NAME:AVINASH BHIWALE ROLL NO.:T1851050 PRN NO.:71901284J DIV:A

Assignment No. 7

Aim: Implement nested sub queries. Perform a test for set membership (in, not in), set comparison (<some, >=some, <all etc.) and set cardinality (unique, not unique).

Objective:

- To learn different types of Joins.
- To implement different sub queries.

Theory :

MySQL JOINS are used with SELECT statement. It is used to retrieve data from multiple tables. It is performed whenever you need to fetch records from two or more tables.

There are three types of MySQL joins:

- MySQL INNER JOIN (or sometimes called simple join)
- MySQL LEFT OUTER JOIN (or sometimes called LEFT JOIN)
- MySQL RIGHT OUTER JOIN (or sometimes called RIGHT JOIN)

MySQL Inner JOIN (Simple Join)

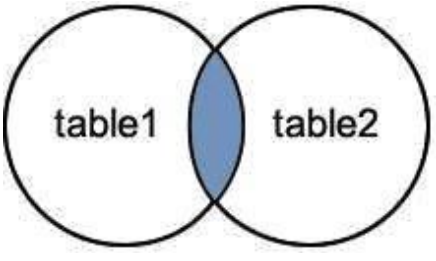
The MySQL INNER JOIN is used to return all rows from multiple tables where the join condition is satisfied. It is the most common type of join.

Syntax:

```
SELECT columns
FROM table1

INNER JOIN table2
ON table1.column = table2.column;
```

Image representation:



Let's take an example:

Consider two tables "officers" and "students", having the following data.
Execute the following query:

```
SELECT officers.officer_name, officers.address, students.course_name
FROM officers
INNER JOIN students
ON officers.officer_id = students.student_id;
```

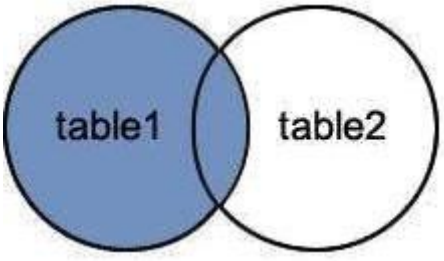
MySQL Left Outer Join

The LEFT OUTER JOIN returns all rows from the left hand table specified in the ON condition and only those rows from the other table where the join condition is fulfilled.

Syntax:

```
SELECT columns
FROM table1
LEFT [OUTER] JOIN table2
ON table1.column = table2.column;
```

Image representation:



Let's take an example:

Consider two tables "officers" and "students", having the following data.
Execute the following query:

```
SELECT officers.officer_name, officers.address, students.course_name
FROM officers
LEFT JOIN students
ON officers.officer_id = students.student_id;
```

MySQL Right Outer Join

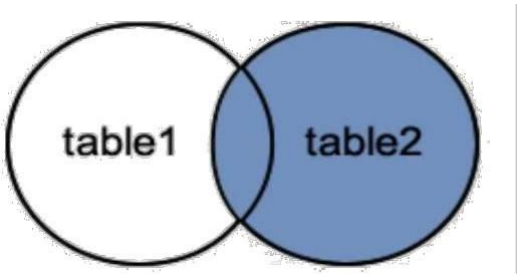
The MySQL Right Outer Join returns all rows from the RIGHT-hand table specified in the ON condition and only those rows from the other table where he join condition is fulfilled.

Syntax:

```
SELECT columns
```

```
FROM table1
RIGHT [OUTER] JOIN table2
ON table1.column = table2.column;
```

Image representation:



Let's take an example:

Consider two tables "officers" and "students", having the following data.
Execute the following query:

```
SELECT officers.officer_name, officers.address,
students.course_name,
students.student_name
FROM officers
RIGHT JOIN students
ON officers.officer_id = students.student_id;
```

SPECIAL OPERATOR:

MySQL IN Condition

The MySQL IN condition is used to reduce the use of multiple OR conditions in a SELECT, INSERT, UPDATE and DELETE statement.

Syntax:

expression IN (value1, value2,.....value_n);

Parameters:

expression: It specifies a value to test.
value1, value2, ...or value_n: These are the values to test against expression. If any of these values matches expression, then the IN condition will evaluate to true. This is a quick method to test if any one of the values matches expression.
Execute the following query:

```
SELECT *

FROM officers
WHERE officer_name IN ('Ajeet', 'Vimal', 'Deepika');
```

MySQL NOT Condition

The MySQL NOT condition is opposite of MySQL IN condition. It is used to negate a condition in a SELECT, INSERT, UPDATE or DELETE statement.

Syntax:

NOT condition

Parameter:

condition: It specifies the conditions that you want to negate.

MySQL NOT Operator with IN condition

Consider a table "officers", having the following data.
Execute the following query:

```
SELECT *

FROM officers
WHERE officer_name NOT IN ('Ajeet','Vimal','Deepika');
```

MySQL IS NULL Condition

MySQL IS NULL condition is used to check if there is a NULL value in the expression. It is used with SELECT, INSERT, UPDATE and DELETE statement.

Syntax:

expression IS NULL

Parameter:

expression: It specifies a value to test if it is NULL
Execute the following query:

```
SELECT *
FROM officers
WHERE officer_name IS NULL;
```

MySQL IS NOT NULL Condition

MySQL IS NOT NULL condition is used to check the NOT NULL value in the expression. It is used with SELECT, INSERT, UPDATE and DELETE statements.

Syntax:

expression IS NOT NULL

Parameter:

expression: It specifies a value to test if it is not NULL value.

Execute the following query:

```
SELECT *
FROM officers
WHERE officer_name IS NOT NULL;
```

SET OPERATORS:

The Set operator combines the result of 2 queries into a single result. The following are the operators:

- Union

Union all

- Intersect
- Minus

LAB PRACTICE ASSIGNMENT:

Consider the following table structure for this assignment:

- Location(Location_Id integer, Reginal_Group varchar(20))
- Department (Department_Id, Name, Location_Id)
- Job(Job_Id Integer,Function Varchar(30))
- Employee(Employee_Id, Lastname ,Firstname,Middlename, Job_Id, Manager_Id, Hiredate, Salary, Department_Id)
- Loan(Employee_Id, Firstname , Loan_Amount)

LOCATION TABLE

LOCATION_ID	REGINAL_GROUP
122	New York
123	Dallas
124	Chicago
167	Boostan

DEPARTMENT TABLE

DEPARTMENT_ID	NAME	LOCATION_ID
10	Accounting	122

20	Research	124
30	Sale	123
40	Operation	164

JOB TABLE

JOB_ID	FUNCTION
667	Cleark
668	Staff
669	Analyst
670	Saleperson
671	Manager
672	President

EMPLOYEE TABLE

EMPLOYEE_ID	LASTNAME	FIRSTNAME	MIDDLENAME	JOB_ID	MANAGER_ID	HIREDATE	SALARY	DEPARTMENT_ID
7369	Smith	Jon	Q	667	7902	17-DEC-84	800	10
7499	Allen	Kevin	J	670	7698	20-FEB-85	1600	20
7505	Doyle	Jean	K	671	7839	04-APR-85	2850	20
7506	Dennis	Lynn	S	671	7839	15-MAY-85	2750	30
7507	Baker	Leslie	D	671	7839	10-JUN-85	2200	40
7521	wark	cynthia	D	670	7698	22-FEB-85	1250	10

Perform the following queries on the above table:

1) Perform all types of JOIN operations on Employee and Loan tables.

2) Perform all types of set operations on Employee and Loan tables.
3) Find out no.of employees working in **"**Sales**"** department
4) Find out the employees who are not working in department 10 or 30.
5) List out employee id, last name in descending order based on the salary column.
6) How many employees who are working in different departments wise in the organization
7) List out the department id having at least four employees
8) Display the employee who got the maximum salary.
9) Update the employees**'** salaries, who are working as Clerk on the basis of 10%.
10) Delete the employees who are working in accounting department.
11) Find out whose department has not employees.
12) List out the department wise maximum salary, minimum salary, average salary of the employees
13) How many employees who are joined in 1985.
14) Display the employees who are working in **"**New York**"**
15) List our employees with their department names

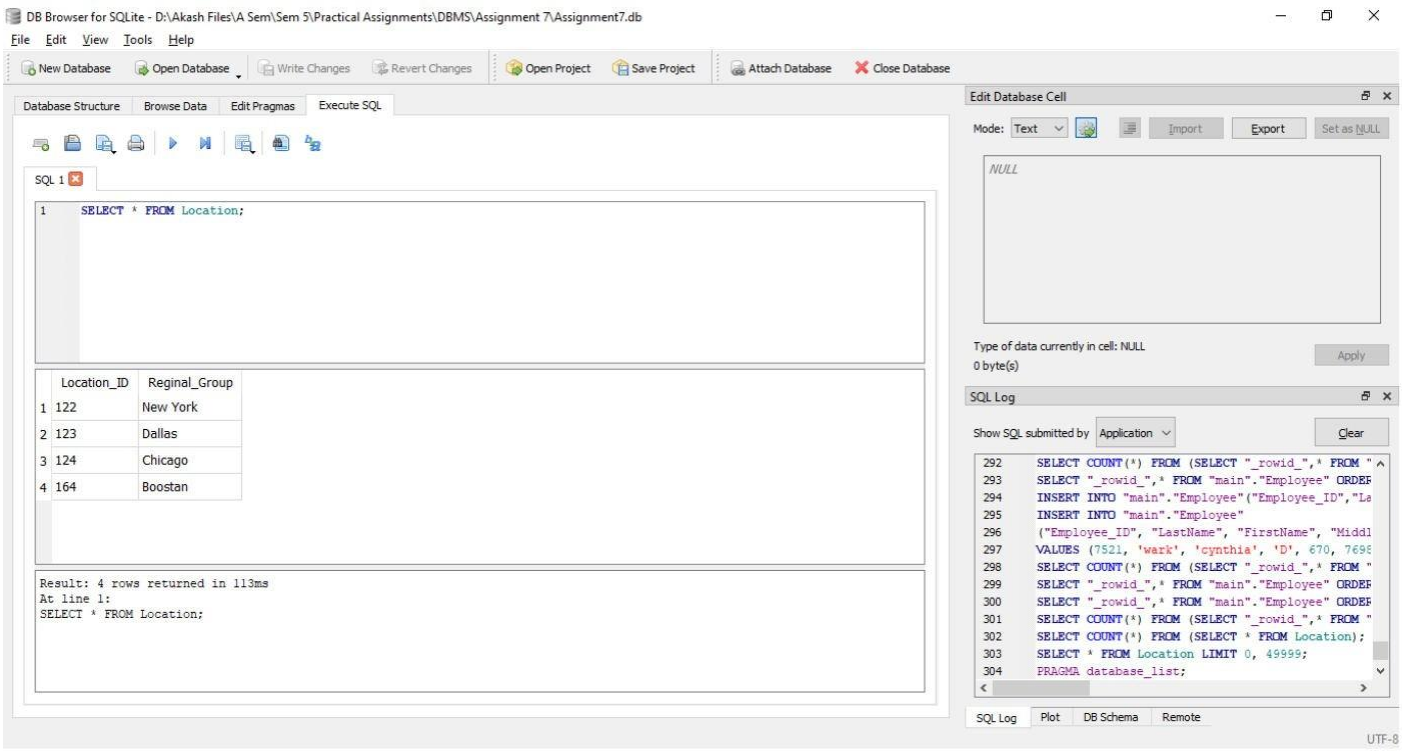
Conclusion:

We have implemented join, set operations, set cardinalities and nested sub queries.

