Southampton Solent University

COMPUTING

SAD 600

Data Science

JAKE LAMBERT

STUDENT ID: Q10158341

# Question 1

Name

Time 2 Vibe.

Purpose

The purpose of this music application lets you rewind back to the past and discover old albums, singles & remixes based on deep house music (personal genre favourites), from the last 10 years. You can now look back at how deep house music (your favourite genres) has changed and evolved over time.

How people will use it

Users will be presented with an interface where they can scroll through a diverse amount of genres and upon selecting a genre they will then pick a time frame (generation) they want to listen from. This will present the user with playlists sorted by given genres and a period in time.

Tables

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

# Question 2

Key-value Stores

With the lack of development flexibility in MusicBrainz, key-value stores offer flexible data modelling. Because 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/>

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.

They are very efficient when querying and analysing because columnar stores can be loaded extremely fast, with billion row tables can be loaded within a few seconds.

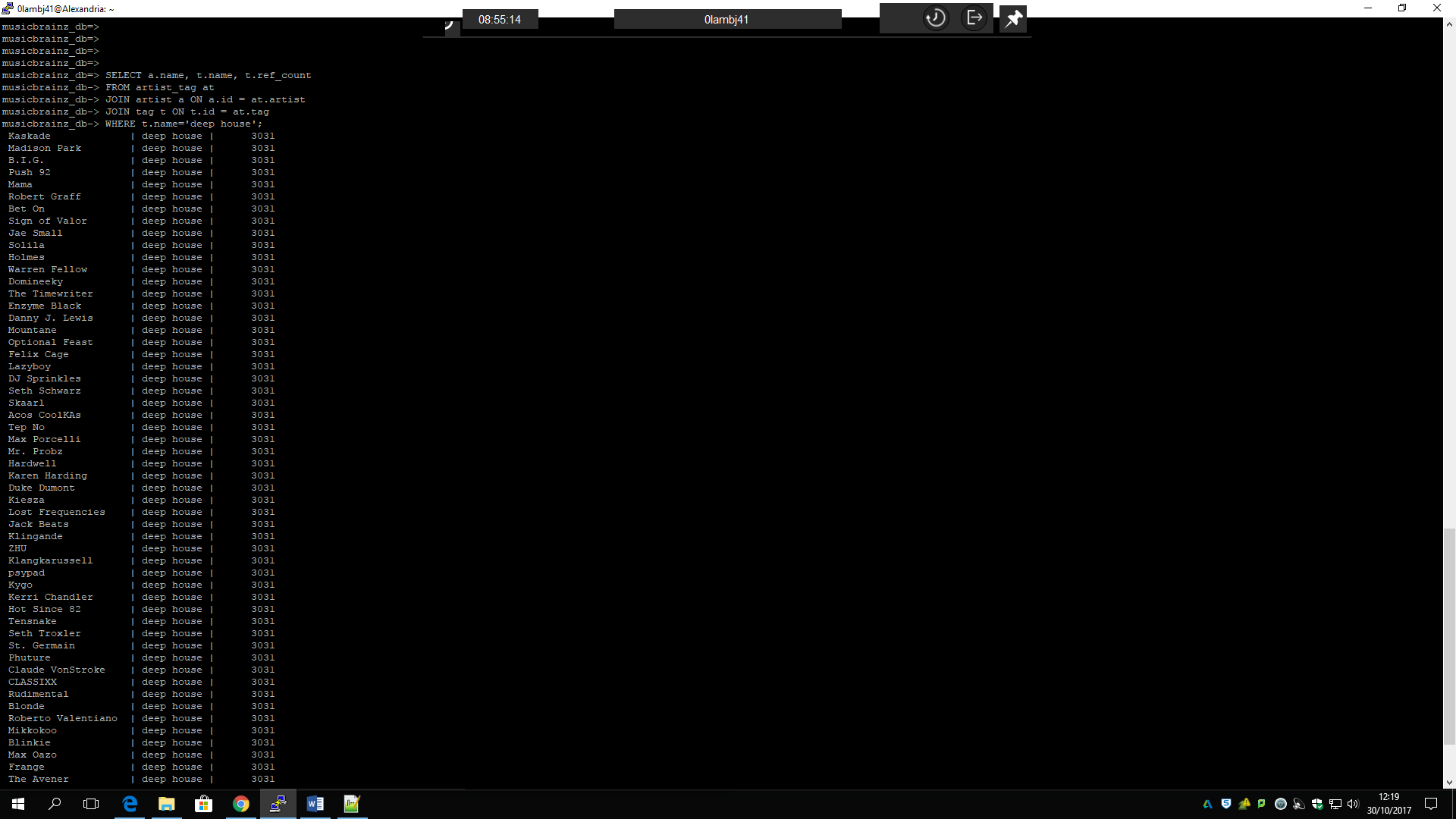
<http://database.guide/what-is-a-column-store-database/>

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.

