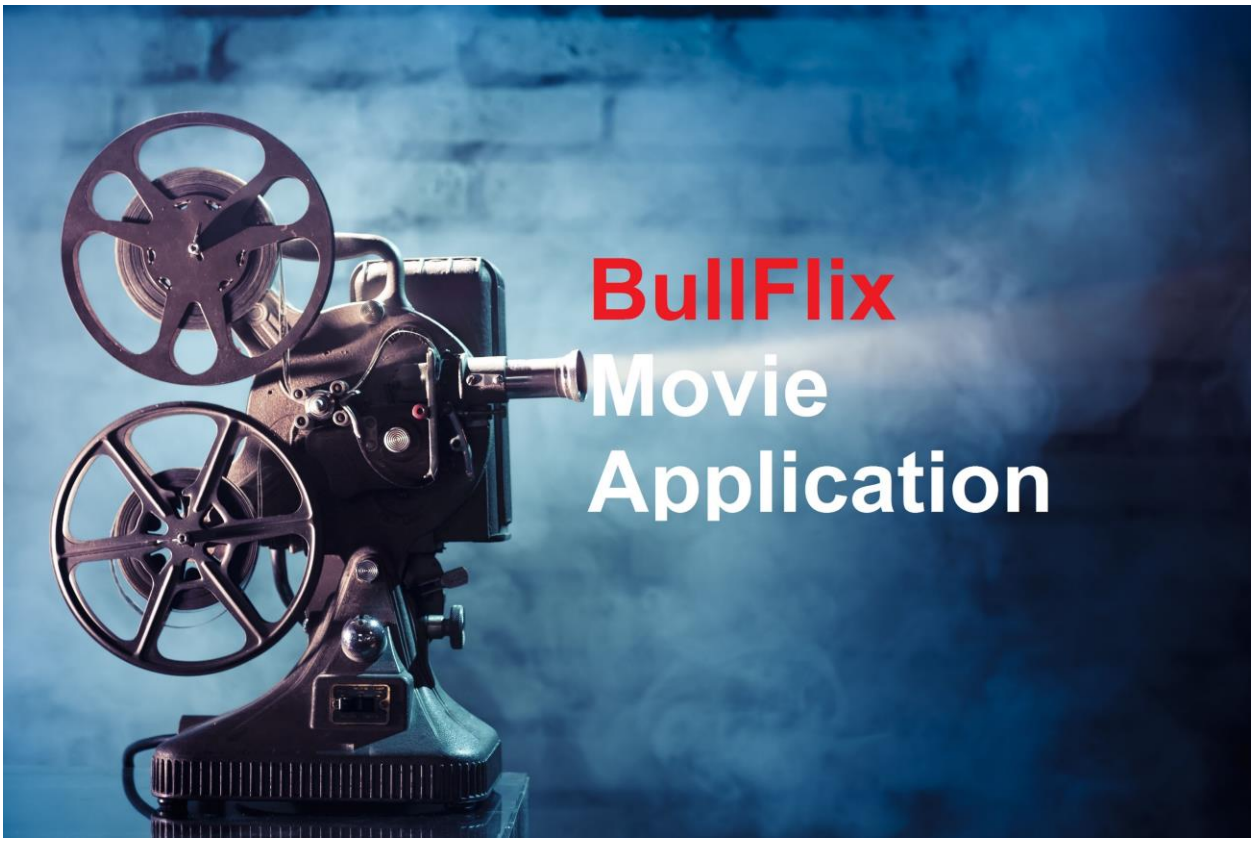
ISM 6124 – Advanced Database Management
(Fall Semester 2017)

Final Group Project

BullFlix Movie Application



By Group 8

Project Members
Rohit Sharma
Samil Shah
Jugal Mer

Table of Contents

1. Executive Summary

Through this report we will be developing database for movie application named ‘BullFlix’. This application is built around the main three parameters like movie ratings, critiques and recommendations. We have extended USF movie application database design and added the functionalities like number of Like tracking, fan rating, creating fixed and random genre list, providing final movie rating based on average rating of fan and critique. Fan authentication will be implemented as important features. The aim of the database is to lend support to online application which allows user to like, rate movies also to see favorite genre list. All the users can search the movie, like the movie, rate the movie, rate the theaters. Users can see the information related to movies, directors, producers, genre, franchises and awards.

Also, it also provides theater related information, genre list through email notification also using database package. BullFlix system has been developed using Oracle database and the same can be used as production systems.

2. Requirements

2.1 Critical Assumptions

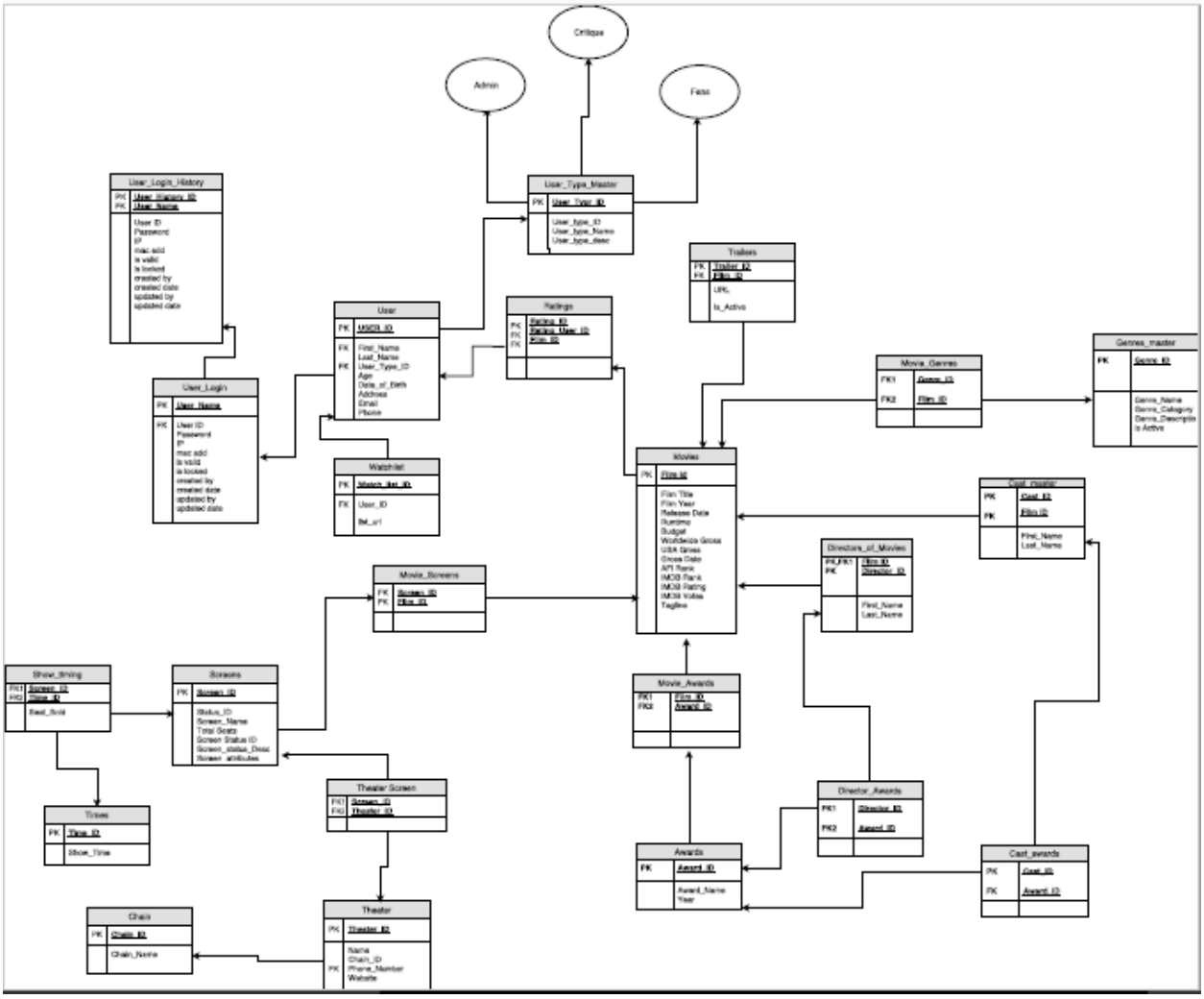
- The assumptions made while designing the system are as follows:
- While rating a movie, not many fans or critique will be rating movie at the same time. It may affect its overall rating.
- As the design does not need distribution of data across various node in different cluster, the database is maintained at one location. So, there is no need of distributed database architecture.
- There is no need to replicate the data across distributed system.
- IF there is need of scalability it can also execute the distributed queries like Analyst requiring data of users of the system like Fans and Critique to send them promotional messages and other service related messages.
- Database servers are running Oracle DBMS.
- To establish distributed database there will be more than one clusters located across different location.
- Multiplex may have more than one theaters inside the auditorium.

- A movie may be screened at different time also simultaneously in multiple auditoriums in multiplex.
- Fan or critique can manage genre list created at any point in time.
- User type id is determined by the admin and must be a part of user master for searching operations.
- Recommendation email is sent to the users only when they agree on the email service at the time of opening the account
- Email notification are sent to the user basis selected time cycle.
- Generation of email notification is generated by the package in database rather than stored procedure in SQL scripting.
- Specific Assumptions
- The number of processor cores used is 16 to smoothly run the operations.
- Cloud resident database will reduce the efforts to manage and administrator the hardware, operating systems, installing and upgrading activities.
- Periodically backup is taken to recover the system in unusual situation like crash
- To provide the stable and performative system accessing wrapper consisting materialized view will be useful. It will also help to make changes in the database without changing the program.
- Target audience for the current system is USF associated people.
- Model View controller has been used to develop the supporting application.

3. Logical Design

The database is divided into functional areas. These functional areas include Movies, Users, Theaters and Awards. The Complete ERD is displayed below. These areas are discussed further in the database decomposition section.

