Assignment CA Individual Report

Data Storage Solutions for Data Analytics

M.Sc. in Data Analytics (Group C)

January 2020 Intake

By:

Yogeshwaran Shanmuganathan

Contents

1. Introduction:

The movie industry is multi trillion Euro industry, and it is one of the most influential mediums around the globe. It is the riskiest business to get into and being an investor or producer of a movie comes with high risk.

There are various external and internal factor involved for a movie to be a success. If we talk about risks a person going for a movie keep track of attributes like: "Director of the Movie", "Actor", "Language", "Category" and etc before watching a movie and it is also associated with the geographical location where it is being launched.

Different movies depending on the "Category" it belonged attracts distinct crowd and as well as the Country in which it was launched cherishes the movie if the "Ticket price" is right.

In this project, we will use the IMDB-Movie rank data as well as the movie data from "rottentomato", developing the data warehouse and validate the facts and will try to answer the questions as an analyst for an Investment firm who wishes to invest in movies on the following criteria:

- Does the success of movie depend on the "Director" and "Actor"?
- Which type of movie category is mostly liked?
- Which county movie industry is profitable?
- Does success of the movie depend on its release date or month?

2. Data Sources:

- Relational Dataset Repository - https://relational.fit.cvut.cz/
- IMDB Rating website - https://www.imdb.com/chart/top/
- List of Countries by Continent - http://statisticstimes.com/geography/countries-by-continents.php

3. Data Model:

The objective of a data warehouse design is to create a schema that is optimized for decision support processing. The relational model is used in systems where many transactions are executed, most of them concurrently. A transaction inserts, updates or in any other way processes data in a database. In many occasions a transaction is an integral part of the business process. A Relational Database is a set of database tables that are related using keys from other database tables.

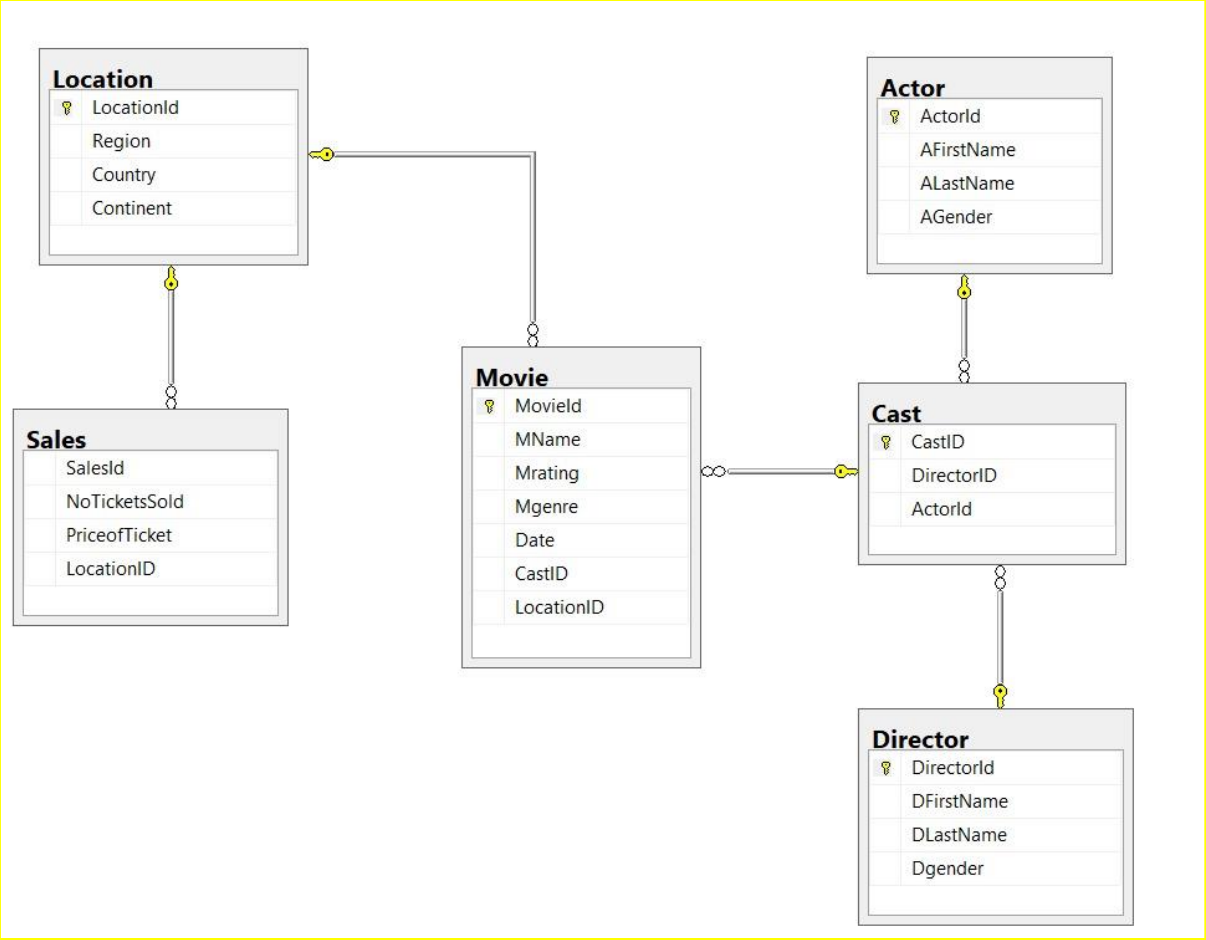


Fig. ENTITY RELATIONSHIP DIAGRAM

Each point of data in each data base can be split into two basic entities, namely attributes and measures. Dimensions or attributes are descriptive details about various objects allowing for their detailed analysis, measures are those that can be added across all dimensions. Measures are called facts. Facts are the reporting unit of any data warehouse, i.e. those are the ones that are manipulated, measured, and reported for analysis. Dimensions are those against which these facts are reported. So, it can be said that dimensions give extra information regarding the facts. A star schema consists of a central fact table which is linked directly to each of the dimension tables via a "one to one" OR "one to many" relationships.

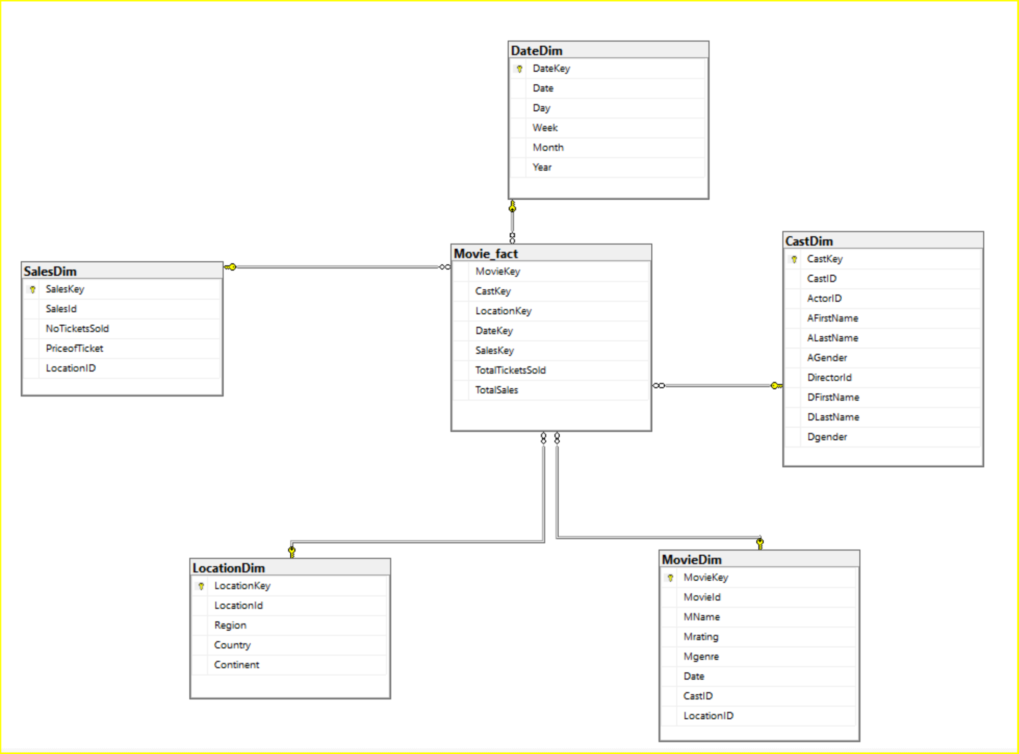


Fig. STAR SCHEMA

4. Dimensions Tables:

- Sale Dimension Table/SaleDim: The Sale dimension table consists of five attributes having Sales key as Primary Key, SalesId, No of Tickets sold, LocationId
- Date Dimension Table/DateDim: The data dimension table has 6 attributes with DateKey being the Primary Key
- Cast Dimension Table/CastDim: The cast dimension table has 10 distinct attributes with CastKey being the Primary Key
- Location Dimension Table/LocationDim: The location dimension table has 5 attributes with Locationkey being the Primary key
- Movie Dimension Table/MovieDim: The Movie dimension table has 8 attributes with MovieKey being the Primary Key.

5. ETL Process:

An ETL process is defined for stage-wise loading of different data sources into raw data tables, generating dimensions and facts, and deploying the cube for further BI analysis. This ETL process is configured in Visual Studio(SSIS) tool for integration and different components are used to perform the process. The following 5 steps are configured in a sequential manner and triggered on successful completion of the prior task.

➢ 5.1 Source Data Extraction:

To start with, data from different sources provided in Section 2 are extracted by means of Python code. Although, some data source like Statista which sits behind the authentication is manually downloaded and referred through one place from Python Code, cleaned and transform to generate structured CSV file for loading into the staging tables.

A meaningful flow structure is implemented for processing different files. All independent data sources are downloaded/cleaning/processed in parallel. Then, dependent processes are executed.

➢ 5.2 Staging Arena Cleaning:

In this stage, All the raw tables are truncated through SQL scripts. This ensures that fresh data is loaded into the database every time entire ETL flow is executed.

