Storage in MoK: experimenting with graph DBs

Matteo Corradin

October 14, 2016

Abstract

MoK needs to store and handle a huge quantity of data. Experimenting with graph DBs, Orient DB and Node4j.

1 Introduction

MoK has to handle a huge quantity of data, stored on distributed consumer devices; needs to move / replicate data between devices with high frequency, aggregate data (link, merge, update, ecc...) also this with high frequency. We need to store and handle these data, for this purpose we will discuss and will analyze the graph DBs to see if they can be useful in MoK.

2 Goal

Acquire know-how on graphDB, OrientDB and Neo4j as case study.
Based on the requirements of MoK about the persistence of textual information, understand which technologies best meet this requirements.
Build a first prototype (a software module) able to use (one of) these technologies.

3 The Molecules of Knowledge model

3.1 MoK Overview

[Mariani, 2012] The Molecules of Knowledge model is a biochemically-inspired coordination model exploiting and promoting self-organisation of knowledge. The basic motivation behind MoK is the idea that knowledge should autonomously aggregate and diffuse to reach knowledge consumers rather than "be searched". The main pillars of MoK are represented by the following (bio)chemical abstractions:
atoms — the smallest unit of knowledge in MoK, an atom contains information from a source, and belongs to a compartment where it "floats"—and where it is subject to its "laws of nature";
molecules — the MoK units for knowledge aggregation, molecules bond together somehow-related atoms;
enzymes — emitted by MoK catalysts, enzymes influence MoK reactions, thus affecting the dynamics of knowledge evolution within MoK compartments to better match the catalyst's needs;
reactions — working at a given rate, reactions are the biochemical laws regulating the evolution of each MoK compartment, by ruling knowledge aggregation, diffusion, and decay within compartments.
Atoms are the most primitive living pieces of knowledge within the model. A MoK atom is produced by a knowledge source, and conveys a piece of information spawning from the source itself. Hence, along with the content they store, atoms should also store some contextual information to refer to the content's origin, and to preserve its original meaning. As a result, a MoK atom is a triple of the form atom(src, val, attr)c where: src identifies unambiguously the source of knowledge; val is the actual piece of knowledge carried by the atom – any kind of content –; attr is essentially the content's attribute, that is, the additional information that helps understanding it – possibly expressed according to some well-defined ontology or controlled vocabulary –; c is the current concentration of the atom, which is essentially the number of atoms of the kind in the compartment.
Molecules are spontaneous, stochastic, "environment-driven" aggregations of atoms, which in principle are meant to reify some semantic relationship between atoms, thus possibly adding new knowledge to the system—for instance self-aggregated news chunks could shape the conceptual "path" toward a novel interesting news story. Each molecule is simply interpreted as a set of atoms, that is, an unordered collection of somehow semantically-related atoms. A MoK molecule has then a structure of the form molecule(Atoms)c where c is the current concentration of the molecule, and Atoms is the collection of all the atoms currently bonded together in the molecule.
Enzymes One of the key features of MoK is that it interprets prosumer's knowledge- related actions as positive

feedbacks that increase the concentration of related atoms and molecules within the prosumer's compartment, producing the positive feedback required to enable self-organisation of knowledge. A MoK enzyme has then a structure of the form enzyme(Atoms)c where every enzyme – with its own concentration c – explicitly refers to a collection of Atoms that the catalyst's actions have in any way pointed out as of interest for the catalyst him/herself.

3.2 MoK storage requirements

MoK has to handle a huge quantity of data, stored on distributed consumer devices; needs to move / replicate data between devices with high frequency, aggregate data (link, merge, update) also this with high frequency.
MoK has to work on consumer devices, could be also smartphone and tablet instead of conventional pc.
Graph database seems to perfect fit the need of MoK to aggregate, link, merge data with high frequency and also with high flexibility.

4 Graph database

A graph database is a database that uses graph structures for semantic queries with nodes, edges and properties to represent and store data.

4.1 Structure

Graph databases are based on graph theory. Graph databases employ nodes, properties, and edges.

- Nodes represent entities, any item you might want to keep track of.
- Properties are pertinent information that relate to nodes. For instance, if "Wikipedia" were one of the nodes, one might have it tied to properties such as "website", "reference material", or "word that starts with the letter 'w'", depending on which aspects of "Wikipedia" are pertinent to the particular database.
- Edges are the lines that connect nodes to nodes or nodes to properties and they represent the relationship between the two. Most of the important information is really stored in the edges. Meaningful patterns emerge when one examines the connections and interconnections of nodes, properties, and edges.

4.2 Properties

Compared with relational databases, graph databases are often faster for associative data sets and map more directly to the structure of object-oriented applications. They can scale more naturally to large data sets as they do not typically require expensive join operations. As they depend less on a rigid schema, they are more suitable to manage ad hoc and changing data with evolving schemas. Conversely, relational databases are typically faster at performing the same operation on large numbers of data elements.
Graph databases are a powerful tool for graph-like queries, for example computing the shortest path between two nodes in the graph. Other graph-like queries can be performed over a graph database in a natural way (for example graph's diameter computations or community detection).
[Wikipedia, 2015]

4.3 OrientDB

OrientDB is an open source NoSQL database management system written in Java. It is a multi-model database, supporting graph, document, key/value, and object models, but the relationships are managed as in graph databases with direct connections between records. It supports schema-less, schema-full and schema-mixed modes. It has a strong security profiling system based on users and roles and supports querying with Gremlin along with SQL extended for graph traversal.

- Written in: Java
- Main point: Document-based graph database
- License: Apache 2.0
- Protocol: binary, HTTP REST/JSON, or Java API for embedding

- Has transactions, full ACID conformity
- Can be used both as a document and as a graph database (vertices with properties)
- Both nodes and relationships can have metadata
- Multi-master architecture
- Supports relationships between documents via persistent pointers (LINK, LINKSET, LINKMAP, LINKLIST field types)
- SQL-like query language (Note: no JOIN, but there are pointers)
- Web-based GUI (quite good-looking, self-contained)
- Inheritance between classes. Indexing of nodes and relationships
- User functions in SQL or JavaScript
- Sharding
- Advanced path-finding with multiple algorithms and Gremlin traversal language
- Advanced monitoring, online backups are commercially licensed

4.4 Neo4j

Neo4j is an open-source graph database implemented in Java and accessible from software written in other languages using the Cypher query language through a transactional HTTP endpoint. The developers describe Neo4j as an ACID-compliant transactional database with native graph storage and processing. Neo4j is the most popular graph database.
In Neo4j, everything is stored in form of either an edge, a node or an attribute. Each node and edge can have any number of attributes. Both the nodes and edges can be labelled. Labels can be used to narrow searches.

- Written in: Java
- Main point: Graph database - connected data
- License: GPL, some features AGPL/commercial
- Protocol: HTTP/REST (or embedding in Java)
- Standalone, or embeddable into Java applications
- Full ACID conformity (including durable data)
- Both nodes and relationships can have metadata
- Integrated pattern-matching-based query language ("Cypher")
- Also the "Gremlin" graph traversal language can be used
- Indexing of nodes and relationships
- Nice self-contained web admin
- Advanced path-finding with multiple algorithms
- Indexing of keys and relationships
- Optimized for reads
- Has transactions (in the Java API)
- Scriptable in Groovy
- Clustering, replication, caching, online backup, advanced monitoring and High Availability are commercially licensed

5 Neo4j as Case study

I decided to build a prototype in Neo4j, it's the leader in graph db, it's more stable (in terms of bugs) than OrientDB, also it can be embedded in Android.

