**Big Data Platforms**

**Module Code: MMI227050**

**Course Work 01**

**From Geeta Yadav**

**(S2120665)**

**For Professor: James Paterson**

**Acknowledgment: This course is work is truly made by me.**

Contents

[1. Introduction 3](#_Toc98179734)

[2. Part A - Online Auction Scenario for Database creation 4](#_Toc98179735)

[A.1 Data Model 5](#_Toc98179736)

[A.2. Mongo DB Database 7](#_Toc98179737)

[A.3 Queries of the key use cases 16](#_Toc98179738)

[Part B – Neo4j Graph Database 22](#_Toc98179739)

[B1. Creating Music Library database 23](#_Toc98179740)

[B2. Queries 28](#_Toc98179741)

[References 31](#_Toc98179742)

# Introduction

In this Big Data Platform course work Part-A, creating a data model on the online auction scenario in the JSON form and creating a database using MongoDB methods. In Part-B, will create a Neo4j database on the music library scenario, which represent the data in the graph form.

JavaScript Object Notation is widely used for records interchange and garage due to the fact it's far the only and simplest answer for operating with records the usage of an internet browser. JSON systems may be evaluated and used as JavaScript gadgets inside the internet browser environment, and records is embedded inside the records meaning is self-describing.

MongoDB is an open supply NoSQL database control program. NoSQL is used as an opportunity to conventional relational databases. NoSQL databases are pretty beneficial for running with massive units of allotted data. MongoDB is a tool that might manage document-oriented information, store or retrieve information [3].

Task A includes creation of data model for application on online auction scenario in JSON form and then inserting data and running queries in command prompt using MongoDB based on different aspects like auction details including user who creating an auction, auction details, Items details, bid and bidders details.[3]

Task B includes creation of graph database on the on the music library scenario and presenting model using Neo4j, using Cypher queries to takeout information from the database.

# Part A - Online Auction Scenario for Database creation

For the Online scenario there could be the multiple solutions for creating a database and data model. There are the different relationships considered for built the model collection. For an example, Auction can have a multiple bids and users, bidder can bid on multiple auction Items, as shown in below given model (figure1).

In Auction collection I am going to take objects on the basis of the relation scenarios which can give the required output on the further queries such as provide all item auctions of specified categories, which user created how many auctions, new bid happened on auction, give all the details with status and bid of the auction. As per the Scenario the model will be based on the many to many relationships.

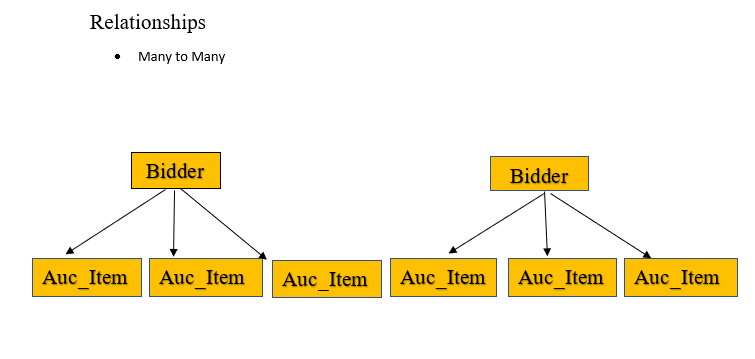


Figure.1

## A.1 Data Model

The model will be preparing on the given scenario including two collections one is the name of Auction and another from the name of Auction\_Bidders. Find the below mention JSON model with example in figure 2.

**Auction Collection**

The objects taken in the Auction collection on the basis of detail including user name who created the auction, given the eight different users, each auction has different Item and its name, category and description. Auction details like closing date, close status of the auction when end date reached, current price of the bid and updated price of the bid when bid done. Created the ten different Auction IDs in the Auction table, with the three different categories (Jewellery, Toys and Gadgets).  
Updating price manually in Auction collection as per the highest price put by the bidder weather the status is in open or close state.

**Auction\_Bidders Collection**

The objects taken in the Auction Bidders collection in such a way so that it could link with the Auction collection and provide the details of Bid and Bidders. Created twenty-six Bidder Id’s which include details of the Bidder, their name, mail ID and the address, bid date and time and bid amount. Each bidder data has an Auction ID on which he is bidding that auction ID would help in link the bidders table to the Auction table and give the all details that who bid on which item, at what price, when and so on.

**Schema and the properties of the model**

**Collection: Auction**

* \_id – It is the unique identifier of the Auction number.
* Auc\_Creator- Is the user of the auction who is hosting the item auction.
* Prod\_Name- It is the name of the particular item.
* Class- defining the category of the item i.e. (Jewellery, Toys and Gadgets).
* Details- It is the description of the product, its feature and quality.
* Auc\_EndDate- The end date of the Auction.
* Bid\_Opening\_Rates- The starting rate of the bid.
* Auc\_State- It is referring the status of the auction.
* Bid\_Updated\_Price- It is the updated price of the bid which is the final price of the id on which auction made.

**Collection: Auction\_Bidders**

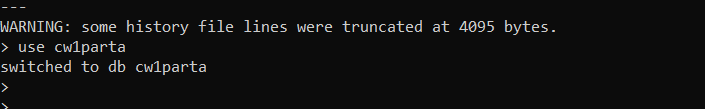
* \_id- The unique identifier of the bidder’s detail.
* BidderName- Name of the Bidder
* EmailID- Email ID of the Bidder
* Location- Address of the Bidder
* AuctionID- The unique Id of the Auction
* bid datetime- The date and time of the Bid happened.
* bid amount- Amount put by the bidder

Figure.2

The above collection model designed as per the fulfilling requirements of the online auction scenario where Auction table can give the details of the user who created the auction and all the details of the Item, status and end date of the auction and also the updated price of the Bid. And Auction\_Bidders collection contains the Bidder and Bid details with the Auction ID.

# A.2. Mongo DB Database

**Step.1** Start the MongoDB server in the system command prompt, it specify the data directory, then run the Mongo command in another command prompt terminal and run the first command to create the database. Find the below command with example and also find screenshot of command prompt. use command here creating a new database called cw1parta.



**Step.2** Inserting and creating the JSON document for creating an Auction collection. Find the below command with example and screenshot also.

**Query:** **db. Auction.insertOne ( )** command is help in inserting the values in the collection. With the help of this command inserting ten documents of the auctions. Can see screenshot of command prompt.

