Southampton Solent University

COMPUTING

SAD 600

Data Science

JAKE LAMBERT

STUDENT ID: Q10158341

Contents

[Question 1 3](#_Toc503254766)

[Question 2 3](#_Toc503254767)

[Key-value Stores 3](#_Toc503254768)

[Document Databases 4](#_Toc503254769)

[Column Family Stores 4](#_Toc503254770)

[Graph Databases 4](#_Toc503254771)

[Question 3 5](#_Toc503254772)

[Query 1 5](#_Toc503254773)

[Query 2 6](#_Toc503254774)

[Query 3 7](#_Toc503254775)

[Query 4 9](#_Toc503254776)

[Question 4 11](#_Toc503254777)

[Python script 1 11](#_Toc503254778)

[Python script 2 13](#_Toc503254779)

[Python script 3 15](#_Toc503254780)

[Python script 4 17](#_Toc503254781)

[XML Schemas 19](#_Toc503254782)

[Query 1 19](#_Toc503254783)

[Query 2 20](#_Toc503254784)

[Query 3 21](#_Toc503254785)

[Query 4 22](#_Toc503254786)

[Question 5 23](#_Toc503254787)

[Question 6 23](#_Toc503254788)

[Question 7 23](#_Toc503254789)

# Question 1

Name

Time 2 Vibe.

Purpose

The purpose of this music application lets you look back and discover old albums, EP’s, singles & remixes that falls into the deep house music genre from the last 17 years (2000’sq). You will be able to feed your ears with the evolution of the genre and enjoy the tempos, soft keyboard sounds and soulful vocals.

How people will use it

Users will be presented with an interface where they can decide what year they want to find and explore tracks from, this will give them recommended playlists like remixed classics, soulful vocals, deep beats etc. Also users will be presented with the most popular DJ’s from a particular country at a particular year, this popularity will be determined by the number of downloaded tracks in the calendar year.

Tables

* Artist
* Artist\_tag
* Tag
* Release
* Recording
* Track
* Artist\_credit
* Artist\_credit\_name
* Release\_group
* Release\_country

# Question 2

*I have included linked from where I derived this information from.*

## Key-value Stores

With the lack of development flexibility in MusicBrainz, key-value stores offer flexible data modelling. A key-value store does not enforce any structure on the data, to match the requirements of the application, it offers fantastic flexibility for data modelling.

*https://cynere.com/advantages-of-document-databases*

## Document Databases

Storing data as an independent document can enhance the performance and distribution across various servers to become easier.

As mobile, web and IOT based applications change the development of application data models, document removes the need to force-fit relational data models to support new types of application data models, this causes for flexible data modelling.

With a flexible schema, any number of fields can be added.

[*http://basho.com/resources/document-databases/*](http://basho.com/resources/document-databases/)

[*https://cynere.com/advantages-of-document-databases/*](https://cynere.com/advantages-of-document-databases/)

## Column Family Stores

As MusicBrainz is a large database, columnar databases have great scalability. They are well suited to MPP (massively parallel processing), which involves having large amounts of data spread out across many machines, sometimes thousands.

Columnar stores are very efficient when querying and analysing as they can be loaded extremely fast, a billion row tables can be loaded within a few seconds.

[*http://database.guide/what-is-a-column-store-database/*](http://database.guide/what-is-a-column-store-database/)

## Graph Databases

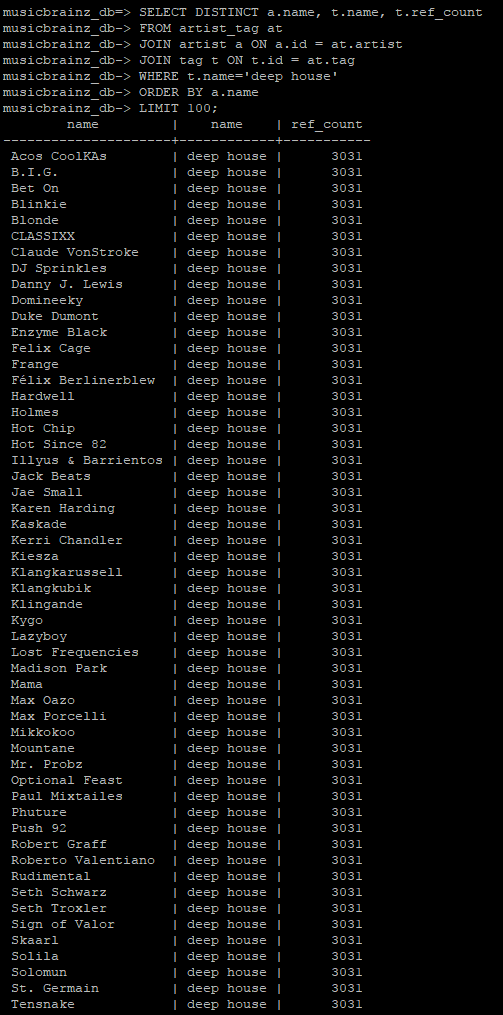
Graph models provides an advantage in handling data changes because the structure and schema is flexible when applications and industries change. So when new data needs to be included in a graph structure the schema will then update when it is written to the graph.

*https://neo4j.com/why-graph-databases/*

# Question 3

## Query 1

This query gathers all the artists to have produced deep house tracks. The artist\_tag is the link between artist and tag tables. I have linked the tag table and the artist table to link artist to the ‘deep house’ genre.



