Project Report
CIS - 556

Team Member and Responsibilities:

1. **Abhishek Kumar Pathak (UMID-36874067):** ER Diagram, DDL and DML
2. **Faiyaz Ahmad (UMID- 74610524):** SQL Queries and Performance Profiling
3. **Sridhar Gangadhara (UMID- 94017183):** Data Preparation, Indexing and Final Report.

Project Description:

The aim of this project is to compare the performance of different queries in PostgreSQL against the IMDb dataset with and without indexing.

This project will cover following concepts of CIS 556:

1. To design an ER Model for a real-world dataset and translate into SQL schema.
2. To populate data into database using DML statements.
3. To write SQL queries.
4. To be able to deploy indexing into database.
5. To measure the performance of SQL queries with and without indexing.

Files Attached:

1. **Code.zip:** In the Code folder we have DDL & DML statements and SQL Queries
2. **Output.zip:** In the Output folder we have Query output images, CSV files of Queries EXPLAIN ANALYZE with and without indexing and CSV files for Query outputs.
3. **movies_data_preprocessing.py:** In this file we have python script

Dataset:

Dataset name: IMDb

Description: Real database of IMDb movies. It contains 7 tables. It contains details of several relation between movies, directors, genre, actors and roles.

Drive link for csv file:
https://drive.google.com/drive/folders/1V_IxQPI6ecEuD6bJjrVVQIrqqpwh_Df0?usp=share_link

Drive link for Preprocessed file:

https://drive.google.com/drive/folders/15w7b6ar95Ilu1CY50FdT_DxlxV0Uq11X?usp=sharing

Source Link: https://relational.fit.cvut.cz/dataset/IMDb

Dataset preprocessing:

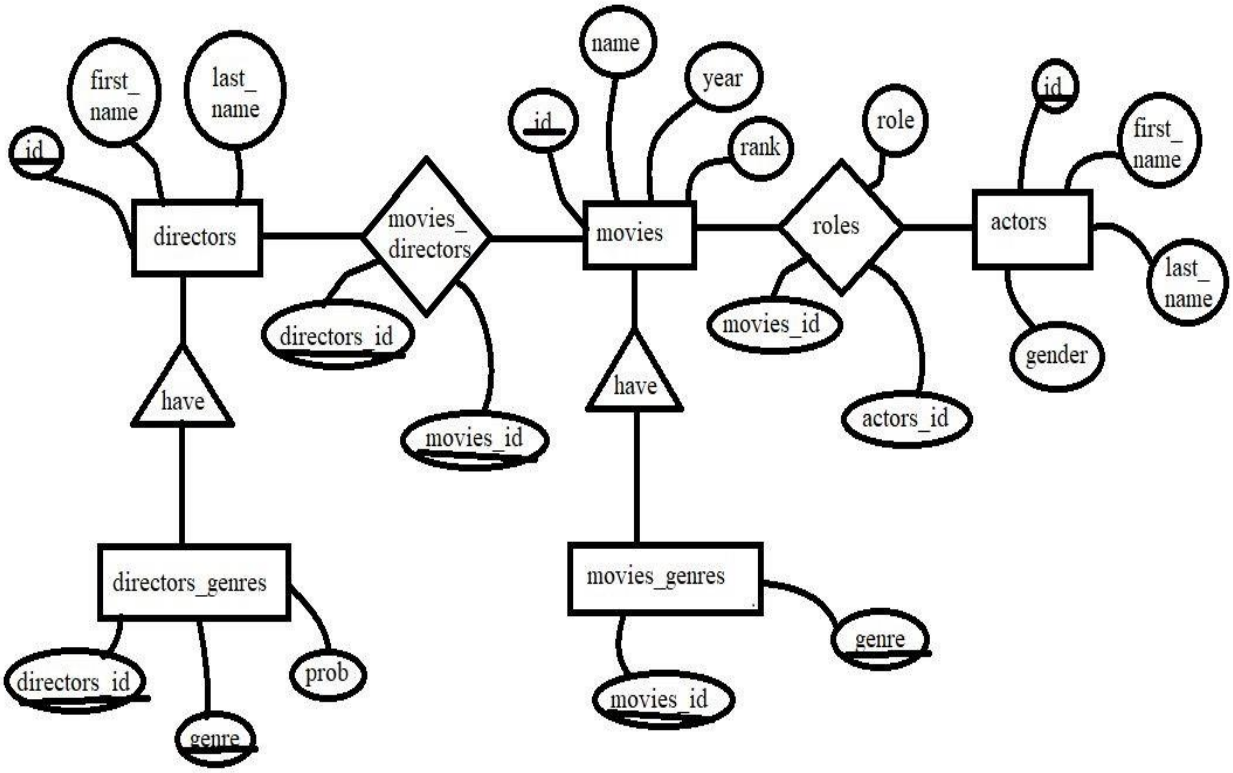
The rank column of the movies table had 'NULL' string and due to float data type constraint, we were unable to populate the data,

