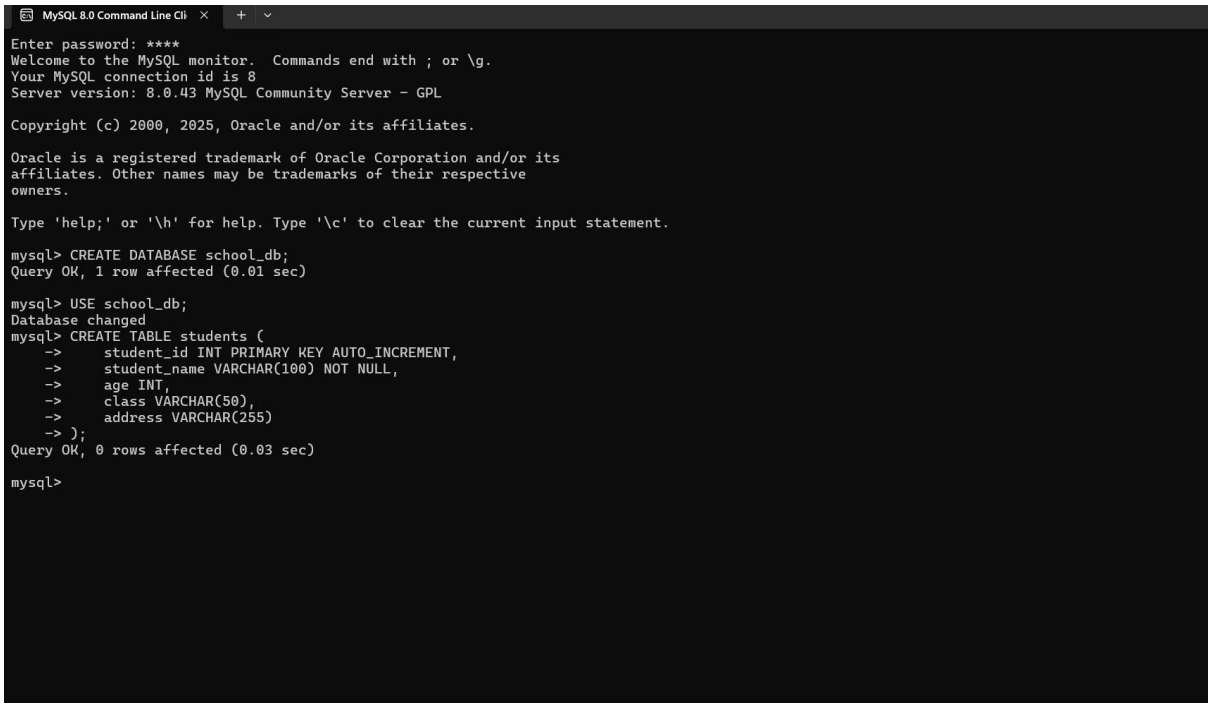
Introduction to SQL

- **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.**



```
MySQL 8.0 Command Line Cli

Enter password: ****
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 8
Server version: 8.0.43 MySQL Community Server - GPL

Copyright (c) 2000, 2025, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> CREATE DATABASE school_db;
Query OK, 1 row affected (0.01 sec)

mysql> USE school_db;
Database changed
mysql> CREATE TABLE students (
    ->     student_id INT PRIMARY KEY AUTO_INCREMENT,
    ->     student_name VARCHAR(100) NOT NULL,
    ->     age INT,
    ->     class VARCHAR(50),
    ->     address VARCHAR(255)
    -> );
Query OK, 0 rows affected (0.03 sec)

mysql>
```

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

```
mysql> INSERT INTO students (student_name, age, class, address)
    -> VALUES
    -> ('Shlok Patel', 15, '10th', 'Ahmedabad'),
    -> ('Ananya Sharma', 14, '9th', 'Surat'),
    -> ('Rohan Mehta', 16, '11th', 'Vadodara'),
    -> ('Priya Desai', 15, '10th', 'Rajkot'),
    -> ('Kabir Joshi', 13, '8th', 'Gandhinagar');
Query OK, 5 rows affected (0.02 sec)
Records: 5  Duplicates: 0  Warnings: 0

mysql> SELECT * FROM students;
+------------+---------------+------+-------+-------------+
| student_id | student_name  | age  | class | address     |
+------------+---------------+------+-------+-------------+
|          1 | Shlok Patel   |   15 | 10th  | Ahmedabad   |
|          2 | Ananya Sharma |   14 | 9th   | Surat       |
|          3 | Rohan Mehta   |   16 | 11th  | Vadodara    |
|          4 | Priya Desai   |   15 | 10th  | Rajkot      |
|          5 | Kabir Joshi   |   13 | 8th   | Gandhinagar |
+------------+---------------+------+-------+-------------+
5 rows in set (0.01 sec)

mysql>
```

2. SQL Syntax

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

```
mysql> SELECT student_name, age
    -> FROM students;
+---------------+------+
| student_name  | age  |
+---------------+------+
| Shlok Patel   |   15 |
| Ananya Sharma |   14 |
| Rohan Mehta   |   16 |
| Priya Desai   |   15 |
| Kabir Joshi   |   13 |
+---------------+------+
5 rows in set (0.00 sec)
```

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

```
mysql> SELECT *
    -> FROM students
    -> WHERE age > 10;
+------------+--------------+------+-------+-------------+
| student_id | student_name | age  | class | address     |
+------------+--------------+------+-------+-------------+
|          1 | Shlok Patel  |   15 | 10th  | Ahmedabad   |
|          2 | Ananya Sharma|   14 | 9th   | Surat       |
|          3 | Rohan Mehta  |   16 | 11th  | Vadodara    |
|          4 | Priya Desai  |   15 | 10th  | Rajkot      |
|          5 | Kabir Joshi  |   13 | 8th   | Gandhinagar |
+------------+--------------+------+-------+-------------+
5 rows in set (0.01 sec)

mysql>
```

