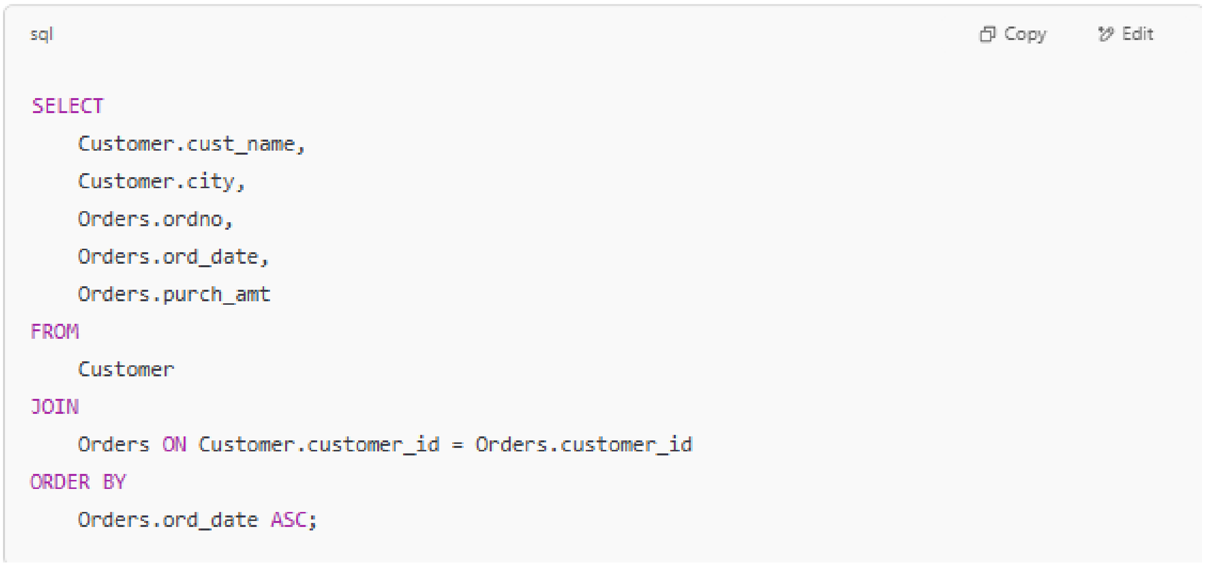
1. **Write a SQL statement to make a report with customer name, city, order number, order date, and order amount in ascending order according to the order date to determine whether any of the existing customers have placed an order or not. Relation: Orders(Ordno,Purch_amt,ord_date,customer_id,salesman_id) Customer(customer_id,cust_name,city,grade,salesman_id)**



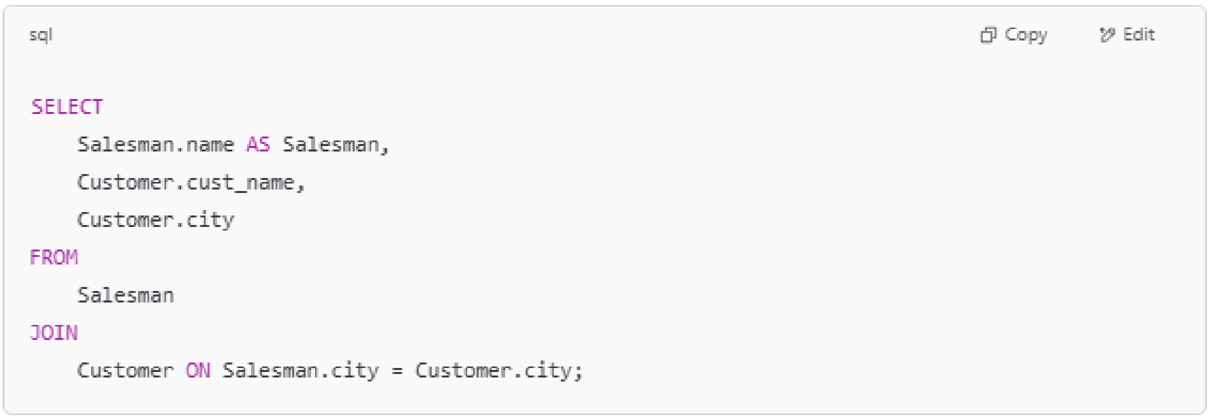
```sql
SELECT
    Customer.cust_name,
    Customer.city,
    Orders.ordno,
    Orders.ord_date,
    Orders.purch_amt
FROM
    Customer
JOIN
    Orders ON Customer.customer_id = Orders.customer_id
ORDER BY
    Orders.ord_date ASC;
```

```
SELECT
  Customer.cust_name,
  Customer.city,
  Orders.ordno,
  Orders.ord_date,
  Orders.purch_amt
FROM
  Customer
JOIN
  Orders ON Customer.customer_id = Orders.customer_id
ORDER BY
  Orders.ord_date ASC;
```

☑ This query:

- Retrieves **customer name, city, order number, order date, and order amount**.
- Ensures only customers **who have placed orders** are included (via JOIN).
- **Orders the result in ascending order of order date**.

2. **write a SQL query to find the salesperson and customer who reside in the same city. Return Salesman, cust_name and city. Relation: salesman(salesman_id,name,city,commission) Customer(customer_id,cust_name,city,grade,salesman_id)**



```sql
SELECT
    Salesman.name AS Salesman,
    Customer.cust_name,
    Customer.city
FROM
    Salesman
JOIN
    Customer ON Salesman.city = Customer.city;
```

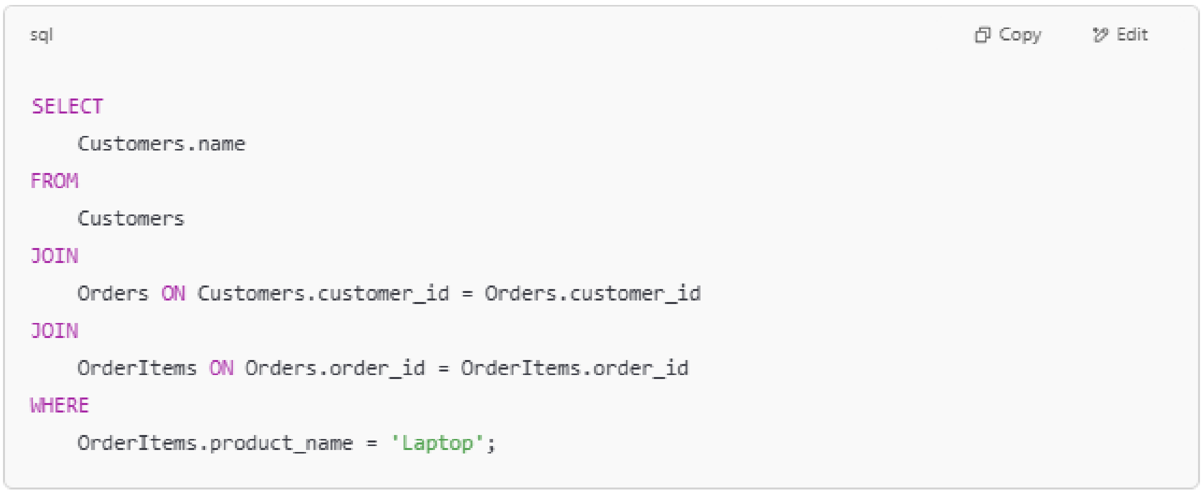
```
SELECT
  Salesman.name AS Salesman,
  Customer.cust_name,
  Customer.city
FROM
  Salesman
JOIN
  Customer ON Salesman.city = Customer.city;
```

☑ This query:

- Uses a JOIN between the salesman and customer tables.
- Returns only those **salesman-customer pairs who reside in the same city**.
- Displays **Salesman name, Customer name, and their shared City**.

4o

3. **Get names of customers who have ordered 'Laptop'. Tables: Customers(customer_id, name) Orders(order_id, customer_id) OrderItems(order_id, product_name)**



```sql
SELECT
    Customers.name
FROM
    Customers
JOIN
    Orders ON Customers.customer_id = Orders.customer_id
JOIN
    OrderItems ON Orders.order_id = OrderItems.order_id
WHERE
    OrderItems.product_name = 'Laptop';
```

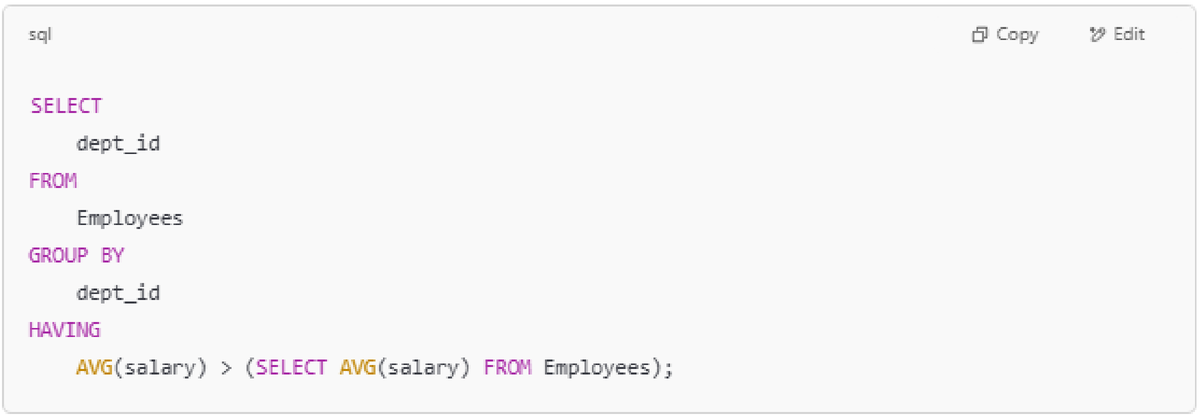
```
SELECT

  Customers.name
```

```
FROM
  Customers
JOIN
  Orders ON Customers.customer_id = Orders.customer_id
JOIN
  OrderItems ON Orders.order_id = OrderItems.order_id
WHERE
  OrderItems.product_name = 'Laptop';
```

☑ This query:

- Joins the three tables (Customers, Orders, OrderItems).
- Filters for rows where the **product name is 'Laptop'**.
- Returns only the **names of customers who placed such orders**.

4. **Write a query that finds departments with average salary > overall average salary? Table:Employees(emp_id, name, salary, dept_id)**



```sql
SELECT
    dept_id
FROM
    Employees
GROUP BY
    dept_id
HAVING
    AVG(salary) > (SELECT AVG(salary) FROM Employees);
```

```
SELECT
  dept_id
FROM
  Employees
GROUP BY
```

dept_id

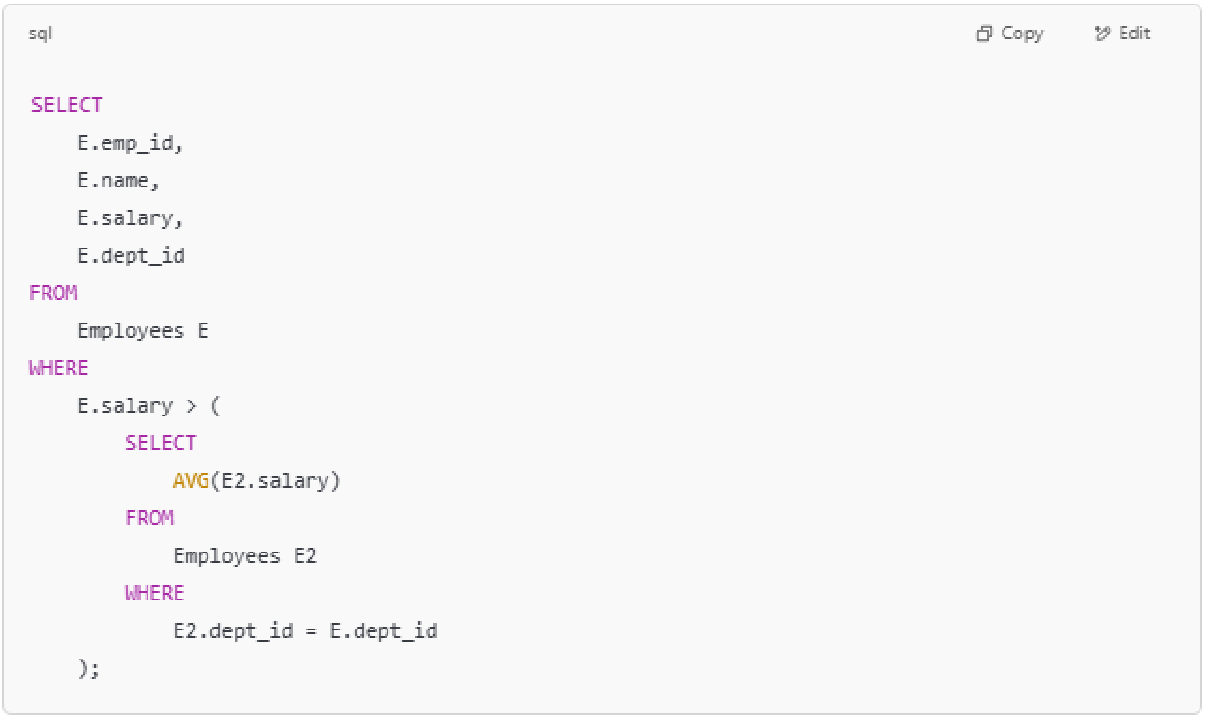
HAVING

AVG(salary) > (SELECT AVG(salary) FROM Employees);

☑ This query:

- Groups employees by dept_id.
- Calculates average salary for each department.
- Compares it with the **overall average salary** using a subquery.
- Returns **only the dept_id** of qualifying departments.

5. **Select employees who earn more than the average salary of their department. Table:Employees(emp_id, name, salary, dept_id)**



```sql
SELECT
    E.emp_id,
    E.name,
    E.salary,
    E.dept_id
FROM
    Employees E
WHERE
    E.salary > (
        SELECT
            AVG(E2.salary)
        FROM
            Employees E2
        WHERE
            E2.dept_id = E.dept_id
    );
```

SELECT

E.emp_id,

E.name,

E.salary,

E.dept_id

FROM

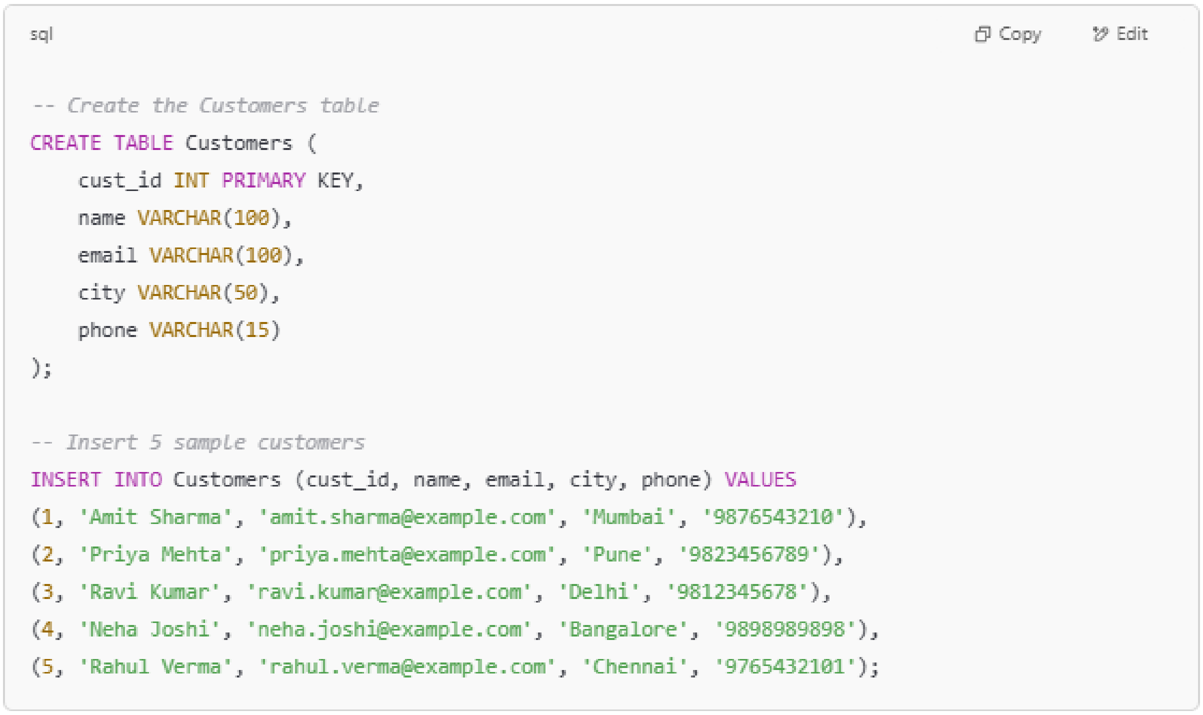
Employees E

```
WHERE
  E.salary > (
    SELECT
      AVG(E2.salary)
    FROM
      Employees E2
    WHERE
      E2.dept_id = E.dept_id
  );
```

☑ This query:

- Compares each employee's salary to the **average salary within their department**.
- Uses a **correlated subquery** to filter those who earn **more than their department average**.

6. **Create a table named Customers with the following structure: Customers(cust_id primary key, name, email, city, phone) Insert at least 5 customers into the Customers table with different cities and emails.**



```sql
-- Create the Customers table
CREATE TABLE Customers (
    cust_id INT PRIMARY KEY,
    name VARCHAR(100),
    email VARCHAR(100),
    city VARCHAR(50),
    phone VARCHAR(15)
);

-- Insert 5 sample customers
INSERT INTO Customers (cust_id, name, email, city, phone) VALUES
(1, 'Amit Sharma', 'amit.sharma@example.com', 'Mumbai', '9876543210'),
(2, 'Priya Mehta', 'priya.mehta@example.com', 'Pune', '9823456789'),
(3, 'Ravi Kumar', 'ravi.kumar@example.com', 'Delhi', '9812345678'),
(4, 'Neha Joshi', 'neha.joshi@example.com', 'Bangalore', '9898989898'),
(5, 'Rahul Verma', 'rahul.verma@example.com', 'Chennai', '9765432101');
```

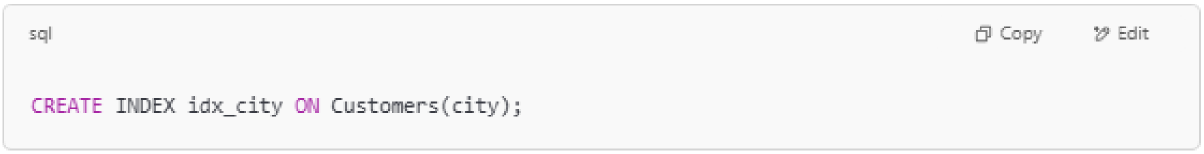
```
-- Create the Customers table
```

```sql
CREATE TABLE Customers (
  cust_id INT PRIMARY KEY,
  name VARCHAR(100),
  email VARCHAR(100),
  city VARCHAR(50),
  phone VARCHAR(15)
);
-- Insert 5 sample customers
INSERT INTO Customers (cust_id, name, email, city, phone) VALUES
(1, 'Amit Sharma', 'amit.sharma@example.com', 'Mumbai', '9876543210'),
(2, 'Priya Mehta', 'priya.mehta@example.com', 'Pune', '9823456789'),
(3, 'Ravi Kumar', 'ravi.kumar@example.com', 'Delhi', '9812345678'),
(4, 'Neha Joshi', 'neha.joshi@example.com', 'Bangalore', '9898989898'),
(5, 'Rahul Verma', 'rahul.verma@example.com', 'Chennai', '9765432101');
```

☑ This code:

- Creates a table with a PRIMARY KEY on cust_id.
- Inserts 5 customers with **unique cities and emails**.

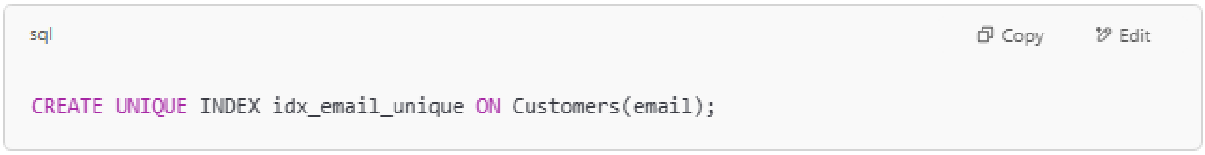
a) Create an index on the city column to improve city-based queries.



```sql
CREATE INDEX idx_city ON Customers(city);
```

```sql
CREATE INDEX idx_city ON Customers(city);
```

b) Create a unique index on the email column to ensure no duplicate emails are stored.



```sql
CREATE UNIQUE INDEX idx_email_unique ON Customers(email);
```

```sql
CREATE UNIQUE INDEX idx_email_unique ON Customers(email);
```

c) Create a composite index on city and name.

```sql
CREATE INDEX idx_city_name ON Customers(city, name);
```

CREATE INDEX idx_city_name ON Customers(city, name);

d) Drop the index on city.

```sql
DROP INDEX idx_city;
```

DROP INDEX idx_city;

e) Drop the unique index on email.

```sql
DROP INDEX idx_email_unique;
```

DROP INDEX idx_email_unique;

7. **Create a table named Books with the following structure: Books(book_id primary key, title not null, author, genre, isbn unique, published_year) Insert at least 3 books details into the Books table.**

```sql
-- Create the Books table
CREATE TABLE Books (
    book_id INT PRIMARY KEY,
    title VARCHAR(255) NOT NULL,
    author VARCHAR(100),
    genre VARCHAR(50),
    isbn VARCHAR(20) UNIQUE,
    published_year INT
);

-- Insert 3 sample book records
INSERT INTO Books (book_id, title, author, genre, isbn, published_year) VALUES
(1, 'The Silent Patient', 'Alex Michaelides', 'Thriller', '9781250301697', 2019),
(2, 'Atomic Habits', 'James Clear', 'Self-help', '9780735211292', 2018),
(3, 'To Kill a Mockingbird', 'Harper Lee', 'Fiction', '9780061120084', 1960);
```

```
-- Create the Books table
CREATE TABLE Books (
    book_id INT PRIMARY KEY,
    title VARCHAR(255) NOT NULL,
    author VARCHAR(100),
```

```sql
    genre VARCHAR(50),
    isbn VARCHAR(20) UNIQUE,
    published_year INT
);


-- Insert 3 sample book records
INSERT INTO Books (book_id, title, author, genre, isbn, published_year) VALUES
(1, 'The Silent Patient', 'Alex Michaelides', 'Thriller', '9781250301697', 2019),
(2, 'Atomic Habits', 'James Clear', 'Self-help', '9780735211292', 2018),
(3, 'To Kill a Mockingbird', 'Harper Lee', 'Fiction', '9780061120084', 1960);
```

a) Create an index on the genre column to improve genre-based queries.

```sql
CREATE INDEX idx_genre ON Books(genre);
```

```sql
CREATE INDEX idx_genre ON Books(genre);
```

b) Create an index on the author column to improve author search performance.

```sql
CREATE INDEX idx_author ON Books(author);
```

```sql
CREATE INDEX idx_author ON Books(author);
```

c) Create a composite index on the author and published_year columns.

```sql
CREATE INDEX idx_author_year ON Books(author, published_year);
```

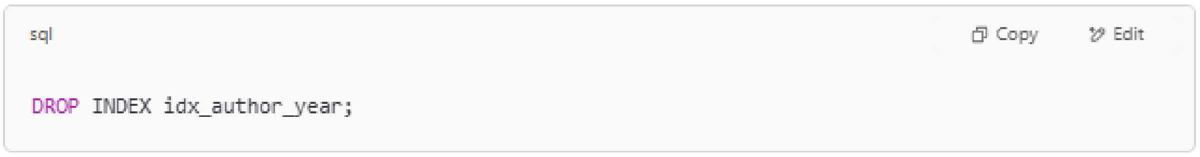
```sql
CREATE INDEX idx_author_year ON Books(author, published_year);
```

d) Drop the index on the genre column.

```sql
DROP INDEX idx_genre;
```

```sql
DROP INDEX idx_genre;
```

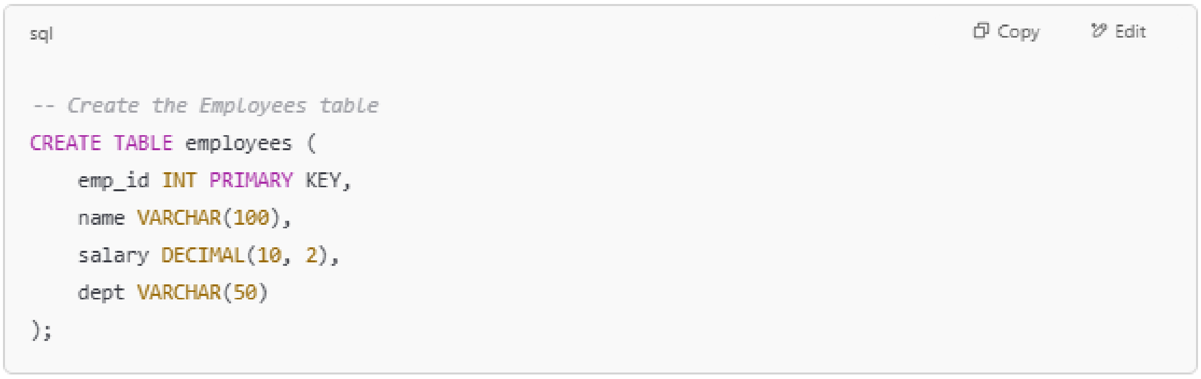
e) Drop the composite index on author and published_year.



```sql
DROP INDEX idx_author_year;
```

DROP INDEX idx_author_year;

8. **Create a table named employees with the following structure: employees(emp_id primary key, name, salary, dept)**



```sql
-- Create the Employees table
CREATE TABLE employees (
    emp_id INT PRIMARY KEY,
    name VARCHAR(100),
    salary DECIMAL(10, 2),
    dept VARCHAR(50)
);
```

-- Create the Employees table

CREATE TABLE employees (

emp_id INT PRIMARY KEY,

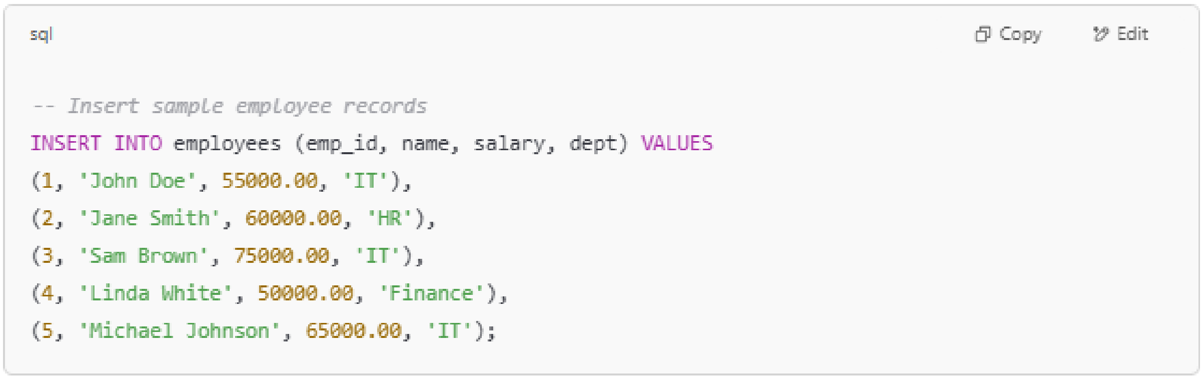
name VARCHAR(100),

salary DECIMAL(10, 2),

dept VARCHAR(50)

);

a) Insert records into the employees table.



```sql
-- Insert sample employee records
INSERT INTO employees (emp_id, name, salary, dept) VALUES
(1, 'John Doe', 55000.00, 'IT'),
(2, 'Jane Smith', 60000.00, 'HR'),
(3, 'Sam Brown', 75000.00, 'IT'),
(4, 'Linda White', 50000.00, 'Finance'),
(5, 'Michael Johnson', 65000.00, 'IT');
```

-- Insert sample employee records
INSERT INTO employees (emp_id, name, salary, dept) VALUES

(1, 'John Doe', 55000.00, 'IT'),
(2, 'Jane Smith', 60000.00, 'HR'),
(3, 'Sam Brown', 75000.00, 'IT'),
(4, 'Linda White', 50000.00, 'Finance'),
(5, 'Michael Johnson', 65000.00, 'IT');

b) write a stored procedure that uses a cursor to retrieve employees's names and salaries from a particular department, then print out the results.

