SQL Commands

SQL Languages:

1. DDL (Data Definition Language) -> Create, Alter, Drop, Truncate
2. DML (Data Manipulation Language) -> Insert, Update, Delete
3. DCL (Data Control Language) -> Grant, Revoke
4. TCL (Transaction Control Language) ->commit, rollback, savepoint
5. DQL (Data Query Language) -> select

**Data Types**

**Numeric Data Types**

| **Data Type** | **Description** | **Example** |
| --- | --- | --- |
| **INT** | Standard integer type. Holds whole numbers. Commonly used for IDs or counts. Range: -2,147,483,648 to 2,147,483,647. | INT(11) - 12345 |
| **TINYINT** | Very small integer. Often used for boolean or small-range values. Range: -128 to 127. | TINYINT(4) - 127 |
| **SMALLINT** | Small integer. Suitable for small-range values. Range: -32,768 to 32,767. | SMALLINT(6) - 32767 |
| **MEDIUMINT** | Medium-sized integer. Fits between SMALLINT and INT. Range: -8,388,608 to 8,388,607. | MEDIUMINT(9) - 8388607 |
| **BIGINT** | Large integer. Used for very large numbers. Range: -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807. | BIGINT(20) - 9223372036854775807 |
| **FLOAT** | Floating-point number. Used for single-precision floating-point numbers. Approximate numerical value with a floating decimal point. | FLOAT(7,4) - 123.4567 |
| **DOUBLE** | Double precision floating-point number. More precision than FLOAT. Suitable for precise calculations. | DOUBLE(16,8) - 123.45678901 |
| **DECIMAL** | Exact numeric type. Used for fixed-point arithmetic. Good for monetary values to avoid rounding errors. | DECIMAL(10,2) - 12345.67 |

**Date and Time Data Types**

| **Data Type** | **Description** | **Example** |
| --- | --- | --- |
| **DATE** | Stores dates in 'YYYY-MM-DD' format. Useful for representing calendar dates. | DATE - 2024-07-30 |
| **DATETIME** | Stores date and time in 'YYYY-MM-DD HH:MM  ' format. Used for exact timestamps. | DATETIME - 2024-07-30 14:30:00 |
| **TIMESTAMP** | Stores date and time as the number of seconds since the Unix epoch ('1970-01-01 00:00:00' UTC). Automatically updates to the current time when records change. | TIMESTAMP - 2024-07-30 14:30:00 |
| **TIME** | Stores time in 'HH:MM  ' format. Useful for representing time of day or duration. | TIME - 14:30:00 |
| **YEAR** | Stores a year in 4-digit format. Useful for representing year values. | YEAR - 2024 |

**String Data Types**

| **Data Type** | **Description** | **Example** |
| --- | --- | --- |
| **CHAR** | Fixed-length string. Pads with spaces to meet the specified length. Efficient for small and fixed-length data. | CHAR(10) - 'hello' |
| **VARCHAR** | Variable-length string. More flexible than CHAR, no padding with spaces. Efficient for variable-length data. | VARCHAR(255) - 'hello world' |
| **TINYTEXT** | Very small text. Limited to 255 characters. Suitable for short texts. | TINYTEXT - 'small text' |
| **TEXT** | Small text. Can store up to 65,535 characters. Suitable for moderate-sized texts. | TEXT - 'some text data' |
| **MEDIUMTEXT** | Medium-length text. Can store up to 16,777,215 characters. Suitable for larger texts. | MEDIUMTEXT - 'medium length text' |
| **LONGTEXT** | Large text. Can store up to 4,294,967,295 characters. Suitable for very large texts like documents. | LONGTEXT - 'very large text data' |
| **BINARY** | Fixed-length binary data. Similar to CHAR but for binary data. Efficient for fixed-length binary storage. | BINARY(10) - b'hello' |
| **VARBINARY** | Variable-length binary data. Similar to VARCHAR but for binary data. Efficient for variable-length binary storage. | VARBINARY(255) - b'hello world' |
| **TINYBLOB** | Very small BLOB (Binary Large Object). Stores up to 255 bytes. Suitable for small binary data. | TINYBLOB - binary data |
| **BLOB** | Small BLOB. Can store up to 65,535 bytes. Suitable for moderate-sized binary data. | BLOB - binary data |
| **MEDIUMBLOB** | Medium-sized BLOB. Can store up to 16,777,215 bytes. Suitable for larger binary data. | MEDIUMBLOB - binary data |
| **LONGBLOB** | Large BLOB. Can store up to 4,294,967,295 bytes. Suitable for very large binary data. | LONGBLOB - binary data |

**Spatial Data Types**

| **Data Type** | **Description** | **Example** |
| --- | --- | --- |
| **GEOMETRY** | Stores any type of spatial data. General spatial data type that can hold any geometry value. | GEOMETRY - spatial data |
| **POINT** | Stores a point in 2D space. Suitable for representing a single location. | POINT - (1 1) |
| **LINESTRING** | Stores a series of points in a line. Suitable for representing paths or line segments. | LINESTRING - (0 0,1 1) |
| **POLYGON** | Stores a polygon. Suitable for representing areas bounded by a series of points. | POLYGON - ((0 0,1 1,1 0,0 0)) |
| **MULTIPOINT** | Stores multiple points. Suitable for representing multiple locations. | MULTIPOINT - (1 1,2 2) |
| **MULTILINESTRING** | Stores multiple lines. Suitable for representing multiple paths or line segments. | MULTILINESTRING - ((0 0,1 1),(2 2,3 3)) |
| **MULTIPOLYGON** | Stores multiple polygons. Suitable for representing multiple areas. | MULTIPOLYGON - (((0 0,1 1,1 0,0 0)),((1 1,2 2,2 1,1 1))) |
| **GEOMETRYCOLLECTION** | Stores a collection of geometry objects. Suitable for representing a mixture of different spatial data types in one column. | GEOMETRYCOLLECTION - GEOMETRYCOLLECTION(POINT(1 1),LINESTRING(0 0,1 1)) |

Constraints

**MySQL Constraints**

| **Constraint** | **Description** | **Example** |
| --- | --- | --- |
| **NOT NULL** | Ensures that a column cannot have a NULL value. | CREATE TABLE Students (ID INT NOT NULL, Name VARCHAR(50) NOT NULL); |
| **UNIQUE** | Ensures that all values in a column are unique. | CREATE TABLE Users (ID INT, Email VARCHAR(100) UNIQUE); |
| **PRIMARY KEY** | Uniquely identifies each record in a table. Combines NOT NULL and UNIQUE. | CREATE TABLE Employees (ID INT PRIMARY KEY, Name VARCHAR(50)); |
| **FOREIGN KEY** | Ensures referential integrity by linking a column to the primary key of another table. | CREATE TABLE Orders (OrderID INT, CustomerID INT, FOREIGN KEY (CustomerID) REFERENCES Customers(ID)); |
| **CHECK** | Ensures that all values in a column satisfy a specific condition. | CREATE TABLE Products (ID INT, Price DECIMAL(10, 2), CHECK (Price >= 0)); |
| **DEFAULT** | Sets a default value for a column when no value is specified. | CREATE TABLE Orders (OrderID INT, OrderDate DATE DEFAULT CURRENT\_DATE); |
| **AUTO\_INCREMENT** | Automatically generates a unique value for each new record in a numeric primary key column. | CREATE TABLE Items (ItemID INT AUTO\_INCREMENT PRIMARY KEY, ItemName VARCHAR(50)); |
| **INDEX** | Improves the speed of data retrieval operations on a table by creating an index on specified columns. | CREATE TABLE Books (ID INT, Title VARCHAR(100), INDEX (Title)); |

**Detailed Examples**

1. **NOT NULL Constraint**: Ensures a column cannot have NULL values.

sql

Copy code

CREATE TABLE Students (

ID INT NOT NULL,

Name VARCHAR(50) NOT NULL

);

1. **UNIQUE Constraint**: Ensures all values in a column are unique.

sql

Copy code

CREATE TABLE Users (

ID INT,

Email VARCHAR(100) UNIQUE

);

1. **PRIMARY KEY Constraint**: Uniquely identifies each record in a table. Combines NOT NULL and UNIQUE.

sql

Copy code

CREATE TABLE Employees (

ID INT PRIMARY KEY,

Name VARCHAR(50)

);

1. **FOREIGN KEY Constraint**: Ensures referential integrity by linking a column to the primary key of another table.

sql

Copy code

CREATE TABLE Orders (

OrderID INT,

CustomerID INT,

FOREIGN KEY (CustomerID) REFERENCES Customers(ID)

);

1. **CHECK Constraint**: Ensures all values in a column satisfy a specific condition.

sql

Copy code

CREATE TABLE Products (

ID INT,

Price DECIMAL(10, 2),

CHECK (Price >= 0)

);

1. **DEFAULT Constraint**: Sets a default value for a column when no value is specified.

sql

Copy code

CREATE TABLE Orders (

OrderID INT,

OrderDate DATE DEFAULT CURRENT\_DATE

);

1. **AUTO\_INCREMENT Constraint**: Automatically generates a unique value for each new record in a numeric primary key column.

sql

Copy code

CREATE TABLE Items (

ItemID INT AUTO\_INCREMENT PRIMARY KEY,

ItemName VARCHAR(50)

);

1. **INDEX Constraint**: Improves the speed of data retrieval operations on a table by creating an index on specified columns.

sql

Copy code

CREATE TABLE Books (

ID INT,

Title VARCHAR(100),

INDEX (Title)

);

**Data Definition Language (DDL)**

**To clear mysql command line client**

**mysql> \! cls**

Create Database

* show databases;
* create database <database name>
* use <database name>
* select database (); -> to select working database
* desc <table name> -> to get table details
* show tables;
* SHOW CREATE TABLE orders;
  + This command will show the entire CREATE TABLE statement, including foreign key constraints.
* SHOW TABLE STATUS LIKE 'orders';
* select constraint\_name, column\_name,referenced\_table\_name from information\_schema.key\_column\_usage where table\_name='orders';

**Delete Database**

* drop database <database name>

**Alter Table Add Column**

Alter Table

alter table employees

add column

salary int not null

default 25000;

**Alter Table Drop column**

alter table contact

drop column name;

**Alter Table Rename Column**

Alter table contacts

Rename column to new name;

alter table contact

rename column name to cust\_name;

**Alter Table Rename Table**

Rename table tablename to newtablename

Rename table contact to vi;

**Alter table to add constraints**

alter table orders

add constraint foreign key (cust\_id)

references customers(cust\_id);

**Alter table to add cascade on existing relationship**

* alter table orders drop constraint orders\_ibfk\_1;
* Using alter -> alter table orders add constraint foreign key (cust\_id) references customers(cust\_id) on delete cascade ;

**Modifying a Column (alter - modify)**

Changing data type

* alter table vi modify column mob varchar(15) default 'unknwon';

**Data Manipulation Language (DML)**

Create Table (Oracle)

* create table <tablename> (

att1 datatype(length) constraint,

att2 datatype(length) constraint,

constraint pk primary key (att1),

constraint check\_val check (val>0)

);

**Example (mysql)**

create table student (

stud\_id int,

stud\_name varchar(100),

);

**Creating a table with constraints**

create table account (

id int primary key auto\_increment not null,

name varchar (20) not null,

acc\_type varchar (20) not null default 'saving'

);

Insert Data

**insert into student values(4,'raj','mumbai');**

**insert into student (stud\_id,stud\_name) values(3,'sunny')**

**insert into student values(4,'raj','mumbai'),(5,'manish','thane');**

updating Data

update student

set name=”xxx”

where name=”xyxy”;

deleting a data

delete from student where name=”xxx”;

**String Functions**

**­­­­­­­­­­­­**

**Concat(‘ ‘ ,’ ‘ ); -> to concate to string**

Ex -> select employees.\* , concat(emp\_name,' ',emp\_desig) as name\_desig from employees;

**Concat\_WS(‘-‘,’ ’,’ ’); width separator -> combine words with seperator**

Ex -> select employees.\* , concat\_WS('->',emp\_name,emp\_desig)

as name\_desig from employees;

