

001-rel-type-refactor-1

chapter01\$

```
//Step 1
MATCH (p:Playlist)-[hastrack:HAS_TRACK]->(t:Track)
//Step 2
MERGE (t)-[onplaylist:ON_PLAYLIST]->(p)
SET onplaylist.position=hastrack.position;
```

The image displays two screenshots of the Neo4j Cypher Shell interface, showing the results of a database refactor operation.

Top Screenshot:

- Database information:** Nodes (127,932), Relationships (306,339). Labels: Album, Artist, Playlist, Track, User. Relationships: ARTIST, HAS_TRACK, ON_PLAYLIST, OWNS, SIMILAR.
- Query:** chapter01\$
1 //Step 1
2 MATCH (p:Playlist)-[hastrack:HAS_TRACK]->(t:Track)
3 //Step 2
4 MERGE (t)-[onplaylist:ON_PLAYLIST]->(p)
5 SET onplaylist.position=hastrack.position;
Created 73,230 relationships, set 73,231 properties

Bottom Screenshot:

- Database information:** Nodes (127,932), Relationships (306,339). Labels: Album, Artist, Playlist, Track, User. Relationships: ARTIST, HAS_TRACK, ON_PLAYLIST, OWNS, SIMILAR. Property keys: id, name, notSamePosition, position, samePosition, uri.
- Query:** chapter01\$ MATCH p=()-[:ON_PLAYLIST]->() RETURN p
- Results overview:** Nodes (1,000), Relationships (1,145). Initial display limit hit at 1,000 nodes. Edit settings. (1000) Playlist (111) Track (889) (1145) ON_PLAYLIST (1145)
- Graph:** A visualization of the data, showing a large cluster of nodes and relationships.
- Query:** 1 //Step 1
2 MATCH (p:Playlist)-[hastrack:HAS_TRACK]->(t:Track)
3 //Step 2
4 MERGE (t)-[onplaylist:ON_PLAYLIST]->(p)
5 SET onplaylist.position=hastrack.position;
Created 73,230 relationships, set 73,231 properties

There are important case sensitivity rules in Cypher:

1. **Keywords and functions:** Case-insensitive (MATCH, RETURN, collect, COLLECT)
2. **Variables, labels, and relationship types:** Case-sensitive

002-recommendation-1

chapter01\$

```
MATCH (popularTrack:Track)-[:ON_PLAYLIST]-(:Playlist)
WITH popularTrack, count(*) as playlistCount
WHERE playlistCount > 5
RETURN popularTrack.name, playlistCount
```

chapter01\$

```
MATCH (p)-[:ON_PLAYLIST]-(t)
WHERE r.position = COUNT { (p)-[:ON_PLAYLIST]-(t)}
RETURN p.name, t.name, r.position
limit 10
```

Find tracks (t) on playlists (p) where the track's position in the playlist equals the total number of tracks in that playlist. In other words, find tracks that are in the **last position** of their playlists

chapter01\$

```
MATCH (p)-[:ON_PLAYLIST]-(t)
WHERE r.position = COUNT { (p)-[:ON_PLAYLIST]-() }
RETURN p.name, t.name, r.position
```

This is a **count subquery** that counts all :ON_PLAYLIST relationships to node p. However, there are two problems:

1. **Subqueries in WHERE clauses** need to be boolean expressions, not numeric comparisons
2. **Aggregation context** - you're mixing per-row matching with aggregated counts

chapter01\$

```
MATCH (p)-[:ON_PLAYLIST]-(t)
WHERE r.position = COUNT {
  MATCH (p)-[:ON_PLAYLIST]-(t2)
  RETURN r2
}
RETURN p.name, t.name, r.position
```

```

1 MATCH (p)-[:ON_PLAYLIST]-(t)
2 WHERE r.position = 9468 and t.name="Me"
3 RETURN p, t, r.position

```

Graph Table RAW

Relationship details

ON_PLAYLIST

| Key | Value |
|----------|--|
| <id> | 5:6a285308-c838-4bc7-bbee-44ffb8169012:1152925902653440025 |
| position | 9468 |

chapter01\$

```

MATCH (p)-[:ON_PLAYLIST]-(t)
WITH p, COUNT(t) AS total_tracks
MATCH (p)-[:ON_PLAYLIST]-(t)
WHERE r.position = total_tracks
RETURN p.name, t.name, r.position

```

chapter01\$

```

MATCH (p)-[:ON_PLAYLIST]-(t)
WHERE r.position = size([(p)-[:ON_PLAYLIST]-(other_t) | other_t])
RETURN p.name, t.name, r.position

```

Query Overview

The goal is to recommend tracks that:

1. Appear in playlists similar to "all that jazz"
2. Aren't already in "all that jazz"
3. Aren't overly popular tracks
4. Are weighted by how frequently they appear in similar contexts

chapter01\$

```

//Find popular tracks
MATCH (popularTrack:Track)-[:ON_PLAYLIST]-(Playlist)
WITH popularTrack, count(*) as playlistCount
ORDER BY playlistCount DESC
LIMIT 10
WITH collect(elementId(popularTrack)) as popularTracks
//Finds the 10 tracks that appear in the most playlists
//Stores their IDs to exclude them from recommendations (avoid recommending overplayed tracks)
// For a given Playlist

```

```

MATCH (p:Playlist) WHERE p.name = "all that jazz"
// Find the last track
MATCH (p)-[:ON_PLAYLIST]-(t)
WHERE r.position = COUNT { (p)-[:ON_PLAYLIST]-() }
//Finds the playlist named "all that jazz"
//Finds the last track in that playlist (where position = total track count)
//Note: This has the same syntax issue mentioned earlier - should use a subquery

WITH p AS playlist, t AS lastTrack, popularTracks
// Get the previous tracks
WITH playlist, lastTrack, popularTracks, COLLECT {
MATCH (playlist)-[:ON_PLAYLIST]-(previous)
  WHERE previous <> lastTrack
RETURN elementId(previous)
} AS previousTracks
//Collects IDs of all tracks in "all that jazz" EXCEPT the last track
//This creates an exclusion list of tracks already in the playlist

// Find other playlists that have the same the last track
MATCH (lastTrack)-[:ON_PLAYLIST]->(otherPlaylist)-[:SIMILAR]-(playlist)
  //Finds other playlists that:
    ○ Contain the same last track as "all that jazz"
    ○ Are marked as SIMILAR to "all that jazz"
  //This finds contextually relevant playlists

// Find other tracks which are not in the given playlist
MATCH (otherPlaylist)-[:ON_PLAYLIST]-(recommendation)
WHERE NOT elementId(recommendation) IN previousTracks
AND NOT elementId(recommendation) IN popularTracks
//Gets tracks from these similar playlists
//Filters out tracks that are:
  ○ Already in "all that jazz" (using previousTracks)
  ○ Overly popular (using popularTracks)

// Score them by how frequently they appear
RETURN recommendation.id as recommendedTrackId, recommendation.name AS
recommendedTrack, otherPlaylist.name AS fromPlaylist, count(*) AS score
ORDER BY score DESC
LIMIT 5
//Scores recommendations by frequency - how many similar playlists contain each track
//Returns top 5 recommendations with highest scores

```

Instance: neo4j://localhost:7687 Database: chapter01 cypher User: neo4j

Database information

Nodes (127,932)

Album Artist Playlist Track User

Relationships (306,339)