5.1 Basics

A graph database can store any kind of data using a few simple concepts:
Nodes - graph data records
Relationships - connect nodes
Properties - named data values
The simplest graph has just a single node with some named values called Properties.
Nodes are the name for data records in a graph.
Data is stored as Properties.
Properties are simple name/value pairs, can be strings, numbers, or boolean.
Nodes can be grouped together by applying a Label to each member. A node can have zero or more labels. Labels do not have any properties.
Relationship describes how the records are related.
Relationships always have direction.
Relationships always have a type.
Relationships form patterns of data.
Relationships are equally well traversed in either direction.
This means that there is no need to add duplicate relationships in the opposite direction (with regard to traversal or performance).
While relationships always have a direction, you can ignore the direction where it is not useful in your application.

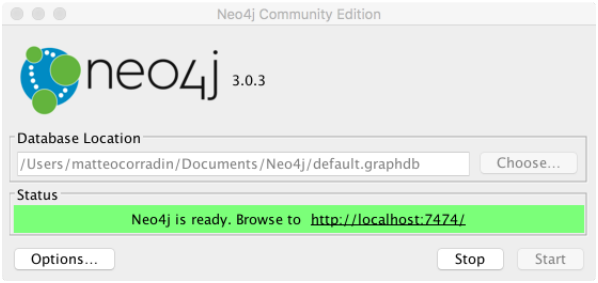
5.2 Neo4j highlights

Connected data is all around us. Neo4j supports rapid development of graph powered systems that take advantage of the rich connectedness of data.
A native graph database: Neo4j is built from the ground up to be a graph database. The architecture is designed for optimizing fast management, storage, and traversal of nodes and relationships. In Neo4j, relationships are first class citizens that represent pre-materialized connections between entities. An operation known in the relational database world as a join, whose performance degrades exponentially with the number of relationships, is performed by Neo4j as navigation from one node to another, whose performance is linear.
This different approach to storing and querying connections between entities provides traversal performance of up to 4 million hops per second and core. As most graph searches are local to the larger neighborhood of a node, the total amount of data stored in a database will not affect operations runtime. Dedicated memory management, and highly scalable and memory efficient operations, contribute to the benefits.
Whiteboard friendly: The property graph approach allows consistent use of the same model throughout conception, design, implementation, storage, and visualization of any domain or use case. This allows all business stakeholders to participate throughout the development cycle. With the schema optional model, the domain model can be evolved continuously as requirements change, without penalty of expensive schema changes and migrations.
Cypher, the declarative graph query language, is designed to visually represent graph patterns of nodes and relationships. This highly capable, yet easily readable, query language is centered around the patterns that express concepts or questions from a specific domain. Cypher can also be extended for narrow optimizations for specific use cases.
Supports rapid development: Neo4j supports fast development of graph powered systems. Neo4j's development stems from the need to run real-time queries on highly related information; something no other database can provide. These unique Neo4j features get you up and running quickly and sustain fast application development for highly scalable applications.
Provides true data safety through ACID transactions: Neo4j uses transactions to guarantee that data is persisted in the case of hardware failure or system crashes.
Designed for business-critical and high-performance operations: Neo4j uses a replicated master-slave cluster setup. It can store hundreds of trillions of entities for the largest datasets imaginable while being sensitive to compact storage. Neo4j can be deployed as a scalable, fault-tolerant cluster of machines. Due to its high scalability, Neo4j clusters require only tens of machines, not hundreds or thousands, saving on cost and operational complexity. Other features for production applications include hot-backups and extensive monitoring.

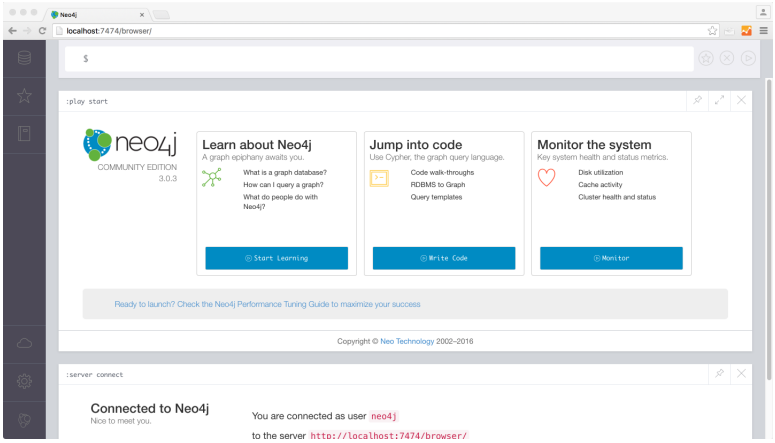
5.3 Getting started

5.3.1 Setup

The latest available version of Neo4J was used for this project. (3.0.3 Community Edition)



Neo4j Desktop App



Neo4j Browser Visualization

5.3.2 Driver

To dialog with Neo4j DB, we can choose between:

- The binary Bolt Protocol
  Wikipedia: https://en.wikipedia.org/wiki/Bolt_(network_protocol)
- Built-in Neo4j Browser application
  Application available on ip:7474.
- Neo4j REST API
  Reference: https://neo4j.com/docs/rest-docs/current/
- Neo4j official drivers
  - Java: the Neo4j Java driver is officially supported by Neo4j and connects to the database using the binary protocol. It aims to be minimal, while being idiomatic to Java.
    Whit maven it's easy.

    ```
    <groupId>org.neo4j.driver</groupId>
    <artifactId>neo4j-java-driver</artifactId>
    <version>1.0.4</version>
    ```

  - .NET
  - Spring
  - JavaScript
  - Python
  - Ruby
  - PHP

- R
- Go
- Erlang / Elixir
- C/C++
- Clojure
- Perl
- Haskell

Some of the language drivers use the Bolt protocol, some other HTTP API.
Reference: https://neo4j.com/developer/language-guides/

6 Experiments

We want to see how Neo4j, and in general a graph database, perform on large dataset. In particular performance on query that involves search of similar elements.

6.1 Cypher Query Language

Cypher is the declarative query language for Neo4j.
Key principles and capabilities of Cypher are as follows:

- Cypher matches patterns of nodes and relationship in the graph, to extract information or modify the data.
- Cypher has the concept of identifiers which denote named, bound elements and parameters.
- Cypher can create, update, and remove nodes, relationships, labels, and properties.
- Cypher manages indexes and constraints.

