**MySQL**

## *Introduction:*

SQL is a query language which is used to access, update, and manipulate the data stored in a database.

MySQL is a relational database that uses SQL to query a database. MySQL stores the existing data in a database in an organized manner.

STRUCTURE:

----------

SELECT field\_name1, field\_name2,... field\_nameN

FROM table\_name1, table\_name2...

[WHERE condition]

[GROUP BY field\_name(s)]

[HAVING condition]

[ORDER BY field\_name(s)]

[OFFSET M ][LIMIT N];

## *Keys & Constraints:*

**Candidate Key** - A set of one or more columns that can uniquely identify a record in a table are known as candidate keys, and you can use each candidate key as a Primary Key.

**Primary Key** - It is the first key used to uniquely identify a record in table

**Super Key** - A super key is a set of single or multiple keys whose combined values can uniquely identifies rows in a table.Any candidate key is a super key, but not all super keys are candidate keys.

**Alternate Key** - A candidate key other than the primary key is called an alternate key (also called as secondary key)

**Foreign Key** - Foreign keys are the column of the table used to point to the primary key of another table. It must not contain NULL values

**Composite Key** - It acts as a primary key if there is no primary key in a table. Two or more columns are used together to make a composite key.

Constraint can be used to specify the limit on the data type of table. These can be specified while creating or altering the table statement.

NOT NULL

CHECK //It is a rule that limits the values that can be placed in a column.

Ex: age INT CHECK (age >= 18)

DEFAULT //middle\_name VARCHAR(50) DEFAULT 'Unknown'

UNIQUE //to create unique key

PRIMARY KEY

FOREIGN KEY

## *Maintain USERS & Schemas:*

Syntax --> CREATE USER [IF NOT EXISTS] account\_name IDENTIFIED BY 'password';

Ex: CREATE USER IF NOT EXISTS adam@localhost IDENTIFIED BY 'jtp123456';

//If you want to give all privileges to a newly created user, execute the following command.

GRANT ALL PRIVILEGES ON \*.\* TO adam@localhost;

DROP USER john@localhost, peter@localhost;

Select user from mysql.user; //to see all users

Maintain DATABASES/Schemas

CREATE DATABASE database\_name;

SHOW DATABASES;

USE database\_name;

DROP SCHEMA/DATABASE IF EXISTS database\_name;

## *Data Definition Language (DDL):*

//cannot undone implicitly saved

DDL commands are used to define, modify, and manage the structure of database objects such as tables, indexes, and views.

CREATE TABLE employee\_table(

id int NOT NULL AUTO\_INCREMENT,

name varchar(45) NOT NULL,

occupation varchar(35) NOT NULL,

age int NOT NULL,

PRIMARY KEY (id)

);

CREATE TABLE Orders (

Order\_ID int NOT NULL PRIMARY KEY,

Order\_Num int NOT NULL,

ID int,

FOREIGN KEY (ID) REFERENCES employee\_table(id)

);

CREATE TABLE IF NOT EXISTS duplicate\_table\_name //copy other tables data

SELECT \* FROM original\_table;

SHOW TABLES;

SHOW TABLES FROM database\_name; //from another db also

DESC table\_name;

DROP TABLE table\_name; //Drop removes both data & structure of table

TRUNCATE TABLE table\_name; //removes the complete data without removing its structure. It is a part of DDL/data definition language command.

ALTER TABLE table\_name

ADD cus\_age varchar(40) NOT NULL; //add new columns

ALTER TABLE table\_name

ADD cus\_address varchar(100) NOT NULL AFTER cus\_surname, // It tells MySQL where to create the column in the table

ADD cus\_salary int(100) NOT NULL AFTER cus\_age ;

ALTER TABLE table\_name

MODIFY cus\_surname varchar(50) NULL; //MODIFY the datatype of existing column in the table

--or---

ALTER TABLE table\_name

ALTER COLUMN column\_name TYPE varchar(50) NOT NULL; //in postgress

ALTER TABLE table\_name

DROP COLUMN column\_name;

ALTER TABLE table\_name

CHANGE COLUMN cus\_surname cus\_title varchar(20) NOT NULL; //rename column

---- or ----

ALTER TABLE table\_name

RENAME COLUMN old\_column\_name TO new\_column\_name;

ALTER TABLE table\_name RENAME TO cus\_table; //rename table

--- or ---

RENAME old\_table TO new\_table;

## *Data Manipulation Language (DML):*

//can be undone

DML commands are used to manipulate data stored in database objects.

