

Alexandra Mejia Vielman

11 May 2025

Music Streaming Graph Database

-My graph database models a music streaming platform, showing how users interact with artists and songs. It includes data on users, the artists they follow, and the songs they listen to. By using a graph database, we can easily track connections and patterns between users, songs, and artists—giving us a clear picture of how music is experienced on the platform. We can answer questions like, for example: Which songs are most listened to? Which artists have the most followers? What genres are most popular among users from different countries?

Node Labels:

- User – listeners on the platform
- Artist – musicians who create songs
- Song – individual tracks

Relationship Types:

- LISTENED_TO (User -->Song)
- FOLLOWS (User -->Artist)
- PERFORMS (Artist -->Song)

Properties (attributes):

- User: username, age, country
- Artist: name, genre
- Song: title, release_year, duration

-- Cypher Code –

```
LOAD CSV WITH HEADERS FROM 'file:///users.csv' AS row  
RETURN row LIMIT 5;
```

```
LOAD CSV WITH HEADERS FROM 'file:///users.csv' AS row  
WITH row  
WHERE row.username IS NOT NULL AND row.username <> ""  
MERGE (u:User {username: row.username})  
SET u.name = row.name,  
    u.age = toInteger(row.age),  
    u.country = row.country;
```

```
LOAD CSV WITH HEADERS FROM 'file:///artists.csv' AS row  
WITH row  
WHERE row.name IS NOT NULL AND row.name <> ""  
MERGE (a:Artist {name: row.name})  
SET a.genre = row.genre;
```

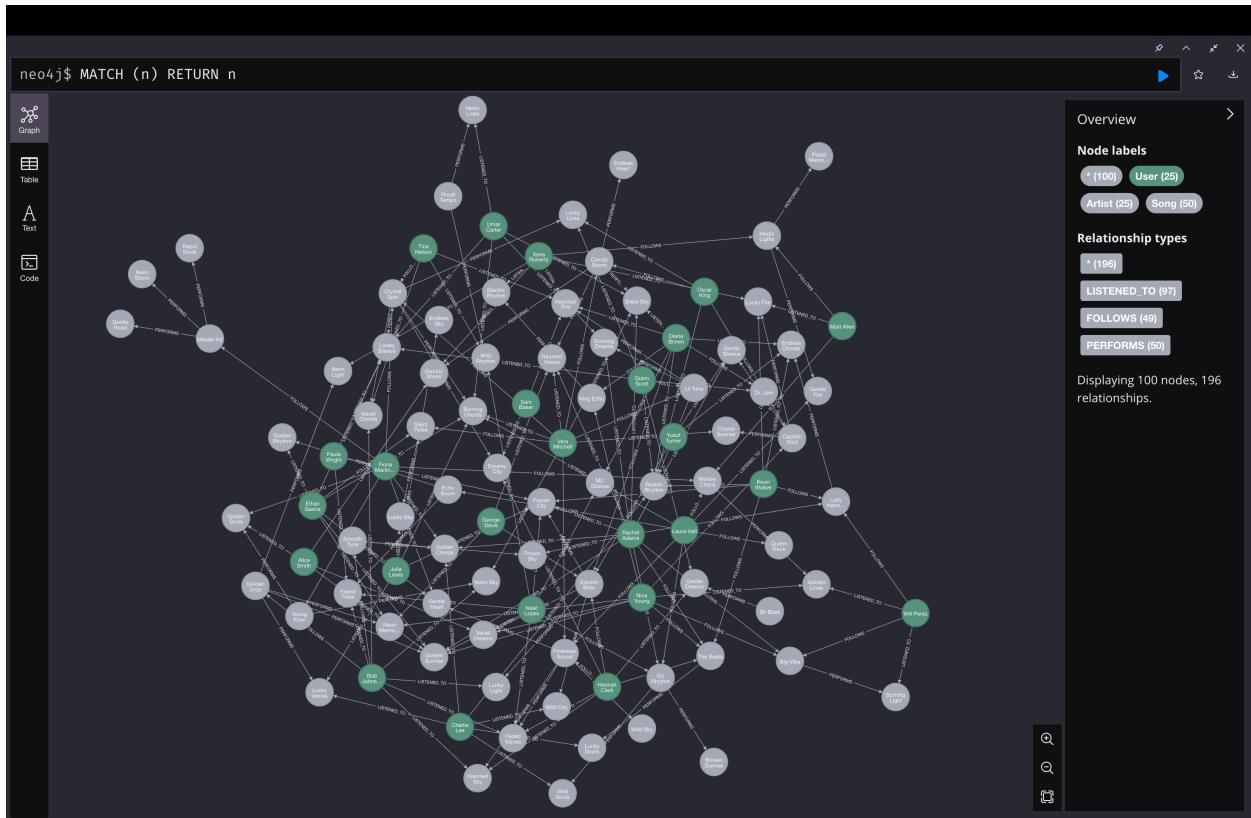
```
LOAD CSV WITH HEADERS FROM 'file:///songs.csv' AS row  
WITH row  
WHERE row.title IS NOT NULL AND row.title <> ""  
MERGE (s:Song {title: row.title})  
SET s.release_year = toInteger(row.release_year),  
    s.duration = toInteger(row.duration);
```

```
LOAD CSV WITH HEADERS FROM 'file:///performs.csv' AS row
```

```
WITH row  
WHERE row.artist IS NOT NULL AND row.song IS NOT NULL  
MATCH (a:Artist {name: row.artist})  
MATCH (s:Song {title: row.song})  
MERGE (a)-[:PERFORMS]->(s);
```

```
LOAD CSV WITH HEADERS FROM 'file:///listens.csv' AS row  
WITH row  
WHERE row.user IS NOT NULL AND row.song IS NOT NULL  
MATCH (u:User {username: row.user})  
MATCH (s:Song {title: row.song})  
MERGE (u)-[:LISTENED_TO]->(s);
```

```
LOAD CSV WITH HEADERS FROM 'file:///follows.csv' AS row  
WITH row  
WHERE row.user IS NOT NULL AND row.artist IS NOT NULL  
MATCH (u:User {username: row.user})  
MATCH (a:Artist {name: row.artist})  
MERGE (u)-[:FOLLOWS]->(a);  
  
MATCH (n) RETURN n
```



