Introduction to SQL - DML

SELECT Query

SELECT query is used to retrieve data from a table. It is the most used SQL query. We can retrieve complete table data, or partial by specifying conditions using the WHERE clause.

Syntax

```
SELECT
        column_name1,
        column_name2,
        column_name3,
        ...
        column_nameN
FROM
        table_name;
```

Example

```
SELECT
        name, age
FROM
        students;
```

SELECT Query

Student_id	Name	Age	City
1	Arun	17	Chennai
2	Ashok	18	Bangalore
3	Vikas	16	Pune
4	Sekar	20	Mumbai

Let us Create a Student Table with above columns and insert the rows into the table.

To Select ALL Columns

```
SELECT *
FROM
    student;
```

To Select few Columns

```
SELECT
    name, age
FROM
    student;
```

WHERE Clause

WHERE Clause is used to filter or select certain data based on some condition.

Syntax

```
SELECT
    column_name1,
    column_name2
FROM
    table_name
WHERE
    [condition];
```

Example

```
SELECT
    name, age
FROM
    student
WHERE
    age = 16;
```

WHERE Clause

Operators for WHERE clause condition
Following is a list of operators that can be used while specifying the WHERE clause condition.

Operator	Description
=	Equal to
!=	Not Equal to
<	Less than
>	Greater than
<=	Less than or Equal to
>=	Greate than or Equal to
BETWEEN	Between a specified range of values
LIKE	This is used to search for a pattern in value.
IN	In a given set of values

DISTINCT Clause

DISTINCT Clause is used to remove the duplicates from the data while retrieving the data from the table.

Syntax

```
SELECT  DISTINCT column-list
FROM    table-name
ORDER BY ASC | DESC;
```

Example

```
SELECT DISTINCT age
FROM    student
ORDER BY age DESC:

SELECT DISTINCT salary
FROM employee;
```

LIKE Clause

LIKE clause is used in the condition in SQL query with the WHERE clause. LIKE clause compares data with an expression using wildcard operators to match pattern given in the condition.

Wildcard operators

There are two wildcard operators that are used in LIKE clause.

- Percent sign %: represents zero, one or more than one character.
- Underscore sign _: represents only a single character.

Example with %

```
SELECT
    name, age
FROM
    student
WHERE
    name LIKE “A%”;
```

LIKE Clause

Example with % and _

```
SELECT
    name, age
FROM
    student
WHERE
    name LIKE "_s%";
```

Will retrieve the data with name has second character 's' irrespective of First Character.

```
SELECT
    name, age
FROM
    student
WHERE
    name LIKE "%n";
```

Will retrieve the data with name has last character 'n'

ORDER BY Clause

ORDER BY is used to display the data in a sorted order based on certain attributes. It has ASC and DESC orders. The ASC and DESC can be mixed in a single query for different columns.

Syntax

```
SELECT  column-list|*
FROM    table-name
ORDER BY ASC | DESC;
```

Example

```
SELECT * from student
ORDER BY name ASC;

SELECT * from student
ORDER BY name ASC, age DESC;
```

GROUP BY Clause

Group by clause is used to group the results of a SELECT query based on one or more columns. It is also used with SQL functions to group the result from one or more tables.

Syntax

```
SELECT   column_name, function(column_name)
FROM     table_name
WHERE    condition
GROUP BY column_name;
```

Example

```
SELECT name , age
 FROM    employee
GROUP BY salary;

SELECT name, age FROM student
WHERE   age > 25
GROUP BY salary;
```

Employee
Emp_id
Emp_name
age
City
Salary

HAVING Clause

HAVING clause is used with SQL Queries to give more precise condition for a statement. It is used to mention condition in Group by based SQL queries, just like WHERE clause is used with SELECT query.