**substring(string,beg,end);**

**ex -> select employees.\*, substring(emp\_id,2) as short from employees;**

**replace(string, from string , to string)**

ex -> select employees.\*,replace(emp\_desig,"associate","trainee") as emp\_desig from employees;

reverse(string)

ex -> select employees.\*, reverse(emp\_id) from employees;

upper(string) and lower(string)

ex -> select employees.\*, ucase(emp\_name) from employees;

ex -> select employees.\*, upper(emp\_name) from employees;

char\_length(String)

ex -> select employees.\*, char\_length(emp\_name) from employees;

ex -> select employees.\*, char\_length(emp\_name) as name\_length from

employees where char\_length(emp\_name)>4;

insert(‘hello sbcjd’,index,shift,’raj’);

ex -> select employees.\*, insert(emp\_name,1,0,"mr.") as emp\_new from employees;

left(‘wvchja’,3); to print characters from left side

ex -> select employees.\*, left(emp\_id,2) as new from employees;

right(‘abcdef’,4); to print characters from right side

select employees.\*, right(emp\_id,2) as new from employees;

repeat(‘o’,5); repeats mention characters

select employees.\*, repeat(emp\_id,2) as new from employees;

trim(‘ string! ’); to removes tailing spaces from string

ex -> select employees.\*, trim(emp\_desig) as new from employees;

**Some more example**

* **select employees.\*, concat(left(emp\_dept,1),emp\_id) as newTable from employees limit 2;**
* **select employees.\*, concat\_WS(":",emp\_id,concat\_ws(' ',emp\_name,"rasto**

**gi"),emp\_desig) as new from employees where (emp\_name='raju');**

* **select employees.\*, concat\_WS(":",emp\_id,emp\_name,upper(emp\_desig),emp**

**\_dept) as new from employees where (emp\_name='raju');**

* **select employees.\*, concat(left(emp\_dept,1),emp\_id) as newTable from employees limit 2;**

**DISTINCT**

**To retrive the values without duplicates**

**Ex -> 1) select employees.\* from employees order by emp\_id , emp\_name desc;**

**2) select employees.\* from employees order by emp\_name ;**

**LIKE**

**Retrive the values which records mathes with defined pattern**

**Ex -> 1) select \* from employees**

**where (emp\_name like "%r%")&&( emp\_name like "%a%");**

2) mysql> select employees.\* from employees where emp\_name like "\_\_\_\_";

**LIMIT**

**return the records only specified rows;**

**ex-> select \* from employees order by salary desc limit 3;**

**Aggregate Functions**

**Aggregate Functions**

* **count()**
* **mysql> select count(salary) from employees;**
* **select emp\_name, salary,count(salary) from employees group by emp\_desi**

**g having salary>1 ;**

* **sum()**
* **avg()**
* **min()**
* **max()**
  + **select dept, max(salary) from employee group by dept having max(salary)>=40000;**
* **GROUP\_CONCAT()**

The GROUP\_CONCAT() function **returns the concatenated string from multiple rows into a single string**. If the group contains at least one non-null value, it always returns a string value. Otherwise, we will get a null value.

If we want to concatenate the designation of the same dept\_id on the employee table, we need to use the following query:

1. mysql> **SELECT** emp\_id, emp\_fname, emp\_lname, dept\_id,

GROUP\_CONCAT(designation) **as** "designation" **FROM** employee **group** **by** emp\_id;

* **first()**
* **last()**

**MySQL Clauses**

* **WHERE**
* **DISTINCT**
* **FROM**

The MySQL FROM Clause is used to select some records from a table. It can also be used to retrieve records from multiple tables using JOIN condition.

**MySQL FROM Clause: Retrieve data from two tables with inner join**

**SELECT officers.officer\_id, students.student\_name**

**FROM students**

**INNER JOIN officers**

**ON students.student\_id = officers.officer\_id;**

**MySQL FROM Clause: Retrieve data from two tables using outer join**

**SELECT officers.officer\_id, students.student\_name**

**FROM officers**

**LEFT OUTER JOIN students**

**ON officers.officer\_id = students.student\_id;**

* **ORDER BY**
* **GROUP BY**

The MYSQL GROUP BY Clause is used to collect data from multiple records and group the result by one or more column. It is generally used in a SELECT statement.

You can also use some aggregate functions like COUNT, SUM, MIN, MAX, AVG etc. on the grouped column.

**select count(emp\_id) from employee group by dept;**

**SELECT emp\_name, AVG(working\_hours) AS "Average working hour"**

**FROM employees**

**GROUP BY emp\_name;**

* **HAVING**
  + MySQL HAVING Clause is used with GROUP BY clause. It always returns the rows where condition is TRUE.
* **select count(\*),dept,salary from employee group by dept having salary>40000;**
  + **select count(\*),dept ,sum(salary) from employee group by dept having sum(salary)>40000;**
  + **HAVING condition: It is used to restrict the groups of returned rows. It shows only those groups in result set whose conditions are TRUE.**
* **Rollup**
  + **To get total**
  + **Used with group by clause**
  + **select ifnull(name,'total') , sum(fees) from sc\_view group by name with roll**

**up;**

**Table 14.33 Miscellaneous Functions**

| **Name** | **Description** |
| --- | --- |
| [ANY\_VALUE()](https://dev.mysql.com/doc/refman/8.4/en/miscellaneous-functions.html#function_any-value) | Suppress ONLY\_FULL\_GROUP\_BY value rejection |
| [BIN\_TO\_UUID()](https://dev.mysql.com/doc/refman/8.4/en/miscellaneous-functions.html#function_bin-to-uuid) | Convert binary UUID to string |
| [DEFAULT()](https://dev.mysql.com/doc/refman/8.4/en/miscellaneous-functions.html#function_default) | Return the default value for a table column |
| [GROUPING()](https://dev.mysql.com/doc/refman/8.4/en/miscellaneous-functions.html#function_grouping) | Distinguish super-aggregate ROLLUP rows from regular rows |
| [INET\_ATON()](https://dev.mysql.com/doc/refman/8.4/en/miscellaneous-functions.html#function_inet-aton) | Return the numeric value of an IP address |
| [INET\_NTOA()](https://dev.mysql.com/doc/refman/8.4/en/miscellaneous-functions.html#function_inet-ntoa) | Return the IP address from a numeric value |
| [IS\_UUID()](https://dev.mysql.com/doc/refman/8.4/en/miscellaneous-functions.html#function_is-uuid) | Whether argument is a valid UUID |
| [NAME\_CONST()](https://dev.mysql.com/doc/refman/8.4/en/miscellaneous-functions.html#function_name-const) | Cause the column to have the given name |
| [SLEEP()](https://dev.mysql.com/doc/refman/8.4/en/miscellaneous-functions.html#function_sleep) | Sleep for a number of seconds |
| [UUID()](https://dev.mysql.com/doc/refman/8.4/en/miscellaneous-functions.html#function_uuid) | Return a Universal Unique Identifier (UUID) |
| [UUID\_SHORT()](https://dev.mysql.com/doc/refman/8.4/en/miscellaneous-functions.html#function_uuid-short) | Return an integer-valued universal identifier |
| [UUID\_TO\_BIN()](https://dev.mysql.com/doc/refman/8.4/en/miscellaneous-functions.html#function_uuid-to-bin) | Convert string UUID to binary |
| [VALUES()](https://dev.mysql.com/doc/refman/8.4/en/miscellaneous-functions.html#function_values) | Define the values to be used during an INSERT |

**Sub Queries**

A subquery in MySQL is a query, which is nested into another SQL query and embedded with SELECT, INSERT, UPDATE or DELETE statement along with the various operators. We can also nest the subquery with another subquery. A subquery is known as the inner query, and the query that contains subquery is known as the outer query. The inner query executed first gives the result to the outer query, and then the main/outer query will be performed. MySQL allows us to use subquery anywhere, but it must be closed within parenthesis.

**The following are the rules to use subqueries:**

* **Subqueries should always use in parentheses.**
* **If the main query does not have multiple columns for subquery, then a subquery can have only one column in the SELECT command.**
* **We can use various comparison operators with the subquery, such as >, <, =, IN, ANY, SOME, and ALL. A multiple-row operator is very useful when the subquery returns more than one row.**
* **We cannot use the ORDER BY clause in a subquery, although it can be used inside the main query.**
* **If we use a subquery in a set function, it cannot be immediately enclosed in a set function.**

**The following are the advantages of using subqueries:**

* **The subqueries make the queries in a structured form that allows us to isolate each part of a statement.**
* **The subqueries provide alternative ways to query the data from the table; otherwise, we need to use complex joins and unions.**
* **The subqueries are more readable than complex join or union statements.**

**SYNTAX**

**SELECT** column\_list (s) **FROM**  table\_name

**WHERE**  column\_name OPERATOR

   (**SELECT** column\_list (s)  **FROM** table\_name [**WHERE**])

**A correlated subquery in MySQL is a subquery that depends on the outer query. It uses the data from the outer query or contains a reference to a parent query that also appears in the outer query. MySQL evaluates it once from each row in the outer query.**

SELECT emp\_name, city, income

FROM employees emp WHERE income > (

SELECT AVG(income) FROM employees WHERE city = emp.city);

**Ex -> select fname , dept ,salary from employee emp where salary >= (select avg(salary) from employee where dept=emp.dept);**

**select employee.\* from employee where salary = (select max(salary) from employee);**

**MySQL Subqueries with EXISTS or NOT EXISTS**

**The EXISTS operator is a Boolean operator that returns either true or false result. It is used with a subquery and checks the existence of data in a subquery. If a subquery returns any record at all, this operator returns true. Otherwise, it will return false. The NOT EXISTS operator used for negation that gives true value when the subquery does not return any row. Otherwise, it returns false. Both EXISTS and NOT EXISTS used with correlated subqueries.**

**The below SQL statements uses EXISTS operator to find the name, occupation, and age of the customer who has placed at least one order.**

**SELECT name, occupation, age FROM customer C**

**WHERE EXISTS (SELECT \* FROM Orders O**

**WHERE C.cust\_id = O.cust\_id);**

**This statement uses NOT EXISTS operator that returns the customer details who have not placed an order.**

**SELECT name, occupation, age FROM customer C**

**WHERE NOT EXISTS (SELECT \* FROM Orders O**

**WHERE C.cust\_id = O.cust\_id);**

**Date Time DateTime functions**

**Ex->** create table person( jd date not null default(current\_date),jt time not null default(now()),jdt datetime not null default(current\_timestamp));

* **Curdate()**
* **Curtime()**
* **Now()**
* **Dayname(date)**
* **Daymonth(date)**
* **Dayofweek(date)**
* **Monthname(date)**
* **Hour(time)**
* **Minute(time)**
* **Year(date);**

**Date Formating**

* Date\_format()
* Date\_format(now(),’%D %d at %T’) -> 21st Tue at 21:20:28
* Date\_format(now(),’%m/%d/%y’) -> 04/16/23
* For more [click here](https://dev.mysql.com/doc/refman/8.4/en/date-and-time-functions.html)

**Date Time Maths**

* Datediff(exp1,exp2); -> diff between days
* Date\_add(date,interval expr unit) -> date\_add(now(),interval 1 day);

Ex -> select date\_add(now(),interval 1 day);

* Date\_sub(date,interval expr unit)
* Timediff(expr1,expr2) -> select timediff('20:12:00','19:13:00');

**Default and On Update Timestamp**

* create table blogs(

blog varchar(150),

ct datetime default(now()),

ut datetime on update now());

update blogs set blog=concat("this is the recent update at",now());

**Operators**

* Relational Operators

(<,>,<=,>=,=,!=)

* Logical Operator

(AND ,OR, NOT)

* IN, NOT IN -> select \* from employee where dept Not in("Account","Cash","Loan");
* Between -> select \* from employee where salary between 40000 and 80000 ;
* Not Between -> select \* from employee where salary not between 40000 and 65000 ;
* Any

The ANY keyword is a MySQL operator that returns the Boolean value TRUE if the comparison is TRUE for ANY of the subquery condition. In other words, this keyword returns true if any of the subquery condition is fulfilled when the SQL query is executed. The ANY keyword must follow the comparison operator. It is noted that ALL SQL operator works related to ANY operator, but it returns true when all the subquery values are satisfied by the condition in MySQL.