> db. Auction.insertOne

(

{

"\_id":"Auc\_1.1",

"Auc\_Creator":"Geeta Yadav”,

"Prod\_Name”:”Diamond Ring",

"Class":"Jewellery",

"Details":"1.34 Carat white Gold Ct Diamonds","Auc\_EndDate":"14/3/2022",

"Bid\_Opening\_Rates":"3000",

"Auc\_State":"Open","Bid\_Updated\_Price":"3030"

}

){ "acknowledged": true, "insertedId":" Auc\_1.5" }

**Result:** With the find () command can show the insert data result of the Auction collection. Below find the pasted query and the screenshot of the result.

|  |
| --- |
| > db.Auction.find().pretty()  {  "\_id" : "Auc\_1.1",  "Auc\_Creator" : "Geeta Yadav",  "Prod\_Name" : "Diamond Ring",  "Class" : "Jewellery",  "Details" : "1.34 Carat white Gold Ct Diamonds",  "Auc\_EndDate" : "14/3/2022",  "Bid\_Opening\_Rates" : 3000,  "Auc\_State" : "Open",  "Bid\_Updated\_Price" : 3030  }  {  "\_id" : "Auc\_1.2",  "Auc\_Creator" : "Geeta Yadav",  "Prod\_Name" : "Yellow Gold Necklace",  "Class" : "Jewellery",  "Details" : "22 kt. Yellow Gold Ct – 0.35 Diamond",  "Auc\_EndDate" : "5/3/2022",  "Bid\_Opening\_Rates" : 4000,  "Auc\_State" : "Close",  "Bid\_Updated\_Price" : 4030  }  {  "\_id" : "Auc\_1.3",  "Auc\_Creator" : "Angela",  "Prod\_Name" : "Diamond Bracelet",  "Class" : "Jewellery",  "Detail" : "18 carat Platinum – Bracelet 20.10 Ct Diamonds- Diamond",  "AUC\_EndDate" : "14/3/2022",  "Bid\_OpeningRates" : 2000,  "Auc\_State" : "open",  "Bid\_Updated\_Price" : 2040  }  {  "\_id" : "Auc\_1.4",  "Auc\_Creator" : "Rose",  "Prod\_Name" : "Dino Robot",  "Class" : "Toys",  "Detail" : "Dinosaur Toy is made up of solid plastic and has durable battery",  "AUC\_EndDate" : "8/3/2022",  "Bid\_OpeningRates" : 4000,  "Auc\_State" : "Close",  "Bid\_Updated\_Price" : 4080  }  {  "\_id" : " Auc\_1.5",  "Auc\_Creator" : "Rose",  "Prod\_Name" : " Army Fighter Jet ",  "Class" : "Toys",  "Detail" : " Fighter Jet is made of solid plastic and has durable battery, green colour and recommended age 9 years and above ",  "AUC\_EndDate" : "9/3/2022",  "Bid\_OpeningRates" : 309,  "Auc\_State" : "Close",  "Bid\_Updated\_Price" : 380  }  {  "\_id" : " Auc\_1.6,",  "Auc\_Creator" : "Mr.Bean",  "Prod\_Name" : " Barbie Doll House ",  "Class" : "Toys",  "Detail" : "“ Beautiful pink Barbie Doll house available with all accessories ",  "AUC\_EndDate" : "4/3/2022",  "Bid\_OpeningRates" : 200,  "Auc\_State" : "Close",  "Bid\_Updated\_Price" : 240  }  {  "\_id" : " Auc\_1.7,",  "Auc\_Creator" : "Arnold",  "Prod\_Name" : " Lenovo Laptop ",  "Class" : "Gadgets",  "Detail" : " ThinkPad black Lenovo laptop contains features are 16gb ram, 250 storage ,i7 series",  "AUC\_EndDate" : "14/3/2022",  "Bid\_OpeningRates" : 305,  "Auc\_State" : "Open",  "Bid\_Updated\_Price" : 385  }  {  "\_id" : " Auc\_1.8,",  "Auc\_Creator" : "Tiger",  "Prod\_Name" : " Smart Watch ",  "Class" : "Gadgets",  "Detail" : " White Fashionable smart watch with advance features like display notifications and alerts ",  "AUC\_EndDate" : "14/3/2022",  "Bid\_OpeningRates" : 700,  "Auc\_State" : "Open",  "Bid\_Updated\_Price" : 800  }  {  "\_id" : "Auc\_1.9",  "Auc\_Creator" : "Deepak",  "Prod\_Name" : "Electrical Mercedese car",  "Class" : "Toys",  "Detail" : "Cute design kids electrical car, design Mercedese, Black colour with 2 years warranty",  "AUC\_EndDate" : "15/3/2022",  "Bid\_OpeningRates" : 1005,  "Auc\_State" : "Open",  "Bid\_Updated\_Price" : 1050  }  {  "\_id" : " Auc\_1.10",  "Auc\_Creator" : "Harry",  "Prod\_Name" : " I-Phone 12 Pro Max",  "Class" : "Gadgets",  "Detail" : "Brand New Black colour I-Phone 256 GB",  "AUC\_EndDate" : "9/3/2022",  "Bid\_OpeningRates" : 990,  "Auc\_State" : "Close",  "Bid\_Updated\_Price" : 1120  } |

**Screenshot:**



Secondly, inserting and creating data for Auction\_Bidders collection, using insertMany command for the multiple bidders data. Find the insert() and the result below.