ARTIST HAS_TRACK ON_PLAYLIST

OWNS SIMILAR

Property keys

id name notSamePosition position

samePosition uri

chapter01\$

```

9 // Find the last track
10 MATCH (p)-[:ON_PLAYLIST]-(t)
11 WHERE r.position = COUNT { (p)-[:ON_PLAYLIST]-(t) }
12 WITH p AS playlist, t AS lastTrack, popularTracks
13 // Get the previous tracks
14 WITH playlist, lastTrack, popularTracks, COLLECT {
15 MATCH (playlist)-[:ON_PLAYLIST]-(previous)
16 WHERE previous <> lastTrack
17 RETURN elementId(previous)
18 } AS previousTracks
19 // Find other playlists that have the same the last track
20 MATCH (lastTrack)-[:ON_PLAYLIST]-(otherPlaylist)-[:SIMILAR]-(playlist)
21 // Find other tracks which are not in the given playlist

```

Table RAW

| recommendedTrack | recommendedTrack | fromPlaylist | score |
|------------------------------|--|---------------|-------|
| "3Jv6RMN6A7Fvdb d5Fq5dq6" | "Softly As In A Morning Sunris e" | "smooth jazz" | 1 |
| "4zRweMHazlqCTU e6GvviYf" | "Can't Help Lov ing" | "smooth jazz" | 1 |
| "59bCluxmIw22x8 YcRji4HY" | "Moonlight In V ermont / Stormy Wheather / I Kn ow Why - Slow f ox" | "smooth jazz" | 1 |
| "7ysmJhXFQtiBQL k6EZ6sks" | "Journey Into M elody - 2007 Di gital Remaster/ Rudy Van Gelder Edition" | "smooth jazz" | 1 |

The Recommendation Logic

This query uses **collaborative filtering**:

- "Playlists that are similar to yours and share your last track probably have other tracks you'd like"
- It avoids recommending tracks you already have or that are too mainstream
- It prioritizes tracks that appear frequently across multiple similar playlists

Fixed Query:

chapter01\$

```

// Find popular tracks to exclude
MATCH (popularTrack:Track)-[:ON_PLAYLIST]-(Playlist)
WITH popularTrack, count(*) as playlistCount
ORDER BY playlistCount DESC
LIMIT 10
WITH collect(elementId(popularTrack)) as popularTracks

```

```

// For the target playlist
MATCH (p:Playlist)
WHERE p.name = "all that jazz"

```

```

// Find the last track efficiently
MATCH (p)-[:ON_PLAYLIST]-(t)

```

```

WITH p, t, r, popularTracks, count { (p)<-[ON_PLAYLIST]-() } as total_tracks
WHERE r.position = total_tracks
WITH p AS playlist, t AS lastTrack, popularTracks

// Get all tracks from the playlist (for exclusion)
MATCH (playlist)<-[ON_PLAYLIST]-(previous)
WITH playlist, lastTrack, popularTracks,
     collect(elementId(previous)) as allTrackIds

// Find similar playlists that share the last track
MATCH (lastTrack)-[:ON_PLAYLIST]->(otherPlaylist)-[:SIMILAR]-(playlist)
WHERE otherPlaylist <> playlist

// Find recommendation candidates
MATCH (otherPlaylist)<-[ON_PLAYLIST]-(recommendation)
WHERE NOT elementId(recommendation) IN allTrackIds
      AND NOT elementId(recommendation) IN popularTracks

// Score by frequency across similar playlists
RETURN recommendation.id as recommendedTrackId,
       recommendation.name AS recommendedTrack,
       otherPlaylist.name AS fromPlaylist,
       count(*) AS score
ORDER BY score DESC
LIMIT 5

```

Instance: neo4j://localhost:7687 Database: chapter01 cypher User: neo4j

Database information

Nodes (127,932)

Album Artist Playlist Track User

Relationships (306,339)

ARTIST HAS_TRACK ON_PLAYLIST

OWNS SIMILAR

Property keys

id name notSamePosition position

samePosition uri

```

chapter01$
10 WHERE p.name = "all that jazz"
11
12 // Find the last track efficiently
13 MATCH (p)-[:ON_PLAYLIST]-(t)
14 WITH p, t, r, popularTracks, count { (p)-[:ON_PLAYLIST]-(t) } as total_tracks
15 WHERE r.position = total_tracks
16 WITH p AS playlist, t AS lastTrack, popularTracks
17
18 // Get all tracks from the playlist (for exclusion)
19 MATCH (playlist)-[:ON_PLAYLIST]-(previous)
20 WITH playlist, lastTrack, popularTracks,
21     collect(elementId(previous)) as allTrackIds

```

Table RAW

| | recommendedTrac | recommendedTrac | fromPlaylist | score |
|---|------------------------------|---|---------------|-------|
| 1 | "3Jv6RMN6A7Fvdb d5Fq5dq6" | "Softly As In A Morning Sunris e" | "smooth jazz" | 1 |
| 2 | "4zRweMHazlqCTU e6GvviYf" | "Can't Help Lov ing" | "smooth jazz" | 1 |
| 3 | "59bC1uxmIw22x8 YcRji4HY" | "Moonlight In V ermont / Stormy Wheather / I Kn ow Why - Slow f ox" | "smooth jazz" | 1 |
| 4 | "6zofY2yVlGkQf BHMWGx8q" | "All the Way (f eat. Denis Sole e)" | "smooth jazz" | 1 |

Optimized Query:

chapter01\$

```

// Find popular tracks and target playlist in parallel
MATCH (popularTrack:Track)-[:ON_PLAYLIST]-(Playlist)
WITH popularTrack, count(*) as playlistCount
ORDER BY playlistCount DESC
LIMIT 10
WITH collect(elementId(popularTrack)) as popularTracks
MATCH (playlist:Playlist {name: "all that jazz"})

```

```

// Get all tracks from target playlist and identify last track
MATCH (playlist)-[:ON_PLAYLIST]-(track)
WITH playlist, popularTracks,
    collect(elementId(track)) as existingTrackIds,
    collect({track: track, position: r.position}) as tracks
WITH playlist, popularTracks, existingTrackIds,
    [t in tracks WHERE t.position = size(tracks) | t.track][0] as lastTrack

```

```

// Find recommendations from similar playlists
MATCH (lastTrack)-[:ON_PLAYLIST]->(otherPlaylist)-[:SIMILAR]-(playlist)
WHERE otherPlaylist <> playlist

```

```

MATCH (otherPlaylist)-[:ON_PLAYLIST]-(recommendation)
WHERE NOT elementId(recommendation) IN existingTrackIds
AND NOT elementId(recommendation) IN popularTracks

```

```

RETURN recommendation.id as recommendedTrackId,
       recommendation.name AS recommendedTrack,
       otherPlaylist.name AS fromPlaylist,
       count(*) AS score
ORDER BY score DESC
LIMIT 5

```

Instance: neo4j://localhost:7687 Database: chapter01 CYPHER 5 User: neo4j

Database information

Nodes (127,932)

Album Artist Playlist Track User

Relationships (306,339)

ARTIST HAS_TRACK ON_PLAYLIST

OWNS SIMILAR

Property keys

id name notSamePosition position samePosition uri

chapter01\$

```

13 collect({track: track, position: r.position}) as tracks
14 WITH playlist, popularTracks, existingTrackIds,
15 [t in tracks WHERE t.position = size(tracks) | t.track][0] as lastTrack
16
17 // Find recommendations from similar playlists
18 MATCH (lastTrack)-[:ON_PLAYLIST]->(otherPlaylist)-[:SIMILAR]-(playlist)
19 WHERE otherPlaylist <> playlist
20 MATCH (otherPlaylist)-[:ON_PLAYLIST]-(recommendation)
21 WHERE NOT elementId(recommendation) IN existingTrackIds
22 AND NOT elementId(recommendation) IN popularTracks
23
24 RETURN recommendation.id as recommendedTrackId,
25       recommendation.name AS recommendedTrack,
26       otherPlaylist.name AS fromPlaylist,
27       count(*) AS score
28 ORDER BY score DESC
29 LIMIT 5