# Question 3

Query 1

This query gathers all the artists that have produced deep house tracks. The artist\_tag is the link between artist and tag tables.

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

FROM artist\_tag at

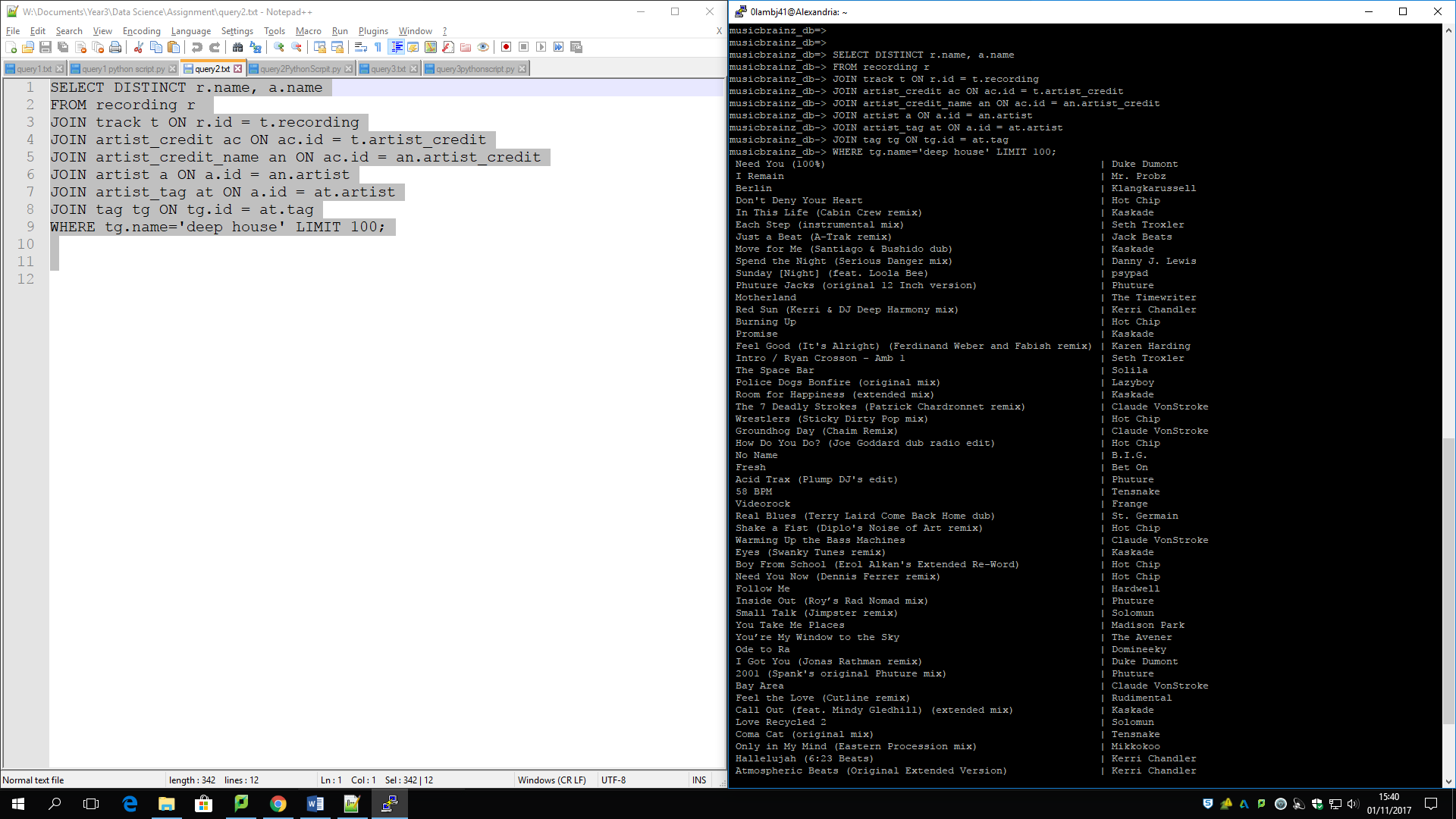
JOIN artist a ON a.id = at.artist

JOIN tag t ON t.id = at.tag

WHERE t.name='deep house';