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).**

```
mysql> CREATE TABLE teachers (
    ->     teacher_id INT PRIMARY KEY AUTO_INCREMENT,
    ->     teacher_name VARCHAR(100) NOT NULL,
    ->     subject VARCHAR(100) NOT NULL,
    ->     email VARCHAR(100) UNIQUE
    -> );
Query OK, 0 rows affected (0.04 sec)

mysql>
```

- **Lab 2: Implement a FOREIGN KEY constraint to relate the teacher_id from the teachers table with the students table.**

```
mysql> ALTER TABLE students
    -> ADD COLUMN teacher_id INT;
Query OK, 0 rows affected (0.03 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> ALTER TABLE students
    -> ADD CONSTRAINT fk_teacher
    -> FOREIGN KEY (teacher_id) REFERENCES teachers(teacher_id);
Query OK, 5 rows affected (0.07 sec)
Records: 5  Duplicates: 0  Warnings: 0
```

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.**

```
mysql> CREATE TABLE courses (
    ->     course_id INT PRIMARY KEY AUTO_INCREMENT,
    ->     course_name VARCHAR(100) NOT NULL,
    ->     course_credits INT NOT NULL
    -> );
Query OK, 0 rows affected (0.03 sec)

mysql>
```

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

```
mysql> CREATE DATABASE university_db;
Query OK, 1 row affected (0.01 sec)

mysql> USE university_db;
Database changed
mysql>
```

5. ALTER Command

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

```
mysql> ALTER TABLE courses
    -> ADD COLUMN course_duration VARCHAR(50);
Query OK, 0 rows affected (0.05 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

- **Lab 2: Drop the course_credits column from the courses table.**

```
mysql> ALTER TABLE courses
    -> DROP COLUMN course_credits;
Query OK, 0 rows affected (0.05 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> DESCRIBE courses;
+-----------------+--------------+------+-----+---------+----------------+
| Field           | Type         | Null | Key | Default | Extra          |
+-----------------+--------------+------+-----+---------+----------------+
| course_id       | int          | NO   | PRI | NULL    | auto_increment |
| course_name     | varchar(100) | NO   |     | NULL    |                |
| course_length   | varchar(50)  | YES  |     | NULL    |                |
| course_duration | varchar(50)  | YES  |     | NULL    |                |
+-----------------+--------------+------+-----+---------+----------------+
4 rows in set (0.00 sec)
```

6. DROP Command

- **Lab 1: Drop the teachers table from the school_db database.**

```
mysql> USE school_db;
Database changed
mysql>
mysql> DROP TABLE teachers;
Query OK, 0 rows affected (0.03 sec)
```

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

```
mysql> SHOW TABLES;
+--------------------+
| Tables_in_school_db |
+--------------------+
| courses            |
| students           |
+--------------------+
2 rows in set (0.02 sec)

mysql> USE school_db;
Database changed
mysql> DROP TABLE students;
Query OK, 0 rows affected (0.02 sec)

mysql> SHOW TABLES;
+--------------------+
| Tables_in_school_db |
+--------------------+
| courses            |
+--------------------+
1 row in set (0.00 sec)
```

7. Data Manipulation Language (DML)

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

```
mysql> INSERT INTO courses (course_name, course_duration)
    -> VALUES
    -> ('Python Programming', '3 Months'),
    -> ('Web Development', '6 Months'),
    -> ('Data Science', '4 Months');
Query OK, 3 rows affected (0.01 sec)
Records: 3  Duplicates: 0  Warnings: 0

mysql> SELECT * FROM courses;
+-----------+--------------------+-----------------+
| course_id | course_name        | course_duration |
+-----------+--------------------+-----------------+
|         1 | Python Programming | 3 Months        |
|         2 | Web Development    | 6 Months        |
|         3 | Data Science       | 4 Months        |
+-----------+--------------------+-----------------+
3 rows in set (0.00 sec)
```

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

```
mysql> UPDATE courses
    -> SET course_duration = '5 Months'
    -> WHERE course_id = 1;
Query OK, 1 row affected (0.01 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> SELECT * FROM courses;
+-----------+--------------------+-----------------+
| course_id | course_name        | course_duration |
+-----------+--------------------+-----------------+
|         1 | Python Programming | 5 Months        |
|         2 | Web Development    | 6 Months        |
|         3 | Data Science       | 4 Months        |
+-----------+--------------------+-----------------+
3 rows in set (0.00 sec)
```

- **Lab 3: Delete a course with a specific course_id from the courses table using the DELETE command.**

```
mysql> DELETE FROM courses
    -> WHERE course_id = 2;
Query OK, 1 row affected (0.01 sec)

mysql> SELECT * FROM courses;
+-----------+--------------------+-----------------+
| course_id | course_name        | course_duration |
+-----------+--------------------+-----------------+
|         1 | Python Programming | 5 Months        |
|         3 | Data Science       | 4 Months        |
+-----------+--------------------+-----------------+
2 rows in set (0.00 sec)
```

8. Data Query Language (DQL)

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

```
mysql> SELECT * FROM courses;
+-----------+--------------------+-----------------+
| course_id | course_name        | course_duration |
+-----------+--------------------+-----------------+
|         1 | Python Programming | 5 Months        |
|         3 | Data Science       | 4 Months        |
+-----------+--------------------+-----------------+
2 rows in set (0.00 sec)
```

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

```
mysql> SELECT course_id, course_name, course_duration
    -> FROM courses
    -> ORDER BY course_duration DESC;
+-----------+--------------------+-----------------+
| course_id | course_name        | course_duration |
+-----------+--------------------+-----------------+
|         1 | Python Programming | 5 Months        |
|         3 | Data Science       | 4 Months        |
+-----------+--------------------+-----------------+
2 rows in set (0.00 sec)
```

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

