DBMS Practical

1) Create the following table and perform the necessary commands given below:

(a) Create Table Student with following attributes:

1. Roll_No

2. Name

3. Date_of_Birth

4. Branch

5. Semester

6. Address

7. Year_of _Admission

```
-- (a) Create Table Student
CREATE TABLE Student (
   Roll_No INT PRIMARY KEY,
   Name VARCHAR(50),
   Date_of_Birth DATE,
   Branch VARCHAR(50),
   Semester INT,
   Address VARCHAR(100),
   Year_of_Admission INT
);
```

(b) Enter at least 10 records in the above table and answer the following queries using SQL:

```
-- (b) Insert at least 10 records
INSERT INTO Student (Roll_No, Name, Date_of_Birth, Branch, Semester, Address,
Year_of_Admission)
VALUES
   (1, 'John Doe', '2008-01-01', 'EXTC', 4, '123 Main St, City', 2014),
   (2, 'Jane Smith', '2008-01-01', 'EXTC', 3, '456 Elm St, Town', 2014),
   (3, 'Alice Johnson', '2000-05-15', 'CSE', 5, '789 Oak St, Village', 2013),
   (4, 'Bob Brown', '2001-03-20', 'IT', 6, '321 Pine St, County', 2012),
   (5, 'Emily Davis', '2005-11-10', 'CSE', 4, '654 Cedar St, Hamlet', 2016),
   (6, 'David Wilson', '2004-09-25', 'IT', 3, '987 Maple St, Valley', 2015),
   (7, 'Sophia Martinez', '2002-07-30', 'IT', 5, '135 Walnut St, Hill', 2014),
```

(8, 'Michael Anderson', '1999-12-05', 'CSE', 7, '246 Birch St, Forest', 2011),

(9, 'Olivia Taylor', '2003-08-18', 'IT', 2, '579 Spruce St, Canyon', 2017),

(10, 'William Thomas', '2000-04-03', 'CSE', 3, '357 Fir St, Ridge', 2015);

i. Find the name of all the students who are enrolled in EXTC branch and having date of birth as 01/01/2008

```
SELECT Name
FROM Student
WHERE Branch = 'EXTC' AND Date_of_Birth = '2008-01-01';
```

ii. List the name and roll number of all the students who are enrolled in year 2015.

```
SELECT Roll_No, Name
FROM Student
WHERE Year_of_Admission = 2015;
```

iii. List the name and address of all the students who are currently in fifth semester for Computer department

```
SELECT Name, Address
FROM Student
WHERE Semester = 5 AND Branch = 'CSE';
```

iv. Retrieve total number of students enrolled in IT department

```
SELECT COUNT(*)
FROM Student
WHERE Branch = 'IT';
```

2) Write a query in SQL to create a table employee and department.

Employee (empno, ename, deptno, job, hiredate) Department (deptno, dname, loc)

A. Include the following constraints on column of Employee table.

i) make empno as primary key of the table

ii) ename attribute does not contain NULL values

iii) job attribute allows only UPPERCASE entries

iv) put the current date as default date in hire date column in case data is not supplied for the column.

B. Include the following constraints on column of dept table.

i) make deptno as primary key.

ii) dname, loc attributes does not contain NULL values

iii) enforce REFERENTIAL INTEGRITY where deptno attribute of dept table as primary key and deptno attribute of emp table as foreign key.

iv) put default value of loc as “Mumbai”

```
-- Create Employee table
CREATE TABLE Employee (
   empno INT PRIMARY KEY,
   ename VARCHAR(50) NOT NULL,
   deptno INT,
   job VARCHAR(50) CHECK (job = UPPER(job)),
   hiredate DATE DEFAULT CURRENT_DATE
);
-- Create Department table
CREATE TABLE Department (
   deptno INT PRIMARY KEY,
   dname VARCHAR(50) NOT NULL,
   loc VARCHAR(50) DEFAULT 'Mumbai' NOT NULL
);
-- Add Foreign Key Constraint
ALTER TABLE Employee
ADD CONSTRAINT fk_deptno
FOREIGN KEY (deptno) REFERENCES Department(deptno);
```

Explanation:

A. Constraints on the Employee table:

i) `empno` is set as the primary key using `PRIMARY KEY` constraint.

ii) `ename` is set as `NOT NULL` to ensure it does not contain NULL values.

iii) `job` is enforced to be uppercase using the `CHECK` constraint.

iv) `hiredate` is given a default value of the current date using the `DEFAULT` constraint.