SELECT \* FROM table\_name;

INSERT INTO table\_name (id, name, occupation, age) VALUES (101, 'Peter', 'Engineer', 32);

---- or ----

INSERT INTO table\_name VALUES // to insert multiple values at a time

(102, 'Joseph', 'Developer', 30),

(103, 'Mike', 'Leader', 28),

(104, 'Stephen', 'Scientist', 45);

REPLACE INTO table\_name (id, city) VALUES (101, 'India'); //If the duplicate record found, the replace command will delete the existing row and then adds the new record in the table, otherwise acts as a standard INSERT statement is performed.

INSERT INTO table\_name2 SELECT \* FROM table\_name1 WHERE condition; //to copy all data from one table into another table

UPDATE table\_name SET email = 'mike@tutorialandexamples.com' WHERE course\_name = 'Java';

DELETE FROM table\_name WHERE emp\_id=107;

## *Data Control Language (DCL):*

DCL commands are used to control access to the database.

GRANT: Provides specific privileges to database users.

REVOKE: Removes previously granted privileges from users.

Transaction Control Language (TCL):

TCL commands are used to manage transactions within a database.

COMMIT: Saves changes made during the current transaction.

ROLLBACK: Undoes changes made during the current transaction.

SAVEPOINT: Sets a point within a transaction to which you can later roll back.

## *Clauses:*

These are used to limit the resultSet by providing conditions to the query.

FROM //to specify the table or tables from which you want to retrieve data

DISTINCT //only unique values

WHERE //used to filter the Non-grouped data.

GROUP BY //It groups the duplicate data into one

HAVING //used to filter the grouped data, should be used after group by.

ORDER BY //It arranges the data either in ascending or descending order.It should be used always as the last statement in the SQL query

Note : job column is present in select query but not in the group by query. If it is enclosed in any of the group functions like sum(sal) etc – then it is not an error. But whatever column is included in the select query must also be included in the group by query.

SELECT emp\_name, job, MAX(salary) AS "Sal"

FROM employees

GROUP BY emp\_name; //to fix add job column to GROUP BY clause or perform aggrigation on job like MAX(job) AS job

Contol Statements

------------------

SELECT IF(200>350,'YES','NO'); //IF function

SELECT lastname, IF(age>20,"Mature","Immature") As Result FROM student; //this is MySQL-specific. In standard SQL, the IF function is not used. Instead, we use a CASE expression for conditional logic.

SELECT IFNULL(NULL,5); //If the first expression is not NULL, it will return the 1st expression. Otherwise, it will return the 2nd expression

SELECT NULLIF("Hello", "404"); //if the 1st expression is equal to the 2nd expression, it returns the NULL. Otherwise, it returns the 1st expression.

## *Stored Procedures:*

A stored procedure is a program that performs a set of actions repeatedly. Several SQL statements are consolidated into a stored procedure(single unit) and execute them whenever and wherever required.

delimiter $$ //since our stored procedure contains multiple statement separated with ";"

we need to tell the MySQL shell donot try to execute the statement when it encounters a ";" like it normally would.

Instead, it should wait for "$$" (we can give any delimitter ex; //, $$ etc)

CREATE PROCEDURE myResult(IN original\_rate NUMERIC(6,2), OUT discount\_rate NUMERIC(6,2))

BEGIN

IF (original\_rate>200) THEN

SET discount\_rate=original\_rate\*0.5;

ELSE

SET discount\_rate=original\_rate;

END IF;

select discount\_rate;

END $$

delimiter ; //now that we're done defining our procedure, change the delimiter back to ";"

CALL procedure\_name(x, y); //CALL statement to call a stored procedure.

