**Show databases;**

**Create database actors;**

**Show tables;**

**Describe actors;**

**Create table**

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

create table actors(

id int,

firstname varchar(20),

lastname varchar(20),

age int,

networth int

);

**Insert table**

—---------------------------------------------------------------------------------------------------------

insert into actors(id, firstname, lastname, age, networth) values(1,"jr", "ntr", 40, 200);

insert into actors(firstname,lastname,dob,networth)

values("johnny", "depp", "1963-06-09",200.00),

("Rajnikanth", "", "1965-12--18",150.00),

("ram", "charan", "1990-12--18",280.00),

("jr", "ntr", "1988-12--18",210.00);

**Select commands**

—----------------------------------------------------------------------------------------------------------------------------

select \* from actors;

select \* from actors where lastname="ntr";

select firstname, lastname from actors;

select firstname, lastname from actors where firstname="brad";

select firstname, lastname from actors where networth > 200;

select firstname, lastname from actors where networth > 0;

select \* from actors where networth >= 210 and networth < 250;

select \* from actors where firstname = "brad" or networth >200;

select \* from actors where not networth = 210;

**explain analyze select \* from employee where salary >100;**

**SELECT \* FROM employees**

**WHERE age > 25 AND department = 'Sales';**

**SELECT \* FROM products**

**WHERE category = 'Electronics' OR price > 500;**

**SELECT \* FROM orders**

**WHERE NOT order\_status = 'Cancelled';**

**SELECT \* FROM customers**

**WHERE country IN ('USA', 'Canada', 'Mexico');**

**select \* from employee where salary in(1300, 1400);**

**SELECT \* FROM products**

**WHERE price BETWEEN 100 AND 500;**

**select \* from employee where salary between 1300 and 1400;**

**SELECT \* FROM products**

**WHERE product\_name LIKE 'Electronics%';**

**select \* from employee where name like 'h%';**

**This query retrieves products with names starting with "Electronics".**

**SELECT \* FROM customers**

**WHERE email LIKE '%gmail.com';**

**select \* from employee where name like '%h';**

**This query retrieves customers with email addresses ending in "gmail.com".**

**SELECT \* FROM employees**

**WHERE first\_name LIKE '%a%';**

**This query retrieves employees with first names containing the letter "a".**

**SELECT \* FROM products**

**WHERE product\_code LIKE 'P\_5678';**

**update actors set networth = 300 where firstname = "ram";**

**update actors set networth = 280 where firstname = "ram";**

**insert into employee(name,id,salary)values("harshit",101,1200);**

**update employee set salary = salary\*1.2 where name='harshit';**

**delete from actors where networth = 280;**

**delete from actors where 1 > 0;**

**insert into actors values**

**("Rajnikanth", "anna", "1965-12--18",150.00),**

**("ram", "charan", "1990-12--18",280.00),**

**("jr", "ntr", "1988-12--18",210.00),**

**("brad", "pitt", "1965-12--18",250.00),**

**("srk", "", "1970-12--18",580.00),**

**("ranveer", "singh", "1988-12--18",230.00);**

**Limit and Offset**

**—------------------------------------------------------------------------------------------------------------------**

**select \* from actors limit 2;**

**select \* from actors limit 4;**

**select \* from actors limit 4 offset 1;**

**select \* from actors limit 2 offset 1;**

**select \* from actors limit 1 offset 1;**

**select \* from actors limit 1 offset 2;**

**Alter Commands**

**—------------------------------------------------------------------------------------------------------------------**

**alter table employee add column isActive boolean;**

**update employee set isActive=true where 1>0;**

**update employee set salary = salary \* 1.1 where isActive=true;**

**alter table actors add column country;**

**alter table actors drop column country;**

**alter table actors add column country varchar(20);**

**alter table employee add column byMistake int;**

**alter table employee drop column byMistake;**

**update actors set country="India" where not networth=250;**

**update actors set country="USA" where networth=250;**

**Order by commands**

**—-------------------------------------------------------------------------------------------------------------**