Output:

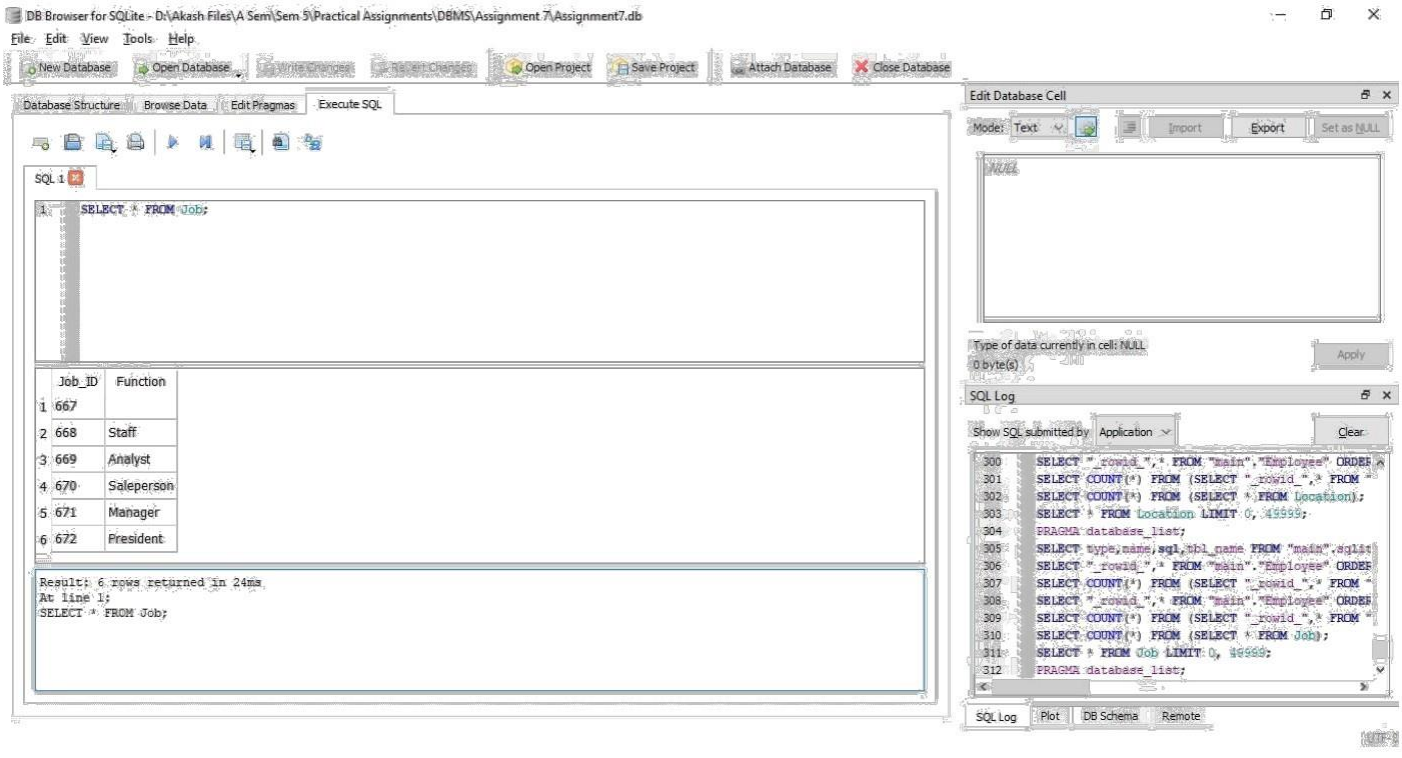
1. **CREATE TABLE:**
   a) **LOCATION:**

```
CREATE TABLE "Location" ( "Location_ID"
    INTEGER NOT NULL, "Reginal_Group"
    VARCHAR(20), PRIMARY
    KEY("Location_ID")
);
```



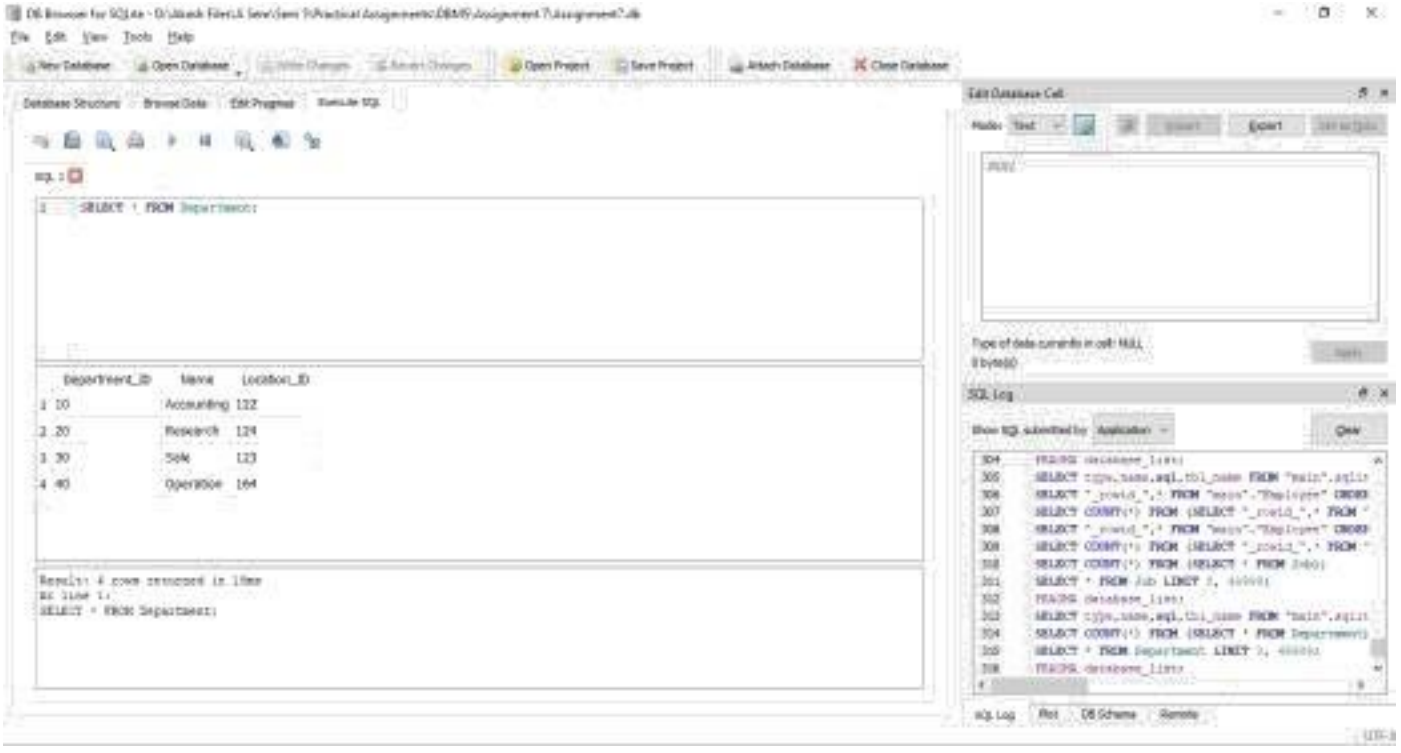
b) JOB:

```
CREATE TABLE "Job" (
    "Job_ID" INTEGER NOT NULL,
    "Function" TEXT NOT NULL,
    PRIMARY KEY("Job_ID")
);
```



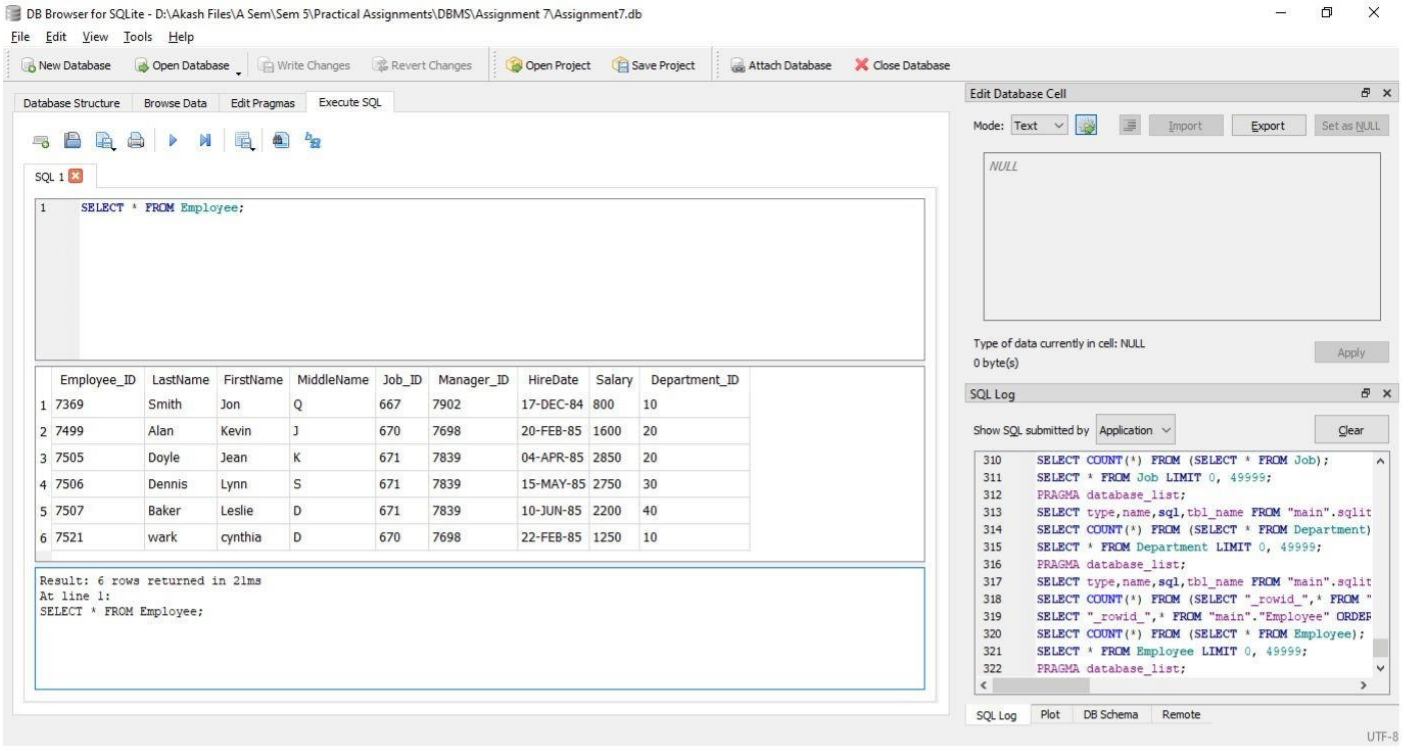
c) DEPARTMENT:

```
CREATE TABLE "Department" (
"Department_ID" INTEGER NOT NULL,
"Name"    TEXT NOT NULL,
"Location_ID"     INTEGER,
PRIMARY KEY("Department_ID"),
FOREIGN KEY("Location_ID") REFERENCES Location);
```