```
mysql> SELECT * FROM courses;
+-----------+---------------------+-----------------+
| course_id | course_name         | course_duration |
+-----------+---------------------+-----------------+
|         1 | Python Programming  | 3 Months        |
|         2 | Web Development     | 6 Months        |
|         3 | Data Science        | 4 Months        |
|         4 | Java Programming    | 5 Months        |
|         5 | Database Management | 2 Months        |
+-----------+---------------------+-----------------+
5 rows in set (0.00 sec)

mysql> SELECT course_id, course_name, course_duration
    -> FROM courses
    -> LIMIT 2;
+-----------+--------------------+-----------------+
| course_id | course_name        | course_duration |
+-----------+--------------------+-----------------+
|         1 | Python Programming | 3 Months        |
|         2 | Web Development    | 6 Months        |
+-----------+--------------------+-----------------+
2 rows in set (0.00 sec)
```

9. Data Control Language (DCL)

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

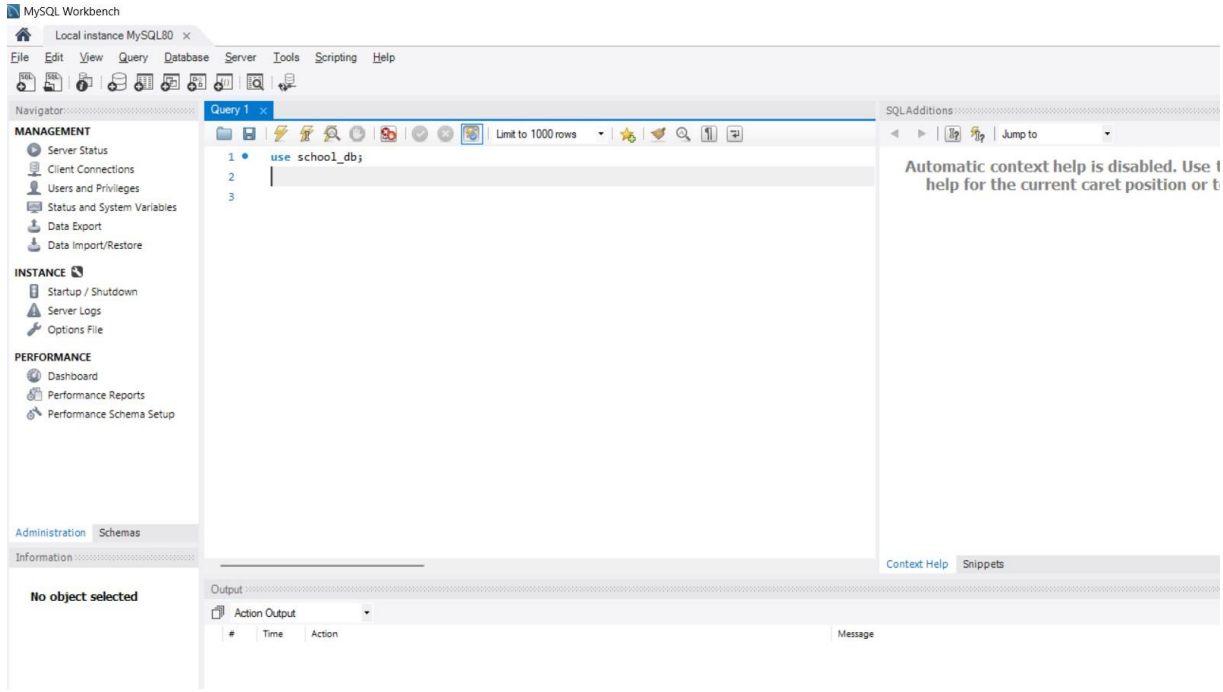
```
mysql> CREATE USER 'user1'@'localhost' IDENTIFIED BY 'password1';
Query OK, 0 rows affected (0.03 sec)

mysql> CREATE USER 'user2'@'localhost' IDENTIFIED BY 'password2';
Query OK, 0 rows affected (0.01 sec)