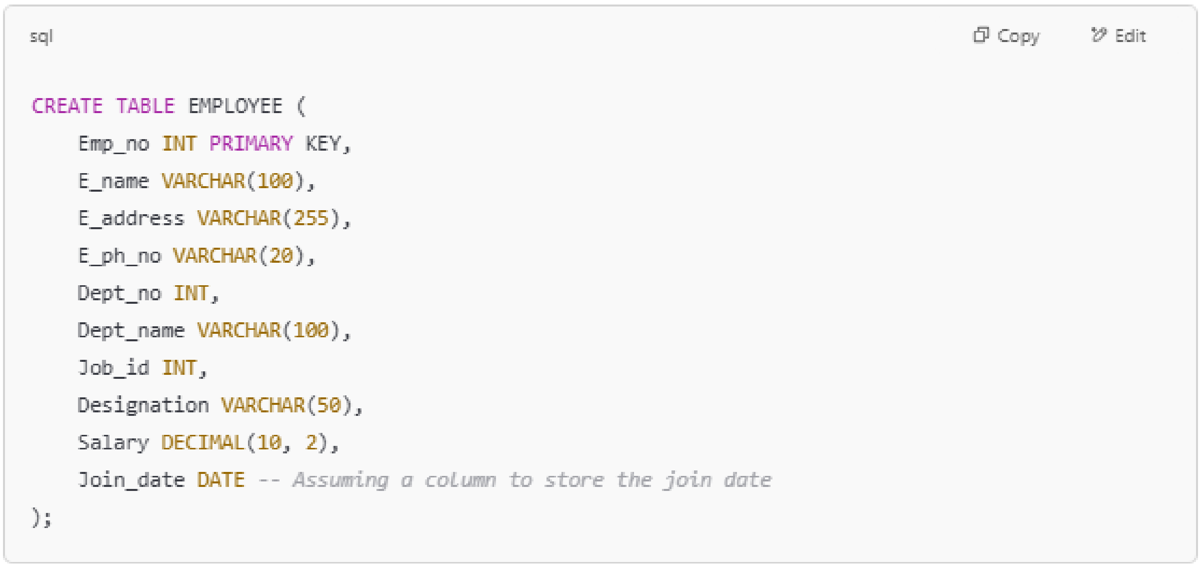
```
DELIMITER $$

CREATE PROCEDURE GetEmployeesByDept(department VARCHAR(50))
BEGIN
    -- Declare variables for cursor
    DECLARE done INT DEFAULT 0;
    DECLARE emp_name VARCHAR(100);
    DECLARE emp_salary DECIMAL(10, 2);

    -- Declare cursor for fetching employee names and salaries
    DECLARE employee_cursor CURSOR FOR
        SELECT name, salary FROM employees WHERE dept = department;

    -- Declare handler for when cursor is done
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;

    -- Open the cursor
    OPEN employee_cursor;

    -- Loop through the cursor
    read_loop: LOOP
        FETCH employee_cursor INTO emp_name, emp_salary;

        IF done THEN
            LEAVE read_loop;
        END IF;

        -- Print the employee name and salary
        SELECT emp_name AS 'Employee Name', emp_salary AS 'Salary';
    END LOOP;

    -- Close the cursor
    CLOSE employee_cursor;
END $$

DELIMITER ;
```

DELIMITER $$

CREATE PROCEDURE GetEmployeesByDept(department VARCHAR(50))
BEGIN
  -- Declare variables for cursor

```
    DECLARE done INT DEFAULT 0;
    DECLARE emp_name VARCHAR(100);
    DECLARE emp_salary DECIMAL(10, 2);

    -- Declare cursor for fetching employee names and salaries
    DECLARE employee_cursor CURSOR FOR
      SELECT name, salary FROM employees WHERE dept = department;

    -- Declare handler for when cursor is done
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;

    -- Open the cursor
    OPEN employee_cursor;

    -- Loop through the cursor
    read_loop: LOOP
      FETCH employee_cursor INTO emp_name, emp_salary;

      IF done THEN
        LEAVE read_loop;
      END IF;

      -- Print the employee name and salary
      SELECT emp_name AS 'Employee Name', emp_salary AS 'Salary';
    END LOOP;

    -- Close the cursor
    CLOSE employee_cursor;
  END $$

  DELIMITER ;
```

9. **Create a table EMPLOYEE with following schema: (Emp_no, E_name, E_address, E_ph_no, Dept_no, Dept_name,Job_id, Designation , Salary) Write SQL statements for the following query.**



```sql
CREATE TABLE EMPLOYEE (
    Emp_no INT PRIMARY KEY,
    E_name VARCHAR(100),
    E_address VARCHAR(255),
    E_ph_no VARCHAR(20),
    Dept_no INT,
    Dept_name VARCHAR(100),
    Job_id INT,
    Designation VARCHAR(50),
    Salary DECIMAL(10, 2),
    Join_date DATE -- Assuming a column to store the join date
);
```

CREATE TABLE EMPLOYEE (

Emp_no INT PRIMARY KEY,

E_name VARCHAR(100),

E_address VARCHAR(255),

E_ph_no VARCHAR(20),

Dept_no INT,

Dept_name VARCHAR(100),

Job_id INT,

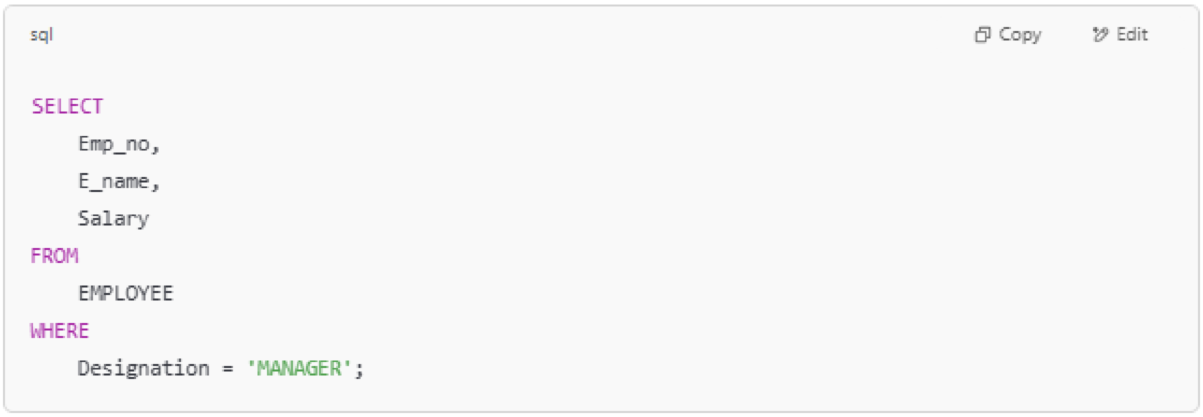
Designation VARCHAR(50),

Salary DECIMAL(10, 2),

Join_date DATE -- Assuming a column to store the join date

);

1. List the E_no, E_name, Salary of all employees working for MANAGER.

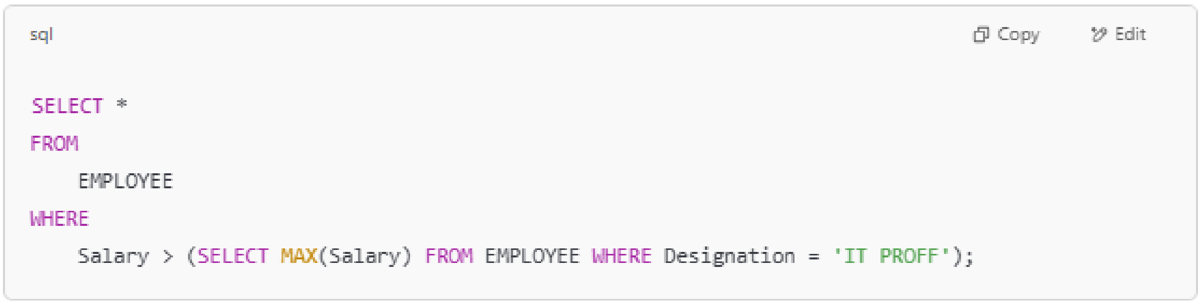


```sql
SELECT
    Emp_no,
    E_name,
    Salary
FROM
    EMPLOYEE
WHERE
    Designation = 'MANAGER';
```

SELECT

```
  Emp_no,
  E_name,
  Salary
FROM
  EMPLOYEE
WHERE
  Designation = 'MANAGER';
```

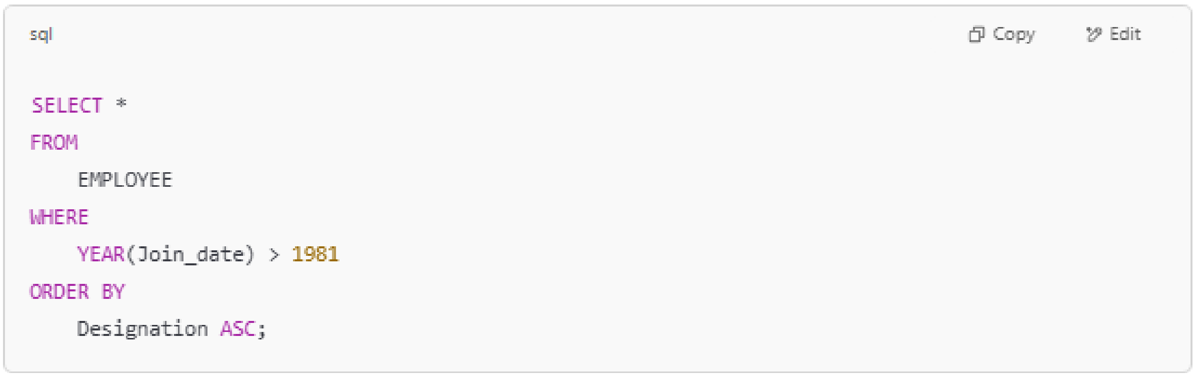
2. Display all the details of the employee whose salary is more than the Sal of any IT PROFF.



```sql
SELECT *
FROM
    EMPLOYEE
WHERE
    Salary > (SELECT MAX(Salary) FROM EMPLOYEE WHERE Designation = 'IT PROFF');
```

```
SELECT *
FROM
  EMPLOYEE
WHERE
  Salary > (SELECT MAX(Salary) FROM EMPLOYEE WHERE Designation = 'IT PROFF');
```

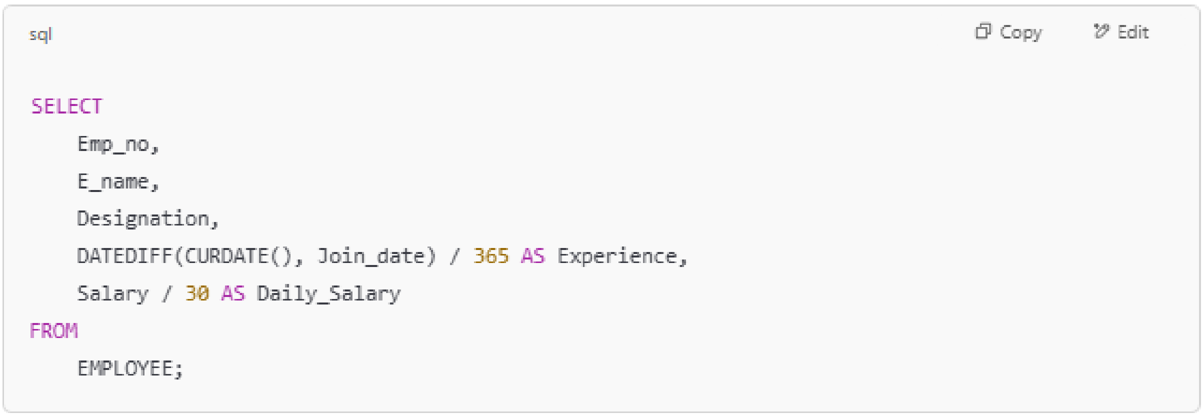
3. List the employees in the ascending order of Designations of those joined after 1981.



```sql
SELECT *
FROM
    EMPLOYEE
WHERE
    YEAR(Join_date) > 1981
ORDER BY
    Designation ASC;
```

```
SELECT *
FROM
  EMPLOYEE
WHERE
  YEAR(Join_date) > 1981
ORDER BY
  Designation ASC;
```

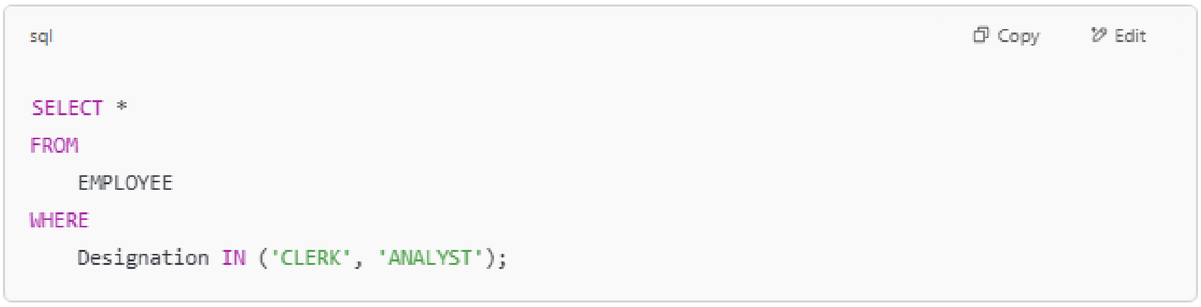
4. List the employees along with their Experience and Daily Salary.



```sql
SELECT
    Emp_no,
    E_name,
    Designation,
    DATEDIFF(CURDATE(), Join_date) / 365 AS Experience,
    Salary / 30 AS Daily_Salary
FROM
    EMPLOYEE;
```

```
SELECT
  Emp_no,
  E_name,
  Designation,
  DATEDIFF(CURDATE(), Join_date) / 365 AS Experience,
  Salary / 30 AS Daily_Salary
FROM
  EMPLOYEE;
```

5. List the employees who are either ‘CLERK’ or ‘ANALYST’.

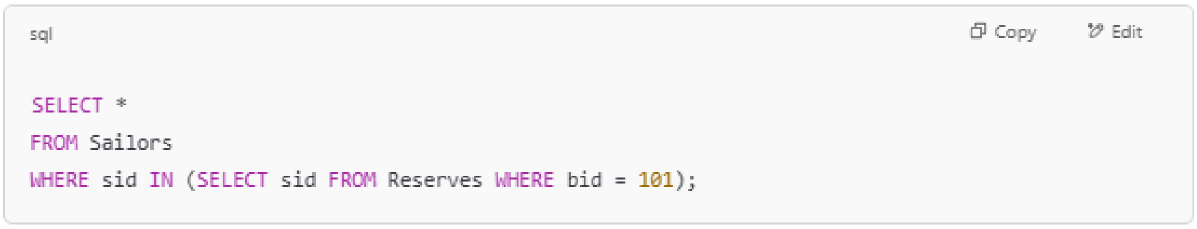


```sql
SELECT *
FROM
    EMPLOYEE
WHERE
    Designation IN ('CLERK', 'ANALYST');
```

```
SELECT *
FROM
  EMPLOYEE
WHERE
  Designation IN ('CLERK', 'ANALYST');
```

10. Consider the following schema: Sailors (sid, sname, rating, age) Boats (bid, bname, color) Reserves (sid, bid, day(date))

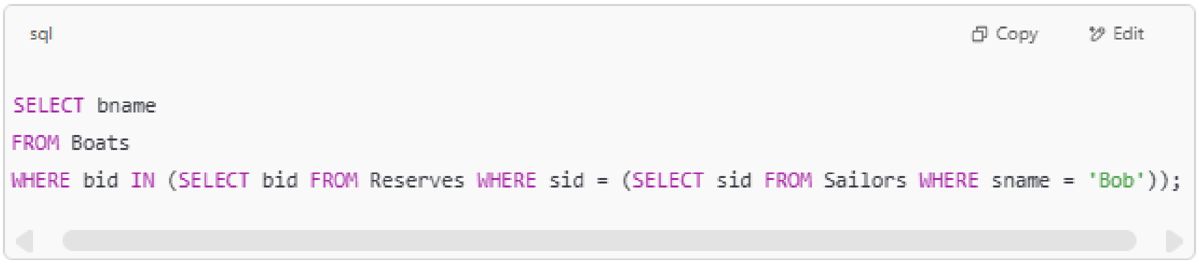
1. Find all information of sailors who have reserved boat number 101.



```sql
SELECT *
FROM Sailors
WHERE sid IN (SELECT sid FROM Reserves WHERE bid = 101);
```

SELECT *
FROM Sailors
WHERE sid IN (SELECT sid FROM Reserves WHERE bid = 101);

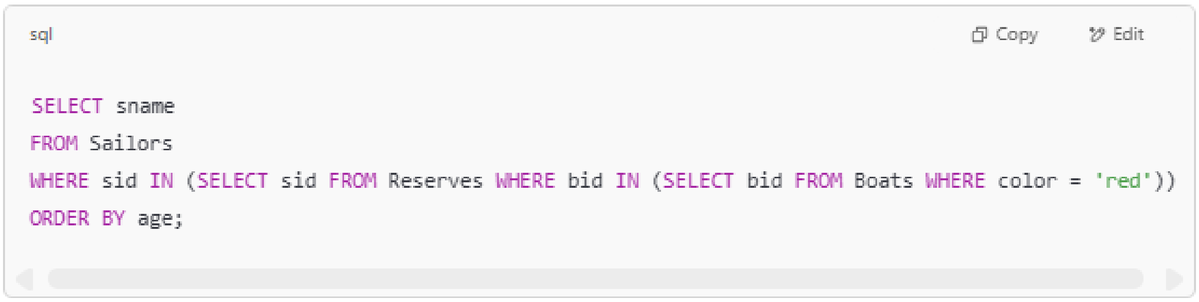
2. Find the name of the boat reserved by Bob.



```sql
SELECT bname
FROM Boats
WHERE bid IN (SELECT bid FROM Reserves WHERE sid = (SELECT sid FROM Sailors WHERE sname = 'Bob'));
```

SELECT bname
FROM Boats
WHERE bid IN (SELECT bid FROM Reserves WHERE sid = (SELECT sid FROM Sailors WHERE sname = 'Bob'));

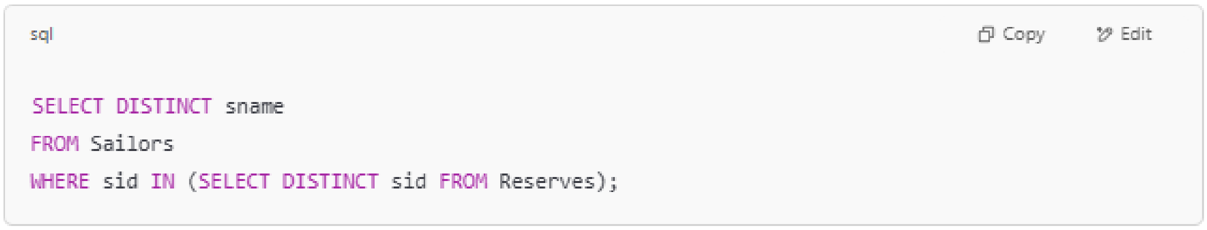
3. Find the names of sailors who have reserved a red boat, and list in the order of age.



```sql
SELECT sname
FROM Sailors
WHERE sid IN (SELECT sid FROM Reserves WHERE bid IN (SELECT bid FROM Boats WHERE color = 'red'))
ORDER BY age;
```

SELECT sname
FROM Sailors
WHERE sid IN (SELECT sid FROM Reserves WHERE bid IN (SELECT bid FROM Boats WHERE color = 'red'))
ORDER BY age;

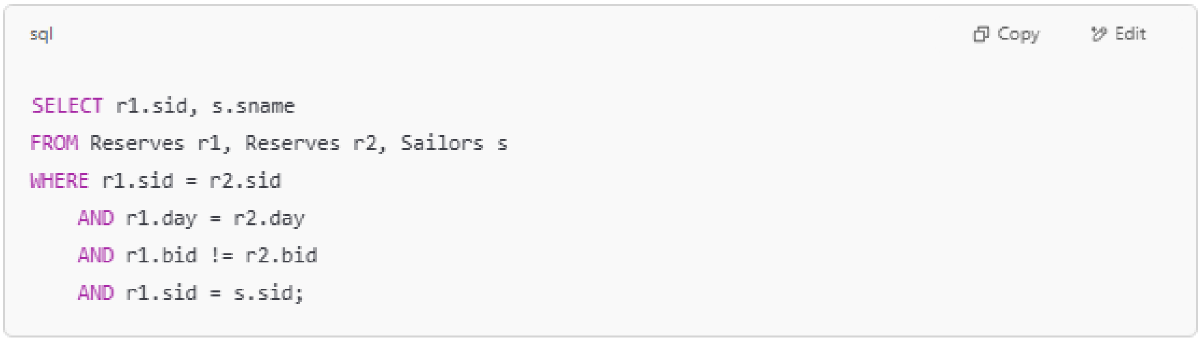
4. Find the names of sailors who have reserved at least one boat.



```sql
SELECT DISTINCT sname
FROM Sailors
WHERE sid IN (SELECT DISTINCT sid FROM Reserves);
```

SELECT DISTINCT sname
FROM Sailors
WHERE sid IN (SELECT DISTINCT sid FROM Reserves);

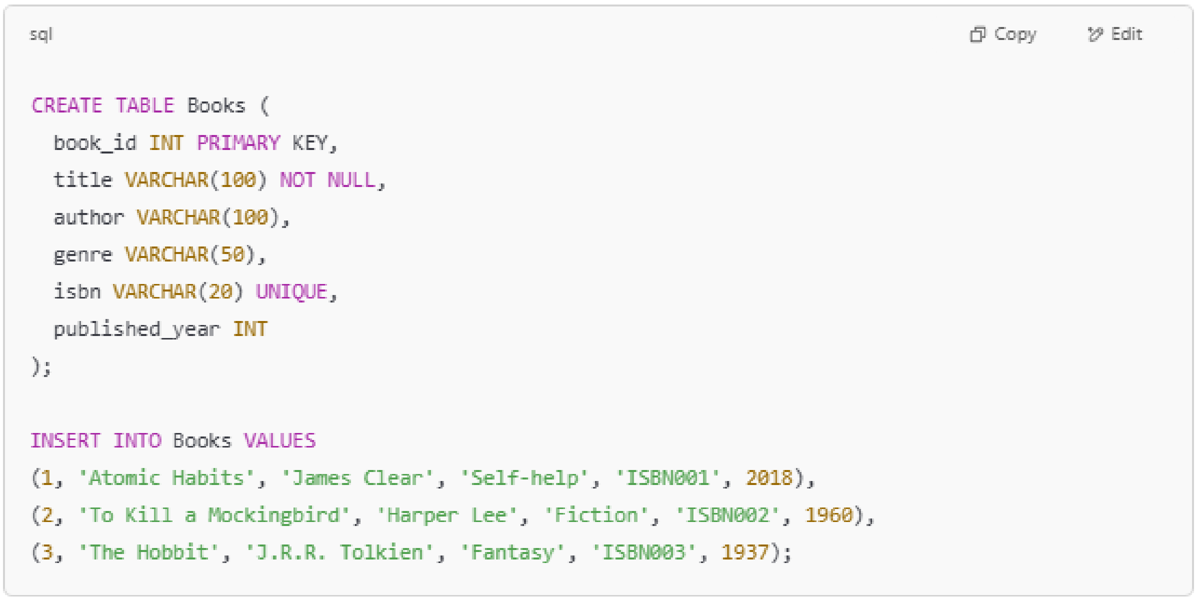
5. Find the ids and names of sailors who have reserved two different boats on the same Day



```sql
SELECT r1.sid, s.sname
FROM Reserves r1, Reserves r2, Sailors s
WHERE r1.sid = r2.sid
    AND r1.day = r2.day
    AND r1.bid != r2.bid
    AND r1.sid = s.sid;
```

SELECT r1.sid, s.sname
FROM Reserves r1, Reserves r2, Sailors s
WHERE r1.sid = r2.sid
  AND r1.day = r2.day
  AND r1.bid != r2.bid
  AND r1.sid = s.sid;

11. Create a table named Books with the following structure: Books(book_id primary key, title not null, author, genre, isbn unique, published_year) Insert at least 3 books details into the Books table.

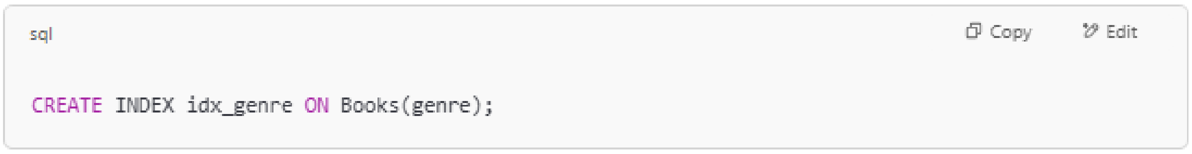


```sql
CREATE TABLE Books (
  book_id INT PRIMARY KEY,
  title VARCHAR(100) NOT NULL,
  author VARCHAR(100),
  genre VARCHAR(50),
  isbn VARCHAR(20) UNIQUE,
  published_year INT
);

INSERT INTO Books VALUES
(1, 'Atomic Habits', 'James Clear', 'Self-help', 'ISBN001', 2018),
(2, 'To Kill a Mockingbird', 'Harper Lee', 'Fiction', 'ISBN002', 1960),
(3, 'The Hobbit', 'J.R.R. Tolkien', 'Fantasy', 'ISBN003', 1937);
```

CREATE TABLE Books (
 book_id INT PRIMARY KEY,
 title VARCHAR(100) NOT NULL,
 author VARCHAR(100),
 genre VARCHAR(50),
 isbn VARCHAR(20) UNIQUE,
 published_year INT
);
INSERT INTO Books VALUES
(1, 'Atomic Habits', 'James Clear', 'Self-help', 'ISBN001', 2018),

(2, 'To Kill a Mockingbird', 'Harper Lee', 'Fiction', 'ISBN002', 1960),
(3, 'The Hobbit', 'J.R.R. Tolkien', 'Fantasy', 'ISBN003', 1937);

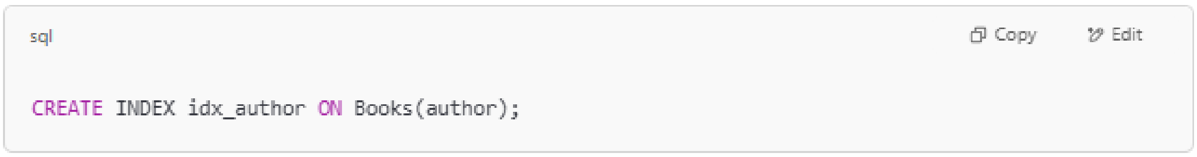
a) Create an index on the genre column to improve genre-based queries.



```sql
CREATE INDEX idx_genre ON Books(genre);
```

CREATE INDEX idx_genre ON Books(genre);

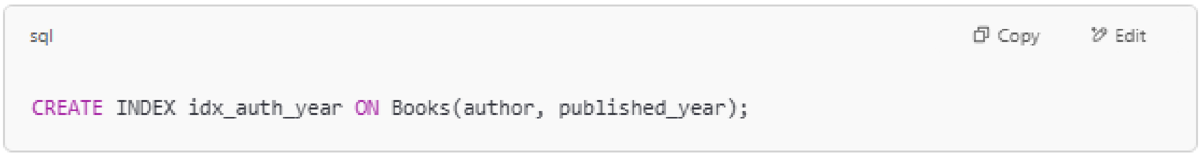
b) Create an index on the author column to improve author search performance.



```sql
CREATE INDEX idx_author ON Books(author);
```

CREATE INDEX idx_author ON Books(author);

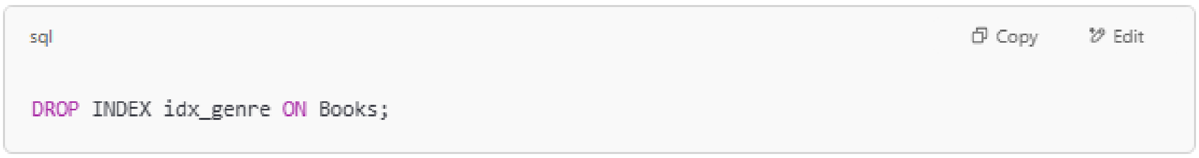
c) Create a composite index on the author and published_year columns.



```sql
CREATE INDEX idx_auth_year ON Books(author, published_year);
```

CREATE INDEX idx_auth_year ON Books(author, published_year);

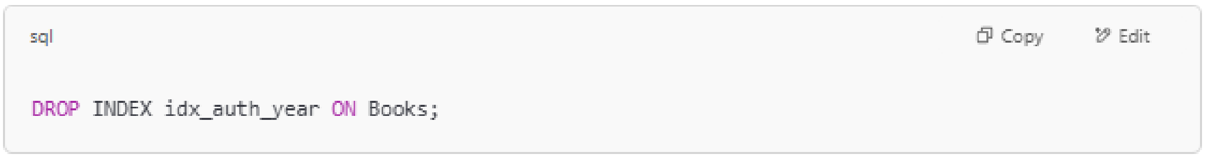
d) Drop the index on the genre column.



```sql
DROP INDEX idx_genre ON Books;
```

DROP INDEX idx_genre ON Books;

e) Drop the composite index on author and published_year.