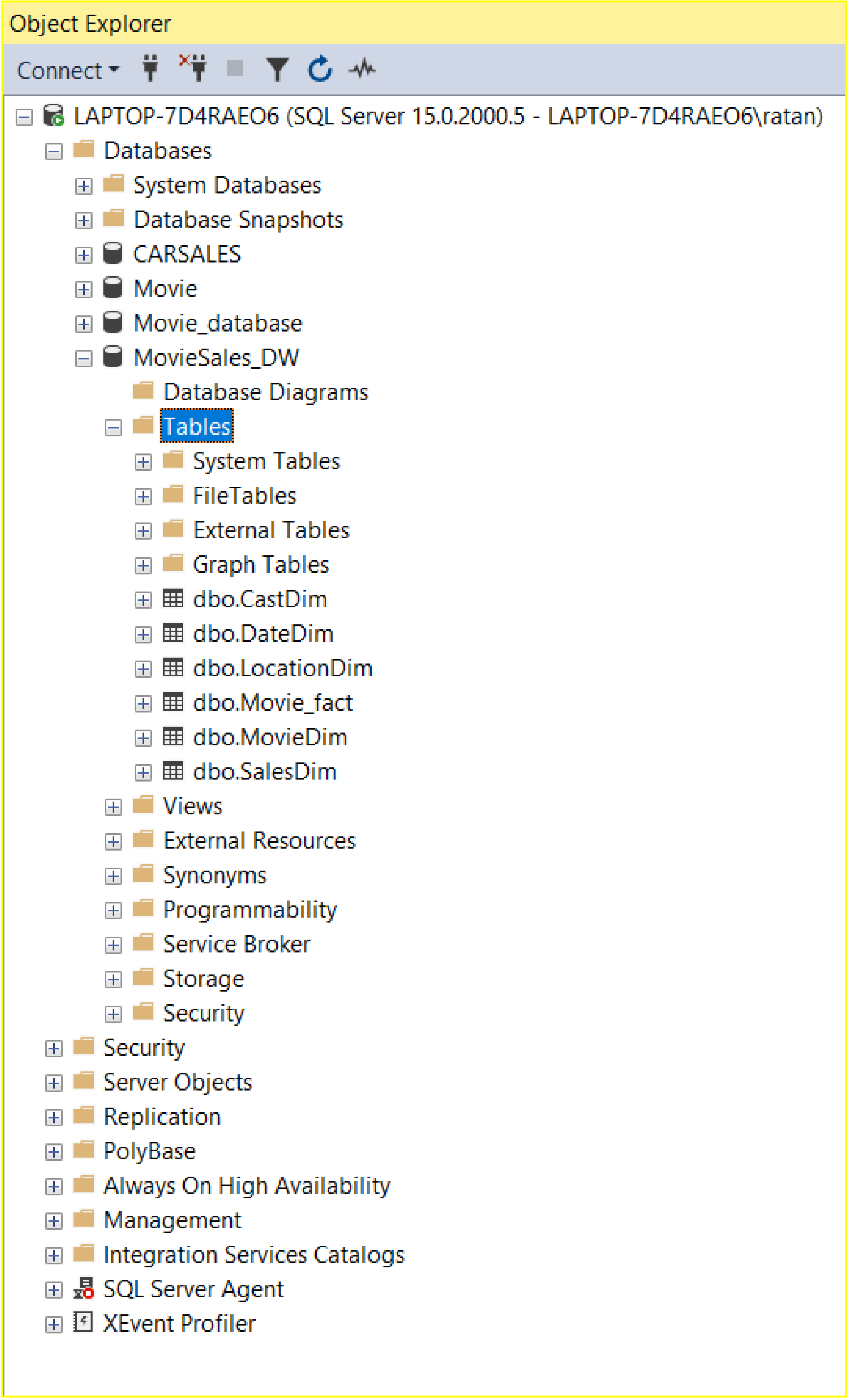
- Creating Dimension tables and Fact tables using SQL script:

```
CREATE DATABASE Movie_database
USE Movie_database;

ALTER TABLE Movie add foreign key (LocationID ) references Location
(LocationID);

ALTER TABLE Movie add foreign key (CastID ) references Cast(CastID);

ALTER TABLE Cast add foreign key (DirectorID ) references
Director(DirectorID);
```

```
ALTER TABLE Cast add foreign key (ActorId ) references Actor(ActorId);

ALTER TABLE Sales add foreign key (LocationId ) references
Location(locationId);


CREATE DATABASE MovieSales_DW

USE MovieSales_DW;

CREATE TABLE CastDim
(
CastKey int identity Primary key not null,
CastID Varchar(50),
ActorID int,
AFirstName varchar(50),
ALastName varchar(50),
AGender varchar(50),
DirectorId varchar(50),
DFirstName varchar(50),
DLastName varchar(50),
Dgender varchar(50)
);

CREATE TABLE MovieDim
(
MovieKey int identity Primary key not null,
MovieId INT ,
MName varchar(50),
Mrating float,
Mgenre varchar(50),
Date date,
CastID varchar(50),
LocationID varchar(50)
);

CREATE TABLE LocationDim
(
LocationKey int identity Primary key not null,
LocationId INT ,
Region varchar(50),
Country varchar(50),
Continent varchar(50)
);
```

```
CREATE TABLE DateDim
(
DateKey int identity Primary key not null,
Date DATE,
Day INT,
Week INT,
Month INT,
Year INT,
);

Create Table SalesDim
(
SalesKey int identity Primary key not null,
SalesId int,
NoTicketsSold int,
PriceofTicket int,
LocationID varchar(50)
);

CREATE TABLE Movie_fact
(
MovieKey INT FOREIGN KEY REFERENCES MovieDim(MovieKey),
CastKey int foreign key references CastDim(CastKey),
LocationKey int foreign key references LocationDim(LocationKey),
DateKey int foreign key references DateDim(DateKey),
SalesKey int foreign key references SalesDim(SalesKey),
TotalTicketsSold int,
TotalSales int
)
```

- SQL Script Output:

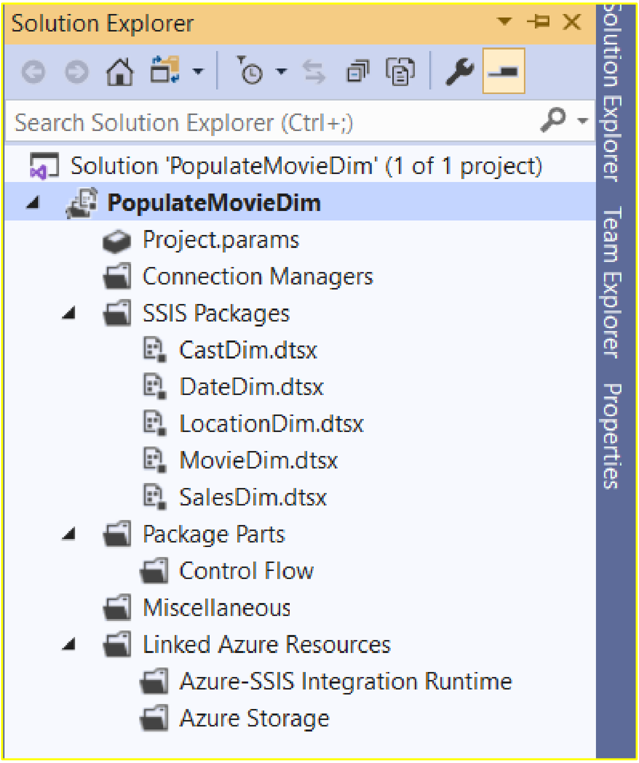


```
Object Explorer
Connect ▾
⊟ LAPTOP-7D4RAEO6 (SQL Server 15.0.2000.5 - LAPTOP-7D4RAEO6\ratan)
    ⊟ Databases
        ⊞ System Databases
        ⊞ Database Snapshots
        ⊞ CARSALES
        ⊞ Movie
        ⊞ Movie_database
        ⊟ MovieSales_DW
              Database Diagrams
            ⊟ Tables
                ⊞ System Tables
                ⊞ FileTables
                ⊞ External Tables
                ⊞ Graph Tables
                ⊞ dbo.CastDim
                ⊞ dbo.DateDim
                ⊞ dbo.LocationDim
                ⊞ dbo.Movie_fact
                ⊞ dbo.MovieDim
                ⊞ dbo.SalesDim
            ⊞ Views
            ⊞ External Resources
            ⊞ Synonyms
            ⊞ Programmability
            ⊞ Service Broker
            ⊞ Storage
            ⊞ Security
    ⊞ Security
    ⊞ Server Objects
    ⊞ Replication
    ⊞ PolyBase
    ⊞ Always On High Availability
    ⊞ Management
    ⊞ Integration Services Catalogs
    ⊞ SQL Server Agent
    ⊞ XEvent Profiler
```

Fig. Dimension tables and Fact tables

5.3 Load data into Staging Arena:

This step loads all the processed flat files into the raw tables with the help of SSIS data flow task. The loading of files is executed in parallel to optimise the overall time.



Solution Explorer

Search Solution Explorer (Ctrl+;)

- Solution 'PopulateMovieDim' (1 of 1 project)
  - PopulateMovieDim
    - Project.params
    - Connection Managers
    - SSIS Packages
      - CastDim.dtsx
      - DateDim.dtsx
      - LocationDim.dtsx
      - MovieDim.dtsx
      - SalesDim.dtsx
    - Package Parts
      - Control Flow
    - Miscellaneous
    - Linked Azure Resources
      - Azure-SSIS Integration Runtime
      - Azure Storage

Fig. SSIS Staging Area

5.4 Populate Dimensions & Facts:

Once raw tables are populated and available to use from the staging arena, there is a need to transform and load these data into dimensions and fact tables.

The dimensions tables are truncated initially and then followed by populating it again with staging tables.

All the dimensions are cleaned before loading data from staging tables so that it can be re-runnable. Additionally, each dimension table maintains a unique identity key which is generated while inserting and it is used to link with fact tables rows. Once dimensions are loaded, then the next stage loads the data into the fact table.

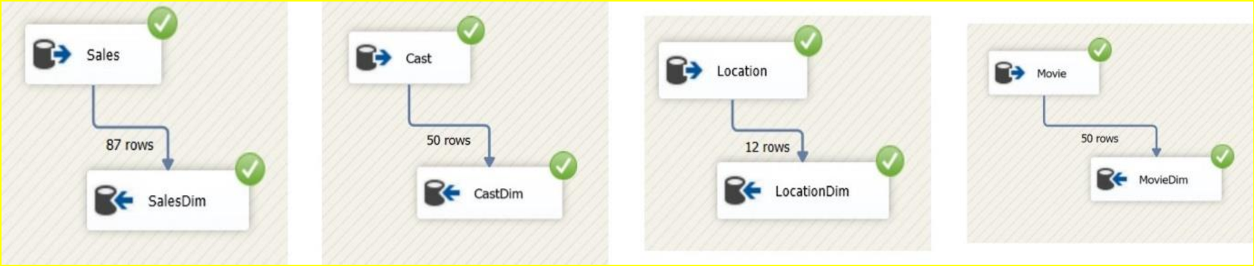


Fig. Populating Dimension and fact table in SSIS

➢ 5.5 Dimensions & Fact Table deployment:

Once data is available into facts and dimensions table, Dimensions and facts table are constructed through SSIS multidimensional project.

