# Contents

# 1 Basic Test Results

```
Extracting Archive:
Archive:  /tmp/bodek.lPpMXx/db/ex3/tzvog/presubmission/submission
  inflating: ex3.pdf
  inflating: q1.sql
  inflating: q2.sql
  inflating: q3.sql
  inflating: q4.sql
  inflating: q5.sql
  inflating: README

************************************************************
** Testing that all necessary files were submitted:
README:
    SUBMITTED
ex3.pdf:
    SUBMITTED
q1.sql:
    SUBMITTED
q2.sql:
    SUBMITTED
q3.sql:
    SUBMITTED
q4.sql:
    SUBMITTED
q5.sql:
    SUBMITTED

************************************************************
** Checking for correct README format:
Output:
CREATE TABLE
CREATE TABLE
CREATE TABLE

Inserting movies.csv
Output:
COPY 10000

Inserting actors.csv
Output:
COPY 7369

Inserting playsInSmall.csv
Output:
COPY 999

Note: The output is capped at 500 characters.
Running q1.sql
Output:
 actorid | max | min |         avg
---------+-----+-----+---------------------
     875 | 120 | 120 | 120.0000000000000000
    1908 |     |     |
    2001 |     |     |
    2154 |  52 |  52 |  52.0000000000000000
    2161 |     |     |
    2503 |     |     |
    3193 |     |     |
    5959 | 110 | 110 | 110.0000000000000000
```

```
    7215 |  53 |  53 |  53.0000000000000000
    7221 |
Running q2.sql
Output:
 movieid |             title
---------+-------------------------------
    3599 | The Adventures of Buffalo Bill
(1 row)


Running q3.sql
Output:
 actorid |     name
---------+---------------
   95809 | Sydney Booth
  697944 | Herbert Prior
 1372000 | Willis Secord
(3 rows)


Running q4.sql
Output:
 num
-----
 108
(1 row)


Running q5.sql
Output:
 actorid |    name
---------+-------------
   45780 | Frank Bacon
(1 row)
```

# 2 README

```
1 tzvog,tzory
```

**Ex3**

## Question 1

a.

```
public=> select actorId from playsin where character like 'Sheriff';
```

(no mention of distinct)

b. if there is no use of distinct, it only takes one scan of the complete scheme.

running time = 4.121

```
public=> explain analyse select actorId from playsin where character like 'Sheriff';
                                                 QUERY PLAN
-------------------------------------------------------------------------------------------------------------
 Seq Scan on playsin  (cost=0.00..615.15 rows=53 width=4) (actual time=0.688..3.965 rows=50 loops=1)
   Filter: (("character")::text ~~ 'Sheriff'::text)
   Rows Removed by Filter: 32602
 Planning Time: 0.094 ms
 Execution Time: 4.027 ms
(5 rows)
```

c.

```
public=> create index character_name on playsIn(character);
```

D.

running time = 1.414

to calculate this query using index, we need searching down the B+ tree, and then go over all the fits leaves and eventually look for the actorid number in the scheme.

```
                                                 QUERY PLAN

-------------------------------------------------------------------------------------------------------------
-----------------------
 HashAggregate  (cost=126.31..126.84 rows=53 width=4) (actual time=0.439..0.469 rows=44 loops=1)
   Group Key: actorid
   ->  Bitmap Heap Scan on playsin  (cost=4.70..126.17 rows=53 width=4) (actual time=0.130..0.392 rows=5
0 loops=1)
         Filter: (("character")::text ~~ 'Sheriff'::text)
         Heap Blocks: exact=37
         ->  Bitmap Index Scan on character_name  (cost=0.00..4.68 rows=53 width=0) (actual time=0.094..
0.094 rows=50 loops=1)
               Index Cond: (("character")::text = 'Sheriff'::text)
 Planning Time: 0.813 ms
 Execution Time: 0.601 ms
(9 rows)
```

## Question 2

A.

