

Using VSCode to write the queries in a file and then execute that file from the terminal using “\i file.sql”

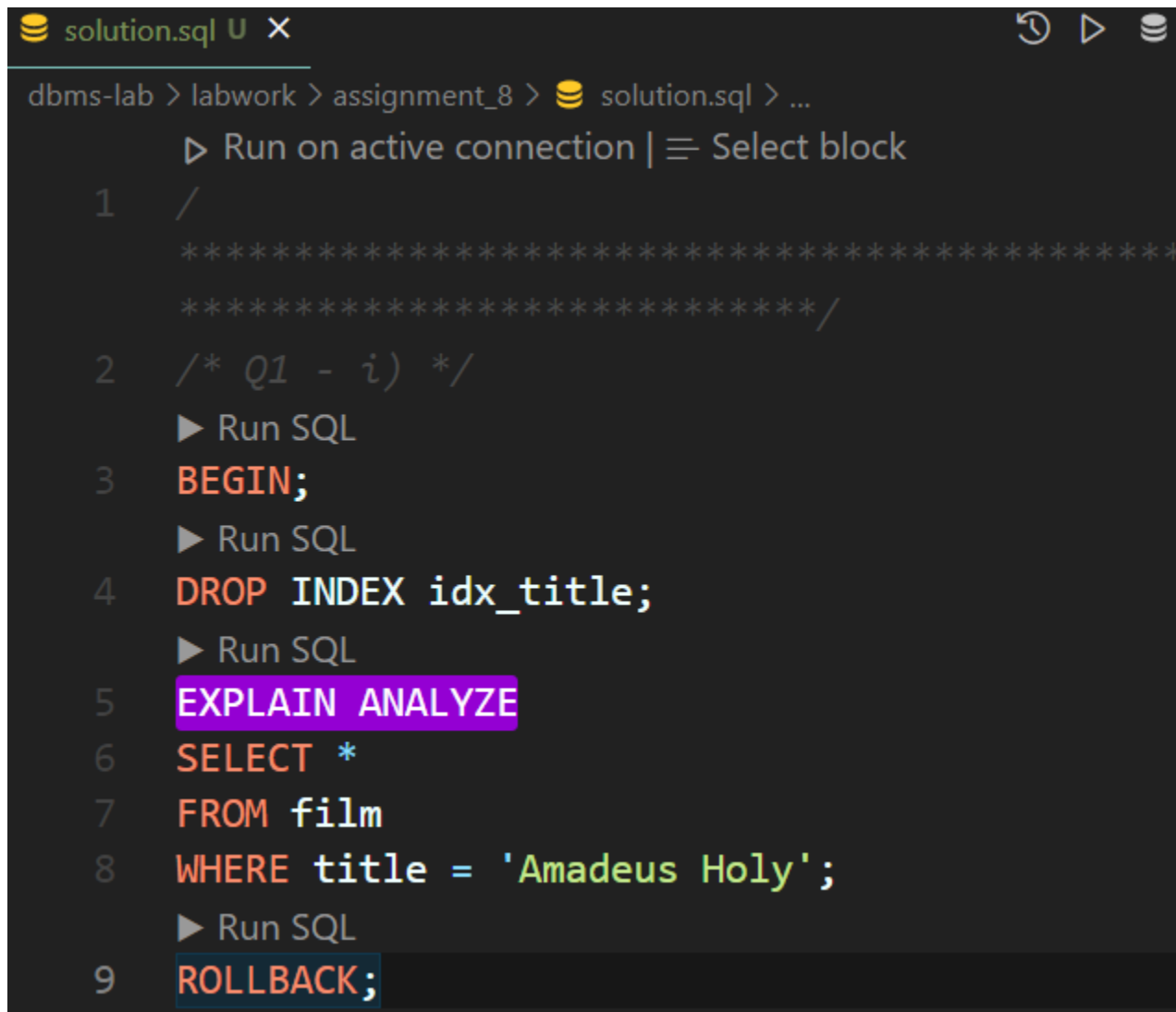
Q1. Consider the following queries to the ‘film’ table from the dvdrental database:

- find the details of the film with the title ‘Amadeus Holy’.
- find all films which have rental rate > 2.