**select \* from actors order by firstname;**

**select \* from actors order by firstname desc;**

**select \* from actors order by networth;**

**select \* from actors order by networth desc;**

**SELECT first\_name, last\_name, salary**

**FROM Employees**

**ORDER BY salary DESC;**

**SELECT product\_name, category, price**

**FROM Products**

**ORDER BY category ASC, price DESC;**

**Put New Data**

**-----------------------------------------------------------------------------------------------------------------**

**create table product(id int, name varchar(20), sales int, category varchar(20));**

**mysql> insert into product(id,name,sales,category)values(1,'iphone12',1000,'mobile');**

**Query OK, 1 row affected (0.01 sec)**

**mysql> insert into product(id,name,sales,category)values(2,'galaxy s2',2000,'mobile');**

**Query OK, 1 row affected (0.00 sec)**

**mysql> insert into product(id,name,sales,category)values(3,'mac m2',8000,'laptop');**

**Query OK, 1 row affected (0.00 sec)**

**mysql> insert into product(id,name,sales,category)values(4,'mac m1',5000,'laptop');**

**Aggregate Function**

**-----------------------------------------------------------------------------------------------------------------**

**select sum(sales) from product;**

**select count(\*) from product;**

**select avg(sales) from product;**

**select min(sales) as minimum\_sales from product;**

**select max(sales) as maximum\_sales from product;**

**Group by Commands**

**-----------------------------------------------------------------------------------------------------------------**

**1. Calculate the average sales for each category in the "product";**

**select category,avg(sales) from product group by category;**

**2. Calculate Maximum sales value for each category in the "product" table.**

**select category,max(sales) from product group by category;**

**3. Calculate Minimum sales value for each category in the "product" table.**

**select category,min(sales) from product group by category;**

**4. Calculate total sales value for each category in the "product" table.**

select category,sum(sales) from product group by category;

**Having Command**

**-----------------------------------------------------------------------------------------------------------------**

The HAVING clause in SQL is used to filter the results of a query that includes the GROUP BY clause.

It allows you to apply conditions to the aggregated data that are generated by the GROUP BY operation.

select category,sum(sales) from product group by category having sum(sales) > 3000;

**You want to find movies with an average rating greater than 8:**

SELECT movie\_name, AVG(rating) AS avg\_rating

FROM Movies

GROUP BY movie\_name

HAVING AVG(rating) > 8;

**Keys**

**First Way Primary Key**

**create table employee(p\_id int not null primary key, p\_name varchar(20) not null);**

**Second Way Primary Key**

**mysql> create table employee(**

**-> e\_id int not null unique,**

**-> e\_name varchar(20) not null,**

**-> primary key(e\_id)**

**-> );**

**Third Way Primary Key**

**mysql> create table employee(**

**-> name varchar(20)**

**-> );**

**alter table employee add column id int;**

**alter table employee add constraint primary key(id);**

**Foreign Key**

**First Create a Department table**

**CREATE TABLE department (**

**department\_id INT PRIMARY KEY,**

**department\_name VARCHAR(255)**

**);**

**First Create a Student table**

**CREATE TABLE student (**

**student\_id INT PRIMARY KEY,**

**student\_name VARCHAR(255),**

**department\_id INT,**

**FOREIGN KEY (department\_id) REFERENCES department(department\_id)**

**);**

**INSERT DATA INTO**

**INSERT INTO department (department\_id, department\_name)**

**VALUES (1, 'Computer Science'),**

**(2, 'Electrical Engineering'),**

**(3, 'Mechanical Engineering'),**

**(4, 'Civil Engineering'),**

**(5, 'Biology'),**

**(6, 'Business Administration');**

**INSERT INTO student (student\_id, student\_name, department\_id)**

**VALUES (101, 'John Doe', 1),**

**(102, 'Jane Smith', 2),**

**(103, 'Michael Johnson', 1),**

**(104, 'Emily Brown', 3),**

**(105, 'Daniel Wilson', 1),**

**(106, 'Sophia Martinez', 4),**

**(107, 'Liam Taylor', 2);**

**INNER JOIN:**

This query retrieves

Students along with their associated department information.

