Module-4: Introduction to DBMS

Lab exercise

* 1.INTRODUCTION TO SQL:-

🡪 Lab 1: Create a new database named school\_db and a table called students with the following columns: student\_id, student\_name, age, class, and address.

Query:-

create database school\_db//for database create

use school\_db//use database

Query:-

create table student(student\_id int primary key auto\_increment,student\_name varchar(20), age int,

class varchar(20),address varchar(20))

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

🡪 Lab 2: Insert five records into the students table and retrieve all records using the SELECT statement.

Query:-

insert into student(student\_name,age,class,address) values(“Yogesh”,20,”graduate”,”Modasa,Gujarat”),

("Vanraj",20,"graduate","Megraj,Gujarat"),

("Ayan",20,"graduate","Ahmedabad,Gujarat"),

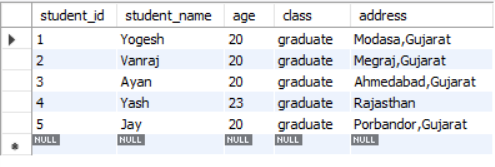
("Yash",23,"graduate","Rajasthan"),

("Jay",20,"graduate","Porbandor,Gujarat")//data inserted successfully

Query:-

Select \* from student//retrive all inserted data

//output:-



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

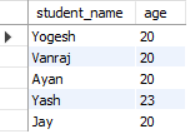
* 2.SQL SYNTAX-

🡪Lab 1: Write SQL queries to retrieve specific columns (student\_name and age) from the students table.

Query:-

select student\_name,age from student//select specific data

//output



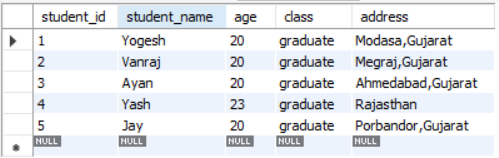
-------------------------------------------------------------------------------------

🡪Lab 2: Write SQL queries to retrieve all students whose age is greater than 10.

Query:-

select \* from student where age>10

//output:-



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

* Que-3. SQL CONSTRAINTS:-

🡪Lab 1: Create a table teachers with the following columns: teacher\_id (Primary Key), teacher\_name (NOT NULL), subject (NOT NULL), and email (UNIQUE).

Query:-

create table teachers(teacher\_id int primary key auto\_increment,teacher\_name varchar(20) not null,subject varchar(20) not null,email varchar(30) unique )

//output



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

🡪Lab 2: Implement a FOREIGN KEY constraint to relate the teacher\_id from the teachers table with the students table

🡪First add teacher\_id field in student table:

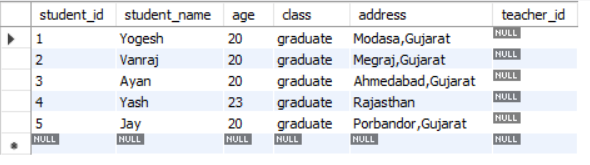
Query:-

alter table student add column teacher\_id int//add new column in student.

🡪foreign key add in column:-

alter table student add constraint foreign key(teacher\_id) references teachers(teacher\_id)

//output



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

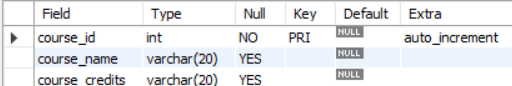
* 4. Main SQL Commands and Sub-commands(DDL):-

🡪Lab 1: Create a table courses with columns: course\_id, course\_name, and course\_credits. Set the course\_id as the primary key.

Query:-

create table course(course\_id int primary key auto\_increment,course\_name varchar(20),course\_credits varchar(20))

//output



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

🡪Lab 2: Use the CREATE command to create a database university\_db.

Query:-

create database university\_db//database created successfully.

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

* 5. ALTER COMMAND:-

🡪Lab 1: Modify the courses table by adding a column course\_duration using the ALTER command.

Query:-

alter table course add course\_duration varchar(20) //colum added

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

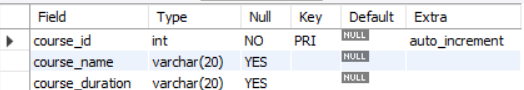
🡪Lab 2: Drop the course\_credits column from the courses table.

Query:-

alter table course drop column course\_credits//column dropped

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

//output:



* 6. DROP COMMAND:-

🡪Lab 1: Drop the teachers table from the school\_db database.