Create an appropriate index on the search keys to make the queries efficient. Justify your index choice by comparing the execution time of the above queries with and without using the index.

i)

Dropping the already present default index and analyzing the query without any index.



```
dbms-lab > labwork > assignment_8 > solution.sql > ...
  ▶ Run on active connection | ≡ Select block
1  /
   ****
   ****/
2  /* Q1 - i) */
   ▶ Run SQL
3  BEGIN;
   ▶ Run SQL
4  DROP INDEX idx_title;
   ▶ Run SQL
5  EXPLAIN ANALYZE
6  SELECT *
7  FROM film
8  WHERE title = 'Amadeus Holy';
   ▶ Run SQL
9  ROLLBACK;
```

```

dvdrental=# \i solution.sql
BEGIN
DROP INDEX

                                QUERY PLAN
-----
Seq Scan on film (cost=0.00..123.50 rows=1 width=391) (actual time=0.016..0.477 rows=1 loops=1)
  Filter: ((title)::text = 'Amadeus Holy'::text)
  Rows Removed by Filter: 999
Planning Time: 0.225 ms
Execution Time: 0.540 ms
(5 rows)

ROLLBACK
dvdrental=# \i solution.sql
BEGIN
DROP INDEX

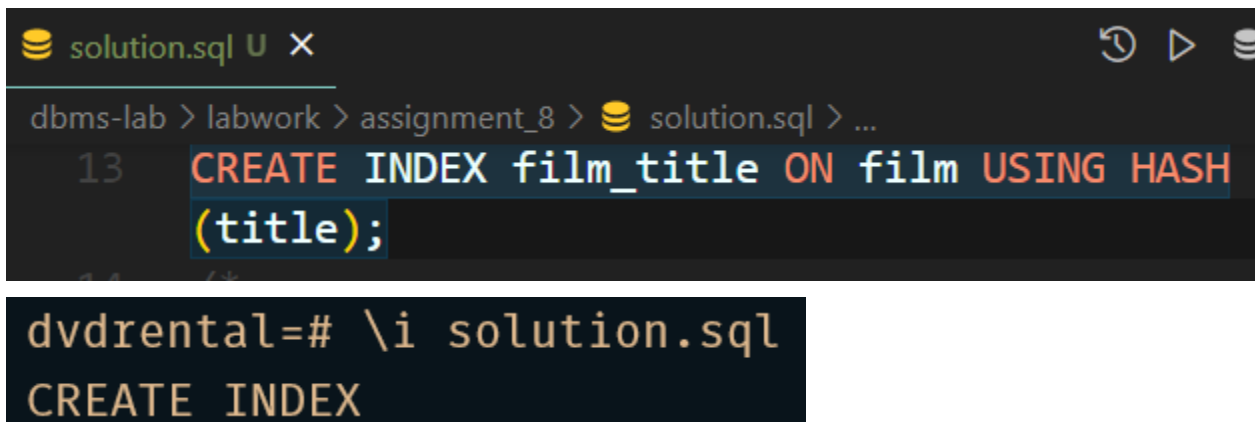
                                QUERY PLAN
-----
Seq Scan on film (cost=0.00..123.50 rows=1 width=391) (actual time=0.014..0.328 rows=1 loops=1)
  Filter: ((title)::text = 'Amadeus Holy'::text)
  Rows Removed by Filter: 999
Planning Time: 0.163 ms
Execution Time: 0.349 ms
(5 rows)

ROLLBACK

```

We can observe that the execution time is around greater than 0.3 ms without using indexing.

Using Hash indexing first, because it is recommended for queries that involve only the = operator.