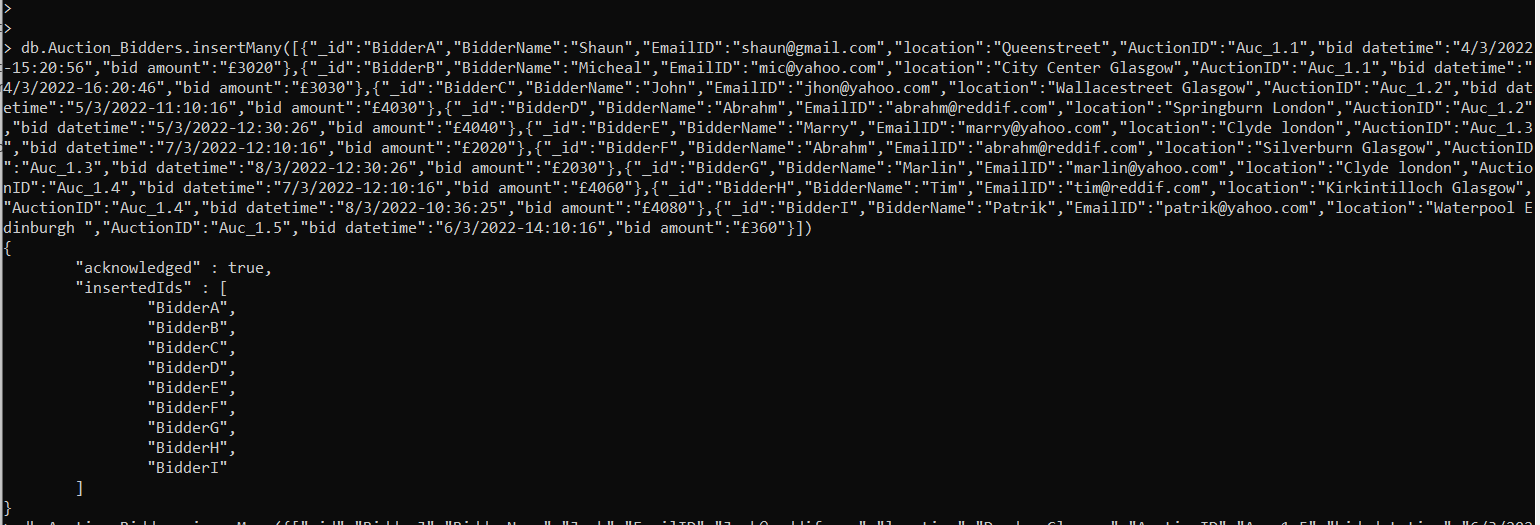
**Query:**

|  |
| --- |
| db.auction\_bidders.insertOne(  (    {  "\_id":"BidderA",  "BidderName":"Shaun",  "EmailID":"shaun@gmail.com",  "location":"Queenstreet",  "AuctionID":"Auc\_1.1",  "bid datetime":"4/3/2022-15:20:56",  "bid amount":"£3020"  }  ) |

**Result:** With the find () command can show the insert data result of the Auction\_Bidders collection. Below find the pasted query and the screenshot of the result.

|  |
| --- |
| > db.Auction\_Bidders.find().pretty()  {  "\_id" : "BidderA",  "BidderName" : "Shaun",  "EmailID" : "shaun@gmail.com",  "location" : "Queenstreet",  "AuctionID" : "Auc\_1.1",  "bid datetime" : "4/3/2022-15:20:56",  "bid amount" : "£3020"  }  {  "\_id" : "BidderB",  "BidderName" : "Micheal",  "EmailID" : "mic@yahoo.com",  "location" : "City Center Glasgow",  "AuctionID" : "Auc\_1.1",  "bid datetime" : "4/3/2022-16:20:46",  "bid amount" : "£3030"  }  {  "\_id" : "BidderC",  "BidderName" : "John",  "EmailID" : "jhon@yahoo.com",  "location" : "Wallacestreet Glasgow",  "AuctionID" : "Auc\_1.2",  "bid datetime" : "5/3/2022-11:10:16",  "bid amount" : "£4030"  }  {  "\_id" : "BidderD",  "BidderName" : "Abrahm",  "EmailID" : "abrahm@reddif.com",  "location" : "Springburn London",  "AuctionID" : "Auc\_1.2",  "bid datetime" : "5/3/2022-12:30:26",  "bid amount" : "£4040"  }  {  "\_id" : "BidderE",  "BidderName" : "Marry",  "EmailID" : "marry@yahoo.com",  "location" : "Clyde london",  "AuctionID" : "Auc\_1.3",  "bid datetime" : "7/3/2022-12:10:16",  "bid amount" : "£2020"  }  {  "\_id" : "BidderF",  "BidderName" : "Abrahm",  "EmailID" : "abrahm@reddif.com",  "location" : "Silverburn Glasgow",  "AuctionID" : "Auc\_1.3",  "bid datetime" : "8/3/2022-12:30:26",  "bid amount" : "£2030"  }  {  "\_id" : "BidderG",  "BidderName" : "Marlin",  "EmailID" : "marlin@yahoo.com",  "location" : "Clyde london",  "AuctionID" : "Auc\_1.4",  "bid datetime" : "7/3/2022-12:10:16",  "bid amount" : "£4060"  }  {  "\_id" : "BidderH",  "BidderName" : "Tim",  "EmailID" : "tim@reddif.com",  "location" : "Kirkintilloch Glasgow",  "AuctionID" : "Auc\_1.4",  "bid datetime" : "8/3/2022-10:36:25",  "bid amount" : "£4080"  }  {  "\_id" : "BidderI",  "BidderName" : "Patrik",  "EmailID" : "patrik@yahoo.com",  "location" : "Waterpool Edinburgh ",  "AuctionID" : "Auc\_1.5",  "bid datetime" : "6/3/2022-14:10:16",  "bid amount" : "£360"  }  {  "\_id" : "BidderJ",  "BidderName" : "Jack",  "EmailID" : "Jack@reddif.com",  "location" : "Dundee Glasgow",  "AuctionID" : "Auc\_1.5",  "bid datetime" : "6/3/2022-20:15:45",  "bid amount" : "£380"  }  {  "\_id" : "BidderK",  "BidderName" : "Shaun",  "EmailID" : "shaun@gmail.com",  "location" : "Queenstreet",  "AuctionID" : "Auc\_1.6",  "bid datetime" : "4/3/2022-15:20:56",  "bid amount" : "£210"  }  {  "\_id" : "BidderL",  "BidderName" : "Micheal",  "EmailID" : "mic@yahoo.com",  "location" : "City Center Glasgow",  "AuctionID" : "Auc\_1.6",  "bid datetime" : "4/3/2022-16:20:46",  "bid amount" : "£230"  }  {  "\_id" : "BidderM",  "BidderName" : "Marry",  "EmailID" : "marry@yahoo.com",  "location" : "Manchester",  "AuctionID" : "Auc\_1.7",  "bid datetime" : "6/3/2022-18:10:16",  "bid amount" : "£380"  }  {  "\_id" : "BidderN",  "BidderName" : "Abrahm",  "EmailID" : "abrahm@reddif.com",  "location" : "Leeds",  "AuctionID" : "Auc\_1.7",  "bid datetime" : "7/3/2022-17:30:26",  "bid amount" : "£375"  }  {  "\_id" : "BidderO",  "BidderName" : "Marry",  "EmailID" : "marry@gmail.com",  "location" : "Oban",  "AuctionID" : "Auc\_1.8",  "bid datetime" : "8/3/2022-15:10:16",  "bid amount" : "£780"  }  {  "\_id" : "BidderP",  "BidderName" : "Mr.Wick",  "EmailID" : "wick@reddif.com",  "location" : "London",  "AuctionID" : "Auc\_1.8",  "bid datetime" : "8/3/2022-18:30:26",  "bid amount" : "£795"  }  {  "\_id" : "BidderQ",  "BidderName" : "Shiva",  "EmailID" : "shiva@gmail.com",  "location" : "Oban",  "AuctionID" : "Auc\_1.9",  "bid datetime" : "5/3/2022-15:10:16",  "bid amount" : "£1035"  }  {  "\_id" : "BidderR",  "BidderName" : "Mr.Rambo",  "EmailID" : "rambo@reddif.com",  "location" : "Dundee",  "AuctionID" : "Auc\_1.9",  "bid datetime" : "9/3/2022-18:30:26",  "bid amount" : "£1050"  }  {  "\_id" : "BidderS",  "BidderName" : "Daniel",  "EmailID" : "danny@reddif.com",  "location" : "Stirling",  "AuctionID" : "Auc\_1.10",  "bid datetime" : "8/3/2022-20:15:45",  "bid amount" : "£1120"  }  {  "\_id" : "BidderT",  "BidderName" : "Rock",  "EmailID" : "rock@gmail.com",  "location" : "Manchester",  "AuctionID" : "Auc\_1.10",  "bid datetime" : "7/3/2022-15:20:56",  "bid amount" : "£1110"  } |