The ANY operator works like comparing the value of a table to each value in the result set provided by the subquery condition. And then, if it finds any value that matches at least one value/row of the subquery, it returns the TRUE result.

* SELECT column\_lists FROM table\_name1 WHERE column\_name Operator ANY (SELECT column\_name FROM table\_name2 WHERE condition);
* SELECT colm1 FROM table1 WHERE colm1 > ANY (SELECT colm1 FROM table2);
* =  >  <  >=  <=  <>  !=
* SELECT num\_value FROM table1
* WHERE num\_value > ANY (SELECT num\_val FROM table2);

The **IN keyword** in MySQL is an **alias for = ANY** when used with a subquery in the statement. Hence, the below two statements are identical in MySQL:

1. **SELECT** colm1 **FROM** table1 **WHERE** colm1 = ANY (**SELECT** colm1 **FROM** table2);
2. **SELECT** colm1 **FROM** table1 **WHERE** colm1 IN (**SELECT** colm1 **FROM** table2);

But we cannot say that IN and = ANY were synonyms when we used it with an expression list. It is because IN can take a list of expressions, but = ANY cannot.

Also, **NOT IN** cannot be an alias for **<> ANY** operator, but it can be used for **<> ALL**.

The word **SOME** in MySQL can be an alias for ANY. Therefore, these two [SQL](https://www.javatpoint.com/sql-tutorial) statements are equivalent:

1. **SELECT** colm1 **FROM** table1 **WHERE** colm1 <>ANY (**SELECT** colm1 **FROM** table2);
2. **SELECT** colm1 **FROM** table1 **WHERE** colm1 <> SOME (**SELECT** colm1 **FROM** table2);

* **Exists**

The EXISTS operator in MySQL is a type of Boolean operator which returns the **true or false** result. It is used in combination with a subquery and checks the existence of data in a subquery. It means if a subquery returns any record, this operator returns true. Otherwise, it will return false. The true value is always represented numeric value 1, and the false value represents 0. We can use it with SELECT, UPDATE, DELETE, INSERT statement.

**SELECT name, occupation FROM customer**

**WHERE EXISTS (SELECT \* FROM Orders**

**WHERE customer.cust\_id = Orders.cust\_id);**

**select exists (select \* from employee) as result;**

**CASE**

MySQL CASE expression is a part of the control flow function that provides us to write an **if-else or if-then-else** logic to a query. This expression can be used anywhere that uses a valid program or query, such as SELECT, WHERE, ORDER BY clause, etc.

The CASE expression validates various conditions and returns the result when the first condition is **true**. Once the condition is met, it stops traversing and gives the output. If it will not find any condition true, it executes the **else block**. When the else block is not found, it returns a **NULL** value. The main goal of [MySQL](https://www.javatpoint.com/mysql-tutorial) CASE statement is to deal with multiple IF statements in the SELECT clause.

SELECT studentid, firstname,

CASE class

WHEN 'CS' THEN 'Computer Science'

WHEN 'EC' THEN 'Electronics and Communication'

ELSE 'Electrical Engineering'

END AS department from students;

select employee.\* ,case when salary>40000 then 'higher salary' else 'low salary' end as 'sal cat' from employee;

mysql> select dept, case when dept ='IT' then "Good" else "not Good" end as "res" from employee

**RELATIONSHIPS**

* **One to one**
* **One to many**
* **Many to many**
  + **Create junction table to store relationship :**

**create table student\_course(sid int ,cid int ,constraint foreign key(sid) references students(id),constraint foreign key(cid) references courses(id));**

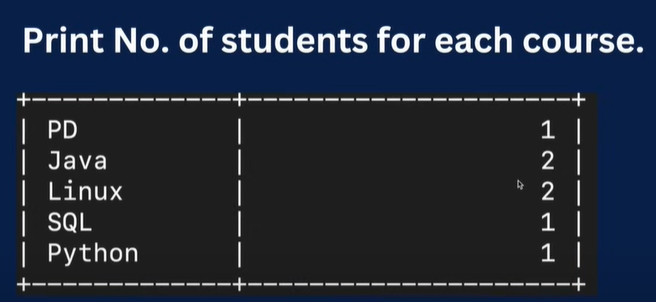
* + **select name,course\_name from students**

**join**

**student\_course on student\_course.sid=students.id**

**join**

**courses on student\_course.cid=courses.id**

****

**select course\_name , count(cid) from courses join student\_course on**

**student\_course.cid=courses.id**

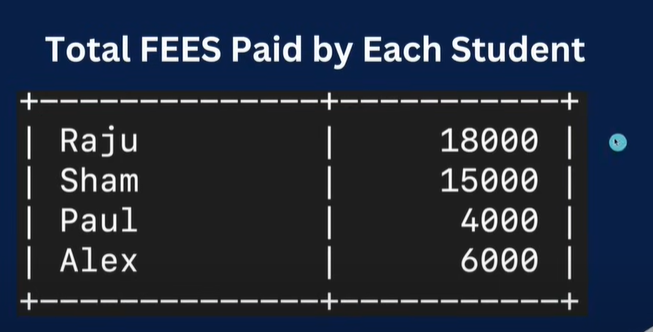
**group by(cid);**

****

* **select name , count(sid) from students join student\_course on student\_course**

**.sid=students.id group by sid;**

* **select name , count(sid) from student\_course join students on students.id=student\_course.sid group by id;**

****

* **select name , sum(fees) from students join student\_course on student\_course.sid=students.id join courses on courses.id=student\_course.cid group by name;**

**Views**

**Virtual tables , stored queries**

* **[-- Show tables] command will show us name if you forget**
  + Create VIEW myView (id,first\_name,city)   
    AS   
    SELECT id, first\_name, city   
    FROM   
    mployee;
  + **Ex-> create view sc\_view as**

**select name , course\_name,fees from students inner join student\_course on student\_course.sid=students.id inner join courses on courses.id=student\_course.cid;**

* + **ALTER VIEW view\_name column\_list AS select\_statement;**
  + ALTER VIEW myView (id,first\_name,city)   
    AS   
    SELECT id, upper(first\_name), city   
    FROM employee;
* **2. Adding One More Column**
  + ALTER VIEW myView (id,first\_name,city,emp\_details)   
    AS   
    SELECT id, upper(first\_name), city, description   
    FROM   
    employee;

**3. Removing a Column**

* + ALTER VIEW myView (id,first\_name,emp\_details)   
    AS   
    SELECT id, upper(first\_name), description   
    FROM   
    employee;

**4. Changing Data in a Column**

* + UPDATE myView set emp\_details = "GFG"   
    WHERE id = 6;
* **Drop VIEW**
* DROP view myView;

**Types Of Join**

* Cross join

-> select \* from customers,orders;

* Inner join
  + select \* from customers inner join orders on orders.cust\_id=customers.cust\_id;
  + select name , sum(amount) from customers inner join orders on orders.cust\_id=customers.cust\_id group by customers.cust\_id;
  + select name , sum(amount) from customers cuss inner join orders ord on ord.cust\_id=cuss.cust\_id group by cuss.cust\_id;
* Left join
  + mysql> select \* from customers left outer join orders on orders.cust\_id=customers.cust\_id;
  + select name,ifnull(sum(amount),0) from customers left join orders on orders.cust\_id = customers.cust\_id group by name;
* Right join
  + mysql> select \* from orders right join customers on orders.cust\_id = customers.cust\_id;

**Cascade On Delete**

* alter table orders drop constraint orders\_ibfk\_1;
* Using alter -> alter table orders add constraint foreign key (cust\_id) references customers(cust\_id) on delete cascade ;

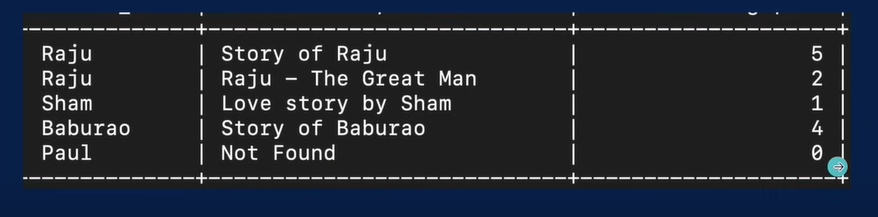
**Data Query Language (DQL)**

Select \* from <table name>

Select column from <table name>

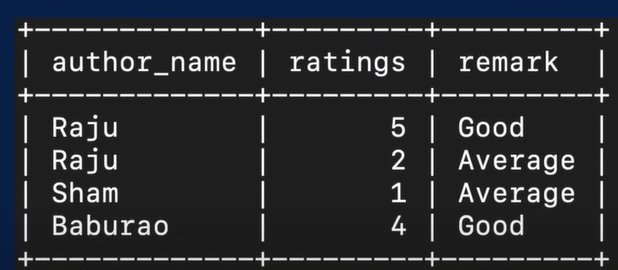
select student.\*,stud\_id from student;

select student.\*,stud\_id from student where stud\_id = 5;



select auth\_name, ifnull(title,'notfound'),ifnull(au\_id,0) from authors left join books on books.au\_id

=authors.auth\_id;



select auth\_name,rating,

case

when ifnull(rating,0)>=3 then 'Good'

else 'Ave'

end as remark

from authors inner join books on books.au\_id=authors.auth\_id;

* **stored routine** 
  + **an sql statement or a set of sql statement that can be stored on database server which can be call on of times.**
  + **Types** 
    - **Stored procedure** 
      * **These are routines that contain a series of sql statements and procedural logic.**
      * **Often used for performing action like data modification, transaction control, and executing sequences of statements.**
      * **Ex: delimiter $$**

**create procedure allemp()**

**begin select \* from employee;**

**end$$**

**delimiter ;**

* + - * **Call() db.procedure();**
      * **Ex with arguments**

**delimiter $$**

**create procedure getemp(in ename varchar(30))**

**begin**

**select \* from employee where fname=ename;**

**end$$**

**delimiter ;**

* + - * **Call bank\_db.getemp(baburao);**
      * **Ex with return output in variable**

**delimiter $$**

**create procedure getsumdepsal(in dname varchar(30),out depsum decimal(10,2))**

**begin**

**select sum(salary) into depsum from employee where dept=dname;**

**end$$**

**delimiter ;**

* + - * **Calling**

**set @depsum = 0;**

**call bank\_db.getsumdepsal('Loan', @depsum);**

**select @depsum;**

* + - **User defined functions**

delimiter $$

create function get\_high\_emp() returns varchar(30)

Deterministic no sql reads sql data

begin

declare name varchar(30);

select fname into name from employee where salary = (select max(salary) from employee);

return name;

end$$

delimiter ;

To call function -> select bank\_db.get\_high\_emp();

Ex to get name of employee with highest salary from dept

delimiter $$

create function get\_highest\_from\_dep2(dname varchar (30)) returns varchar(30)

Deterministic no sql

begin

declare ename varchar(30);

select fname into ename from employee where dept=dname AND salary = (select max(salary) from employee where dept=dname);

return ename;

end$$

delimiter ;

**Window Function**

**Also known as analytic functions allow you to perform calculations across a set of rows related to the current row**

**Define by an over() clause**

**select fname,salary,sum(salary) over(order by emp\_id) from employee;**

**select fname,salary,dept,sum(salary) over(partition by dept) from employee;**

**Other functions**

* + - * **Row\_number()**
        + **select fname,salary,dept,row\_number() over(partition by dept) from employee;**
      * **Rank()**
        + **select fname,salary,dept,rank() over(order by salary) from employee;**
      * **Dense\_rank()**
        + **select fname,salary,dept,dense\_rank() over(order by salary) from employee;**
      * **Lag()**
        + **select fname,salary,dept,salary-lag(salary) over(order by salary desc) from employee;**
      * **Lead()**
        + **select fname,salary,dept,salary-lead(salary) over(order by salary desc) from employee;**

# **MySQL Transaction**

A transaction in MySQL is a **sequential group of statements**, queries, or operations such as select, insert, update or delete to perform as a one single work unit that can be committed or rolled back. If the transaction makes multiple modifications into the database, two things happen:

* Either all modification is successful when the transaction is committed.
* Or, all modifications are undone when the transaction is rollback.

