Read Me

*Team 5*

Purpose

This is a neo4j databases project created by Nathan Sanders, Lily Zephyr, Jono Schwan, and Mauricio Aquino for our CS 4420 Database Systems course at the University of Colorado Springs Colorado. This document introduces the project and covers basic instructions for loading the database from the dump file.

Synopsis

This project utilizes a guitar database with the purpose of displaying information on various chord iterations. The database has information on chords, the notes and finger positions used for those chords, and the fret each note is played on. We chose this database with the intent to view potential patterns between notes and fingers with different chord iterations. We have fine-tuned this database to fit our needs for this project which allows us to view the data in a meaningful way. This has allowed our team to get granular with specific notes and iterations to view their relationships and patterns.

About our team

Our team consists of 2 computer science majors in the Bachelor of Science program and 2 computer security majors in the bachelors of innovation program. Amongst our group members we have three piano players, and a guitar player. All members of the group have a basic understanding of music theory which sparked the interest in this project and the curiosity to explore this database.

Resources

The following is a GitHub link which contains all data for our project:

<https://github.com/nsander2/Guitar-Chords>

The specific final database we used can be found here:

<https://github.com/nsander2/Guitar-Chords/blob/main/Chords-Database_Fixed-Final.csv>

Note: The excel spreadsheet which contains our database will be provided with submission as well.

How to Load Database:

The best way to upload our database is to load in what Neo4j calls a dump file. A dump file is how Neo4j exports databases so that you can send a database to a different computer and load in the database. We found this very effective in a team setting since we each needed to work on the same database. To upload the dump file into Neo4j simply create a new project, go to manage, then open terminal. Once in the terminal cd into bin and run the following command:

neo4j-admin load --from=/backups/neo4j/2016-10-02.dump --database=neo4j –force

Where the red highlight after from= is the file path of our dump file. After running the command, you will be able to open the database in Neo4j and see all of the relationships we have created.

An easier way to load our database is just opening a Neo4j terminal and running our code we have used for this entire project. Here is our code:

LOAD CSV WITH HEADERS FROM 'https://raw.githubusercontent.com/nsander2/Guitar-Chords/main/Chords-Database\_Fixed-Final.csv' AS Line

MERGE(r:Chord {Name: Line.Chord})

MERGE(e:E {Name: Line.E})

MERGE(a:A {Name: Line.A})

MERGE(d:D {Name: Line.D})

MERGE(g:G {Name: Line.G})

MERGE(b:B {Name: Line.B})

MERGE(e2:E2 {Name: Line.E2})

MERGE(s:Structure {Name: Line.Structure})

MERGE(n1:Note1 {Name: Line.Note1})

MERGE(n2:Note2 {Name: Line.Note2})

MERGE(n3:Note3 {Name: Line.Note3})

MERGE(n4:Note4 {Name: Line.Note4})

MERGE(n5:Note5 {Name: Line.Note5})

MERGE(n6:Note6 {Name: Line.Note6})

MERGE (c) - [:IS\_CHORD]-> (r)

MERGE (r) - [:HAS\_E\_STRING]-> (e)

MERGE (r) - [:HAS\_A\_STRING]-> (a)

MERGE (r) - [:HAS\_D\_STRING]-> (d)

MERGE (r) - [:HAS\_G\_STRING]-> (g)

MERGE (r) - [:HAS\_B\_STRING]-> (b)

MERGE (r) - [:HAS\_E2\_STRING]-> (e2)

MERGE (r) - [:HAS\_CHORD\_STRUCTURE]-> (s)

MERGE (r) - [:HAS\_NOTE\_1]-> (n1)

MERGE (r) - [:HAS\_NOTE\_2]-> (n2)