Sales
Order_id
Order_dt
Customer_name
Previous_bal

Syntax

```
SELECT column_name, function(column_
FROM t  able_name
WHERE  column_name condition
GROUP BY column_name
HAVING       function(column_name) cor
```

Example

```
SELECT  customer_name, sum(previous_bal)
FROM    sales
GROUP BY customer_name
HAVING sum(previous_bal) > 3000;
```

SQL Joins

SQL Join is used to fetch data from two or more tables, which is joined to appear as single set of data. It is used for combining column from two or more tables by using values common to both tables.

JOIN Keyword is used in SQL queries for joining two or more tables. Minimum required condition for joining table, is (n-1) where n, is number of tables. A table can also join to itself, which is known as, **Self Join**.

Types of JOIN

Following are the types of JOIN that can be used in SELECT statements.

- Inner
- Outer
- Left
- Right

SQL Joins

SQL Join is used to fetch data from two or more tables, which is joined to appear as single set of data. It is used for combining column from two or more tables by using values common to both tables.

JOIN Keyword is used in SQL queries for joining two or more tables. Minimum required condition for joining table, is (n-1) where n, is number of tables. A table can also join to itself, which is known as, **Self Join**.

Types of JOIN

Following are the types of JOIN that can be used in SELECT statements.

- Inner
- Outer
- Left
- Right

SQL Joins

Cross JOIN or Cartesian Product : This type of JOIN returns the cartesian product of rows from the tables in Join. It will return a table which consists of records which combines each row from the first table with each row of the second table.

Example

```
SELECT * FROM subject CROSS JOIN hobbies;
```

INNER Join or EQUI Join: This is a simple JOIN in which the result is based on matched data as per the equality condition specified in the SQL query.

Example

```
select    s.stu_name, sub.subject_name, s.location
from      student s
inner join student_subject ss on s.student_id = ss.student_id
inner join subject sub on sub.subject_id = ss.subject_id;
```

SQL Joins

LEFT Outer Join : The left outer join returns a result set table with the matched data from the two tables and then the remaining rows of the left table and null from the right table's columns.

Example

```
select      sub.subject_name, s.stu_name, s.location
from       subject sub
left outer join student_subject ss on ss.subject_id = sub.subject_id
left outer join student s on s.student_id = ss.student_id ;
```

subject_name	stu_name	location
DBMS	Shanmugam	Bangalore
Java	Shanmugam	Bangalore
Python	Shanmugam	Bangalore
Web Design	NULL	NULL

SQL Joins

RIGHT Outer Join : The right outer join returns a result set table with the matched data from the two tables being joined, then the remaining rows of the right table and null for the remaining left table's columns.

Example

```
select      s.stu_name, h.hobby_name
from student_hobbies sh
right outer join student s on s.student_id = sh.student_id
right outer join hobbies h on h.hobby_id = sh.hobby_id;
```

NULL	Athletics
Shanmugam	Cricket
Shanmugam	Hockey
NULL	Tennis
NULL	Volley Ball

SQL Joins

Natural JOIN : Natural Join is a type of Inner join which is based on column having same name and same datatype present in both the tables to be joined. It helps to eliminate redundant columns.

Example

```
select student_id, stu_name, subject_id
from student natural join student_subject;
```

Alternate way

```
select * from student join student_subject using (student_id);
```

1	Shanmugam	1
1	Shanmugam	2
1	Shanmugam	3

SQL Joins

STRAIGHT_JOIN scans and combines matching rows (if specified any condition) which are stored in associated tables otherwise it behaves like an INNER JOIN or JOIN of without any condition.

STRAIGHT_JOIN is similar to JOIN, except that the left table is always read before the right table. This can be used for those (few) cases for which the join optimizer puts the tables in the wrong order.