DELIMITER //

CREATE PROCEDURE GetEmployeeName (IN emp\_id INT)

BEGIN

DECLARE emp\_name VARCHAR(100);

SELECT employee\_name INTO emp\_name

FROM employees WHERE employee\_id = emp\_id; //Retrieves the employee\_name corresponding to the input emp\_id and assigns it to the variable emp\_name.

SELECT emp\_name AS employee\_name; //Returns the employee\_name as the result

END //

DELIMITER ;

CALL GetEmployeeName(123);

## *FUNCTIONS:*

Functions enable users to perform operations on data, manipulate strings, perform calculations, work with dates and times, and more.

functions are named, reusable SQL queries that take zero or more parameters as input and return a single value. They are used to encapsulate logic that can be reused throughout queries within a database.

--- String Functions

CONCAT(): Concatenates two or more strings.

SUBSTRING(): Returns a substring from a string.

UPPER(): Converts a string to uppercase.

LOWER(): Converts a string to lowercase.

LENGTH(): Returns the length of a string.

TRIM(): Removes leading and trailing spaces from a string.

REPLACE(): Replaces occurrences of a specified string.

LEFT(), RIGHT(): Returns the leftmost or rightmost characters of a string.

--- Numeric Functions

ABS(): Returns the absolute value of a number.

ROUND(): Rounds a number to a specified number of decimal places.

CEILING(), FLOOR(): Returns the smallest integer greater than or equal to, or the largest integer less than or equal to, a given number.

RAND(): Returns a random number.

MOD(): Returns the remainder of a division operation.

--- Date and Time Functions

NOW(): Returns the current date and time.

CURDATE(): Returns the current date.

CURTIME(): Returns the current time.

DATE\_FORMAT(): Formats a date as specified.

DATEDIFF(): Calculates the difference between two dates.

DATE\_ADD(), DATE\_SUB(): Adds or subtracts a specified time interval from a date.

--- Aggregate Functions

SUM(): Calculates the sum of values.

AVG(): Calculates the average of values.

COUNT(): Counts the number of rows.

MAX(), MIN(): Returns the maximum or minimum value.

GROUP\_CONCAT(): concatenates values from multiple rows into a single string.

--- Control Flow Functions

IF(), CASE: Conditional functions used to control the flow of execution based on conditions.

--- Miscellaneous Functions

NULLIF(): Returns NULL if two expressions are equal.

COALESCE(): Returns the first non-NULL value in a list of expressions.

IFNULL(): Returns a specified value if the expression is NULL.

Stored procedure vs Function

1) Functions can only have input parameters, whereas procedures can have either input or output parameters.

2) A function has a return type and returns a value, whereas A procedure does not have a return type But it returns values using the OUT parameters.

3) Functions can be called from procedures, but procedures cannot be called from functions.

4) Exceptions can be handled in try-catch blocks within procedures, but try-catch blocks cannot be used within functions.

5) Procedures cannot be used in SELECT statements, but functions can be embedded in SELECT statements

## *OPERATORS:*

// AND/OR --> AND - both conditions to be true, OR - any one condition has to be true

SELECT \* FROM cus\_tbl WHERE cus\_firstname = 'Abijeet' AND/OR cus\_id > 3;

MySQL does not contain built-in Boolean or Bool data type. They provide a TINYINT data type instead of Boolean or Bool data types

// IN –-> it is used for evaluating multiple values

SELECT \* FROM officers WHERE officer\_name IN ('Ajeet', 'Vimal', 'Deepika');

SELECT \* FROM officers WHERE officer\_name NOT IN ('Ajeet','Vimal','Deepika');

//LIKE –-> used for pattern matching

% - matches 0 or n characters

\_ - matches exactly one character

SELECT officer\_name FROM officers WHERE address LIKE 'Luck%';

