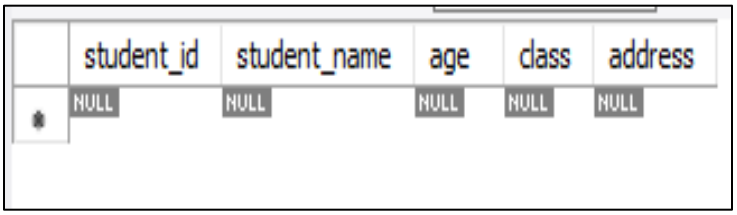
MODULE: 4
Introduction to DBMS

Introduction to SQL

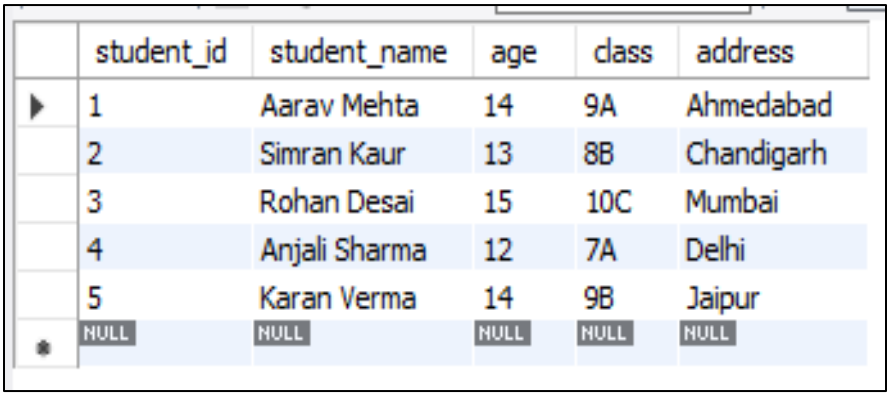
1. **Lab 1: Create a new database named school_db and a table called students with the following columns: student_id, student_name, age, class, and address.**

```
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)       );
```



