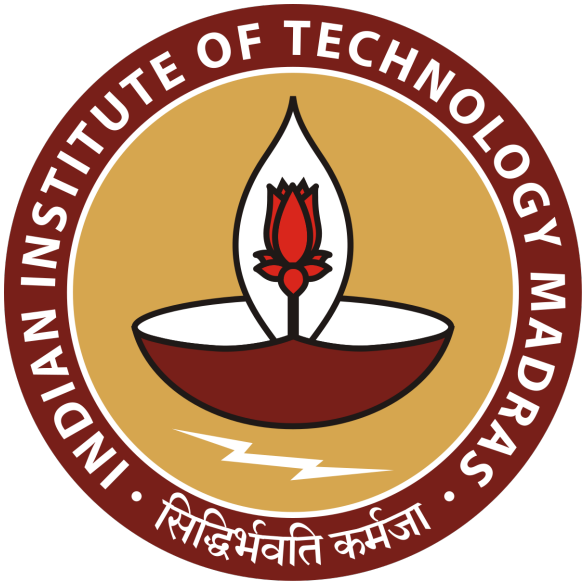
CS3700 - Introduction to Database Systems

Assignment 3 - SQL on Group's DB



Name - Jay Prajapati
Roll Number - ME21B143
Group No. - 11

Table of Contents

Note: The following queries are based on the database schema (GrabYourTicket.sql) provided in the zip file. The queries are designed to extract specific information from the database, and the results are presented in tabular format. Each query is explained in detail, including its breakdown and output. The last two queries involve aggregate functions and group by clauses, which are also explained in detail. The output of each query is presented in a separate figure for clarity.

1 Query 1

Give all movies and sequels along with their venues where they are being played.

```
SELECT
  DISTINCT m1.name AS movie_name,
  v1.type AS movie_venue_type,
  v1.city AS movie_city,
  m2.name AS sequel_name,
  v2.type AS sequel_venue_type,
  v2.city AS sequel_city
FROM Movie m1
JOIN Sequel s ON m1.movieID = s.movieID
JOIN Movie m2 ON s.sequelID = m2.movieID
JOIN PlayedAt pa1 ON m1.movieID = pa1.movieID
JOIN Venue v1 ON pa1.venueID = v1.venueID
JOIN PlayedAt pa2 ON m2.movieID = pa2.movieID
JOIN Venue v2 ON pa2.venueID = v2.venueID;
```

Breakdown of the query:

1. **Original Movies (m1):** The query starts by selecting a list of movies, referred to as "original movies."
2. **Sequel Relationships (s):** It identifies which movies have sequels using a table that links movies to their sequels.
3. **Sequel Movies (m2):** Once the sequel is identified, it fetches details about the sequel movie, such as its name.
4. **Where Original Movies Were Shown (pa1 and v1):** The query looks at a table that keeps track of where each original movie was shown. It then gathers details about these locations, including:
   - The type of venue
   - The city where it was shown
5. **Where Sequels Were Shown (pa2 and v2):** Similarly, it checks where each sequel was shown and collects information about those venues, including:
   - The type of venue
   - The city where it was shown

6. **Final Output:** Combines all this information to produce a list showing:
    - The name of the original movie
    - The type of venue and city where it was shown
    - The name of its sequel
    - The type of venue and city where the sequel was shown

Output of the query:

movie_name	movie_venue_type	movie_city	sequel_name	sequel_venue_type	sequel_city
Avengers: Endgame	Concert Hall	Mumbai	The Avengers	Stadium	Kolkata
The Lord of the Rings: The Fellowship of the Ring	Auditorium	Chennai	The Lord of the Rings: The Two Towers	Opera House	Hyderabad
The Lord of the Rings: The Fellowship of the Ring	Opera House	Hyderabad	The Lord of the Rings: The Two Towers	Opera House	Hyderabad
Star Wars: Episode IV - A New Hope	Amphitheater	Pune	Star Wars: Episode V - The Empire Strikes Back	Amphitheater	Pune
Star Wars: Episode IV - A New Hope	Amphitheater	Pune	Star Wars: Episode V - The Empire Strikes Back	Music Venue	Ahmedabad
Star Wars: Episode V - The Empire Strikes Back	Amphitheater	Pune	Star Wars: Episode VI - Return of the Jedi	Music Venue	Ahmedabad
Star Wars: Episode V - The Empire Strikes Back	Amphitheater	Pune	Star Wars: Episode VI - Return of the Jedi	Art Gallery	Jaipur
Star Wars: Episode V - The Empire Strikes Back	Music Venue	Ahmedabad	Star Wars: Episode VI - Return of the Jedi	Music Venue	Ahmedabad
Star Wars: Episode V - The Empire Strikes Back	Music Venue	Ahmedabad	Star Wars: Episode VI - Return of the Jedi	Art Gallery	Jaipur
The Godfather	Opera House	Hyderabad	The Godfather: Part II	Cinema	Lucknow