SELECT DISTINCT a.name, t.name, t.ref\_count

FROM artist\_tag at

JOIN artist a ON a.id = at.artist

JOIN tag t ON t.id = at.tag

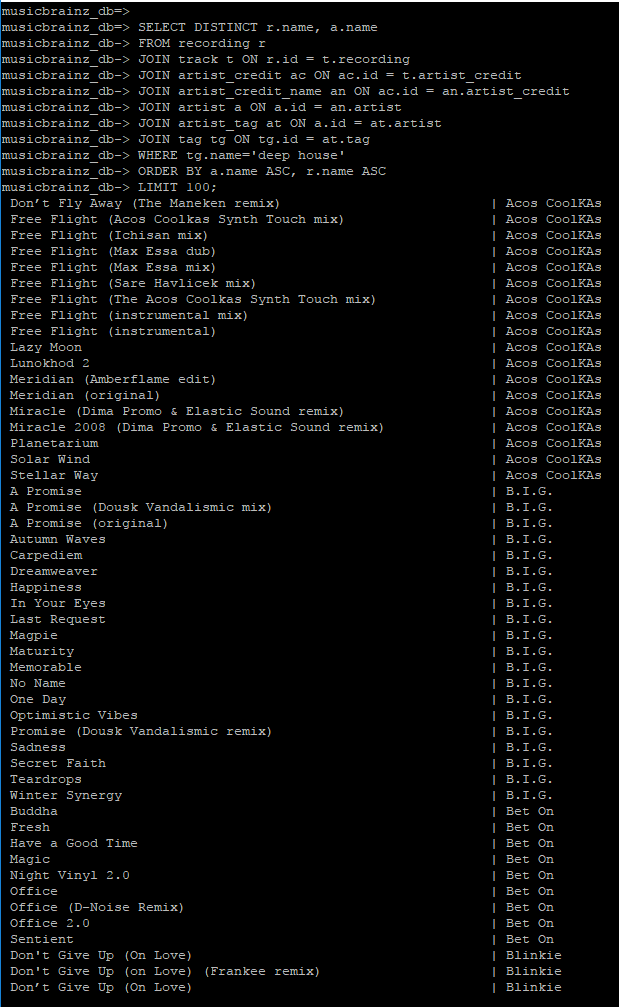
WHERE t.name='deep house'

ORDER BY a.name

LIMIT 100;

## Query 2

For this query, I queried all the unique mixes or edits that come under the Deep House genre. This will also show the original version of the song as well as the additional versions. Deep House music is always releasing remixes to songs, so for one song at times can have 8 remixes, for this reason I thought it would be important to use for this query.



SELECT DISTINCT r.name, a.name

FROM recording r

JOIN track t ON r.id = t.recording

JOIN artist\_credit ac ON ac.id = t.artist\_credit

JOIN artist\_credit\_name an ON ac.id = an.artist\_credit

JOIN artist a ON a.id = an.artist

JOIN artist\_tag at ON a.id = at.artist

JOIN tag tg ON tg.id = at.tag

WHERE tg.name='deep house'

ORDER BY a.name ASC, r.name ASC

LIMIT 100;

## Query 3

This query displays all releases from singles, albums & EP’s release between the years of 2000 and 2017. I have ordered them by the artist name so there is an equal mix of artists, releases and year spread out.

SELECT DISTINCT a.name, r.name, t.name, rc.date\_year

FROM artist a

JOIN artist\_credit\_name acn ON a.id = acn.artist

JOIN artist\_credit ac ON ac.id = acn.artist\_credit

JOIN track t ON ac.id = t.artist\_credit

JOIN release r ON r.artist\_credit = ac.id

JOIN release\_country rc ON r.id = rc.release

JOIN artist\_tag at ON a.id = at.artist

JOIN tag tg ON tg.id = at.tag

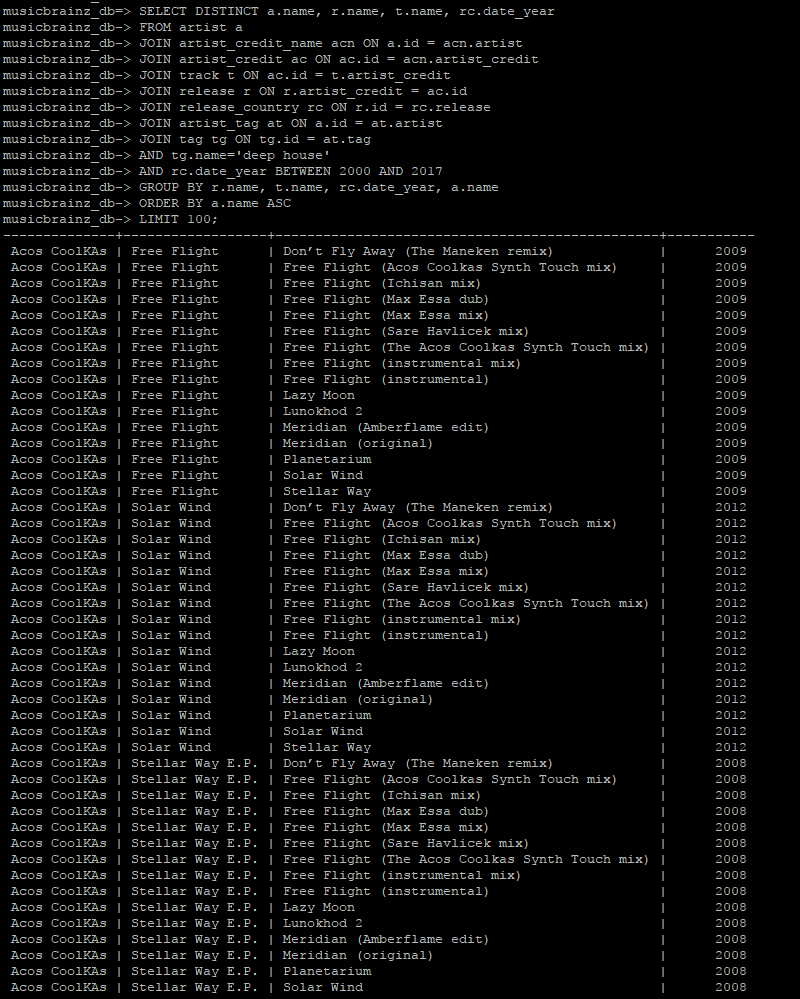
AND tg.name='deep house'