The screenshot shows a database terminal window with the title 'solution.sql'. The command prompt is 'dbms-lab > labwork > assignment_8 > solution.sql > ...'. The user has entered the command 'CREATE INDEX film_title ON film USING HASH (title);'. Below this, the user has entered 'dvdrental=# \i solution.sql' and 'CREATE INDEX'.

```

solution.sql U X
dbms-lab > labwork > assignment_8 > solution.sql > ...
13 CREATE INDEX film_title ON film USING HASH
    (title);
dvdrental=# \i solution.sql
CREATE INDEX

```

```
dbms-lab > labwork > assignment_8 > solution.sql > ...
```

```
16 EXPLAIN ANALYZE
17 SELECT *
18 FROM film
19 WHERE title = 'Amadeus Holy';
```

```
dvdrental=# \i solution.sql
```

QUERY PLAN

```
-----
Index Scan using film_title on film (cost=0.00..8.02 rows=1 width=391) (actual time=0.067..0.068 rows=1 loops=1)
  Index Cond: ((title)::text = 'Amadeus Holy'::text)
Planning Time: 0.366 ms
Execution Time: 0.100 ms
(4 rows)
```

```
dvdrental=# \i solution.sql
```

QUERY PLAN

```
-----
Index Scan using film_title on film (cost=0.00..8.02 rows=1 width=391) (actual time=0.018..0.020 rows=1 loops=1)
  Index Cond: ((title)::text = 'Amadeus Holy'::text)
Planning Time: 0.101 ms
Execution Time: 0.045 ms
(4 rows)
```

```
dvdrental=# \i solution.sql
```

QUERY PLAN

```
-----
Index Scan using film_title on film (cost=0.00..8.02 rows=1 width=391) (actual time=0.019..0.020 rows=1 loops=1)
  Index Cond: ((title)::text = 'Amadeus Holy'::text)
Planning Time: 0.102 ms
Execution Time: 0.046 ms
(4 rows)
```


```
dvdrental=# \i solution.sql
```


QUERY PLAN

```
-----
Index Scan using film_title on film (cost=0.00..8.02 rows=1 width=391) (actual time=0.017..0.019 rows=1 loops=1)
  Index Cond: ((title)::text = 'Amadeus Holy'::text)
Planning Time: 0.104 ms
Execution Time: 0.046 ms
(4 rows)
```

We can observe that after using Hash indexing on the film title, the execution time reduced to around 0.046 ms.


Using with B-Tree indexing now and dropping the previous hash index. (Just for comparison)


 solution.sql U X

dbms-lab > labwork > assignment_8 >  solution.sql > ...

```
16 CREATE INDEX film_title ON film USING BTREE(title);
```

```
dvdrental=# DROP INDEX film_title;
DROP INDEX
dvdrental=# \i solution.sql
CREATE INDEX
dvdrental=# |
```

 solution.sql U X

dbms-lab > labwork > assignment_8 >  solution.sql > ...

```
16 EXPLAIN ANALYZE
17 SELECT *
18 FROM film
19 WHERE title = 'Amadeus Holy';
```

```

dvdrental=# \i solution.sql

                                QUERY PLAN
-----
Index Scan using film_title on film (cost=0.28..8.29 rows=1 width=391) (actual time=0.099..0.101 rows=1 loops=1)
  Index Cond: ((title)::text = 'Amadeus Holy'::text)
Planning Time: 0.350 ms
Execution Time: 0.136 ms
(4 rows)

dvdrental=# \i solution.sql

                                QUERY PLAN
-----
Index Scan using film_title on film (cost=0.28..8.29 rows=1 width=391) (actual time=0.027..0.028 rows=1 loops=1)
  Index Cond: ((title)::text = 'Amadeus Holy'::text)
Planning Time: 0.103 ms
Execution Time: 0.055 ms
(4 rows)

dvdrental=# \i solution.sql

                                QUERY PLAN
-----
Index Scan using film_title on film (cost=0.28..8.29 rows=1 width=391) (actual time=0.026..0.028 rows=1 loops=1)
  Index Cond: ((title)::text = 'Amadeus Holy'::text)
Planning Time: 0.103 ms
Execution Time: 0.054 ms
(4 rows)

dvdrental=# \i solution.sql

                                QUERY PLAN
-----
Index Scan using film_title on film (cost=0.28..8.29 rows=1 width=391) (actual time=0.026..0.028 rows=1 loops=1)
  Index Cond: ((title)::text = 'Amadeus Holy'::text)
Planning Time: 0.134 ms
Execution Time: 0.054 ms
(4 rows)