In other words, a transaction cannot be successful without completing each operation available in the set. It means if any statement fails, the transaction operation cannot produce results.

A transaction in MySQL starts with the first executable SQL statement and ends when it finds a commit or rolled back either explicitly or implicitly. It explicitly uses COMMIT or ROLLBACK statement and implicitly when a DDL statement is used.

### **Properties of Transaction**

The transaction contains mainly four properties, which referred to as **ACID** property. Now, we are going to discuss the ACID property in detail. The ACID property stands for:

1. Atomicity
2. Consistency
3. Isolation
4. Durability

**Atomicity:** This property ensures that all statements or operations within the transaction unit must be executed successfully. Otherwise, if any operation is failed, the whole transaction will be aborted, and it goes rolled back into their previous state. It includes features:

* COMMIT statement.
* ROLLBACK statement.
* Auto-commit setting.
* Operational data from the INFORMATION\_SCHEMA tables.

**Consistency:** This property ensures that the database changes state only when a transaction will be committed successfully. It is also responsible for protecting data from crashes. It includes features:

* InnoDB doublewrite buffer.
* InnoDB crash recovery.

**Isolation:** This property guarantees that each operation in the transaction unit operated independently. It also ensures that statements are transparent to each other. It includes features:

* SET ISOLATION LEVEL statement.
* Auto-commit setting.
* The low-level details of InnoDB locking.

**Durability:** This property guarantees that the result of committed transactions persists permanently even if the system crashes or failed. It includes features:

* Write buffer in a storage device.
* Battery-backed cache in a storage device.
* Configuration option innodb\_file\_per\_table.
* Configuration option innodb\_flush\_log\_at\_trx\_commit.
* Configuration option sync\_binlog.

### **MySQL Transaction Statement**

MySQL control transactions with the help of the following statement:

* MySQL provides a START TRANSACTION statement to begin the transaction. It also offers a "BEGIN" and "BEGIN WORK" as an alias of the START TRANSACTION.
* We will use a COMMIT statement to commit the current transaction. It allows the database to make changes permanently.
* We will use a ROLLBACK statement to roll back the current transaction. It allows the database to cancel all changes and goes into their previous state.
* We will use a SET auto-commit statement to disable/enable the auto-commit mode for the current transaction. By default, the COMMIT statement executed automatically. So if we do not want to commit changes automatically, use the below statement:

1. **SET** autocommit = 0;
2. OR,
3. **SET** autocommit = **OFF**:
4. **SET** autocommit = 1;
5. OR,
6. **SET** autocommit = **ON**:

### **COMMIT Example**

1. -- 1. Start a new transaction
3. START **TRANSACTION**;
5. -- 2. Get the highest income
7. **SELECT** @income:= **MAX**(income) **FROM** employees;
9. -- 3. Insert a new record into the employee table
11. **INSERT** **INTO** employees(emp\_id, emp\_name, emp\_age, city, income)
12. **VALUES** (111, 'Alexander', 45, 'California', 70000);
14. -- 4. Insert a new record into the order table
16. **INSERT** **INTO** Orders(order\_id, prod\_name, order\_num, order\_date)
17. **VALUES** (6, 'Printer', 5654, '2020-01-10');
19. -- 5. Commit changes
20. **COMMIT**;

### **ROLLBACK Example**

We can understand the rollback transaction with the help of the following illustration. First, open the MySQL command prompt and log into the database server using the password. Next, we have to select a database.

Suppose our database contains the "**Orders**" table. Now, the following are the scripts that perform the rollback operations:

1. -- 1. Start a new transaction
2. START **TRANSACTION**;
4. -- 2. Delete data from the order table
5. **DELETE** **FROM** Orders;
6. -- 3. Rollback changes
7. **ROLLBACK**;
9. -- 4. Verify the records in the first session
10. **SELECT** \* **FROM** Orders;

**Statements that cannot be a rollback in using MySQL Transaction.**

MySQL Transaction cannot be able to roll back all statements. For example, these statements include DDL (Data Definition Language) commands such as CREATE, ALTER, or DROP database as well as CREATE, UPDATE, or DROP tables or stored routines. We have to make sure that when we design our transaction, these statements do not include.

### **SAVEPOINT, ROLLBACK TO SAVEPOINT, RELEASE SAVEPOINT**

The **SAVEPOINT** statement creates a special mark with the name of the **identifier** inside a transaction. It allows all statements that are executed after savepoint would be rolled back. So that the transaction restores to the previous state it was in at the point of the savepoint. If we have set multiple savepoints in the current transaction with the same name, the newly savepoint is responsible for rollback.

The **ROLLBACK TO SAVEPOINT** statement allows us to rolls back all transactions to the given savepoint was established without aborting the transaction.

The **RELEASE SAVEPOINT** statement destroys the named savepoint from the current transaction without undoing the effects of queries executed after the savepoint was established. After these statements, no rollback command occurs. If the savepoint does not exist in the transaction, it gives an error.

1. SAVEPOINT savepoint\_name
2. **ROLLBACK** **TO** [SAVEPOINT] savepoint\_name
3. RELEASE SAVEPOINT savepoint\_name

### **Example**

Let us understand how to use these statements through the example. In the below example, we are going to use SAVEPOINT and ROLLBACK TO SAVEPOINT statements that explain how a savepoint determines which records of the current transaction can be rolled back.

1. START **TRANSACTION**;
3. **SELECT** \* **FROM** Orders;
5. **INSERT** **INTO** Orders(order\_id, prod\_name, order\_num, order\_date)
6. **VALUES** (6, 'Printer', 5654, '2020-01-10');
8. SAVEPOINT my\_savepoint;
10. **INSERT** **INTO** Orders(order\_id, prod\_name, order\_num, order\_date)
11. **VALUES** (7, 'Ink', 5894, '2020-03-10');
13. **ROLLBACK** **TO** SAVEPOINT my\_savepoint;
15. **INSERT** **INTO** Orders(order\_id, prod\_name, order\_num, order\_date)
16. **VALUES** (8, 'Speaker', 6065, '2020-02-18');
18. **COMMIT**;

### **What is SQLyog?**

SQLyog program is the most popular GUI tool for admin. It is the most popular MySQL manager and admin tool. It combines the features of MySQL administrator, phpMyadmin, and others. MySQL front ends and MySQL GUI tools.

### **Write a query to count the number of rows of a table in MySQL.**

**SELECT COUNT** user\_id FROM users;

### **Write a query to retrieve a hundred books starting from 20th.**

**SELECT** book\_title FROM books LIMIT 20, 100;

### **How is the MyISAM table stored?**

MyISAM table is stored on disk in three formats.

* '.frm' file : storing the table definition
* '.MYD' (MYData): data file
* '.MYI' (MYIndex): index file

### **What is the usage of ENUMs in MySQL?**

ENUMs are string objects. By defining ENUMs, we allow the end-user to give correct input as in case the user provides an input that is not part of the ENUM defined data, then the query won't execute, and an error message will be displayed which says "The wrong Query". For instance, suppose we want to take the gender of the user as an input, so we specify ENUM('male', 'female', 'other'), and hence whenever the user tries to input any string any other than these three it results in an error.

ENUMs are used to limit the possible values that go in the table:

**For example:**

CREATE TABLE months (month ENUM 'January', 'February', 'March'); INSERT months VALUES ('April').

### **What are the advantages of MyISAM over InnoDB?**

MyISAM follows a conservative approach to disk space management and stores each MyISAM table in a separate file, which can be further compressed if required. On the other hand, InnoDB stores the tables in the tablespace. Its further optimization is difficult.

### **What are the differences between MySQL\_fetch\_array(), MySQL\_fetch\_object(), MySQL\_fetch\_row()?**

Mysql\_fetch\_object is used to retrieve the result from the database as objects, while mysql\_fetch\_array returns result as an array. This will allow access to the data by the field names.

**For example:**

Using mysql\_fetch\_object field can be accessed as $result->name.

Using mysql\_fetch\_array field can be accessed as $result->[name].

Using mysql\_fetch\_row($result) where $result is the result resource returned from a successful query executed using the mysql\_query() function.

**Example:**

1. $result = mysql\_query("SELECT \* from students");
2. while($row = mysql\_fetch\_row($result))
3. {
4. Some statement;
5. }

### **What is the difference between mysql\_connect and mysql\_pconnect?**

Mysql\_connect() is used to open a new connection to the database, while mysql\_pconnect() is used to open a persistent connection to the database. It specifies that each time the page is loaded, mysql\_pconnect() does not open the database.

### **What is the use of mysql\_close()?**

Mysql\_close() cannot be used to close the persistent connection. However, it can be used to close a connection opened by mysql\_connect().

### **What is MySQL data directory?**

MySQL data directory is a place where MySQL stores its data. Each subdirectory under this data dictionary represents a MySQL database. By default, the information managed my MySQL = server mysqld is stored in the data directory.

### **What is the usage of regular expressions in MySQL?**

In MySQL, regular expressions are used in queries for searching a pattern in a string.

* \* Matches 0 more instances of the string preceding it.
* + matches one more instances of the string preceding it.
* ? Matches 0 or 1 instances of the string preceding it.
* . Matches a single character.
* [abc] matches a or b or z
* | separates strings
* ^ anchors the match from the start.
* "." Can be used to match any single character. "|" can be used to match either of the two strings
* REGEXP can be used to match the input characters with the database.

The following statement retrieves all rows where column employee\_name contains the text 1000 (example salary):

1. **Select** employee\_name
2. **From** employee
3. **Where** employee\_name REGEXP '1000'
4. **Order** **by** employee\_name

### **What is the usage of the "i-am-a-dummy" flag in MySQL?**

In MySQL, the "i-am-a-dummy" flag makes the MySQL engine to deny the UPDATE and DELETE commands unless the WHERE clause is present.

### **Which command is used to view the content of the table in MySQL?**

The SELECT command is used to view the content of the table in MySQL.

**Explain Access Control Lists.**

An ACL is a list of permissions that are associated with an object. MySQL keeps the Access Control Lists cached in memory, and whenever the user tries to authenticate or execute a command, MySQL checks the permission required for the object, and if the permissions are available, then execution completes successfully.

### **What is InnoDB?**

InnoDB is a storage database for SQL. The ACID-transactions are also provided in InnoDB and also includes support for the foreign key. Initially owned by InnobaseOY now belongs to Oracle Corporation after it acquired the latter since 2005.

### **What is ISAM?**

It is a system for file management developed by IBM, which allows records to access sequentially or even randomly.

### **How can we run batch mode in MySQL?**

To perform batch mode in MySQL, we use the following command:

1. mysql;
2. mysql mysql.**out**;

### **What are federated tables?**

Federated tables are tables that point to the tables located on other databases on some other server.

### **What is the difference between primary key and candidate key?**

To identify each row of a table, we will use a primary key. For a table, there exists only one primary key.

A candidate key is a column or a set of columns, which can be used to uniquely identify any record in the database without having to reference any other data.