Query:-

drop table teachers;//manually

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

🡪Lab 2: Drop the students table from the school\_db database and verify that the table has been removed.

Query:-

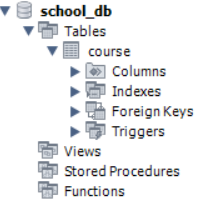
drop table student;//manually

**Note** :- this manually query can’t work because we can use foreign key that connect teacher\_id in both student and teacher table so these query use for it

Query:-

drop tables teacher,student

//verification |Output:-

//only course table there

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

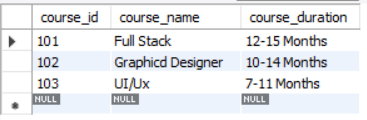
* 7. Data Manipulation Language (DML):-

🡪Lab 1: Insert three records into the courses table using the INSERT command.

Query:-//bydefault value of course\_id start with 101

insert into course(course\_name,course\_duration) values("Full Stack","12-15 Months"),("Graphicd Designer","10-14 Months"),("UI/Ux","7-11 Months")

//output



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

🡪 Lab 2: Update the course duration of a specific course using the UPDATE command.

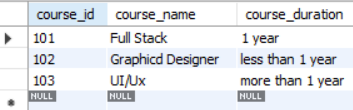
Query:-

update course set course\_duration="1 year" where course\_id=101//for 101

update course set course\_duration=" less than 1 year" where course\_id=102// for 102

update course set course\_duration="more than 1 year" where course\_id=103//for 103

//output



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

🡪Lab 3: Delete a course with a specific course\_id from the courses table using the DELETE command.

Query:-

delete from course where course\_id=102//delete 102 course

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

* 8. Data Query Language (DQL):-

🡪Lab 1: Retrieve all courses from the courses table using the SELECT statement.

Query:- select \* from course;

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

🡪Lab 2: Sort the courses based on course\_duration in descending order using ORDER BY.

Query:-

select \* from course order by course\_duration desc;

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

🡪Lab 3: Limit the results of the SELECT query to show only the top two courses using LIMIT.

Query:- select \* from course order by course\_duration desc limit 1;

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

* 9. Data Control Language (DCL):-

🡪Lab 1: Create two new users user1 and user2 and grant user1 permission to SELECT from the courses table.

Query:-

create user 'user1'@'localhost' identified by 'Mysql';

create user 'user2'@'localhost' identified by 'Mysql';

grant select on course to user1@localhost

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

🡪Lab 2: Revoke the INSERT permission from user1 and give it to user2

Query:-

revoke insert on course from user1@localhost

grant insert on course to user2@localhost

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

* 10. Transaction Control Language (TCL):-

🡪Lab 1: Insert a few rows into the courses table and use COMMIT to save the changes.

Query:-

insert into course(course\_name,course\_duration) values

('C++ Programming', '1 month'),

('Java Fundamentals', '6 months'),

('Python for Beginners', '7 months'),

('Web Development', '2 months'),

('Database Management', '3 months')

Commit;

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

🡪Lab 2: Insert additional rows, then use ROLLBACK to undo the last insert operation.

Query:-

start transaction;

Insert into course(course\_name,course\_duration) values

('Advance php','3 Months'),

('Advance java','3 Months');

rollback;

🡪Lab 3: Create a SAVEPOINT before updating the courses table, and use it to roll back specific changes.

Query:-

start transaction;

savepoint sp2;

update course set course\_duration='1 Month'

where course\_id=103;

rollback to sp2;

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

* 11. SQL JOIN:-

🡪Lab 1: Create two tables: departments and employees. Perform an INNER JOIN to display employees along with their respective departments.

Query:-//create table

create table department

(dept\_id int primary key auto\_increment,

dept\_name varchar(20));

create table employee

(emp\_id int primary key auto\_increment,

emp\_name varchar(20),

dept\_id int,

foreign key(dept\_id) references department(dept\_id))

Query : //insert data into table

INSERT INTO department VALUES (1, 'HR');

INSERT INTO department VALUES (2, 'IT');

INSERT INTO department VALUES (3, 'Finance');

-- Insert employees

INSERT INTO employee VALUES (1001, 'Yogesh', 2);

INSERT INTO employee VALUES (1002, 'Ram', 1);

INSERT INTO employee VALUES (1003, 'Rohit', 3);

INSERT INTO employee VALUES (1004, 'Sai', 2);

Query:-//Inner Join

Select emp\_id,emp\_name,department.dept\_name

from employee

Inner join department

on employee.dept\_id=department.dept\_id

order by emp\_id;

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

🡪Lab 2: Use a LEFT JOIN to show all departments, even those without employees.