6.1.1 Cypher Refcard

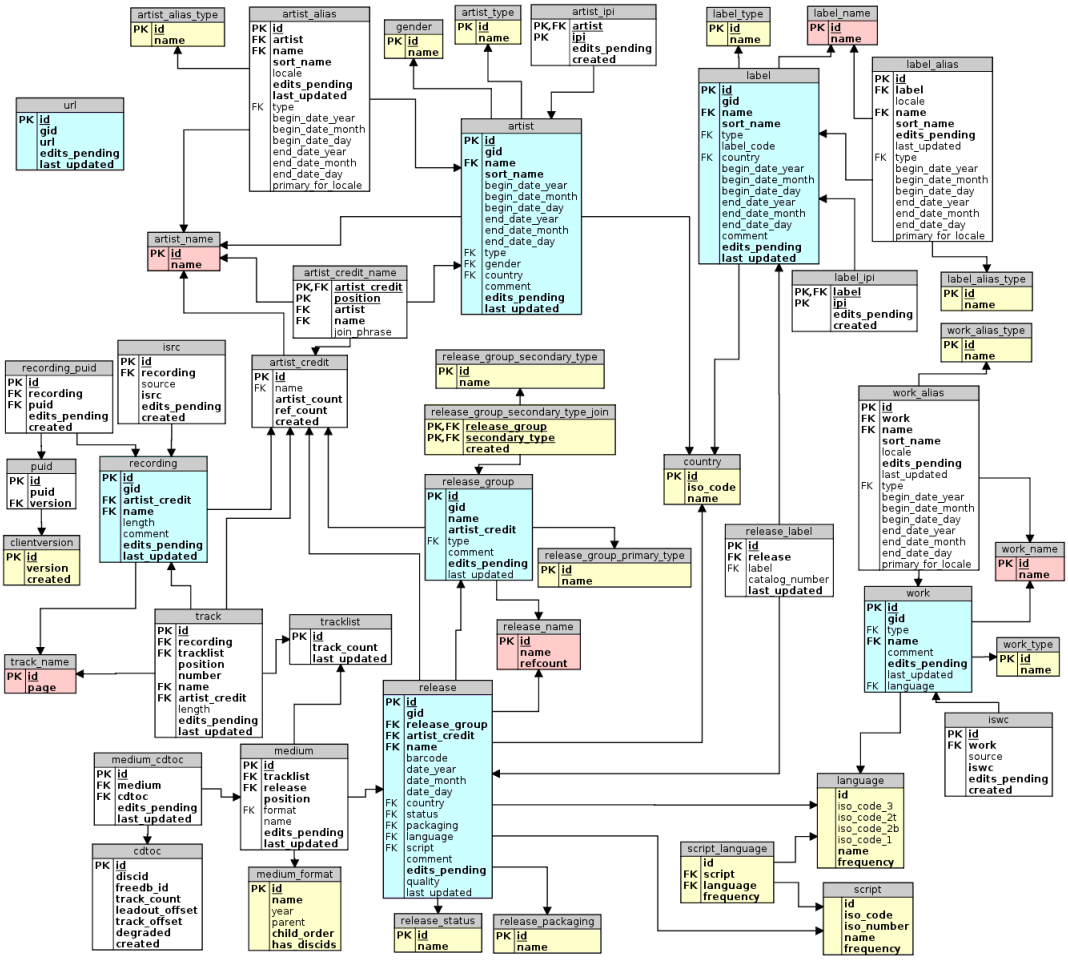
https://neo4j.com/docs/cypher-refcard/current/

6.2 Dataset

To conduct the experiments we need a Dataset. The largest dataset for Neo4j ready-to-use with relevant data is The Musicbrainz DB (15 GiB).
Reference: https://neo4j.com/developer/example-data/
musicbrainz_21.zip (4.5GB)
MusicBrainz is an open content music database, captures information about artists, their recorded works, and the relationships between them.
The MusicBrainz server is written in Perl with a PostgreSQL backend. It's actually open-source. The table schema is on github: https://github.com/metabrainz/musicbrainz-server/blob/master/admin/sql/CreateTables.sql

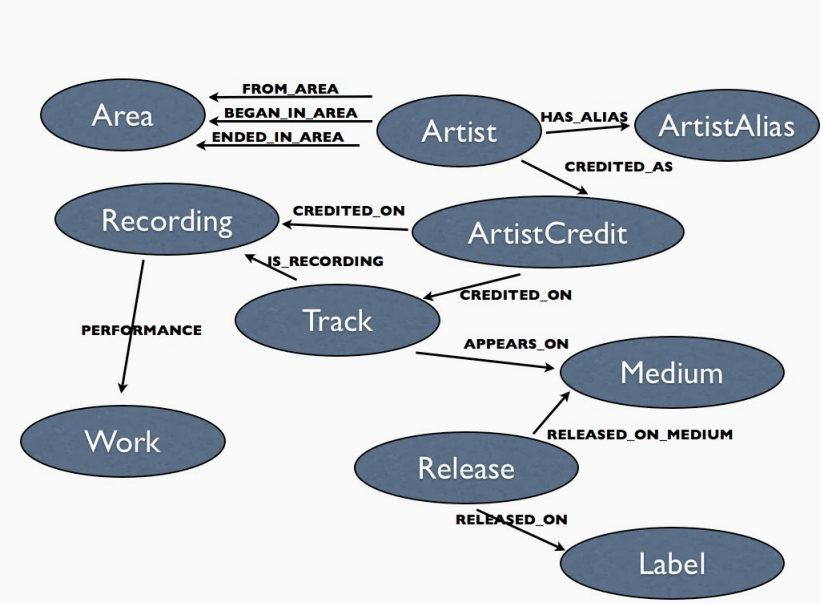
6.2.1 MusicBrainz's relational model

https://musicbrainz.org/doc/MusicBrainz_Database/Schema



6.2.2 MusicBrainz's graph model

The entities in our graph will be assigned Neo4j Labels that are roughly corresponding to the SQL table names (the names in the bubbles).



In the appendix (A) we can see what is related and how. It's calculation required 32 minutes.

```
// What is related, and how
MATCH (a)-[r]->(b)
WHERE labels(a) <> [] AND labels(b) <> []
RETURN DISTINCT head(labels(a)) AS This, type(r) as To, head(labels(b)) AS That
```

6.3 Hardware

The experiments are running on a MacBookPro11,3.
CPU: Core i7 (I7-4850HQ)
RAM: 16 GB, 1600 MHz DDR3L SDRAM
HD: 512 GB SSD PCIe

6.4 Neo4j Memory Management

The underlying engine of Neo4j keeps data stored as nodes, and relationships, and runs some of its operations in-memory, after their first execution.
The first time that we run a query, Neo4j it's inevitably slow because it needs to access the physical storage to obtain the data.
Ideally, MoK is made to start and never go offline.
For this reason the first launch of a query is not counted.
Reference: http://neo4j.com/docs/operations-manual/current/performance/

6.5 Experiments: Introduction

As we sad before, we want to see how Neo4j, and in general a graph database, perform on query that involves search of similar elements.
We now identify, using the MusicBrainz dataset, a list of potential interesting query for MoK.

- British artists signed on American record labels
- UK Bands with USA members
- Artist that have collaborated with parent
- Retrieve all artists

Limitation: The graph visualization tool integrated in Neo4j has bad performance if we display more than 300 nodes (on the Hardware 6.3).
The visualizer request JSON data for all or part of the graph data then dynamically creates an in-memory JavaScript visualization on the client side.

