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 student\_data;

CREATE TABLE student\_int (

student\_id INT PRIMARY KEY,

student\_name VARCHAR(20),

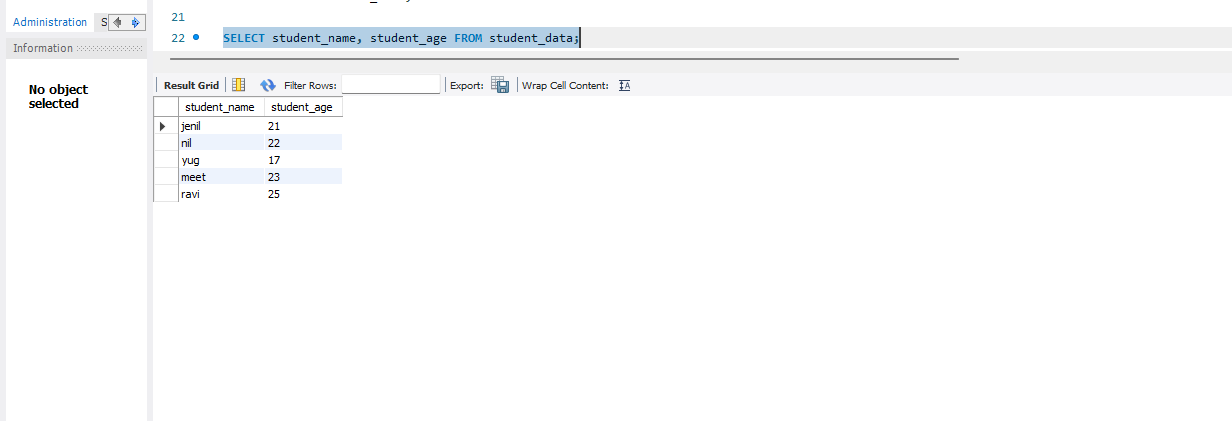
student\_age INT,

student\_address VARCHAR(50),

student\_class INT,

);

select \* from student\_int;



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

INSERT INTO xyz1 (id, student\_name, student\_email\_id, student\_phone\_number, student\_address, student\_age, student\_class)

VALUES (101, 'Jenil', 'jenil90@gmail.com', '9737300756', 'Parvat Patiya, Surat', 20, 12);

INSERT INTO xyz1 (id, student\_name, student\_email\_id, student\_phone\_number, student\_address, student\_age, student\_class)

VALUES (102, 'ravi', 'ravi1090@gmail.com', '9737300756', 'Parvat Patiya, Surat', 25, 12);

INSERT INTO xyz1 (id, student\_name, student\_email\_id, student\_phone\_number, student\_address, student\_age, student\_class)

VALUES (103, 'vraj', 'vraj1190@gmail.com', '9737300756', 'Parvat Patiya, Surat', 23, 14);

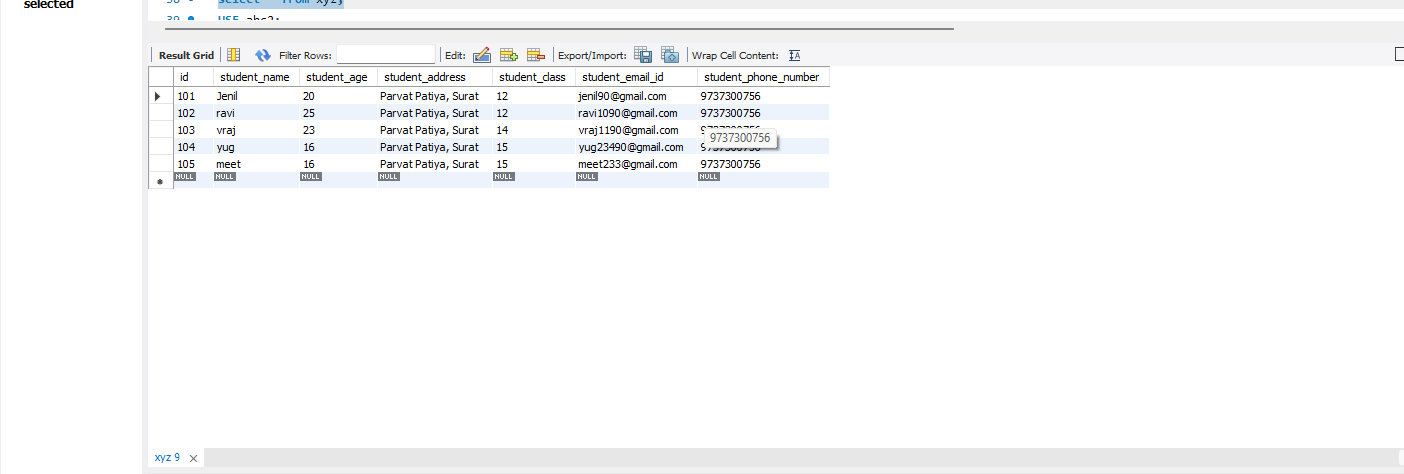
INSERT INTO xyz1 (id, student\_name, student\_email\_id, student\_phone\_number, student\_address, student\_age, student\_class)

VALUES (104, 'yug', 'yug23490@gmail.com', '9737300756', 'Parvat Patiya, Surat', 16, 15);

INSERT INTO xyz1 (id, student\_name, student\_email\_id, student\_phone\_number, student\_address, student\_age, student\_class)

VALUES (105, 'meet', 'meet233@gmail.com', '9737300756', 'Parvat Patiya, Surat', 16, 15);

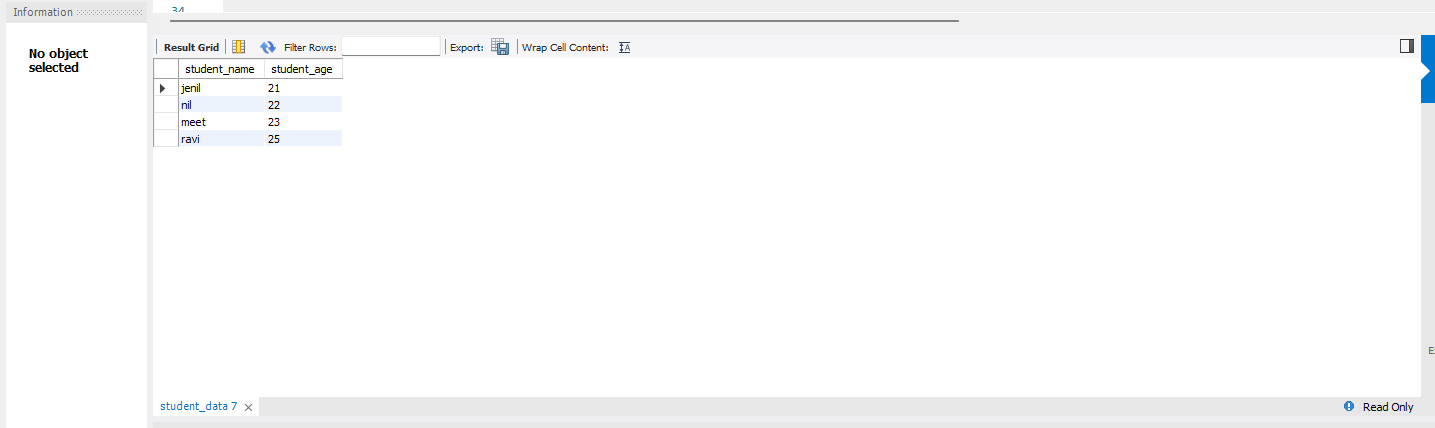
select \* from student\_int;



2. SQL Syntax

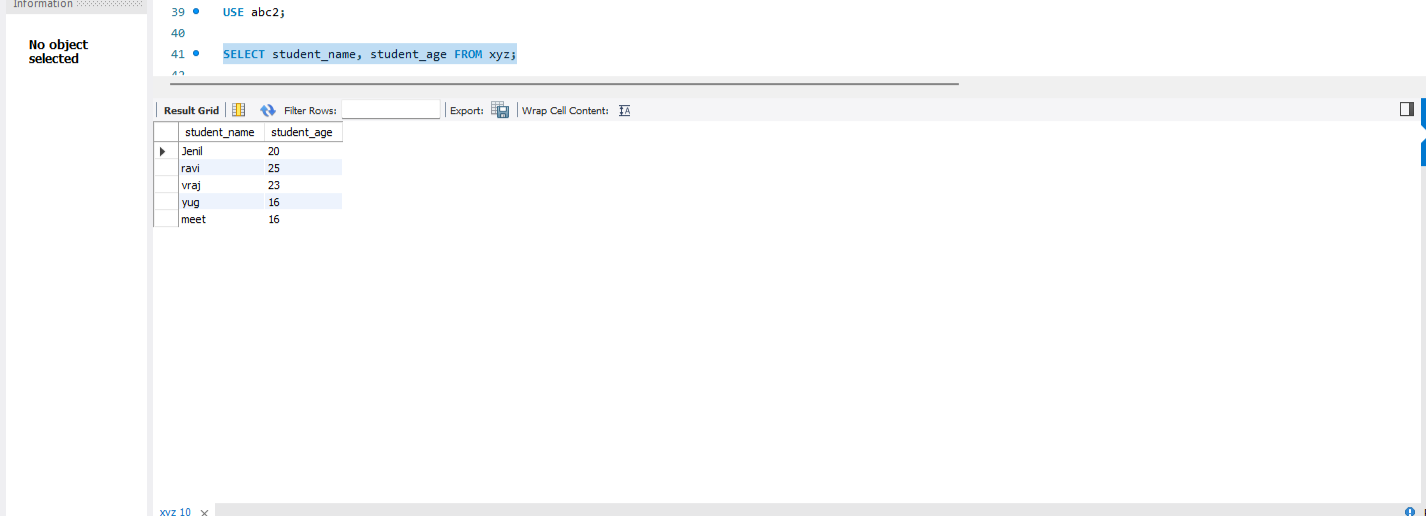
• Lab 1: Write SQL queries to retrieve specific columns (student\_name and age) from the students table