1. floor(1000/150) = 6 rows at each block, roof(10,000/6) = 1667 blocks

Therefor, without index, we need 1667 I/O actions.

2. optimal separation degree is $\frac{1000+8}{8+8} = 63$

3. down the B+ tree, takes $log_{32}10000 = 3$

Passing one leaf (because of DISTINCT) with duration > 100, takes $1$ action

No need to access the row. There for, total actions: $3 + 1 = 4$

B.

1. As before, without index, passing all rows takes 1667 I/O actions.

2. optimal separation degree is $\frac{1000+8}{8+8} = 63$

3. down the B+ tree, takes $log_{32}10000 = 3$

Passing all leaves with duration > 100, takes $\frac{750}{31} = 25$

No need to access the row. There for, total actions: $3 + 25 = 28$

C.

1. As before, without index, passing all rows takes 1667 I/O actions.

2. optimal separation degree is $\frac{1000+8}{8+8} = 63$

3. down the B+ tree, takes $log_{32}10000 = 3$

movieId is unique therefore one matching row $\frac{1}{31} = 1$ it is only in one leaf giving us 1

need to access the row therefore we add 1 for a total we get 3 + 1 + 1 = 5

D.

1. As before, without index, passing all rows takes 1667 I/O actions.

2. optimal indexing degree is $\frac{1000+10}{8+10} = 56$

3. down the B+ tree, takes $log_{28}10000 = 3$

we have $\frac{10000}{4} = 2500$ rows with values of drama as genre

therefore $\frac{2500}{27} = 93$ number of branches we need to look at

for a total we will get 93 + 2500 + 3 = 2596

E.

1. As before, without index, passing all rows takes 1667 I/O actions.

2. just like before where we only look at genre optimal indexing degree is $\frac{1000+10}{8+10} = 56$

3. down the B+ tree, takes $log_{28}10000 = 3$

we have $\frac{10000}{4} = 2500$ rows with values of drama as genre

therefore $\frac{2500}{27} = 93$

for a total of 93 + 3 = 96 (since there is no need to leave the branch since the data is already there)

# 4 q1.sql

```
1  select actorId, max(duration) ,min(duration), avg(duration)
2  from playsIn natural join movies
3  group by actorId
4  order by actorId;
```

# 5 q2.sql

```
1  select movieId, title
2  from playsIn natural join movies natural join actors
3  group by movieId, title
4  HAVING avg(movies.year - actors.byear) >= 70
5  ;
```

# 6 q3.sql

```
1  select actorId, name
2  from movies natural join actors natural join playsIn
3  group by actorId, name
4  HAVING avg(rating) >= ALL(
5      select avg(rating)
6      from movies natural join actors natural join playsIn
7      where rating is not null
8      group by actorId
9      )
10 ;
```

# 7 q4.sql

```
1  select count(*) as Num
2  from (
3           select actorId
4           from (
5                    select movieId
6                    from playsIn
7                    group by movieId
8                    having count(*) >= 6) as T1
9                    natural join playsIn
10       except
11
12          select actorId
13          from (
14            select movieId
15            from playsIn
16            group by movieId
17            having count(*) < 6) as T2
18            natural join playsIn
19
20 ) as T
21 ;
```

# 8 q5.sql

```
1  WITH RECURSIVE BaconTable(actorId, movieId, counter) AS
2  (
3      SELECT actors.actorId, movieId, 0
4      FROM actors left join playsIn on
5          actors.actorId = playsIn.actorId
6      WHERE name = 'Frank Bacon'
7      UNION
8      SELECT distinct PI2.actorId, PI2.movieId, counter + 1
9      from PlaysIn PI1 inner join BaconTable on
10         PI1.movieId = BaconTable.movieId and
11         counter < 5 inner join PlaysIn PI2 on
12             PI1.actorId = PI2.actorId
13 )
14 SELECT distinct actorId, name
15 FROM Actors NATURAL JOIN BaconTable
16 order by actorId
17 ;
18
```