS** Alias_Name;

In this SELECT syntax, we have to pass that numeric number in the function whose rand value we want to find.

In the Structured Query Language, we can also use the RAND function in SELECT query with the table field:

1. **SELECT** RAND(column_Name) **AS** Alias_Name **FROM** Table_Name;

In this SELECT query, we have to define the name and field of that table on which we want to perform the RAND function.

```
SELECT
Teacher_First_Name,Teacher_ID,RAND(Teacher_ID)
FROM Teacher_Info;
```

```
Teacher_First_Name      Teacher_ID      RAND(Teacher_ID)
Arush   1001    0.5885127331954546
Bulbul  1002    0.8386958426224961
Saurabh 1004    0.33906206147657897
Shivani 1005    0.5892451709036205
Avinash 1006    0.839428280330662
Shyam   1007    0.08961138975770343

[Execution complete with exit code 0]
```

ATN2 Function in SQL

The ATN2 is a mathematics function which returns the arc tangent of two specified numbers.

Syntax of ATN2 Function

1. **SELECT** ATN2(Number1, Number2) **AS** Alias_Name;

In the ATN2 syntax, we have to pass two numeric numbers in the function from which we can find the value of arc tangent.

In the Structured Query Language, we can also use the ATN2 function in the SELECT query with the table fields:

1. **SELECT** ATN2(Column_Name1, Column_Name2) **AS** Alias_Name **FROM** Table_Name;

In this SELECT query, we have to define the name and fields of that table on which we want to perform the ATN2 function.

```
SELECT
Teacher_First_Name,Teacher_ID,ATN2(Teacher_ID,Teacher_Dept_Id) FROM Teacher_Info;
```

```
ERROR 1370 (42000) at line 23: execute command denied to user
'mycompiler'@'localhost' for routine 'mycompiler.ATN2'
```

```
[Execution complete with exit code 1]
```

LOG Function in SQL

The LOG is a string function in SQL which returns the logarithm of the given number. Or, we can say that, it shows the logarithms of the number to the given base.

Syntax of LOG Function

1. **SELECT** LOG(Number1, Number2) **AS** Alias_Name;

In the LOG syntax, Number1 must be greater than 0 and it is the number whose log we want to find. Number2 is the base.

In the Structured Query Language, we can also use the LOG function with the columns of the table as shown in the following block:

1. **SELECT** LOG(Column_Name1, Column_Name2) **AS** Alias_Name **FROM** Table_Name;

In this syntax, we have to define the name and columns of that table on which we want to perform the LOG function.

SELECT Teacher_First_Name,Teacher_marks,LOG(Teacher_marks) FROM Teacher_Info;

```
Teacher_First_Name      Teacher_marks   LOG(Teacher_marks)
Arush   0       NULL
Bulbul  45      3.8066624897703196
Saurabh -78     NULL
Shivani -56     NULL
Avinash 0       NULL
Shyam   54      3.9889840465642745

[Execution complete with exit code 0]
```

LOG2 Function in SQL

The LOG2 is a numeric function in SQL which returns the natural logarithm of the number to the base 2.

Syntax of LOG2 Function

1. **SELECT** LOG2(Number) **AS** Alias_Name;

In the LOG2 syntax, we have to pass that decimal number whose log base 2 value we want to find.

In the Structured Query Language, we can also use the LOG2 function with the column of the table as shown in the following block:

1. **SELECT** LOG2(column_Name) **AS** Alias_Name **FROM** Table_Name;

In this syntax, we have to define the name and column of that table on which we want to perform the LOG2 function

SELECT Teacher_First_Name,Teacher_ID,LOG2(Teacher_ID) FROM Teacher_Info;

```
Teacher_First_Name      Teacher_ID      LOG2(Teacher_ID)
Arush   1001    9.967226258835993
Bulbul  1002    9.968666793195208
Saurabh 1004    9.971543553950772
Shivani 1005    9.972979786066292
Avinash 1006    9.974414589805527
Shyam   1007    9.975847968006784

[Execution complete with exit code 0]
```

LOG10 Function in SQL

The LOG10() is a mathematics function which returns the logarithm of the specified number to base 10.

Syntax of LOG10 Function

1. **SELECT** LOG10(Number) **AS** Alias_Name;

In this SELECT syntax, we have to pass that numeric number in the function whose log 10 value we want to find.

In the Structured Query Language, we can also use the LOG10 function in SELECT query with the table field:

1. **SELECT** LOG10(column_Name) **AS** Alias_Name **FROM** Table_Name;

SELECT Teacher_First_Name,Teacher_ID,LOG10(Teacher_ID) FROM Teacher_Info;

```
Teacher_First_Name      Teacher_ID      LOG10(Teacher_ID)
Arush   1001    3.000434077479319
Bulbul  1002    3.000867721531227
Saurabh 1004    3.0017337128090005
Shivani 1005    3.002166061756508
Avinash 1006    3.0025979807199086
Shyam   1007    3.003029470553618

[Execution complete with exit code 0]
```

GREATEST Function in SQL

The GREATEST is a SQL numeric function which shows the greatest value from the specified inputs in Structured Query Language.

Syntax of GREATEST Function

1. **SELECT** GREATEST(Number1, Number2, Number3, Number4, ..., NumberN) **AS** Alias_Name;

In the GREATEST syntax, we have to pass those numbers from which we want to find greatest value.

In the Structured Query Language, we can also use the GREATEST function with the column of the table as shown in the following block:

1. **SELECT** GREATEST(Integer_column_Name1, Integer_column_Name1, Integer_column_Name1, Integer_column_Name1, ..., Integer_column_NameN,) **AS** Alias_Name **FROM** Table_Name;

In this syntax, we have to define the name and columns of that table on which we want to perform the GREATEST function.

SELECT Teacher_First_Name,Teacher_ID,Teacher_Dept_Id,GREATEST(Teacher_Dept_Id,Teacher_ID) AS greater_value FROM Teacher_Info;

```
Teacher_First_Name      Teacher_ID      Teacher_Dept_Id greater_value
Arush   1001    4001    4001
```

```
Bulbul  1002    4002    4002
Saurabh 1004    4001    4001
Shivani 1005    4001    4001
Avinash 1006    4002    4002
Shyam   1007    4003    4003

[Execution complete with exit code 0]
```

POW Function in SQL

The POW is a mathematical function in SQL which returns the value of a number raised to the power of another number. This function is similar to the POWER function. In the POW function, we have to pass the two number as the argument in which one number acts as exponent and other acts as the base.

Syntax of POW String Function

1. **SELECT** POW(Number1, Number2) **AS** Alias_Name;

In this POW function, following are the two arguments:

1. Number1: It acts as the base in the power function

2. Number2: And, it acts as the exponent.

In Structured Query Language, we can also use the POW function with the integer columns of the table as shown in the following block:

1. **SELECT** POW(column_Name1, column_Name2) **AS** Alias_Name **FROM** Table_ Name;

In this syntax, we used the POW function with existing table of SQL. Here, we have to define the name and columns of that table on which we want to perform the POW function.

SELECT Teacher_First_Name,Teacher_marks,POW(Teacher_marks,5) AS greater_value FROM Teacher_Info;

```
Teacher_First_Name      Teacher_marks   greater_value
Arush   0       0
Bulbul  45      184528125
Saurabh -78     -2887174368
Shivani -56     -550731776
Avinash 0       0
```

```
Shyam   54      459165024

[Execution complete with exit code 0]
```

DIV Function in SQL

The DIV is a string function in SQL which returns the quotient by dividing the first number from second number.

Syntax of DIV Function

1. **SELECT** DIV(Number1, Number2) **AS** Alias_Name;

In the DIV syntax, Number1 is the dividend and Number2 is the divisor.

In the Structured Query Language, we can also use the DIV function with the columns of the table as shown in the following block:

1. **SELECT** DIV(Column_Name1, Column_Name2) **AS** Alias_Name **FROM** Table_Name;

In this syntax, we have to define the name and columns of that table on which we want to perform the DIV function.

SELECT Teacher_First_Name,Teacher_marks,POW(Teacher_Dept_Id, Teacher_marks) AS greater_value FROM Teacher_Info;

```
Teacher_First_Name      Teacher_marks   greater_value
Arush   0       1
Bulbul  45      1.2661022877005214e162
Saurabh -78     1.0736259312179085e-281
Shivani -56     1.8991581115525343e-202
Avinash 0       1
Shyam   54      3.3792620077547213e194

[Execution complete with exit code 0]
```

LEAST Function in SQL

The LEAST is a SQL numeric function which shows the least value from the specified inputs in Structured Query Language.

Syntax of LEAST Function

1. **SELECT** LEAST(Number1, Number2, Number3, Number4, ……, NumberN) **AS** Al ias_Name;

In the LEAST syntax, we have to pass those numbers from which we want to find least value.

In the Structured Query Language, we can also use the LEAST function with the column of the table as shown in the following block:

1. **SELECT** LEAST(Integer_column_Name1, Integer_column_Name1, Integer_column _Name1, Integer_column_Name1, ……., Integer_column_NameN,) **AS** Alias_Name **FROM** Table_Name;

In this syntax, we have to define the name and columns of that table on which we want to perform the LEAST function.

SELECT Teacher_First_Name,Teacher_ID,Teacher_marks,LEAST(Tea cher_Dept_Id,Teacher_marks) AS least_value FROM Teacher_Info;

```
Teacher_First_Name      Teacher_ID      Teacher_marks   least_value
Arush   1001    0       0
Bulbul  1002    45      45
Saurabh 1004    -78     -78
Shivani 1005    -56     -56
Avinash 1006    0       0
Shyam   1007    54      54

[Execution complete with exit code 0]
```

LN Function in SQL

The LN is a numeric function in SQL which returns the natural logarithm of the specified integer number.

Syntax of LN Function

1. **SELECT** LN(Number) **AS** Alias_Name;

In the LN syntax, we have to pass that decimal number whose log base 2 value we want to find.

In the Structured Query Language, we can also use the LN function with the column of the table as shown in the following block:

```
1. SELECT LN(column_Name) AS Alias_Name FROM Table_Name;
```

SELECT Teacher_First_Name,Teacher_ID,Teacher_marks,LN(Teacher_marks) AS value1 FROM Teacher_Info;

```
Teacher_First_Name      Teacher_ID      Teacher_marks   value1
Arush   1001    0       NULL
Bulbul  1002    45      3.8066624897703196
Saurabh 1004    -78     NULL
Shivani 1005    -56     NULL
Avinash 1006    0       NULL
Shyam   1007    54      3.9889840465642745

[Execution complete with exit code 0]
```

UNICODE Function in SQL

The UNICODE function of Structured Query Language shows the unicode (integer) value of the first character of the string. We can also use the UNICODE function with the string fields of the SQL table.

Syntax of UNICODE String Function

Syntax1: This syntax uses the UNICODE function with the column names of the SQL table:

```
1. SELECT UNICODE (Column_Name) AS Alias_Name FROM Table_Name;
```

In the syntax, we have to specify the column's name on which we want to use the UNICODE string function.

Syntax2: This syntax uses the UNICODE function with the set of characters (string):

```
1. SELECT UNICODE (String);
```

Syntax2: This syntax uses the UNICODE function with the individual character:

```
1. SELECT UNICODE (character);
```

SELECT Teacher_First_Name, UNICODE(Teacher_Last_Name) AS value1 FROM Teacher_Info;

```
 ERROR 1064 (42000) at line 23: You have an error in your SQL syntax; check
the manual that corresponds to your MySQL server version for the right syntax
to use near '(Teacher_Last_Name) AS value1 FROM Teacher_Info' at line 1

[Execution complete with exit code 1]
```

REPLICATE Function in SQL

The REPLICATE is a string function in SQL. It shows the inputted string in the output to the given number of times.

Syntax of REPLICATE String Function

Syntax1: This syntax describes how to use the REPLICATE with the fields of the structured table.

1. **SELECT** REPLICATE(Column_Name, Repetation_value) **AS** Alias_Name **FROM** T able_Name;

If we want to perform Replicate function, then we have to specify the name of that column from the table whose values we want to repeat.

Syntax2: This syntax descibes how to use the REPLICATE function with the string or sentence:

1. **SELECT** REPLICATE(Original_String, Repetation_value) **AS** Alias_Name;

Syntax3: This syntax descibes how to use the REPLICATE function with the individual character:

1. **SELECT** REPLICATE(**Character**, Repetation_value) **AS** Alias_Name;

SELECT Teacher_First_Name,Teacher_marks, REPLICATE(Teacher_marks) AS value1 FROM Teacher_Info;

```
ERROR 1370 (42000) at line 23: execute command denied to user
'mycompiler'@'localhost' for routine 'mycompiler.REPLICATE'

[Execution complete with exit code 1]
```

TRUNCATE Function in SQL

The TRUNCATE is a numeric function in SQL which truncates the number according to the particular decimal points.

Syntax of TRUNCATE Function

1. **SELECT TRUNCATE**(X, D) **AS** Alias_Name;

In the TRUNCATE syntax, X is the integer number and D is the decimal points.

In the SQL, you can also use the TRUNCATE function with the integer field of the table as shown in the following block:

1. **SELECT TRUNCATE**(Integer_Column_Name, Decimal_point) **AS** Alias_Name **F ROM** Table_Name;

In this syntax, we have to define the name and columns of that table on which we want to perform the TRUNCATE function.

SELECT Teacher_First_Name,Teacher_Salary, TRUNCATE(Teacher_Salary,1) AS value1 FROM Teacher_Info;

```
Teacher_First_Name      Teacher_Salary  value1
Arush   2000.91 2000.9
Bulbul  3800.56 3800.5
Saurabh 4500.76 4500.7
Shivani 4200.65 4200.6
Avinash 3800.56 3800.5
Shyam   3000.34 3000.3

[Execution complete with exit code 0]
```

DATALENGTH Function in SQL

The DATALENGTH string function of Structured Query Language returns the number of bytes used to indicate the expression.

Syntax of DATALENGTH String Function

Syntax1: This syntax uses the DATALENGTH function with the column name of the SQL table:

```
1. SELECT DATALENGTH(Column_Name) AS Alias_Name FROM Table_Name;
```

In the syntax, we have to specify the name of that column on which we want to perform the DATALENGTH string function for finding the number of bytes used to represent the string or expression.

Syntax2: This syntax uses the DATALENGTH function with the string:

```
1. SELECT DATALENGTH(Original_String);
```

```
SELECT
Teacher_First_Name,DATALENGTH(Teacher_First_Name)
AS value1 FROM Teacher_Info;
```

```
ERROR 1370 (42000) at line 23: execute command denied to user
'mycompiler'@'localhost' for routine 'mycompiler.DATALENGTH'

[Execution complete with exit code 1]
```

NCHAR Function in SQL

The NCHAR string function shows the unicode value of the integer passed in the function. This function takes only one parameter or argument. If we pass the integer value which exceeds the given range then it shows NULL value.

Syntax of NCHAR String Function

In SQL, we can use the NCHAR function with the columns of the table, strings, and characters.

Syntax1:

```
1. SELECT NCHAR(Integer_Column_Name) AS Alias_Name FROM Table_Name;
```

In this syntax, we used the NCHAR function with existing table of SQL. Here, we have to define the name and integer column of that table on which we want to perform NCHAR function.

OCTET_LENGTH Function in SQL

The OCTET_LENGTH string function of Structured Query Language returns the number of characters of the given string or word.

Syntax of OCTET_LENGTH String Function

Syntax1: This syntax uses the OCTET_LENGTH function with the column name of the SQL table:

```
1. SELECT OCTET_LENGTH(Column_Name) AS Alias_Name FROM Table_Name;
```

In this first syntax, we have to specify the name of that column on which we want to execute the OCTET_LENGTH string function for finding the number of characters of each value.

Syntax2: This syntax uses the OCTET_LENGTH function with the string:

```
1. SELECT OCTET_LENGTH(Original_String);
```

Syntax2:

```
1. SELECT NCHAR(Integer_Value) AS Alias_Name;
```

In this syntax, we used the NCHAR function with the integer value.

SELECT Teacher_First_Name, OCTET_LENGTH(Teacher_First_Name) AS value1 FROM Teacher_Info;

```
Teacher_First_Name      value1
Arush   5
Bulbul  6
Saurabh 7
Shivani 7
Avinash 7
Shyam   5

[Execution complete with exit code 0]
```

PATINDEX Function in SQL

The PATINDEX string function in Structured Query Language returns the position of the specified pattern in the original string. If Sub-string is omitted in the original string, the PATINDEX function returns 0. The first position of the original string is indicated as 1.

Syntax of PATINDEX String Function

Syntax1: This syntax uses the PATINDEX function with the column name of the SQL table:

```
1. SELECT PATINDEX(Pattern, Column_Name) AS Alias_Name FROM Table_Nam
   e;
```

In the syntax, we have to specify the name of that column on which we want to find the position of string.

Syntax2: This syntax uses the PATINDEX function with the string:

```
1. SELECT PATINDEX(Pattern, Original_String) AS Alias_Name;
```

SELECT Teacher_First_Name, PATINDEX('a',Teacher_First_Name) AS value1 FROM Teacher_Info;

```
ERROR 1370 (42000) at line 23: execute command denied to user
'mycompiler'@'localhost' for routine 'mycompiler.PATINDEX'

[Execution complete with exit code 1]
```

PI Function in SQL

The PI() is a mathematics function which returns the pi value. In this section you will also learn how to use round function with the PI function.

In Structured Query Language, this function does not take any argument.

Syntax of PI Function

```
1. SELECT PI() AS Alias_Name;
```

In this SELECT syntax, we have to pass that numeric number in the function whose tan value we want to find.

Examples of PI function

Example 1: This example returns the pi value of the specified number:

```
1. SELECT PI() + 8 AS pi_plus8;
```

SELECT PI()+8 AS pi ;

```
pi
11.141593

[Execution complete with exit code 0]
```

ORD Function in SQL

The ORD function of Structured Query Language shows the code of left-most character of the specified string or word. SQL also allows you to perform the ORD function on the String fields of the table.

Syntax of ORD String Function

Syntax1: This syntax uses the ORD function with the column names of the SQL table:

```
1. SELECT ORD(Column_Name) AS Alias_Name FROM Table_Name;
```

In the syntax, we have to specify the column's name on which we want to use the ORD string function.

Syntax2: This syntax uses the ORD function with the set of characters (string):

```
1. SELECT ORD(String);
```

Syntax2: This syntax uses the ORD function with the individual character:

```
1. SELECT ORD(character);
```

SELECT Teacher_First_Name,ORD(Teacher_First_Name) AS value1 FROM Teacher_Info;

```
Teacher_First_Name      value1
Arush   65
Bulbul  66
Saurabh 83
Shivani 83
Avinash 65
Shyam   83

[Execution complete with exit code 0]
```

SQL GENERAL FUNCTIONS

SQL General Functions: NVL, NVL2, DECODE, COALESCE, LNNVL and NANVL

Introduction

Today, we are going to learn about SQL general Functions. The General Functions we are going to learn about are:

- NVL()
- NVL2()
- DECODE()
- COALESCE()
- LNNVL()

1.) NVL()

This is one of the functions of SQL extensively used in Structured Query Language (SQL).

This function can hold two input values only. If input values are more than 2 then an error is returned. This functions returns the first NOT NULL value when searched in the function.

If both the inputs are NULL, then there is no output for this function.

The input data type can be Integer, Floating Point Number, String, Character input, etc.

Syntax

```
NVL (input 1, input 2)
SQL> select NVL(1, 2) from dual;

 NVL(1,2)
_ _ _ _ _ _
        1
SQL> select NVL(NULL, 1) from dual;


NVL(NULL,1)
```

```
– – – – – – – –
        1
SQL> select NVL(1.029384, 1.029384) from dual;

NVL(1.029384,1.029384)
– – – – – – – – – – – – – –

            1.029384

SQL> select NVL(NULL, 1.029384) from dual;

NVL(NULL,1.029384)
– – – – – – – – – – – –
        1.029384
```

NVL2()

This is one of the functions of SQL extensively used in Structured Query Language (SQL).

This function can hold three input values only. If input values are more than three then an error is returned. This function returns the first value after NOT NULL value is found, when searched in the function.

If the first value is NOT NULL, second value is returned.

If the first value is NULL and the second value is NOT NULL, third value is returned.

The returned value can also be a NULL value too.

The working of this functioning is as same as NVL () in SQL.

If both the inputs are NULL, then there is no output for this function.

The input data type can be Integer, Floating Point Number, String, Character input, etc.

Syntax

1. NVL2(input 1, input 2, input 3)

Example Queries

```
SQL> select NVL2(1, 2, 3) from dual;

NVL2(1,2,3)
- - - - - - - -
        2

SQL> select NVL2(2, 2, 3) from dual;

NVL2(2,2,3)
- - - - - - - -
        2

SQL> select NVL2(2, 4, 3) from dual;

NVL2(2,4,3)
- - - - - - - -
        4

SQL> select NVL2(2, NULL, 3) from dual;

NVL2(2,NULL,3)
- - - - - - - - -

SQL> select NVL2('Kevin', 'Pitersen', ' SA / ENG ') from dual;

NVL2('KE
- - - - - -
Pitersen
```

DECODE()

This is also one of the expressions used in SQL. This Decode expression is used as IF, ELSE IF, ELSE IF Ladder style. This decode works on the basis of the condition specified.

Any kind of operation specified is going to work here.

The input types must chosen based on the data types specified.

Syntax

```
DECODE (column  name,  number 1 to be searched, result 1 to be updated
                , number 2 to be searched, result 2 to be updated
                , number 3 to be searched, result 3 to be updated
          . . . . . .
                number n to be searched, result n to be updated , default)
```

```
SQL > select * from ipla;


SID  SNAME          SAL       AGE
---  -----      ----  ----
   1   mahi           12        40
   2  kohli           14         33
   3   DK             6.25      33
   4  warner         6.75      33
   5  rahul          16         29
   6  pandya         14        27
```

```
SQL > SELECT Sname, sid, sal,
  2       DECODE (sid, 1, 1.5*sal,
  3                       2, 4*sal,
  4                       3, 9*sal,
  5                       4, 10.25*sal,
  6              sal)
  7       "REVISED SALARY"
  8  from ipla ;
```

```
SNAME       SID       SAL       REVISED SALARY
---  -----      ----  ----------
mahi             1       12          18
kohli            2       14          56
DK               3     6.25          56.25
warner           4     6.75          69.1875
rahul            5       16          16
```

pandya 6 14 14

```
CREATE TABLE Teacher_Info
(
Teacher_ID INT ,
Teacher_First_Name VARCHAR (100),
Teacher_Last_Name VARCHAR (100),
Teacher_Dept_Id INT ,
Teacher_marks INT,
Teacher_Address Varchar (80),
Teacher_City Varchar (80),
Teacher_Salary FLOAT
);
```

```
INSERT INTO Teacher_Info (Teacher_ID,
Teacher_First_Name, Teacher_Last_Name,
Teacher_Dept_Id,Teacher_marks, Teacher_Address,
Teacher_City, Teacher_Salary) VALUES
    (1001, 'Arush', 'Sharma', 4001,0, '22 street', 'New Delhi',
NULL),
    (NULL, 'Bulbul', 'Roy', 4002,45, '120 street', 'New Delhi',
3800.560),
    (1004, 'Saurabh', 'Sharma', NULL,-78, '221 street',
'Mumbai', 4500.760),
```

```
    (NULL, 'Shivani', 'Singhania', 4001,-56, '501 street',
'Kolkata', 4200.650),
    (NULL, 'Avinash', 'Sharma', NULL,0, '12 street', 'Delhi',
3800.560),
    (1007, 'Shyam', 'Besas', NULL,54, '202 street', 'Lucknow',
3000.340);



SELECT Teacher_First_Name, Teacher_ID, Teacher_Salary,
DECODE (Teacher_ID, 1001, 1.5*Teacher_Salary,
1004, 4*Teacher_Salary,
1007, 10.25*Teacher_Salary,
Teacher_Salary)
"REVISED SALARY"
from Teacher_Info ;
```

COALESCE()

This also one of the expression used in SQL. This expression works similar to NVL () expression. The only difference it can accept inputs greater than two. It returns the first NOT NULL input element.

The input data type can be anything. The inputs can be int, float, string, character, number, etc.

Syntax

1. COALESCE (input 1, input 2, input 3, ..., input n)

Example Queries

```
SQL> select COALESCE(NULL, 1) from dual;

COALESCE(NULL,1)
----------------
               1
SQL> select COALESCE(1, 2, 2) from dual;

COALESCE(1,2,2)
---------------
              1

SQL> select COALESCE(NULL, 2, 2) from dual;

COALESCE(NULL,2,2)
------------------
                 2

SQL> select COALESCE(NULL, NULL, 2) from dual;

COALESCE(NULL,NULL,2)
---------------------
                    2

SQL> select COALESCE(NULL, NULL, NULL) from dual;

C
-


SQL> select COALESCE(NULL, NULL, NULL, 1, 2, 3, 4, 5, 6) from dual;

COALESCE(NULL,NULL,NULL,1,2,3,4,5,6)
------------------------------------
                                   1
SQL> select COALESCE(NULL, 'NULL', 'Stuart Broad', 'Adam Gilchrist') from dual;
```

```
COAL
----
NULL
```

LNNVL()

This is one of the function of SQL which is used in SQL. This is used to convert True to False and False to True.

The LNNVL () function has the capacity to hold a condition. This makes the condition go reverse.

If the condition is SID = 2. Then LNNVL (SID = 2) is equivalent to SID ! = 2.

Syntax

1. LNNVL (Condition)

Example Queries

```
SQL> select * from ipla;

SID  SNAME          SAL       AGE
---  -----   ----  ----
 1   mahi            12        40
 2  kohli           14        33
 3   DK             6.25       33
 4  warner           6.75       33
 5  rahul           16        29
 6  pandya           14         27
 7  Tim David         8.25       26

7 rows selected.

SQL> select * from ipla where sid=2;
SID  SNAME          SAL       AGE
---  -----    ----  ----
  2     kohli          14          33
```

```
SQL> select * from ipla where LNNVL (sid = 2) ;

SID  SNAME          SAL       AGE
_ _ _ _ _ _ _ _    _ _ _ _   _ _ _ _
1    mahi           12         40
3    DK            6.25        33
4   warner          6.75        33
5   rahul          16          29
6   pandya          14          27
7   Tim David       8.25         26
```

NANVL ()

If the input value n2 is NaN (not a number), this method returns an alternative value n1, and if n2 is not NaN, it returns n2. Only floating-point numbers of the types BINARY FLOAT or BINARY DOUBLE can be used with this function.

The function accepts any numeric or nonnumeric data type as an input, with the ability to implicitly convert to a numeric data type.

The method returns BINARY DOUBLE if the parameter is BINARY FLOAT. If not, the function returns a numeric data type that matches the parameter.

Syntax

NANVL (input 1, input 2)

```
SID  SNAME          SAL       AGE
_ _ _   _ _ _ _ _    _ _ _ _   _ _ _ _
   1   mahi           12         40
   2  kohli          14         33
   3   DK            6.25        33
   4  warner          6.75        33
   5  rahul          16          29
   6  pandya          14          27
   7   Tim David       8.25         26
SQL> select Sname, sid, sal,
  2         DECODE (SID, 1,  (sal + sid * 5),
  3                      2, (sal + sid*4),
```

```
 4                  3, NVL(sal*4, NULL),
 5                  4, NVL2(NULL, sal*3, sal*4),
 6                  5, COALESCE(NULL, NULL, sal+4),
 7                  6, 3 * sal,
 8                  sal*2)
 9                  "REVISED SALARY "
10                 from ipla;
```

SID	SNAME	SAL	REVISED SALARY
1	mahi	12	17
2	kohli	14	22
3	DK	6.25	25
4	warner	6.75	27
5	rahul	16	20
6	pandya	14	42
7	Tim David	8.25	16.5