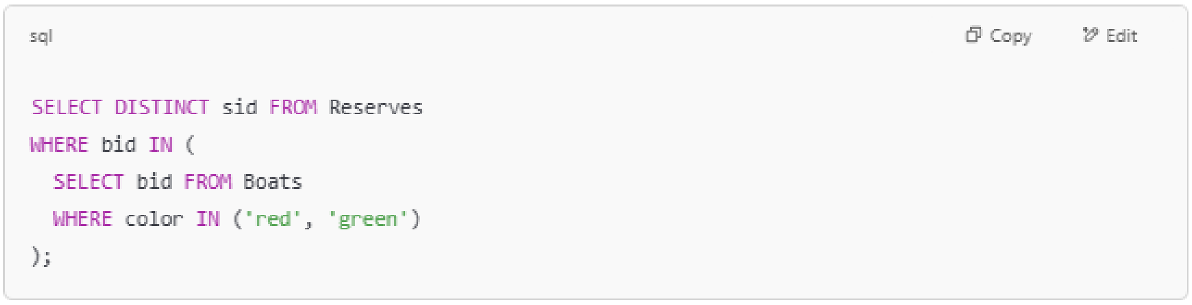


```sql
DROP INDEX idx_auth_year ON Books;
```

DROP INDEX idx_auth_year ON Books;

12. Consider the following schema: Sailors (sid, sname, rating, age) Boats (bid, bname, color) Reserves (sid, bid, day(date))

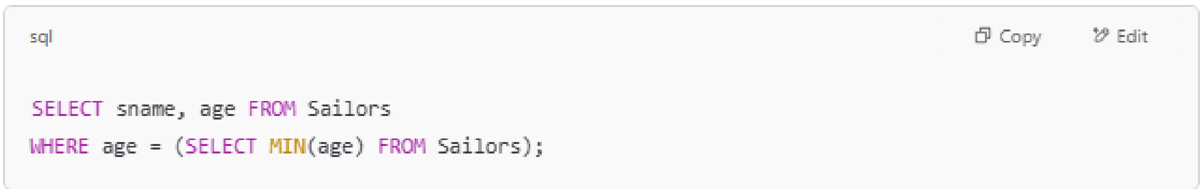
1. Find the ids of sailors who have reserved a red boat or a green boat.



```sql
SELECT DISTINCT sid FROM Reserves
WHERE bid IN (
  SELECT bid FROM Boats
  WHERE color IN ('red', 'green')
);
```

SELECT DISTINCT sid FROM Reserves
WHERE bid IN (
 SELECT bid FROM Boats
 WHERE color IN ('red', 'green')
);

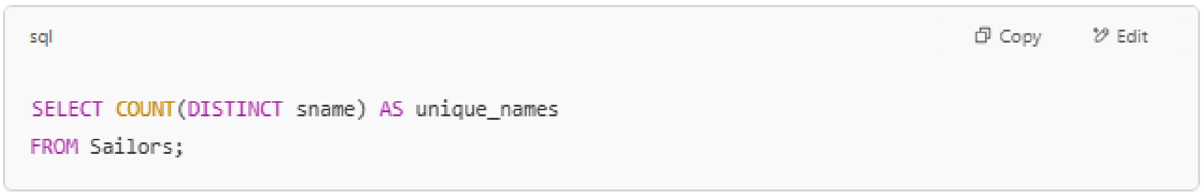
2. Find the name and the age of the youngest sailor.



```sql
SELECT sname, age FROM Sailors
WHERE age = (SELECT MIN(age) FROM Sailors);
```

SELECT sname, age FROM Sailors
WHERE age = (SELECT MIN(age) FROM Sailors);

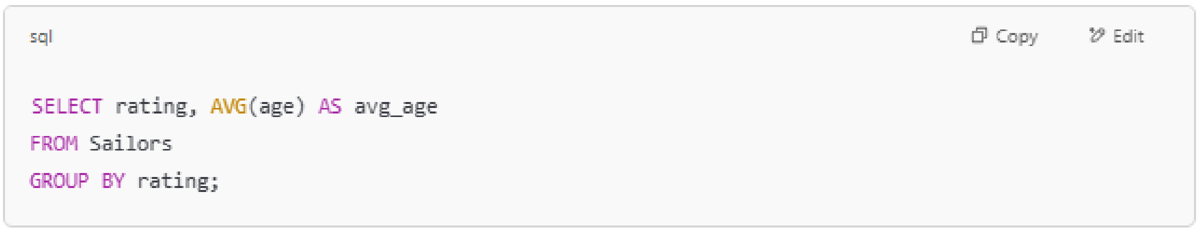
3. Count the number of different sailor names.



```sql
SELECT COUNT(DISTINCT sname) AS unique_names
FROM Sailors;
```

SELECT COUNT(DISTINCT sname) AS unique_names
FROM Sailors;

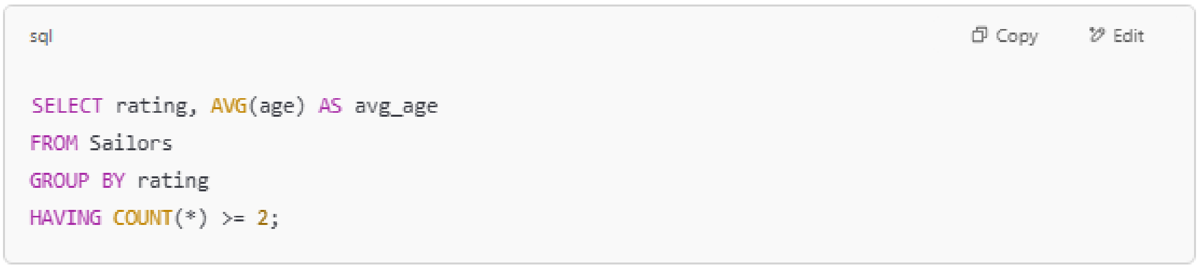
4. Find the average age of sailors for each rating level.



```sql
SELECT rating, AVG(age) AS avg_age
FROM Sailors
GROUP BY rating;
```

SELECT rating, AVG(age) AS avg_age
FROM Sailors
GROUP BY rating;

5. Find the average age of sailors for each rating level that has at least two sailors.

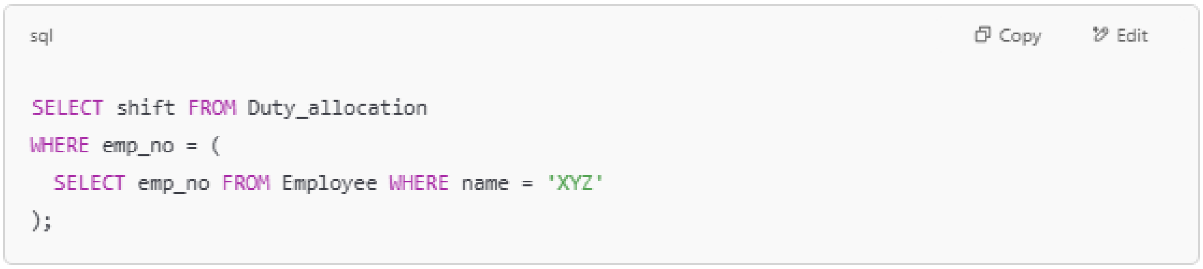


```sql
SELECT rating, AVG(age) AS avg_age
FROM Sailors
GROUP BY rating
HAVING COUNT(*) >= 2;
```

SELECT rating, AVG(age) AS avg_age
FROM Sailors
GROUP BY rating
HAVING COUNT(*) >= 2;

13. Consider the following database: Employee (emp_no, name, skill, pay rate) Position (posting_no, skill) Duty_allocation (posting_no, emp_no, day, shift(day/night))

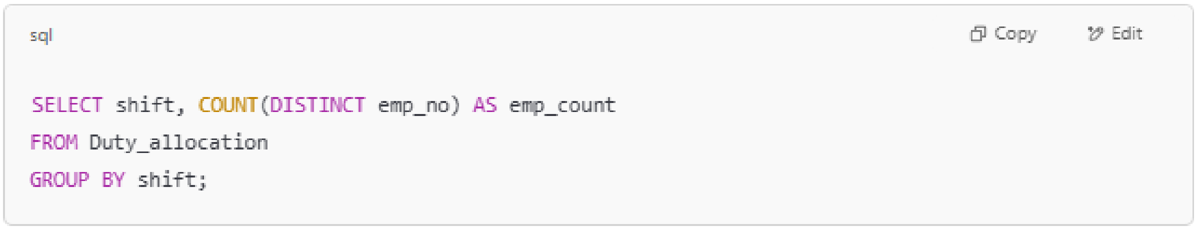
1.Find the shift for the employee 'XYZ':



```sql
SELECT shift FROM Duty_allocation
WHERE emp_no = (
  SELECT emp_no FROM Employee WHERE name = 'XYZ'
);
```

SELECT shift FROM Duty_allocation
WHERE emp_no = (
 SELECT emp_no FROM Employee WHERE name = 'XYZ'
);

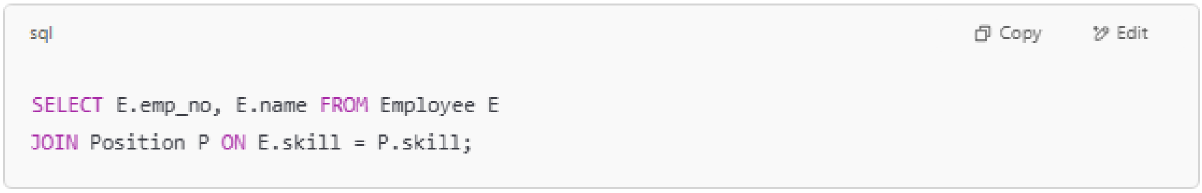
2.Get count of different employee of each shift:



```sql
SELECT shift, COUNT(DISTINCT emp_no) AS emp_count
FROM Duty_allocation
GROUP BY shift;
```

SELECT shift, COUNT(DISTINCT emp_no) AS emp_count
FROM Duty_allocation
GROUP BY shift;

3.Find the employee eligible to fill a position:

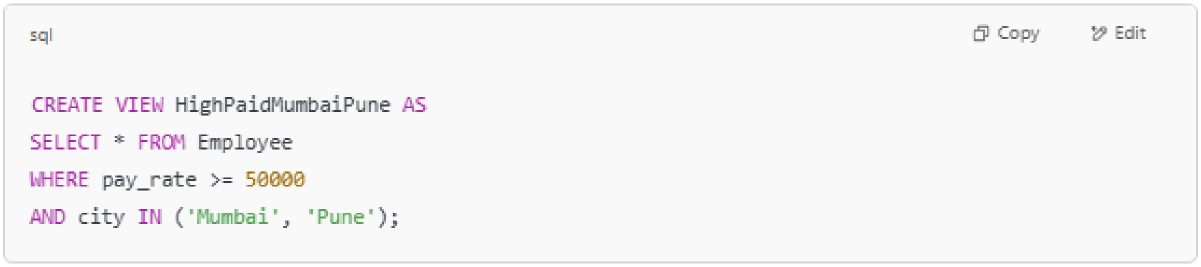


```sql
SELECT E.emp_no, E.name FROM Employee E
JOIN Position P ON E.skill = P.skill;
```

SELECT E.emp_no, E.name FROM Employee E

JOIN Position P ON E.skill = P.skill;

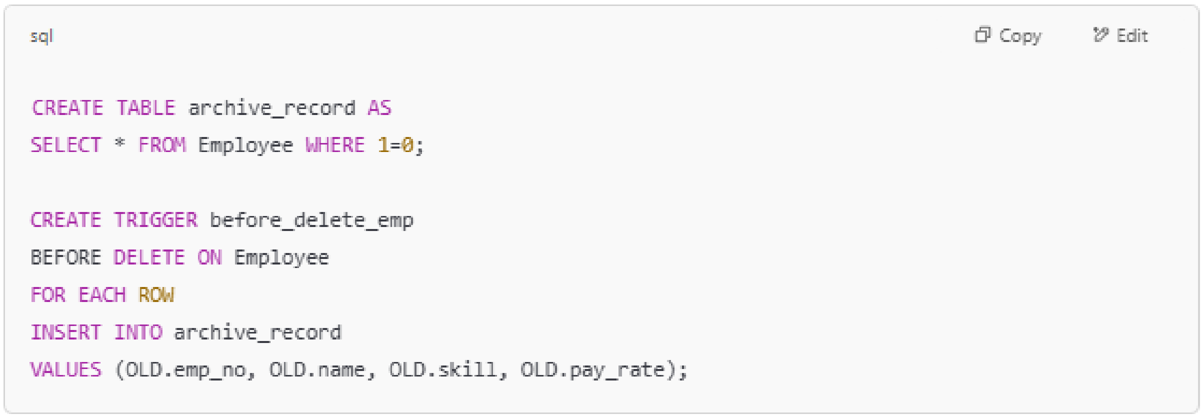
4.Create a view for employees having salary >=50000 and stays in 'Mumbai and Pune'



```sql
CREATE VIEW HighPaidMumbaiPune AS
SELECT * FROM Employee
WHERE pay_rate >= 50000
AND city IN ('Mumbai', 'Pune');
```

CREATE VIEW HighPaidMumbaiPune AS
SELECT * FROM Employee
WHERE pay_rate >= 50000
AND city IN ('Mumbai', 'Pune');

5. Create trigger before delete to archive the deleted employee record into 'archive_record' table.



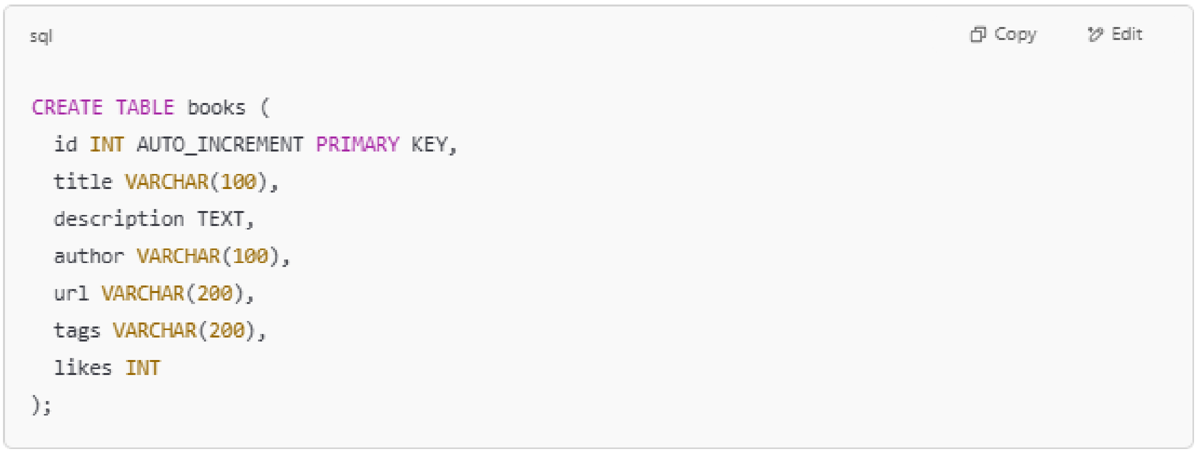
```sql
CREATE TABLE archive_record AS
SELECT * FROM Employee WHERE 1=0;

CREATE TRIGGER before_delete_emp
BEFORE DELETE ON Employee
FOR EACH ROW
INSERT INTO archive_record
VALUES (OLD.emp_no, OLD.name, OLD.skill, OLD.pay_rate);
```

CREATE TABLE archive_record AS
SELECT * FROM Employee WHERE 1=0;

CREATE TRIGGER before_delete_emp
BEFORE DELETE ON Employee
FOR EACH ROW
INSERT INTO archive_record
VALUES (OLD.emp_no, OLD.name, OLD.skill, OLD.pay_rate);

14. Implement the following Queries

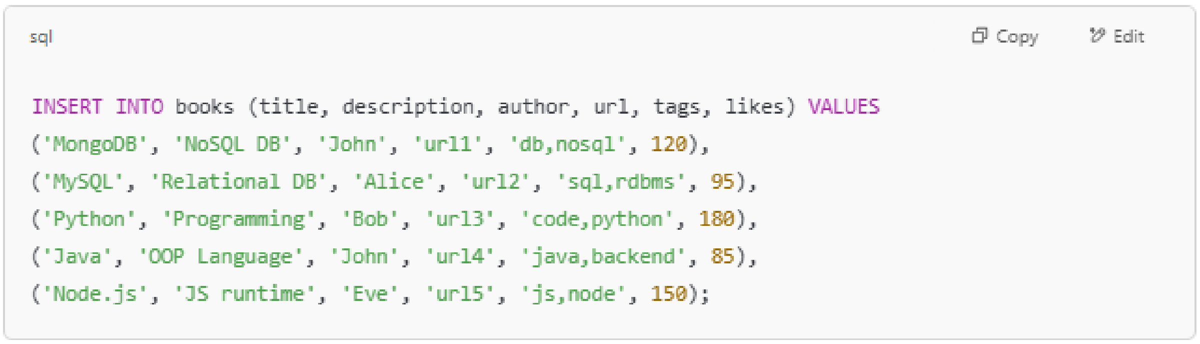
1. Create a collection named books.



```sql
CREATE TABLE books (
  id INT AUTO_INCREMENT PRIMARY KEY,
  title VARCHAR(100),
  description TEXT,
  author VARCHAR(100),
  url VARCHAR(200),
  tags VARCHAR(200),
  likes INT
);
```

```
CREATE TABLE books (
 id INT AUTO_INCREMENT PRIMARY KEY,
 title VARCHAR(100),
 description TEXT,
 author VARCHAR(100),
 url VARCHAR(200),
 tags VARCHAR(200),
 likes INT
);
```

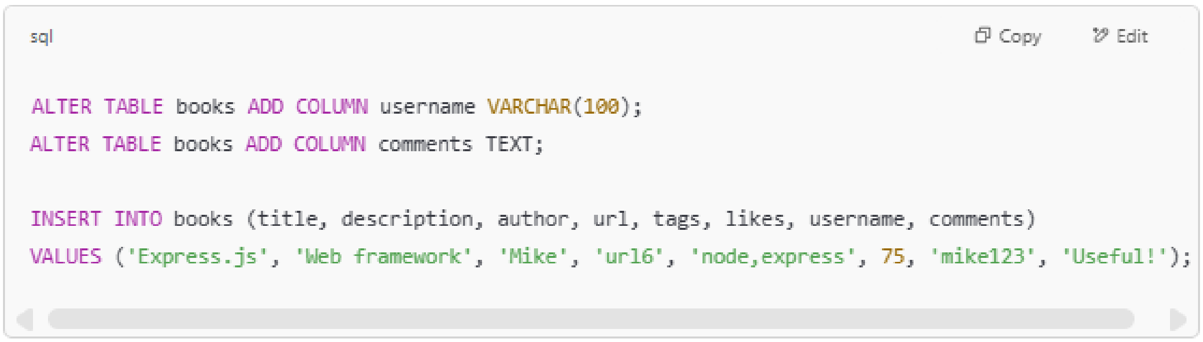
2. Insert 5 records with field TITLE, DESCRIPTION, BY, URL, TAGS AND LIKES



```sql
INSERT INTO books (title, description, author, url, tags, likes) VALUES
('MongoDB', 'NoSQL DB', 'John', 'url1', 'db,nosql', 120),
('MySQL', 'Relational DB', 'Alice', 'url2', 'sql,rdbms', 95),
('Python', 'Programming', 'Bob', 'url3', 'code,python', 180),
('Java', 'OOP Language', 'John', 'url4', 'java,backend', 85),
('Node.js', 'JS runtime', 'Eve', 'url5', 'js,node', 150);
```

```
INSERT INTO books (title, description, author, url, tags, likes) VALUES
('MongoDB', 'NoSQL DB', 'John', 'url1', 'db,nosql', 120),
('MySQL', 'Relational DB', 'Alice', 'url2', 'sql,rdbms', 95),
('Python', 'Programming', 'Bob', 'url3', 'code,python', 180),
('Java', 'OOP Language', 'John', 'url4', 'java,backend', 85),
('Node.js', 'JS runtime', 'Eve', 'url5', 'js,node', 150);
```

3. Insert 1 more document in collection with additional field of user name and comments.

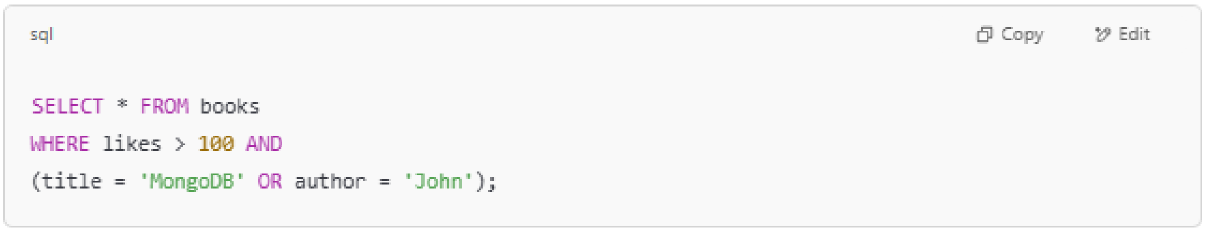


```sql
ALTER TABLE books ADD COLUMN username VARCHAR(100);
ALTER TABLE books ADD COLUMN comments TEXT;

INSERT INTO books (title, description, author, url, tags, likes, username, comments)
VALUES ('Express.js', 'Web framework', 'Mike', 'url6', 'node,express', 75, 'mike123', 'Useful!');
```

ALTER TABLE books ADD COLUMN username VARCHAR(100);
ALTER TABLE books ADD COLUMN comments TEXT;

INSERT INTO books (title, description, author, url, tags, likes, username, comments)
VALUES ('Express.js', 'Web framework', 'Mike', 'url6', 'node,express', 75, 'mike123',
'Useful!');

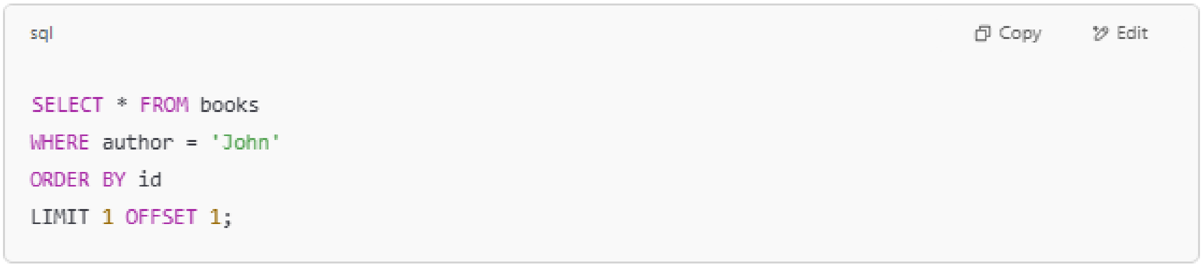
4. Display all the documents whose like is greater than 100 and whose title is either 'mongodb' or written by 'john'.



```sql
SELECT * FROM books
WHERE likes > 100 AND
(title = 'MongoDB' OR author = 'John');
```

SELECT * FROM books
WHERE likes > 100 AND
(title = 'MongoDB' OR author = 'John');

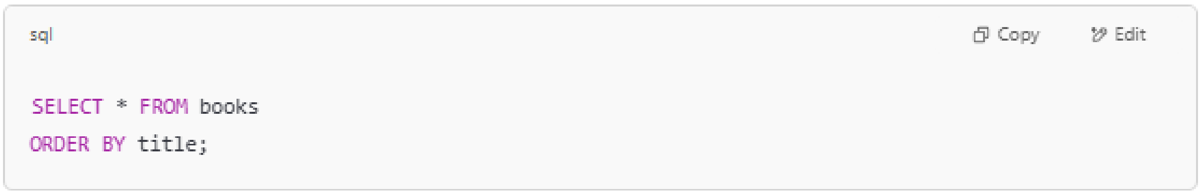
5. Display the second document published by 'john'.



```sql
SELECT * FROM books
WHERE author = 'John'
ORDER BY id
LIMIT 1 OFFSET 1;
```

SELECT * FROM books
WHERE author = 'John'
ORDER BY id
LIMIT 1 OFFSET 1;

6. Display all the books in the sorted fashion.

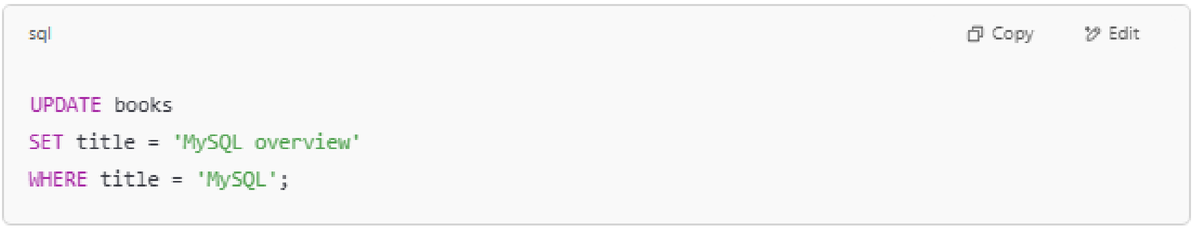


```sql
SELECT * FROM books
ORDER BY title;
```

SELECT * FROM books

ORDER BY title;

7. Update the title of 'MySQL' document to 'MySQL overview'

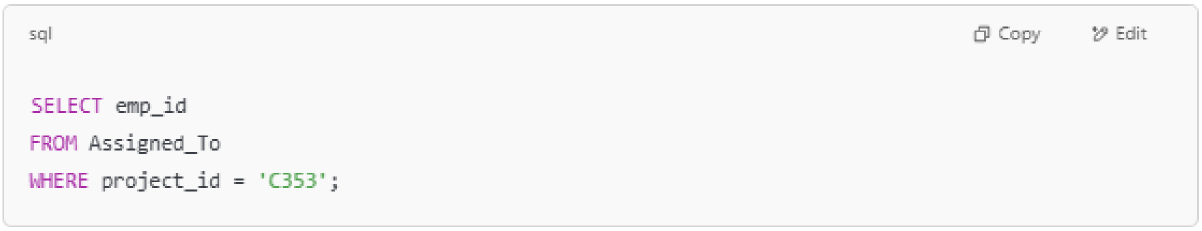


```sql
UPDATE books
SET title = 'MySQL overview'
WHERE title = 'MySQL';
```

UPDATE books
SET title = 'MySQL overview'
WHERE title = 'MySQL';

15. Consider the given database: Project (project_id, proj_name, chief_arch) Employee (emp_id, emp_name) Assigned_To (project_id ,emp_id) Find the SQL queries for the following statements:

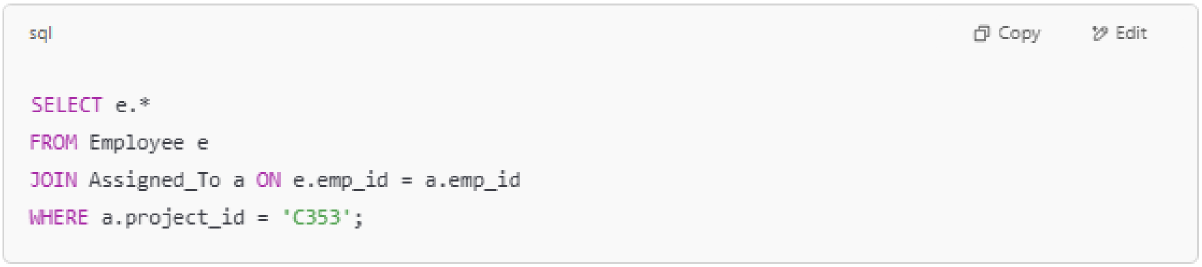
1. Get the employee number of employee working on project C353:



```sql
SELECT emp_id
FROM Assigned_To
WHERE project_id = 'C353';
```

SELECT emp_id
FROM Assigned_To
WHERE project_id = 'C353';

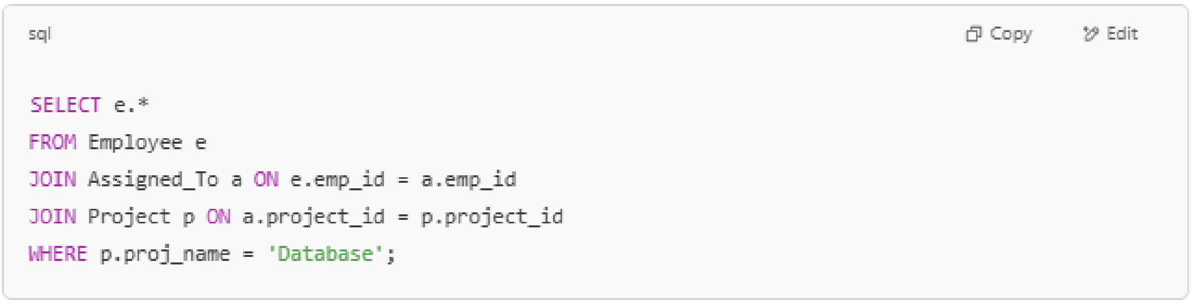
2. Get details of employees working on project C353:



```sql
SELECT e.*
FROM Employee e
JOIN Assigned_To a ON e.emp_id = a.emp_id
WHERE a.project_id = 'C353';
```

SELECT e.*
FROM Employee e
JOIN Assigned_To a ON e.emp_id = a.emp_id
WHERE a.project_id = 'C353';

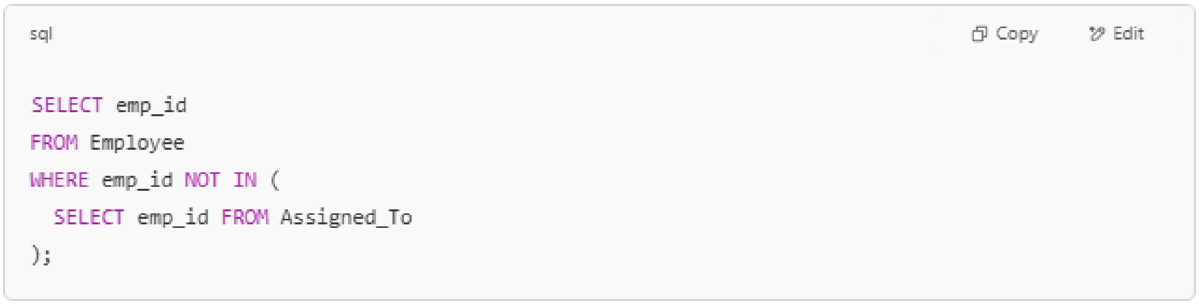
3. Obtain details of employees working on database project



```sql
SELECT e.*
FROM Employee e
JOIN Assigned_To a ON e.emp_id = a.emp_id
JOIN Project p ON a.project_id = p.project_id
WHERE p.proj_name = 'Database';
```

SELECT e.*
FROM Employee e
JOIN Assigned_To a ON e.emp_id = a.emp_id
JOIN Project p ON a.project_id = p.project_id
WHERE p.proj_name = 'Database';

4. Get the employee number of employees who are not on any project.

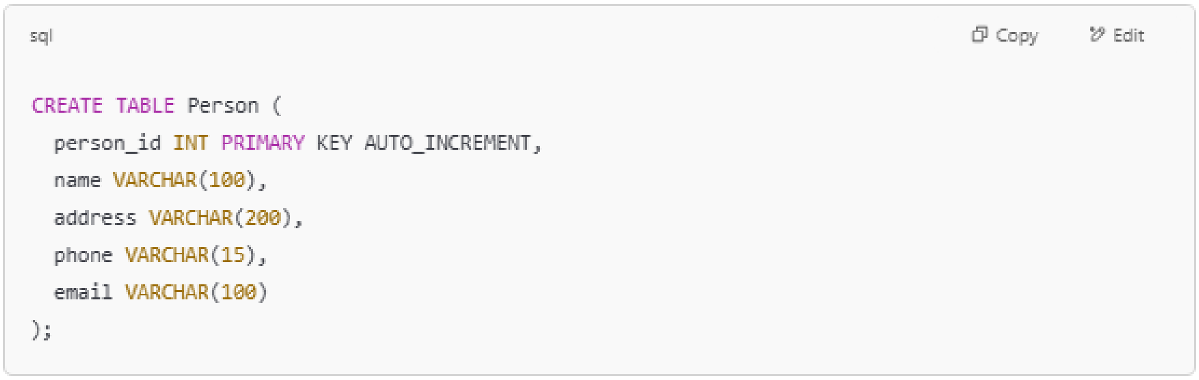


```sql
SELECT emp_id
FROM Employee
WHERE emp_id NOT IN (
  SELECT emp_id FROM Assigned_To
);
```

SELECT emp_id
FROM Employee
WHERE emp_id NOT IN (
 SELECT emp_id FROM Assigned_To
);

16. A insurance company wants to keep following information

1.Name, address, phone number and email of person.