SELECT student\_name, student\_age FROM student\_int;



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

SELECT student\_age FROM student\_int WHERE student\_age > 10;



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).

create database student\_data;

use student\_data;

CREATE TABLE student\_int (

student\_id INT PRIMARY KEY,

student\_name VARCHAR(20),

student\_age INT,

student\_address VARCHAR(50),

student\_class INT,

teacher\_id INT,

FOREIGN KEY (teacher\_id) REFERENCES teacher\_data(teacher\_id)

);

USE student\_data;

create table teacher data(

teacher\_id int primary key,

teacher\_name varchar(10) not null,

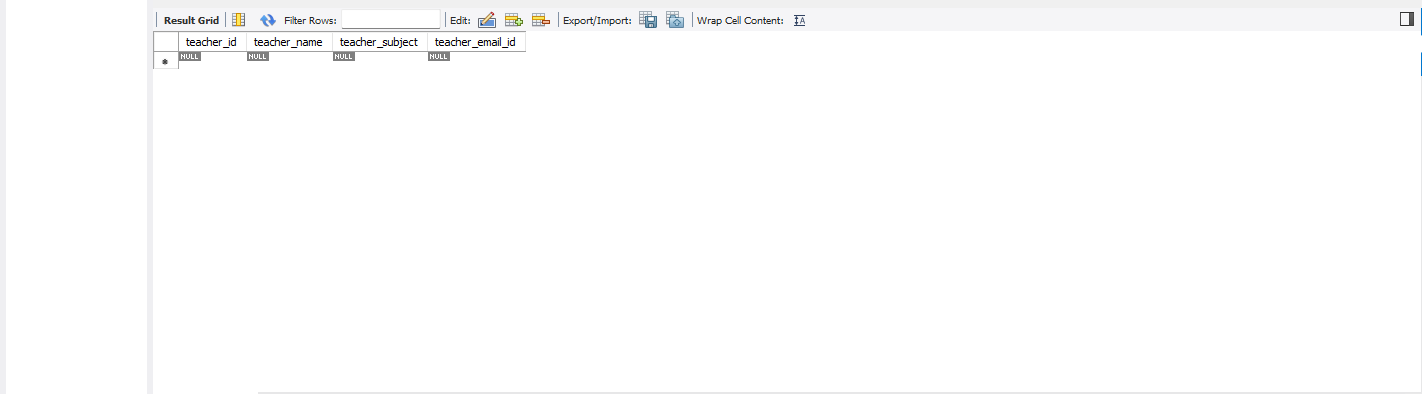
teacher\_subject varchar (20) not null,

teacher\_email\_id varchar(20) unique

);

select \* from student\_int;

select \* from teacher data;



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.

create database student\_data;

create database university dB;

use student\_data;

create table courses(

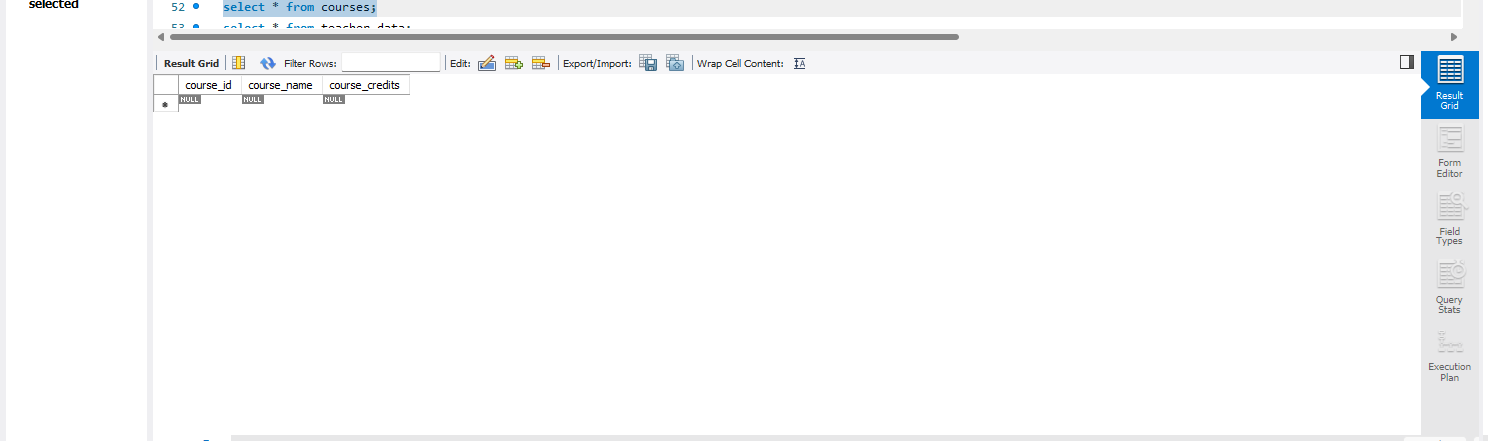
course\_id int primary key,

course\_name varchar (20),

course\_credits varchar(20)

);

select \* from courses;



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

create database university\_db;

use university\_db;

5. ALTER Command

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

create database student\_data;

create database university\_db;

use student\_data;

create table courses(

course\_id int primary key,

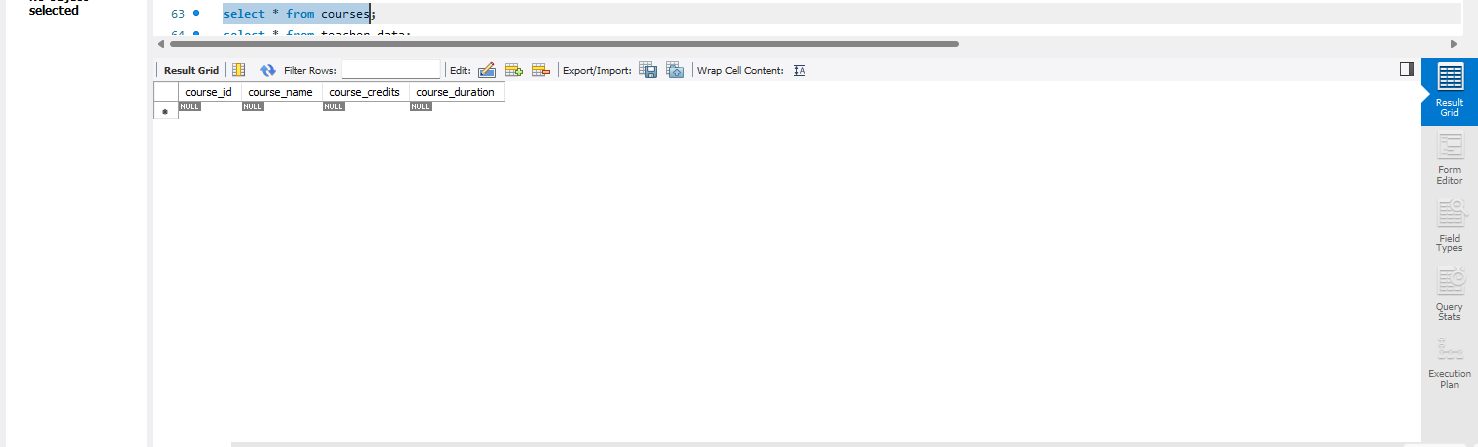
course\_name varchar (20),

course\_credits varchar(20)

);

