IMDb Movies and T.V. Shows

Advanced Database Management

Gurmeet Kaur, Suryateja Chalapati and Tanawat Tejathavon
MS in Business Analytics and Information Systems
University of South Florida
{gurmeetkaur, chalapati, ttejathavon}@usf.edu

November 14, 2020

1 Executive Summary

The objective of this project is to design and implement the Entity Relation Diagram for the IMDb data set for movies and T.V. shows. This design can be used to explore and view information about movies and T.V. shows. Motivation behind doing this project came after working on movies database for class assignments. If a person wants to find details about any movie or show they have to look at different places to get complete information. Data for this project is fetched from IMDb website and is created as one stop destination for users to access the following information:

1. User can access list of movies and shows which are coming soon.
2. Region wise and language wise movies and shows can be found.
3. Type of display for the movies/shows can also be checked. For example, if the display is original, it is just the trailer, display is 3-D version etc.
4. List of movies/shows labeled based on their run time minutes - video, short, movie, tv series, tv mini series.
5. User can gets the details of tv shows about the total seasons and total episodes.
6. User can search for the list of movies and tv shows of their favorite actors/ actress, director, producer, writer and other roles.
7. User can look at the average rating given to their movie/show of choice.
8. User can also get the information and make sure if the video is fit for children to watch.

   (Explain about data elaborately)

Contents

2 Creating SQL Tables

SQL table consists of all the data in the IMDb database. For creating tables, Entity Relationship Diagram is designed (ERD) to see the relationship between entity and its attributes. Records of database are stored in relational database and data dictionary is one of its crucial component. Data dictionary is created to describe the contents, format and structure of the database. It consists of the name of the column, description, data type, size, identity. constraints of the table like keys, unique, index, nulls etc. Relationship between the columns of the database can be used to access the information and manipulate the database.

2.1 Entity Relationship Diagram

To design ERD for IMDb data set, it was carefully read and entities identified. ERD for the IMDb movies and tv shows is created to visualise entities, its attributes and relationships between them as shown in Fig. 1. For our database, TITLE_EPISODE, TITLE_RATINGS, TITLE_BASICS, SHOW_ATTRIBUTES, TITLE_PRINCIPALS and NAME_BASICS have been identified as entities.

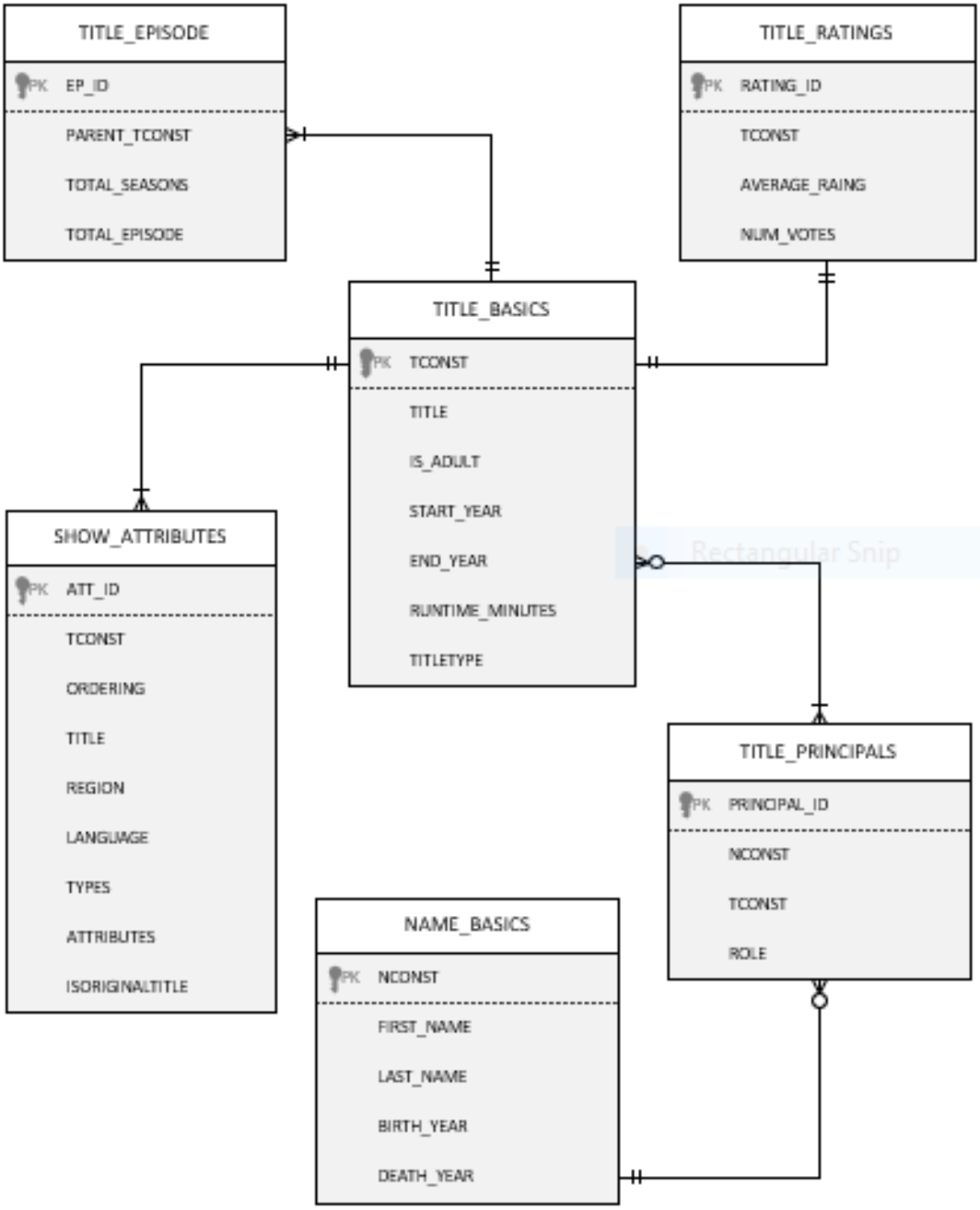

Figure 1: Entity Relationship Diagram

2.2 Relational Database and Data Dictionary

Records are stored within relational database as tables. Data dictionary consists of tables which are entities, columns represents attributes and rows represents records. It also consists of column description, data type, size of data, constraint, null and index.

Table 1: NAME_BASICS

NAME_BASICS							
Column name	Description	Data Type	Size	Unique	Constraint Type	Allow Nulls	Index
NCONST	alphanumeric unique identifier of the name/person	Varchar	25	Yes	Check	Not Null	Yes
FIRST_NAME	First name of the person	Varchar	20		Check	Not Null	Yes
LAST_NAME	Last name of the person	Varchar	20		Check	Not Null	
BIRTH_YEAR	Birth year of the person	Number					
DEATH_YEAR	Death year of the person	Number					

Table 2: SHOW_ATTRIBUTES

SHOW_ATTRIBUTES							
Column name	Description	Data Type	Size	Unique	Constraint Type	Allow Nulls	Index
ATT_ID	alphanumeric unique identifier of the name/person	Varchar	20	Yes	Check	Not Null	Yes
TCONST	alphanumeric unique identifier of the title	Varchar	20	Yes	Check	Not Null	Yes
ORDERING	a number to uniquely identify rows for a given titleId	Number		Yes	Check	Not Null	Yes
TITLE	the localized title	Varchar	45				
REGION	the region for this version of the title	Varchar	20				
LANGUAGE	the language of the title	Varchar	20				
TYPES	Enumerated set of attributes for this alternative title	Varchar	20		alternative, dvd, festival, tv, video, working, original, imdbDisplay		
ATTRIBUTES	Additional terms to describe this alternative title, not enumerated	Number					
ISORIGINALTITLE	Originality of the title	Number			0: not original title; 1: original title	Not Null	

Table 3: TITLE_BASICS

TITLE_BASICS							
Column name	Description	Data Type	Size	Unique	Constraint Type	Allow Nulls	Index
TCONST	alphanumeric unique identifier of the title	Nvarchar	20	Yes	Check	Not Null	Yes
TITLE	the localized title	Nvarchar	250		Check	Not Null	
IS_ADULT	Adult or not	Number			0: non-adult title; 1: adult title	Not Null	
START_YEAR	represents the release year of a title. In the case of TV Series, it is the series start year	Number			Check	Not Null	
END_YEAR	TV Series end year. '\N' for all other title types	Number					
RUNTIME_MINUTES	primary runtime of the title, in minutes	Number					
TITLETYPE	the type/format of the title	Varchar	20		movie, short, tvseries, tvepisode, video		

Table 4: TITLE_EPISODE

TITLE_EPISODE							
Column name	Description	Data Type	Size	Unique	Constraint Type	Allow Nulls	Index
EP_ID	alphanumeric unique identifier of the name/person	Varchar	20	Yes	Check	Not Null	Yes
PARENT_TCONST	alphanumeric identifier of the parent TV Series	Varchar	20		Check	Not Null	Yes
TOTAL_SEASONS	Total number of seasons	Number			Check	Not Null	
TOTAL_EPISODE	Total number of episodes in a show	Number			Check	Not Null	

Table 5: TITLE_PRINCIPALS

TITLE_PRINCIPALS							
Column name	Description	Data Type	Size	Unique	Constraint Type	Allow Nulls	Index
PRINCIPAL_ID	alphanumeric unique identifier of the Principal	Varchar	20	Yes	Check	Not Null	Yes
NCONST	alphanumeric unique identifier of the name/person	Varchar	20	Yes	Check	Not Null	Yes
TCONST	alphanumeric unique identifier of the title	Varchar	20	Yes	Check	Not Null	Yes
ROLE	specific job title	Varchar	20				

Table 6: TITLE_RATINGS

TITLE_RATINGS							
Column name	Description	Data Type	Size	Unique	Constraint Type	Allow Nulls	Index
RATING_ID	alphanumeric unique identifier of the Rating	Varchar	20	Yes	Check	Not Null	Yes
TCONST	alphanumeric unique identifier of the title	Varchar	20	Yes	Check	Not Null	Yes
AVARAGE_RAING	weighted average of all the individual user ratings	Number					
NUM_VOTES	number of votes the title has received	Number					

3 Scripts for reading tab separated files, data cleaning and inserting data into SQL Tables

The data set was in tab separated format so to read it pandas library was used. Original data set was huge and each table consisted of 7 million rows. Number of rows in TITLE_BASICS and NAME_BASICS were limited to 15000 and data after 2019 was inserted into the tables, rest all the tables were built on that. TITLE_EPISODES consists of total number of episodes and total number of seasons. One table had roles and other had names with no roles. We put relationship for both with and without roles in TITLE_PRINCIPALS table. We exported all tables to csv for syncing purposes.

```
import cx_Oracle
cx_Oracle.init_oracle_client(lib_dir="instantclient_19_8")
dsn_tns = cx_Oracle.makedsn('reade.forest.usf.edu', '1521','cdb9')
conn = cx_Oracle.connect(user='DB372', password='<password>', dsn=dsn_tns)
cursor = conn.cursor()

