**MODULE: 4**

**Introduction to DBMS**

* **Introduction to SQL**

1. **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.**

* create database school\_db;

use school\_db;

create table students(student\_id int primary key auto\_increment,

student\_name varchar(50),

age int,

class varchar(20),

address varchar(100) );



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

* INSERT INTO students (student\_name, age, class, address) VALUES

('Aarav Mehta', 14, '9A', 'Ahmedabad'),

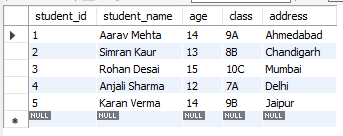
('Simran Kaur', 13, '8B', 'Chandigarh'),

('Rohan Desai', 15, '10C', 'Mumbai'),

('Anjali Sharma', 12, '7A', 'Delhi'),

('Karan Verma', 14, '9B', 'Jaipur');

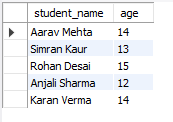
SELECT \* FROM students;



* **SQL Syntax**

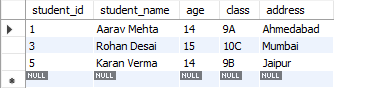
1. **Lab 1: Write SQL queries to retrieve specific columns (student\_name and age) from the students table.**

* select student\_name, age from students;



1. **Lab 2: Write SQL queries to retrieve all students whose age is greater than 13.**

* select \* from students where age > 13;



* **SQL Constraints**

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

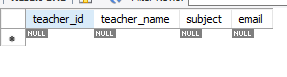
* create table teachers(teacher\_id int primary key auto\_increment,

teacher\_name varchar(50) not null,

subject varchar(50) not null,

email varchar(100) unique

);



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

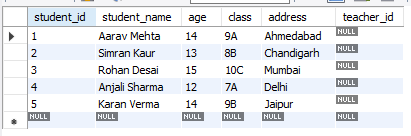
* ALTER TABLE students ADD teacher\_id INT;

ALTER TABLE students

ADD CONSTRAINT fk\_teacher

FOREIGN KEY (teacher\_id)

REFERENCES teachers(teacher\_id);



* **Main SQL Commands and Sub-commands (DDL)**

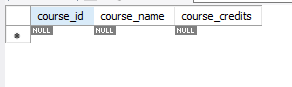
1. **Lab 1: Create a table courses with columns: course\_id, course\_name, and course\_credits. Set the course\_id as the primary key.**

* create table courses(course\_id int primary key,

course\_name varchar(50),

course\_credits int

);



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

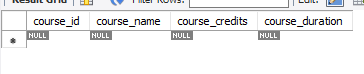
* create database university\_db;

use university\_db;

* **ALTER Command**

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

* alter table courses add course\_duration varchar(50);



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

* alter table courses drop column course\_credits;



* **DROP Command**

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

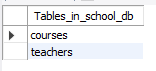
* drop table teachers;



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

* drop table students;

show tables;



* **Data Manipulation Language (DML)**

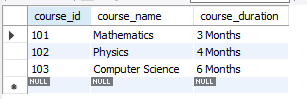
1. **Lab 1: Insert three records into the courses table using the INSERT command.**

* insert into courses (course\_id, course\_name, course\_duration) values

(101, 'Mathematics', '3 Months'),

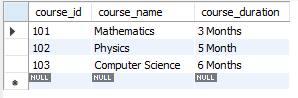
(102, 'Physics', '4 Months'),

(103, 'Computer Science', '6 Months');



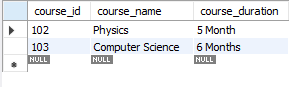
1. **Lab 2: Update the course duration of a specific course using the UPDATE command.**

* update courses set course\_duration = '5 Month' where course\_id = 102;



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

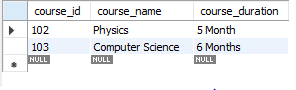
* delete from courses where course\_id = 101;



* **Data Query Language (DQL)**

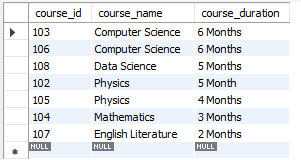
1. **Lab 1: Retrieve all courses from the courses table using the SELECT statement.**

* select \* from courses;



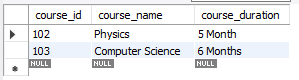
1. **Lab 2: Sort the courses based on course\_duration in descending order using ORDER BY.**

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



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

* select \* from courses limit 2;



* **Data Control Language (DCL)**

1. **Lab 1: Create two new users user1 and user2 and grant user1 permission to SELECT from the courses table.**
2. **Lab 2: Revoke the INSERT permission from user1 and give it to user2.**

* **Transaction Control Language (TCL)**

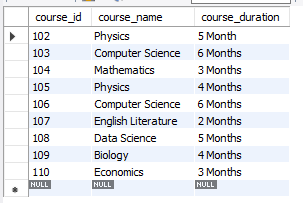
1. **Lab 1: Insert a few rows into the courses table and use COMMIT to save the changes.**

* start transaction;

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

(109, 'Biology', '4 Months'), (110, 'Economics', '3 Months');

commit;



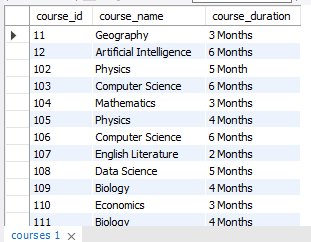
1. **Lab 2: Insert additional rows, then use ROLLBACK to undo the last insert operation.**

* start transaction;

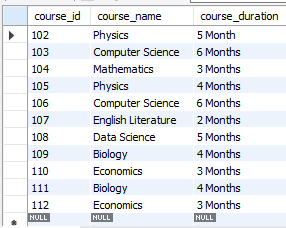
Insert into courses (course\_id, course\_name, course\_duration) values

