Assignment A4

(d) Multi table join (minimum of 4 tables) – Review the join order of the tables.

(e) Demonstrate performance improvement by comparing the execution plans.

Name: Arvind Krishna SRN: PES1UG19CS090	Sem: 6 Sec: B

Creating a table called director:

```
dbt=# DROP TABLE IF EXISTS director;
CREATE TABLE director(
    id int PRIMARY KEY,
    name varchar(10)
);
INSERT INTO director (id, name)
SELECT (random()*100000)::integer,
        substr(md5(random()::text), 1, 10)
FROM generate_series(1, 10000)
ON CONFLICT (id) DO NOTHING;
select count(*) from director;
NOTICE:  table "director" does not exist, skipping
DROP TABLE
CREATE TABLE
INSERT 0 9512
 count
-------
  9512
(1 row)
```

Creating a table called movie:

```
dbt=# DROP TABLE IF EXISTS movie;
CREATE TABLE movie(
    id int PRIMARY KEY,
    movie_name varchar(10)
);
INSERT INTO movie (id, movie_name)
SELECT (random()*100000)::integer,
        substr(md5(random()::text), 1, 10)
FROM generate_series(1, 20000)
ON CONFLICT (id) DO NOTHING;
select count(*) from movie;
NOTICE:  table "movie" does not exist, skipping
DROP TABLE
CREATE TABLE
INSERT 0 18081
 count
-------
 18081
(1 row)
```

Creating a table called actor:

```
dbt=# DROP TABLE IF EXISTS actor;
CREATE TABLE actor(
    id int PRIMARY KEY,
    name varchar(10)
);
INSERT INTO actor (id, name)
SELECT (random()*100000)::integer,
        substr(md5(random()::text), 1, 10)
FROM generate_series(1, 30000)
ON CONFLICT (id) DO NOTHING;
select count(*) from actor;
NOTICE:  table "actor" does not exist, skipping
DROP TABLE
CREATE TABLE
INSERT 0 26004
 count
-------
 26004
(1 row)
```

Creating a table called tickets:

```
dbt=# DROP TABLE IF EXISTS tickets;
CREATE TABLE tickets(
    id int PRIMARY KEY,
    cust_name varchar(10)
);
INSERT INTO tickets (id, cust_name)
SELECT (random()*100000)::integer,
        substr(md5(random()::text), 1, 10)
FROM generate_series(1, 40000)
ON CONFLICT (id) DO NOTHING;
select count(*) from tickets;
DROP TABLE
CREATE TABLE
INSERT 0 32956
 count
-------
 32956
(1 row)
```

Table Summary:

```
dbt=# SELECT schemaname,relname,n_live_tup
  FROM pg_stat_user_tables
ORDER BY n_live_tup;

 schemaname | relname  | n_live_tup
------------+----------+------------
 public     | director |       9512
 public     | movie    |      18081
 public     | actor    |      26004
 public     | tickets  |      32956
(4 rows)
```

Time taken for execution in the descending order of table size

```
dbt=# EXPLAIN ANALYZE SELECT COUNT(*) FROM tickets natural join actor natural join movie natural join director;
                                                               QUERY PLAN
-------------------------------------------------------------------------------------------------------------------------------------------
 Aggregate  (cost=828.01..828.02 rows=1 width=8) (actual time=24.713..24.720 rows=1 loops=1)
   ->  Nested Loop  (cost=290.38..828.00 rows=1 width=0) (actual time=24.707..24.712 rows=0 loops=1)
         ->  Nested Loop  (cost=290.09..827.69 rows=1 width=12) (actual time=24.706..24.710 rows=0 loops=1)
               ->  Hash Join  (cost=289.80..827.36 rows=1 width=8) (actual time=24.705..24.709 rows=0 loops=1)
                     Hash Cond: ((actor.id = director.id) AND ((actor.name)::text = (director.name)::text))
                     ->  Seq Scan on actor  (cost=0.00..401.04 rows=26004 width=15) (actual time=0.013..5.642 rows=26004 loops=1)
                     ->  Hash  (cost=147.12..147.12 rows=9512 width=15) (actual time=6.745..6.747 rows=9512 loops=1)
                           Buckets: 16384  Batches: 1  Memory Usage: 565kB
                           ->  Seq Scan on director  (cost=0.00..147.12 rows=9512 width=15) (actual time=0.007..2.017 rows=9512 loops=1)
               ->  Index Only Scan using tickets_pkey on tickets  (cost=0.29..0.33 rows=1 width=4) (never executed)
                     Index Cond: (id = actor.id)
                     Heap Fetches: 0
         ->  Index Only Scan using movie_pkey on movie  (cost=0.29..0.31 rows=1 width=4) (never executed)
               Index Cond: (id = tickets.id)
               Heap Fetches: 0
 Planning Time: 3.769 ms
 Execution Time: 24.805 ms
(17 rows)
```

Time taken for execution in the ascending order of table size

```
dbt=# EXPLAIN ANALYZE SELECT COUNT(*) FROM director natural join movie natural join actor natural join tickets;
                                                              QUERY PLAN
-------------------------------------------------------------------------------------------------------------------------------------------
 Aggregate  (cost=828.03..828.04 rows=1 width=8) (actual time=6.131..6.133 rows=1 loops=1)
   ->  Nested Loop  (cost=290.38..828.03 rows=1 width=0) (actual time=6.128..6.130 rows=0 loops=1)
         Join Filter: (director.id = movie.id)
         ->  Nested Loop  (cost=290.09..827.70 rows=1 width=12) (actual time=6.128..6.130 rows=0 loops=1)
               Join Filter: (director.id = tickets.id)
               ->  Hash Join  (cost=289.80..827.36 rows=1 width=8) (actual time=6.127..6.129 rows=0 loops=1)
                     Hash Cond: ((actor.id = director.id) AND ((actor.name)::text = (director.name)::text))
                     ->  Seq Scan on actor  (cost=0.00..401.04 rows=26004 width=15) (actual time=0.004..1.382 rows=26004 loops=1)
                     ->  Hash  (cost=147.12..147.12 rows=9512 width=15) (actual time=1.646..1.647 rows=9512 loops=1)
                           Buckets: 16384  Batches: 1  Memory Usage: 565kB
                           ->  Seq Scan on director  (cost=0.00..147.12 rows=9512 width=15) (actual time=0.002..0.499 rows=9512 loops=1)
               ->  Index Only Scan using tickets_pkey on tickets  (cost=0.29..0.33 rows=1 width=4) (never executed)
                     Index Cond: (id = actor.id)
                     Heap Fetches: 0
         ->  Index Only Scan using movie_pkey on movie  (cost=0.29..0.31 rows=1 width=4) (never executed)
               Index Cond: (id = tickets.id)
               Heap Fetches: 0
 Planning Time: 0.731 ms
 Execution Time: 6.162 ms
(19 rows)
```

From the above execution times it is clear that the join in which smaller tables are joined first is relatively efficient than the other way around.