**Screenshot:**



# A.3 Queries of the key use cases

**1. Find all auctions for items in a specified category**

**Query**: **-**

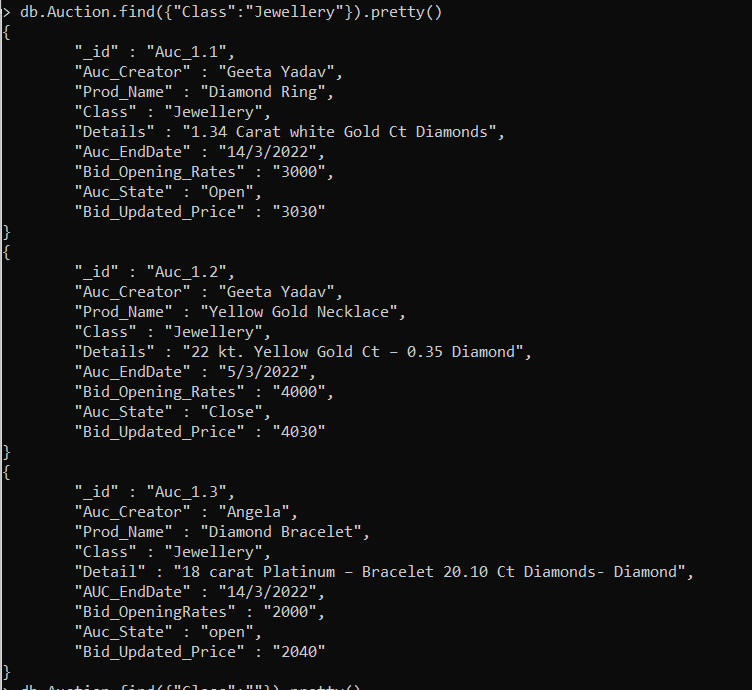
|  |
| --- |
| > db. Auction. Find({"Class”:"Jewellery"}). Pretty () |

The above query is providing the all-items details falls in the category or Jewellery in the Auction data. In this query **DB** define as a database, Auction is the name of the collection data, **find** command helps in find the details as specifying **({"Class”:"Jewellery"}),** where class is a category and Jewellery is the name of the category.

**Result**: - Find the three Items of the jewellery category in the Auction collection data. Similarly, we can find the other categories item auctions also.

|  |
| --- |
| > db.Auction.find({"Class":"Jewellery"}).pretty()  {  "\_id" : "Auc\_1.1",  "Auc\_Creator" : "Geeta Yadav",  "Prod\_Name" : "Diamond Ring",  "Class" : "Jewellery",  "Details" : "1.34 Carat white Gold Ct Diamonds",  "Auc\_EndDate" : "14/3/2022",  "Bid\_Opening\_Rates" : 3000,  "Auc\_State" : "Open",  "Bid\_Updated\_Price" : 3030  }  {  "\_id" : "Auc\_1.2",  "Auc\_Creator" : "Geeta Yadav",  "Prod\_Name" : "Yellow Gold Necklace",  "Class" : "Jewellery",  "Details" : "22 kt. Yellow Gold Ct – 0.35 Diamond",  "Auc\_EndDate" : "5/3/2022",  "Bid\_Opening\_Rates" : 4000,  "Auc\_State" : "Close",  "Bid\_Updated\_Price" : 4030  }  {  "\_id" : "Auc\_1.3",  "Auc\_Creator" : "Angela",  "Prod\_Name" : "Diamond Bracelet",  "Class" : "Jewellery",  "Detail" : "18 carat Platinum – Bracelet 20.10 Ct Diamonds- Diamond",  "AUC\_EndDate" : "14/3/2022",  "Bid\_OpeningRates" : 2000,  "Auc\_State" : "open",  "Bid\_Updated\_Price" : 2040  } |

**Screenshot:**



**2. Find all auctions created by a specified user**

**Query:** Below query will show the all auctions created by the Geeta Yadav, find the result below.

|  |
| --- |
| db. Auction.find({"Auc\_Creator":"Geeta Yadav"}). pretty () |

**Result**:

|  |
| --- |
| > db.Auction.find({"Auc\_Creator":"Geeta Yadav"}).pretty()  {  "\_id" : "Auc\_1.1",  "Auc\_Creator" : "Geeta Yadav",  "Prod\_Name" : "Diamond Ring",  "Class" : "Jewellery",  "Details" : "1.34 Carat white Gold Ct Diamonds",  "Auc\_EndDate" : "14/3/2022",  "Bid\_Opening\_Rates" : "3000",  "Auc\_State" : "Open",  "Bid\_Updated\_Price" : "3030"  }  {  "\_id" : "Auc\_1.2",  "Auc\_Creator" : "Geeta Yadav",  "Prod\_Name" : "Yellow Gold Necklace",  "Class" : "Jewellery",  "Details" : "22 kt. Yellow Gold Ct – 0.35 Diamond",  "Auc\_EndDate" : "5/3/2022",  "Bid\_Opening\_Rates" : "4000",  "Auc\_State" : "Close",  "Bid\_Updated\_Price" : "4030"  } |

**Screenshot:**



**3. Make a new bid on an auction**

**Query**: Using insertOne command to showing the new bid, with the same details of bidder and bid mentioned in the

|  |
| --- |
| > db.Auction\_Bidders.insertOne({"\_id":"Bidderxyz","BidderName":"Ricky","EmailID":"ricky@gmail.com","location":"Kirkintilloch","AuctionID":"Auc\_1.10","bid datetime":"8/3/2022-19:14:18","bid amount":"1130"})  { "acknowledged" : true, "insertedId" : "Bidderxyz" } |

**Result:**

|  |
| --- |
| > db.Auction\_Bidders.find({"\_id":"Bidderxyz"}).pretty()  {  "\_id" : "Bidderxyz",  "BidderName" : "Ricky",  "EmailID" : "ricky@gmail.com",  "location" : "Kirkintilloch",  "AuctionID" : "Auc\_1.10",  "bid datetime" : "8/3/2022-19:14:18",  "bid amount" : "1130"  }  > |

