CSE 560

Data Models and Query Languages

Project 2

Submitted By

Karan Manchandia

UBIT Name: karanman

Person # 50290755

Problem.1

The relational schema for problem 1 is:

Employee (Ename, Salary);

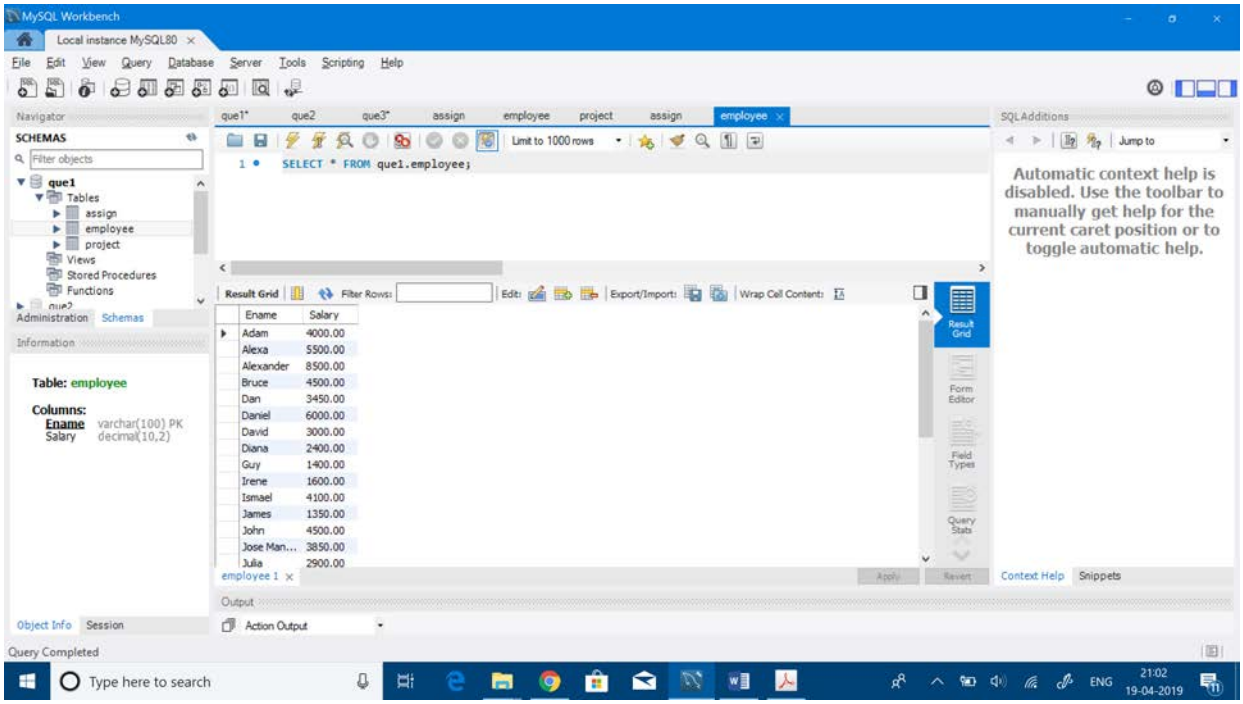
Project (Pname, Agency Budget);

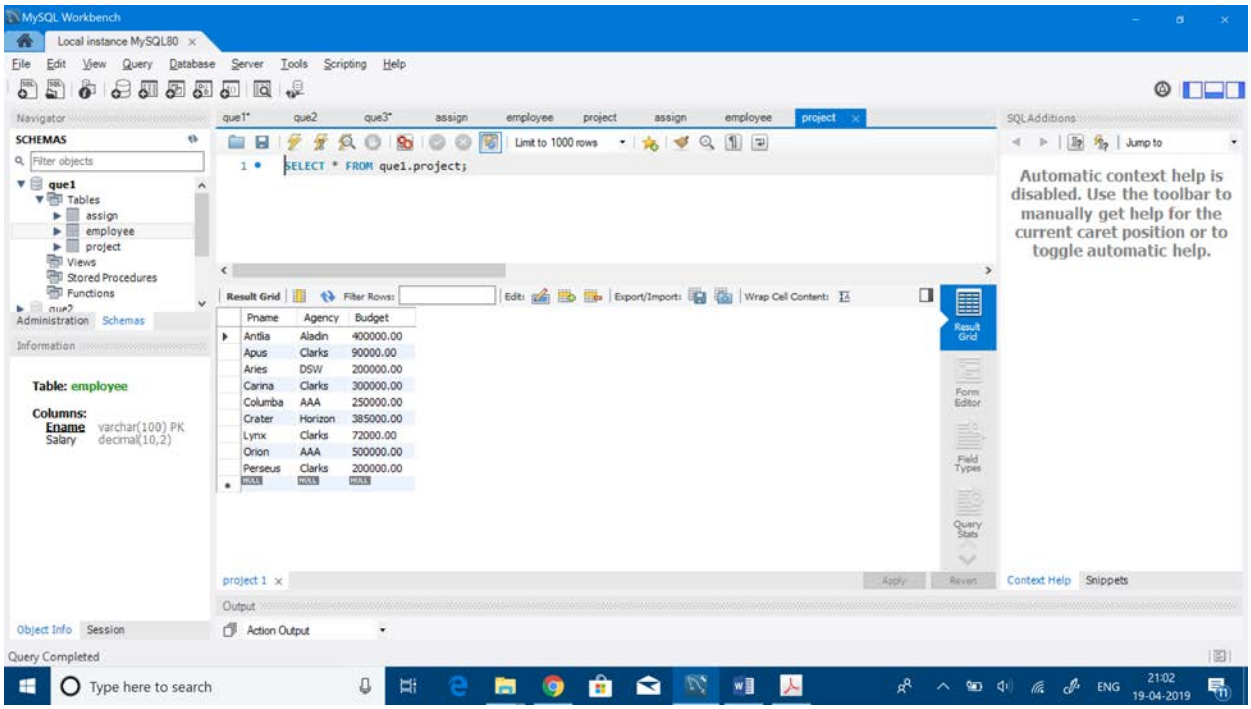
Assign (Pname, Ename), foreign keys: Pname references Project(Pname) and Ename references Employee(Ename).