It only returns rows where there is a match between the department\_id in the student table and the department\_id in the department table.

**SELECT s.student\_id, s.student\_name, d.department\_id, d.department\_name**

**FROM student s**

**INNER JOIN department d ON s.department\_id = d.department\_id;**

**LEFT JOIN:**

This query retrieves all students and their associated department information.

If a student doesn't have a matching department, the result will still include that student with NULL values for department information.

SELECT s.student\_id, s.student\_name, d.department\_id, d.department\_name

FROM student s

LEFT JOIN department d ON s.department\_id = d.department\_id;

Add a student with no dept\_id and then again RUN.

insert into student(student\_id, student\_name)values(1,'harshit');

RIGHT JOIN:

This query retrieves all departments and the students associated with each department.

If a department doesn't have any matching students.

The result will still include that department with NULL values for student information.

SELECT s.student\_id, s.student\_name, d.department\_id, d.department\_name

FROM student s

RIGHT JOIN department d ON s.department\_id = d.department\_id;

FULL OUTER JOIN: NOT SUPPORTED BY MYSQL

This query retrieves all students and all departments.

Showing the relationships between students and departments.

If there are unmatched records on either side, they will be included with NULL values.

SELECT s.student\_id, s.student\_name, d.department\_id, d.department\_name

FROM student s

LEFT JOIN department d ON s.department\_id = d.department\_id

UNION

SELECT s.student\_id, s.student\_name, d.department\_id, d.department\_name

FROM department d

LEFT JOIN student s ON s.department\_id = d.department\_id

WHERE s.student\_id IS NULL;

INDEXING IN MYSQL

Indexing in MySQL is a database optimization technique used to improve the performance of queries by creating data structures.

Primary Key Index: Automatically created when you define a primary key for a table. It ensures that each value in the primary key column is unique and is used for efficient row retrieval.

Benefits of Indexing:

Faster data retrieval: Indexes allow the database engine to quickly locate specific rows, reducing the need for full table scans.

Improved query performance: Queries that involve indexed columns can be executed faster.

**Creating Indexes:**

**create table employee(id int, name varchar(10));**

Indexes can be created during table creation

or

Added later using the CREATE INDEX statement.

**create index idx\_id on employee(id);**

**show index from employee;**

**explain analyze select \* from employee;**

**Dropping Indexes:**

Indexes can be dropped using the DROP INDEX statement.

**drop index idx\_id on student;**

**VIEWS:**

* Views can be used to encapsulate complex queries or joins into a single, easily accessible object.
* A view is a virtual table that is created by selecting data from one or more existing tables.
* A view does not store data itself; instead
* it provides a way to present data from one or multiple tables in a structured and organized manner.
* Views can be queried just like regular tables, and they can simplify complex queries, enhance security, and improve maintainability.

Creating Views:

**Views are created using the CREATE VIEW statement, which defines the columns and the query that the view is based on.**

CREATE VIEW view\_name AS

SELECT column1, column2

FROM original\_table

WHERE condition;

create view v1 as SELECT s.student\_id, s.student\_name, d.department\_id, d.department\_name FROM student s LEFT JOIN department d ON s.department\_id = d.department\_id UNION SELECT s.student\_id, s.student\_name, d.department\_id, d.department\_name FROM department d LEFT JOIN student s ON s.department\_id = d.department\_id WHERE s.student\_id IS NULL;

select \* from v1;

Using Views:

Once a view is created, you can query it like you would query a regular table.

SELECT \* FROM s1;

drop view s1;

show full tables where table\_type = 'VIEW';

To list all views in a MySQL database,

DELIMITER

delimiter .

delimiter ;

Stored Procedure

CREATE TABLE orders (

order\_id INT AUTO\_INCREMENT PRIMARY KEY,

customer\_id INT,

product\_id INT,

price DECIMAL(10, 2)

);

INSERT INTO orders (customer\_id, product\_id, price) VALUES

(1, 101, 25.00),

(1, 102, 35.00),

(2, 103, 50.00),

(3, 101, 25.00),

(3, 104, 40.00);

Example 1: Retrieve Customer's Order Count

Option - 1

select count(\*) from orders;

Option - 2

Delimiter .

create procedure getTotalOrders()

-> begin

-> select count(\*) from orders;

-> end

-> .

call getTotalOrders().