```

Table RAW

| | recommendedTrac | recommendedTrac | fromPlaylist | score |
|---|------------------------------|---|---------------|-------|
| 1 | "3Jv6RMN6A7Fvdb d5Fq5dq6" | "Softly As In A Morning Sunris e" | "smooth jazz" | 1 |
| 2 | "4zRweMHazlqCTU e6GvviYf" | "Can't Help Lov ing" | "smooth jazz" | 1 |
| 3 | "59bC1uxmIw22x8 YcRji4HY" | "Moonlight In V ermont / Stormy Wheather / I Kn ow Why - Slow f ox" | "smooth jazz" | 1 |
| 4 | "6zofY2yVELGkQf" | "All the Way (f | "smooth jazz" | 1 |

003-rel-type-refactor-2

chapter01\$

```

MATCH (p:Playlist)-[hastrack:HAS_TRACK]->(t:Track)
DELETE hastrack

```


Instance: neo4j://localhost:7687 Database: chapter01 cypher 5 User: neo4j

Database information

Nodes (127,932)

Album Artist Playlist Track User

Relationships (233,108)

ARTIST HAS_TRACK ON_PLAYLIST OWNS SIMILAR

chapter01\$

chapter01\$ //Step 3 MATCH (p:Playlist)-[hastrack:HAS_TRACK]->(t:Track) DELETE hastrack

Deleted 73,231 relationships

```

9 // Get all tracks from target playlist and identify last track
10 MATCH (playlist)-[r:ON_PLAYLIST]->(track)
11 WITH playlist, popularTracks,
12    collect(elementId(track)) as existingTrackIds

```

004-artistNameIndex

chapter01\$

```
CREATE INDEX artist_name FOR (n:Artist) ON n.name;
```

Instance: neo4j://localhost:7687 Database: chapter01 cypher 5 User: neo4j

Database information

Nodes (127,932)

Album Artist Playlist Track User

Relationships (233,108)

ARTIST HAS_TRACK ON_PLAYLIST OWNS SIMILAR

chapter01\$

chapter01\$ CREATE INDEX artist_name FOR (n:Artist) ON n.name;

Added 1 index

005-loadGenres

chapter01\$

```

LOAD CSV WITH HEADERS FROM "file:///genres.csv" AS row
WITH row
WITH row.Artist as artist, split(row.Genre,"/") AS genreList
UNWIND genreList AS genre
WITH artist, collect(trim(toLower(genre))) AS genres
MATCH (a:Artist {name:artist})
SET a.genres=genres

```

Instance: neo4j://localhost:7687 Database: chapter01 cypher 5 User: neo4j [Go back to old Br](#)

Database information

Nodes (127,932)

Album Artist Playlist Track User

Relationships (233,108)

ARTIST HAS_TRACK ON_PLAYLIST OWNS SIMILAR

Property keys

genres id name notSamePosition position samePosition uri

chapter01\$

```

1 LOAD CSV WITH HEADERS FROM "file:///genres.csv" AS row
2 WITH row
3 WITH row.Artist as artist, split(row.Genre,"/") AS genreList
4 UNWIND genreList AS genre
5 WITH artist, collect(trim(toLower(genre))) AS genres
6 MATCH (a:Artist {name:artist})
7 SET a.genres=genres

```

Set 113 properties Completed after 7

chapter01\$ CREATE INDEX artist_name FOR (n:Artist) ON n.name;

Added 1 index Completed after 1

chapter01\$

```
MATCH (a)
where a.genres is NOT NULL
RETURN a.name , a.genres
```

chapter01\$

```
MATCH (a)-[:ARTIST]-(t)
WHERE a.genres is NOT NULL
RETURN a.name , a.genres , t.name
LIMIT 100
```

chapter01\$

```
MATCH (n:Artist {name:"David Bowie"
}) RETURN n;
```

006-genreConstraint

chapter01\$

```
CREATE CONSTRAINT genre_name
FOR (genre:Genre) REQUIRE genre.name IS UNIQUE
```

The screenshot shows the Neo4j Cypher Shell interface. At the top, it displays the instance 'neo4j://localhost:7687', the database 'chapter01 CYPHER 5', and the user 'neo4j'. On the left, there is a sidebar with 'Database information' and a list of nodes (127,932) including Album, Artist, Genre, Playlist, Track, and User. The main area shows the Cypher query being executed: 'chapter01\$' followed by '1 CREATE CONSTRAINT genre_name' and '2 FOR (genre:Genre) REQUIRE genre.name IS UNIQUE'. A status message at the bottom indicates 'Added 1 constraint'.

007-refactor-genre

chapter01\$

```
//Step 1
MATCH (a:Artist)
WHERE a.genres IS NOT NULL
//Step 2
WITH a
UNWIND a.genres as genreName
//Step 3
MERGE (g:Genre {name:genreName})
//Step 4
```

REMOVE a.genres

Instance: neo4j://localhost:7687 Database: chapter01 cypher v3 User: neo4j

[Go back to old version](#)

Database information

Nodes (127,986)

* Album Artist Genre Playlist Track
User

Relationships (233,362)

* ARTIST GENRE HAS_TRACK
ON_PLAYLIST OWNS SIMILAR

Property keys

genres id name notSamePosition
position samePosition uri

chapter01\$

```
chapter01$ MATCH (n:Genre) RETURN n LIMIT 25;
```

Graph Table RAW

Results overview

Nodes (57)

* (57) Artist (1) Genre (25)
Track (31)

Relationships (33)

* (33) ARTIST (31) GENRE (2)

Started streaming 25 records after 41 ms and completed a...

MATCH (p:Playlist)

```
//The relationship r contains the position property
```

```

CALL {
  WITH p
  MATCH (t:Track)-[r:ON_PLAYLIST]->(p:Playlist)

  //Creates a new PlaylistTrack node for each track-playlist combination
  //Establishes a relationship from the track to this junction node
  //Step 2
  CREATE (pt:PlaylistTrack)
  CREATE (t)-[:PLAYLIST_ITEM]->(pt)

  //Creates a relationship from the playlist to the junction node
  //Preserves the position data by copying it from the original ON_PLAYLIST relationship
  //Step 3
  CREATE (p)-[:TRACK_ITEM_TEMP {position:r.position}]->(pt)

  //Executes the operations in batched transactions
  //Prevents memory issues when processing large datasets
  //Neo4j automatically manages transaction boundaries
} IN TRANSACTIONS

```

Instance: neo4j://localhost:7687 Database: chapter01 CYPHER 5 User: neo4j

Database information

Nodes (201,216)

- Album
- Artist
- Genre
- Playlist
- PlaylistTrack
- Track
- User

Relationships (379,822)