2 Query 2

Give list of friends who are from different city, but have seen the same movie in theatre. Also give their email ids.

```
SELECT
    u1.name AS user1_name,
    u1.email AS user1_email,
    u2.name AS user2_name,
    u2.email AS user2_email,
    m1.name AS movie_name
FROM User u1
JOIN Friend f ON u1.userID = f.userID
JOIN User u2 ON f.friendID = u2.userID
JOIN Seat s1 ON u1.userID = s1.reservingUser
JOIN Shows sh1 ON s1.showID = sh1.showID
JOIN Movie m1 ON sh1.movieID = m1.movieID
JOIN Seat s2 ON u2.userID = s2.reservingUser
JOIN Shows sh2 ON s2.showID = sh2.showID
JOIN Movie m2 ON sh2.movieID = m2.movieID
WHERE m1.movieID = m2.movieID AND u1.city <>
   u2.city;
```

Breakdown of the query:

1. **What We're Looking For**
   - `user1_name` and `user1_email`: First person's name and email
   - `user2_name` and `user2_email`: Their friend's name and email
   - `movie_name`: Title of the movie they both watched
2. **How We Find It**
   - Start with the first person's profile
   - Find their friends through the friendship connections
   - Check both people's ticket reservations
   - Look up which movie screenings these tickets correspond to
   - Verify they watched the same movie
   - Confirm they live in different cities
3. **Key Connections**
   - People → Friends: Linked through user IDs
   - Tickets → Screenings: Each ticket corresponds to a movie showing
   - Screenings → Movies: Each showing is for a specific movie

Output of the query:

user1_name	user1_email	user2_name	user2_email	movie_name
Guru Sai Kiran Kammara	kgskiran2004@gmail.com	Dhruv Aggarwal	dhruvagg.96@gmail.com	Avatar
Guru Sai Kiran Kammara	kgskiran2004@gmail.com	Dhruv Aggarwal	dhruvagg.96@gmail.com	Fight Club
Nikhil Narula	nikhilnarula05@gmail.com	Pranita Vasudeva Rao	raopranita10@gmail.com	Inception
Sushanth Battu	srichandra197206@gmail.com	Tejonivas	tejonivas.oruganti01@gmail.com	The Godfather
Sushanth Battu	srichandra197206@gmail.com	Tejonivas	tejonivas.oruganti01@gmail.com	The Avengers
Thiruvarul	thiruvarul12345@gmail.com	Viswajit	svv230804@gmail.com	The Avengers
Ujjayan	NULL	Yeswanth Narayana Patnana	dragonyeswanth2049@gmail.com	The Avengers

3 Query 3

Give list of students from Chennai who have watched a live show/Gigs in Bangalore and movie in Delhi, along with the movie and show names and their release dates.

```
SELECT
   u.name AS student_name,
   m.name AS movie_name,
   m.releaseDate AS movie_release_date,
   gg.title AS live_show_title,
   gg.date AS live_show_date
FROM User u
JOIN Ticket t1 ON u.userID = t1.purchasingUser
JOIN Gigs gg ON t1.gigsID = gg.gigsID
JOIN Venue v1 ON gg.venueID = v1.venueID
JOIN Seat s ON u.userID = s.reservingUser
JOIN Shows sh ON s.showID = sh.showID
JOIN PlayedAt pa ON sh.showID = pa.showID
JOIN Venue v2 ON pa.venueID = v2.venueID
JOIN Movie m ON sh.movieID = m.movieID
WHERE u.city = 'Chennai'
AND v1.city = 'Bangalore'
AND v2.city = 'Delhi';
```

Breakdown of the query:

1. From the `User` table, select users who live in Chennai.
2. Check if they have:
   - Purchased a `Ticket` for a `Gig` (live show) in Bangalore.
   - Reserved a `Seat` for a `Movie` that played at a venue in Delhi.
3. For such users, retrieve:
   - Their name (`u.name`)
   - Movie name and release date (`m.name`, `m.releaseDate`)
   - Gigs title and date (`gg.title`, `gg.date`)

Output of the query:

student_name	movie_name	movie_release_date	live_show_title	live_show_date
Gokulakrishnan R	Avatar	2009-12-18	Tech Conference	2024-04-10

4 Query 4

Give list of collaborators who have directed some movies released after 2005 along with movie name and release date who have also performed in some liveshow in Maharashtra.

```
SELECT
  DISTINCT c.name AS collaborator_name,
  m.name AS movie_name,
  m.releaseDate
FROM Collaborator c
JOIN Featuring f ON c.colID = f.colID
JOIN Movie m ON f.movieID = m.movieID
JOIN PerformsIn p ON c.colID = p.colID
JOIN Gigs gg ON p.gigsID = gg.gigsID
JOIN Venue v ON gg.venueID = v.venueID
WHERE
  f.role = 'Director'
  AND YEAR(m.releaseDate) > 2005
  AND v.state = 'Maharashtra';
```

Breakdown of the query:

1. From the `Collaborator` table, select collaborators who:
   - Are listed in the `Featuring` table as a Director.
   - Directed a `Movie` released after 2005.
   - Also performed in a `Gig` (live show) held in the state of Maharashtra.
2. For each such collaborator, retrieve:
   - Their name (`c.name`)
   - The movie name and its release date (`m.name`, `m.releaseDate`)
3. Use `DISTINCT` to remove any duplicate entries.

