GDMA Project:
CDDB as Graph Database

Julian Schelb (1069967)

June 2022

1 Importing Data

This chapter describes design decisions made to map the relational CDDB database to a graph database. See Jupyter notebook `01_Import_Data.ipynb`.

1.1 Mappping of Tables

Entries in tables *artists*, *albums*, *songs* and *cds* represent entities and are mapped to nodes of appropriate type. The cross table *cdtracks* is mapped as relations between CDs and Songs with the property *track*. The cross table *artist2album* is mapped as a triangular relationships, using six relations each to connect Artists, Albums and CDs in both directions.

1.2 Bidirectional Relations

To consgtruct the queries necessary for subsequent tasks, it was structly speaking not necessary to create relations for both directions. However, doing so allows for more readable queries. In addition, the relational SQL database does not define a direction for relations either, and this property is thereby preserved.

1.3 Unique Identifiers

The unique identifiers present in the SQL database are preserved as node properties. Although Neo4j automatically assigns different IDs internally, this preserves compatibility with the SQL version. It is conceivable that in the real worlds, queries may reference those IDs directly. For example as hardcoded list as part of a GUI.

1.4 Entity Names

As in the SQL version, the actual name of a *song*, *genre*, *album* and *artist* is a property named after the entity type and therefore different per node label. This was done to map the SQL version as closely as possible to the graph

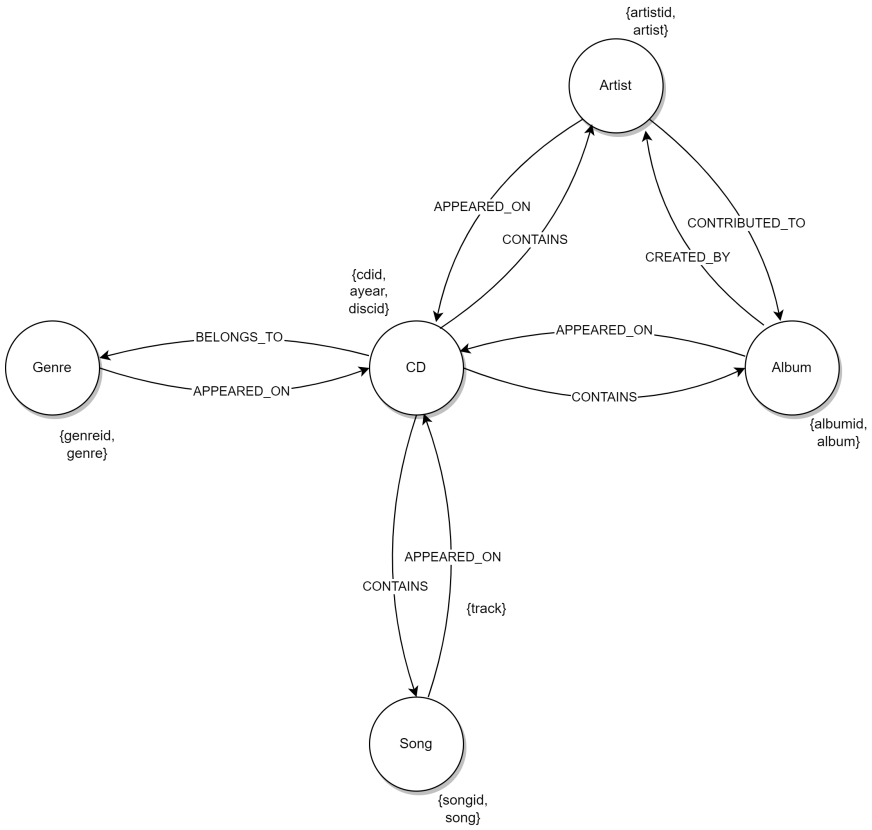

Figure 1: CDDB Graph Schema

version. In real world scenarios it may be beneficial to rename these properties to *name* across all nodes. This would allow for simpler queries in cases where nodes of different types are filtered by name. Also, a joint full-text index would be possible.

1.5 Indices

To speed up `MERGE` and `MATCH` operations, the IDs for nodes with label *Artist*, *Album*, *Genre*, *Song* or *CD* are indexed. In addition, a full text index is used for the names of *Artists*, *Albums* and *Songs* to allow for imprecise searches (fuzzy search).