//BETWEEN –-> used for searching based on range of values

SELECT \* FROM officers WHERE salary BETWEEN 10000 AND 300000;

//IS –-> it is used to compare nulls

SELECT \* FROM officers WHERE commision IS NULL;

SELECT \* FROM officers WHERE officer\_name IS NOT NULL;

## *JOINS:*

Joins are used when we need to fetch the data from multiple tables based on the relationship between the fields of the tables

------

1) CROSS JOIN is also known as CARTESIAN JOIN, produces a Cartesian product of two tables. It combines each row from the first table with every row from the second table, resulting in a result set that is the product of the two tables.

SELECT customer.customer\_id, customer.cust\_name, orders.order\_id, orders.price

FROM

customer CROSS JOIN orders;

//If table1 has 3 rows and table2 has 4 rows, a cross join between them will produce 12 rows (3 \* 4 = 12)

----->In the absence of a WHERE condition the CARTESIAN JOIN will behave like a CARTESIAN PRODUCT . i.e., the number of rows in the result-set is the product of the number of rows of the two tables.

In the presence of WHERE condition this JOIN will function like a INNER JOIN. Generally speaking, Cross join is similar to an inner join where the join-condition will always evaluate to True

2) INNER JOIN selects all rows from both the tables as long as the condition satisfies.

SELECT students.stud\_fname, students.city, technologies.technology

FROM

students INNER JOIN technologies ON students.student\_id = technologies.tech\_id; //INNER JOIN or JOIN or EQUI-JOIN

3) Left Joins returns all records from the left-hand table and only the matching records from the right side table

SELECT customers.customer\_id, customers.cust\_name, customers.occupation, orders.price, orders.date

FROM

customers LEFT JOIN orders ON customers.customer\_id = orders.customer\_id; //LEFT OUTER JOIN

---> the table customers and orders have the same column name, which is customer\_id. In that case, it can also be used with the USING clause to access the records.

So, instead of table1.column\_name = table2.column\_name, you can use the USING(column\_name) clause.

4) Right Join returns all rows from the right-hand table, and only those results from the left table that fulfilled the join condition.

SELECT customers.customer\_id, Customers.cust\_name, orders.price, orders.date

FROM

customers RIGHT JOIN orders USING(customer\_id) //RIGHT OUTER JOIN

ORDER BY customer\_id;

5) FULL OUTER JOIN contains all rows from both tables, but not like cross join. If a row from one table doesn't have a matching row in the other table, NULL values are used for the columns from the missing row.

SELECT customers.customer\_id, Customers.cust\_name, orders.price, orders.date

FROM

customers FULL OUTER JOIN orders ON customers.customer\_id = orders.customer\_id; //OUTER JOIN or FULL JOIN

//If table1 has 5 rows and table2 has 4 rows, a full join between them will produce at least 5 rows in the result set, including all rows from both tables, and NULL values for columns from the table with missing rows.

6) SELF JOIN is a join where a table is joined with itself. Self Join uses table aliases, This is used to compare values in a column with other values in the same column in the same table

SELECT a.ROLL\_NO , b.NAME

FROM Student a, Student b //No join command inbetween

WHERE a.ROLL\_NO < b.ROLL\_NO;

## *Normalization:*

Normalization is the process to eliminate data redundancy and enhance data integrity in the table. It also helps to organize the data in the database.

1NF:

1. Every column/cell need to have single value (atomicity of the table is 1)

2. Each row should be unique through single/multiple columns. Not mandatory to have PK

2NF:

1. the table has to be in 1NF

2. All non-key attributes must be fully dependent on candidate-keys

- if non-key columns partially dependent on candidate-key then split them into diff tables

3. Every table should have a PK and relationship between the tables should be formed using FK

3NF:

1. the table has to be in 2NF

2. Avoid transitive dependency for non-key attributes (non-key attributes should not depend on other non-key attributes in a table)

//Candidate key is a set of one or more columns that can uniquely identify a record in a table, and you can use each candidate key as a Primary Key.