### **What are the drivers in MySQL?**

Following are the drivers available in MySQL:

* PHP Driver
* JDBC Driver
* ODBC Driver
* C WRAPPER
* PYTHON Driver
* PERL Driver
* RUBY Driver
* CAP11PHP Driver
* Ado.net5.mxz

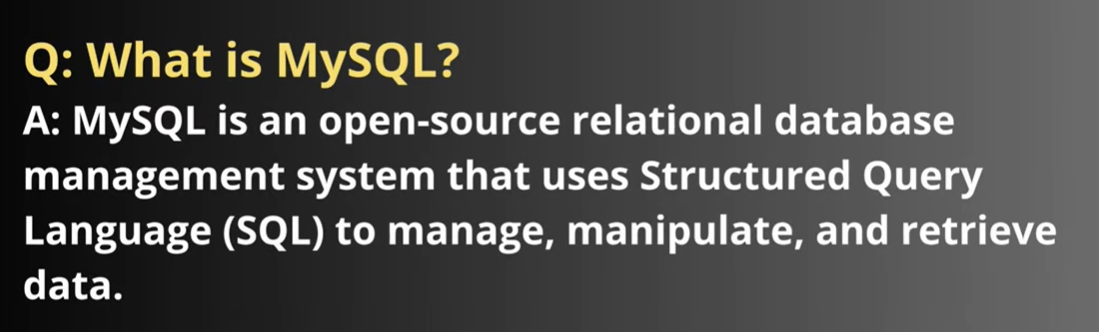
### **What are DDL, DML, and DCL?**

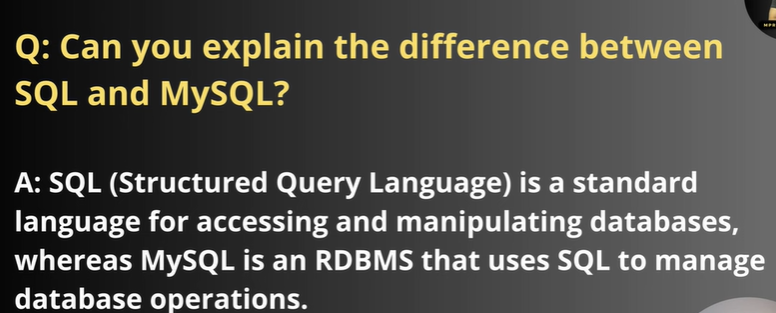
Majorly SQL commands can be divided into three categories, i.e., DDL, DML & DCL. Data Definition Language (DDL) deals with all the database schemas, and it defines how the data should reside in the database. Commands like CreateTABLE and ALTER TABLE are part of DDL.

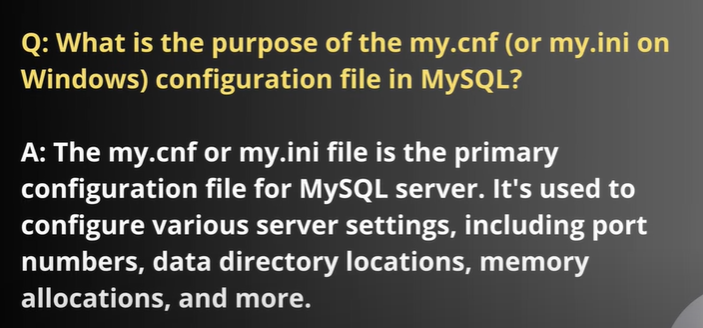
Data Manipulative Language (DML) deals with operations and manipulations on the data. The commands in DML are Insert, Select, etc.

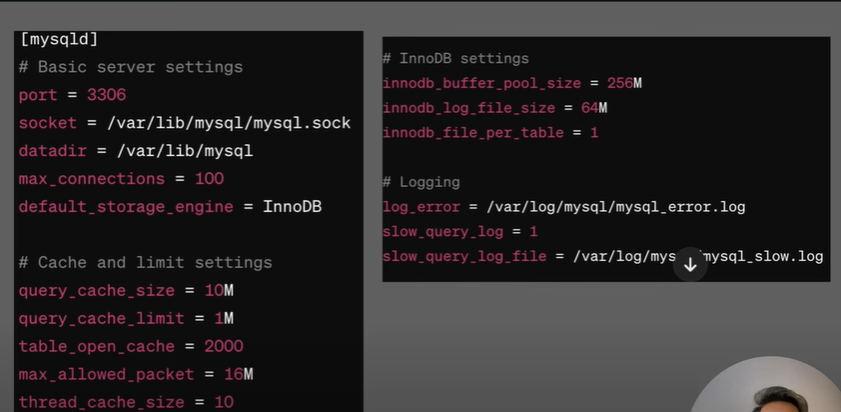
Data Control Languages (DCL) are related to the Grant and permissions. In short, the authorization to access any part of the database is defined by these.

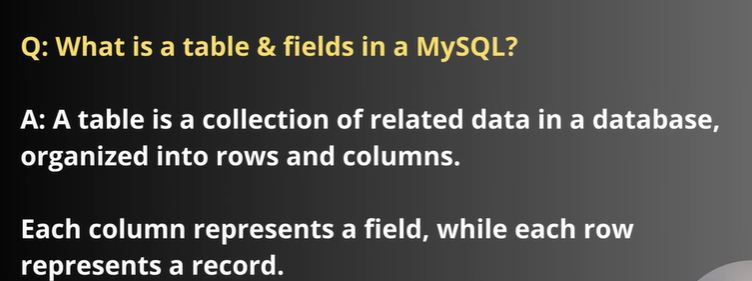
**50 MySQL Interview Questions For Beginners for JOB and EXAM**