Hence, we preprocessed the movies table (movies.csv) by using python script: **movies_data_preprocessing.py**

Command: % python movies_data_preprocessing.py movies.csv

This python script replaces the 'NULL' string with empty value in 'rank' column in movies table.

ER Diagram:


first_
name
last_
name
id
directors
movies_
directors
directors_id
movies_id
have
directors_genres
directors_id
genre
prob
name
year
id
rank
movies
have
movies_genres
movies_id
genre
role
roles
movies_id
actors_id
id
first_
name
actors
last_
name
gender

Database Schema:

We have modified the above ER into the following DDL Statements:

```
DROP TABLE IF EXISTS directors CASCADE;
CREATE TABLE directors (
        id int PRIMARY KEY,
        first_name varchar(100),
        last_name varchar(100)
);
DROP TABLE IF EXISTS movies CASCADE;
CREATE TABLE movies (
        id int PRIMARY KEY,
        name varchar(100),
            year int,
        rank float
);

DROP TABLE IF EXISTS actors CASCADE;
CREATE TABLE actors (
        id int PRIMARY KEY,
        first_name varchar(100),
        last_name varchar(100),
        gender char(1)
);

DROP TABLE IF EXISTS movies_directors CASCADE;
CREATE TABLE movies_directors(
                director_id int references directors(id) NOT NULL,
```

```
        movie_id int references movies(id) NOT NULL,
        PRIMARY KEY (movie_id,director_id)
);

DROP TABLE IF EXISTS roles CASCADE;
CREATE TABLE roles(
        actor_id int references actors(id) NOT NULL,
        movie_id int references movies(id) NOT NULL,
        role varchar
);

DROP TABLE IF EXISTS directors_genres CASCADE;
CREATE TABLE directors_genres(
        director_id int references directors(id) NOT NULL,
        genre varchar(100) ,
        prob float,
        PRIMARY KEY (director_id, genre)
);

DROP TABLE IF EXISTS movies_genres CASCADE;
CREATE TABLE movies_genres(
        movie_id int references movies(id) NOT NULL,
        genre varchar(100) ,
        PRIMARY KEY (movie_id, genre)
);
```

DML Statements:

With the following DML statements, we have populated the data into our schema

```
COPY directors
FROM 'C:\Program Files\PostgreSQL\15\pgAdmin 4\New folder\directors.csv'
DELIMITER ',' CSV ;

COPY movies
FROM 'C:\Program Files\PostgreSQL\15\pgAdmin 4\New folder\movies.csv'
DELIMITER ',' CSV ;

COPY actors
FROM 'C:\Program Files\PostgreSQL\15\pgAdmin 4\New folder\actors.csv'
DELIMITER ',' CSV ;

COPY movies_directors
FROM 'C:\Program Files\PostgreSQL\15\pgAdmin 4\New folder\movies_directors.csv'
DELIMITER ',' CSV;

COPY roles
FROM 'C:\Program Files\PostgreSQL\15\pgAdmin 4\New folder\roles.csv'
DELIMITER ',' CSV ;


COPY directors_genres
FROM 'C:\Program Files\PostgreSQL\15\pgAdmin 4\New folder\directors_genres.csv'
DELIMITER ',' CSV HEADER;
```

```
COPY movies_genres
FROM 'C:\Program Files\PostgreSQL\15\pgAdmin 4\New folder\movies_genres.csv'
DELIMITER ',' CSV;
```

Methodology:

We have used the Queries given below to test the performance.

Queries:

Query1: Retrieve id,first_name and last_name of actor whose first_name is 'Luis'

```
select id,first_name,last_name from actors where first_name='Luis';
```

Query2: Retrieve all the movies name where 'painter' role is casted.

```
select movies.name from movies join
(select distinct movie_id from roles where role='painter') as sq1
on (sq1.movie_id=movies.id);
```

Query3: List the first and last names of all the actors who were cast in the 'Rush Hour' movie:

```
select first_name,last_name
from
actors as ac
join
(select distinct actor_id from roles as rc
 join
```

```
(select id from movies where name='Rush Hour' ) as mv on (mv.id=rc.movie_id))
as pq
on (pq.actor_id=ac.id)
order by first_name, last_name;
```

Query4: List all the directors (first and last names) who directed movies who casted the role of 'Musician'.

```
select first_name,last_name
from
directors as dc
join
(select distinct director_id from movies_directors as md
 natural join
(select distinct movie_id from roles where role='Musician') as di)
as dq
on (dq.director_id=dc.id)
order by first_name, last_name;
```

Query5: List all the actors who acted in a film before 1950 and also in a film after 2000.

```
select * from
(select first_name,last_name from actors join
(select distinct actor_id from roles join (select id from movies where year>=2000) as m
on (roles.movie_id=m.id)) as dcm
on (dcm.actor_id=actors.id)) as l1
INTERSECT
(select first_name,last_name from actors join
```

```
(select distinct actor_id from roles join (select id from movies where year<=1950) as m
on (roles.movie_id=m.id)) as dcm
on (dcm.actor_id=actors.id))
order by first_name, last_name;
```

Query6: Retrieve all the directors name who directed more than 10 'Thriller' movies in movies_genres.

```
select first_name,last_name from directors natural join
(select director_id as id, count(*) as cs from movies_directors natural join
(select movie_id from movies_genres where genre='Thriller') as md
group by director_id) as sq
where cs>10
order by cs DESC;
```

Query7: Write query to retrieve all the actors' first and last name, the movie name, the number of distinct roles where one actor played more than 5 roles in that particular movie.