(11, 'Geography', '3 Months'),

(12, 'Artificial Intelligence', '6 Months');



rollback;

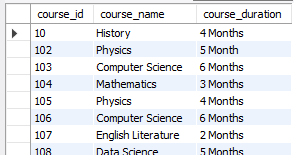


1. **Lab 3: Create a SAVEPOINT before updating the courses table, and use it to roll back specific changes.**

* start transaction;

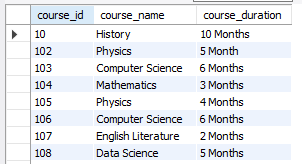
insert into courses (course\_id, course\_name, course\_duration)

values (10, 'History', '4 Months');

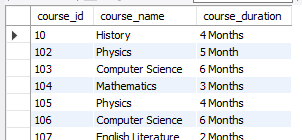


savepoint abc;

update courses set course\_duration = '10 Months' where course\_id = 10;



rollback to abc;



commit;

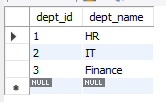
* **SQL Joins:**

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

* create table departments(dept\_id int primary key, dept\_name varchar(50) );

insert into departments (dept\_id, dept\_name) values

(1, 'HR'), (2, 'IT'), (3, 'Finance');

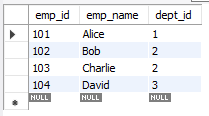


create table employees(emp\_id int primary key, emp\_name varchar(50), dept\_id int,

foreign key (dept\_id) references departments(dept\_id) );

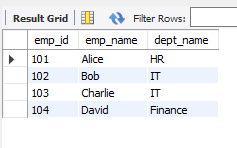
insert into employees (emp\_id, emp\_name, dept\_id) values

(101, 'Alice', 1), (102, 'Bob', 2), (103, 'Charlie', 2), (104, 'David', 3);



select employees.emp\_id, employees.emp\_name, departments.dept\_name

from employees inner join departments on employees.dept\_id = departments.dept\_id;

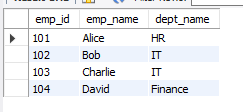


1. **Lab 2: Use a LEFT JOIN to show all departments, even those without employees.**

* select e.emp\_id, e.emp\_name, d.dept\_name from

employees e join departments d

on e.dept\_id = d.dept\_id;



* **SQL Group By**

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

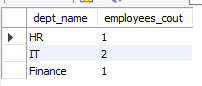
* select departments.dept\_name,

count(employees.emp\_id) AS employees\_cout

from employees

inner join departments on employees.dept\_id = departments.dept\_id

group by departments.dept\_name;



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

* alter table employees add emp\_salary decimal(10, 2);

update employees set emp\_salary = 50000 where emp\_id = 101;

update employees set emp\_salary = 45000 where emp\_id = 102;

update employees set emp\_salary = 60000 where emp\_id = 103;

update employees set emp\_salary = 58000 where emp\_id = 104;

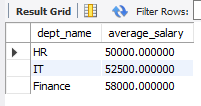
select departments.dept\_name,

avg(employees.emp\_salary) AS average\_salary

from employees

inner join departments on employees.dept\_id = departments.dept\_id

group by departments.dept\_name;



* **SQL Stored Procedure**

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

* CREATE DEFINER=`root`@`localhost` PROCEDURE `GetemployeesBydepartment`(In deptName varchar(100))

BEGIN

select emp\_id, emp\_name, emp\_salary

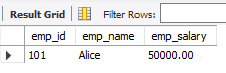
from employees

inner join departments on employees.dept\_id = departments.dept\_id

where departments.dept\_name = deptName;

END

CALL GetEmployeesByDepartment('HR');



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

* CREATE DEFINER=`root`@`localhost` PROCEDURE `GetCoursesDetails`(In input\_course\_id INT)

BEGIN

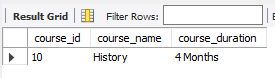
select course\_id, course\_name, course\_duration

from courses

where course\_id = input\_course\_id;

END

call GetCoursesDetails(10);

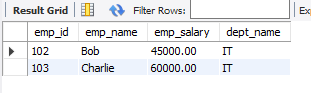


* **SQL View**

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

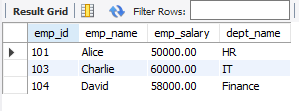
* select \* from school\_db.employeedepartmentview

where dept\_name = ‘IT’;



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

* select \* from empdeptview;



* **SQL Triggers**

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

* delimiter //

create trigger after\_employee\_insert

after insert on employees

for each row

begin

insert into employees\_log (emp\_id, emp\_name)

values (new.emp\_id, new.emp\_name);

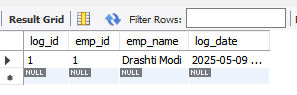
end;

//

delimiter ;

insert into employees (emp\_id, emp\_name)

values (1, 'drashti modi');



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

* alter table employees

add last\_modified timestamp default current\_timestamp on update current\_timestamp;

delimiter //

create trigger before\_employee\_update

before update on employees

for each row

begin

set new.last\_modified = current\_timestamp;

end;

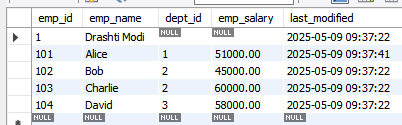
//

delimiter ;

update employees

set emp\_salary = emp\_salary + 1000

where emp\_id = 101;



* **Rollback and Commit Savepoint**

1. **Lab 1: Perform a transaction where you create a savepoint, insert records, then rollback to the savepoint.**
2. **Lab 2: Commit part of a transaction after using a savepoint and then rollback the remaining changes.**