```

```
import pandas as pd
import sys

sql='insert into TITLE_BASICS values(:1,:2,:3,:4,:5,:6,:7)'
n=0
m=0
dft = pd.read_csv('title.episode.tsv', sep='\t')

for df in pd.read_csv('title.basics.tsv', sep='\t', chunksize=5000):
    df = df.replace('\\N', '')
    print(n)
    for index, row in df.iterrows():
         if row['startYear'] == '' or int(row['startYear']) < 2015 or
         len(dft.loc[dft['tconst']==row['tconst']]) > 0 : continue
         li = [row['tconst'], row['originalTitle'], row['isAdult'],
               row['startYear'], row['endYear'] ,
               row['runtimeMinutes'], row['titleType']]
         try:
             cursor.execute(sql,li)
             m += 1
             print("inserted: ", m)
             if m >= 10000:
                 conn.commit()
                 conn.close()
                 sys.exit(1)
         except Exception as e:
             print(row)
             print(str(e))
    n+=5000
    conn.commit()

conn.close()
```

```
import cx_Oracle
cx_Oracle.init_oracle_client(lib_dir="instantclient_19_8")
dsn_tns = cx_Oracle.makedsn('reade.forest.usf.edu', '1521','cdb9')
conn = cx_Oracle.connect(user='DB372', password='<password>', dsn=dsn_tns)
cursor = conn.cursor()


import pandas as pd
import sys

```

```

sql='insert into NAME_BASICS values(:1,:2,:3,:4,:5)'


n=0
m = 0
dft = pd.read_csv('export3.csv')

for df in pd.read_csv('name.basics.tsv', sep='\t', chunksize=5000):
    df = df.replace('\\N', '')
    print(n)
    for index, row in df.iterrows():
        known = row['knownForTitles'].split(',')
        for i in known:
           if len(dft.loc[dft['TCONST']==i]) == 0: continue
           names = row['primaryName'].split()
           li = [row['nconst'], names[0], names[-1],
           row['birthYear'], row['deathYear']]
           print(i)
           try:
               cursor.execute(sql,li)
               conn.commit()
               m += 1
               if m >= 15000:
                   conn.close()
                   sys.exit(1)
               break
           except Exception as e:
             print(row)
             print(str(e))
    n+=5000
    conn.commit()
```

```
import cx_Oracle
cx_Oracle.init_oracle_client(lib_dir="instantclient_19_8")
dsn_tns = cx_Oracle.makedsn('reade.forest.usf.edu', '1521','cdb9')
conn = cx_Oracle.connect(user='DB372', password='<password>', dsn=dsn_tns)
cursor = conn.cursor()

import pandas as pd
import sys

sql='insert into TITLE_PRINCIPALS values(:1,:2,:3, :4)'
```

```
m = 7200
ided = 1
dft = pd.read_csv('export3.csv')
dtf = pd.read_csv('export4.csv')
ins = pd.read_csv('export7.csv')
n=0
for df in pd.read_csv('name.basics.tsv', sep='\t', chunksize=5000):
    df = df.replace('\\N', '')
    print(n)
    for index, row in df.iterrows():
        n+=1
        known = row['knownForTitles'].split(',')
        if len(dtf.loc[dtf['NCONST'] == row['nconst']]) == 0: continue
        for i in known:
           if len(dft.loc[dft['TCONST']==i]) == 0: continue
           if not len(ins[(ins['NCONST']==row['nconst']) &
           (ins['TCONST']==i)]) == 0: continue
           li = [m, row['nconst'], i, ""]
           try:
               cursor.execute(sql,li)
               conn.commit()
               m += 1
           except Exception as e:
             print(row)
             print(str(e))


conn.close()
```

```
import cx_Oracle
cx_Oracle.init_oracle_client(lib_dir="instantclient_19_8")
dsn_tns = cx_Oracle.makedsn('reade.forest.usf.edu', '1521','cdb9')
conn = cx_Oracle.connect(user='DB372', password='<password>', dsn=dsn_tns)
cursor = conn.cursor()

import pandas as pd
import sys

sql='insert into SHOW_ATTRIBUTES values(:1,:2,:3,:4,:5,:6,:7,:8,:9)'
ided = 0

dft = pd.read_csv('export3.csv')
n=340000
```

```
for df in pd.read_csv('title.akas.tsv', sep='\t', chunksize=5000,
skiprows=range(1,340000)):
    df = df.replace('\\N', '')
    print(n)
    for index, row in df.iterrows():
        n+=1
        if len(dft.loc[dft['TCONST']==row['titleId']]) == 0 : continue
        pid = "att" + str(ided).zfill(9)
        print(row)

        li = [pid, row['titleId'], row['ordering'], row['title'],
              row['region'], row['language'],row['types'],
              row['attributes'], row['isOriginalTitle']]
        try:
            cursor.execute(sql,li)
            conn.commit()
            ided+=1
        except Exception as e:
            print(row)
            print(str(e))

conn.close()
```

```
import cx_Oracle
cx_Oracle.init_oracle_client(lib_dir="instantclient_19_8")
dsn_tns = cx_Oracle.makedsn('reade.forest.usf.edu', '1521','cdb9')
conn = cx_Oracle.connect(user='DB372', password='<password>', dsn=dsn_tns)
cursor = conn.cursor()

import pandas as pd
import sys

sql='insert into TITLE_PRINCIPALS values(:1,:2,:3,:4)'

n=455000
m = 0

ided = 1
dft = pd.read_csv('export3.csv')
dtf = pd.read_csv('export4.csv')

for df in pd.read_csv('title.principals.tsv', sep='\t',
chunksize=5000, skiprows=range(1,455000)):
```

```
    df = df.replace('\\N', '')
    print(n)
    for index, row in df.iterrows():

        if len(dtf.loc[dtf['NCONST']==row['nconst']]) == 0 or
        len(dft.loc[dft['TCONST']==row['tconst']]) == 0 : continue

        pid = str(ided).zfill(7)
        li = [pid, row['nconst'], row['tconst'], row['category']]
        try:
            cursor.execute(sql,li)
            conn.commit()
            ided+=1
        except Exception as e:
            print(row)
            print(str(e))


    n+=5000
conn.close()
```

```
import cx_Oracle
cx_Oracle.init_oracle_client(lib_dir="instantclient_19_8")
dsn_tns = cx_Oracle.makedsn('reade.forest.usf.edu', '1521','cdb9')
conn = cx_Oracle.connect(user='DB372', password='<password>', dsn=dsn_tns)
cursor = conn.cursor()

import pandas as pd
import sys

sql='insert into SHOW_ATTRIBUTES values(:1,:2,:3,:4,:5,:6,:7,:8,:9)'
ided = 0

dft = pd.read_csv('export3.csv')
n=340000
for df in pd.read_csv('title.akas.tsv', sep='\t', chunksize=5000,
skiprows=range(1,340000)):
    df = df.replace('\\N', '')
    for index, row in df.iterrows():
        n+=1
        if len(dft.loc[dft['TCONST']==row['titleId']]) == 0 : continue
        pid = "att" + str(ided).zfill(9)

```

```
        li = [pid, row['titleId'], row['ordering'], row['title'],
              row['region'], row['language'],row['types'],
              row['attributes'], row['isOriginalTitle']]
        try:
            cursor.execute(sql,li)
            conn.commit()

            ided+=1
        except Exception as e:
            print(row)
            print(str(e))

conn.close()
```

```
import cx_Oracle
cx_Oracle.init_oracle_client(lib_dir="instantclient_19_8")
dsn_tns = cx_Oracle.makedsn('reade.forest.usf.edu', '1521','cdb9')
conn = cx_Oracle.connect(user='DB372', password='<password>', dsn=dsn_tns)
cursor = conn.cursor()

import pandas as pd
import sys

sql='insert into TITLE_RATINGS values(:1,:2,:3,:4)'
ided = 0

dft = pd.read_csv('export3.csv')
n=40000
for df in pd.read_csv('title.ratings.tsv', sep='\t',
chunksize=5000,skiprows=range(1,40000)):
    df = df.replace('\\N', '')
    print(n)
    for index, row in df.iterrows():
        n+=1
        if len(dft.loc[dft['TCONST']==row['tconst']]) == 0 : continue
        pid = "rid" + str(ided).zfill(7)
        li = [pid, row['tconst'], row['averageRating'], row['numVotes']]
        try:
            cursor.execute(sql,li)
            conn.commit()
            ided+=1
        except Exception as e:
            print(row)
```

```
            print(str(e))

conn.close()
```

4 Data Exploration and Query Writing

In this section, the database would be investigated to understand the structure. Each table will be explained in the subsections and determine the restriction of the database in hand.

4.1 Table 1 : title_basics

According to the ERD shown in Fig 1, the table basic is the center which the others table connect. Firstly, this table did not have primary key as a default shown in Fig 2 and the Fig 3 shows the primary key, tconst, after being altered.

	CONSTRAINT_NAME	CONSTRAINT_TYPE	SEARCH_CONDITION	R_OWNER	R_TABLE_NA
1	SYS_C00116749	Check	"TCONST" IS NOT NULL	(null)	(null)
2	SYS_C00116750	Check	"TITLE" IS NOT NULL	(null)	(null)
3	SYS_C00116751	Check	"IS_ADULT" IS NOT NULL	(null)	(null)
4	SYS_C00116752	Check	"START_YEAR" IS NOT NULL	(null)	(null)

Figure 2: title_basics without primary key

```
ALTER TABLE title_basics
ADD CONSTRAINT title_basics_pk
PRIMARY KEY (tconst);
```

	CONSTRAINT_NAME	CONSTRAINT_TYPE	SEARCH_CONDITION	R_OWNER	R_TABLE_NAME	R_CON
1	SYS_C00116749	Check	"TCONST" IS NOT NULL	(null)	(null)	(null)
2	SYS_C00116750	Check	"TITLE" IS NOT NULL	(null)	(null)	(null)
3	SYS_C00116751	Check	"IS_ADULT" IS NOT NULL	(null)	(null)	(null)
4	SYS_C00116752	Check	"START_YEAR" IS NOT NULL	(null)	(null)	(null)
5	TITLE_BASICS_PK	Primary_Key	(null)	(null)	(null)	(null)

Figure 3: title_basics with primary key

4.2 Table 2 : title_episode

Same pattern is applied to this table. However, there are duplicated data exist in this table shown in Fig 4 with prevent from creating primary key. After delete it, Fig 5 show the primary key, which is ep_id. Surprisingly, parent_tconst is the subset of tconst in title_basic.

```
SELECT *
FROM title_episode
WHERE ep_id = 'epid000000000';
```

SQL | All Rows Fetched: 2 in 0.011 seconds

	EP_ID	PARENT_TCONST	TOTAL_SEASONS	TOTAL_EPISODE
1	epid000000000	tt10004066	1	26
2	epid000000000	tt10004066	1	26

Figure 4: title_episode duplicate data

```
DELETE FROM title_episode
WHERE rowid not in (
SELECT MAX(rowid)
FROM title_episode
GROUP BY ep_id);
```

	CONSTRAINT_NAME	CONSTRAINT_TYPE	SEARCH_CONDITION	R_
1	SYS_C00117209	Check	"EP_ID" IS NOT NULL	(nul
2	SYS_C00117210	Check	"PARENT_TCONST" IS NOT NULL	(nul
3	SYS_C00117211	Check	"TOTAL_SEASONS" IS NOT NULL	(nul
4	SYS_C00117212	Check	"TOTAL_EPISODE" IS NOT NULL	(nul
5	TITLE_EPISODE_PK	Primary_Key	(null)	(nul