1.6 Importing Songs

Songs with a trailing backslashcoused problems because the closing quote is escaped. To mitigate this, an extra whitespace character is added to those song

titles. Effected rows can be found with the following sql statement:

```
SELECT *
from cddb.songs s
where song like '%\\'
  and song not like '%\\\\'
```

Statement to update the rows:

```
UPDATE cddb.songs
SET song = song || '␣'
WHERE song LIKE '%\\'
  AND song NOT LIKE '%\\\\'
```

2 Cypher Queries

See Jupyter notebook `02_Cypher_Queries.ipynb`

3 SQL to Cypher

See Jupyter notebook `03_SQL_to_Cypher.ipynb`

4 Searching and Ranking

See Jupyter notebook `05_Searching_Demo.ipynb` for examples search results.

5 Implementation Details

This search incorporates nodes corresponding to the input string on the one hand, and the previously liked albums, artists and songs on the other hand. Two separate scores are calculated in a similar manner. Separately calculated Centrality Scores form the basis. Separate graph projections are created for nodes that match the search input and for nodes that have been previously liked. Subsequently, a centrality score is calculated for each node in both. The basic idea is that the most relevant CDs will have a high centrality score. The "Content Match" score and the "User Preference" score are then combined in a weighted manner. The ten CDs with the highest combined score are presented to the user.

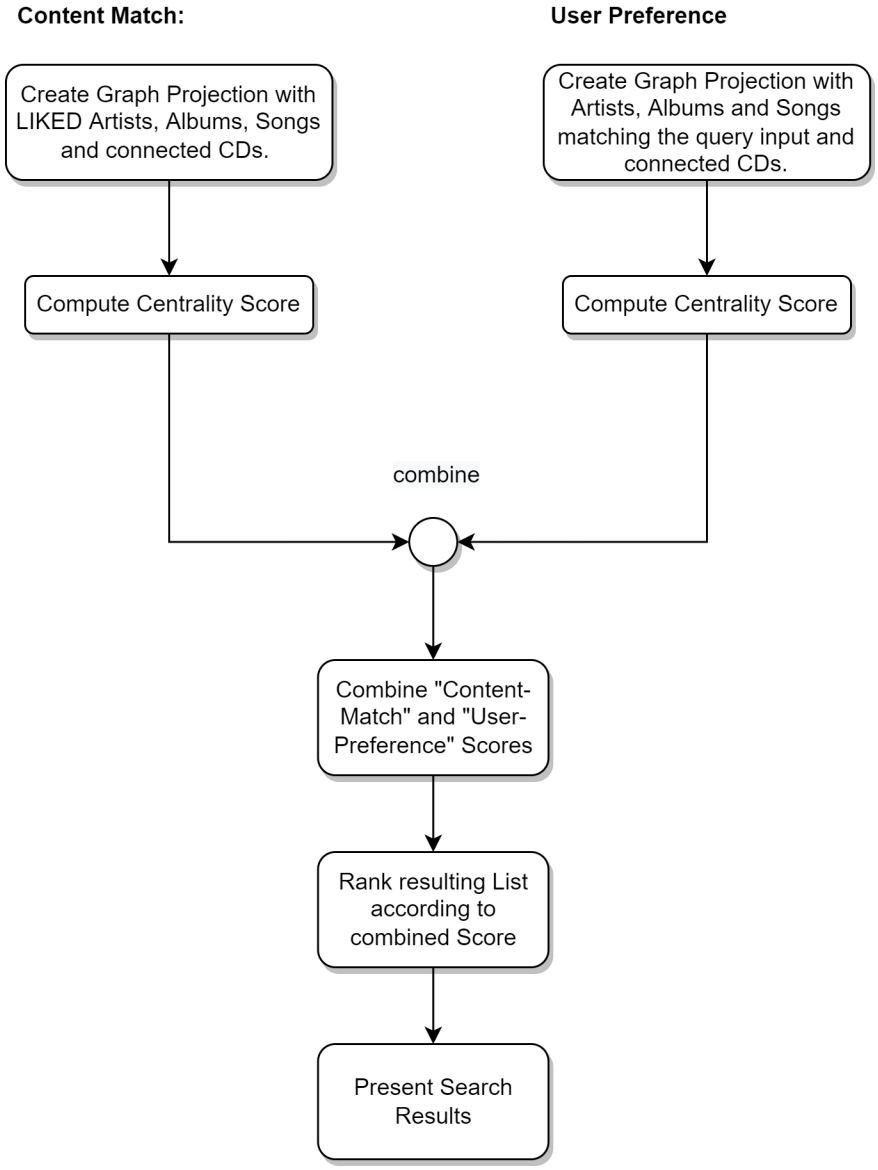

Figure 2: Process of calculating a "Content Match Score" and a "User Preference Score" to produce a combined result list.