2. **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;
```



student_id	student_name	age	class	address
1	Aarav Mehta	14	9A	Ahmedabad
2	Simran Kaur	13	8B	Chandigarh
3	Rohan Desai	15	10C	Mumbai
4	Anjali Sharma	12	7A	Delhi
5	Karan Verma	14	9B	Jaipur
NULL	NULL	NULL	NULL	NULL

SQL Syntax

3. Lab 1: Write SQL queries to retrieve specific columns (student_name and age) from the students table.

- select student_name, age from students;

student_name	age
Aarav Mehta	14
Simran Kaur	13
Rohan Desai	15
Anjali Sharma	12
Karan Verma	14

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

- select * from students where age > 13;

student_id	student_name	age	class	address
1	Aarav Mehta	14	9A	Ahmedabad
3	Rohan Desai	15	10C	Mumbai
5	Karan Verma	14	9B	Jaipur
NULL	NULL	NULL	NULL	NULL

SQL Constraints

5. 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
  );

teacher_id	teacher_name	subject	email
NULL	NULL	NULL	NULL

6. **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);

student_id	student_name	age	class	address	teacher_id
1	Aarav Mehta	14	9A	Ahmedabad	NULL
2	Simran Kaur	13	8B	Chandigarh	NULL
3	Rohan Desai	15	10C	Mumbai	NULL
4	Anjali Sharma	12	7A	Delhi	NULL
5	Karan Verma	14	9B	Jaipur	NULL
NULL	NULL	NULL	NULL	NULL	NULL

Main SQL Commands and Sub-commands (DDL)

7. **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
);

course_id	course_name	course_credits
NULL	NULL	NULL

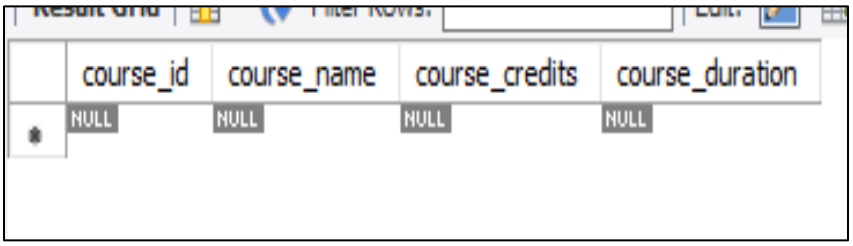
8. **Lab 2: Use the CREATE command to create a database university_db.**

➢ create database university_db;
use university_db;

ALTER Command

9. Lab 1: Modify the courses table by adding a column course_duration using the ALTER command.

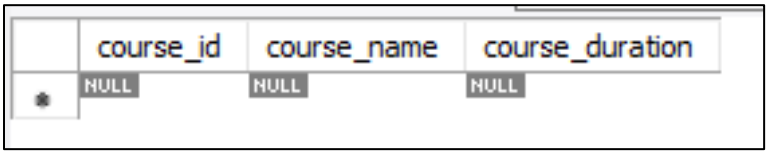
- alter table courses add course_duration varchar(50);



	course_id	course_name	course_credits	course_duration
*	NULL	NULL	NULL	NULL

10. Lab 2: Drop the course_credits column from the courses table.

- alter table courses drop column course_credits;

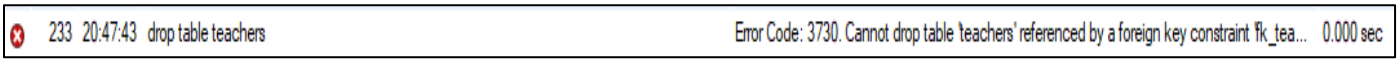


	course_id	course_name	course_duration
*	NULL	NULL	NULL

DROP Command

11. Lab 1: Drop the teachers table from the school_db database.

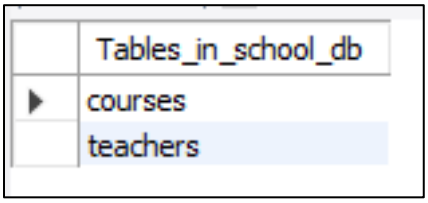
- drop table teachers;



233 20:47:43 drop table teachers — Error Code: 3730. Cannot drop table 'teachers' referenced by a foreign key constraint 'fk_tea... 0.000 sec

12. Lab 2: Drop the students table from the school_db database and verify that the table has been removed.

- drop table students;
  show tables;



	Tables_in_school_db
▶	courses
	teachers

Data Manipulation Language (DML)

13. 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');

course_id	course_name	course_duration
101	Mathematics	3 Months
102	Physics	4 Months
103	Computer Science	6 Months
NULL	NULL	NULL

14. 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;

course_id	course_name	course_duration
101	Mathematics	3 Months
102	Physics	5 Month
103	Computer Science	6 Months
NULL	NULL	NULL

15. 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;

course_id	course_name	course_duration
102	Physics	5 Month
103	Computer Science	6 Months
NULL	NULL	NULL

Data Query Language (DQL)

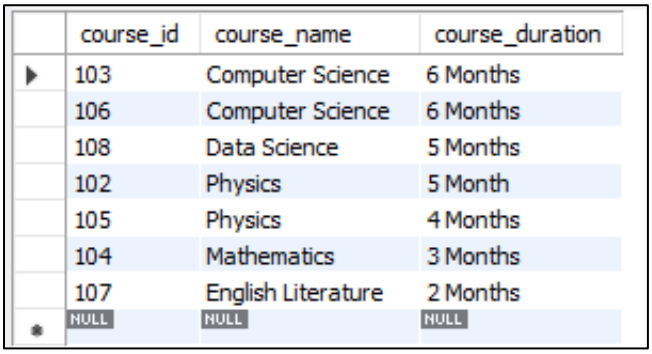
16. Lab 1: Retrieve all courses from the courses table using the SELECT statement.

- select * from courses;

course_id	course_name	course_duration
102	Physics	5 Month
103	Computer Science	6 Months
NULL	NULL	NULL

17. Lab 2: Sort the courses based on course_duration in descending order using ORDER BY.

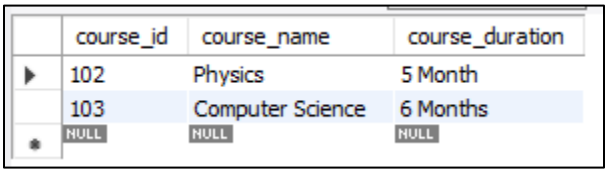
- select * from courses order by course_duration desc;



course_id	course_name	course_duration
103	Computer Science	6 Months
106	Computer Science	6 Months
108	Data Science	5 Months
102	Physics	5 Month
105	Physics	4 Months
104	Mathematics	3 Months
107	English Literature	2 Months
NULL	NULL	NULL

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

- select * from courses limit 2;



course_id	course_name	course_duration
102	Physics	5 Month
103	Computer Science	6 Months
NULL	NULL	NULL

Data Control Language (DCL)

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

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

Transaction Control Language (TCL)

21. 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;

course_id	course_name	course_duration
102	Physics	5 Month
103	Computer Science	6 Months
104	Mathematics	3 Months
105	Physics	4 Months
106	Computer Science	6 Months
107	English Literature	2 Months
108	Data Science	5 Months
109	Biology	4 Months
110	Economics	3 Months
NULL	NULL	NULL

22. 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');

course_id	course_name	course_duration
11	Geography	3 Months
12	Artificial Intelligence	6 Months
102	Physics	5 Month
103	Computer Science	6 Months
104	Mathematics	3 Months
105	Physics	4 Months
106	Computer Science	6 Months
107	English Literature	2 Months

rollback;

course_id	course_name	course_duration
102	Physics	5 Month
103	Computer Science	6 Months
104	Mathematics	3 Months
105	Physics	4 Months
106	Computer Science	6 Months
107	English Literature	2 Months

23. 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');

course_id	course_name	course_duration
10	History	4 Months
102	Physics	5 Month
103	Computer Science	6 Months
104	Mathematics	3 Months
105	Physics	4 Months
106	Computer Science	6 Months
107	English Literature	2 Months
108	Data Science	5 Months

savepoint abc;

update courses set course_duration = '10 Months' where course_id = 10;

course_id	course_name	course_duration
10	History	10 Months
102	Physics	5 Month
103	Computer Science	6 Months
104	Mathematics	3 Months
105	Physics	4 Months
106	Computer Science	6 Months
107	English Literature	2 Months
108	Data Science	5 Months

rollback to abc;

course_id	course_name	course_duration
10	History	4 Months
102	Physics	5 Month
103	Computer Science	6 Months
104	Mathematics	3 Months
105	Physics	4 Months
106	Computer Science	6 Months
107	English Literature	2 Months

commit;

SQL Joins:

24. 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');
```