AND rc.date\_year BETWEEN 2000 AND 2017

GROUP BY r.name, t.name, rc.date\_year, a.name

ORDER BY a.name ASC

LIMIT 100;



## Query 4

This query displays all the releases of a particular artist under the Deep House genre. It retrieves the data of whether the release is either a single, album or EP. It displays the release between the years 2000 and 2017, the modern era. I have ordered the select statement by artist name, this is so within the 100 results there is a mixture of EP’s, albums & singles.

SELECT DISTINCT rp.name “Type”, a.name “Artist”, r.name “Name”, rc.date\_year “Year”

FROM artist a

JOIN artist\_credit\_name acn ON a.id = acn.artist

JOIN artist\_credit ac ON ac.id = acn.artist\_credit

JOIN track t ON ac.id = t.artist\_credit

JOIN release r ON r.artist\_credit = ac.id

JOIN release\_country rc ON r.id = rc.release

JOIN artist\_tag at ON a.id = at.artist

JOIN tag tg ON tg.id = at.tag

JOIN release\_group rg ON rg.id = r.release\_group

JOIN release\_group\_primary\_type rp ON rp.id = rg.type

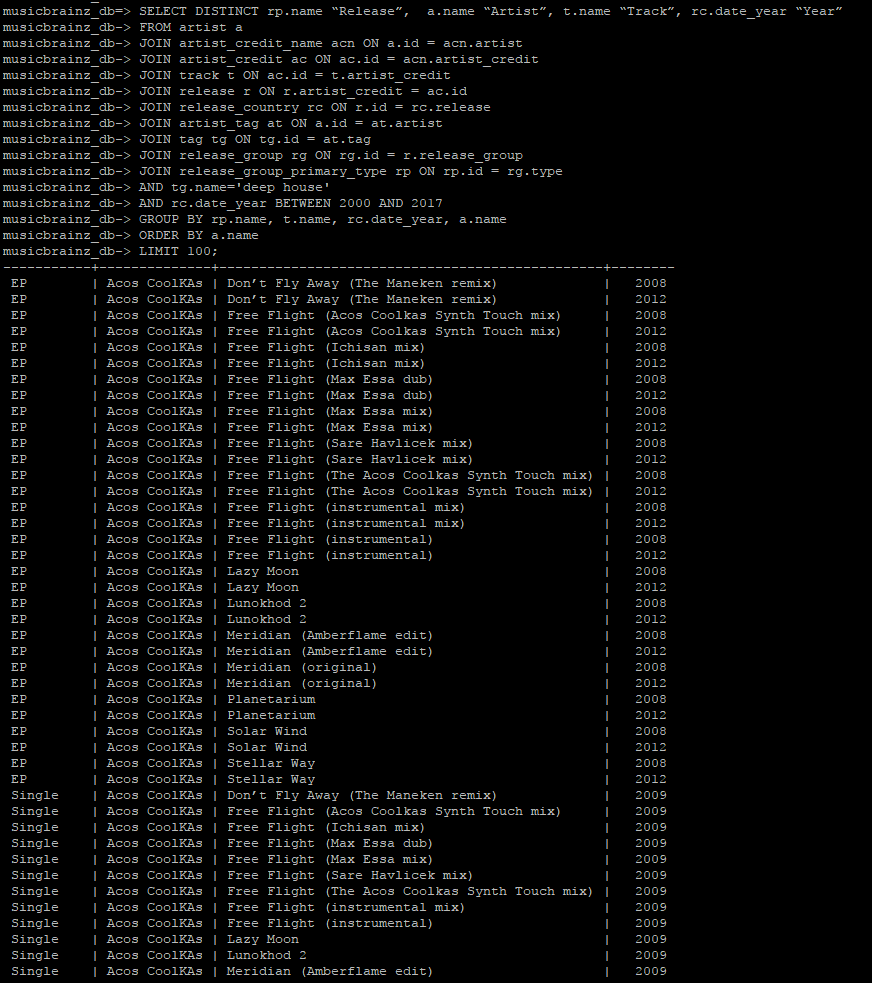
AND tg.name='deep house'

AND rc.date\_year BETWEEN 2000 AND 2017

GROUP BY rp.name, r.name, rc.date\_year, a.name

ORDER BY a.name

LIMIT 100;



# Question 4

## Python script 1

Here we have a python script that connects to the database Musicbrainz and outputs an xml document representing the data fetched from the database. In this example, the fetched data is finding the artists in the genre deep house. The artist name and genre are outputted in string format whereas the ref count is outputted as integer. These three tags are nested in artist collection tag.