**4. Show all the details of an auction including bids and status**

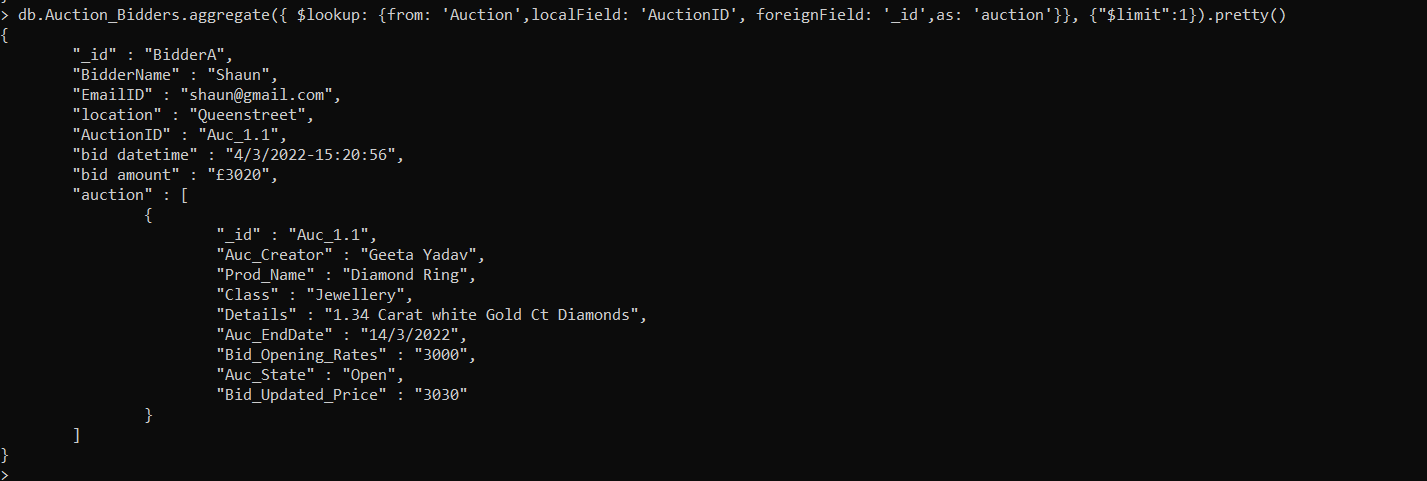
**Query**: The mentioned query is giving the all details of an Auction including bid and status.

|  |
| --- |
| db. Auction\_Bidders. Aggregate({ $lookup: {from: 'Auction',localField: 'AuctionID', foreignField: '\_id',as: 'auction'}}, {"$limit":1}).pretty() |

**Results:**

|  |
| --- |
| {  "\_id" : "BidderA",  "BidderName" : "Shaun",  "EmailID" : "shaun@gmail.com",  "location" : "Queenstreet",  "AuctionID" : "Auc\_1.1",  "bid datetime" : "4/3/2022-15:20:56",  "bid amount" : "£3020",  "auction" : [  {  "\_id" : "Auc\_1.1",  "Auc\_Creator" : "Geeta Yadav",  "Prod\_Name" : "Diamond Ring",  "Class" : "Jewellery",  "Details" : "1.34 Carat white Gold Ct Diamonds",  "Auc\_EndDate" : "14/3/2022",  "Bid\_Opening\_Rates" : "3000",  "Auc\_State" : "Open",  "Bid\_Updated\_Price" : "3030"  }  ]  } |

**Screenshot**



**5. MongoDB aggregation pipeline query to find the average current price for each item category**

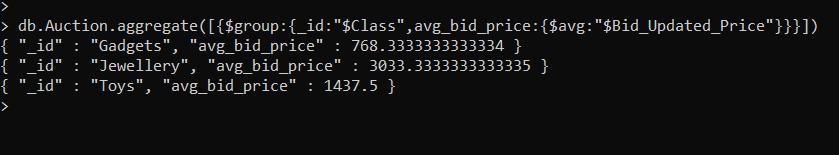
**Query:** The below command will give the average current price for each item category, find the result also.

|  |
| --- |
| Auction.aggregate([{$group:{\_id:"$Class",avg\_bid\_price:{$avg:"$Bid\_Updated\_Price"}}}]) |

**Result:**

|  |
| --- |
| > db.Auction.aggregate([{$group:{\_id:"$Class",avg\_bid\_price:{$avg:"$Bid\_Updated\_Price"}}}])  { "\_id" : "Gadgets", "avg\_bid\_price" : 768.3333333333334 }  { "\_id" : "Jewellery", "avg\_bid\_price" : 3033.3333333333335 }  { "\_id" : "Toys", "avg\_bid\_price" : 1437.5 } |

**Screenshot:**



# Part B – Neo4j Graph Database

In this task creating a Neo4j graph database on the scenario of the Music Library. For this graph database I am creating two albums, each album will have four songs and each song will have three different artists with different performance like (Singer, writer and musician).

Also, try to link the graph in such a way where one artist involved in two activities for example, he has written the song and also performing on it.

Cypher statements makes graph data easy to understand and read by describing the nodes, relationships **(Relationship Symbol 🡪)** and properties with text-based symbols [1].

**Schema and description are given below of the database queries:**

* Create- command to create the statement for the graph.
* album1, 1stsong and Artist1- They are all the node labels.
* Title- Property to define the title of the album, song or artist.
* Length- variable to define the length of the music
* Name- variable to define the name of the album, song or artist
* AppearsOn- Is the edge label between album and song
* PerformedOn- Is the edge label between song and Artist
* Performance\_Type- Is the property of the performance of the Artist.
* Return- Is the command, which gives result on you node, edge and record statement.

# B1. Creating Music Library database

Further, creating a node, edges, and record step by step using a Cypher statement.

Step1. Create statement will create the Album of title GreatestHits.

|  |
| --- |
| CREATE (album1:Album {Title:"GreatestHits"}), |

Step2. In this step creating the list of the songs with the titles and length.

|  |
| --- |
| CREATE(1stsong:song {Title:"Titanic",Length:'3min47sec'}),  CREATE(2ndsong:song {Title:"Sanorita",Length:'2min50sec'}),  CREATE(3rdsong:song {Title:"Cold",Length:'2min50sec'}),  CREATE(4thsong:song {Title:"Luv me like you do",Length:'3min46sec'}), |

Step3. Linking and making the relationship as per the above two statements and between the album and song, where all four songs will appear on the Album with help of return command. The graph output shown in below figure

|  |
| --- |
| CREATE (album1:SongsAlbum {Title:"GreatestHits"})-[:AppearsOn]->(song1:song {Title:"Titanic",Length:'3.47'}),  (album1)-[:AppearsOn]->(song2:song {Title:"Sanorita",Length:'2.50'}),  (album1)-[:AppearsOn]->(song3:song {Title:"Cold",Length:'2.50'}),  (album1)-[:AppearsOn]->(song4:song {Title:"Luv me like you do",Length:'3.46'})  RETURN album1,song1,song2,song3,song4 |