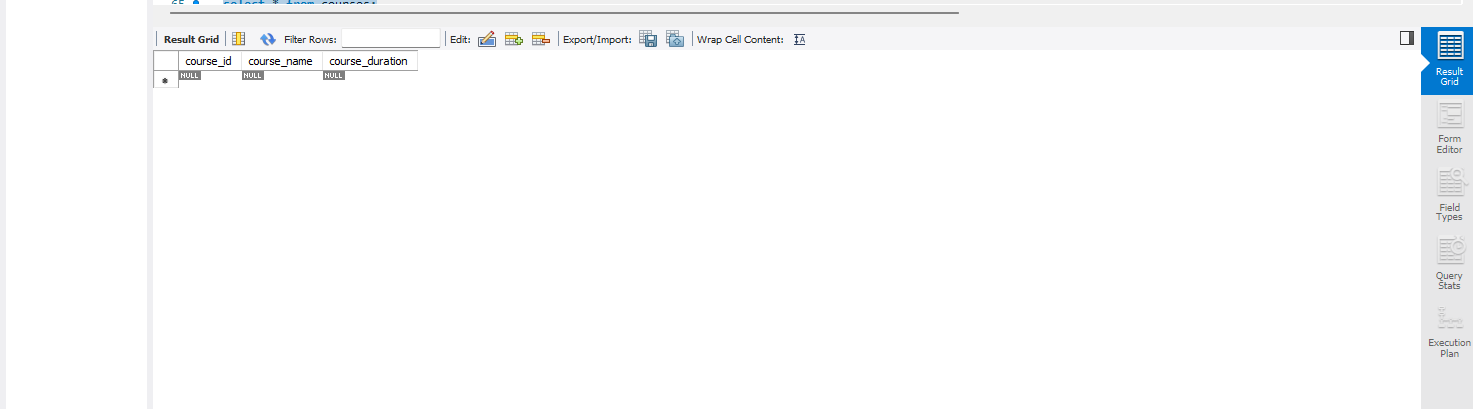
select \* from courses;

ALTER TABLE courses ADD COLUMN course\_duration VARCHAR(50);



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

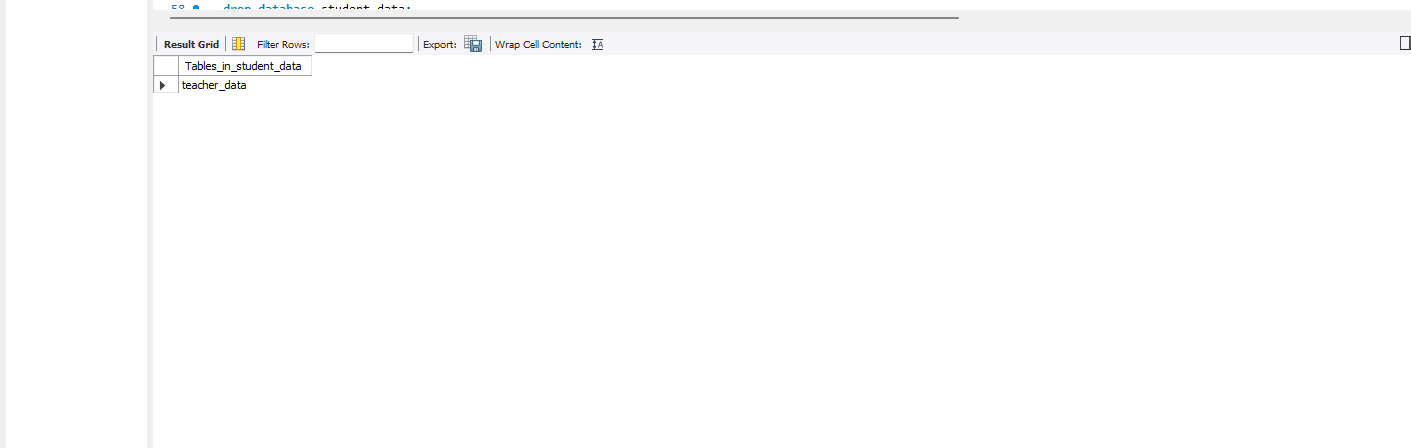
ALTER TABLE courses DROP COLUMN course\_credits;



6. DROP Command

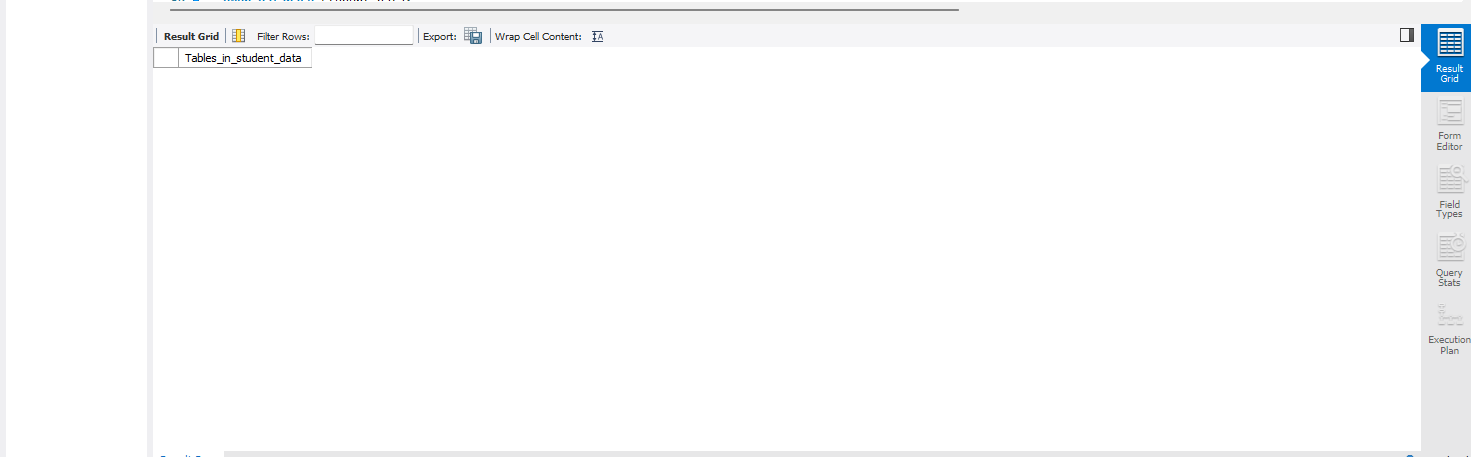
Lab 1: Drop the teachers table from the school\_db database.

drop table teacher\_data;



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

show tables;



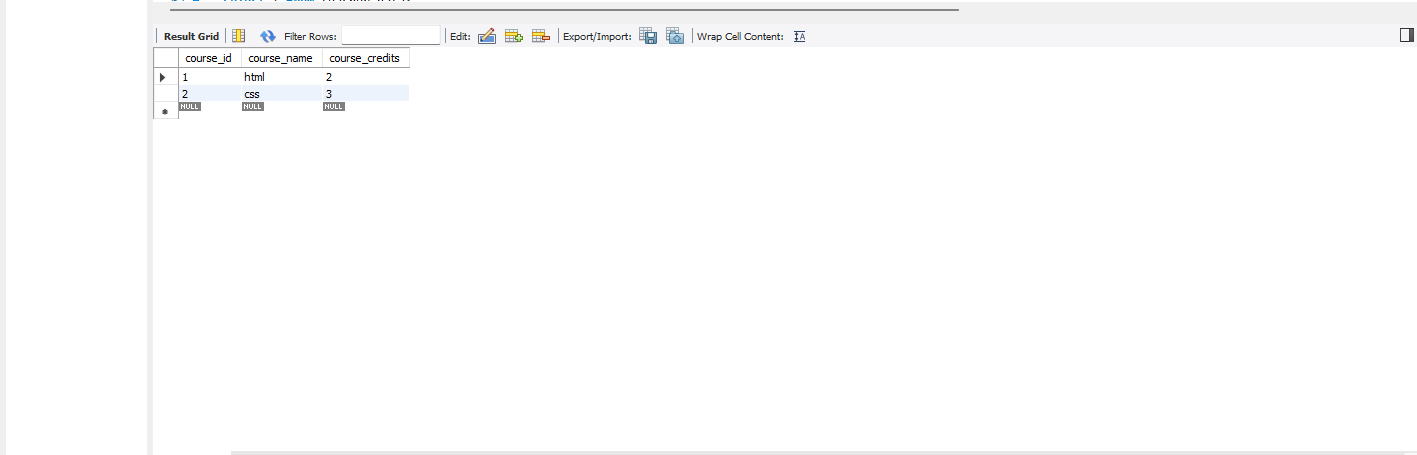
7. Data Manipulation Language (DML)

