Q no	QUESTIONS
1.	Sailors database

SAILORS (sid, sname, rating, age)
BOAT(bid, bname, color)
RSERVERS (sid, bid, date)
Queries, View and Trigger

1. Find the colours of boats reserved by Albert
2. Find all sailor id's of sailors who have a rating of at least 8 or reserved boat 103
3. Find the names of sailors who have not reserved a boat whose name contains the string "storm". Order the names in ascending order.
4. Find the names of sailors who have reserved all boats.
5. Find the name and age of the oldest sailor.
6. For each boat which was reserved by at least 5 sailors with age >= 40, find the boat id and the average age of such sailors.
7. Create a view that shows the names and colours of all the boats that have been reserved by a sailor with a specific rating.
8. A trigger that prevents boats from being deleted If they have active reservations.

```
Create database SAILORS;

Use SAILORS;

create table sailors(
  sid int not null,
  primary key(sid),
  sname varchar(25),
  rating int not null,
  age int not null
  );

 create table boat(
   bid int not null,
   primary key(bid),
   bname varchar(30),
   color varchar(15)
   );

 create table rservers(
   sid int not null,
   bid int not null,
   foreign key(sid) references sailors(sid) on delete cascade,
   foreign key(bid) references boat(bid) on delete cascade,
   dte date
   );

insert into sailors values
(1,"john",9,65),
(2,"albert",8,57),
(3,"harrystormr",6,45),
(4,"james",8,41),
(5,"peterstorm",9,47),
(6,"kevin",8.5,43);

insert into boat values
(101,"b101","blue"),
(102,"b102","red"),
(103,"b103","yellow"),
(104,"b104","orange"),
(105,"b105","green");

insert into rservers values
(1,101,"2003-01-01"),
(1,102,"2003-06-09"),
(1,103,"2003-01-01"),
(1,104,"2003-01-01-"),
(1,105,"2003-05-23"),
(2,101,"2006-01-01"),
(2,104,"2018-09-08"),
(1,102,"2003-01-01"),
(6,101,"2008-01-01"),
(4,105,"2019-09-08"),
(4,101,"2003-01-01"),
(5,101,"2003-01-09");

select * from sailors;
select * from boat;
select * from rservers;

select b.color
from boat as b,sailors as s,rservers as r
where s.sid = r.sid and b.bid = r.bid and s.sname = "albert";
```

Find the colors of boats reserved by 'Albert'.

```
(SELECT DISTINCT s.sid
FROM sailors s
JOIN rservers r ON s.sid = r.sid
WHERE s.rating >= 8)
UNION
(SELECT DISTINCT s.sid
FROM sailors s
JOIN rservers r ON s.sid = r.sid
JOIN boat b ON b.bid = r.bid
WHERE r.bid = 103);
```

Find all sailors ids of sailors who have a rating of atleast 8 or reserved the boat 103

```
SELECT DISTINCT s.sname
FROM sailors s
LEFT JOIN rservers r ON s.sid = r.sid
WHERE r.sid IS NULL AND s.sname LIKE "%storm%"
ORDER BY s.sname ASC;
```

Find the names of sailors who have not reserved a boat whose name contains the string 'strom'. Order the names in ascending order.

```
SELECT s.sname
FROM sailors s
WHERE NOT EXISTS (
    SELECT *
    FROM boat b
    WHERE NOT EXISTS (
        SELECT *
        FROM rservers r
        WHERE r.sid = s.sid AND r.bid = b.bid
    )
);
```

Find the names of sailors who have reserved all boats

```
SELECT s.sname, MAX(s.age)
FROM sailors s;
```

Find the name and age of oldest sailor

```
SELECT r.bid, AVG(s.age) AS avg_age
FROM rservers r
JOIN sailors s ON r.sid = s.sid
WHERE r.bid IN (
    SELECT r2.bid
    FROM rservers r2
    JOIN sailors s2 ON r2.sid = s2.sid
    WHERE s2.age >= 40
    GROUP BY r2.bid
    HAVING COUNT(DISTINCT r2.sid) >= 5
)
GROUP BY r.bid;
```

