Database Management System Project - UE19CS301
Movie ticket booking system

Assignment 4

Team Details:

Name	**SRN**
Prakruti PK	PES1UG19CS338
Pruthvi P	PES1UG19CS355
Raghav Pandit	PES1UG19CS364

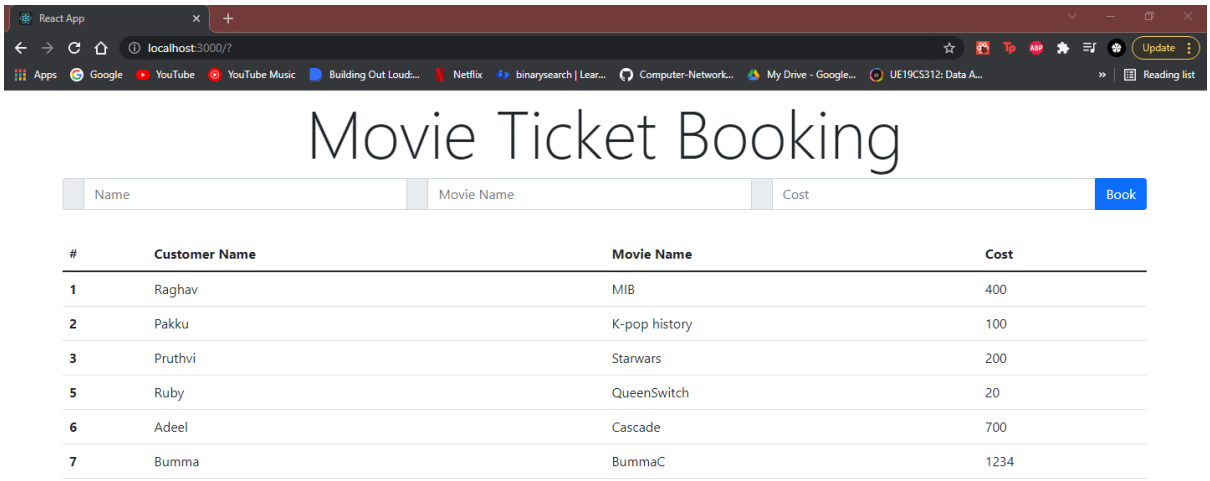
USER INTERFACE

PERN stack was used to implement the linking of the front-end(React) and the backend(nodeJS) to the database(PostgreSQL) and express was used to build the website quickly and relatively easily

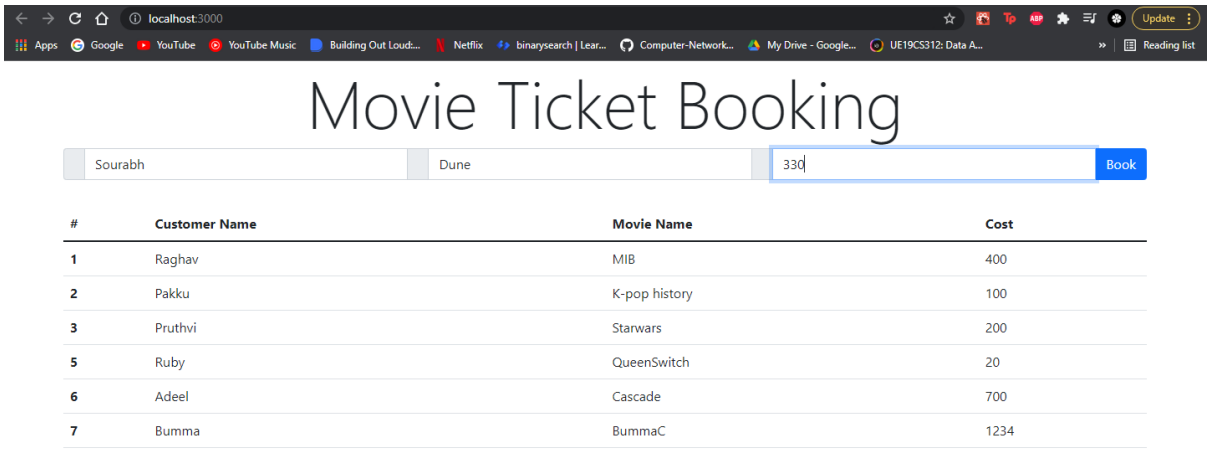
PERN stack was a choice of front-end package because:

- Backend code was easier to write and implement
- Supports many middleware
- Creating efficient and robust API is quick and easy
- PostgreSQL is a great choice as it's stable and is great for complex duties

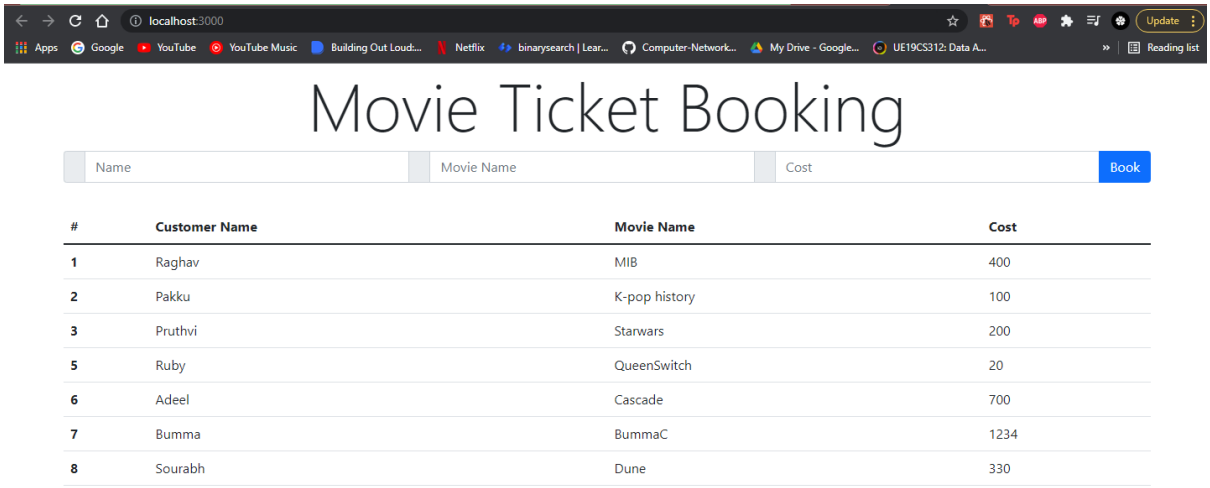
1.Pre-Existing Movie tickets



2.New Ticket information added to the input fields



3.New Ticket data inserted into the database and shown on the front-end screen and is confirmed by viewing the console




```
movie=# select * from Ticket_info;
 t_id | c_name   |     m_name     | cost
------+----------+----------------+------
    1 | Raghav   | MIB            |  400
    2 | Pakku    | K-pop history  |  100
    3 | Pruthvi  | Starwars       |  200
    5 | Ruby     | QueenSwitch    |   20
    6 | Adeel    | Cascade        |  700
    7 | Bumma    | BummaC         | 1234
(6 rows)


movie=# select * from Ticket_info;
 t_id | c_name   |     m_name     | cost
------+----------+----------------+------
    1 | Raghav   | MIB            |  400
    2 | Pakku    | K-pop history  |  100
    3 | Pruthvi  | Starwars       |  200
    5 | Ruby     | QueenSwitch    |   20
    6 | Adeel    | Cascade        |  700
    7 | Bumma    | BummaC         | 1234
    8 | Sourabh  | Dune           |  330
(7 rows)
```

SCHEMA CHANGES:

1) **Addition of a director column to the movie table help identify movies better:**
   ALTER table Movie
   ADD column Director VARCHAR;

```
movie=# ALTER table Movie
ADD column Director VARCHAR;
ALTER TABLE
movie=# Select * from Movie;
 m_id | m_name  | duration | release_date | rating | director
------+---------+----------+--------------+--------+----------
 001  | Hella   |      120 | 2012-12-10   |      4 |
 002  | Ava     |      122 | 2002-11-15   |      4 |
 003  | Batman  |      100 | 2013-04-23   |      4 |
 004  | Run     |       98 | 1982-03-17   |      4 |
 005  | Time    |      128 | 2007-07-28   |      4 |
 006  | Dune    |      140 | 2021-09-09   |      4 |
 007  | Vertigo |      210 | 2011-11-11   |      4 |
 008  | Big     |      138 | 2010-08-10   |      4 |
 009  | Up      |      120 | 2015-11-09   |      4 |
(9 rows)
```

2) **Adding director names to the newly created column**
   >UPDATE Movie set director = 'Anna' WHERE movie.m_id = '003';
   >UPDATE Movie set director = 'David' WHERE movie.m_id = '007';
   >UPDATE Movie set director = 'Hatiti' WHERE movie.m_id = '004';
   >UPDATE Movie set director = 'Lux' WHERE movie.m_id = '001';

```
movie=# UPDATE Movie
set director = 'Anna'
WHERE movie.m_id = '003';
UPDATE 1
movie=# UPDATE Movie
set director = 'David'
WHERE movie.m_id = '007';
UPDATE 1
movie=# UPDATE Movie
set director = 'Hatiti'
WHERE movie.m_id = '004';
UPDATE 1
movie=# UPDATE Movie
set director = 'Lux'
WHERE movie.m_id = '001';
UPDATE 1
movie=# Select * from Movie;
 m_id | m_name  | duration | release_date | rating | director
------+---------+----------+--------------+--------+----------
 002  | Ava     |      122 | 2002-11-15   |      4 |
 005  | Time    |      128 | 2007-07-28   |      4 |
 006  | Dune    |      140 | 2021-09-09   |      4 |
 008  | Big     |      138 | 2010-08-10   |      4 |
 009  | Up      |      120 | 2015-11-09   |      4 |
 003  | Batman  |      100 | 2013-04-23   |      4 | Anna
 007  | Vertigo |      210 | 2011-11-11   |      4 | David
 004  | Run     |       98 | 1982-03-17   |      4 | Hatiti
 001  | Hella   |      120 | 2012-12-10   |      4 | Lux
(9 rows)
```

3) **Make phone number a unique constraint in customer table so that it can be used for login.**

Alter table customer ADD UNIQUE (phone);

```
movie=# Alter table customer ADD
movie-# UNIQUE (phone);
ALTER TABLE
movie=# \d+ customer
                                        Table "public.customer"
 Column |         Type          | Collation | Nullable | Default | Storage  | Stats target | Description
--------+-----------------------+-----------+----------+---------+----------+--------------+-------------
 c_id   | character varying(20) |           | not null |         | extended |              |
 c_name | character varying(20) |           | not null |         | extended |              |
 age    | integer               |           |          |         | plain    |              |
 mail   | character varying(40) |           |          |         | extended |              |
 phone  | bigint                |           | not null |         | plain    |              |
Indexes:
    "customer_pkey" PRIMARY KEY, btree (c_id)
    "customer_phone_key" UNIQUE CONSTRAINT, btree (phone)
Referenced by:
    TABLE "discount" CONSTRAINT "discount_c_id_fkey" FOREIGN KEY (c_id) REFERENCES customer(c_id)
    TABLE "soldto" CONSTRAINT "soldto_c_id_fkey" FOREIGN KEY (c_id) REFERENCES customer(c_id)
Access method: heap
```

4)Make mail ID a unique constraint in customer table so that it can be used for login.

Alter table customer ADD UNIQUE (mail);

```
movie=# Alter table customer ADD
UNIQUE (mail);
ALTER TABLE
movie=# \d+ customer
                                        Table "public.customer"
 Column |         Type          | Collation | Nullable | Default | Storage  | Stats target | Description
--------+-----------------------+-----------+----------+---------+----------+--------------+-------------
 c_id   | character varying(20) |           | not null |         | extended |              |
 c_name | character varying(20) |           | not null |         | extended |              |
 age    | integer               |           |          |         | plain    |              |
 mail   | character varying(40) |           |          |         | extended |              |
 phone  | bigint                |           | not null |         | plain    |              |
Indexes:
    "customer_pkey" PRIMARY KEY, btree (c_id)
    "customer_mail_key" UNIQUE CONSTRAINT, btree (mail)
    "customer_phone_key" UNIQUE CONSTRAINT, btree (phone)
Referenced by:
    TABLE "discount" CONSTRAINT "discount_c_id_fkey" FOREIGN KEY (c_id) REFERENCES customer(c_id)
    TABLE "soldto" CONSTRAINT "soldto_c_id_fkey" FOREIGN KEY (c_id) REFERENCES customer(c_id)
Access method: heap
```

5) Add phone number for ticket master so they can be contacted easily.

ALTER table ticketmaster ADD column phone VARCHAR;

```
movie=# ALTER table ticketmaster
ADD column phone VARCHAR;
ALTER TABLE
movie=# Select * from ticketmaster;
 tm_id |        tm_name        | phone
-------+-----------------------+-------
 12521 | Ava                   |
 12564 | Olivia                |
 12123 | Liam                  |
 12632 | Elizabeth             |
 12742 | Noah                  |
 12835 | Amy                   |
 12832 | Oliver                |
 12735 | Glenn                 |
 12111 | Elijah                |
(9 rows)
```

DATABASE MIGRATION AND SUPPORT

We are currently using a postgresSQL as our database to store all our data. However, we plan on explaining the functionalities that our product can perform.

Some functionalities include:

- Allow customers to give ratings and reviews for movies
- Allow users to provide various preferences like language, genre.
- Introduce a recommendation system based on their preferences and past ratings and reviews which helps users decide which movie they'd want to watch.
- Introduce hashtag use to better move through the application.

After viewing the possible expansion ideas it makes best sense to migrate to a NoSQL database because it allows for

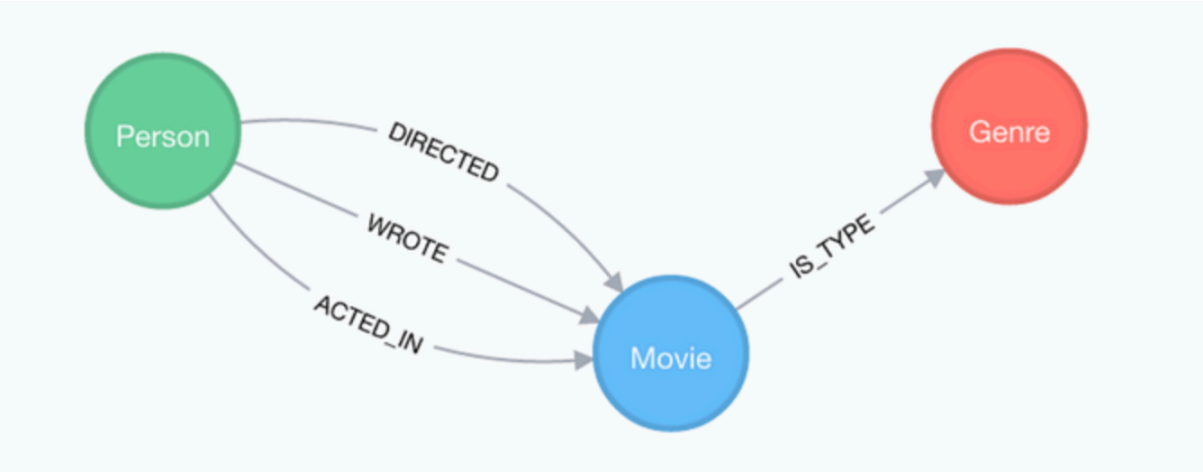
- Flexible schemas
- Horizontal scaling
- Fast queries due to the data model
- Ease of use for developers