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 of the song as well as the other 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 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

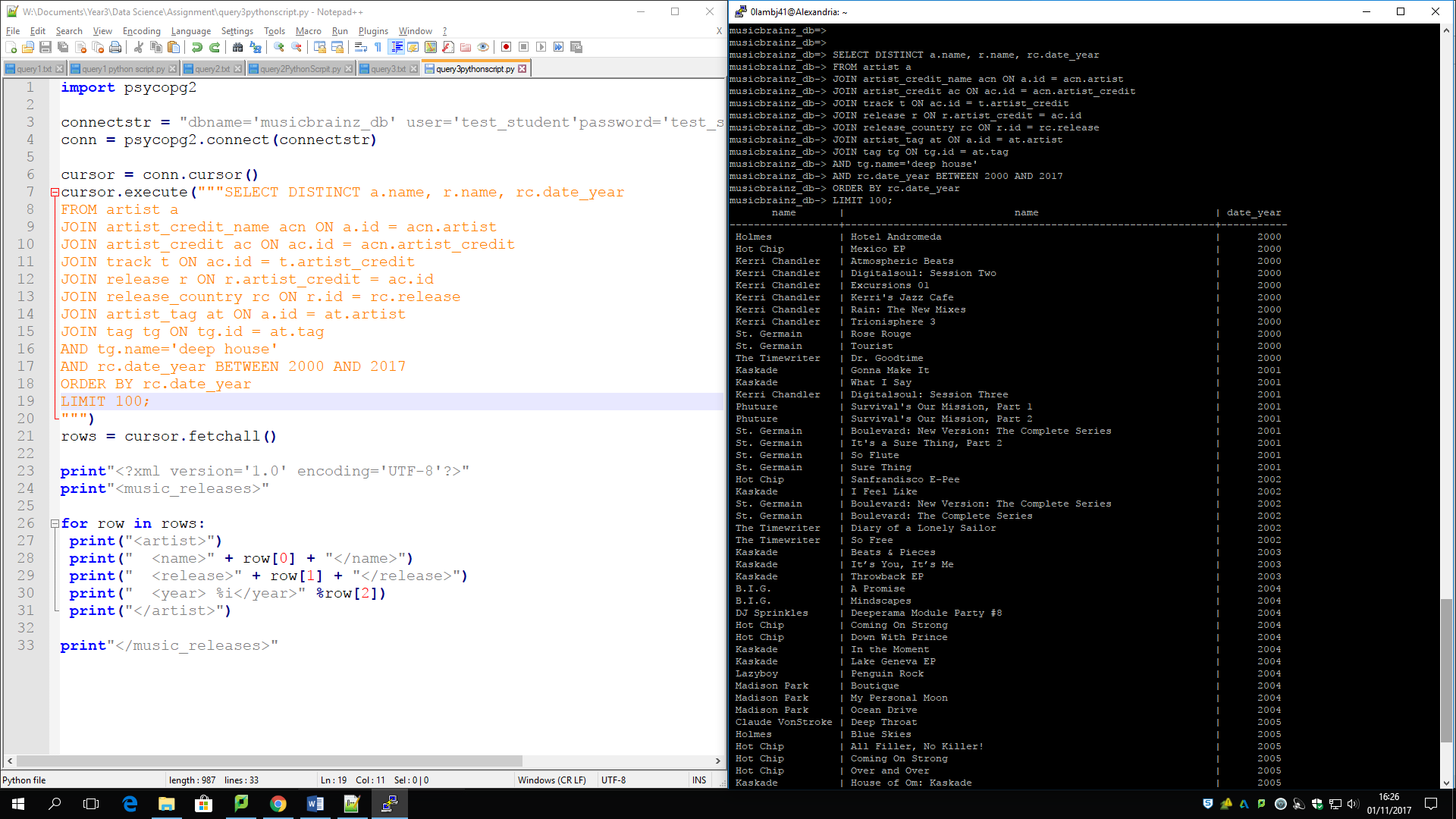
JOIN tag tg ON tg.id = at.tag

WHERE tg.name='deep house' LIMIT 100;

Query 3

This query displays all the releases of a particular artist under the Deep House genre. It retrieves the data to state if the release is either a single, album or EP, so it is a group of releases with a specified type. It displays the release between the years 2000 and 2017, the modern era.

