

IFT 530

**Advanced Database Management Systems
Project Proposal**

Prof. Robert Rucker

Submitted By:

Niharika LNU

Abhilash Nataraja

Namratha Ganesh Chandrashekaraya

Priyanka Menghani

Date: 1st May 2022

SECTION - 1

INTRODUCTION

Many of us are deeply affected by movies because the combination of pictures, music, conversation, lighting, sound, and special effects may provoke strong emotions and prompt us to reflect on our lives. They may assist us in better understanding our own lives, the lives of others around us, and even the functioning of our society and culture.

When an organization decides to centralize a job or position within its operating system, it considers a number of variables before deciding whether or not to do so. Some of the most crucial questions to ask are if this change will bring value to the organization and whether it will assist in considerably decreasing operating costs. The same is true when it comes to running a movie theater. Because, even if you own a cinema, you should constantly be looking for methods to cut costs and increase profits.

The Movie management system is a set of digital solutions that lets you manage all of the operations in your cinema in a more efficient manner, decreasing the workload and simplifying all of the operations. It aids in the streamlining and automation of all activities, providing you with the ideal management system, and allowing you to maintain totalitarian management of overall cinema operations. The cinema management system will aid in keeping track of ticket sales and movie attendance when people purchase tickets.

IMPORTANCE

Growing up in India, a cinema-loving country, our love for movies began at quite an early age. The Film industry has been significantly popular for a very long time now. It is always a topic of great interest when our parents talk about the famous movies of their time and compare how the industry has evolved since then. The proposed project helps us provide the stats of these movies in the database. Changes can be made based on the reviews of the movies and be used as a criterion for screening.

There are several benefits to using a Movie Management System, including

Reduction of human error: - Because human error is unavoidable when processes are handled by employees, yet when errors do occur, they may have potentially disastrous effects on the company. Consider the ramifications of showing an incorrect trailer in front of a family film. The harm to your theater's and brand's reputation that this causes may be irreversible, and you may have to pay a high price for it. This is something which you want to avoid. So when you want to reduce the potential errors that can occur from your staff, the best possible way that you can do it is to automate the many processes that go on within the cinema. And this is exactly what the Movie Management System offers you.

Reduces the number of manual tasks: - This reduces the cost of running a cinema in a variety of ways. Because lowering the number of jobs that must be completed manually allows you to have fewer employees on hand because fewer hands are required on site. A Movie management system can handle duties including playlist creation and scheduling, content switching across several displays in the theater, and hardware monitoring. This tends to greatly minimize the strain on the employees, allowing them to

focus more on the more vital responsibilities, such as creating stronger customer connections, and so on.

Helps you to optimize your operations: A theater management system not only does the work for you, but it also provides you with analytical and statistical data on the many

processes within your cinema, giving you insight into how well everything is going. And this information may be really useful in assisting you in improving the cinema's operational capabilities and in generating new ideas for improving the way things are done. It may warn you about things you're doing incorrectly and aspects of your operations that aren't very cost-effective, as well as give you solutions to help you improve.

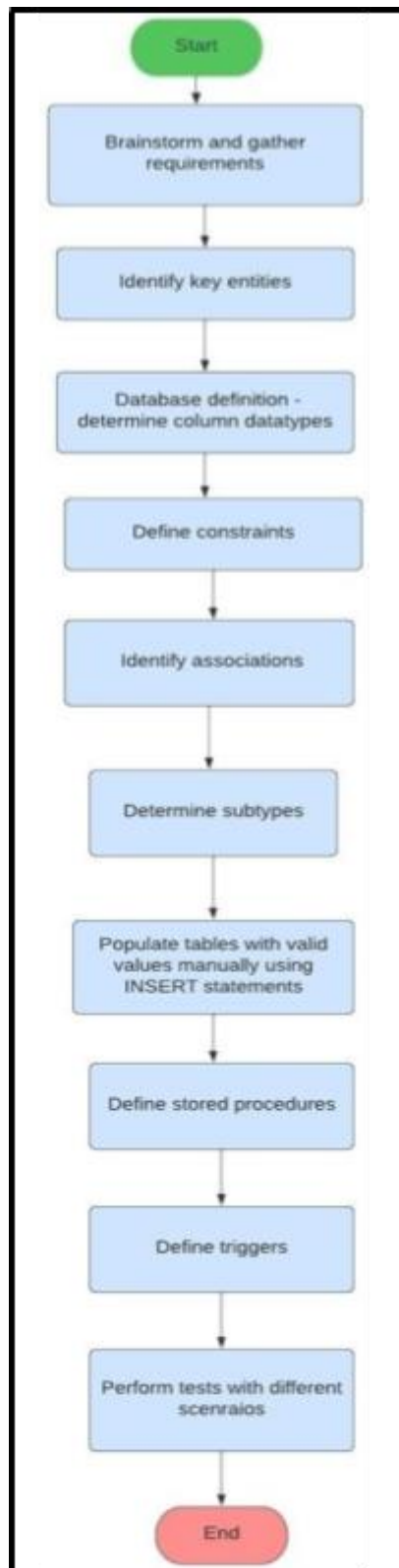
Brings about consistency in your operations: This is another crucial component that most people overlook. When you have to deal with the same processes on a daily basis, it's human nature to want to break up the monotony. As a result, you may not find the consistency you desire across all jobs. Having a Theater Management System in place helps to provide uniformity to all aspects of the cinema's operations, making it easier to deal with problems in a consistent manner. This is especially useful if you have many websites to manage. This kind of uniformity across all of your sites may assist build your brand's reputation, which is a benefit.

SECTION - 2

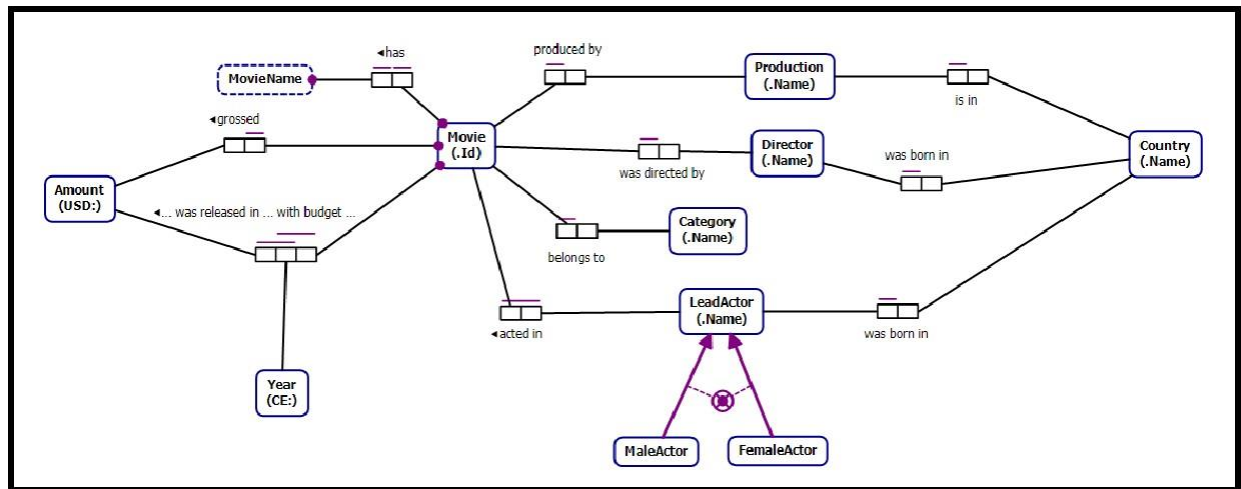
DESIGN REQUIREMENTS

Entity Requirements	Data Requirements
Can it provide information on Movie?	Movie: MovieName, ProductionName, CategoryName .
Can it provide information on Production?	Production: ProductionName, countryName
Can it provide the stats on the Lead Actor?	LeadActorActedInMovie : leadActorName, movieid
Can it check for the genre of the movie?	Movie: MovieName, ProductionName, CategoryName .
Can it provide information about the sales of the movie production wise?	Movie: MovieName, ProductionName, amount.