--20 queries—

1. List all users

```
MATCH (u:User) RETURN u.name, u.username;
```

	u.name	u.username
1	"Alice Smith"	"user1"
2	"Bob Johnson"	"user2"
3	"Charlie Lee"	"user3"
4	"Diana Brown"	"user4"
5	"Ethan Garcia"	"user5"
6	"Fiona Martinez"	"user6"
7		

2. List all artists by genre

```
MATCH (a:Artist) RETURN a.name, a.genre ORDER BY a.genre;
```

The screenshot shows the Neo4j browser interface with a query results table. The table has two columns: 'a.name' and 'a.genre'. The data is as follows:

a.name	a.genre
"Big Vibe"	"Electronic"
"Young Flow"	"Electronic"
"Professor Sound"	"Electronic"
"Golden Drop"	"Electronic"
"Grand Wave"	"Electronic"
"MC Groove"	"Hip-hop"

3. List all songs released after 2015

```
MATCH (s:Song) WHERE s.release_year > 2015 RETURN s.title, s.release_year;
```

The screenshot shows the Neo4j browser interface with a query results table. The table has two columns: 's.title' and 's.release_year'. The data is as follows:

s.title	s.release_year
"Frozen Sky"	2024
"Velvet Chords"	2022
"Gentle Silence"	2020
"Wild City"	2023
"Neon Memories"	2017
"Frozen City"	2023

4. Which songs did user1 listen to?

```
MATCH (:User {username: "user1"})-[:LISTENED_TO]->(s:Song) RETURN s.title;
```

The screenshot shows the Neo4j browser interface with a query results table. The table has one column labeled 's.title' and four rows of data:

s.title
1 "Electric Storm"
2 "Golden Souls"
3 "Velvet Dreams"
4 "Golden Sunrise"

5. Which artists does user1 follow?

```
MATCH (:User {username: "user1"})-[:FOLLOWS]->(a:Artist) RETURN a.name;
```

The screenshot shows the Neo4j browser interface with a query results table. The table has one column labeled 'a.name' and displays the message '(no changes, no records)'.