I was meant to include the name row the release\_group table but I was unable to test it in Putty as it kept saying No space left on device



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

ORDER BY r.name ASC

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 data has been outputted in string format, so under artist tags and nested within this tag are the name and genre tags.

def artistGenre();

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';

""")

rows = cursor.fetchall()

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

print"<artist\_collection>"

for row in rows:

print("<artist>")

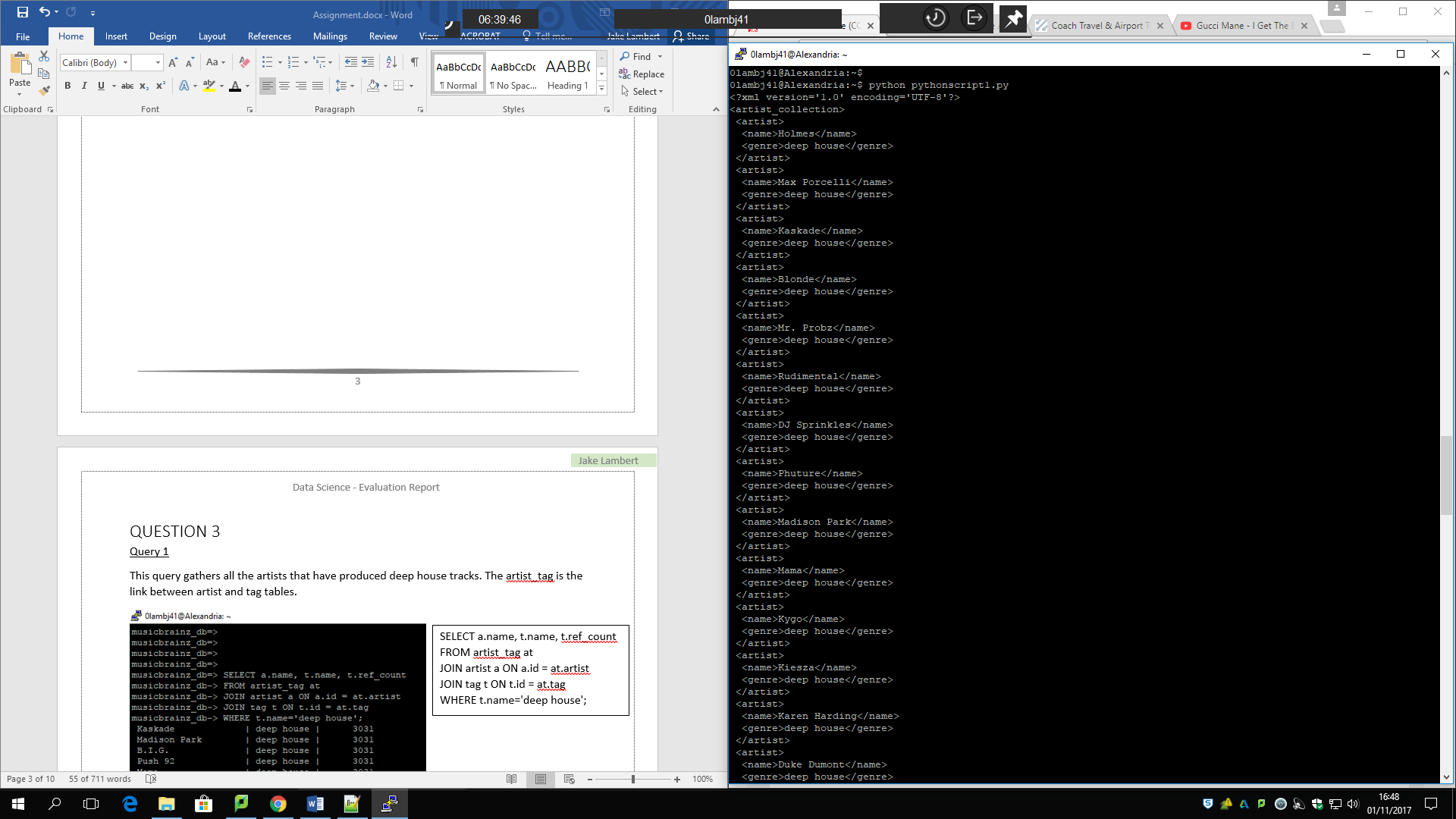
print(" <name>" + row[0] + "</name>")