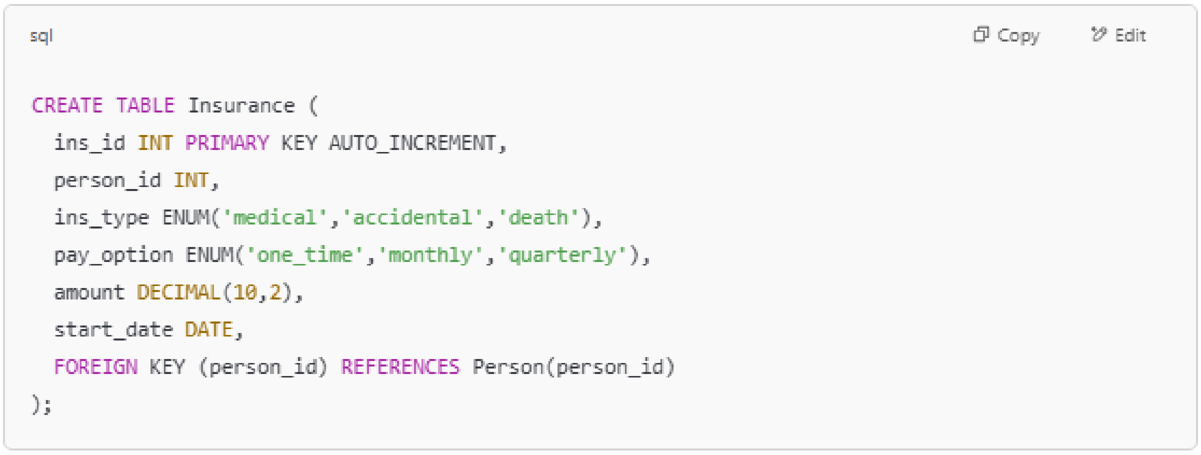


```sql
CREATE TABLE Person (
  person_id INT PRIMARY KEY AUTO_INCREMENT,
  name VARCHAR(100),
  address VARCHAR(200),
  phone VARCHAR(15),
  email VARCHAR(100)
);
```

CREATE TABLE Person (
 person_id INT PRIMARY KEY AUTO_INCREMENT,
 name VARCHAR(100),
 address VARCHAR(200),
 phone VARCHAR(15),
 email VARCHAR(100)

);

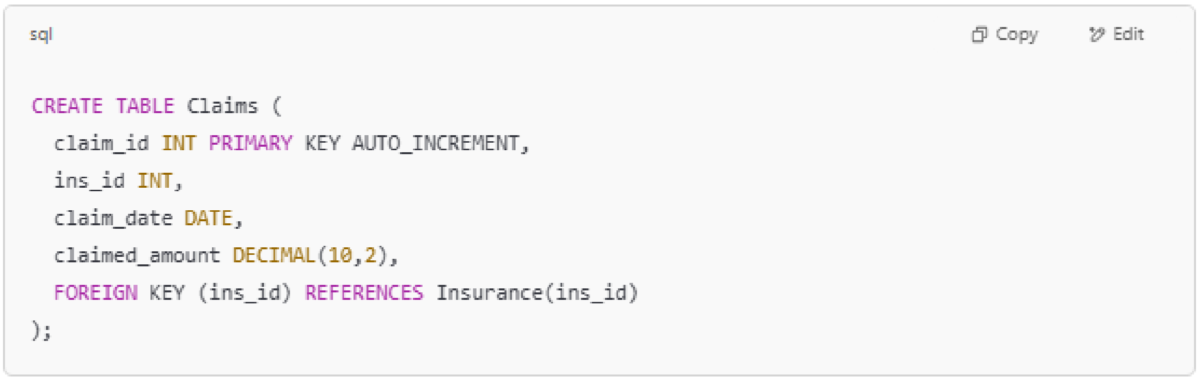
2.Type of insurance: medical, accidental, death.



```sql
CREATE TABLE Insurance (
  ins_id INT PRIMARY KEY AUTO_INCREMENT,
  person_id INT,
  ins_type ENUM('medical','accidental','death'),
  pay_option ENUM('one_time','monthly','quarterly'),
  amount DECIMAL(10,2),
  start_date DATE,
  FOREIGN KEY (person_id) REFERENCES Person(person_id)
);
```

```
CREATE TABLE Insurance (
  ins_id INT PRIMARY KEY AUTO_INCREMENT,
  person_id INT,
  ins_type ENUM('medical','accidental','death'),
  pay_option ENUM('one_time','monthly','quarterly'),
  amount DECIMAL(10,2),
  start_date DATE,
  FOREIGN KEY (person_id) REFERENCES Person(person_id)
);
```

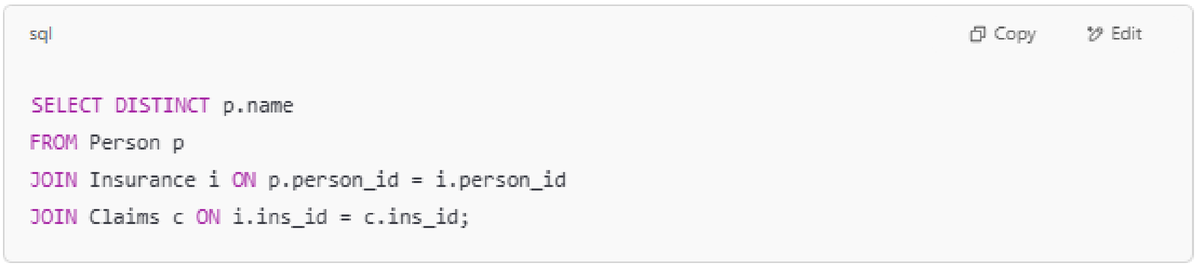
3. Insurance payment options one time, monthly, quarterly.



```sql
CREATE TABLE Claims (
  claim_id INT PRIMARY KEY AUTO_INCREMENT,
  ins_id INT,
  claim_date DATE,
  claimed_amount DECIMAL(10,2),
  FOREIGN KEY (ins_id) REFERENCES Insurance(ins_id)
);
```

```
CREATE TABLE Claims (
  claim_id INT PRIMARY KEY AUTO_INCREMENT,
  ins_id INT,
  claim_date DATE,
  claimed_amount DECIMAL(10,2),
  FOREIGN KEY (ins_id) REFERENCES Insurance(ins_id)
);
```

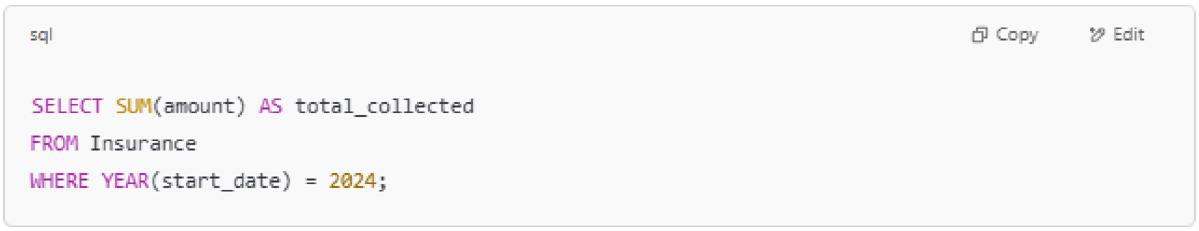
4.List of persons who claimed against their insurance.



```sql
SELECT DISTINCT p.name
FROM Person p
JOIN Insurance i ON p.person_id = i.person_id
JOIN Claims c ON i.ins_id = c.ins_id;
```

SELECT DISTINCT p.name
FROM Person p
JOIN Insurance i ON p.person_id = i.person_id
JOIN Claims c ON i.ins_id = c.ins_id;

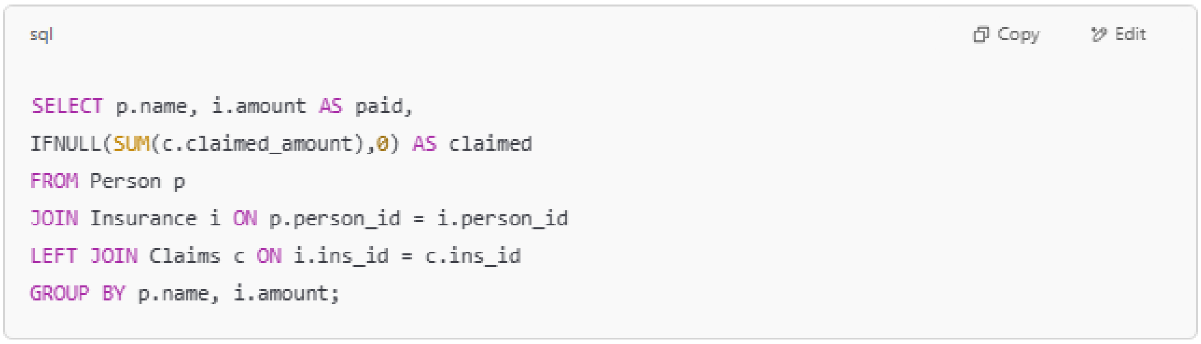
5.Amount collected in given period



```sql
SELECT SUM(amount) AS total_collected
FROM Insurance
WHERE YEAR(start_date) = 2024;
```

SELECT SUM(amount) AS total_collected
FROM Insurance
WHERE YEAR(start_date) = 2024;

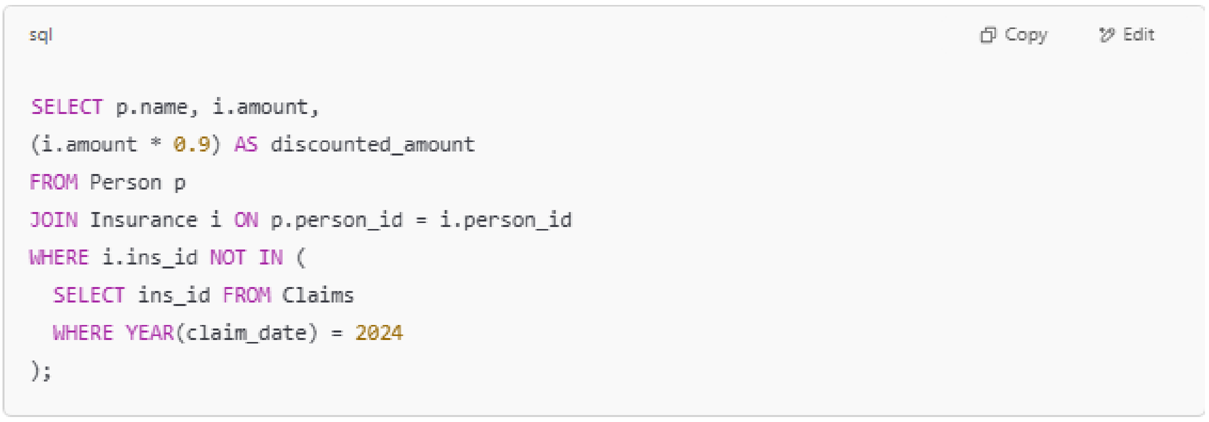
6.Name of person, amount paid as insurance, amount claimed against insurance.



```sql
SELECT p.name, i.amount AS paid,
IFNULL(SUM(c.claimed_amount),0) AS claimed
FROM Person p
JOIN Insurance i ON p.person_id = i.person_id
LEFT JOIN Claims c ON i.ins_id = c.ins_id
GROUP BY p.name, i.amount;
```

SELECT p.name, i.amount AS paid,
IFNULL(SUM(c.claimed_amount),0) AS claimed
FROM Person p
JOIN Insurance i ON p.person_id = i.person_id
LEFT JOIN Claims c ON i.ins_id = c.ins_id
GROUP BY p.name, i.amount;

7.If a person has not claimed during the year then give 10% discount on insurance amount during next year.

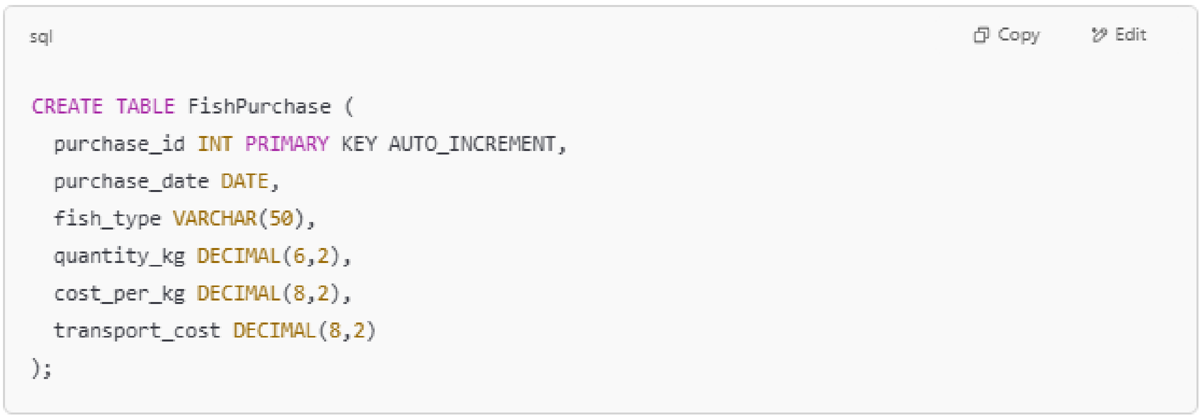


```sql
SELECT p.name, i.amount,
(i.amount * 0.9) AS discounted_amount
FROM Person p
JOIN Insurance i ON p.person_id = i.person_id
WHERE i.ins_id NOT IN (
  SELECT ins_id FROM Claims
  WHERE YEAR(claim_date) = 2024
);
```

```
SELECT p.name, i.amount,
(i.amount * 0.9) AS discounted_amount
FROM Person p
JOIN Insurance i ON p.person_id = i.person_id
WHERE i.ins_id NOT IN (
 SELECT ins_id FROM Claims
 WHERE YEAR(claim_date) = 2024
);
```

17. A fish shop is selling approximately 100 Kg fish daily. Shop is having 10 types of different fishes in the shop. Cost of each fish is different and it is per Kg. Shop man wants to keep the following record.

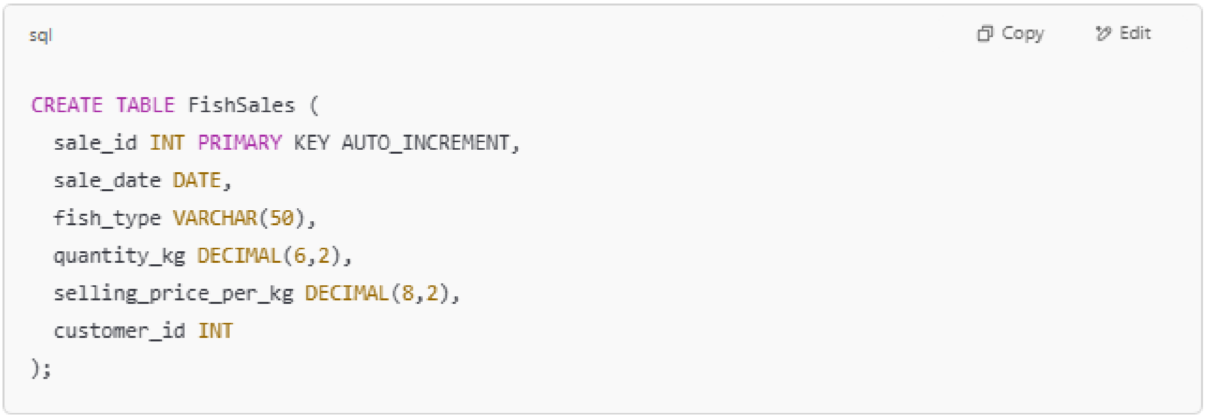
1.Quantity of fish purchase each day, cost of purchase, transportation cost.



```sql
CREATE TABLE FishPurchase (
  purchase_id INT PRIMARY KEY AUTO_INCREMENT,
  purchase_date DATE,
  fish_type VARCHAR(50),
  quantity_kg DECIMAL(6,2),
  cost_per_kg DECIMAL(8,2),
  transport_cost DECIMAL(8,2)
);
```

```
CREATE TABLE FishPurchase (
 purchase_id INT PRIMARY KEY AUTO_INCREMENT,
 purchase_date DATE,
 fish_type VARCHAR(50),
 quantity_kg DECIMAL(6,2),
 cost_per_kg DECIMAL(8,2),
 transport_cost DECIMAL(8,2)
);
```

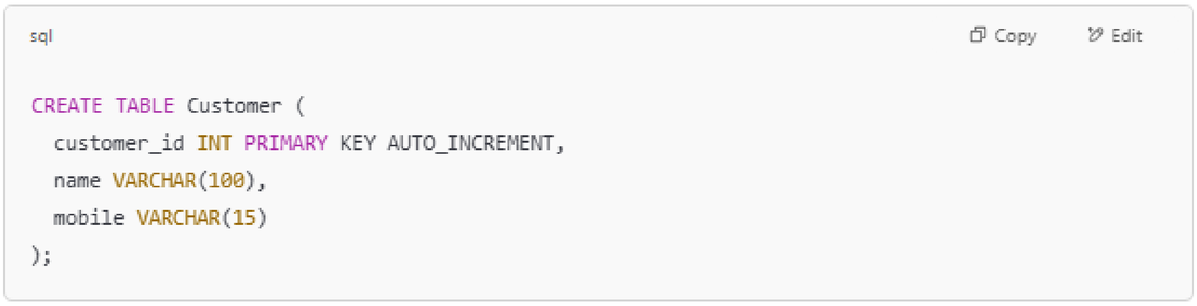
2.Quantity of fishes sold per day and selling price



```sql
CREATE TABLE FishSales (
  sale_id INT PRIMARY KEY AUTO_INCREMENT,
  sale_date DATE,
  fish_type VARCHAR(50),
  quantity_kg DECIMAL(6,2),
  selling_price_per_kg DECIMAL(8,2),
  customer_id INT
);
```

```
CREATE TABLE FishSales (
 sale_id INT PRIMARY KEY AUTO_INCREMENT,
 sale_date DATE,
 fish_type VARCHAR(50),
 quantity_kg DECIMAL(6,2),
 selling_price_per_kg DECIMAL(8,2),
 customer_id INT
);
```

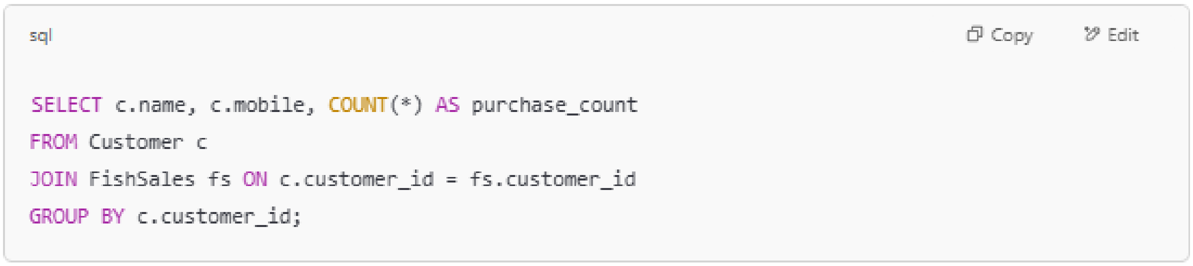
3.Name and mobile number of person purchasing fish and types of fishes he purchases



```sql
CREATE TABLE Customer (
  customer_id INT PRIMARY KEY AUTO_INCREMENT,
  name VARCHAR(100),
  mobile VARCHAR(15)
);
```

```
CREATE TABLE Customer (
 customer_id INT PRIMARY KEY AUTO_INCREMENT,
 name VARCHAR(100),
 mobile VARCHAR(15)
);
```

4.Find frequency of a person purchasing fishes

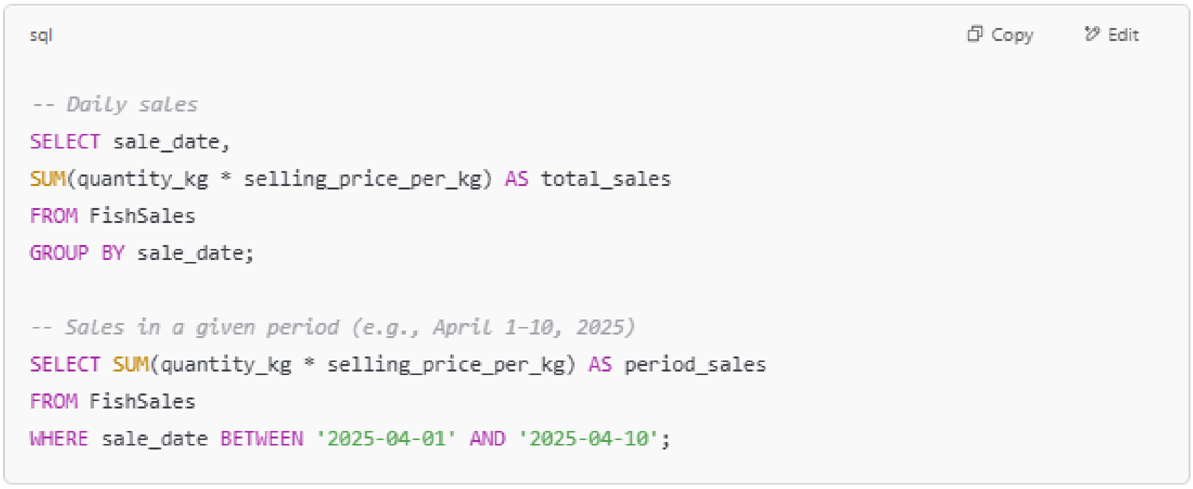


```sql
SELECT c.name, c.mobile, COUNT(*) AS purchase_count
FROM Customer c
JOIN FishSales fs ON c.customer_id = fs.customer_id
GROUP BY c.customer_id;
```

```
SELECT c.name, c.mobile, COUNT(*) AS purchase_count
FROM Customer c
JOIN FishSales fs ON c.customer_id = fs.customer_id
```

GROUP BY c.customer_id;

5.Find daily sales amount and sales amount between given period



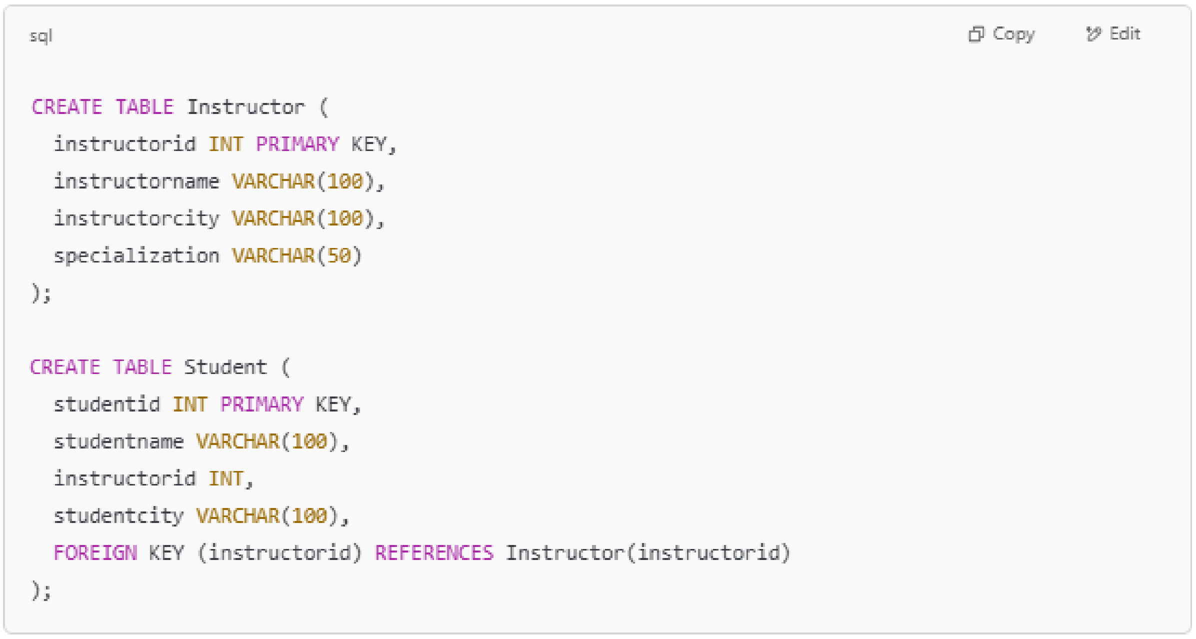
```sql
-- Daily sales
SELECT sale_date,
SUM(quantity_kg * selling_price_per_kg) AS total_sales
FROM FishSales
GROUP BY sale_date;

-- Sales in a given period (e.g., April 1-10, 2025)
SELECT SUM(quantity_kg * selling_price_per_kg) AS period_sales
FROM FishSales
WHERE sale_date BETWEEN '2025-04-01' AND '2025-04-10';
```

-- Daily sales
SELECT sale_date,
SUM(quantity_kg * selling_price_per_kg) AS total_sales
FROM FishSales
GROUP BY sale_date;

-- Sales in a given period (e.g., April 1–10, 2025)
SELECT SUM(quantity_kg * selling_price_per_kg) AS period_sales
FROM FishSales
WHERE sale_date BETWEEN '2025-04-01' AND '2025-04-10';

18. Consider the given database schema: Student (studentid , studentname,instructorid,studentcity) Instructor (instructorid,Instructorname,instructor city,specialization) Use all types of Joins and set operation

1. Add primary and foreign keys

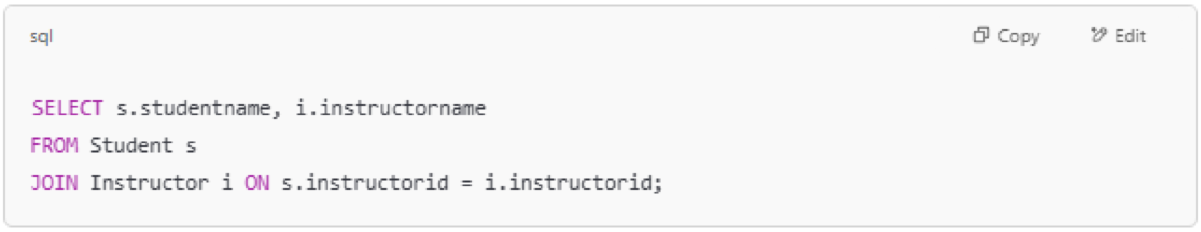


```sql
CREATE TABLE Instructor (
  instructorid INT PRIMARY KEY,
  instructorname VARCHAR(100),
  instructorcity VARCHAR(100),
  specialization VARCHAR(50)
);

CREATE TABLE Student (
  studentid INT PRIMARY KEY,
  studentname VARCHAR(100),
  instructorid INT,
  studentcity VARCHAR(100),
  FOREIGN KEY (instructorid) REFERENCES Instructor(instructorid)
);
```

CREATE TABLE Instructor (
 instructorid INT PRIMARY KEY,
 instructorname VARCHAR(100),
 instructorcity VARCHAR(100),
 specialization VARCHAR(50)
);

CREATE TABLE Student (
 studentid INT PRIMARY KEY,
 studentname VARCHAR(100),
 instructorid INT,
 studentcity VARCHAR(100),
 FOREIGN KEY (instructorid) REFERENCES Instructor(instructorid)
);