Delimiter ;

Let's create a stored procedure that retrieves the total number of orders for a given customer.

Option 1

select count(\*) from orders where customer\_id=1;

Option 2

-> create procedure getCustomerOrderCount(in c\_id int)

-> begin

-> select count(\*) from orders where customer\_id=c\_id;

-> end

-> .

call getCustomerOrderCount(2).

Let's create a stored procedure that calculates the total price of a customer's orders.

select sum(price) from orders where customer\_id=1;

-> create procedure totalPrice(in id int)

-> begin

-> select sum(price) from orders where customer\_id = id;

-> end

-> .

> call totalPrice(1);

Saving Thing in Output Variable

> create procedure f1(out result int)

-> begin

-> select count(\*) into result from orders;

-> end

-> //

mysql> delimiter ;

mysql> call f1(@x);

mysql> select @x;

Let’s Calculate this Area of Rectangle

mysql> create procedure calArea(in len int, in wid int, out area int)

-> begin

-> declare temp int;

-> set temp = len \* wid;

-> select temp into area;

-> end

-> .

mysql> delimiter ;

mysql> CALL calArea(10, 5, @result);

SELECT @result;

**Best Syntax**

**DELIMITER //**

**CREATE PROCEDURE calArea(IN len INT, IN wid INT, OUT area INT)**

**BEGIN**

**DECLARE temp INT;**

**SET temp = len \* wid;**

**SELECT temp INTO area;**

**END;**

**Composite Key and Candidate Key**

* Primary keys must contain UNIQUE values, and cannot contain NULL values.
* A table can have only ONE primary key in the table.
* So, while you cannot have two separate primary keys in a single table. You can have a composite primary key that consists of multiple columns.

**mysql> CREATE TABLE orders (**

**-> customer\_id INT,**

**-> order\_number INT,**

**-> order\_date DATE,**

**-> /\* Other columns related to the order \*/**

**-> PRIMARY KEY (customer\_id, order\_number)**

**-> );**

**mysql> INSERT INTO orders (customer\_id, order\_number, order\_date)**

**-> VALUES**

**-> (1, 1001, '2023-08-19'),**

**-> (2, 1002, '2023-08-20'),**

**-> (1, 1003, '2023-08-21');**

**mysql> describe orders;**

**We can have multiple FK although**

**CREATE TABLE customers (**

**customer\_id INT PRIMARY KEY,**

**customer\_name VARCHAR(50)**

**);**

**CREATE TABLE products (**

**product\_id INT PRIMARY KEY,**

**product\_name VARCHAR(50)**

**);**

**CREATE TABLE orders (**

**order\_id INT PRIMARY KEY,**

**customer\_id INT,**

**product\_id INT,**

**order\_date DATE,**

**FOREIGN KEY (customer\_id) REFERENCES customers(customer\_id),**

**FOREIGN KEY (product\_id) REFERENCES products(product\_id)**

**);**

**How to add foreign key using alter command**

**CREATE TABLE customers (**

**customer\_id INT PRIMARY KEY,**

**customer\_name VARCHAR(50)**

**);**

**CREATE TABLE orders (**

**order\_id INT PRIMARY KEY,**

**order\_date DATE,**

**customer\_id int**

**);**

**alter table orders add constraint foreign key(customer\_id) references customers(customer\_id);**

Functional Dependency

Super Key:

A super key is a set of one or more attributes (columns) that can uniquely identify a tuple (row) in a relation (table).

It doesn't necessarily have to be minimal, meaning it can contain more attributes than required to uniquely identify a tuple.

Example:

Consider a table "Students" with attributes: StudentID, FirstName, LastName, and Email.

A super key could be {StudentID}, {FirstName, LastName}, or even {StudentID, FirstName, LastName}.

Candidate Key:

A candidate key is a minimal super key, meaning it's a set of attributes that can uniquely identify a tuple without any unnecessary attributes.