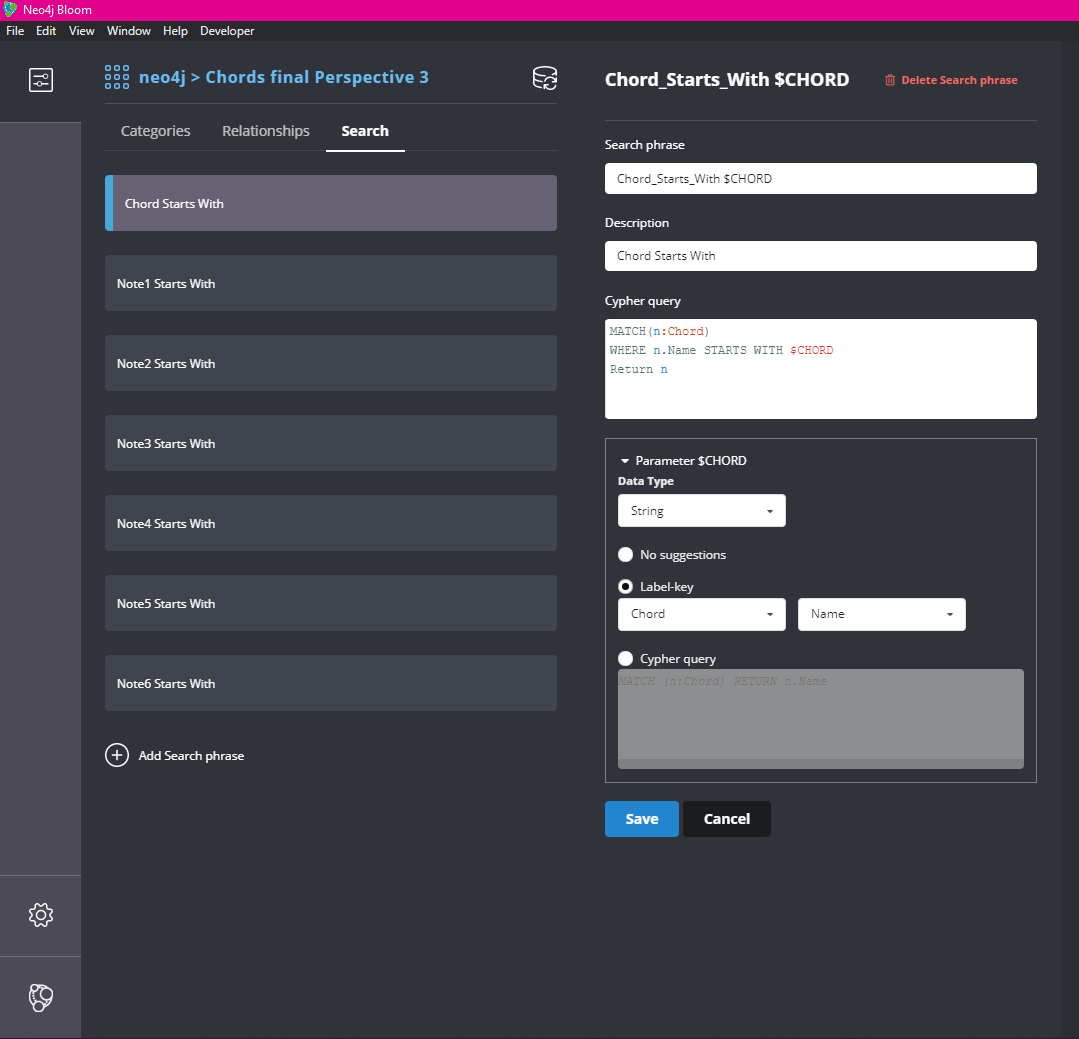
MERGE (r) - [:HAS\_NOTE\_3]-> (n3)

MERGE (r) - [:HAS\_NOTE\_4]-> (n4)

MERGE (r) - [:HAS\_NOTE\_5]-> (n5)

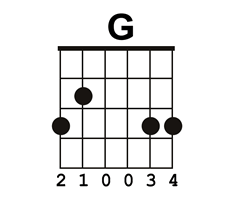
MERGE (r) - [:HAS\_NOTE\_6]-> (n6)

The only downfall of this option is it will not have our rule-based color scheme we created in bloom or our search functions, but those can be easily added using this color scheme and this code:



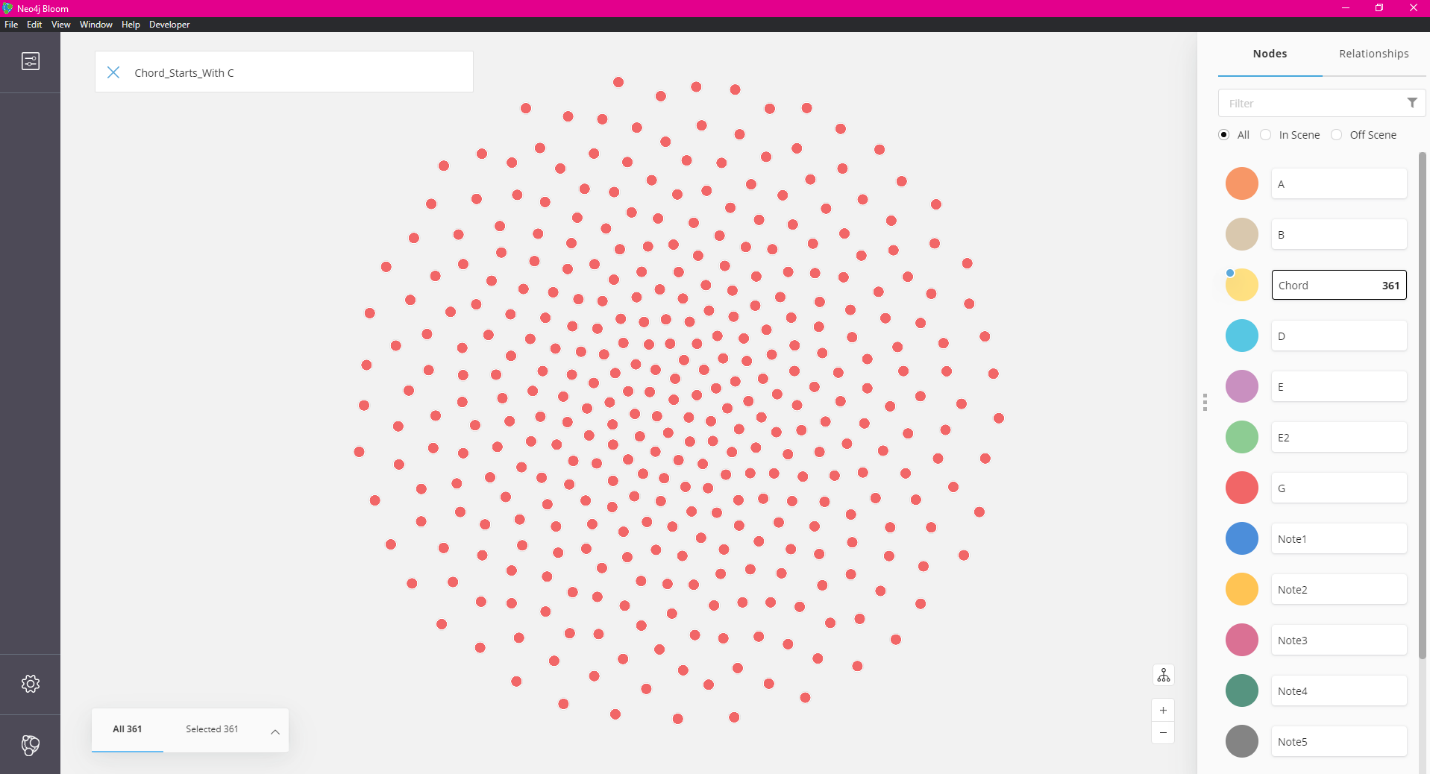


Here are some screenshots from our Neo4j Bloom with our rule-based color scheme, and search functions. In this example we were able to search custom C chords that fit a specific guitar pivoting example. We take this specific “Rock” G chord