Lab 1: Insert two records into the courses table using the INSERT command

INSERT INTO courses(course\_id,course\_name,course\_credits) values

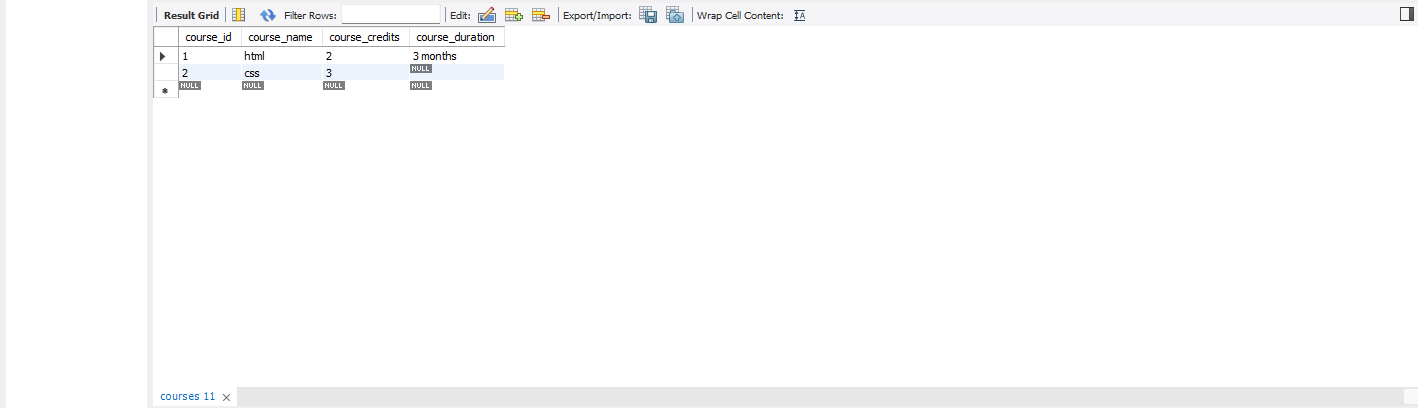
('1','html','2'),

('2','css','3');



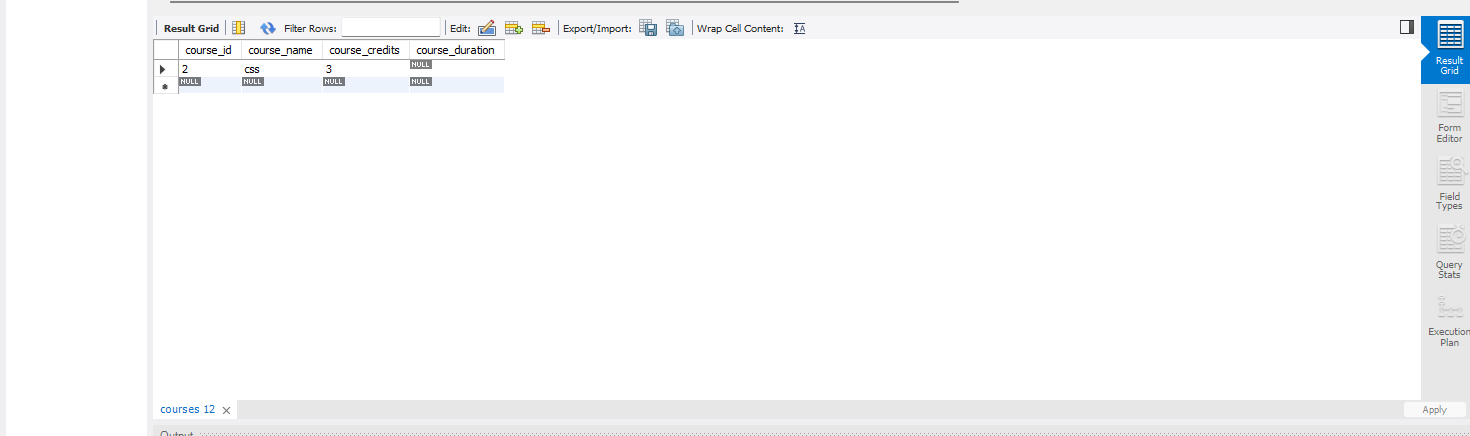
Lab 2: Update the course duration of a specific course using the UPDATE command

UPDATE courses SET course\_duration = '3 months ‘WHERE course\_id = 1;



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

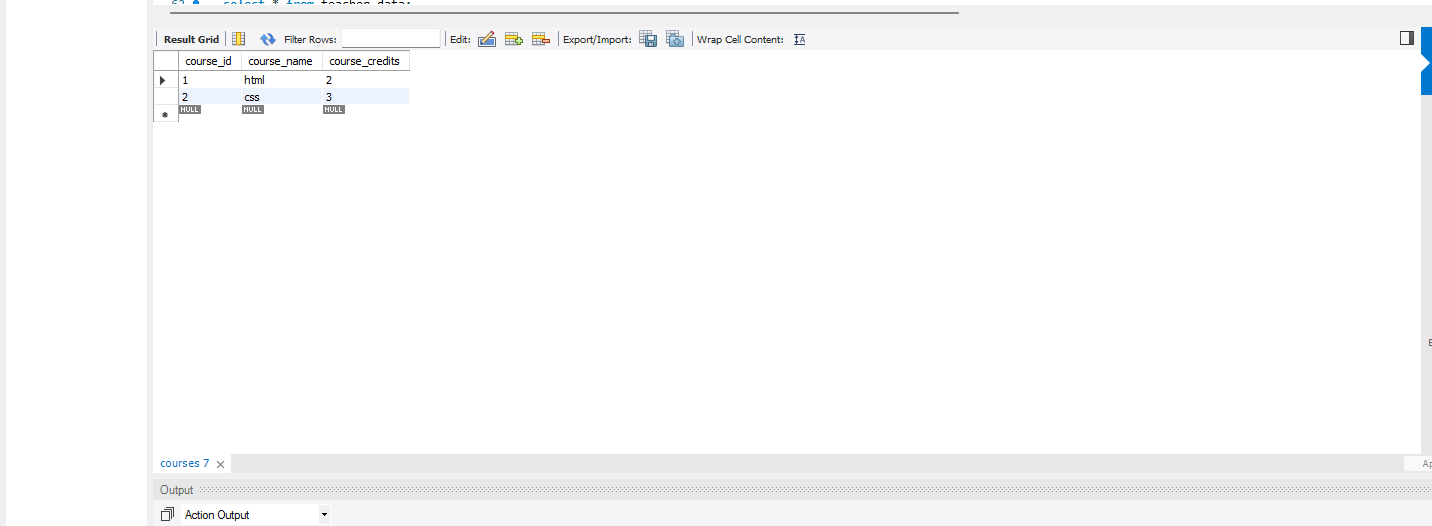
DELETE FROM courses WHERE course\_id = 1;



8. Data Query Language (DQL)

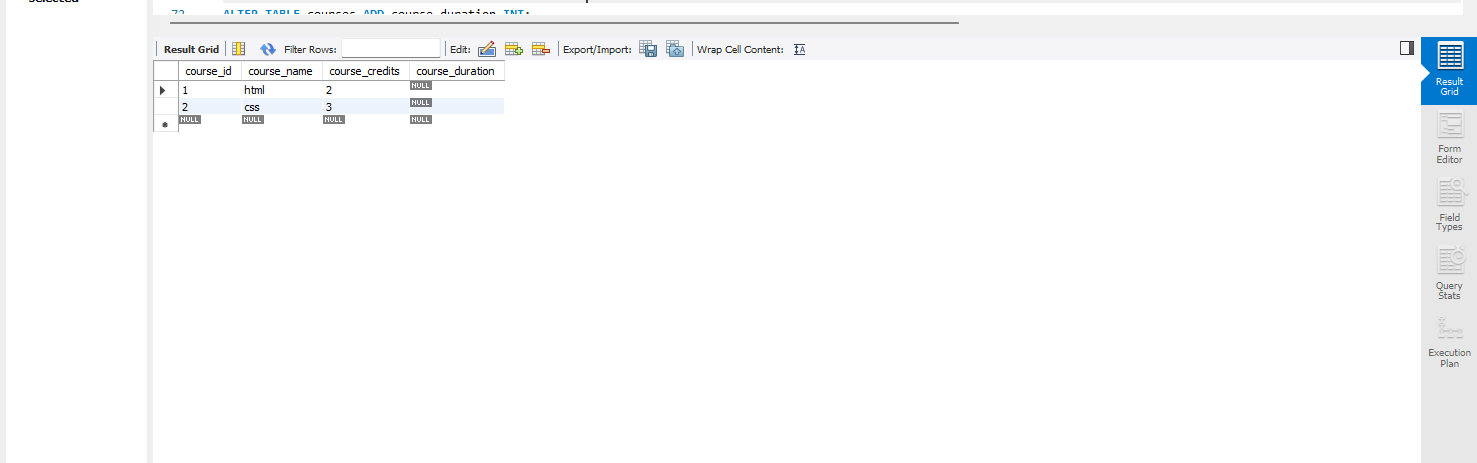
• Lab 1: Retrieve all courses from the courses table using the SELECT statement