Figure 5: title_episode primary key

4.3 Table 3 : show_attributes

In contrast to table: title_episode, there is no problem exist. Fig 6 show the primary key of this table.

```
ALTER TABLE show_attributes
ADD CONSTRAINT show_attributes_pk
PRIMARY KEY (att_id);
```

	CONSTRAINT_NAME	CONSTRAINT_TYPE	SEARCH_CONDITION	R_OWN
1	SHOW_ATTRIBUTES_PK	Primary_Key	(null)	(null)
2	SYS_C00117182	Check	"ATT_ID" IS NOT NULL	(null)
3	SYS_C00117183	Check	"TCONST" IS NOT NULL	(null)
4	SYS_C00117184	Check	"ORDERING" IS NOT NULL	(null)
5	SYS_C00117185	Check	"ISORIGINALTITLE" IS NOT NULL	(null)

Figure 6: show_attributes primary key

4.4 Table 3 : title_rating

Fig 7 show the primary key of this table.

```
ALTER TABLE title_ratings
ADD CONSTRAINT title_ratings_pk
PRIMARY KEY (rating_id);
```

	CONSTRAINT_NAME	CONSTRAINT_TYPE	SEARCH_CONDITION	R
1	SYS_C00117169	Check	"RATING_ID" IS NOT NULL	(nu
2	TITLE_RATINGS_PK	Primary_Key	(null)	(nu

Figure 7: title_rating primary key

4.5 Table 4 : title_principals

Fig 8 show the primary key of this table.

```
ALTER TABLE title_principals
ADD CONSTRAINT title_principals_pk
PRIMARY KEY (principal_id);
```

	CONSTRAINT_NAME	CONSTRAINT_TYPE	SEARCH_CONDITION	R_OWNER
1	SYS_C00117004	Check	"PRINCIPAL_ID" IS NOT NULL	(null)
2	SYS_C00117005	Check	"NCONST" IS NOT NULL	(null)
3	SYS_C00117006	Check	"TCONST" IS NOT NULL	(null)
4	TITLE_PRINCIPALS_PK	Primary_Key	(null)	(null)

Figure 8: title_principal primary key

4.6 Table 5 : name_basics

Fig 9 show the primary key of this table.

```
ALTER TABLE name_basics
ADD CONSTRAINT name_basics_pk
PRIMARY KEY (nconst);
```

	CONSTRAINT_NAME	CONSTRAINT_TYPE	SEARCH_CONDITION	
1	NAME_BASICS_PK	Primary Key	(null)	(
2	SYS_C00116978	Check	"NCONST" IS NOT NULL	(
3	SYS_C00116979	Check	"FIRST_NAME" IS NOT NULL	(
4	SYS_C00116980	Check	"LAST_NAME" IS NOT NULL	(

Figure 9: name_basics primary key

4.7 Interesting Queries

4.7.1 Display movie name, people who are responsible with their roles, year of the movie, and age restriction

```
SELECT
    tb.title,
    nb.first_name,
    nb.last_name,
    tp.role,
    tb.start_year,
    tb.is_adult
FROM
    title_basics tb
```

```
    INNER JOIN title_principals tp
    USING (tconst)
    INNER JOIN name_basics nb
    USING (nconst)
ORDER BY tb.start_year DESC, nb.first_name;
```

	TITLE	FIRST_NAME	LAST_NAME	ROLE	START_YEAR	IS_ADULT
1	Feroce	Anthony	Burns	director	2022	0
2	The Christmas Kill	Laura	Sosin	producer	2022	0
3	Shazam! Fury of the Gods	Peter	Safran	producer	2022	0
4	Jungle Cruise	Abraham	IV	(null)	2021	0
5	Jungle Cruise	Abraham	Takatsuki	(null)	2021	0
6	Chinese Speaking Vampires	Adalina	Aladro	(null)	2021	0
7	Mortal Kombat	Aimee	Mullins	(null)	2021	0
8	Perspectiva	Alba	Fernández	(null)	2021	0
9	Perspectiva	Alba	Lozano	(null)	2021	0
10	The Little Things	Albert	Penuelas	(null)	2021	0
11	Jungle Cruise	Alex	Anduze	(null)	2021	0
12	Port Radacini	Alexis	Gogoasa	director	2021	0
13	Satria Dewa: Gatotkaca	Ali	Fikry	(null)	2021	0
14	Mortal Kombat	Alisha	Hennessy	(null)	2021	0
15	Mare of Easttown	Allen	Fawcett	(null)	2021	0
16	Without Remorse	Ambra	Fisser	(null)	2021	0
17	Inside the Notebook	Anastasia	Perevozova	actress	2021	0
18	My Dinner with Chuck	Andrew	Kim	(null)	2021	0
19	Cinderella	Andrew	Phizacklea	(null)	2021	0
20	Mortal Kombat	Angela	Heesom	(null)	2021	0
21	Inside the Notebook	Annabel	Lavkraft	(null)	2021	0
22	The Naked Ghost	Ar	Noe	cinematographer	2021	0

Figure 10: 6.7.1 Result

4.7.2 The movies which has been changed the name after broadcast in another region

```
SELECT
    tb.title Original_name,
    sa.title Alternative_name,
    sa.region
FROM
    title_basics tb
    INNER JOIN show_attributes sa
    USING (tconst)
WHERE tb.title != sa.title;
```

	ORIGINAL_NAME	ALTERNATIVE_NAME	REGION
1	The Other Side of the Wind	Al otro lado del viento	AR
2	The Other Side of the Wind	¿¿¿¿¿¿ ¿¿¿¿¿¿¿ ¿¿¿¿¿	RU
3	The Other Side of the Wind	L'altra faccia del vento	IT
4	The Other Side of the Wind	¿ ¿¿¿¿ ¿¿e¿¿¿ t¿¿ ¿¿¿µ¿¿	GR
5	The Other Side of the Wind	O Outro Lado do Vento	BR
6	The Other Side of the Wind	Phía Bên Kia Con Gió	VN
7	The Other Side of the Wind	O Outro Lado do Vento	PT
8	The Other Side of the Wind	A szél másik oldala	HU
9	The Other Side of the Wind	Druga strona wiatru	PL
10	The Other Side of the Wind	¿¿¿¿¿¿	JP
11	The Other Side of the Wind	La otra cara del viento	VE
12	The Other Side of the Wind	De l'autre côté du vent	FR
13	The Other Side of the Wind	Al otro lado del viento	ES
14	Toula ou Le génie des eaux	Toula, or the Water Genie	US
15	Toula ou Le génie des eaux	Toula oder Der Geist des Wassers	DDDE
16	Toula ou Le génie des eaux	Toula or the Water Spirit	(null)
17	A Dangerous Practice	Predestinado	BR
18	A Dangerous Practice	Arigó	US
19	A Dangerous Practice	Predestinado, Arigó e o Espírito do Dr. Fritz	BR
20	La Telenovela Errante	Objazdowa opera mydlana	PL
21	La Telenovela Errante	The Wandering Soap Opera	US
22	La Telenovela Errante	The Wandering Soap Opera	XWW
23	La bague du roi Koda	King Koda's Ring	(null)

Figure 11: 6.7.2 Result

4.7.3 According to the previous query, count the movies which does not contain Latin alphabet in alternative name

```
SELECT
    tb.title,
    COUNT(tb.title)
FROM
    title_basics tb
    INNER JOIN show_attributes sa
    USING (tconst)
WHERE tb.title != sa.title AND sa.title LIKE '%¿%'
GROUP BY tb.title;
```

	TITLE	COUNT(TB.TITLE)
1	Nappily Ever After	4
2	The Stanford Prison Experiment	4
3	A Million Little Pieces	1
4	Alita: Battle Angel	8
5	Suburbicon	6
6	Tulip Fever	5
7	The Danish Girl	7
8	Road to Red	1
9	Pelé: Birth of a Legend	6
10	Child of Wisdom	1
11	The Boiling Water LAMA	3
12	Protivostoyanie	1
13	Alien: Containment	1
14	Neele Gagan Ke Tale	1
15	Channeru wa sono mama!	2
16	Guilty	2
17	The Menarche	3
18	Antebellum	7
19	Ningen shikkaku: Dazai Osamu to 3-nin no onnatachi	2
20	Nguoi Phán Xu	1
21	Kevin Hart: Irresponsible	2
22	Wo men yu e de ju li	2

Figure 12: 6.7.3 Result

4.7.4 Show the average IMDB ranking and the number of voter, which should be more than 1000, for each movie. Higher rank comes first.

```
SELECT
    tb.title,
    tr.avarage_raing,
    tr.num_votes
FROM
    title_basics tb
    INNER JOIN title_ratings tr
    USING (tconst)
WHERE tr.num_votes >1000
ORDER BY tr.avarage_raing DESC;
```

	TITLE	AVARAGE_RAING	NUM_VOTES
1	Wo men yu e de ju li	9.2	1737
2	Westworld	8.7	411688
3	Medically Yourrs	8.6	1755
4	Baarish	8.6	1319
5	Julie and the Phantoms	8.5	1475
6	Fittrat	8.5	1889
7	Coldd Lassi Aur Chicken Masala	8.2	1961
8	I Know This Much Is True	8.2	7654
9	Borderlands 3	8.2	1022
10	Final Fantasy XV	8.2	2111
11	Bekaaboo	8.1	2470
12	Love, Victor	8.1	5129
13	Street Food: Asia	8	1986
14	Drzavni sluzbenik	8	1163
15	Linda Ronstadt: The Sound of My Voice	8	1372
16	BOSS: Baap of Special Services	8	1942
17	Carnival Row	7.9	41762
18	Never Have I Ever	7.9	20434
19	I Think You Should Leave with Tim Robinson	7.7	5074
20	Home for Christmas	7.6	4844
21	Afsos	7.6	2083

Figure 13: 6.7.4 Result

5 Performance Tuning

This section explores various performance tuning methods which can increase query speed and efficiency. The main purpose of database tuning is to increase query speed. Database tuning is a more broader term which includes optimization, database management system applications and database environment configuration which includes OS optimizer, CPU single and multi-threading, memory etc. In this project, we will be specifically looking at database performance tuning techniques like indexing, partitioning, parallel execution etc.