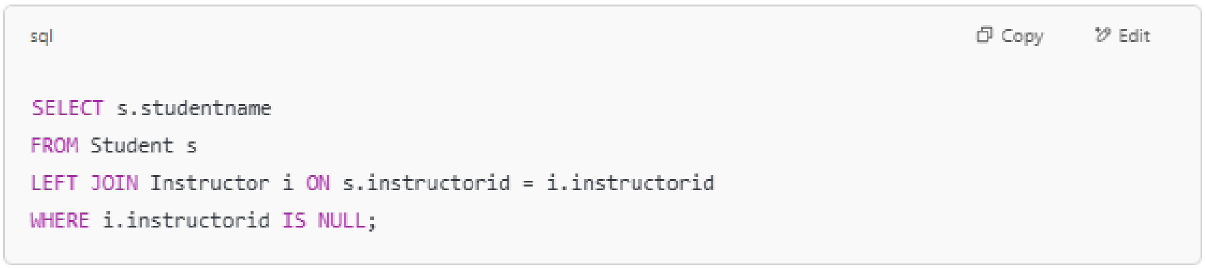
2. Find the instructor of each student.



```sql
SELECT s.studentname, i.instructorname
FROM Student s
JOIN Instructor i ON s.instructorid = i.instructorid;
```

SELECT s.studentname, i.instructorname
FROM Student s
JOIN Instructor i ON s.instructorid = i.instructorid;

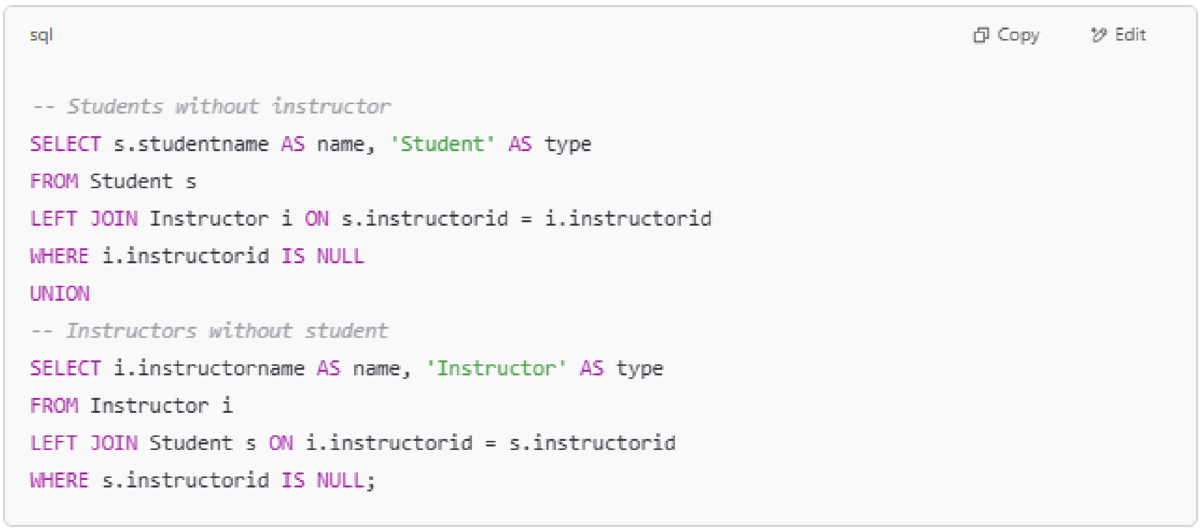
3. Find the student who is not having any instructor.



```sql
SELECT s.studentname
FROM Student s
LEFT JOIN Instructor i ON s.instructorid = i.instructorid
WHERE i.instructorid IS NULL;
```

SELECT s.studentname
FROM Student s
LEFT JOIN Instructor i ON s.instructorid = i.instructorid
WHERE i.instructorid IS NULL;

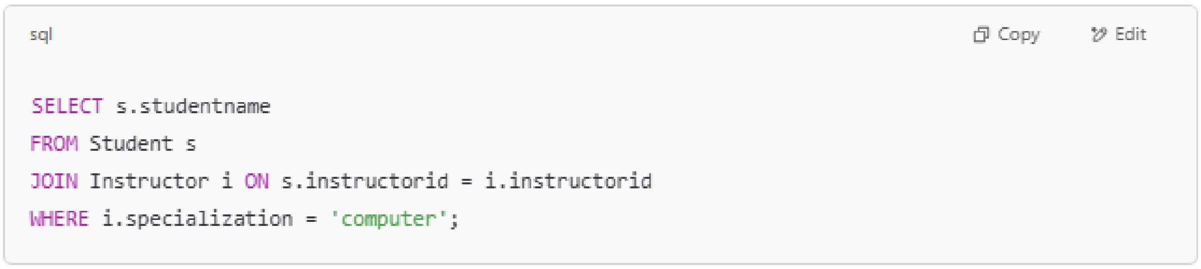
4. Find the student who is not having any instructor as well as the instructor who is not having a student.



```sql
-- Students without instructor
SELECT s.studentname AS name, 'Student' AS type
FROM Student s
LEFT JOIN Instructor i ON s.instructorid = i.instructorid
WHERE i.instructorid IS NULL
UNION
-- Instructors without student
SELECT i.instructorname AS name, 'Instructor' AS type
FROM Instructor i
LEFT JOIN Student s ON i.instructorid = s.instructorid
WHERE s.instructorid IS NULL;
```

-- Students without instructor
SELECT s.studentname AS name, 'Student' AS type
FROM Student s
LEFT JOIN Instructor i ON s.instructorid = i.instructorid
WHERE i.instructorid IS NULL
UNION
-- Instructors without student
SELECT i.instructorname AS name, 'Instructor' AS type
FROM Instructor i
LEFT JOIN Student s ON i.instructorid = s.instructorid
WHERE s.instructorid IS NULL;

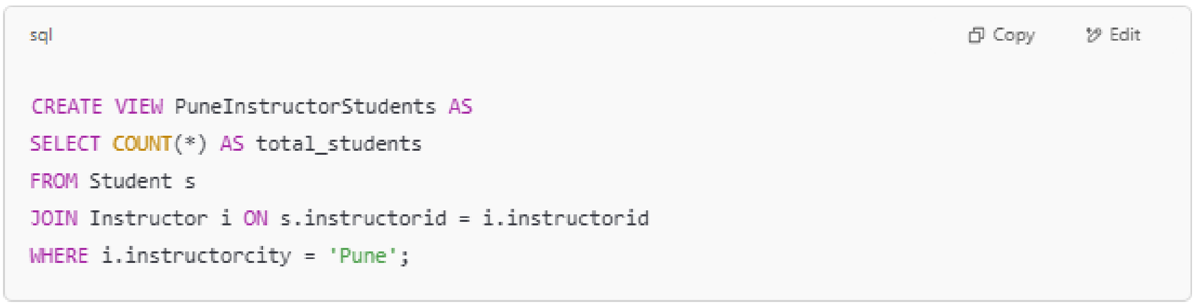
5. Find the students whose instructor’s specialization is computer.



```sql
SELECT s.studentname
FROM Student s
JOIN Instructor i ON s.instructorid = i.instructorid
WHERE i.specialization = 'computer';
```

SELECT s.studentname
FROM Student s
JOIN Instructor i ON s.instructorid = i.instructorid
WHERE i.specialization = 'computer';

6. Create a view containing the total number of students whose instructor belongs to “Pune”.



```sql
CREATE VIEW PuneInstructorStudents AS
SELECT COUNT(*) AS total_students
FROM Student s
JOIN Instructor i ON s.instructorid = i.instructorid
WHERE i.instructorcity = 'Pune';
```

CREATE VIEW PuneInstructorStudents AS
SELECT COUNT(*) AS total_students
FROM Student s
JOIN Instructor i ON s.instructorid = i.instructorid
WHERE i.instructorcity = 'Pune';

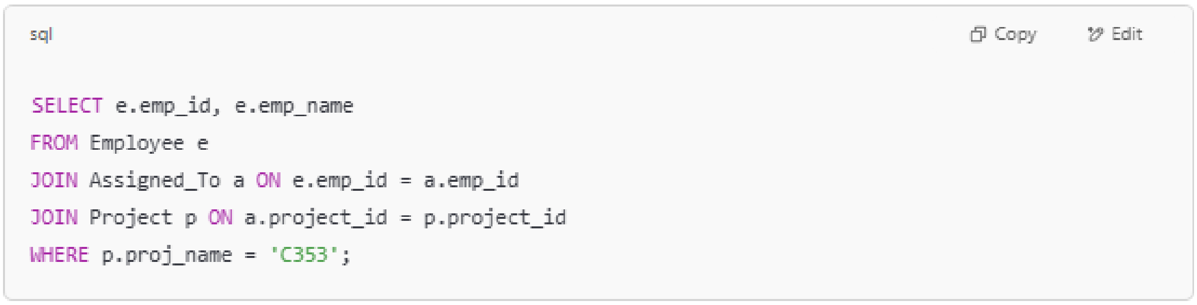
19. Consider the given database: Project (project_id, proj_name, chief_arch) Employee (emp_id, emp_name) Assigned_To (project_id ,emp_id) Find the SQL queries for the following statements:

1. Get the employee number of employee working on project C353:

```sql
SELECT e.emp_id
FROM Employee e
JOIN Assigned_To a ON e.emp_id = a.emp_id
JOIN Project p ON a.project_id = p.project_id
WHERE p.proj_name = 'C353';
```

SELECT e.emp_id
FROM Employee e
JOIN Assigned_To a ON e.emp_id = a.emp_id
JOIN Project p ON a.project_id = p.project_id
WHERE p.proj_name = 'C353';

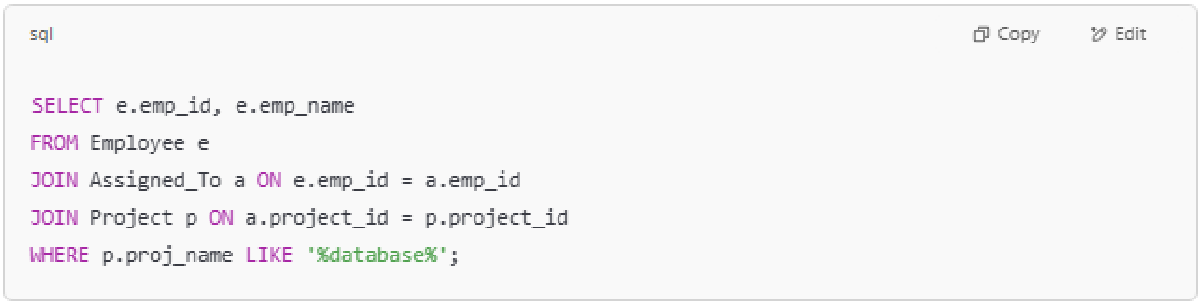
2. Get details of employees working on project C353:



```sql
SELECT e.emp_id, e.emp_name
FROM Employee e
JOIN Assigned_To a ON e.emp_id = a.emp_id
JOIN Project p ON a.project_id = p.project_id
WHERE p.proj_name = 'C353';
```

SELECT e.emp_id, e.emp_name
FROM Employee e
JOIN Assigned_To a ON e.emp_id = a.emp_id
JOIN Project p ON a.project_id = p.project_id
WHERE p.proj_name = 'C353';

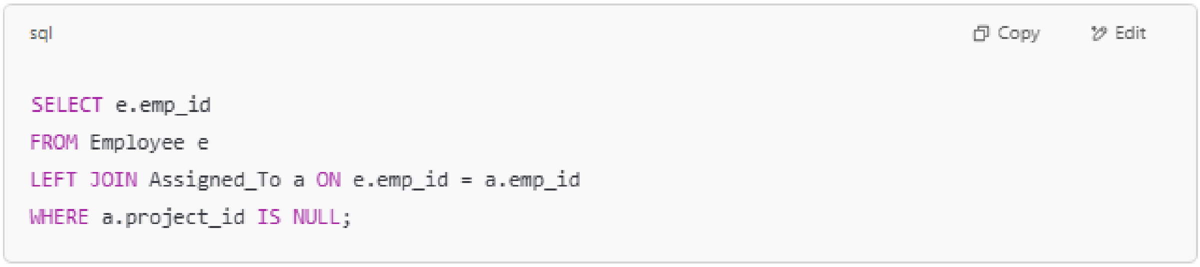
3. Obtain details of employees working on database project



```sql
SELECT e.emp_id, e.emp_name
FROM Employee e
JOIN Assigned_To a ON e.emp_id = a.emp_id
JOIN Project p ON a.project_id = p.project_id
WHERE p.proj_name LIKE '%database%';
```

SELECT e.emp_id, e.emp_name
FROM Employee e
JOIN Assigned_To a ON e.emp_id = a.emp_id
JOIN Project p ON a.project_id = p.project_id
WHERE p.proj_name LIKE '%database%';

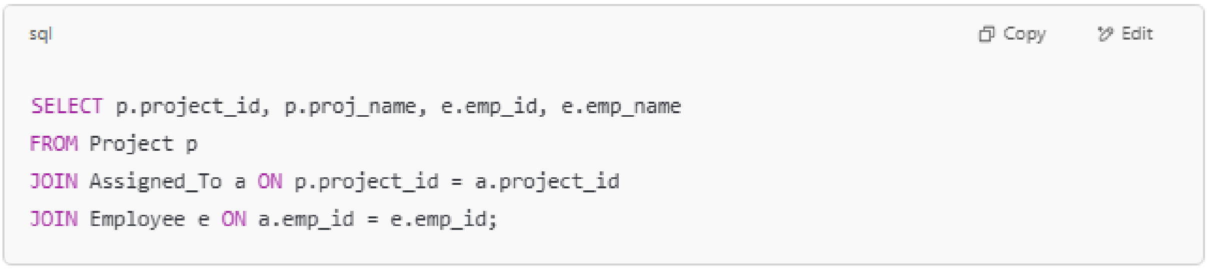
4. Get the employee number of employees who are not on any project.



```sql
SELECT e.emp_id
FROM Employee e
LEFT JOIN Assigned_To a ON e.emp_id = a.emp_id
WHERE a.project_id IS NULL;
```

SELECT e.emp_id
FROM Employee e
LEFT JOIN Assigned_To a ON e.emp_id = a.emp_id
WHERE a.project_id IS NULL;

5. Display Project_id,Proj_name,emp_id and emp_name

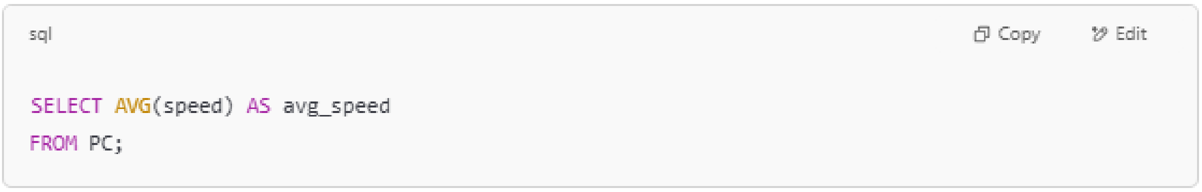


```sql
SELECT p.project_id, p.proj_name, e.emp_id, e.emp_name
FROM Project p
JOIN Assigned_To a ON p.project_id = a.project_id
JOIN Employee e ON a.emp_id = e.emp_id;
```

SELECT p.project_id, p.proj_name, e.emp_id, e.emp_name
FROM Project p
JOIN Assigned_To a ON p.project_id = a.project_id
JOIN Employee e ON a.emp_id = e.emp_id;

20. Write the following queries in SQL. Product (maker, model, type) PC(mode1, speed, ram, hd, rd, price) Laptop(mode1, speed, ram, hd, screen, price) Printer(mode1, color, type, price)

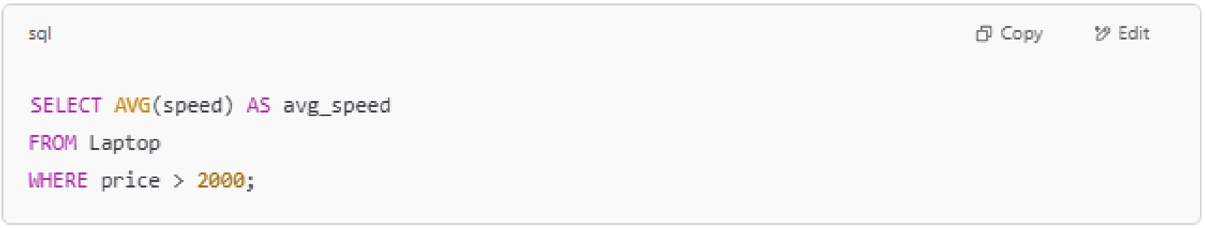
1. Find the average speed of PC's.



```sql
SELECT AVG(speed) AS avg_speed
FROM PC;
```

SELECT AVG(speed) AS avg_speed
FROM PC;

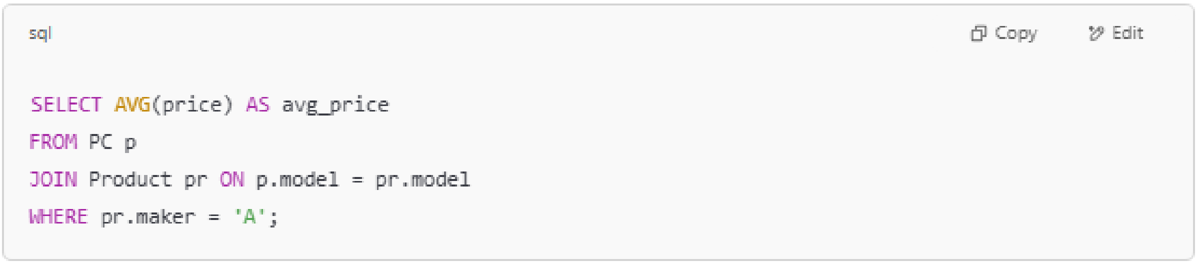
2. Find the average speed of laptops costing over Rs. 2000.



```sql
SELECT AVG(speed) AS avg_speed
FROM Laptop
WHERE price > 2000;
```

SELECT AVG(speed) AS avg_speed
FROM Laptop
WHERE price > 2000;

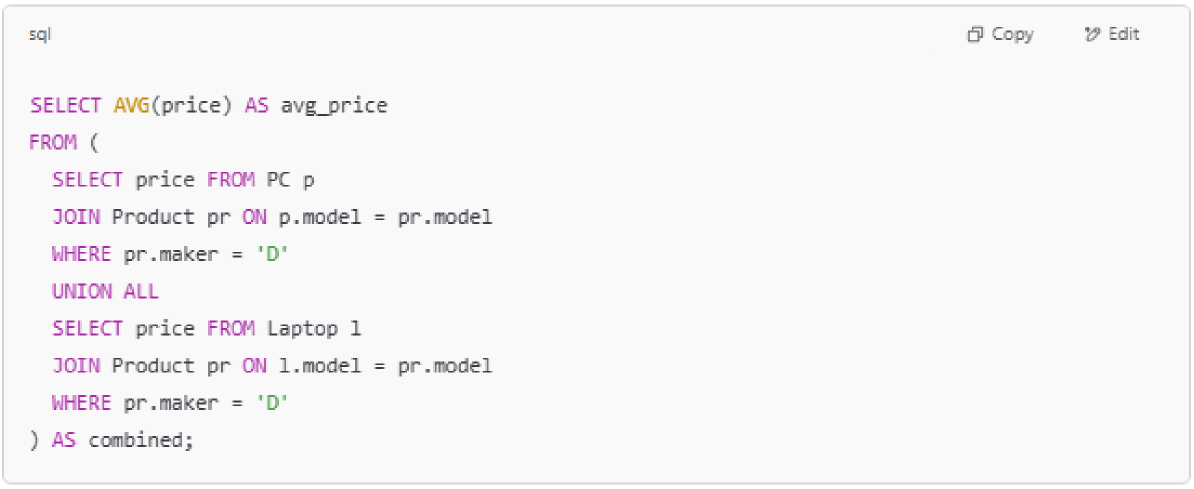
3. Find the average price of PC's made by manufacturer A.



```sql
SELECT AVG(price) AS avg_price
FROM PC p
JOIN Product pr ON p.model = pr.model
WHERE pr.maker = 'A';
```

SELECT AVG(price) AS avg_price
FROM PC p
JOIN Product pr ON p.model = pr.model
WHERE pr.maker = ‘A’

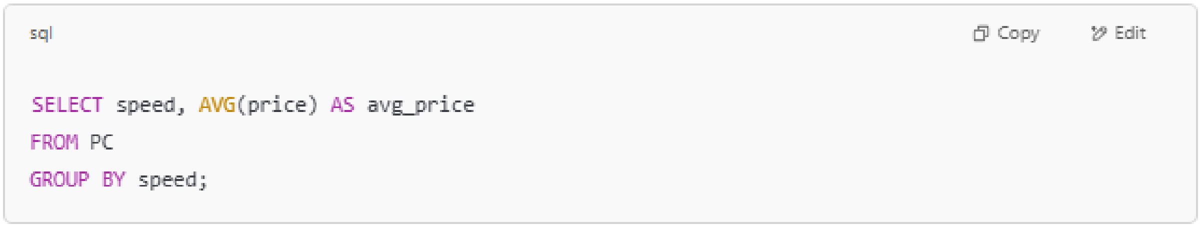
4. Find the average price of PC's and laptops made by manufacturer D.



```sql
SELECT AVG(price) AS avg_price
FROM (
  SELECT price FROM PC p
  JOIN Product pr ON p.model = pr.model
  WHERE pr.maker = 'D'
  UNION ALL
  SELECT price FROM Laptop l
  JOIN Product pr ON l.model = pr.model
  WHERE pr.maker = 'D'
) AS combined;
```

SELECT AVG(price) AS avg_price
FROM (
 SELECT price FROM PC p
 JOIN Product pr ON p.model = pr.model
 WHERE pr.maker = 'D'
 UNION ALL
 SELECT price FROM Laptop l
 JOIN Product pr ON l.model = pr.model
 WHERE pr.maker = 'D'
) AS combined;

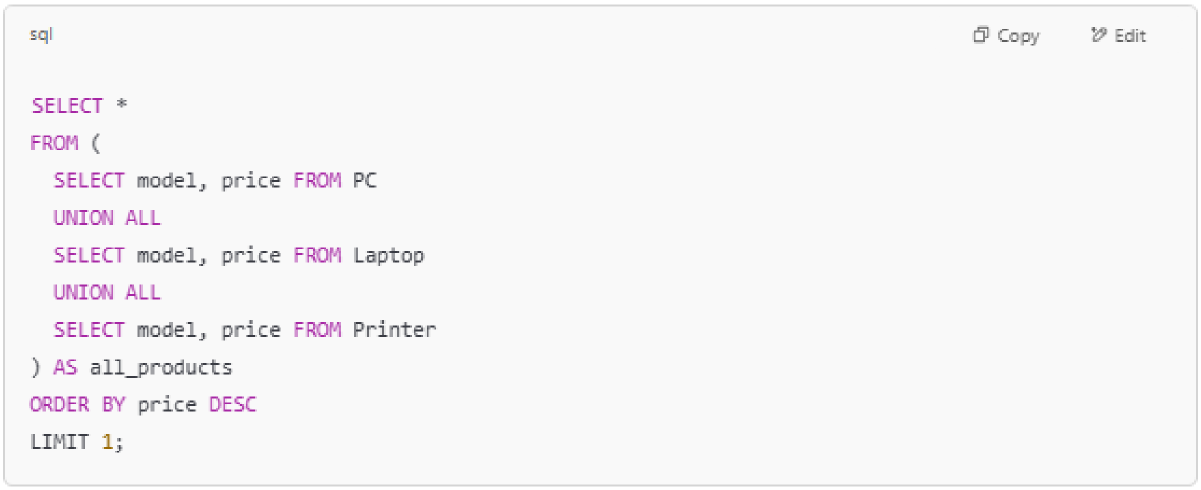
5. Find, for each different speed the average price of a PC.



```sql
SELECT speed, AVG(price) AS avg_price
FROM PC
GROUP BY speed;
```

SELECT speed, AVG(price) AS avg_price
FROM PC
GROUP BY speed;

6. Find the product with the highest price.

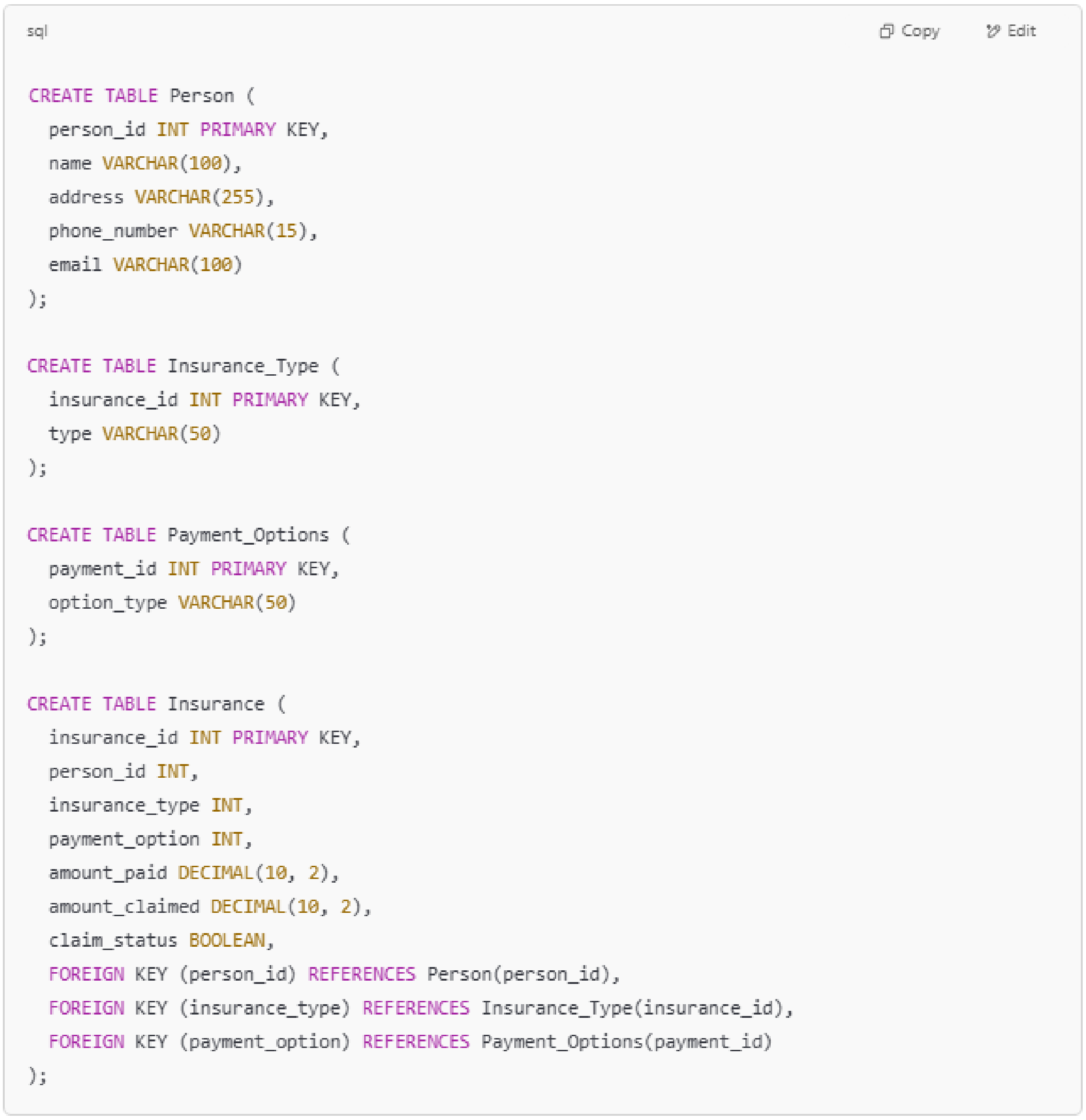


```sql
SELECT *
FROM (
  SELECT model, price FROM PC
  UNION ALL
  SELECT model, price FROM Laptop
  UNION ALL
  SELECT model, price FROM Printer
) AS all_products
ORDER BY price DESC
LIMIT 1;
```

```
SELECT *
FROM (
 SELECT model, price FROM PC
 UNION ALL
 SELECT model, price FROM Laptop
 UNION ALL
 SELECT model, price FROM Printer
) AS all_products
ORDER BY price DESC
LIMIT 1;
```

