**Assignment 7 | Neha Chede**

1.

Input Tables:

create table PC (p varchar(30), c varchar(30));

insert into PC (p, c) values

('Jay', 'Claire'),

('Jay', 'Mitchell'),

('Mitchell', 'Lily'),

('Claire', 'Haley'),

('Claire', 'Alex'),

('Claire', 'Luke');

create table Male (p varchar(30));

insert into Male (p) values

('Jay'),

('Mitchell'),

('Luke');

create table Female (p varchar(30));

insert into Female (p) values

('Claire'),

('Lily'),

('Haley'),

('Alex');

Query:

with recursive ancestors(ance, id) as (

select p as ance, c as id

from PC

union

select a.ance, PC.c

from ancestors a

join PC

on a.id = PC.p

),

descendants(desce, id) as (

select c as desce, p as id

from PC

union

select a.desce, PC.p

from descendants a

join PC

on a.id = PC.c

)

select a.ance as x, m.p as y, d.desce as z

from ancestors a, descendants d, Male m, Female f

where a.id = m.p and m.p = d.id and d.desce = f.p;

Output:

|  |  |  |
| --- | --- | --- |
| x | y | z |
| Jay | Mitchell | Lily |

2.

A: Number of people known by Person, p1

B: Number of people known by Person, p2

Query:

create or replace function k\_cnt(pid int)

returns int as

$$

select count(k.pid2)

from Knows k

where k.pid1 = k\_cnt.pid

group by k.pid1;

$$ language sql;

select distinct k1.pid1 as p1, k2.pid1 as p2

from Knows k1, Knows k2

where k1.pid1 <> k2.pid1 and not exists(

select \*

from k\_cnt(k1.pid1)

except

select \*

from k\_cnt(k2.pid1)

);

3.

select distinct p1.pid

from personHasSkills p1

where not exists(

select 1

from personHasSkills p2

where cardinality(p2.skills) > cardinality(p1.skills)

and p1.pid<>p2.pid

);

Output:

A screenshot of a computer

Description automatically generated

4.

Considering constants, a = 5 and c = 14,

create table r(a integer, b integer);

create table s(b integer, c integer);

create table v\_cnt (cnt integer);

insert into v\_cnt values (0);

create or replace function upd\_cnt()

returns trigger as

$$

begin

if tg\_op = 'INSERT' then

if (new.a!=5 and new.b in (select b from s where c!=14)) then

update v\_cnt

set cnt = cnt + 1;

end if;

elsif tg\_op = 'DELETE' then

if (old.a!=5 and old.b in (select b from s where c!=14)) then

update v\_cnt

set cnt = cnt - 1;

end if;

end if;

return new;

end;

$$ language plpgsql;

create or replace trigger insert\_r after insert on r

for each row

execute function upd\_cnt();

create or replace trigger delete\_r after delete on r

for each row

execute function upd\_cnt();

insert into r values (1,2), (3,4), (5,6), (7,8), (9,10);

insert into s values (2,11), (4,12), (6,13), (8,14), (10,15);

create or replace view ques as

SELECT r.a, s.c

FROM R r, S s

WHERE r.a != 5 AND r.b = s.b AND s.c != 14;

select \* from v\_cnt;

Output:

"cnt"

3

insert into r values (2,20);

select \* from v\_cnt;

Output:

"cnt"

4

5. With buffers, and block size, , if we use the block nested-loop algorithm to implement natural join operations, to evaluate the relational algebra expression , the time complexity is:

where is the number of blocks to store .

Given the assumption, , the overall time complexity depends on the number of block transfers required for each join operation.

6. (a) Given, r = 300,000 records, B = 4,096 bytes, length of record = 100 bytes

Number of Block Accesses =

6. (b) Given, V = 9 bytes, P = 6 bytes

Records per block =

Entries per block, b = indexes

Number of blocks in the index file =

With binary search as a primary index is constructed, number of block access required =

7. (a) R1(x); R2(y); R1(z); R2(x); R1(y)

The schedule is conflict serializable as there are all read operations involved in the transaction and there are no cycles in the precedence graph. A conflict equivalent serial schedule would be: R1(x);R1(z);R2(y);R2(x);R1(y).

7. (b) R1(x); W2(y); R1(z); R3(z); W2(x); R1(y)

There is a cycle in the precedence graph for the given schedule [Transactions (1, 2, 3, 1)], thus, this schedule is not conflict serializable.

8. (a) The relation:

Patients (Patient\_ID, Name, DOB)

Doctors (Doctor\_ID, name, specialty)

Nurses (Nurse\_ID, name, department)

Relationship tables: pat\_doc (Patient\_ID, Doctor\_ID) pat\_nur (Patient\_ID, Nurse\_ID)

A screenshot of a computer screen

Description automatically generated

**1**

*Fig. Entity Relationship Diagram for the given schema using pgAdmin*

8. (b)

create table Patients(

Patient\_ID integer primary key,

name text,

DOB date

);

create table Doctors(

Doctor\_ID integer primary key,

name text,

specialty text

);

create table Nurses(

Nurse\_ID integer primary key,

name text,

department text

);

create table pat\_doc(

Patient\_ID integer references Patients(Patient\_ID),

Doctor\_ID integer references Doctors(Doctor\_ID),

primary key(Patient\_ID, Doctor\_ID)

);

create table pat\_nur(

Patient\_ID integer references Patients(Patient\_ID),

Nurse\_ID integer references Nurses(Nurse\_ID),

primary key(Patient\_ID, Nurse\_ID)

);