I have put the python script into a function called artistGenre().

def artistGenreXML();

import psycopg2

connectstr = "dbname='musicbrainz\_db' user='test\_student' password='test\_student' host='localhost'"

conn = psycopg2.connect(connectstr)

cursor = conn.cursor()

cursor.execute("""SELECT DISTINCT a.name, t.name, t.ref\_count

FROM artist\_tag at

JOIN artist a ON a.id = at.artist

JOIN tag t ON t.id = at.tag

WHERE t.name='deep house'

ORDER BY a.name

LIMIT 100;

""")

rows = cursor.fetchall()

print"<?xml version='1.0' encoding='UTF-8'?>"

print"<artist\_collection>"

for row in rows:

print(" <artist>")

print(" <name>" + row[0] + "</name>")

print(" <genre>" + row[1] + "</genre>")

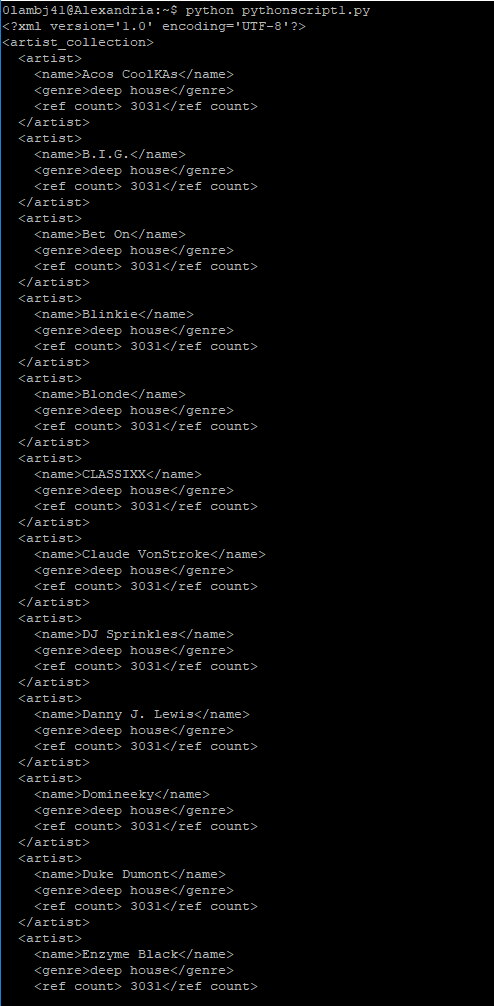
print(" <ref count> %i</ref count>" %row[2])

print(" </artist>")

print("</artist\_collection>")

artistGenreXML()

This is the python script being run in Alexandria.



## Python script 2

This python script shows the different versions of a particular song. The string formatting has the artist name and song title nested in the artist tag.

The python script is in a function called trackVersions().

def trackVersionsXML();

import psycopg2

connectstr = "dbname='musicbrainz\_db' user='test\_student'password='test\_student' host='localhost'"

conn = psycopg2.connect(connectstr)

cursor = conn.cursor()

cursor.execute("""SELECT DISTINCT r.name, a.name

FROM recording r

JOIN track t ON r.id = t.recording

JOIN artist\_credit ac ON ac.id = t.artist\_credit

JOIN artist\_credit\_name an ON ac.id = an.artist\_credit

JOIN artist a ON a.id = an.artist

JOIN artist\_tag at ON a.id = at.artist

JOIN tag tg ON tg.id = at.tag

WHERE tg.name='deep house'

ORDER BY a.name ASC, r.name ASC

LIMIT 100;

""")

rows = cursor.fetchall()

print"<?xml version='1.0' encoding='UTF-8'?>"

print"<remix\_collection>"

for row in rows:

print("<recording>")

print(" <track name>" + row[1] + "</track name>")

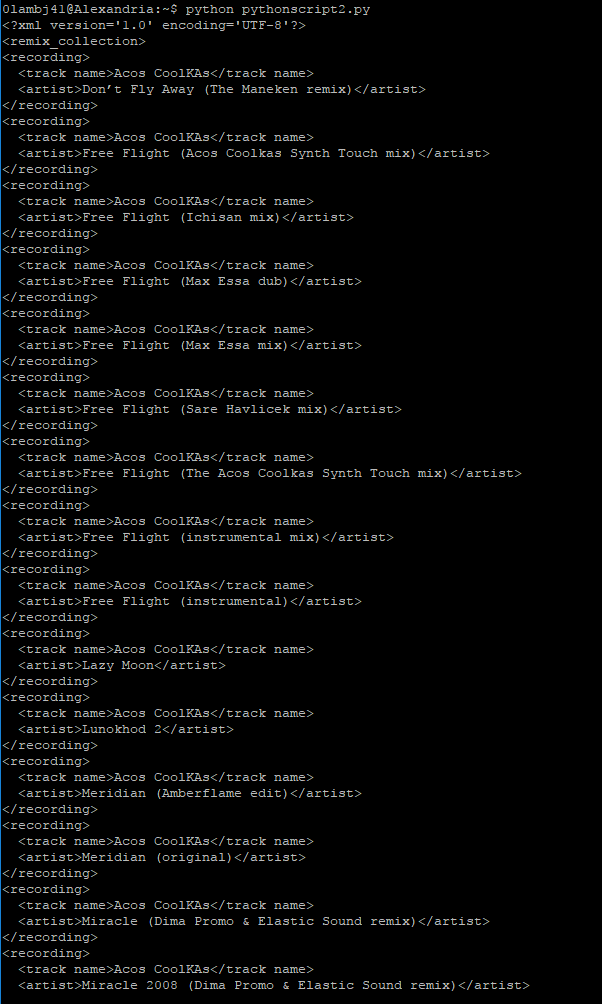
print(" <artist>" + row[0] + "</artist>")

print("</recording>")

print"</remix\_collection>"

trackVersionsXML()

This is the python script being run in Alexandria.