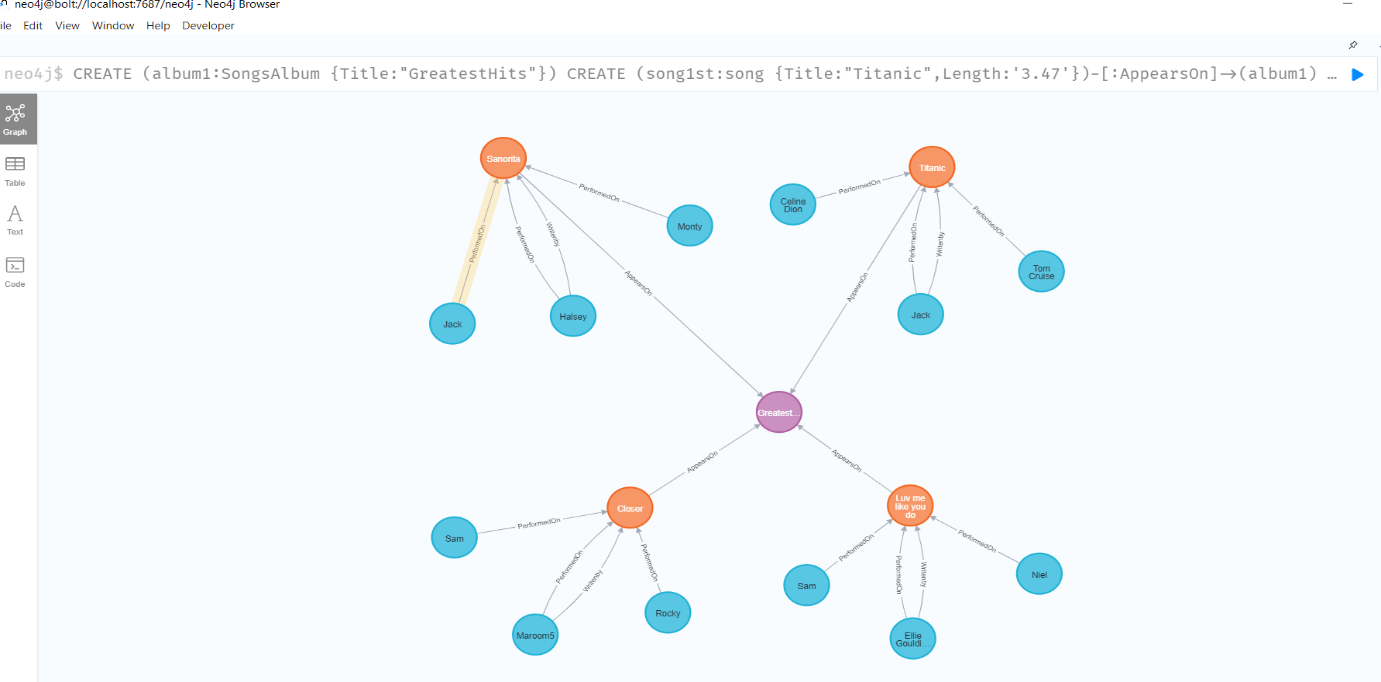


Step4. Now, creating the three artists on each song of the album, linking them bu the edge PerformanceOn with the Performance\_Type properties.

|  |
| --- |
| create (Artst1:Artist {Name:"Celine Dion"})-[:PerformedOn{Performance\_Type:"Singer"}]->(song1st)  create (Artst2:Artist {Name:"Jack"})-[:PerformedOn{Performance\_Type:"Guitarist"}]->(song1st)  create (Artst3:Artist {Name:"Tom Cruise"})-[:PerformedOn{Performance\_Type:"Pianist"}]->(song1st)  create (Artst2)-[:Writenby]->(song1st)  create (Artst4:Artist {Name:"Halsey"})-[:PerformedOn{Performance\_Type:"Singer"}]->(song2nd)  create (Artst5:Artist {Name:"Jack"})-[:PerformedOn{Performance\_Type:"Guitarist"}]->(song2nd)  create (Artst6:Artist {Name:"Monty"})-[:PerformedOn{Performance\_Type:"Drummer"}]->(song2nd)  create (Artst4)-[:Writenby]->(song2nd)  create (Artst7:Artist {Name:"Maroom5"})-[:PerformedOn{Performance\_Type:"Singer"}]->(song3rd)  create (Artst8:Artist {Name:"Rocky"})-[:PerformedOn{Performance\_Type:"Violinist"}]->(song3rd)  create (Artst9:Artist {Name:"Sam"})-[:PerformedOn{Performance\_Type:"Trumpist"}]->(song3rd)  create (Artst7)-[:Writenby]->(song3rd)  create (Artst10:Artist {Name:"Ellie Goulding"})-[:PerformedOn{Performance\_Type:"Singer"}]->(song4th)  create (Artst11:Artist {Name:"Niel"})-[:PerformedOn{Performance\_Type:"Pianist"}]->(song4th)  create (Artst12:Artist {Name:"Sam"})-[:PerformedOn{Performance\_Type:"TromBonist"}]->(song4th)  create (Artst10)-[:Writenby]->(song4th) |

Step4. Return statement will show the graph of the 1st Album with the relationship of songs and artists. As shown in the figure

|  |
| --- |
| RETURN album1,song1st,song2nd,song3rd,song4th,Artst1,Artst2,Artst3,Artst4,Artst5,Artst6,Artst7,Artst8,Artst9,Artst10,Artst11,Artst12 |



Step5. Now similarly, creating a data for the second album and make the all required relationship with the songs and Artist.

|  |
| --- |
| CREATE (album2:SongsAlbum {Title:"ForeverMemories"}) |

Step6. Creating four songs on the album2, which will appear on the album2.

|  |
| --- |
| CREATE (song5th:song {Title:"Let me luv you",Length:2.55})-[:AppearsOn]->(album2)  CREATE (song6th:song {Title:"Amplifier",Length:3.40})-[:AppearsOn]->(album2)  CREATE (song7th:song {Title:"Hip's dont lie",Length:3.50})-[:AppearsOn]->(album2)  CREATE (song8th:song {Title:"Believer",Length:3.25})-[:AppearsOn]->(album2) |