21. An insurance company wants to keep following information

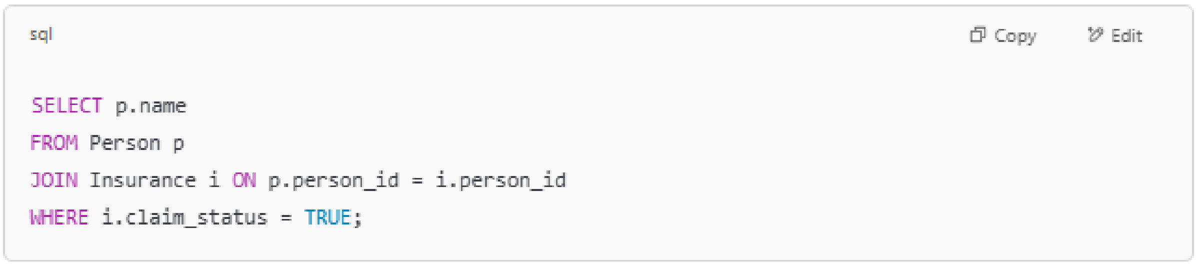
1.Name, address, phone number and email of person.



```sql
CREATE TABLE Person (
  person_id INT PRIMARY KEY,
  name VARCHAR(100),
  address VARCHAR(255),
  phone_number VARCHAR(15),
  email VARCHAR(100)
);

CREATE TABLE Insurance_Type (
  insurance_id INT PRIMARY KEY,
  type VARCHAR(50)
);

CREATE TABLE Payment_Options (
  payment_id INT PRIMARY KEY,
  option_type VARCHAR(50)
);

CREATE TABLE Insurance (
  insurance_id INT PRIMARY KEY,
  person_id INT,
  insurance_type INT,
  payment_option INT,
  amount_paid DECIMAL(10, 2),
  amount_claimed DECIMAL(10, 2),
  claim_status BOOLEAN,
  FOREIGN KEY (person_id) REFERENCES Person(person_id),
  FOREIGN KEY (insurance_type) REFERENCES Insurance_Type(insurance_id),
  FOREIGN KEY (payment_option) REFERENCES Payment_Options(payment_id)
);
```

```
CREATE TABLE Person (
 person_id INT PRIMARY KEY,
 name VARCHAR(100),
 address VARCHAR(255),
 phone_number VARCHAR(15),
 email VARCHAR(100)
);

CREATE TABLE Insurance_Type (
 insurance_id INT PRIMARY KEY,
 type VARCHAR(50)
);

CREATE TABLE Payment_Options (
 payment_id INT PRIMARY KEY,
 option_type VARCHAR(50)
);

CREATE TABLE Insurance (
 insurance_id INT PRIMARY KEY,
 person_id INT,
 insurance_type INT,
 payment_option INT,
 amount_paid DECIMAL(10, 2),
 amount_claimed DECIMAL(10, 2),
 claim_status BOOLEAN,
 FOREIGN KEY (person_id) REFERENCES Person(person_id),
 FOREIGN KEY (insurance_type) REFERENCES Insurance_Type(insurance_id),
 FOREIGN KEY (payment_option) REFERENCES Payment_Options(payment_id)
);
```

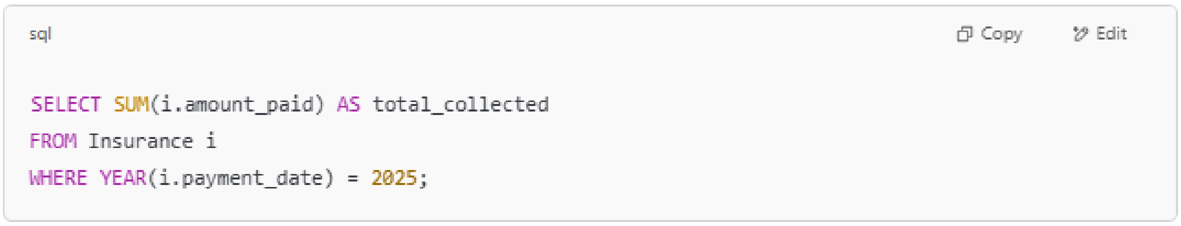
2.Type of insurance medical, accidental, death.



```sql
SELECT p.name
FROM Person p
JOIN Insurance i ON p.person_id = i.person_id
WHERE i.claim_status = TRUE;
```

```
SELECT p.name
FROM Person p
JOIN Insurance i ON p.person_id = i.person_id
WHERE i.claim_status = TRUE;
```

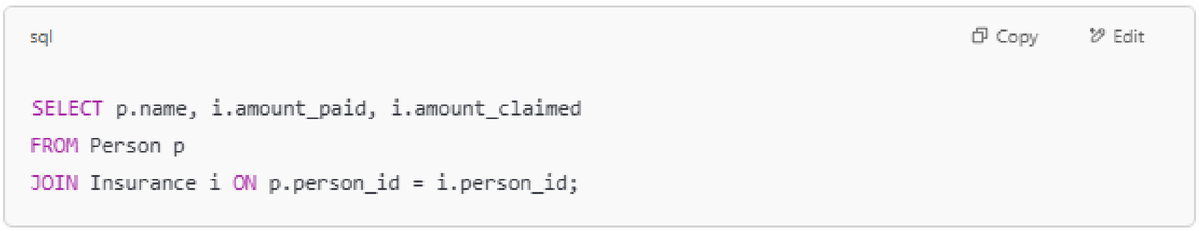
3.Insurance payment options one time, monthly, quarterly.



```sql
SELECT SUM(i.amount_paid) AS total_collected
FROM Insurance i
WHERE YEAR(i.payment_date) = 2025;
```

```
SELECT p.name
FROM Person p
JOIN Insurance i ON p.person_id = i.person_id
WHERE i.claim_status = TRUE;
```

4.List of persons who claimed against their insurance.

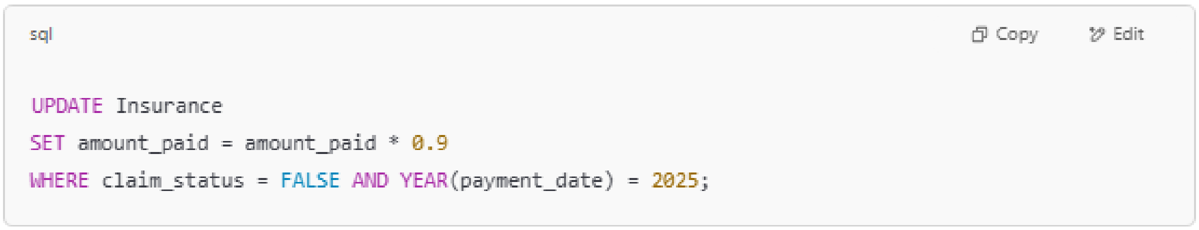


```sql
SELECT p.name, i.amount_paid, i.amount_claimed
FROM Person p
JOIN Insurance i ON p.person_id = i.person_id;
```

```
SELECT p.name, i.amount_paid, i.amount_claimed
```

FROM Person p
JOIN Insurance i ON p.person_id = i.person_id;

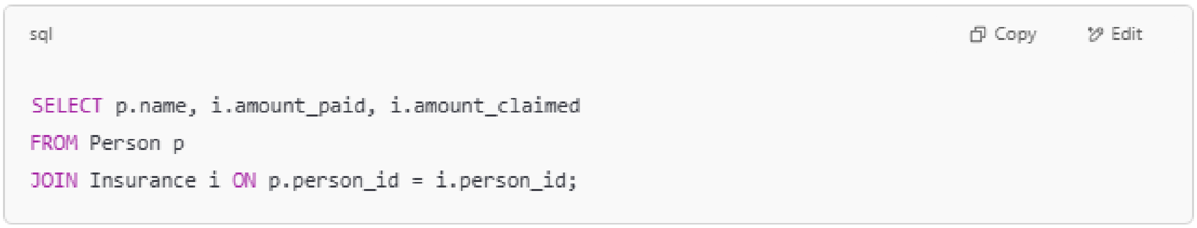
5.Amount collected in given period



```sql
UPDATE Insurance
SET amount_paid = amount_paid * 0.9
WHERE claim_status = FALSE AND YEAR(payment_date) = 2025;
```

UPDATE Insurance
SET amount_paid = amount_paid * 0.9
WHERE claim_status = FALSE AND YEAR(payment_date) = 2025;

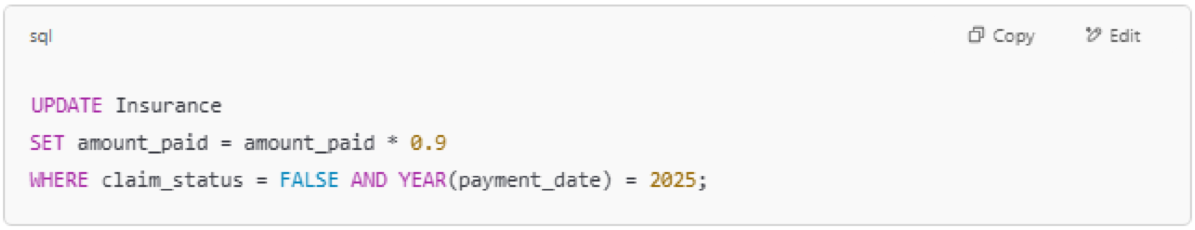
6.Name of person, amount paid as insurance, amount claimed against insurance.



```sql
SELECT p.name, i.amount_paid, i.amount_claimed
FROM Person p
JOIN Insurance i ON p.person_id = i.person_id;
```

SELECT p.name, i.amount_paid, i.amount_claimed
FROM Person p
JOIN Insurance i ON p.person_id = i.person_id;

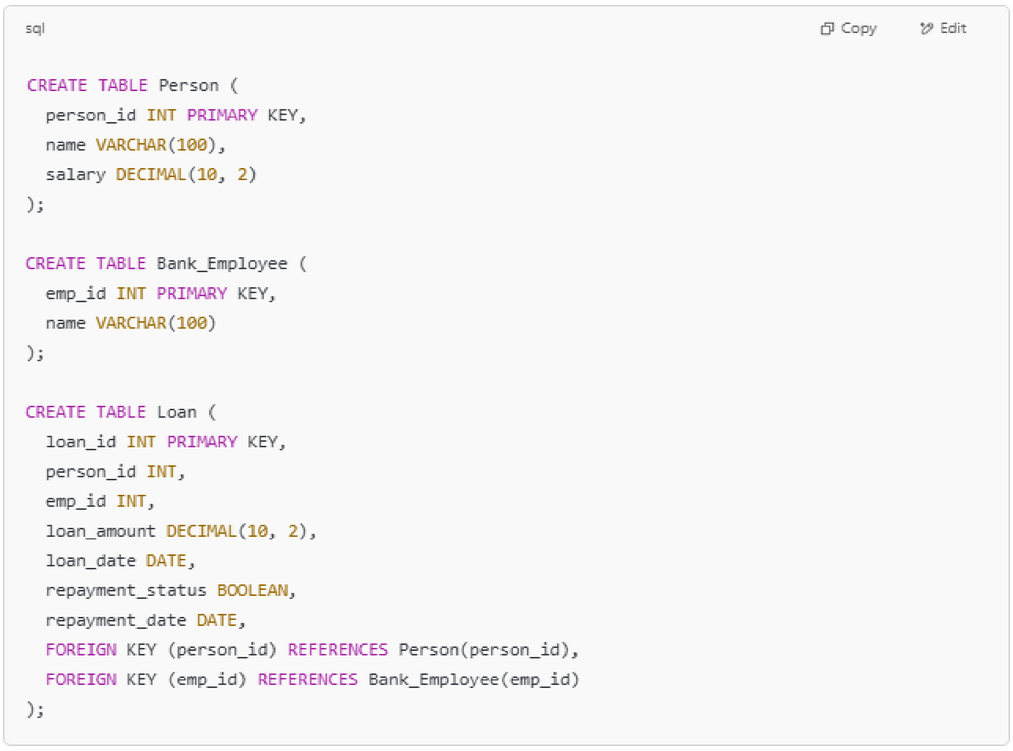
7.If a person has not claimed during the year then give 10% discount on insurance amount during next year.



```sql
UPDATE Insurance
SET amount_paid = amount_paid * 0.9
WHERE claim_status = FALSE AND YEAR(payment_date) = 2025;
```

UPDATE Insurance
SET amount_paid = amount_paid * 0.9
WHERE claim_status = FALSE AND YEAR(payment_date) = 2025;

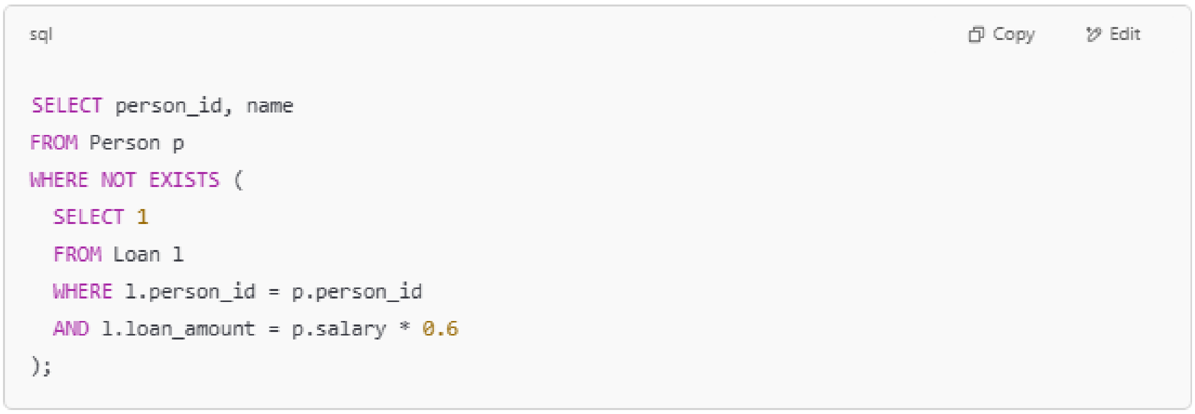
22. A bank gives loans to people for their home and vehicle wants to keep following record A person is eligible for a loan amounting 60% of his salary.



```sql
CREATE TABLE Person (
  person_id INT PRIMARY KEY,
  name VARCHAR(100),
  salary DECIMAL(10, 2)
);

CREATE TABLE Bank_Employee (
  emp_id INT PRIMARY KEY,
  name VARCHAR(100)
);

CREATE TABLE Loan (
  loan_id INT PRIMARY KEY,
  person_id INT,
  emp_id INT,
  loan_amount DECIMAL(10, 2),
  loan_date DATE,
  repayment_status BOOLEAN,
  repayment_date DATE,
  FOREIGN KEY (person_id) REFERENCES Person(person_id),
  FOREIGN KEY (emp_id) REFERENCES Bank_Employee(emp_id)
);
```

```
CREATE TABLE Person (
 person_id INT PRIMARY KEY,
 name VARCHAR(100),
 salary DECIMAL(10, 2)
);

CREATE TABLE Bank_Employee (
 emp_id INT PRIMARY KEY,
 name VARCHAR(100)
);

CREATE TABLE Loan (
 loan_id INT PRIMARY KEY,
 person_id INT,
 emp_id INT,
 loan_amount DECIMAL(10, 2),
 loan_date DATE,
 repayment_status BOOLEAN,
 repayment_date DATE,
 FOREIGN KEY (person_id) REFERENCES Person(person_id),
 FOREIGN KEY (emp_id) REFERENCES Bank_Employee(emp_id)
);
```

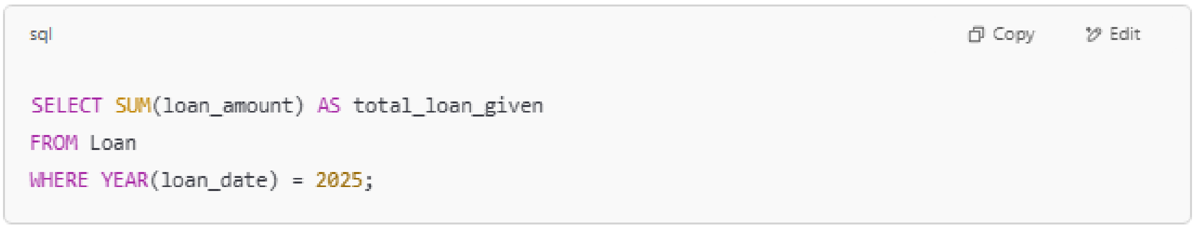
1.If a person has already taken loan 60% of his salary then he will not be given another loan.



```sql
SELECT person_id, name
FROM Person p
WHERE NOT EXISTS (
  SELECT 1
  FROM Loan l
  WHERE l.person_id = p.person_id
  AND l.loan_amount = p.salary * 0.6
);
```

```
SELECT person_id, name
FROM Person p
WHERE NOT EXISTS (
 SELECT 1
 FROM Loan l
 WHERE l.person_id = p.person_id
 AND l.loan_amount = p.salary * 0.6
);
```

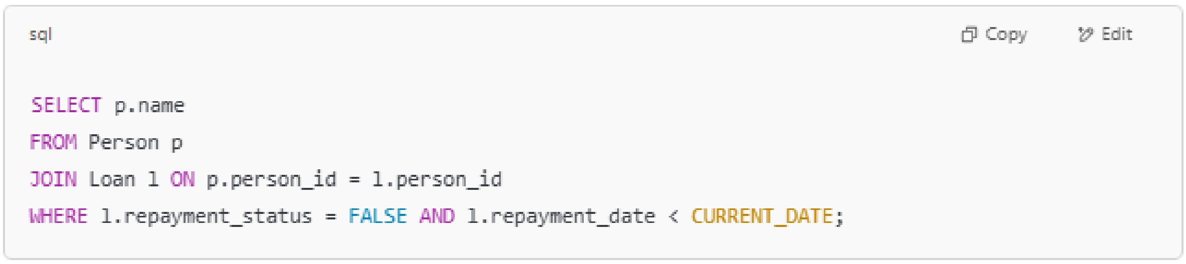
2.Calculate total loan given during specific period.



```sql
SELECT SUM(loan_amount) AS total_loan_given
FROM Loan
WHERE YEAR(loan_date) = 2025;
```

```
SELECT SUM(loan_amount) AS total_loan_given
FROM Loan
WHERE YEAR(loan_date) = 2025;
```

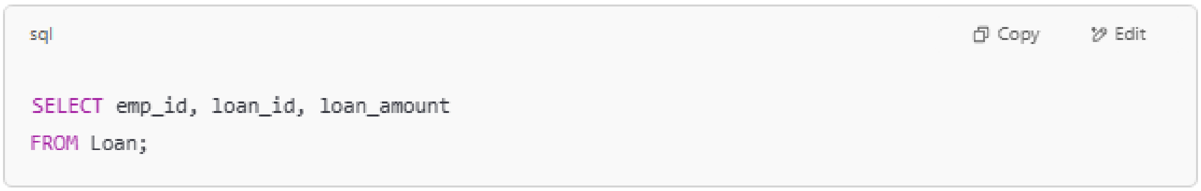
3.List names of persons not repaying loan in time.



```sql
SELECT p.name
FROM Person p
JOIN Loan l ON p.person_id = l.person_id
WHERE l.repayment_status = FALSE AND l.repayment_date < CURRENT_DATE;
```

```
SELECT p.name
FROM Person p
JOIN Loan l ON p.person_id = l.person_id
WHERE l.repayment_status = FALSE AND l.repayment_date < CURRENT_DATE;
```

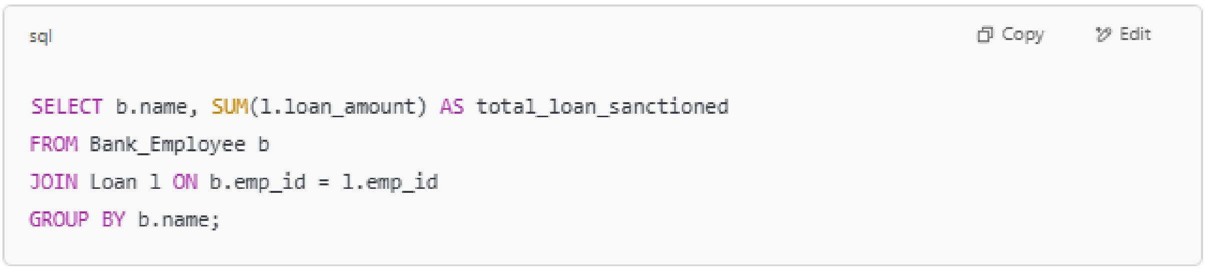
4.Each loan is sanctioned by bank employees.



```sql
SELECT emp_id, loan_id, loan_amount
FROM Loan;
```

SELECT emp_id, loan_id, loan_amount
FROM Loan;

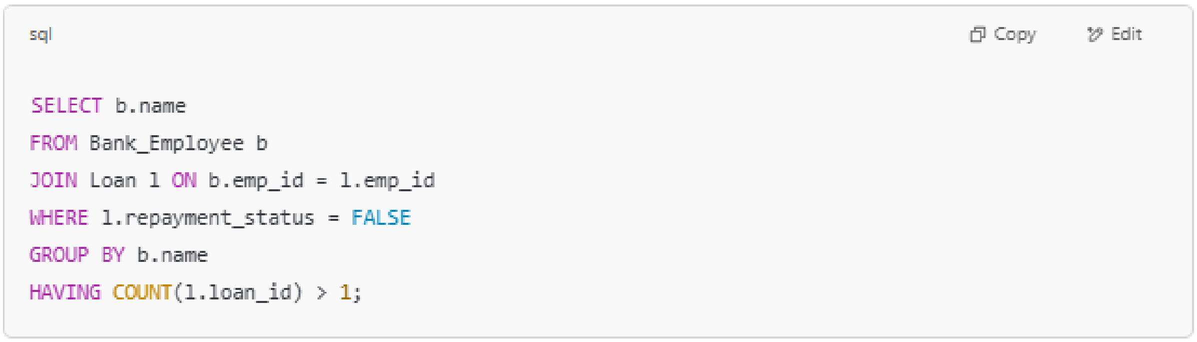
5.List name of bank employee and amount of loan sanctioned by him.



```sql
SELECT b.name, SUM(l.loan_amount) AS total_loan_sanctioned
FROM Bank_Employee b
JOIN Loan l ON b.emp_id = l.emp_id
GROUP BY b.name;
```

SELECT b.name, SUM(l.loan_amount) AS total_loan_sanctioned
FROM Bank_Employee b
JOIN Loan l ON b.emp_id = l.emp_id
GROUP BY b.name;

6.List names of bank employees whose sanctioned loan is not been repaid regularly.



```sql
SELECT b.name
FROM Bank_Employee b
JOIN Loan l ON b.emp_id = l.emp_id
WHERE l.repayment_status = FALSE
GROUP BY b.name
HAVING COUNT(l.loan_id) > 1;
```

SELECT b.name
FROM Bank_Employee b
JOIN Loan l ON b.emp_id = l.emp_id
WHERE l.repayment_status = FALSE
GROUP BY b.name
HAVING COUNT(l.loan_id) > 1;

23. A mobile manufacturing company wants keep following record

1. Name of distributor, mobile number, email address, state, city

```sql
CREATE TABLE Distributor (
  distributor_id INT PRIMARY KEY,
  name VARCHAR(100),
  mobile_number VARCHAR(15),
  email VARCHAR(100),
  state VARCHAR(50),
  city VARCHAR(50)
);

CREATE TABLE Mobile_Order (
  order_id INT PRIMARY KEY,
  distributor_id INT,
  number_of_mobiles INT,
  amount DECIMAL(10, 2),
  order_date DATE,
  FOREIGN KEY (distributor_id) REFERENCES Distributor(distributor_id)
);
```

```
CREATE TABLE Distributor (
 distributor_id INT PRIMARY KEY,
 name VARCHAR(100),
 mobile_number VARCHAR(15),
 email VARCHAR(100),
 state VARCHAR(50),
 city VARCHAR(50)
);

CREATE TABLE Mobile_Order (
 order_id INT PRIMARY KEY,
 distributor_id INT,
 number_of_mobiles INT,
 amount DECIMAL(10, 2),
 order_date DATE,
 FOREIGN KEY (distributor_id) REFERENCES Distributor(distributor_id)
);
```

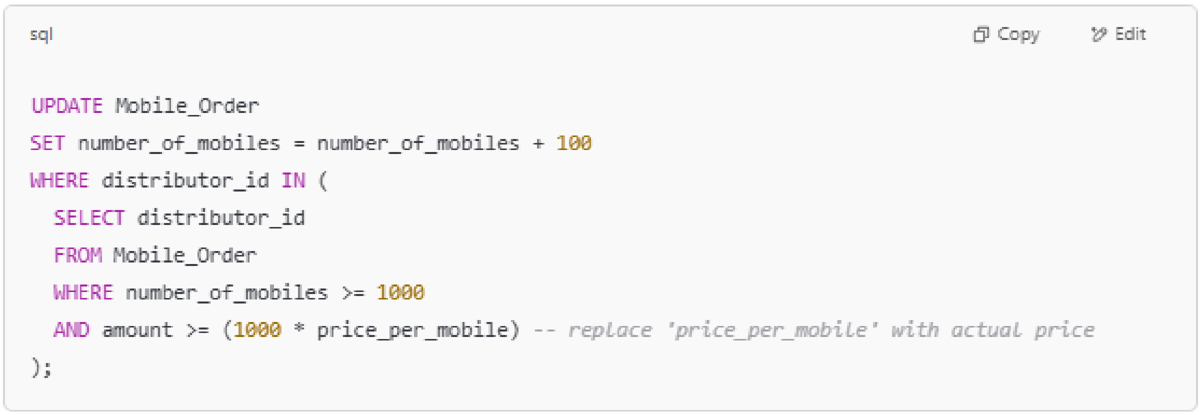
2. Number of mobiles ordered by distributor

```sql
SELECT distributor_id, SUM(number_of_mobiles) AS total_mobiles_ordered
FROM Mobile_Order
GROUP BY distributor_id;
```

```
SELECT distributor_id, SUM(number_of_mobiles) AS total_mobiles_ordered
FROM Mobile_Order
```

GROUP BY distributor_id;

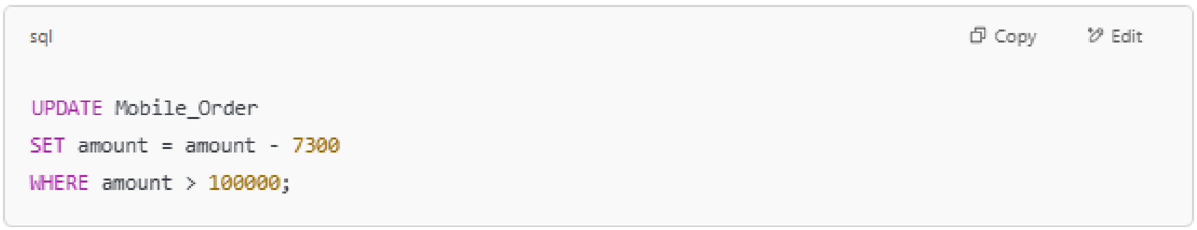
3. If a distributor has ordered 1000 mobiles and deposited amount required for those mobiles then he must be given 100 mobiles on credit.



```sql
UPDATE Mobile_Order
SET number_of_mobiles = number_of_mobiles + 100
WHERE distributor_id IN (
  SELECT distributor_id
  FROM Mobile_Order
  WHERE number_of_mobiles >= 1000
  AND amount >= (1000 * price_per_mobile) -- replace 'price_per_mobile' with actual price
);
```

UPDATE Mobile_Order
SET number_of_mobiles = number_of_mobiles + 100
WHERE distributor_id IN (
 SELECT distributor_id
 FROM Mobile_Order
 WHERE number_of_mobiles >= 1000
 AND amount >= (1000 * price_per_mobile) -- replace 'price_per_mobile' with actual price
);

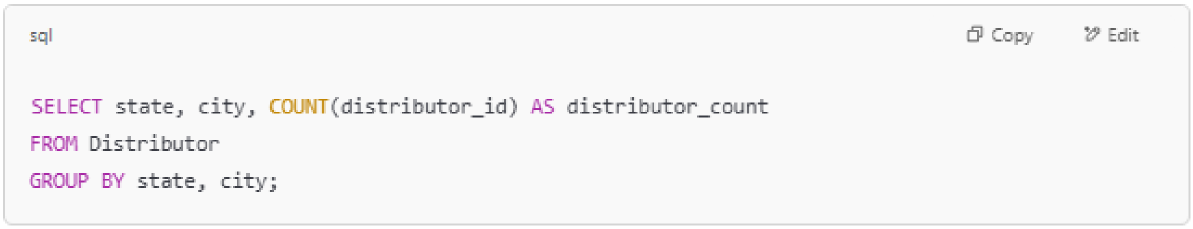
4. If the order crosses Rs. 100000/- then the distributor is given discount of Rs. 7300/-



```sql
UPDATE Mobile_Order
SET amount = amount - 7300
WHERE amount > 100000;
```

UPDATE Mobile_Order
SET amount = amount - 7300
WHERE amount > 100000;

5. Count of distributors in each state, city.

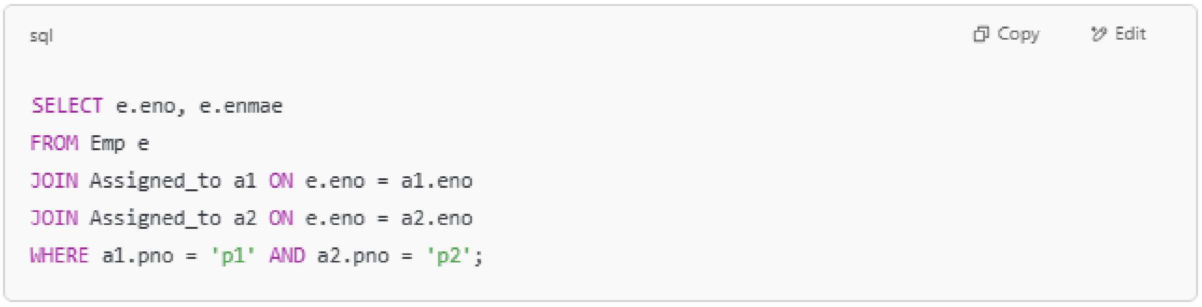


```sql
SELECT state, city, COUNT(distributor_id) AS distributor_count
FROM Distributor
GROUP BY state, city;
```

SELECT state, city, COUNT(distributor_id) AS distributor_count
FROM Distributor
GROUP BY state, city;

24. Consider the following relations Project(pno,pname,chief_architect) Emp(eno,enmae) Assigned_to(pno,eno) Implement the following queries.