B. Constraints on the Department table:

i) `deptno` is set as the primary key using `PRIMARY KEY` constraint.

ii) Both `dname` and `loc` are set as `NOT NULL` to ensure they do not contain NULL values.

iii) Referential integrity is enforced by creating a foreign key constraint (`fk_deptno`) on the `deptno` column of the Employee table that references the `deptno` column of the Department table.

iv) `loc` is given a default value of "Mumbai" using the `DEFAULT` constraint.

3) Write a query in SQL to create a table employee and department with following attributes.

Employee (empno, ename, deptno, job, hiredate) Department (deptno, dname, loc)

```
-- Create Employee table
CREATE TABLE Employee (
   empno INT PRIMARY KEY,
   ename VARCHAR(50),
   deptno INT,
   job VARCHAR(50),
   hiredate DATE
);

-- Create Department table
CREATE TABLE Department (
   deptno INT PRIMARY KEY,
   dname VARCHAR(50),
   loc VARCHAR(50)
);
```

1. Give list of emp name & their job spec who are working in dept no 20?

```
SELECT ename, job
FROM Employee
WHERE deptno = 20;
```

2. Retrieve the details of emp working in dept no 30?

```
SELECT *
FROM Employee
WHERE deptno = 30;
```

3. Find list of emp whose empno is greater than manager no?

```
SELECT *
FROM Employee
WHERE empno > (SELECT MAX(empno) FROM Employee);
```

4. Find all manager not working in dept no 10?

```
SELECT *
```

```
FROM Employee
WHERE job = 'Manager' AND deptno != 10;
```

5. To find the total number of employees.

```
SELECT COUNT(*) AS total_employees
FROM Employee;
```

4) Write a query in sql to create a table employee and department.

Employee (empno, ename, deptno, job, hiredate) Department (deptno, dname, loc)

```
-- Create Employee table
CREATE TABLE Employee (
   empno INT PRIMARY KEY,
   ename VARCHAR(50),
   deptno INT,
   job VARCHAR(50),
   hiredate DATE
);
-- Create Department table
CREATE TABLE Department (
   deptno INT PRIMARY KEY,
   dname VARCHAR(50),
   loc VARCHAR(50)
);
```

1. To find the total number of clerks hired after 13-Jan-2001.

```
SELECT COUNT(*) AS total_clerks_hired_after_2001
FROM Employee
WHERE job = 'Clerk' AND hiredate > '2001-01-13';
```

2. Determine which department having more than two people holding a same job?

```
SELECT deptno, job, COUNT(*) AS job_count
FROM Employee
GROUP BY deptno, job
HAVING COUNT(*) > 2;
```

3. Find all departments that have at least two clerks?

```
SELECT deptno
```

```
FROM Employee
WHERE job = 'Clerk'
GROUP BY deptno
HAVING COUNT(*) >= 2;
```

4. Retrieve emp name and job who have the same job as that of „Allen‟?

```
SELECT ename, job
FROM Employee
WHERE job = (SELECT job FROM Employee WHERE ename = 'Allen');
```

5. List all emp name and their job of those department that are located at Chicago?

```
SELECT e.ename, e.job
FROM Employee e
JOIN Department d ON e.deptno = d.deptno
WHERE d.loc = 'Chicago';
```

5) Write a query in sql to create a table employee and department.

Employee (empno, ename, deptno, job, hiredate, salary) Department (deptno, dname, loc)

```
-- Create Employee table
CREATE TABLE Employee (
   empno INT PRIMARY KEY,
   ename VARCHAR(50),
   deptno INT,
   job VARCHAR(50),
   hiredate DATE,
   salary DECIMAL(10, 2)
);
-- Create Department table
CREATE TABLE Department (
   deptno INT PRIMARY KEY,
   dname VARCHAR(50),
   loc VARCHAR(50)
);
```

1. To get all employees working for dept 10 and 20.

```
SELECT *
FROM Employee
WHERE deptno IN (10, 20);
```

2. To list all employees whose name begins with „J“.

```
SELECT *
FROM Employee
WHERE ename LIKE 'J%';
```

3. Retrieve all details of employees whose name is either Smith, Blake, Allen, Scott, Clark and King?

```
SELECT *
FROM Employee
WHERE ename IN ('Smith', 'Blake', 'Allen', 'Scott', 'Clark', 'King');
```

4. Create view on appropriate tables to display ename , job , salary , dept name?

```
CREATE VIEW EmployeeDetails AS
SELECT e.ename, e.job, e.salary, d.dname AS dept_name
FROM Employee e
JOIN Department d ON e.deptno = d.deptno;
```

5. Drop the above view

```
DROP VIEW EmployeeDetails;
```

6) Write a query in sql to create a table employee and department.