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

print("</artist>")

print"</artist\_collection>"

artistGenre()



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.

def differentRecordings();

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' LIMIT 100;

""")

rows = cursor.fetchall()

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

print"<remix\_collection>"

for row in rows:

print("<artist>")

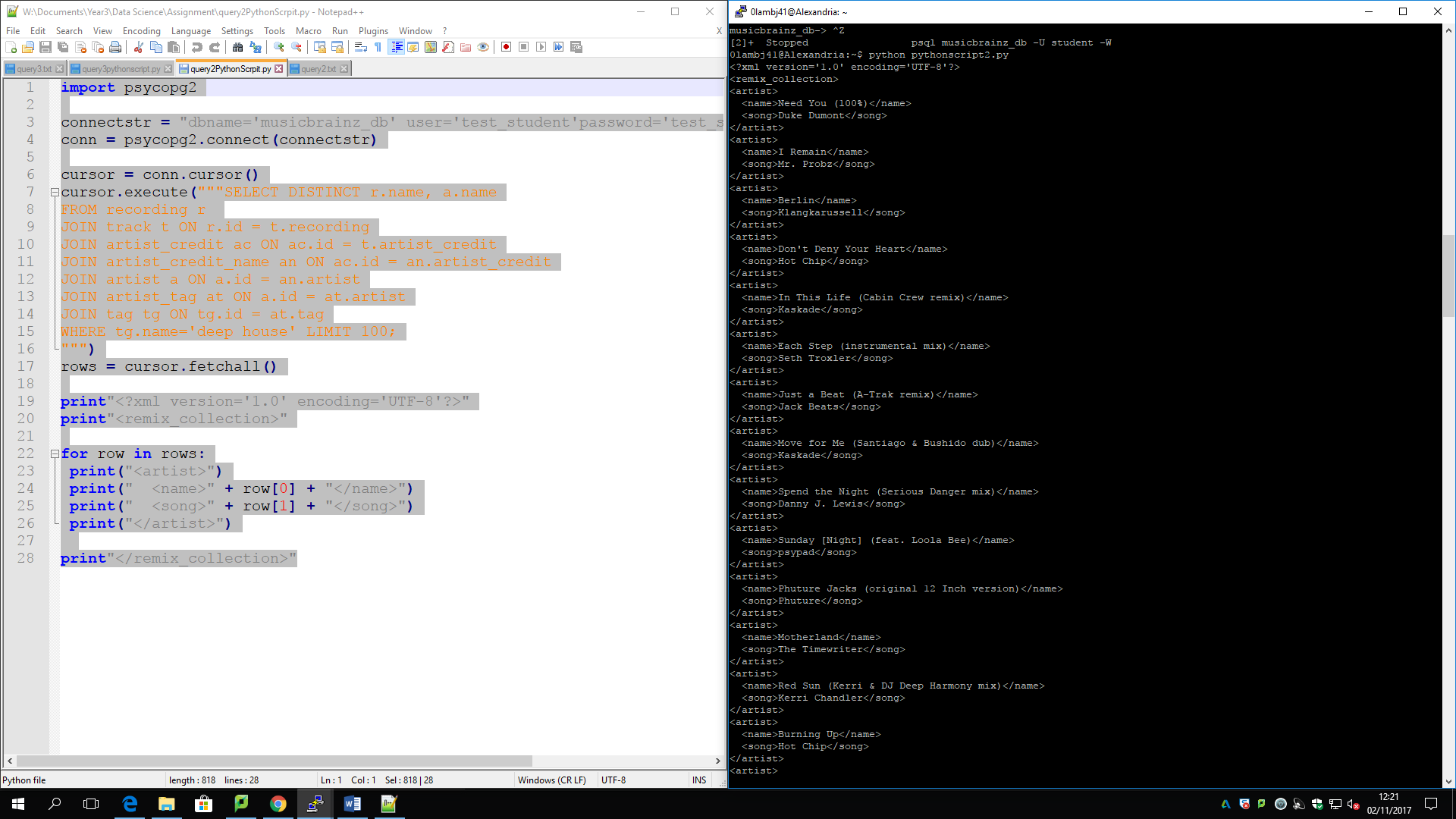
print(" <name>" + row[0] + "</name>")

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

print("</artist>")

print"</remix\_collection>"

differentRecordings()



Python script 3

This python script shows all the releases from the 2000’s. The string data formatting is outputted inside again an tag with the artist name, release title and year or released tags, and is ordered by the release year.