## Python script 3

This python script shows all the releases from the 2000’s. All the tags are nested under track so each individual line is a unique track.

The python script is in a function called releaseYear().

def releaseYearXML();

import psycopg2

connectstr = "dbname='musicbrainz\_db' user='test\_student'password='test\_student' host='localhost'"

conn = psycopg2.connect(connectstr)

cursor = conn.cursor()

cursor.execute("""SELECT DISTINCT a.name, r.name, t.name, rc.date\_year

FROM artist a

JOIN artist\_credit\_name acn ON a.id = acn.artist

JOIN artist\_credit ac ON ac.id = acn.artist\_credit

JOIN track t ON ac.id = t.artist\_credit

JOIN release r ON r.artist\_credit = ac.id

JOIN release\_country rc ON r.id = rc.release

JOIN artist\_tag at ON a.id = at.artist

JOIN tag tg ON tg.id = at.tag

AND tg.name='deep house'

AND rc.date\_year BETWEEN 2000 AND 2017

GROUP BY r.name, t.name, rc.date\_year, a.name

ORDER BY a.name ASC

LIMIT 100;

""")

rows = cursor.fetchall()

print"<?xml version='1.0' encoding='UTF-8'?>"

print"<music\_releases>"

for row in rows:

print("<track>")

print(" <artist>" + row[0] + "</artist>")

print(" <release>" + row[1] + "</release>")

print(" <title>" + row[2] + "</title>")

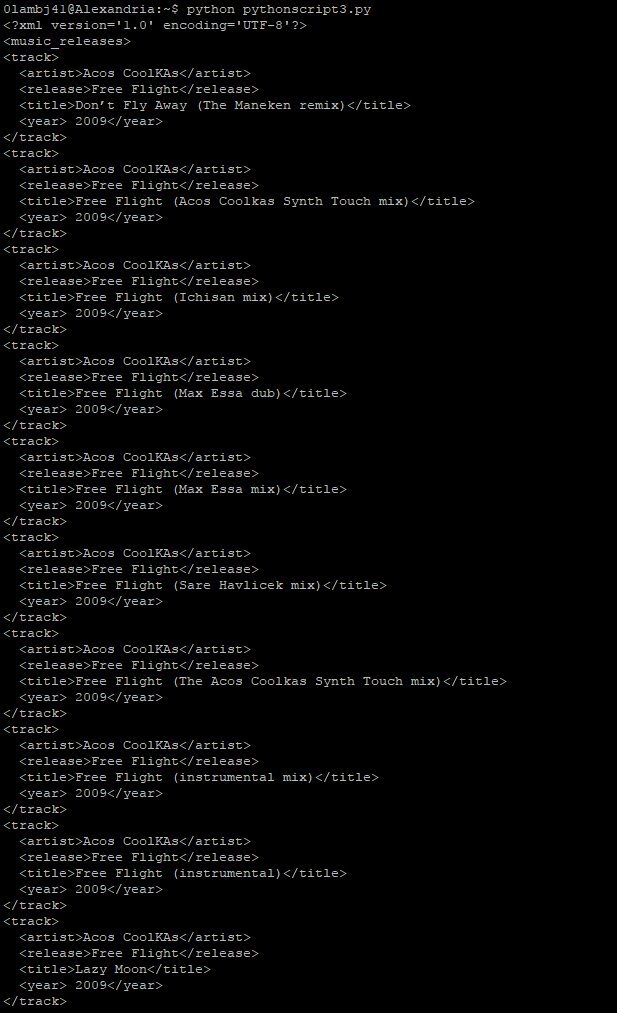
print(" <year> %i</year>" %row[3])

print("</track>")

print"</music\_releases>"

releaseYearXML()

This is the python script being run in Alexandria.



## Python script 4

This python script displays whether the release is an EP, single or album.

The python script is in a function called releaseYear().

def releaseTypeXML();

import psycopg2

connectstr = "dbname='musicbrainz\_db' user='test\_student'password='test\_student' host='localhost'"

conn = psycopg2.connect(connectstr)

cursor = conn.cursor()

cursor.execute("""SELECT DISTINCT rp.name, a.name , r.name , rc.date\_year

FROM artist a

JOIN artist\_credit\_name acn ON a.id = acn.artist

JOIN artist\_credit ac ON ac.id = acn.artist\_credit

JOIN track t ON ac.id = t.artist\_credit

JOIN release r ON r.artist\_credit = ac.id

JOIN release\_country rc ON r.id = rc.release

JOIN artist\_tag at ON a.id = at.artist

JOIN tag tg ON tg.id = at.tag

JOIN release\_group rg ON rg.id = r.release\_group

JOIN release\_group\_primary\_type rp ON rp.id = rg.type

AND tg.name='deep house'

AND rc.date\_year BETWEEN 2000 AND 2017

GROUP BY rp.name, r.name, rc.date\_year, a.name

ORDER BY a.name

LIMIT 100;

""")

rows = cursor.fetchall()

print"<?xml version='1.0' encoding='UTF-8'?>"

print"<release\_category>"

for row in rows:

print("<release>")

print(" <type>" + row[0] + "</type>")

print(" <artist>" + row[1] + "</artist>")

print(" <name>" + row[2] + "</name>")