```

We can observe that after using B-Tree indexing on the film title, the execution time reduced to around 0.054 ms.

ii)

```

solution.sql U X
dbms-lab > labwork > assignment_8 > solution.sql > ...
26  EXPLAIN ANALYZE
27  SELECT *
28  FROM film
29  WHERE rental_rate > 2;

```

```
dvdrental=# \i solution.sql
```

QUERY PLAN

```
-----  
Seq Scan on film (cost=0.00..123.50 rows=659 width=391) (actual time=0.014..0.642 rows=659 loops=1)  
  Filter: (rental_rate > '2'::numeric)  
    Rows Removed by Filter: 341  
Planning Time: 0.103 ms  
Execution Time: 0.703 ms  
(5 rows)
```

```
dvdrental=# \i solution.sql
```

QUERY PLAN

```
-----  
Seq Scan on film (cost=0.00..123.50 rows=659 width=391) (actual time=0.011..0.643 rows=659 loops=1)  
  Filter: (rental_rate > '2'::numeric)  
    Rows Removed by Filter: 341  
Planning Time: 0.092 ms  
Execution Time: 0.706 ms  
(5 rows)
```

```
dvdrental=# \i solution.sql
```

QUERY PLAN

```
-----  
Seq Scan on film (cost=0.00..123.50 rows=659 width=391) (actual time=0.011..0.615 rows=659 loops=1)  
  Filter: (rental_rate > '2'::numeric)  
    Rows Removed by Filter: 341  
Planning Time: 0.092 ms  
Execution Time: 0.671 ms  
(5 rows)
```


```
dvdrental=# \i solution.sql
```

QUERY PLAN


```
-----  
Seq Scan on film (cost=0.00..123.50 rows=659 width=391) (actual time=0.024..0.642 rows=659 loops=1)  
  Filter: (rental_rate > '2'::numeric)  
    Rows Removed by Filter: 341  
Planning Time: 0.206 ms  
Execution Time: 0.718 ms  
(5 rows)
```

We can observe that the execution time is around greater than 0.7 ms without using indexing.

Using B-Tree indexing, because it is preferred for the queries involving the comparison operators.

 solution.sql U X



```
dbms-lab > labwork > assignment_8 >  solution.sql > ...
```

```
33 CREATE INDEX film_rental_rate ON film USING BTREE  
    (rental_rate);
```

```
dvdrental=# \i solution.sql
CREATE INDEX
```

🍌 solution.sql U X

dbms-lab > labwork > assignment_8 > 🍌 solution.sql > ...

```
26  EXPLAIN ANALYZE
27  SELECT *
28  FROM film
29  WHERE rental_rate > 2;
```

QUERY PLAN

```
-----
Seq Scan on film  (cost=0.00..123.50 rows=659 width=391) (actual time=0.009..0.505 rows=659 loops=1)
  Filter: (rental_rate > '2'::numeric)
  Rows Removed by Filter: 341
Planning Time: 0.077 ms
Execution Time: 0.566 ms
(5 rows)
```

QUERY PLAN

```
-----
Seq Scan on film  (cost=0.00..123.50 rows=659 width=391) (actual time=0.009..0.471 rows=659 loops=1)
  Filter: (rental_rate > '2'::numeric)
  Rows Removed by Filter: 341
Planning Time: 0.072 ms
Execution Time: 0.529 ms
(5 rows)
```

QUERY PLAN

```
-----
Seq Scan on film  (cost=0.00..123.50 rows=659 width=391) (actual time=0.008..0.485 rows=659 loops=1)
  Filter: (rental_rate > '2'::numeric)
  Rows Removed by Filter: 341
Planning Time: 0.076 ms
Execution Time: 0.543 ms
(5 rows)
```

QUERY PLAN

```
-----
Seq Scan on film  (cost=0.00..123.50 rows=659 width=391) (actual time=0.008..0.490 rows=659 loops=1)
  Filter: (rental_rate > '2'::numeric)
  Rows Removed by Filter: 341
Planning Time: 0.071 ms
Execution Time: 0.547 ms
(5 rows)
```

We can observe that after using B-Tree indexing on the film rental rate, the execution time is reduced to around 0.5 ms.