For each boat that was reserved by atleast 5 sailors with age >=40, find the boat id and average age of such sailors.

```
CREATE VIEW ReservedBoats AS
SELECT b.bname, b.color, s.rating
FROM boat b
JOIN rservers r ON b.bid = r.bid
JOIN sailors s ON r.sid = s.sid;
```

Create a view that shows the names and colours of all the boats that have been reserved by a sailor with a specific rating.

```
DELIMITER //

CREATE TRIGGER prevent_delete
BEFORE DELETE ON boat
FOR EACH ROW
```

```
IN
   DECLARE reservation_count INT;
   SELECT COUNT(*) INTO reservation_count
   FROM rservers
   WHERE bid = OLD.bid;

   IF reservation_count > 0 THEN
       SIGNAL SQLSTATE '45000'
       SET MESSAGE_TEXT = 'Cannot delete. Active reservations exist.';
   END IF;
END//

DELIMITER ;
```

A trigger that prevents boats from being deleted if they have active reservations.

Insurance database
PERSON (driver id#: string, name: string, address: string)
CAR (regno: string, model: string, year: int)
ACCIDENT (report_ number: int, acc_date: date, location: string)
OWNS (driver id#: string, regno: string)
PARTICIPATED(driver id#:string, regno:string, report_ number: int,damage_amount: int)

1. Find the total number of people who owned cars that were involved in accidents in 2021.
2. Find the number of accidents in which the cars belonging to "Smith" were involved.
3. Add a new accident to the database; assume any values for required attributes.
4. Delete the Mazda belonging to "Smith".
5. Update the damage amount for the car with license number "KA09MA1234" in the accident with report.
6. A view that shows models and year of cars that are involved in accident.
7. A trigger that prevents a driver from participating in more than 3 accidents in a given year.

```
create table PERSON(
    driver_id varchar(10),
    name varchar(25),
    address varchar(50),
    primary key(driver_id)
);
create table CAR(
    regno varchar(10),
    primary key(regno),
    model varchar(10),
    year int
);
create table ACCIDENT(
    report_number int not null,
    primary key(report_number),
    acc_date date,
    location varchar(50)
);
create table OWNS(
    driver_id varchar(10),
    foreign key(driver_id) references PERSON(driver_id) on delete cascade,
    regno varchar(10),
    foreign key(regno) references CAR(regno) on delete cascade
);
create table PARTICIPATED(
    driver_id varchar(10),
    foreign key(driver_id) references PERSON(driver_id) on delete cascade,
    regno varchar(10),
    foreign key(regno) references CAR(regno) on delete cascade,
    report_number int not null,
    foreign key(report_number) references ACCIDENT(report_number) on delete cascade,
    damage_amt int not null
);

insert into PERSON values
("d1","john","vijayanagar"),
("d2","smith","jp nagar"),
("d3","jane","kuvempunagar"),
("d4","kevin","jayalakshmipuram"),
("d5","rock","lakshmipuram");

insert into CAR values
("KA09MA1234","swift",2000),
("KA09SF5634","mazda",2015),
("KA09HA7239","toyota",2009),
("KA09RG4367","mazda",2019),
("KA09RK1889","honda",2021);

insert into ACCIDENT values
(12,"2007-09-09","vijayanagar"),
(14,"2007-09-03","brigade rd"),
(32,"2018-09-06","kuvempunagar"),
(23,"2021-11-09","lakshmipuram"),
(65,"2021-09-25","jayalakshmipuram");

insert into OWNS values
("d1","KA09SF5634"),
("d2","KA09RG4367"),
("d3","KA09MA1234"),
("d4","KA09RK1889"),
("d5","KA09HA7239");

insert into PARTICIPATED values
("d2","KA09RG4367",12,25000),
("d2","KA09RG4367",14,67000),
("d3","KA09MA1234",32,12000),
("d4","KA09RK1889",23,200000),
("d5","KA09HA7239",65,3400);

SELECT COUNT(DISTINCT P.driver_id) AS total_people_involved
FROM PERSON P
JOIN OWNS O ON P.driver_id = O.driver_id
JOIN PARTICIPATED PA ON O.regno = PA.regno
JOIN ACCIDENT A ON PA.report_number = A.report_number
WHERE YEAR(A.acc_date) = 2021;
```