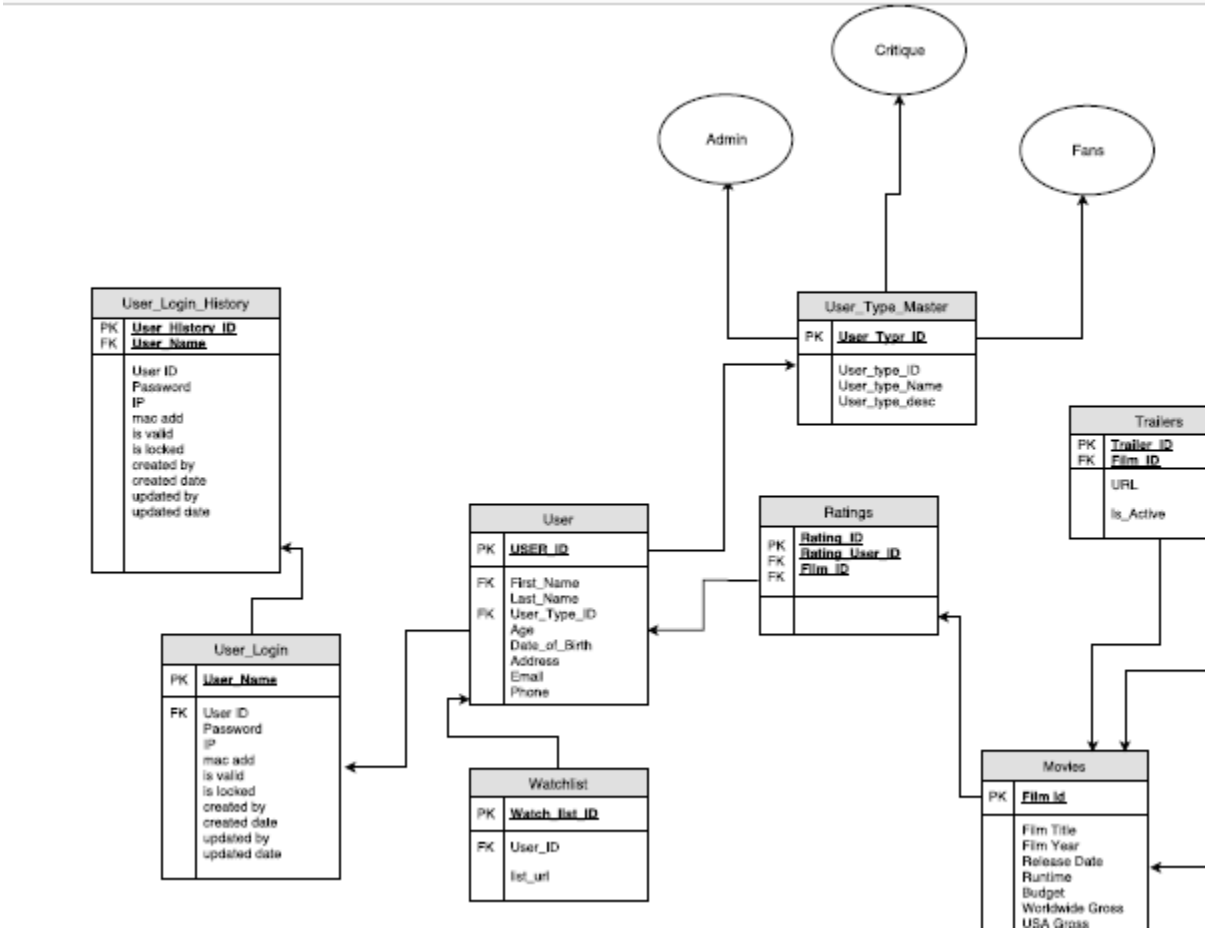
3.1 Entity Relationship Diagram for BullFlix Movie Database



3.2 Functional Decomposition

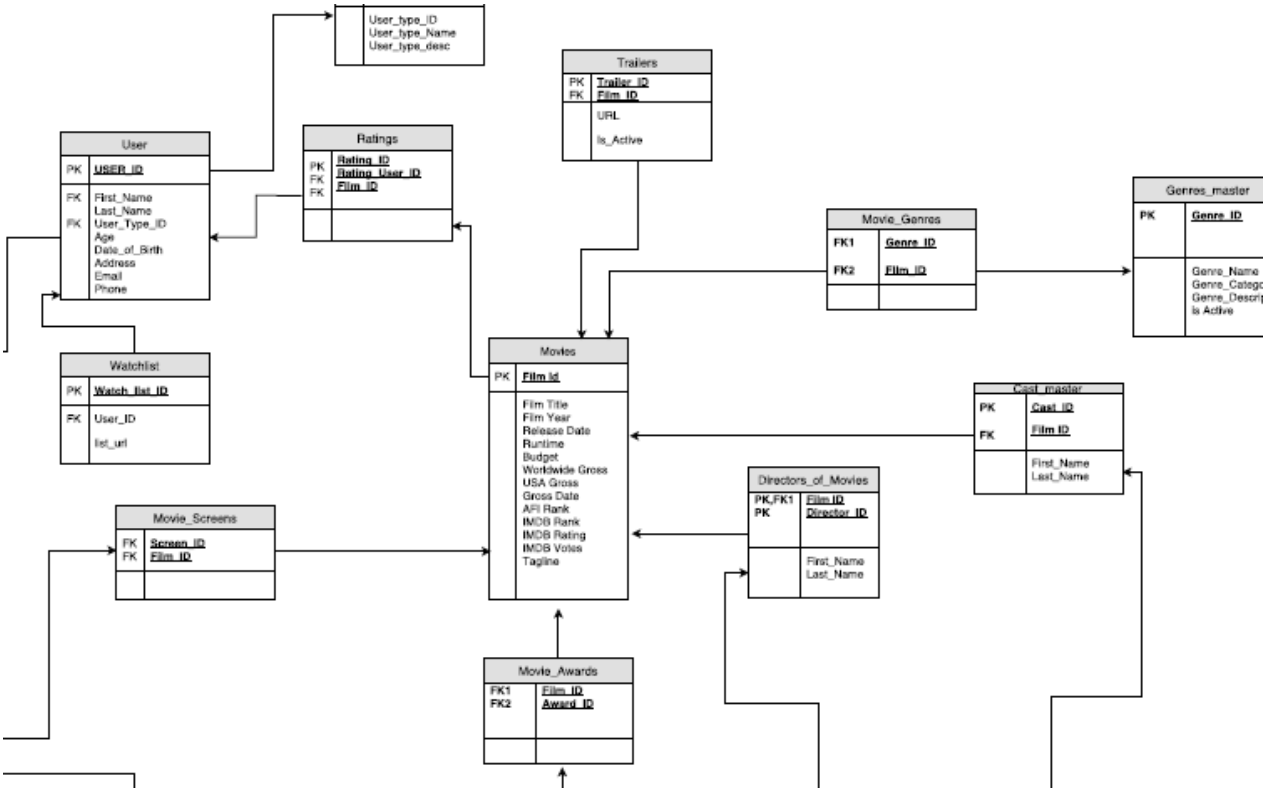
Users

The first of four functional subsections of the database is that of the Users. It is assumed that all users see movies and give movie ratings. User has got user login access where it has user id, password, address and has got a user history of when he or she has rated the movie which that person has seen. There is a rating table where all the movie ratings are stored associated with the user. There is a user type which could be either fan or critic or an admin. User has got watchlist which the user would like to see the movie again depending on the attributes of the movie. User rates movie from user mapping of the film id and the user role played in that circumstances.



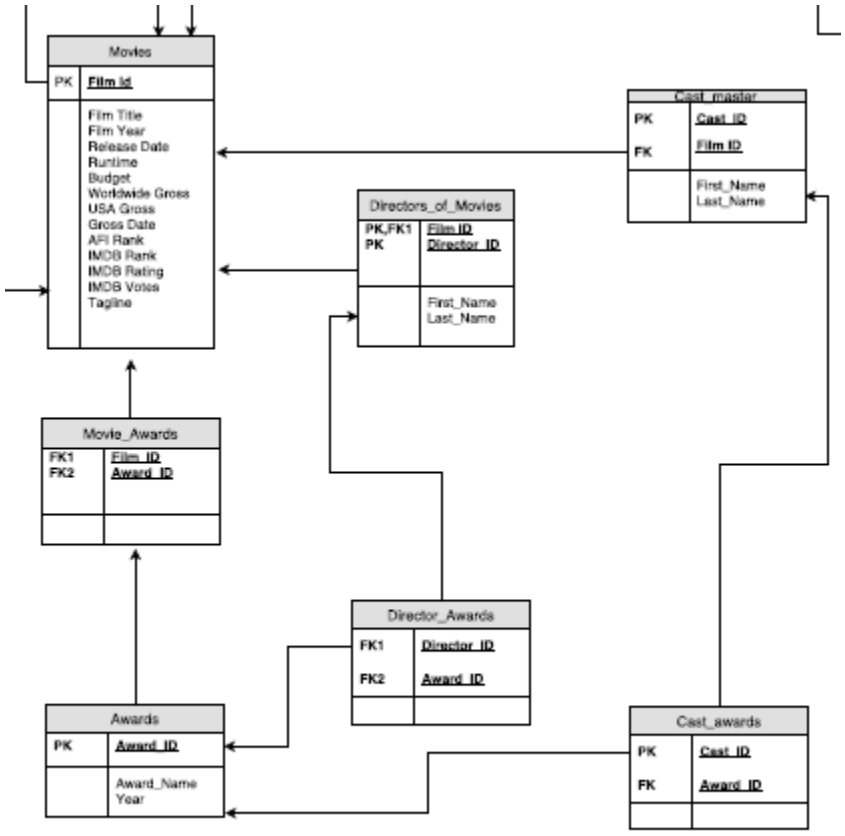
Movies

The second functional subsection of the database is for Movies. The user mapping from the user consisting of film id is connected to the movie table. The movie contains parameters such as the film id, year of release, budget, gross, IMDB rank and few more. All of these play a crucial role in rating of the movie. With all the movies there are associated few entities such as cast members, director, genre and the awards achieved by the film. Each of these entities have got its unique ids for the associations.



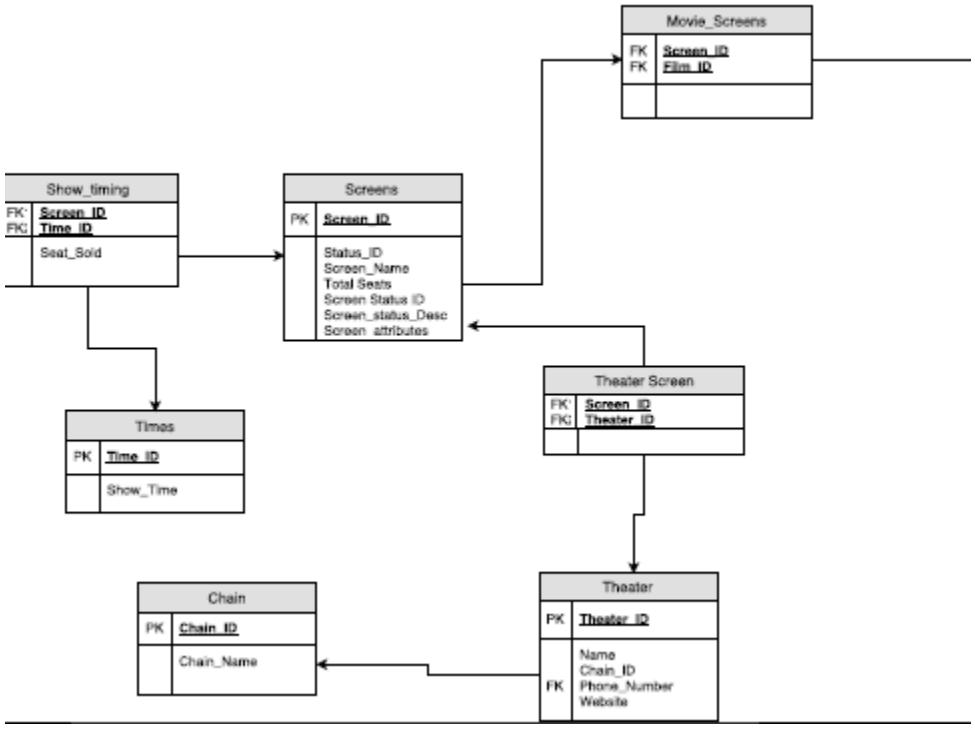
Awards

The third functional subsection of the database is for the awards received by the movie. Depending on the movie entities such as the director, movie casts and the awards received by them based on their successful achievements counts to the awards associated to the movie. Each award won by directors, casts or the movie over all add up the award count of the particular movie.