Q2. Create a generalized inverted index on the ``special features`` column of the film table in the dvdrental database. Compare the response time of the following queries with or without using the index.

- find all films with the special feature "Drama"
- find all films with the special feature "Commentaries"

Executing the first query without indexing.

```
dvdrental=# EXPLAIN ANALYZE SELECT * FROM film WHERE 'Drama' = ANY(special_features);
              QUERY PLAN
-----
Seq Scan on film (cost=0.00..133.50 rows=5 width=391) (actual time=0.530..0.531 rows=0 loops=1)
  Filter: ('Drama'::text = ANY (special_features))
  Rows Removed by Filter: 1000
Planning Time: 0.116 ms
Execution Time: 0.555 ms
(5 rows)

dvdrental=# EXPLAIN ANALYZE SELECT * FROM film WHERE 'Drama' = ANY(special_features);
              QUERY PLAN
-----
Seq Scan on film (cost=0.00..133.50 rows=5 width=391) (actual time=0.551..0.551 rows=0 loops=1)
  Filter: ('Drama'::text = ANY (special_features))
  Rows Removed by Filter: 1000
Planning Time: 0.111 ms
Execution Time: 0.581 ms
(5 rows)

dvdrental=# EXPLAIN ANALYZE SELECT * FROM film WHERE 'Drama' = ANY(special_features);
              QUERY PLAN
-----
Seq Scan on film (cost=0.00..133.50 rows=5 width=391) (actual time=0.558..0.559 rows=0 loops=1)
  Filter: ('Drama'::text = ANY (special_features))
  Rows Removed by Filter: 1000
Planning Time: 0.116 ms
Execution Time: 0.583 ms
(5 rows)

dvdrental=# EXPLAIN ANALYZE SELECT * FROM film WHERE 'Drama' = ANY(special_features);
              QUERY PLAN
-----
Seq Scan on film (cost=0.00..133.50 rows=5 width=391) (actual time=0.514..0.515 rows=0 loops=1)
  Filter: ('Drama'::text = ANY (special_features))
  Rows Removed by Filter: 1000
Planning Time: 0.146 ms
Execution Time: 0.541 ms
(5 rows)
```

We can observe that the execution time is around greater than 0.5 ms without using indexing.

Executing the second query without indexing.

```
dvdrental=# EXPLAIN ANALYZE SELECT * FROM film WHERE 'Commentaries' = ANY(special_features);
               QUERY PLAN
-----
Seq Scan on film (cost=0.00..133.50 rows=539 width=391) (actual time=0.013..0.591 rows=539 loops=1)
  Filter: ('Commentaries'::text = ANY (special_features))
  Rows Removed by Filter: 461
  Planning Time: 0.074 ms
  Execution Time: 0.641 ms
(5 rows)

dvdrental=# EXPLAIN ANALYZE SELECT * FROM film WHERE 'Commentaries' = ANY(special_features);
               QUERY PLAN
-----
Seq Scan on film (cost=0.00..133.50 rows=539 width=391) (actual time=0.017..0.615 rows=539 loops=1)
  Filter: ('Commentaries'::text = ANY (special_features))
  Rows Removed by Filter: 461
  Planning Time: 0.086 ms
  Execution Time: 0.715 ms
(5 rows)

dvdrental=# EXPLAIN ANALYZE SELECT * FROM film WHERE 'Commentaries' = ANY(special_features);
               QUERY PLAN
-----
Seq Scan on film (cost=0.00..133.50 rows=539 width=391) (actual time=0.016..0.625 rows=539 loops=1)
  Filter: ('Commentaries'::text = ANY (special_features))
  Rows Removed by Filter: 461
  Planning Time: 0.084 ms
  Execution Time: 0.680 ms
(5 rows)

dvdrental=# EXPLAIN ANALYZE SELECT * FROM film WHERE 'Commentaries' = ANY(special_features);
               QUERY PLAN
-----
Seq Scan on film (cost=0.00..133.50 rows=539 width=391) (actual time=0.017..0.626 rows=539 loops=1)
  Filter: ('Commentaries'::text = ANY (special_features))
  Rows Removed by Filter: 461
  Planning Time: 0.090 ms
  Execution Time: 0.689 ms
(5 rows)
```

We can observe that the execution time is around greater than 0.6 ms without using indexing.