Example

```
select * from student straight_join student_hobbies;
```

SET Operations

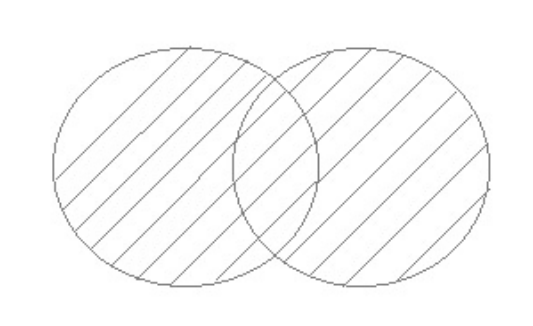
SQL standard supports few Set operations which can be performed on the table data. These are used to get meaningful results from data stored in the table, under different special conditions. They are,

- UNION
- UNION ALL
- INTERSECT
- MINUS

But, MySQL only support UNION, UNION ALL and UNION DISTINCT operations. It doesn't support INTERSECT and MINUS Operations. But, it can be emulated using indirect way. We can see through examples.

SET Operations

UNION is used to combine the results of two or more SELECT statements. However, it will eliminate duplicate rows from its result set. In case of union, number of columns and datatype must be same in both the tables, on which UNION operation is being applied.



Example

```
SELECT * from t1
UNION
SELECT * from t2;
```

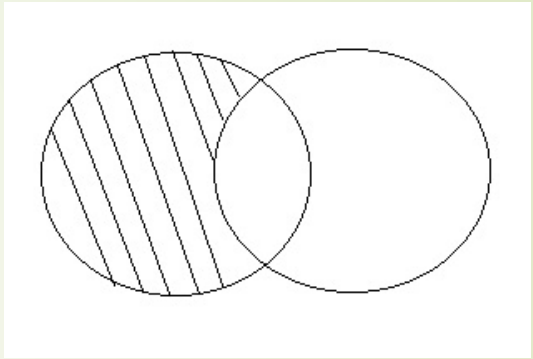
Just UNION operation will bring the rows from both tables and eliminate any duplicate data.

```
SELECT * from t1
UNION ALL
SELECT * from t2;
```

UNION ALL operation will bring all the rows from both tables irrespective of any duplicate data.

SET Operations

INTERSECT operation is used to combine two SELECT statements, but it only retuns the records which are common from both SELECT statements. In case of Intersect the number of columns and datatype must be same.

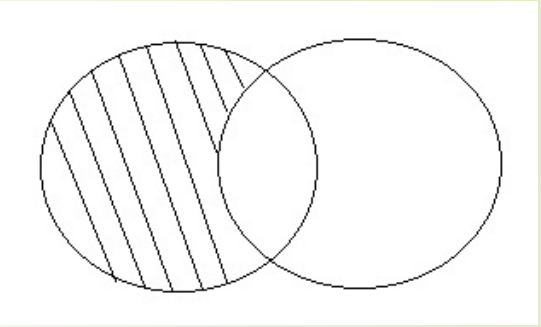


There is NO INTERSECT operation in MySQL. But, the same can be achieved using DISTINCT and INNER JOIN operations.

Example

```
select DISTINCT * from t1
inner join t2 on t2.t2_id = t1.t1_id;
```

SET Operations



The MINUS operation combines results of two SELECT statements and return only those in the final result, which belongs to the first set of the result.

There is NO MINUS operation in MySQL. But, the same can be achieved using NOT EXISTS Clause from SELECT Query.

Example

```
## Emulating Minus
select * from t1
where not exists
      (select 1 from t2
      where t2.t2_id = t1.t1_id);
```

FUNCTIONS

MySQL provides many built-in functions to perform operations on data. These functions are useful while performing mathematical calculations, string concatenations, sub-strings etc.

Some of the built-in functions are,

- Arithmetic Functions
- String Functions
- Date and Time Functions
- Aggregate Functions, etc

The complete list of functions are available in MySQL Manual which can be accessed using the following link.

https://dev.mysql.com/doc/refman/8.0/en/functions.html

FUNCTIONS

MySQL provides many built-in functions to perform operations on data. These functions are useful while performing mathematical calculations, string concatenations, sub-strings etc.