Theatres

The last functional subsection of the database is for the theatres where the movie runs. Movies run on many screens and each screen runs many movies. The movies associated with the screen depends on the movie timings, screen status whether it is free or not, the demand or the success of the movie and the theatres location. There ae lots of theatres at particular location and the movie could be running simultaneously at different on same location at same time. Theatres have got its name, contact number, website and the chain id if the theatre have got its chain or a brand value, making the same theatre available at many locations with the same name.



3.3 Data dictionaries

Table: User_Login_History; 100 rows loaded

Columns:

PK/FK	Name	Data Type	MaxLength (Bytes)	Required
PK	UserHistory_id	NUMBER	22	Y
FK	User_Name	VHARCHAR	50	Y
	IP	VARCHAR	20	Y
	Mac_address	VARCHAR2	20	Y
	Is_Valid	VARCHAR2	20	Y
	Is_locked	BOOLEAN	8	Y

Indexes:

PK/FK	Name	Column	Type

PK	UserHistory_PK	UserHistory _id	Unique

Used By: User_History

Table: Trailers; 100 rows loaded

Columns:

PK/FK	Name	Data Type	MaxLength (Bytes)	Required
PK	Trailer_id	NUMBER	22	Y
FK	Film_id	NUMBER	22	Y
	URL	VARCHAR2	20	Y
	Is_Active	VARCHAR2	15	Y

Indexes:

PK/FK	Name	Column	Type

PK	Trailer_PK	Trailer_id	Unique
FK	Film_FK	Film_id	Unique

Used By: User_Login

Table: User_Login; 100 rows loaded

Columns:

PK/FI	Name	Data Type	Max Length (Bytes)	Required
PK	User_Name	VARCHAR2	20	Y
FK	User_id	NUMBER	22	Y
FK	User_History_id	NUMBER	22	Y
	Password	VARCHAR2	22	Y
	IP	VARCHAR	20	Y

	Mac_address	VARCHAR2	20	Y
	Is_Valid	VARCHAR2	20	Y
	Is_locked	BOOLEAN	8	Y

Indexes:

PK/FK	Name	Column	Type
PK	User_Name_PK	User_Name	Unique
FK	User_FK	User_id	Unique
FK	User_History_FK	User_History	Unique

Used By: User_login, User_History, User

Table: User; 100 rows loaded

Columns:

PK/FI	Name	Data Type	Max Length (Bytes)	Required
PK	User_id	NUMBER	22	Y
FK	FirstName	VARCHAR2	20	Y
	LastName	VARCHAR2	20	Y
FK	User_Type_id	NUMBER	22	Y
	Age	NUMBER	22	Y
	DateOfBirth	VARCHAR2	20	Y
	Address	VARCHAR2	50	Y
	Email	VARCHAR2	20	Y

Indexes:

PK/FK	Name	Column	Type

PK	User_PK	User_id	Unique
FK	FirstName_FK	FirstName	Unique
FK	User_Type_FK	User_Type_id	Unique

Used By: User, User_login, Ratings, User_Type_Master

Table: Genres; 100 rows loaded

Columns:

PK/FK	Name	Data Type	Max Length (Bytes)	Required
PK	Genres_id	NUMBER	22	Y
	Genres_Name	VARCHAR2	20	Y
	Genres_Category	VARCHAR2	20	Y
	Genres_Description	VARCHAR2	50	Y

Indexes:

PK/FK	Name	Column	Type
PK	Genres_PK	Genres_id	Unique

Used By: Genres

Table: Watchlist; 100 rows loaded

Columns:

PK/FK	Name	Data Type	Max Length (Bytes)	Required
PK	Watchlist_id	NUMBER	22	Y
FK	User_id	NUMBER	22	Y
	List_URL	VARCHAR2	20	Y

Indexes:

PK/FK	Name	Column	Type
PK	Watchlist_PK	Watchlist	Unique
FK	User_PK	User _id	Unique

Used By: Watchlist, User

Table: Film; 100 rows loaded

Columns:

PK/F	Name	Data Type	Max Length (Bytes)	Required
PK	Film_id	NUMBER	22	Y
	FilmTitle	VARCHAR2	20	Y
	FlimYear	NUMBER	22	Y

	ReleaseDate	VARCHAR2	20	Y
	Runtime	NUMBER	22	Y
	Budget	NUMBER	22	Y
	WorldwideGross	NUMBER	22	Y
	USAgross	NUMBER	22	Y
	GrossDate	VARCHAR2	20	Y
	AFIRank	NUMBER	22	Y
	IMDBrank	NUMBER	22	Y
	IMDBrating	NUMBER	22	Y
	IMDBvotes	NUMBER	22	Y
	Tagline	VARCHAR2	20	Y

Indexes:

PK/FK	Name	Column	Type

PK	Film_PK	Film_id	Unique

Used By: Film

Table: Show_Timing; 100 rows loaded

Columns:

PK/FK	Name	Data Type	Max Length (Bytes)	Required
PK	Time_id	NUMBER	22	Y
FK	Screen_id	NUMBER	22	Y
	Seats_Sold	NUMBER	50	Y

Indexes:

PK/FK	Name	Column	Type
PK	Screen_PK	Screen _id	Unique

FK	Time_FK	Time_id	Unique

Used By: Show_Timing, Times, Theatre_Screen

Table: Times; 100 rows loaded

Columns:

PK/FK	Name	Data Type	Max Length (Bytes)	Required
PK	Time_id	NUMBER	22	Y
	Show_Time	VARCHAR2	20	Y

Indexes:

PK/FK	Name	Column	Type
PK	Time_PK	Time_id	Unique

Used By: Times

Table: Screens; 100 rows loaded

Columns:

PK/FI	Name	Data Type	Max Length (Bytes)	Required
PK	Screen_id	NUMBER	22	Y
	Status_id	NUMBER	22	Y
	Screen_Name	VARCHAR2	20	Y
	Total_Seats	NUMBER	22	Y

Indexes:

PK/FK	Name	Column	Type
PK	Screen_PK	Screen_id	Unique

Used By: Screens

Table: Directors; 100 rows loaded

Columns:

PK/FI	Name	Data Type	Max Length (Bytes)	Required
PK	Director_id	NUMBER	22	Y
	FirstName	VARCHAR2	20	Y
	LastName	VARCHAR2	20	Y

Indexes:

PK/FK	Name	Column	Type
PK	Director_PK	Director_id	Unique

Used By: Directors

Table: Cast; 100 rows loaded

Columns:

PK/FK	Name	Data Type	Max Length (Bytes)	Required
PK	Cast_id	NUMBER	22	Y
	FirstName	VARCHAR2	20	Y
	LastName	VARCHAR2	20	Y

Indexes:

PK/FK	Name	Column	Type
PK	Cast_PK	Cast_id	Unique

Used By: Cast

Table: Awards; 100 rows loaded

Columns:

PK/FK	Name	Data Type	Max Length (Bytes)	Required
FK1	Award_id	NUMBER	22	Y
	Award_Name	VARCHAR2	20	Y
	Year	NUMBER	22	Y

Indexes:

PK/FK	Name	Column	Type
PK	Award_PK	Award_id	Unique

Used By: Awards, Movie_Awards, Director_Awards, Cast_Awards

Table: Chains; 100 rows loaded

Columns:

PK/FI	Name	Data Type	Max Length (Bytes)	Requir ed
PK	Chain_id	NUMBER	22	Y
	Chain_Name	VARCHAR2	20	Y

Indexes:

PK/FK	Name	Column	Type
PK	Chain_PK	Chain_id	Unique

Used By: Chain

Table: Theatre; 100 rows loaded

Columns:

PK/FI	Name	Data Type	Max Length (Bytes)	Required
PK	Theatre_id	NUMBER	22	Y
	Name	VARCHAR2	20	Y
	Chain_id	NUMBER	22	Y
FK	Phone_Number	NUMBER	22	Y
	Website	VARCHAR2	20	Y

Indexes:

PK/FK	Name	Column	Type
PK	Theatre_PK	Theatre_id	Unique
FK	Phone_Number_FK	Phone_Number	Unique

Used By: Theatre, Theatre_Screen

4. Physical Design

The Movie Database is linked to the Login website and will allow users to create an account, rate the movie and can also look for show times , awards to cast and director and for the movie. The database will power the front end in a way that is conducive to searches and filters for buyers looking for specific items. Images will be stored separately in BLOB or filesystem storage. The database can be useful for generating revenue reports for stakeholders, as well as highlighting frequent sale types to consider future site improvements to encourage further growth in the sector.

5. Data generation and loading

5.1 Test Data Generation and loading

After the initial creation of the database, the DbForge Data Generator for Oracle was used connected and used to generate sample data. Based on the constraints and metadata built into the database, the generator simulates types of data that could be possible in the database. We decided to only generate data for some tables based on anticipated needs for performance tuning: Users, Awards, Theatres, Cast, Director using simple text files, we were able to feed certain types of data to limit an attribute to. For example, we generated a text file listing the rating given by a user to a movie. This helped us later query these tables and run performance experiments. For the varchar data type attributes lorem ipsum was used to generate text based on the data constraint. One issue we found with this free tool was that if we had a constraint specified for an attribute or multiple attributes, we were not also allowed to set constraints within the program.

Overall, while this data proved to be marginally useful for generating data on our budget (free), we realize that in generating data to test a real database we would develop, a paid tool would have made the process smoother and more reliable. Another issue we found was that the data loading itself was very time consuming. To load our data, it took about long hours. It was also very difficult to know exactly what tests we would want to conduct on which tables from the beginning of the project. We were somewhat limited by the tables we could run tests on and would have benefited from loading additional data into more tables.

We created small database with 500 records each table for development purpose. Also, we took reference data from RELMDB schema provided.

We have used following data count for records based on the data provided in schema

Theaters: 7719
Movies: 283
Fans: 7655
Genre: 127
Director: 52
Cast: 604

Rest of the data is populated with standard 500 records.

Additional data has been added to some tables using stored procedures.

Populating user's data
Insert procedure has been written to populate the user table with the corresponding data. At start, User_ID, FirstName, LastName, User_Type_ID, Rating_User_ID, Age, DOB, Address, Email. As shown below it accepts the various parameter for required values and eases the insert data

operation into database. User_ID is set to be incremental which will increase with every new added in the table.