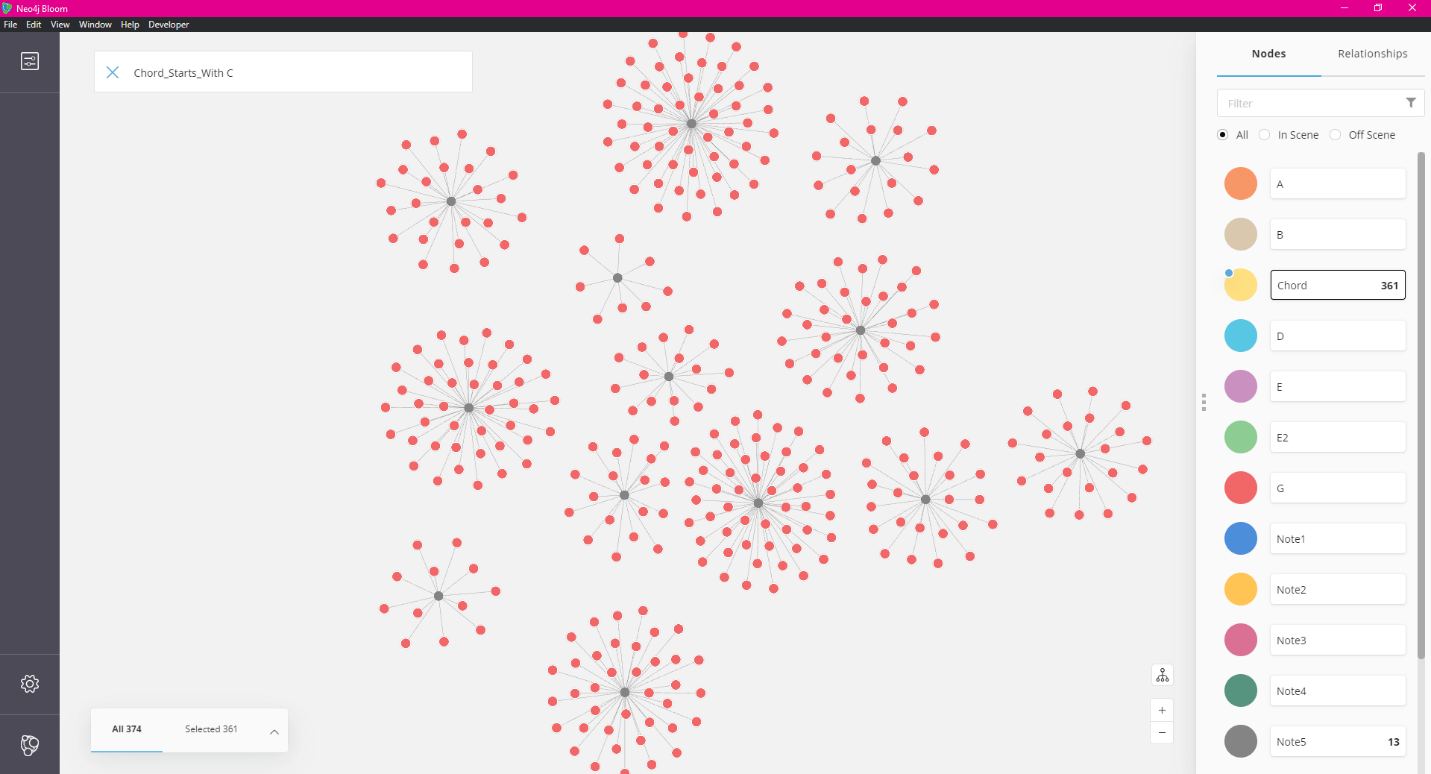


And what we want to do is find a C chord that keeps the third and fourth finger in the same spots. We can use our filter function and our relationships to narrow down all C chords until we have the few that fight the 3rd and 4th finger specifications.