Employee (empno, ename, deptno, job, hiredate, salary) Department (deptno, dname, loc)

```
-- Create Employee table
CREATE TABLE Employee (
   empno INT PRIMARY KEY,
   ename VARCHAR(50),
   deptno INT,
   job VARCHAR(50),
   hiredate DATE,
   salary DECIMAL(10, 2)
);
-- Create Department table
CREATE TABLE Department (
   deptno INT PRIMARY KEY,
   dname VARCHAR(50),
   loc VARCHAR(50)
);
```

1. To select the employees whose salary is greater than the salary of all employees working in dept no. 30

```
SELECT *
FROM Employee
WHERE salary > ALL (SELECT salary FROM Employee WHERE deptno = 30);
```

2. To list all employees in the ascending order by name.

```
SELECT *
FROM Employee
ORDER BY ename ASC;
```

3. To select all employees sorted dept wise in ascending order and within dept salary wise in descending order.

```
SELECT *
FROM Employee
ORDER BY deptno ASC, salary DESC;
```

4. To select all employees working in location whose name is starting with L

```
SELECT *
FROM Employee e
JOIN Department d ON e.deptno = d.deptno
WHERE d.loc LIKE 'L%';
```

5. To find the minimum salary of managers in various depts.

```
SELECT MIN(salary) AS min_manager_salary
FROM Employee
WHERE job = 'Manager'
GROUP BY deptno;
```

7) Database Schema:

Person (driver-id, name, address)
car (license, model, year)
accident (report-number, date location)
owns (driver-id, license)

participated (driver-id, car, report-number, damage, amount)

(i) Create relations persons owns in sql

```
CREATE TABLE Person (
   driver_id INT PRIMARY KEY,
   name VARCHAR(50),
   address VARCHAR(100)
);
CREATE TABLE Car (
   license VARCHAR(20) PRIMARY KEY,
   model VARCHAR(50),
   year INT
);
CREATE TABLE Accident (
   report_number INT PRIMARY KEY,
   date DATE,
   location VARCHAR(100)
);
CREATE TABLE Owns (
   driver_id INT,
   license VARCHAR(20),
   FOREIGN KEY (driver_id) REFERENCES Person(driver_id),
   FOREIGN KEY (license) REFERENCES Car(license),
   PRIMARY KEY (driver_id, license)
);
CREATE TABLE Participated (
   driver_id INT,
   license VARCHAR(20),
   report_number INT,
   damage VARCHAR(100),
   amount DECIMAL(10, 2),
   FOREIGN KEY (driver_id) REFERENCES Person(driver_id),
   FOREIGN KEY (license) REFERENCES Car(license),
   FOREIGN KEY (report_number) REFERENCES Accident(report_number),
   PRIMARY KEY (driver_id, license, report_number)
);
```

(ii) Add a new accident to the database, assume any values for required attribute.

```
INSERT INTO Accident (report_number, date, location) VALUES (12345, '2024-04-28', 'City Center');
```

(iii) Delete the SKODA belonging to 'Sachin Parker'.

```
DELETE FROM Owns
WHERE driver_id = (SELECT driver_id FROM Person WHERE name = 'Sachin Parker')
AND license IN (SELECT license FROM Car WHERE model = 'SKODA');
```

(iv) Find the total number of people who owned cars that were involved in accident in 1999.

```
SELECT COUNT(DISTINCT p.driver_id) AS total_owners
FROM Person p
JOIN Owns o ON p.driver_id = o.driver_id
JOIN Participated pa ON o.license = pa.license
JOIN Accident a ON pa.report_number = a.report_number
WHERE YEAR(a.date) = 1999;
```

(v) Find the person whose names starts with 'S' and arrange in decreasing order of driver-id.

```
SELECT *
FROM Person
WHERE name LIKE 'S%'
ORDER BY driver_id DESC;
```