mysql> GRANT SELECT ON school_db.courses TO 'user1'@'localhost';
Query OK, 0 rows affected (0.02 sec)
```

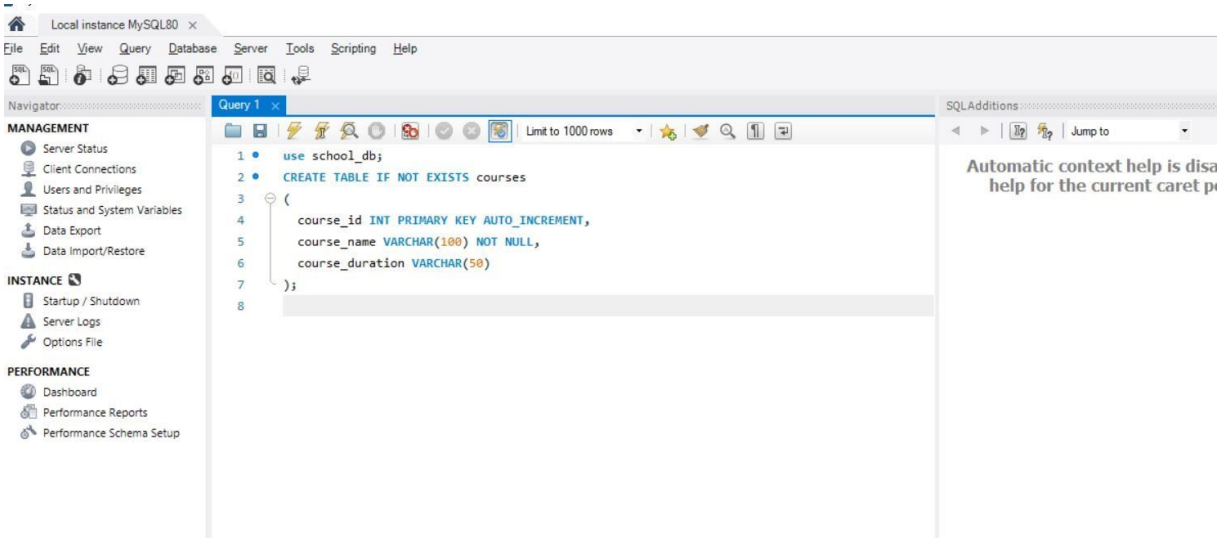
- **Lab 2: Revoke the INSERT permission from user1 and give it to user2.**

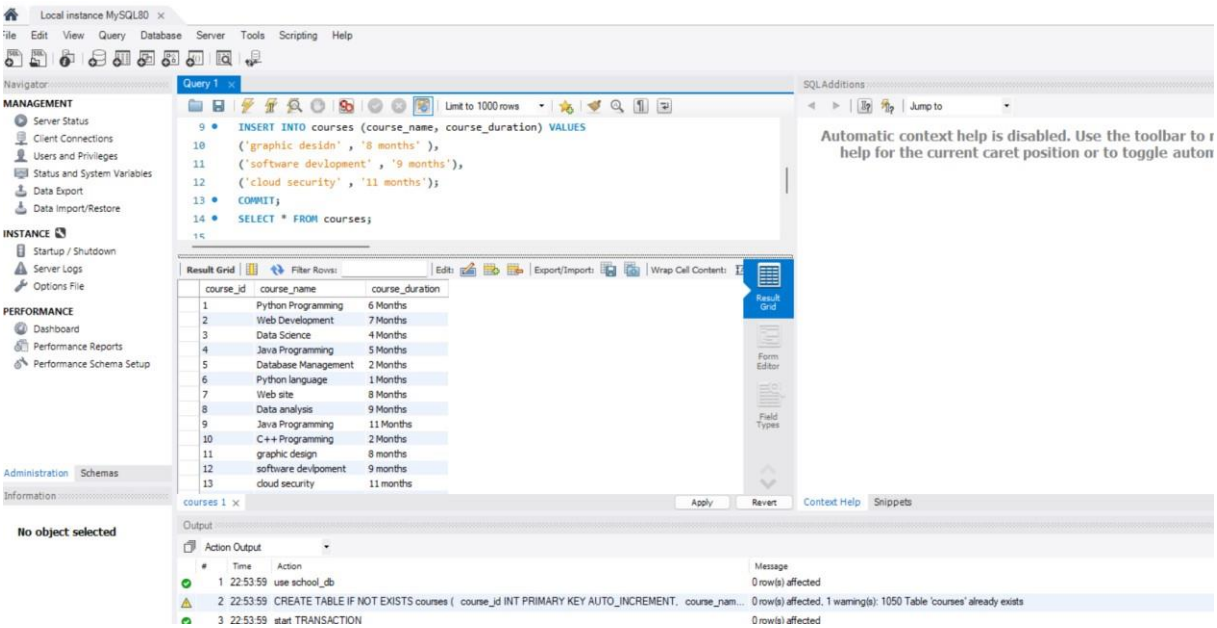
```
mysql> REVOKE INSERT ON school_db.courses FROM 'user1'@'localhost';
Query OK, 0 rows affected (0.01 sec)