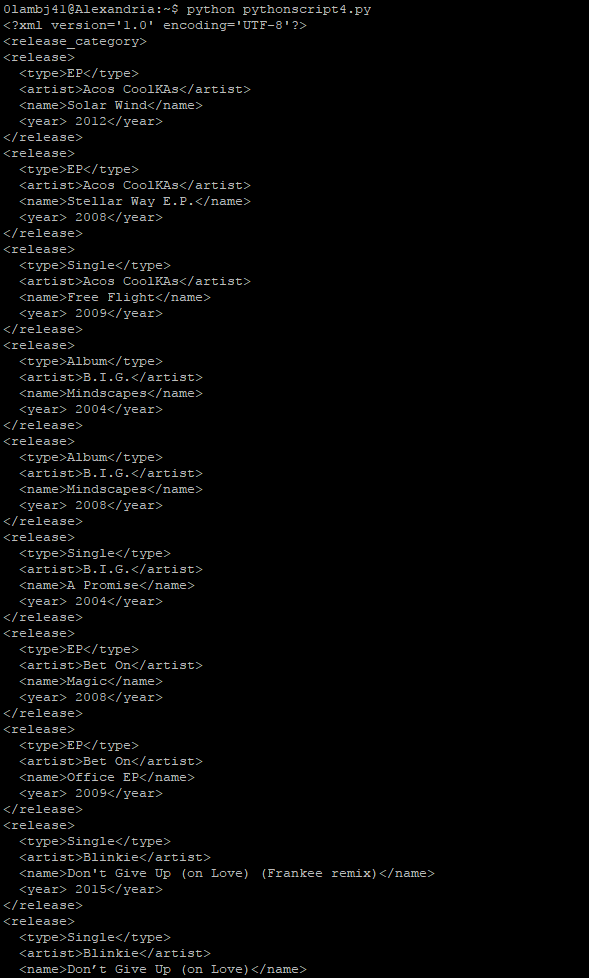
print(" <year> %i</year>" %row[3])

print("</release>")

print"</release\_category>"

releaseType()

This is the python script being run in Alexandria.



## XML Schemas

Here is the four XML schemas for my four separate queries. Some are mixed with displaying strings and integers.

### Query 1

<?xml version="1.0" encoding="UTF-8"?>

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

<xs:element name="artist\_collection">

<xs:complexType>

<xs:sequence>

<xs:element name="artist" type="xs:string"/>

<xs:complexType>

<xs:sequence>

<xs:element name="name" type="xs:string"/>

<xs:element name="genre" type="xs:string"/>

</xs:sequence>

</xs:complexType>

</xs:sequence>

</xs:complexType>

</xs:element>

</xs:schema>

### Query 2

<?xml version="1.0" encoding="UTF-8"?>

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

<xs:element name="remix\_collection">

<xs:complexType>

<xs:sequence>

<xs:element name="recording" type="xs:string"/>

<xs:complexType>

<xs:sequence>

<xs:element name="track name" type="xs:string"/>

<xs:element name="artist" type="xs:string"/>

</xs:sequence>

</xs:complexType>

</xs:sequence>

</xs:complexType>

</xs:element>

</xs:schema>

### Query 3

<?xml version="1.0" encoding="UTF-8"?>

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

<xs:element name="music\_releases">

<xs:complexType>

<xs:sequence>

<xs:element name="track" type="xs:string"/>

<xs:complexType>

<xs:sequence>

<xs:element name="artist" type="xs:string"/>

<xs:element name="release" type="xs:string"/>

<xs:element name="title" type="xs:string"/>

<xs:element name="year" type="xs:integer"/>

</xs:sequence>

</xs:complexType>

</xs:sequence>

</xs:complexType>

</xs:element>

</xs:schema>

### Query 4

<?xml version="1.0" encoding="UTF-8"?>

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

<xs:element name="release\_category">

<xs:complexType>

<xs:sequence>

<xs:element name="release" type="xs:string"/>

<xs:complexType>

<xs:sequence>

<xs:element name="type" type="xs:string"/>

<xs:element name="artist" type="xs:string"/>

<xs:element name="name" type="xs:string"/>

<xs:element name="year" type="xs:integer"/>

</xs:sequence>

</xs:complexType>

</xs:sequence>

</xs:complexType>

</xs:element>

</xs:schema>

# Question 5

We are writing a similar Python script with functions which for each of the four queries connects to the database and produces JSON documents.

## Python script JSON 1

This JSON format displays all artist under the Deep House genre.

def artistGenreJSON():

import psycopg2

import json

connectstr = "dbname='musicbrainz\_db' user='test\_student' password='test\_student' host='localhost'"

conn = psycopg2.connect(connectstr)

cursor = conn.cursor()

cursor.execute("""SELECT DISTINCT a.name, t.name, t.ref\_count

FROM artist\_tag at

JOIN artist a ON a.id = at.artist

JOIN tag t ON t.id = at.tag

WHERE t.name='deep house'

ORDER BY a.name

LIMIT 100;

""")

rows = cursor.fetchall()

result = []

for row in rows:

d = dict()

d['Artist'] = row[0]

d['Genre'] = row[1]