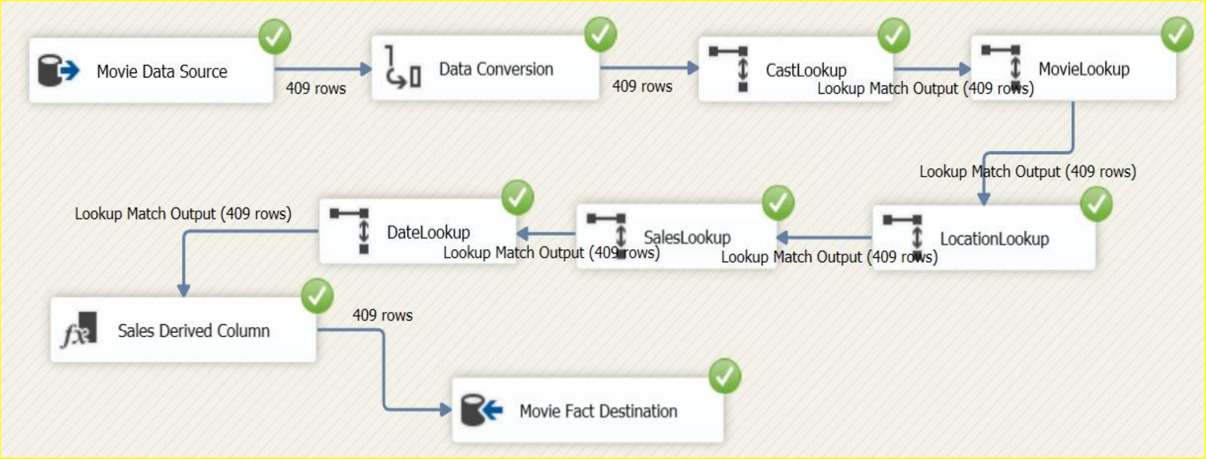
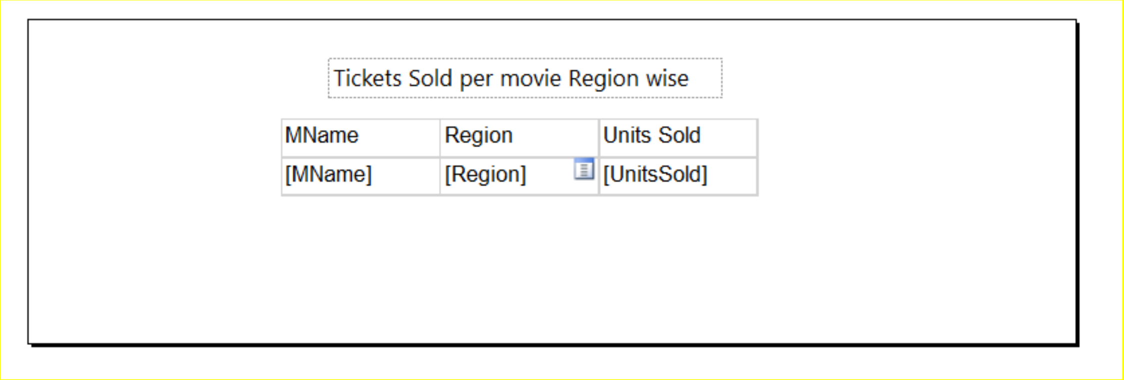


Fig. SSIS Deployment

6. Reports & Visualisation:

➢ 6.1 SSRS:

- **Report_1:**
  - **Design:**

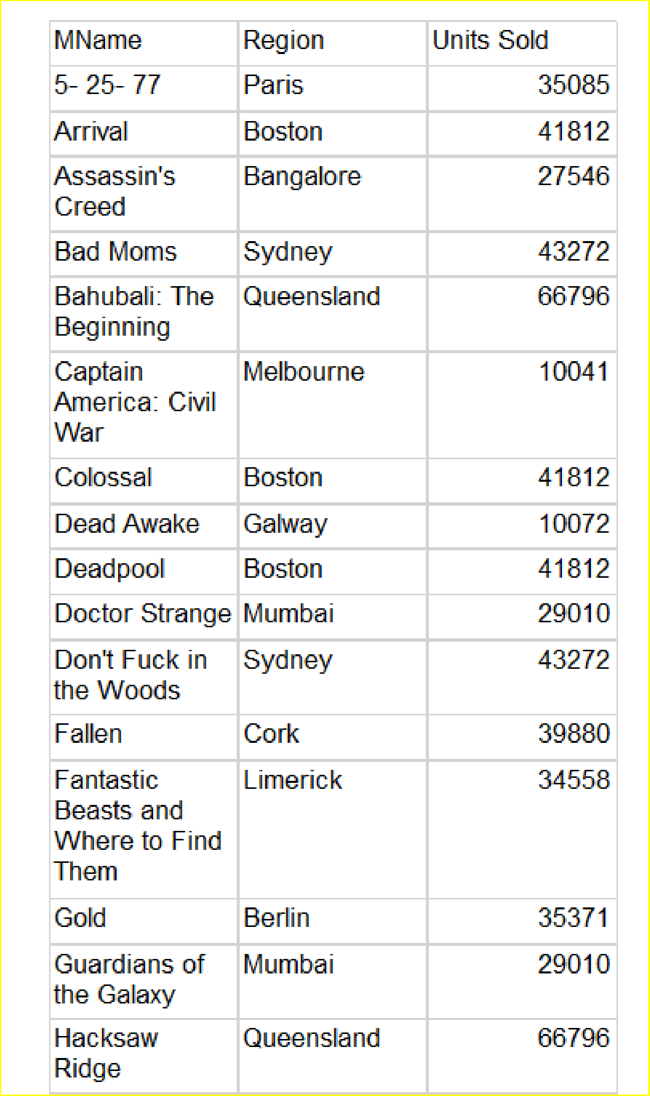


Tickets Sold per movie Region wise

MName	Region	Units Sold
[MName]	[Region]	[UnitsSold]

Fig. Report_1 Design

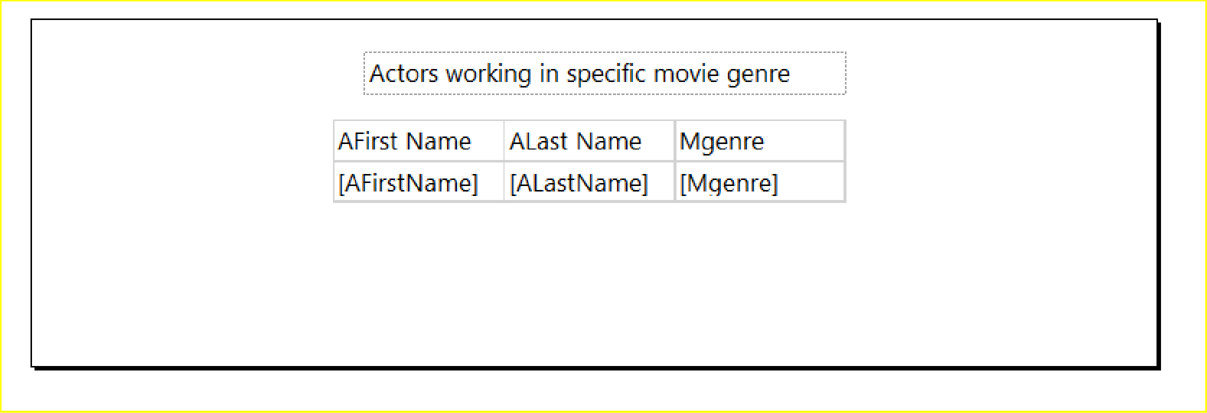
- **Preview:**



MName	Region	Units Sold
5- 25- 77	Paris	35085
Arrival	Boston	41812
Assassin's Creed	Bangalore	27546
Bad Moms	Sydney	43272
Bahubali: The Beginning	Queensland	66796
Captain America: Civil War	Melbourne	10041
Colossal	Boston	41812
Dead Awake	Galway	10072
Deadpool	Boston	41812
Doctor Strange	Mumbai	29010
Don't Fuck in the Woods	Sydney	43272
Fallen	Cork	39880
Fantastic Beasts and Where to Find Them	Limerick	34558
Gold	Berlin	35371
Guardians of the Galaxy	Mumbai	29010
Hacksaw Ridge	Queensland	66796

Fig. Report_1 Preview

- **Report_2:**
  - **Design:**



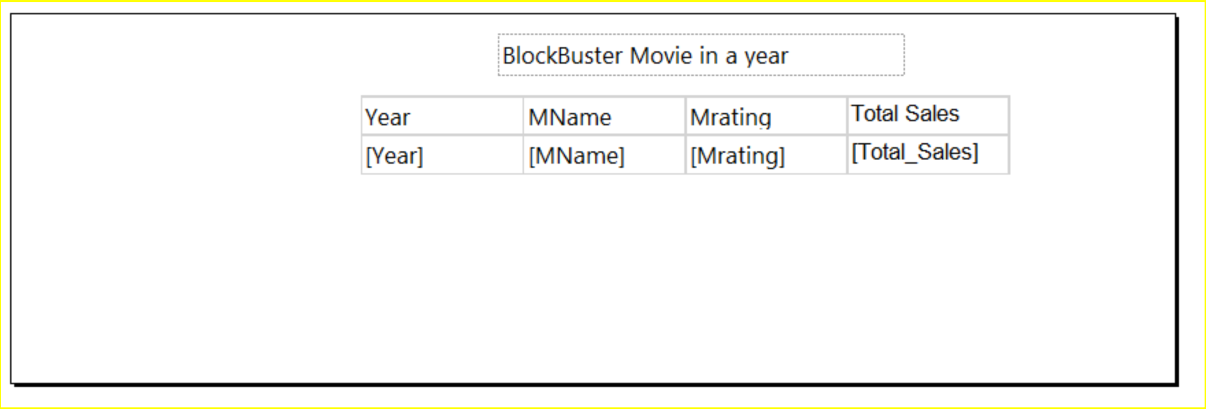
Actors working in specific movie genre

AFirst Name	ALast Name	Mgenre
[AFirstName]	[ALastName]	[Mgenre]