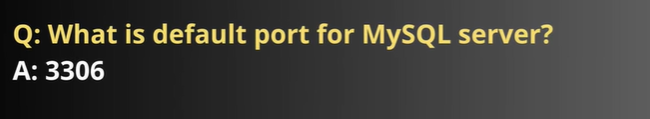


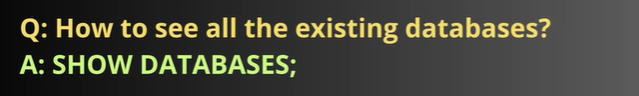


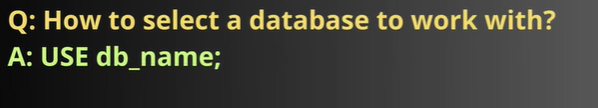


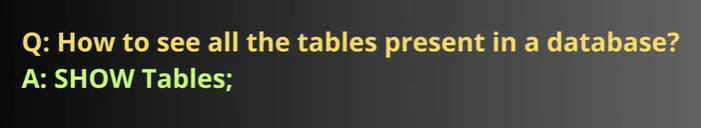


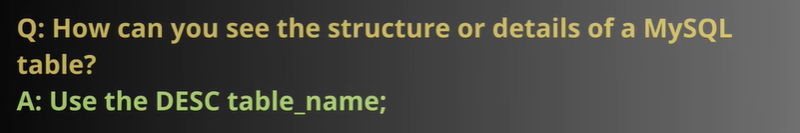


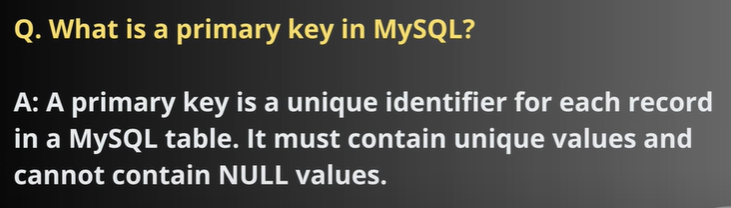


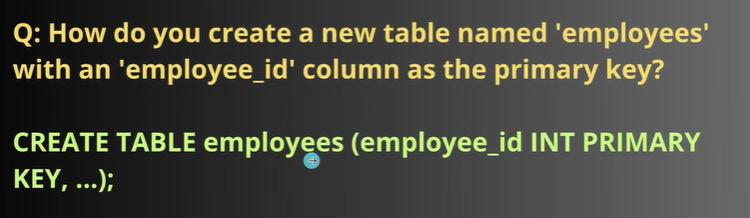


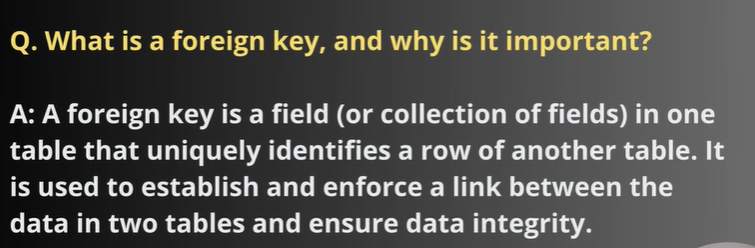


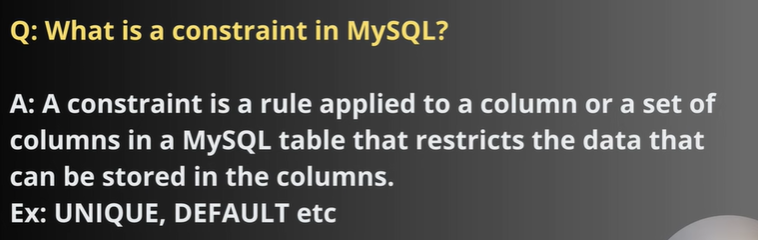


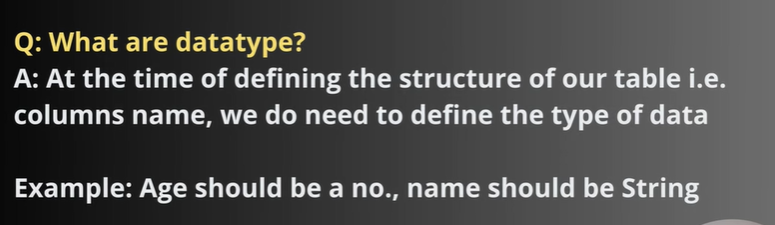


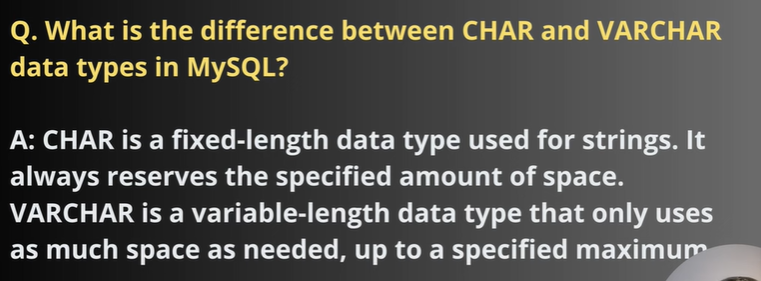


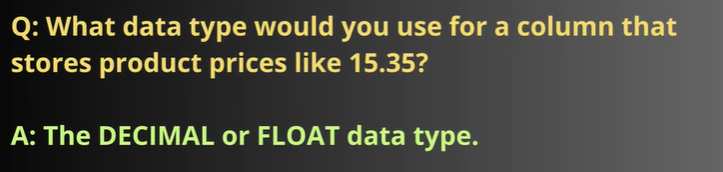


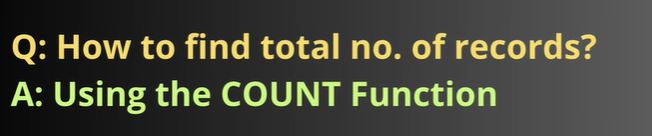




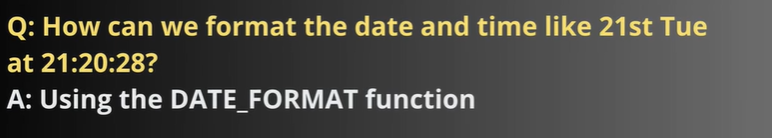


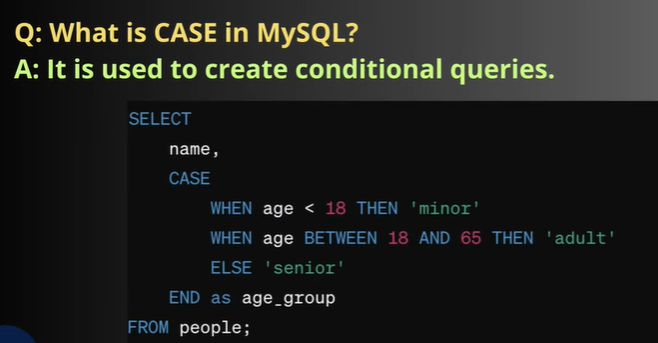


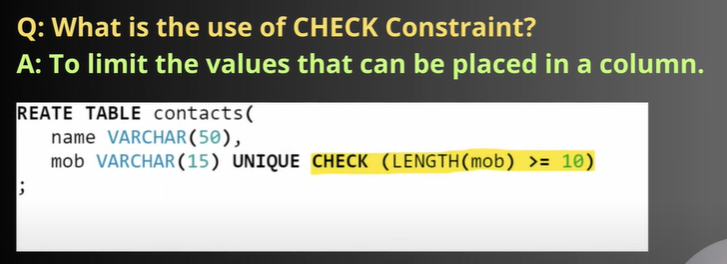


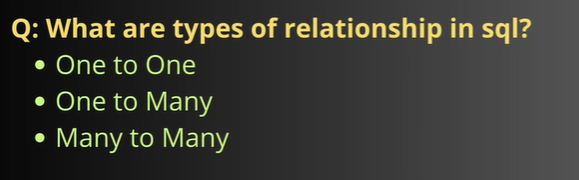


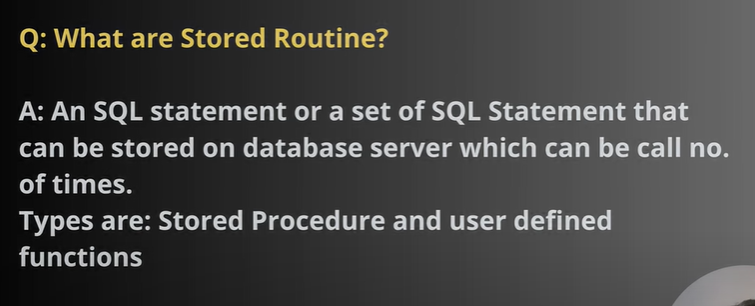


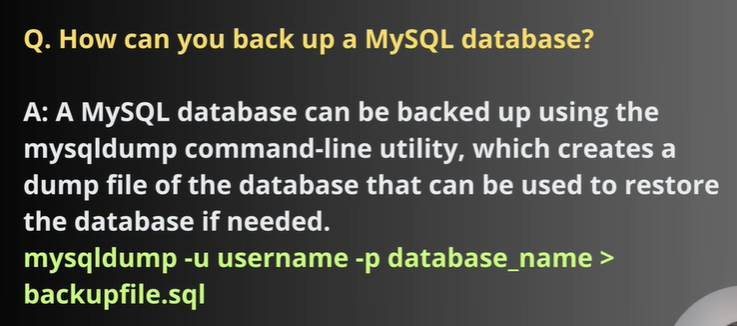


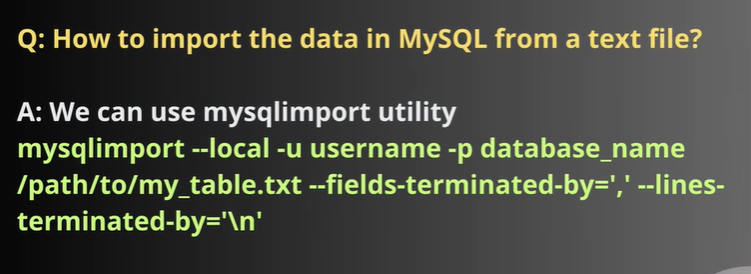


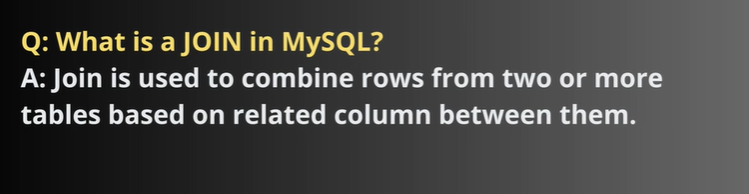




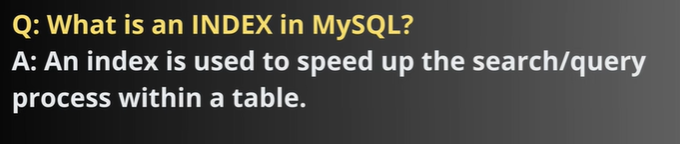


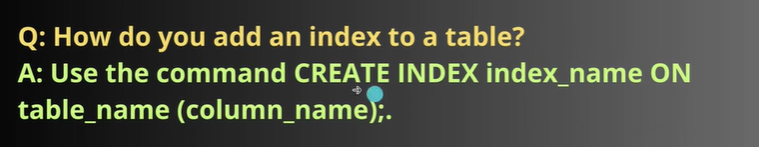


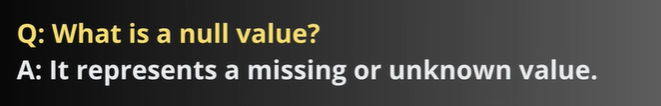


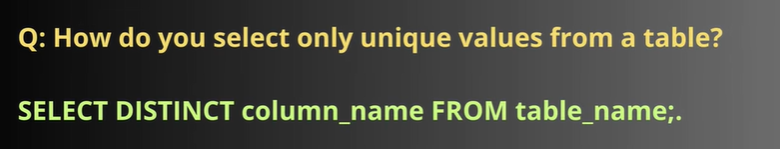




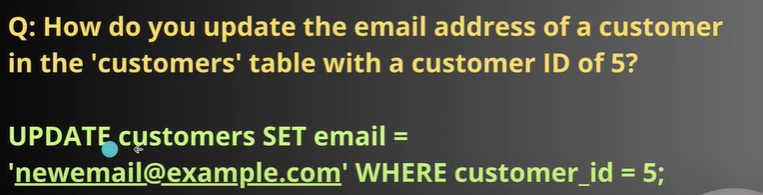


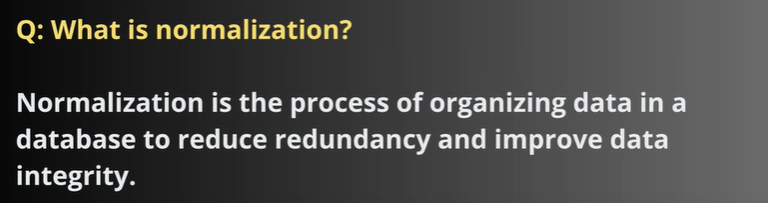


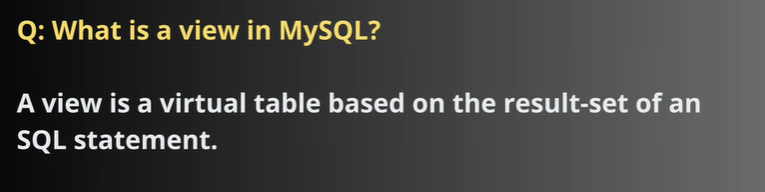


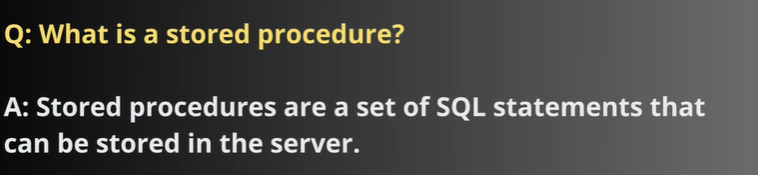


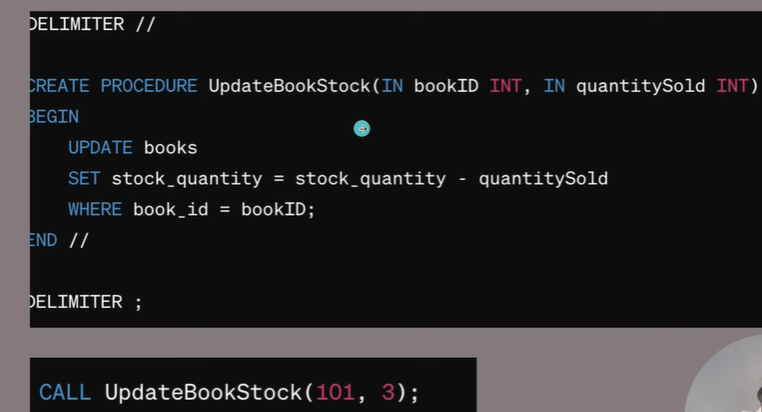


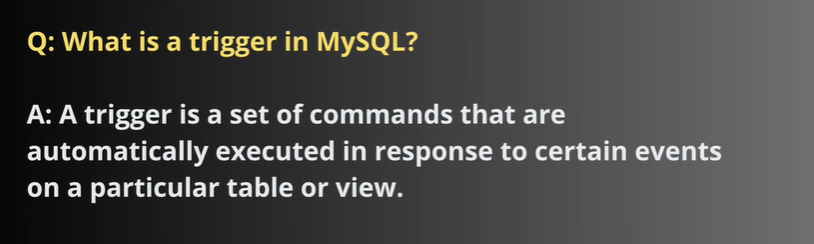


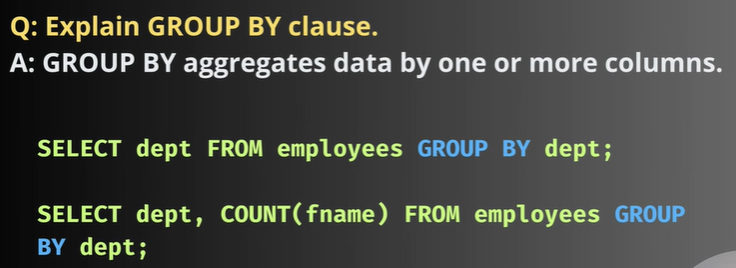


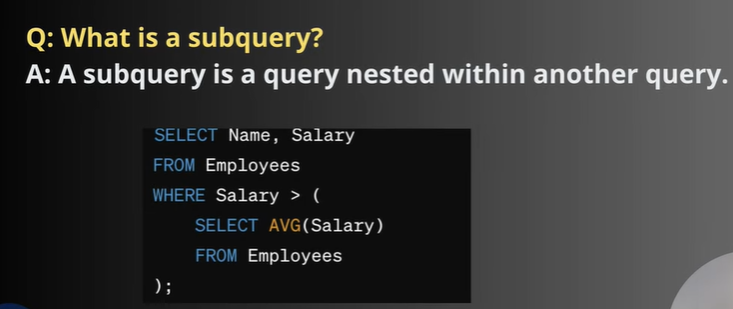


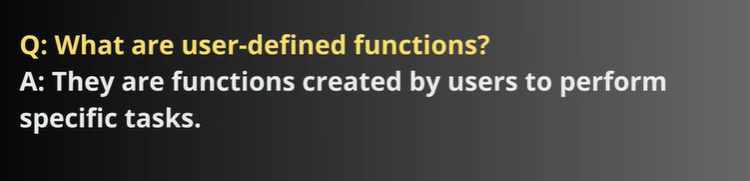


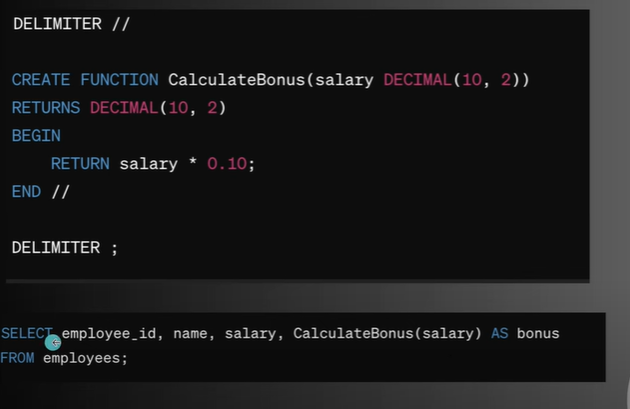


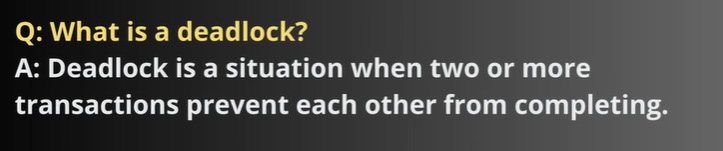


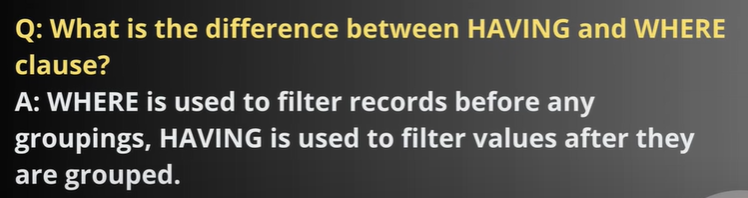


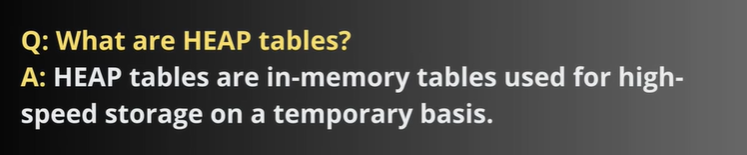


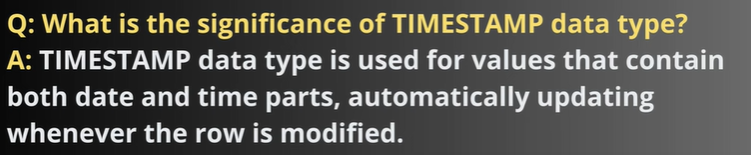


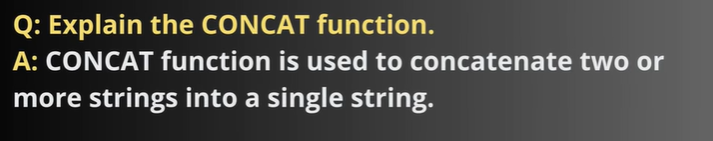


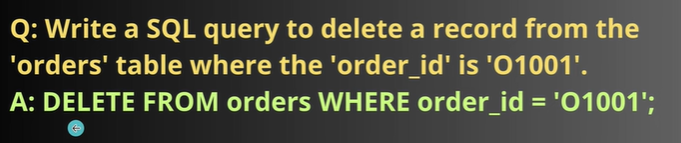


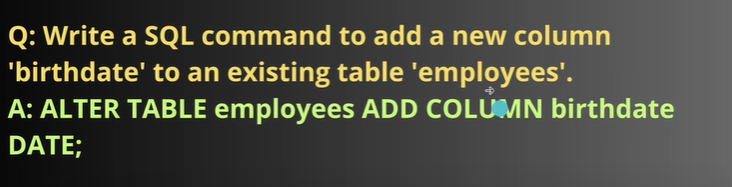


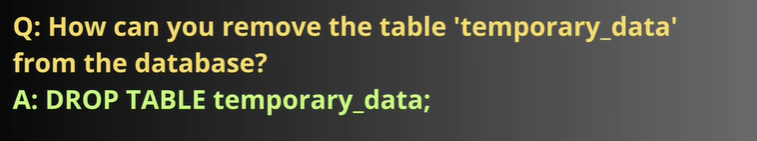
****

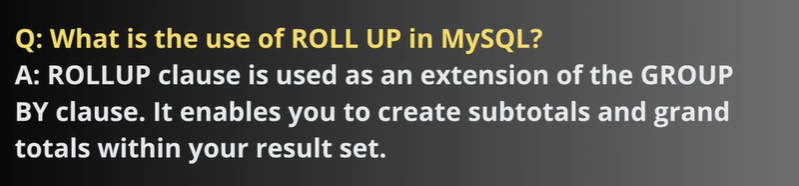
****

****

****

****

****

****

How to change the database name in MySQL?

1. mysqldump -u username -p "password" -R oldDbName > oldDbName.sql
2. mysql -u username -p"password" newDbName < oldDbName.sql

How to import a database in MySQL?

* Command Line Tool
* MySQL Workbench

**Step 1:** Execute the below statement to show all databases available on [MySQL](https://www.javatpoint.com/mysql-tutorial) Server.

1. mysql> SHOW DATABASES;

**Step 2:** Access the command line on the computer where the database is stored. You can open a DOS or terminal window to access the command line if you have physical access to the computer. **For example**, if we have installed the MySQL in the **C folder**, copy the following folder and paste it in our DOS command. Now, press **Enter key**.

1. C:\Users\javatpoint> CD C:\Program Files\MySQL\MySQL Server 8.0\bin

**Step 3:** Now, we will use the **mysqldump** tool to export the database. This tool uses the login credentials of MySQL user for the operation. The following command is used to export the database in your desired place.

1. $ mysqldump -u username -p database\_name > desiredplace\dbname.sql

In the below statement, we will give a username: **root**, database name: **mytestdb**, folder name: **BackupFile**, and output database name: **testdb.sql.** Now, press the Enter key.

1. mysqldump -u root -p mytestdb > D:\BackupFile\testdb.sql

**Step 4:** After pressing the Enter key, it will ask the password of the username we have specified. Then, press the Enter key again. It will create the backup file with a **.sql suffix** in the specified location. We will get the output as below that means the database exported successfully:

MySQL Import Database

**Step 1:** Create a blank database named **"mytestdb\_copy"** by using the below statement:

1. mysql> **CREATE** **DATABASE** mytestdb\_copy;

**Step 2:** Next, we need to execute the below commands to verify that it does not have any table.

1. mysql> USE mytestdb\_copy;
2. mysql> SHOW TABLES;

**Step 3:** Access the command line on the computer where the database is stored. You can open a DOS or terminal window to access the command line if you have physical access to the computer. **For example**, if we have installed the MySQL in the **C folder**, copy the following folder link and paste it in your DOS command. Now, press Enter key.

1. C:\Users\javatpoint> CD C:\Program Files\MySQL\MySQL Server 8.0\bin

**Step 4:** Next, we will use the following command to import the dump file in your desired database.

1. $ mysql -u username -p database\_name < desiredplace\dbname.sql

In the below statement, we will give a username: **root**, database name: **mytestdb\_copy**, location of backup file: **BackupFile**, and input database name: **testdb.sql**. Now, press Enter key.

1. $ mysql -u root -p mytestdb\_copy < D:\BackupFile\testdb.sql

**Step 5:** After pressing the Enter key, it will ask the password of the username we have specified. Then, press the Enter key again. If the command executed correctly, it does not display any content on the screen. We will get the below screen that means the database imported successfully:

**Step 6:** Open the MySQL client tool again and execute the below command to verify the database. In the output, we can see that this database now contains two tables:

 How to change the column name in MySQL?

1. **ALTER** **TABLE** table\_name
2. CHANGE **COLUMN** old\_column\_name new\_column\_name column\_definition [**FIRST**|**AFTER** existing\_column];

How to delete columns in MySQL?

1. **ALTER** **TABLE** table\_name **DROP** **COLUMN** column\_name1, column\_name2....;

How to insert data in MySQL?

1. **INSERT** **INTO** table\_name ( field1, field2,...fieldN )
2. **VALUES**  ( value1, value2,...valueN );
3. **INSERT** **INTO** **table**(field1, field2,...fieldN)
4. **VALUES**
5. (value1, value 2, ...),
6. (value1, value2, ...),
7. ...
8. (value1, value2, ...);

How to delete a row in MySQL?

1. **DELETE** **FROM** table\_name **WHERE** Condition\_specified;

How to join three tables in MySQL?

Sometimes we need to fetch data from three or more tables. There are two types available to do these types of joins. Suppose we have three tables named Student, Marks, and Details.