Output of the query:

collaborator_name	movie_name	releaseDate
Aarav	Avengers: Endgame	2019-04-26
Aarav	Gully Boy	2019-02-14
Arnav	The Dark Knight	2008-07-18
Vivaan	Interstellar	2014-11-07
Vivaan	Golmaal: Fun Unlimited	2006-07-14
Ansh	The Wolf of Wall Street	2013-12-25

Note: The Next two queries will be containing aggreate functions and group by clauses.

5 Query 5

Count the number of seats bought, movies rented and movies in wishlist for movies produced by each studio. Also arrange them in descending order of sum for above three counts.

```
SELECT
  m.studio,
  COUNT(DISTINCT s.seatID) AS total_seats_booked,
  COUNT(DISTINCT r.movieID) AS total_movies_rented,
  COUNT(DISTINCT w.movieID) AS
     total_movies_in_wishlist
FROM Movie m
LEFT JOIN Shows sh ON m.movieID = sh.movieID
LEFT JOIN Seat s ON sh.showID = s.showID
LEFT JOIN Rents r ON m.movieID = r.movieID
LEFT JOIN Wishlist w ON m.movieID = w.movieID
GROUP BY m.studio
ORDER BY total_seats_booked + total_movies_rented +
total_movies_in_wishlist DESC;
```

Breakdown of the query:

1. For every `Movie`, get its `Studio` name.
2. Count the number of:
   - Seats booked for its shows (total_seats_booked from `Seat` table).
   - Times the movie was rented (total_movies_rented from `Rents` table).
   - Times the movie was added to wishlists (total_movies_in_wishlist from `Wishlist` table).
3. Use `LEFT JOINs` to ensure studios are included even if some counts are zero.
4. Group the results by the studio name.
5. Sort the studios by the total of the three counts (engagement score) in descending order:

   Engagement Score = Seats Booked+Movies Rented+Wishlist Adds

Output of the query:

studio	total_seats_booked	total_movies_rented	total_movies_in_wishlist
Warner Bros.	36	8	10
Paramount Pictures	27	8	8
20th Century Fox	18	6	6
Universal Pictures	13	5	5
New Line Cinema	9	3	3
Orion Pictures	12	1	2
Marvel Studios	12	1	1
Walt Disney Pictures	12	1	1
Columbia Pictures	9	1	1

6 Query 6

Give for all cities the number of venues and number of movies and liveshows being performed in the city, along with average movie rating in that cities' movies having this average rating more than 2.5 in descending order of rating.

```
CREATE VIEW MovieStats AS
  SELECT v.city, AVG(m.userRating) AS
     average_movie_rating
  FROM Venue v
  JOIN PlayedAt pa ON v.venueID = pa.venueID
  JOIN Shows sh ON pa.showID = sh.showID
  JOIN Movie m ON sh.movieID = m.movieID
  GROUP BY v.city;

CREATE VIEW LiveShowStats AS
  SELECT v.city
  FROM Venue v
  JOIN Gigs gg ON v.venueID = gg.venueID
  GROUP BY v.city;

SELECT v.city,
    COUNT(DISTINCT v.venueID) AS total_venues,
    COUNT(DISTINCT m.movieID) AS total_movies,
    COUNT(DISTINCT gg.gigsID) AS total_liveshows,
    ms.average_movie_rating
FROM Venue v
LEFT JOIN MovieStats ms ON v.city = ms.city
LEFT JOIN LiveShowStats lss ON v.city = lss.city
LEFT JOIN PlayedAt pa ON v.venueID = pa.venueID
LEFT JOIN Movie m ON pa.movieID = m.movieID
LEFT JOIN Gigs gg ON v.venueID = gg.venueID
GROUP BY v.city, ms.average_movie_rating
HAVING ms.average_movie_rating > 2.5
ORDER BY ms.average_movie_rating DESC;
```

Breakdown of the query:

1. Computes the average user rating of movies played at each city's venues.
2. Joins `Venue`, `PlayedAt`, `Shows`, and `Movie`.
3. Lists cities that hosted at least one live show (gig).

4. Joins `Venue` and `Gigs`.
5. For each city (from the `Venue` table), it computes:
    - Total number of unique venues
    - Total number of unique movies shown
    - Total number of unique live gigs held
    - Average movie rating
6. Uses `LEFT JOIN` with views to retain all cities, even if they have missing values.
7. Filters cities where the average movie rating is greater than 2.5.
8. Sorts results in descending order of average rating.

Output of the query:

city	total_venues	total_movies	total_liveshows	average_movie_rating
Hyderabad	5	6	3	3.96364
Mumbai	5	6	3	3.80909
Kolkata	5	6	3	3.75000
Delhi	5	7	3	3.55385
Ahmedabad	5	8	3	3.46667
Chennai	5	7	3	3.30000
Pune	5	6	3	3.24545
Bangalore	5	6	3	3.15000
Jaipur	5	4	3	3.03750
Lucknow	5	4	3	2.95714