- ARTIST
- GENRE
- HAS_TRACK
- ON_PLAYLIST
- OWNS
- PLAYLIST_ITEM
- SIMILAR
- TRACK_ITEM_TEMP

Property keys

- genres
- id
- name
- notSamePosition
- position
- samePosition
- uri

chapter01\$

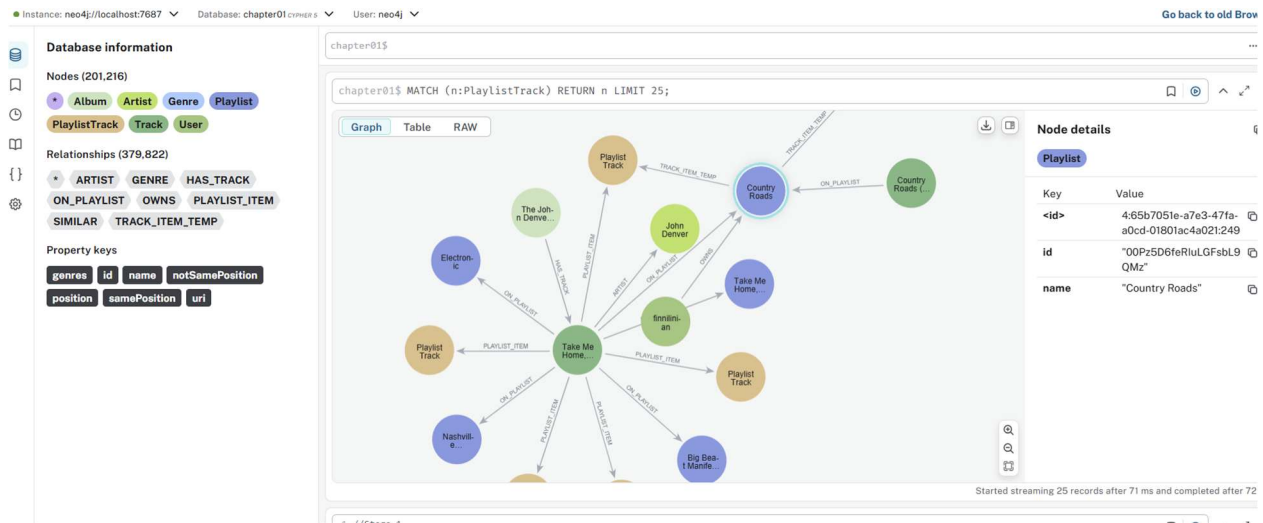
```

1 //Stage 1
2 //Step 1
3 MATCH (p:Playlist)
4 CALL {
5   WITH p
6   MATCH (t:Track)-[r:ON_PLAYLIST]->(p:Playlist)
7   //Step 2
8   CREATE (pt:PlaylistTrack)
9   CREATE (t)-[:PLAYLIST_ITEM]->(pt)
10  //Step 3
11  CREATE (p)-[:TRACK_ITEM_TEMP {position:r.position}]->(pt)
12 } IN TRANSACTIONS

```

✓ Created 73,230 nodes, created 146,460 relationships, set 73,230 properties, added 73,230 labels

> ⚠ This feature is deprecated and will be removed in future versions.



Query Overview

The query is creating a **junction node** pattern to replace a direct relationship between Playlists and Tracks.

Original Structure (before):

(Track)-[:ON_PLAYLIST {position: X}]->(Playlist)

New Structure (after):

(Track)-[:PLAYLIST_ITEM]->(PlaylistTrack)<-[:TRACK_ITEM_TEMP {position: X}]->(Playlist)

Why This Pattern is Used

This transformation is common when you need to:

1. **Add metadata** to playlist-track relationships
2. **Normalize the data model** for many-to-many relationships
3. **Prepare for additional relationships** to the junction node
4. **Migrate to a more flexible schema**

009-refactor-playlist-linked-2

Don't execute original:

```
//Stage 2
MATCH (p:Playlist)
CALL {
  WITH p
```

```

MATCH (p)-[:TRACK_ITEM_TEMP {position:1}]->(firstTrack:PlaylistTrack)
CREATE (p)-[:PLAYLIST_TRACK]->(firstTrack)
WITH p
MATCH (p)-[r:TRACK_ITEM_TEMP]->(lastTrack:PlaylistTrack)
  WHERE r.position = COUNT {()-[:ON_PLAYLIST]->(p)}
CREATE (p)-[:LAST_PLAYLIST_TRACK]->(lastTrack)
} IN TRANSACTION

```

chapter01\$

// Stage 2

```
MATCH (p:Playlist)
```

```
CALL {
```

```
  WITH p
```

```
  // Find first track (position 1)
```

```
  MATCH (p)-[:TRACK_ITEM_TEMP {position: 1}]->(firstTrack:PlaylistTrack)
```

```
  CREATE (p)-[:PLAYLIST_TRACK]->(firstTrack)
```

```
WITH p
```

```
// Find last track using original ON_PLAYLIST count
```

```
MATCH (p)-[r:TRACK_ITEM_TEMP]->(lastTrack:PlaylistTrack)
```

```
WITH p, r, lastTrack, count { (:Track)-[:ON_PLAYLIST]->(p) } as totalTracks
```

```
WHERE r.position = totalTracks
```

```
CREATE (p)-[:LAST_PLAYLIST_TRACK]->(lastTrack)
```

```
} IN TRANSACTIONS
```

Instance: neo4j://localhost:7687 Database: chapter01 CYPHER 5 User: neo4j

Database information

Nodes (201,216)

- Album
- Artist
- Genre
- Playlist
- PlaylistTrack
- Track
- User

Relationships (380,302)

- ARTIST
- GENRE
- HAS_TRACK
- LAST_PLAYLIST_TRACK
- ON_PLAYLIST
- OWNS
- PLAYLIST_ITEM
- PLAYLIST_TRACK
- SIMILAR
- TRACK_ITEM_TEMP

Property keys

- genres
- id
- name
- notSamePosition
- position
- samePosition
- uri

chapter01\$

```

1 // Stage 2
2 MATCH (p:Playlist)
3 CALL {
4   WITH p
5   // Find first track (position 1)
6   MATCH (p)-[:TRACK_ITEM_TEMP {position: 1}]->(firstTrack:PlaylistTrack)
7   CREATE (p)-[:PLAYLIST_TRACK]->(firstTrack)
8
9   WITH p
10  // Find last track using original ON_PLAYLIST count
11  MATCH (p)-[r:TRACK_ITEM_TEMP]->(lastTrack:PlaylistTrack)
12  WITH p, r, lastTrack, count { (:Track)-[:ON_PLAYLIST]->(p) } as totalTracks
13  WHERE r.position = totalTracks
14  CREATE (p)-[:LAST_PLAYLIST_TRACK]->(lastTrack)
15 } IN TRANSACTIONS

```

Created 480 relationships

This feature is deprecated and will be removed in future versions.

If you are not sure that you ran the query twice, you can use below queries to verify that:

```
MATCH (p:Playlist)
WHERE size([(p)-[:PLAYLIST_TRACK]->>() | 1]) > 1
  OR size([(p)-[:LAST_PLAYLIST_TRACK]->>() | 1]) > 1
RETURN count(p) as playlistsWithDuplicates;
```

Interpret Results:

- If `playlistsWithDuplicates = 0` → Query ran once (or relationships don't exist)
- If `playlistsWithDuplicates > 0` → Query ran multiple times

If duplicates exist, see details:

```
MATCH (p:Playlist)
WITH p,
  size([(p)-[:PLAYLIST_TRACK]->>() | 1]) as ptCount,
  size([(p)-[:LAST_PLAYLIST_TRACK]->>() | 1]) as lptCount
WHERE ptCount > 1 OR lptCount > 1
RETURN p.name as playlistName, ptCount, lptCount
ORDER BY ptCount DESC, lptCount DESC;
```

Visual Graph Inspection (Optional):

You can also visually check a specific playlist:

// Pick a playlist and view its relationships

```
MATCH (p:Playlist)-[r:PLAYLIST_TRACK|LAST_PLAYLIST_TRACK]->(pt)
WHERE p.name = "all that jazz" // Replace with actual playlist name
RETURN p, r, pt;
```

Look for:

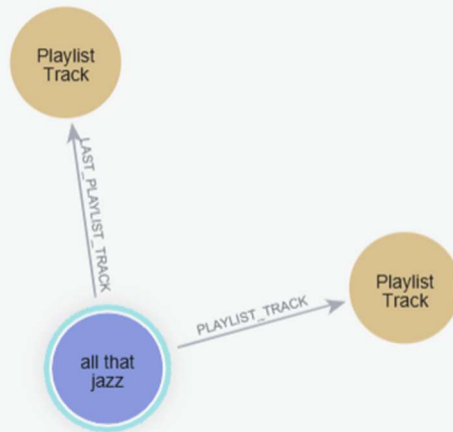
- Multiple `PLAYLIST_TRACK` arrows from the same playlist
- Multiple `LAST_PLAYLIST_TRACK` arrows from the same playlist

```

1 MATCH (p:Playlist)-[r:PLAYLIST_TRACK|LAST_PLAYLIST_TRACK]->(pt)
2 WHERE p.name = "all that jazz" // Replace with actual playlist name
3 RETURN p, r, pt;