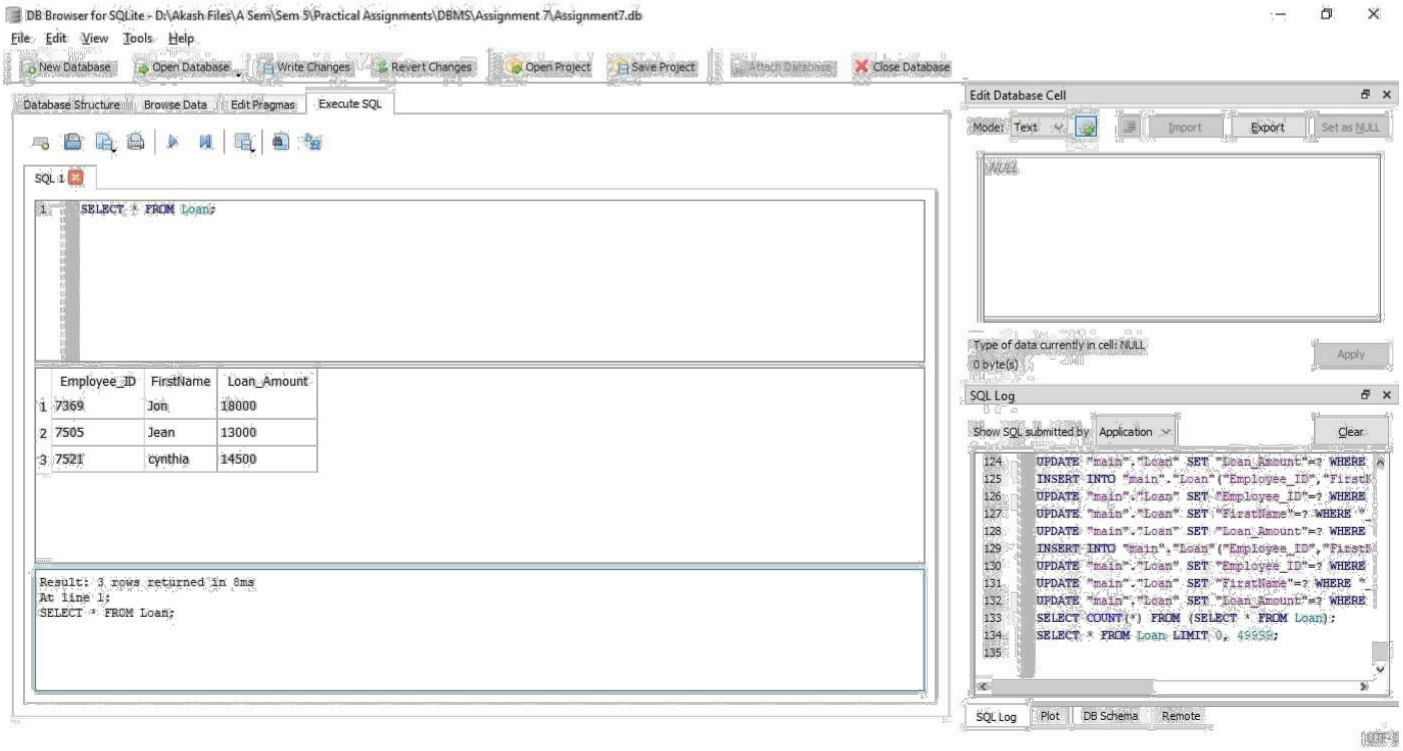
d) EMPLOYEE:

```
CREATE TABLE "Employee" (
    "Employee_ID"    INTEGER NOT NULL,
    "LastName"       TEXT,
    "FirstName"      TEXT NOT NULL,
    "MiddleName"     TEXT,
    "Job_ID" INTEGER NOT NULL,
    "Manager_ID"     INTEGER NOT NULL,
    "HireDate" TEXT NOT NULL, "Salary"
    INTEGER NOT NULL, "Department_ID"
    INTEGER NOT NULL, PRIMARY
    KEY("Employee_ID","Manager_ID"),
    FOREIGN KEY("Department_ID") REFERENCES "Department"("Department_ID"),
    FOREIGN KEY("Job_ID") REFERENCES "Job"("Job_ID")
);
```



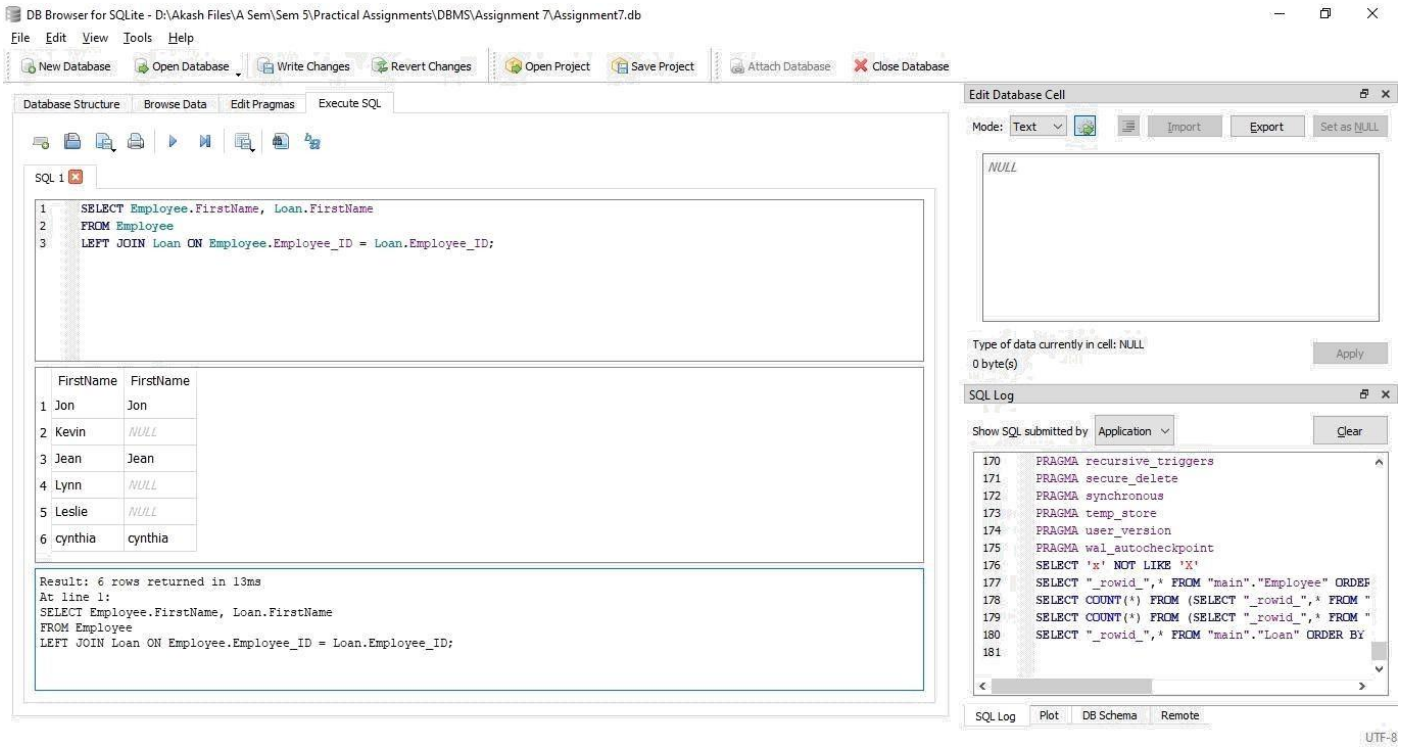
e) LOAN:

```
CREATE TABLE "Loan" (
    "Employee_ID"    INTEGER NOT NULL,
    "FirstName"      TEXT NOT NULL,
    "Loan_Amount" INTEGER NOT NULL);
```



2. JOIN:

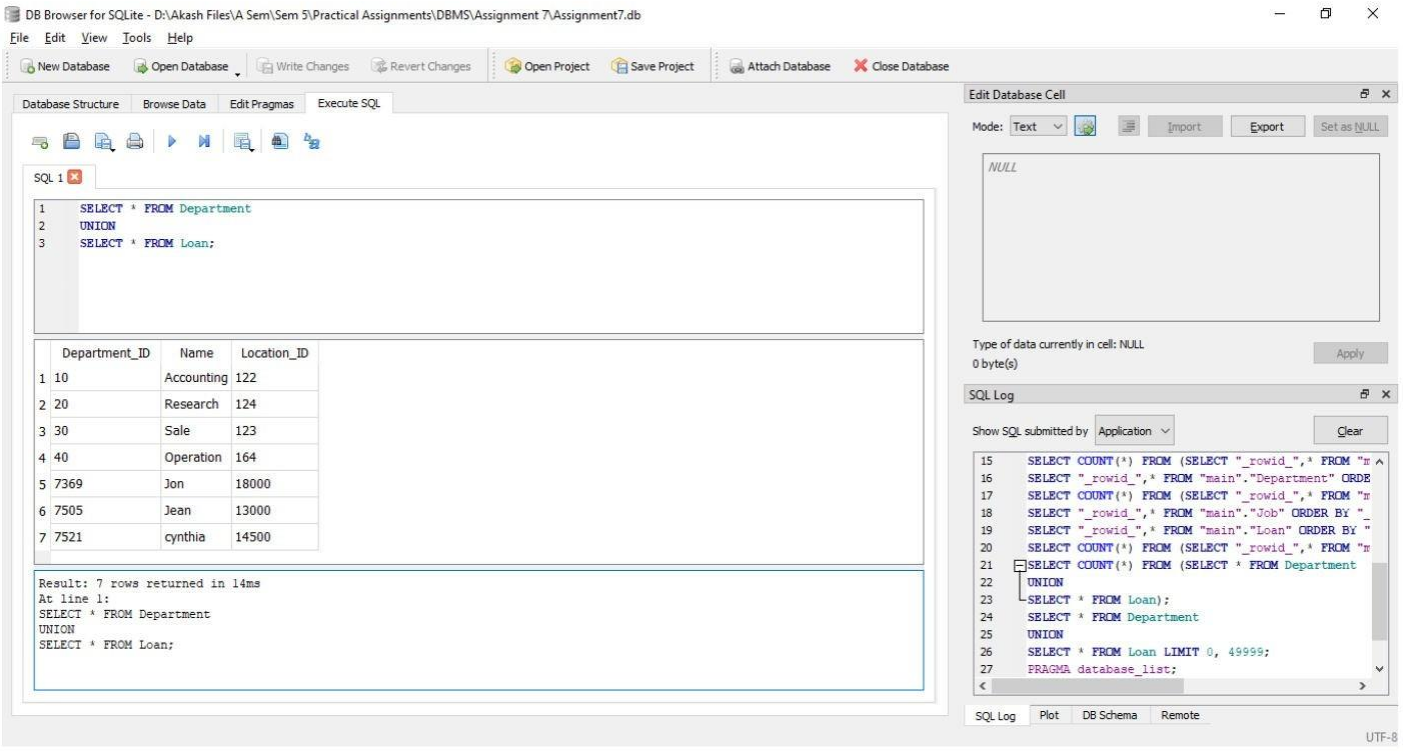
```
SELECT Employee.FirstName, Loan.FirstName
FROM Employee
LEFT JOIN Loan ON Employee.Employee_ID = Loan.Employee_ID;
```