5.1 Indexing

Let's start-off with indexing. Indexes are widely used to quickly search through the data without having to search every single row in a table when a database is accessed. Indexes can be created using a single or multiple columns of a database table, which provides both rapid random lookups and efficient access of ordered records.

```
SELECT *
FROM DB372.MOVIES
WHERE FILM_TITLE = 'Life of Pi';
```

The required query output is as follows:

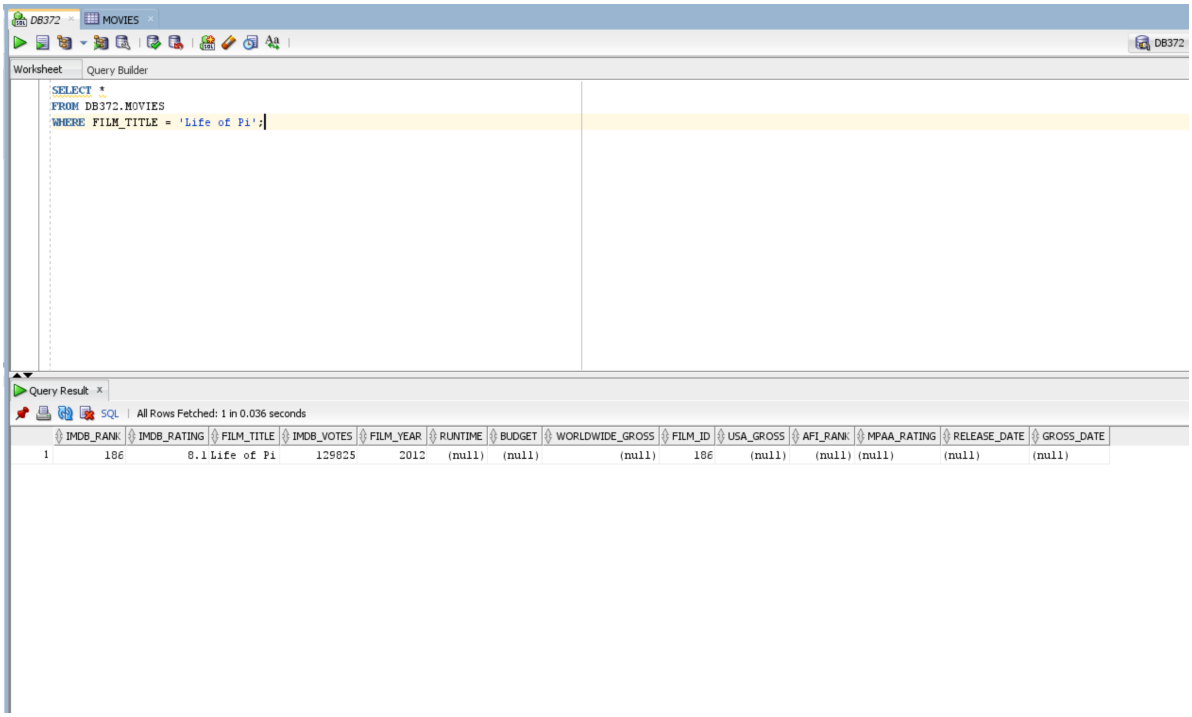

Figure 14: Query Output

When we look at Execution Plan, we can see that Indexing is not being used and the session logical reads is 93.

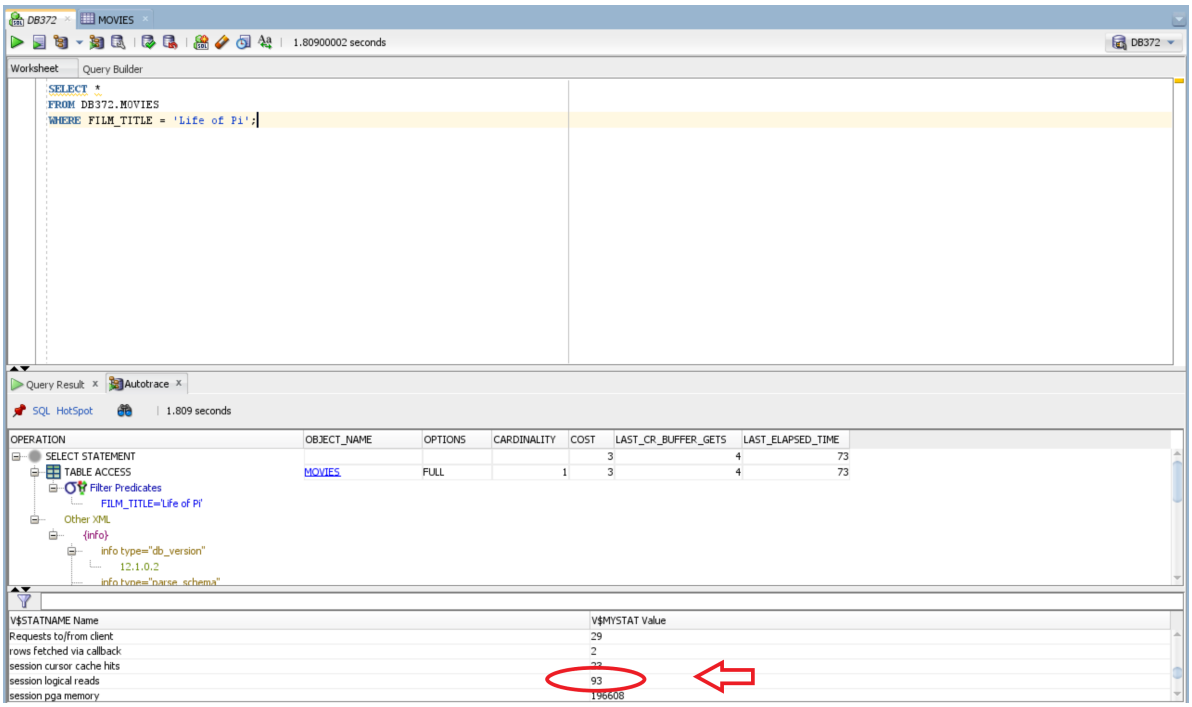

Figure 15: Execution Plan without Indexing

After querying with the indexing id and looking at Execution Plan, we can see that Indexing is being used and the session logical reads have reduced to 7. The cost of querying has also dropped from 3 to 1.

```
1 SELECT *
2 FROM DB372.MOVIES
3 WHERE FILM_ID = 186;
```

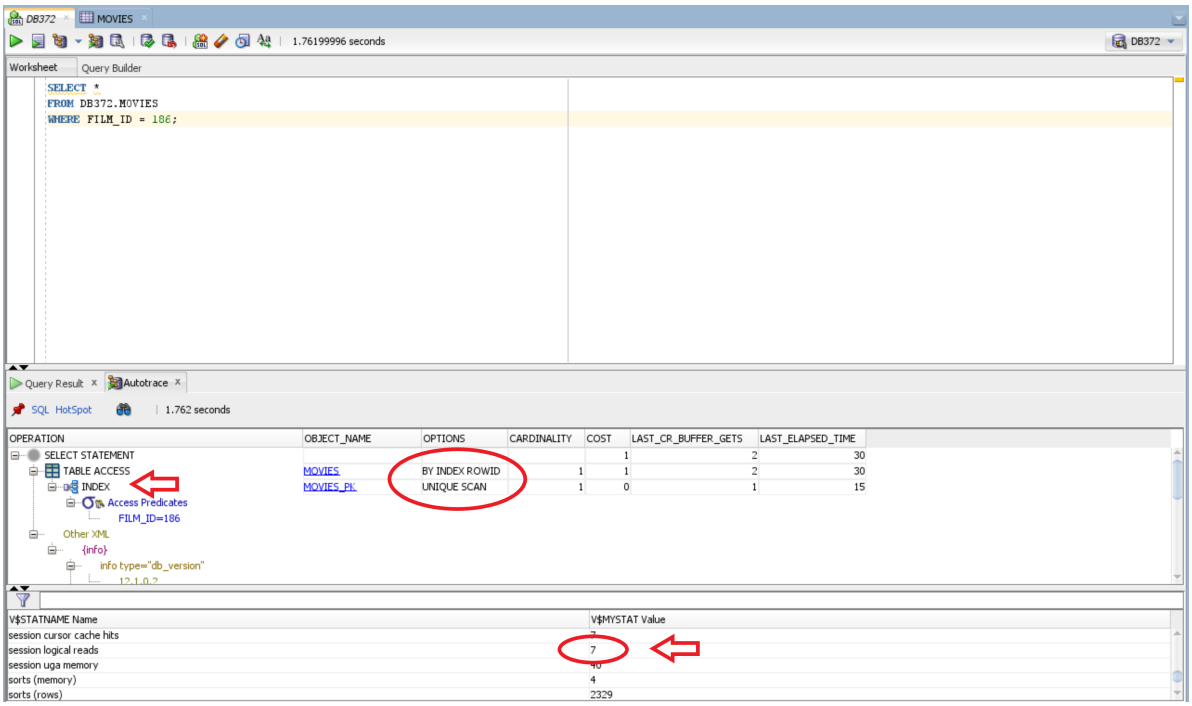

Figure 16: Execution Plan with Indexing

5.2 Function-Based Indexing

As we can see, indexing is performing a full scan, this can be further optimized by using Function-based Indexing. In Function-Based Indexing, we use something called a range scan, instead of full scan, by doing this the query will be faster where the optimizer does complex querying.

5.2.1 Case - 1: Without Function-Based Indexing

Here we fetch a range of years from FILM_YEAR column on MOVIES. We are using a CEIL() function to fetch the range on FILM_YEAR. Upon running the auto-trace on the execution plan, we find out that indexing is not being used and a ”Full” scan takes place.

```
SELECT
    IMDB_RANK,
    IMDB_RATING,
    FILM_TITLE,
    FILM_YEAR,
    FILM_ID
FROM DB372.MOVIES
WHERE CEIL(FILM_YEAR) > 1990 AND CEIL(FILM_YEAR) < 2000
AND IMDB_RANK IS NOT NULL;
```

We can see the query output below:

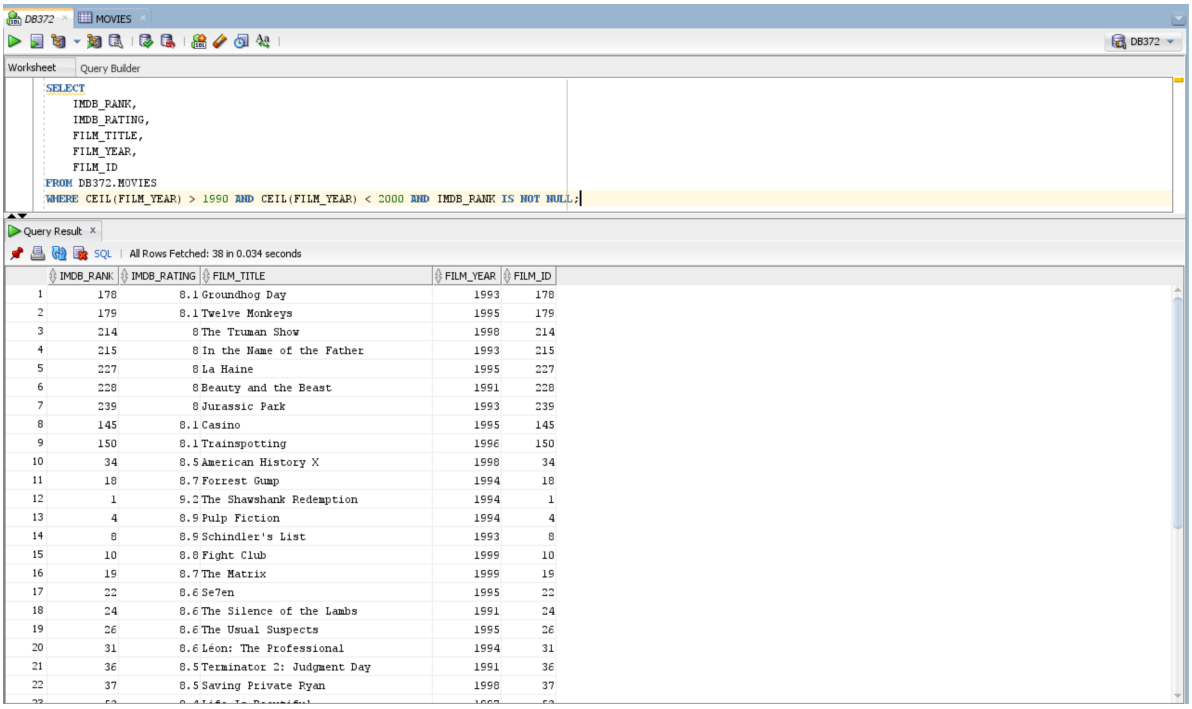

Figure 17: Execution Plan with Indexing

The execution plan shows that a ”Full” scan takes place:

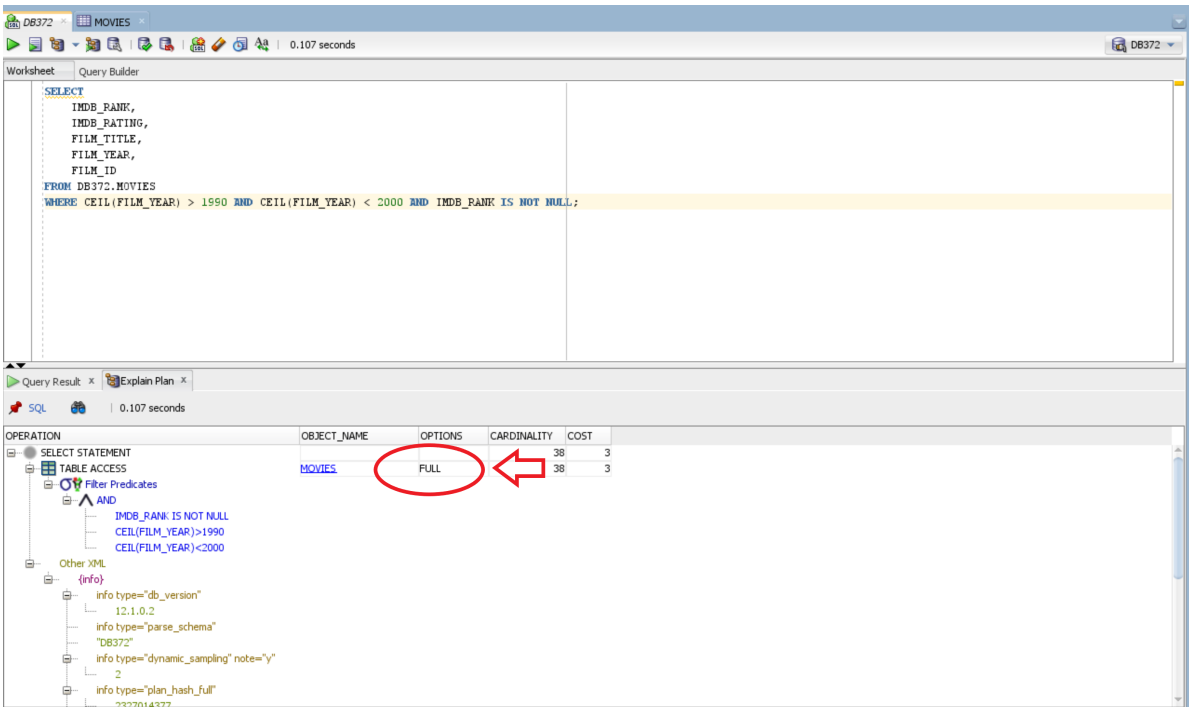

Figure 18: Execution Plan with Indexing

5.2.2 Case - 2: With Function-Based Indexing

Now we use Function-Based indexing by creating an index on MOVIES. Then we fetch a range of years from FILM_YEAR column on MOVIES again. We are using a CEIL() function to fetch the range on FILM_YEAR. Upon running the auto-trace on the execution plan, we find out that indexing is being used and a ”Range” scan takes place.

```
1  CREATE INDEX MOVIES_YEAR_INDEX ON MOVIES(CEIL(FILM_YEAR));
2
3  SELECT
4      IMDB_RANK,
5      IMDB_RATING,
6      FILM_TITLE,
7      FILM_YEAR,
8      FILM_ID
9  FROM DB372.MOVIES
10 WHERE CEIL(FILM_YEAR) > 1990 AND CEIL(FILM_YEAR) < 2000
11 AND IMDB_RANK IS NOT NULL;
```

Index created on MOVIES_YEAR_MOVIES as seen below:

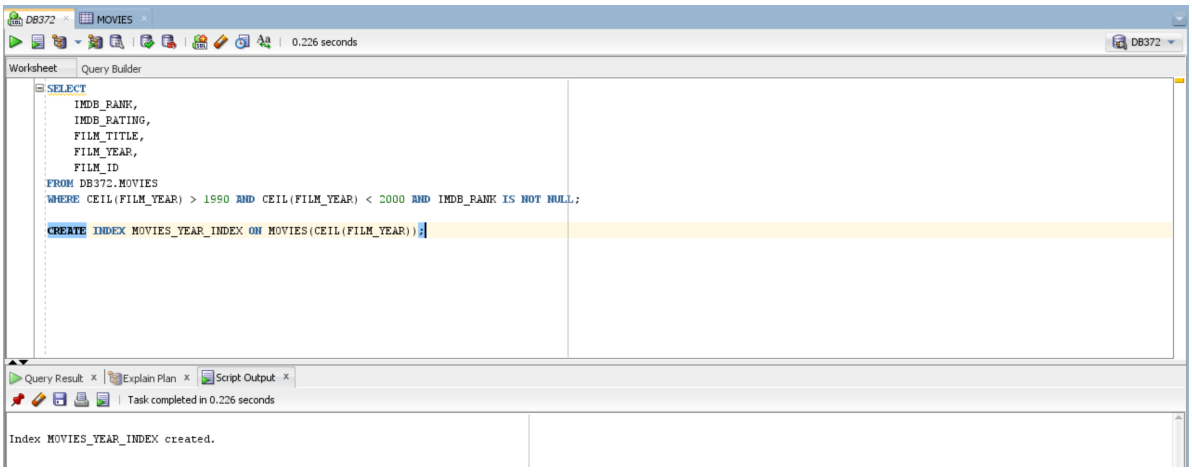

Figure 19: Execution Plan with Indexing

The execution plan shows that a ”Range” scan takes place:

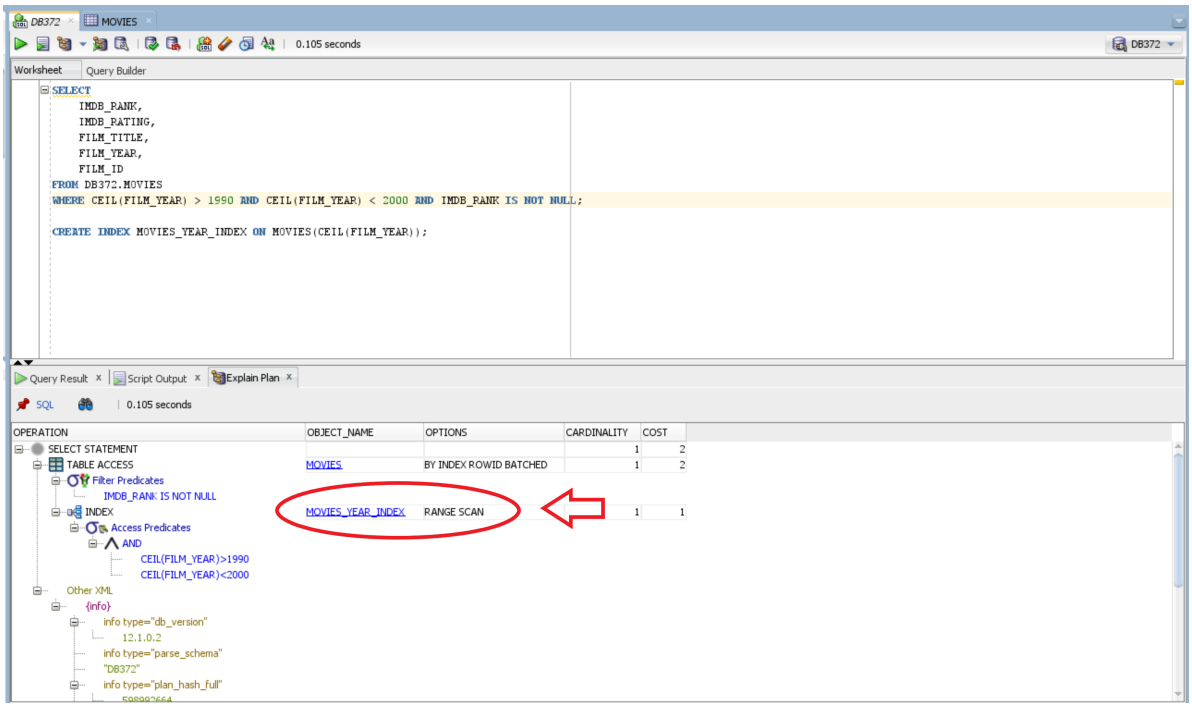

Figure 20: Execution Plan with Indexing

5.3 Parallel Execution

In this section, the CUSTOMER table is used to show how parallelism works. First of all we begin by creating a table and executing a simple query. Then we proceed to enable parallelism on the same query and execute again.

5.3.1 Step - 1: Creating a Table and Executing a Simple Query

We start by creating a new table from CUSTOMER. Then we will run a simple query with a condition.

```
1 CREATE TABLE CUSTOMER_PARALLEL AS
2 SELECT * FROM DB372.CUSTOMER;
3
4 SELECT COUNT(*)
5 FROM CUSTOMER_PARALLEL
6 WHERE CUSTOMER_SEGMENT = 'Home Office' AND REGION = 'South';
```