```

Graph Table RAW



If Duplicates Found

// Remove duplicates in one go

```
MATCH (p:Playlist)-[r:PLAYLIST_TRACK]->(pt)
```

```
WITH p, pt, collect(r) as rels
```

```
WHERE size(rels) > 1
```

```
UNWIND tail(rels) as extraRel
```

```
DELETE extraRel;
```

```
MATCH (p:Playlist)-[r:LAST_PLAYLIST_TRACK]->(pt)
```

```
WITH p, pt, collect(r) as rels
```

```
WHERE size(rels) > 1
```

```
UNWIND tail(rels) as extraRel
```

```
DELETE extraRel;
```


010-refactor-playlist-linked-3

chapter01\$

```
//Stage 3
//Step 1
MATCH (p:Playlist)
CALL {
  WITH p
  MATCH (p)-[r:TRACK_ITEM_TEMP]->(t:PlaylistTrack)
  WITH r,t
  ORDER BY r.position
  WITH COLLECT(t) AS playlistTracks
  UNWIND RANGE(0,SIZE(playlistTracks) - 2) as idx
  WITH playlistTracks[idx] AS t1, playlistTracks[idx+1] AS t2
  MERGE (t1)-[:PLAYLIST_TRACK]->(t2)
} IN TRANSACTIONS
```

The screenshot shows the Neo4j Browser interface. On the left, the 'Database information' panel displays nodes (201,216) and relationships (453,292). The nodes include Album, Artist, Genre, Playlist, PlaylistTrack, Track, and User. The relationships include ARTIST, GENRE, HAS_TRACK, LAST_PLAYLIST_TRACK, ON_PLAYLIST, OWNS, PLAYLIST_ITEM, PLAYLIST_TRACK, SIMILAR, and TRACK_ITEM_TEMP. The 'Property keys' section lists genres, id, name, notSamePosition, position, samePosition, and uri. The main query editor shows the Cypher query from the previous block. Below the query, a message states 'Created 72,990 relationships' and 'This feature is deprecated and will be removed in future versions.' The bottom of the interface features a 'GUIDE' section with a puzzle piece icon and a 'DATASET' section with a film camera icon.

Issues with above query:

1. **Missing OF 10000 in IN TRANSACTIONS** - This specifies the batch size
2. **No cleanup of temporary relationships** - You're creating new relationships but not removing the old ones
3. **Potential performance issues** with large playlists due to the UNWIND approach

chapter01\$

// Count how many TRACK_ITEM_TEMP relationships exist

MATCH ()-[r:TRACK_ITEM_TEMP]->()

RETURN COUNT(r) AS tempRelationshipsToDelete;

```
1 // Count how many TRACK_ITEM_TEMP relationships exist
2 MATCH ()-[r:TRACK_ITEM_TEMP]->()
3 RETURN COUNT(r) AS tempRelationshipsToDelete;
```

Table RAW

tempRelationships

1 73230

chapter01\$

// View sample TRACK_ITEM_TEMP relationships with node details

MATCH (p)-[r:TRACK_ITEM_TEMP]->(t)

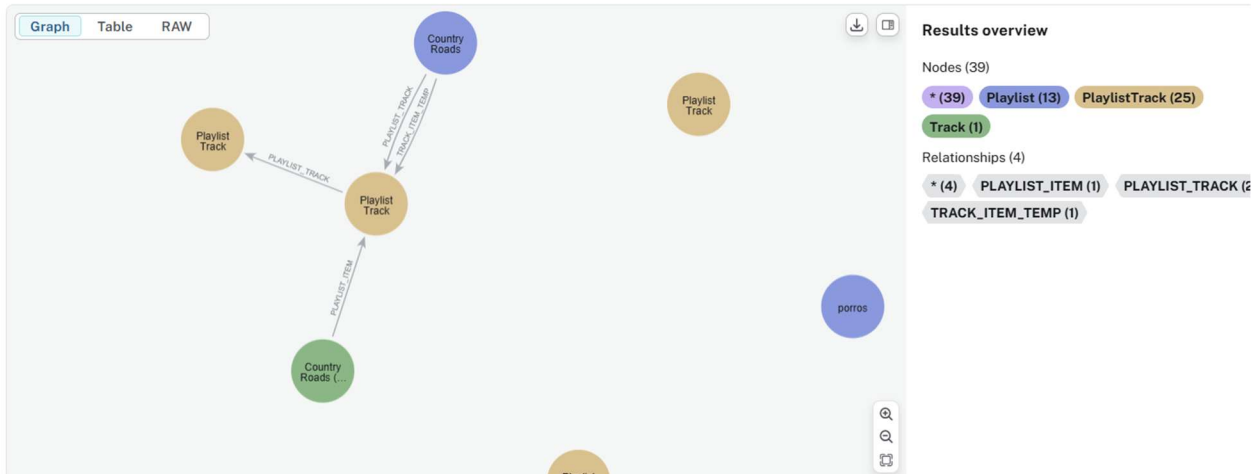
//RETURN p.name AS playlist, t.name AS track, r.position AS position

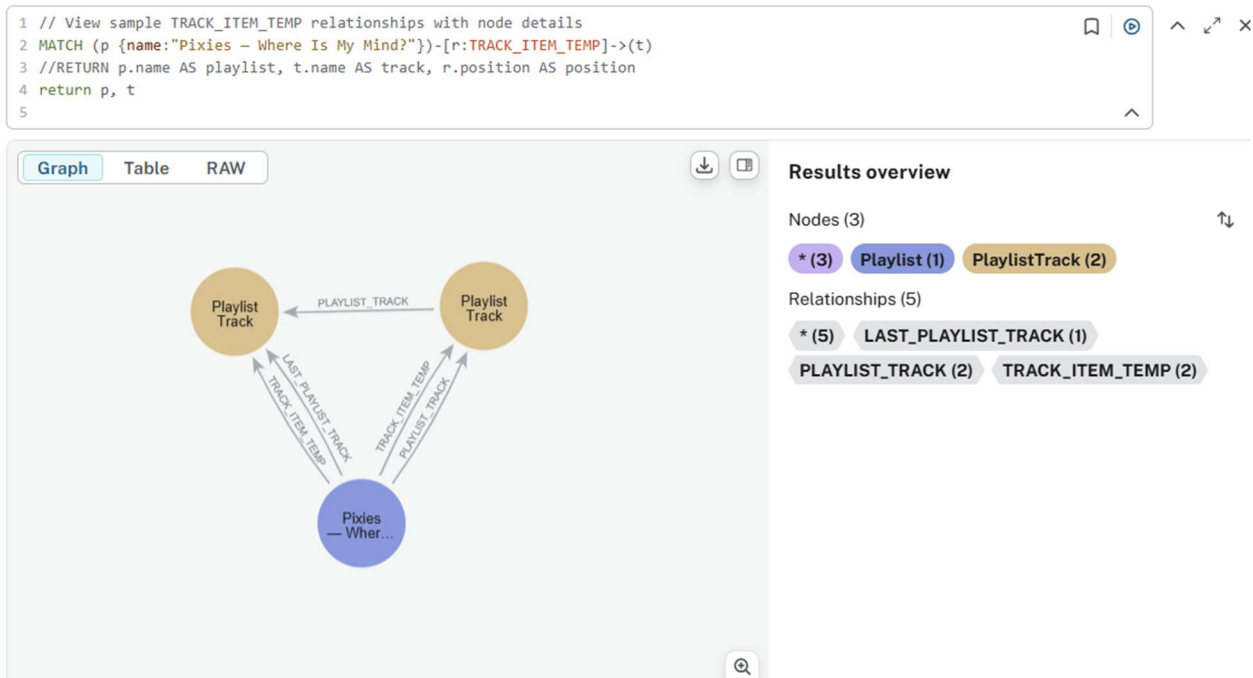
return p, t

LIMIT 25;

```
1 // View sample TRACK_ITEM_TEMP relationships with node details
2 MATCH (p)-[r:TRACK_ITEM_TEMP]->(t)
3 //RETURN p.name AS playlist, t.name AS track, r.position AS position
4 return p, t
5 LIMIT 25;
```