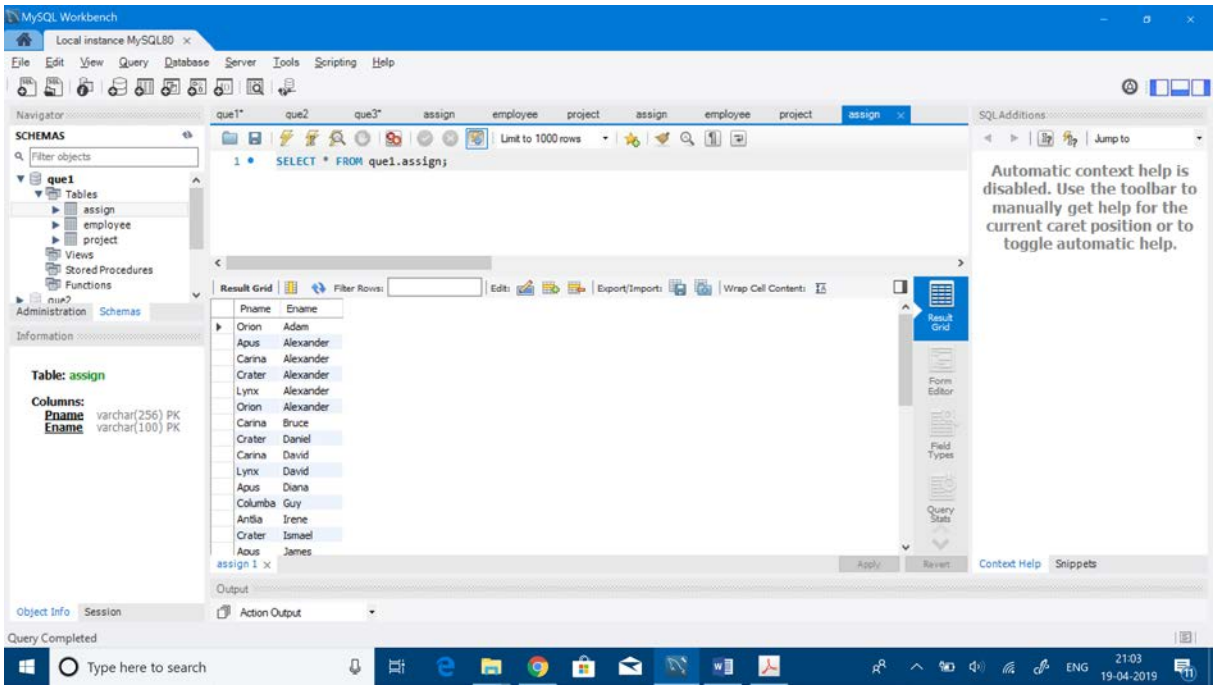
The create Table statements:

The create table SQL queries for the three relations could be found in the submitted .sql file.

The screenshots for the tables created are attached below:





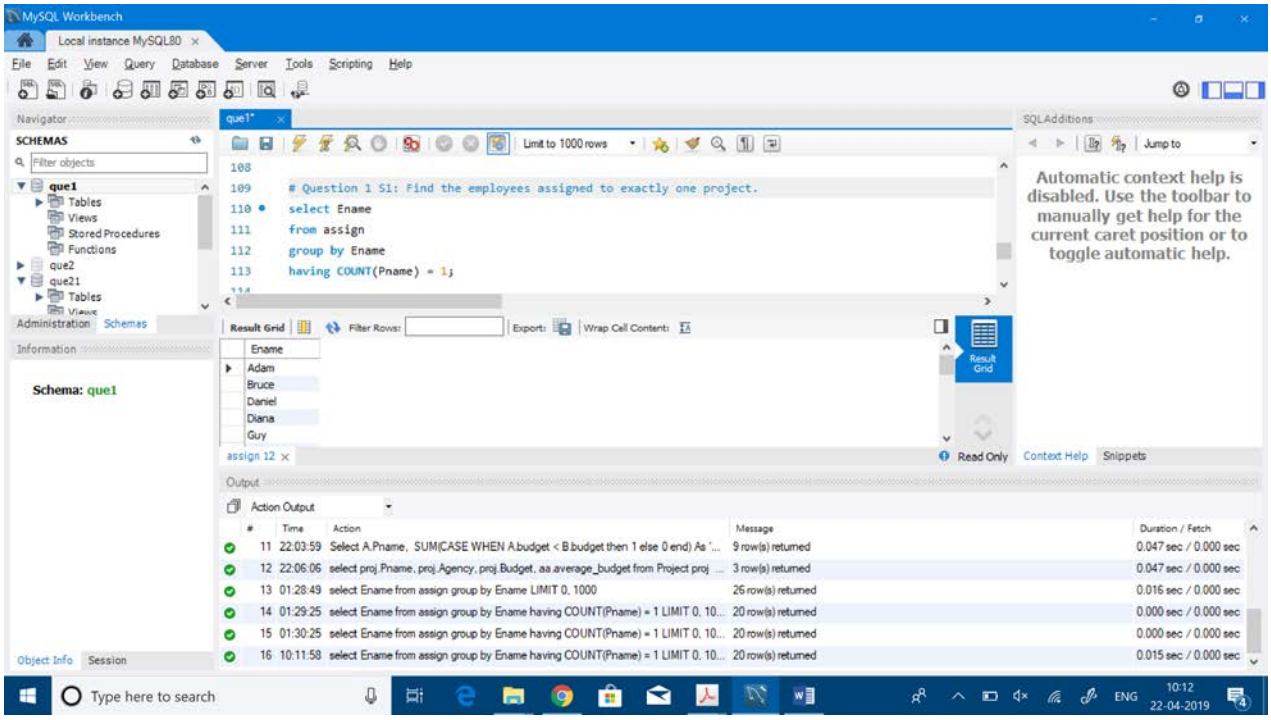


SQL Queries:

S1: Find the employees assigned to exactly one project.

```
select Ename
from assign
group by Ename
having COUNT(Pname) = 1;
```

Screenshot:



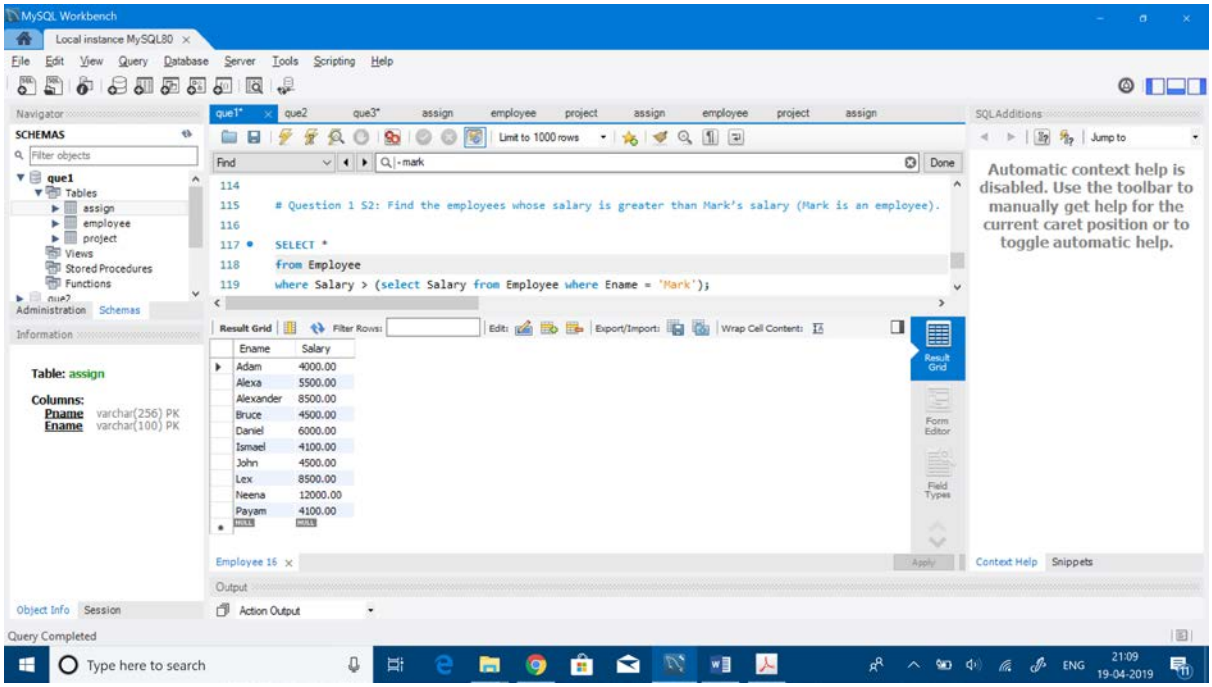
S2: Find the employees whose salary is greater than Mark's salary (Mark is an employee).

SELECT *

from Employee

where Salary > (select Salary from Employee where Ename = 'Mark');

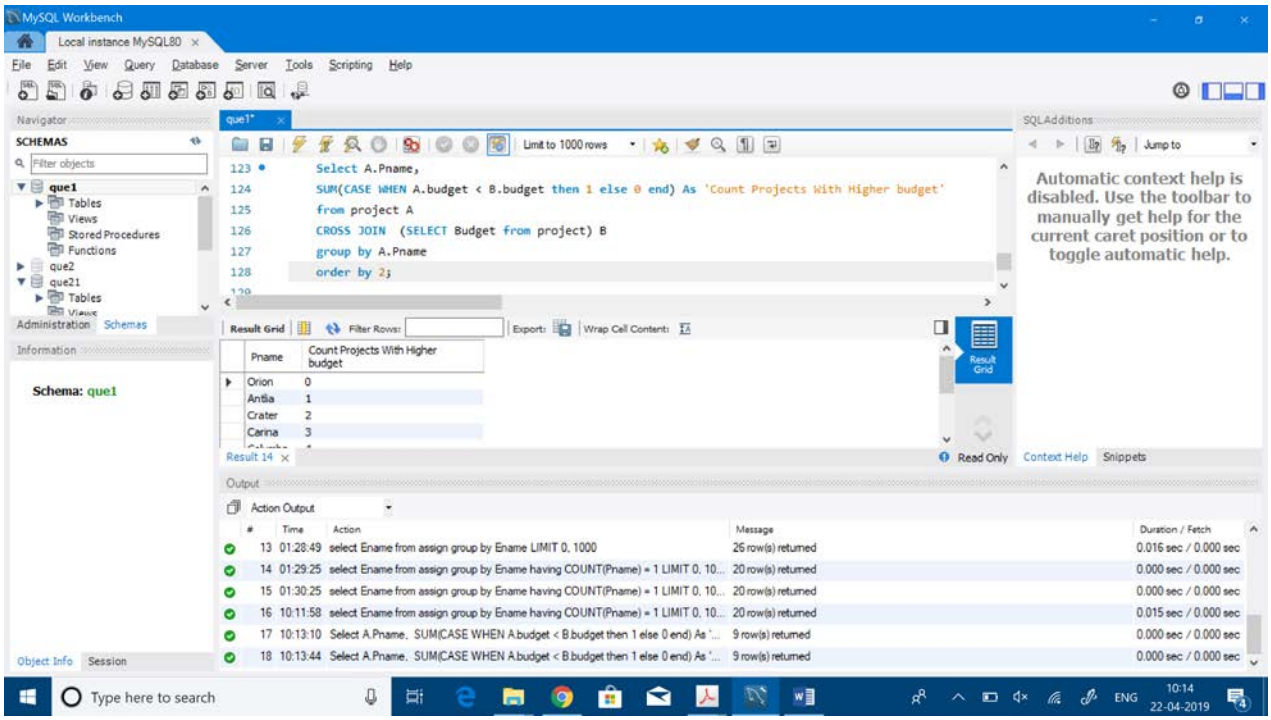
Screenshot:



S3: For every project, compute the number of projects whose budget is higher.

```
Select A.Pname,
SUM(CASE WHEN A.budget < B.budget then 1 else 0 end) As 'Count Projects With Higher budget'
from project A
CROSS JOIN  (SELECT Budget from project) B
group by A.Pname
order by 2;
```

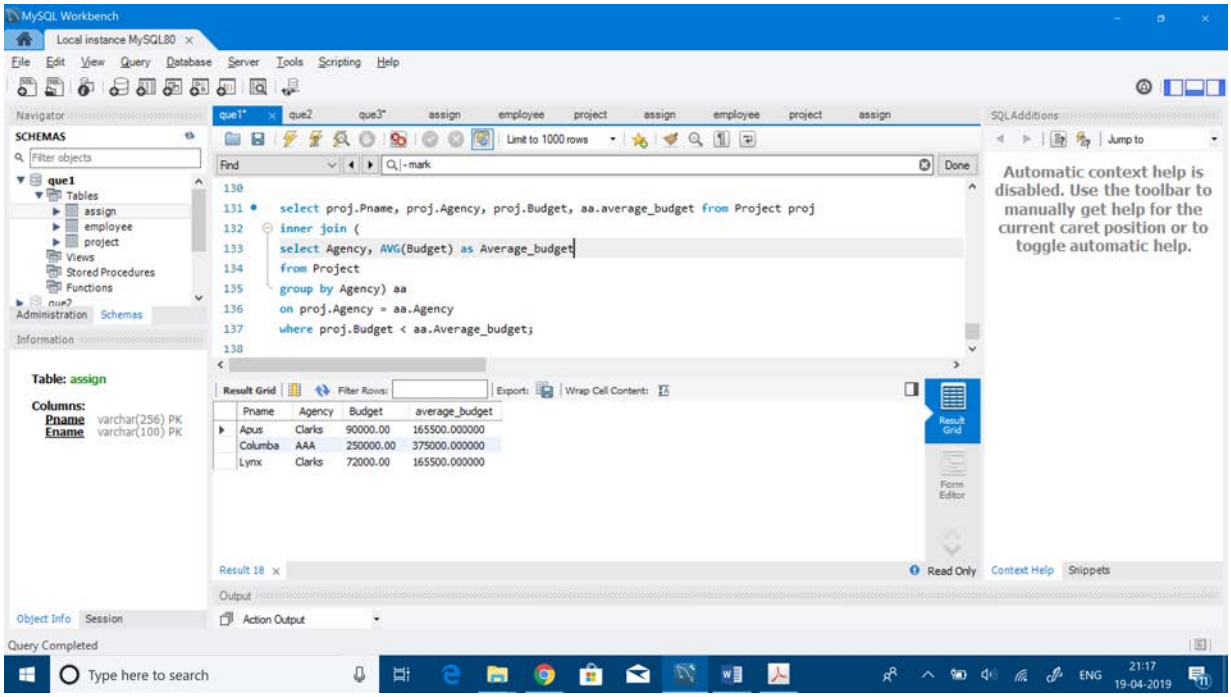
Screenshot:



S4: Find the projects with the budget lower than the average project budget at the same agency.

```
select proj.Pname, proj.Agency, proj.Budget, aa.average_budget from Project proj
inner join (
select Agency, AVG(Budget) as Average_budget
from Project
group by Agency) aa
on proj.Agency = aa.Agency
where proj.Budget < aa.Average_budget;
```

Screenshot:



Which of the above queries has an equivalent relational algebra formulation? Explain your answer. For the queries that have an equivalent relational algebra formulation provide this formulation.

Answer:

Only the query S2: Find the employees whose salary is greater than Mark's salary, has an equivalent relational algebra formulation. This is because S2 is the only query among all the four queries that do not use 'group by' or any aggregate function. Queries using aggregate functions MIN and MAX could only be written in Relational Algebra. Queries using other aggregate functions such as COUNT, SUM etc. could not have an equivalent relational algebra formulation. To represent 'group by', there is a symbol in the relational algebra, but it has not been included in our syllabus. Hence, I could only represent query S2 in relational algebra.

The equivalent relational algebra formulation for query S2 is:

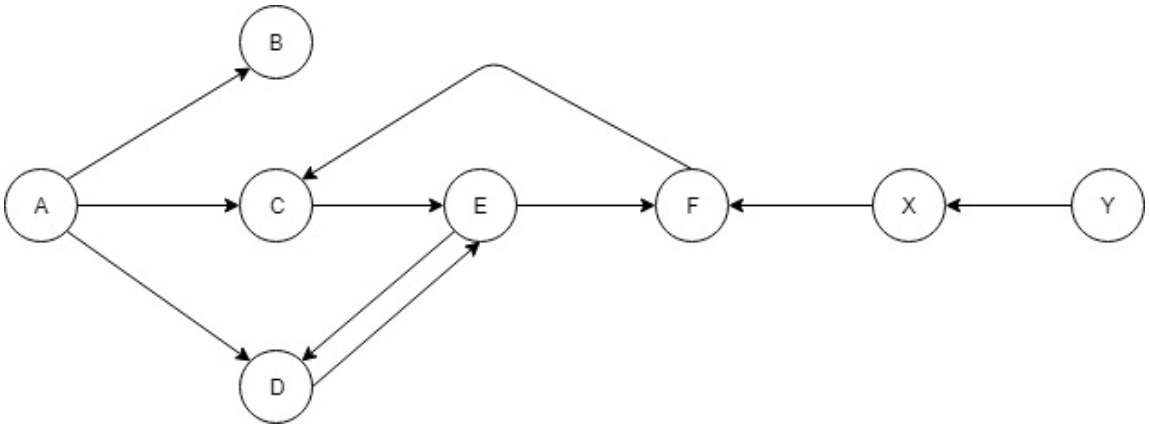
$R1 := \pi_{Salary} (\sigma_{Ename = \text{'Mark'}} (Employees))$

The final relational algebra query for S2 is:

$\pi_{Ename, Salary} (\sigma_{Salary > R1.Salary} (Employees))$

Problem.2

The directed graph chosen for the given problem is shown below:



- **Define an appropriate relational schema?**

  For representing this graph in a relational schema we define a relation called graph with two attributes types StartNode and EndNode. Each StartNode and EndNode describes a directed link going from StartNode to EndNode in the graph.