Let's say Student has (stud\_id, name) columns, Marks has (school\_id, stud\_id, scores) columns, and Details has (school\_id, address, email) columns.

**1. Using SQL Join Clause**

This approach is similar to the way we join two tables. The following query returns result from three tables:

1. **SELECT** **name**, scores, address, email **FROM** Student s
2. **INNER** JOIN Marks m **on** s.stud\_id = m.stud\_id
3. **INNER** JOIN Details d **on** d.school\_id = m.school\_id;

**2. Using Parent-Child Relationship**

It is another approach to join more than two tables. In the above tables, we have to create a parent-child relationship. First, create column X as a primary key in one table and as a foreign key in another table. Therefore, stud\_id is the primary key in the Student table and will be a foreign key in the Marks table. Next, school\_id is the primary key in the Marks table and will be a foreign key in the Details table. The following query returns result from three tables:

1. **SELECT** **name**, scores, address, email
2. **FROM** Student s, Marks m, Details d
3. **WHERE** s.stud\_id = m.stud\_id AND m.school\_id = d.school\_id;

 How to update the table in MySQL?

1. **UPDATE** table\_name
2. **SET** field1=new-value1, field2=new-value2, ...
3. [**WHERE** Clause]

How to drop the primary key in MySQL?

1. **ALTER** **TABLE** table\_name  **DROP** **PRIMARY** **KEY**;

How to create a Stored Procedure in MySQL?

A stored procedure is a group of SQL statements that we save in the database. The SQL queries, including INSERT, UPDATE, DELETE, etc. can be a part of the stored procedure. A procedure allows us to use the same code over and over again by executing a single statement. It stores in the database data dictionary.

1. DELIMITER &&
2. **CREATE** **PROCEDURE** procedure\_name [[IN | **OUT** | INOUT] parameter\_name datatype [, parameter datatype]) ]
3. **BEGIN**
4. Declaration\_section
5. Executable\_section
6. **END** &&
7. DELIMITER ;
8. DELIMITER $$
9. **CREATE** **PROCEDURE** get\_student\_info()
10. **BEGIN**
11. **SELECT** \* **FROM** Student\_table;
12. **END**$$

**MySQL procedure parameter has one of three modes:**

**IN parameter**

It is the default mode. It takes a parameter as input, such as an attribute. When we define it, the calling program has to pass an argument to the stored procedure. This parameter's value is always protected.

**OUT parameters**

It is used to pass a parameter as output. Its value can be changed inside the stored procedure, and the changed (new) value is passed back to the calling program. It is noted that a procedure cannot access the OUT parameter's initial value when it starts.

**INOUT parameters**

It is a combination of IN and OUT parameters. It means the calling program can pass the argument, and the procedure can modify the INOUT parameter, and then passes the new value back to the calling program.

How to execute a stored procedure in MySQL?

1. CALL stored\_procedure\_name (argument\_list);

1. CALL Product\_Pricing (@pricelow, @pricehigh);

Procedures with INOUT Parameter

1. DELIMITER &&
2. **CREATE** **PROCEDURE** display\_marks (INOUT var1 **INT**)
3. **BEGIN**
4. **SELECT** marks **INTO** var1 **FROM** student\_info **WHERE** stud\_id = var1;
5. **END** &&
6. DELIMITER ;

After successful execution, we can call the procedure as follows:

1. mysql> **SET** @M = '3';
2. mysql> CALL display\_marks(@M);
3. mysql> **SELECT** @M;

How to show or list stored procedures in MySQL?

1. SHOW **PROCEDURE** STATUS [LIKE 'pattern' | **WHERE** search\_condition]
2. mysql> SHOW **PROCEDURE** STATUS **WHERE** db = 'mystudentdb';

How to delete/drop stored procedures in MySQL?

1. **DROP** **PROCEDURE** [ IF EXISTS ] procedure\_name;
2. mysql> **DROP** **PROCEDURE** display\_marks;

How to create a View in MySQL?

A view is a database object that has no values. Its contents are based on the base table. It contains rows and columns similar to the real table. In MySQL, the View is a **virtual table** created by a query by joining one or more tables. It is operated similarly to the base table but does not contain any data of its own. The View and table have one main difference that the views are definitions built on top of other tables (or views). If any changes occur in the underlying table, the same changes reflected in the View also.

1. **CREATE** [OR REPLACE] **VIEW** view\_name **AS**
2. **SELECT** columns
3. **FROM** tables
4. [**WHERE** conditions];

We can see the created view by using the following syntax:

1. **SELECT** \* **FROM** view\_name;

MySQL Update VIEW

1. **ALTER** **VIEW** trainer **AS**
2. **SELECT** id, course\_name, trainer
3. **FROM** courses;

MySQL Drop VIEW

**DROP** **VIEW** [IF EXISTS] view\_name;

 How to create a Trigger in MySQL?

A trigger in MySQL is a set of SQL statements that reside in a system catalog. **It is a special type of stored procedure that is invoked automatically in response to an event**. Each trigger is associated with a table, which is activated on any DML statement such as **INSERT, UPDATE**, or **DELETE**.

A trigger is called a special procedure because it cannot be called directly like a stored procedure. The main difference between the trigger and procedure is that a trigger is called automatically when a data modification event is made against a table. In contrast, a stored procedure must be called explicitly.

Generally, **triggers are of two types** according to the [SQL](https://www.javatpoint.com/sql-tutorial) standard: row-level triggers and statement-level triggers.

**Row-Level Trigger:** It is a trigger, which is activated for each row by a triggering statement such as insert, update, or delete. For example, if a table has inserted, updated, or deleted multiple rows, the row trigger is fired automatically for each row affected by the [insert](https://www.javatpoint.com/mysql-insert), [update](https://www.javatpoint.com/mysql-update), or [delete statement](https://www.javatpoint.com/mysql-delete).

