Ans - 1

1 .
2 . CREATE TABLE students(student_id int PRIMARY KEY AUTO_INCREMENT , student_name varchar(50) , age int , class int , address varchar(50));

3 . INSERT INTO `students`(`student_id`, `student_name`, `age`, `class`, `address`) VALUES ('1','shubham','22','1','valasan')

4 . SELECT * FROM `students` ;

Ans - 2

1 . SELECT student_name , age FROM `students`;

2 . SELECT age FROM `students` WHERE age > 10;

Ans - 3

1. CREATE TABLE Teachers(teacher_id int PRIMARY KEY AUTO_INCREMENT, teacher_name varchar(50),subject varchar(50),email varchar(50)UNIQUE);

2 . ALTER TABLE students ADD COLUMN teacher_id varchar(50);

3 . ALTER TABLE students ADD CONSTRAINT teacher_id FOREIGN KEY (teacher_id) REFERENCES teachers(teacher_id);

Ans - 4

1 . CREATE TABLE courses (course_id int PRIMARY KEY AUTO_INCREMENT , course_name varchar(50) , course_credits int);

2 . CREATE DATABASE university_db;

Ans - 5

1 . ALTER TABLE courses ADD course_duration int;

2 . ALTER TABLE courses DROP COLUMN course_credits;

Ans - 6

1 . DROP TABLE students;
2 . DROP TABLE teachers;

Ans - 7

```
1 . INSERT INTO `courses`(`course_id`, `course_name`, `course_duration`) VALUES
('1','shubham','5')
2 . UPDATE `courses` SET `course_duration`='7' WHERE course_id = 1;
3 . DELETE FROM courses WHERE course_id = 1;
```

Ans - 8

```
1 . SELECT * FROM `courses` ;
2 . SELECT * FROM courses ORDER by course_duration DESC;
3 . SELECT * FROM `courses` LIMIT 2;
```

Ans - 9

```
1 . GRANT SELECT ON courses TO USER 1 ;
2 . REVOKE SELECT ON courses TO USER 1 ;
```

Ans - 10

```
1 . INSERT INTO `courses`(`course_id`, `course_name`, `course_duration`) VALUES
('[value-1]','[value-2]','[value-3]');
commit;

2 . INSERT INTO `courses`(`course_id`, `course_name`, `course_duration`) VALUES
('[value-1]','[value-2]','[value-3]');
Rollback;

3 . START TRANSACTION;

SAVEPOINT before_update;

UPDATE courses
SET course_duration = 10
WHERE course_id = 3;

ROLLBACK TO SAVEPOINT before_update;

COMMIT;
```

Ans - 11

1 . CREATE TABLE dpt(d_id int PRIMARY KEY AUTO_INCREMENT); && CREATE TABLE employees(e_id int PRIMARY KEY AUTO_INCREMENT);

```
 ALTER TABLE dpt ADD CONSTRAINT e_id FOREIGN KEY(e_id)REFERENCES
employees(e_id);
```

```
 SELECT employees.e_id AS employee_name, dpt.d_id
FROM employees
INNER JOIN dpt
ON employees.e_id = dpt.d_id;
```

2 . SELECT dpt.d_id, employees.e_id AS employee_name

```
FROM dpt
LEFT JOIN employees
ON dpt.d_id= employees.e_id;
```

Ans - 12

1 . SELECT *,

```
   COUNT(*) AS employee_count
FROM
   employee
GROUP BY
   Employee_name;
```

```
ALTER TABLE department ADD COLUMN salary int;
```

2 . SELECT emp_id, AVG(salary)

```
 AS average_salary FROM department
GROUP BY emp_id LIMIT 1;
```

Ans - 13

1 . DELIMITER $$

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

DELIMITER ;
```

2 . DELIMITER $$

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

DELIMITER ;
```

Ans - 14

1 . DELIMITER $$

```
CREATE PROCEDURE CreateEmployeeDepartmentView()
BEGIN
   CREATE OR REPLACE VIEW EmployeeDepartmentView AS
   SELECT e.employee_id,
         e.name AS employee_name,
         e.salary,
         d.department_name
   FROM employees e
   JOIN departments d ON e.department_id = d.department_id;
END$$

DELIMITER ;
```

```
2 . DELIMITER $$

CREATE PROCEDURE UpdateEmployeeDepartmentView()
BEGIN
   CREATE OR REPLACE VIEW EmployeeDepartmentView AS
   SELECT e.employee_id,
        e.name AS employee_name,
        e.salary,
        d.department_name
   FROM employees e
   JOIN departments d ON e.department_id = d.department_id
   WHERE e.salary >= 50000;
END$$

DELIMITER ;
```

Ans - 15

```
1 . DELIMITER $$

CREATE PROCEDURE CreateInsertLogTrigger()
BEGIN
   CREATE TRIGGER LogNewEmployee
   AFTER INSERT ON employees
   FOR EACH ROW
   BEGIN
      INSERT INTO employee_log (employee_id, action)
      VALUES (NEW.employee_id, 'INSERT');
   END;
END$$

DELIMITER ;
2 . DELIMITER $$

CREATE PROCEDURE CreateUpdateTimestampTrigger()
BEGIN
   CREATE TRIGGER UpdateEmployeeTimestamp
   BEFORE UPDATE ON employees
   FOR EACH ROW
   BEGIN
      SET NEW.last_modified = CURRENT_TIMESTAMP;
   END;
END$$

DELIMITER ;
```

Ans - 16

```
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;
/
```

```
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;
/
```

Ans - 17

```
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;
/
```

Ans - 18

```
 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;
/
```

Ans - 19

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;
/
```