1.Get details of emps working on both p1 and p2 projects.



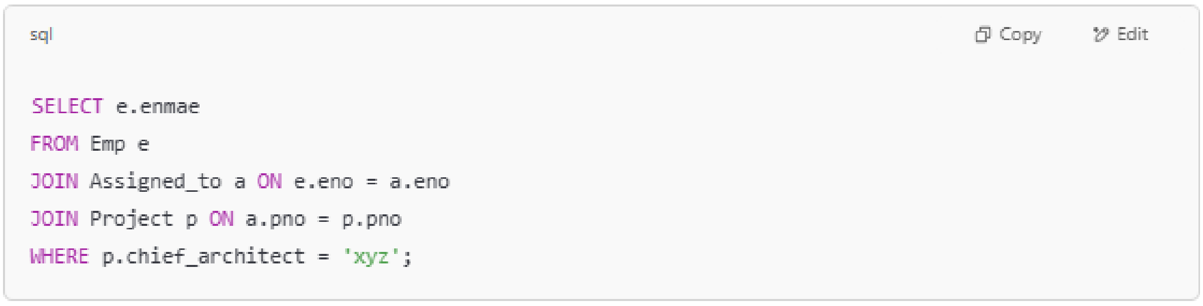
```sql
SELECT e.eno, e.enmae
FROM Emp e
JOIN Assigned_to a1 ON e.eno = a1.eno
JOIN Assigned_to a2 ON e.eno = a2.eno
WHERE a1.pno = 'p1' AND a2.pno = 'p2';
```

SELECT e.eno, e.enmae
FROM Emp e
JOIN Assigned_to a1 ON e.eno = a1.eno
JOIN Assigned_to a2 ON e.eno = a2.eno
WHERE a1.pno = 'p1' AND a2.pno = 'p2';

2.Get emp nos of emps who work at least all those projects that emp 'abc' works on.

```
        SELECT e.eno
        FROM Emp e
        JOIN Assigned_to a ON e.eno = a.eno
        WHERE NOT EXISTS (
        SELECT 1
        FROM Assigned_to a2
        WHERE a2.pno NOT IN
          (SELECT pno FROM Assigned_to WHERE eno =
              (SELECT eno FROM Emp WHERE enmae = 'abc'))
         AND a.pno = a2.pno
);
```

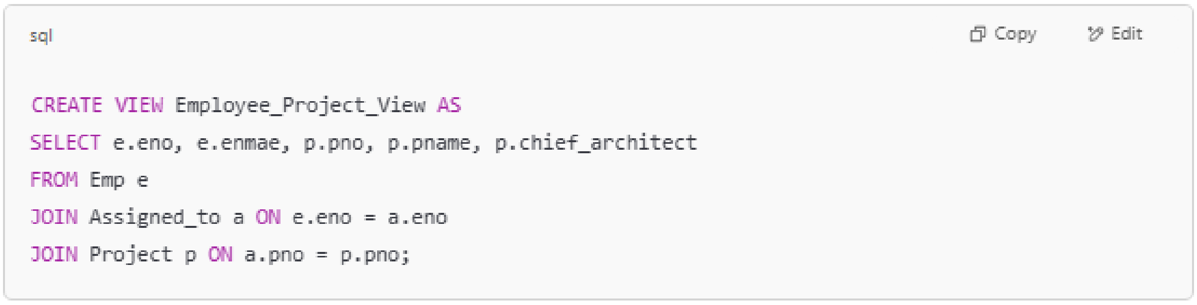
3.Get names of emps who are assigned to projects designed by chief_architect 'xyz'.



```sql
SELECT e.enmae
FROM Emp e
JOIN Assigned_to a ON e.eno = a.eno
JOIN Project p ON a.pno = p.pno
WHERE p.chief_architect = 'xyz';
```

SELECT e.enmae
FROM Emp e
JOIN Assigned_to a ON e.eno = a.eno
JOIN Project p ON a.pno = p.pno
WHERE p.chief_architect = 'xyz';

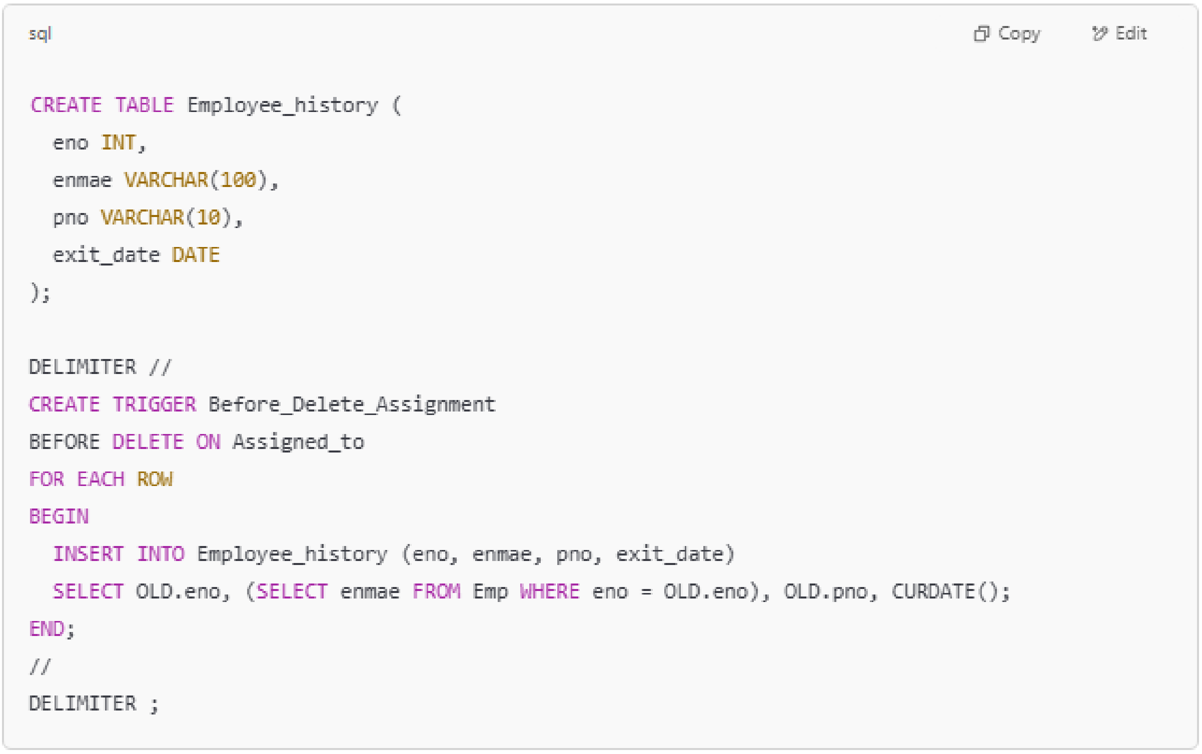
4.Create a view that displays employee details and the project details they are working on.



```sql
CREATE VIEW Employee_Project_View AS
SELECT e.eno, e.enmae, p.pno, p.pname, p.chief_architect
FROM Emp e
JOIN Assigned_to a ON e.eno = a.eno
JOIN Project p ON a.pno = p.pno;
```

CREATE VIEW Employee_Project_View AS
SELECT e.eno, e.enmae, p.pno, p.pname, p.chief_architect
FROM Emp e
JOIN Assigned_to a ON e.eno = a.eno
JOIN Project p ON a.pno = p.pno;

5.Create a trigger which will generate the new table Employee_history, that will be storing the data of employees who left the project.



```sql
CREATE TABLE Employee_history (
  eno INT,
  enmae VARCHAR(100),
  pno VARCHAR(10),
  exit_date DATE
);

DELIMITER //
CREATE TRIGGER Before_Delete_Assignment
BEFORE DELETE ON Assigned_to
FOR EACH ROW
BEGIN
  INSERT INTO Employee_history (eno, enmae, pno, exit_date)
  SELECT OLD.eno, (SELECT enmae FROM Emp WHERE eno = OLD.eno), OLD.pno, CURDATE();
END;
//
DELIMITER ;
```

CREATE TABLE Employee_history (
 eno INT,
 enmae VARCHAR(100),
 pno VARCHAR(10),
 exit_date DATE
);

DELIMITER //
CREATE TRIGGER Before_Delete_Assignment

```
BEFORE DELETE ON Assigned_to
FOR EACH ROW
BEGIN
  INSERT INTO Employee_history (eno, enmae, pno, exit_date)
  SELECT OLD.eno, (SELECT enmae FROM Emp WHERE eno = OLD.eno), OLD.pno,
CURDATE();
END;
//
DELIMITER ;
```

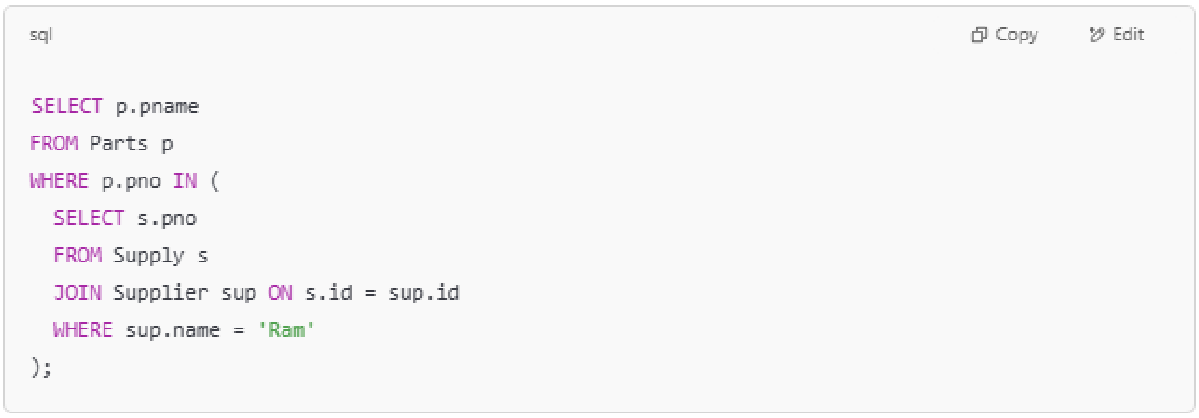
25. Consider the following database schema to write nested queries in SQL

Supplier (id, name, city)

Parts(pno, pname, pdescription)

Supply(id, pno, cost)

i) Find the names of the parts supplied by "Ram"

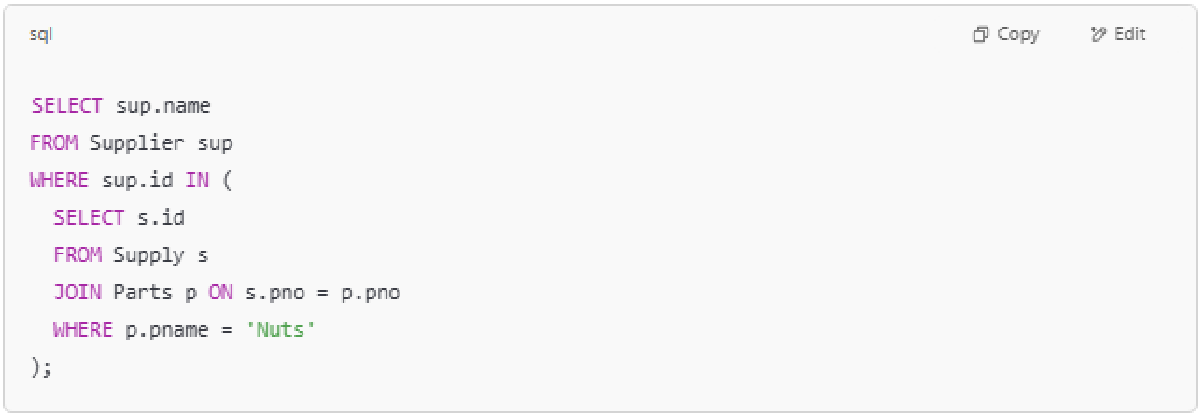


sql Copy Edit

```sql
SELECT p.pname
FROM Parts p
WHERE p.pno IN (
  SELECT s.pno
  FROM Supply s
  JOIN Supplier sup ON s.id = sup.id
  WHERE sup.name = 'Ram'
);
```

```
SELECT p.pname
FROM Parts p
WHERE p.pno IN (
  SELECT s.pno
  FROM Supply s
  JOIN Supplier sup ON s.id = sup.id
  WHERE sup.name = 'Ram'
);
```

ii) Find the names of the suppliers who supply "Nuts"



```sql
SELECT sup.name
FROM Supplier sup
WHERE sup.id IN (
  SELECT s.id
  FROM Supply s
  JOIN Parts p ON s.pno = p.pno
  WHERE p.pname = 'Nuts'
);
```

SELECT sup.name

FROM Supplier sup

WHERE sup.id IN (

 SELECT s.id

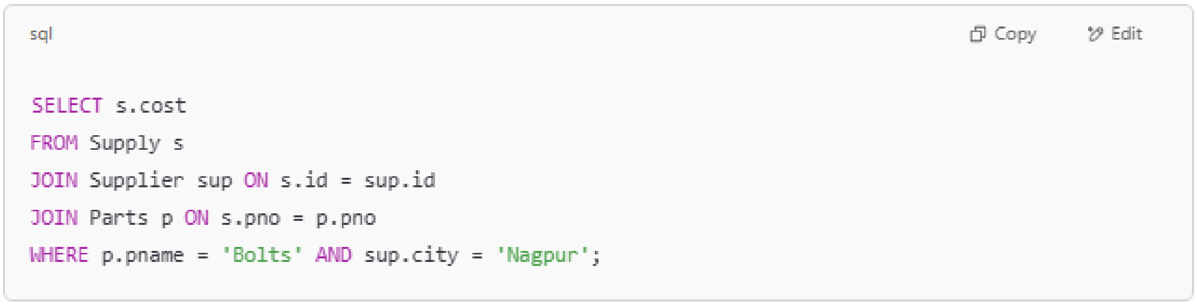
 FROM Supply s

 JOIN Parts p ON s.pno = p.pno

 WHERE p.pname = 'Nuts'

);

iii) Find the cost of bolts being supplied by Nagpur suppliers.



```sql
SELECT s.cost
FROM Supply s
JOIN Supplier sup ON s.id = sup.id
JOIN Parts p ON s.pno = p.pno
WHERE p.pname = 'Bolts' AND sup.city = 'Nagpur';
```

SELECT s.cost

FROM Supply s

JOIN Supplier sup ON s.id = sup.id

JOIN Parts p ON s.pno = p.pno

WHERE p.pname = 'Bolts' AND sup.city = 'Nagpur';

26. A relational database contains two tables EMP & DEPT. Create the above 2 tables with the following structure:
EMP Table

Name	**Type**	**Description**
EMPNO	NUMERIC(4) Primary key	Employee no
ENAME	VARCHAR(20) NOT NULL	Employee name
JOB	CHAR(10)	Designation
MGR	NUMERIC(4)	Respective manager's empno
HIREDATE	DATETIME	Date of Joining
SAL	NUMERIC(9,2)	Salary
COMM	NUMERIC(7,2)	Commission
DEPTNO	NUMERIC(2)	Department

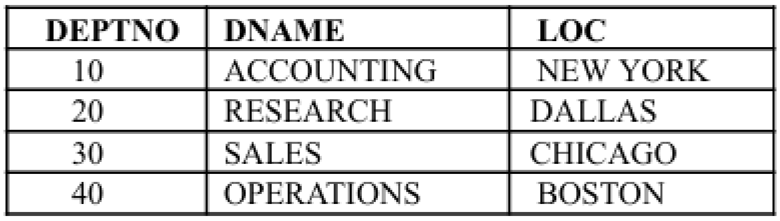
DEPT table

Name	**Type**	**Description**
DEPTNO	NUMRIC(2) Primary key	Dept no
DNAME	VARCHAR(20) NOT NULL	Department name
LOC	VARCHAR(10)	Department location

Insert the following records into EMP Table:

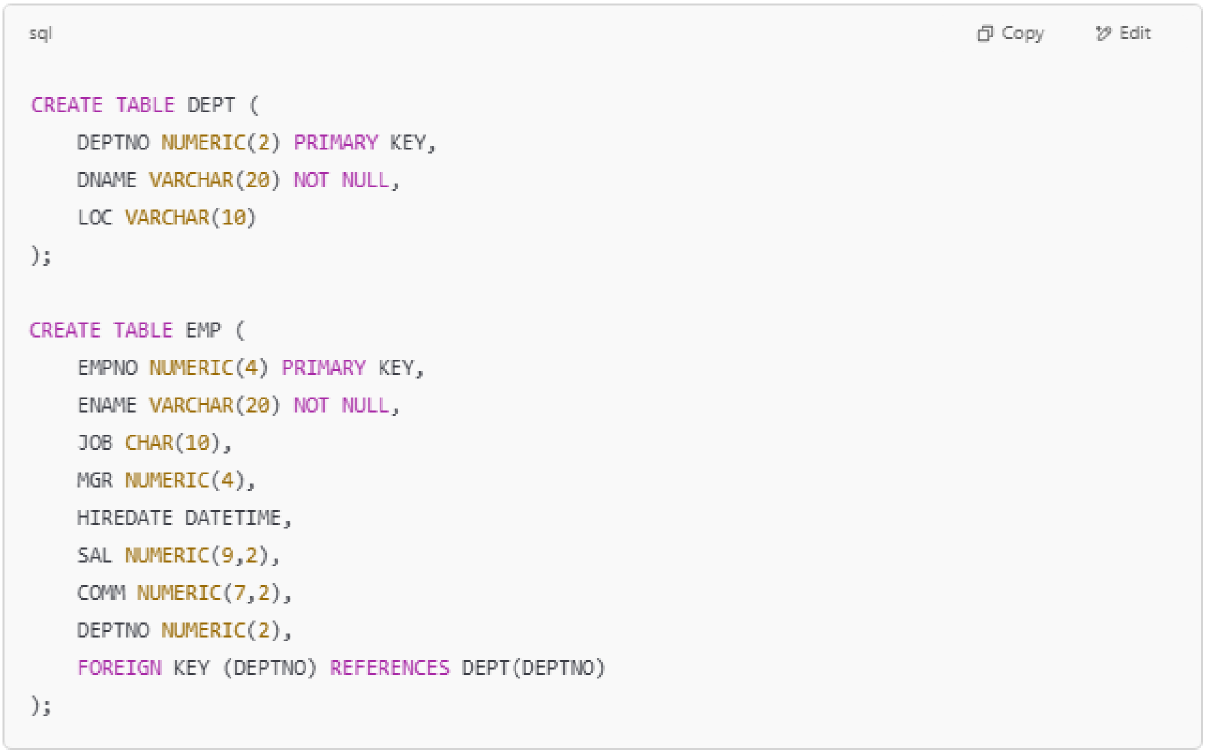
EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7369	**SMITH**	**CLERK**	**7902**	**1980-12-17**	**800**		**20**
7499	**ALLEN**	**SALESMAN**	**7698**	**1981-02-20**	**1600**	**300**	**30**
7521	WARD	SALESMAN	7698	**1981-02-22**	1250	500	30
7566	JONES	MANAGER	7839	**1981-04-02**	2975		20
7654	MARTIN	SALESMAN	7698	**1981-09-28**	1250	1400	30
7698	BLAKE	MANAGER	7839	**1981-05-01**	2850		30
7782	CLARK	MANAGER	7839	**1981-06-09**	2450		10
7788	SCOTT	ANALYST	7566	**1987-04-19**	3000		20
7839	KING	PRESIDENT		**1981-11-17**	5000		10
7844	TURNER	SALESMAN	7698	**1981-09-08**	1500	0	30
7876	ADAMS	CLERK	7788	**1987-05-23**	1100		20
7900	JAMES	CLERK	7698	**1981-12-03**	950		30
7902	FORD	ANALYST	7566	**1981-12-03**	3000		20
7934	**MILLER**	**CLERK**	**7782**	**1982-01-23**	**1300**		**10**

Insert the following records into DEPT Table:



DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON

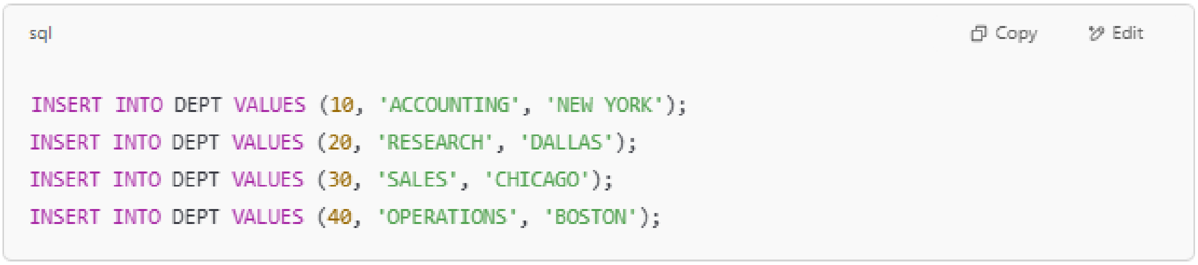
Create the EMP and DEPT Tables



```sql
CREATE TABLE DEPT (
    DEPTNO NUMERIC(2) PRIMARY KEY,
    DNAME VARCHAR(20) NOT NULL,
    LOC VARCHAR(10)
);

CREATE TABLE EMP (
    EMPNO NUMERIC(4) PRIMARY KEY,
    ENAME VARCHAR(20) NOT NULL,
    JOB CHAR(10),
    MGR NUMERIC(4),
    HIREDATE DATETIME,
    SAL NUMERIC(9,2),
    COMM NUMERIC(7,2),
    DEPTNO NUMERIC(2),
    FOREIGN KEY (DEPTNO) REFERENCES DEPT(DEPTNO)
);
```

```sql
CREATE TABLE DEPT (
   DEPTNO NUMERIC(2) PRIMARY KEY,
   DNAME VARCHAR(20) NOT NULL,
   LOC VARCHAR(10)
);

CREATE TABLE EMP (
   EMPNO NUMERIC(4) PRIMARY KEY,
   ENAME VARCHAR(20) NOT NULL,
   JOB CHAR(10),
   MGR NUMERIC(4),
   HIREDATE DATETIME,
   SAL NUMERIC(9,2),
   COMM NUMERIC(7,2),
   DEPTNO NUMERIC(2),
   FOREIGN KEY (DEPTNO) REFERENCES DEPT(DEPTNO)
);
```

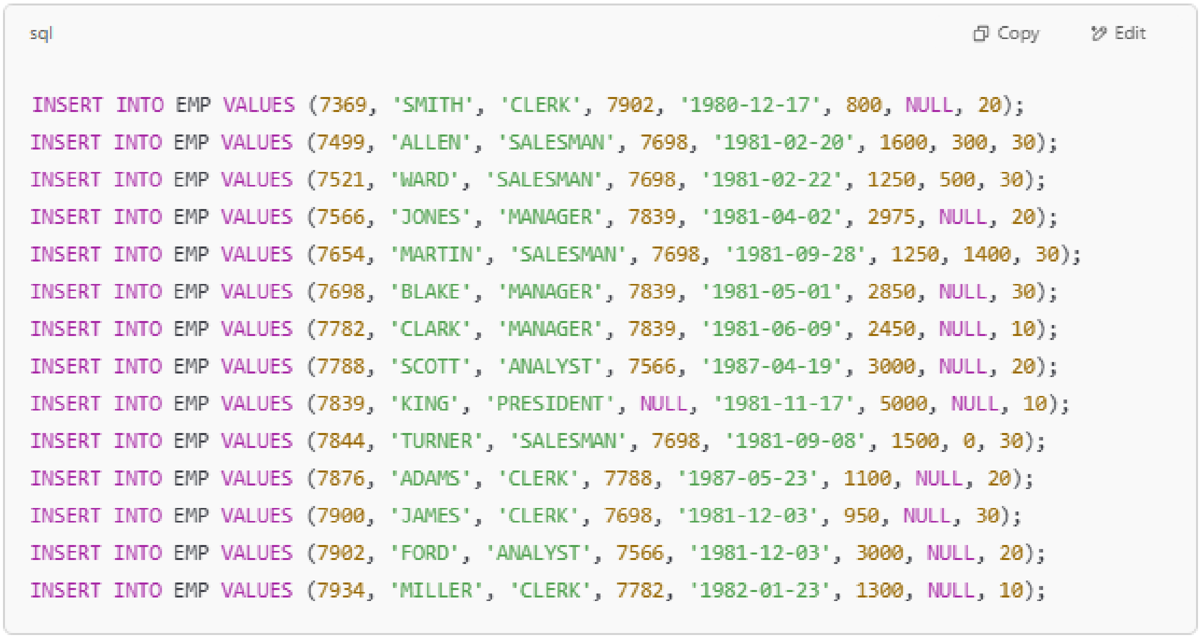
Insert into DEPT Table



```sql
INSERT INTO DEPT VALUES (10, 'ACCOUNTING', 'NEW YORK');
INSERT INTO DEPT VALUES (20, 'RESEARCH', 'DALLAS');
INSERT INTO DEPT VALUES (30, 'SALES', 'CHICAGO');
INSERT INTO DEPT VALUES (40, 'OPERATIONS', 'BOSTON');
```

INSERT INTO DEPT VALUES (10, 'ACCOUNTING', 'NEW YORK');
INSERT INTO DEPT VALUES (20, 'RESEARCH', 'DALLAS');
INSERT INTO DEPT VALUES (30, 'SALES', 'CHICAGO');
INSERT INTO DEPT VALUES (40, 'OPERATIONS', 'BOSTON');

Insert into EMP Table



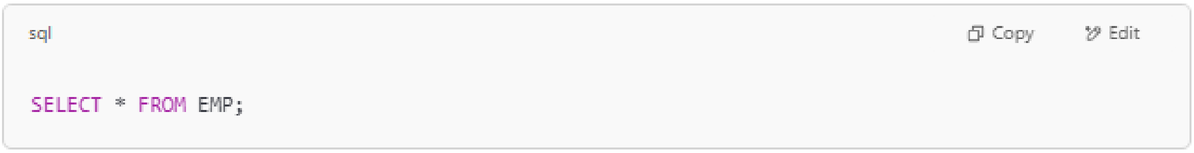
```sql
INSERT INTO EMP VALUES (7369, 'SMITH', 'CLERK', 7902, '1980-12-17', 800, NULL, 20);
INSERT INTO EMP VALUES (7499, 'ALLEN', 'SALESMAN', 7698, '1981-02-20', 1600, 300, 30);
INSERT INTO EMP VALUES (7521, 'WARD', 'SALESMAN', 7698, '1981-02-22', 1250, 500, 30);
INSERT INTO EMP VALUES (7566, 'JONES', 'MANAGER', 7839, '1981-04-02', 2975, NULL, 20);
INSERT INTO EMP VALUES (7654, 'MARTIN', 'SALESMAN', 7698, '1981-09-28', 1250, 1400, 30);
INSERT INTO EMP VALUES (7698, 'BLAKE', 'MANAGER', 7839, '1981-05-01', 2850, NULL, 30);
INSERT INTO EMP VALUES (7782, 'CLARK', 'MANAGER', 7839, '1981-06-09', 2450, NULL, 10);
INSERT INTO EMP VALUES (7788, 'SCOTT', 'ANALYST', 7566, '1987-04-19', 3000, NULL, 20);
INSERT INTO EMP VALUES (7839, 'KING', 'PRESIDENT', NULL, '1981-11-17', 5000, NULL, 10);
INSERT INTO EMP VALUES (7844, 'TURNER', 'SALESMAN', 7698, '1981-09-08', 1500, 0, 30);
INSERT INTO EMP VALUES (7876, 'ADAMS', 'CLERK', 7788, '1987-05-23', 1100, NULL, 20);
INSERT INTO EMP VALUES (7900, 'JAMES', 'CLERK', 7698, '1981-12-03', 950, NULL, 30);
INSERT INTO EMP VALUES (7902, 'FORD', 'ANALYST', 7566, '1981-12-03', 3000, NULL, 20);
INSERT INTO EMP VALUES (7934, 'MILLER', 'CLERK', 7782, '1982-01-23', 1300, NULL, 10);
```

INSERT INTO EMP VALUES (7369, 'SMITH', 'CLERK', 7902, '1980-12-17', 800, NULL, 20);
INSERT INTO EMP VALUES (7499, 'ALLEN', 'SALESMAN', 7698, '1981-02-20', 1600, 300, 30);
INSERT INTO EMP VALUES (7521, 'WARD', 'SALESMAN', 7698, '1981-02-22', 1250, 500, 30);
INSERT INTO EMP VALUES (7566, 'JONES', 'MANAGER', 7839, '1981-04-02', 2975, NULL, 20);
INSERT INTO EMP VALUES (7654, 'MARTIN', 'SALESMAN', 7698, '1981-09-28', 1250, 1400, 30);
INSERT INTO EMP VALUES (7698, 'BLAKE', 'MANAGER', 7839, '1981-05-01', 2850, NULL, 30);
INSERT INTO EMP VALUES (7782, 'CLARK', 'MANAGER', 7839, '1981-06-09', 2450, NULL, 10);
INSERT INTO EMP VALUES (7788, 'SCOTT', 'ANALYST', 7566, '1987-04-19', 3000, NULL, 20);