Example:

A set of attributes that uniquely identify the records in the table is called the Candidate key.

Every SQL table has at least one candidate key.

A table can have multiple candidate keys, but only one primary key.

In our example, both Student\_ID and Roll\_no are candidate keys.

We then select Student\_ID as the primary key.

Alternate Key:

Alternate keys are candidate keys that are not chosen as Primary Keys.

In our example, the attribute Roll\_no is an Alternate key.

**Key Attribute:**

**A key attribute is an attribute that is used to uniquely identify a record (row) within a database table.**

**It helps ensure that each record has a distinct identity.**

**In a relational database, a primary key is a specific type of key attribute.**

**It uniquely identifies each record in the table and serves as the main method of referencing and accessing the data.**

**Example: In a "Students" table, the "StudentID" could be the primary key, serving as a key attribute.**

**Non-Key Attribute:**

**A non-key attribute is an attribute in a database table that is not part of the primary key.**

**It provides additional information about the record but is not used to uniquely identify records.**

**Example: In the same "Students" table, attributes like "StudentName," "DateOfBirth," and "Address" could be non-key attributes.**

**Prime Attribute:**

**A prime attribute is an attribute that is part of the primary key.**

**Meaning it contributes to the uniqueness of a record in a table.**

**In essence, all key attributes, whether they are single or composite keys, are considered prime attributes.**

**Example: In a "Sales" table, if the primary key is composed of both "InvoiceID" and "ProductID," both of these attributes are prime attributes.**

**Non-Prime Attribute:**

**A non-prime attribute is an attribute that is not part of the primary key.**

**It includes both non-key attributes and attributes that are part of secondary keys (keys other than the primary key).**

**Example: In the "Sales" table, if "SalesDate" is not part of the primary key but provides information about when a sale occurred, it is a non-prime attribute.**

Normalization

**1st NF**

A relation (table) in a relational database is said to be in 1NF if it meets the following criteria:

**Atomic Values:** Each attribute (column) in a table must hold only atomic (indivisible) values.This means that a column should not contain lists, arrays, or nested structures. Each value should be simple and single.

**2nd NF**

**A relation is said to be in 2NF if it meets the following criteria:**

* **It is already in 1NF.**
* **Table should not have any partial dependency.**
* When a non prime attribute depend on part of prime attribute
* Instead depending on entire prime attribute when it depend on part of prime attribute.

**3rd NF**

**A relation is said to be in 3NF if it meets the following criteria:**

**It is already in 2NF.**

**Table should not have any Transitive dependency.**

* When a non prime attribute finds another non prime attribute.