Select \* from course



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

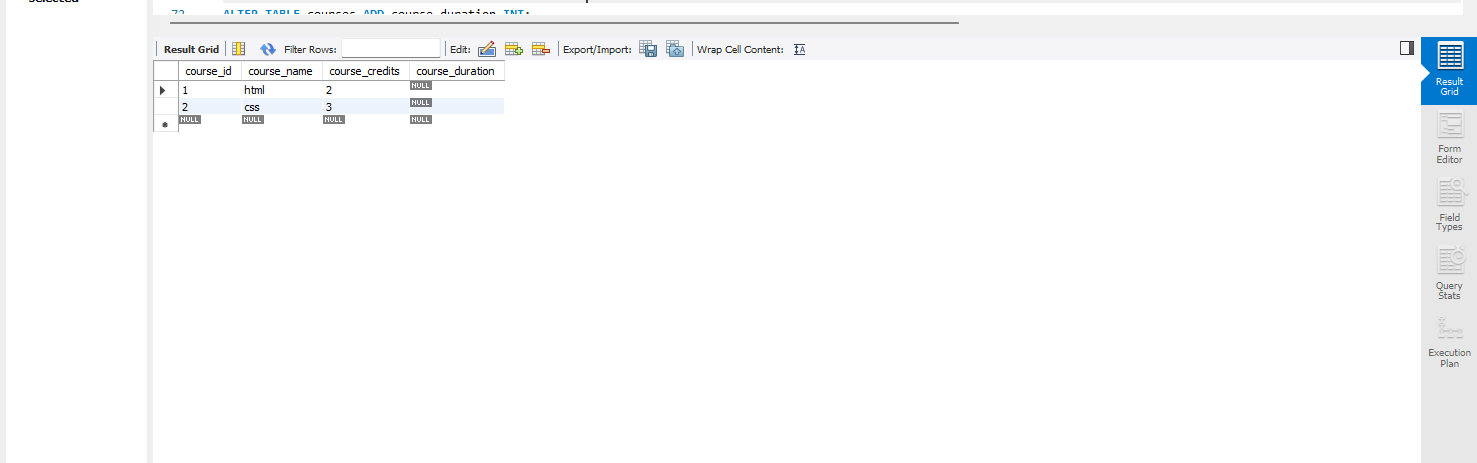
SELECT \* FROM courses ORDER BY course\_duration DESC;



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

SELECT \* FROM courses ORDER BY course\_duration DESC

LIMIT 2;



9. Data Control Language (DCL)

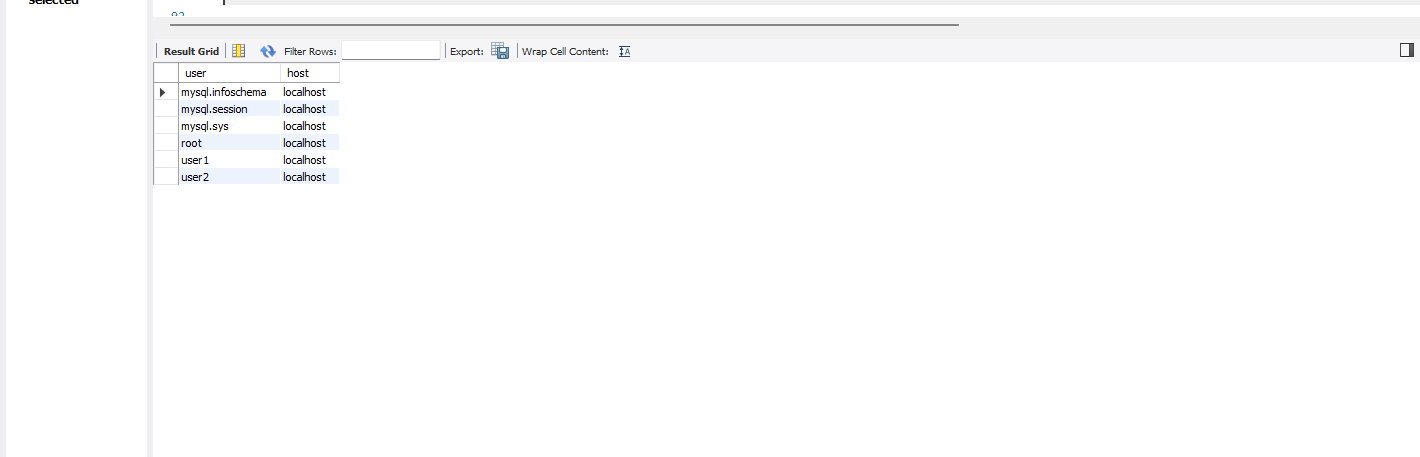
• Lab 1: Create two new users user1 and user2 and grant user1 permission to SELECT from the courses table

CREATE USER 'user2'@'localhost' IDENTIFIED BY 'password2';

DROP USER IF EXISTS 'user2'@'localhost';

CREATE USER 'user2'@'localhost' IDENTIFIED BY 'password2';

SELECT user, host FROM mysql.user;



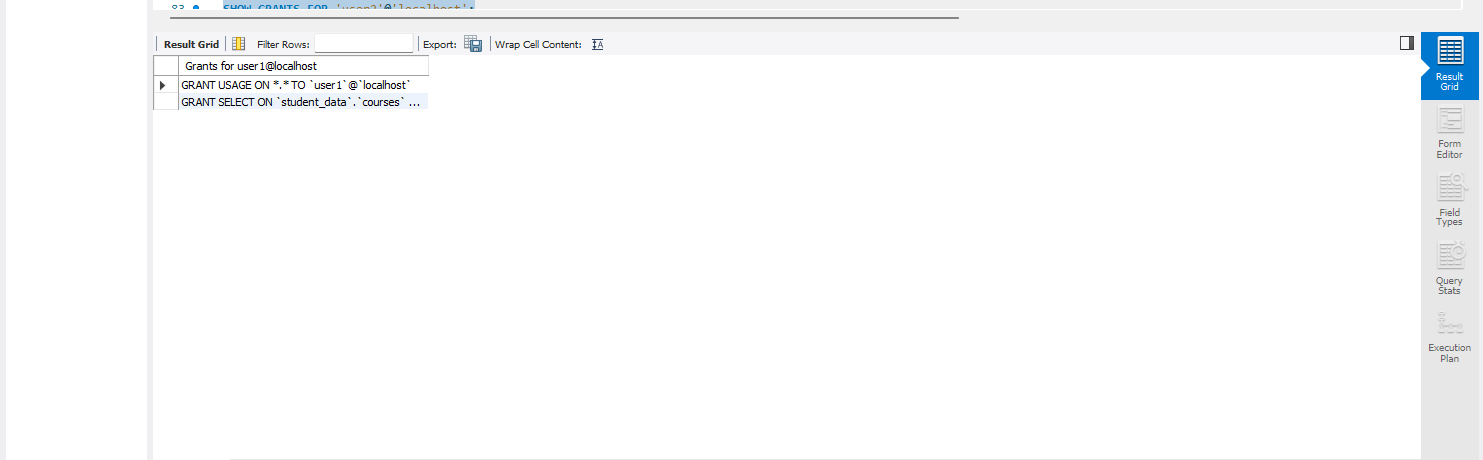
• Lab 2: Revoke the INSERT permission from user1 and give it to user2.

REVOKE INSERT ON student\_data.courses FROM 'user1'@'localhost';

GRANT INSERT ON student\_data.courses TO 'user2'@'localhost';

SHOW GRANTS FOR 'user1'@'localhost';

SHOW GRANTS FOR 'user2'@'localhost';



10. Transaction Control Language (TCL)

• 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, course\_fee)

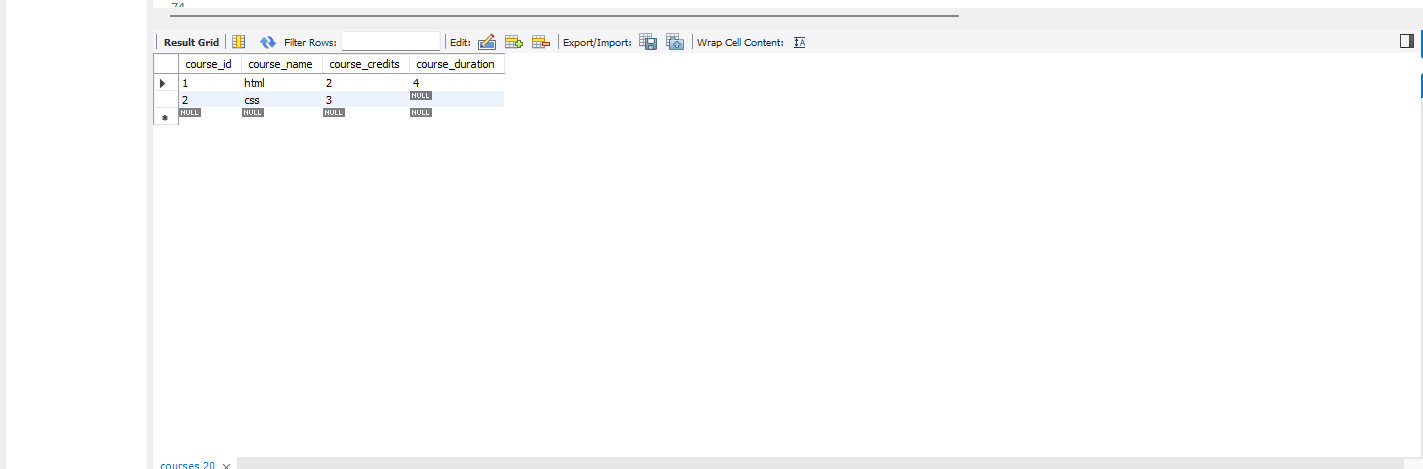
VALUES

(1, 'Python Programming', 6, 5000),

(2, 'Web Development', 8, 6000),

(3, 'Data Science', 10, 8000);

commit;