8) Consider the employee database where the primary keys are underlined. Give an expression in SQL for the following queries:

employee (employee-name, street, city)
works (employee-name, company-name, salary)
company (company-name, city)
manages (empolyee-name, manager-name)

```
CREATE TABLE employee (
   employee_name VARCHAR(50) PRIMARY KEY,
   street VARCHAR(100),
   city VARCHAR(50)
);
CREATE TABLE works (
   employee_name VARCHAR(50),
   company_name VARCHAR(50),
   salary DECIMAL(10, 2),
   PRIMARY KEY (employee_name, company_name),
   FOREIGN KEY (employee_name) REFERENCES employee(employee_name)
);
CREATE TABLE company (
   company_name VARCHAR(50) PRIMARY KEY,
   city VARCHAR(50)
);
CREATE TABLE manages (
   employee_name VARCHAR(50),
   manager_name VARCHAR(50),
   PRIMARY KEY (employee_name),
   FOREIGN KEY (employee_name) REFERENCES employee(employee_name),
   FOREIGN KEY (manager_name) REFERENCES employee(employee_name)
);
```

(i) Find all employees in the database who earn more than each employee of Small Bank Corporation.

```
SELECT DISTINCT e.employee_name
FROM employee e
JOIN works w1 ON e.employee_name = w1.employee_name
WHERE NOT EXISTS (
   SELECT *
   FROM works w2
   WHERE w2.company_name = 'Small Bank Corporation'
   AND w2.salary >= w1.salary
);
```

(ii) Find all employees in the database who do not work for First Bank Corporation.

```
SELECT employee_name
FROM employee
WHERE employee_name NOT IN (
   SELECT w.employee_name
   FROM works w
   WHERE w.company_name = 'First Bank Corporation'
);
```

(iii) Find all employees who earn more than the average salary of all employees of their company.

```
SELECT e.employee_name
FROM employee e
JOIN works w ON e.employee_name = w.employee_name
WHERE w.salary > (
   SELECT AVG(w2.salary)
   FROM works w2
   WHERE w2.company_name = w.company_name
);
```

(iv) Find the names of all employees who work for First Bank Corporation.

```
SELECT e.employee_name
FROM employee e
JOIN works w ON e.employee_name = w.employee_name
WHERE w.company_name = 'First Bank Corporation';
```

9) Create table employee with following attributes and insert 10 records (Apply following query)

Employee (empno, ename, deptno, job, hiredate, salary) Department (deptno, dname, loc)

```
-- Create Employee table
CREATE TABLE Employee (
   empno INT PRIMARY KEY,
   ename VARCHAR(50),
   deptno INT,
   job VARCHAR(50),
   hiredate DATE,
    salary DECIMAL(10, 2)
);
-- Create Department table
CREATE TABLE Department (
    deptno INT PRIMARY KEY,
    dname VARCHAR(50),
    loc VARCHAR(50)
);
-- Insert 10 records into Employee table
INSERT INTO Employee (empno, ename, deptno, job, hiredate, salary)
VALUES
    (1, 'John', 10, 'Manager', '2023-01-15', 5000.00),
    (2, 'Jane', 20, 'Engineer', '2023-02-20', 4000.00),
    (3, 'Jack', 10, 'Clerk', '2023-03-25', 3000.00),
    (4, 'Jill', 20, 'Analyst', '2023-04-30', 3500.00),
    (5, 'James', 30, 'Manager', '2023-05-05', 4500.00),
    (6, 'Jessica', 30, 'Engineer', '2023-06-10', 4200.00),
    (7, 'Justin', 40, 'Analyst', '2023-07-15', 3800.00),
    (8, 'Jennifer', 40, 'Clerk', '2023-08-20', 3200.00),
    (9, 'Jordan', 10, 'Manager', '2023-09-25', 4800.00),
    (10, 'Julia', 20, 'Engineer', '2023-10-30', 3900.00);
```

1. To list employees whose name begins with „J" and has „N" as the third character?

```
SELECT *
FROM Employee
WHERE ename LIKE 'J_N%';
```

2. To list all employees not entitled for commission.

```
SELECT *
FROM Employee
WHERE job != 'Commission';
```

3. To get all the employees whose salary is greater than the average salary of the company.

```
SELECT *
FROM Employee
```

```
WHERE salary > (SELECT AVG(salary) FROM Employee);
```