  Relational Schema:
  Graph (StartNode, EndNode)

  The create table statements, the insert statements and the screenshots for the schema are shown below:

  Create table statement for graph table having 2 columns StartNode and EndNode:

```
create table graph
(
StartNode char(1),
EndNode char(1),
constraint graphPK primary key (StartNode, EndNode)
);
```

  Statements for inserting values in the graph table:

```
insert into graph values ('A','B');
insert into graph values ('A','C');
insert into graph values ('A','D');
insert into graph values ('C','E');
```

insert into graph values ('D','E');

insert into graph values ('E','D');

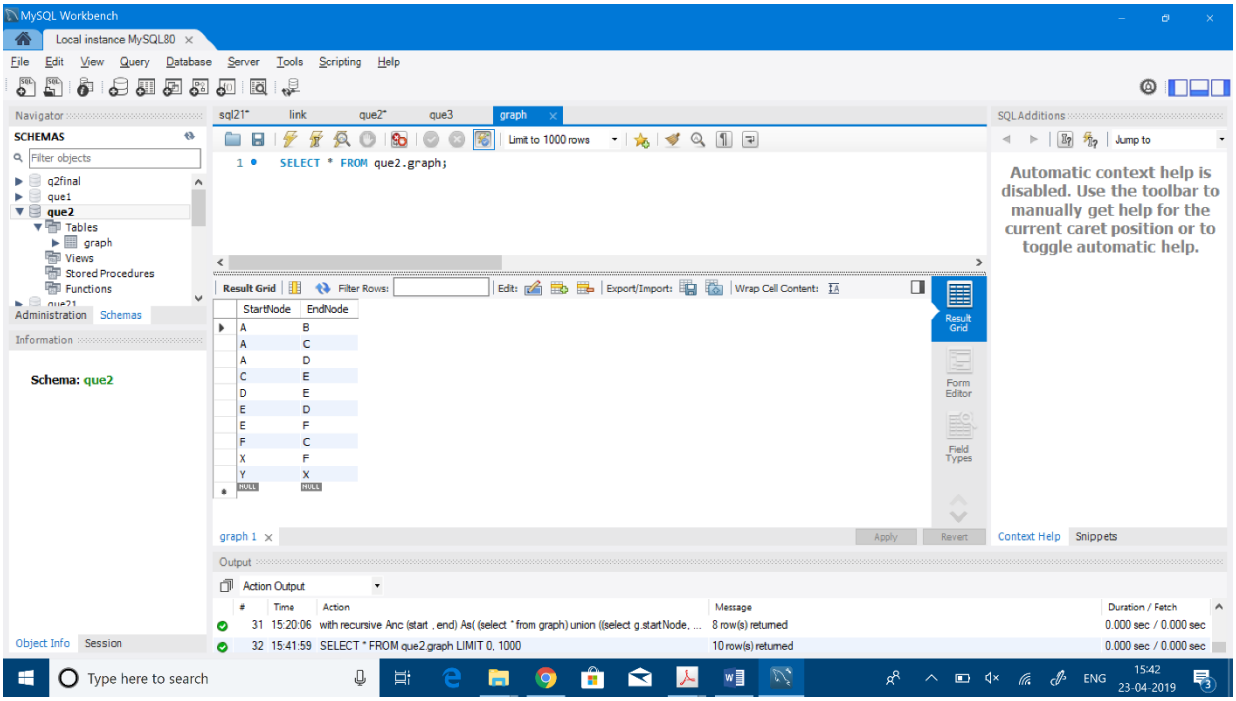
insert into graph values ('E','F');

insert into graph values ('F','C');

insert into graph values ('Y','X');

insert into graph values ('X','F');

Screenshot:



- **Write the following queries:**
  - **For every node, compute the number of the nodes directly reachable from the node.**

