

Assignment 3 - Report

Department of Industrial and Enterprise Systems Engineering

Zhenye Na (zna2)

April 2, 2018

1 SINGLE-RELATION QUERIES (30 PTS)

1. Consider the following relation:

Graph(n1 , n2)

A tuple (n1, n2) in Graph stores a directed edge from a node n1 to a node n2 in the corresponding graph. Your goal is to, for *every* node in the graph, count the number of outgoing edges of that node. Note that for nodes without any outgoing edges, their edge count would be zero; you need to output this as well. You can assume that:

- a) there are no duplicates or null values in the table; and
- b) every node in the graph is involved in at least one edge.

Solution:

```
SELECT n1 , COUNT(n1)
FROM Graph
GROUP BY Graph.n1 ;
UNION
SELECT n2, 0
FROM Graph as g1
WHERE g1.n2 NOT IN (SELECT g2.n1
                    FROM Graph as g2
                    );
```

2. Consider the following relation:
Trained(student, master, year)

A tuple (S, M, Y) in Trained specifies that a SQL Master M trained student S who graduated in year Y. Your goal is to find *the count of* SQL Masters who trained a student who graduated in the same year that 'Alice' or 'Bob' graduated.

Solution:

```
SELECT COUNT(DISTINCT t1.master) as CountofMaster
FROM Trained as t1
WHERE t1.year IN ( SELECT t2.year
                   FROM Trained as t2
                   WHERE t2.student = "Alice"
                      OR t2.student = "Bob"
                 );
```

3. Consider the following relation:

DBMS(operator, system, performance)

A tuple (O, S, P) in DBMS specifies an operator O in system S and has the performance value P. Your goal is to find those systems whose operators achieves a higher performance value on average than the average performance value in a system named 'PostgreSQL'.

Solution:

```
SELECT d3.system
FROM ( SELECT d1.system as system, AVG(d1.performance) as performance
      FROM DBMS as d1
      GROUP BY d1.system
    ) as d3
WHERE d3.performance > ( SELECT AVG(d2.performance)
                       FROM DBMS as d2
                       WHERE d2.system = "PostgreSQL"
                     );
```

2 MULTI-RELATION QUERIES (20 PTS)

Consider the following relations representing student information at UIUC:

Mentorship(mentee_sid, mentor_sid)

Study(sid, credits)

Enrollment(did, sid)

Student(sid, street, city)

- A tuple (M1, M2) in Mentorship specifies that M2 is a mentor of another student M1.
- A tuple (S, C) in Study specifies that the student S has taken C credits.
- A tuple in Enrollment (D, S) specifies that student S is enrolled in department D.
- A (ST, S, C) in Student specifies that student ST lives on street S in city C.

1. Find all students who live in the same city and on the same street as their mentor.

Solution:

```
SELECT ml.mentee_sid
FROM Mentorship as ml, Student as s1, Student as s2
WHERE ml.mentee_sid = s1.sid
      AND ml.mentor_sid = s2.sid
      AND s1.street = s2.street
      AND s1.city = s2.city;
```

2. Find all students(i.e., distinct sid) who have taken more credits than the average credits of all of the students of their department.

Solution:

```
SELECT s1.sid
FROM Study as s1, Enrollment as e1
WHERE e1.sid = s1.sid
      AND s1.credits > ( SELECT AVG(s2.credits)
                        FROM Study as s2, Enrollment as e2
                        WHERE s2.sid = e2.sid
                        AND e1.did = e2.did
                        );
```

3 DATABASE MANIPULATION AND VIEWS (25 PTS)

1. In the Study relation, insert a new student, whose id is 66666 and has 0 credits.

Solution:

```
INSERT INTO Study
VALUES('66666', '0');
```

2. In the Study relation, delete students who have graduated (i.e., the ones who have more than 200 credits).

Solution:

```
DELETE FROM Study
WHERE Study.credits > 200;
```

3. In the Study relation, add 2 credits for students who are mentors.

Solution:

```
UPDATE Study
SET Study.credits = Study.credits + 2
WHERE Study.sid IN ( SELECT ml.mentor_sid
                    FROM Mentorship as ml
                    );
```

4. Incoming students are those who have been accepted (i.e., exist in the **Student** relation) but have not registered in any department (i.e., do not exist in the **Enrollment** relation). Create a View that contains **sid** of all incoming students.

Solution:

```
CREATE VIEW Incoming AS
  SELECT s1.sid
  FROM Student as s1
  WHERE s1.sid NOT IN ( SELECT e1.sid
                        FROM Enrollment as e1
                      );
```

4 CONSTRAINTS AND TRIGGERS (25 PTS)

1. Consider the following relation:

Payment(salary, bonus)

Write a schema-level assertion using the "CREATE ASSERTION" statement to ensure that no bonus is larger than the maximum salary in the **Payment** relation.

Solution:

```
CREATE ASSERTION Bonus
CHECK(
NOT EXISTS( SELECT Payment.bonus
            FROM Payment
            WHERE Payment.bonus > MAX(Payment.salary)
          ));
```

2. Consider the following relation:

Study(sid, major, GPA)

Write a trigger T1 that increases the GPA by 10% for those students who transform their major from any Non-CS major to 'CS'.

Solution:

```
CREATE TRIGGER T1
  AFTER UPDATE OF major ON Study
  REFERENCING
    OLD ROW AS old
    NEW ROW AS new
  FOR EACH ROW
  WHEN(
    old.major <> 'CS'
    AND new.major = 'CS'
  )
  UPDATE Study
  SET new.GPA = old.GPA * 1.1;
```