4. To find out average minimum and maximum salary of each dept.

```
SELECT deptno, AVG(salary) AS avg_salary, MIN(salary) AS min_salary,
MAX(salary) AS max_salary
FROM Employee
GROUP BY deptno;
```

5. Create view on emp to display sum of salary grouped according to deptno

```
CREATE VIEW DeptSalarySum AS
SELECT deptno, SUM(salary) AS total_salary
FROM Employee
GROUP BY deptno;
```

10) Consider the following schema:

Suppliers (sid: integer, sname: string, address: string)

Parts (pid: integer, pname: string, color: string)

Catalog (sid: integer, pid: integer, cost: real)

The Catalog relation lists the prices charged for parts by Suppliers.

```
-- Create Suppliers table
CREATE TABLE Suppliers (
   sid INT PRIMARY KEY,
   sname VARCHAR(50),
   address VARCHAR(100)
);
-- Create Parts table
CREATE TABLE Parts (
   pid INT PRIMARY KEY,
   pname VARCHAR(50),
   color VARCHAR(20)
);
-- Create Catalog table
CREATE TABLE Catalog (
   sid INT,
   pid INT,
   cost REAL,
   PRIMARY KEY (sid, pid),
   FOREIGN KEY (sid) REFERENCES Suppliers(sid),
   FOREIGN KEY (pid) REFERENCES Parts(pid)
);
```

Write the following queries in SQL:

1. For each part, find the sname of the supplier who charges the most for that part.

```
SELECT Parts.pname, Suppliers.sname
FROM Parts
JOIN Catalog ON Parts.pid = Catalog.pid
JOIN Suppliers ON Catalog.sid = Suppliers.sid
WHERE Catalog.cost = (
  SELECT MAX(cost)
  FROM Catalog
  WHERE Catalog.pid = Parts.pid
);
```

2. Find the sids of suppliers who supply only red parts.

```
SELECT DISTINCT Catalog.sid
FROM Catalog
JOIN Parts ON Catalog.pid = Parts.pid
WHERE Parts.color = 'red'
AND NOT EXISTS (
  SELECT *
  FROM Catalog AS C2
  JOIN Parts AS P2 ON C2.pid = P2.pid
  WHERE C2.sid = Catalog.sid
  AND P2.color != 'red'
);
```

3. Find the sids of suppliers who supply a red part and a green part.

```
SELECT sid
FROM Catalog
WHERE pid IN (
  SELECT pid
  FROM Parts
```

```
    WHERE color IN ('red', 'green')
)
GROUP BY sid
HAVING COUNT(DISTINCT color) = 2;
```

4. Find the snames of suppliers who supply every red part.

```
SELECT sname
FROM Suppliers
WHERE NOT EXISTS (
    SELECT pid
    FROM Parts
    WHERE color = 'red'
    AND pid NOT IN (
        SELECT pid
        FROM Catalog
        WHERE sid = Suppliers.sid
    )
);
```

5. Find the pnames of parts supplied by Acme Widget Suppliers and no one else.

```
SELECT pname
FROM Parts
WHERE pid IN (
    SELECT pid
    FROM Catalog
    WHERE sid = (
        SELECT sid
        FROM Suppliers
        WHERE sname = 'Acme Widget Suppliers'
    )
)
AND NOT EXISTS (
    SELECT pid
    FROM Catalog
    WHERE pid = Parts.pid
    AND sid != (
        SELECT sid
        FROM Suppliers
        WHERE sname = 'Acme Widget Suppliers'
    )
);
```

11) Create a table customer (acc_no, cust_name, avail_balance)

Create table mini_statement (acc_no, avail_balance)

Insert into customer following records:

Customer (1000, “Fanny”, 7000);

Customer (1001, ”Peter”, 12000);