WORKFLOW FLOWCHART

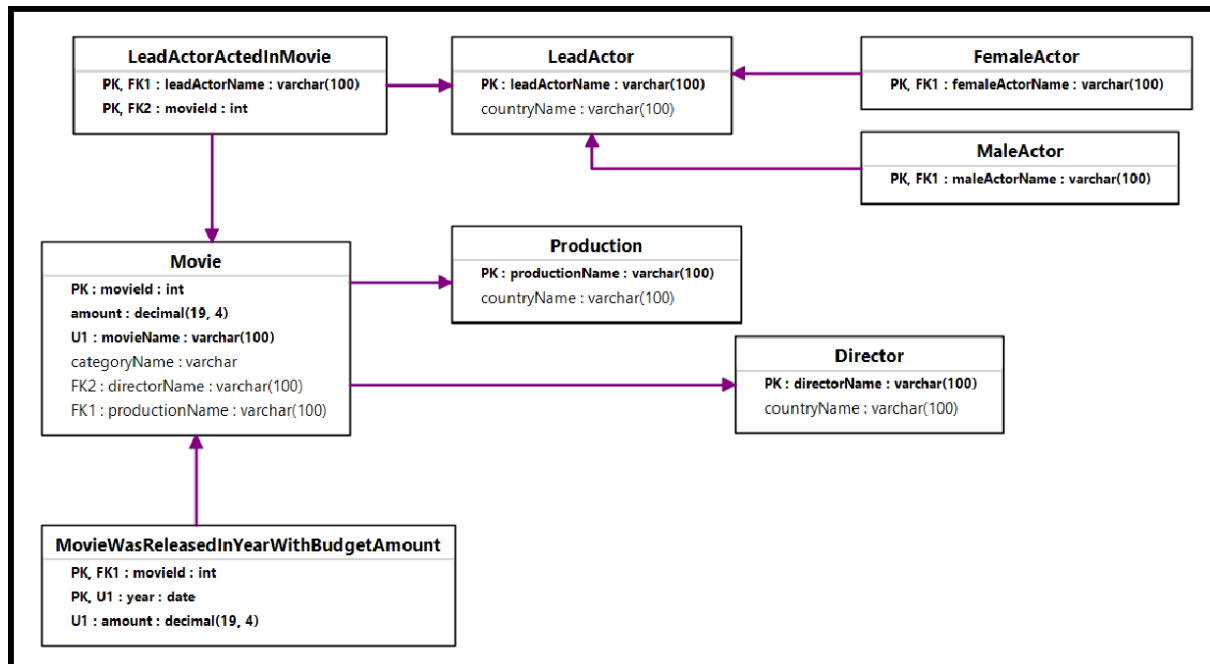


ORM DIAGRAM



SECTION - 3

RELATIONAL



SECTION - 4

SQL CODE

```
IF DB_ID('ORMModel1') IS NOT NULL DROP
DATABASE ORMModel1;
GO
CREATE DATABASE ORMModel1; GO
USE ORMModel1;
GO
CREATE TABLE Movie ( movieId int IDENTITY
    (1, 1) NOT NULL amount
    decimal(19,4) NOT NULL, movieName
    nvarchar(100) NOT NULL,
    categoryName nvarchar(max),
    directorName nvarchar(100),
    productionName nvarchar(100),
    CONSTRAINT Movie_PK PRIMARY KEY(movieId),
    CONSTRAINT Movie_UC UNIQUE(movieName)
)
GO
CREATE TABLE Production ( productionName
    nvarchar(100) NOT NULL, countryName
    nvarchar(100),
    CONSTRAINT Production_PK PRIMARY KEY(productionName)
)
GO
CREATE TABLE Director ( directorName
    nvarchar(100) NOT NULL, countryName
    nvarchar(100),
    CONSTRAINT Director_PK PRIMARY KEY(directorName)
)
GO
CREATE TABLE LeadActor ( leadActorName
    nvarchar(100) NOT NULL, countryName
    nvarchar(100),
    CONSTRAINT LeadActor_PK PRIMARY KEY(leadActorName)
```

```
CREATE TABLE LeadActorActedInMovie (  
    leadActorName nvarchar(100) NOT NULL,  
    movieId int NOT NULL,  
    CONSTRAINT LeadActorActedInMovie_PK PRIMARY KEY(movieId, leadActorName)  
)  
GO
```

```
CREATE TABLE MaleActor (  
    maleActorName  
        nvarchar(100) NOT NULL,  
    CONSTRAINT MaleActor_PK PRIMARY KEY(maleActorName)  
)  
GO
```

```
CREATE TABLE FemaleActor (  
    femaleActorName  
        nvarchar(100) NOT NULL,  
    CONSTRAINT FemaleActor_PK PRIMARY KEY(femaleActorName)  
)  
GO
```

```
CREATE TABLE MovieWasReleasedInYearWithBudgetAmount  
    (  
        movieId int NOT NULL, "year" date NOT NULL,  
        amount decimal(19,4) NOT NULL,  
        CONSTRAINT MovieWasReleasedInYearWithBudgetAmount_PK PRIMARY  
KEY(movieId, "year"),  
        CONSTRAINT MovieWasReleasedInYearWithBudgetAmount_UC UNIQUE("year",  
amount)  
    )  
GO
```

```
ALTER TABLE Movie ADD CONSTRAINT Movie_FK1 FOREIGN KEY (productionName)  
REFERENCES Production (productionName) ON DELETE NO ACTION ON UPDATE NO  
ACTION  
GO
```

```
ALTER TABLE Movie ADD CONSTRAINT Movie_FK2 FOREIGN KEY (directorName) REFERENCES  
Director (directorName) ON DELETE NO ACTION ON UPDATE NO ACTION GO  
ALTER TABLE LeadActorActedInMovie ADD CONSTRAINT LeadActorActedInMovie_FK1 FOREIGN
```

```
KEY (leadActorName) REFERENCES LeadActor (leadActorName) ON DELETE NO ACTION ON  
UPDATE NO ACTION  
GO
```

```
ALTER TABLE LeadActorActedInMovie ADD CONSTRAINT LeadActorActedInMovie_FK2  
FOREIGN KEY (movieId) REFERENCES Movie (movieId) ON DELETE NO ACTION ON UPDATE  
NO ACTION  
GO
```

```
ALTER TABLE MaleActor ADD CONSTRAINT MaleActor_FK FOREIGN KEY  
(maleActorName) REFERENCES LeadActor (leadActorName) ON DELETE NO ACTION ON  
UPDATE NO ACTION  
GO
```

```
ALTER TABLE FemaleActor ADD CONSTRAINT FemaleActor_FK FOREIGN KEY (femaleActorName)  
REFERENCES LeadActor (leadActorName) ON DELETE NO ACTION ON UPDATE NO ACTION  
GO
```

```
ALTER TABLE MovieWasReleasedInYearWithBudgetAmount ADD CONSTRAINT  
MovieWasReleasedInYearWithBudgetAmount_FK FOREIGN KEY (movieId) REFERENCES Movie  
(movieId) ON DELETE NO ACTION ON UPDATE NO ACTION  
GO  
GO
```

DATA INSERTION

SET IDENTITY_INSERT Movie ON

SET IDENTITY_INSERT MovieWasReleasedInYearWithBudgetAmount ON insert

into Director(directorName,countryName) values

('Frank Darabont','United States'),

('Francis Ford Coppola','United States'),

('Christopher Nolan','United Kingdom'),

('Bong Joon Ho','South Korea'), ('Ramesh

Sippy','India');

insert into Production(productionName,countryName) values

('Hollywood Rental Company','United States'), ('Paramount','United

States'),

('Warner Bros','United States'),

('Barunson E&A','South Korea'),

('Sippy Films Pvt Ltd','India');

insert into Movie(movieId,amount,movieName,categoryName,directorName,productionName)

values (001,28340000,'The Shawshank Redemption','Drama','Frank Darabont','Hollywood Rental Company'),

(002,134970000,'The Godfather','Crime','Francis Ford Coppola','Paramount'),

(003,534860000,'The Dark Knight','Action','Christopher Nolan','Warner Bros'),

(004,292580000,'Inception','Scifi','Christopher Nolan','Warner Bros'),

(005,53370000,'Parasite','Comedy','Bong Joon Ho','Barunson E&A'),

(006,39150000,'Sholay','Drama','Ramesh Sippy','Sippy Films Pvt Ltd');

insert into LeadActor(leadActorName,countryName) values

('Tim Robbins','United States'),

('Marlon Brando','United States'),

('Christian Bale','United Kingdom'),

('Leonardo Dicaprio','United States'),

('Kang-ho Song','South Korea'),

('Amitabh Bachchan','India');

insert into MaleActor(maleActorName) values

('Tim Robbins'),

('Marlon Brando'),

('Christian Bale'),

('Leonardo Dicaprio'), ('Kang-

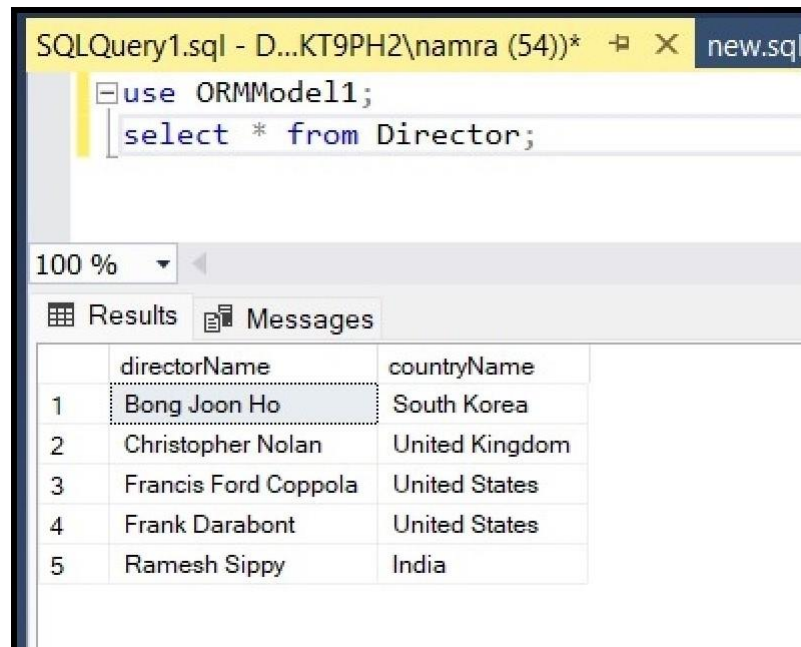
ho Song'),

('Amitabh Bachchan');

```
insert into LeadActorActedInMovie(leadActorName,movieId) values ('Tim
Robbins',001),
('Marlon Brando',002),
('Christian Bale',003),
('Leonardo Dicaprio',004), ('Kang-
ho Song',005),
('Amitabh Bachchan',006); insert into
MovieWasReleasedInYearWithBudgetAmount(movieId,year,amount) values
(001,'1994',25000000),
(002,'1972',6000000),
(003,'2008',185000000),
(004,'2010',160000000),
(005,'2019',11400000),
(006,'1975',75000);
insert into MovieHasRating(movieId, ratingNr)
values(001, 1),
(002, 2),
(003, 5),
(004, 4),
(005, 1),
(006, 5);
```