def releaseYear();

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, 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

ORDER BY rc.date\_year

LIMIT 100;

""")

rows = cursor.fetchall()

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

print"<music\_releases>"

for row in rows:

print("<artist>")

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

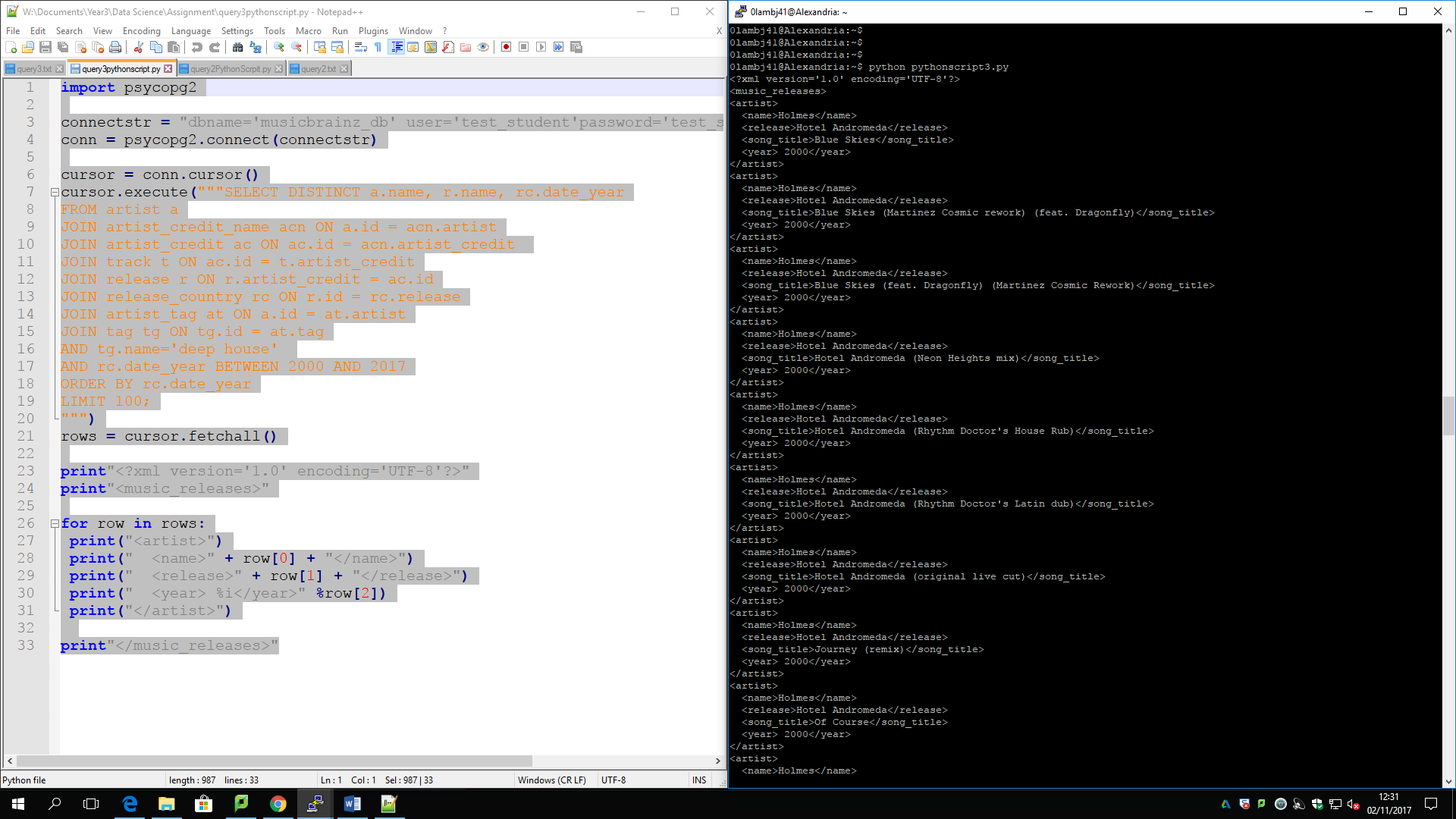
print(" <release>" + row[1] + "</release>")

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

print("</artist>")

print"</music\_releases>"

releaseYear()



# Question 5

Python and databases week work

# Question 6

Nosql.pdf

Use Pymongo

# Question 7

Nosql.pdf