6.6 Benchmark: How to

Neo4j provides a REST API to interact with the db. The REST API return a JSON. A simple Node.js app was made for benchmark.

```
var username = "neo4j",
    password = "1234",
    request = require("request"),
    auth = "Basic " + new Buffer(username + ":" + password).toString("base64"),
    host = 'localhost',
    port = 7474,
    httpUrlForTransaction = 'http://' + host + ':' + port + '/db/data/cypher';

function runCypherQuery(query) {
    console.time("query");
    request.post({
            uri: httpUrlForTransaction,
            headers : {
                "Authorization" : auth
            },
            form: {query:query}
        },
        function (err, res, body) {
            console.timeEnd("query");
        })
}

runCypherQuery(
```

```
    'QUERY'
);
```

Even if we don't directly print the query result, the system save the result in memory so it invalidate the tests. We want to measure the performance of queries only about getting results intended as usable by the system. We are not interested in printing the results in a human readable version or JSON or any other representation. Moreover also the network delay is not counted.
To avoid this the call to the rest API was made with curl. The authorization header was hard coded.

```
curl -H accept:application/json -H content-type:application/json \
    -d '{"statements":[{"statement":"QUERY"}]}' \
    http://localhost:7474/db/data/transaction/commit \
    --header 'authorization: Basic bmVvNGo6MTIzNA==' \
    -s -w "%{time_total}\n" \
    -o /dev/null
```

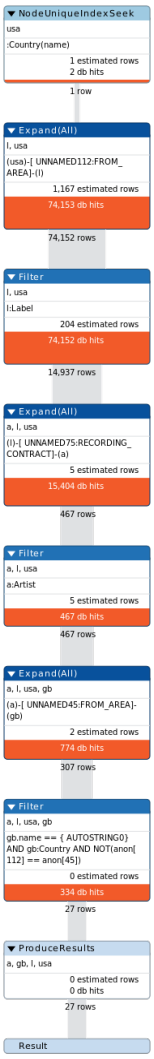
6.7 Experiments: Benchmark and test

British artists signed on American record labels

Query

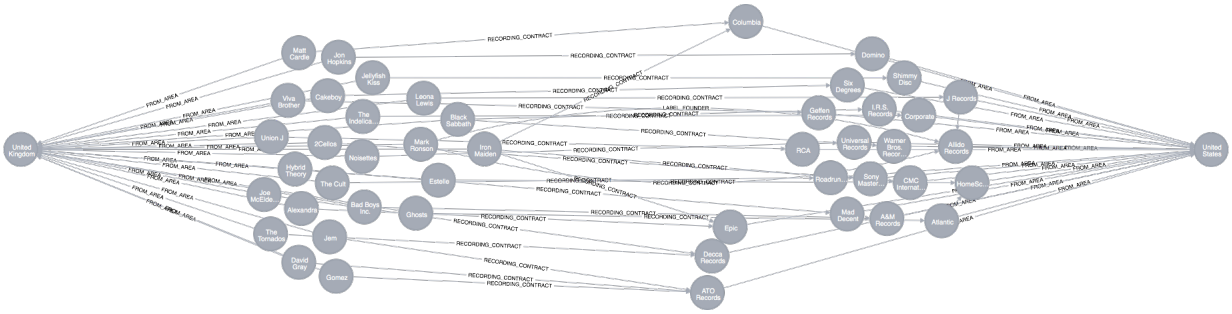
```
MATCH (gb:Country),(usa:Country),
(a:Artist)-[:FROM_AREA]-(gb),
(a:Artist)-[:RECORDING_CONTRACT]-(l:Label),
(l)-[:FROM_AREA]-(usa)
WHERE (gb.name = "United Kingdom") AND (usa.name = "United States")
RETURN a,l,usa,gb
```

Query plan



1. Select USA (2 db hits, 1 row returned)
2. Select recording contracts from USA, cartesian product (74153 db hits, 74152 nodes returned)
3. Filter recording contracts, select only the ones from USA (74152 db hits, 14937 nodes returned)
4. Get artists (15404 db hits, 467 nodes returned)
5. Filter to select the correct artists (467 db hits, 467 nodes returned)
6. Get artist from GB (774 db hits, 307 nodes returned)
7. Filter artist from GB (334 db hits, 27 nodes returned)
8. Produce result

Graph



Benchmark

#	Time (ms)
1	65
2	66
3	67
4	67
5	67
Average	66,4

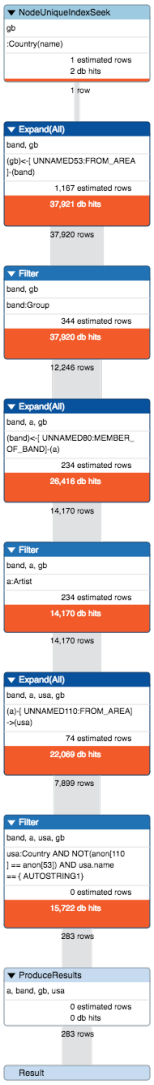
Total db hits: 165286

UK Bands with USA members

Query

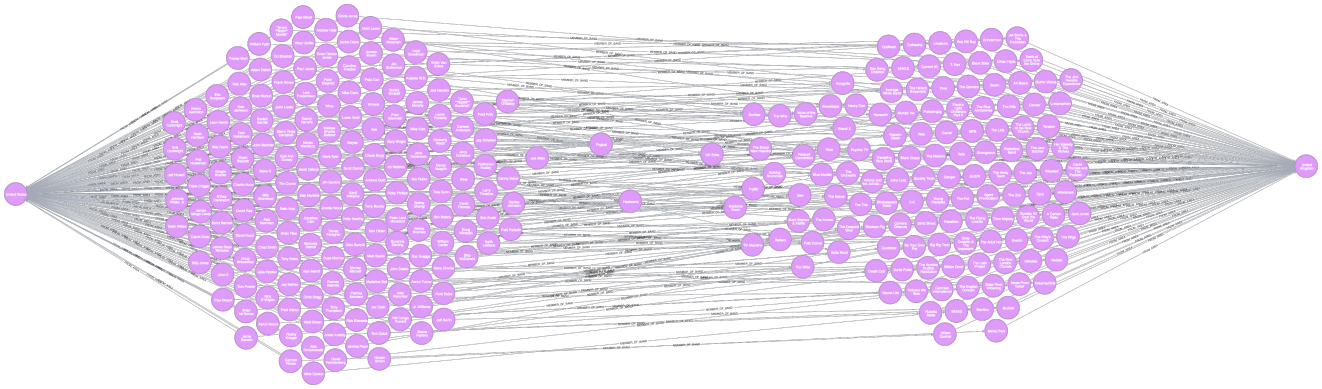
```
MATCH
(gb:Country),
(usa:Country),
(gb)<-[:FROM_AREA]-(band:Group)<-[:MEMBER_OF_BAND]-(a:Artist)-[:FROM_AREA]->(usa)
WHERE
(gb.name = "United Kingdom") AND (usa.name = "United States")
RETURN
usa,gb, a, band
```