    select StartNode, COUNT(DISTINCT EndNode) AS Node_Cnt from Graph

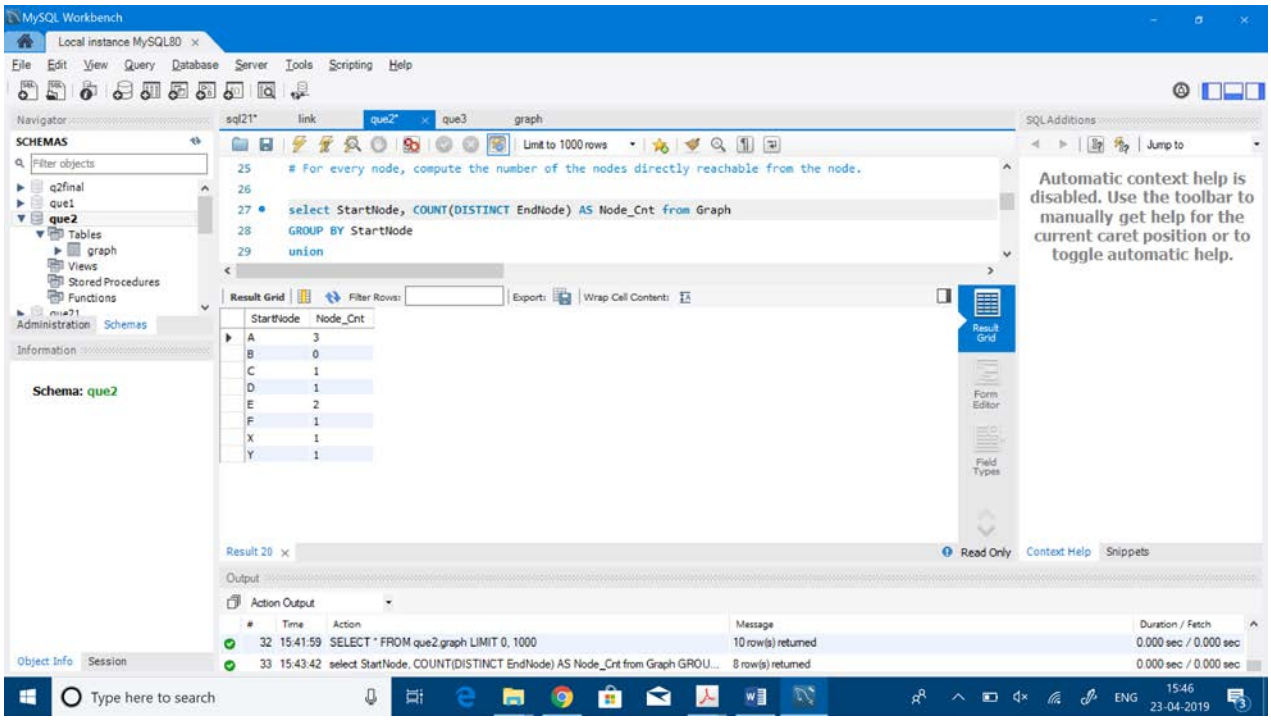
    GROUP BY StartNode

    union

    select EndNode, 0 from Graph where

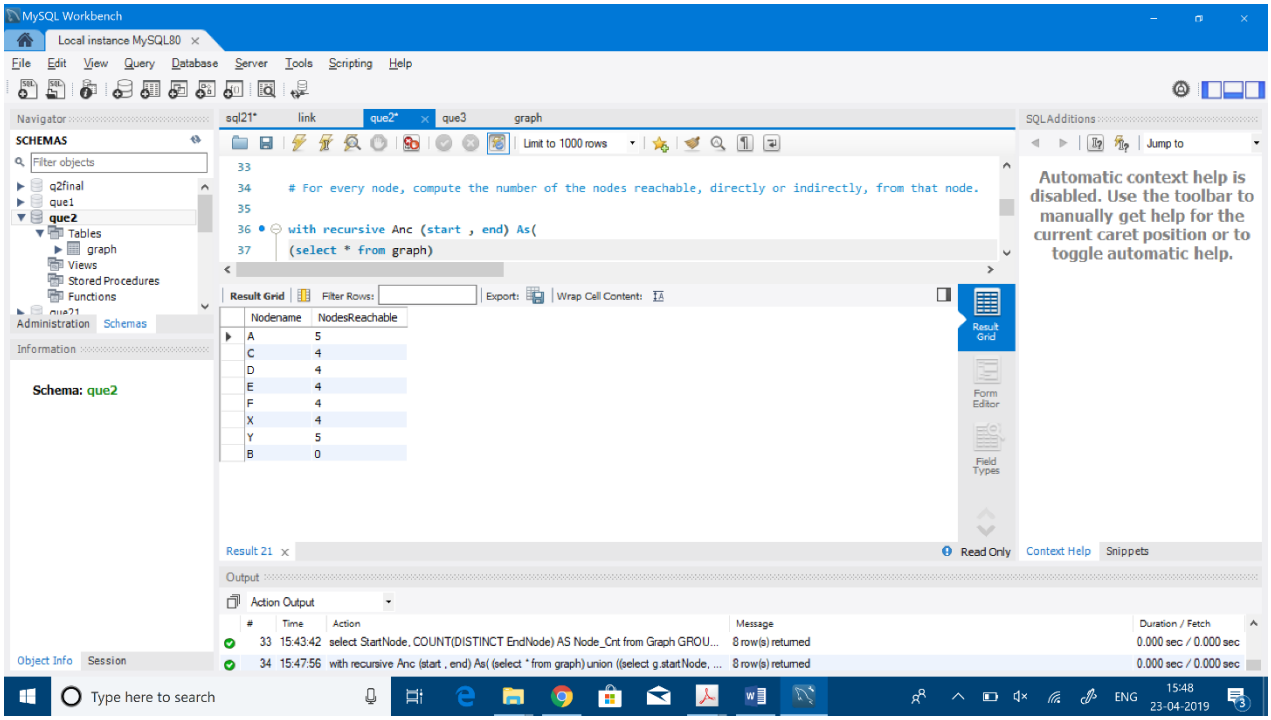
    EndNode not in (select StartNode from Graph)

    order by StartNode;