Write a trigger to insert old values into mini_statement table (including acc_no, avail_balance as parameters) before updating any record in customer table.

```
-- Create Customer table
CREATE TABLE customer (
   acc_no INT PRIMARY KEY,
   cust_name VARCHAR(50),
   avail_balance DECIMAL(10, 2)
);
-- Create Mini_Statement table
CREATE TABLE mini_statement (
   acc_no INT,
   avail_balance DECIMAL(10, 2),
   change_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
-- Insert records into customer table
INSERT INTO customer (acc_no, cust_name, avail_balance) VALUES
(1000, 'Fanny', 7000),
(1001, 'Peter', 12000);
-- Create Trigger
DELIMITER //
CREATE TRIGGER before_customer_update
BEFORE UPDATE ON customer
```

```
FOR EACH ROW
BEGIN
    INSERT INTO mini_statement (acc_no, avail_balance)
    VALUES (OLD.acc_no, OLD.avail_balance);
END;
//
DELIMITER ;
```

12) Create a table customer (acc_no, cust_name, avail_balance)

Create table micro_statement (acc_no, avail_balance)

Insert following record in table customer:

Customer (1000, "Fanny", 7000);

Customer (1001, ”Peter”, 12000);

Customer (1002, ”Janitor”, 4500)

Write a trigger to insert new values of acc_no and avail_balance in micro_statement after an update has occurred.

```
-- Create Customer table
CREATE TABLE customer (
    acc_no INT PRIMARY KEY,
    cust_name VARCHAR(50),
    avail_balance DECIMAL(10, 2)
);
-- Create Micro_Statement table
CREATE TABLE micro_statement (
    acc_no INT,
    avail_balance DECIMAL(10, 2),
    change_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
-- Insert records into customer table
INSERT INTO customer (acc_no, cust_name, avail_balance) VALUES
(1000, 'Fanny', 7000),
(1001, 'Peter', 12000),
(1002, 'Janitor', 4500);
-- Create Trigger
```

```
DELIMITER //
CREATE TRIGGER after_customer_update
AFTER UPDATE ON customer
FOR EACH ROW
BEGIN
   INSERT INTO micro_statement (acc_no, avail_balance)
   VALUES (NEW.acc_no, NEW.avail_balance);
END;
//
DELIMITER ;
```

13) Create a table customer (cust_id, cust_name, balance) and insert 3 records to it Write a transaction which update the balance of all 3 customers and using TCL Commands (Commit, Rollback and Savepoint) show the changes made to actual records.

--lets create customer table and insert 3 records:

```
CREATE TABLE customer (
   cust_id INT PRIMARY KEY,
   cust_name VARCHAR(50),
   balance DECIMAL(10, 2)
);
INSERT INTO customer (cust_id, cust_name, balance) VALUES
(1, 'Alice', 1000.00),
(2, 'Bob', 1500.00),
(3, 'Charlie', 2000.00);
```

--let's perform a transaction to update the balance of all 3 customers and demonstrate the use of TCL commands (COMMIT, ROLLBACK, SAVEPOINT):

```
-- Start the transaction
START TRANSACTION;
-- Update the balance for all 3 customers
```

```
UPDATE customer SET balance = balance + 500 WHERE cust_id IN (1, 2, 3);
-- Show the updated records before committing
SELECT * FROM customer;
-- Savepoint to mark a point in the transaction
SAVEPOINT before_update;
-- Attempt to update the balance for all 3 customers again
UPDATE customer SET balance = balance + 300 WHERE cust_id IN (1, 2, 3);
-- Show the updated records after the second update (before rollback)
SELECT * FROM customer;
-- Rollback to the savepoint to undo the second update
ROLLBACK TO before_update;
-- Show the records after rollback
SELECT * FROM customer;
-- Commit the transaction to make the changes permanent
COMMIT;
```

14) Create a table student (student_id, stud_name, percentage) and insert 3 records to it Write a transaction which update the percentage of all 3 students and using TCL Commands (Commit, Rollback and Savepoint) show the changes made to actual records.

```
-- Create the student table
CREATE TABLE student (
   student_id INT PRIMARY KEY,
   stud_name VARCHAR(50),
   percentage DECIMAL(5, 2)
);
-- Insert 3 records into the student table
INSERT INTO student (student_id, stud_name, percentage) VALUES
(1, 'Alice', 80.00),
(2, 'Bob', 75.50),
(3, 'Charlie', 90.25);

-- Start the transaction
START TRANSACTION;
```

```
-- Update the percentage for all 3 students
UPDATE student SET percentage = percentage + 5;
-- Show the updated records before committing
SELECT * FROM student;
-- Savepoint to mark a point in the transaction
SAVEPOINT before_update;
-- Attempt to update the percentage for all 3 students again
UPDATE student SET percentage = percentage + 3;
-- Show the updated records after the second update (before rollback)
SELECT * FROM student;
-- Rollback to the savepoint to undo the second update
ROLLBACK TO before_update;
-- Show the records after rollback
SELECT * FROM student;
-- Commit the transaction to make the changes permanent
COMMIT;
```