6. Which songs were performed by 'DJ Rhythm'?

```
MATCH (:Artist {name: "DJ Rhythm"})-[:PERFORMS]->(s:Song) RETURN s.title;
```

The screenshot shows the Neo4j browser interface with a query results table. The table has one column labeled 's.title' and two rows of data:

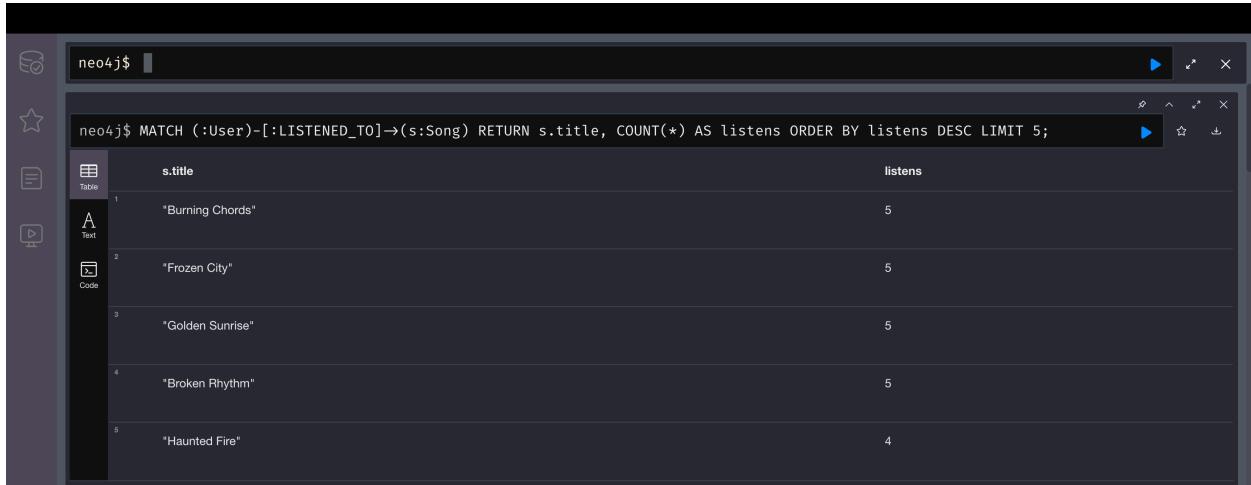
s.title
1 "Wild Souls"
2 "Broken Sunrise"

7. Top 5 most listened-to songs

```
MATCH (:User)-[:LISTENED_TO]->(s:Song)
```

```
RETURN s.title, COUNT(*) AS listens
```

```
ORDER BY listens DESC LIMIT 5;
```



The screenshot shows the Neo4j browser interface with a query results table. The table has two columns: 's.title' and 'listens'. The results are as follows:

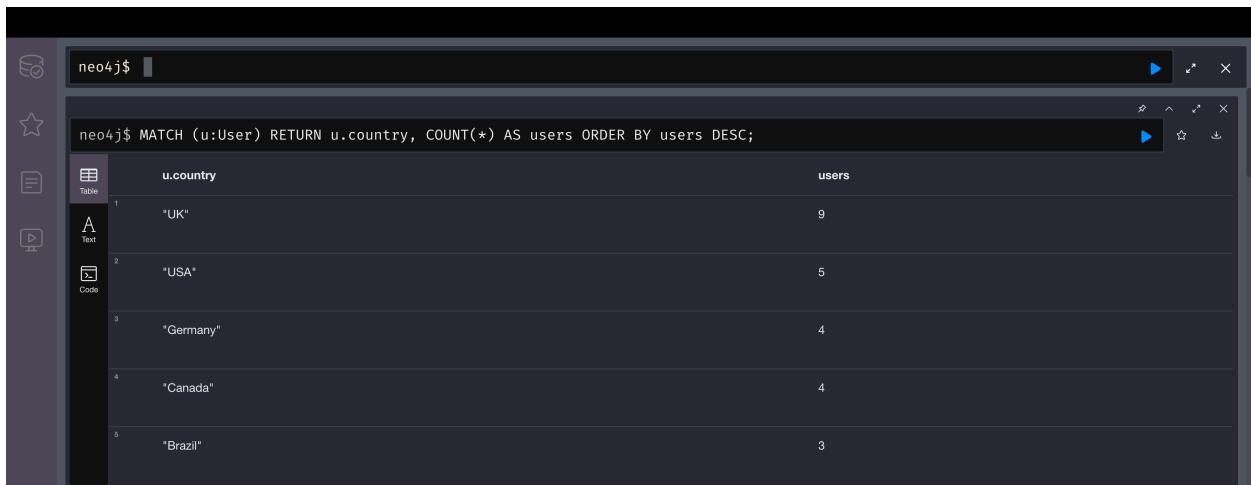
s.title	listens
"Burning Chords"	5
"Frozen City"	5
"Golden Sunrise"	5
"Broken Rhythm"	5
"Haunted Fire"	4

8. Number of users per country

```
MATCH (u:User)
```

```
RETURN u.country, COUNT(*) AS users
```

```
ORDER BY users DESC;
```



The screenshot shows the Neo4j browser interface with a query results table. The table has two columns: 'u.country' and 'users'. The results are as follows:

u.country	users
"UK"	9
"USA"	5
"Germany"	4
"Canada"	4
"Brazil"	3

9. How many users follow each artist?

```
MATCH (u:User)-[:FOLLOWS]->(a:Artist)
```

```
RETURN a.name, COUNT(u) AS followers
```

```
ORDER BY followers DESC;
```

The screenshot shows the Neo4j browser interface with a query results table. The query is:

```
neo4j$ MATCH (u:User)-[:FOLLOWERS]→(a:Artist) RETURN a.name, COUNT(u) AS followers ORDER BY followers DESC;
```

The results table has two columns: "a.name" and "followers". The data is as follows:

a.name	followers
"The Beats"	4
"Electric Bass"	4
"DJ Rhythm"	3
"Big Vibe"	3
"Professor Sound"	3
"Silent Pulse"	3

10. Average song duration per genre

```
MATCH (a:Artist)-[:PERFORMS]->(s:Song)  
RETURN a.genre, AVG(s.duration) AS avg_duration  
ORDER BY avg_duration DESC;
```

The screenshot shows the Neo4j browser interface with a query results table. The query is:

```
neo4j$ MATCH (a:Artist)-[:PERFORMS]→(s:Song) RETURN a.genre, AVG(s.duration) AS avg_duration ORDER BY avg_duration DESC;
```

The results table has two columns: "a.genre" and "avg_duration". The data is as follows:

a.genre	avg_duration
"Hip-hop"	207.3
"Jazz"	203.55555555555554
"Pop"	203.2857142857143
"Electronic"	200.18181818181816
"Rock"	191.53846153846155

11. Users who listened to songs longer than 4 minutes

```
MATCH (u:User)-[:LISTENED_TO]->(s:Song)  
WHERE s.duration > 240  
RETURN DISTINCT u.name, s.title;
```

The screenshot shows the Neo4j browser interface with a query results table. The table has two columns: 'u.name' and 's.title'. The data is as follows:

	u.name	s.title
1	"Diana Brown"	"Endless Chords"
2	"Kevin Walker"	"Endless Chords"
3	"Yusuf Turner"	"Endless Chords"
4	"Diana Brown"	"Silent Sky"
5	"Rachel Adams"	"Silent Sky"
6	"Umar Carter"	"Silent Sky"
7		

12. Songs that no one has listened to

MATCH (s:Song)

WHERE NOT (s)<-[LISTENED_TO]-(:User)

RETURN s.title;

The screenshot shows the Neo4j browser interface with a query results table. The table has one column: 's.title'. The data is as follows:

s.title
"Endless Heart"
"Rapid Memories"
"Neon Storm"
"Endless Sky"
"Gentle Road"
"Wild Sky"
7