mysql> GRANT INSERT ON school_db.courses TO 'user2'@'localhost';
Query OK, 0 rows affected (0.01 sec)
```

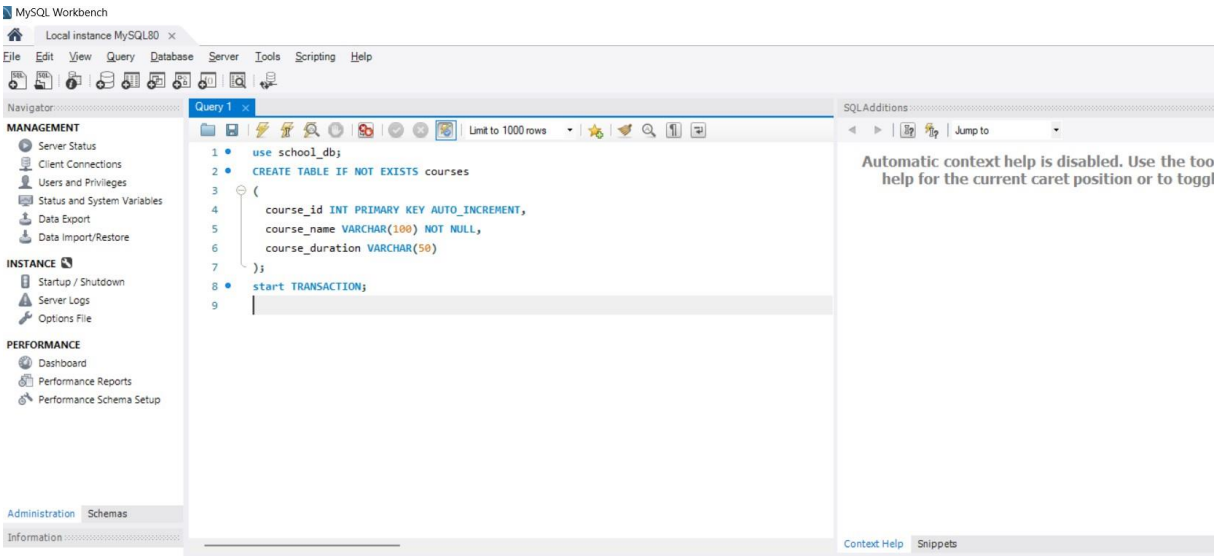
10. Transaction Control Language (TCL)

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









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

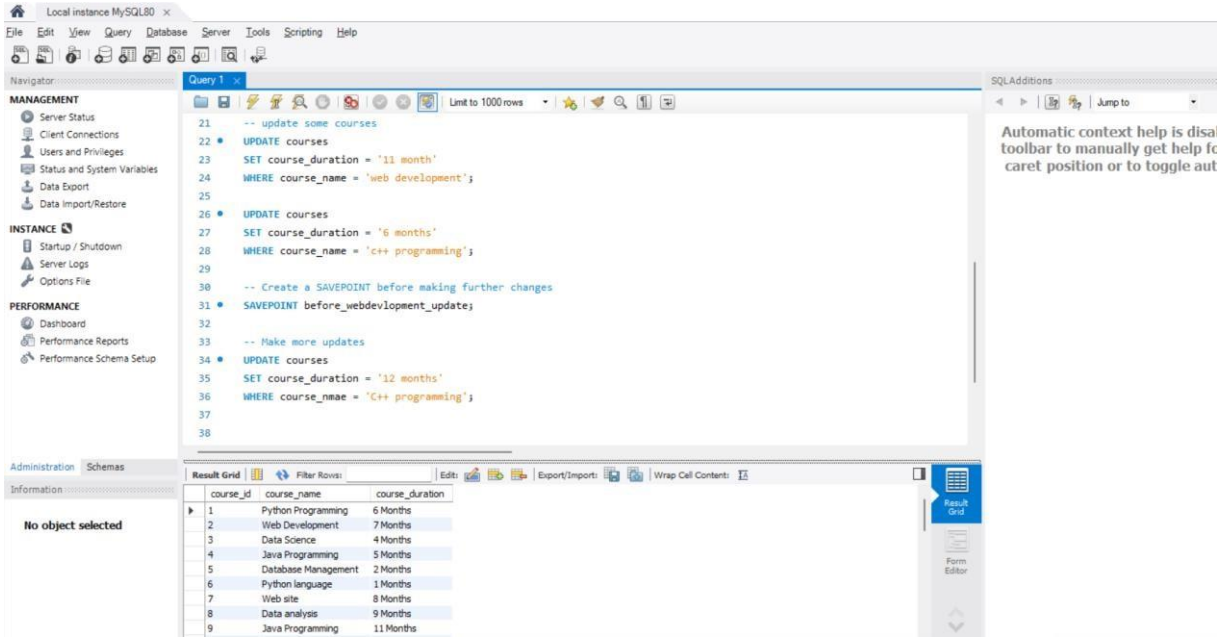
```
15 • START TRANSACTION;
16 • INSERT INTO courses (course_name , course_duration) VALUES
17   ('artificial intelligence' , '12 months'),
18   ('machine learning' , '10 months');
19 • ROLLBACK;
20 • SELECT * FROM courses;
21   -- update some courses
22 • UPDATE courses
```

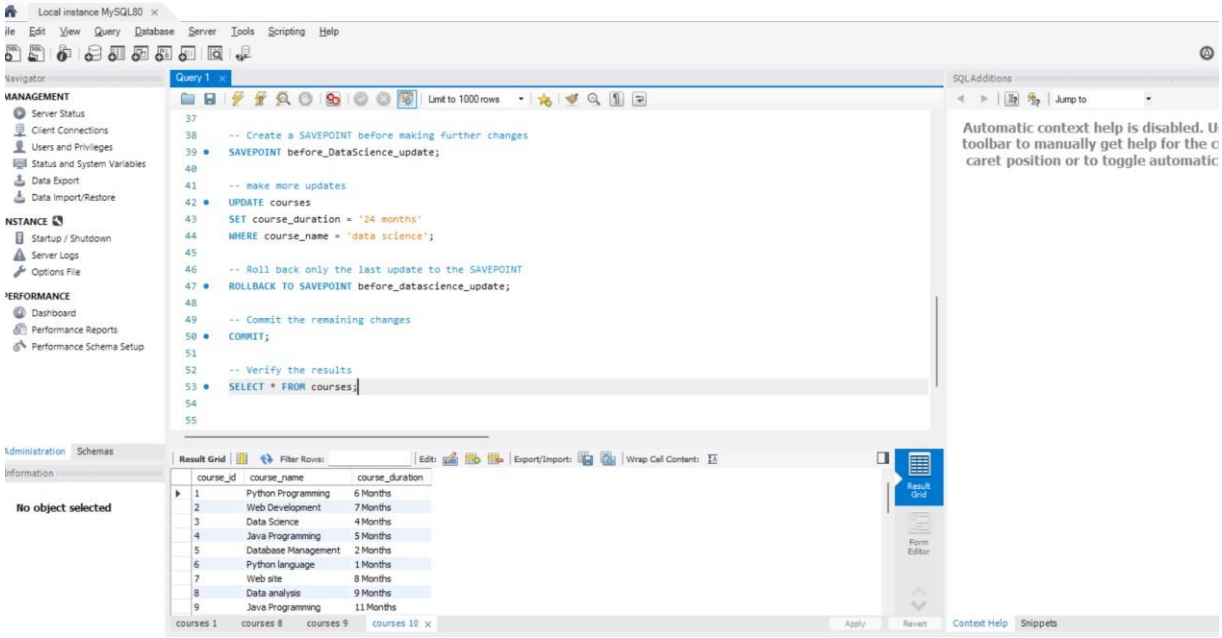
Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