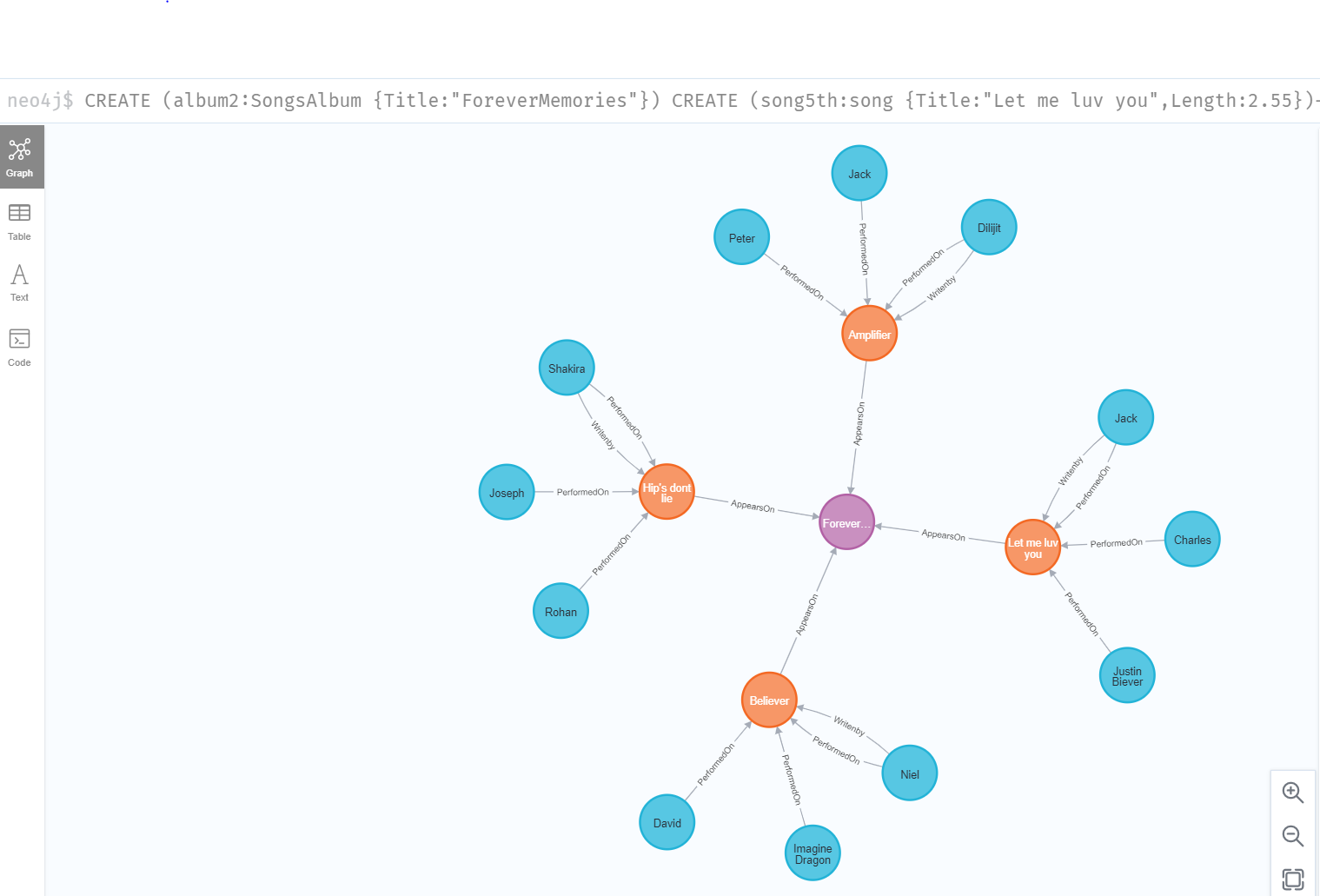
Step7. Creating and linking three songs on each songs, total twelve artists will appear on the four songs.

|  |
| --- |
| create (Artst13:Artist {Name:"Justin Biever"})-[:PerformedOn{Performance\_Type:"Singer"}]->(song5th)  create (Artst14:Artist {Name:"Charles"})-[:PerformedOn{Performance\_Type:"Drummer"}]->(song5th)  create (Artst15:Artist {Name:"Jack"})-[:PerformedOn{Performance\_Type:"Guitarist"}]->(song5th)  create (Artst15)-[:Writenby]->(song5th)  create (Artst16:Artist {Name:"Dilijit"})-[:PerformedOn{Performance\_Type:"Singer"}]->(song6th)  create (Artst17:Artist {Name:"Jack"})-[:PerformedOn{Performance\_Type:"Guitarist"}]->(song6th)  create (Artst18:Artist {Name:"Peter"})-[:PerformedOn{Performance\_Type:"Drummer"}]->(song6th)  create (Artst16)-[:Writenby]->(song6th)  create (Artst19:Artist {Name:"Shakira"})-[:PerformedOn{Performance\_Type:"Singer"}]->(song7th)  create (Artst20:Artist {Name:"Rohan"})-[:PerformedOn{Performance\_Type:"Pianist"}]->(song7th)  create (Artst21:Artist {Name:"Joseph"})-[:PerformedOn{Performance\_Type:"Tubist"}]->(song7th)  create (Artst19)-[:Writenby]->(song7th)  create (Artst22:Artist {Name:"Imagine Dragon"})-[:PerformedOn{Performance\_Type:"Singer"}]->(song8th)  create (Artst23:Artist {Name:"Niel"})-[:PerformedOn{Performance\_Type:"Pianist"}]->(song8th)  create (Artst24:Artist {Name:"David"})-[:PerformedOn{Performance\_Type:"Drummer"}]->(song8th)  create (Artst23)-[:Writenby]->(song8th) |

Step8. In the final step, returning all the nodes graph, it can be shown in figure:

|  |
| --- |
| RETURN album,album2,song1st,song2nd,song3rd,song4th,song5th,song6th,song7th,song8th,Artst1,Artst2,Artst3,Artst4,Artst5,Artst6,Artst7,Artst8,Artst9,Artst10,Artst11,Artst12,Artst13,Artst14,Artst15,Artst16,Artst17,Artst18,Artst19,Artst20,Artst21,Artst22,Artst23,Artst24 |

