**Question:** **Multiple studies have been done comparing the performance of graph databases and relational ones. Use a relational database such as Adventure Works (or you can use any other relational database of your choice) as data source and implement a graph database in Neo4J. Write any seven queries in SQL and CQL to use as evidence to compare both databases and discuss the results to demonstrate the differences of storing/retrieving data in/from relational and graph databases. Note: These queries must provide useful information from the database.**

We have used **Chinook Database** for the assignment question to implement graph database in Neo4J. Chinook database is a relational database that is designed for digital music store. It includes the tables for Artist, Album, Customer, Invoice, InvoiceLine. As an initial step the database file .bak is downloaded is been used in SQL Server to restore the database.

**Key Tables:**

* Artist and Album: Manages music data
* Customer: Stores information about customer details
* Invoice and InvoiceLines: Manages purchase details
* Track and Genre: Includes track details of album

**Relationship between tables:**

* Each Album is link to Artist
* Track belong to Album with defined Genre
* Each Customer has Invoice, and Invoice includes InvoiceLines

**Importing Data into Neo4J:**

Loading excel sheet saved from SQL Server to Neo4J for each tables using Cypher Query Language (CQL).

**Different Nodes properties and relationship:**

**Nodes:**

* Artist: (ArtistId, Name}
* Album: {AlbumId, Title}
* Track: {TrackId, Name, Composer, Milliseconds, Bytes, UnitPrice}
* Customer: {CustomerId, FirstName, LastName, Email, Country}
* Invoice: {InvoiceId, InvoiceDate, Total}

**Relationship:**

* (:Artist)-[CREATED]->(:Album)
* (:Album)-[CONTAINS]->(:Track)
* (:Track)-[HAS\_Genre]->(:Genre)
* (:Customer)-[PURCHASED]->(:Invoice)

**Creating Nodes and Relationship:**

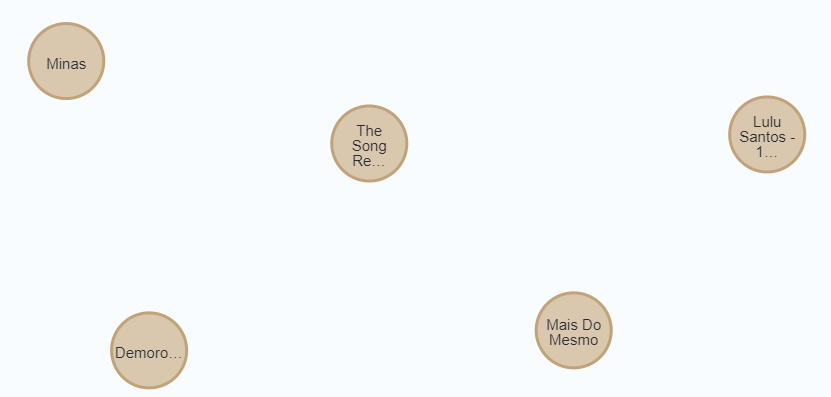
* LOAD CSV WITH HEADERS FROM 'file:///artist.csv' as row CREATE (a:Artist) SET a={artist\_id: row.ArtistId, artist\_name: row.Name} return a
* CREATE CONSTRAINT FOR (a:Artist) REQUIRE a.artist\_id IS UNIQUE



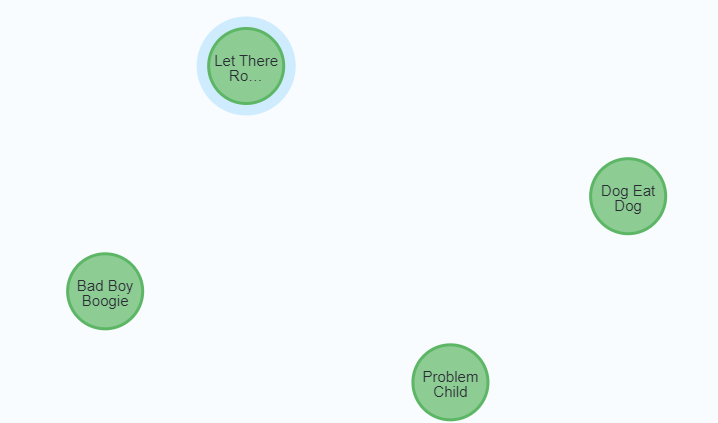
* LOAD CSV WITH HEADERS FROM 'file:///Album.csv' as row CREATE (a:Album) SET a = {album\_id:row.AlbumId, album\_title:row.Title, artist\_id:row.ArtistId}