15) Consider the relational database. Write an expression in SQL for following schema

Employee (employee-name, street, city)

works (employee-name, company-name, salary)

company (company-name, city)

manages (employee-name,manager-name)

```
-- Create Employee table
CREATE TABLE Employee (
   employee_name VARCHAR(50),
   street VARCHAR(100),
   city VARCHAR(50),
   PRIMARY KEY (employee_name)
);
-- Create works table
CREATE TABLE works (
   employee_name VARCHAR(50),
   company_name VARCHAR(50),
   salary DECIMAL(10, 2),
    FOREIGN KEY (employee_name) REFERENCES Employee(employee_name)
);
-- Create company table
CREATE TABLE company (
    company_name VARCHAR(50) PRIMARY KEY,
    city VARCHAR(50)
);
-- Create manages table
CREATE TABLE manages (
    employee_name VARCHAR(50),
    manager_name VARCHAR(50),
    FOREIGN KEY (employee_name) REFERENCES Employee(employee_name),
    FOREIGN KEY (manager_name) REFERENCES Employee(employee_name)
);
```

Write following SQL queries

i) Retrieve details of all employees working for “Infosys” company

```
SELECT e.employee_name, e.street, e.city
FROM Employee e
JOIN works w ON e.employee_name = w.employee_name
WHERE w.company_name = 'Infosys';
```

ii) Retrieve employee-name in uppercase for all employees

```
SELECT UPPER(employee_name) AS employee_name_uppercase
FROM Employee;
```

iii) Replace existing company name of employees from Infosys to TCS

```
UPDATE works
SET company_name = 'TCS'
WHERE company_name = 'Infosys';
```

iv) Retrieve manager name along with employee name working for “TCS” company

```
SELECT e.employee_name, m.manager_name
FROM Employee e
JOIN manages m ON e.employee_name = m.employee_name
JOIN works w ON e.employee_name = w.employee_name
WHERE w.company_name = 'TCS';
```

v) Retrieve details of all employees whose ename starts with “P”

```
SELECT *
FROM Employee
WHERE employee_name LIKE 'P%';
```

16) Consider an online bookstore database with the following tables:

** books: Contains information about books such as book_id, title, author_id, genre_id, and price.**

** authors: Contains information about authors such as author_id, author_name, and country.**

** genres: Contains information about book genres such as genre_id and genre_name.**

** customers: Contains information about customers such as customer_id, name, email, and city.**

** orders: Contains information about orders such as order_id, customer_id, order_date, and total_amount.**

** order_details: Contains information about the details of each order such as order_detail_id, order_id, book_id, quantity, and subtotal.**

```
-- Create authors table
CREATE TABLE authors (
   author_id INT PRIMARY KEY,
   author_name VARCHAR(100),
   country VARCHAR(100)
);
-- Create genres table
CREATE TABLE genres (
   genre_id INT PRIMARY KEY,
   genre_name VARCHAR(100)
);
-- Create books table
CREATE TABLE books (
   book_id INT PRIMARY KEY,
   title VARCHAR(255),
   author_id INT,
   genre_id INT,
    price DECIMAL(10, 2),
    FOREIGN KEY (author_id) REFERENCES authors(author_id),
    FOREIGN KEY (genre_id) REFERENCES genres(genre_id)
);
-- Create customers table
CREATE TABLE customers (
    customer_id INT PRIMARY KEY,
    name VARCHAR(100),
    email VARCHAR(255),
    city VARCHAR(100)
);
-- Create orders table
CREATE TABLE orders (
    order_id INT PRIMARY KEY,
    customer_id INT,
    order_date DATE,
    total_amount DECIMAL(10, 2),
    FOREIGN KEY (customer_id) REFERENCES customers(customer_id)
);
-- Create order_details table
CREATE TABLE order_details (
    order_detail_id INT PRIMARY KEY,
    order_id INT,
    book_id INT,
    quantity INT,
    subtotal DECIMAL(10, 2),
    FOREIGN KEY (order_id) REFERENCES orders(order_id),
    FOREIGN KEY (book_id) REFERENCES books(book_id)
);
```