Some of the built-in functions are,

- Mathematical/Numerical Functions
- String Functions
- Date and Time Functions
- Aggregate Functions, etc

The complete list of functions are available in MySQL Manual which can be accessed using the following link.

https://dev.mysql.com/doc/refman/8.0/en/functions.html

FUNCTIONS

The Aggregate functions are mostly used in SQL Query when using the GROUP BY Clause. Some of the popular AGGREGATE Functions are,

min(), max(), avg(), count(*), sum(), etc

The list of all aggregate functions are available in the following link.

https://dev.mysql.com/doc/refman/8.0/en/aggregate-functions.html

INSERT Statement

INSERT command

Insert command is used to insert data into a table. Following is its general syntax,

```
INSERT INTO table_name VALUES(data1, data2, ...);
```

Example

Inserting values to all the columns in a table

```
INSERT INTO Subject values (1, 'Mathematics');
```

Inserting values to some columns in a table

```
INSERT INTO student_subject (student_id, subject_id, marks) values (1,1, 89);
```

INSERT Statement

INSERTING Multiple rows in a single insert statement using ROW()

```
create table t1 (t1_id int, t1_name varchar(30));
create table t2 (t2_id int, t2_name varchar(30));
insert into t1 values ROW(1, 'aa'), ROW(2,'bb'), ROW(3,'cc’);
insert into t2 values ROW(2,'cccc'),ROW(3,'dddd'), ROW(4,'eeee’);
```

INSERTING rows into a table using SELECT query

The columns and data type should match with target table in the SELECT Query of the source table.

```
insert into t1
select * from t2;
```

UPDATE Statement

UPDATE command

UPDATE command is used to update any record of data in a table. Following is its general syntax,

```
UPDATE table_name SET
    column_name = new_value
WHERE some_condition;
```

Example

```
update student set
    stu_name = 'Ajay'
where stu_id = 123
```

UPDATE Statement

UPDATE Multiple values

```
update student set
        stu_name = 'Ajay',
        age = 21
where stu_id = 123;
```

Updating all the values of a column

```
update student set
        age = age + 1;
```

Updating a value from another table

```
UPDATE items, month
        SET items.price=month.price
WHERE items.id=month.id and month.month=04;
```

UPDATE Statement

UPDATE using Select Statement

```
CREATE TABLE items (
        id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
        wholesale DECIMAL(6,2) NOT NULL DEFAULT 0.00,
        retail DECIMAL(6,2) NOT NULL DEFAULT 0.00,
        quantity INT NOT NULL DEFAULT 0 );

UPDATE items,
        (SELECT id FROM items
        WHERE id IN (
                SELECT id FROM items
                WHERE retail / wholesale >= 1.3 AND quantity < 100)) AS discounted
SET items.retail = items.retail * 0.9
WHERE items.id = discounted.id;
```

UPDATE Statement

The same update statement can also be written using the following query.

```
UPDATE items,
    (SELECT id, retail / wholesale AS markup, quantity FROM items) AS discounted
SET   items.retail = items.retail * 0.9
WHERE    discounted.markup >= 1.3 AND
         discounted.quantity < 100 AND
         items.id = discounted.id;
```

DELETE Statement

DELETE statement/command is used to delete all the rows or a particular row from a table.

The general syntax is,

DELETE from Table_name

WHERE Condition;

Example

Delete a particular row(s) from a table

delete from student

student_id = 123;

Delete all the rows from a table

delete from student;

DELETE Statement

Deleting data from multiple tables using some other table

```
DELETE t1, t2 FROM t1 INNER JOIN t2 INNER JOIN t3
WHERE t1.id=t2.id AND t2.id=t3.id;
## The same query can be written as follows:
DELETE FROM t1, t2 USING t1 INNER JOIN t2 INNER JOIN t3
WHERE t1.id=t2.id AND t2.id=t3.id;
```

VIEW

VIEWS are virtual tables that do not store any data of their own but display data stored in other tables. In other words, VIEWS are nothing but SQL Queries. A view can contain all or a few rows from a table. A MySQL view can show data from one table or many tables.

