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

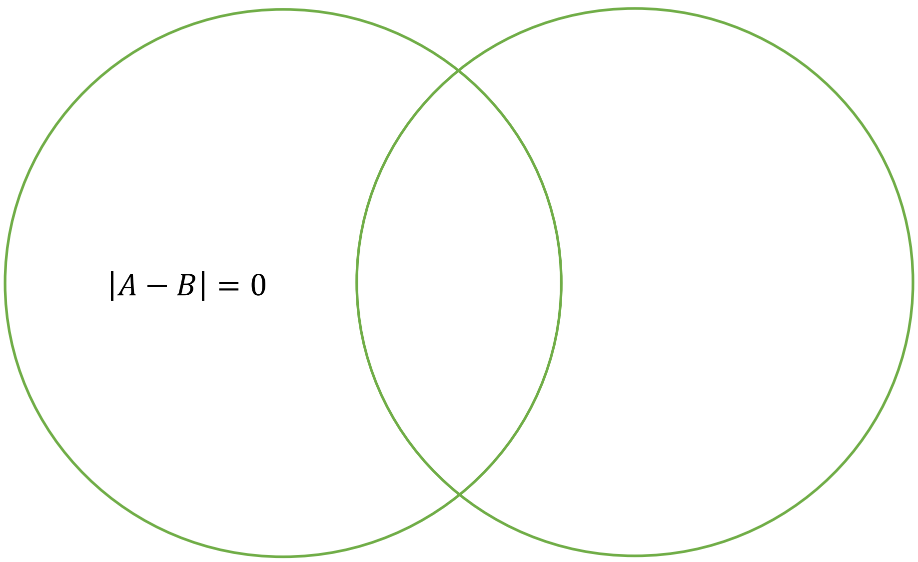
2.

$|A - B| = 0$

$\neg\exists\, x\ (x \in A - B)$

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:

	pid integer
1	1011

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, $B(R), B(S), B(T)$ and block size, M, if we use the block nested-loop algorithm to implement natural join operations, to evaluate the relational algebra expression $(R \bowtie S) \bowtie T$, the time complexity is:

$$B(R \bowtie S) = B(R) + \frac{B(R) \times B(S)}{M}$$

where $B(R \bowtie S)$ is the number of blocks to store $(R \bowtie S)$.

$$B\big((R \bowtie S) \bowtie T\big) = B(R \bowtie S) + \frac{B(R \bowtie S) \times B(T)}{M}$$

Given the assumption, $B(R \bowtie S) \leq M^2$, 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 = $\lceil \log_2(N+1) \rceil = \left\lceil \log_2(\frac{300000 \times 100}{4096} + 1) \right\rceil = \lceil \log_2(7325) \rceil = \lceil 12.84 \rceil = 13$

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

Records per block = $\left\lfloor \frac{4096}{100} \right\rfloor = 40$

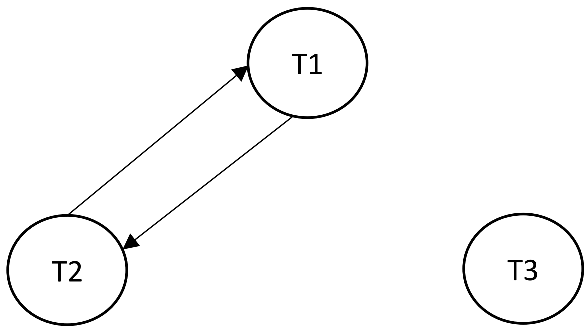
Entries per block, b = $\left\lfloor \frac{4096}{9+6} \right\rfloor = \lfloor 273.06 \rfloor = 273$ indexes

Number of blocks in the index file = $\left\lceil \frac{300000/40}{273} \right\rceil = \lceil 27.47 \rceil = 28$

With binary search as a primary index is constructed, number of block access required = $\lceil \log_2 28 \rceil = \lceil 4.80 \rceil = 5$

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 (<u>Patient_ID</u>, Name, DOB)
Doctors (<u>Doctor_ID</u>, name, specialty)
Nurses (<u>Nurse_ID</u>, name, department)

Relationship tables: pat_doc (Patient_ID, Doctor_ID) pat_nur (Patient_ID, Nurse_ID)

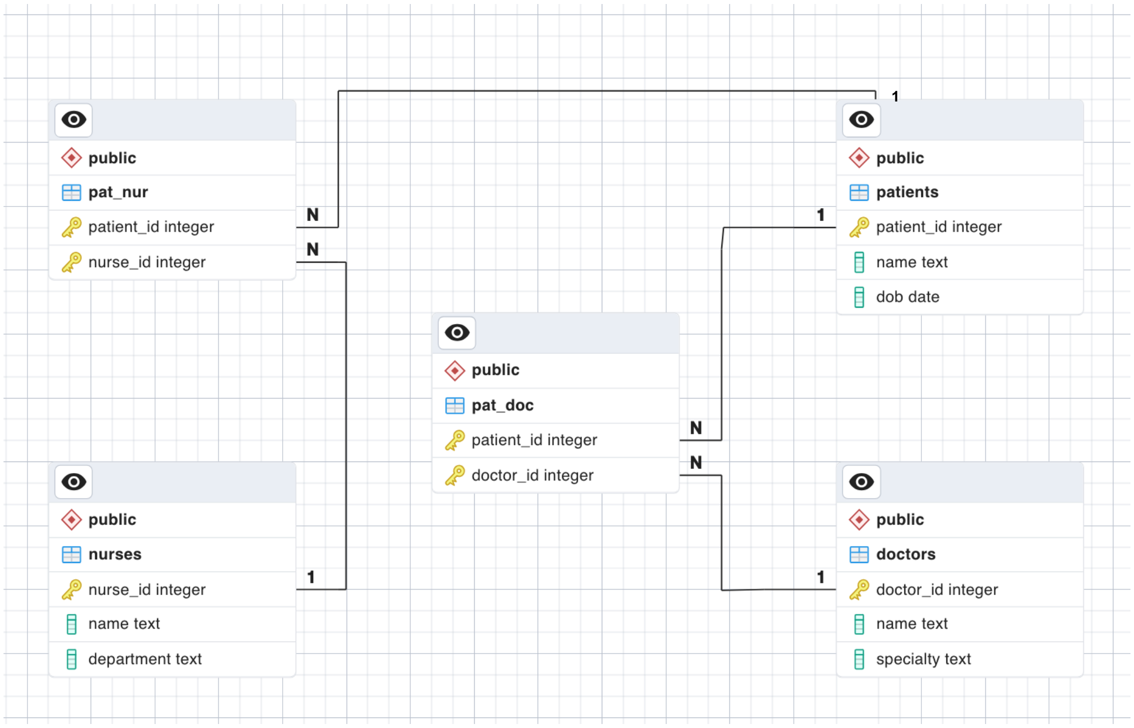


Fig. Entity Relationship Diagram for the given schema using pgAdmin

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)
);
```