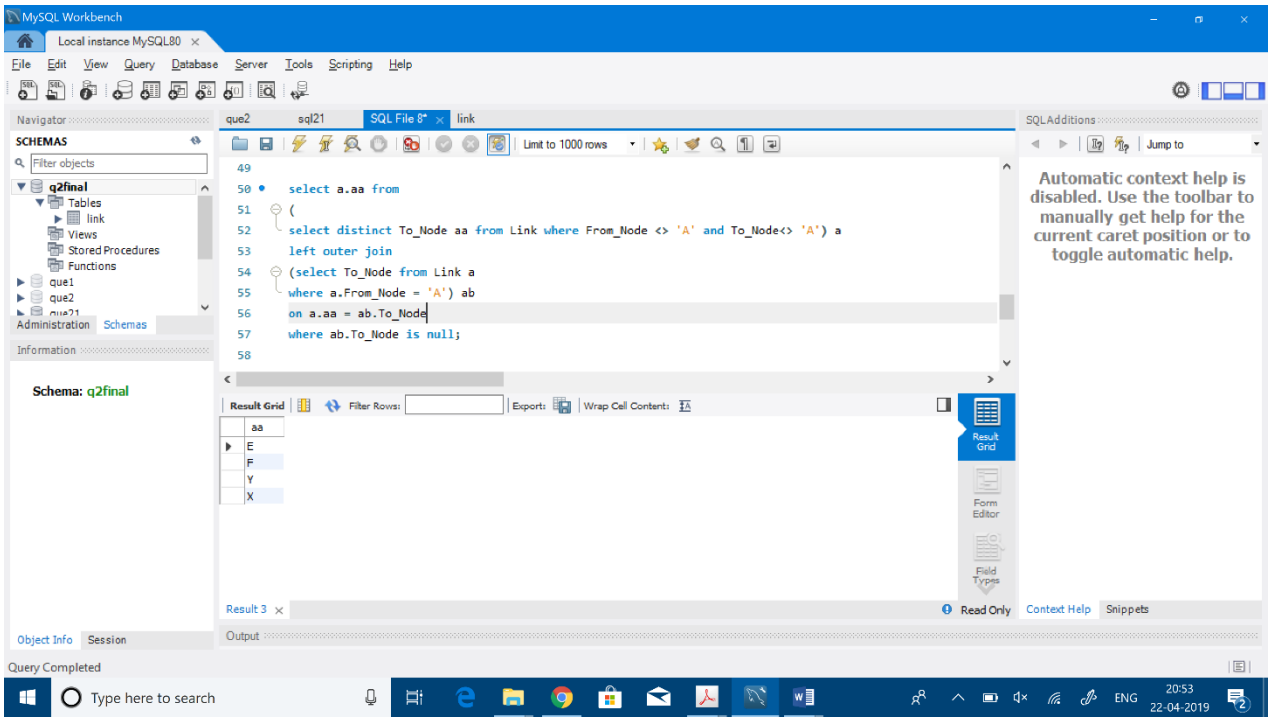
- **For every node, compute the number of the nodes reachable, directly or indirectly, from that node.**

```
with recursive Anc (start , end) As(
(select * from graph)
union
((select g.startNode, A.end
from graph g, Anc A
where g.endNode = A.start And A.start <> A.end)))
select Anc.start as Nodename, Count(anc.end)as NodesReachable from Anc group by
anc.start
union
select EndNode, 0 from Graph where
EndNode not in ( select StartNode from Graph);
```



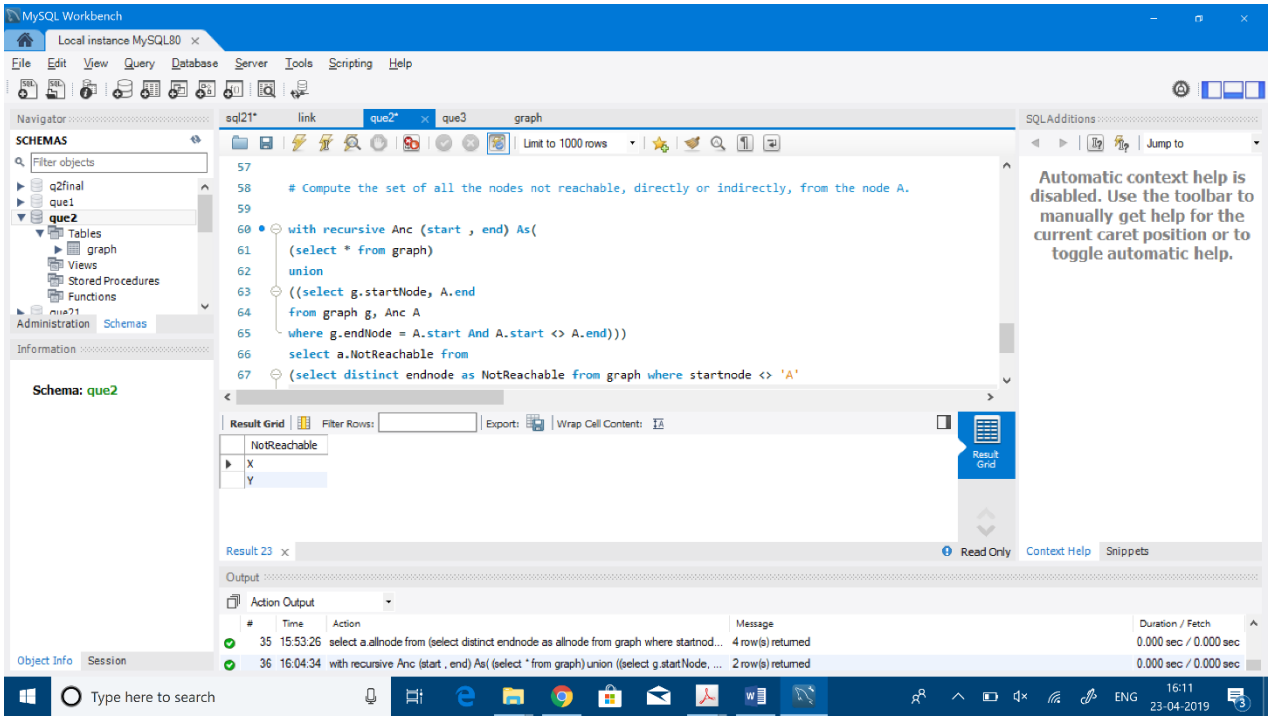
- **Compute the set of all the nodes not directly reachable from node A.**

```
select a.allnode from
(select distinct endnode as allnode from graph where startnode <> 'A'
union
select distinct startnode from graph where startnode <> 'A'
order by allnode) a
where allnode not in
(select distinct endnode as allnode from graph
where startnode ='A');
```



- **Compute the set of all the nodes not reachable, directly or indirectly, from the node A.**

```
with recursive Anc (start , end) As(
(select * from graph)
union
((select g.startNode, A.end
from graph g, Anc A
where g.endNode = A.start And A.start <> A.end)))
select a.NotReachable from
(select distinct endnode as NotReachable from graph where startnode <> 'A'
union
select distinct startnode from graph where startnode <> 'A'
order by NotReachable) a
where NotReachable not in
(select Anc.end as NotReachable from Anc where Anc.start = 'A');
```



- **Construct a small database instance, test your queries over it and report the results.**
  The create table queries and the insert queries for the database schema representing the graph shown above were written above. The same could also be found in the .sql file.