Graph Table RAW





+++++

// Step 2a - First verify what will be deleted

```
MATCH (p)-[r:TRACK_ITEM_TEMP]->(t)
```

```
RETURN
```

```
  p.name AS playlistName,
```

```
  COUNT(r) AS trackCount,
```

```
  COLLECT(t.name)[0..5] AS sampleTracks
```

```
LIMIT 10;
```

// Step 2b - Then proceed with deletion (if correct)

```
MATCH ()-[r:TRACK_ITEM_TEMP]->()
```

```
DELETE r
```

```
RETURN "Deleted " + COUNT(r) + " TRACK_ITEM_TEMP relationships";
```

+++++

// Safe approach that shows what was deleted

```
MATCH ()-[r:TRACK_ITEM_TEMP]->()
```

```
WITH COLLECT(r) AS relationshipsToDelete
```

```
FOREACH (r IN relationshipsToDelete | DELETE r)
```

```
RETURN SIZE(relationshipsToDelete) AS deletedRelationshipCount;
```

+++++

011-refactor-playlist-linked-4

Don't execute original query:

```
//Stage 3
//Step 2
MATCH ()-[r:TRACK_ITEM_TEMP]-()
CALL(r){
  DELETE r
}
IN TRANSACTIONS
```

chapter01\$

```
MATCH ()-[r:TRACK_ITEM_TEMP]->()
WITH COLLECT(r) AS relationshipsToDelete
FOREACH (r IN relationshipsToDelete | DELETE r)
RETURN SIZE(relationshipsToDelete) AS deletedRelationshipCount;
```

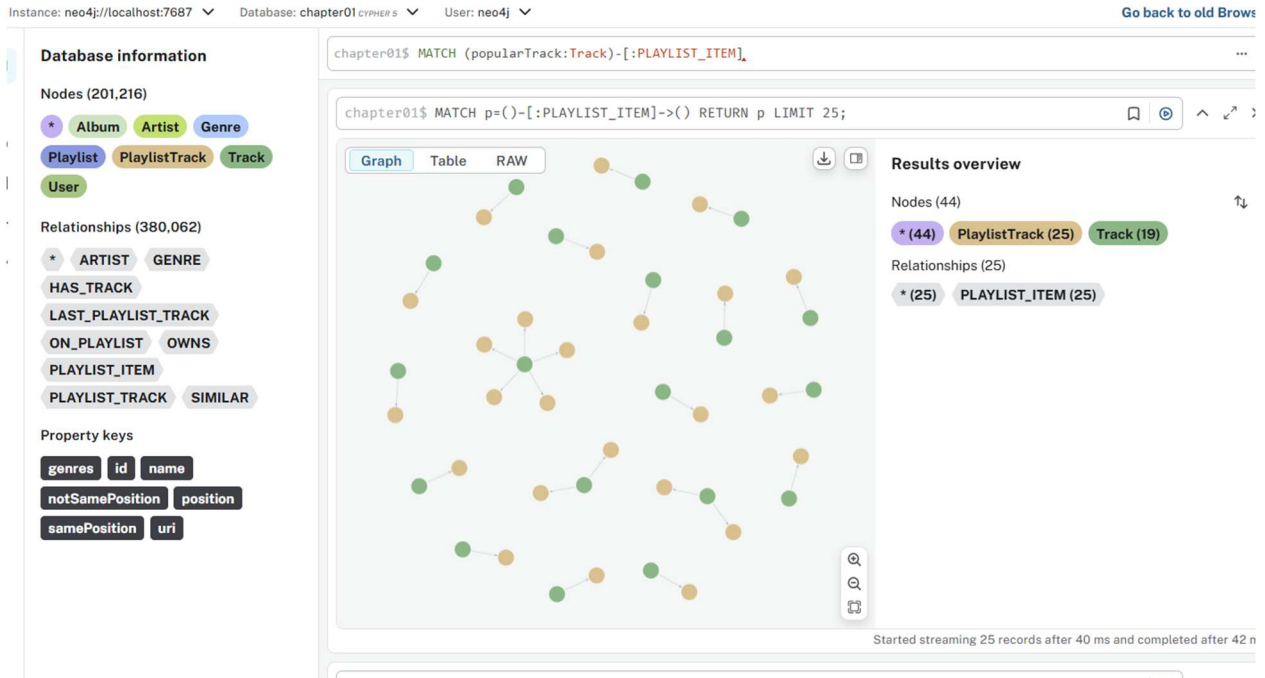
The screenshot shows a Cypher query execution interface. At the top, the query is displayed with line numbers 1 through 4. Below the query, there are two tabs: 'Table' (selected) and 'RAW'. The results are shown in a table with one column, 'deletedRelationshipCount', and one row with the value '73230'. At the bottom, there is a status bar with a green checkmark icon, the text 'Deleted 73,230 relationships', and the text 'Started stream'.

```
1 MATCH ()-[r:TRACK_ITEM_TEMP]->()
2 WITH COLLECT(r) AS relationshipsToDelete
3 FOREACH (r IN relationshipsToDelete | DELETE r)
4 RETURN SIZE(relationshipsToDelete) AS deletedRelationshipCount;
```

Table RAW

| deletedRelationshipCount |
|--------------------------|
| 73230 |

Deleted 73,230 relationships Started stream



012-recommendation-2

chapter01\$

//Find popular tracks

MATCH (popularTrack:Track)-[:PLAYLIST_ITEM]->()

WITH popularTrack, count(*) as playlistCount

ORDER BY playlistCount DESC

LIMIT 10

WITH collect(elementId(popularTrack)) as popularTracks

// For a given Playlist

MATCH (playlist:Playlist) WHERE playlist.name = "all that jazz"

// Find the last track

MATCH (playlist)-[:LAST_PLAYLIST_TRACK]->(lastTrackItem)

// Get the previous tracks

WITH playlist, lastTrackItem, popularTracks, COLLECT {

MATCH (playlist) ((-[:PLAYLIST_TRACK]->())) {1,100} (previousTrackItem)

WHERE previousTrackItem <> lastTrackItem

RETURN elementId(previousTrackItem)

} AS previousTrackItems

// Find other playlists that have the same the last track

```
MATCH (playlist)-[:SIMILAR]-(otherPlaylist:Playlist)-[:LAST_PLAYLIST_TRACK]-
>(otherLastTrack)<-[:PLAYLIST_ITEM]-(:Track)-[:PLAYLIST_ITEM]->()<-
[:LAST_PLAYLIST_TRACK]-(playlist)
```

```
// Find other tracks which are not in the given playlist
```

```
MATCH (otherPlaylist) (()-[:PLAYLIST_TRACK]->()) {1,100} (recommendationItem)<-
[:PLAYLIST_ITEM]-(recommendation)
```

```
WHERE NOT elementId(recommendationItem) IN previousTrackItems
```

```
AND NOT elementId(recommendationItem) IN popularTracks
```

```
// Score them by how frequently they appear
```

```
RETURN recommendation.id as recommendedTrackId, recommendation.name AS
recommendedTrack, otherPlaylist.name AS fromPlaylist, count(*) AS score
```

```
ORDER BY score DESC
```

```
LIMIT 5
```