course_id	course_name	course_duration
1	Python Programming	6 Months
2	Web Development	7 Months
3	Data Science	4 Months
4	Java Programming	5 Months
5	Database Management	2 Months
6	Python language	1 Months
7	Web site	8 Months
8	Data analysis	9 Months
9	Java Programming	11 Months

courses 1 | courses 8 | courses 9 | courses 10 | Apply | Revert

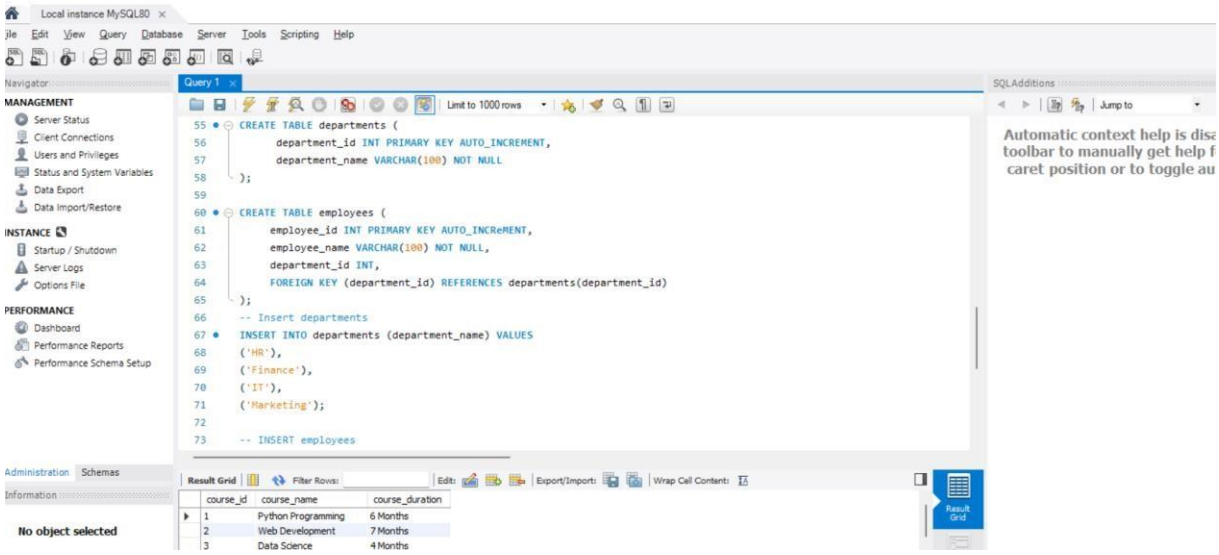
- **Lab 3: Create a SAVEPOINT before updating the courses table, and use it to roll back specific changes.**

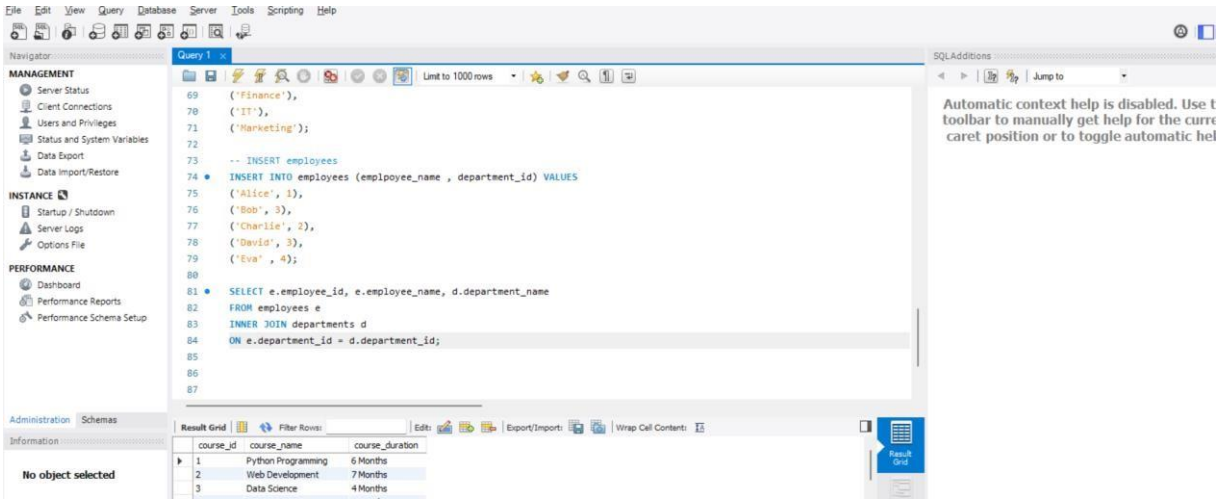




11. SQL Joins

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





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

