**Module 4 – Introduction to DBMS**

**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.

**Ans: -**

CREATE DATABASE school\_db;

USE school\_db;

CREATE TABLE students (

student\_id INT PRIMARY KEY AUTO\_INCREMENT,

student\_name VARCHAR(100) NOT NULL,

age INT,

class VARCHAR(50),

address VARCHAR(255)

);

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

statement.

**Ans: -**

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

VALUES

('Drashti Dave', 20, '12', 'Ahmedabad'),

('Jaydeep Raval', 20, '12', 'Ahmedabad'),

('Sweta', 18, '12', 'Botad'),

('Brij', 18, '12', 'Morbi'),

('Varsha', 16, '10', 'Morbi');

**Retrieve All Records:-**

SELECT \* FROM students;

**2. SQL Syntax**

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

students table.

**Ans: -**

SELECT student\_name, age FROM students;

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

**Ans: -**

SELECT \* FROM students WHERE 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).

**Ans: -**

CREATE TABLE teachers (

teacher\_id INT PRIMARY KEY,

teacher\_name VARCHAR(100) NOT NULL,

subject VARCHAR(100) NOT NULL,

email VARCHAR(100) UNIQUE

);

**Lab 2:**

**Ans: -**

ALTER TABLE students

ADD teacher\_id INT,

ADD CONSTRAINT fk\_teacher

FOREIGN KEY (teacher\_id) REFERENCES teachers(teacher\_id);

**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.

**Ans: -**

CREATE TABLE courses (

course\_id INT PRIMARY KEY,

course\_name VARCHAR(100),

course\_credits INT

);

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

**Ans: -**

CREATE DATABASE university\_db;

**5. ALTER Command**

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

command.

**Ans: -**

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

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

**Ans: -**

ALTER TABLE courses DROP COLUMN course\_credits;

**6. DROP Command**

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

**Ans: -**

DROP TABLE teachers;

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

been removed.

**Ans: -**

DROP TABLE students;

To verify:

SHOW TABLES;

**7. DML (Data Manipulation Language)**

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

**Ans: -**

INSERT INTO courses (course\_id, course\_name, course\_duration)

VALUES

(1, 'Math', '3 Months'),

(2, 'Science', '4 Months'),

(3, 'English', '2 Months');

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

**Ans: -**

UPDATE courses SET course\_duration = '5 Months' WHERE course\_id = 2;

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

command.

**Ans: -**

DELETE FROM courses WHERE course\_id = 3;

**8. DQL (Data Query Language)**

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

**Ans: -**

SELECT \* FROM courses;

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

**Ans: -**

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.

**Ans: -**

SELECT \* FROM courses LIMIT 2;

**9. DCL (Data Control Language)**

**Lab 1:** Create two new users user1 and user2 and grant user1 permission to SELECT

from the courses table.

**Ans: -**

CREATE USER 'user1'@'localhost' IDENTIFIED BY 'password';

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

GRANT SELECT ON university\_db.courses TO 'user1'@'localhost';

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

**Ans: -**

REVOKE INSERT ON university\_db.courses FROM 'user1'@'localhost';

GRANT INSERT ON university\_db.courses TO 'user2'@'localhost';

**10. TCL (Transaction Control Language)**

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

**Ans: -**

START TRANSACTION;

INSERT INTO courses VALUES (4, 'History', '3 Months');

COMMIT;

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

**Ans: -**

START TRANSACTION;

INSERT INTO courses VALUES (5, 'Geography', '4 Months');

ROLLBACK;

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

specific changes.

**Ans: -**

START TRANSACTION;

SAVEPOINT sp1;

UPDATE courses SET course\_duration = '6 Months' WHERE course\_id = 1;

ROLLBACK TO sp1;

**11. SQL Joins**

**Lab 1:** Create two tables: departments and employees. Perform an INNER JOIN to

display employees along with their respective departments.

**Ans: -**

CREATE TABLE departments (

dept\_id INT PRIMARY KEY,

dept\_name VARCHAR(100)

);

CREATE TABLE employees (

emp\_id INT PRIMARY KEY,

emp\_name VARCHAR(100),

dept\_id INT,

FOREIGN KEY (dept\_id) REFERENCES departments(dept\_id)

);

SELECT emp\_name, dept\_name

FROM employees

INNER JOIN departments ON employees.dept\_id = departments.dept\_id;

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

**Ans: -**

SELECT dept\_name, emp\_name

FROM departments

LEFT JOIN employees ON departments.dept\_id = employees.dept\_id;

**12. SQL GROUP BY**

**Lab 1:** Group employees by department and count the number of employees in each

department using GROUP BY.

**Ans: -**

SELECT dept\_id, COUNT(\*) AS num\_employees

FROM employees

GROUP BY dept\_id;

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

department.

