SQL

Structured Query Language

- SQL (Structured Query Language) is a programming language used to communicate with data stored in a relational database management system.
- SQL syntax is like the English language, which makes it relatively easy to write, read, and interpret.
- SQL is not case sensitive.
- SQL is a domain-specific language used in programming and designed for managing data held in database management system (RDBMS).

PURPOSE OF SQL:

- It is used for storing and managing data in a relational database management system (RDMS).
- It is a standard language for relational database systems. It enables a user to create, read, update and delete relational databases and tables.
- 5 types of SQL queries.
- Data Definition Language (DDL).
- Data Manipulation Language (DML).
- Data control Language (DCL)
- Transaction control language (TCL)
- Data Query Language (DQL).

DATA DEFINITION LANGUAGE (DDL):

DDL helps to define the database structure or schema.

5 types of DDL commands in SQL are:

CREATE

CREATE statement is used to define the database structure schema:

Syntax:

CREATE TABLE TABLE_NAME (COLUMN_NAME DATA TYPE [ ]);

For example:

Create database student;

DROP

Drop command remove tables and databases from RDMS.

Syntax:

DROP TABLE TABLE_NAME;

For example:

Drop table student;

ALTER

Alters command allows you to alter the structure of the database.

Syntax: To add a new column in the table

ALTER TABLE table_name ADD column_name Column-defination;

To modify an existing column in the table:

ALTER TABLE MODFIY (COLUMN DEFINITION);

For example:

Alter table student add subject varchar;

TRUNCATE

This command is used to delete rows from the table and free the space containing the table.

Syntax: TRUNCATE TABLE table_name;

DATA MANIPULATION LANGUAGE(DML):

DML allows you to modify the database instance by inserting modifying and deleting its data. It is responsible for performing all types of data modification in a database.

DML COMMANDS IN SQL:

- INSERT
- UPDATE
- DELETE

INSERT

This command is used to insert data into the row of a table.

Syntax:

INSERT INTO TABLE_NAME (col1, col2, col3,……col N) VALUES(value1, value2, value3,…..valueN);

UPDATE

This command is used to update or modify the value of the value of a column in the table.

Syntax:

UPDATE table_name SET [column_name1 = value1,…column_nameN = valueN] [WHERE CONDITION]

DELETE

This command is used to remove or more rows from a table.

Syntax:

DELETE FROM table_name [WHERE condtion];

DATA CONTROL LANGUAGE:

DCL includes commands like GRANT and REVOKE, which are useful to give “rights & permissions”.

DCL commands:

- GRANT
- REVOKE

GRANT

This command is use to give user access privileges to a database.

Syntax:

GRANT SELECT, UPDATE ON MY_TABLE TO SOME_USER , ANOTHER_USER;

For example:

GRANT SELECT ON users TO ‘Tom@’localhost;

REVOKE

It is useful to back permissions from the user.

Syntax:

REVOKE privilege_nameON object_name From {user_name|PUBILC|role_name}

For example:

REVOKE SELECT, UPDATE ON student FROM BCA, MCA;

TRANSACTION CONTROL LANGUAGE(TCL):

Transaction control language or TCL deals within the database.

COMMIT

This command is used to save all the transaction to database.

Syntax:

Commit;

ROLLBACK

Rollback command allows you to undo the transactions that have not already saved into the database.
Syntax:
Rollback;

SAVEPOINT

This command helps to sets a savepoint within a transaction.
Syntax:
SAVEPOINT SAVEPOINT_NAME;
For example:
SAVEPOINT Roll_number;
DATA QUERY LANGUAGE(DQL):
DQL is used to fetch data from database.

SELECT

This command will help you to select the attributes based on the condition described by the WHERE clause.

Syntax:

SELECT expressions FROM TABLE WHERE conditions;

DATA TYPES:

Data type of a column in a table defines what value the column can hold: integer, character, date and time, binary and so on.

Each column in a database table is required to have a name and a data type. The data type is a guideline for SQL to understand what type of data is expected inside each column, and it also identifies how SQL will interact.