Query :- //first we can add new department

insert into department values(4,'Management');

Query:-//Left Join

Select emp\_id,emp\_name,department.dept\_name

from department left join employee

on employee.dept\_id=department.dept\_id

order by emp\_id;

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

* 12. SQL Group By :-

🡪Lab 1: Group employees by department and count the number of employees in each department using GROUP BY.

Query:-

select count(emp\_id) as Total\_Emp,dept\_name

from employee

right join department

on employee.dept\_id=department.dept\_id

group by dept\_name;

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

🡪Lab 2: Use the AVG aggregate function to find the average salary of employees in each department.

Query:-//alter the new row emp\_sal

Alter table employee add emp\_sal int ;

Query:-update all row

update employee set emp\_sal=100000 where emp\_id=1001;

update employee set emp\_sal=70000 where emp\_id=1002;

update employee set emp\_sal=40000 where emp\_id=1003;

update employee set emp\_sal=80000 where emp\_id=1004;

Query:- Search Average

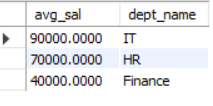
select avg(emp\_sal) as avg\_sal,dept\_name

from employee,department

where employee.dept\_id=department.dept\_id

group by dept\_name;

Output:-



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

* 13. SQL Stored Procedure:-

🡪Lab 1: Write a stored procedure to retrieve all employees from the employees table based on department.

Query:-//create procedure

delimiter //

create procedure getinfo(deptname varchar(20))

begin

select emp\_id,emp\_name,emp\_sal,dept\_name from employee

join department

on employee.dept\_id=department.dept\_id

where dept\_name=deptname;

end//

Query:-//call procedure

call getinfo('IT');

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

🡪Lab 2: Write a stored procedure that accepts course\_id as input and returns the course details

Query:-//create procedure

delimiter //

create procedure getcourseinfo(courseid int)

begin

select \* from course where course\_id=courseid;

end//

Query:- //Call procedure

call getcourseinfo(101);

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

* 14. SQL View:-

🡪Lab 1: Create a view to show all employees along with their department names.

Query:-//create view

create view f\_view

as

select emp\_id,emp\_name,emp\_sal,dept\_name

from employee,department

where employee.dept\_id=department.dept\_id;

Query:-//select all data

Select \* from f\_view;

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

🡪Lab 2: Modify the view to exclude employees whose salaries are below $50,000.

Query:-//drop old view

Drop view if exists f\_view;

Query:-// create modified view

create view f\_view

as

select emp\_id,emp\_name,emp\_sal,dept\_name

from employee,department

where employee.dept\_id=department.dept\_id and emp\_sal>50000;

Query:-

Select \* from f\_view;

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

* 15. SQL Triggers :-

🡪Lab 1: Create a trigger to automatically log changes to the employees table when a new employee is added.

Query:-

delimiter //

create trigger log

after insert on employee

for each row

begin

insert into employee(emp\_name,emp\_sal,dept\_id) values(new.emp\_name,new.emp\_sal,new.dept\_id);

end //

Query:-

insert into employee(emp\_name,emp\_sal,dept\_id) value('Rahul',85000,2);

Note:- Must Be Run on Online Editor .

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

🡪Lab 2: Create a trigger to update the last\_modified timestamp whenever an employee record is updated.

Query:-//first we can add new column for time data added/or modified

alter table employee add column last\_modified timestamp default current\_timestamp ;

delimiter //

create trigger last\_edited

before update on employee

for each row

begin

set new.last\_modified=current\_timestamp;

end//

delimiter ;

Query:-//update something to check trigger

update employee set emp\_name='Rahul' where emp\_id=1004;

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

* 16. Introduction to PL/SQL:-

🡪Lab 1: Write a PL/SQL block to print the total number of employees from the employees table.

Note :- cannot create pl/sql block derectly in mysql

delimiter //

create procedure get\_total\_emp()

begin

declare total int;

select count(emp\_id) into total from employee;

select concat('total employee: ',total) as Result;

end//

delimiter ;

call get\_total\_emp();

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

🡪Lab 2: Create a PL/SQL block that calculates the total sales from an orders table.