//non-key columns -> columns which are not part of candidate-key or PK

//partially dependent -> if a Candidate-key is a combination of 2 or more columns, then every non-key column should be fully dependent on all Candidate-key columns.

## *Indexing*

//improves DB performance

Indexing offers an efficient way to quickly access the records from the database files stored on the disk drive. It optimizes the database querying speed by serving as an organized lookup table with pointers to the location of the requested data. Indexing improves database performance by minimizing the number of disc visits required to fulfill a query.

Index Architectures:

Indexes typically use a balanced-tree (i.e., B-tree) structure to store the data in a sorted manner. Usage of B-tree also enhances the speed of data search and access operations in a database. The data structure associated with the index has two fields. The first field stores the values of the database column the index is sorting. The other field stores a group of pointers helping to identify the disk location of the column values.

Three methods of indexing -> Clustered or Primary Indexing, Non-clustered or Secondary Indexing

Ex: create INDEX index\_name ON table\_name (column\_name);

create INDEX index\_name ON table\_name (column\_name1, column\_nmae2);

## *IMP Questions:*

1) UNION operator allows us to combine two or more results from multiple SELECT queries into a single result set. It has a default feature to remove the duplicate rows from the tables

- The number and order of the columns should be the same in all queries.

- Union and Join are different.

SELECT City FROM student1

UNION

SELECT City FROM student2 ORDER BY City;

* The UNION ALL operator combines two or more results from multiple SELECT queries and returns all records into a single result set. It does not remove the duplicate rows from the output

2) **Unique Key** - A unique key is a constraint that ensures the values in a column or combination of columns are unique across all rows in table.

Example:

| Roll\_no. | Name | Address | Personal\_id |
| --- | --- | --- | --- |
| 1 | John | Pune | John@123 |
| 4 | Merry | Mumbai | NULL |
| 18 | Sheero | Nagpur | 32467 |

In the above example, **Roll\_no** can act as a primary since it is unique and there is no null value. **Personal\_id** can act as a unique key because it is also unique and for one time we can keep its value as null

| **Primary Key** | **Unique Key** |
| --- | --- |
| A primary key cannot have a NULL value. | Unique keys can have one NULL value. |
| The table can have only one primary key. | The table can have more than one unique key. |
| Any change in primary key value may necessitate updates in related tables to maintain referential integrity. | A unique key can be updated or deleted; it doesn’t affect the table’s integrity constraints. |
| In some databases, the primary key can generate a clustered index by default. | In some databases, a unique key can generate a non-clustered index by default. |

3) Limit query is used to restrict the number of rows returns from the result set, rather than fetching the whole set in the MySQL database.

SELECT [column\_list] FROM table\_name LIMIT offset, row\_count;

//offset = tells which line to begin with the data recovery (offset of row1 is '0'), //row\_count = tells no. of rows to be returned.

4)find Nth highest salary in each table **with & without using limit**

**using LIMIT :**

SELECT DISTINCT salary FROM employees ORDER BY salary DESC LIMIT n-1, 1;

**using SUBQUERIES :**

SELECT salary FROM employee e1 WHERE N = (

SELECT COUNT(DISTINCT salary)

FROM employee e2

WHERE e2.salary >= e1.salary); //gives rank for each salary, and we will fetch req no.

5)find the duplicate values in a table

SELECT name, COUNT(name) AS duplicate\_count

FROM student\_contacts

GROUP BY name

HAVING COUNT(name) > 1;

6)Delete Duplicate Record from the table student\_contacts based on the email column Using Inner Join //only records with the same email but different IDs are selected for deletion.

DELETE S1 FROM

student\_contacts AS S1 INNER JOIN student\_contacts AS S2

WHERE S1.id < S2.id AND S1.email = S2.email; //use < to keep the lowest id in the table

7)REGEXP\_INSTR() Function returns the index value of the substring by matching the given string. This function returns 0 if no match is found Else, it returns 1.