3. SET OPERATIONS:

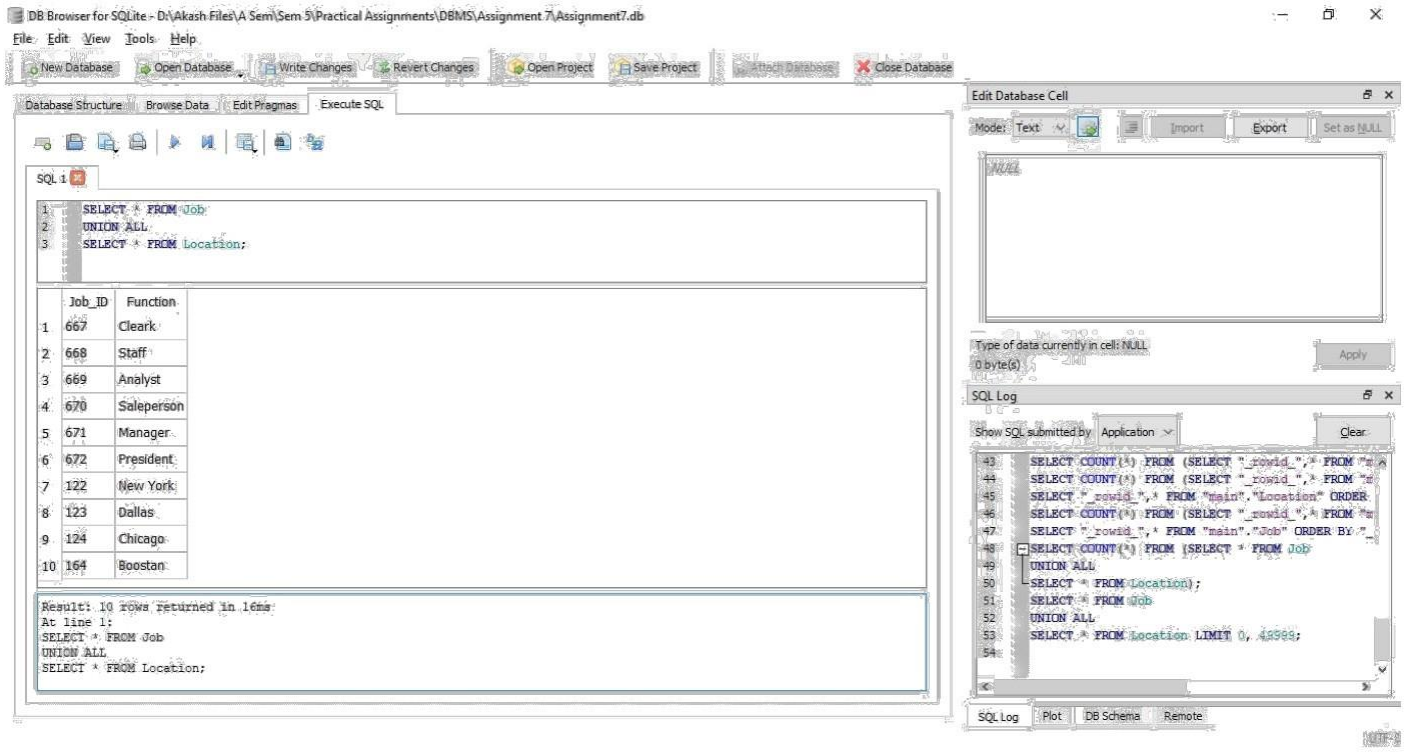
a) UNION:

```
SELECT * FROM Department
UNION
SELECT * FROM Loan;
```



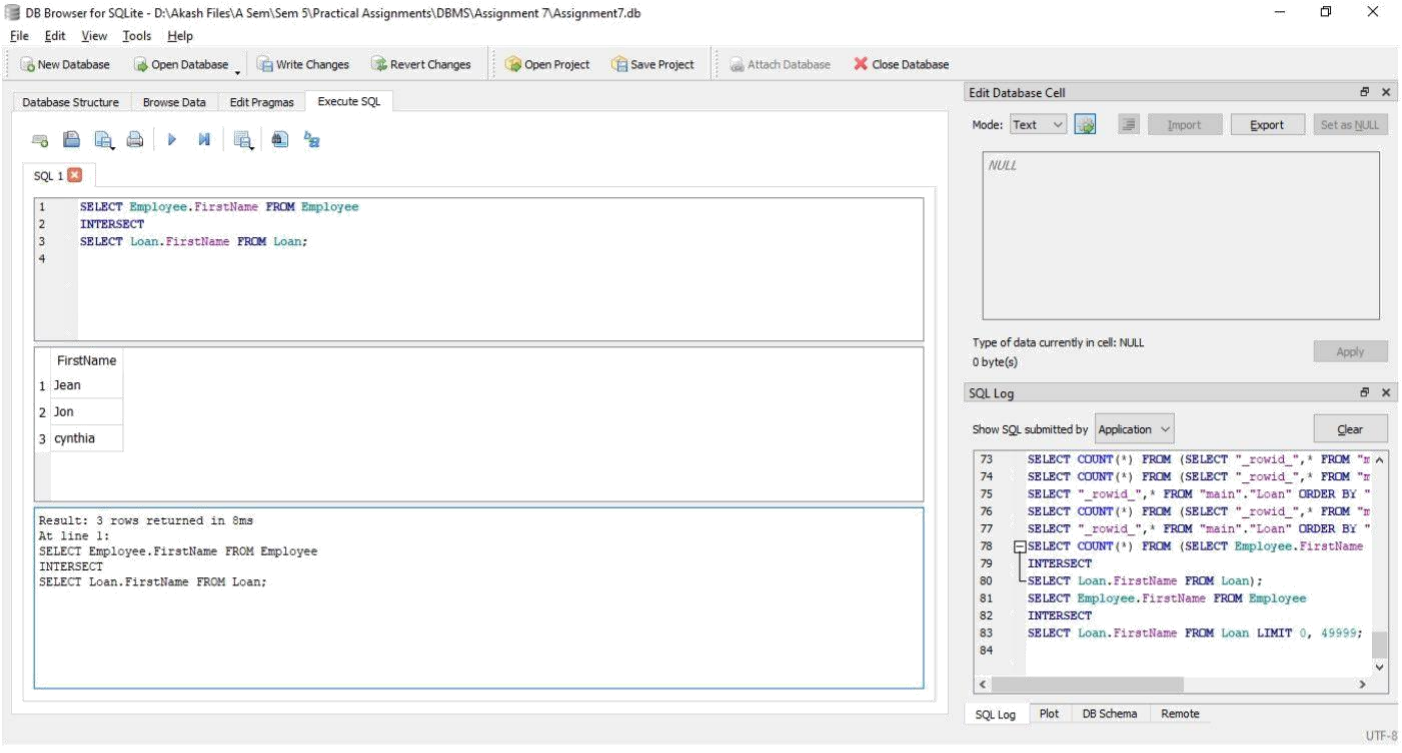
b) UNION ALL:

SELECT * FROM Job
UNION ALL
SELECT * FROM Location;



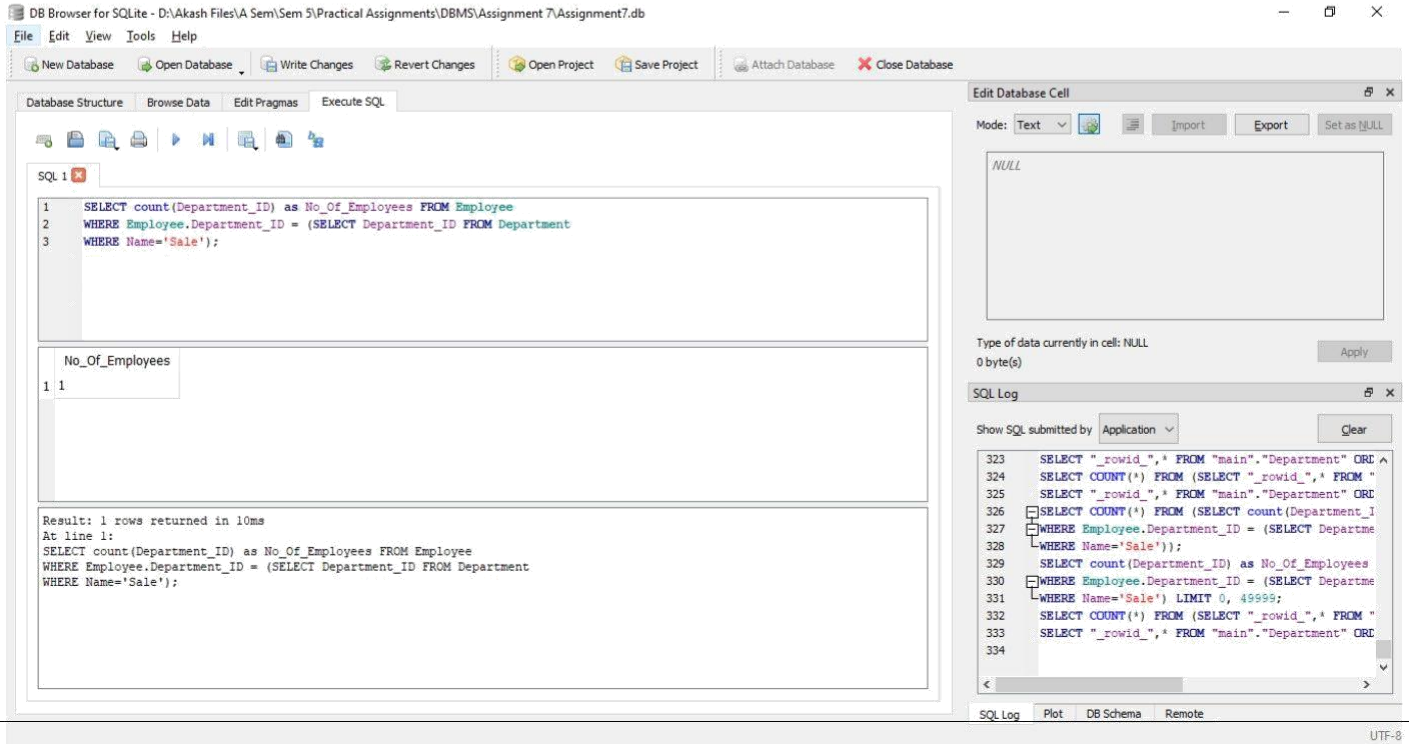
c) INTERSECT:

SELECT Employee.FirstName FROM Employee
INTERSECT
SELECT Loan.FirstName FROM Loan;