Query:

delimiter //

create procedure cal\_total\_sale()

begin

declare total int;

select sum(sales) into total from orders;

select concat('total sales:- ',total) as result;

end//

delimiter ;

call cal\_total\_sale();

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

* 17. PL/SQL Control Structures:-

🡪Lab 1: Write a PL/SQL block using an IF-THEN condition to check the department of an employee.

Query:-

delimiter //

create procedure check\_dept(employ\_id int)

begin

declare emp\_dept int;

select dept\_id into emp\_dept from employee where emp\_id=employ\_id;

if emp\_dept= 1 then

select concat('Employee',employ\_id,'work in HR department') as message;

elseif emp\_dept=2 then

select concat('Employee',employ\_id,'work in IT department') as message;

elseif emp\_dept=3 then

select concat('Employee',employ\_id,'work in Finance department') as message;

elseif emp\_dept=4 then

select concat('Employee',employ\_id,'work in Management department') as message;

end if;

end//

delimiter ;

call check\_dept(1001);

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

🡪Lab 2: Use a FOR LOOP to iterate through employee records and display their names.

Query:-

delimiter //

create procedure loop\_ex()

begin

declare done int default false;

declare empname varchar(20);

declare emp\_cur cursor for

select emp\_name from employee;

declare continue handler for not found set done=true;

open emp\_cur;

emp\_loop:loop

fetch emp\_cur into empname;

if done then

leave emp\_loop;

end if;

select empname as 'employee name ';

end loop;

close emp\_cur;

end//

delimiter ;

call loop\_ex();

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

* 18. SQL Cursors:-

🡪Lab 1: Write a PL/SQL block using an explicit cursor to retrieve and display employee details.

delimiter //

create procedure cur()

begin

declare c\_emp\_id int;

declare c\_emp\_name varchar(20);

declare c\_emp\_sal int;

declare done int default false;

declare emp\_cur cursor for

select emp\_id,emp\_name,emp\_sal from employee;

declare continue handler for not found set done=true;

open emp\_cur;

read\_loop:loop

fetch emp\_cur into c\_emp\_id,c\_emp\_name,c\_emp\_sal;

if done then

leave read\_loop;

end if;

select concat('ID= ',c\_emp\_id,' Name= ',c\_emp\_name,' Salary= ',c\_emp\_sal)as emp\_details;

end loop;

end//

delimiter ;

call cur();

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

🡪Lab 2: Create a cursor to retrieve all courses and display them one by one.

delimiter //

create procedure cur\_2()

begin

declare done int default false;

declare c\_course\_id int;

declare c\_course\_name varchar(30);

declare c\_course\_duration varchar(20);

declare course\_cur cursor for

select course\_id,course\_name,course\_duration from course;

declare continue handler for not found set done=true;

create temporary table if not exists tmp\_course(

cid int,c\_name varchar(20),c\_dura varchar(20));

truncate table tmp\_course;

open course\_cur;

read\_loop:loop

fetch course\_cur into c\_course\_id,c\_course\_name,c\_course\_duration;

if done then

leave read\_loop;

end if;

insert into tmp\_course values(c\_course\_id,c\_course\_name,c\_course\_duration);

end loop;

close course\_cur;

select \* from tmp\_course;

end//

delimiter ;

call cur\_2();

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

* 19. Rollback and Commit Savepoint :-

🡪Lab 1: Perform a transaction where you create a savepoint, insert records, then rollback to the savepoint.

start transaction;

select \* from employee; -- check already exists data

insert into employee(emp\_name,dept\_id,emp\_sal) values('om',4,150000);

insert into employee(emp\_name,dept\_id,emp\_sal) values('Anuj',2,50000);

savepoint sp1;

insert into employee(emp\_name,dept\_id,emp\_sal) values('rudra',1,20000);

insert into employee(emp\_name,dept\_id,emp\_sal) values('prince',3,10000);

rollback to sp1;

commit;

select \* from employee; -- check update data

🡪Lab 2: Commit part of a transaction after using a savepoint and then rollback the remaining changes

start transaction;

insert into employee(emp\_name,dept\_id,emp\_sal) values('rudra',1,20000);

insert into employee(emp\_name,dept\_id,emp\_sal) values('prince',3,10000);

savepoint sp2;

commit;

insert into employee(emp\_name,dept\_id,emp\_sal) values('rahul',2,20000);

insert into employee(emp\_name,dept\_id,emp\_sal) values('pandya',4,10000);

rollback;

select \* from employee;

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