Script:

```
CREATE PROC InsertUser
@UserID numeric(100),
@FirstName varchar(4,0),
@LastName varchar(10,0),
@Age numeric(12,0),
@DOB Date,
@Address varchar(10,2)
AS
BEGIN
INSERT INTO USER (USERID, FIRSTNAME, LASTNAME, AGE,
DOB, ADDRESS)
VALUES (@UserID, @FirstName, @LastName, @Age,
@DOB, @Address)
END;
```

Populating data into Cast/Actor table

This inserts data of new cast to the records. It simplifies the insertion of the record into database

Script:

```
CREATE PROC NewCast
@Actor_Name varchar(50),
@Film_Id numeric(10,0),
@Role varchar(100)
AS
BEGIN
INSERT INTO CASTS (FILM_ID, ACTOR_NAME, ROLE)
VALUES (@Film_Id, @Actor_Name, @Role)
END;
```

Populating data into Director table

This procedure will add the new directors data, Name Film_ID to the director's table.
It takes two parameters as input and inserts the record easily into the director table.

Script:

```
CREATE PROC NewDirector
@Film_Id numeric(10,0),
@Director_Name varchar(50)
AS
```

```
BEGIN
INSERT INTO DIRECTORS (FILM_ID, DIRECTOR_NAME)
VALUES (@Film_Id, @Director_Name)
END
```

Populating data into Theater table

This procedure adds the theater details like name, location, address.

```
CREATE PROC NewTheater
@Film_Id numeric(10,0),
@Director_Name varchar(50)
AS
BEGIN
INSERT INTO DIRECTORS (FILM_ID, DIRECTOR_NAME)
VALUES (@Film_Id, @Director)
END
```

6. Architectural Issues

Given these initial calculations, a NoSQL option would not be necessary, and this data is well suited for a RDBM system. The data is well-structured, which favours RDBMS over NoSQL; the schemas are a good fit and allow for consistent data. Additionally, seeing as the data is spread out over multiple tables to achieve 3rd normal form, joins will be an important feature.

7. Indexing Strategies

All primary keys are by default indexed in Oracle. We created a few in the Performance tuning section of this report where we determined three of the four were actually very beneficial. There were still several more that we suspected might improve performance of the database but due to the limited data that was loaded it was difficult to actually run any experiments.

8. Performance tuning

8.1 Usage

Movie Fan/Critique

- To search movies, entire cast, genre, rating, theater
- To get information like all the movie director has directed or actor has acted in
- To get information on overall rating of the movie. (Fan and critique rating)
- To get the information on theater rating and to rate the theater visited
- To get the information on the awards won by an actor and director
- To fetch information about the movie show timings at given location

Database management

- To recommend genre list based on the search history of the user
- To notify user via email or sms about the latest list posted
- To display the report on top 10 highest gross movie of the year
- To recommend the movies based on the fan ratings
- To fetch the information on fan and critique user in state and country. So, that this can be used to send the promotional information

Most queried table in the database is the Movies table, so it would be wise idea to implement indexing over a field in table to increase the performance of the system

8.2 Indexing

1. Film_ID

We have to tried to analyze the importance of creating indexing over Film_ID while joining Movies and User table through this approach. Having index on Film_ID helped in accessing movie information faster and allowed fast fetching of queries for analysis like
Does top gross movie always spends the huge money on a project and which region is responsible for contributing the large amount to movie's business.

To test this approach we have first run a query showing number of movies watched by user.

Query without Indexing:
If we run query without creating index over Film_ID it results in cost of '25'.

SELECT COUNT(FILM_ID),

```
USER_ID
FROM RELMDB.MOVIES
NATURAL JOIN RELMDB.USERS
GROUP BY USER_ID
ORDER BY COUNT(FILM_ID);
```

OutPut:

S.NO	Predicate	With Index			Without Index		
		Consistent Reads	Physical Reads	Cost	Consistent Reads	Physical Reads	Cost
1	MOVIES.FILMID = USERS.FILM_ID	50	0	25	200	0	18

Query with indexing:

Running the same query with indexing will result in cost of '18' which is much less than the cost without indexing. So rather than scanning the whole table oracle was able to get the information from the index quickly.

2. User_Type_ID

In this experiment we added index on User_Type_ID. We thought the addition of index on this field will help search faster through the data about the users like Fans, Critiques and Admin. As fan, critiques are may frequently view the list of genre recommended continuously, running this query using index seemed to be efficient operation. In result it turned out to be more costly operation and our assumption were ruled out. Implementing index actually increased the cost significantly. Without index cost is just 30 and forcing the index shoots cost to 120.

```
SELECT /* INDEX (USERS USER_TYPE_ID)
        USERS.*,
        GENRE.*
FROM
        USERS
        INNER JOIN GENRE
        ON USER.USER_TYPE_ID = GENRE.USER_TYPE_ID

WHERE
        GENRE = 'HORROR'
```

S.NO	Predicate	With Index			Without Index		
		Consistent Reads	Physical Reads	Cost	Consistent Reads	Physical Reads	Cost
1	GENRE.GENRE = 'HORROR'	100	0	120	50	0	30

So, looking at the cost in both the scenarios we can say that putting index every time will not result in performance, it may slow down the operation. This might have happened because there's small number of categories which is not sparse. At any point in time given category will have distributed data through the table resulting in every block needs to be load causing no performance gain. Hence, the performance loss because of cost of loading and scanning.

3. This experiment will show the result of B+ tree on movies table. It would be helpful for users to search movies based on the name string. It's been conducted on relmdb.movies table

Part1: Creating a test table

Create table movie_test as
select *
from relmdb.movies

Part2: Analyzing the execution plan for the query when particular movie is being searched

a. Title Search

Select * from movie_test
where title like '"$Shawshank redemption$"(1980)';

b. Text in title search

Select * from movie_test
where title like '%the%'

Part 3: creating index over title key

Create index index_title on movie_test(title)

Part4:

a. Title search

Select * from movie_test
where title like ‘“$Shawshank redemption$”(1980)’;

b. Text in title search

Select * from movie_test
where title like ‘%the%’

8.3 Bitmap Indexes

Let’s say if the expected query outcome is to find all the movies of 2010 whose overall rating is greater than 7. The second query would be to find the movies in specific genre. So, we will be using AND/OR operations in the likes of the queries. In such scenarios Bitmap indexes are very helpful. This experiment demonstrates the use of bitmap indexes on queries having logical operators.

The sample used for this operation is movie table relmdb table.

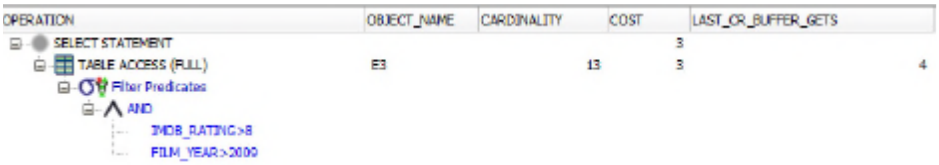
Part1: Creation of test table

Create table movie_test1 as
select *
from relmdb.movies

Part2: Query to identify movies having overall rating of greater than 7 in the year 2009

Select * from movie_test1
where Rating>7 and Film_year >2010

Execution plan:



Observations:

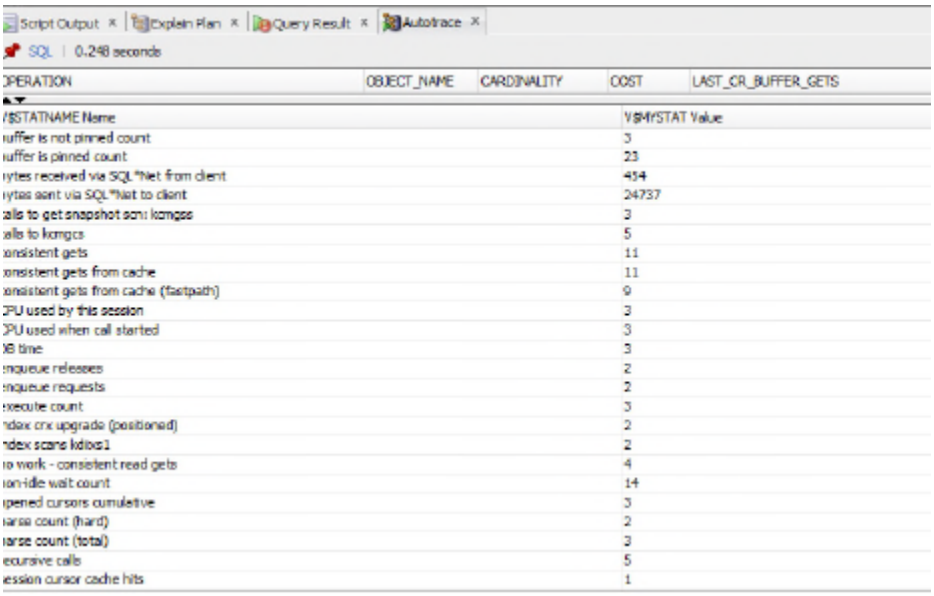
V$STATNAME.Name	V$MYSTAT.Value
buffer is not pinned count	25
bytes received via SQL*Net from client	454
bytes sent via SQL*Net to client	24738
calls to get snapshot scn: kcmgas	12
calls to kcmgcs	6
cluster key scan block gets	3
cluster key scans	3
consistent gets	40
consistent gets - examination	13
consistent gets from cache	40
consistent gets from cache (fastpath)	26
CPU used by this session	4
CPU used when call started	4
DB time	4
enqueue releases	2
enqueue requests	2
execute count	12
index fetch by key	4
index scans kdiixs1	6
no work - consistent read gets	21
non-idle wait count	14
opened cursors cumulative	12
parse count (hard)	2
parse count (total)	11
parse time cpu	2
parse time elapsed	2
recursive calls	205
recursive cpu usage	1

Part4: Now creating bitmap indexes on Rating and Film_year to check the performance

CREATE BITMAP INDEX bitmap_rating
ON movie_test1 (Rating)
CREATE BITMAP INDEX bitmap_film_year
ON movie_test1 (Film_Year)

Part5: Executing the query after indexing

Select * from movie_test1
where Rating>8 and Film_year >2009



So, from observing above values we can say that bitmap indexes are definitely increasing the performance for the query.

8.4 Strategy used for the performance tuning

Below is the Proposed plan for the tuning of the database to increase the performance

- Indexing large tables like Movies, Fan, Critique with the help of B+ tree on title, film_id as a search key.
- We will be sending the notification to the users consisting of fixed or dynamic list of genres suggested also company might send the promotional offers to the users residing in different state. We can create Bitmap index on fans and the user master, and genre table which will significantly reduce the query time and enhance the performance. Bitmap indexes will search for the relevant columns in the database.
- Similarly bitmap indexes can also be created for the theater and user master to search for the user rating for the theater.