MySQL supports 3 main data types:

- String
- Numeric
- Date and time

STRING DATA TYPES:

CHAR(size) – A Fixed length(can contain letters, Numbers, and special characters). The size parameter specific is the column length in character can be from zero to 255. Default is 1.
VARCHAR(size) – A VARIABLE length string(can contain letters, numbers, and special characters). The size parameter specifies the maximum column length in characters – can be from 0 to 65535.
BINARY(size)- Equal to CHAR( ) , but stores binary byte strings. The size parameter specifies the column length in bytes. Default is 1.
VARBINARY(size)- Equal to VARCHAR( ), but stores binary byte strings. The size parameter specifes the maximum column length in bytes.
TINYBLOB – For BLOBs(Binary Large Objects). Max length: 255 bytes.
TINYTEXT – Holds a string with a maximum length of 255 characters.
TEXT(size)- Holds a string with a maximum length of 65,535 bytes.

NUMERIC DATA TYPES:

numeric(p,s) – Fixed precision and scale numbers. Allows numbers from -10^38 + 1 to 10^38 -1.

P parameter indicates the maximum total number of digits that can be stored. p must be a Value from one to 38. Default is 18. There's parameter indicates the maximum number of digits stored to the right of the decimal point. s must be a value from 0 to p. default value is 0. 5-17 bytes.
smallmoney – Monetary data from -214,748.3648 to 214,748.3647. 4bytes.
money – Monetary data from -922,337,203,685,477.5808 to 922,337,203,685,477.5807. 8 bytes.
float(n) – floating precision number data from -1.79E + 308 to 1.79E + 308. The n parameter indicates whether the field should hold 4 or 8 bytes. Float(24) holds a 4-byte field and float(53) holds an 8 byte field. Default value of n is 53. 4 or 8 bytes.
real – floating precision Number data from 3.40 E + 38 to 3.40 E + 38. 4 bytes.

DATE AND TIME:
DATE stores a date value in the format YYYY-MM-DD. The range is from 1000-01-01 to 9999-12-31 and it uses 3 bytes of storage. Example: '2024-08-03'.
DATETIME stores both date and time in the format YYYY-MM-DD HH:MM:SS. The range is from 1000-01-01 00:00:00 to 9999-12-31 23:59:59 and it uses 5 bytes of storage (up to 8 bytes if fractional seconds are used). It supports fractional seconds up to microseconds. Example: '2024-08-03 14:30:00'.
TIMESTAMP stores both date and time in the format YYYY-MM-DD HH:MM:SS. The range is from 1970-01-01 00:00:01 UTC to 2038-01-19 03:14:07 UTC and it uses 4 bytes of storage. It is affected by the time zone setting and supports fractional seconds up to microseconds. Example: '2024-08-03 14:30:00'.
TIME stores the time in the format HH:MM:SS. The range is from -838:59:59 to 838:59:59 and it uses 3 bytes of storage. It supports fractional seconds up to microseconds. Example: '14:30:00'.
YEAR stores a year in either a 2-digit or 4-digit format. The 4-digit format range is from 1901 to 2155 and 0000. The 2-digit format range is from 70 to 69 representing years from 1970 to 2069. It uses 1 byte of storage. Example: '2024'.

SQL SERVER DATA TYPES:
Char(n) – Fixed width character string. Maximum size 8000 characters.
Varchar(n) – Variable width character string maximum size 8000 characters.
Varchar(max) variable width character string maximum size 1,073,741,824 characters.
Text variable width character Singh maximum size 2GB of text data
Nchar- Fixed width Unicode Singh maximum size 4000 characters
NVARCHAR - Variable width Unicode Singh maximum size 536,870,912 characters.
Ntext – Variable with Unicode Singh maximum size 2GB of text data
Binary (n) – Fixed width binary string maximum size 8000 bytes
Varbinary- variable width binary string maximum size 8000 bytes
Binary Max- variable width binary string maximum size 2GB and
Image- variable width binary string 2GB maximum size.

CONSTRAINTS

Constraint or the rules that were applied to the type of the data in a table. That is we can specify the limit on the type of data that can be stored in a particular column in our table using constraints.
The available constraints in sql are:
NOT NULL: These constraints tell us that we can't store a null value in a column.
That is if a column is specified as not null then we will not be able to store a null in the column anymore.
Example: CREATE TABLE Student(ID int (6) NOT NULL, NAME varchar(10) NOT NULL, ADDRESS varchar(20));

UNIQUE

This constraint when specified with a column, tells that all the values in the column must be unique. That is the values in any row of a column must not be repeated.