SAMPLE DATA

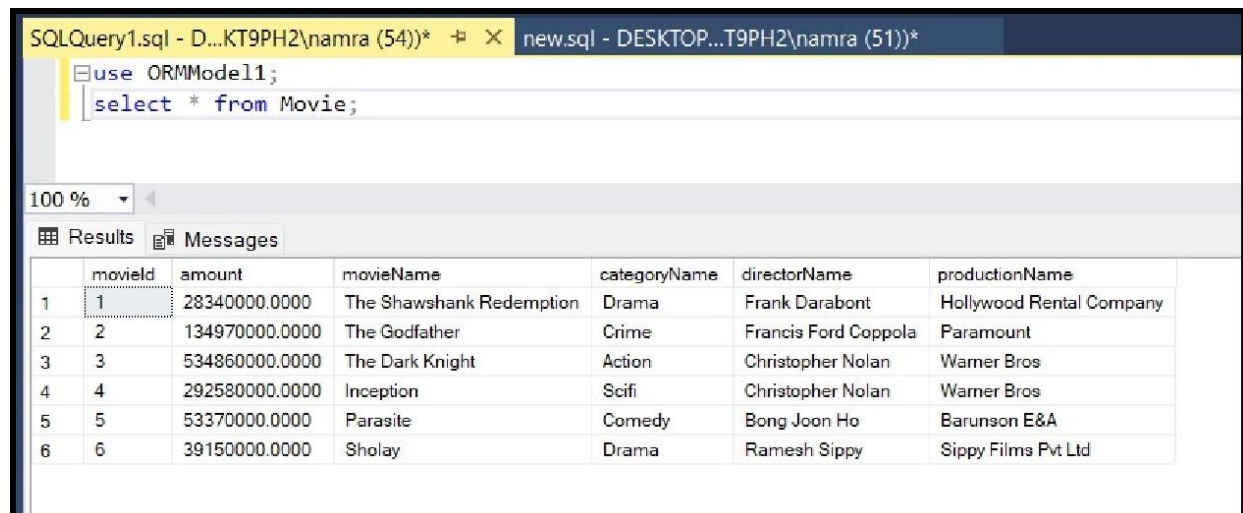
Table: Director



The screenshot shows a SQL query window with the following text: `use ORMModel1;` and `select * from Director;`. Below the query, the 'Results' tab is active, displaying a table with 5 rows and 2 columns: `directorName` and `countryName`. The data is as follows:

	directorName	countryName
1	Bong Joon Ho	South Korea
2	Christopher Nolan	United Kingdom
3	Francis Ford Coppola	United States
4	Frank Darabont	United States
5	Ramesh Sippy	India

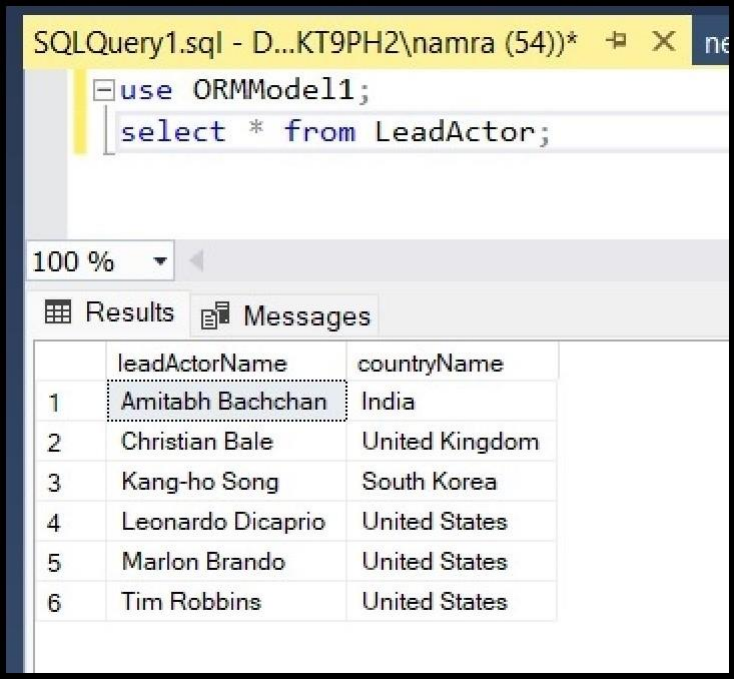
Table: Movie



The screenshot shows a SQL query window with the following text: `use ORMModel1;` and `select * from Movie;`. Below the query, the 'Results' tab is active, displaying a table with 6 rows and 7 columns: `movieId`, `amount`, `movieName`, `categoryName`, `directorName`, and `productionName`. The data is as follows:

	movieId	amount	movieName	categoryName	directorName	productionName
1	1	28340000.0000	The Shawshank Redemption	Drama	Frank Darabont	Hollywood Rental Company
2	2	134970000.0000	The Godfather	Crime	Francis Ford Coppola	Paramount
3	3	534860000.0000	The Dark Knight	Action	Christopher Nolan	Warner Bros
4	4	292580000.0000	Inception	Scifi	Christopher Nolan	Warner Bros
5	5	53370000.0000	Parasite	Comedy	Bong Joon Ho	Barunson E&A
6	6	39150000.0000	Sholay	Drama	Ramesh Sippy	Sippy Films Pvt Ltd

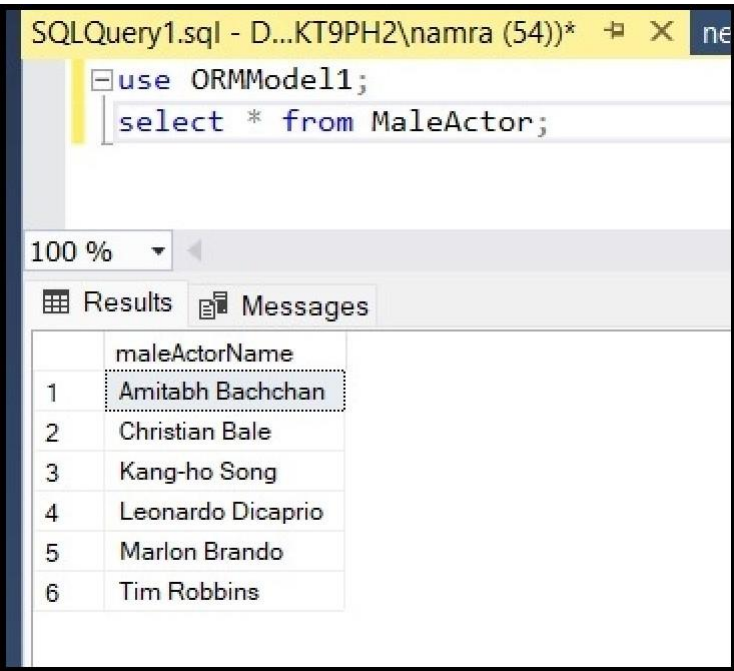
Table: LeadActor



The screenshot shows a SQL query window titled "SQLQuery1.sql - D...KT9PH2\namra (54))*". The query is: `use ORMModel1;` followed by `select * from LeadActor;`. Below the query, the "Results" tab is active, displaying a table with two columns: "leadActorName" and "countryName". The table contains six rows of data, with the first row highlighted.

	leadActorName	countryName
1	Amitabh Bachchan	India
2	Christian Bale	United Kingdom
3	Kang-ho Song	South Korea
4	Leonardo Dicaprio	United States
5	Marlon Brando	United States
6	Tim Robbins	United States

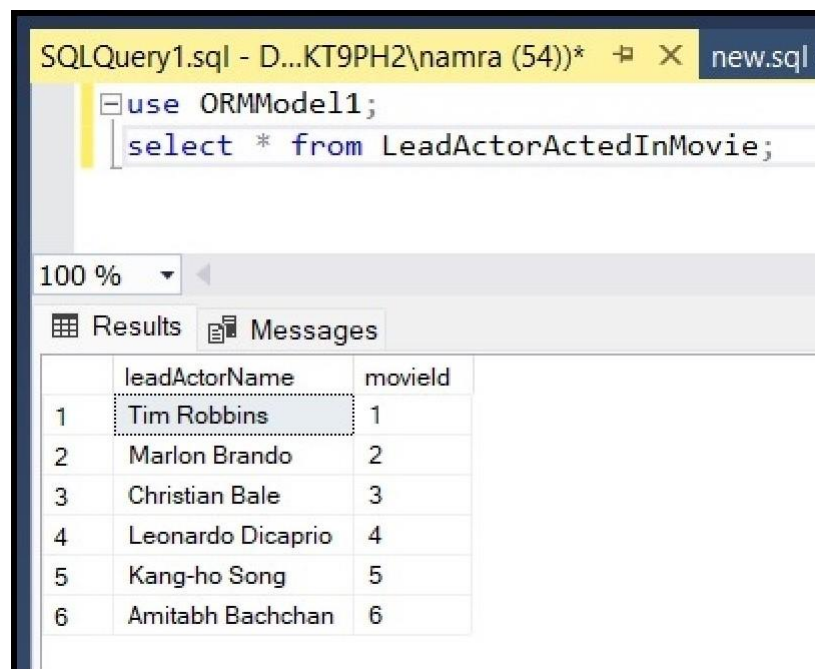
Table: MaleActor



The screenshot shows a SQL query window titled "SQLQuery1.sql - D...KT9PH2\namra (54))*". The query is: `use ORMModel1;` followed by `select * from MaleActor;`. Below the query, the "Results" tab is active, displaying a table with one column: "maleActorName". The table contains six rows of data, with the first row highlighted.

	maleActorName
1	Amitabh Bachchan
2	Christian Bale
3	Kang-ho Song
4	Leonardo Dicaprio
5	Marlon Brando
6	Tim Robbins

Table: LeadActorActedInMovie



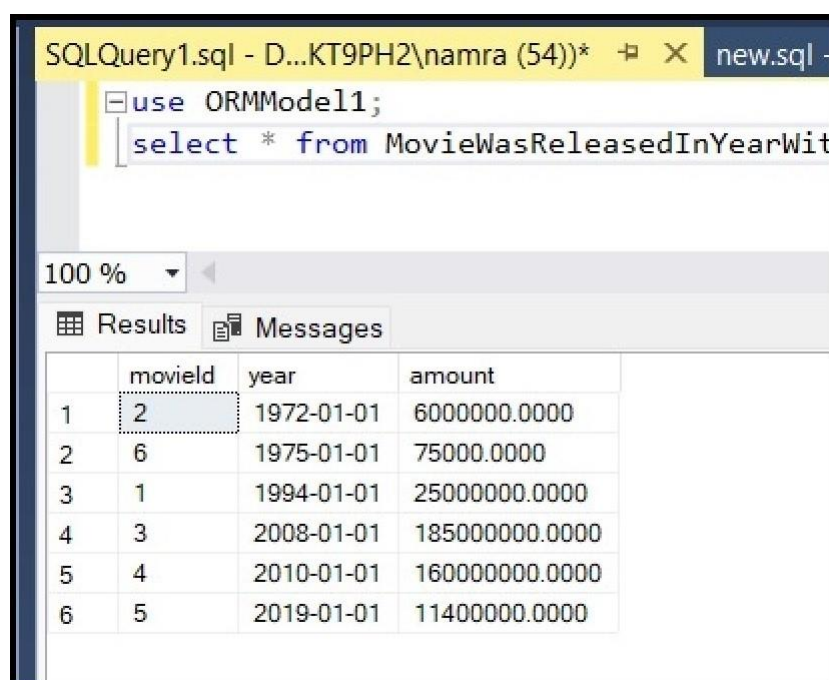
The screenshot shows a SQL query window with the following text:

```
SQLQuery1.sql - D...KT9PH2\namra (54))* X new.sql -
use ORMModel1;
select * from LeadActorActedInMovie;
```