Fig. Report_2 Design

- **Report_3:**

- **Design:**

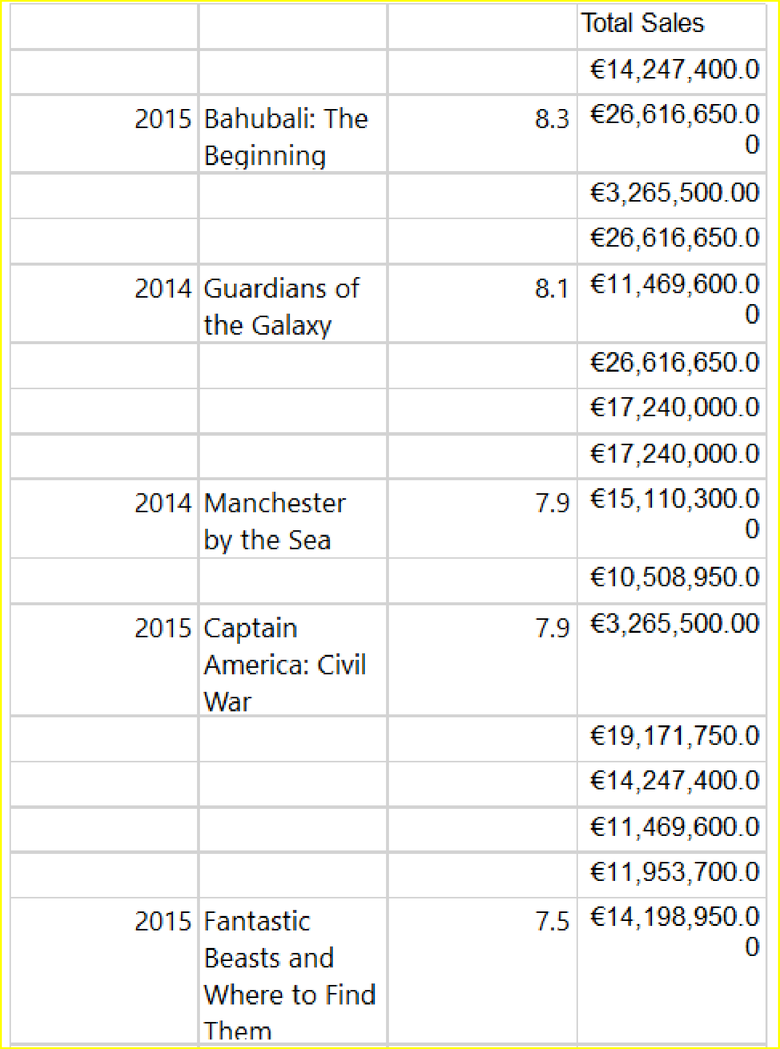


BlockBuster Movie in a year

Year	MName	Mrating	Total Sales
[Year]	[MName]	[Mrating]	[Total_Sales]

Fig. Report_3 Design

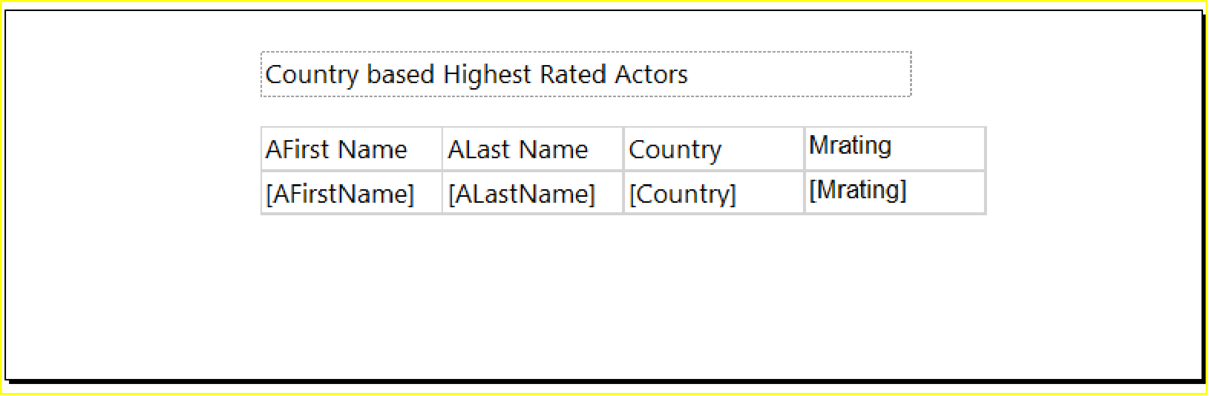
- **Preview:**



			Total Sales
			€14,247,400.0
2015	Bahubali: The Beginning	8.3	€26,616,650.00
			€3,265,500.00
			€26,616,650.0
2014	Guardians of the Galaxy	8.1	€11,469,600.00
			€26,616,650.0
			€17,240,000.0
			€17,240,000.0
2014	Manchester by the Sea	7.9	€15,110,300.00
			€10,508,950.0
2015	Captain America: Civil War	7.9	€3,265,500.00
			€19,171,750.0
			€14,247,400.0
			€11,469,600.0
			€11,953,700.0
2015	Fantastic Beasts and Where to Find Them	7.5	€14,198,950.00

Fig. Report_3 Preview

- **Report_4:**
  - **Design:**

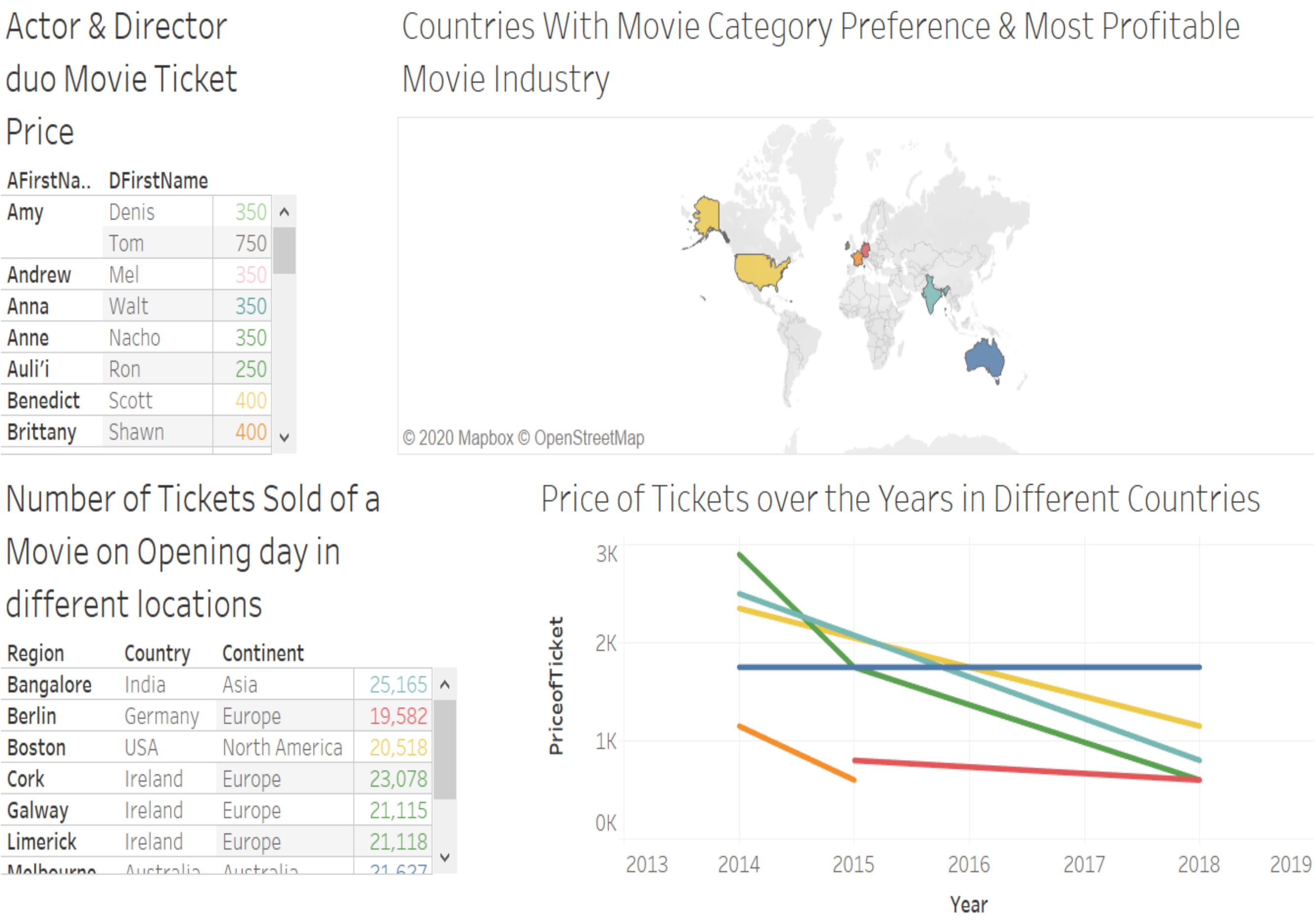


Country based Highest Rated Actors

AFirst Name	ALast Name	Country	Mrating
[AFirstName]	[ALastName]	[Country]	[Mrating]

Fig. Report_4 Design

- **6.2 Tableau:** It is an analytical software tool used in analysing and visualisation of data.