return a

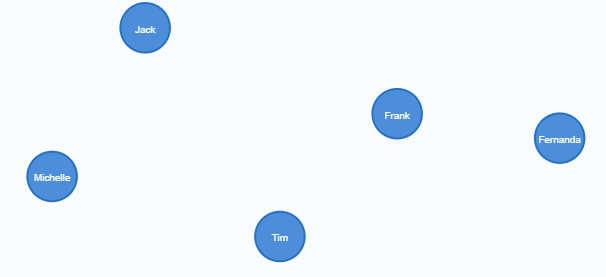
* CREATE CONSTRAINT FOR (a:Album) REQUIRE a.album\_id IS UNIQUE



* LOAD CSV WITH HEADERS FROM 'file:///track.csv' AS row CREATE (t:Track) SET t= {track\_id:row.TrackId, track\_name:row.Name, album\_id:row.AlbumId, media\_typeid:row.MediaTypeId, genre\_id:row.GenreId, composer:row.Composer, milli\_seconds:row.Milliseconds, bytes:row.Bytes, unit\_price:row.UnitPrice} return t
* CREATE CONSTRAINT FOR (t:Track) REQUIRE t.track\_id IS UNIQUE



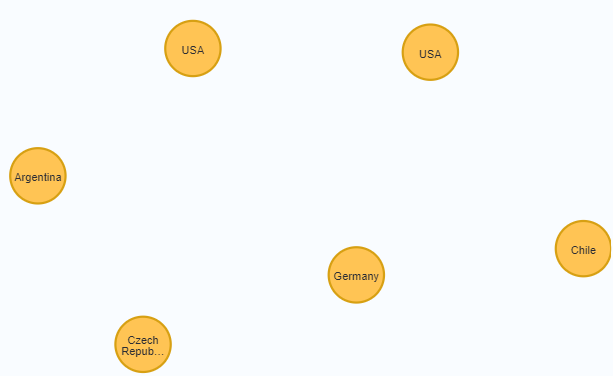
* LOAD CSV WITH HEADERS FROM 'file:///customer.csv' as row CREATE (c:Customer) SET c={customer\_id:row.CustomerId, first\_name:row.FirstName, last\_name:row.LastName, company:row.Company, address:row.Address, city:row.City, state:row.State, country:row.Country, postal\_code: row.PostalCode, phone:row.Phone, fax:row.Fax, email:row.Email, support\_rep\_id: row.SupportRepId} return c
* CREATE CONSTRAINT FOR (c:Customer) REQUIRE c.customer\_id IS UNIQUE



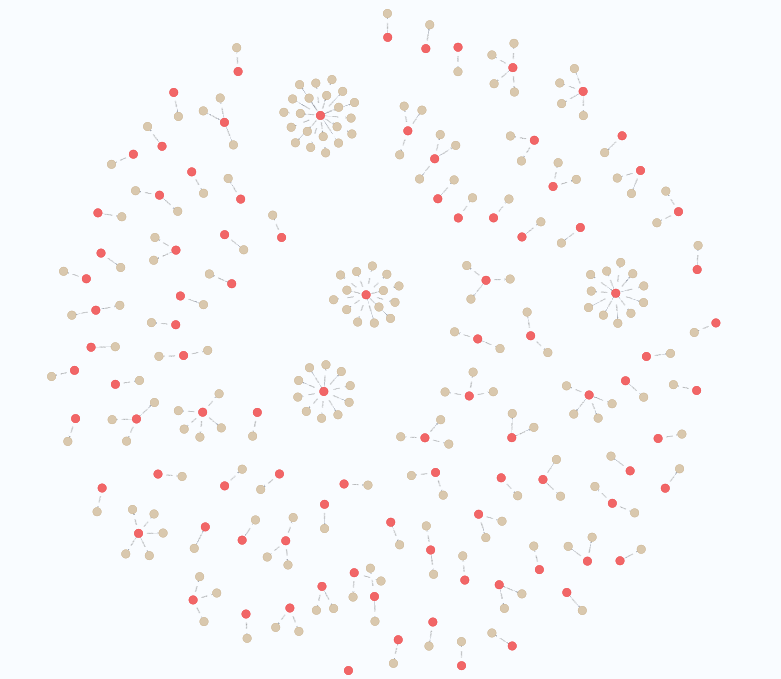
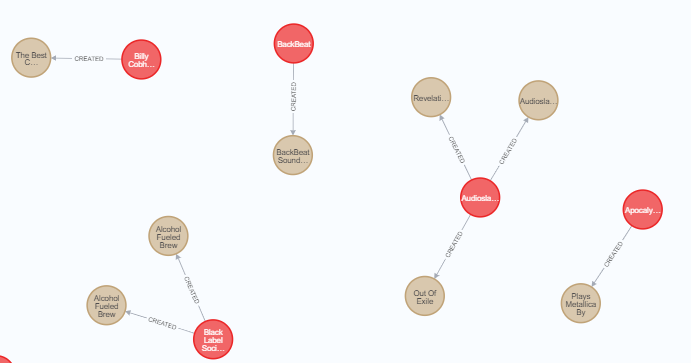
* LOAD CSV WITH HEADERS FROM 'file:///invoice.csv' AS row CREATE