Syntax:

CREATE TABLE Table_name (column_name1 DataType NOT NULL UNIQUE, column_name2 DataType NOT NULL, column_name3 DataType,……,Column_nameN DataType);

PRIMARY KEY:

A primary key is a field that can uniquely identify each row in a table. These constraint is used to specify a field in a table as the primary key.

Syntax : Create table table_ name(Column_name1 datatype NOT NULL, column_name2 datatype, column_nameN datatype, PRIMARY KEY(column_name1))

FOREIGN KEY:

A foreign key is a feel that can uniquely identify each row in another table these constraint is used to specify a field as a foreign key.

Syntax:

```
CREATE TABLE table_name1 (
   column_name1 datatype NOT NULL,
   column_name2 datatype,
   ...
   FOREIGN KEY (foreign_key_column) REFERENCES parent_table(parent_column)
);
```

CHECK:

These consent helps to validate the values of a column to meet a particular condition.

It helps to ensure that the value stored in a column meets a specific condition.

Syntax:

ALTER TABLE(table_name) ADD CONSTRAINT (constraint_name) CHECK (Boolean_expression).

DEFAULT:

These constraints specify a default value for the column when no value is specified by the user

Syntax:

ALTER TABLE(table_name) ADD CONSTRAINT (constraint_name) DEFAULT (default_value) FOR (existing_column_name).

Operators

An operator is a reserved word, Or a character used primarily in an sql statement to perform operations like comparisons and arithmetic operations.

- SQL Arithmetic operators.
- SQL Bitwise operators
- SQL Comparsion operators.
- SQL Compound Operators.

<u>SQL ARITHMETIC OPERATORS:</u>

Assume variable a holds 10 and variable B holds 20, then:

+ Addition- Adds value on a the site of the operator.

-Subtraction-subtracts right hand operand from the left hand operand.

* Multiplication- Multiplies value on either side of the operator.

/Division- Divides left hand operand by right hand operand.

%Modulus-divides left hand operand by right hand operand and returns reminder.

Another important in SQL is a comparison operator, which is used to compare one expression's value to other expressions. SQL supports different types of the comparison operator

COMPARISON OPERATOR:
= Equal to
> Greater Than
< Less Than
>= Greater than equal to
<= Less than equal to
<> Not equal

LOGICAL OPERATORS
The Logical operators are those that are true or false. They return true or false values to combine one or more true or false values.
AND Logical AND compares between two Booleans as Expression and returns true when both expressions are true
OR Logical OR compares between two Booleans as expressions and return true when one of the expressions is true.
NOT = Not takes a single Boolean as an argument and changes its value from false to true or from true to false.

SQL COMPOUND OPERATORS:
OR-The OR operator is used to combine multiple conditions in SQL Statements.
AND-The AND operator allows all the existence of multiple conditions in an SQL statement's.
NOT-The NOT operator is the used to negate the logical operator with which it is used.
BETWEEN-The BETWEEN operator is used to search for values that are within a set of values
IN-The IN operator is used to compare a value to a list of values that are specified.
LIKE-The LIKE operator is used to compare a value to similar values using wildcard operators
EXISTS-The EXISTS operator is used to search for the presence of a row in a specified table that meets certain criteria.
ALL-The ALL operator is used to compare a value to all values in another value set.
ANY-The ANY operator is used to compare a value to any applicable value in the list according to the condition.
UNIQUE-The UNIQUE operator searches every row of a specified table for uniqueness no duplicates.

SUB-QUERY
Following are a few rules that subqueries must follow:

Subqueries must be enclosed within parentheses.
A subquery can have only one column in the SELECT clause unless multiple columns are in the main query for the subquery to compare its selected columns.
An ORDER BY command cannot be used in a subquery, although the main query can use an ORDER BY. The GROUP BY command can be used to perform the same function as the ORDER BY in a subquery.
Subqueries that return more than one row can only be used with multiple value operators such as the IN operator.
The SELECT list cannot include any references to values that evaluate a BLOB, ARRAY, CLOB, or NCLOB.
A subquery cannot be immediately enclosed in a set function.

The BETWEEN operator cannot be used with a subquery. However, the BETWEEN operator can be used within the subquery.

SUBQUERY WITH SELECT STATEMENT:-

Subqueries are most frequently used with the SELECT statement. The basic syntax is as follows:

Syntax:- SELECT column_name [, column_name]
FROM table1 [, table2]
WHERE column_name OPERATOR
(SELECT column_name [, column_name]
FROM table1 [, table2][WHERE])

SUBQUERIES WITH THE INSERT STATEMENT:

INSERT statement can be used with subqueries. The INSERT statement uses the data returned from the subquery to insert into another table. The selected data in the subquery can be modified with any of the character, date, or number functions.