13. Artists who performed more than 2 songs

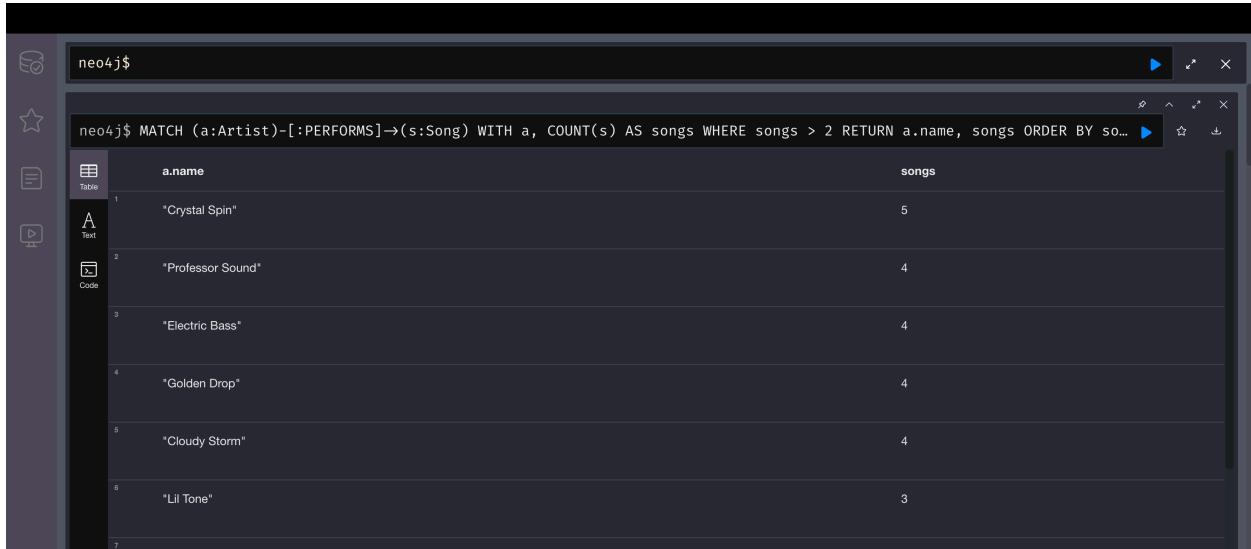
MATCH (a:Artist)-[:PERFORMS]->(s:Song)

WITH a, COUNT(s) AS songs

WHERE songs > 2

```
RETURN a.name, songs
```

```
ORDER BY songs DESC;
```



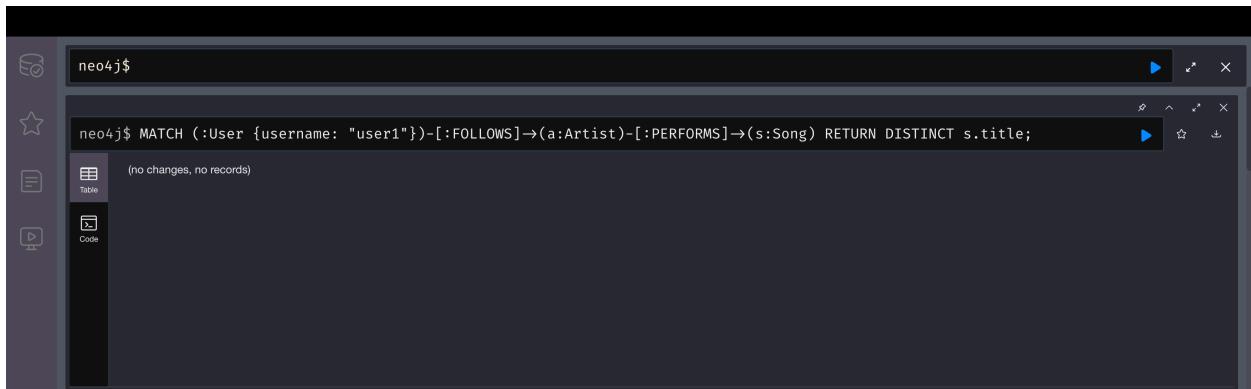
The screenshot shows the Neo4j browser interface with a query results table. The table has two columns: 'a.name' and 'songs'. The data rows are:

a.name	songs
"Crystal Spin"	5
"Professor Sound"	4
"Electric Bass"	4
"Golden Drop"	4
"Cloudy Storm"	4
"Lil Tone"	3