Below the query window, the 'Results' tab is active, displaying a table with 6 rows and 2 columns: leadActorName and movieId.

	leadActorName	movieId
1	Tim Robbins	1
2	Marlon Brando	2
3	Christian Bale	3
4	Leonardo Dicaprio	4
5	Kang-ho Song	5
6	Amitabh Bachchan	6

Table: MovieWasReleasedInYearwithSales



The screenshot shows a SQL query window with the following text:

```
SQLQuery1.sql - D...KT9PH2\namra (54))* X new.sql -
use ORMModel1;
select * from MovieWasReleasedInYearWit
```

Below the query window, the 'Results' tab is active, displaying a table with 6 rows and 4 columns: movieId, year, and amount.

	movieId	year	amount
1	2	1972-01-01	6000000.0000
2	6	1975-01-01	75000.0000
3	1	1994-01-01	25000000.0000
4	3	2008-01-01	185000000.0000
5	4	2010-01-01	160000000.0000
6	5	2019-01-01	11400000.0000

Table: Production

SQLQuery1.sql - D...KT9PH2\namra (54))* ✕ new.sql -

```
use ORMModel1;  
select * from Production;
```

100 %

Results Messages

	productionName	countryName
1	Barunson E&A	South Korea
2	Hollywood Rental Company	United States
3	Paramount	United States
4	Sippy Films Pvt Ltd	India
5	Warner Bros	United States

SECTION – 5

STORED PROCEDURES

Between the user interface and the database, a stored procedure adds an important layer of security. Because end users may enter or edit data but do not develop processes, it promotes security through data access restrictions.

We have used 4 stored procedures:

- MovieRating
- Movie_LeadActor_Details
- Movie_Details
- Movie_Details123

Stored Procedure 1 : MovieRating

We have developed a parameterised stored procedure which takes the Rating value as a parameter and returns the details of the movie based on the value passed.

Code:

Create PROC MovieRating

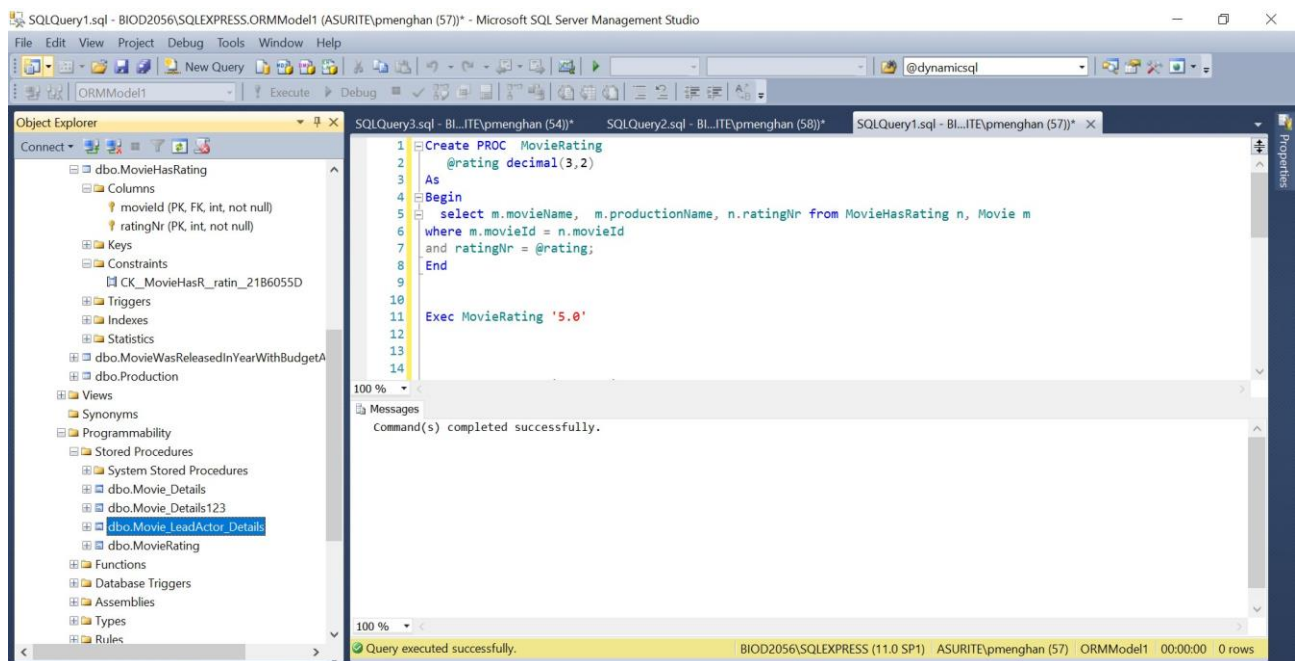
 @rating decimal(3,2)

As

Begin

select m.movieName, m.productionName, n.ratingNr from MovieHasRating n, Movie m where m.movieId = n.movieId and ratingNr = @rating;

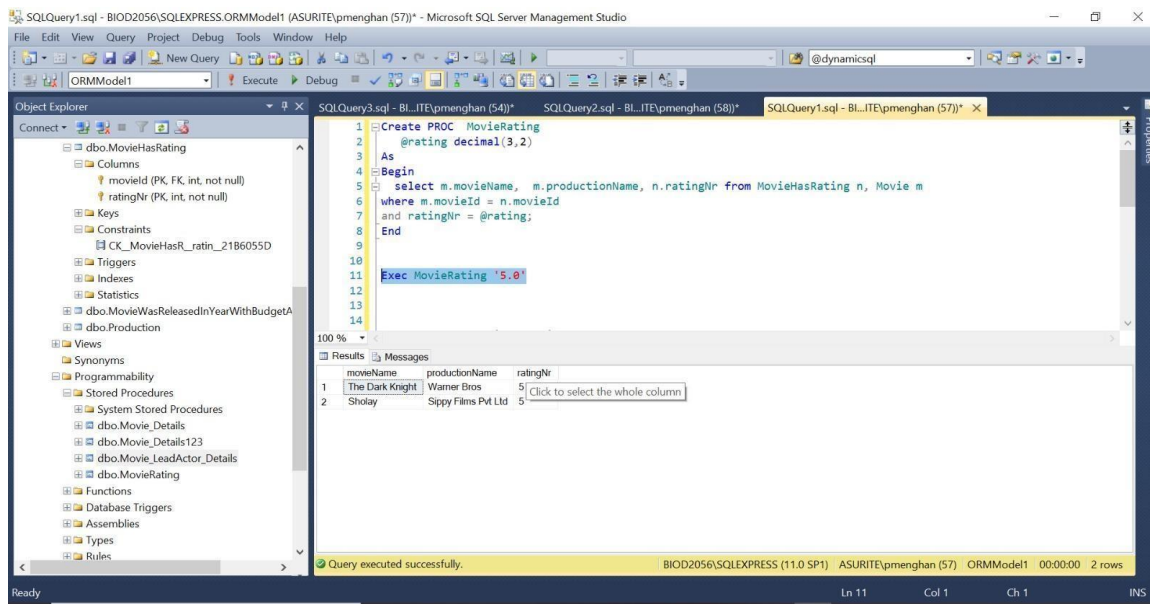
End



Exec MovieRating '5.0'

Output:

Here a value of 5.0 was passed as rating and movies with rating 5 were returned.



Stored Procedure 2 : Movie_LeadActor_Details

We have developed a stored procedure to display the movie and the movie's lead actor details.

CODE:

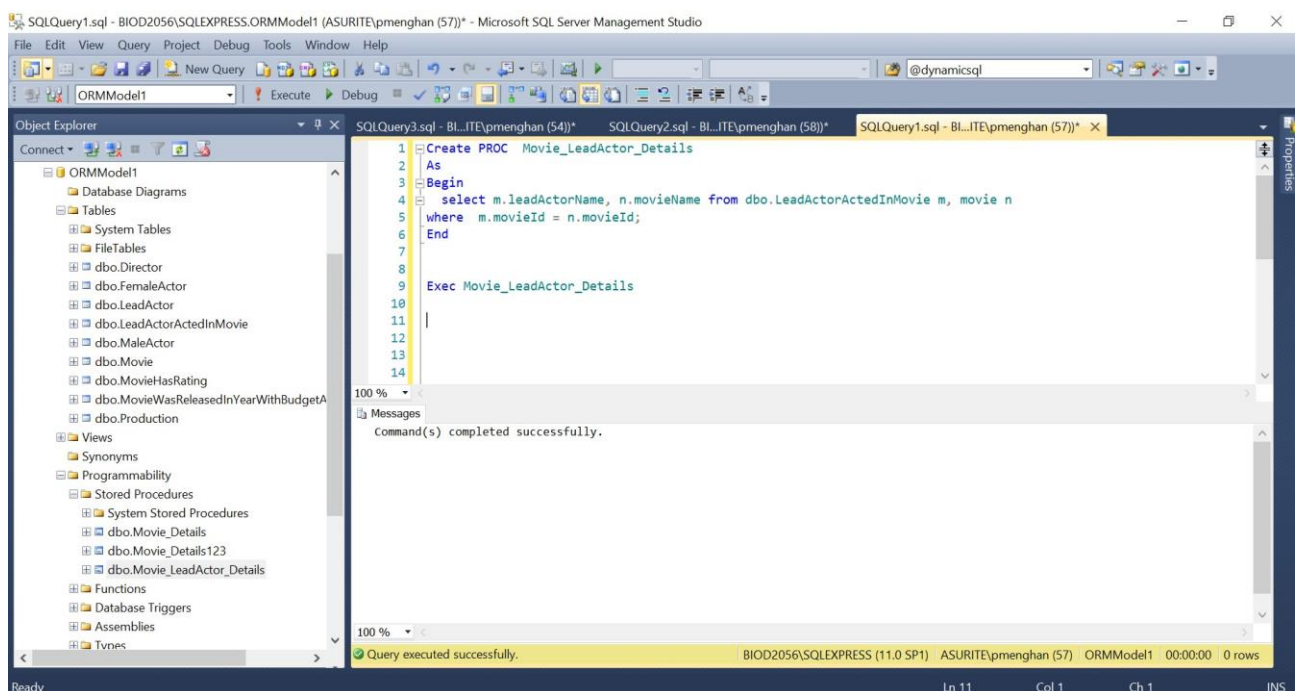
Create PROC Movie_LeadActor_Details

As

Begin

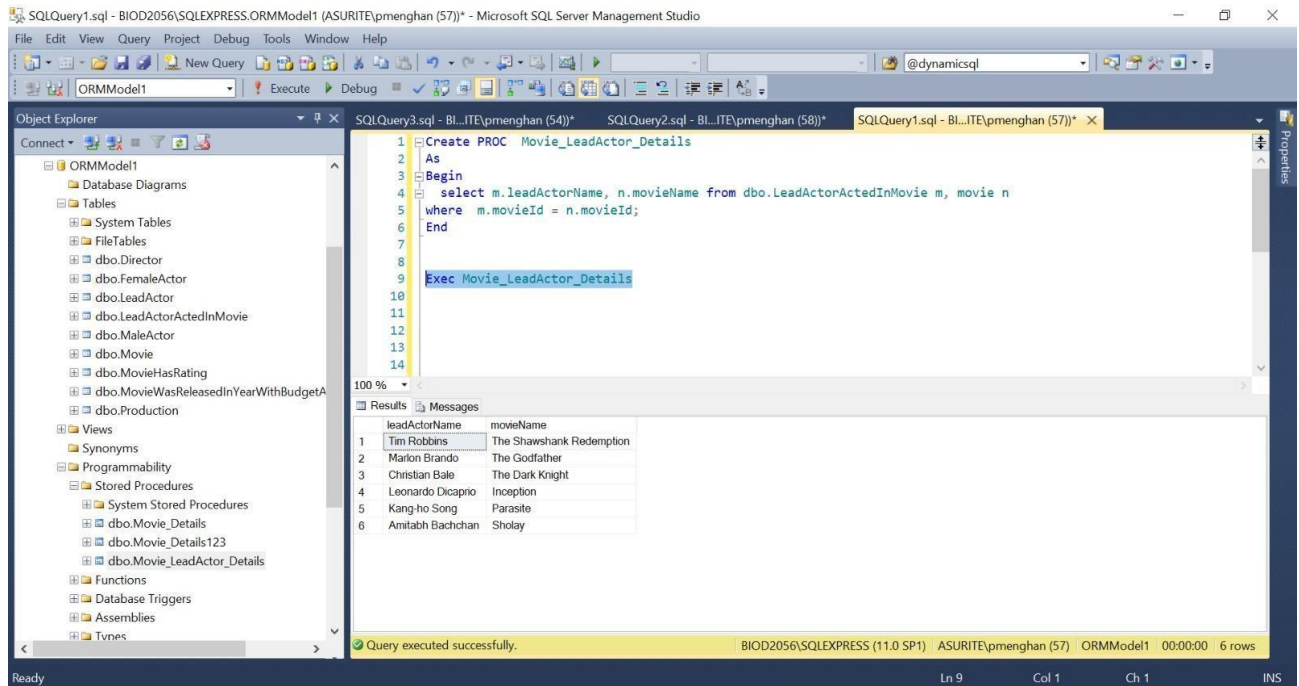
select m.leadActorName, n.movieName from dbo.LeadActorActedInMovie m , movie n
where m.movieId = n.movieId

End



Exec Movie_LeadActor_Details

Output:



Stored Procedure 3 : Movie_Details

We have developed a stored procedure that returns some more details about movies. It displays the movie name along with its category and year released.

CODE:

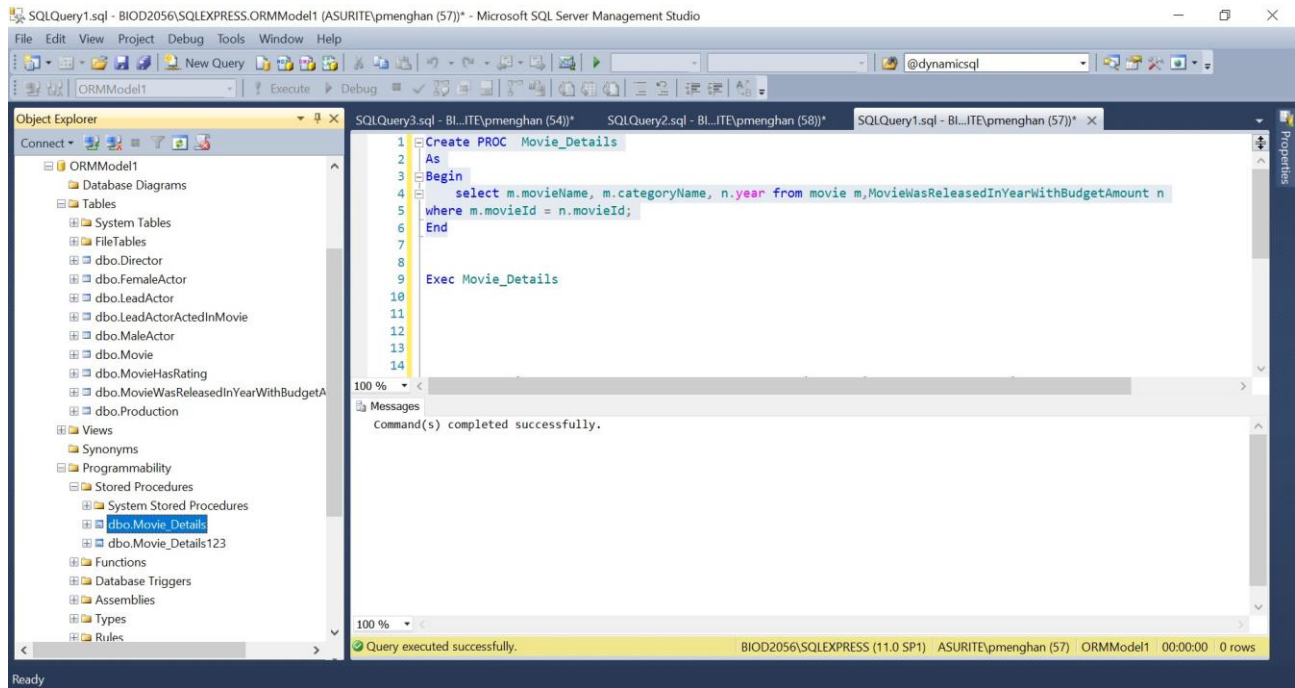
Create PROC Movie_Details

As

Begin

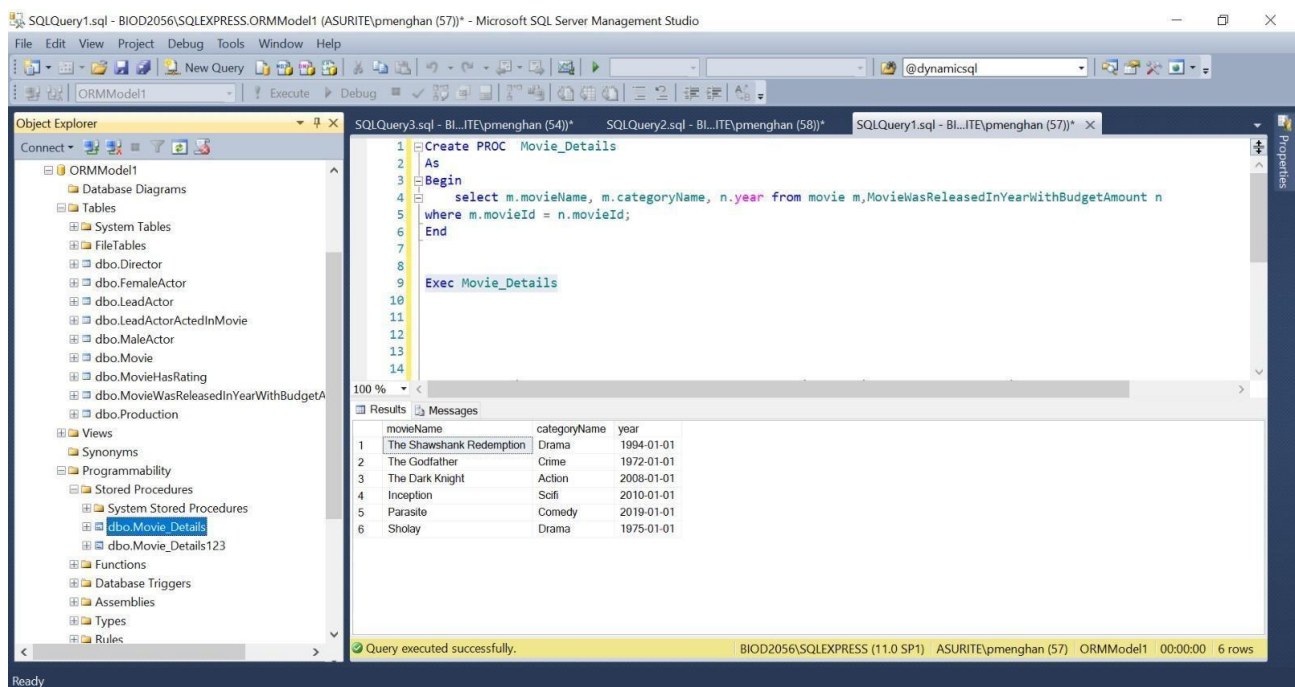
select m.movieName, m.categoryName, n.year from movie
m, MovieWasReleasedInYearWithBudgetAmount n where
m.movieId = n.movieId

End



Exec Movie_Details

Output:



Stored Procedure 4

We had developed this stored procedure in the last phase.