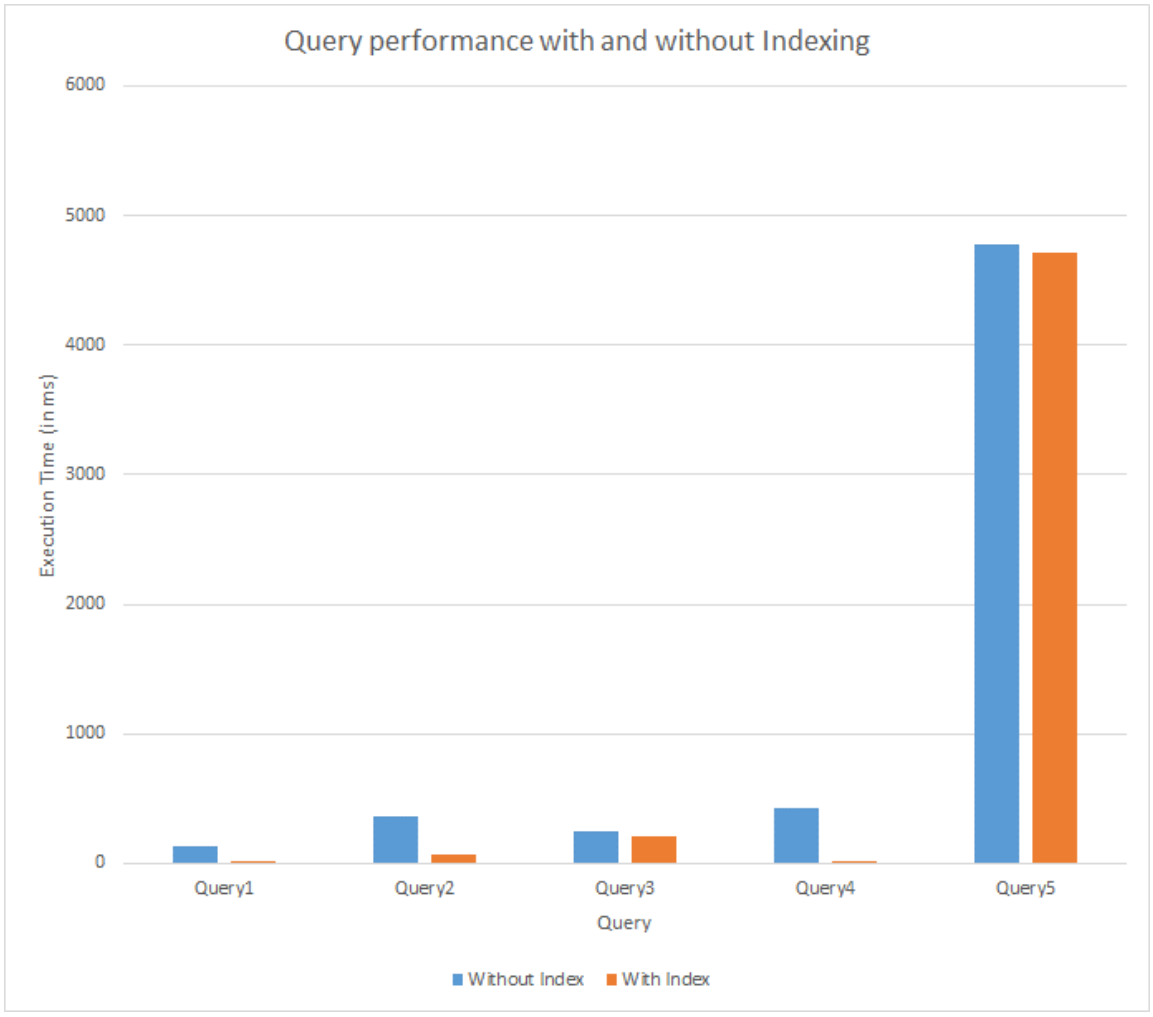
```
select name as movie_name,first_name,last_name,role_n from actors join
(select movies.name,actor_id,role_n from movies join
(select movie_id,actor_id,count(distinct role) as role_n from roles
group by movie_id,actor_id) as sq1
on (movies.id=sq1.movie_id)) as sq2
on (actors.id=sq2.actor_id)
where role_n>5
order by role_n DESC;
```