Instance: neo4j://localhost:7687 Database: chapter01 cypher.s User: neo4j

Database information

Nodes (201,216)

- Album
- Artist
- Genre
- Playlist
- PlaylistTrack
- Track
- User

Relationships (380,062)

- ARTIST
- GENRE
- HAS_TRACK
- LAST_PLAYLIST_TRACK
- ON_PLAYLIST OWNS
- PLAYLIST_ITEM
- PLAYLIST_TRACK
- SIMILAR

Property keys

- genres
- id
- name
- notSamePosition
- position
- samePosition
- uri

chapter01\$

```
25 AND NOT elementId(recommendationItem) IN popularTracks
26
27 // Score them by how frequently they appear
28 RETURN recommendation.id as recommendedTrackId, recommendation.name AS recommendedTrack, otherPlaylist.name AS fromPlaylist, count(*) AS score
29 ORDER BY score DESC
30 LIMIT 5
31
```

Table RAW

| | recommendedTrac | recommendedTrac | fromPlaylist | score |
|---|------------------------------|---|---------------|-------|
| 1 | "04md2B5nmsojIk TSUWff2C" | "I Can't Give Y ou Anything But Love" | "smooth jazz" | 1 |
| 2 | "084QYJ08vynWzF gHEMMAeN" | "For All We Kno w" | "smooth jazz" | 1 |
| 3 | "07fTNXLb42wb1N gBMDwHk5" | "What's New" | "smooth jazz" | 1 |
| 4 | "07Nlg9PvXqVwDV JtX0gjDs" | "Since I Fell F or You" | "smooth jazz" | 1 |
| 5 | "016HKF7A407tat oJ9BgZM1" | "Say It (Over A nd Over Again)" | "smooth jazz" | 1 |

Started streaming 5 records

Let's debug if relationships are missing:

```
chapter01$
```

```
MATCH (playlist:Playlist {name: "all that jazz"})
```

```
CALL (playlist) {
```

```
//WITH playlist
```

```
MATCH (playlist)-[r]->(target)
```

```
RETURN type(r) AS relationshipType, COUNT(r) AS count
```

```
UNION
```

```

//WITH playlist
MATCH (source)-[r]->(playlist)
RETURN type(r) AS relationshipType, COUNT(r) AS count
}
RETURN relationshipType, count

```

Instance: neo4j://localhost:7687 Database: chapter01 CYPHER 5 User: neo4j

Database information

Nodes (201,216)

- Album Artist Genre
- Playlist PlaylistTrack Track
- User

Relationships (380,062)

- ARTIST GENRE
- HAS_TRACK
- LAST_PLAYLIST_TRACK
- ON_PLAYLIST OWNS
- PLAYLIST_ITEM
- PLAYLIST_TRACK SIMILAR

Property keys

- genres id name
- notSamePosition position
- samePosition uri

chapter01\$

```

1 MATCH (playlist:Playlist {name: "all that jazz"})
2 CALL (playlist) {
3   //WITH playlist
4   MATCH (playlist)-[r]->(target)
5   RETURN type(r) AS relationshipType, COUNT(r) AS count
6 UNION
7   //WITH playlist
8   MATCH (source)-[r]->(playlist)
9   RETURN type(r) AS relationshipType, COUNT(r) AS count
10 }
11 RETURN relationshipType, count |

```

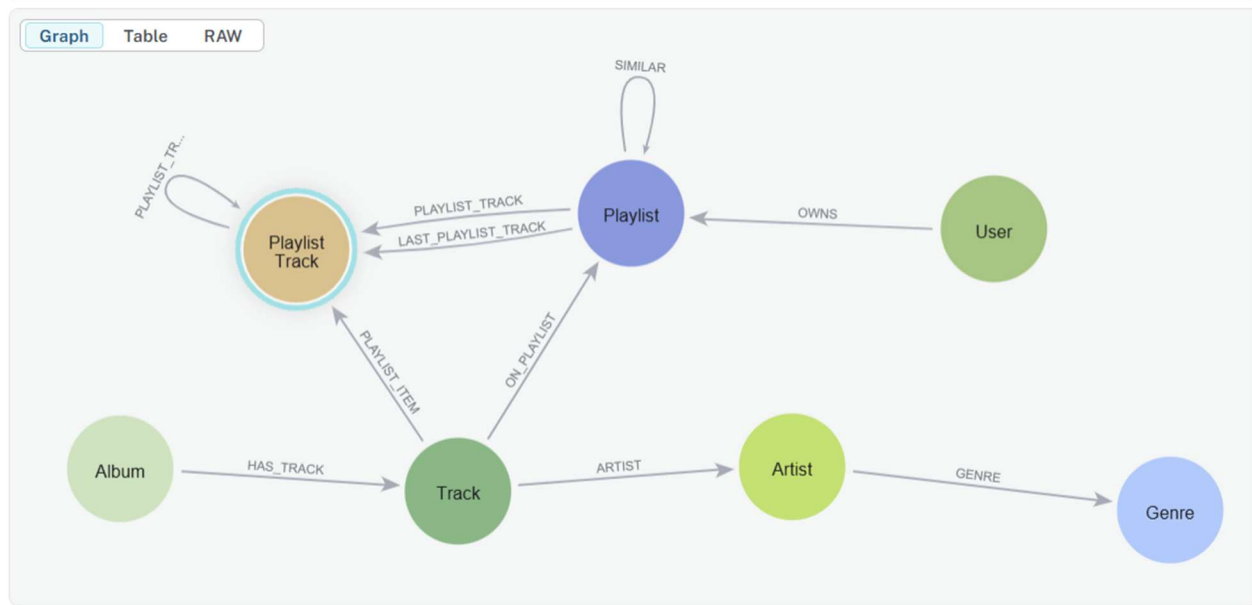
Table RAW

| | relationshipType | count |
|---|-----------------------|-------|
| 1 | "PLAYLIST_TRACK" | 1 |
| 2 | "LAST_PLAYLIST_TRACK" | 1 |
| 3 | "OWNS" | 1 |
| 4 | "SIMILAR" | 2 |
| 5 | "ON_PLAYLIST" | 382 |

Let's see the structure of the Graph: There are a couple of ways to do this depending on whether you have APOC installed or not.

CALL apoc.meta.graph()

```
chapter01$ call apoc.meta.graph()
```



`CALL apoc.meta.schema()`

Returns a map with all **labels, relationships, and property types**. Great for structural analysis (not visual).

Without APOC – Using Built-in Neo4j Procedures

If APOC is **not installed**, you can still get basic structure info.