Here are the syntax and an example of subqueries using the INSERT statement.

Syntax:- INSERT INTO table_name [(column1 [,column2])]
SELECT [*|column1 [, column2] FROM table1 [, table2]
[WHERE VALUE OPERATOR]

SUBQUERIES WITH THE UPDATE STATEMENT:

The subquery can be used in conjunction with the UPDATE statement. Either single or multiple columns in a table can be updated when using a subquery with the UPDATE statement.

Here are the syntax and an example of subqueries using the INSERT statement.

Syntax:- UPDATE table SET column_name = new_value [
WHERE OPERATOR [VALUE] SELECT COLUMN_NAME FROM TABLE_NAME) TABLE NAME
[WHERE)]

SUBQUERIES WITH THE DELETE STATEMENT:

The subquery can be used in conjunction with the DELETE statement like with any other statements mentioned above.

Here are the syntax and an example of subqueries using the INSERT statement.

Syntax:- DELETE FROM TABLE_NAME
[WHERE OPERATOR [VALUE](SELECT COLUMN_NAME FROM TABLE_NAME)
[WHERE)]

JOINS

JOINS in SQL commands, are used to combine rows from two or more tables, based on a related column between tables. There are predominantly used when a user is trying to extract data from tables that have one-to-many or many-to-many relationships between them.

TYPES OF JOINS USED IN SQL:

INNER JOIN records that have matching values in both tables.
Example: select name, course, age from students inner join courses on name = name;
LEFT(OUTER) JOIN Returns all the records from the left table, and the matched records from the right table.
Example:

select students.name, age, course from students left join courses on students.name = courses.name where courses.name is null;

RIGHT (OUTER) JOIN Returns all records from the right table, and the matched records from the left table.

Example:

select students.name, age, courses.name, course from students right join courses on students.name = courses.name where students.name is null;

FULL (OUTER) JOIN Returns all records when there is a match in either left or right table.

Example:

select students.name, age, courses.name, course from students full join courses on students.name = courses.name where students.name is null or courses.name is null;

CROSS JOIN is used to combine each row of the first table with each row of the second table. It is also known as the Cartesian join since it returns the Cartesian product of the sets of rows from the joined tables.

Example:

select students.name, age, courses.name, course from students cross join courses;

VIEWS IN SQL

Views in SQL are considered as a virtual table. A view also contains rows and columns.

To create the view, we can select the fields from one or more tables present in the database.

A view can either have specific rows based on certain condition or all the rows of a table.

CREATING VIEW

A view can be created using the CREATE VIEW statement. We can create a view from a single table or multiple tables.

Syntax:

```
CREATE VIEW view_name AS
SELECT column1, column2…
FROM table_name
WHERE condition;
```

CREATING VIEW FROM A SINGLE TABLE

In this example, we create a view named DetailsView from the table Student_Detail

```
CREATE VIEW Detailsview AS
SELECT NAME, ADDRESS
FROM Student_Details
WHERE STU_ID < 4;
```

We can query the view to view the data

```
SELECT * FROM DetailsView;
```

CREATING VIEW FROM MULTIPLE TABLES

View from multiple tables can be created by simply include multiple tables in the SELECT statement. In the given example, a view is created named MarksView from two tables Student_Detail and Student_Marks.

```
CREATE VIEW MarksView AS
SELECT Student_Detail.NAME,Student_Detail.ADDRESS, Student_Marks.MARKS
FROM Student_Detail, Student_Mark
WHERE Student_Detail.NAME = Student_Marks.NAME;
SELECT * FROM MarksView;
```

DELETING VIEW

A view can be deleted using the Drop View statement.

SYNTAX:

DROP VIEW view_name;
Example:
If we want to delete the View MarksView, we can do this as:
DROP VIEW MarksView;

TRIGGER

Triggers in are SQL is a procedural code that is automatically executed in response to certain events on a specified table. It is a important to understand how these small codes make such a huge difference in database performance.
A trigger in SQL works similarly to a real-world trigger. For example, when the gun trigger is pulled a bullet is fired.
Note: There cannot be two triggers with similar action time and events for one table.
For example, we cannot have two BEFORE UPDATE triggers for a table.
But, we can have a BEFORE UPDATE and a BEFORE INSERT trigger, or a BEFORE UPDATE and an AFTER UPDATE trigger:
Application of Triggers:
Sharat is the marketing officer of a company.
As part of his daily activity, when a new customer data is entered into the company's database, a welcome message has to be sent to each of this new customer.