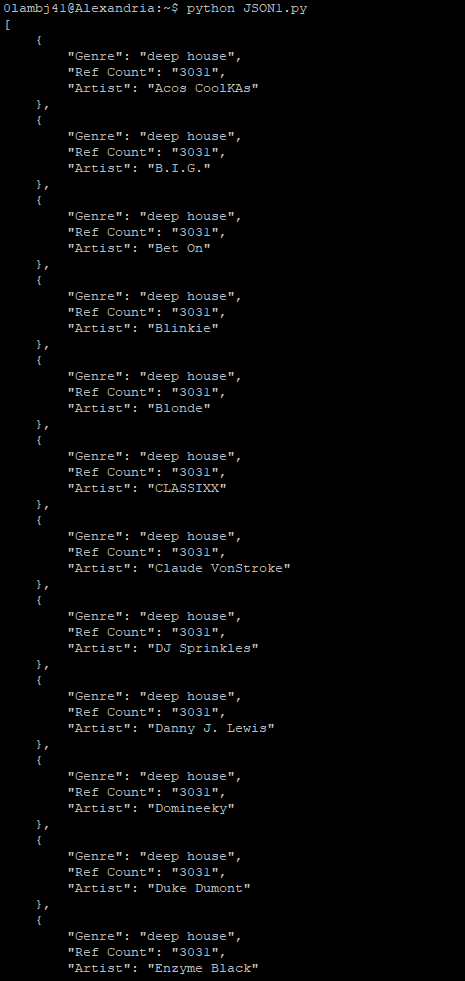
d['Ref Count'] = str(row[2])

result.append(d)

subjects = json.dumps(result, indent=4)

print subjects

artistGenre()

Here is the extracts from the JSON document

## Python script JSON 2

This JSON format displays all versions for each tracks including remixes and features.

def trackVersionsJSON()

import psycopg2

import json

connectstr = "dbname='musicbrainz\_db' user='test\_student' password='test\_student' host='localhost'"

conn = psycopg2.connect(connectstr)

cursor = conn.cursor()

cursor.execute("""SELECT DISTINCT r.name, a.name

FROM recording r

JOIN track t ON r.id = t.recording

JOIN artist\_credit ac ON ac.id = t.artist\_credit

JOIN artist\_credit\_name an ON ac.id = an.artist\_credit

JOIN artist a ON a.id = an.artist

JOIN artist\_tag at ON a.id = at.artist

JOIN tag tg ON tg.id = at.tag

WHERE tg.name='deep house'

ORDER BY a.name ASC, r.name ASC

LIMIT 100;

""")

rows = cursor.fetchall()

result = []

for row in rows:

d = dict()

d['Track Name'] = row[0]

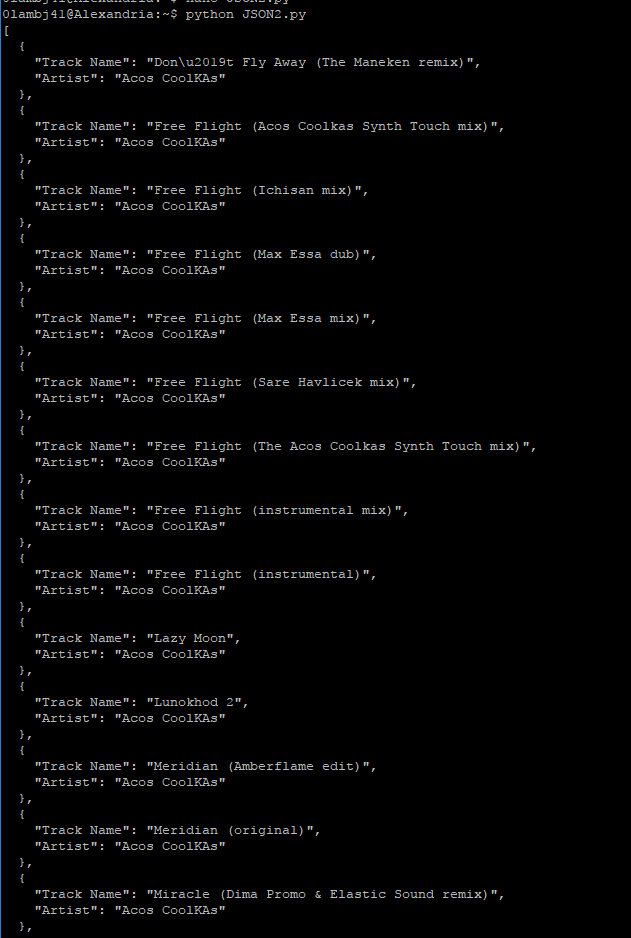
d['Artist'] = row[1]

result.append(d)

subjects = json.dumps(result, indent=2)

print subjects

trackVersionsJSON()

Here is the extracts from the JSON document

## Python script JSON 3

This JSON format displays the year for the artists releases.

def releaseYearJSON();

import psycopg2

import json

connectstr = "dbname='musicbrainz\_db' user='test\_student' password='test\_student' host='localhost'"

conn = psycopg2.connect(connectstr)

cursor = conn.cursor()

cursor.execute("""SELECT DISTINCT a.name, r.name, t.name, rc.date\_year

FROM artist a

JOIN artist\_credit\_name acn ON a.id = acn.artist

JOIN artist\_credit ac ON ac.id = acn.artist\_credit

JOIN track t ON ac.id = t.artist\_credit

JOIN release r ON r.artist\_credit = ac.id

JOIN release\_country rc ON r.id = rc.release

JOIN artist\_tag at ON a.id = at.artist

JOIN tag tg ON tg.id = at.tag

AND tg.name='deep house'

AND rc.date\_year BETWEEN 2000 AND 2017

GROUP BY r.name, t.name, rc.date\_year, a.name

ORDER BY a.name ASC

LIMIT 100;

""")

rows = cursor.fetchall()

result = []

for row in rows:

d = dict()

d['Artist'] = row[0]

d['Release'] = row[1]

d['Title'] = row[2]

d['Year'] = str(row[3])