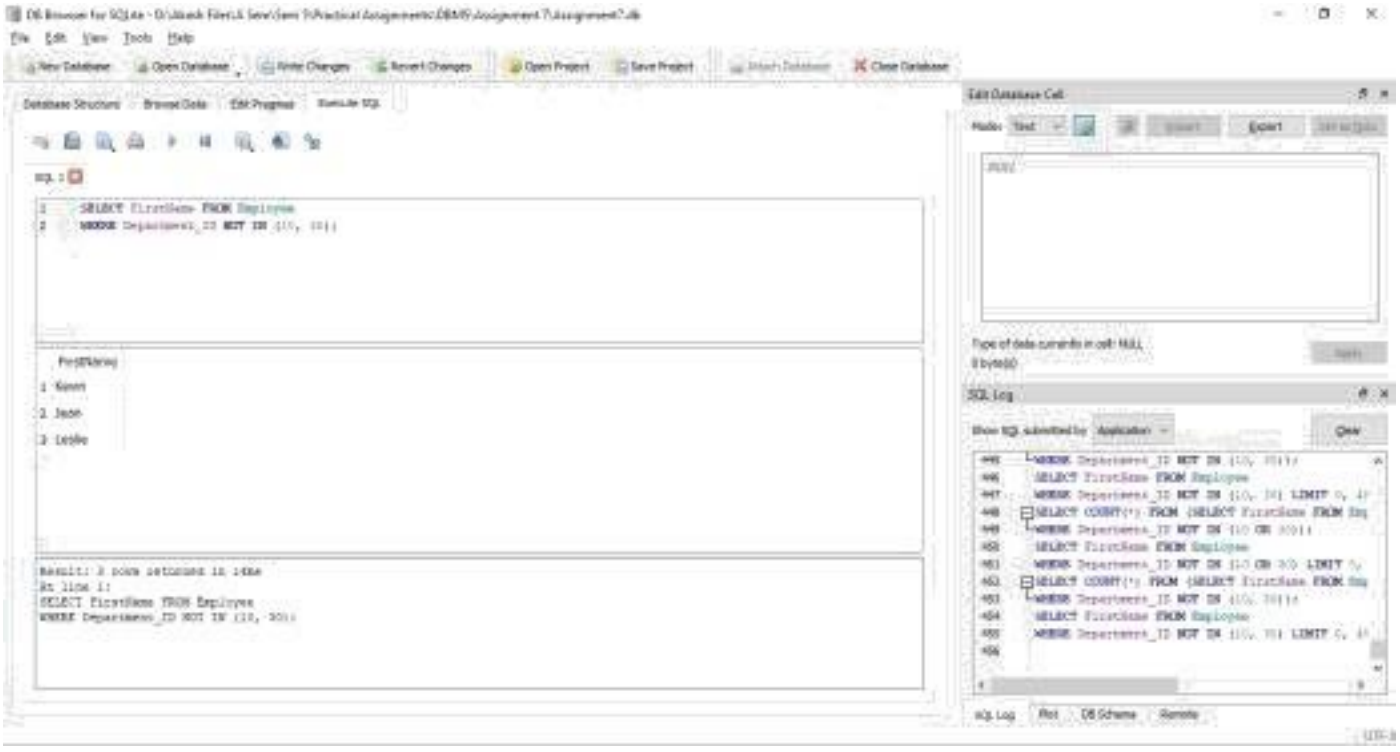
4. No. of employees working in sales department:

SELECT count(Department_ID) as No_Of_Employees FROM Employee
WHERE Employee.Department_ID = (SELECT Department_ID FROM
Department WHERE Name='Sale');



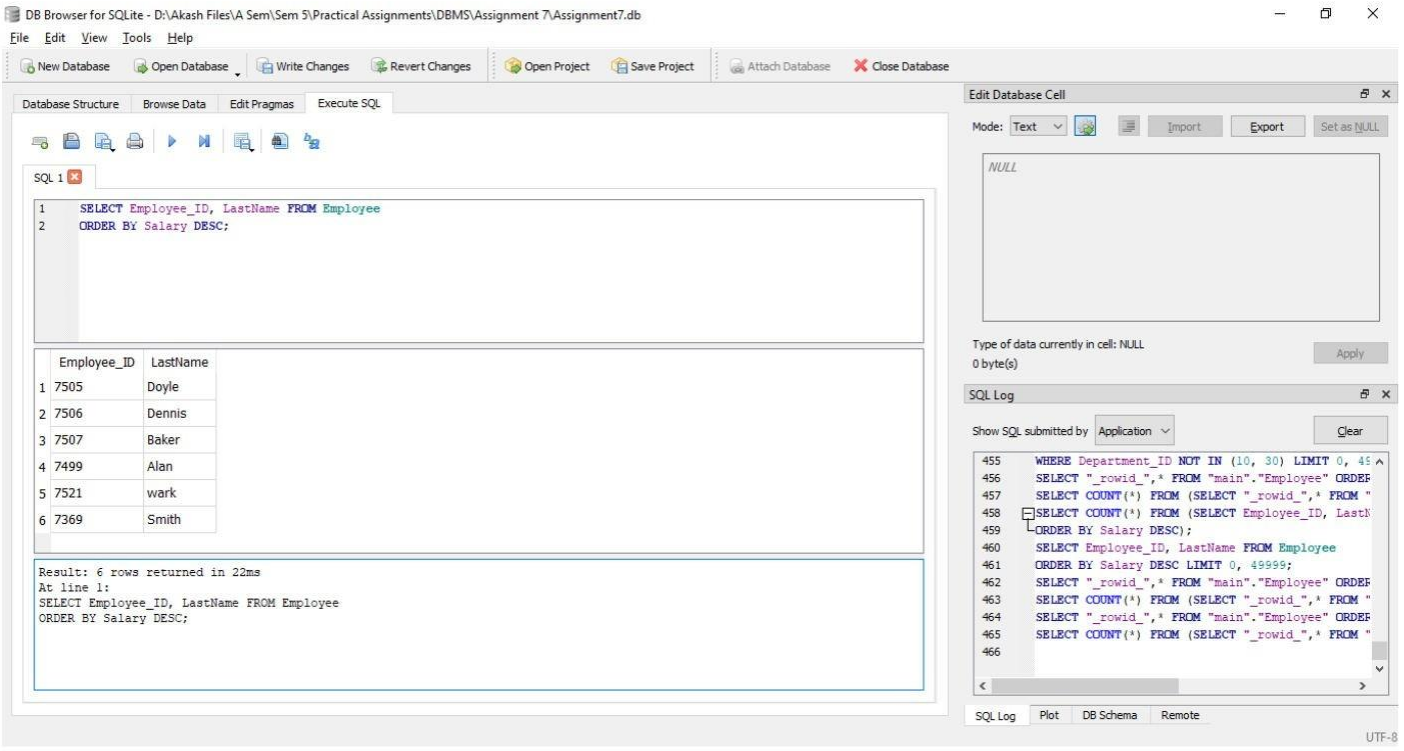
5. Find out the employees who are not working in department 10 or 30.

SELECT FirstName FROM Employee
WHERE Department_ID NOT IN (10, 30);



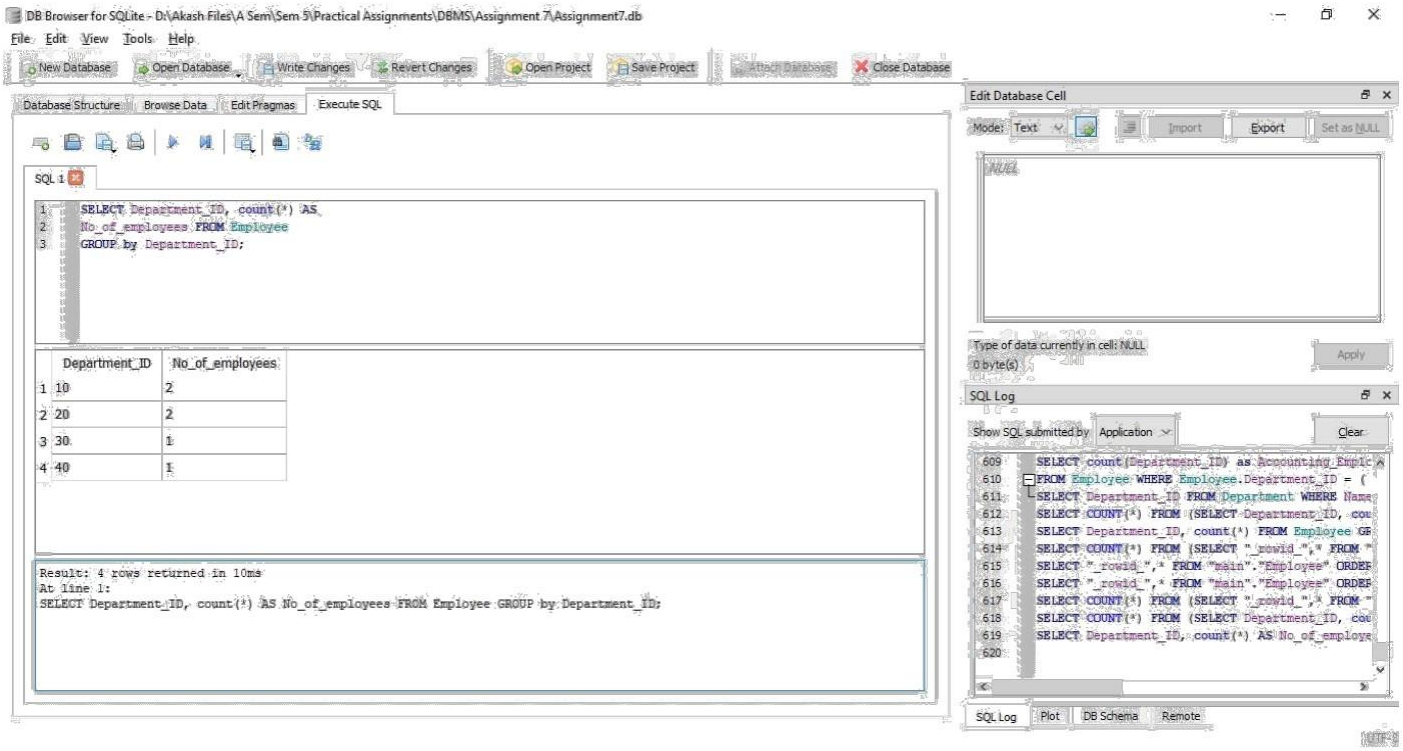
6. List out employee id, last name in descending order based on the salary column.

SELECT Employee_ID, LastName FROM
Employee ORDER BY Salary DESC;



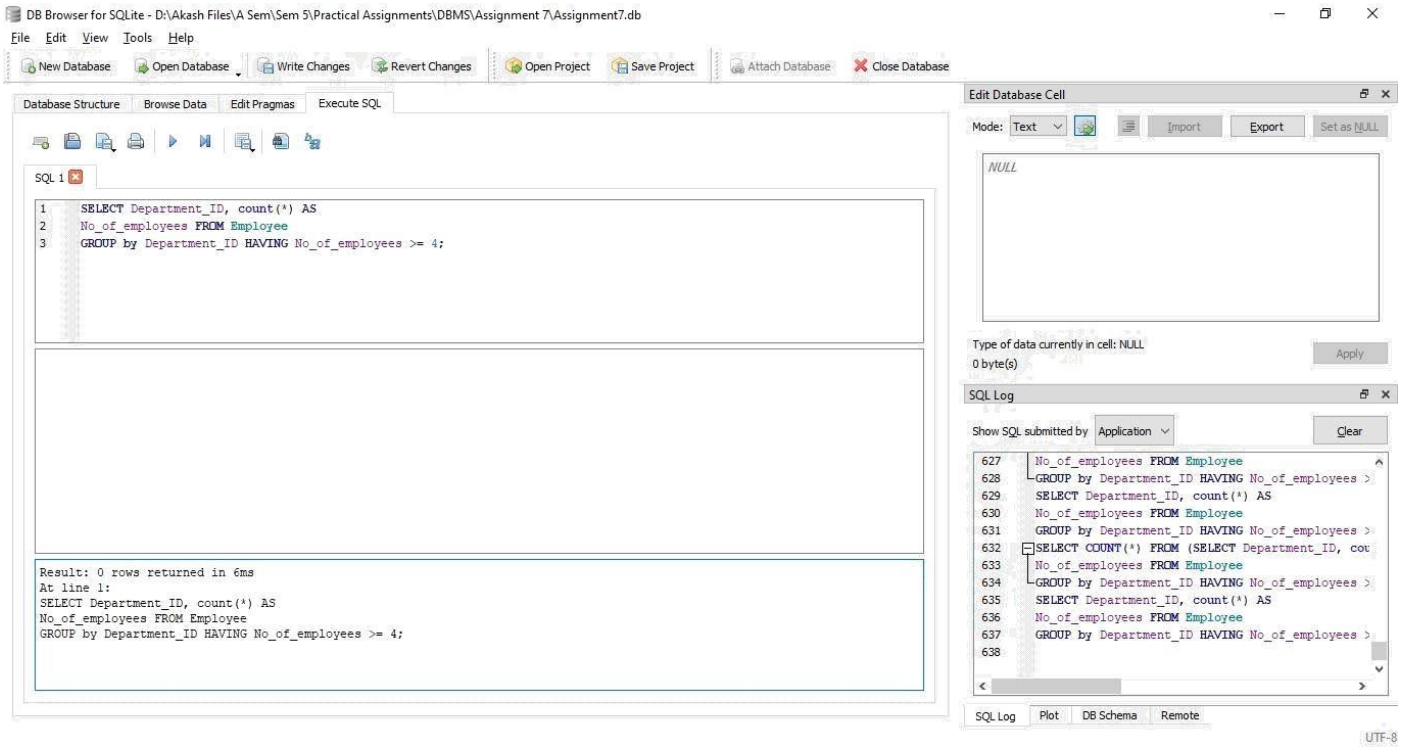
7. How many employees who are working in different departments wise in the organization