(i:Invoice) SET i={invoice\_id:row.InvoiceId, customer\_id:row.CustomerId, invoice\_date:row.InvoiceDate, billing\_address:row.BillingAddress, billing\_city:row.BillingCity,billing\_state:row.BillingState, billing\_country:row.BillingCountry, billing\_postal\_code:row.BillingPostalCode, total\_bill:row.Total} return i

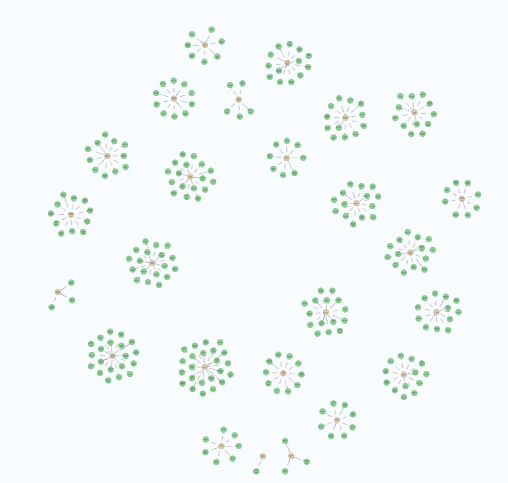
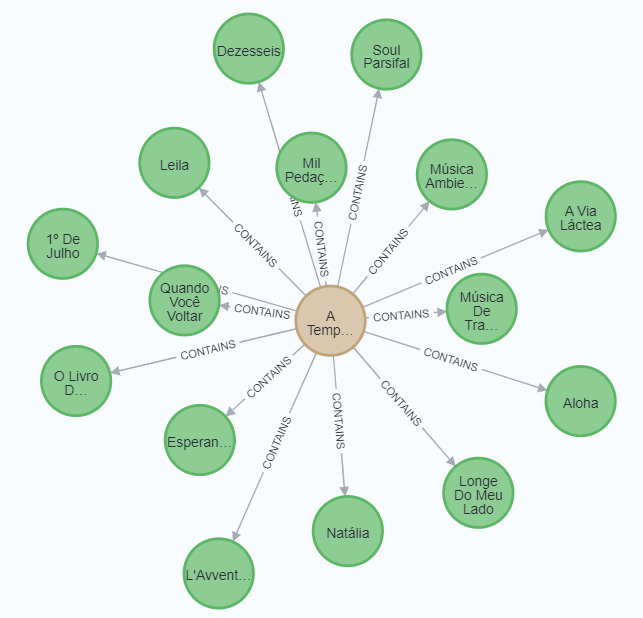
* CREATE CONSTRAINT FOR (i:Invoice) REQUIRE i.invoice\_id IS UNIQUE