Actor & Director duo Movie Ticket Price

AFirstNa..	DFirstName	
Amy	Denis	350
	Tom	750
Andrew	Mel	350
Anna	Walt	350
Anne	Nacho	350
Auli'i	Ron	250
Benedict	Scott	400
Brittany	Shawn	400

Countries With Movie Category Preference & Most Profitable Movie Industry

© 2020 Mapbox © OpenStreetMap

Number of Tickets Sold of a Movie on Opening day in different locations

Region	Country	Continent	
Bangalore	India	Asia	25,165
Berlin	Germany	Europe	19,582
Boston	USA	North America	20,518
Cork	Ireland	Europe	23,078
Galway	Ireland	Europe	21,115
Limerick	Ireland	Europe	21,118
Melbourne	Australia	Australia	21,627

Price of Tickets over the Years in Different Countries

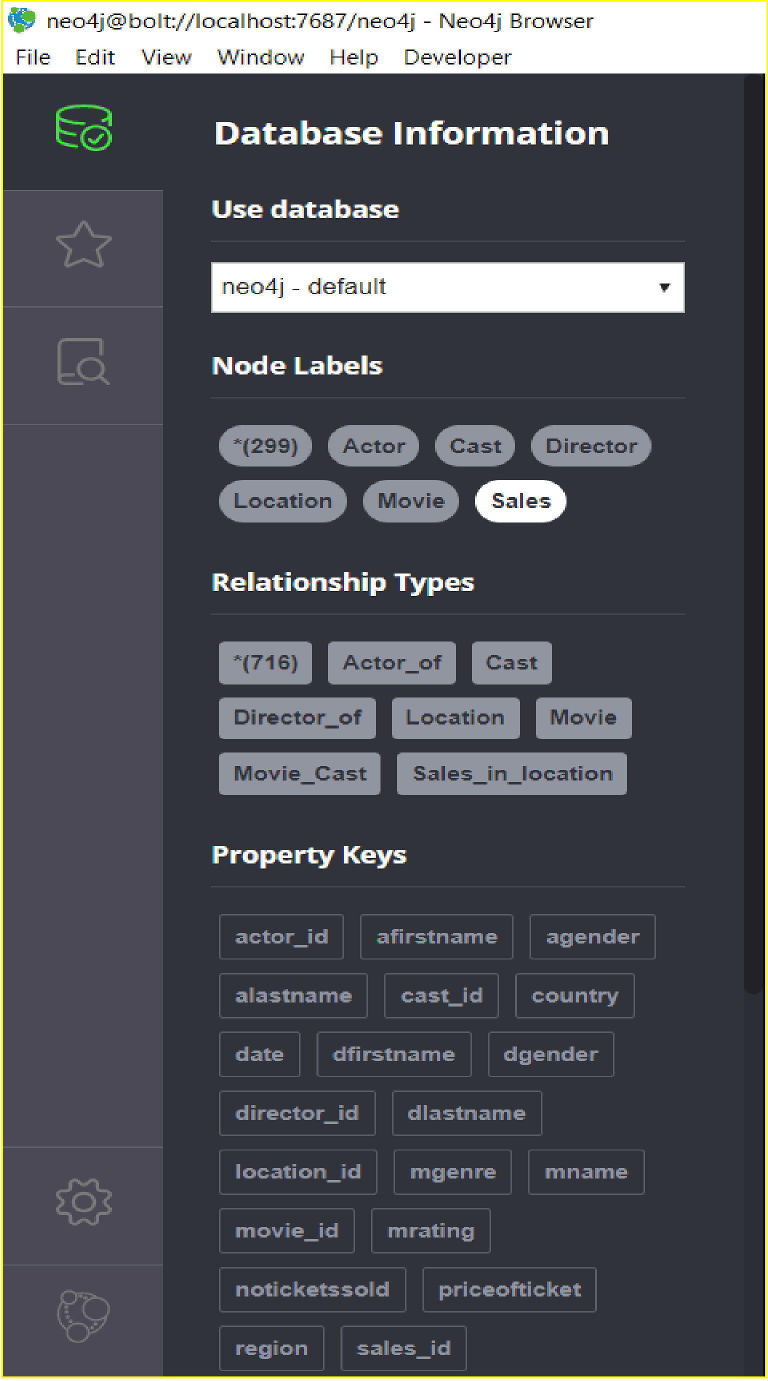
Fig. Tableau Dashboard

7. Graph Database- Neo4J:

We use Neo4j's Cypher Query Language to compare our relational database models. First step involves creating the "Movie Sales" model into the database and populating them.

In Neo4j, nodes represent each entity having attributes and a relationship can be established between these nodes to connect the nodes.

These are the nodes and relationships created for our "Movie Sales" database.



neo4j@bolt://localhost:7687/neo4j - Neo4j Browser
File Edit View Window Help Developer

Database Information

Use database

neo4j - default

Node Labels

*(299) Actor Cast Director Location Movie Sales

Relationship Types

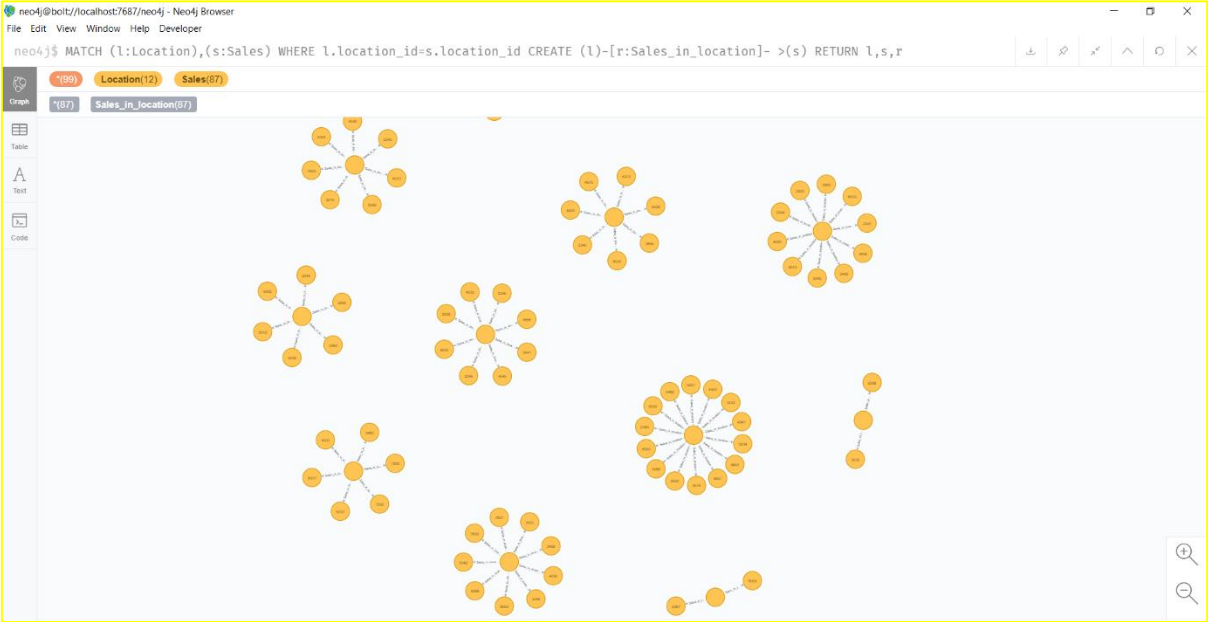
*(716) Actor_of Cast Director_of Location Movie Movie_Cast Sales_in_location

Property Keys

actor_id afirstname agender alastname cast_id country date dfirstname dgender director_id dlastname location_id mgenre mname movie_id mrating noticketssold priceofticket region sales_id

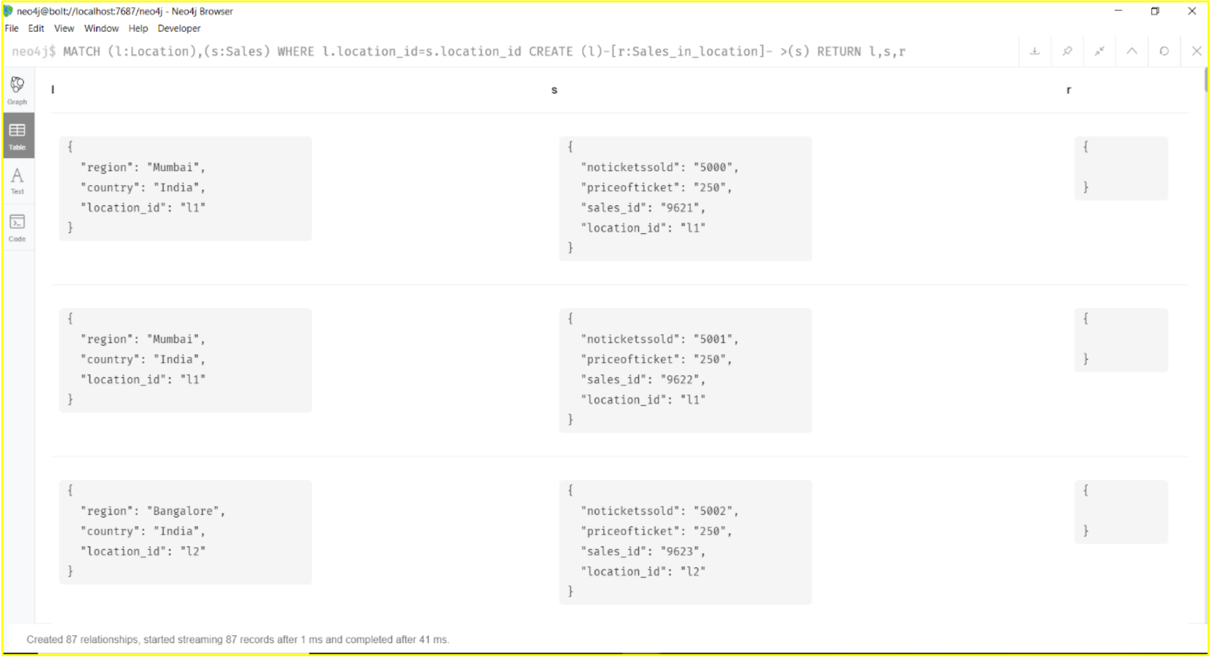
Fig. Neo4j Database Information

We can click on each of the nodes as well as the relationships to check its graphical representation.


neo4j@bolt://localhost:7687/neo4j - Neo4j Browser
File Edit View Window Help Developer
Graph
Table
Text
Code
*(99)
Location(12)
Sales(87)
*(87)
Sales_in_location(87)

```
neo4j$ MATCH (l:Location),(s:Sales) WHERE l.location_id=s.location_id CREATE (l)-[r:Sales_in_location]- >(s) RETURN l,s,r
```

Fig. Neo4j Graph Database


neo4j@bolt://localhost:7687/neo4j - Neo4j Browser
File Edit View Window Help Developer
Graph
Table
Text
Code

```
neo4j$ MATCH (l:Location),(s:Sales) WHERE l.location_id=s.location_id CREATE (l)-[r:Sales_in_location]- >(s) RETURN l,s,r
```

l	s	r

{
"region": "Mumbai",
"country": "India",
"location_id": "l1"
}
	{
	"noticketssold": "5000",
	"priceofticket": "250",
	"sales_id": "9621",
	"location_id": "l1"
	}
		{
		}

{
"region": "Mumbai",
"country": "India",
"location_id": "l1"
}
	{
	"noticketssold": "5001",
	"priceofticket": "250",
	"sales_id": "9622",
	"location_id": "l1"
	}
		{
		}