Find the total number of people who owned cars that were involved in accidents in 2021.

```
SELECT COUNT(*) AS accidents_involving_smith
FROM PERSON P
JOIN OWNS O ON P.driver_id = O.driver_id
JOIN PARTICIPATED PA ON O.regno = PA.regno
WHERE P.name = 'smith';
```

Find the no of accidents car belonging to smith were involved

```
INSERT INTO ACCIDENT VALUES
(78, '2022-01-01', 'ABC rd');
SELECT * FROM ACCIDENT;
```

Add a new accident to the database, assume any values for required attributes.

```
DELETE FROM OWNS
WHERE driver_id = (SELECT driver_id
                   FROM PERSON
                   WHERE name = 'Smith')
  AND
  regno IN (SELECT regno
            FROM CAR
            WHERE model = 'Mazda');
SELECT * FROM OWNS;
```

Delete the 'Mazda' belonging to smith were involved.

```
UPDATE PARTICIPATED
SET damage_amt = 170000
WHERE regno = 'KA09MA1234';
--SELECT * FROM PARTICIPATED;
```

Update the damage amount for the car with license no "KA09MA1234" in the accident with report.

```
CREATE VIEW AccidentView AS
SELECT DISTINCT C.model, C.year
FROM CAR C
JOIN PARTICIPATED PA ON C.regno = PA.regno;
SELECT * FROM AccidentView;
```

A view that shows models and year of cars that are involved in accident

```
DELIMITER//
```

```
CREATE TRIGGER PreventExcessiveAccidents
BEFORE INSERT ON PARTICIPATED
FOR EACH ROW
BEGIN
  DECLARE accident_count INT;

  SELECT COUNT(*)
  INTO accident_count
  FROM PARTICIPATED
  WHERE driver_id# = NEW.driver_id#
    AND YEAR((SELECT acc_date FROM ACCIDENT WHERE report_number = NEW.report_number)) = YEAR(NOW(
));

  IF accident_count >= 3 THEN
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'Driver cannot participate in more than 3 accidents in a year';
  END IF;
END//

DELIMITER;
```

A trigger that prevents driver from participating in more than 3 accidents in a given year.

3. Order processing database

```
Customer (Cust#:int, cname: string, city: string)
Order (order#:int, odate: date, cust#: int, order-amt: int)
Order-item (order#:int, Item#: int, qty: int)
Item (item#:int, unitprice: int)
Shipment (order#:int, warehouse#: int, ship-date: date)
Warehouse (warehouse#:int, city: string)
```