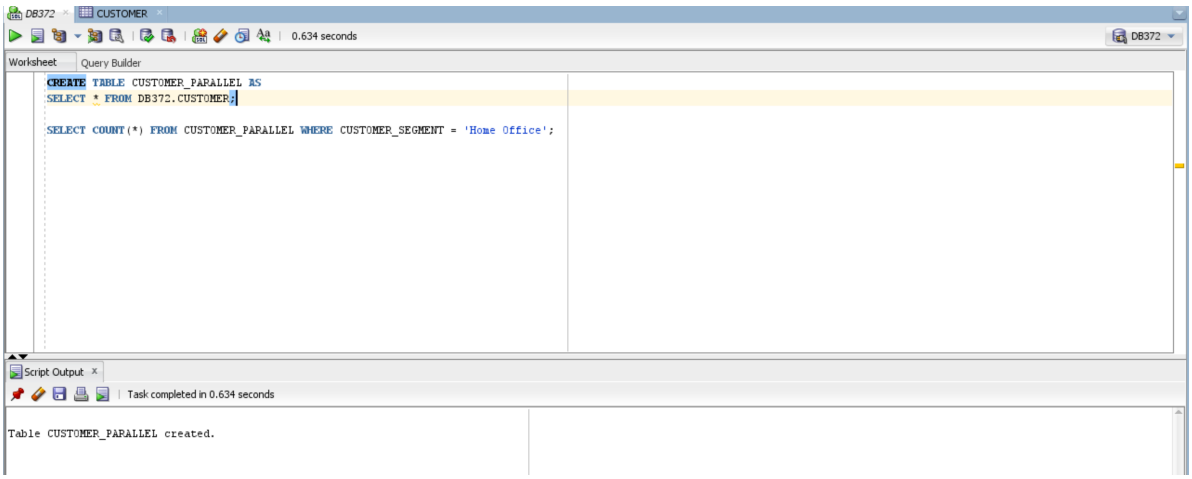

Figure 21: Table Created

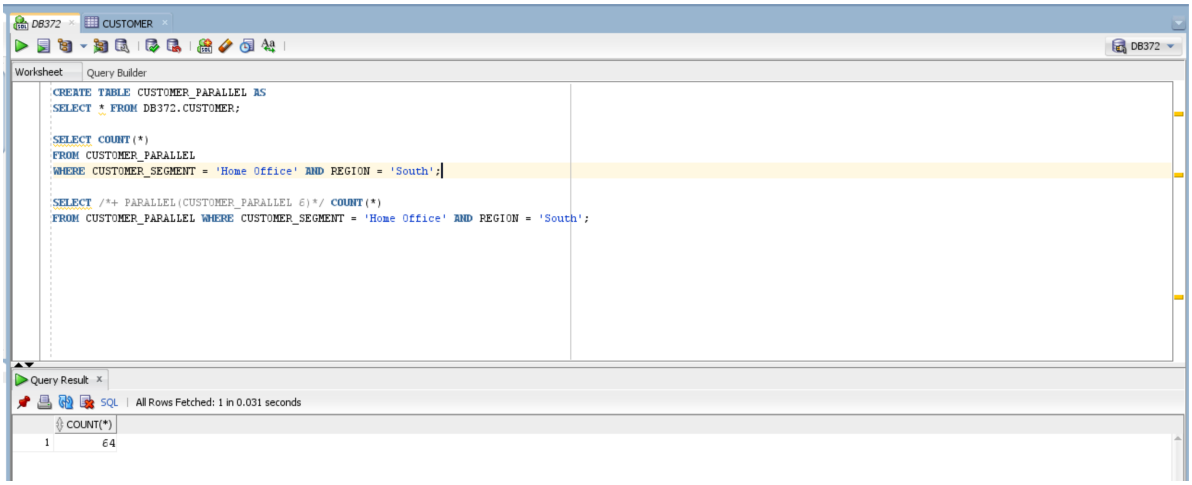

Figure 22: Query Output

5.3.2 Step - 2: Running the Same Query with Parallelism

Now we enable parallelism and run the same query again. Here we have applied 6 parallel process for this query. When the query is run once again, the output shows the value 64.

```
SELECT /*+ PARALLEL(CUSTOMER_PARALLEL 6)*/ COUNT(*)
FROM CUSTOMER_PARALLEL WHERE CUSTOMER_SEGMENT = 'Home Office'
AND REGION = 'South';
```

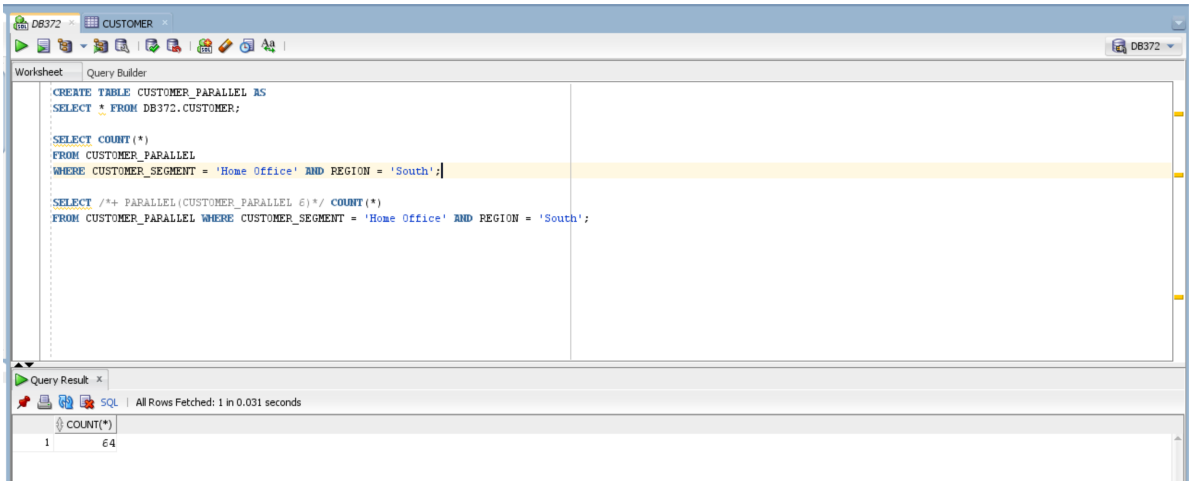

Figure 23: Query Output

5.3.3 Step - 3: Comparing the Execution Plans

Lastly, lets compare the execution plans for both the scenarios. We can clearly see the (PX) in the execution plan.

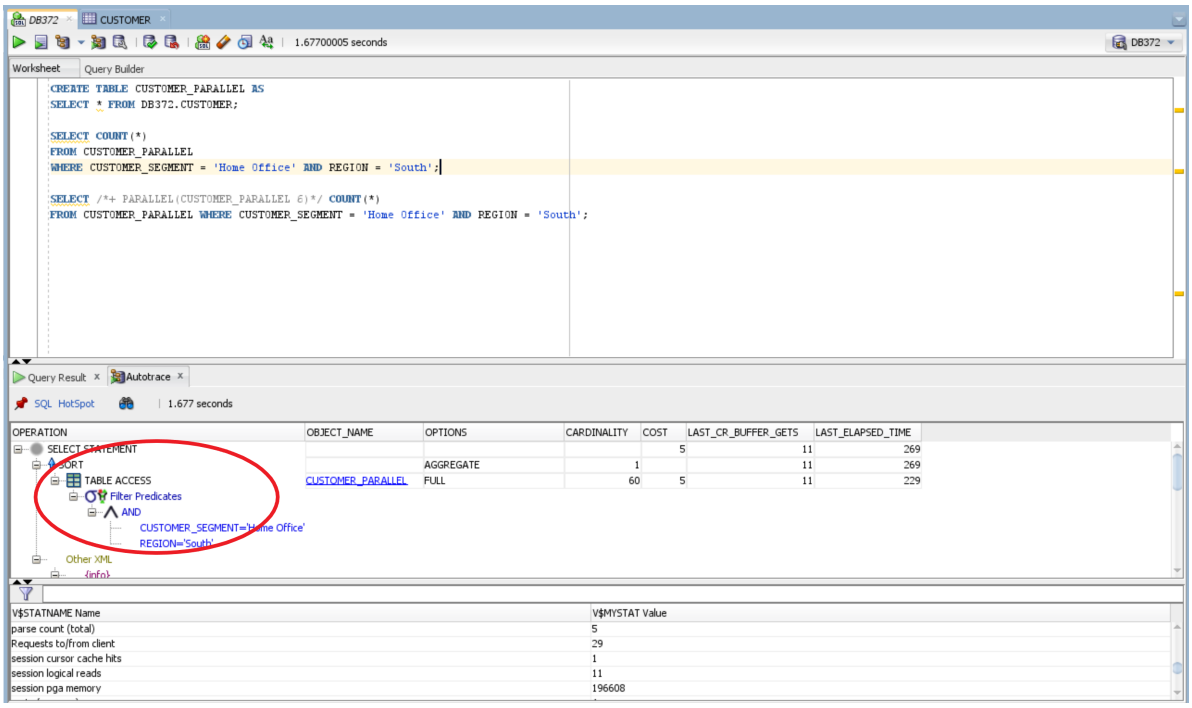

Figure 24: With PX

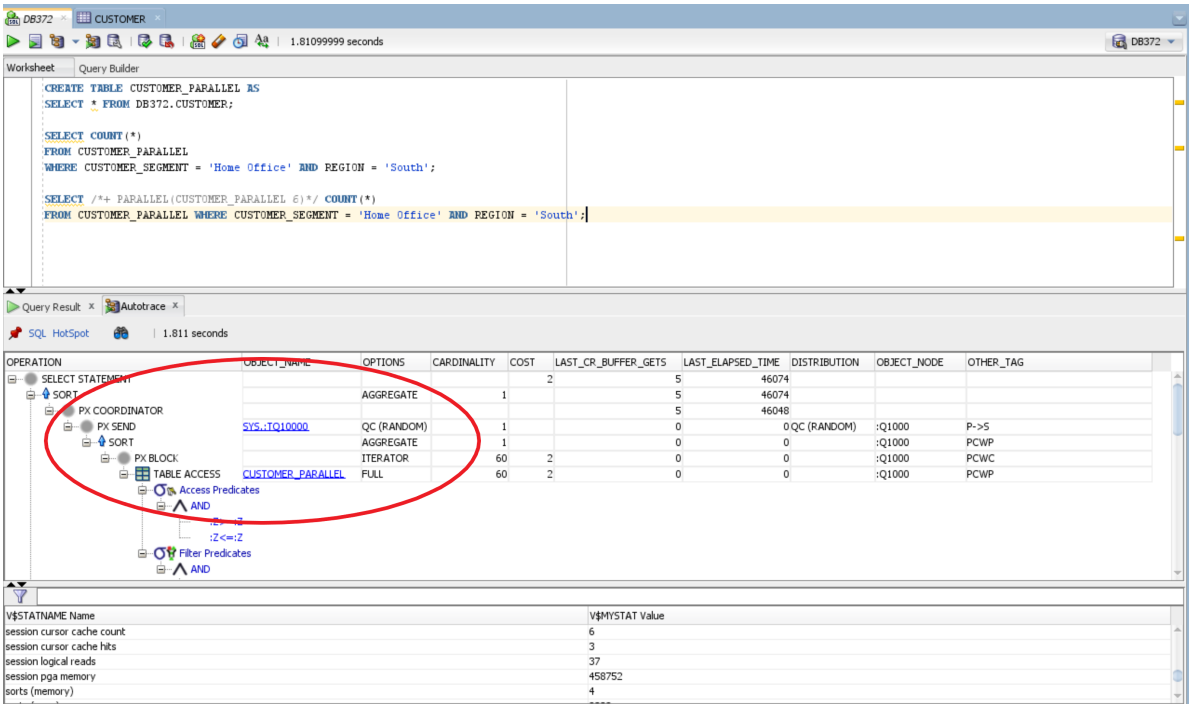

Figure 25: Without PX

5.4 Transitions in More Complex Parallel Queries

In this section, we use the MOVIES tale to show various transitions like parallel-to-parallel and parallel-to-serial using complex queries. In this example we see the best transition which is parallel-to-parallel, showing we had no bottlenecks for this processing.