The major kinds of NoSQL databases are

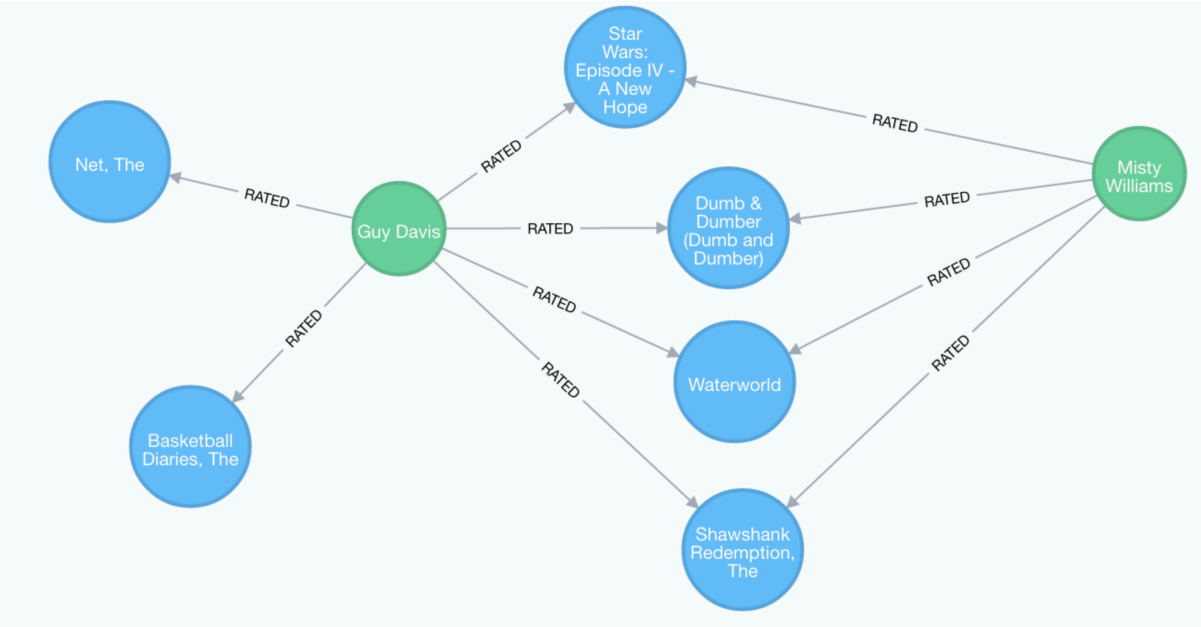
1. Document DB
2. Key-value DB
3. Column oriented DB
4. Graph DB

For our requirements it best to migrate to a graph based database.
graph database is a database designed to treat the relationships between data as equally important to the data itself. It is intended to hold data without constricting it to a pre-defined model. Instead, the data is stored like we first draw it out - showing how each individual entity connects with or is related to others.

Neo4j is the only enterprise strength graph database that combines native graph storage, advanced security, and scalable speed- optimized architecture. Neo4j also allows for the recommendation system to be built with ease with it method of representation.



A possible representation:



Steps for data migration from Postgres to Neo4j:

1. Prepare SQL query to select data and export them to CSV file.
2. Prepare CYPHER queries with usage of LOAD CSV tool to import data.
3. Run script from first step on postgres
4. Get exported csv files from postgres container.
5. Move exported csv files to neo4j container to “import” directory
6. Move script prepared in step 2 also to this docker container but to /tmp directory
7. Use cypher-shell tool with prepared script as input.
8. Remove temporary files

CONTRIBUTIONS

Name	Work	Time
Pruthvi P	Schema changes, Database Migration and support, Report,	4
Raghav Pandit	Frontend, Report	10