A Natural Language Interface over the MusicBrainz Database

Johan Granberg and Michael Minock

Department of Computing Science: Umeå University

Abstract. This paper demonstrates a way to build a natural language interface (NLI) over semantically rich data. Specifically we show this over the MusicBrainz domain, inspired by the second shared task of the QALD-1 workshop. Our approach uses the tool C-Phrase [4] to build an NLI over a set of views defined over the original MusicBrainz database. C-Phrase uses a limited variant of X-Bar theory [3] for syntax and tuple calculus for semantics. The C-Phrase authoring tool works over any domain and only the end configuration has to be redone for each new database covered – a task that does not require deep knowledge about linguistics and the system internals. Working over the MusicBrainz dataset was a challenge due to the size of the database – quite a lot of effort went into optimizing computation times and memory usage to manageable levels. This paper reports on this work.

Keywords: Natural Language Interfaces, Relational Databases, MusicBrainz, C-Phrase

1 Introduction

It has often been noted that natural language interfaces to databases suffer when the manner in which data is stored does not correspond to the user's conceptual view of such data [1, 2, 5]. This mismatch between the way data is structured and the user's conceptual model can be for a variety of reasons, but here we speculate that the three most common reasons are:

1. The database is highly normalized (e.g. to BCNF) for the sake of eliminating update anomalies.
2. The database is highly abstracted, including many attributes per relation so as to avoid cost associated with joins.
3. The database is stored in a semi-structured form corresponding to RDF triples.

It is the thesis of this paper that the user does in fact prefer to query the data in a conceptual form similar to what is represented in the entity-relationship diagram of the database base domain. Naturally if we are given a database in one of the three forms above, we assume that it is possible, via standard view definitions, to transform the data so that it may be accessed (and perhaps even in cases updated) via the conceptual model.

This paper explores these ideas and shows some preliminary results over the MusicBrainz database paired with the 50 natural language queries in the shared task for MusicBrainz the QALD workshop. In section 2 we present our approach to building a natural language interface supporting these queries over the MusicBrainz set. Section 3 discusses our initial results and some anecdotes from our development efforts. Section 4 summarizes our findings and points toward near term and longer term future plans.

2 Approach

2.1 The conceptual model

After looking at the set of 50 natural language queries to be supported over MusicBrainz, we defined the conceptual model appearing in figure 1.

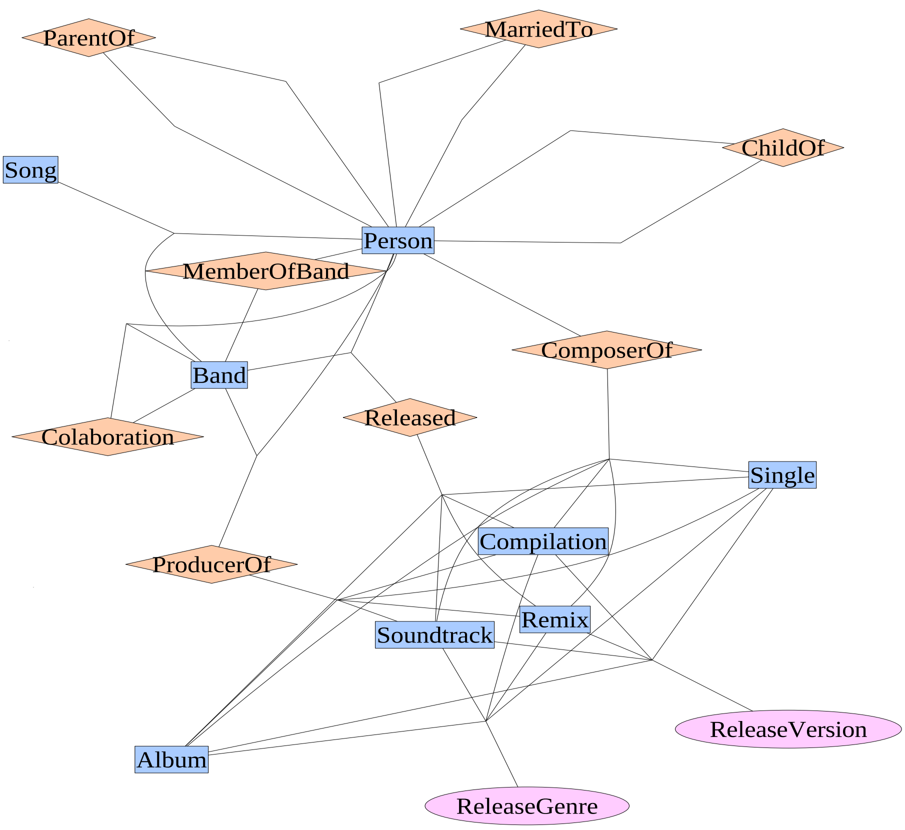

Fig. 1.

2.2 The data source

We had two choices for what we used as the data source for the actual MusicBrainz data. The first was the original data stored in a PostgreSQL database and the second was the RDF dump of the data built for the QALD shared task. Because of our rooting in relational databases, we chose to simply draw on the data in the former relational database.

The database schema of the MusicBrainz database is primarily designed according to option 2 from above. That is, several highly abstracted relations with many attributes represent abstract entities such as `L_ARTIST_ARTIST` and `LT_ARTIST_ARTIST` which has tuples that represent both individual artists (e.g. Bob Dylan) as well as bands (e.g. REM). The database itself is rather large. There are approximately 10.6 million songs, 0.8 million albums and 0.6 million individual and bands. In our experiment we remove the tuples containing non-ascii characters and arrive at 9.7 million songs, 0.7 million albums and 0.4 million artists.