Query plan



1. Select GB (2 db hits, 1 row returned)
2. Select bands from GB (37921 db hits, 32920 nodes returned)
3. Filter bands, bands are Group (37920 db hits, 12246 nodes returned)
4. Get member of bands (26416 db hits, 14170 nodes returned)
5. Filter artist (14170 db hits, 14170 nodes returned)
6. Get artist from USA (22069 db hits, 7899 nodes returned)
7. Filter artist from USA (15722 db hits, 283 nodes returned)
8. Produce result

Graph



Benchmark

#	Time (ms)
1	80
2	71
3	73
4	78
5	75
Average	75,4

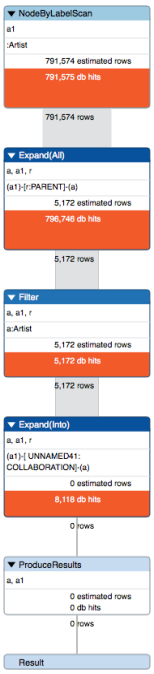
Total db Hits: 154220

Artist that have collaborated with parent

Query

```
MATCH (a:Artist)-[r:PARENT]-(a1:Artist)-[:COLLABORATION]-(a)
RETURN a, a1
```

Query plan



1. Select artists (791575 db hits, 791574 row returned)
2. Select only artists who has a parent relation (796746 db hits, 5172 nodes returned)
3. Filter artists (remove nodes that are not ARTIST) (5172 db hits, 5172 nodes returned)
4. Get the artists tha have collaborated with parent (8118 db hits, 0 nodes returned)
5. Produce result

Result

Total db hits: 1601611 No one have collaborated with parent.

Benchmark

#	Time (ms)
1	592
2	566
3	564
4	576
5	601
Average	579,8

Retrieve all artists

Query

```
MATCH (a:Artist) RETURN a
```

Benchmark

#	Time (ms)
1	2739
2	2713
3	2722
4	2745
5	2754
Average	2734,6

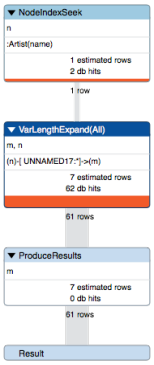
Total db hits: 791574

Given an artist get all the nodes that are at a certain distance

Query

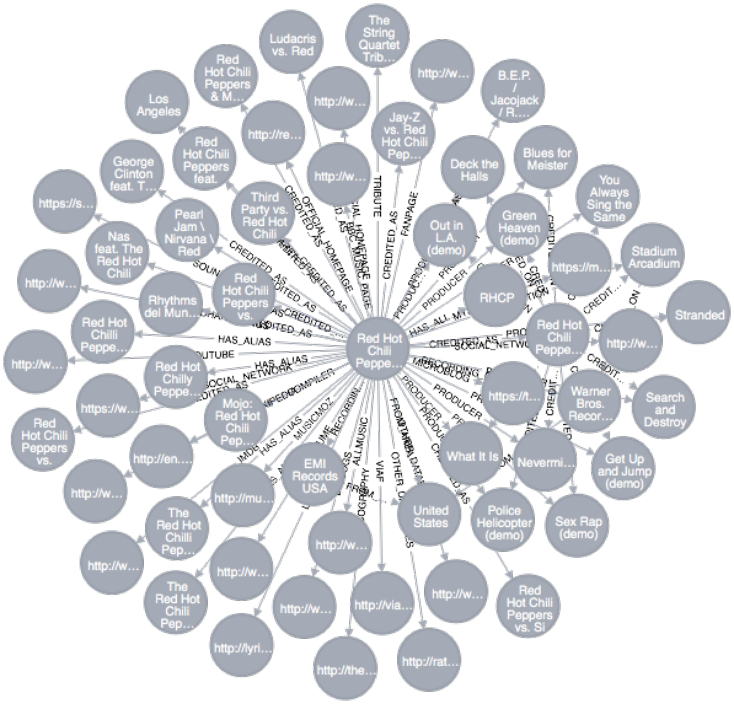
```
MATCH (n:Artist)-[*1]->(m) WHERE n.name = "Red Hot Chili Peppers" RETURN m
MATCH (n:Artist)-[*2]->(m) WHERE n.name = "Red Hot Chili Peppers" RETURN m
MATCH (n:Artist)-[*3]->(m) WHERE n.name = "Red Hot Chili Peppers" RETURN m
MATCH (n:Artist)-[*4]->(m) WHERE n.name = "Red Hot Chili Peppers" RETURN m
MATCH (n:Artist)-[*5]->(m) WHERE n.name = "Red Hot Chili Peppers" RETURN m
MATCH (n:Artist)-[*6]->(m) WHERE n.name = "Red Hot Chili Peppers" RETURN m
MATCH (n:Artist)-[*7]->(m) WHERE n.name = "Red Hot Chili Peppers" RETURN m
```

Query plan



1. Get Red Hot Chili Peppers node (2 db hits, 1 row returned)
2. Get all the nodes that are at a certain distance (example for N=1, 62 db hits , 61 nodes returned)
3. Produce result

Graph (for N=1)


Red Hot Chili Peppe...
Ludacris vs. Red
The String Quartet Trib...
http://w...
B.E.P. / Jacojack / R....
Los Angeles
Red Hot Chili Peppers & M...
http://w...
http://re...
Jay-Z vs. Red Hot Chili Pep...
Deck the Halls
Blues for Meister
You Always Sing the Same
George Clinton feat. T...
Red Hot Chili Peppers feat.
https://s...
Third Party vs. Red Hot Chili
Pearl Jam \ Nirvana \ Red
Out in L.A. (demo)
Green Heaven (demo)
https://m...
Stadium Arcadium
Nas feat. The Red Hot Chili
http://w...
Rhythms del Mun...
Red Hot Chili Peppers vs.
RHCP
Red Hot Chili Peppe...
Stranded
http://w...
Red Hot Chilli Peppe...
http://w...
Red Hot Chilly Peppe...
https://w...
Mojo: Red Hot Chili Pep...
https://t...
Warner Bros. Recor...
Search and Destroy
Red Hot Chili Peppers vs.
http://en...
What It Is
Nevermi...
Get Up and Jump (demo)
http://w...
http://mu...
EMI Records USA
United States
Police Helicopter (demo)
Sex Rap (demo)
The Red Hot Chilli Pep...
http://w...
http://w...
http://w...
http://w...
http://via...
http://w...
Red Hot Chili Peppers vs. Si
The Red Hot Chili Pep...
http://lyri...
http://the...
http://rat...