9. Query writing

We have written below queries that will utilize the database system. These queries will be useful in finding the answers to the questions that will provide insight to the data and will help to derive more analysis using the similar queries. These can be used by the Analyst, Marketing people**.**

Query statement:

1. Displaying top 5 cast members who's movies have highest collection at the box office. It will be very useful in business point of view, while signing actors/actress many production houses look for the brand value of the actors and then finalize the final casts of the movie. Every producer is concerned with the return of the money invested. Picking up the cast members who's movies have done well in past provides the producer security to some extent. Although it wont assure that the movie is going to do well, it always provides the production houses less risk factors as many times people go to watch movies with just the name of actor/actress.

Below query explain how the data corresponding to the cast members and their gross movie collection can be fetched from Casts and Movies table simultaneously.

```
/*
DISPLAY TOP 5 CAST MEMBERS WHOS MOVIES HAVE HIGHEST COLLECTION AT BOX OFFICE.
*/



SELECT CAST_MEMBER,TOTAL_GROSS
FROM(
SELECT
    C.CAST_MEMBER,
    SUM(M.USA_GROSS) + SUM(M.WORLDWIDE_GROSS) AS TOTAL_GROSS
FROM RELMDB.MOVIES M
    INNER JOIN RELMDB.CASTS C
    ON C.FILM_ID = M.FILM_ID
WHERE
    USA_GROSS IS NOT NULL
    AND  WORLDWIDE_GROSS IS NOT NULL
GROUP BY C.CAST_MEMBER
ORDER BY TOTAL_GROSS DESC
)
WHERE  ROWNUM <= 5;
```

2. This query is responsible for fetching the data for all the movies with IMDB rating more than 8 and with more than 100000 IMDB votes also which were released from 2007 to 2013. Showing highest IMDB movie first. This type of queries will be very useful for doing analysis for top rated movies and current likings of the movie goers. It provides different movie analyst with the data through which the they can study different traits of the movie.

```
/* 1. Display the name of all movies that have an IMDB rating of at least 8.0,
with more than 100,000 IMDB votes, and were released from 2007 to 2013.
Show the movies with the highest IMDB ratings first. */

SELECT
FILM_TITLE
FROM RELMDB.MOVIES
WHERE IMDB_RATING >= 8.0
    AND IMDB_VOTES > 100000
    AND FILM_YEAR BETWEEN 2007 AND 2013
ORDER BY IMDB_RATING DESC;
```

3. This query displays how many movies have an MPAA rating of G, PG, PG-13 and R. Also, show the results in alphabetical order by MPAA rating. This can be used to categorize the movie on the basis of their grades and how many PG-13 rating are being made in an year, and how well they are doing in the business.

```
/*
3.Display how many movies have an MPAA rating of G, PG, PG-13, and R.
Show the results in alphabetical order by MPAA rating.
*/

SELECT
MPAA_RATING,
COUNT(*) AS "NO OF MOVIES"
FROM RELMDB.MOVIES
WHERE MPAA_RATING IN ('G','PG','PG-13','R')
GROUP BY MPAA_RATING
ORDER BY MPAA_RATING; --DEFAULT SORTING ORDER IS ASC;
```

4. This query displays the title of any movies where Tom Hanks or Tim Allen were cast members. Each movie title should be shown only once. Using such queries, it ca target on specificity of the results restricted to the specific actors.

```sql
/* 4.
Display the titles of any movies where Tom Hanks or Tim Allen were cast members.
Each movie title should be shown only once. */


SELECT
DISTINCT -- REMOVING DUPLICATES
M.FILM_TITLE
FROM RELMDB.MOVIES M
    JOIN CASTS C -- DEFAULT IS INNER JOIN
    ON M.FILM_ID=C.FILM_ID
WHERE CAST_MEMBER IN ('Tom Hanks','Tim Allen'); -- where Tom Hanks or Tim Allen were cast members.
```

5. This query is about the displaying each movie's title and total gross, where total gross is USA gross and world wide gross combined. Exclude any movies that do not have values for either USA gross or worldwide gross. And showing the highest grossing movie first. This will give an idea to the management about what kind of movies are doing what kind of business, are the content based film are working more or the entertainment based. How well the movies are performing in international market. Also, it gives the sense to what is international audience taste of the movies.

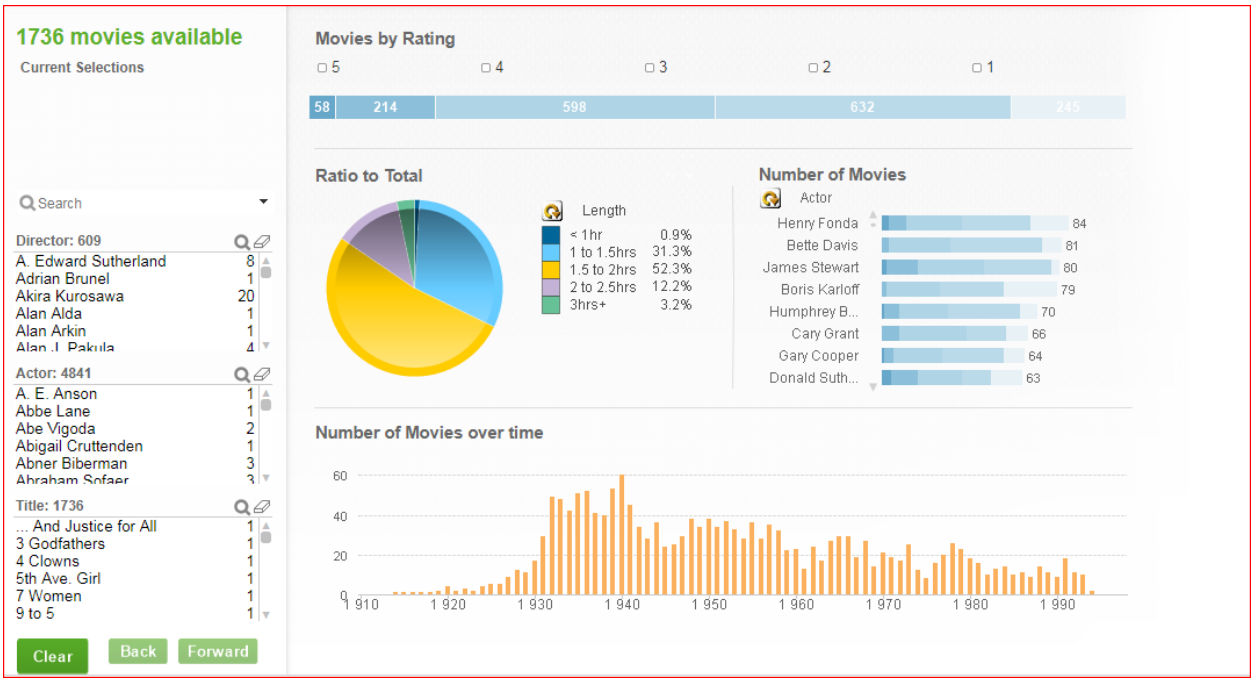
```sql
/* 2.Display each movie's title and total gross, where total gross
is USA gross and worldwide gross combined. Exclude any movies that do not have
values for either USA gross or worldwide gross.  Show the highest grossing movies first.*/

SELECT
FILM_TITLE,
(WORLDWIDE_GROSS+USA_GROSS) AS TOTAL_GROSS
FROM RELMDB.MOVIES
WHERE USA_GROSS IS NOT NULL
    AND WORLDWIDE_GROSS IS NOT NULL
ORDER BY TOTAL_GROSS DESC;
```

6. This query will fetch movie's title, year and how many cast members were part of the movie. Excluding movies with five or less cast members. Also displaying movies with the most cast members first, followed by movie year and title. This type of analysis helps in analyzing the data of whether multi starrer films are working good now a days or movies with less cast members.

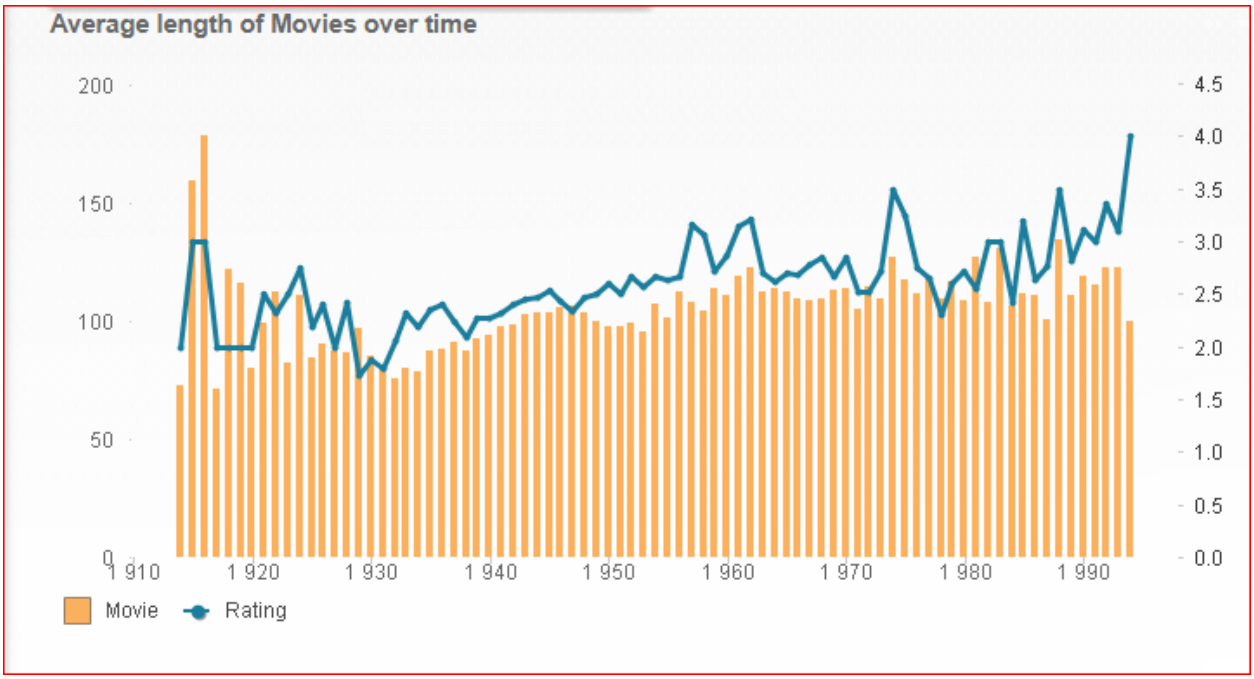
```sql
/*5.
For each movie display its movie title, year,
and how many cast members were a part of the movie.
Exclude movies with five or fewer cast members.
Display movies with the most cast members first, followed by movie year and title.*/


SELECT
FILM_TITLE,
FILM_YEAR,
COUNT(C.FILM_ID) AS CAST_MEMBER_COUNT
FROM RELMDB.MOVIES M
    INNER JOIN CASTS C
    ON M.FILM_ID=C.FILM_ID
GROUP BY FILM_TITLE,FILM_YEAR
HAVING COUNT(C.CAST_MEMBER)>5
ORDER BY COUNT(C.CAST_MEMBER) DESC,M.FILM_YEAR ASC,M.FILM_TITLE ASC;
```