• 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)

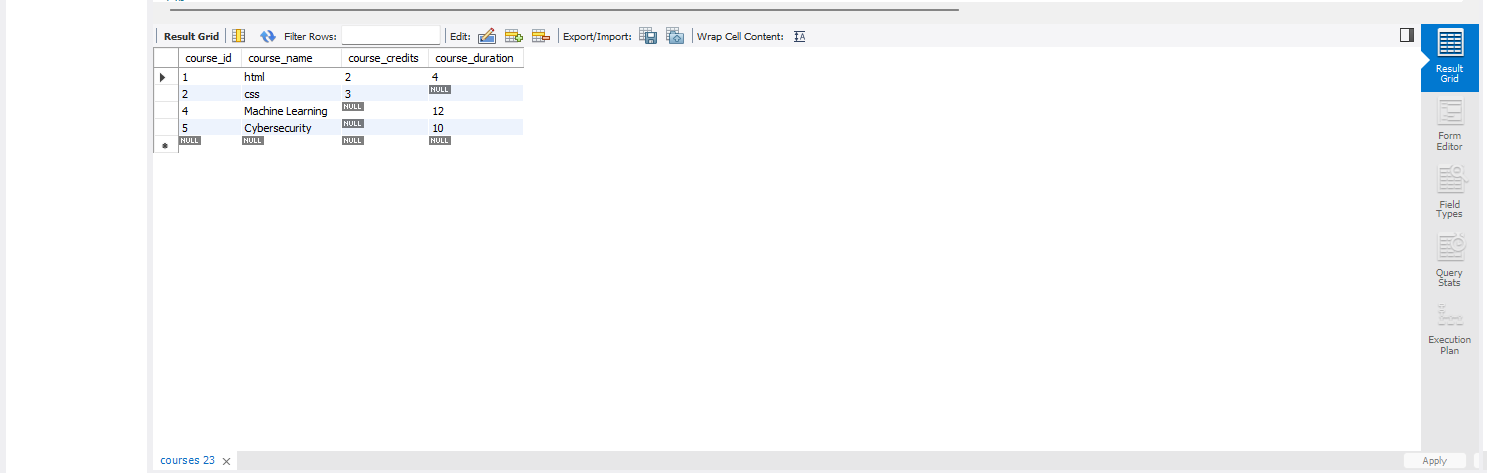
VALUES

(4, 'Machine Learning', 12),

(5, 'Cybersecurity', 10);

ROLLBACK;

SELECT \* FROM courses;



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

START TRANSACTION;

SAVEPOINT before\_update;

UPDATE courses

SET course\_duration = 14

WHERE course\_id = 2;

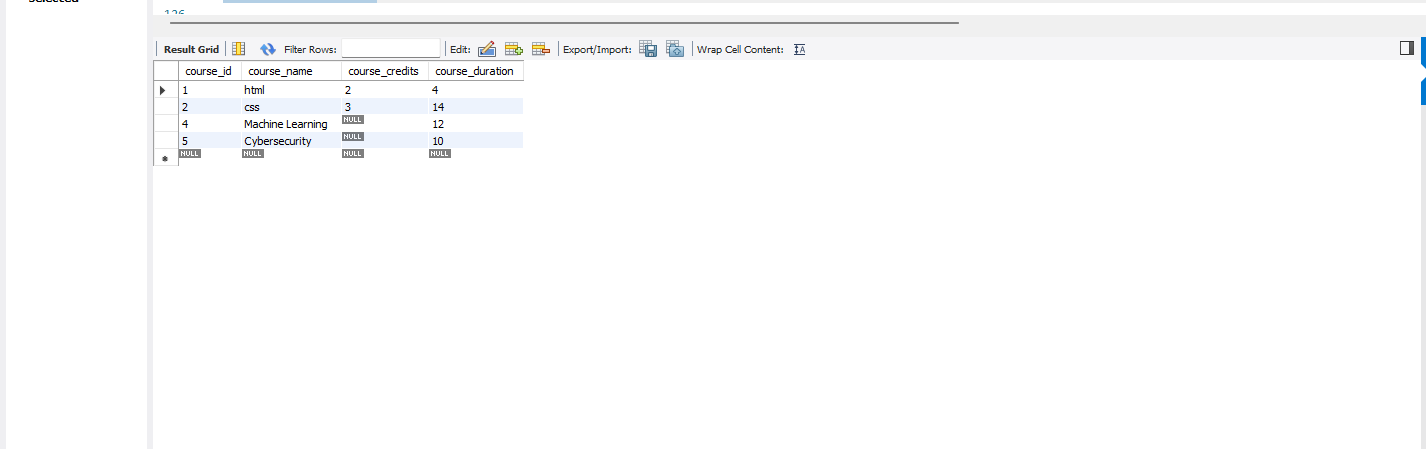
UPDATE courses

SET course\_duration = 16

WHERE course\_id = 3;

ROLLBACK TO SAVEPOINT before\_update;

COMMIT;



11. SQL Joins

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

SELECT

e.Employee\_id,

e.Employee\_First\_name,

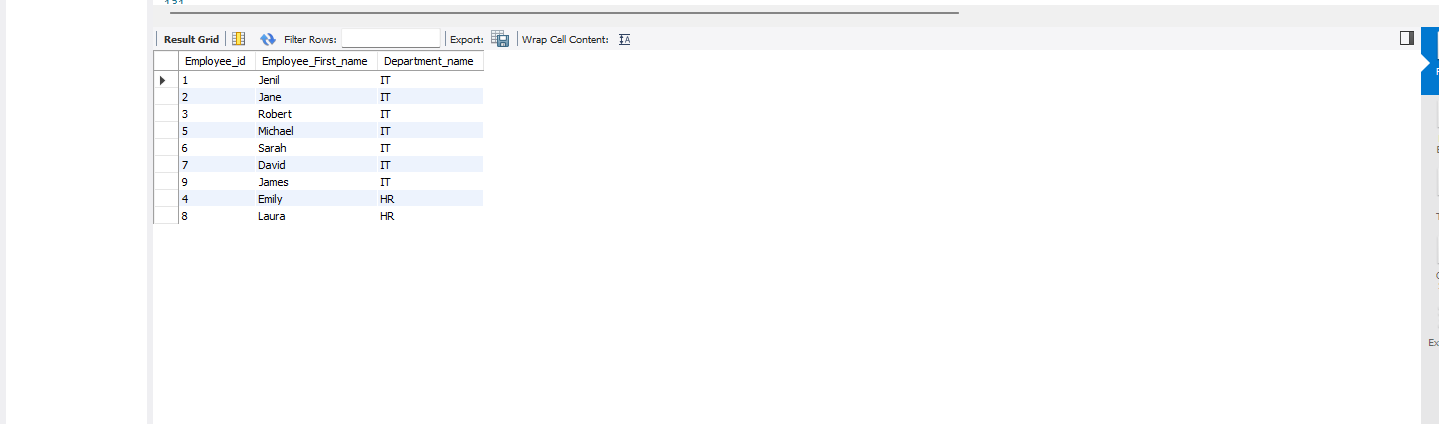
d.Department\_name

FROM

employee e

INNER JOIN

Depatment1 d ON e.Department\_id = d.Department\_id;



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

SELECT

d.Department\_id,

d.Department\_name,

e.Employee\_id,

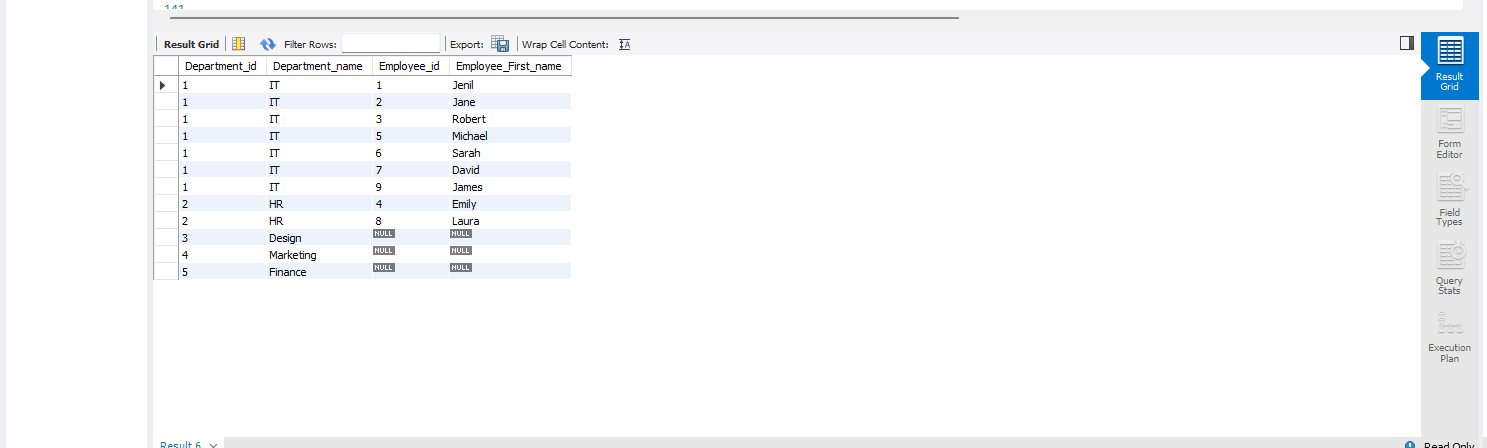
e.Employee\_First\_name

FROM

Depatment1 d

LEFT JOIN

employee e ON d.Department\_id = e.Department\_id;