Benchmark

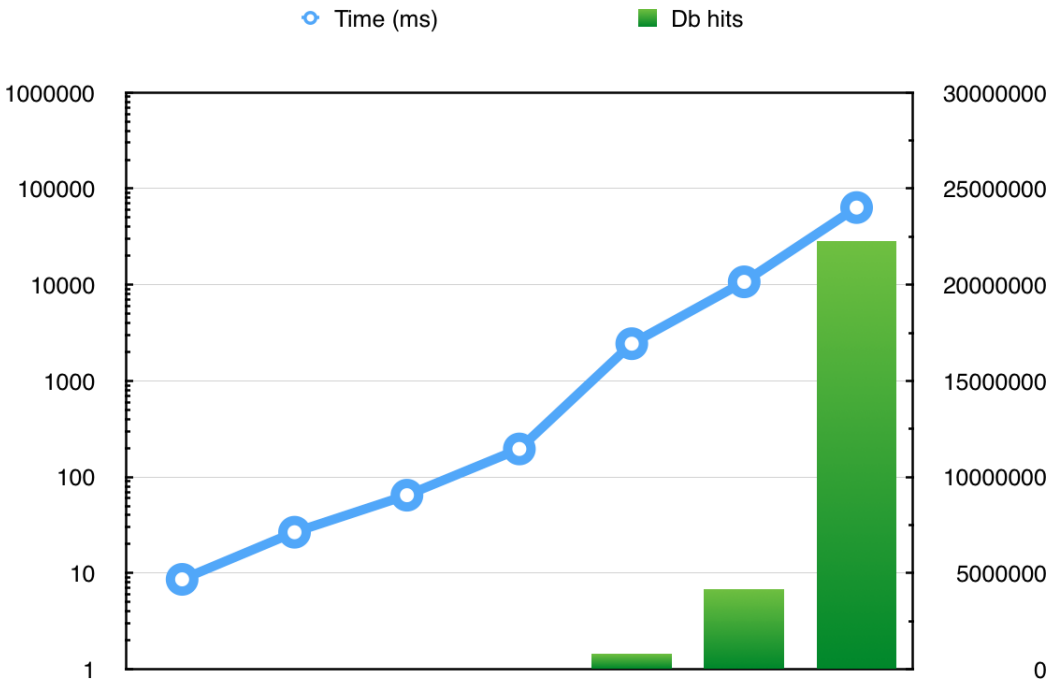
Distance	#	Time (ms)
1	1	9
1	2	8
1	3	9
1	4	9
1	5	8
1	Average	8,6
2	1	27
2	2	26
2	3	28
2	4	27
2	5	25
2	Average	26,6
3	1	63
3	2	66
3	3	64
3	4	66
3	5	64
3	Average	64,6
4	1	197
4	2	193
4	3	198
4	4	194
4	5	196
4	Average	195,6
5	1	2440
5	2	2437
5	3	2422
5	4	2428
5	5	2452
5	Average	2435,8
6	1	10717
6	2	10704
6	3	10725
6	4	10729
6	5	10720
6	Average	10719
7	1	63582
7	2	63538
7	3	63543
7	4	63512
7	5	63529
7	Average	63540.8

Distance	# Nodes
1	61
2	4695
3	15559
4	45722
5	690308
6	2657548
7	15446800

Distance	# Db hits
1	64
2	4820
3	25074
4	86357
5	822391
6	4170439
7	22275098

Is there a correlation between db hits and time?
In the graphs below we can see time and db hit plotted for the query:

```
MATCH (n:Artist)-[*N]->(m)WHERE n.name = "Red Hot Chili Peppers" RETURN m
```



Linear scale

We can clearly see that the time is a function of the number of hits and the trend is linear.

6.8 Experiments: Considerations

Every query result return also the corresponding graph. So for example if we are interested in what kind of relationship involves between nodes and the relative path for the query

```
MATCH (n:Artist)-[*N]->(m)WHERE n.name = "Red Hot Chili Peppers" RETURN m
```

We can get the information by adding to the return statement n (the Artists) and the relations.

```
MATCH (n:Artist)-[r *2]->(m)WHERE n.name = "Red Hot Chili Peppers" RETURN m,n,r
```

Example:

```
"graph": {
            "nodes": [
              {
                "id": "48854",
                "labels": [
                  "Artist",
                  "Group",
                  "Entry"
                ],
                "properties": {
                  "mbid": "8bfac288-ccc5-448d-9573-c33ea2aa5c30",
```

```
      "disambiguation": "\"\"",
      "name": "Red Hot Chili Peppers",
      "length": "0",
      "position": "0"
    }
  },
  {
    "id": "1705090",
    "labels": [
      "Label",
      "OriginalProduction",
      "Entry"
    ],
    "properties": {
      "mbid": "c595c289-47ce-4fba-b999-b87503e8cb71",
      "disambiguation": "\"\"",
      "name": "Warner Bros. Records",
      "length": "0",
      "position": "0"
    }
  },
  {
    "id": "16582214",
    "labels": [
      "Release",
      "Official",
      "JewelCase",
      "Entry"
    ],
    "properties": {
      "mbid": "e9e285e8-4855-3b1f-8cba-a5f44b82c215",
      "disambiguation": "\"\"",
      "name": "Queer",
      "length": "0",
      "position": "0"
    }
  }
],
"relationships": [
  {
    "id": "463479",
    "type": "RECORDING_CONTRACT",
    "startNode": "48854",
    "endNode": "1705090",
    "properties": {
      "month": "0",
      "year": "0",
      "position": "0",
      "day": "0"
    }
  },
  {
    "id": "35888090",
    "type": "MANUFACTURED",
    "startNode": "1705090",
    "endNode": "16582214",
    "properties": {
      "month": "0",
      "year": "0",
      "position": "0",
      "day": "0"
```

```
            }
        }
    ]
}
```

Appendix

A Relationship table

This	To	That
Area	PART_OF	Area
Area	HAS_ALIAS	AreaAlias
Area	RECORDED_IN	Recording
Area	WIKIDATA	Url
Area	WIKIPEDIA	Url
Area	ANTHEM	Work
Artist	BEGAN_IN_AREA	Area
Artist	ENDED_IN_AREA	Area
Artist	FROM_AREA	Area
Artist	CATALOGUED	Artist
Artist	COLLABORATION	Artist
Artist	CONDUCTOR_POSITION	Artist
Artist	INSTRUMENTAL_SUPPORTING_MUSICIAN	Artist
Artist	INVOLVED_WITH	Artist
Artist	IS_PERSON	Artist
Artist	MARRIED	Artist
Artist	MEMBER_OF_BAND	Artist
Artist	PARENT	Artist
Artist	SIBLING	Artist
Artist	SUPPORTING_MUSICIAN	Artist
Artist	VOCAL_SUPPORTING_MUSICIAN	Artist
Artist	VOICE_ACTOR	Artist
Artist	HAS_ALIAS	ArtistAlias
Artist	CREDITED_AS	ArtistCredit
Artist	CREATIVE_POSITION	Label
Artist	ENGINEER_POSITION	Label
Artist	LABEL_FOUNDER	Label
Artist	PRODUCER_POSITION	Label
Artist	RECORDING_CONTRACT	Label
Artist	ARRANGER	Other
Artist	ART_DIRECTION	Other
Artist	ARTISTS_AND_REPERTOIRE	Other
Artist	AUDIO	Other
Artist	BOOKING	Other
Artist	CHORUS_MASTER	Other
Artist	COMPILER	Other
Artist	COMPOSER	Other
Artist	CONDUCTOR	Other
Artist	CREATIVE_DIRECTION	Other
Artist	DESIGN/ILLUSTRATION	Other
Artist	EDITOR	Other
Artist	ENGINEER	Other
Artist	GRAPHIC_DESIGN	Other
Artist	HELD_A_CONCERT_AT	Other
Artist	INSTRUMENT	Other
Artist	INSTRUMENT_ARRANGER	Other
Artist	LEGAL_REPRESENTATION	Other
Artist	LIBRETTIST	Other
Artist	LINER_NOTES	Other
Artist	LYRICIST	Other
Artist	MASTERING	Other
Artist	MISC	Other
Artist	MIX	Other

Continued...