10. Data visualization

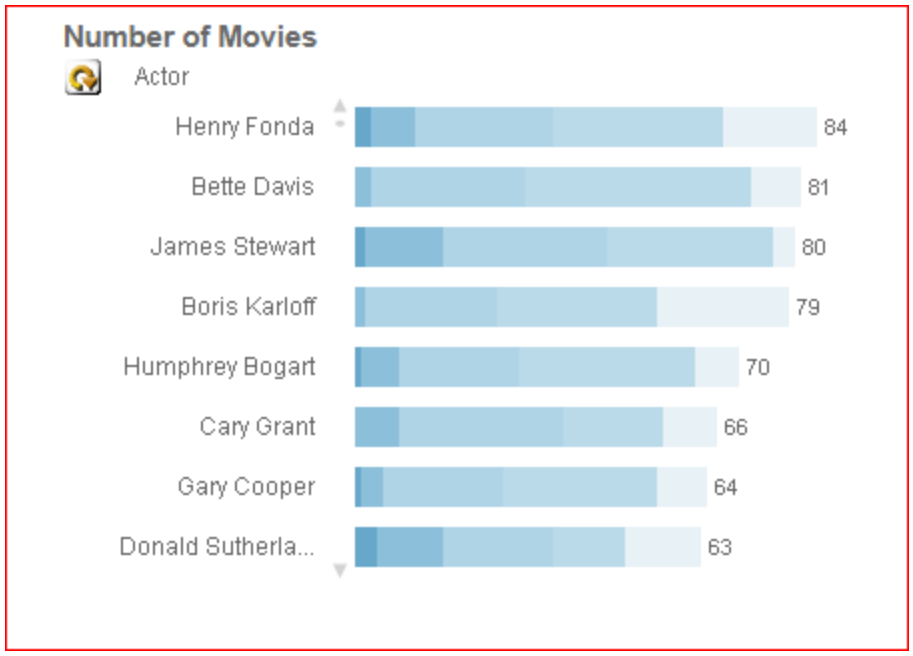
We have done below visualizations in QlikView visualization tool, input data to the tools was provided from sample data created. These are of great help in analyzing, drawing conclusion and developing the recommendation.

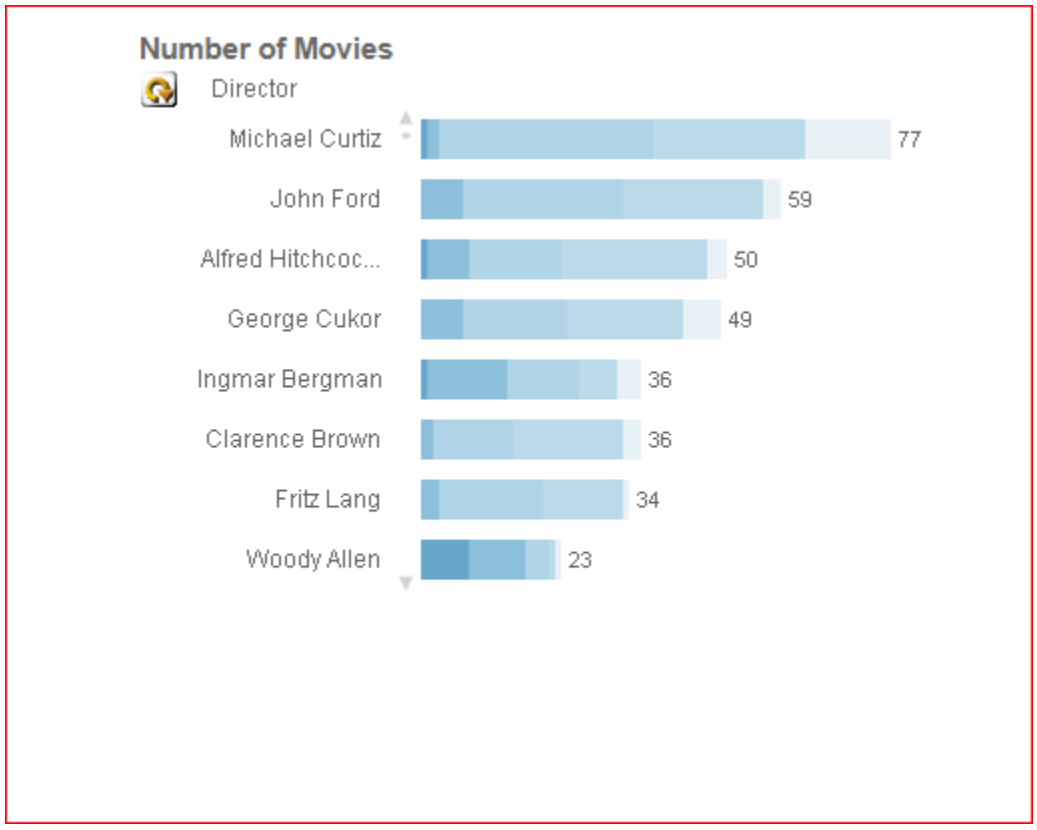


1. This Combo chart (Bar + Line) displays the average length of the movies over a time. Through this chart we can say with increase in the number of years average length of the movie has been increased. Earlier audience were more interested in watching shorter length of the movies which has changes for this decade. Dimension used in this chart is Year movie was released and a expression for calculating average length of movie has been calculated



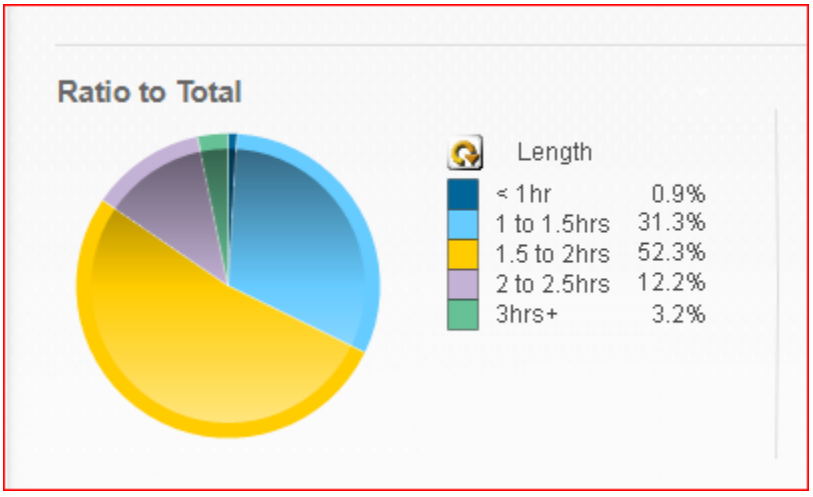
2. Below is the multilevel cyclic drill bar graph which depicts the number of movies done by actors and director. The cyclic group at the top left corner helps to change between multiple dimension such as actor and director. This will be very useful to maintain and analyze the records of director's and actors movies.


Number of Movies
Actor
Henry Fonda 84
Bette Davis 81
James Stewart 80
Boris Karloff 79
Humphrey Bogart 70
Cary Grant 66
Gary Cooper 64
Donald Sutherla... 63

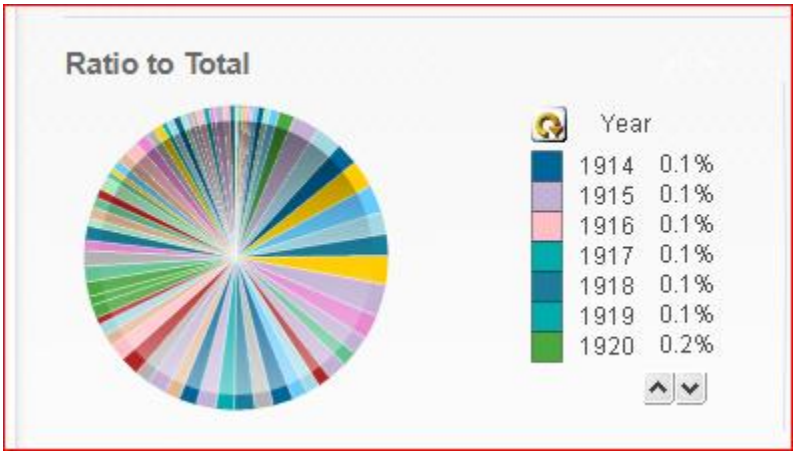

Number of Movies
Director
Michael Curtiz 77
John Ford 59
Alfred Hitchcoc... 50
George Cukor 49
Ingmar Bergman 36
Clarence Brown 36
Fritz Lang 34
Woody Allen 23

3. Below pie charts depicts the contribution percentage length, decade, and year wise.

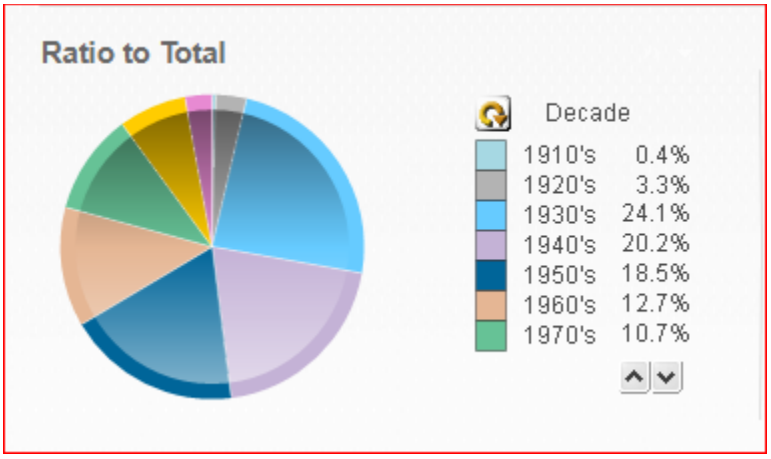
This pie chart depicts that 1.5 to 2 hrs. movie hold the largest among the all length of movie made and it is 52 % in yellow. Next closest duration is 1 to 1.5 hrs.