This is to show all of the movie data. Here we used a simple non-parameterized stored procedure. These details may then be exported to Excel or used to create any other report or dashboard.

CODE:

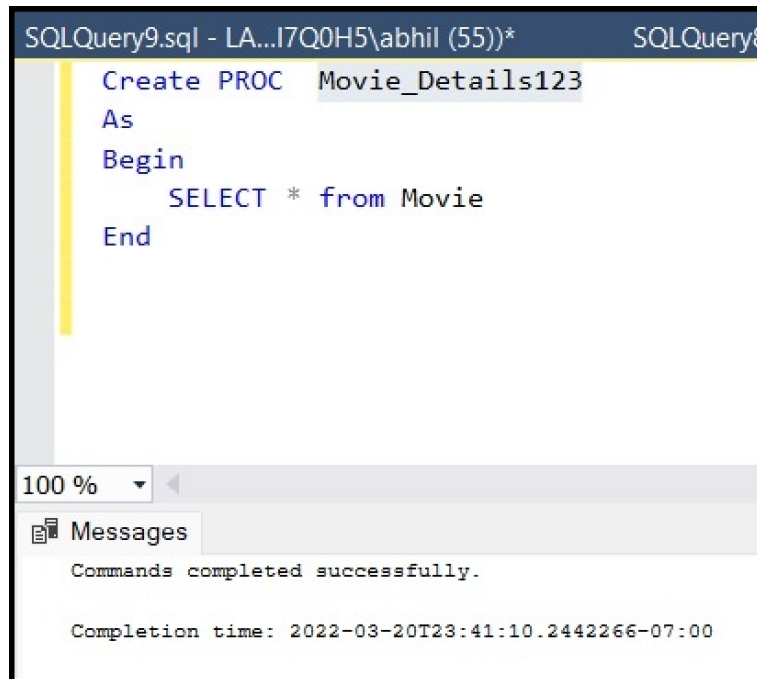
Create PROC Movie_Details123

As

Begin

 SELECT * from Movie

End



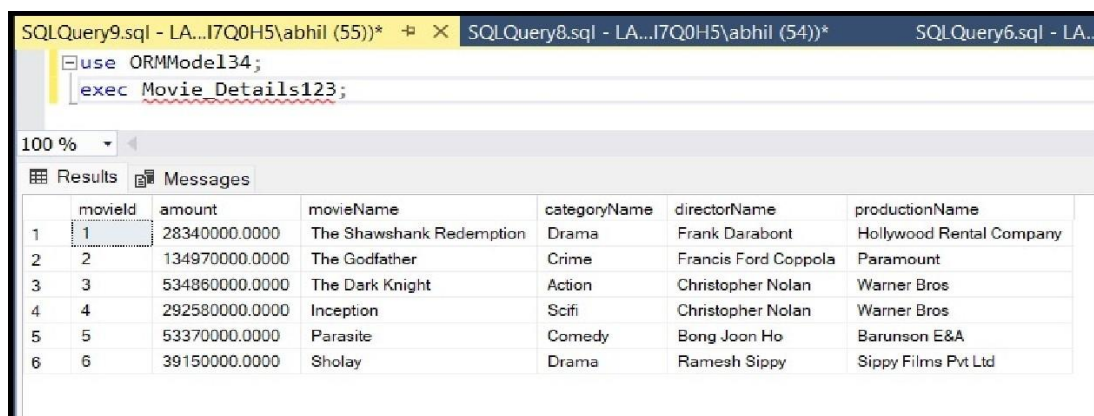
The screenshot shows a SQL Query Editor window titled 'SQLQuery9.sql - LA...I7Q0H5\abhil (55))*'. The editor contains the following SQL code:

```
Create PROC Movie_Details123
As
Begin
    SELECT * from Movie
End
```

Below the editor, the 'Messages' tab is selected, displaying the message: 'Commands completed successfully.' and 'Completion time: 2022-03-20T23:41:10.2442266-07:00'.

Exec Movie_Details123

Output:



The screenshot shows a SQL Query Editor window with two tabs: 'SQLQuery9.sql - LA...I7Q0H5\abhil (55))*' and 'SQLQuery8.sql - LA...I7Q0H5\abhil (54))*'. The 'SQLQuery9.sql' tab is active, showing the following SQL code:

```
use ORMModel134;
exec Movie_Details123;
```

Below the editor, the 'Results' tab is selected, displaying a table with 6 rows and 7 columns. The columns are: 'movielid', 'amount', 'movieName', 'categoryName', 'directorName', and 'productionName'. The data is as follows:

	movielid	amount	movieName	categoryName	directorName	productionName
1	1	28340000.0000	The Shawshank Redemption	Drama	Frank Darabont	Hollywood Rental Company
2	2	134970000.0000	The Godfather	Crime	Francis Ford Coppola	Paramount
3	3	534860000.0000	The Dark Knight	Action	Christopher Nolan	Warner Bros
4	4	292580000.0000	Inception	Scifi	Christopher Nolan	Warner Bros
5	5	53370000.0000	Parasite	Comedy	Bong Joon Ho	Barunson E&A
6	6	39150000.0000	Sholay	Drama	Ramesh Sippy	Sippy Films Pvt Ltd

TRIGGERS

Triggers allow you to validate the data's integrity. When a change occurs in the database, the triggers might make changes to the whole database. Triggers aid in keeping the User Interface light. Instead of repeating the same function call throughout the program, you may use a trigger to have it run.

We have used 2 triggers:

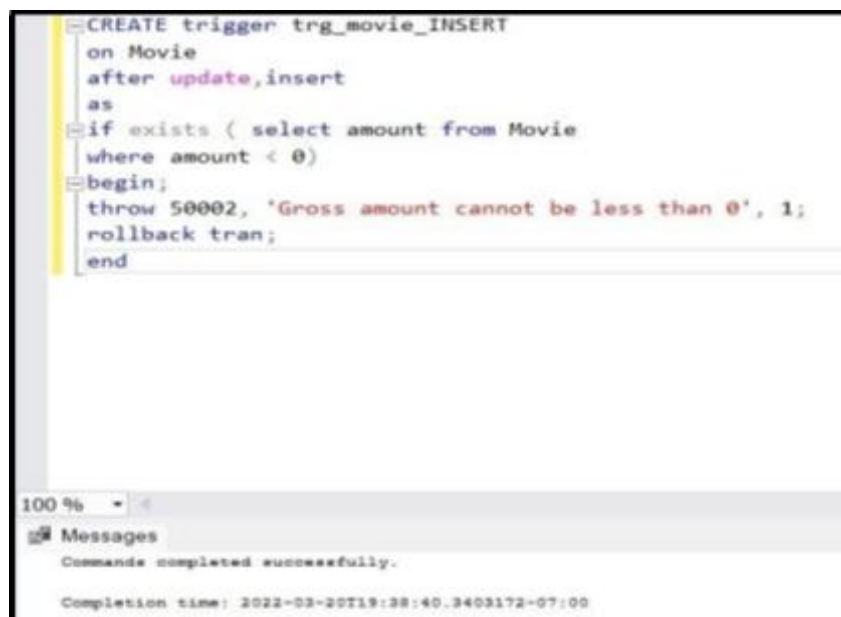
- tgr_movie_INSERT
- tgr_movie_neg_rating_INSERT

Trigger 1: tgr_movie_INSERT

The gross profit figure for the film has been set as a trigger. We tested the constraint that the amount cannot be negative.

Code :

```
CREATE trigger
trg_movie_INSERT on movie after
update,insert as
if exists ( select amount from
movie where amount < 0) begin;
throw 50002, 'Gross Amount cannot be less than 0',
1; rollback tran; end
insert into
Movie(movieId,amount,movieName,categoryName,directorName,productionName)
values(008,-24000, 'KGF', 'Action', 'Prashanth Neel', 'Hombale Films');
```



```
CREATE trigger trg_movie_INSERT
on Movie
after update,insert
as
if exists ( select amount from Movie
where amount < 0)
begin;
throw 50002, 'Gross amount cannot be less than 0', 1;
rollback tran;
end
```

100 %

Messages

Commands completed successfully.

Completion time: 2022-03-20T19:38:40.3403172-07:00

```
use ORMModel1;  
set identity_insert Movie on  
insert into Movie(movieId,amount,movieName,categoryName,directorName,productionName) values  
(008,-24000,'KGF','Action','Prashanth Neel','Hombale Films');
```

100 %

Messages

Msg 50002, Level 16, State 1, Procedure trg_movie_INSERT, Line 11 [Batch Start Line 0]
Gross amount cannot be less than 0