dept_id	dept_name
1	HR
2	IT
3	Finance
NULL	NULL

```
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);
```

emp_id	emp_name	dept_id
101	Alice	1
102	Bob	2
103	Charlie	2
104	David	3
NULL	NULL	NULL

```
select employees.emp_id, employees.emp_name, departments.dept_name
    from employees  inner join departments on employees.dept_id = departments.dept_id;
```

Result Grid | Filter Rows:

emp_id	emp_name	dept_name
101	Alice	HR
102	Bob	IT
103	Charlie	IT
104	David	Finance

25. 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;
```

emp_id	emp_name	dept_name
101	Alice	HR
102	Bob	IT
103	Charlie	IT
104	David	Finance

SQL Group By

26. 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;
```

dept_name	employees_cout
HR	1
IT	2
Finance	1

27. 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;
```

Result Grid | Filter Rows:

dept_name	average_salary
HR	50000.000000
IT	52500.000000
Finance	58000.000000

SQL Stored Procedure

28. 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');
```

Result Grid | Filter Rows:

emp_id	emp_name	emp_salary
101	Alice	50000.00

29. 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);
```

Result Grid | Filter Rows:

course_id	course_name	course_duration
10	History	4 Months

SQL View

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

- select * from school_db.employeedepartmentview
  where dept_name = 'IT';

Result Grid | Filter Rows:

emp_id	emp_name	emp_salary	dept_name
102	Bob	45000.00	IT
103	Charlie	60000.00	IT

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

- select * from empdeptview;

Result Grid | Filter Rows:

emp_id	emp_name	emp_salary	dept_name
101	Alice	50000.00	HR
103	Charlie	60000.00	IT
104	David	58000.00	Finance

SQL Triggers

32. 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');
```

Result Grid | Filter Rows:

log_id	emp_id	emp_name	log_date
1	1	Drashti Modi	2025-05-09 …
NULL	NULL	NULL	NULL

33. 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;
```

emp_id	emp_name	dept_id	emp_salary	last_modified
1	Drashti Modi	NULL	NULL	2025-05-09 09:37:22
101	Alice	1	51000.00	2025-05-09 09:37:41
102	Bob	2	45000.00	2025-05-09 09:37:22
103	Charlie	2	60000.00	2025-05-09 09:37:22
104	David	3	58000.00	2025-05-09 09:37:22
NULL	NULL	NULL	NULL	NULL

Rollback and Commit Savepoint

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

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