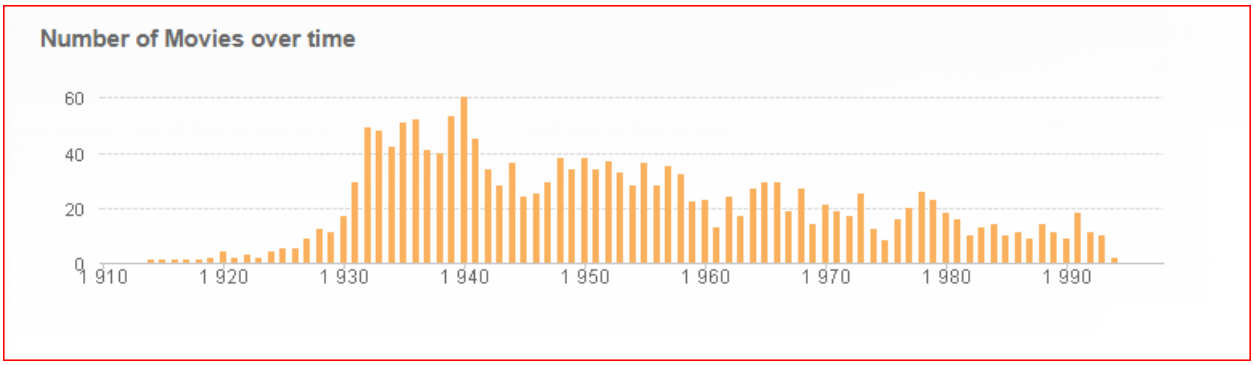
As described earlier in bar chart cyclic group is provided in this chart which will facilitate the changing of dimensions like year, length and decade with just a click. This chart depicts the in which year maximum number of movies were made.



This pie chart is made between the dimension Decade and the number of movies made that decade. As shown in the image 1930's decade is the one in which maximum number of movies were made.



This is the bar chart which depicts the number of movies made over the years. As shown in the image maximum number of movies were made in 1940's.



QlikView helps in self driven analysis rather than fetching records only for specific queries. It is interactive, and any numbers of question can be asked, and analysis be done in real time. In-memory processing makes it unique and results are returned in no time.

11. Stored procedures and Triggers

11.1 Email Procedure

We are writing this procedure to send auto email notifications to users giving them recommendations based on their search history. We would schedule this daily at 5 PM for the users who have subscribed for the notifications. Here we are using UTL_SMTP package to achieve this. it is used for sending electronic mails over Simple mail transfer protocol. IT CONSISTS OF set of commands for an email client to dispatch the email to SMTP server. The various commands include UTL_SMTP functions such as open_connection,helo, rcpt, write_data,close, quit to perform handshake and send the emails

```
CREATE OR REPLACE PROCEDURE PROC_SEND_EMAIL (V_ERROE_CODE   OUT NUMBER
                            ,V_ERROR_MSG   OUT VARCHAR2)SS
AS

V_TO        VARCHAR2(100);
V_FROM      VARCHAR2(100):= 'DB_MAIL_USF.COM';
V_MSG           VARCHAR2(4000);
V_SMTP_HOST     VARCHAR2(100):= '10.0.0.127';
V_SMTP_PORT     NUMBER:= 25;
V_SUB       VARCHAR2(200):= 'RECOMMENDATIONS FROM BULL FLIX';
V_CONN          UTL_SMTP.CONNECTION;


CURSOR CUR_DATA IS
    --write a query to find out top 3 rated, genre specific movie or whatever u want

BEGIN


    V_CON := UTL_SMTP.OPEN_CONNECTION(V_SMTP_HOST,V_SMTP_PORT);

    UTL_SMTP.HELO(V_CONN,V_SMTP_HOST);
    UTL_SMTP.MAIL(V_CONN,V_FROM);

    FOR REC_DATA IN CUR_DATA LOOP
        UTL_SMTP.RCPT(V_CONN,V_TO);
    END LOOP;
```

Worksheet Query Builder

```
        UTL_SMTP.OPEN_DATA(V_CONN);

        UTL_SMTP.WRITE_DATA(V_CONN, 'Date:' || TO_CHAR(SYSDATE,'DD/MM/YYYY HH24:MI:SS')
        FOR REC_DATA IN CUR_DATA LOOP
            UTL_SMTP.WRITE_DATA(V_CONN, ' To:'|| REC_DATA.EMAIL_ID);

            v_msg := 'Dear' ||rec_data.user_name|| '--write some text'|| --give some count from above que
                  write in cursor;
        END LOOP;
        UTL_SMTP.WRITE_DATA(V_CONN, 'From:' ||v_from);
        UTL_SMTP.WRITE_DATA(V_CONN, 'Subject:' ||v_sub);
        UTL_SMTP.WRITE_DATA(V_CONN, V_MSG);
        UTL_SMTP.CLOSE_DATA(V_CONN);

        UTL_SMTP.QUIT(V_CONN);

    EXCEPTION

    WHEN NO_DATA_FOUND THEN
        V_ERROR_CODE := SQLCODE;
        V_ERROR_MSG  := 'WHEN NO_DATA_FOUND'|| sQLERRM;
        ROLLBACK;
```

```
WHEN OTHERS THEN
    V_ERROR_CODE := SQLCODE;
    V_ERROR_MSG  := 'WHEN NO_DATA_FOUND'|| sQLERRM;
    ROLLBACK;


END PROC_SEND_EMAIL;
```

11.2 Trigger

This trigger is written for inserting values in user_ login_history table. At the time of inserting records in user_login table the trigger would automatically insert records in the history table

```
Create or replace trigger trg_login_list

After insert on user_login

For each row

Declare

V_hist_id number (10)

Begin

Userid = : old.userid

Username = :old.username

Password = :old.password


Select list id seq.nextval into v_hist_id from dual;

Insert into user_login (user_hist_id, userid, username, password) values

(user_hist_id, userid, username, password)
```

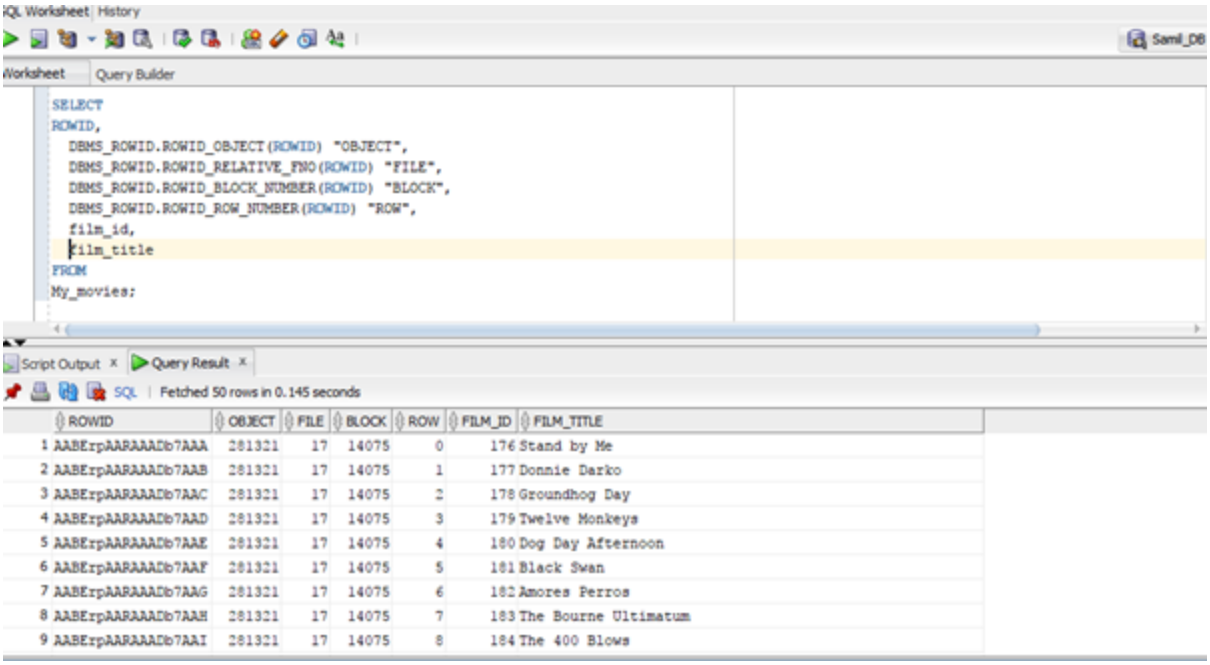
12. DBA Scripts

Some of the scripts which can be useful for DBA for monitoring and managing are discussed below

1. Scripts to obtain information on ROWID's, FILE, OBJECTS or table space in which a table is stored.

a)

```
SELECT
ROWID,
  DBMS_ROWID.ROWID_OBJECT(ROWID) "OBJECT",
  DBMS_ROWID.ROWID_RELATIVE_FNO(ROWID) "FILE",
  DBMS_ROWID.ROWID_BLOCK_NUMBER(ROWID) "BLOCK",
  DBMS_ROWID.ROWID_ROW_NUMBER(ROWID) "ROW",
  film_id,
  film_title
FROM
My_movies;
```



```
SELECT
ROWID,
  DBMS_ROWID.ROWID_OBJECT(ROWID) "OBJECT",
  DBMS_ROWID.ROWID_RELATIVE_FNO(ROWID) "FILE",
  DBMS_ROWID.ROWID_BLOCK_NUMBER(ROWID) "BLOCK",
  DBMS_ROWID.ROWID_ROW_NUMBER(ROWID) "ROW",
  film_id,
  film_title
FROM
My_movies;
```

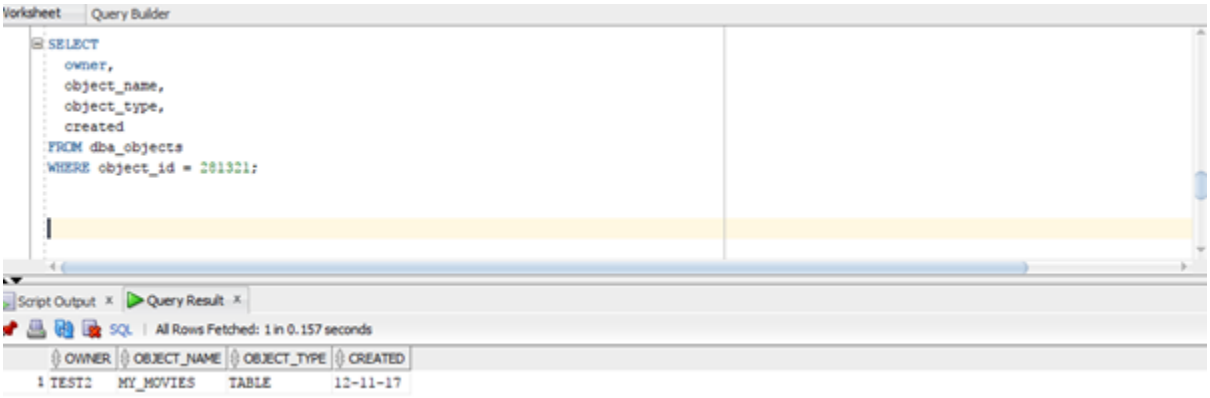
Fetched 50 rows in 0.145 seconds

	ROWID	OBJECT	FILE	BLOCK	ROW	FILM_ID	FILM_TITLE
1	AABErpAARAAADb7AAA	281321	17	14075	0	176	Stand by Me
2	AABErpAARAAADb7AAB	281321	17	14075	1	177	Donnie Darko
3	AABErpAARAAADb7AAC	281321	17	14075	2	178	Groundhog Day
4	AABErpAARAAADb7AAD	281321	17	14075	3	179	Twelve Monkeys
5	AABErpAARAAADb7AAE	281321	17	14075	4	180	Dog Day Afternoon
6	AABErpAARAAADb7AAF	281321	17	14075	5	181	Black Swan
7	AABErpAARAAADb7AAG	281321	17	14075	6	182	Amores Perros
8	AABErpAARAAADb7AAH	281321	17	14075	7	183	The Bourne Ultimatum
9	AABErpAARAAADb7AAI	281321	17	14075	8	184	The 400 Blows

Table My_movies in the database is stored in file_id = 17. This information can be further used to obtain information on logical storage or table spaces which store this table.