12. SQL Group By

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

SELECT

d.Department\_id,

d.Department\_name,

COUNT(e.Employee\_id) AS employee\_count

FROM

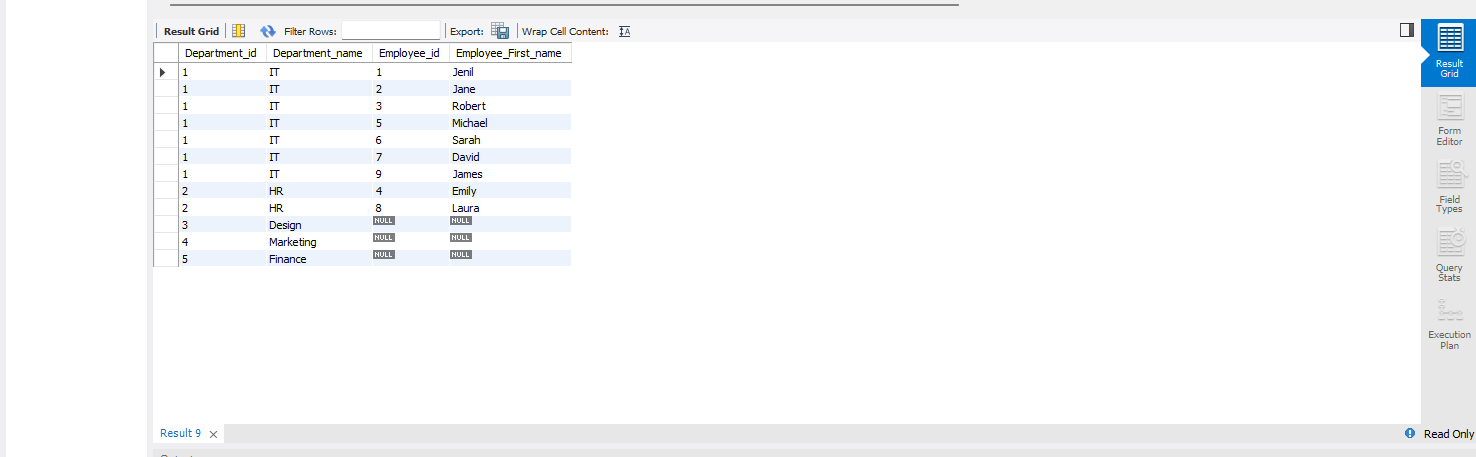
Depatment1 d

LEFT JOIN

employee e ON d.Department\_id = e.Department\_id

GROUP BY

d.Department\_id, d.Department\_name;



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

SELECT

d.Department\_id,

d.Department\_name,

AVG(e.Employee\_salary) AS average\_salary

FROM

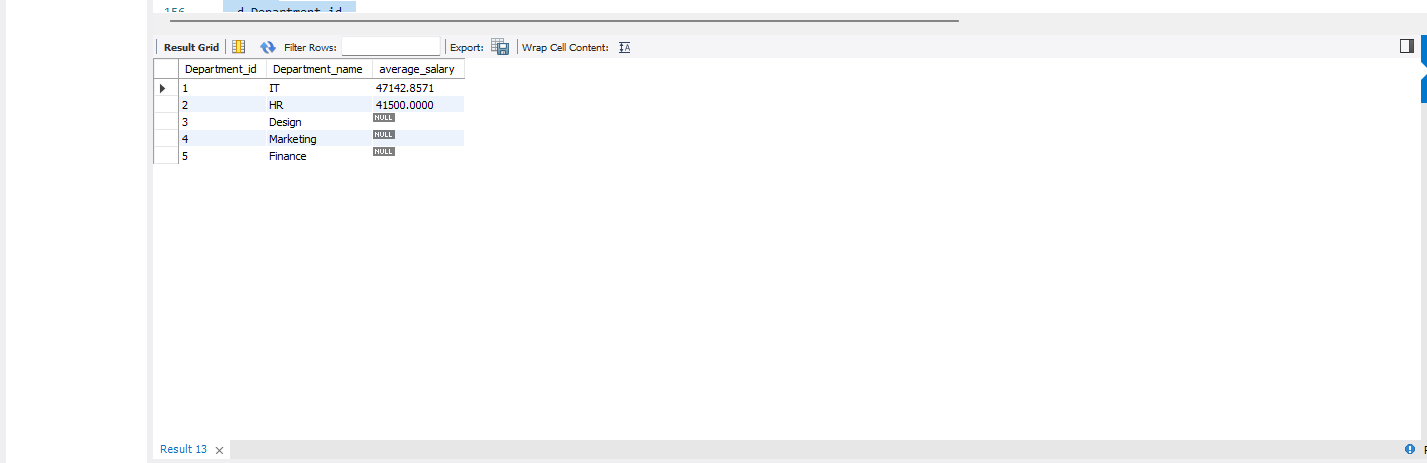
Depatment1 d

LEFT JOIN

employee e ON d.Department\_id = e.Department\_id

GROUP BY

d.Department\_id, d.Department\_name;



13. SQL Stored Procedure

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

DELIMITER $$

CREATE PROCEDURE getemid(IN emp\_id INT)

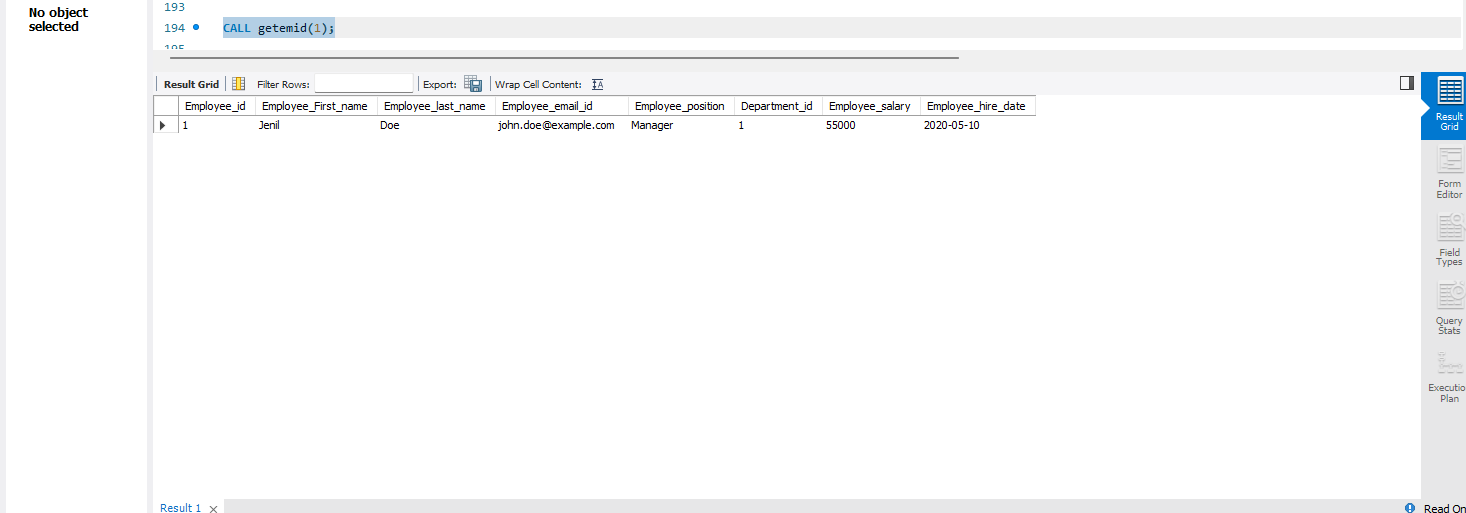
BEGIN

SELECT \* FROM employee WHERE Employee\_id = emp\_id;

END$$

DELIMITER ;

CALL getemid(1);



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

DELIMITER $$

CREATE PROCEDURE getCourseDetails (

IN input\_course\_id INT

)