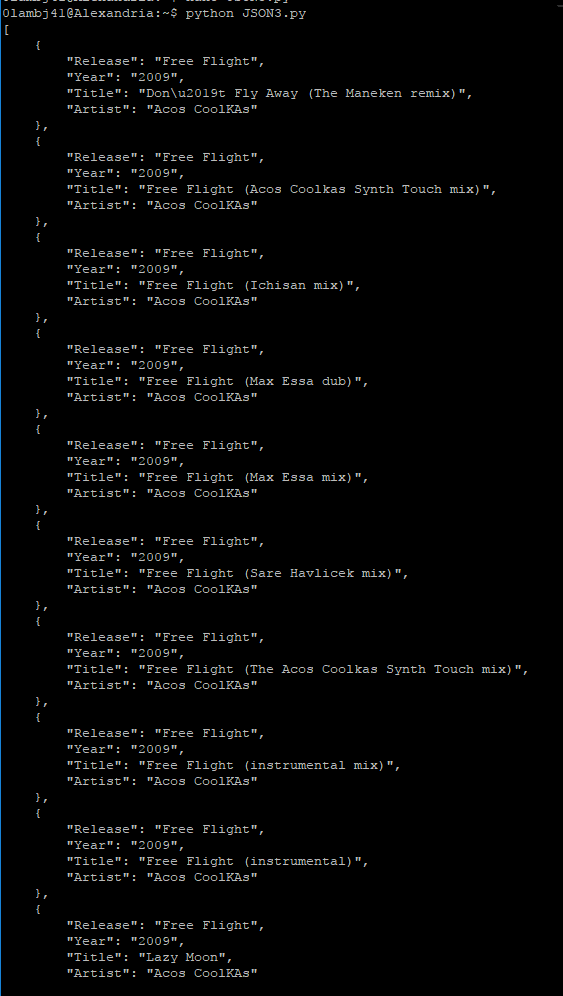
result.append(d)

subjects = json.dumps(result, indent=4)

print subjects

releaseYearJSON()

Here is the extracts from the JSON document



## Python script JSON 4

This JSON format displays the types of releases from artists.

def releaseTypeJSON();

import psycopg2

import json

connectstr = "dbname='musicbrainz\_db' user='test\_student' password='test\_student' host='localhost'"

conn = psycopg2.connect(connectstr)

cursor = conn.cursor()

cursor.execute("""SELECT DISTINCT rp.name, a.name , r.name , rc.date\_year

FROM artist a

JOIN artist\_credit\_name acn ON a.id = acn.artist

JOIN artist\_credit ac ON ac.id = acn.artist\_credit

JOIN track t ON ac.id = t.artist\_credit

JOIN release r ON r.artist\_credit = ac.id

JOIN release\_country rc ON r.id = rc.release

JOIN artist\_tag at ON a.id = at.artist

JOIN tag tg ON tg.id = at.tag

JOIN release\_group rg ON rg.id = r.release\_group

JOIN release\_group\_primary\_type rp ON rp.id = rg.type

AND tg.name='deep house'

AND rc.date\_year BETWEEN 2000 AND 2017

GROUP BY rp.name, r.name, rc.date\_year, a.name

ORDER BY a.name

LIMIT 100;

""")

rows = cursor.fetchall()

result = []

for row in rows:

d = dict()

d['Type'] = row[0]

d['Artist'] = row[1]

d['Release Title'] = row[2]

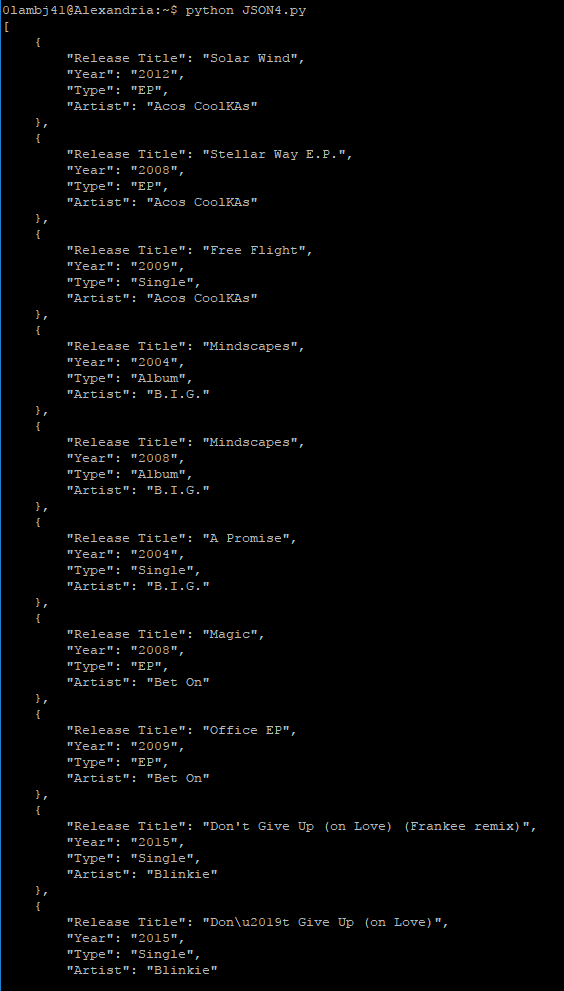
d['Year'] = str(row[3])

result.append(d)

subjects = json.dumps(result, indent=4)

print subjects

releaseTypeJSON()

Here is the extracts from the JSON document

# Question 6

Here we are migrating the necessary data from Musicbrainz to MongoDB.