**Find the graph result of the album2 after creating and linking the all songs and artists.**



# B2. Queries

**Ques.1 Show all the nodes and edges in the database?**

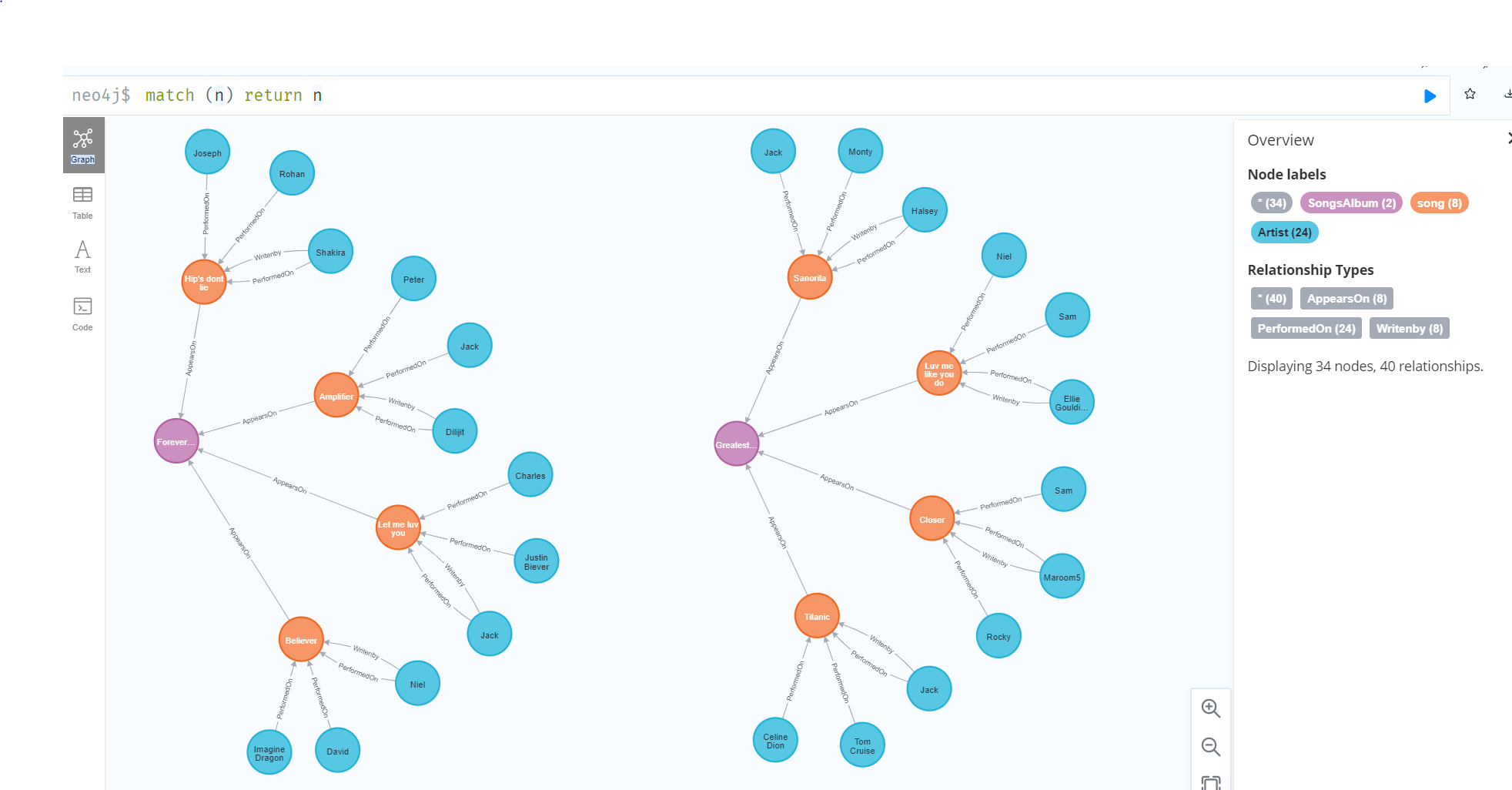
**Query Statement:**

match (n) return n gives the result as below: -

**Node labels** contains 2 SongsAlbum, eight songs and 24 Artist, total 34 node labels.

**Relationship Types** contains 8 AppearsOn, 24 PerformedOn, 8 writenby and total 40 relationships**.**

Displaying 34 nodes, 40 relationships.



**Ques.2 Show all the songs on a specified album?**

**Query Statement:**

MATCH (n:song)-[:AppearsOn]-(:SongsAlbum {Title:'ForeverMemories'}) return n

**Result:**

╒════════════════════════════════════════╕

│"n" │

╞════════════════════════════════════════╡

│{"Length":3.25,"Title":"Believer"} │

├────────────────────────────────────────┤

│{"Length":3.5,"Title":"Hip's dont lie"} │

├────────────────────────────────────────┤

│{"Length":3.4,"Title":"Amplifier"} │

├────────────────────────────────────────┤

│{"Length":2.55,"Title":"Let me luv you"}│

└────────────────────────────────────────┘

**Ques.3 Show all the musicians who have both written and performed on any song?**

**Query Statement:**

**match (n:Artist)-[:PerformedOn]->(x:song)return n**

**Result:**

╒═════════════════════════╕

│"n" │

╞═════════════════════════╡

│{"Name":"Tom Cruise"} │

├─────────────────────────┤

│{"Name":"Jack"} │

├─────────────────────────┤

│{"Name":"Celine Dion"} │

├─────────────────────────┤

│{"Name":"Monty"} │

├─────────────────────────┤

│{"Name":"Jack"} │

├─────────────────────────┤

│{"Name":"Halsey"} │

├─────────────────────────┤

│{"Name":"Sam"} │

├─────────────────────────┤

│{"Name":"Rocky"} │

├─────────────────────────┤

│{"Name":"Maroom5"} │

├─────────────────────────┤

│{"Name":"Sam"} │

├─────────────────────────┤

│{"Name":"Niel"} │

├─────────────────────────┤

│{"Name":"Ellie Goulding"}│

├─────────────────────────┤

│{"Name":"Jack"} │

├─────────────────────────┤

│{"Name":"Charles"} │

├─────────────────────────┤

│{"Name":"Justin Biever"} │

├─────────────────────────┤

│{"Name":"Peter"} │

├─────────────────────────┤

│{"Name":"Jack"} │

├─────────────────────────┤

│{"Name":"Dilijit"} │

├─────────────────────────┤

│{"Name":"Joseph"} │

├─────────────────────────┤

│{"Name":"Rohan"} │

├─────────────────────────┤

│{"Name":"Shakira"} │

├─────────────────────────┤

│{"Name":"David"} │

├─────────────────────────┤

│{"Name":"Imagine Dragon"}│

├─────────────────────────┤

│{"Name":"Niel"} │

└─────────────────────────┘

**Ques.4 Show all the musicians who have both written and performed on any song?**

**Query Statement:**

**match (n:Artist)-[:PerformedOn]->(x:song)return n**

**Result:**

╒═════════════════════╕

│"n" │

╞═════════════════════╡

│{"Name":"Niel"} │

├─────────────────────┤

│{"Name":"Tom Cruise"}│

└─────────────────────┘

**Ques.5 Show all the musicians who have both written and performed on any song?**

**Query Statement:**

**match (n:song)-[:AppearsOn]-:SongsAlbum{Title:'GreatestHits'})return sum(n.Length)**

**Result:**

╒══════════════════╕

│"sum(n.Length)" │

╞══════════════════╡

│11.930000000000001│

└──────────────────┘

# References

1. https://neo4j.com/product/ - figure

2. <https://neo4j.com/docs/graphql-manual/current/>

3. https://docs.mongodb.com/manual/tutorial/getting-started/