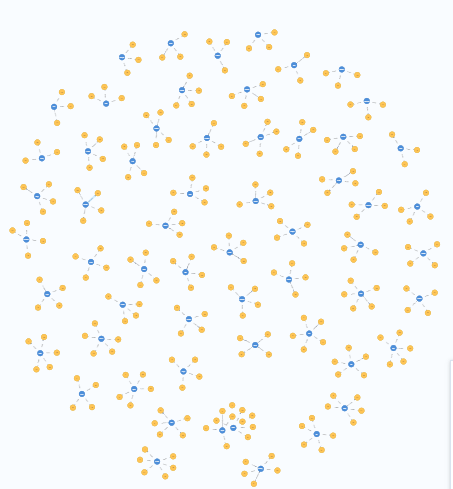
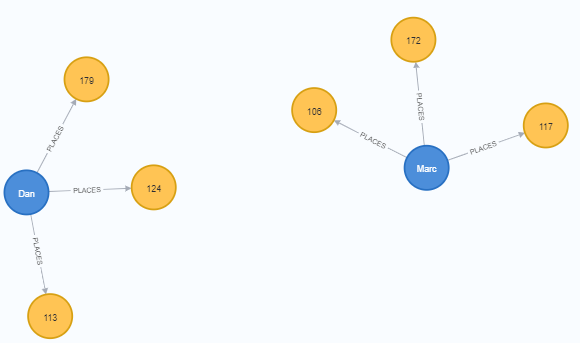
* MATCH (a:Artist),(b:Album) where a.artist\_id=b.artist\_id CREATE (a)-[c:CREATED]->(b) return a,c,b

* MATCH (a:Album),(t:Track) where a.album\_id=t.album\_id CREATE (a)-[c:CONTAINS]->(t) return a,c,t

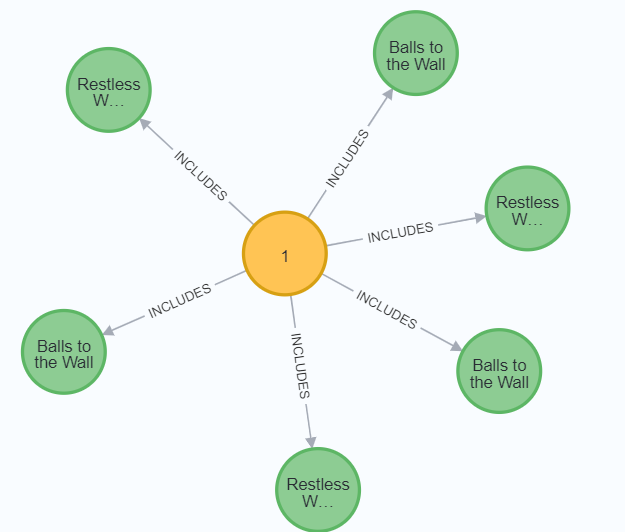
* MATCH (c:Customer),(i:Invoice) where i.customer\_id=c.customer\_id CREATE (c)-[p:PLACES]->(i) return c,p,i

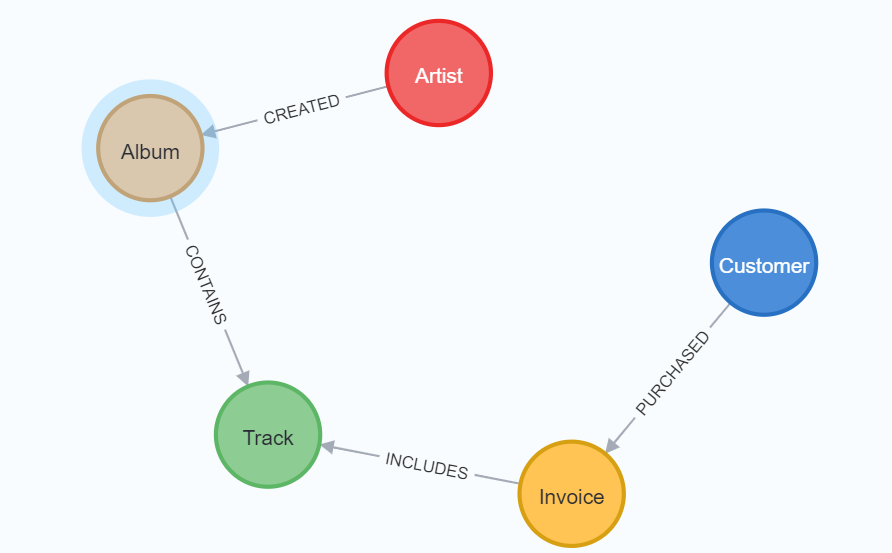
* LOAD CSV WITH HEADERS FROM 'file:///genre.csv' as row CREATE (a:Genre) SET a={genre\_id: row.GenreId, genre\_name: row.Name} return a
* LOAD CSV WITH HEADERS FROM 'file:///invoiceline.csv' AS row

MATCH (invoice:Invoice {invoice\_id: row.InvoiceId})

MATCH (track:Track {track\_id: row.TrackId}) CREATE (invoice)-[:INCLUDES]->(track)