2.3 The view definitions

Views were defined in the standard way over the base relations of the MusicBrainz database. One consideration was whether to materialize these views for perhaps slightly quicker access to the data. This essentially doubles the database size. Our findings are that this leads to a speed up factor of approximately 3. For example using regular views the test query, "which singles did the Dead Kennedys release?" took 2.0 ms on average. Using materialized views it took an average of 0.7 ms.

2.4 Authoring with the C-Phrase administration interface

Once several errors were dealt with (see section 3.1) the authoring process proceeded well. It followed the name, tailor and define method laid out in [4]. Well over half the training set can be authored for within 90 minutes. There are several queries that involve dates that are still not able to be captured, but work to integrate for sophisticated date processing is ongoing.

3 Preliminary Results

3.1 Difficulties

There were several difficulties we encountered that, while perhaps anecdotal, are still worth mentioning. To date the C-Phrase system has been applied only over small databases. MusicBrainz is a more sizable database, so this brought up some scalability issues that we had not earlier experienced. We assume other NLIs attempting to scale to this size of a database might face similar problems.

Large main-memory hash tables The first issue related to memory management issues in CLISP, the version of LISP that C-Phrase is implemented over. There are some unfortunate memory bugs that corrupt CLISP memory when utilization climbs over a certain threshold. It is difficult to track, because the corruption generally causes a `Segmentation Fault` at later steps when memory is accessed. Under normal circumstances this problem does not surface. However to allow for named entity recognition, C-Phrase materializes string values from the database into hash tables to scan for matching values in the user's typed request. In the case of MusicBrainz this means building main memory hash tables containing millions of constants. This was too much for CLISP to handle.

After several false starts, the solution to this problem was to implement a remote hash facility that maintained the main memory hash tables remotely outside of CLISP. The overhead access time for these hash tables is negligible and the implementation is stable and scalable, bounded ultimately by the size of virtual memory.

Limitations of the PostgreSQL SQL optimizer C-Phrase maps English to logical expressions in Codd's tuple calculus. From such logical expressions, SQL is in turn generated. Before our experiment we would generate SQL such as the following to answer the query, "Which singles did the Dead Kennedys release?"

```
SELECT DISTINCT NAME
FROM SINGLE AS x
WHERE
 EXISTS(
   SELECT *
   FROM BAND as y1
   WHERE
     x.artist = y1.id  AND
     y1.name = 'Dead Kennedys').
```

Unfortunately such queries are not taken up by PostgreSQL's optimizer. This query in fact takes 23 minutes to answer on an older solaris server where we run our database.

In contrast the equivalent query

```
SELECT DISTINCT x.NAME
FROM SINGLE AS x,BAND as y1
WHERE x.artist = y1.id  AND y1.name = 'Dead Kennedys'"
```

takes 0.7 seconds on the same server. Here the query time is entirely dominated by the time to establish the connection, the actual query execution is reported as 2 ms. We assume that this is due to the fact that the optimizer has access to both relations on the second line of the query and can plan accordingly. Instead of pestering PostgreSQL about 'improving their optimizer', we altered our translator to produce SQL of the later variety.

3.2 Performance

We are still in the process of collecting performance data. The running example query of "Which singles did the Dead Kennedys release?" shows that the performance is adequate in the case of single join queries. We have run additional tests on queries that exercise more joins and are convinced that the current approach is feasible. We intend to have much more elaborate performance measures and findings by the time this paper goes to press.

3.3 Current coverage

The QALD-Shared task published 50 queries for the MusicBrainz example. We expect to essentially cover all these queries (expect perhaps the query, "Is Liz Story a person or a group?") before the release of the final test set. It will be interesting to see how many of the new queries in the test set will be cover with our current configuration. Naturally we will present this 'result' in the final version of this paper.

4 Conclusions

We were very pleased when we heard of the QALD shared task. We decided to focus our efforts on the more database oriented aspects of the shared task for queries over the MusicBrainz data set. We based out data on the original MusicBrainz database and, after confronting several technical difficulties in scaling C-Phrase, we have managed to build a natural language interface that is on the way to covering all the example in the QALD shared task for MusicBrainz.

We await the next round of testing with the next release of queries. We will document how well we cover these new queries without alteration and the time it takes for us to patch the system to cover the new test set. In addition it is our hope that we will be able to solve several key technical issues and put out an NLI for real-time querying of MusicBrainz by the public.

References

1. I. Androutsopoulos and G.D. Ritchie. Database interfaces. In R. Dale, H. Moisl, and H. Somers, editors, *Handbook of Natural Language Processing*, pages 209–240. Marcel Dekker Inc., 2000.
2. A. Copestake and K. Sparck Jones. Natural language interfaces to databases. *The Natural Language Review*, 5(4):225–249, 1990.
3. R. Jackendoff. *X-bar-Syntax: A Study of Phrase Structure, Linguistic Inquiry Monograph 2.* MIT Press, 1977.
4. M. Minock. C-phrase: A system for building robust natural language interfaces to databases. *Journal of Data and Knowledge Engineering (DKE)*, 3(69):290–302, 2010.
5. W. Ogden and P. Bernick. Using natural language interfaces. Technical Report MCCS-96-299, Computing Research Laboratory, New Mexicon State University, Las Cruces, May 1996.