Syntax

```
CREATE VIEW `view_name` AS SELECT statement;
```

Example

```
CREATE VIEW stu_sub_view AS
    SELECT s.stu_name, sub.subject_name, ss.marks, ss.grade
    FROM student s
    inner join student_subject ss using(student_id)
    inner join subject sub using (subject_id)
```

VIEW

To Drop a view:

```
DROP VIEW <View_name>;
```

- VIEWS increase re-usability. You will not have to create complex queries involving joins repeatedly. All the complexity is converted into a single line of query use VIEWS. Such condensed code will be easier to integrate in your application. This will eliminate chances of typos and your code will be more readable.
- VIEWS help in data security. You can use views to show only authorized information to users and hide sensitive data like credit card numbers.

TRIGGERS

A TRIGGER is a stored program invoked automatically in response to an event such as **insert, update, or delete** that occurs in the associated table. For example, you can define a trigger that is invoked automatically before a new row is inserted into a table.

MySQL supports triggers that are invoked in response to the INSERT, UPDATE or DELETE event.

Advantages of triggers

- Triggers provide another way to check the integrity of data.
- Triggers handle errors from the database layer.
- Triggers can be useful for auditing the data changes in tables.

Disadvantages of triggers

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

TRIGGERS

A TRIGGER is a stored program invoked automatically in response to an event such as **insert, update, or delete** that occurs in the associated table. For example, you can define a trigger that is invoked automatically before a new row is inserted into a table.

MySQL supports triggers that are invoked in response to the INSERT, UPDATE or DELETE event. Triggers can be created BEFORE and AFTER of any of INSERT, UPDATE and DELETE event.

To list the triggers available in the database, SHOW TRIGGERS command can be used.

Example

```
SHOW TRIGGERS from <Database_name>;

## To list the triggers for a specic table, we need to use WHERE clause

SHOW Triggers from database_name

WHERE table = Table_name;
```

TRIGGERS

Advantages of triggers

- Triggers provide another way to check the integrity of data.
- Triggers handle errors from the database layer.
- Triggers can be useful for auditing the data changes in tables.