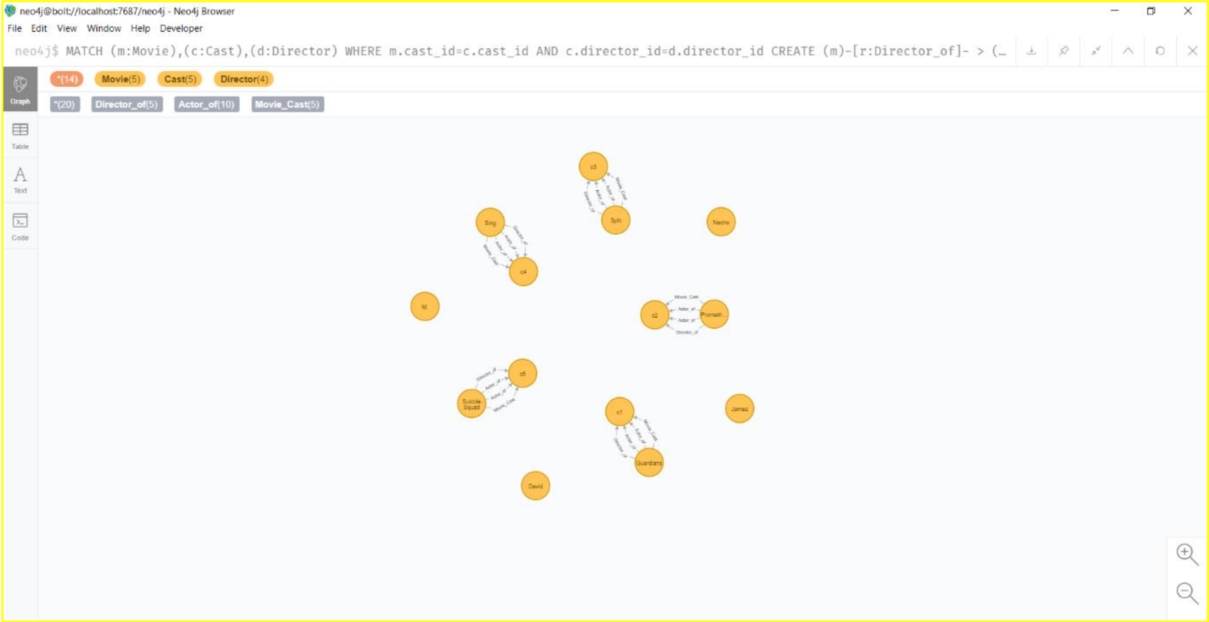
{
"region": "Bangalore",
"country": "India",
"location_id": "l2"
}
	{
	"noticketssold": "5002",
	"priceofticket": "250",
	"sales_id": "9623",
	"location_id": "l2"
	}
		{
		}

Created 87 relationships, started streaming 87 records after 1 ms and completed after 41 ms.

Fig. Neo4j Table Database

7.1 Graph Database Query:

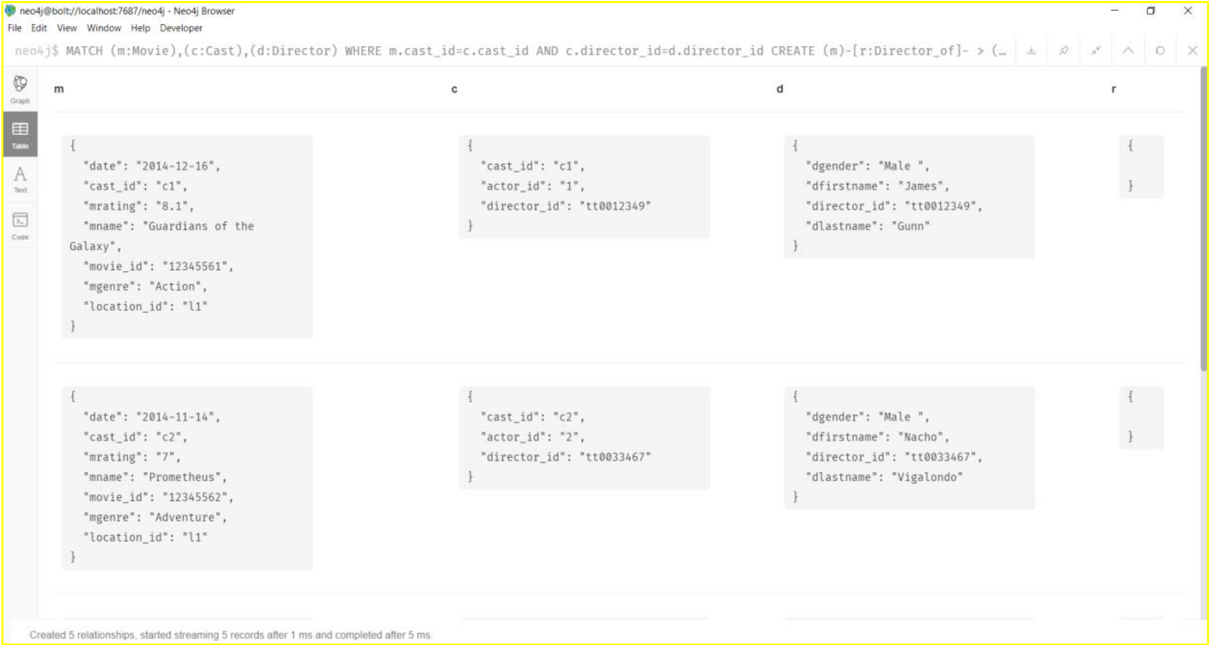
- **Query_1: Getting relationships between Movie, Cast and Director tables and Matching the similar values:**



neo4j@bolt://localhost:7687/neo4j - Neo4j Browser

```
neo4j$ MATCH (m:Movie),(c:Cast),(d:Director) WHERE m.cast_id=c.cast_id AND c.director_id=d.director_id CREATE (m)-[r:Director_of]- > (…
```

Fig. Neo4j Graph Database



neo4j@bolt://localhost:7687/neo4j - Neo4j Browser

```
neo4j$ MATCH (m:Movie),(c:Cast),(d:Director) WHERE m.cast_id=c.cast_id AND c.director_id=d.director_id CREATE (m)-[r:Director_of]- > (…
```

```
{
  "date": "2014-12-16",
  "cast_id": "c1",
  "mrating": "8.1",
  "mname": "Guardians of the
Galaxy",
  "movie_id": "12345561",
  "mgenre": "Action",
  "location_id": "l1"
}
```

```
{
  "cast_id": "c1",
  "actor_id": "1",
  "director_id": "tt0012349"
}
```

```
{
  "dgender": "Male ",
  "dfirstname": "James",
  "director_id": "tt0012349",
  "dlastname": "Gunn"
}
```

```
{
}
```

```
{
  "date": "2014-11-14",
  "cast_id": "c2",
  "mrating": "7",
  "mname": "Prometheus",
  "movie_id": "12345562",
  "mgenre": "Adventure",
  "location_id": "l1"
}
```

```
{
  "cast_id": "c2",
  "actor_id": "2",
  "director_id": "tt0033467"
}
```

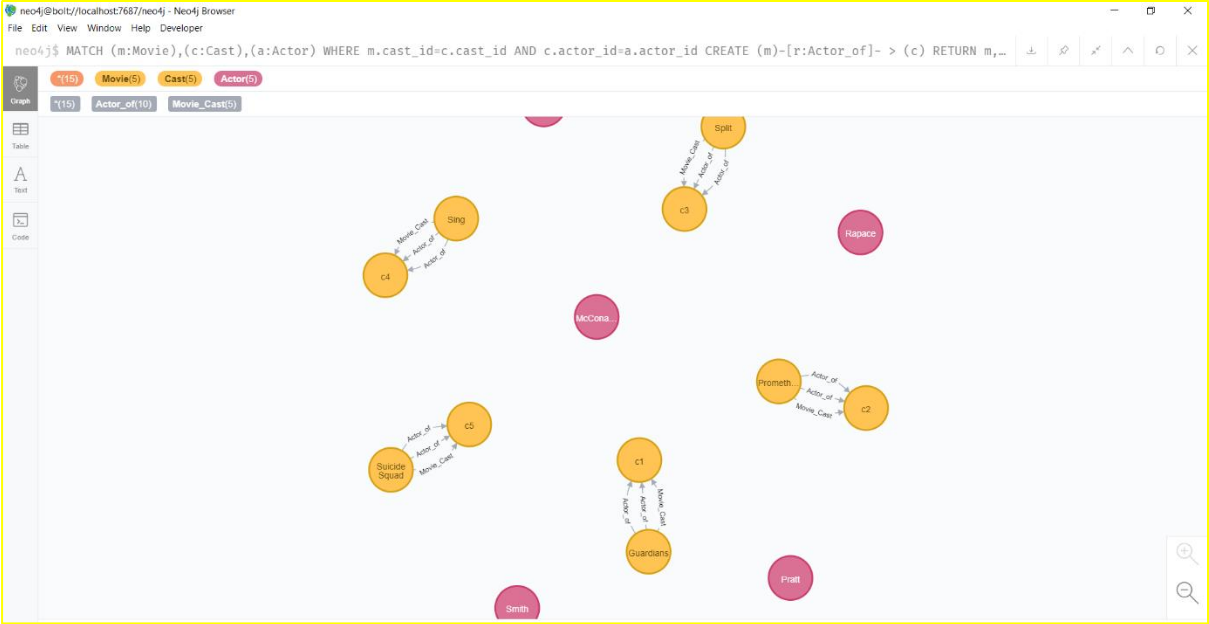
```
{
  "dgender": "Male ",
  "dfirstname": "Nacho",
  "director_id": "tt0033467",
  "dlastname": "Vigalondo"
}
```

```
{
}
```

Created 5 relationships, started streaming 5 records after 1 ms and completed after 5 ms.