1. List the Order# and Ship_date for all orders shipped from Warehouse# "W2".
2. List the Warehouse information from which the Customer named "Kumar" was supplied his orders. Produce a listing of Order#, Warehouse#.
3. Produce a listing: Cname, #ofOrders, Avg_Order_Amt, where the middle column is the total number of orders by the customer and the last column is the average order amount for that customer. (Use aggregate functions)
4. Delete all orders for customer named "Kumar".
5. Find the item with the maximum unit price.
6. A trigger that updates order_amout based on quantity and unitprice of order_item
7. Create a view to display orderID and shipment date of all orders shipped from a warehouse 5.

```
CREATE TABLE Customer (
  cust_no INT PRIMARY KEY,
  cname VARCHAR(255),
  city VARCHAR(255)
);

CREATE TABLE Orders (
  order_no INT PRIMARY KEY,
  odate DATE,
  cust_no INT,
  order_amt INT,
  FOREIGN KEY (cust_no) REFERENCES Customer(cust_no)  ON DELETE CASCADE

);

CREATE TABLE Item (
  item INT PRIMARY KEY,
  unitprice INT
);

CREATE TABLE Order_item (
  order_no INT,
  Item INT,
  qty INT,
  PRIMARY KEY (order_no, Item),
  FOREIGN KEY (order_no) REFERENCES Orders(order_no)  ON DELETE CASCADE,
  FOREIGN KEY (Item) REFERENCES Item(item)  ON DELETE CASCADE
);

CREATE TABLE Warehouse (
  warehouse_no INT PRIMARY KEY,
  city VARCHAR(255)
);

CREATE TABLE Shipment (
  order_no INT PRIMARY KEY,
  warehouse_no INT,
  ship_date DATE,
  FOREIGN KEY (order_no) REFERENCES Orders(order_no)  ON DELETE CASCADE,
  FOREIGN KEY (warehouse_no) REFERENCES Warehouse(warehouse_no)  ON DELETE CASCADE

);

INSERT INTO Customer VALUES
 (1, 'John Doe', 'New York'),
 (2, 'Jane Smith', 'Los Angeles'),
 (3, 'kumar', 'Chicago'),
 (4, 'Alice Brown', 'Houston'),
 (5, 'Charlie White', 'San Francisco');

INSERT INTO Orders VALUES
(101, '2023-01-15', 1, 500),
(102, '2023-02-02', 2, 750),
(103, '2023-03-10', 3, 300),
(104, '2023-04-05', 4, 900),
(105, '2023-05-12', 5, 600);


INSERT INTO Item VALUES
(1, 50),
(2, 30),
(3, 25),
(4, 15),
(5, 40);

INSERT INTO Order_item VALUES
(101, 1, 2),
(101, 2, 3),
(102, 3, 1),
(103, 4, 5),
(104, 5, 2);

INSERT INTO Warehouse VALUES
(1, 'C1'),
(2, 'C2'),
(3, 'C3'),
(4, 'C4'),
(5, 'C5');

INSERT INTO Shipment VALUES
(101, 1, '2023-01-20'),
(102, 2, '2023-02-25'),
(103, 3, '2023-03-15'),
(104, 4, '2023-04-10'),
(105, 5, '2023-05-17');

SELECT o.order_no, s.ship_date
FROM Orders o
JOIN Shipment s ON o.order_no = s.order_no
WHERE s.warehouse_no = 2;
```

} List the order and shipdate for all ~~orderds~~ orders shiped from warehouse "W2"

```
SELECT o.order_no, w.*
FROM Customer c
JOIN Orders o ON c.cust_no = o.cust_no
```

} List the Warehouse info from ~~the~~ which customer named "Kumar" was

```
JOIN Shipment s ON o.order_no = s.order_no
JOIN Warehouse w ON s.warehouse_no = w.warehouse_no
WHERE c.cname = "kumar";
```
} supplied his orders
Produce a listing of order#, Warehouse#

```
SELECT c.cname, COUNT(o.order_no) AS no_of_orders, AVG(o.order_amt) AS avg_order_amt
FROM Customer c
LEFT JOIN Orders o ON c.cust_no = o.cust_no
GROUP BY c.cname;
```
} produce a listing

```
DELETE FROM Orders
WHERE cust_no = (SELECT cust_no
                 FROM Customer
                 WHERE cname = 'Kumar');
```
} Delete all orders for customer named Kumar

```
SELECT item,MAX(unitprice)
FROM Item;
```
} Find the item with maximum unit price

```
DELIMITER //
create trigger UpdateOrderAmt
after insert on Order_item
for each row
BEGIN
       update Orders set order_amt=(new.qty*(select distinct unitprice
                                              from Item
                                              NATURAL JOIN Order_item where item=new.item))
where Orders.order_no=new.order_no;
END; //
DELIMITER ;
```
A trigger that updates order amount based on quantity and unitprice of order item

```
CREATE VIEW Orders_Warehouse_5 AS
SELECT o.order_no, s.ship_date
FROM Orders o
JOIN Shipment s ON o.order_no = s.order_no
WHERE s.warehouse_no = 5;
SELECT * FROM Orders_Warehouse_5;
```
} Create a view to display order ID and shipment date of all orders shipped from Warehouse 5.