5.4.1 Step - 1: Executing a Query without Parallelism

Now we run a complex query to fetch movie details within a range of IMDB ratings. We get an output with only two entries as shown below

```
SELECT
    FILM_ID,
    IMDB_RATING,
    FILM_TITLE,
    FILM_YEAR
FROM
    DB372.MOVIES
WHERE IMDB_RATING IS NOT NULL AND IMDB_RATING BETWEEN 7 AND 7.9
ORDER BY FILM_YEAR ASC;
```

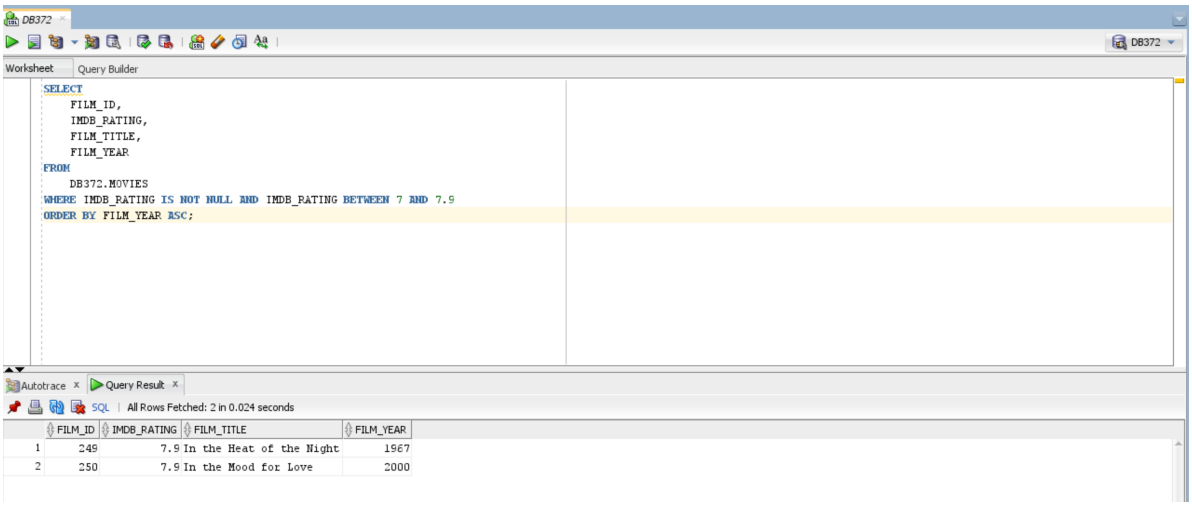

Figure 26: Without Parallelism

5.4.2 Step - 2: Execution with Parallelism and Checking the Execution Plan

Lets now check execute the same query with parallelism and check the execution plans. We check to see if there is any P-P, P-S transitions when the query executed. The result shows that there was in fact multiple transitions at the time of execution.

```
SELECT /*+ PARALLEL(MOVIES 6)*/
    FILM_ID,
    IMDB_RATING,
    FILM_TITLE,
    FILM_YEAR
FROM
    DB372.MOVIES
WHERE IMDB_RATING IS NOT NULL AND IMDB_RATING BETWEEN 7 AND 7.9
ORDER BY FILM_YEAR ASC;
```

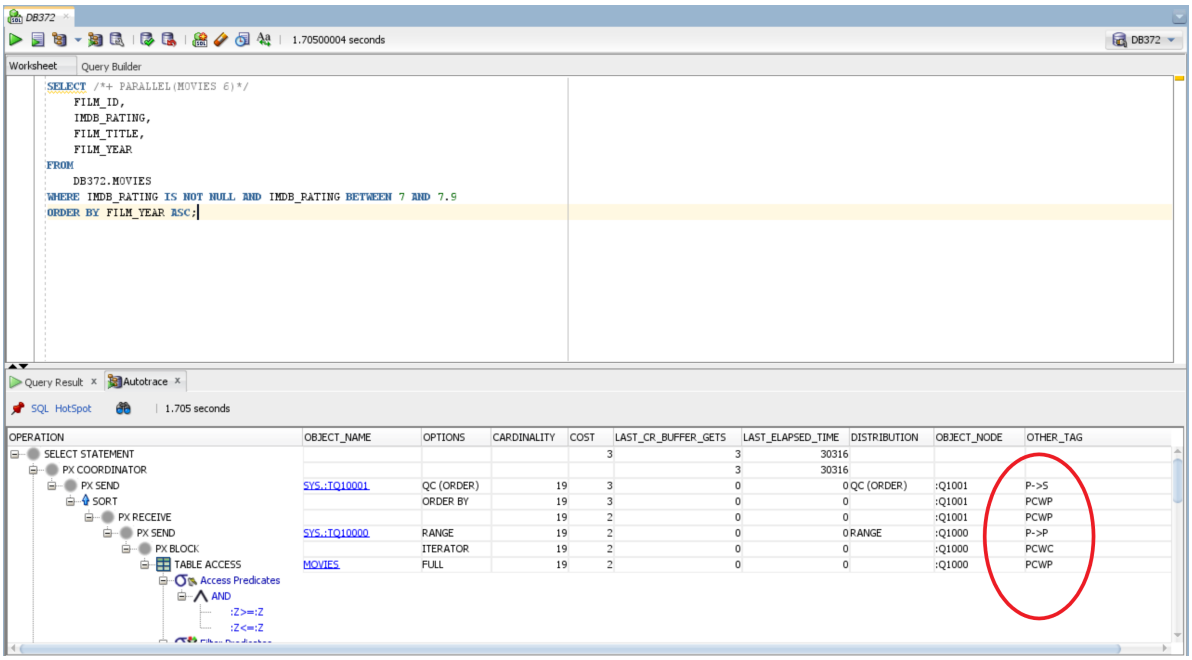

Figure 27: Execution Plan

5.5 Partitioned Tables

In this section, we show how partitioning works. We will write some interesting queries to test out how the range partition works and see the partitioned result.

5.5.1 Step - 1: Creating a New Partitioned Table

We start by creating a new table called MOVIES_RPART. We use the range partition to create this new table.

```
CREATE TABLE MOVIES_RPART (
    FILM_ID NUMBER(10,0),
    FILM_TITLE VARCHAR2(100))
PARTITION BY RANGE (FILM_ID)
    (PARTITION P100 VALUES LESS THAN (100),
     PARTITION P200 VALUES LESS THAN (200),
     PARTITION P300 VALUES LESS THAN (300),
     PARTITION P400 VALUES LESS THAN (400),
     PARTITION P500 VALUES LESS THAN (500));
```

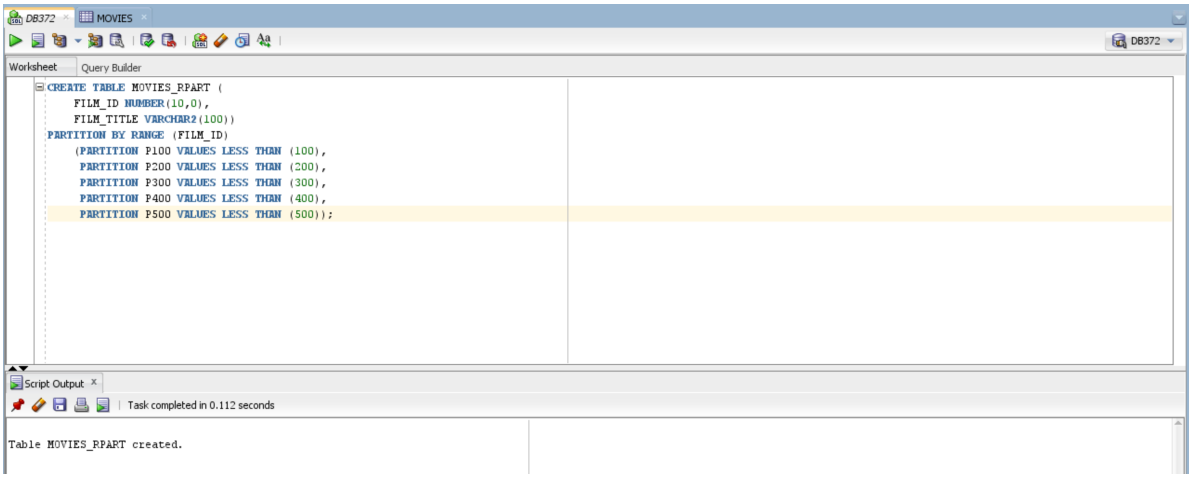

Figure 28: Table Created

5.6 Step - 2: Creating Partitioning and Inserting Data

Now we insert the data into the range partitioned table, commit and analyse for compute statistics. We have partitioned the data based on the FILM_ID in the new table with a range of 100 values in each partition.

```
1  INSERT INTO MOVIES_RPART (
2      FILM_ID,
3      FILM_TITLE)
4  SELECT
5      FILM_ID,
6      FILM_TITLE
7  FROM MOVIES;
8
9  COMMIT;
10
11 ANALYZE TABLE MOVIES_RPART COMPUTE STATISTICS;
```

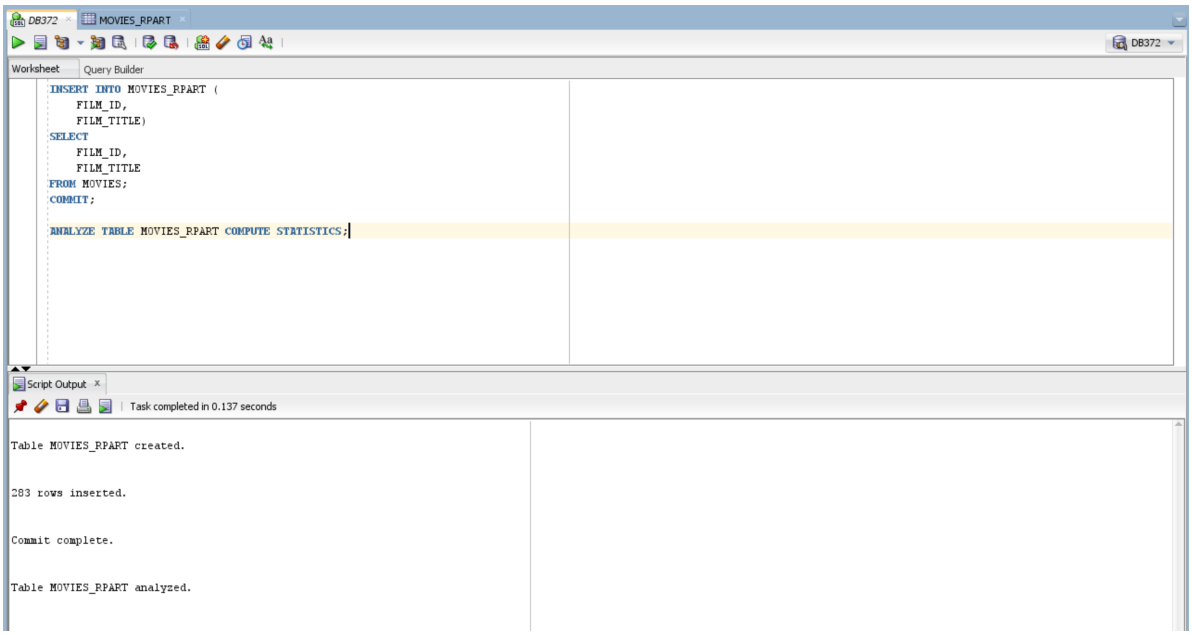

Figure 29: Query Executed

5.7 Step - 3: Inserting Data into the New Table

Here, we compare the partition tab before and after analysing to see the range partitioned results.