BEGIN

SELECT \*

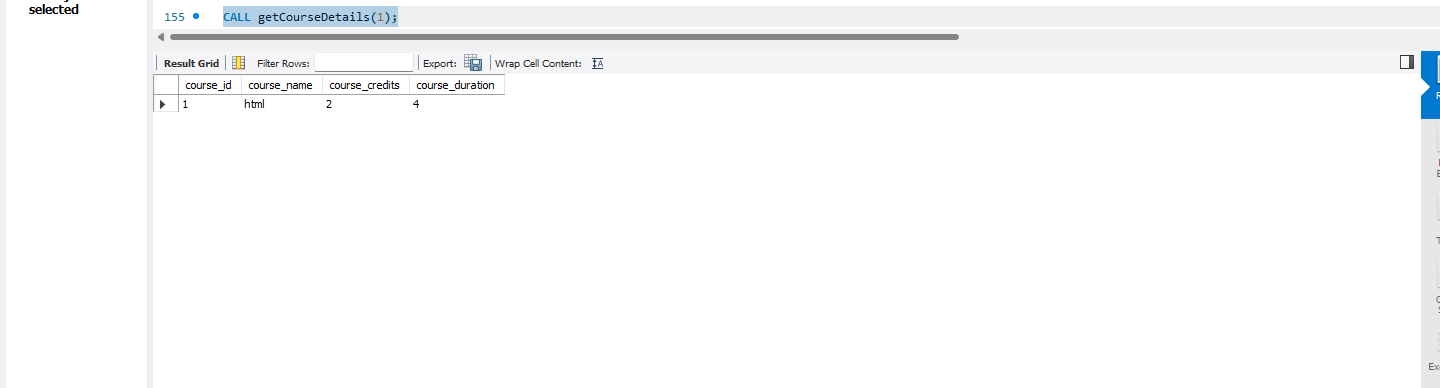
FROM courses

WHERE course\_id = input\_course\_id;

END$$

DELIMITER ;

CALL getCourseDetails(1);



14. SQL View

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

CREATE VIEW employee\_with\_Depatment1 AS

SELECT

e.Employee\_id,

e.Employee\_First\_name,

e.Employee\_last\_name,

e.Employee\_position,

e.Employee\_salary,

e.Department\_id,

d.Department\_name

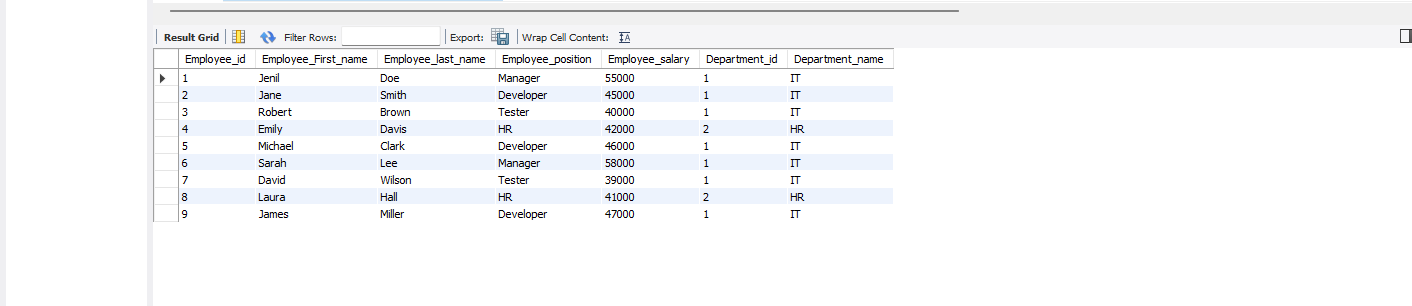
FROM

employee e

JOIN

Depatment1 d ON e.Department\_id = d.Department\_id;

SELECT \* FROM employee\_with\_Depatment1;



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

CREATE VIEW employee\_with\_department\_view AS

SELECT

e.Employee\_id,

e.Employee\_First\_name,

e.Employee\_last\_name,

e.Employee\_position,

e.Employee\_salary,

e.Department\_id,

d.Department\_name

FROM

employee e

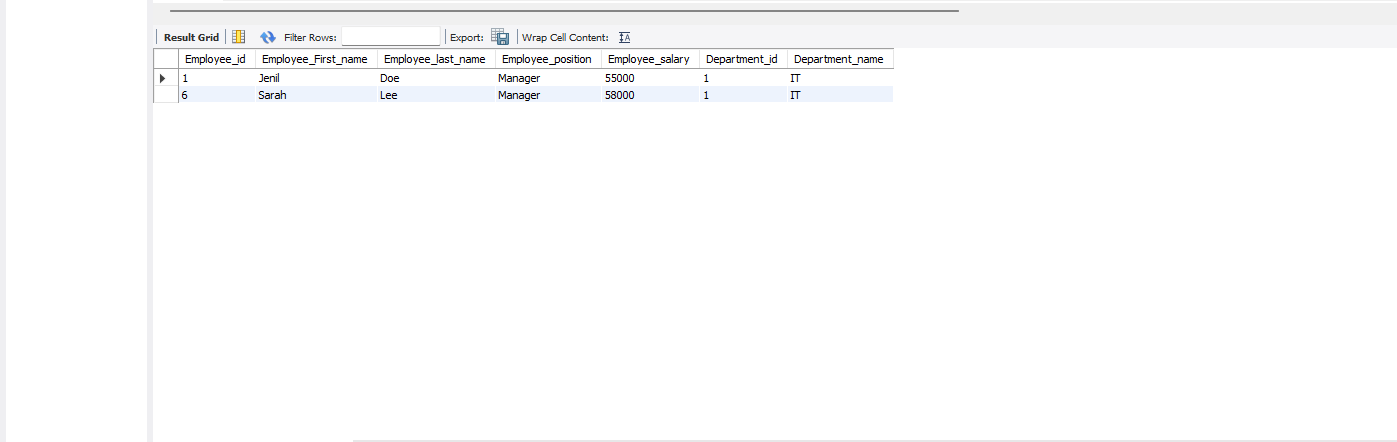
JOIN

Depatment1 d ON e.Department\_id = d.Department\_id

WHERE

e.Employee\_salary >= 50000;

SELECT \* FROM employee\_with\_department\_view;



15. SQL Triggers

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

create table temp(

tid int primary key auto\_increment,

empid int,

time\_of\_event datetime,

event\_name varchar(30)

);

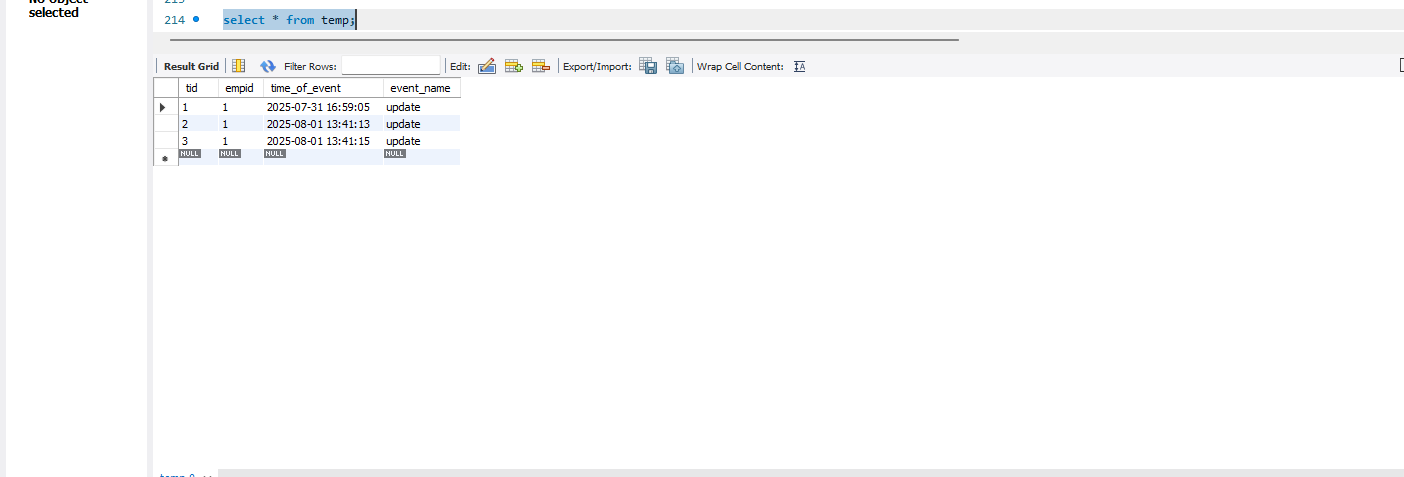
create trigger trimp after update on employee for each row

insert into temp(empid,time\_of\_event,event\_name)values

(old.Employee\_id,now(),"update");

UPDATE employee SET Employee\_First\_name = 'Jenil' WHERE Employee\_id = 1;

select \* from temp;



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

ALTER TABLE employee

ADD COLUMN last\_modified TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

ON UPDATE CURRENT\_TIMESTAMP

DELIMITER $$

CREATE TRIGGER before\_employee\_update

BEFORE UPDATE ON employee

FOR EACH ROW

BEGIN

SET NEW.last\_modified = CURRENT\_TIMESTAMP;

END$$

DELIMITER ;

;

UPDATE employee

SET Employee\_salary = 70000

WHERE Employee\_id = 101;

SELECT Employee\_id, last\_modified FROM employee WHERE Employee\_id = 1;