Disadvantages of triggers

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

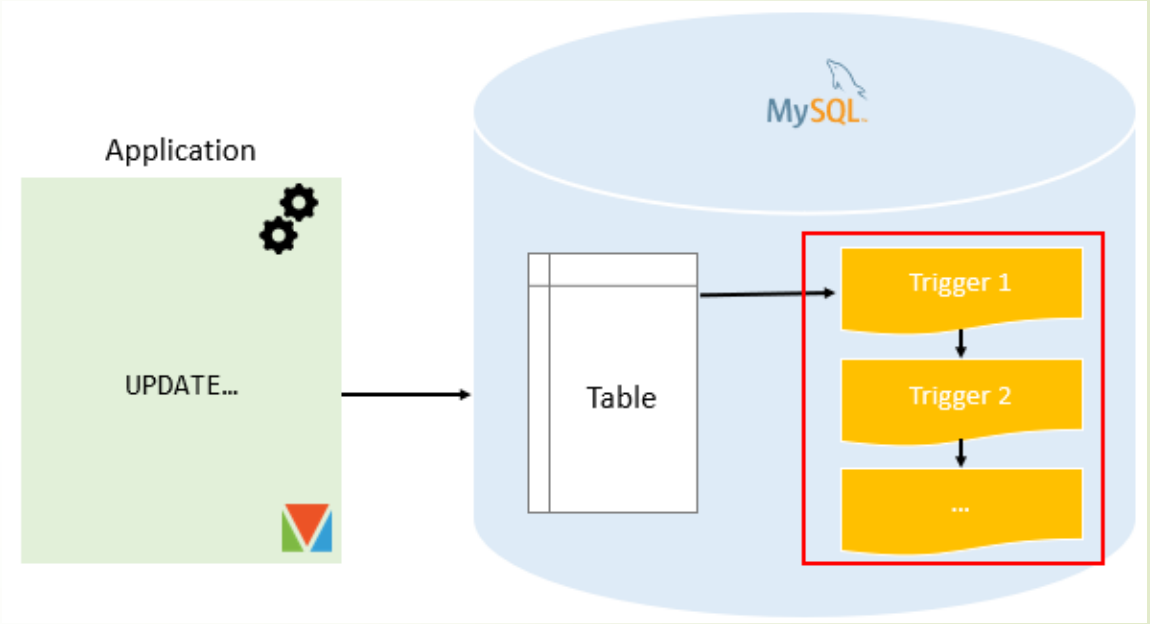
TRIGGERS

MySQL allows to create multiple triggers on a table.

```
DELIMITER $$

CREATE TRIGGER trigger_name
{BEFORE|AFTER}{INSERT|UPDATE|DELETE}
ON table_name FOR EACH ROW
{FOLLOWS|PRECEDES} existing_trigger_name
BEGIN
   -- statements
END$$

DELIMITER ;
```



STORED PROCEDURE

A **stored procedure** is a segment of declarative SQL statements stored inside the MySQL Server.

After the stored procedure is created, it can be invoked by using the CALL statement:

```
CALL procedure_name;
```

And the procedure executes the statements inside the procedure.

- When a stored procedure is invoked first time, MySQL looks up for the name in the database catalog, compiles the stored procedure's code, place it in a memory area known as a cache, and execute the stored procedure.
- If the same stored procedure invoked again in the same session again, MySQL just executes the stored procedure from the cache without having to recompile it.

STORED PROCEDURE

- A stored procedure may contain control flow statements such as IF, CASE, and LOOP that allow you to implement the code in the procedural way.
- A stored procedure can call other stored procedures or stored functions, which allows you to modularize your code.

Advantages of Stored Procedures

Reduce network traffic: Stored procedures help reduce the network traffic between applications and MySQL Server. Because instead of sending multiple lengthy SQL statements, applications have to send only the name and parameters of stored procedures.

Centralize business logic in the database: You can use the stored procedures to implement business logic that is reusable by multiple applications. The stored procedures help reduce the efforts of duplicating the same logic in many applications and make your database more consistent.

Make database more secure: The database administrator can grant appropriate privileges to applications that only access specific stored procedures without giving any privileges on the underlying tables.

STORED PROCEDURE – Handling Exceptions

Handling Exceptions

When an error occurs inside a stored procedure, it is important to handle it appropriately, such as continuing or exiting the current code block's execution and issuing a meaningful error message.

MySQL provides an easy way to define handlers that handle from general conditions such as warnings or exceptions to specific conditions e.g., specific error codes.

Declaring a handler

To declare a handler, you use the DECLARE HANDLER statement as follows:

DECLARE action HANDLER FOR condition_value statement;

If a condition whose value matches the condition_value , MySQL will execute the statement and continue or exit the current code block based on the action .

The action accepts one of the following values:

- **CONTINUE :** the execution of the enclosing code block (BEGIN ... END) continues**.**
- **EXIT :** the execution of the enclosing code block, where the handler is declared, terminates.

Cursors

To handle a result set inside a stored procedure, you use a cursor. A cursor allows you to iterate a set of rows returned by a query and process each row individually.

Working with MySQL cursor

The sequence of steps to create and use the cursor

```
DECLARE cursor_name CURSOR FOR SELECT_statement;

OPEN cursor_name;

FETCH cursor_name INTO variables list;

 ## All the operations or SQL Statements to process the fetched row.

CLOSE cursor_name;
```


thank
you