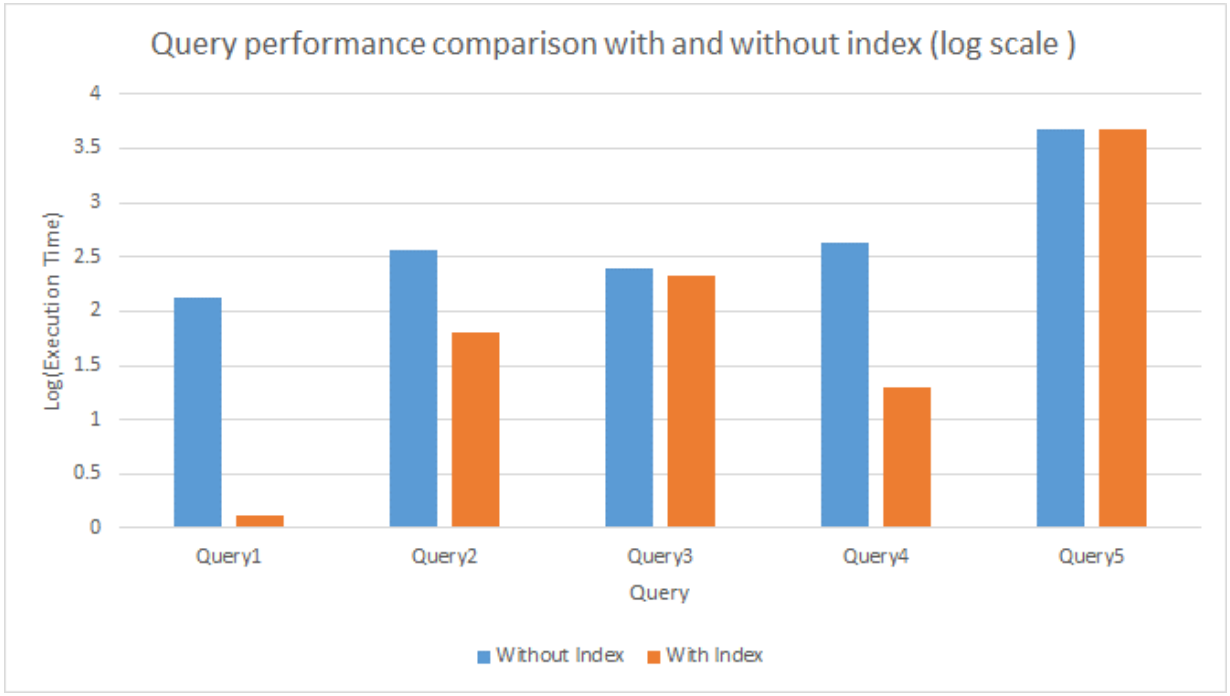
Query8: Calculate the percentage of Male and Female actors in actors table.

```
select sq1.gender,sq1.count*100/sum(sq1.count) over () as percentage from
(select gender,count(*) from actors group by gender) as sq1;
```

Benchmarks:

Query	Execution time (in ms)	
	Without Index	With Index
Query1	135.193	1.329
Query2	366.578	63.739
Query3	250.752	213.216
Query4	426.115	20.117
Query5	4772.872	4709.094


Query performance with and without Indexing
6000
5000
4000
3000
2000
1000
0
Execution Time (in ms)
Query1
Query2
Query3
Query4
Query5
Query
Without Index
With Index


Query performance comparison with and without index (log scale)
4
3.5
3
2.5
2
1.5
1
0.5
0
Log(Execution Time)
Query1
Query2
Query3
Query4
Query5
Query
Without Index
With Index

Instructions for reproducing the experiments:

1. Download the data set (in CSV format) from the given site https://relational.fit.cvut.cz/dataset/IMDb
2. We preprocessed the movies table (movies.csv) by using python script:
   movies_data_preprocessing.py
   Command: % python movies_data_preprocessing.py movies.csv
   This python script replaces the 'NULL' string with empty value in 'rank' column in movies table.
3. Using the DDL statements, we have created the schema in PostgreSQL
4. To populate the data in the database, we used the DML statements.
5. Using the Queries, we have recorded the performance without indexing using EXPLAIN ANALYZE commands.
6. We created indexes on certain columns on which the above queries work
7. We executed the queries again after indexing to record the performance using EXPLAIN ANALYZE commands.
8. Then we compared the results and tabulate it to compare the performance.

Conclusion:

From the above output results and the consequent graphs, it can be seen that the performance of the lookup Queries is significantly higher with indexing as compared to that of without indexing.