Creating the GIN indexing on the `special_features` attribute.

```
dvdrental=# CREATE INDEX gin_special_features ON film USING GIN(special_features);
CREATE INDEX
dvdrental=# |
```

```

dvdrental=# EXPLAIN ANALYZE SELECT * FROM film WHERE special_features @> ARRAY['Drama'];
               QUERY PLAN
-----
Bitmap Heap Scan on film  (cost=8.04..24.92 rows=5 width=391) (actual time=0.011..0.011 rows=0 loops=1)
  Recheck Cond: (special_features @> '{Drama}':text[])
  -> Bitmap Index Scan on gin_special_features  (cost=0.00..8.04 rows=5 width=0) (actual time=0.009..0.009 rows=0 loops=1)
       Index Cond: (special_features @> '{Drama}':text[])
Planning Time: 0.121 ms
Execution Time: 0.040 ms
(6 rows)

dvdrental=# EXPLAIN ANALYZE SELECT * FROM film WHERE special_features @> ARRAY['Drama'];
               QUERY PLAN
-----
Bitmap Heap Scan on film  (cost=8.04..24.92 rows=5 width=391) (actual time=0.010..0.011 rows=0 loops=1)
  Recheck Cond: (special_features @> '{Drama}':text[])
  -> Bitmap Index Scan on gin_special_features  (cost=0.00..8.04 rows=5 width=0) (actual time=0.009..0.009 rows=0 loops=1)
       Index Cond: (special_features @> '{Drama}':text[])
Planning Time: 0.143 ms
Execution Time: 0.043 ms
(6 rows)

dvdrental=# EXPLAIN ANALYZE SELECT * FROM film WHERE special_features @> ARRAY['Drama'];
               QUERY PLAN
-----
Bitmap Heap Scan on film  (cost=8.04..24.92 rows=5 width=391) (actual time=0.010..0.010 rows=0 loops=1)
  Recheck Cond: (special_features @> '{Drama}':text[])
  -> Bitmap Index Scan on gin_special_features  (cost=0.00..8.04 rows=5 width=0) (actual time=0.008..0.008 rows=0 loops=1)
       Index Cond: (special_features @> '{Drama}':text[])
Planning Time: 0.119 ms
Execution Time: 0.040 ms
(6 rows)

dvdrental=# EXPLAIN ANALYZE SELECT * FROM film WHERE special_features @> ARRAY['Drama'];
               QUERY PLAN
-----
Bitmap Heap Scan on film  (cost=8.04..24.92 rows=5 width=391) (actual time=0.012..0.012 rows=0 loops=1)
  Recheck Cond: (special_features @> '{Drama}':text[])
  -> Bitmap Index Scan on gin_special_features  (cost=0.00..8.04 rows=5 width=0) (actual time=0.009..0.010 rows=0 loops=1)
       Index Cond: (special_features @> '{Drama}':text[])
Planning Time: 0.192 ms
Execution Time: 0.051 ms
(6 rows)