**Overall Schema Visualization:** CALL db.schema.visualization()



**Lets compare queries with respect to Structured Query Language (SQL) and Cypher Query Language (CQL)**

|  |  |
| --- | --- |
| **Query 1: Retrieve all albums created by specific artist** | |
| SELECT Album.Title AS Album\_Title  FROM Album  JOIN Artist ON Album.ArtistId = Artist.ArtistId  WHERE Artist.Name = 'Metallica';    Data is stored with use of foreign key. Query is evaluated joining Album and Artist tables on ground of ArtistId column.  Where clause helps in filtering the specific artist.  Complexity of Join increases on having more relationship.  Also if there are large records, join can be slower. | MATCH (artist:Artist {artist\_name: 'Metallica'})-[:CREATED]->(album:Album)  RETURN album.album\_title;    Data is stored as nodes (Artist, Album) and a relationship is defined between them as [CREATED].  Instead of Join, Graph relationship is made that performs faster retrieval. |
| **Query 2: Retrieve all tracks in specific album** | |
| SELECT t.TrackId, t.Name AS Track\_Name, t.Composer, t.Milliseconds /(1000.0\*60.0) AS Duration\_in\_Minutes  FROM Track AS t  JOIN Album AS a ON t.AlbumId = a.AlbumId  WHERE a.Title = 'Restless and Wild'; | MATCH (a:Album {album\_title: 'Restless and Wild'})-[:CONTAINS]->(t:Track)  RETURN t.track\_id AS TrackId,         t.track\_name AS Track\_Name,         t.composer AS Composer,         toFloat(t.milli\_seconds) / (1000\*60) AS Duration\_in\_Minutes |
| **Query 3:** **Show Customers, their Invoices and purchased Tracks** | |
| SELECT  c.\*, i.\*, t.\*  FROM  Customer c  JOIN Invoice i ON c.CustomerId = i.CustomerId  JOIN InvoiceLine il ON i.InvoiceId = il.InvoiceId  JOIN Track t ON il.TrackId = t.TrackId | MATCH (customer:Customer)-[:PLACES]->(invoice:Invoice)-[:INCLUDES]->(track:Track)  RETURN customer, invoice, track; |
| **Query 4:** **Show Artist, their Albums and the Tracks they contain** | |
| SELECT Artist.Name, Album.Title, Track.Name  FROM Artist  JOIN Album ON Artist.ArtistId = Album.ArtistId  JOIN Track ON Album.AlbumId = Track.AlbumId; | MATCH (ar:Artist)-[c:CREATED]->(al:Album)-[d:CONTAINS]>- (t:Track)  RETURN ar, c, al, d, t; |
| **Query 5:** **Retrieve Top 5 tracks w.r.t to total count of Purchase**  SQL requires grouping with , while Cypher directly counts relationships. | |
| SELECT TOP 10 Track.Name, COUNT(InvoiceLine.TrackId) AS PurchaseCount  FROM Track  JOIN InvoiceLine ON Track.TrackId = InvoiceLine.TrackId  GROUP BY Track.Name  ORDER BY PurchaseCount DESC; | MATCH (i:Invoice)-[a:INCLUDES]->(t:Track)  RETURN t.track\_name, COUNT(t) AS PurchaseCount  ORDER BY PurchaseCount DESC  LIMIT 5; |
| **Query 6: Retrieve artists with their created albums and tracks with respect to albums** | |
| SELECT Artist.Name AS ArtistName, Album.Title AS AlbumTitle, Track.Name AS TrackName  FROM Artist  JOIN Album ON Artist.ArtistId = Album.ArtistId  JOIN Track ON Album.AlbumId = Track.AlbumId; | MATCH (ar:Artist)-[c:CREATED]->(al:Album)-[d:CONTAINS]->(t:Track)  RETURN ar, c, al, d, t; |
| **Query 7: Show all invoices with date placed by specific Customer** | |
| SELECT i.InvoiceId, i.InvoiceDate, i.Total as Total\_Bill  FROM Invoice AS i  JOIN Customer AS c ON i.CustomerId = c.CustomerId  WHERE c.FirstName = 'Astrid' AND c.LastName = 'Gruber'; | MATCH (c:Customer {first\_name: 'Astrid', last\_name: 'Gruber'})-[p:PLACES]->(i:Invoice)  RETURN i.invoice\_id, i.invoice\_date, i.total\_bill; |