Fig. Neo4j Table Database

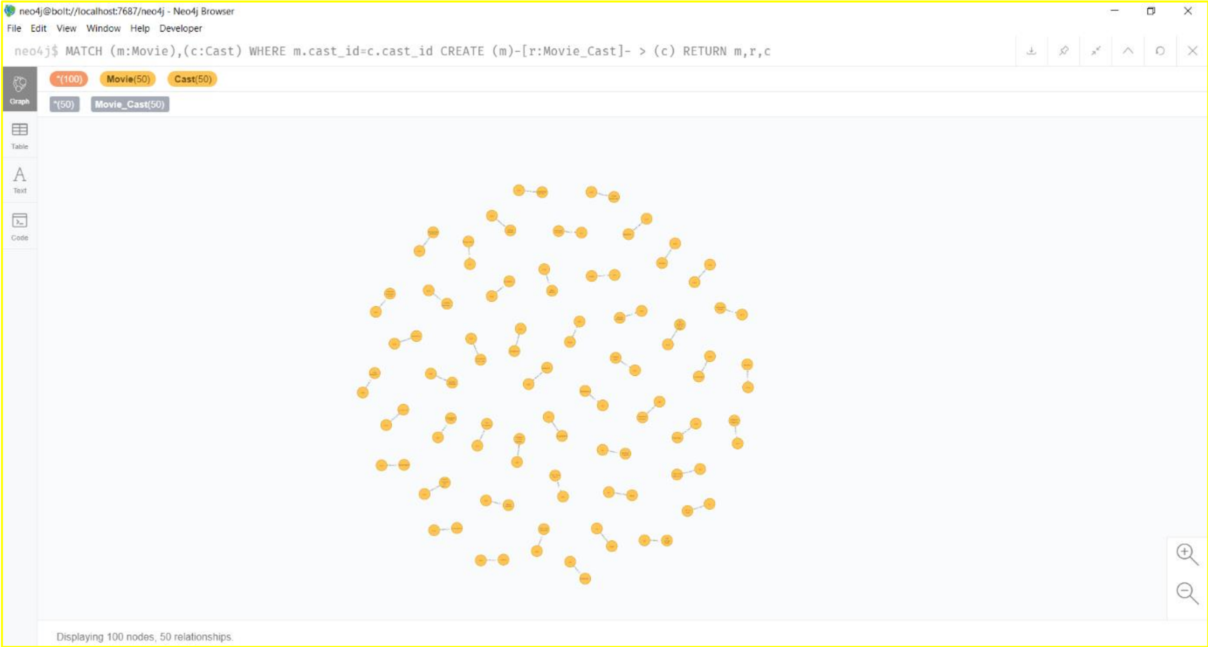
- **Query_2: Getting relationships between Movie, Cast and Actor tables and Matching the similar values:**



```
neo4j$ MATCH (m:Movie),(c:Cast),(a:Actor) WHERE m.cast_id=c.cast_id AND c.actor_id=a.actor_id CREATE (m)-[r:Actor_of]- > (c) RETURN m,…
```

Fig. Neo4j Graph Database

- **Query_3: Getting relationships between Movie and Cast tables and Matching the similar values:**



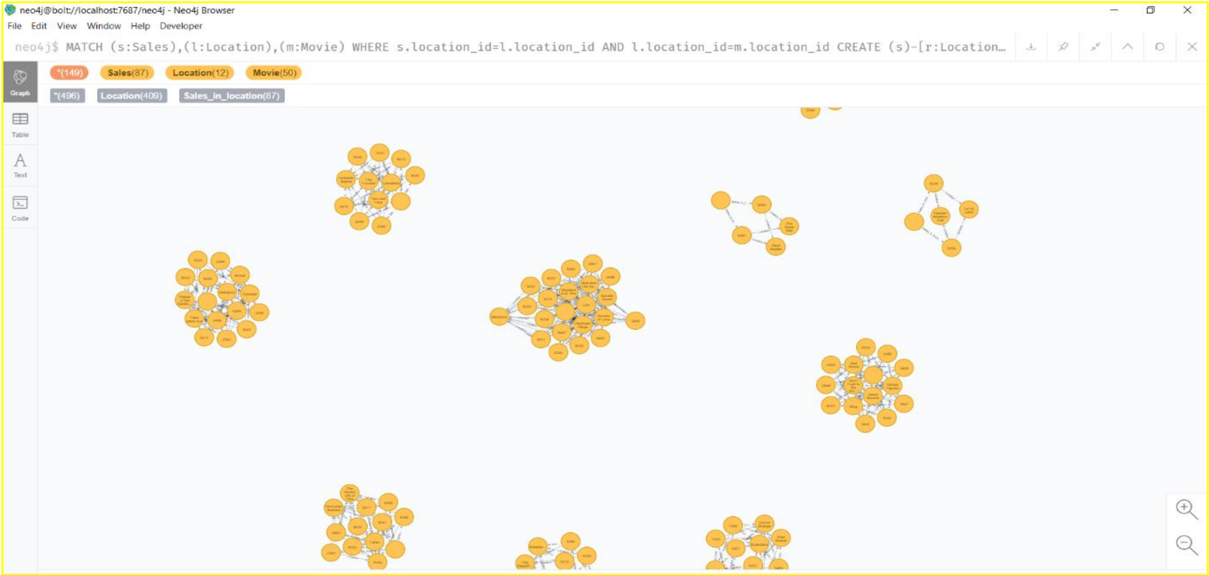
```
neo4j$ MATCH (m:Movie),(c:Cast) WHERE m.cast_id=c.cast_id CREATE (m)-[r:Movie_Cast]- > (c) RETURN m,r,c
```

Displaying 100 nodes, 50 relationships.

neo4j@bolt://localhost:7687/neo4j - Neo4j Browser
File Edit View Window Help Developer
*(100)
Movie(50)
Cast(50)
*(50)
Movie_Cast(50)
Graph
Table
Text
Code

Fig. Neo4j Graph Database

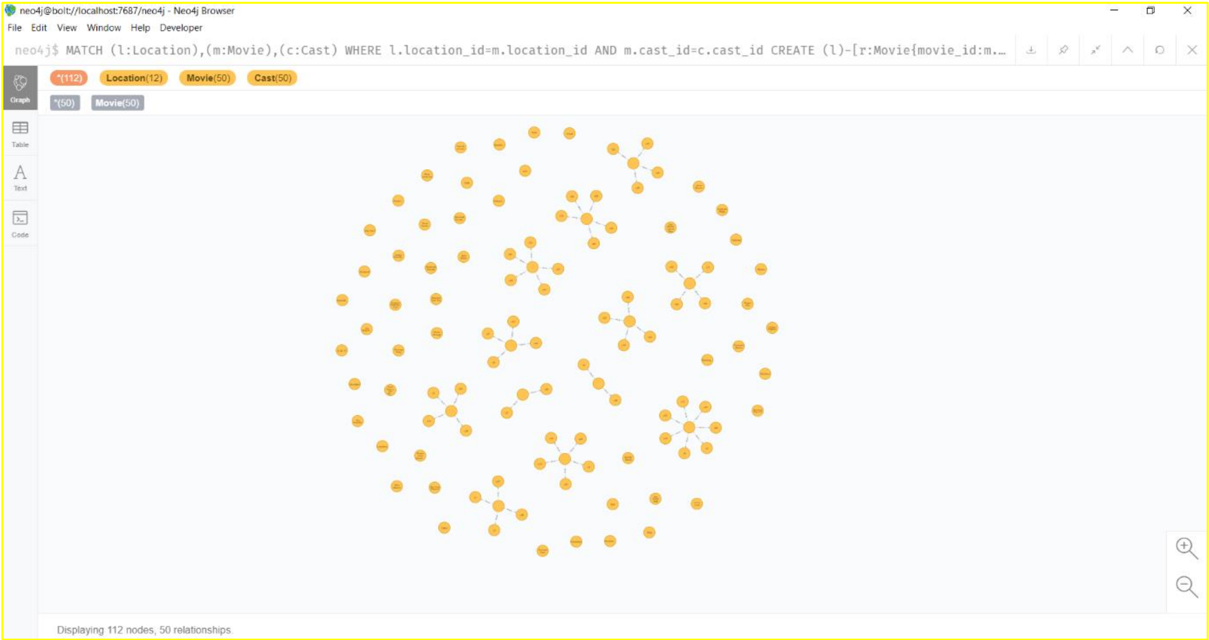
- **Query_4: Getting relationships between Sales, Location and Movie tables and Matching the similar values:**



```
neo4j$ MATCH (s:Sales),(l:Location),(m:Movie) WHERE s.location_id=l.location_id AND l.location_id=m.location_id CREATE (s)-[r:Location…
```

Fig. Neo4j Graph Database

- **Query_5: Getting relationships between Location, Movie and Cast tables and Matching the similar values:**

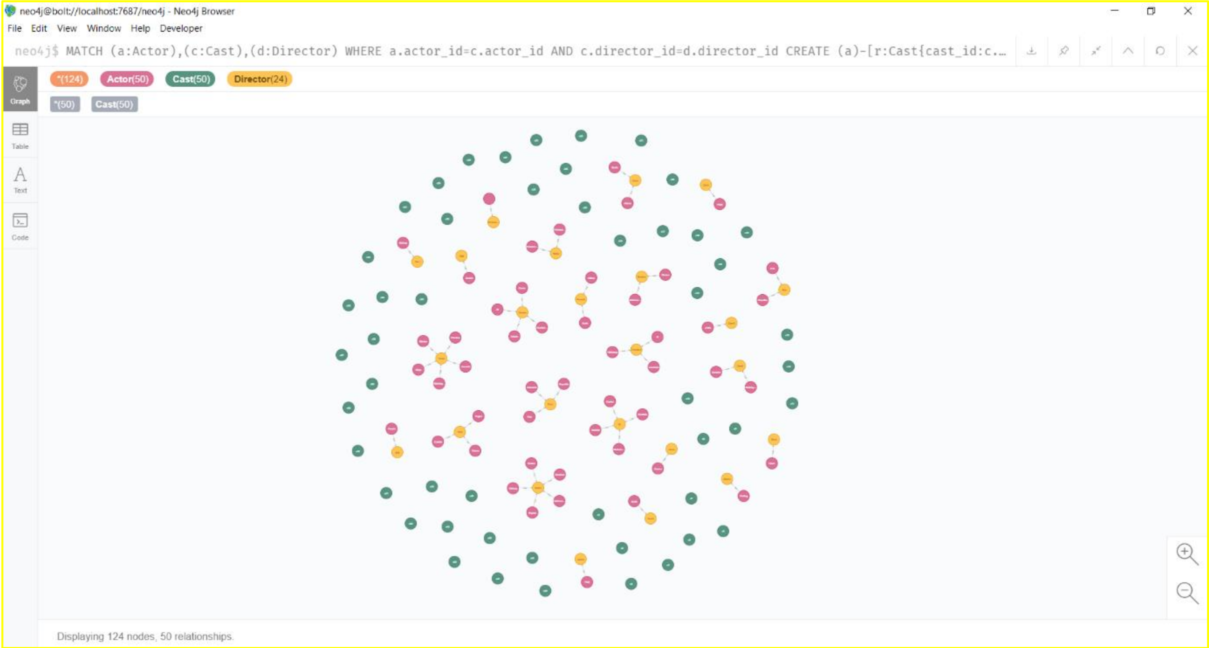

neo4j@bolt://localhost:7687/neo4j - Neo4j Browser
File Edit View Window Help Developer
Graph
Table
Text
Code
*(112)
Location(12)
Movie(50)
Cast(50)
*(50)
Movie(50)

```
neo4j$ MATCH (l:Location),(m:Movie),(c:Cast) WHERE l.location_id=m.location_id AND m.cast_id=c.cast_id CREATE (l)-[r:Movie{movie_id:m.…
```

Displaying 112 nodes, 50 relationships.