This	To	That
Artist	MIX-DJ	Other
Artist	ORCHESTRATOR	Other
Artist	PERFORMER	Other
Artist	PERFORMING_ORCHESTRA	Other
Artist	PHOTOGRAPHY	Other
Artist	PRODUCER	Other
Artist	PROGRAMMING	Other
Artist	PUBLISHING	Other
Artist	RECORDING	Other
Artist	SOUND	Other
Artist	TRIBUTE	Other
Artist	VOCAL	Other
Artist	WRITER	Other
Artist	ENGINEER_POSITION	Place
Artist	HELD_A_CONCERT_AT	Place
Artist	MASTERING_ENGINEER_POSITION	Place
Artist	MIXING_ENGINEER_POSITION	Place
Artist	RECORDING_ENGINEER_POSITION	Place
Artist	ARRANGER	Recording
Artist	ART_DIRECTION	Recording
Artist	ARTISTS_AND_REPERTOIRE	Recording
Artist	AUDIO	Recording
Artist	BOOKING	Recording
Artist	CHORUS_MASTER	Recording
Artist	COMPILER	Recording
Artist	CONDUCTOR	Recording
Artist	CREATIVE_DIRECTION	Recording
Artist	DESIGN/ILLUSTRATION	Recording
Artist	EDITOR	Recording
Artist	ENGINEER	Recording
Artist	GRAPHIC_DESIGN	Recording
Artist	INSTRUMENT	Recording
Artist	INSTRUMENT_ARRANGER	Recording
Artist	INSTRUMENTATOR	Recording
Artist	LEGAL_REPRESENTATION	Recording
Artist	LINER_NOTES	Recording
Artist	MASTERING	Recording
Artist	MISC	Recording
Artist	MIX	Recording
Artist	MIX-DJ	Recording
Artist	ORCHESTRATOR	Recording
Artist	PERFORMER	Recording
Artist	PERFORMING_ORCHESTRA	Recording
Artist	PHOTOGRAPHY	Recording
Artist	PRODUCER	Recording
Artist	PROGRAMMING	Recording
Artist	PUBLISHING	Recording
Artist	RECORDING	Recording
Artist	REMIXER	Recording
Artist	SAMPLES_FROM_ARTIST	Recording
Artist	SOUND	Recording
Artist	VOCAL	Recording
Artist	VOCAL_ARRANGER	Recording
Artist	ARRANGER	Release
Artist	ART_DIRECTION	Release
Artist	AUDIO	Release
Artist	BOOKING	Release
Artist	CHORUS_MASTER	Release
Artist	COMPILER	Release

Continued...

This	To	That
Artist	COMPOSER	Release
Artist	CONDUCTOR	Release
Artist	DESIGN/ILLUSTRATION	Release
Artist	EDITOR	Release
Artist	ENGINEER	Release
Artist	GRAPHIC_DESIGN	Release
Artist	INSTRUMENT	Release
Artist	INSTRUMENT_ARRANGER	Release
Artist	INSTRUMENTATOR	Release
Artist	LEGAL_REPRESENTATION	Release
Artist	LIBRETTIST	Release
Artist	LINER_NOTES	Release
Artist	LYRICIST	Release
Artist	MASTERING	Release
Artist	MISC	Release
Artist	MIX	Release
Artist	MIX-DJ	Release
Artist	ORCHESTRATOR	Release
Artist	PERFORMER	Release
Artist	PERFORMING_ORCHESTRA	Release
Artist	PHOTOGRAPHY	Release
Artist	PRODUCER	Release
Artist	PROGRAMMING	Release
Artist	PUBLISHING	Release
Artist	RECORDING	Release
Artist	REMIXER	Release
Artist	SAMPLES_FROM_ARTIST	Release
Artist	SOUND	Release
Artist	VOCAL	Release
Artist	VOCAL_ARRANGER	Release
Artist	WRITER	Release
Artist	ARTISTS_AND_REPERTOIRE	ReleaseGroup
Artist	CREATIVE_DIRECTION	ReleaseGroup
Artist	TRAVEL	ReleaseGroup
Artist	TRIBUTE	ReleaseGroup
Artist	ALLMUSIC	Url
Artist	BBC_MUSIC_PAGE	Url
Artist	BIOGRAPHY	Url
Artist	BLOG	Url
Artist	DISCOGRAPHY	Url
Artist	DISCOGS	Url
Artist	DOWNLOAD_FOR_FREE	Url
Artist	FANPAGE	Url
Artist	IMAGE	Url
Artist	IMDB	Url
Artist	LYRICS	Url
Artist	MICROBLOG	Url
Artist	MUSICMOZ	Url
Artist	MYSPACE	Url
Artist	OFFICIAL_HOMEPAGE	Url
Artist	ONLINE_COMMUNITY	Url
Artist	OTHER_DATABASES	Url
Artist	PURCHASE_FOR_DOWNLOAD	Url
Artist	PURCHASE_FOR_MAIL-ORDER	Url
Artist	PUREVOLUME	Url
Artist	SECONDHANDSONGS	Url
Artist	SOCIAL_NETWORK	Url
Artist	SOUNDCLOUD	Url
Artist	STREAMING_MUSIC	Url

Continued...

This	To	That
Artist	VGMDB	Url
Artist	VIAF	Url
Artist	VIDEO_CHANNEL	Url
Artist	WIKIDATA	Url
Artist	WIKIPEDIA	Url
Artist	YOUTUBE	Url
Artist	ARRANGER	Work
Artist	COMPOSER	Work
Artist	INSTRUMENT_ARRANGER	Work
Artist	INSTRUMENTATOR	Work
Artist	LIBRETTIST	Work
Artist	LYRICIST	Work
Artist	MISC	Work
Artist	ORCHESTRATOR	Work
Artist	PUBLISHING	Work
Artist	VOCAL_ARRANGER	Work
Artist	WRITER	Work
ArtistCredit	CREDITED_ON	Other
ArtistCredit	CREDITED_ON	Recording
ArtistCredit	CREDITED_ON	Release
ArtistCredit	CREDITED_ON	ReleaseGroup
ArtistCredit	CREDITED_ON	Track
Label	FROM_AREA	Area
Label	LABEL_DISTRIBUTION	Label
Label	LABEL_OWNERSHIP	Label
Label	LABEL_REISSUE	Label
Label	LABEL_RENAME	Label
Label	DISTRIBUTED	Other
Label	MANUFACTURED	Other
Label	PROMOTED	Other
Label	PUBLISHING	Other
Label	RIGHTS_SOCIETY	Other
Label	PUBLISHING	Recording
Label	DISTRIBUTED	Release
Label	MANUFACTURED	Release
Label	PROMOTED	Release
Label	PUBLISHED	Release
Label	PUBLISHING	Release
Label	RIGHTS_SOCIETY	Release
Label	BLOG	Url
Label	CATALOG_SITE	Url
Label	DISCOGS	Url
Label	FANPAGE	Url
Label	HISTORY_SITE	Url
Label	IMDB	Url
Label	LOGO	Url
Label	MICROBLOG	Url
Label	MYSPACE	Url
Label	OFFICIAL_SITE	Url
Label	OTHER_DATABASES	Url
Label	SOCIAL_NETWORK	Url
Label	SOUNDCLOUD	Url
Label	VGMDB	Url
Label	VIAF	Url
Label	VIDEO_CHANNEL	Url
Label	WIKIDATA	Url
Label	WIKIPEDIA	Url
Label	YOUTUBE	Url
Label	PUBLISHING	Work

Continued...