INSERT INTO EMP VALUES (7839, 'KING', 'PRESIDENT', NULL, '1981-11-17', 5000, NULL, 10);
INSERT INTO EMP VALUES (7844, 'TURNER', 'SALESMAN', 7698, '1981-09-08', 1500, 0, 30);
INSERT INTO EMP VALUES (7876, 'ADAMS', 'CLERK', 7788, '1987-05-23', 1100, NULL, 20);
INSERT INTO EMP VALUES (7900, 'JAMES', 'CLERK', 7698, '1981-12-03', 950, NULL, 30);
INSERT INTO EMP VALUES (7902, 'FORD', 'ANALYST', 7566, '1981-12-03', 3000, NULL, 20);
INSERT INTO EMP VALUES (7934, 'MILLER', 'CLERK', 7782, '1982-01-23', 1300, NULL, 10);

Perform the following queries :

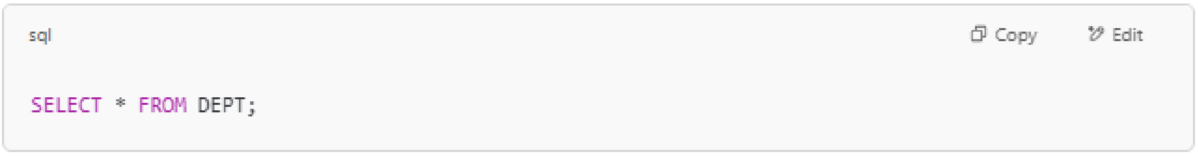
1. SELECT ALL THE RECORDS FROM EMP TABLE



```sql
SELECT * FROM EMP;
```

SELECT * FROM EMP;

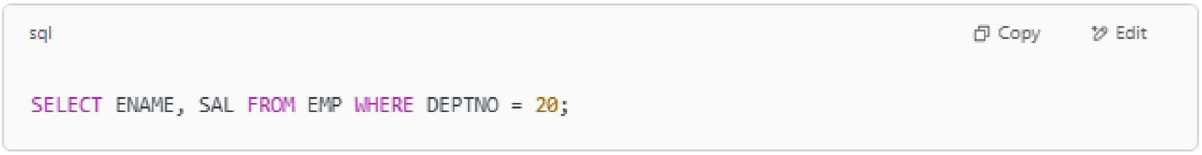
2. SELECT ALL THE RECORDS FROM DEPT TABLE



```sql
SELECT * FROM DEPT;
```

SELECT * FROM DEPT;

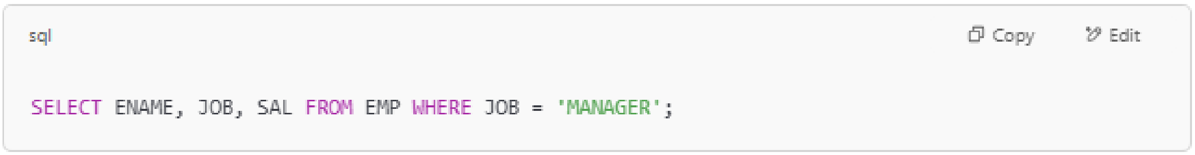
3. FIND THE EMPLOYEE NAME,SALARY WHO IS WORKING IN DEPT NO 20



```sql
SELECT ENAME, SAL FROM EMP WHERE DEPTNO = 20;
```

SELECT ENAME, SAL FROM EMP WHERE DEPTNO = 20;

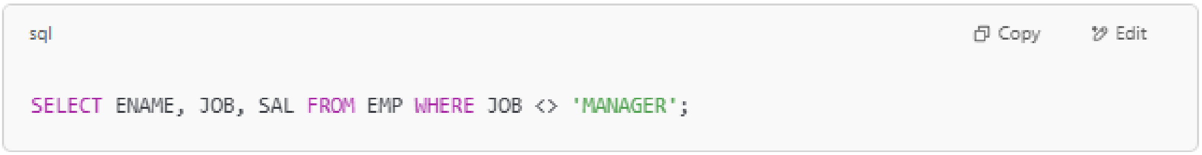
4. FIND THE NAME, JOB, SALARY OF THE EMPLOYEE WHO IS MANAGER



```sql
SELECT ENAME, JOB, SAL FROM EMP WHERE JOB = 'MANAGER';
```

SELECT ENAME, JOB, SAL FROM EMP WHERE JOB = 'MANAGER';

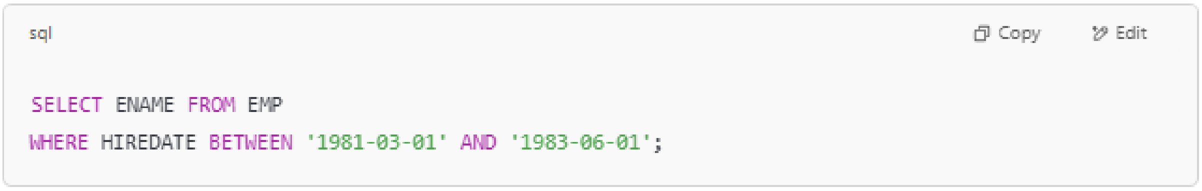
5. FIND THE NAME ,JOB,SALARY OF THE EMPLOYEE WHO IS NOT A MANAGER



```sql
SELECT ENAME, JOB, SAL FROM EMP WHERE JOB <> 'MANAGER';
```

SELECT ENAME, JOB, SAL FROM EMP WHERE JOB <> 'MANAGER';

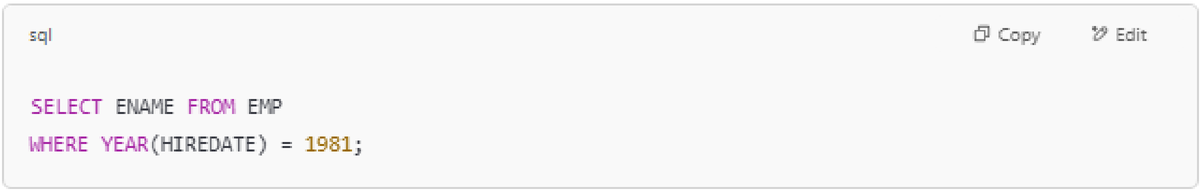
6. FIND THOSE EMPLOYEES WHO WERE HIRED BETWEEN 1 MAR 1981 AND 1 JUN 1983



```sql
SELECT ENAME FROM EMP
WHERE HIREDATE BETWEEN '1981-03-01' AND '1983-06-01';
```

SELECT ENAME FROM EMP
WHERE HIREDATE BETWEEN '1981-03-01' AND '1983-06-01';

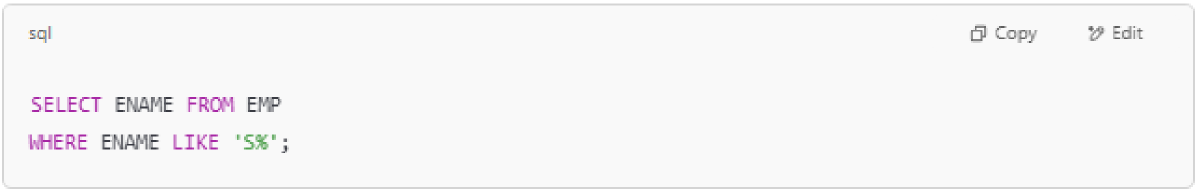
7. FIND EMPLOYEE NAME WHO WERE HIRED IN 1981



```sql
SELECT ENAME FROM EMP
WHERE YEAR(HIREDATE) = 1981;
```

SELECT ENAME FROM EMP
WHERE YEAR(HIREDATE) = 1981;

8. FIND EMPLOYEE NAME WHOSE NAME STARS WITH 'S'

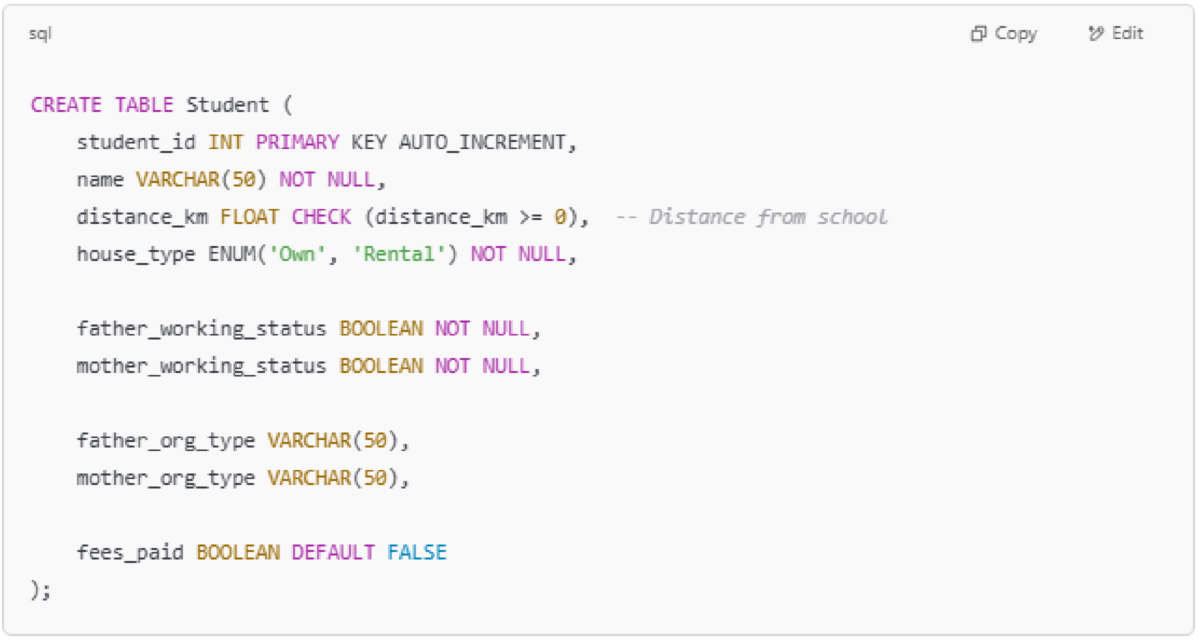


```sql
SELECT ENAME FROM EMP
WHERE ENAME LIKE 'S%';
```

SELECT ENAME FROM EMP
WHERE ENAME LIKE 'S%';

27. A school wants to keep following information of their students

☑ **Create Table: Student:**

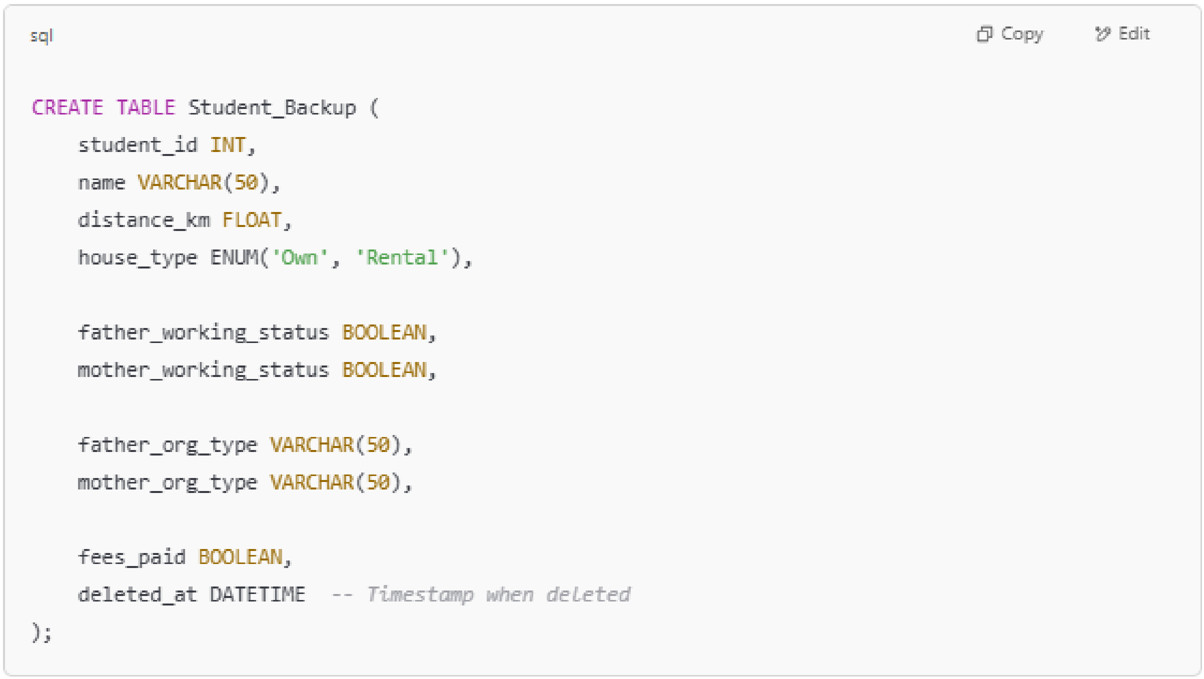


```sql
CREATE TABLE Student (
    student_id INT PRIMARY KEY AUTO_INCREMENT,
    name VARCHAR(50) NOT NULL,
    distance_km FLOAT CHECK (distance_km >= 0),  -- Distance from school
    house_type ENUM('Own', 'Rental') NOT NULL,

    father_working_status BOOLEAN NOT NULL,
    mother_working_status BOOLEAN NOT NULL,

    father_org_type VARCHAR(50),
    mother_org_type VARCHAR(50),

    fees_paid BOOLEAN DEFAULT FALSE
);
```

CREATE TABLE Student (
  student_id INT PRIMARY KEY AUTO_INCREMENT,

```
  name VARCHAR(50) NOT NULL,
  distance_km FLOAT CHECK (distance_km >= 0),  -- Distance from school
  house_type ENUM('Own', 'Rental') NOT NULL,

  father_working_status BOOLEAN NOT NULL,
  mother_working_status BOOLEAN NOT NULL,

  father_org_type VARCHAR(50),
  mother_org_type VARCHAR(50),

  fees_paid BOOLEAN DEFAULT FALSE
);
```

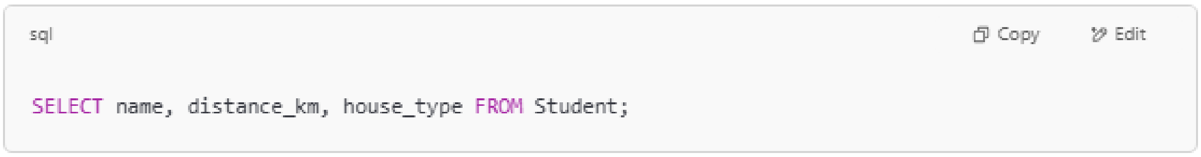
☑ **Create Table: Student_Backup** (For Deleted Records Backup)



```sql
CREATE TABLE Student_Backup (
    student_id INT,
    name VARCHAR(50),
    distance_km FLOAT,
    house_type ENUM('Own', 'Rental'),

    father_working_status BOOLEAN,
    mother_working_status BOOLEAN,

    father_org_type VARCHAR(50),
    mother_org_type VARCHAR(50),

    fees_paid BOOLEAN,
    deleted_at DATETIME   -- Timestamp when deleted
);
```

```
CREATE TABLE Student_Backup (
  student_id INT,
  name VARCHAR(50),
  distance_km FLOAT,
  house_type ENUM('Own', 'Rental'),

  father_working_status BOOLEAN,
  mother_working_status BOOLEAN,
    father_org_type VARCHAR(50),
    mother_org_type VARCHAR(50),

    fees_paid BOOLEAN,
    deleted_at DATETIME  -- Timestamp when deleted
);
```

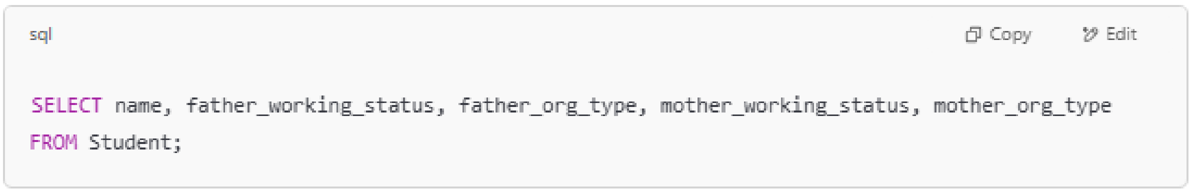
1. Name, distance of his house from school, own or rental house



```sql
SELECT name, distance_km, house_type FROM Student;
```

SELECT name, distance_km, house_type FROM Student;

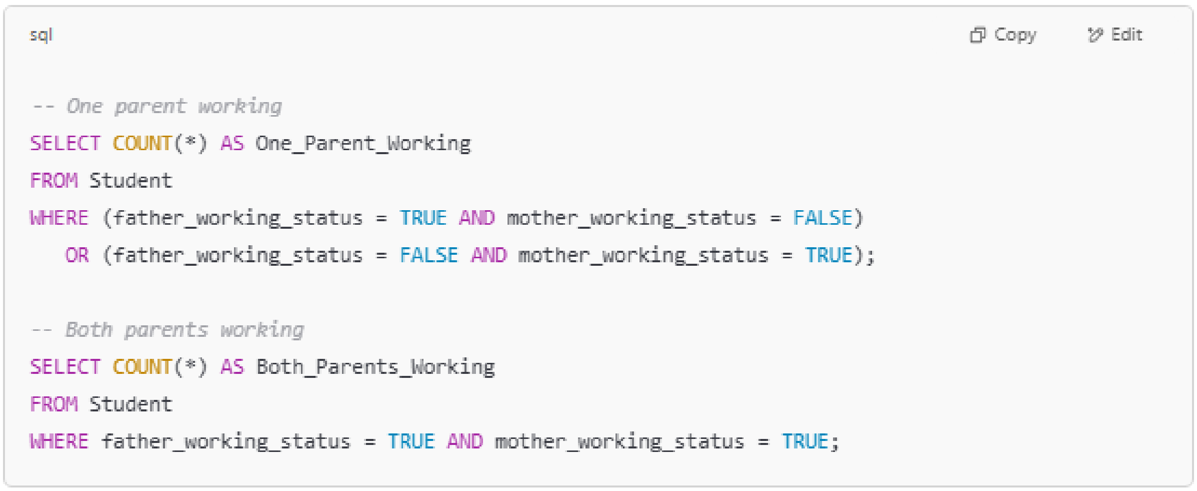
2. Father/Mother working status, type of organization they are working



```sql
SELECT name, father_working_status, father_org_type, mother_working_status, mother_org_type
FROM Student;
```

SELECT name, father_working_status, father_org_type, mother_working_status, mother_org_type
FROM Student;

3. Count of students whose one parent working and both parent working



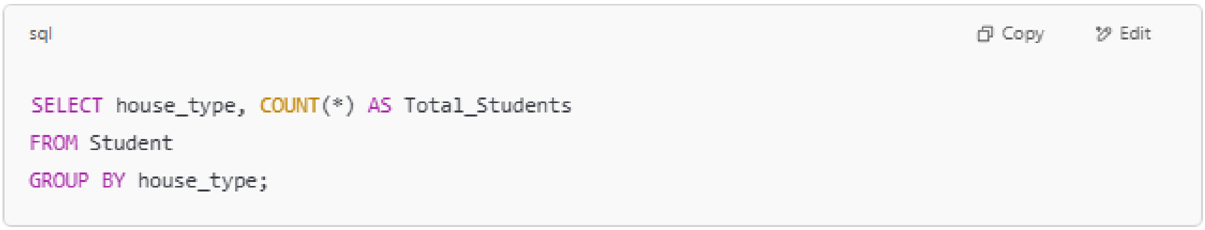
```sql
-- One parent working
SELECT COUNT(*) AS One_Parent_Working
FROM Student
WHERE (father_working_status = TRUE AND mother_working_status = FALSE)
   OR (father_working_status = FALSE AND mother_working_status = TRUE);

-- Both parents working
SELECT COUNT(*) AS Both_Parents_Working
FROM Student
WHERE father_working_status = TRUE AND mother_working_status = TRUE;
```

-- One parent working
SELECT COUNT(*) AS One_Parent_Working
FROM Student
WHERE (father_working_status = TRUE AND mother_working_status = FALSE)
   OR (father_working_status = FALSE AND mother_working_status = TRUE);

-- Both parents working
SELECT COUNT(*) AS Both_Parents_Working

FROM Student
WHERE father_working_status = TRUE AND mother_working_status = TRUE;

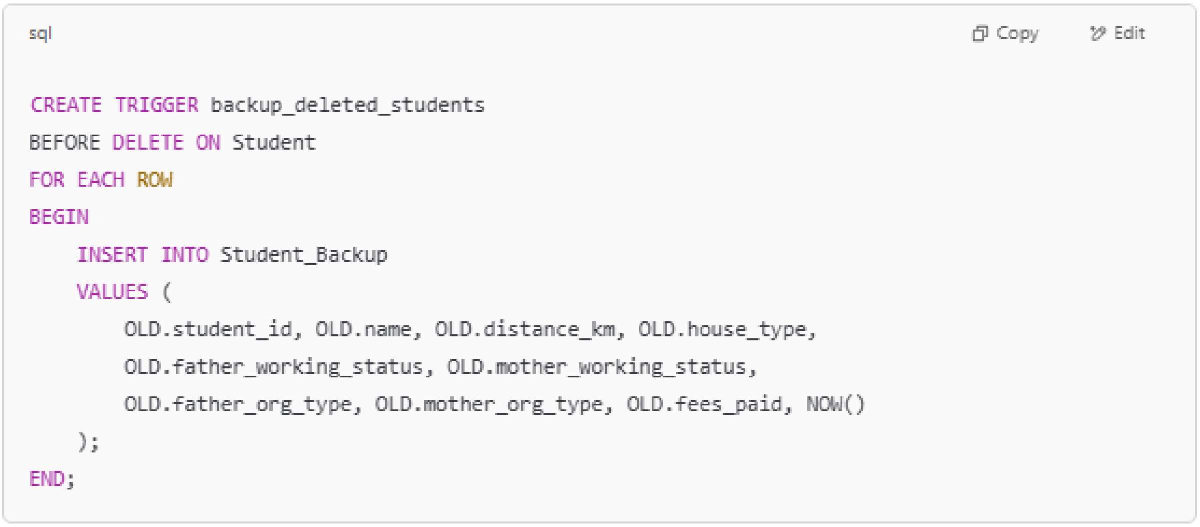
4. Count of students leaving in own house and rental.



```sql
SELECT house_type, COUNT(*) AS Total_Students
FROM Student
GROUP BY house_type;
```

SELECT house_type, COUNT(*) AS Total_Students
FROM Student
GROUP BY house_type;

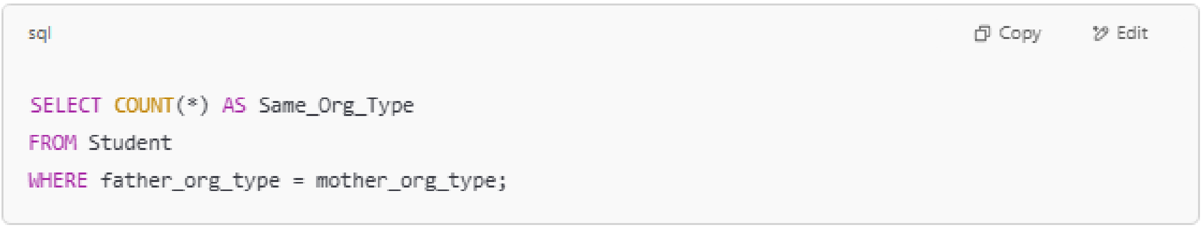
5. If the student is deleted from the table then his information must be transferred to other table.



```sql
CREATE TRIGGER backup_deleted_students
BEFORE DELETE ON Student
FOR EACH ROW
BEGIN
    INSERT INTO Student_Backup
    VALUES (
        OLD.student_id, OLD.name, OLD.distance_km, OLD.house_type,
        OLD.father_working_status, OLD.mother_working_status,
        OLD.father_org_type, OLD.mother_org_type, OLD.fees_paid, NOW()
    );
END;
```

CREATE TRIGGER backup_deleted_students
BEFORE DELETE ON Student
FOR EACH ROW
BEGIN
  INSERT INTO Student_Backup
  VALUES (
    OLD.student_id, OLD.name, OLD.distance_km, OLD.house_type,
    OLD.father_working_status, OLD.mother_working_status,
    OLD.father_org_type, OLD.mother_org_type, OLD.fees_paid, NOW()
  );
END;

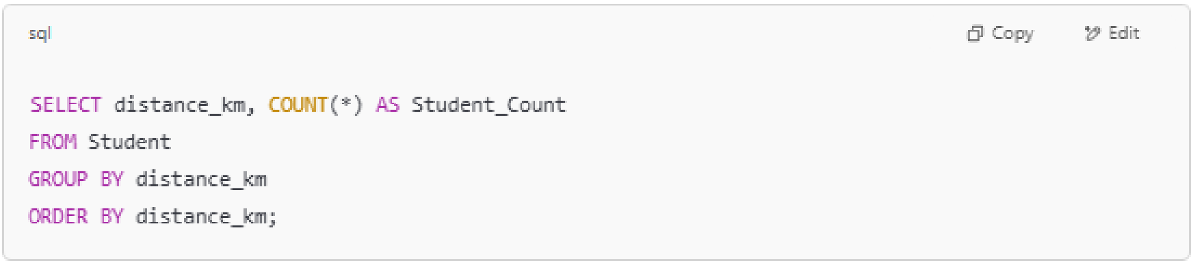
6. Count of students whose father and mother type of organization is same.



```sql
SELECT COUNT(*) AS Same_Org_Type
FROM Student
WHERE father_org_type = mother_org_type;
```

SELECT COUNT(*) AS Same_Org_Type
FROM Student
WHERE father_org_type = mother_org_type;

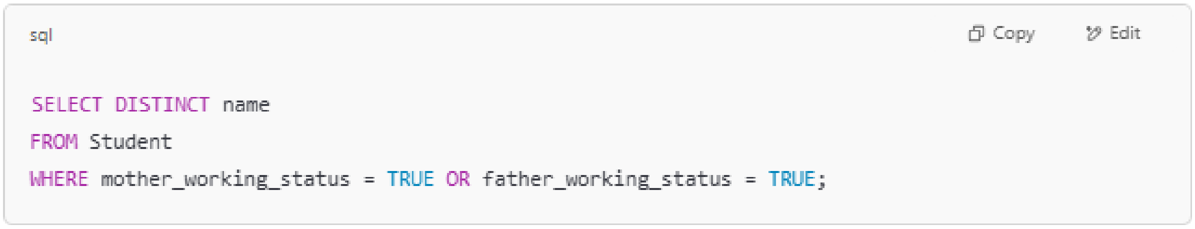
7. Count of students according to their distances from school.



```sql
SELECT distance_km, COUNT(*) AS Student_Count
FROM Student
GROUP BY distance_km
ORDER BY distance_km;
```

SELECT distance_km, COUNT(*) AS Student_Count
FROM Student
GROUP BY distance_km
ORDER BY distance_km;

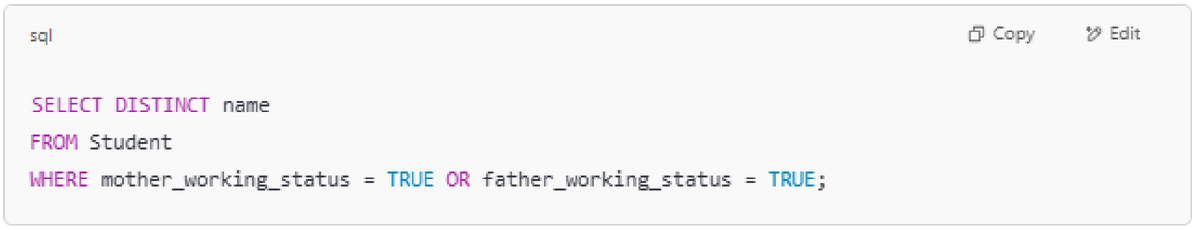
8. Name of students whose mother is working and name of students whose father is working. List name once if it is coming more than once.



```sql
SELECT DISTINCT name
FROM Student
WHERE mother_working_status = TRUE OR father_working_status = TRUE;
```

SELECT DISTINCT name
FROM Student
WHERE mother_working_status = TRUE OR father_working_status = TRUE;

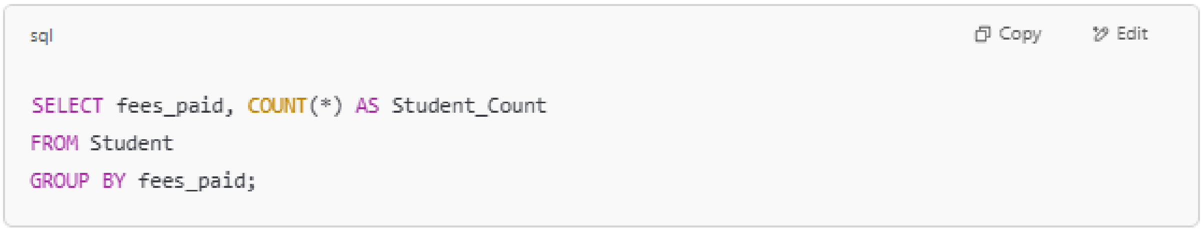
9. Student whose only one parent is working will be given 10% discount in fees.



```sql
SELECT DISTINCT name
FROM Student
WHERE mother_working_status = TRUE OR father_working_status = TRUE;
```

SELECT name
FROM Student
WHERE (father_working_status = TRUE AND mother_working_status = FALSE)
  OR (father_working_status = FALSE AND mother_working_status = TRUE);

10. Count of students who have paid their fees and not paid their fees.



```sql
SELECT fees_paid, COUNT(*) AS Student_Count
FROM Student
GROUP BY fees_paid;
```

```
SELECT fees_paid, COUNT(*) AS Student_Count
FROM Student
GROUP BY fees_paid;
```