Fig. Neo4j Graph Database

- **Query_6: Getting relationships between Actor, Cast and Director tables and Matching the similar values:**

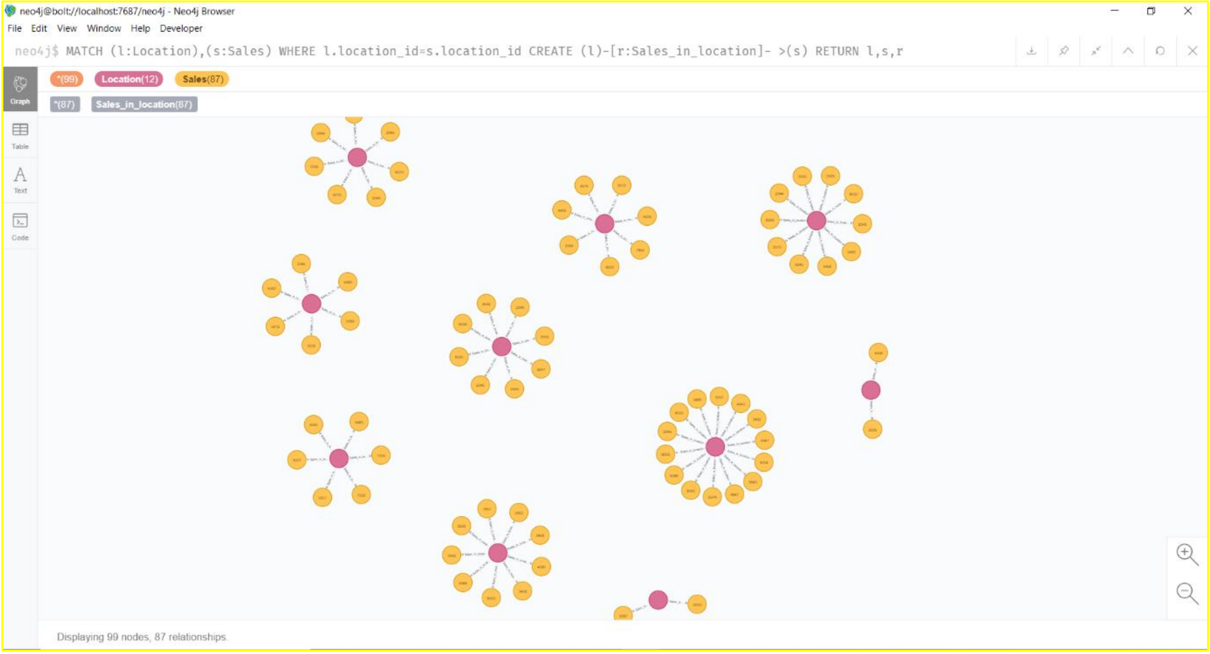

neo4j@bolt://localhost:7687/neo4j - Neo4j Browser
File Edit View Window Help Developer
*(124) Actor(50) Cast(50) Director(24)
*(50) Cast(50)
Graph
Table
Text
Code

```
neo4j$ MATCH (a:Actor),(c:Cast),(d:Director) WHERE a.actor_id=c.actor_id AND c.director_id=d.director_id CREATE (a)-[r:Cast{cast_id:c.…
```

Displaying 124 nodes, 50 relationships.

Fig. Neo4j Graph Database

- **Query_7: Getting relationships between Location and Sales tables and Matching the similar values:**


neo4j@bolt://localhost:7687/neo4j - Neo4j Browser
File Edit View Window Help Developer
*(99) Location(12) Sales(87)
*(87) Sales_in_location(87)
Graph
Table
Text
Code

```
neo4j$ MATCH (l:Location),(s:Sales) WHERE l.location_id=s.location_id CREATE (l)-[r:Sales_in_location]- >(s) RETURN l,s,r
```

Displaying 99 nodes, 87 relationships.

Fig. Neo4j Graph Database

8. Conclusion:

- We have used relational and dimensional modelling to design a data warehouse.
- We fetched the data and moved it across the data warehouse tables using ETL and we used SQL queries to insert data into the fact table and dimensions.
- We have used different types of visualization using SSRS package and Tableau.
- We used Neo4j to create graphical databases and created relations between them.

9. Bibliography:

- Data Source: https://www.dofactory.com/sql/sample-database
- For ETL:
- SQL Server: https://docs.microsoft.com/en-us/sql/ssms/download-sql-server-management-studio-ssms?view=sql-server-ver15 [Accessed 13 April 2020]
- Studied through ZagiSales Example on Moodle: https://elearning.dbs.ie/mod/folder/view.php?id=853499
- ERD and StarSchema: https://app.diagrams.net/
- https://epublications.marquette.edu/cgi/viewcontent.cgi?article=1118&context=theses_open
- For SSRS:
- SQL Server Data Tools for Visual Studio 2013: https://www.microsoft.com/en-ie/download/details.aspx?id=42313
- Studied through Examples on Moodle:
- https://elearning.dbs.ie/mod/page/view.php?id=855545
- http://ssrstutorials.blogspot.com/2012/07/lesson-4-creating-your-first-ssrs.html
- SSRS Procedure:
- https://www.c-sharpcorner.com/article/creating-your-firsttabular-ssrs-report-with-and-without-wiza/
- https://www.networkworld.com/article/2258477/chapter-1--introduction-to-sql-server-reporting-services--ssrs-.html
- Microsoft.com. (2017). *Download Microsoft SQL Server 2017 Reporting Services from Official Microsoft Download Center*. [online] Available at: https://www.microsoft.com/en-us/download/confirmation.aspx?id=55252
- Neo4j Graph Database Platform. (2017). *Neo4j Graph Platform – The Leader in Graph Databases*. [online] Available at: https://neo4j.com/ [Accessed April 13,2020].

10. Appendix - Neo4J Code:

## Create Actor

```
LOAD CSV WITH HEADERS FROM "file:///Actor.csv" AS line
CREATE(a:Actor) SET a={actor_id:line.ActorId,
afirstname:line.AFirstName,alastname:line.ALastName,agender:line.AGender}
RETURN a
```

## Create Cast

```
LOAD CSV WITH HEADERS FROM "file:///Cast.csv" AS line CREATE(c:Cast)
SET c={cast_id:line.CastID,director_id:line.DirectorID,actor_id:line.ActorId}
RETURN c
```

## Create Director

```
LOAD CSV WITH HEADERS FROM "file:///Director.csv" AS line
CREATE(d:Director) SET
d={director_id:line.DirectorId,dfirstname:line.DFirstName,dlastname:line.DLastName,dgender:line.Dgender} RETURN d
```

## Create Location

```
LOAD CSV WITH HEADERS FROM "file:///Location.csv" AS line
CREATE(l:Location) SET
l={location_id:line.LocationId,region:line.Region,country:line.Country,continent:line.Continet} RETURN l
```

## Create Movie

```
LOAD CSV WITH HEADERS FROM "file:///Movie.csv" AS line
CREATE(m:Movie) SET
m={movie_id:line.MovieId,mname:line.MName,mrating:line.Mrating,mgenre:line.Mgenre,date:line.Date,cast_id:line.CastID,location_id:line.LocationID} RETURN m
```

## Create Sales

```
LOAD CSV WITH HEADERS FROM "file:///Sales.csv" AS line CREATE(s:Sales)
SET
s={sales_id:line.SalesId,noticketssold:line.NoTicketsSold,priceofticket:line.PriceofTicket,location_id:line.LocationID} RETURN s
```

## Creating Constraint script for Unique Value:

```
CREATE CONSTRAINT ON(a:Actor) ASSERT a.actor_id IS UNIQUE

CREATE CONSTRAINT ON(c:Cast) ASSERT c.cast_id IS UNIQUE

CREATE CONSTRAINT ON(d:Director) ASSERT d.director_id IS UNIQUE

CREATE CONSTRAINT ON(l:Location) ASSERT l.location_id IS UNIQUE
CREATE CONSTRAINT ON(m:Movie) ASSERT m.movie_id IS UNIQUE

CREATE CONSTRAINT ON(s:Sales) ASSERT s.sales_id IS UNIQUE
```

## MATCH QUERY:

## Query_1

```
MATCH (l:Location),(s:Sales) WHERE l.location_id=s.location_id CREATE (l)-
[r:Sales_in_location]- >(s) RETURN l,s,r
```

## Query_2

```
MATCH (a:Actor),(c:Cast),(d:Director) WHERE a.actor_id=c.actor_id AND
c.director_id=d.director_id CREATE (a)-[r:Cast{cast_id:c.cast_id}]- >(d) RETURN
a, c, d, r
```

## Query_3

```
MATCH (l:Location),(m:Movie),(c:Cast) WHERE l.location_id=m.location_id
AND m.cast_id=c.cast_id CREATE (l)-
[r:Movie{movie_id:m.movie_id,mname:m.mname,mrating:m.mrating,mgenre:m.mg
enre,date:m.date}]- >(c) RETURN l,m,c,r
```

## Query_4

```
MATCH (s:Sales),(l:Location),(m:Movie) WHERE s.location_id=l.location_id
AND l.location_id=m.location_id CREATE (s)-
[r:Location{region:l.region,country:l.country,continent:l.continent}]- >(m)
RETURN s,l,m,r
```

## Query_5

```
MATCH (m:Movie),(c:Cast) WHERE m.cast_id=c.cast_id CREATE (m)-
[r:Movie_Cast]- > (c) RETURN m,r,c
```

## Query_6

```
MATCH (m:Movie),(c:Cast),(a:Actor) WHERE m.cast_id=c.cast_id AND
c.actor_id=a.actor_id CREATE (m)-[r:Actor_of]- > (c) RETURN m,c,a,r limit 5
```

## Query_7

```
MATCH (m:Movie),(c:Cast),(d:Director) WHERE m.cast_id=c.cast_id AND
c.director_id=d.director_id CREATE (m)-[r:Director_of]- > (c) RETURN m,c,d,r
limit 5
```