```

We can observe that after using GIN indexing on the `special_features`, the execution time is reduced to around 0.04 ms for the first query.

```

dvdrental=# EXPLAIN ANALYZE SELECT * FROM film WHERE special_features @> ARRAY['Commentaries'];
               QUERY PLAN
-----
Seq Scan on film (cost=0.00..123.50 rows=539 width=391) (actual time=0.014..0.751 rows=539 loops=1)
  Filter: (special_features @> '{Commentaries}'::text[])
  Rows Removed by Filter: 461
  Planning Time: 0.142 ms
  Execution Time: 0.806 ms
(5 rows)

dvdrental=# EXPLAIN ANALYZE SELECT * FROM film WHERE special_features @> ARRAY['Commentaries'];
               QUERY PLAN
-----
Seq Scan on film (cost=0.00..123.50 rows=539 width=391) (actual time=0.015..0.777 rows=539 loops=1)
  Filter: (special_features @> '{Commentaries}'::text[])
  Rows Removed by Filter: 461
  Planning Time: 0.173 ms
  Execution Time: 0.834 ms
(5 rows)

dvdrental=# EXPLAIN ANALYZE SELECT * FROM film WHERE special_features @> ARRAY['Commentaries'];
               QUERY PLAN
-----
Seq Scan on film (cost=0.00..123.50 rows=539 width=391) (actual time=0.015..0.787 rows=539 loops=1)
  Filter: (special_features @> '{Commentaries}'::text[])
  Rows Removed by Filter: 461
  Planning Time: 0.114 ms
  Execution Time: 0.898 ms
(5 rows)

dvdrental=# EXPLAIN ANALYZE SELECT * FROM film WHERE special_features @> ARRAY['Commentaries'];
               QUERY PLAN
-----
Seq Scan on film (cost=0.00..123.50 rows=539 width=391) (actual time=0.015..0.748 rows=539 loops=1)
  Filter: (special_features @> '{Commentaries}'::text[])
  Rows Removed by Filter: 461
  Planning Time: 0.113 ms
  Execution Time: 0.808 ms
(5 rows)

```

We can observe that after using GIN indexing on the `special_features`, the execution time is increased to around 0.8 ms for the second query.

Q3. Create a table called 'dummy2' having 2 attributes i.e. 'sub1' and 'sub2', 'sub1' contains randomly generated text sequences and 'sub2' contains random integer values in the range 0-100. Populate the table with 100000 records. Write a query to filter out records having 'ecr' as a substring in 'sub1' and the value of 'sub2' > 5.

- Analyze the execution time of the query without indexing.
- Use appropriate indexing for cutting down the execution time of the above query.

Finally, drop the created index. Write a short justification for the choice of index.

Creating the table and inserting random data.

```
solution.sql U X
dbms-lab > labwork > assignment_8 > solution.sql > ...
68  /* Q3 */
    ► Run SQL
69  CREATE TABLE dummy2 (
70      sub1 TEXT NOT NULL,
71      sub2 INT NOT NULL CHECK(sub2 >= 0 AND sub2 <= 100)
72  );
73
    ► Run SQL
74  INSERT INTO dummy2(sub1, sub2)
75  SELECT MD5(RANDOM()::TEXT), RANDOM() * 100
76  FROM (
77      SELECT *
78      FROM generate_series(1, 100000) AS id
79  ) AS x;
```

```
mayank=# \i solution.sql
CREATE TABLE
INSERT 0 100000
mayank=# |
```

Executing the query without any indexing.

```
mayank=# EXPLAIN ANALYZE SELECT * FROM dummy2 WHERE sub1 ILIKE '%ecr%' AND sub2 > 5;
               QUERY PLAN
-----
Seq Scan on dummy2 (cost=0.00..2334.00 rows=9 width=37) (actual time=84.853..84.856 rows=0 loops=1)
  Filter: ((sub1 ~* '%ecr% '::text) AND (sub2 > 5))
  Rows Removed by Filter: 100000
  Planning Time: 0.221 ms
  Execution Time: 84.879 ms
(5 rows)

mayank=# EXPLAIN ANALYZE SELECT * FROM dummy2 WHERE sub1 ILIKE '%ecr%' AND sub2 > 5;
               QUERY PLAN
-----
Seq Scan on dummy2 (cost=0.00..2334.00 rows=9 width=37) (actual time=110.456..110.457 rows=0 loops=1)
  Filter: ((sub1 ~* '%ecr% '::text) AND (sub2 > 5))
  Rows Removed by Filter: 100000
  Planning Time: 0.237 ms
  Execution Time: 110.475 ms
(5 rows)

mayank=# EXPLAIN ANALYZE SELECT * FROM dummy2 WHERE sub1 ILIKE '%ecr%' AND sub2 > 5;
               QUERY PLAN
-----
Seq Scan on dummy2 (cost=0.00..2334.00 rows=9 width=37) (actual time=117.146..117.147 rows=0 loops=1)
  Filter: ((sub1 ~* '%ecr% '::text) AND (sub2 > 5))
  Rows Removed by Filter: 100000
  Planning Time: 0.389 ms
  Execution Time: 117.214 ms
(5 rows)

mayank=# EXPLAIN ANALYZE SELECT * FROM dummy2 WHERE sub1 ILIKE '%ecr%' AND sub2 > 5;
               QUERY PLAN
-----
Seq Scan on dummy2 (cost=0.00..2334.00 rows=9 width=37) (actual time=86.670..86.671 rows=0 loops=1)
  Filter: ((sub1 ~* '%ecr% '::text) AND (sub2 > 5))
  Rows Removed by Filter: 100000
  Planning Time: 0.357 ms
  Execution Time: 86.693 ms
(5 rows)
```