SELECT Department_ID, count(*) AS
No_of_employees FROM Employee
GROUP by Department_ID;



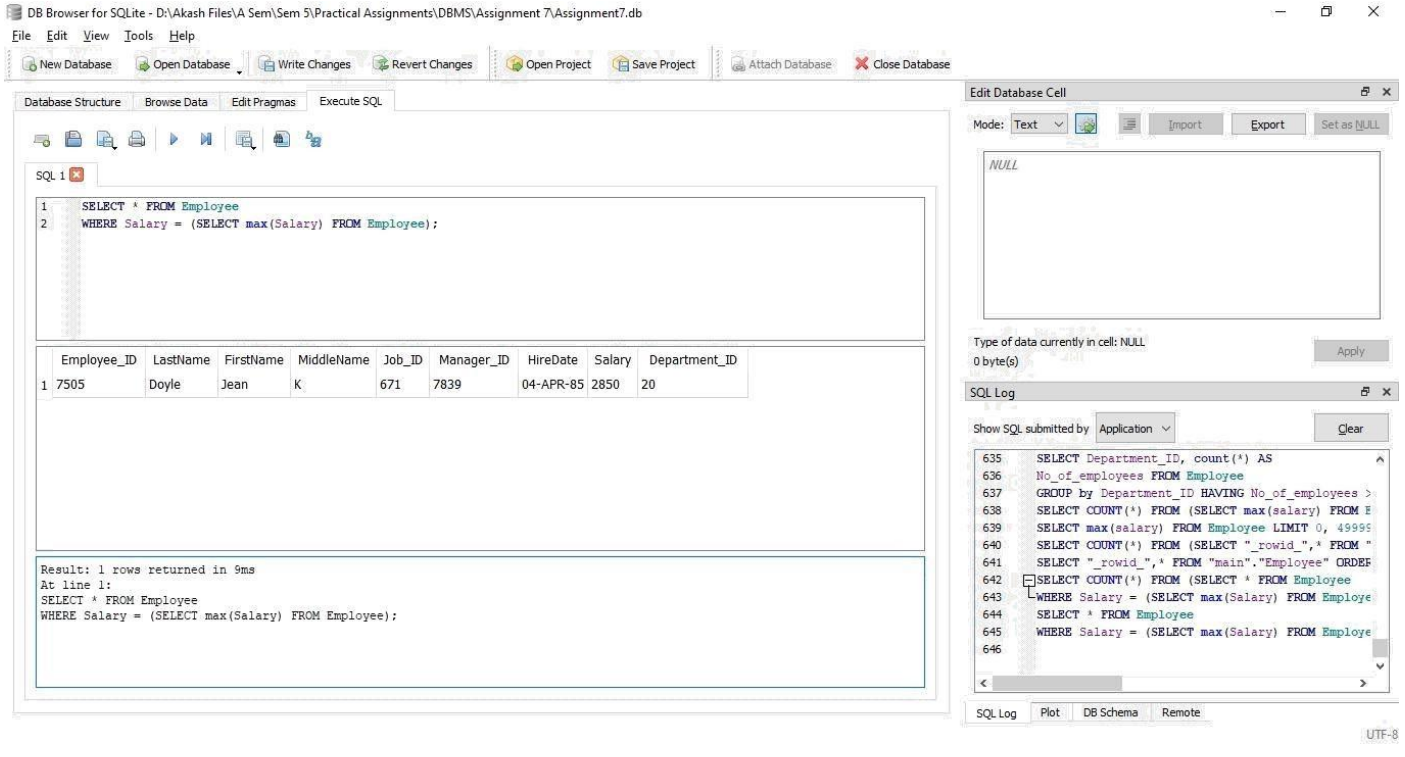
8. List out the department id having at least four employees

```
SELECT Department_ID, count(*) AS
No_of_employees FROM Employee
GROUP by Department_ID HAVING No_of_employees >= 4;
```



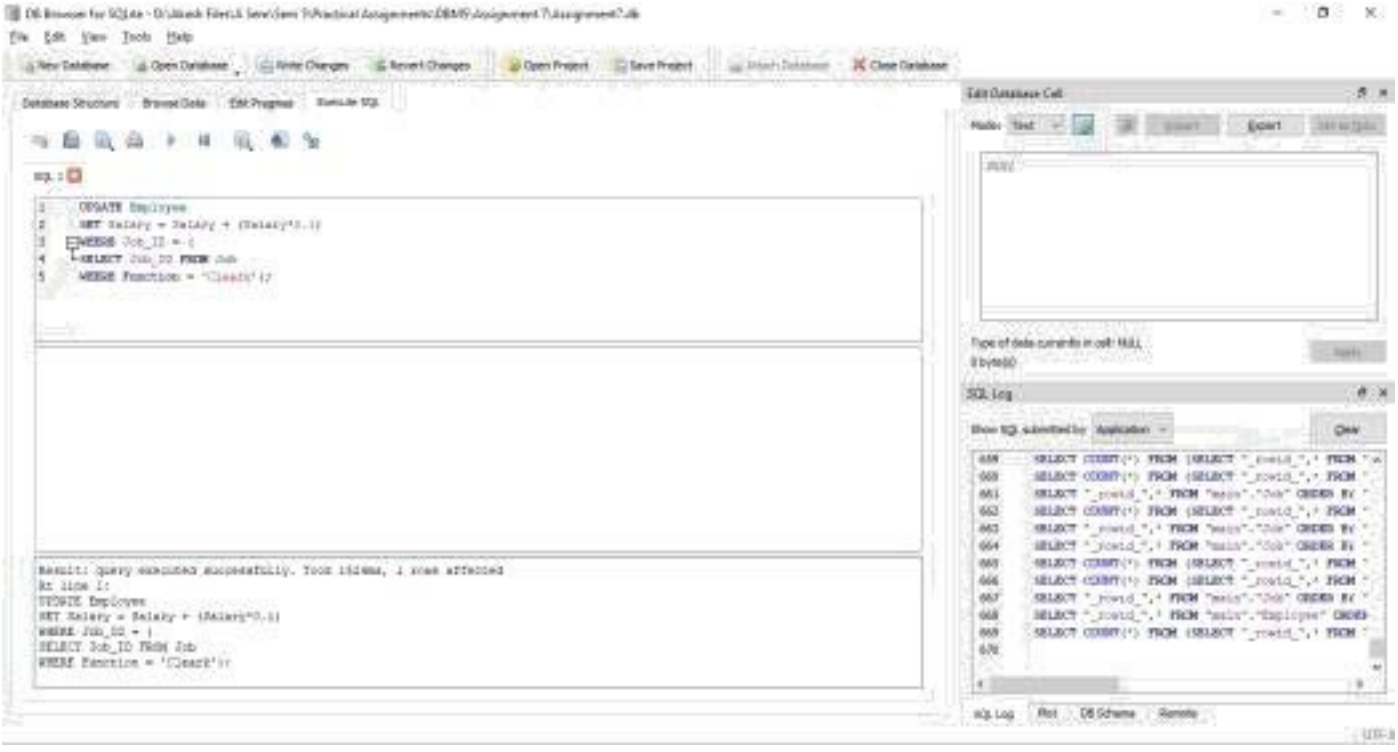
9. Display the employee who got the maximum salary.

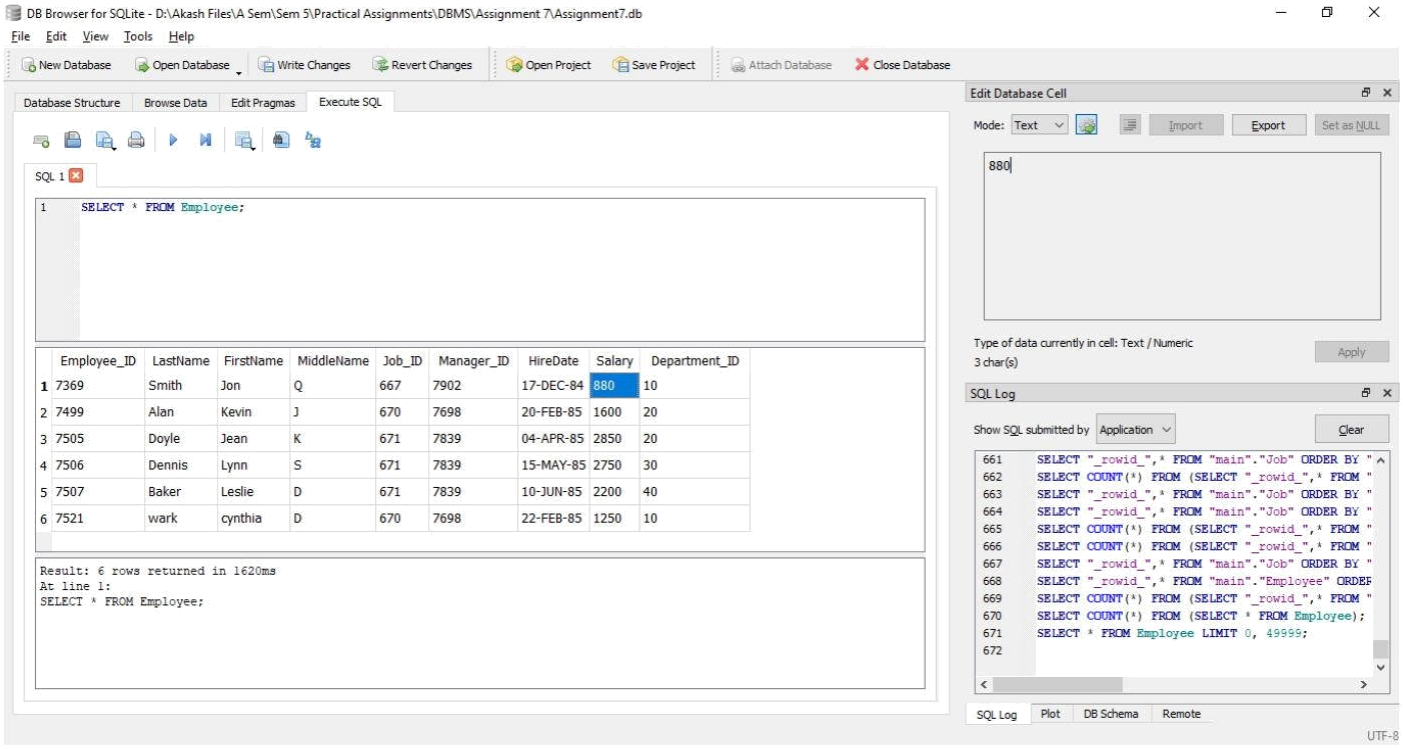
```
SELECT * FROM Employee
WHERE Salary = (SELECT max(Salary) FROM Employee);
```



