Extra Lab Practice…

1.Introduction to sql…

Lab-1:

```
-> CREATE TABLE students (
   student_id INT PRIMARY KEY,
   student_name VARCHAR(100) NOT NULL,
);
```

```
INSERT INTO `students` (`student_id`, `student_name`, `age`, `class`, `address`) VALUES (1, 'rutvi', '20', 'bca', 'abc');

->INSERT INTO `students` (`student_id`, `student_name`, `age`, `class`, `address`) VALUES (2, 'hiral', '15', '2th', 'pqr');

->INSERT INTO `students` (`student_id`, `student_name`, `age`, `class`, `address`) VALUES (3, 'priyanshi', '45', '1st', 'xyz');

->INSERT INTO `students` (`student_id`, `student_name`, `age`, `class`, `address`) VALUES (4, 'vaishvi', '56', 'bba', 'mno');

->INSERT INTO `students` (`student_id`, `student_name`, `age`, `class`, `address`) VALUES (5, 'abc', '50', 'bcom', 'qwty');
```

Lab-2:

```
->select * from students;
```

2.SQL Syntax…

Lab-1:

```
->SELECT student_name,age FROM students;
```

Lab-2:

```
->SELECT age FROM students where age>10;
```

3.SQL Contraints…

Lab-1:

```
-> 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:

```
-> CREATE TABLE students (
   student_id INT PRIMARY KEY,
   student_name VARCHAR(100) NOT NULL,
);
 ALTER TABLE students
ADD CONSTRAINT fk_teacher
FOREIGN KEY (teacher_id) REFERENCES teachers(teacher_id);
```

---

4. Main SQL Commands and Sub-commands (DDL)..

Lab-1:

```
->CREATE TABLE courses (
   course_id INT PRIMARY KEY,
   course_name VARCHAR(100),
   course_credits INT
);
```

Lab-2:

```
-> CREATE DATABASE university_db;
```

---

5. ALTER Command..

Lab-1:

```
-> ALTER TABLE courses ADD COLUMN course_duration VARCHAR(50);
```

Lab-2:

ALTER TABLE courses DROP COLUMN course_credits;

---

6. DROP Command..

Lab-1**:**

->ALTER TABLE students DROP FOREIGN KEY fk_teacher;

->DROP TABLE teachers;

Lab-2:

-> DROP TABLE students;

---

7. Data Manipulation Language (DML)…

Lab-1:

->INSERT INTO courses(course_name)VALUES('rutvi');

INSERT INTO courses(course_name)VALUES('hiral');

INSERT INTO courses(course_name)VALUES('priyanshi');

Lab-2:

-> UPDATE courses SET course_duration = '6 months' WHERE course_id = 1;

Lab-3:

-> DELETE FROM course WHERE course_id = 1;

---

8. Data Query Language (DQL)..

Lab-1:

->select * from courses;

Lab-2:

-> SELECT * FROM courses ORDER BY course_duration DESC;

Lab-3:

-> SELECT * FROM courses LIMIT 2;

---

9. Data Control Language (DCL)..

Lab-1:

->GRANT SELECT ON courses TO user1;

Lab-2:

->REVOKE SELECT ON courses TO user1;

---

10. Transaction Control Language (TCL)..

Lab-1:

```
->START TRANSACTION;
INSERT INTO courses (course_name, course_duration)VALUES ('Networking', '3');
INSERT INTO courses (course_name, course_duration)VALUES ('Networking', '3');
COMMIT;
```

Lab-2:

```
-> INSERT INTO courses (course_name, course_duration)VALUES ('Networking', '3');
ROLLBACK;
```

Lab-3:

```
-> START TRANSACTION;
SAVEPOINT before_insert;
INSERT INTO courses (course_name, course_duration)
VALUES ('Java Advanced', 12);
ROLLBACK TO before_insert;
COMMIT;
```

---

11. SQL Joins..

Lab-1:

```
-> CREATE TABLE  employess (
   emp_id INT PRIMARY KEY,
   emp_name VARCHAR(100)
);
CREATE TABLE  department (
   dpart_id INT PRIMARY KEY,
   dpat_name VARCHAR(100),
   emp_id int
);
```

```
ALTER TABLE department ADD CONSTRAINT emp_id FOREIGN KEY(emp
_id)REFERENCES employees(emp_id);
SELECT
   employees.emp_id,
   employees.emp_name,
   department.department_name
FROM
   employees
INNER JOIN
   department ON employees.emp_id = department.department_id;
```

lab-2:

```
-> SELECT
   department.department_id,
   department.department_name,
   employees.emp_id,
   employees.emp_name
FROM
   department
LEFT JOIN
   employees ON department.emp_id = employees.emp_id;
```

---

12. SQL Group By..

Lab-1:

```
-> SELECT *,
   COUNT(*) AS employee_count
FROM
   employees
```

```
GROUP BY
   emp_id;
```

Lab-2:

```
-> SELECT emp_id, AVG(salary) AS average_salary FROM department GROUP BY emp_id LIMIT 1;
```

---

13. **SQL Stored Procedure..**

Lab-1:

```
-> DELIMITER $$
CREATE PROCEDURE GetEmployeesByDepartment(IN dept_id INT)
BEGIN
   SELECT *
   FROM employees
   WHERE department_id = dept_id;
END $$
DELIMITER ;
```

Lab-2:

```
-> DELIMITER $$

CREATE PROCEDURE GetCourseDetails(IN input_course_id INT)
BEGIN
   SELECT *
   FROM courses
   WHERE course_id = input_course_id;
END $$

DELIMITER ;
```

---

14. SQL View..

Lab-1:

```
-> CREATE VIEW employee_department_view AS
SELECT
    e.emp_id,
    e.emp_name,
    e.salary,
    d.department_id,
    d.department_name
FROM
    employees e
JOIN
    department d ON d.department_id = d.department_id;
```

Lab-2:

```
-> CREATE VIEW employee_department_view AS
SELECT
    e.emp_id,
    e.emp_name,
    e.salary,
    d.department_id,
    d.department_name
FROM
    employees e
JOIN
    department d ON d.department_id = d.department_id
WHERE
    e.salary >= 50000;
```

15.sql trigger..

Lab-1:

```
-> CREATE TABLE employee_log (
    log_id INT AUTO_INCREMENT PRIMARY KEY,
    emp_id INT,
    emp_name VARCHAR(100),
    action_time DATETIME,
    action_type VARCHAR(20)
);
DELIMITER $$

CREATE TRIGGER log_new_employee
AFTER INSERT ON employees
FOR EACH ROW
BEGIN
    INSERT INTO employee_log(emp_id, emp_name,
action_time, action_type)
    VALUES (NEW.emp_id, NEW.emp_name, NOW(), 'INSERT');
END $$

DELIMITER ;
```

Lab-2:

```
-> ALTER TABLE employees
ADD COLUMN last_modified DATETIME;
DELIMITER $$

CREATE TRIGGER update_last_modified
BEFORE UPDATE ON employees
FOR EACH ROW
BEGIN
   SET NEW.last_modified = NOW();
END $$

DELIMITER ;
```

---

16. **Introduction to PL/SQL..**

Lab-1:

```
DECLARE
 v_total_employees NUMBER;
BEGIN
 SELECT COUNT(*) INTO v_total_employees FROM employees;
 DBMS_OUTPUT.PUT_LINE('Total number of employees: ' || v_total_employees);
END;
/
```

Lab-2:

```
DECLARE
  v_total_sales NUMBER;
BEGIN
  SELECT SUM(order_amount) INTO v_total_sales FROM orders;
  DBMS_OUTPUT.PUT_LINE('Total sales amount: $' || v_total_sales);
END;
/
```

17. **Introduction to PL/SQL..**

```
1 . DECLARE
v_employee_id employees.employee_id%TYPE := 101; -- Change as needed
v_department employees.department%TYPE;
BEGIN
SELECT department INTO v_department
FROM employees
WHERE employee_id = v_employee_id;
IF v_department = 'HR' THEN
DBMS_OUTPUT.PUT_LINE('Employee belongs to the HR department.');
ELSE
DBMS_OUTPUT.PUT_LINE('Employee does not belong to the HR department.');
END IF;
END;
/
2 . DECLARE
CURSOR emp_cursor IS
SELECT name FROM employees;
BEGIN
```

```
FOR emp_rec IN emp_cursor LOOP
DBMS_OUTPUT.PUT_LINE('Employee Name: ' || emp_rec.name);
END LOOP;
END;
/
```

18. **Introduction to PL/SQL..**

```
1 . DECLARE
CURSOR emp_cursor IS
SELECT employee_id, name, department, salary
FROM employees;
v_emp_id employees.employee_id%TYPE;
v_name employees.name%TYPE;
v_dept employees.department%TYPE;
v_salary employees.salary%TYPE;
BEGIN
OPEN emp_cursor;
LOOP
FETCH emp_cursor INTO v_emp_id, v_name, v_dept, v_salary;
EXIT WHEN emp_cursor%NOTFOUND;
DBMS_OUTPUT.PUT_LINE('ID: ' || v_emp_id || ', Name: ' || v_name ||
', Department: ' || v_dept || ', Salary: ' || v_salary);
END LOOP;
CLOSE emp_cursor;
END;
/
```

2 . DECLARE

CURSOR course_cursor IS

SELECT course_id, course_name, duration

FROM courses;

v_course_id courses.course_id%TYPE;

v_course_name courses.course_name%TYPE;

v_duration courses.duration%TYPE;

BEGIN

OPEN course_cursor;

LOOP

FETCH course_cursor INTO v_course_id, v_course_name, v_duration;

EXIT WHEN course_cursor%NOTFOUND;

DBMS_OUTPUT.PUT_LINE('Course ID: ' || v_course_id || ', Name: ' ||
v_course_name

||

', Duration: ' || v_duration || ' hours');

END LOOP;

CLOSE course_cursor;

END;

/

19. **Introduction to PL/SQL..**

1 . BEGIN

-- Start Transaction

INSERT INTO employees (employee_id, name, department, salary)

VALUES (201, 'Alice Smith', 'Finance', 60000);

```
SAVEPOINT emp_insert_savepoint;
INSERT INTO employees (employee_id, name, department, salary)
VALUES (202, 'Bob Johnson', 'HR', 55000);
-- Something goes wrong; rollback only the second insert
ROLLBACK TO emp_insert_savepoint;
-- Commit the first insert
COMMIT;
DBMS_OUTPUT.PUT_LINE('Transaction rolled back to savepoint. First insert
committed.');
END;
/
2 . BEGIN
-- First part of the transaction
INSERT INTO employees (employee_id, name, department, salary)
VALUES (203, 'Charlie Brown', 'IT', 70000);
SAVEPOINT part1_done;
-- Second part of the transaction
INSERT INTO employees (employee_id, name, department, salary)
VALUES (204, 'Diana Prince', 'Marketing', 52000);
-- Commit first insert (Charlie)
COMMIT;
-- Something goes wrong with second insert
ROLLBACK TO part1_done;
DBMS_OUTPUT.PUT_LINE('First insert committed. Second insert rolled back.');
END;
/
```