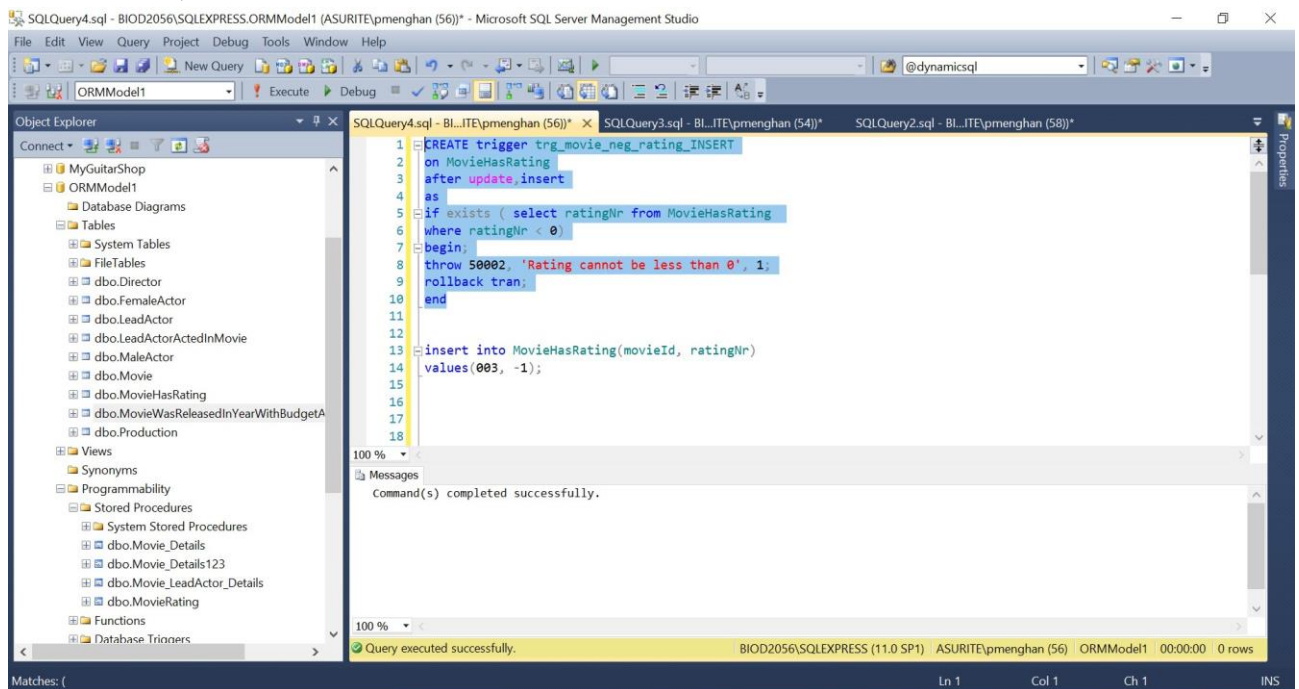
Completion time: 2022-03-20T19:49:20.7040086-07:00

Trigger 2 : tgr_movie_neg_rating_INSERT

We have developed a trigger on the insertion of negative rating. Whenever a record is inserted with a negative rating this trigger will get executed and will show a message that 'Rating cannot be less than 0'.

CODE:

```
CREATE trigger
trg_movie_rating_INSERT on
MovieHasRating after update,insert
as
if exists ( select ratingNr from
MovieHasRating where ratingNr < 0) begin;
throw 50002, 'Rating cannot be less than 0', 1;
rollback tran; end
```



```
insert into MovieHasRating(movieId, ratingNr)
values(003, -1);
```

SQLQuery4.sql - BIOD2056\SQLEXPRESS\ORMModel1 (ASURITE\pmenghan (56))* - Microsoft SQL Server Management Studio

File Edit View Query Project Debug Tools Window Help

ORMModel1 Execute Debug

Object Explorer

- Connect
- MyGuitarShop
- ORMModel1
 - Database Diagrams
 - Tables
 - System Tables
 - FileTables
 - dbo.Director
 - dbo.FemaleActor
 - dbo.LeadActor
 - dbo.LeadActorActedInMovie
 - dbo.MaleActor
 - dbo.Movie
 - dbo.MovieHasRating
 - dbo.MovieWasReleasedInYearWithBudgetA
 - dbo.Production
 - Views
 - Synonyms
 - Programmability
 - Stored Procedures
 - System Stored Procedures
 - dbo.Movie_Details
 - dbo.Movie_Details123
 - dbo.Movie_LeadActor_Details
 - dbo.MovieRating
 - Functions
 - Database Triggers

SQLQuery4.sql - BL...ITE\pmenghan (56))* SQLQuery3.sql - BL...ITE\pmenghan (54))* SQLQuery2.sql - BL...ITE\pmenghan (58))*

```
3 after update,insert
4 as
5 if exists ( select ratingNr from MovieHasRating
6 where ratingNr < 0)
7 begin;
8 throw 50002, 'Rating cannot be less than 0', 1;
9 rollback tran;
10 end
11
12
13 insert into MovieHasRating(movieId, ratingNr)
14 values(003, -1);
15
16
```

100 %

Messages

Msg 50002, Level 16, State 1, Procedure trg_movie_neg_rating_INSERT, Line 8
Rating cannot be less than 0

100 %

Query completed with errors. BIOD2056\SQLEXPRESS (11.0 SP1) ASURITE\pmenghan (56) ORMModel1 00:00:00 0 rows

Matches: (Ln 13 Col 1 Ch 1 INS

SECTION – 6 NOSQL

We downloaded and installed Couchbase Enterprise 7. Couchbase Server is a NoSQL document database that may be used to power interactive web applications. It offers a flexible data model, is readily scalable, delivers constant high performance, and can provide application data with 100% uptime.

Couchbase is the result of the marriage of two well-known NoSQL technologies:

- Membase is a high-performance Memcached technology that enables durability, replication, and sharding.
- CouchDB, is the pioneer of the JSON-based document-oriented approach.

We have used the following entities:-

- LeadActor
- Production
- Movie
- Director
- MovieWasReleasedInYearWithBudgetAmount

BUCKETS

A bucket is a logical container for a group of elements that are connected, such as key-value pairs or documents. In relational databases, buckets are analogous to databases. They offer a resource management facility for the data set that they hold. Applications can store data in one or more buckets.

We created the following two buckets:-

- Actor530
- Movie530

Abhilash530 > Buckets

ADD BUCKET

Dashboard

Servers

Buckets

Backup

XDCR

Security

Settings

Logs

Documents

Query

Indexes

Search

Analytics

Eventing

Views

filter buckets...

name

items

resident

ops/sec

RAM used/quota

disk used

Documents

Scopes & Collections

Actor530

5

100%

0

38.9MiB / 1.28GiB

8.09MiB

Documents

Scopes & Collections

Movie530

5

100%

0

38.9MiB / 1.28GiB

8.09MiB

Documents

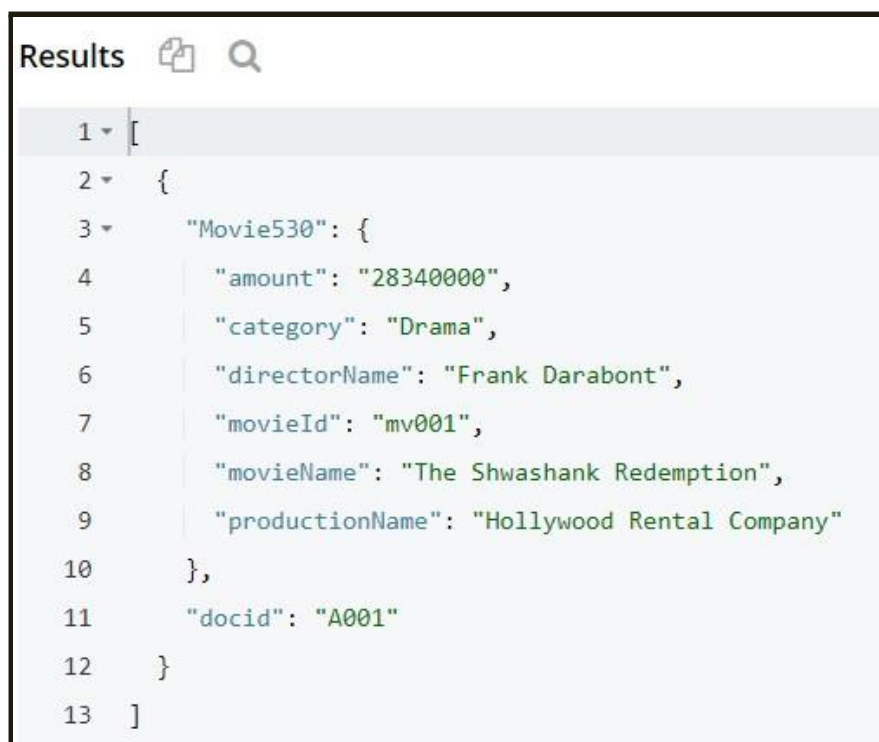
Scopes & Collections

Documents Creation

In Couchbase Server, the fundamental unit of data storage is a JSON document, allowing your application to be free of rigorously defined relational database tables. Schema migrations are not required since application objects are represented as documents.

Although binary data may also be stored in documents, the JSON structure allows the data to be indexed and accessed using views. Couchbase Server has a JavaScript-based query engine that allows you to discover records based on field values.

We have created 5 Documents in the bucket Movie 530 and 4 documents in the bucket Actor530

A screenshot of the Couchbase query results interface. The title bar says 'Results' with a copy icon and a search icon. The results are displayed in a light blue box with line numbers on the left. The JSON document is as follows:

```
1  [
2    {
3      "Movie530": {
4        "amount": "28340000",
5        "category": "Drama",
6        "directorName": "Frank Darabont",
7        "movieId": "mv001",
8        "movieName": "The Shwashank Redemption",
9        "productionName": "Hollywood Rental Company"
10     },
11     "docid": "A001"
12   }
13 ]
```

Results



```
1 ▾ [  
2 ▾  {  
3 ▾   "Movie530": {  
4     "amount": "134970000",  
5     "category": "Crime",  
6     "directorName": "Francis Ford Coppola",  
7     "movieId": "mv002",  
8     "movieName": "The Godfather",  
9     "productionName": "Paramount"  
10    },  
11    "docid": "A002"  
12  }  
13  ]
```

Results



```
1 ▾ [  
2 ▾  {  
3 ▾   "Movie530": {  
4     "amount": "534860000",  
5     "category": "Action",  
6     "directorName": "Christopher Nolan",  
7     "movieId": "mv003",  
8     "movieName": "The Dark Knight",  
9     "productionName": "Warner Bros"  
10    },  
11    "docid": "A003"  
12  }  
13  ]
```


Results

```
1 ▾ [  
2   {  
3     "Movie530": {  
4       "amount": "53370000",  
5       "category": "Comedy",  
6       "directorName": "Bong Joon Ho",  
7       "movieId": "mv004",  
8       "movieName": "Parasite",  
9       "productionName": "Barunson E&A"  
10    },  
11    "docid": "A004"  
12  }  
13 ]
```