4. Student enrollment in courses and books adopted for each course

STUDENT (regno: string, name: string, major: string, bdate: date)
COURSE (course#:int, cname: string, dept: string)
ENROLL(regno:string, course#: int,sem: int,marks: int)
BOOK-ADOPTION (course#:int, sem: int, book-ISBN: int)
TEXT (book-ISBN: int, book-title: string, publisher: string,author: string)

1. Demonstrate how you add a new text book to the database and make this book be adopted by some department.
2. Produce a list of text books (include Course #, Book-ISBN, Book-title) in the alphabetical order for courses offered by the 'CS' department that use more than two books.
3. List any department that has all its adopted books published by a specific publisher.
4. List the students who have scored maximum marks in 'DBMS' course.
5. Create a view to display all the courses opted by a student along with marks obtained.
6. Create a trigger that prevents a student from enrolling in a course if the marks prerequisite is less than 40.

```
CREATE TABLE STUDENT (
    regno VARCHAR(50) PRIMARY KEY,
    name VARCHAR(255),
    major VARCHAR(255),
    bdate DATE
);

CREATE TABLE COURSE (
    course_no INT PRIMARY KEY,
    cname VARCHAR(255),
    dept VARCHAR(255)
);

CREATE TABLE ENROLL (
    regno VARCHAR(50),
    course_no INT,
    sem INT,
    marks INT,
    FOREIGN KEY (regno) REFERENCES STUDENT(regno)ON DELETE CASCADE,
    FOREIGN KEY (course_no) REFERENCES COURSE(course_no) ON DELETE CASCADE
);

CREATE TABLE TEXTBOOK (
    book_ISBN INT PRIMARY KEY,
    book_title VARCHAR(255),
    publisher VARCHAR(255),
    author VARCHAR(255)
);

CREATE TABLE BOOK_ADOPTION (
    course_no INT,
    sem INT,
    book_ISBN INT,
    FOREIGN KEY (course_no) REFERENCES COURSE(course_no) ON DELETE CASCADE,
    FOREIGN KEY (book_ISBN) REFERENCES TEXTBOOK(book_ISBN) ON DELETE CASCADE
);

INSERT INTO STUDENT VALUES
('S001', 'John Doe', 'Computer Science', '1990-01-01'),
('S002', 'Jane Smith', 'Electrical Engineering', '1992-05-15'),
('S003', 'Bob Johnson', 'Mechanical Engineering', '1991-08-20'),
('S004', 'Alice Brown', 'Computer Science', '1993-03-10'),
('S005', 'Charlie Davis', 'Physics', '1992-11-25');

INSERT INTO COURSE VALUES
(1, 'Database Management Systems', 'CS'),
(2, 'Computer Networks', 'CS'),
(3, 'Introduction to Physics', 'Physics'),
(4, 'Mechanics of Materials', 'Mechanical Engineering'),
(5, 'Digital Signal Processing', 'Electrical Engineering');

INSERT INTO ENROLL VALUES
('S001', 1, 1, 85),
('S002', 1, 1, 72),
('S003', 2, 1, 68),
('S004', 1, 1, 90),
('S005', 3, 1, 75);

INSERT INTO TEXTBOOK VALUES
(123456789, 'Database Management Systems', 'Pearson', 'Author1'),
(234567890, 'Computer Networks', 'McGraw Hill', 'Author2'),
(345678901, 'Introduction to Physics', 'Wiley', 'Author3'),
(456789012, 'Mechanics of Materials', 'Springer', 'Author4'),
(567890123, 'Digital Signal Processing', 'Pearson', 'Author5');

INSERT INTO BOOK_ADOPTION VALUES
(1, 1, 123456789),
(1, 2, 234567890),
(3, 8, 345678901),
(4, 4, 123456789),
(5, 3, 234567890);

INSERT INTO TEXTBOOK VALUES
(123456787, 'XYZ', 'ABC', 'PQRST');

INSERT INTO BOOK_ADOPTION VALUES
(1, 1, 123456787);

select c.course_no,t.book_ISBN,t.book_title
from course c
```

```
join book_adoption b on b.course_no = c.course_no
join textbook t on t.book_ISBN = b.book_ISBN
where c.dept = "CS"
and 2<( select count(book_ISBN)
        from book_adoption ba
        where c.course_no = ba.course_no)
order by t.book_title asc;
```