```
mysql> SELECT d.dept_id, d.dept_name, e.emp_id, e.emp_name
    -> FROM departments d
    -> LEFT JOIN employees e
    -> ON d.dept_id = e.dept_id;
+---------+-----------+--------+----------+
| dept_id | dept_name | emp_id | emp_name |
+---------+-----------+--------+----------+
|       1 | HR        |    101 | Alice    |
|       2 | Finance   |    103 | Charlie  |
|       3 | IT        |    102 | Bob      |
|       3 | IT        |    104 | David    |
|       4 | Marketing |    105 | Eva      |
+---------+-----------+--------+----------+
5 rows in set (0.00 sec)
```

12. SQL Group By

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

```
mysql> SELECT d.dept_name, COUNT(e.emp_id) AS num_employees
    -> FROM departments d
    -> INNER JOIN employees e
    -> ON d.dept_id = e.dept_id
    -> GROUP BY d.dept_name;
+-----------+---------------+
| dept_name | num_employees |
+-----------+---------------+
| HR        |             1 |
| Finance   |             1 |
| IT        |             2 |
| Marketing |             1 |
+-----------+---------------+
4 rows in set (0.00 sec)
```

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

```
mysql> DROP TABLE IF EXISTS employees;
Query OK, 0 rows affected (0.02 sec)

mysql>
mysql> CREATE TABLE employees (
    ->     emp_id INT PRIMARY KEY,
    ->     emp_name VARCHAR(50),
    ->     dept_id INT,
    ->     salary DECIMAL(10,2),
    ->     FOREIGN KEY (dept_id) REFERENCES departments(dept_id)
    -> );
Query OK, 0 rows affected (0.03 sec)

mysql> INSERT INTO employees (emp_id, emp_name, dept_id, salary) VALUES
    -> (101, 'Alice', 1, 50000),
    -> (102, 'Bob', 3, 60000),
    -> (103, 'Charlie', 2, 55000),
    -> (104, 'David', 3, 65000),
    -> (105, 'Eva', 4, 52000);
Query OK, 5 rows affected (0.01 sec)
Records: 5  Duplicates: 0  Warnings: 0

mysql> SELECT d.dept_name, AVG(e.salary) AS avg_salary
    -> FROM departments d
    -> INNER JOIN employees e
    -> ON d.dept_id = e.dept_id
    -> GROUP BY d.dept_name;
+-----------+--------------+
| dept_name | avg_salary   |
+-----------+--------------+
| HR        | 50000.000000 |
| Finance   | 55000.000000 |
| IT        | 62500.000000 |
| Marketing | 52000.000000 |
+-----------+--------------+
4 rows in set (0.01 sec)
```

13. SQL Stored Procedure

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

```
mysql> CREATE VIEW employee_department_view AS
    -> SELECT e.emp_id,
    ->        e.emp_name,
    ->        e.salary,
    ->        d.dept_name
    -> FROM employees e
    -> INNER JOIN departments d
    -> ON e.dept_id = d.dept_id;
Query OK, 0 rows affected (0.01 sec)

mysql> SELECT * FROM employee_department_view;
+--------+----------+----------+-----------+
| emp_id | emp_name | salary   | dept_name |
+--------+----------+----------+-----------+
|    101 | Alice    | 50000.00 | HR        |
|    103 | Charlie  | 55000.00 | Finance   |
|    102 | Bob      | 60000.00 | IT        |
|    104 | David    | 65000.00 | IT        |
|    105 | Eva      | 52000.00 | Marketing |
+--------+----------+----------+-----------+
5 rows in set (0.00 sec)
```

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

```
+--------+----------+----------+------------+
| emp_id | emp_name | salary   | dept_name  |
+--------+----------+----------+------------+
|    101 | Alice    | 50000.00 | HR         |
|    102 | Bob      | 60000.00 | IT         |
|    103 | Charlie  | 55000.00 | Finance    |
|    104 | David    | 65000.00 | IT         |
|    105 | Eva      | 52000.00 | Marketing  |
+--------+----------+----------+------------+
5 rows in set (0.00 sec)
```

```
mysql> UPDATE employees
    -> SET salary = 45000
    -> WHERE emp_name = 'Charlie';
Query OK, 0 rows affected (0.00 sec)
Rows matched: 1  Changed: 0  Warnings: 0

mysql> SELECT * FROM employee_department_view;
+--------+----------+----------+-----------+
| emp_id | emp_name | salary   | dept_name |
+--------+----------+----------+-----------+
|    101 | Alice    | 50000.00 | HR        |
|    102 | Bob      | 60000.00 | IT        |
|    104 | David    | 65000.00 | IT        |
|    105 | Eva      | 52000.00 | Marketing |
+--------+----------+----------+-----------+
4 rows in set (0.00 sec)
```

15. SQL Triggers

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

```
mysql> CREATE TABLE employee_log (
    ->     log_id INT AUTO_INCREMENT PRIMARY KEY,
    ->     emp_id INT,
    ->     emp_name VARCHAR(50),
    ->     dept_id INT,
    ->     salary DECIMAL(10,2),
    ->     action_time TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    ->     action_type VARCHAR(20)
    -> );
Query OK, 0 rows affected (0.04 sec)

mysql> DELIMITER //
mysql>
mysql> CREATE TRIGGER after_employee_insert
    -> AFTER INSERT ON employees
    -> FOR EACH ROW
    -> BEGIN
    ->     INSERT INTO employee_log (emp_id, emp_name, dept_id, salary, action_type)
    ->     VALUES (NEW.emp_id, NEW.emp_name, NEW.dept_id, NEW.salary, 'INSERT');
    -> END;
    ->
    -> //
Query OK, 0 rows affected (0.02 sec)

mysql>
mysql> DELIMITER ;
mysql> INSERT INTO employees (emp_id, emp_name, dept_id, salary)
    -> VALUES (106, 'Frank', 2, 48000);
Query OK, 1 row affected (0.01 sec)

mysql> SELECT * FROM employee_log;
+--------+--------+----------+---------+----------+---------------------+-------------+
| log_id | emp_id | emp_name | dept_id | salary   | action_time         | action_type |
+--------+--------+----------+---------+----------+---------------------+-------------+
|      1 |    106 | Frank    |       2 | 48000.00 | 2025-09-26 12:17:59 | INSERT      |
+--------+--------+----------+---------+----------+---------------------+-------------+
1 row in set (0.00 sec)
```

- **Lab 2: Create a trigger to update the last_modified timestamp whenever an employee record is updated.**

```
mysql> ALTER TABLE employees
    -> ADD COLUMN last_modified TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP;
Query OK, 0 rows affected (0.02 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> DELIMITER //
mysql>
mysql> CREATE TRIGGER before_employee_update
    -> BEFORE UPDATE ON employees
    -> FOR EACH ROW
    -> BEGIN
    ->     SET NEW.last_modified = CURRENT_TIMESTAMP;
    -> END;
    -> //
Query OK, 0 rows affected (0.02 sec)

mysql>
mysql> DELIMITER ;
mysql> UPDATE employees
    -> SET salary = 62000
    -> WHERE emp_id = 102;
Query OK, 1 row affected (0.01 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> SELECT emp_id, emp_name, salary, last_modified
    -> FROM employees
    -> WHERE emp_id = 102;
+--------+----------+----------+---------------------+
| emp_id | emp_name | salary   | last_modified       |
+--------+----------+----------+---------------------+
|    102 | Bob      | 62000.00 | 2025-09-26 12:20:09 |
+--------+----------+----------+---------------------+
1 row in set (0.00 sec)
```

16. Introduction to PL/SQL

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