Problem.3

We are given a relation with N columns of the same type. We need to write a SQL query that returns tuples having minimum number of different values.

The relational schema for the problem is:

R (value1, value2, value3,…..,valuen)

Create table statement for relation R having N number of columns. This is a conceptual query, as it has N columns. It does not actually runs in MySQL:

```
create table R
(value1 char(1),
value2 char(1),
value3 char(1),
.....valuen char(1)
);
```

Create table statement for relation having N=3 number of columns (the same queries could be found in .sql file. There is also a second table R2 in the .sql file. In the relation R2, the minimum number of different values is 1):

```
create table R
(value1 char(1),
value2 char(1),
value3 char(1)
);
```

Statements for inserting values in R:

```
insert into R values ('a','b','c');
insert into R values ('d','d','e');
insert into R values ('f','g','g');
insert into R values ('h','i','j');
insert into R values ('k','l','m');
insert into R values ('n','o','n');
insert into R values ('p','o','n');
insert into R values ('n','o','n');
insert into R values ('o','n','n');
```

For a schema with N columns a SQL query that would return all the tuples having minimum number of different values is shown below:
(Note that this query is a conceptual query and cannot actually tested)

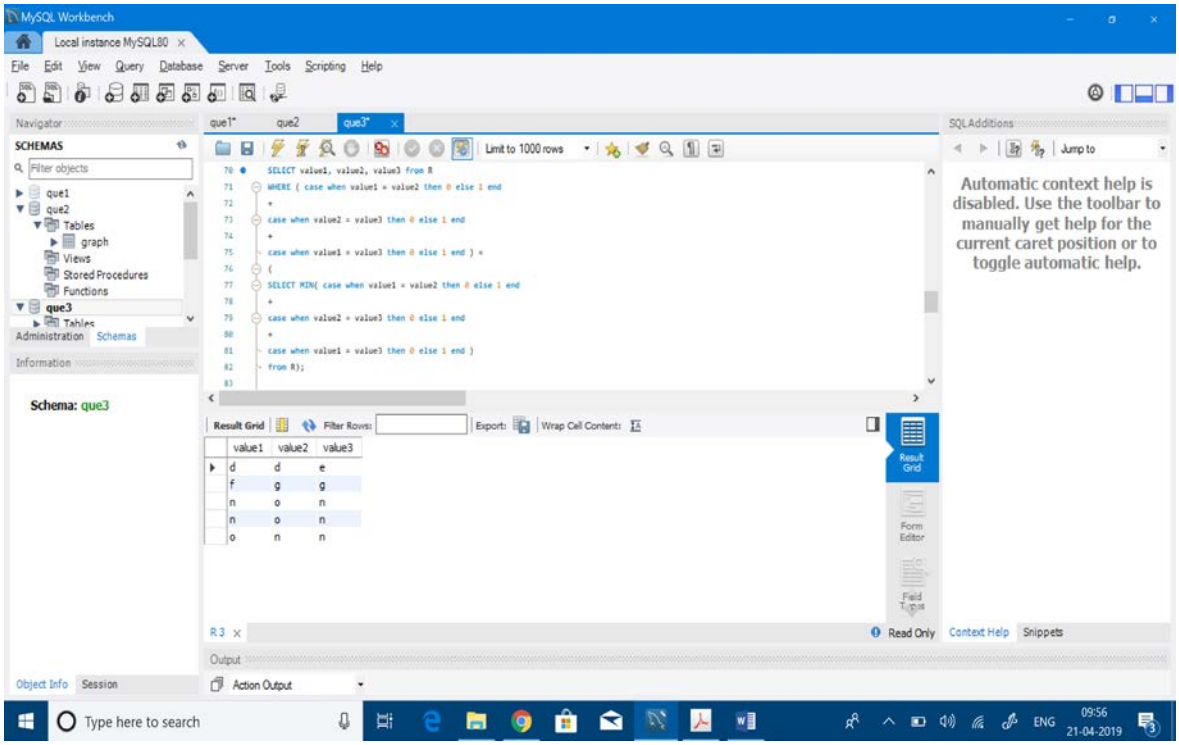
```
SELECT value1, value2, value3,....,valuen from R
WHERE (case when value1 = value2 then 0 else 1 end
+
case when value2 = value3 then 0 else 1 end
+
case when value1 = value3 then 0 else 1 end
+
.....
+
case when value1 = valuen then 0 else 1 end
+
```

```
case when value2 = valuen then 0 else 1 end
+
case when value3 = valuen then 0 else 1 end) =
(
SELECT MIN( case when value1 = value2 then 0 else 1 end
+
case when value2 = value3 then 0 else 1 end
+
case when value1 = value3 then 0 else 1 end
+
.....
+
case when value1 = valuen then 0 else 1 end
+
case when value2 = valuen then 0 else 1 end
+
case when value3 = valuen then 0 else 1 end)
from R);
```

For number of columns N=3, the actual SQL query is:

```
SELECT value1, value2, value3 from R
WHERE (case when value1 = value2 then 0 else 1 end
+
case when value2 = value3 then 0 else 1 end
+
case when value1 = value3 then 0 else 1 end) =
(
SELECT MIN(case when value1 = value2 then 0 else 1 end
+
case when value2 = value3 then 0 else 1 end
+
case when value1 = value3 then 0 else 1 end)
from R);
```

Output Screenshot:



Note that the query shown above use aggregation in an essential way. Also, the query is size polynomial in N. A small proof to see it the query is size polynomial in N and not exponential is:

- For number of columns N = 3, there are in total 6 case statements in the query.
- For number of columns N = 4, there would be in total 12 case statements in the query.

Hence, this shows that the size of query does not increase exponentially. So, its size is polynomial in N.