14. Songs by artists followed by user1

```
MATCH (:User {username: "user1"})-[:FOLLOWS]->(a:Artist)-[:PERFORMS]->(s:Song)
```

```
RETURN DISTINCT s.title;
```



The screenshot shows the Neo4j browser interface with a query results table. The table has one column: 's.title'. The message '(no changes, no records)' is displayed above the table.

s.title
(no changes, no records)

15. Users who listened to songs by 'DJ Rhythm'

```
MATCH (u:User)-[:LISTENED_TO]->(s:Song)<-[ :PERFORMS ]-(a:Artist {name: "DJ Rhythm"})
```

```
RETURN DISTINCT u.name;
```

The screenshot shows the Neo4j browser interface with a query results table. The table has one row with the following data:

u.name
"Charlie Lee"

16. Most active listener (by number of listens)

```
MATCH (u:User)-[:LISTENED_TO]->(s:Song)  
RETURN u.name, COUNT(s) AS total_listens  
ORDER BY total_listens DESC LIMIT 1;
```

The screenshot shows the Neo4j browser interface with a query results table. The table has one row with the following data:

u.name	total_listens
"Rachel Adams"	8

17. Users who follow the same artist

```
MATCH (u1:User)-[:FOLLOWS]->(a:Artist)<-[ :FOLLOWS ]-(u2:User)  
WHERE u1.username <> u2.username  
RETURN DISTINCT u1.username, u2.username, a.name;
```

The screenshot shows the Neo4j browser interface with a query results table. The table has three columns: u1.username, u2.username, and a.name. The data is as follows:

	u1.username	u2.username	a.name
1	"user9"	"user4"	"MC Groove"
2	"user7"	"user4"	"Electric Bass"
3	"user8"	"user4"	"Electric Bass"
4	"user25"	"user4"	"Electric Bass"
5	"user17"	"user4"	"Dr. Jam"
6	"user14"	"user4"	"King Echo"
7			

18. Number of songs each user listened to

```
MATCH (u:User)-[:LISTENED_TO]->(s:Song)  
RETURN u.username, COUNT(s) AS song_count  
ORDER BY song_count DESC;
```

The screenshot shows the Neo4j browser interface with a query results table. The table has two columns: u.username and song_count. The data is as follows:

	u.username	song_count
1	"user18"	8
2	"user2"	6
3	"user3"	6
4	"user5"	6
5	"user9"	6
6	"user21"	5
7		

19. Artists that perform songs longer than 5 minutes

```
MATCH (a:Artist)-[:PERFORMS]->(s:Song)  
WHERE s.duration > 300
```

```
RETURN a.name, s.title, s.duration;
```

The screenshot shows the Neo4j browser interface. The query entered is:

```
neo4j$ MATCH (a:Artist)-[:PERFORMS]-(s:Song) WHERE s.duration > 300 RETURN a.name, s.title,...
```

The results pane displays the message "(no changes, no records)" and "Completed after 13 ms.".

20.Genres of songs listened to by users in Canada

```
MATCH (u:User {country: "Canada"})-[:LISTENED_TO]->(s:Song)<-[ :PERFORMS ]-(a:Artist)
```

```
RETURN DISTINCT a.genre;
```

The screenshot shows the Neo4j browser interface. The query entered is:

```
neo4j$ MATCH (u:User {country: "Canada"})-[:LISTENED_TO]->(s:Song)<-[ :PERFORMS ]-(a:Artist) RETURN DISTINCT a.genre;
```

The results pane displays a table titled "a.genre" with the following data:

a.genre
1 "Rock"
2 "Electronic"
3 "Jazz"
4 "Hip-hop"
5 "Pop"