If it is one or two customers Sharat can do it manually, but if the count is more (a thousand) then the task becomes redundant.
Triggers come in handy in such a scenario.
A trigger can be created to automatically send a welcome email to the all-new customers once their data is entered into the database.
Syntax to create a trigger:-
create trigger Trigger_Name (Before | After) [Insert | Update | Delete] on [Table_Name] [for each row | for each column] [trigger_body]

CREATE TRIGGER
Two keywords are used to specify that a trigger block is going to be declared.

TRIGGER_NAME
It specifies the name of the trigger. Trigger names must be unique and shouldn't repeat.

(BEFORE | AFTER)

This specifies when the trigger will be executed. It tells us the time at which the trigger is initiated, i.e, either before the ongoing event or after.

Before Triggers are used to update or validate record values before they're saved to the database.

After Triggers are used to access field values that are set by the system and to effect changes in other records. The records that activate the after trigger are read-only. We cannot use the After trigger if we want to update a record because it will lead to a read-only error.

[INSERT | UPDATE | DELETE]

These are the DML operations, and we can use either of them in a given trigger.

ON [TABLE_NAME]

We need to mention the table name on which the trigger is being applied. Don't forget to use one keyword and make sure the selected table is present in the database.

[FOR EACH ROW | FOR EACH COLUMN]
Row-level trigger gets executed before or after any column value of a row changes Column Level Trigger gets executed before or after the specified column changes.

[TRIGGER_BODY]
It consists of queries that need to be executed when the trigger is called.

We can perform many operations using triggers. Following are a few examples:

DROP A Trigger

DROP TRIGGER trigger name;

Display A Trigger

The below code will display all the triggers that are present.
SHOW TRIGGERS;
The below code will display all the triggers that are present in a particular database

SHOW TRIGGERS IN database_name;
Example:- SHOW TRIGGERS IN 360digiTMG;

EXAMPLE:-

As we have already understood how to create a trigger, now let's understand the two variants of the trigger those are before inserted and After insert.

Let's create a student table with various columns as shown below:

CREATE TABLE Student(INT NOT NULL AUTO_INCREMENT, FName VARCHAR(20), LName VARCHAR(20), Address VARCHAR(30), City VARCHAR(15), Marks INT, PRIMARY KEY(studentID));

Operations in Triggers:
The first variant i.e., Before Insert: Here when we insert data into the table, the total mark trigger will store the result in the Final mark table.

Syntax:- CREATE TRIGGER calculate before INSERT ON student FOR EACH ROW SET new.marks = new.marks+100;

Advantages:

Forcing security approvals on the table that are present in the database.
Triggers provide another way to check the integrity of data.
Counteracting invalid exchanges.

Triggers handle errors from the database layer.
Normally triggers can be useful for inspecting the data changes in tables.
Triggers give an alternative way to run scheduled tasks. Using triggers, we don't have to wait for the scheduled events to run because the triggers are invoked automatically before or after a change is made to the data in a table.

Disadvantages:
Triggers can only provide extended validations, i.e., not all kinds of validations. For simple validations, you can use the NOT NULL, UNIQUE, CHECK, and FOREIGN KEY constraints.
Triggers may increase the overhead of the database.
Triggers can be difficult to troubleshoot because they execute automatically in the database, which may not be invisible to the client applications.

UNION AND UNION ALL

A union is used for extracting rows using the conditions specified in the query while Union All is used for extracting all the rows from a set of two tables.

Syntax:
Query1 UNION query2

SELECT name, department FROM employees1
UNION
SELECT name, department FROM employees2;

Example UNION ALL

SELECT name, department FROM employees1
UNION ALL
SELECT name, department FROM employees2;

Condition for Union and Union all

In order to use UNION and UNION ALL to combine the results of two or more SELECT statements, there are a few conditions that must be met:
The SELECT statements must have the same number of columns.
The data types of the corresponding columns from each SELECT statement must be compatible or convertible.
The column names in the final result set will be taken from the first SELECT statement, unless aliases are used to specify different names.