Demonstrate how you add a new text book to the database and make this book be adopted by some department.

```
SELECT C.dept
FROM COURSE C
WHERE NOT EXISTS (
    SELECT BA.course_no
    FROM BOOK_ADOPTION BA
    JOIN TEXTBOOK TB ON BA.book_ISBN = TB.book_ISBN
    WHERE BA.course_no = C.course_no AND TB.publisher != 'Wiley'
);
```

produce a list of textbook (include course#, Book-ISBN, Book title) in the alphabetical order for courses offered by the 'CS' department that use more than 2 books.

```
select s.*
from student s
join enroll e on s.regno = e.regno
join course c on c.course_no = e.course_no
where c.cname = "Database Management Systems"
and e.marks in ( select max(marks) from enroll );
```

list the students who have scored marks in 'DBMS' course

```
create view StudentCourses
as select c.cname , s.name ,e.marks
from student s
join enroll e on s.regno = e.regno
join course c on c.course_no = e.course_no
```

Create a view to display all the courses opted by ~~the~~ student along with marks obtained.

```
DELIMITER //

CREATE TRIGGER BeforeEnroll
BEFORE INSERT ON ENROLL
FOR EACH ROW
BEGIN
    IF NEW.marks < 40 THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Marks prerequisite not met';
    END IF;
END//

DELIMITER;
```

create a trigger that prevents a student from enrolling in a course if the marks prerequisite is less than 40.

5. Company Database:

EMPLOYEE (SSN, Name, Address, Sex, Salary, SuperSSN, DNo)
DEPARTMENT (DNo, DName, MgrSSN, MgrStartDate)
DLOCATION (DNo,DLoc)
PROJECT (PNo, PName, PLocation, DNo)
WORKS_ON (SSN, PNo, Hours)

1. Make a list of all project numbers for projects that involve an employee whose last name is 'Scott', either as a worker or as a manager of the department that controls the project.
2. Show the resulting salaries if every employee working on the 'IoT' project is given a 10 percent raise.
3. Find the sum of the salaries of all employees of the 'Accounts' department, as well as the maximum salary, the minimum salary, and the average salary in this department
4. Retrieve the name of each employee who works on all the projects controlled by department number 5 (use NOT EXISTS operator).
5. For each department that has more than five employees, retrieve the department number and the number of its employees who are making more than Rs. 6,00,000.
6. Create a view that shows name, dept name and location of all employees.
7. Create a trigger that prevents a project from being deleted if it is currently being worked by any employee.

```
CREATE TABLE DEPARTMENT (
    DNo INT PRIMARY KEY,
    DName VARCHAR(25),
    MgrSSN INT,
    MgrStartDate DATE

);

CREATE TABLE EMPLOYEE (
    SSN INT PRIMARY KEY,
    Name VARCHAR(25),
    Address VARCHAR(255),
    Sex CHAR(1),
    Salary FLOAT,
    SuperSSN INT,
    DNo INT,
    FOREIGN KEY (DNo) REFERENCES DEPARTMENT(DNo) ON DELETE CASCADE
);


CREATE TABLE DLOCATION (
    DNo INT PRIMARY KEY,
    DLoc VARCHAR(255),
    FOREIGN KEY (DNo) REFERENCES DEPARTMENT(DNo) ON DELETE CASCADE
);

CREATE TABLE PROJECT (
    PNo INT PRIMARY KEY,
    PName VARCHAR(255),
    PLocation VARCHAR(255),
    DNo INT,
    FOREIGN KEY (DNo) REFERENCES DEPARTMENT(DNo) ON DELETE CASCADE
);

CREATE TABLE WORKS_ON (
    SSN INT,
    PNo INT,
    Hours INT,
    PRIMARY KEY (SSN, PNo),
    FOREIGN KEY (SSN) REFERENCES EMPLOYEE(SSN) ON DELETE CASCADE,
    FOREIGN KEY (PNo) REFERENCES PROJECT(PNo) ON DELETE CASCADE
);


INSERT INTO DEPARTMENT VALUES
(1, 'HR', 1, '2023-01-01'),
(2, 'IT', 2, '2023-02-01'),
(3, 'Finance', 4, '2023-03-01'),
(4,'Accounts',3,'2023-09-07'),
(5,'Production',5,'2022-07-06');

INSERT INTO EMPLOYEE VALUES
(1, 'John Doe', '123 Main St', 'M', 500000,3, 1),
(2, 'Jane Smith', '456 Oak St', 'F', 600008, 1, 4),
(3, 'Bob Johnson', '789 Pine St', 'M', 550090, 2, 4),
(4, 'Alice Brown', '101 Elm St', 'F', 700009, 4, 3),
(5, 'Miller Scott', '202 Maple St', 'M', 800700, 2, 1);

INSERT INTO DLOCATION VALUES
(1, 'New York'),
(2, 'San Francisco'),
(3, 'Chicago'),
(4,'Alaska'),
(5,'California');

INSERT INTO PROJECT VALUES
(101, 'IoT', 'New York', 2),
(102, 'System Testing', 'San Francisco', 5),
(103, 'Product optimization', 'Chicago', 3),
(104, 'yield increase','California',1),
(105, 'Product refinement','Alaska',4);
```

```
INSERT INTO WORKS_ON VALUES
(1, 101, 40),
(2, 102, 30),
(3, 101, 20),
(4, 103, 25),
(5, 102, 35);
```

```
SELECT DISTINCT P.PNo
FROM PROJECT P
JOIN WORKS_ON W ON P.PNo = W.PNo
JOIN EMPLOYEE E ON W.SSN = E.SSN
WHERE E.Name LIKE '%Scott';
```