10. Update the employees' salaries, who are working as Clerk on the basis of 10%.

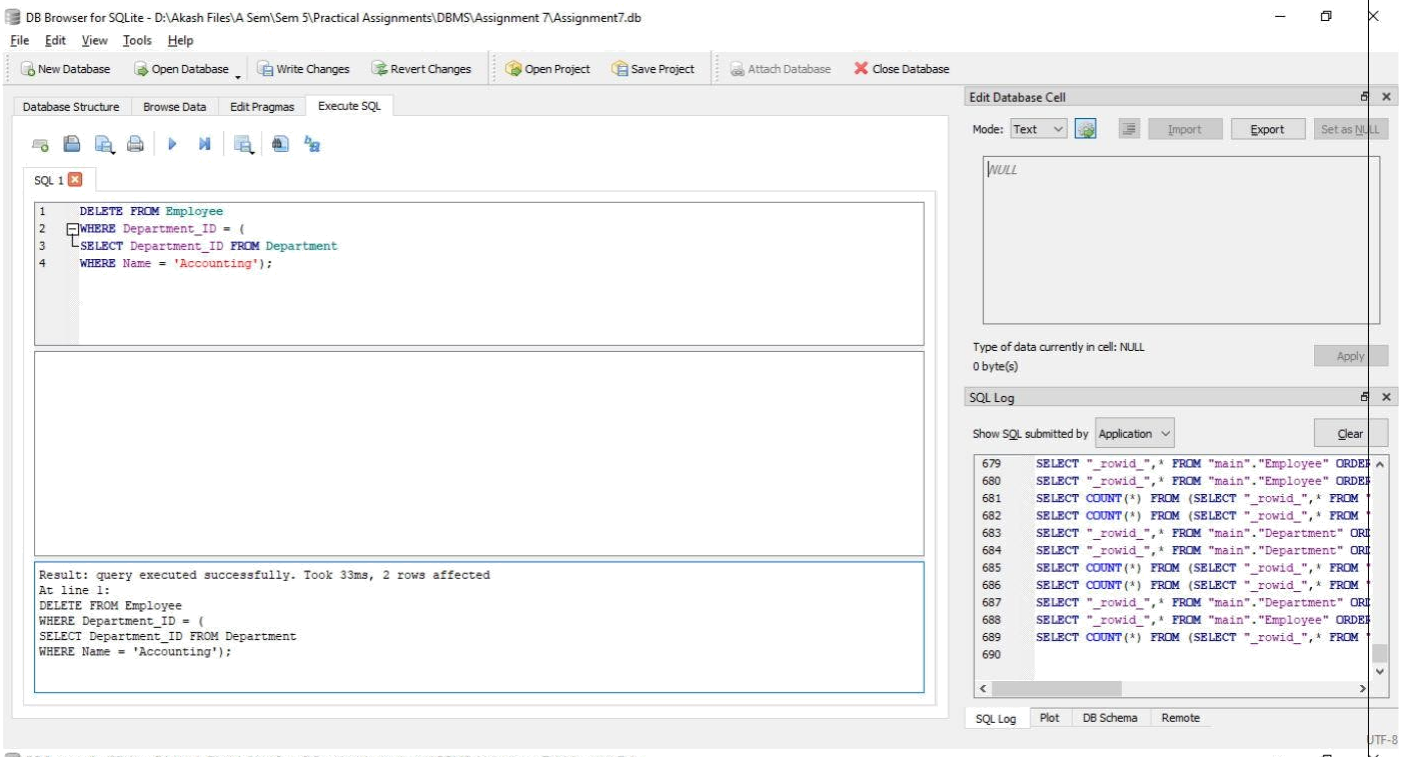
```
UPDATE Employee
SET Salary = Salary + (Salary*0.1)
WHERE Job_ID = (
SELECT Job_ID FROM Job
WHERE Function = 'Cleark');
```

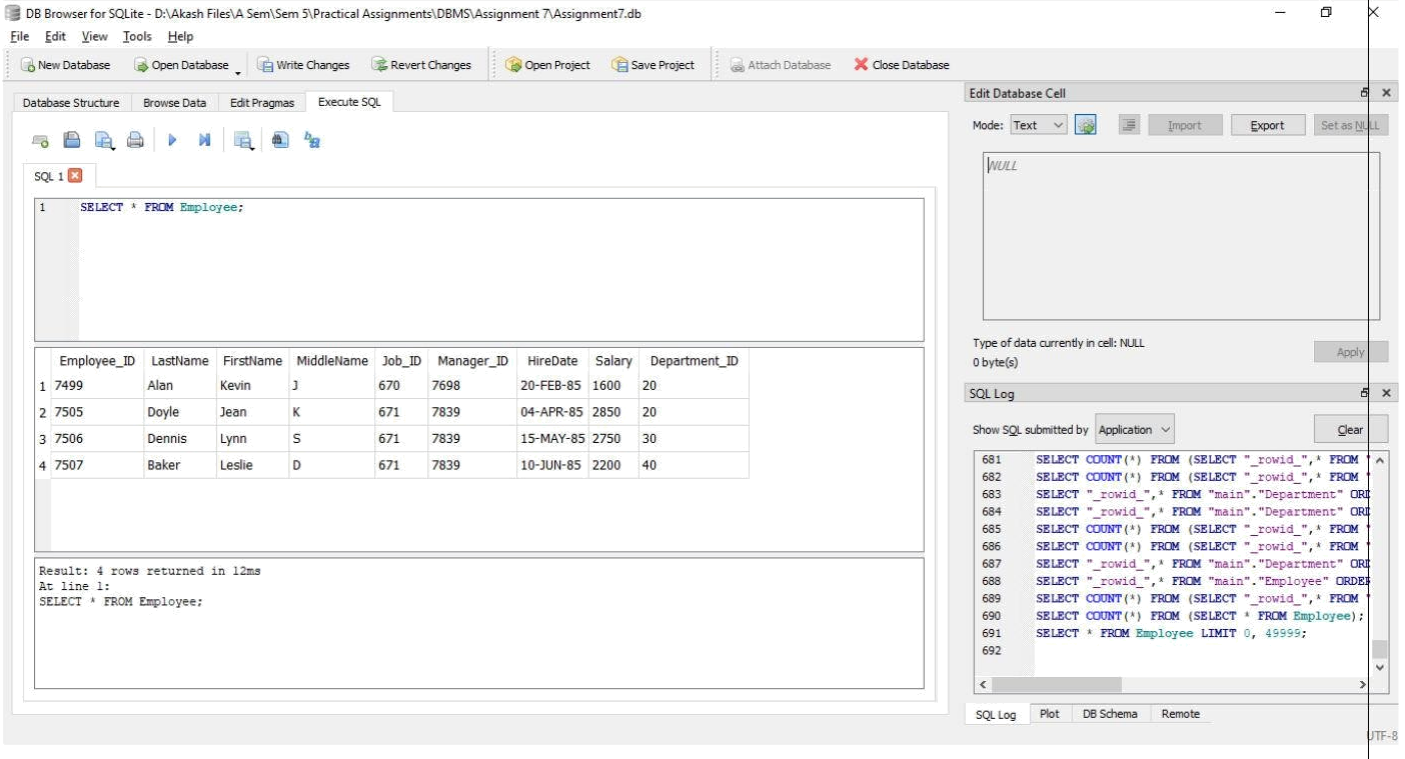




11. Delete the employees who are working in accounting department.

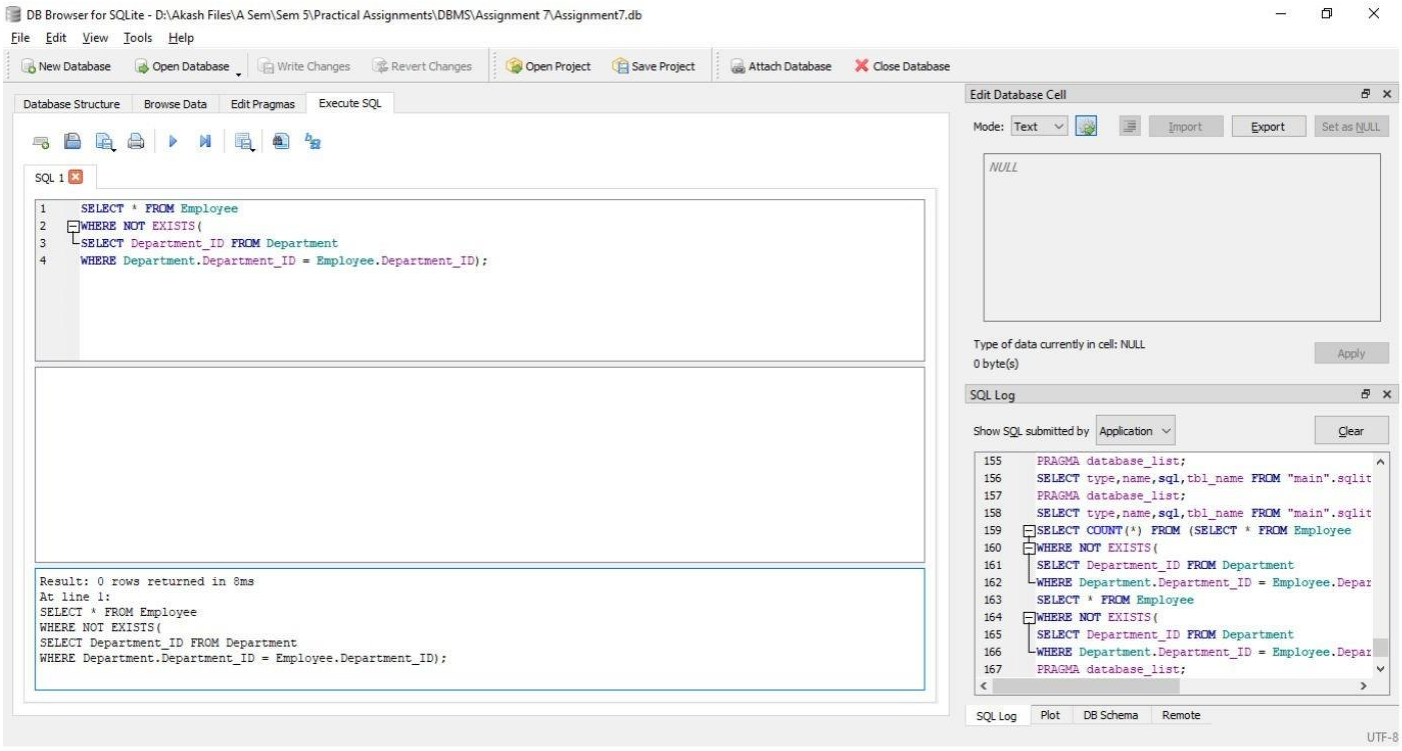
```
DELETE FROM Employee
WHERE Department_ID = (
SELECT Department_ID FROM Department
WHERE Name = 'Accounting');
```