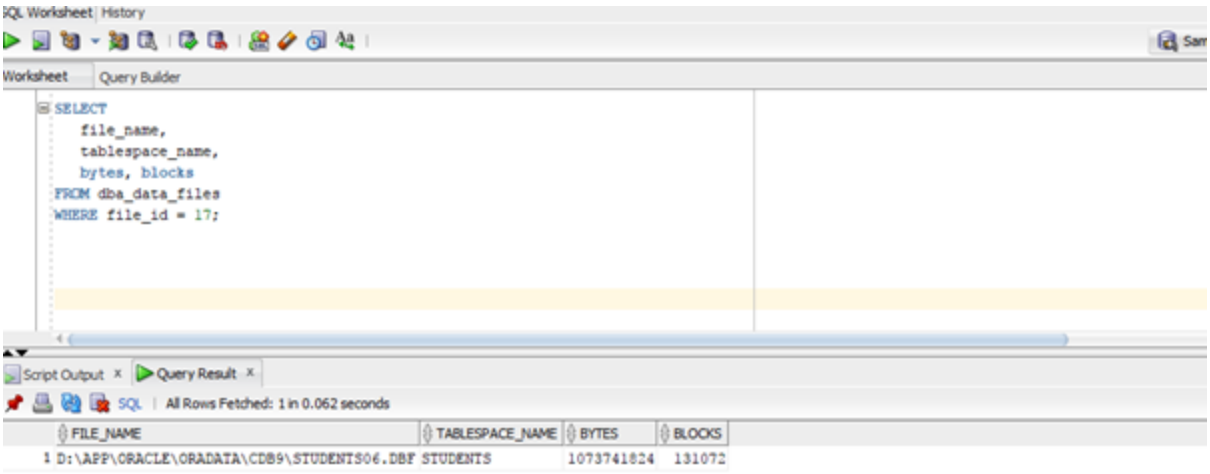
b) To get information on the ownership of an object

```
SELECT
 owner,
 object_name,
 object_type,
 created
FROM dba_objects
WHERE object_id = 281321;
```



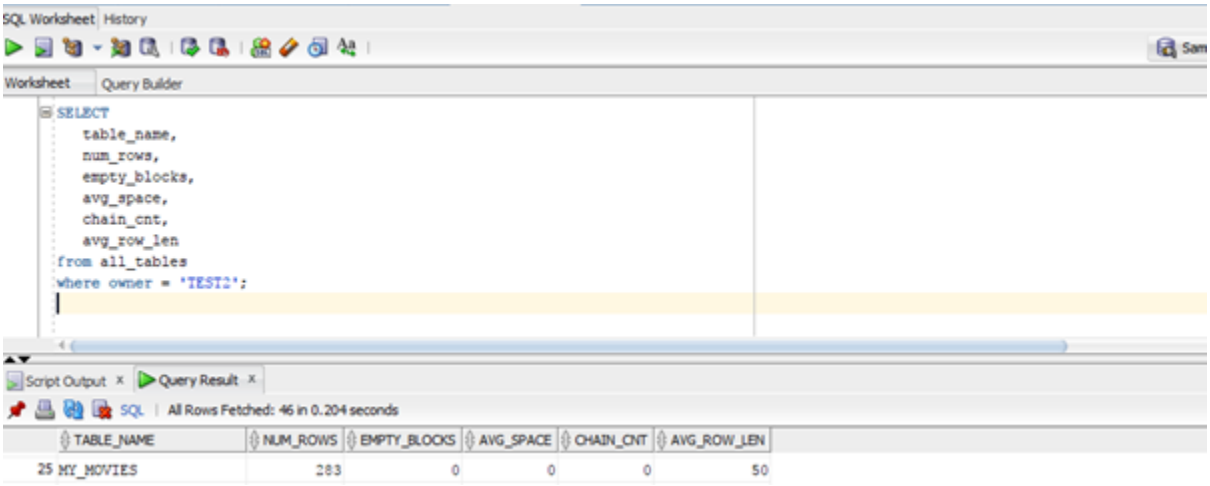
c) Scripts for information on Table space

```
SELECT
   file_name,
   tablespace_name,
   bytes, blocks
FROM dba_data_files
WHERE file_id = 17;
```



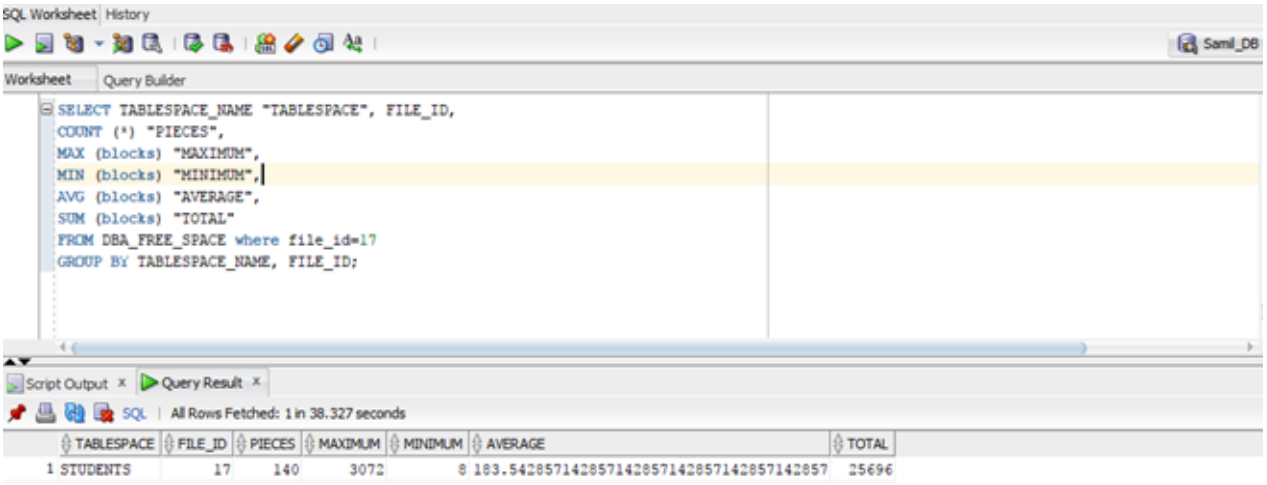
d. Scripts for getting information on all tables and space occupied by them

```
SELECT
   table_name,
   num_rows,
   empty_blocks,
   avg_space,
   chain_cnt,
   avg_row_len
from all_tables
where owner = 'TEST2';
```



e. Script for analyzing space available in various table spaces in the database and maximum and minimum block size.

```
SELECT TABLESPACE_NAME "TABLESPACE", FILE_ID,
COUNT (*) "PIECES",
MAX (blocks) "MAXIMUM",
MIN (blocks) "MINIMUM",
AVG (blocks) "AVERAGE",
SUM (blocks) "TOTAL"
FROM DBA_FREE_SPACE where file_id=17
GROUP BY TABLESPACE_NAME, FILE_ID;
```



Above information is important when any new object is added to a tablespace as it is important to

ascertain if the space is sufficient.

2. Scripts related to indexes

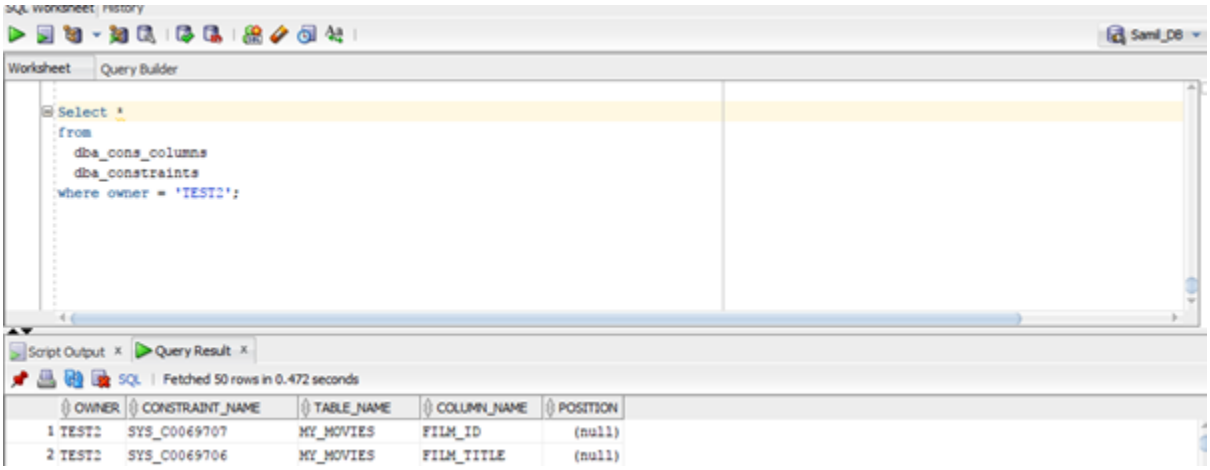
a) Script to list all the indexes of an database owner.

```
SELECT * FROM dba_indexes
WHERE owner = 'TEST2'
```

ORDER BY index_name;

3 DBA scripts for finding constraints in the tables

```
Select *
from
  dba_cons_columns
  dba_constraints
where owner = 'TEST2';
```



Topic Area	Description	Points
Database Design	This part should include a logical database design (for the relational model), using normalization to control redundancy and integrity constraints for data quality.The logical design section should include entity-relationship diagrams (ERDs) and data dictionaries for your database design, as well as any design assumptions. There should also be a complete ERD for your entire project. There is no expectation that you implement all of your design, just indicate the areas built.	30
Query Writing	This part is another chance to write SQL queries, explore transactions, and even do some database programming for stored procedures.Include interesting queries that highlight the types of questions that can be answered by the database.These queries may also be used to illustrate performance tuning.	25
Performance Tuning	In this section, you can capitalize and extend your prior experiments with indexing, optimizer modes, partitioning, parallel execution and any other techniques you want to further explore. Experiments with different indexing strategies, optimizer changes, transaction isolation levels, function-based indexes, and table partitioning can all be interesting. Remember to look at different types of queries (e.g., point, range, scan), execution plans, and I/O burden.	25
Other Topics	Here you are free to explore any other topics of interest. Suggestions include DBA scripts, database security, interface design, data visualization, data mining, and NoSQL databases.	20