This	To	That
Other	RELEASED_IN	Area
Other	RELEASED_ON	Label
Other	RELEASED_ON_MEDIUM	Medium
Other	COVER	Other
Other	PART_OF	Other
Other	REMASTER	Other
Other	TRANSL-TRACKLISTING	Other
Other	HAS_ALIAS	PlaceAlias
Other	RECORDED_AT	Recording
Other	PART_OF_SET	Release
Other	REMASTER	Release
Other	TRANSL-TRACKLISTING	Release
Other	PART_OF	ReleaseGroup
Other	SINGLE_FROM	ReleaseGroup
Other	ALLMUSIC	Url
Other	AMAZON_ASIN	Url
Other	COVER_ART_LINK	Url
Other	DISCOGRAPHY_ENTRY	Url
Other	DISCOGS	Url
Other	DOWNLOAD_FOR_FREE	Url
Other	IMAGE	Url
Other	IMDB	Url
Other	IMDB_SAMPLES	Url
Other	LICENSE	Url
Other	LYRICS	Url
Other	OFFICIAL_HOMEPAGE	Url
Other	OTHER_DATABASES	Url
Other	PURCHASE_FOR_DOWNLOAD	Url
Other	PURCHASE_FOR_MAIL-ORDER	Url
Other	REVIEW	Url
Other	SECONDHANDSONGS	Url
Other	SOCIAL_NETWORK	Url
Other	STREAMING_MUSIC	Url
Other	VGMDB	Url
Other	WIKIDATA	Url
Other	WIKIPEDIA	Url
Place	HAS_ALIAS	PlaceAlias
Place	MIXED_AT	Recording
Place	RECORDED_AT	Recording
Place	MASTERED_AT	Release
Place	MIXED_AT	Release
Place	RECORDED_AT	Release
Place	BLOG	Url
Place	DISCOGS	Url
Place	IMAGE	Url
Place	IMDB	Url
Place	MYSPACE	Url
Place	OFFICIAL_HOMEPAGE	Url
Place	OTHER_DATABASES	Url
Place	SOCIAL_NETWORK	Url
Place	VIDEO_CHANNEL	Url
Place	WIKIDATA	Url
Place	WIKIPEDIA	Url
Place	YOUTUBE	Url
Recording	COMPILATION	Recording
Recording	DJ-MIX	Recording
Recording	EDIT	Recording
Recording	FIRST_TRACK_RELEASE	Recording
Recording	KARAOKE	Recording

Continued...

This	To	That
Recording	MASHES_UP	Recording
Recording	PERFORMANCE	Recording
Recording	REMASTER	Recording
Recording	REMIX	Recording
Recording	SAMPLES_MATERIAL	Recording
Recording	SAMPLES_MATERIAL	Release
Recording	ALLMUSIC	Url
Recording	ARTISTS_AND_REPERTOIRE	Url
Recording	CREATIVE_DIRECTION	Url
Recording	DESIGN/ILLUSTRATION	Url
Recording	DOWNLOAD_FOR_FREE	Url
Recording	GRAPHIC_DESIGN	Url
Recording	IMDB_SAMPLES	Url
Recording	LICENSE	Url
Recording	OTHER_DATABASES	Url
Recording	PUBLISHING	Url
Recording	PURCHASE_FOR_DOWNLOAD	Url
Recording	RECORDING_STUDIO	Url
Recording	STREAMING_MUSIC	Url
Recording	MEDLEY	Work
Recording	PERFORMANCE	Work
Release	RELEASED_IN	Area
Release	RELEASED_ON	Label
Release	RELEASED_ON_MEDIUM	Medium
Release	PART_OF	Other
Release	PART_OF_SET	Other
Release	REMASTER	Other
Release	TRANSL-TRACKLISTING	Other
Release	PART_OF_SET	Release
Release	REMASTER	Release
Release	SUPPORTING_RELEASE	Release
Release	TRANSL-TRACKLISTING	Release
Release	PART_OF	ReleaseGroup
Release	AMAZON_ASIN	Url
Release	COVER_ART_LINK	Url
Release	DISCOGRAPHY_ENTRY	Url
Release	DISCOGS	Url
Release	DOWNLOAD_FOR_FREE	Url
Release	IMDB_SAMPLES	Url
Release	LICENSE	Url
Release	OTHER_DATABASES	Url
Release	PURCHASE_FOR_DOWNLOAD	Url
Release	PURCHASE_FOR_MAIL-ORDER	Url
Release	SECONDHANDSONGS	Url
Release	STREAMING_MUSIC	Url
Release	VGMDB	Url
ReleaseGroup	REMIX	Other
ReleaseGroup	SINGLE_FROM	Other
ReleaseGroup	COVER	ReleaseGroup
ReleaseGroup	DJ-MIX	ReleaseGroup
ReleaseGroup	LIVE_PERFORMANCE	ReleaseGroup
ReleaseGroup	MASHES_UP	ReleaseGroup
ReleaseGroup	REMIX	ReleaseGroup
ReleaseGroup	SINGLE_FROM	ReleaseGroup
ReleaseGroup	ALLMUSIC	Url
ReleaseGroup	DISCOGS	Url
ReleaseGroup	IMDB	Url
ReleaseGroup	LYRICS	Url
ReleaseGroup	MUSICMOZ	Url

Continued...

This	To	That
ReleaseGroup	OFFICIAL_HOMEPAGE	Url
ReleaseGroup	OTHER_DATABASES	Url
ReleaseGroup	RECORDING_STUDIO	Url
ReleaseGroup	REVIEW	Url
ReleaseGroup	WIKIDATA	Url
ReleaseGroup	WIKIPEDIA	Url
Track	APPEARS_ON	Medium
Track	IS_RECORDING	Recording
Url	ALLMUSIC	Work
Url	LYRICS	Work
Url	MISC	Work
Url	OTHER_DATABASES	Work
Url	PUBLISHING	Work
Url	SCORE	Work
Url	SECONDHANDSONGS	Work
Url	SONGFACTS	Work
Url	VIAF	Work
Url	WIKIDATA	Work
Url	WIKIPEDIA	Work
Work	ARRANGEMENT	Work
Work	BASED_ON	Work
Work	MEDLEY	Work
Work	ORCHESTRATION	Work
Work	OTHER_VERSION	Work
Work	PARTS	Work
Work	REVISION_OF	Work

Bibliography

Stefano Mariani. Molecules of knowledge: a novel perspective over knowledge management. 2012.

Wikipedia. Graph database, 2015. URL `https://en.wikipedia.org/wiki/Graph_database`.