Results

```
1 ▾ [  
2   {  
3     "Movie530": {  
4       "amount": "260000000",  
5       "category": "Action",  
6       "directorName": "Prashanth Neel",  
7       "movieId": "mv005",  
8       "movieName": "KGF Chapter 2",  
9       "productionName": "Hombale Films"  
10    },  
11    "docid": "A005"  
12  }  
13 ]
```

Results  

```
1 ▾ [  
2 ▾ {  
3 ▾   "Actor530": {  
4     |   "leadActor": "Tim Robbins",  
5     |   "movieName": "The Shwashank Redemption"  
6     | },  
7     | "docid": "B001"  
8     | }  
9   ]
```

Results  

```
1 ▾ [  
2 ▾ {  
3 ▾   "Actor530": {  
4     |   "leadActor": "Marlon Brando",  
5     |   "movieName": "The Godfather"  
6     | },  
7     | "docid": "B002"  
8     | }  
9   ]
```

Results

```
1 ▾ [  
2 ▾  {  
3 ▾   "Actor530": {  
4     |   "leadActor": "Christian Bale",  
5     |   "movieName": "The Dark Knight"  
6     | },  
7     | "docid": "B003"  
8     | }  
9   ]
```

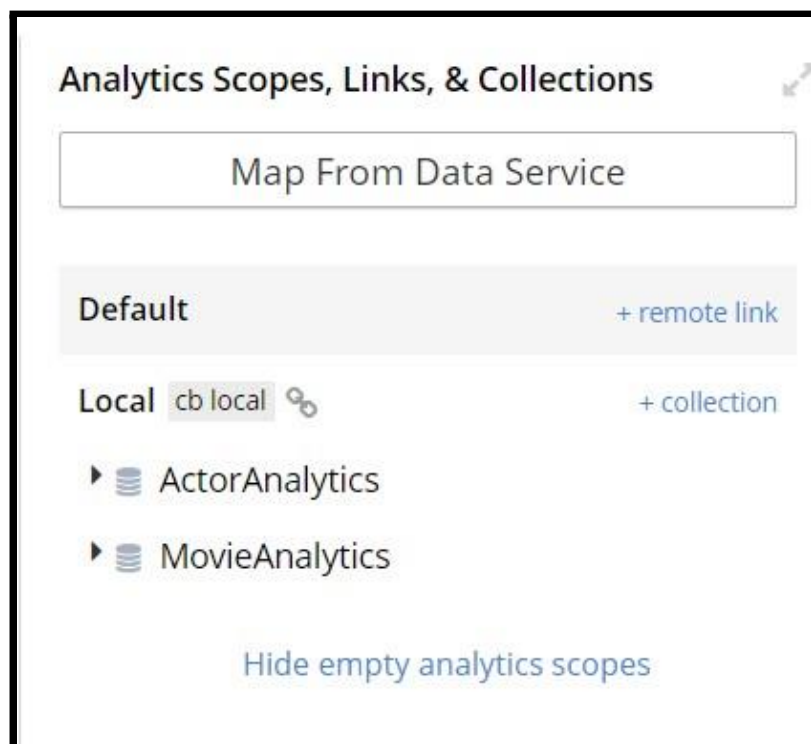
```
20 ▾  {  
21 ▾   "Actor530": {  
22     |   "leadActor": "Kang Ho Song",  
23     |   "movieName": "Parasite"  
24     | }  
25   },  
26 ▾  {  
27 ▾   "Actor530": {  
28     |   "leadActor": "Rocking Star Yash",  
29     |   "movieName": "KGF Chapter 2"  
30     | }
```

ANALYTIC SERVICE

Couchbase Analytics is best suited for conducting big, complicated queries that include data aggregations on enormous volumes of data. The Analytics Service enables you to rapidly and simply create insight-driven applications. There are two main areas of focus: operational analytics and near real-time analytics.

We loaded the Couchbase/Mongo with the data from the analysis above using the Query service.

We then linked and loaded Query services data with the Analytics service.



SQL QUERIES

We then performed 2 SQL++ queries against these buckets using the Analytics service

Query Editor

← history (37/37) →

query context

```
1 SELECT MovieAnalytics.movieId, MovieAnalytics.movieName, MovieAnalytics.productionName, ActorAnalytics.leadActor
2 FROM ActorAnalytics
3 INNER JOIN MovieAnalytics ON ActorAnalytics.movieName=MovieAnalytics.movieName;
```

Execute

Explain

✓ success

elapsed: 313.92ms | execution: 282.74ms | docs scanned: 10 | docs returned: 5 | size: 609 bytes

format

Query Results

JSON

Table

Chart

Plan

Plan Text

leadActor	movieId	movieName	productionName
Tim Robbins	mv001	The Shwashank Redemption	Hollywood Rental Company
Marlon Brando	mv002	The Godfather	Paramount
Christian Bale	mv003	The Dark Knight	Warner Bros
Kang Ho Song	mv004	Parasite	Barunson E&A
Rocking Star Yash	mv005	KGF Chapter 2	Hombale Films

Query Editor

← history (38/38) →

query context

```
1 FROM MovieAnalytics as m JOIN ActorAnalytics as a
2 ON m.movieName = a.movieName
3 WHERE m.category = "Action"
4 SELECT m.productionName, a.leadActor;
```

Execute

Explain

✓ success

elapsed: 255.28ms | execution: 212.38ms | docs scanned: 10 | docs returned: 2 | size: 141 bytes

format

Query Results

JSON

Table

Chart

Plan

Plan Text

leadActor	productionName
Christian Bale	Warner Bros
Rocking Star Yash	Hombale Films

NOSQL CODE :

Create primary index on Actor530

Create primary index on Movie530

Insert into Movie530 (key,value) values (

"A001",

{

 "movieId":"mv001",

 "movieName":"The Shawshank Redemption",

 "directorName":"Frank Darabont",

 "productionName":"Hollywood Rental Company",

 "category":"Drama",

 "amount":"28340000",

}

Insert into Movie530 (key,value) values (

"A002",

{

 "movieId":"mv002",

 "movieName":"The Godfather",

 "directorName":"Francis Ford Coppola",

 "productionName":"Paramount",

 "category":"Crime",

 "amount":"134970000",

}

Insert into Movie530 (key,value) values (

"A003",

{

 "movieId":"mv003",

 "movieName":"The Dark Knight",

 "directorName":"Christopher Nolan",

 "productionName":"Warner Bros",

 "category":"Action",

 "amount":"534860000",

}

"A004",

{

 "movieId":"mv004",

 "movieName":"Parasite",

 "directorName":"Bong Joon Ho",

 "productionName":"Barunson E&A",

```
    "category": "Comedy",
    "amount": "53370000",
```

```
}
```

```
Insert into Movie530 (key,value) values (
```

```
"A005",
```

```
{
```

```
    "movieId": "mv005",
```

```
    "movieName": "KGF Chapter 2",
```

```
    "directorName": "Prashanth Neel",
```

```
    "productionName": "Hombale Films",
```

```
    "category": "Action",
```

```
    "amount": "260000000",
```

```
}
```

```
Analytics created:
```

- ActorAnalytics
- MovieAnalytics

```
).RETURNING META().id as docid, *;
```

```
SELECT MovieAnalytics.movieId, MovieAnalytics.movieName,
```

```
MovieAnalytics.productionName, ActorAnalytics.leadActor FROM ActorAnalytics
```

```
INNER JOIN MovieAnalytics ON ActorAnalytics.movieName= MovieAnalytics.movieName
```

```
FROM MovieAnalytics as m JOIN ActorAnalytics as a
```

```
ON m.movieName = a.movieName
```

```
WHERE m.category = "Action"
```

```
SELECT m.productionName, a.leadActor;
```

SUMMARY

We have specified the core data entities and the structure of the database that we propose to construct after completing the initial phase of this project. This database module is intended to aid a Movie Management company in a variety of areas, including managing movie genres, tracking movie progress, and buying movie tickets. We've articulated certain questions that a movie management company could ask, and we've built our entities around these queries. Apart from the basic stored procedure and triggers, we implemented in phase 1, we implemented several users specified functions, stored procedures, and triggers as part of phase 2 of this project to offer other functionality such as ticket availability, best movie, best production, and actors, production tenure, and so on. We used SQL Server, NORMA, Couchbase Enterprise 7, and SQL Server Management Studio to build this project. The suggested system should be a solid option for managing movies without any human errors. This should be able to handle data as well. The system displays each actor's and movie's statistics, which aids in the management of the Cinema. To do this, we employed several functions such as UDF to define the general flow of the management system. We employed numerous stored procedures because a stored procedure offers an important layer of protection between the user interface and the database. It increases security through data access limits since end users may enter or change data but not construct procedures. We also added two Triggers to help keep the User Interface light. Instead of repeating the same function call throughout the program, you may have it execute using a trigger.

CONCLUSION

Following the development of this project, we saw the relevance of object role modeling and how it can be an exceptionally useful tool for creating databases that bring great levels of convenience to many applications in our daily lives. We wanted to reduce data redundancy and improve database connection using our technique. We connected our current database with other ideas presented in class, like triggers, stored procedures, and user-defined functions, to consider the future aspect of this project. Our objective was to add different functions to our model using the ideas of buckets, JSON structures, and Couchbase.

Because the project is still in its early phases, there are many more features that need to be added to improve existing functionality. It is feasible to utilize data prediction to assist selectors in making squad decisions without having to look at each player's numbers individually. Because the system currently lacks a user interface, users would need to be familiar with SQL and databases. Along with the development of a web application, a graphical user interface should be created. We may develop a dashboard that displays which players have been picked for forthcoming matches, teams that require a new coach, and so on. Although the project is now operating on a basic level, improving its features would make it sparkle and set it apart.

REFERENCES

Textbooks:

- Terry Halpin's Object-Role Modeling Fundamentals: A Practical Guide to Data Modeling with ORM
- Murach's SQL for SQL Server

Articles:

- Halpin, T. 2010, 'Object-Role Modeling: Principles and Benefits', International Journal of Information Systems Modeling and Design,

Websites:

- <https://blog.layoutindex.com/the-importance-of-centralizing-your-operations-in-a-cinema/>
- <https://www.layoutindex.com/business-solutions/products/cinemamanagement-system>