1. Get the list of books with their authors and genres

```
SELECT b.title AS book_title, a.author_name, g.genre_name
FROM books b
JOIN authors a ON b.author_id = a.author_id
JOIN genres g ON b.genre_id = g.genre_id;
```

2. Get the total amount spent by each customer:

```sql
SELECT c.name AS customer_name, SUM(o.total_amount) AS total_spent
FROM customers c
JOIN orders o ON c.customer_id = o.customer_id
GROUP BY c.name;
```

3. Get the list of customers along with the titles of books they have ordered

```sql
SELECT c.name AS customer_name, b.title AS book_title
FROM customers c
JOIN orders o ON c.customer_id = o.customer_id
JOIN order_details od ON o.order_id = od.order_id
JOIN books b ON od.book_id = b.book_id;
```

4. Get the top-selling authors (authors with the highest total number of book sales)

```sql
SELECT a.author_name, COUNT(od.order_detail_id) AS total_sales
FROM authors a
JOIN books b ON a.author_id = b.author_id
JOIN order_details od ON b.book_id = od.book_id
GROUP BY a.author_name
ORDER BY total_sales DESC;
```

17) We have a database for an online bookstore with relations books and customer, and we want to grant specific privileges to different users.

□ Create a new user named 'bookstore_manager' with a password.

□ Grant the SELECT privilege on the 'books' table to the 'bookstore_manager' user, allowing them to retrieve data from the 'books' table.

□ Grant the INSERT, UPDATE, and DELETE privileges on the 'customers' table to the 'bookstore_manager' user, allowing them to insert, update, and delete records in the 'customers' table.

1. Create a new user named 'bookstore_manager' with a password:

```sql
CREATE USER 'bookstore_manager'@'localhost' IDENTIFIED BY 'your_password';
```

Replace `'your_password'` with the desired password for the user.

2. Grant the SELECT privilege on the 'books' table to the 'bookstore_manager' user:

```sql
GRANT SELECT ON books TO 'bookstore_manager'@'localhost';
```

3. Grant the INSERT, UPDATE, and DELETE privileges on the 'customers' table to the 'bookstore_manager' user:

```sql
GRANT INSERT, UPDATE, DELETE ON customers TO 'bookstore_manager'@'localhost';
```

These SQL statements will create a new user named 'bookstore_manager', assign a password, and grant specific privileges to this user for accessing the 'books' and 'customers' tables in the online bookstore database. Make sure to replace `'your_password'` with an actual password.

18) Let's consider a scenario where a customer places an order on our online bookstore. We want to ensure that the order process is treated as a single transaction. Apply TCL commands using following steps

 We begin a transaction using the BEGIN TRANSACTION command.

 We insert the order details (such as customer ID, order date, and total amount) into the 'orders' table.

 We insert the individual items of the order (book ID, quantity, and subtotal) into the 'order_details' table.

 We perform a check to ensure that the total amount matches the sum of individual subtotals. If the validation fails, we rollback the transaction using the ROLLBACK command.

 If the validation succeeds, we commit the transaction using the COMMIT command.

```
-- Begin the transaction
BEGIN TRANSACTION;

-- Step 1: Insert the order details into the 'orders' table
INSERT INTO orders (customer_id, order_date, total_amount)
VALUES (123, '2024-04-18', 150.00);

-- Step 2: Insert the individual items of the order into the 'order_details' table
```

```
INSERT INTO order_details (order_id, book_id, quantity, subtotal)
VALUES
(1, 101, 2, 50.00),
(1, 102, 1, 30.00),
(1, 103, 3, 70.00);

-- Step 3: Perform validation to ensure total amount matches the sum of individual subtotals
DECLARE @total DECIMAL(10, 2);
DECLARE @subtotal DECIMAL(10, 2);

SELECT @total = total_amount FROM orders WHERE order_id = 1;
SELECT @subtotal = SUM(subtotal) FROM order_details WHERE order_id = 1;

IF (@total <> @subtotal) BEGIN
    -- Rollback the transaction if validation fails
    ROLLBACK;
    PRINT 'Validation failed! Rolling back transaction.';
END
ELSE BEGIN
    -- Commit the transaction if validation succeeds
    COMMIT;
    PRINT 'Validation succeeded! Transaction committed.';
END;
```