SELECT REGEXP\_INSTR ('BCA MCA PGDCA BBA MBA BCA', 'BCA', 1, 2, 0) AS BCA\_index\_2; //1=pos at which the search will starts, 2=occurrence of a match, 0=to specify which type of position of the matched substring will return i.e; 0=returns the position of the matched substring's first character & 1=return the position following the matched substring

8)Back up MySQL Database Using mysqldump

syntax -> mysqldump -u [username] -p [database\_name] > [backup\_file\_name].sql //When prompted, enter your password and press Enter to confirm

If you want to take a full backup i.e., all databases, procedures, routines, and events without interrupting any connections:

mysqldump -u [username] -p -A -R -E --triggers --single-transaction > [backup\_file\_name].sql

-A For all databases (you can also use --all-databases)

-R For all routines (stored procedures & triggers)

-E For all events

--single-transaction Without locking the tables i.e., without interrupting any connection (R/W).

If you want to take a backup of only specified database(s):

mysqldump -u [username] -p [database\_name] [other\_database\_name] -R -e --triggers --single-transaction > database\_backup.sql

9) Transaction Isolation Levels in DBMS

The level of isolation defines the degree to which the transaction is isolated during the modification of the data, and the other transaction in the database does these modifications. We can define transaction isolation by the following phenomena.

1. **Dirty Read:** A dirty read is a type of situation that occurs when a transaction reads the data, and that data has not been committed yet. Let us understand this with an example. Let's say transaction-1 updates a row, and that transaction 1 is not committed yet. Then transaction 2 reads the updated row. If transaction 1 rolls back the changes, then transaction-2 will read all the data, and these data will be considered as never existing.
2. **Non-Repeatable Read:** The non-repeatable Read occurs when the transaction reads a single row multiple times and gets a different value on each Read. Let's take an example. Suppose transaction1 reads the row data. But due to a concurrency issue, transaction 2 updates the same data and is committed. Now transaction 2 reads the same row again and gets a different value this time.
3. **Phantom Read:** When the two queries are executed, but they retrieve the two rows differently, at that time, phantom Read occurs. Let's take an example. Suppose transaction 1 retrieves a set of rows, and these rows are satisfied by some conditions. Now transaction 2 generates some new rows that match the search criteria of transaction 1. If transaction 1 executes the statement that reads the row, it gets a different row at different times.

Isolation Levels

Based on the above different phenomena, SQL defines into four isolation levels.

1. **Read Uncommitted:** It is the lowest in the isolation level, **allowing access to data before modifications are performed/ not yet committed**. Thereby allowing dirty reads.
2. **Read Committed:** This isolation level **allows you access to information only after it has been committed to the database**. Thus, it does not allow dirty read. The transaction holds a read or write lock on the current row, and thus prevents other transactions from reading, updating, or deleting it.
3. **Repeatable Read:** This is the most restrictive isolation level. The transaction holds **read locks** on all rows it references and **writes locks** on referenced rows for updates and deletes actions. Since other transactions cannot read, update or delete these rows, consequently, it avoids non-repeatable Read.
4. **Serializable:** This is the highest isolation level. Typically, this isolation mode would **lock the whole table**, to prevent any other transactions from inserting or reading data from it. Serializable execution is defined to be an execution of operations in which concurrently executing transactions appears to be serially executing.

10) **List all the customer IDs and last names who have made more than 2 orders**

Select customer\_id,first\_name,last\_name From Customer

Where customer\_id in

(Select customer\_id From Order group by customer\_id having count(\*) > 2 )

11) **ACID Properties in DBMS**

A [**transaction**](https://www.geeksforgeeks.org/sql-transactions/) is a single logical unit of work that accesses and possibly modifies the contents of a database. Transactions access data using read and write operations.   
To **maintain consistency** in a database, before and after the transaction, certain properties are followed. These are called **ACID** properties.