Creating GIN index for `sub1` because it is efficient with the trigram operations.

Creating B-Tree index for the `sub2` because it is recommended to use with the queries involving comparison operators.

```
mayank=# CREATE INDEX sub1_idx ON dummy2 USING GIN(sub1 GIN_TRGM_OPS);
CREATE INDEX
mayank=# CREATE INDEX sub2_idx ON dummy2 USING BTREE(sub2);
CREATE INDEX
mayank=# |
```

```

mayank=# EXPLAIN ANALYZE SELECT * FROM dummy2 WHERE sub1 ILIKE '%ecr%' AND sub2 > 5;
               QUERY PLAN
-----
Bitmap Heap Scan on dummy2 (cost=16.08..52.94 rows=9 width=37) (actual time=0.013..0.014 rows=0 loops=1)
  Recheck Cond: (sub1 ~* '%ecr% '::text)
  Filter: (sub2 > 5)
  -> Bitmap Index Scan on sub1_idx (cost=0.00..16.07 rows=10 width=0) (actual time=0.011..0.012 rows=0 loops=1)
       Index Cond: (sub1 ~* '%ecr% '::text)
Planning Time: 0.305 ms
Execution Time: 0.038 ms
(7 rows)

mayank=# EXPLAIN ANALYZE SELECT * FROM dummy2 WHERE sub1 ILIKE '%ecr%' AND sub2 > 5;
               QUERY PLAN
-----
Bitmap Heap Scan on dummy2 (cost=16.08..52.94 rows=9 width=37) (actual time=0.013..0.014 rows=0 loops=1)
  Recheck Cond: (sub1 ~* '%ecr% '::text)
  Filter: (sub2 > 5)
  -> Bitmap Index Scan on sub1_idx (cost=0.00..16.07 rows=10 width=0) (actual time=0.011..0.012 rows=0 loops=1)
       Index Cond: (sub1 ~* '%ecr% '::text)
Planning Time: 0.356 ms
Execution Time: 0.039 ms
(7 rows)

mayank=# EXPLAIN ANALYZE SELECT * FROM dummy2 WHERE sub1 ILIKE '%ecr%' AND sub2 > 5;
               QUERY PLAN
-----
Bitmap Heap Scan on dummy2 (cost=16.08..52.94 rows=9 width=37) (actual time=0.013..0.014 rows=0 loops=1)
  Recheck Cond: (sub1 ~* '%ecr% '::text)
  Filter: (sub2 > 5)
  -> Bitmap Index Scan on sub1_idx (cost=0.00..16.07 rows=10 width=0) (actual time=0.011..0.011 rows=0 loops=1)
       Index Cond: (sub1 ~* '%ecr% '::text)
Planning Time: 0.318 ms
Execution Time: 0.037 ms
(7 rows)

```

We can observe that the query execution time is reduced significantly to 0.03 ms.

Dropping the created index.

```

mayank=# DROP INDEX sub1_idx;
DROP INDEX
mayank=# DROP INDEX sub2_idx;
DROP INDEX
mayank=# |

```