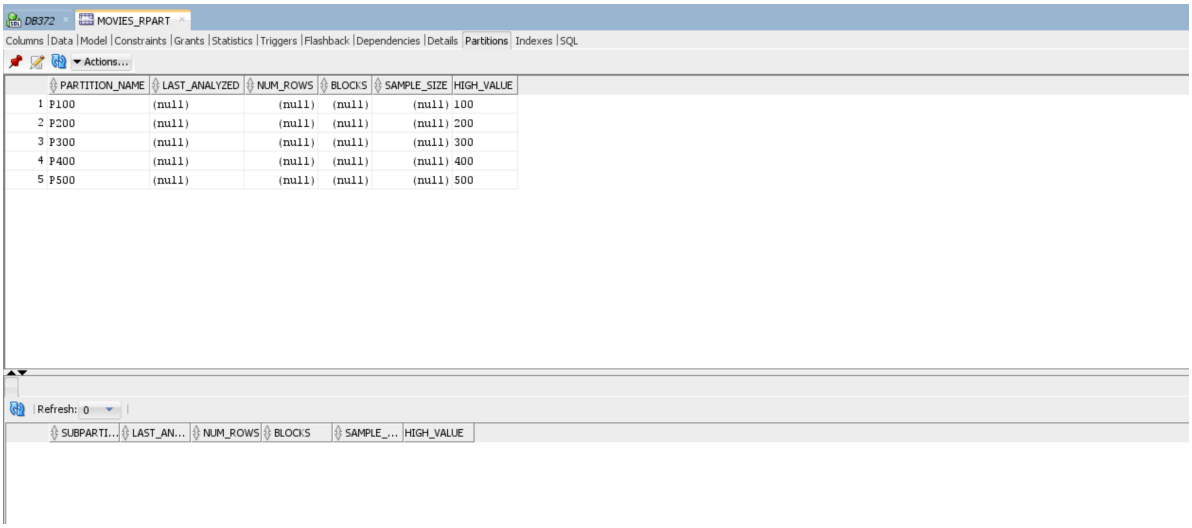

Figure 30: Before Data Insertion

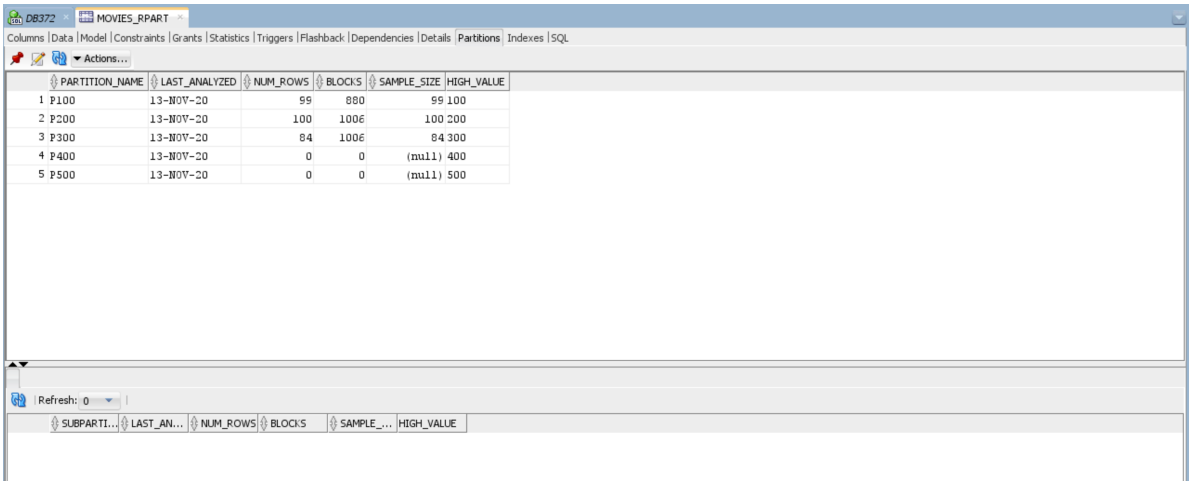

Figure 31: After Data Insertion

Finally, we see the PARTITION_START and PARTITION_STOP, which lists the partition identification numbers that define the range. We ran a simple query to check the execution plan which shows the partition range as well.

```
1 SELECT
2     FILM_ID,
3     FILM_TITLE
4 FROM MOVIES_RPART;
```

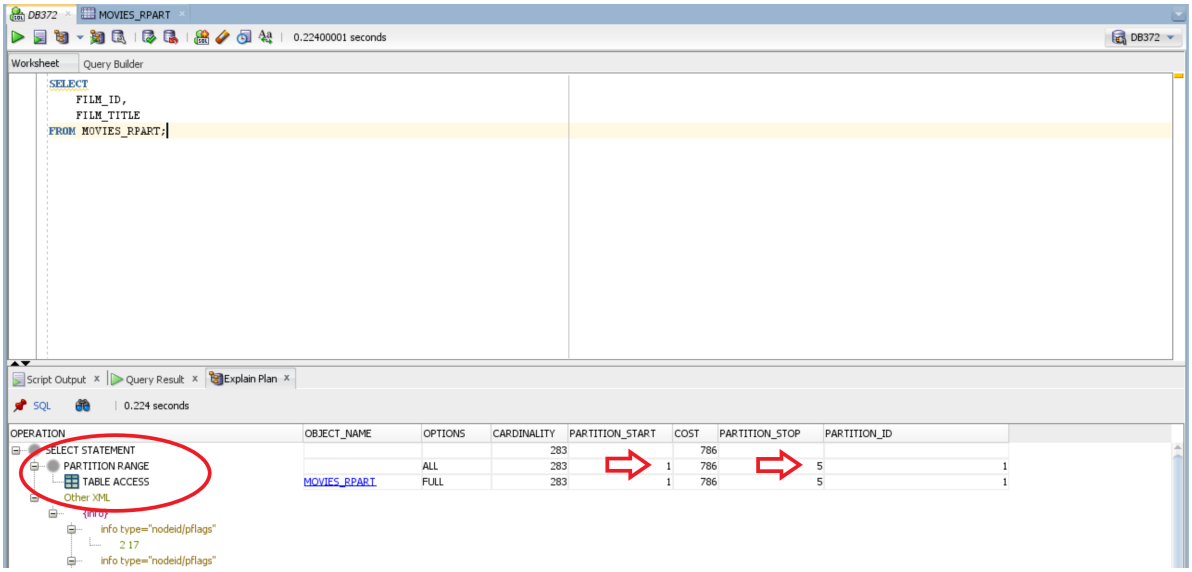

Figure 32: Execution Plan

6 Data Visualization: Regression with Rstudio

According to the difficulty of syncing the database to Rstudio, we decide to export the query result shown below in excel file. Then we decide to do the linear regression test by Rstudio with the table data shown in Fig 33.

```
SELECT
    tb.title,
    tr.avarage_raing,
    tr.num_votes
FROM
    title_basics tb
    INNER JOIN title_ratings tr
    USING (tconst)
WHERE tr.num_votes > 100
ORDER BY tr.avarage_raing DESC;
```

	TITLE	AVARAGE_RAING	NUM_VOTES
1	Call Me Kevin	9.6	135
2	Rammstein: Deutschland	9.3	646
3	Sudu Andagena Kalu Avidin	9.3	210
4	Wo men yu e de ju li	9.2	1737
5	National Theatre Live: The Lehman Trilogy	9.2	140
6	Nouba	9.1	946
7	NXT TakeOver: New York	9	242
8	Precure Miracle Universe Movie	9	114
9	Blame the Hero	8.9	117
10	Mentalno razgibavanje	8.9	107
11	Joe Finds Grace	8.8	264
12	Behind the Line: Escape to Dunkirk	8.8	764
13	NT Live: Cyrano de Bergerac	8.8	190
14	BTS Feat. Halsey: Boy With Luv	8.7	177
15	Westworld	8.7	411688
16	Baarish	8.6	1319
17	Medically Yourrs	8.6	1755

Figure 33: Example of Executed Data

The data show IMDB score, number of voter, restriction age of movies and number of location which the movie goes aboard. After that, we perform the regression model test by comparing hypothesis test of 6 different models. Denote Y = *IMDB ranking*, X_1 = *number of voter*.

```
dv2=import("Data Visualization 2.xlsx")
colnames(dv2)=tolower(make.names(colnames(dv2)))
attach(dv2)

num.rateshow=lm(avarage_raing~num_votes+show_location,data=dv2)
summary(num.rateshow)

plot(num_votes,avarage_raing, main="Scatterplot",
      xlab="Voters ", ylab="IMDB Scores", pch=19)

plot(log(num_votes),avarage_raing, main="Scatterplot",
     xlab="Voters ", ylab="IMDB Scores", pch=19)

plot(1/(num_votes),avarage_raing, main="Scatterplot",
     xlab="Voters ", ylab="IMDB Scores", pch=19)

lognum=lm(avarage_raing~num_votes+I(log(num_votes)),data=dv2)
summary(lognum)


innum=lm(avarage_raing~I(1/num_votes),data=dv2)
summary(innum)
```

Figure 34: Example of coding

6.1 $Y = \beta_0 + \beta_1 X_1$

```
Call:
lm(formula = avarage_raing ~ num_votes, data = dv)

Residuals:
    Min      1Q  Median      3Q     Max
-5.6549 -0.8549  0.1451  1.0451  3.3451

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) 6.655e+00  3.927e-02 169.485   <2e-16 ***
num_votes   1.055e-06  1.265e-06   0.834    0.404
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 1.544 on 1559 degrees of freedom
Multiple R-squared:  0.0004463, Adjusted R-squared:  -0.0001948
F-statistic: 0.6961 on 1 and 1559 DF,  p-value: 0.4042
```

Figure 35: Summary of $Y = \beta_0 + \beta_1 X_1$

According to the P-value and R-squared, they are both surprisingly ugly value. We cannot assume or identify any relationship on them. Therefore, we decide to do nonlinear model which are ln(x) model and inverse x model .

6.2 $\ln(x)$ model

```
Call:
lm(formula = avarage_raing ~ num_votes + I(log(num_votes)), data = dv2)

Residuals:
    Min      1Q  Median      3Q     Max
-4.9531 -0.7718  0.1398  0.8433  3.1460

Coefficients:
                  Estimate Std. Error t value Pr(>|t|)
(Intercept)       6.438e+00  3.076e-01  20.932   <2e-16 ***
num_votes         1.623e-06  1.416e-06   1.146    0.252
I(log(num_votes)) 3.201e-03  4.756e-02   0.067    0.946
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 1.334 on 365 degrees of freedom
Multiple R-squared:  0.006249,  Adjusted R-squared:  0.0008039
F-statistic: 1.148 on 2 and 365 DF,  p-value: 0.3185
```

Figure 36: Summary of $Y = \beta_0 + \beta_1 \ln(X_1)$

6.3 Inverse X model

```
Call:
lm(formula = avarage_raing ~ I(1/num_votes), data = dv2)

Residuals:
    Min      1Q  Median      3Q     Max
-4.9517 -0.7777  0.1228  0.8353  3.2098

Coefficients:
               Estimate Std. Error t value Pr(>|t|)
(Intercept)      6.5562     0.1057  62.044   <2e-16 ***
I(1/num_votes) -22.3990    23.4347  -0.956     0.34
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 1.335 on 366 degrees of freedom
Multiple R-squared:  0.00249,   Adjusted R-squared:  -0.0002356
F-statistic: 0.9136 on 1 and 366 DF,  p-value: 0.3398
```

Figure 37: Summary of $Y = \beta_0 + \beta_1 \frac{1}{x_1}$

6.4 Conclusion of the regression experiment

According to the results, both of the functions still contain ugly values. It means the relationship of those X_1 and Y is very hard to determine. The relationship could be very more complex or it is not related. However, we could imply that the rating of the movies might not explain in the numeric function because the human emotion is too bias.