Show all node labels:

`CALL db.labels()`

Instance: neo4j://localhost:7687 Database: chapter01 CYPHER 5 User: neo4j

Database information

Nodes (201,216)

*

Album

Artist

Genre

Playlist

PlaylistTrack

Track

User

Relationships (380,062)

*

ARTIST

GENRE

HAS_TRACK

LAST_PLAYLIST_TRACK

ON_PLAYLIST

OWNS

PLAYLIST_ITEM

PLAYLIST_TRACK

SIMILAR

Property keys

genres

id

name

notSamePosition

position

samePosition

uri

chapter01\$

chapter01\$ CALL db.labels()

Table

RAW

label

1 "Track"

2 "Album"

3 "Artist"

4 "Playlist"

5 "User"

6 "Genre"

7 "PlaylistTrack"

Show all relationship types:
CALL db.relationshipTypes()

Instance: neo4j://localhost:7687 Database: chapter01 CYPHER S User: neo4j

Database information

Nodes (201,216)

* Album Artist Genre

Playlist PlaylistTrack Track

User

Relationships (380,062)

* ARTIST GENRE

HAS_TRACK

LAST_PLAYLIST_TRACK

ON_PLAYLIST OWNS

PLAYLIST_ITEM

PLAYLIST_TRACK SIMILAR

Property keys

genres id name

notSamePosition position

samePosition uri

chapter01\$

chapter01\$ CALL db.relationshipTypes()

Table RAW

relationshipType

1 "HAS_TRACK"

2 "ARTIST"

3 "OWNS"

4 "SIMILAR"

5 "ON_PLAYLIST"

6 "GENRE"

7 "PLAYLIST_ITEM"

8 "PLAYLIST_TRACK"

9 "LAST_PLAYLIST_TRACK"

Show all property keys:
CALL db.propertyKeys()

Instance: neo4j://localhost:7687 Database: chapter01 CYPHER 5 User: neo4j

Database information

Nodes (201,216)

- * Album Artist Genre
- Playlist PlaylistTrack Track
- User

Relationships (380,062)

- * ARTIST GENRE
- HAS_TRACK
- LAST_PLAYLIST_TRACK
- ON_PLAYLIST OWNS
- PLAYLIST_ITEM
- PLAYLIST_TRACK SIMILAR

Property keys

- genres id name
- notSamePosition position
- samePosition uri

chapter01\$

chapter01\$ CALL db.propertyKeys()

Table RAW

propertyKey

- 1 "id"
- 2 "uri"
- 3 "name"
- 4 "position"
- 5 "samePosition"
- 6 "notSamePosition"
- 7 "genres"

Show schema overview

CALL db.schema.visualization()

Instance: neo4j://localhost:7687 Database: chapter01 CYPHER 5 User: neo4j

Database information

Nodes (201,216)

- * Album Artist Genre
- Playlist PlaylistTrack Track
- User

Relationships (380,062)

- * ARTIST GENRE
- HAS_TRACK
- LAST_PLAYLIST_TRACK
- ON_PLAYLIST OWNS
- PLAYLIST_ITEM
- PLAYLIST_TRACK SIMILAR

Property keys

- genres id name
- notSamePosition position
- samePosition uri

chapter01\$

chapter01\$ CALL db.schema.visualization()

Graph Table RAW

Check if the playlist has any tracks at all:

chapter01\$

```
MATCH (t)-[:ON_PLAYLIST]->(playlist:Playlist {name: "all that jazz", id:
"0kl1NEhdo4bJemAqZn5xJd"})
MATCH (playlist)-[:LAST_PLAYLIST_TRACK]->(lastTrackItem)
WITH count(t) as cnt, playlist.name as pname , lastTrackItem.name as lst
RETURN pname, cnt, lst
```



The screenshot shows a Cypher query execution interface. At the top, the query is displayed in a code editor. Below the query, there are two tabs: 'Table' (selected) and 'RAW'. The results are shown in a table with three columns: 'pname', 'cnt', and 'lst'. The first row of results shows 'all that jazz' for pname, 382 for cnt, and null for lst.

```
1 MATCH (t)-[:ON_PLAYLIST]->(playlist:Playlist {name: "all that jazz", id: "0kl1NEhdo4bJemAqZn5xJd"})
2 MATCH (playlist)-[:LAST_PLAYLIST_TRACK]->(lastTrackItem)
3 WITH count(t) as cnt, playlist.name as pname , lastTrackItem.name as lst
4 RETURN pname, cnt, lst
```

| | pname | cnt | lst |
|---|-----------------|-----|------|
| 1 | "all that jazz" | 382 | null |

chapter01\$

```
MATCH (playlist:Playlist {name: "all that jazz"})
//OPTIONAL MATCH (playlist)-[r:PLAYLIST_TRACK]->(PlaylistTrack)
OPTIONAL MATCH (t)-[:ON_PLAYLIST]-> (playlist)-[r:LAST_PLAYLIST_TRACK]-
>(LastPlaylistTrack)<-[:PLAYLIST_ITEM]-(t2)
RETURN count(*) as lst, playlist.name, t2.name , t2.id, t2.uri
```

chapter01\$

```

1 MATCH (playlist:Playlist {name: "all that jazz"})
2 //OPTIONAL MATCH (playlist)-[r:PLAYLIST_TRACK]->(PlaylistTrack)
3 OPTIONAL MATCH (t)-[:ON_PLAYLIST]-> (playlist)-[r:LAST_PLAYLIST_TRACK]->(LastPlaylistTrack)<-[:PLAYLIST_ITEM]-(t2)
4 RETURN count(*) as lst, playlist.name, t2.name , t2.id, t2.uri

```

Table

RAW

| lst | playlist.name | t2.name | t2.id | t2.uri |
|-------|-----------------|--|------------------------------|--|
| 1 382 | "all that jazz" | "Journey Into M elody - 2007 Di gital Remaster/ Rudy Van Gelder Edition" | "7ysmJhXFQtiBQl k6EZ6sks" | "spotify:track: 7ysmJhXFQtiBQlk 6EZ6sks" |

013-refactor-playlist-linked-5

chapter01\$

//Stage 4

MATCH ()-[r:ON_PLAYLIST]-()

CALL(r){

DELETE r

}

IN TRANSACTIONS

Instance: neo4j://localhost:7687 Database: chapter01 CYPHER 5 User: neo4j

Database information

Nodes (201,216)

- Album
- Artist
- Genre
- Playlist
- PlaylistTrack
- Track
- User

Relationships (306,832)

- ARTIST
- GENRE
- HAS_TRACK
- LAST_PLAYLIST_TRACK
- OWNS
- PLAYLIST_ITEM
- PLAYLIST_TRACK
- SIMILAR

Property keys

- genres
- id
- name
- notSamePosition
- position
- samePosition
- uri

chapter01\$

```

1 //Stage 4
2 MATCH ()-[r:ON_PLAYLIST]-()
3 CALL(r) {
4   DELETE r
5 }
6 IN TRANSACTIONS

```

Deleted 73,230 relationships

```

1 //Find popular tracks
2 MATCH (popularTrack:Track)-[:PLAYLIST_ITEM]->()
3 WITH popularTrack, count(*) as playlistCount
4 ORDER BY playlistCount DESC
5 LIMIT 10
6 WITH collect(elementId(popularTrack)) as popularTracks
7 // For a given Playlist
8 MATCH (playlist:Playlist) WHERE playlist.name = "all that jazz"
9 // Find the last track
10 MATCH (playlist)-[:LAST_PLAYLIST_TRACK]->(lastTrackItem)
11
12 // Get the previous tracks
13 WITH playlist, lastTrackItem, popularTracks, COLLECT {
14 MATCH (playlist) ({}-[:PLAYLIST_TRACK]->()) {1,100} (previousTrackItem)
15 WHERE previousTrackItem <> lastTrackItem
16 RETURN elementId(previousTrackItem)

```

Simple DELETE (Recommended)

//Stage 4 - Clean up ON_PLAYLIST relationships

```

MATCH ()-[r:ON_PLAYLIST]-()
DELETE r

```

Batched DELETE for Large Datasets

//Stage 4 - Batched cleanup of ON_PLAYLIST relationships

```

MATCH ()-[r:ON_PLAYLIST]-()
CALL {
  WITH r
  DELETE r
} IN TRANSACTIONS OF 10000 ROWS

```

Using apoc.periodic.iterate (For Very Large Datasets)

//Stage 4 - Efficient batch deletion for very large datasets

```

CALL apoc.periodic.iterate(
  "MATCH ()-[r:ON_PLAYLIST]-() RETURN r",
  "DELETE r",
  {batchSize: 10000, parallel: false}
)

```