Why we need/use triggers in MySQL?

* Triggers help us to enforce business rules.
* Triggers help us to validate data even before they are inserted or updated.
* Triggers help us to keep a log of records like maintaining audit trails in tables.
* SQL triggers provide an alternative way to check the integrity of data.
* Triggers provide an alternative way to run the scheduled task.
* Triggers increases the performance of SQL queries because it does not need to compile each time the query is executed.
* Triggers reduce the client-side code that saves time and effort.
* Triggers help us to scale our application across different platforms.
* Triggers are easy to maintain.

Limitations of Using Triggers in MySQL

* MySQL triggers do not allow to use of all validations; they only provide extended validations. **For example**, we can use the NOT NULL, UNIQUE, CHECK and FOREIGN KEY constraints for simple validations.
* Triggers are invoked and executed invisibly from the client application. Therefore, it isn't easy to troubleshoot what happens in the database layer.
* Triggers may increase the overhead of the database server.

### **Types of Triggers in MySQL?**

We can define the maximum six types of actions or events in the form of triggers:

1. [**Before Insert**](https://www.javatpoint.com/mysql-before-insert-trigger)**:** It is activated before the insertion of data into the table.
2. [**After Insert**](https://www.javatpoint.com/mysql-after-insert-trigger)**:** It is activated after the insertion of data into the table.
3. [**Before Update**](https://www.javatpoint.com/mysql-before-update-trigger)**:** It is activated before the update of data in the table.
4. [**After Update**](https://www.javatpoint.com/mysql-after-update-trigger)**:** It is activated after the update of the data in the table.
5. [**Before Delete**](https://www.javatpoint.com/mysql-before-delete-trigger)**:** It is activated before the data is removed from the table.
6. [**After Delete**](https://www.javatpoint.com/mysql-after-delete-trigger)**:** It is activated after the deletion of data from the table.

When we use a statement that does not use INSERT, UPDATE or DELETE query to change the data in a table, the triggers associated with the trigger will not be invoked.

### **Naming Conventions**

Naming conventions are the set of rules that we follow to give appropriate unique names. It saves our time to keep the work organize and understandable. Therefore, **we must use a unique name for each trigger associated with a table**. However, it is a good practice to have the same trigger name defined for different tables.

1. (BEFOR | **AFTER**) table\_name (**INSERT** | **UPDATE** | **DELETE**)

Thus,

**Trigger Activation Time:** BEFORE | AFTER

**Trigger Event:** INSERT | UPDATE | DELETE

### **How to create triggers in MySQL?**

We can use the **CREATE TRIGGER** statement for creating a new trigger in MySQL. Below is the syntax of creating a trigger in MySQL:

1. **CREATE** **TRIGGER** trigger\_name
2. (**AFTER** | BEFORE) (**INSERT** | **UPDATE** | **DELETE**)
3. **ON** table\_name **FOR** EACH ROW
4. **BEGIN**
5. --variable declarations
6. --trigger code
7. **END**;
8. **CREATE** **TABLE** employee(
9. **name** **varchar**(45) NOT NULL,
10. occupation **varchar**(35) NOT NULL,
11. working\_date **date**,
12. working\_hours **varchar**(10)
13. );
14. mysql> DELIMITER //
15. mysql> **Create** **Trigger** before\_insert\_empworkinghours
16. BEFORE **INSERT** **ON** employee **FOR** EACH ROW
17. **BEGIN**
18. IF NEW.working\_hours < 0 **THEN** **SET** NEW.working\_hours = 0;
19. **END** IF;
20. **END** //
21. mysql> SHOW TRIGGERS;
22. mysql> SHOW TABLES **FROM** database\_name;
23. mysql> SHOW TRIGGERS LIKE pattern;

OR,

1. mysql> SHOW TRIGGERS **FROM** database\_name LIKE pattern;

1. mysql> SHOW TRIGGERS **WHERE** search\_condition;

OR,

1. mysql> SHOW TRIGGERS **FROM** database\_name **WHERE** search\_condition;
2. **DROP** **TRIGGER** [IF EXISTS] [schema\_name.]trigger\_name;

### **How to clear screen in MySQL?**

1. mysql> SYSTEM CLS;

### **How to create a new user in MySQL?**

1. **CREATE** USER [IF NOT EXISTS] account\_name IDENTIFIED **BY** 'password';

### **How to check USERS in MySQL?**

1. mysql> **SELECT** USER **FROM** mysql.user;

### **How to import a CSV file in MySQL?**

1. **LOAD** DATA INFILE 'C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/filename.csv'
2. **INTO** **TABLE** tablename
3. FIELDS TERMINATED **BY** ','
4. OPTIONALLY ENCLOSED **BY** '"'
5. LINES TERMINATED **BY** '\r\n'
6. **IGNORE** 1 **ROWS**;

### **How to check database size in MySQL?**

MySQL allows us to query the information\_schema.tables table to get the information about the tables and databases. It will return the information about the data length, index length, collation, creation time, etc. We can check the size of the database on the server using the below syntax:

1. **SELECT** table\_schema **AS** 'Database Name',
2. SUM(data\_length + index\_length) 'Size in Bytes',
3. ROUND(SUM(data\_length + index\_length) / 1024 / 1024, 2) 'Size in MB'
4. **FROM** information\_schema.tables
5. **WHERE** table\_schema = 'testdb'
6. **GROUP** **BY** table\_schema;
7. **SELECT** table\_name **AS** 'Table Name',
8. ROUND(((data\_length + index\_length) / 1024 / 1024), 2) **AS** 'Size in MB'
9. **FROM** information\_schema.TABLES
10. **WHERE** table\_schema = 'testdb'
11. **ORDER** **BY** (data\_length + index\_length) **DESC**;

### **How does indexing works in MySQL?**

Indexing is a process to find an unordered list into an ordered list. It helps in maximizing the query's efficiency while searching on tables in MySQL. The working of MySQL indexing is similar to the book index.

Suppose we have a book and want to get information about, say, searching. Without indexing, it is required to go through all pages one by one, until the specific topic was not found. On the other hand, an index contains a list of keywords to find the topic mentioned on pages. Then, we can flip to those pages directly without going through all pages.

### **How to set auto increment in MySQL?**

1. **ALTER** **TABLE** table\_name AUTO\_INCREMENT = value;

### **How to find the second highest salary in MySQL?**

1. **SELECT** salary
2. **FROM** (**SELECT** salary **FROM** employees **ORDER** **BY** salary **DESC** LIMIT 2) **AS** Emp **ORDER** **BY** salary LIMIT 1;
3. **SELECT** **MAX**(salary) **From** employees
4. **WHERE** salary < ( **SELECT** **Max**(salary) **FROM** employees);
5. **SELECT** **MAX**(salary)
6. **FROM** employees
7. **WHERE** salary NOT IN ( **SELECT** **Max**(salary) **FROM** employees);

### **What is the difference between TRUNCATE and DELETE in MySQL?**

* TRUNCATE is a DDL command, and DELETE is a DML command.
* It is not possible to use Where command with TRUNCATE QLbut you can use it with DELETE command.
* TRUNCATE cannot be used with indexed views, whereas DELETE can be used with indexed views.
* The DELETE command is used to delete data from a table. It only deletes the rows of data from the table while truncate is a very dangerous command and should be used carefully because it deletes every row permanently from a table.

### **What is the heap table?**

Tables that are present in memory is known as HEAP tables. When you create a heap table in MySQL, you should need to specify the TYPE as HEAP. These tables are commonly known as memory tables. They are used for high-speed storage on a temporary basis. They do not allow BLOB or TEXT fields.

### **What is BLOB and TEXT in MySQL?**

BLOB is an acronym that stands for a large binary object. It is used to hold a variable amount of data.

There are four types of the BLOB.

1. TINYBLOB
2. BLOB
3. MEDIUMBLOB
4. LONGBLOB

The differences among all these are the maximum length of values they can hold.

TEXT is a case-insensitive BLOB. TEXT values are non-binary strings (character string). They have a character set, and values are stored and compared based on the collation of the character set.

There are four types of TEXT.

1. TINYTEXT MEDIUMTEXT
2. LONGTEXT TEXT

### **What is the difference between the heap table and the temporary table?**

**Heap tables:**

Heap tables are found in memory that is used for high-speed storage temporarily. They do not allow BLOB or TEXT fields.

Heap tables do not support AUTO\_INCREMENT.

Indexes should be NOT NULL.

**Temporary tables:**

The temporary tables are used to keep the transient data. Sometimes it is beneficial in cases to hold temporary data. The temporary table is deleted after the current client session terminates.

**Main differences:**

The heap tables are shared among clients, while temporary tables are not shared.

Heap tables are just another storage engine, while for temporary tables, you need a special privilege (create temporary table).

### **What is the difference between FLOAT and DOUBLE?**

FLOAT stores floating-point numbers with accuracy up to 8 places and allocate 4 bytes. On the other hand, DOUBLE stores floating-point numbers with accuracy up to 18 places and allocates 8 bytes.

### **What are the advantages of MySQL in comparison to Oracle?**

1. MySQL is a free, fast, reliable, open-source relational database while Oracle is expensive, although they have provided Oracle free edition to attract MySQL users.
2. MySQL uses only just under 1 MB of RAM on your laptop, while Oracle 9i installation uses 128 MB.
3. MySQL is great for database enabled websites while Oracle is made for enterprises.
4. MySQL is portable.

### **What are the disadvantages of MySQL?**

1. MySQL is not so efficient for large scale databases.
2. It does not support COMMIT and STORED PROCEDURES functions version less than 5.0.
3. Transactions are not handled very efficiently.
4. The functionality of MySQL is highly dependent on other addons.
5. Development is not community-driven.

### **What is the difference between CHAR and VARCHAR?**

1. CHAR and VARCHAR have differed in storage and retrieval.
2. CHAR column length is fixed, while VARCHAR length is variable.
3. The maximum no. of character CHAR data types can hold is 255 characters, while VARCHAR can hold up to 4000 characters.
4. CHAR is 50% faster than VARCHAR.
5. CHAR uses static memory allocation, while VARCHAR uses dynamic memory allocation.

### **What is the difference between MySQL\_connect and MySQL\_pconnect?**

**Mysql\_connect:**

1. It opens a new connection to the database.
2. Every time you need to open and close the database connection, depending on the request.
3. Opens page whenever it is loaded.

**Mysql\_pconnect:**

1. In Mysql\_pconnect, "p" stands for persistent connection, so it opens the persistent connection.
2. The database connection cannot be closed.
3. It is more useful if your site has more traffic because there is no need to open and close connection frequently and whenever the page is loaded.

### **What does "i\_am\_a\_dummy flag" do in MySQL?**

The "i\_am\_a\_dummy flag" enables the MySQL engine to refuse any UPDATE or DELETE statement to execute if the WHERE clause is not present. Hence it can save the programmer from deleting the entire table my mistake if he does not use WHERE clause.

### **What are the security alerts while using MySQL?**

Install antivirus and configure the operating system's firewall.

Never use the MySQL Server as the UNIX root user.

Change the root username and password Restrict or disable remote access.

### **How to change a password for an existing user via mysqladmin?**

Mysqladmin -u root -p password "newpassword".

### **How to display the nth highest salary from a table in a MySQL query?**

Let us take a table named the employee.

**To find Nth highest salary is:**

select distinct(salary)from employee order by salary desc limit n-1,1

**if you want to find 3rd largest salary:**

select distinct(salary)from employee order by salary desc limit 2,1

### **What is the difference between NOW() and CURRENT\_DATE()?**

NOW() command is used to show current year, month, date with hours, minutes, and seconds while CURRENT\_DATE() shows the current year with month and date only.

### **What is the query to display the top 20 rows?**

SELECT \* FROM table\_name LIMIT 0,20;

### **Write a query to display the current date and time?**

SELECT NOW();

SELECT CURRENT\_DATE();

### **What is the save point in MySQL?**

A defined point in any transaction is known as savepoint.

SAVEPOINT is a statement in MySQL, which is used to set a named transaction savepoint with the name of the identifier.