```sql
DECLARE
    total_employees NUMBER;  -- Variable to store the total count
BEGIN
    -- Get the total number of employees
    SELECT COUNT(*)
    INTO total_employees
    FROM employees;

    -- Print the total number of employees
    DBMS_OUTPUT.PUT_LINE('Total number of employees: ' || total_employees);
END;
/
```

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

```sql
DECLARE
    total_sales NUMBER;  -- Variable to store total sales
BEGIN
    -- Calculate total sales
    SELECT SUM(order_amount)
    INTO total_sales
    FROM orders;

    -- Print the total sales
    DBMS_OUTPUT.PUT_LINE('Total sales: ' || NVL(total_sales, 0));
END;
/
```

17. PL/SQL Control Structures

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

```sql
DECLARE
    emp_id       NUMBER := 101;  -- Employee ID to check
    dept_id      NUMBER;
BEGIN
    -- Get the department of the employee
    SELECT department_id
    INTO dept_id
    FROM employees
    WHERE employee_id = emp_id;
```

```
    -- Check the department using IF-THEN
    IF dept_id = 10 THEN
        DBMS_OUTPUT.PUT_LINE('Employee ' || emp_id || ' works in the Accounting department.');
    ELSIF dept_id = 20 THEN
        DBMS_OUTPUT.PUT_LINE('Employee ' || emp_id || ' works in the Sales department.');
    ELSE
        DBMS_OUTPUT.PUT_LINE('Employee ' || emp_id || ' works in another department.');
    END IF;
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        DBMS_OUTPUT.PUT_LINE('Employee ' || emp_id || ' does not exist.');
END;
/
```

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

```
BEGIN
    -- Loop through all employee records
    FOR emp_rec IN (SELECT first_name, last_name FROM employees) LOOP
        -- Display employee full name
        DBMS_OUTPUT.PUT_LINE('Employee Name: ' || emp_rec.first_name || ' ' || emp_rec.last_name);
    END LOOP;
END;
/
```

18. SQL Cursors

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

```
DECLARE
    -- Declare an explicit cursor to select employee details
    CURSOR emp_cursor IS
        SELECT employee_id, first_name, last_name, department_id
        FROM employees;

    -- Declare a record variable to hold each row fetched from the cursor
    emp_record emp_cursor%ROWTYPE;
BEGIN
    -- Open the cursor
    OPEN emp_cursor;
```

```
    -- Loop through each record
    LOOP
        FETCH emp_cursor INTO emp_record;
        EXIT WHEN emp_cursor%NOTFOUND;  -- Exit loop when no more records

        -- Display employee details
        DBMS_OUTPUT.PUT_LINE('ID: ' || emp_record.employee_id ||
                             ', Name: ' || emp_record.first_name || ' ' || emp_record.last_name ||
                             ', Dept: ' || emp_record.department_id);
    END LOOP;

    -- Close the cursor
    CLOSE emp_cursor;
END;
/
```

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

```
DECLARE
    -- Declare an explicit cursor to select courses
    CURSOR course_cursor IS
        SELECT course_id, course_name
        FROM courses;

    -- Declare a record variable for each row
    course_rec course_cursor%ROWTYPE;
BEGIN
    -- Open the cursor
    OPEN course_cursor;
```

```
    -- Loop through each course
    LOOP
        FETCH course_cursor INTO course_rec;
        EXIT WHEN course_cursor%NOTFOUND;  -- Stop when no more records

        -- Display course details
        DBMS_OUTPUT.PUT_LINE('Course ID: ' || course_rec.course_id ||
                             ', Course Name: ' || course_rec.course_name);
    END LOOP;

    -- Close the cursor
    CLOSE course_cursor;
END;
/
```

19. Rollback and Commit Savepoint

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

```
BEGIN
    -- Insert first record
    INSERT INTO employees (employee_id, first_name, last_name, department_id)
    VALUES (301, 'Amit', 'Sharma', 10);

    -- Create a savepoint after first insert
    SAVEPOINT sp1;

    -- Insert second record
    INSERT INTO employees (employee_id, first_name, last_name, department_id)
    VALUES (302, 'Neha', 'Patel', 20);

    -- Insert third record
    INSERT INTO employees (employee_id, first_name, last_name, department_id)
    VALUES (303, 'Ravi', 'Kumar', 30);

    -- Rollback to the savepoint (removes 302, 303 but keeps 301)
    ROLLBACK TO sp1;
```

```
    -- Commit final (Amit record save hoga)
    COMMIT;

    DBMS_OUTPUT.PUT_LINE('Transaction complete: Only Amit record saved, Neha & Ravi rolled back.');
END;
/
```

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

```
SET SERVEROUTPUT ON;

BEGIN
    -- Insert first record
    INSERT INTO employees (employee_id, first_name, last_name, department_id)
    VALUES (401, 'Karan', 'Mehta', 10);

    -- Create savepoint after first insert
    SAVEPOINT sp1;

    -- Insert second record
    INSERT INTO employees (employee_id, first_name, last_name, department_id)
    VALUES (402, 'Priya', 'Singh', 20);

    -- Insert third record
    INSERT INTO employees (employee_id, first_name, last_name, department_id)
    VALUES (403, 'Rohan', 'Patel', 30);
```

```
    -- Commit the transaction till savepoint (Karan will be permanent)
    COMMIT;

    -- Now rollback remaining changes (Priya & Rohan will be undone)
    ROLLBACK;

    DBMS_OUTPUT.PUT_LINE('Transaction complete: Karan committed, Priya & Rohan rolled back.');
END;
/
```