**Ans: -**

SELECT dept\_id, AVG(salary) AS avg\_salary

FROM employees

GROUP BY dept\_id;

**13. Stored Procedures**

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

on department.

**Ans: -**

CREATE PROCEDURE GetEmployeesByDept(IN dept INT)

BEGIN

SELECT \* FROM employees WHERE dept\_id = dept;

END

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

details.

**Ans: -**

CREATE PROCEDURE GetCourseDetails(IN cid INT)

BEGIN

SELECT \* FROM courses WHERE course\_id = cid;

END

**14. SQL Views**

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

**Ans: -**

CREATE VIEW emp\_dept\_view AS

SELECT emp\_name, dept\_name

FROM employees

JOIN departments ON employees.dept\_id = departments.dept\_id;

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

**Ans: -**

CREATE OR REPLACE VIEW emp\_dept\_view AS

SELECT emp\_name, dept\_name

FROM employees

JOIN departments ON employees.dept\_id = departments.dept\_id

WHERE salary >= 50000;

**15. SQL Triggers**

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

employee is added.

**Ans: -**

CREATE TABLE emp\_log (

log\_id INT AUTO\_INCREMENT PRIMARY KEY,

emp\_name VARCHAR(100),

log\_time TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

);

CREATE TRIGGER log\_new\_employee

AFTER INSERT ON employees

FOR EACH ROW

INSERT INTO emp\_log (emp\_name) VALUES (NEW.emp\_name);

**Lab 2:** Create a trigger to update the last\_modified timestamp whenever an employee

record is updated.

**Ans: -**

ALTER TABLE employees ADD last\_modified TIMESTAMP;

CREATE TRIGGER update\_last\_modified

BEFORE UPDATE ON employees

FOR EACH ROW

SET NEW.last\_modified = CURRENT\_TIMESTAMP;

**16. Introduction to PL/SQL**

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

Table.

**Ans: -**

DECLARE

total\_emps INT;

BEGIN

SELECT COUNT(\*) INTO total\_emps FROM employees;

DBMS\_OUTPUT.PUT\_LINE('Total Employees: ' || total\_emps);

END;

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

**Ans: -**

DECLARE

total\_sales NUMBER;

BEGIN

SELECT SUM(sale\_amount) INTO total\_sales FROM orders;

DBMS\_OUTPUT.PUT\_LINE('Total Sales: ' || total\_sales);

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.

**Ans: -**

DECLARE

dept VARCHAR2(100);

BEGIN

SELECT dept\_name INTO dept FROM employees WHERE emp\_id = 1;

IF dept = 'HR' THEN

DBMS\_OUTPUT.PUT\_LINE('Employee is in HR');

END IF;

END;

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

**Ans: -**

DECLARE

emp\_name VARCHAR2(100);

CURSOR emp\_cursor IS SELECT emp\_name FROM employees;

BEGIN

FOR emp\_record IN emp\_cursor LOOP

DBMS\_OUTPUT.PUT\_LINE('Employee: ' || emp\_record.emp\_name);

END LOOP;

END;

**18. SQL Cursors**

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

**Ans: -**

DECLARE

CURSOR emp\_cursor IS SELECT emp\_id, emp\_name FROM employees;

emp\_record emp\_cursor%ROWTYPE;

BEGIN

OPEN emp\_cursor;

LOOP

FETCH emp\_cursor INTO emp\_record;

EXIT WHEN emp\_cursor%NOTFOUND;

DBMS\_OUTPUT.PUT\_LINE(emp\_record.emp\_id || ' - ' || emp\_record.emp\_name);

END LOOP;

CLOSE emp\_cursor;

END;

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

**Ans: -**

DECLARE

CURSOR course\_cursor IS SELECT course\_name FROM courses;

course\_name\_var courses.course\_name%TYPE;

BEGIN

OPEN course\_cursor;

LOOP

FETCH course\_cursor INTO course\_name\_var;

EXIT WHEN course\_cursor%NOTFOUND;

DBMS\_OUTPUT.PUT\_LINE(course\_name\_var);

END LOOP;

CLOSE course\_cursor;

END;

**19. Savepoint, Rollback, Commit**

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

the savepoint.

**Ans: -**

START TRANSACTION;

INSERT INTO courses VALUES (6, 'Philosophy', '2 Months');

SAVEPOINT sp1;

INSERT INTO courses VALUES (7, 'Music', '3 Months');

ROLLBACK TO sp1;

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

changes.

**Ans: -**

START TRANSACTION;

INSERT INTO courses VALUES (8, 'Drama', '4 Months');

SAVEPOINT sp2;

COMMIT;

INSERT INTO courses VALUES (9, 'Dance', '3 Months');

ROLLBACK TO sp2;