**| TeacherID | TeacherName | Department | DepartmentHead |**

**|------------|--------------------------|-------------------|-------------------------|**

**| 101 | John | Math | Emily |**

**| 102 | Jane | Science | Michael |**

**| 103 | Michael | Math | Emily |**

**In this example,**

**DepartmentHead depends on Department.**

**And Department depends on TeacherID.**

**This is a transitive dependency.**

**To resolve it,**

**You would move the DepartmentHead attribute to a separate table**

**where it directly depends on the primary key (TeacherID):**

**Teachers Table:**

**TeacherID TeacherName Department**

**101 John Math**

**102 Jane Science**

**103 Michael Math**

**Departments Table:**

**Department DepartmentHead**

**Math Emily**

**Science Michael**

SQL Sub Queries Concepts

**mysql> CREATE TABLE students (**

**-> student\_id INT PRIMARY KEY,**

**-> student\_name VARCHAR(50)**

**-> );**

**Query OK, 0 rows affected (0.01 sec)**

**mysql> CREATE TABLE courses (**

**-> course\_id INT PRIMARY KEY,**

**-> course\_name VARCHAR(50)**

**-> );**

**Query OK, 0 rows affected (0.02 sec)**

**mysql> CREATE TABLE student\_courses (**

**-> student\_course\_id INT PRIMARY KEY,**

**-> student\_id INT,**

**-> course\_id INT,**

**-> FOREIGN KEY (student\_id) REFERENCES students (student\_id),**

**-> FOREIGN KEY (course\_id) REFERENCES courses (course\_id)**

**-> );**

**INSERT INTO students (student\_id, student\_name) values(1,'suraj'),(2,'abhi'),(3,'moni');**

**INSERT INTO courses (course\_id, course\_name)**

**VALUES (101, 'Mathematics'),(102,'History'),(103,'Science');**

**INSERT INTO student\_courses (student\_course\_id, student\_id, course\_id)**

**VALUES (1, 1, 101),**

**(2, 1, 102),**

**(3, 2, 102),**

**(4, 2, 103),**

**(5, 3, 101),**

**(6, 3, 103);**

**Q - 1 - Total Number of enrolled\_courses by student\_id = 1 ?**

Answer - select count(\*) enrolled\_courses from student\_courses where student\_id = 1;

**Q - 2 - Total Number of enrolled\_courses by Each Student ?**

Answer - select student\_id,count(\*) enrolled\_courses from student\_courses group by student\_id;

**Q - 3 - Retrieve student names along with the total number of courses they are enrolled in.**

**Using JOINS**

**SELECT s.student\_name, COUNT(sc.student\_id) AS enrolled\_courses**

**FROM students s**

**JOIN student\_courses sc ON s.student\_id = sc.student\_id**

**GROUP BY s.student\_id;**

**Using Sub-Query**

**SELECT student\_name,**

**(SELECT COUNT(\*) FROM student\_courses WHERE student\_id = students.student\_id) AS enrolled\_courses**

**FROM students;**

**Q 4 - Retrieve student id who are enrolled in a specific course. Example - 102**

Answer - select student\_id from student\_courses where course\_id = 102;

**Q 5 - Retrieve student names who are enrolled in a specific course. Example - 102**

**Using Sub-Query**

**select student\_id from student\_courses where course\_id = 102;**

**+------------+**

**| student\_id |**

**+------------+**

**| 1 |**

**| 2 |**

**+------------+**

**2 rows in set (0.00 sec)**

**select student\_name from students where student\_id in (1,2);**

**+--------------+**

**| student\_name |**

**+--------------+**

**| suraj |**

**| abhi |**

**+--------------+**

**2 rows in set (0.00 sec)**

**select student\_name from students where student\_id in (select student\_id from student\_courses where course\_id = 102);**

**+--------------+**

**| student\_name |**

**+--------------+**

**| suraj |**

**| abhi |**

**+--------------+**

**2 rows in set (0.00 sec)**

**Using JOINS**

**SELECT s.student\_name**

**FROM students s**

**JOIN student\_courses sc ON s.student\_id = sc.student\_id**

**WHERE sc.course\_id = 102;**

**Getting Highest and Second Highest Salary from Employee Table**

**CREATE TABLE employee (**

**id INT AUTO\_INCREMENT PRIMARY KEY,**

**name VARCHAR(50),**

**salary DECIMAL(10, 2)**

**);**

**INSERT INTO employee (name, salary)**

**VALUES ('John Doe', 50000.00),**

**('Jane Smith', 60000.00),**

**('Michael Johnson', 75000.00),**

**('Emily Brown', 55000.00);**

Q- 1 Fetch the name, salary of the employee in decreasing way

Answer - select name,salary from employee order by salary desc;

Q - 2 To fetch the name and salary of the employee with the highest salary

Answer - select name,salary from employee order by salary desc limit 1;

Q - 3 To fetch the name and salary of the employee with the second Highest salary

Answer - select name,salary from employee order by salary desc limit 1 offset 1;

Q - 3 To fetch the name and salary of the employee with the Highest salary without using limit and offset

Answer -

1. select max(salary) from employee;

2. select name,salary from employee where salary = (select max(salary) from employee);

Q - 4 To fetch the name and salary of the employee with the Second Highest salary without using limit and offset

Answer -

select name,salary from employee where salary = (select max(salary) from employee where salary < (select max(salary) from employee ));

Third Highest

select name,salary from employee where salary = (select max(salary) from employee where salary < (select max(salary) from employee where salary < (select max(salary) from employee)));