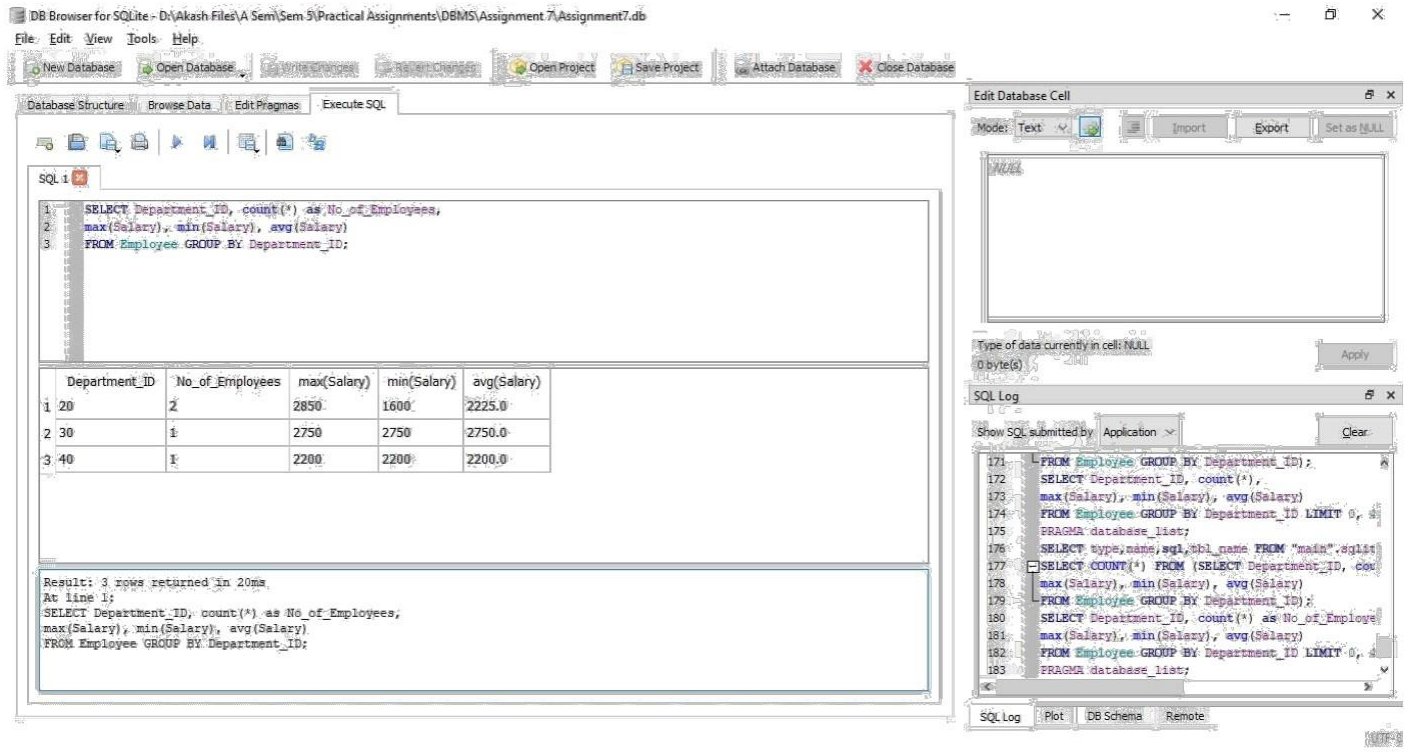
12. Find out whose department has not employees.

```
SELECT * FROM Employee
WHERE NOT EXISTS(
SELECT Department_ID FROM Department
WHERE Department.Department_ID = Employee.Department_ID);
```



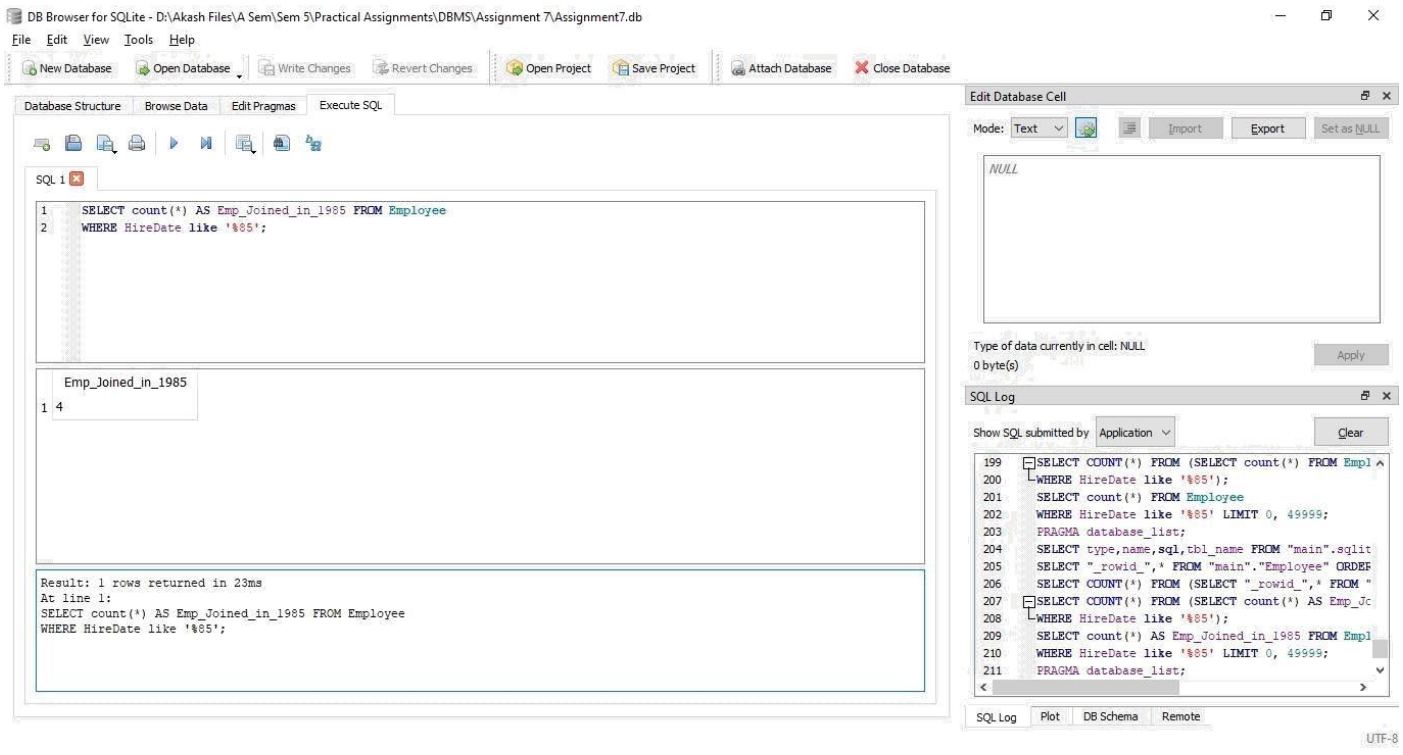
13. List out the department wise maximum salary, minimum salary, average salary of the employees

```
SELECT Department_ID, count(*) as No_of_Employees,
max(Salary), min(Salary), avg(Salary)
FROM Employee GROUP BY Department_ID;
```



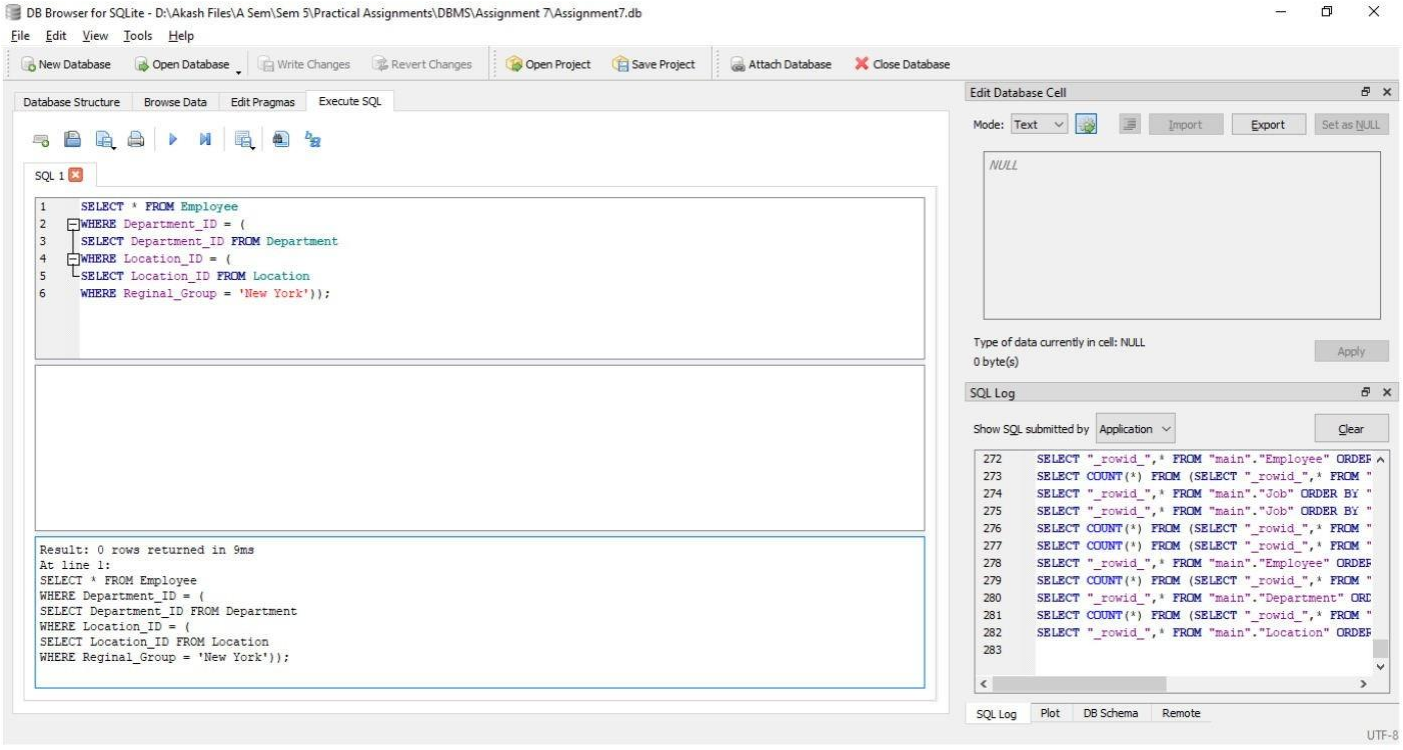
14. How many employees who are joined in 1985.

```
SELECT count(*) AS Emp_Joined_in_1985 FROM Employee
WHERE HireDate like '%85';
```



15. Display the employees who are working in “New York”

```
SELECT * FROM Employee
WHERE Department_ID = (
SELECT Department_ID FROM Department
WHERE Location_ID = (
SELECT Location_ID FROM Location
WHERE Reginal_Group = 'New York'));
```



16. List our employees with their department names

SELECT * FROM Employee, Department
WHERE Employee.Department_ID =
Department.Department_ID GROUP by Name;