Make a list of all project <u>no</u> for projects that involve an employee whose last name is 'Scott', either as a worker or as a manager of the dpt that controls the project.

```
UPDATE EMPLOYEE
SET Salary = Salary * 1.10
WHERE SSN IN (SELECT SSN
              FROM WORKS_ON
              WHERE PNo = (SELECT PNo
                           FROM PROJECT
                           WHERE PName = 'IoT'));
SELECT * FROM EMPLOYEE;
```

Show resulting salaries if every employee working on 'IoT' project is given 10% raise.

```
SELECT
    SUM(Salary) AS TotalSalary,
    MAX(Salary) AS MaxSalary,
    MIN(Salary) AS MinSalary,
    AVG(Salary) AS AvgSalary
FROM EMPLOYEE
WHERE DNo = (SELECT DNo
             FROM DEPARTMENT
             WHERE DName = 'Accounts');
```

Find the sum of the salaries of all employees of the 'Accounts' department as well as the maximum, minimum and average salary in this department.

```
SELECT EMPLOYEE.SSN,Name,DNo
FROM EMPLOYEE
WHERE NOT EXISTS
    (SELECT PNo
    FROM PROJECT p
    WHERE p.DNo=5
    AND
    PNo NOT IN
        (SELECT PNo
        FROM WORKS_ON w
        WHERE w.SSN=EMPLOYEE.SSN));
```

Retrive the name of each employee who works on all the projects controlled by department no. 5 (Use NOT EXISTS operator)

```
SELECT DNo, COUNT(*) AS NumEmployees
FROM EMPLOYEE
WHERE Salary > 600000
GROUP BY DNo
HAVING COUNT(*) > 5;
```

For each department that has more than 5 employees, retrive the department <u>no</u> and the number of its employees who are making more than ₹6,00,000/-

```
CREATE VIEW EmployeeDetails AS
SELECT E.Name, D.DName AS DeptName, DL.DLoc AS DeptLocation
FROM EMPLOYEE E
JOIN DEPARTMENT D ON E.DNo = D.DNo
JOIN DLOCATION DL ON DL.DNo = D.DNo;
SELECT * FROM EmployeeDetails;
```

Create view that shows name, dpt name and ~~all~~ locations of all employees

```
DELIMITER //

CREATE TRIGGER PreventProjectDeletion
BEFORE DELETE ON PROJECT
FOR EACH ROW
BEGIN
    DECLARE projectCount INT;
    SELECT COUNT(*)
    INTO projectCount
    FROM WORKS_ON
    WHERE PNo = OLD.PNo;
```

Create a trigger that prevents a project from being deleted if it is currently being worked ~~on~~ by any employee.

```
    IF projectCount > 0 THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Cannot delete project currently being worked on.';
    END IF;
END//

DELIMITER;
```