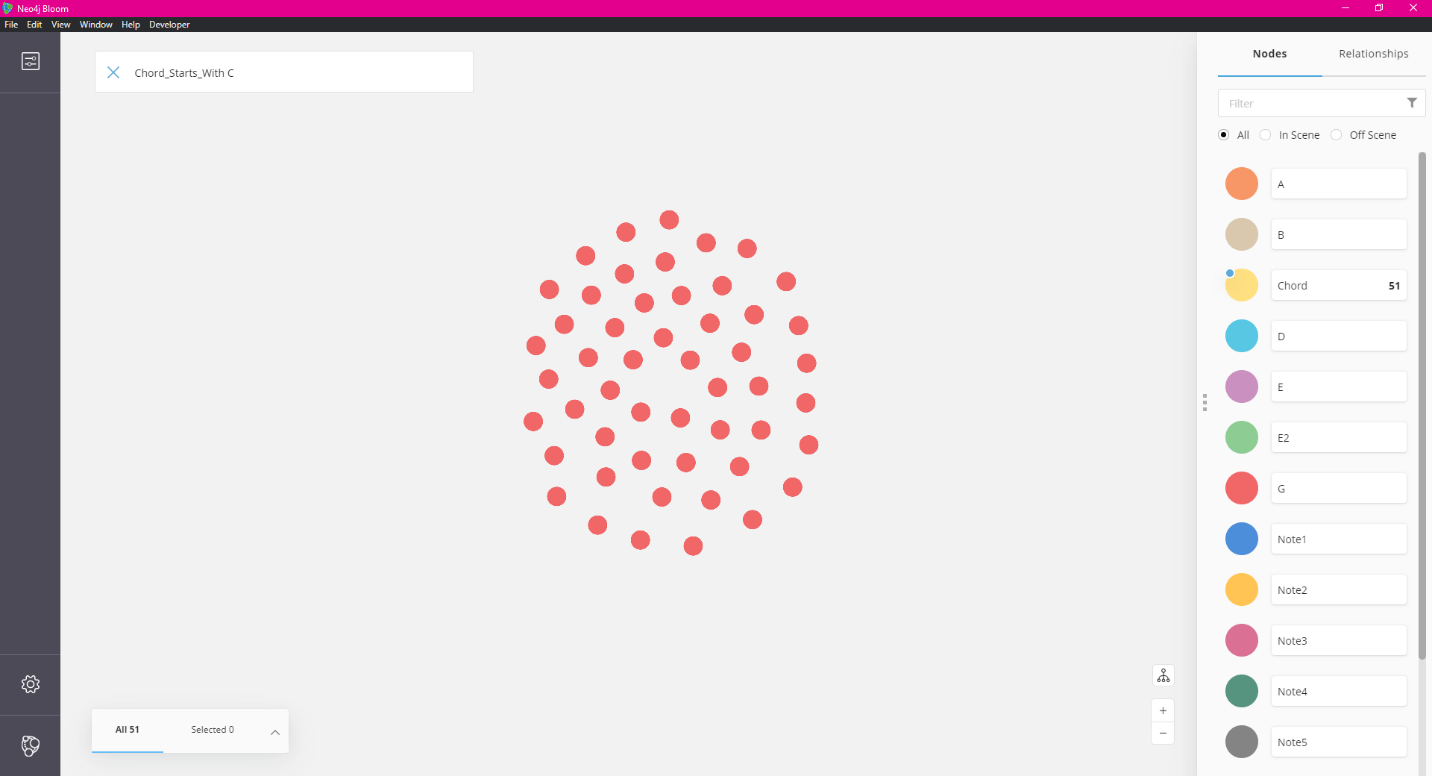
We grab all of the C Chords:



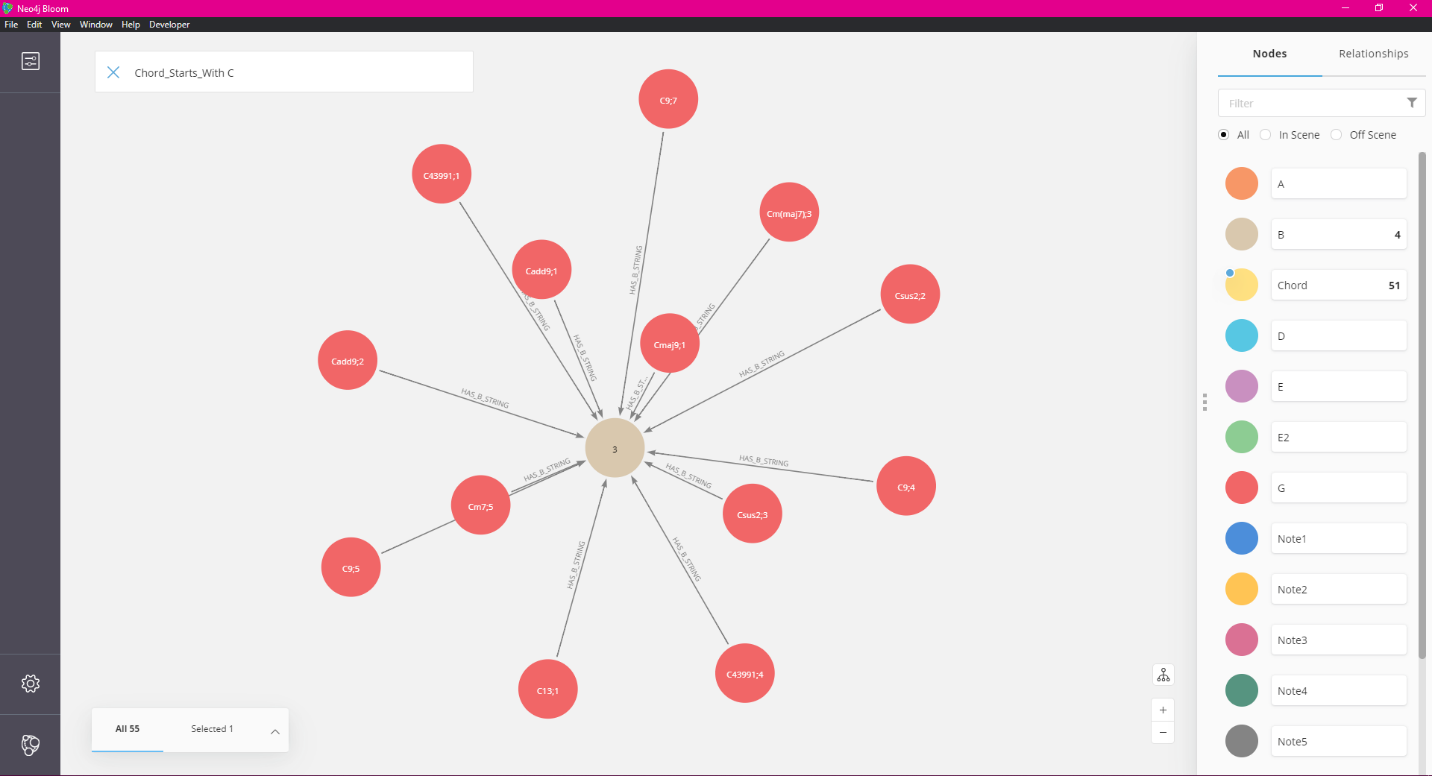
We split them up into the individual note played on the 5th string:



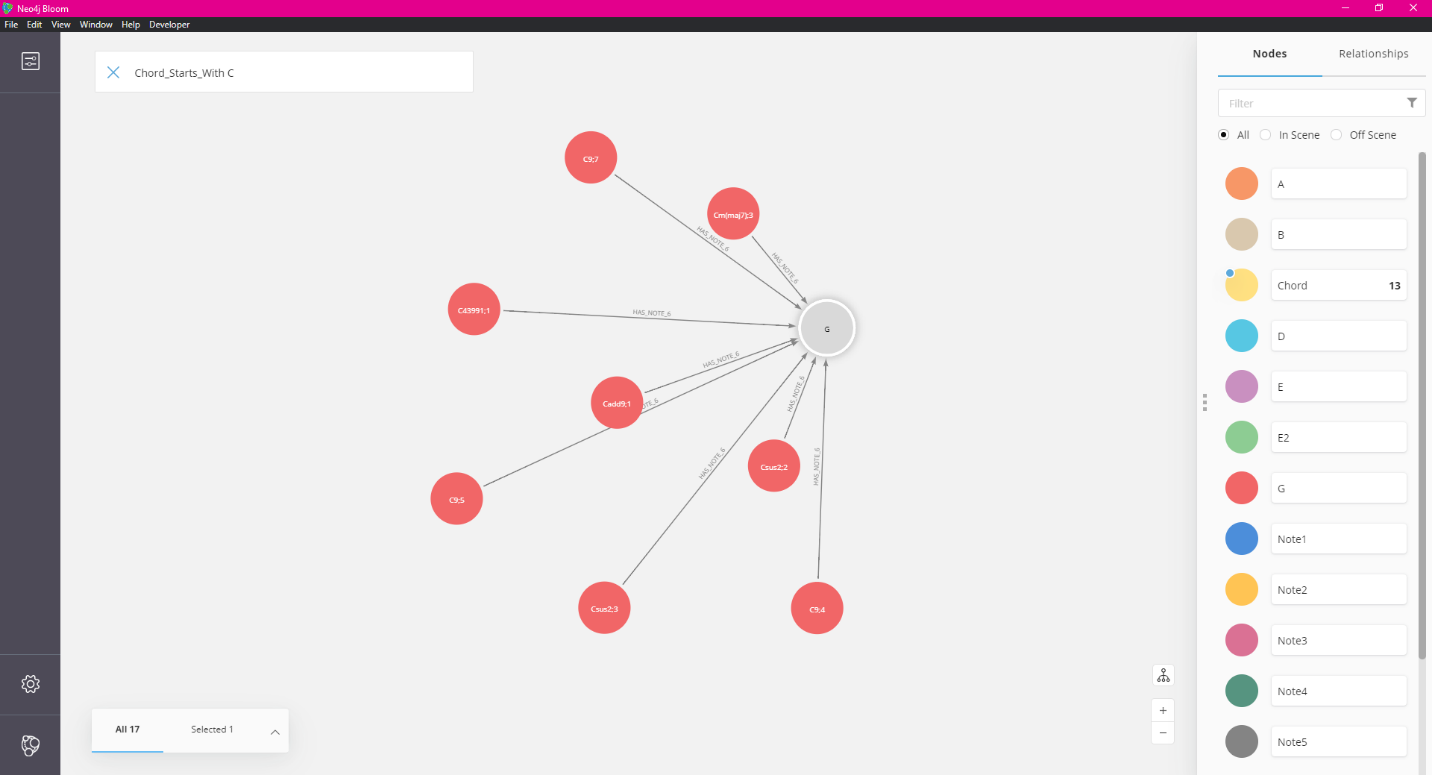
We then only select notes that play a D note on the fifth string since that is the same note as the “Rock” G chord:



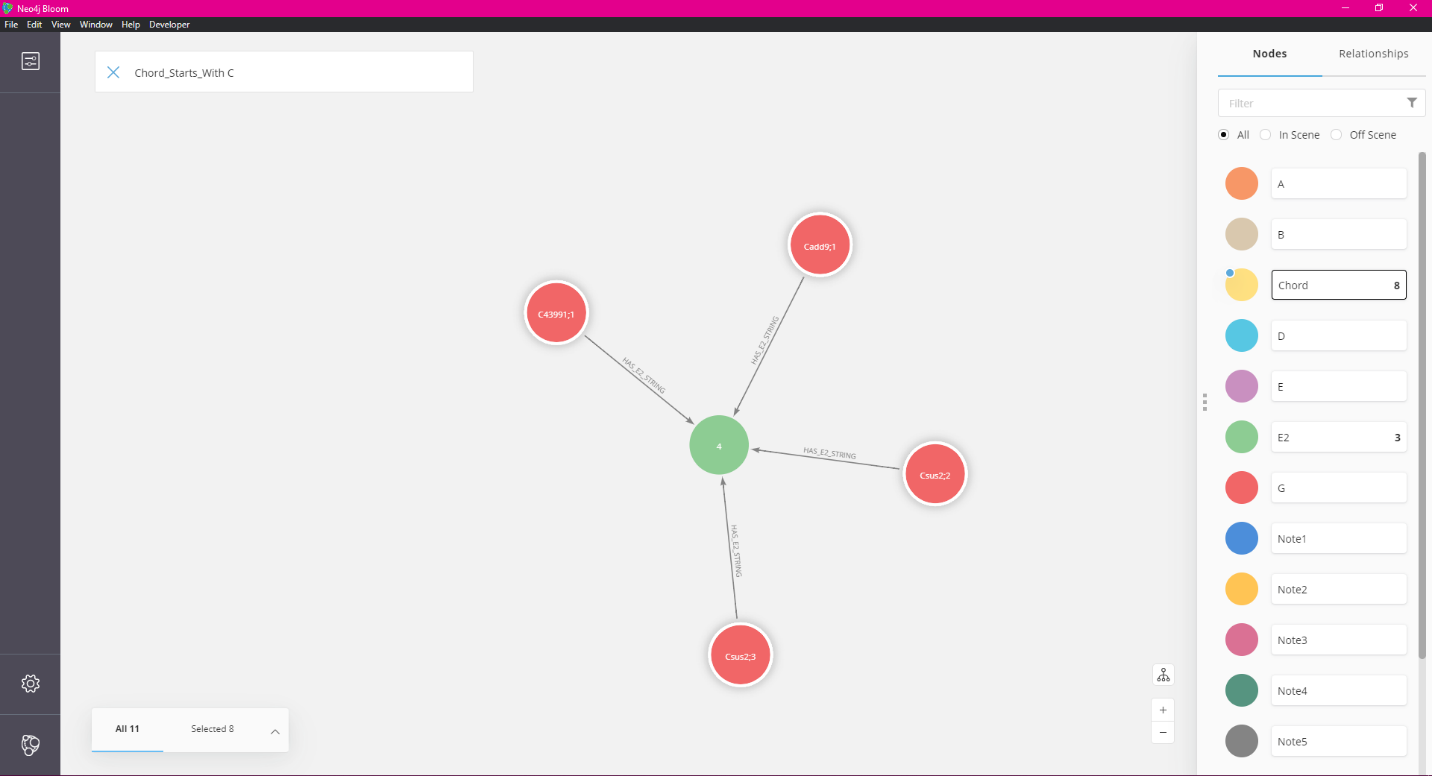
We make sure that we only select those chords that play the correct 3rd finger:



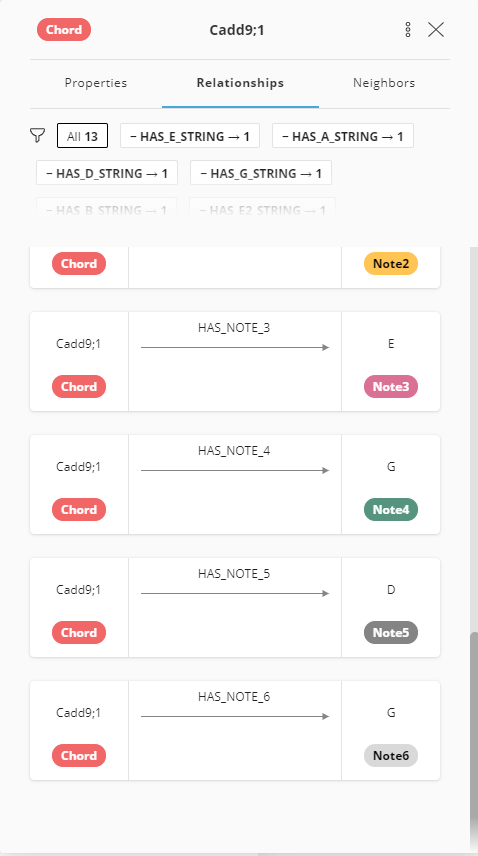
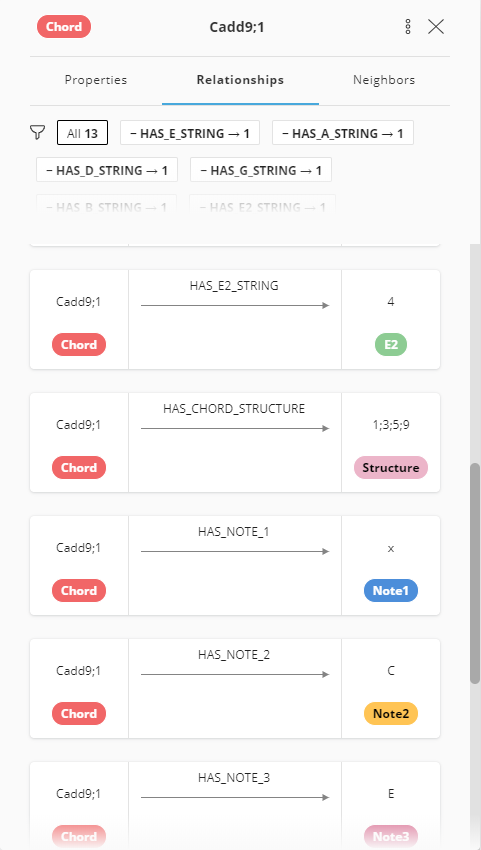
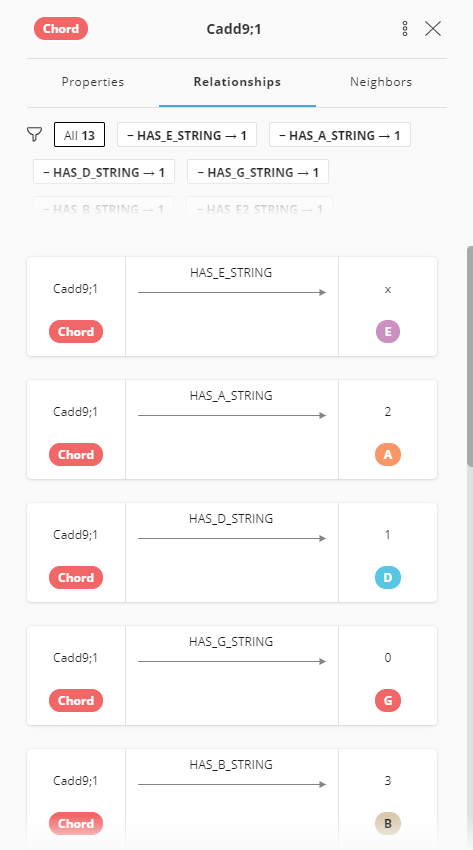
Next up we do the same process but for the 6th string. We select G notes from the sixth string:



And from there we check to make sure that it is indeed played by the fourth finger:



From there we are able to look at specific properties of each node and chose what chord we want to play, in this instance we check out the properties of the Cadd9, 1st iteration chord:



From there we now know both the chord formation and structure and can play that chord. From there we can construct a similar process to create an entire chord progression with the same specifications that looks like:

