Project - Database Model Comparison

The Implementation Dataset : University Enrollment Data

12/12/2020 / By Jonathan Njeunje

Introduction ¶

The main idea of this project is the comparison of three types of the databases: differences in design approach, implementation, query composition and performance.

The general approaches we will follow is to create 3 copies of the same database. Create three databases and an application working with all three. One - relational DB Oracle, MongoDB, and Neo4j.

Scenario

For this implemenetatio we will use the following dataset:

The University Enrollment database - A database that keeps information geared towards students' enrollment in the university: their majors, year enrolled, classes taken, grades received, withdrawals, etc. Also their background info, age, gender, high-school attended, home address, current housing arrangements (dorm, rent, parents) etc. The application should be able to answer , from "Which major has the most students retained after the first year?" to "What course has the highest percent of people with grade W?", "What's a 4-year graduation rate among the female students, grouped by housing arrangements?", etc.

Selected Queries

- Which major has the most students retained after the first year?
- What course has the highest percent of people with grade W?
- What's a 4-year graduation rate among the female students, grouped by majors?
- Rank all courses in terms of their popularity (cumulated number enrollments).
- What is the trend of enrollment over the years?
- How does the number of full-time students vs part-time students compare for both graduate and undergraduate students?
- What are the percentages of female vs male students in STEM programs?
- What is the distribution of students with respect to their race?
- What is the percentage of enrollment per state in the USA?
- Which instructor is most "effective" in the long run? - 80%*score+20%*absenceRate
- What is the most popular course (based on enrollment count) With respect to the country of origin?

Required Installs

In [1]:
```
# pip install requests
```

In [2]:
```
# pip install cx_Oracle --upgrade
```

In [3]:
```
# pip install neo4j-driver
```

Required Libraries

In [376]:
```
# Global imports
import pprint # for pretty printing
import numpy as np
import json
import glob
```

In [377]:
```
# Benchmark

# Find out what type of Python we running
import sys
# https://www.geeksforgeeks.org/python-measure-time-taken-by-program-to-execut
e/
# Code to Measure time taken by program to execute.
import time
import math

import pandas as pds
pds.set_option('display.max_colwidth', -1)
```

In [378]:
```
# Data Generation
import requests
```

In [379]:
```
# Ref: https://www.heatonresearch.com/content/oracle.html
# Use the directory you unzipped the instant client to:
import os
# sys.path.insert(1, 'c:\\Oracle\\instantclient_19_9')
# workspace_dir = os.getcwd()
# oracle_instantclient_dir = "C:\\Oracle\\instantclient_19_9"
# os.chdir(oracle_instantclient_dir)
import cx_Oracle
# os.chdir(workspace_dir)
# display(sys.path)
```

In [380]:
```
# MongoDB
import pymongo
from pymongo import MongoClient
```

In [381]:
```
# Neo4j
from neo4j import GraphDatabase
```

In [382]:
```
# Benchmark
import matplotlib.pyplot as plt
```

Benchmark

Criteria:

- The execution time
- The features availability

Configuration

Python Environment:

In [11]:
```
print("Python version: " + sys.version)
```

```
Python version: 3.7.3 (default, Apr 24 2019, 15:29:51) [MSC v.1915 64 bit (AM
D64)]
```

Oracle Environment:

- Environment: Online, Linux OS
- Version: 12.1.0.2.0

MongoDB Environment:

- Environment: Online, Linux OS, arch: x86_64
- Version: 4.2.11

Neo4j Environment:

- Environment: Localhost, Windows OS, 64
- Version: 4.2.1

Data Generator

- The mock data is generated from www.mockaroo.com.
- Different schemas and APIs (to those schemas) where used to generate the mock data.
- The account use at amockaroo.com, is a free limited account to a 1000 records per API calls, with a max of 200 API calls per 24 hrs

In [70]:
```
# # function to add to JSON
# def write_data_to_json(data, filename='file.json'):
#     data_uni = []
#     for batch in data:
#         for record in batch:
#             data_uni.append(record)

#     with open(filename,'w') as f:
#         json.dump(data_uni, f, indent=4)

#     print("Records count:", len(data_uni))
```

In [13]:
```
# # function to add to JSON
# def write_data_to_json(data, filename='file.json', append = 0):

#     data_uni = []
#     for batch in data:
#         for record in batch:
#             data_uni.append(record)

#     if append == 1:
#         with open(filename) as json_file:
#             try:
#                 temp = json.load(json_file)
#             except:
#                 temp = []
#             temp.append(data_uni)
#             data_uni = temp
#     print("Records count:", len(data_uni))

#     with open(filename,'w') as json_file:
#         json.dump(data_uni, json_file, indent=4)

# # json_file.close()
```

Major:

In [14]:
```
# api_call = "https://my.api.mockaroo.com/universityenrollment_major.json?key=
8e82cd00"
# number_requests = 1

# status_code = []
# data = []
# for idx in range(number_requests):
#     response = requests.get(api_call)
#     status_code.append(response.status_code)
#     data.append(response.json())
# print("API call status code: ", status_code)

# write_data_to_json(data, "dataset/UniversityEnrollment_Major.json")
```

Student:

In [15]:
```
# api_call = "https://my.api.mockaroo.com/universityenrollment_student.json?ke
y=8e82cd00"
# number_requests = 30

# status_code = []
# data = []
# for idx in range(number_requests):
#     init_ID = (1000 * idx)
#     parameters = {
#         "init_ID": init_ID
#     }
#     response = requests.get(api_call, params = parameters)
#     status_code.append(response.status_code)
#     data.append(response.json())
# print("API call status code: ", status_code)

# batch = len(glob.glob("dataset/student/*.json")) + 1

# write_data_to_json(data, "dataset/student/UniversityEnrollment_Student_" + s
tr(batch) + ".json")
```

In [89]:
```
data = []
for file in glob.glob("dataset/student/*.json"):
    with open(file, "r") as json_file:
        data.append(json.load(json_file))


idx = 1
data_uni = []
std_limit = 100000
for batch in data:
    for record in batch:
        std_limit -= 1
        if std_limit < 0: break
        record["Student"]["ID"] = idx
        data_uni.append(record)
        idx += 1

display(len(data_uni))
with open("dataset/UniversityEnrollment_Student.json", "w") as json_file:
     json.dump(data_uni, json_file, indent = 4)
```

```
100000
```

Enrollment:

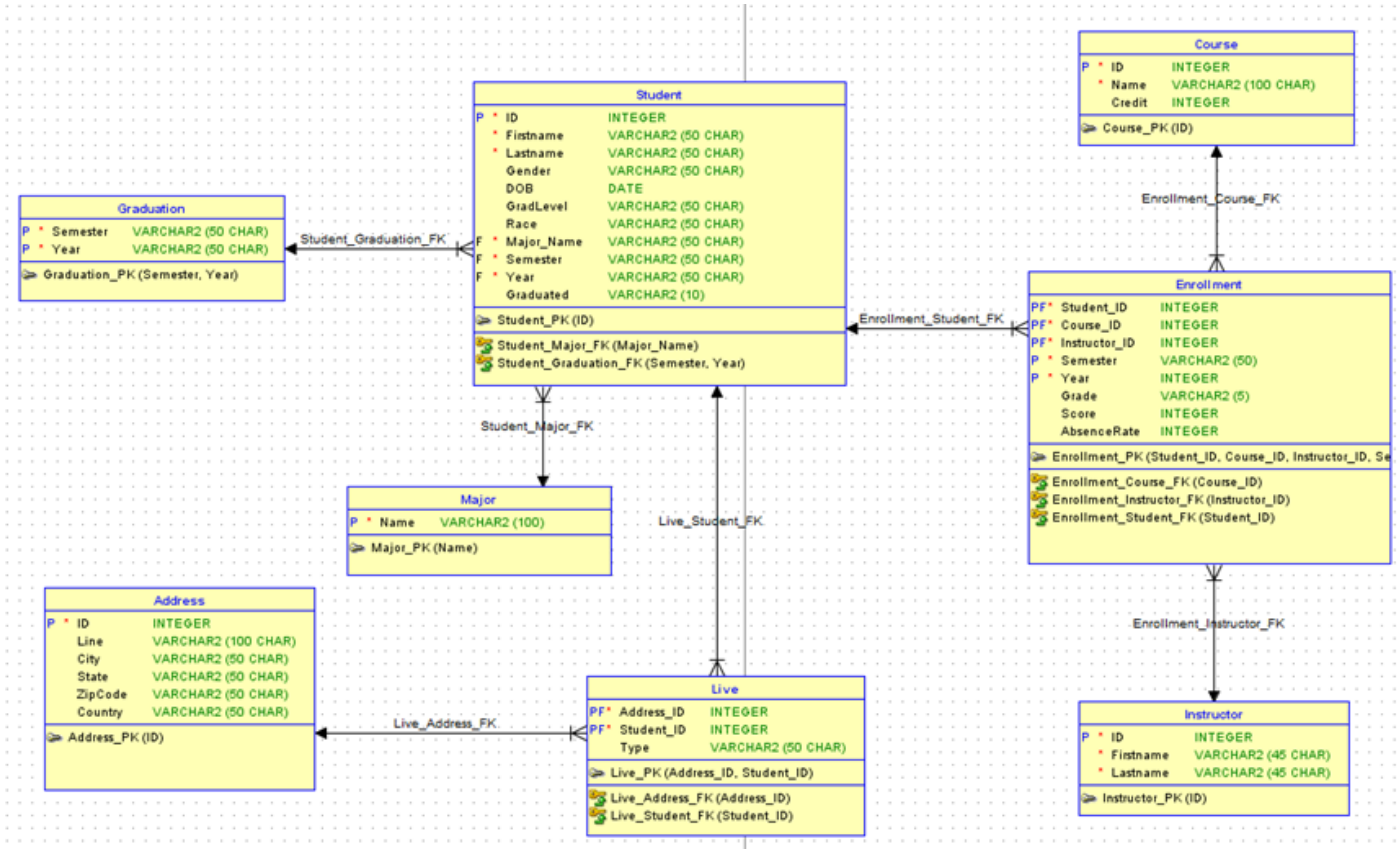
In [90]:
```
# api_call = "https://my.api.mockaroo.com/universityenrollment_enrollment.jso
n?key=8e82cd00"
# number_requests = 40

# status_code = []
# data = []
# for idx in range(number_requests):
#     response = requests.get(api_call)
#     status_code.append(response.status_code)
#     data.append(response.json())
# print("API call status code: ", status_code)

# batch = len(glob.glob("dataset/enrollment/*.json")) + 1

# write_data_to_json(data, "dataset/enrollment/UniversityEnrollment_Enrollment
_" + str(batch) + ".json")
```

In [94]:
```
data = []
for file in glob.glob("dataset/enrollment/*.json"):
    with open(file, "r") as json_file:
        data.append(json.load(json_file))

std_limit = 100000 # make sure it equal that from the student's section.
enr_limit = 200000
data_uni = []
for batch in data:
    for record in batch:
        enr_limit -= 1
        if enr_limit < 0: break
        record["Student"]["ID"] = (record["Student"]["ID"] % (std_limit - 1))
+ 1 # Rematching the student IDs
        data_uni.append(record)


display(len(data_uni))
with open("dataset/UniversityEnrollment_Enrollment.json", "w") as json_file:
     json.dump(data_uni, json_file, indent = 4)
```

```
200000
```

Databases Modeling and Loading

Oracle


Course
P * ID INTEGER
* Name VARCHAR2 (100 CHAR)
Credit INTEGER
Course_PK (ID)
Enrollment_Course_FK
Student
P * ID INTEGER
* Firstname VARCHAR2 (50 CHAR)
* Lastname VARCHAR2 (50 CHAR)
Gender VARCHAR2 (50 CHAR)
DOB DATE
GradLevel VARCHAR2 (50 CHAR)
Race VARCHAR2 (50 CHAR)
F * Major_Name VARCHAR2 (50 CHAR)
F * Semester VARCHAR2 (50 CHAR)
F * Year VARCHAR2 (50 CHAR)
Graduated VARCHAR2 (10)
Student_PK (ID)
Student_Major_FK (Major_Name)
Student_Graduation_FK (Semester, Year)
Graduation
P * Semester VARCHAR2 (50 CHAR)
P * Year VARCHAR2 (50 CHAR)
Graduation_PK (Semester, Year)
Student_Graduation_FK
Enrollment_Student_FK
Enrollment
PF * Student_ID INTEGER
PF * Course_ID INTEGER
PF * Instructor_ID INTEGER
P * Semester VARCHAR2 (50)
P * Year INTEGER
Grade VARCHAR2 (5)
Score INTEGER
AbsenceRate INTEGER
Enrollment_PK (Student_ID, Course_ID, Instructor_ID, Se
Enrollment_Course_FK (Course_ID)
Enrollment_Instructor_FK (Instructor_ID)
Enrollment_Student_FK (Student_ID)
Student_Major_FK
Major
P * Name VARCHAR2 (100)
Major_PK (Name)
Live_Student_FK
Enrollment_Instructor_FK
Address
P * ID INTEGER
Line VARCHAR2 (100 CHAR)
City VARCHAR2 (50 CHAR)
State VARCHAR2 (50 CHAR)
ZipCode VARCHAR2 (50 CHAR)
Country VARCHAR2 (50 CHAR)
Address_PK (ID)
Live_Address_FK
Live
PF * Address_ID INTEGER
PF * Student_ID INTEGER
Type VARCHAR2 (50 CHAR)
Live_PK (Address_ID, Student_ID)
Live_Address_FK (Address_ID)
Live_Student_FK (Student_ID)
Instructor
P * ID INTEGER
* Firstname VARCHAR2 (45 CHAR)
* Lastname VARCHAR2 (45 CHAR)
Instructor_PK (ID)

Connection to the Database:

To connect to the oracle database we need to download and install the instant_client

For Windows machines, you can follow https://www.oracle.com/database/technologies/instant-client/winx64-64-downloads.html (https://www.oracle.com/database/technologies/instant-client/winx64-64-downloads.html) and download a basic instant client (version 19.9). Follow the installation instructions on https://www.oracle.com/database/technologies/instant-client/winx64-64-downloads.html#ic_winx64_inst (https://www.oracle.com/database/technologies/instant-client/winx64-64-downloads.html#ic_winx64_inst).

Other Ref: https://oracle.github.io/python-cx_Oracle/ (https://oracle.github.io/python-cx_Oracle/)

- There is an oracle database restriction of 50MB for every user. The dataset had to be scaled down to 100000 student records and 200000 enrollment records

In [386]:
```
# myscript.py
host = "oracle1.wiu.edu"
port = "1521"
sid = "toolman"
user = "ajmn100"
password = "CS523_3907"

dsn_tns = cx_Oracle.makedsn(host, port, sid=sid) #if connecting to WIU Oracle
 server
# OR
#dsn_tns = cx_Oracle.makedsn(host, port, service_name=sid) #if connecting to l
ocalhost

# Connect as user with password to the dsn service.
oracle_connection = cx_Oracle.connect(user=user, password=password, dsn=dsn_tn
s,  encoding="UTF-8", nencoding="UTF-8")
```

Verify Connection:

In [387]:
```
print("Connected to Oracle: " + oracle_connection.version)
```

```
Connected to Oracle: 12.1.0.2.0
```

In [388]:
```
oracle_cursor = oracle_connection.cursor()
```

Reset Database:

In [98]:
```
def runsqllines(sqllines):
    for sql in sqllines.split(";"):
        if sql.strip() == "": continue
        try:
            oracle_cursor.execute(sql)
        except Exception as e:
            print(e)
```

In [99]:

```
def reset_oracle():
    sqllines = """
    DROP TABLE enrollment CASCADE CONSTRAINTS;
    DROP TABLE live CASCADE CONSTRAINTS;
    DROP TABLE course CASCADE CONSTRAINTS;
    DROP TABLE instructor CASCADE CONSTRAINTS;
    DROP TABLE student CASCADE CONSTRAINTS;
    DROP TABLE graduation CASCADE CONSTRAINTS;
    DROP TABLE major CASCADE CONSTRAINTS;
    DROP TABLE address CASCADE CONSTRAINTS;
    """
    runsqllines(sqllines)

    sqllines= """
    CREATE TABLE address (
        id       INTEGER NOT NULL,
        line     VARCHAR2(100 CHAR),
        city     VARCHAR2(50 CHAR),
        state    VARCHAR2(50 CHAR),
        zipcode  VARCHAR2(50 CHAR),
        country  VARCHAR2(50 CHAR)
    );

    ALTER TABLE address ADD CONSTRAINT address_pk PRIMARY KEY ( id );

    CREATE TABLE course (
        id      INTEGER NOT NULL,
        name    VARCHAR2(100 CHAR) NOT NULL,
        credit  INTEGER
    );

    ALTER TABLE course ADD CONSTRAINT course_pk PRIMARY KEY ( id );

    CREATE TABLE enrollment (
        student_id     INTEGER NOT NULL,
        course_id      INTEGER NOT NULL,
        instructor_id  INTEGER NOT NULL,
        semester       VARCHAR2(50) NOT NULL,
        year           INTEGER NOT NULL,
        grade          VARCHAR2(5),
        score          INTEGER,
        absencerate    INTEGER
    );

    ALTER TABLE enrollment
        ADD CONSTRAINT enrollment_pk PRIMARY KEY ( student_id,
                                                   course_id,
                                                   instructor_id,
                                                   semester,
                                                   year );

    CREATE TABLE graduation (
        semester  VARCHAR2(50 CHAR) NOT NULL,
        year      VARCHAR2(50 CHAR) NOT NULL
    );
```

```
    ALTER TABLE graduation ADD CONSTRAINT graduation_pk PRIMARY KEY ( semeste
r,
                                                                     year );

    CREATE TABLE instructor (
        id         INTEGER NOT NULL,
        firstname  VARCHAR2(45 CHAR) NOT NULL,
        lastname   VARCHAR2(45 CHAR) NOT NULL
    );

    ALTER TABLE instructor ADD CONSTRAINT instructor_pk PRIMARY KEY ( id );

    CREATE TABLE live (
        address_id  INTEGER NOT NULL,
        student_id  INTEGER NOT NULL,
        type        VARCHAR2(50 CHAR)
    );

    ALTER TABLE live ADD CONSTRAINT live_pk PRIMARY KEY ( address_id,
                                                         student_id );

    CREATE TABLE major (
        name VARCHAR2(100) NOT NULL
    );

    ALTER TABLE major ADD CONSTRAINT major_pk PRIMARY KEY ( name );

    CREATE TABLE student (
        id          INTEGER NOT NULL,
        firstname   VARCHAR2(50 CHAR) NOT NULL,
        lastname    VARCHAR2(50 CHAR) NOT NULL,
        gender      VARCHAR2(50 CHAR),
        dob         DATE,
        gradlevel   VARCHAR2(50 CHAR),
        race        VARCHAR2(50 CHAR),
        major_name  VARCHAR2(50 CHAR) NOT NULL,
        semester    VARCHAR2(50 CHAR) NOT NULL,
        year        VARCHAR2(50 CHAR) NOT NULL,
        graduated   VARCHAR2(10)
    );

    ALTER TABLE student ADD CONSTRAINT student_pk PRIMARY KEY ( id );

    ALTER TABLE enrollment
        ADD CONSTRAINT enrollment_course_fk FOREIGN KEY ( course_id )
            REFERENCES course ( id );

    ALTER TABLE enrollment
        ADD CONSTRAINT enrollment_instructor_fk FOREIGN KEY ( instructor_id )
            REFERENCES instructor ( id );

    ALTER TABLE enrollment
        ADD CONSTRAINT enrollment_student_fk FOREIGN KEY ( student_id )
            REFERENCES student ( id );

    ALTER TABLE live
        ADD CONSTRAINT live_address_fk FOREIGN KEY ( address_id )
```

```
        REFERENCES address ( id );

ALTER TABLE live
    ADD CONSTRAINT live_student_fk FOREIGN KEY ( student_id )
        REFERENCES student ( id );

ALTER TABLE student
    ADD CONSTRAINT student_graduation_fk FOREIGN KEY ( semester,
                                                       year )
        REFERENCES graduation ( semester,
                                year );

ALTER TABLE student
    ADD CONSTRAINT student_major_fk FOREIGN KEY ( major_name )
        REFERENCES major ( name );
"""
runsqllines(sqllines)
```

Data Load:

In [100]:

```
%%time
UniversityEnrollment_Student = 'dataset/UniversityEnrollment_Student.json'
UniversityEnrollment_Instructor = 'dataset/UniversityEnrollment_Instructor.jso
n'
UniversityEnrollment_Course = 'dataset/UniversityEnrollment_Course.json'
UniversityEnrollment_Enrollment = 'dataset/UniversityEnrollment_Enrollment.jso
n'
UniversityEnrollment_Major = 'dataset/UniversityEnrollment_Major.json'
UniversityEnrollment_Graduation = 'dataset/UniversityEnrollment_Graduation.jso
n'


# Opening JSON file returns JSON object
f = open(UniversityEnrollment_Instructor,)
data = json.load(f)

instructor_data = []
for record in data:
    instructor_data.append(tuple(record["Instructor"].values()))
# display(instructor_data)

# Closing file
f.close()


# Opening JSON file returns JSON object
f = open(UniversityEnrollment_Student,)
data = json.load(f)

address_data = []; address_id = 0
student_data = []
live_data = []
for record in data:
    address_id += 1
    record["Address"].pop("Latitude")
    record["Address"].pop("Longitude")
    address_data.append((address_id,) + tuple(record["Address"].values()))
#     address_data.encode('utf-8').strip()

    live_data.append((address_id, record["Student"]["ID"], "Home"))

    student_data.append(tuple(record["Student"].values()) + tuple(record["Majo
r"].values()) + tuple(record["Graduation"].values()))
# display(address_data)
# display(student_data)
# display(live_data)

# Closing file
f.close()


# Opening JSON file returns JSON object
f = open(UniversityEnrollment_Major,)
data = json.load(f)

major_data = []
```

```
for record in data:
    major_data.append(tuple(record["Major"].values()))
# display(major_data)

# Closing file
f.close()


# Opening JSON file returns JSON object
f = open(UniversityEnrollment_Graduation,)
data = json.load(f)

graduation_data = []
for record in data:
    graduation_data.append(tuple(record["Graduation"].values()))
# display(graduation_data)

# Closing file
f.close()


# Opening JSON file returns JSON object
f = open(UniversityEnrollment_Instructor,)
data = json.load(f)

instructor_data = []
for record in data:
    instructor_data.append(tuple(record["Instructor"].values()))
# display(instructor_data)

# Closing file
f.close()


# Opening JSON file returns JSON object
f = open(UniversityEnrollment_Course,)
data = json.load(f)

course_data = []
for record in data:
    course_data.append(tuple(record["Course"].values()))
# display(course_data)

# Closing file
f.close()


# Opening JSON file returns JSON object
f = open(UniversityEnrollment_Enrollment,)
data = json.load(f)

enrollment_data = []
for record in data:
    enrollment_data.append(tuple(record["Student"].values()) + tuple(record["C
ourse"].values()) + tuple(record["Instructor"].values()) + tuple(record["Enrol
lment"].values()))
# display(enrollment_data)
```

```
# Closing file
f.close()
```

```
Wall time: 8.08 s
```

In [101]:
```
def batches(data, number_of_batches):
# Yield successive n-sized batches from lst."""
    for idx in range(0, len(data), number_of_batches):
        yield data[idx:idx + number_of_batches]
```

In [102]:
```
def dataload_oracle():

    sql = """Insert into ADDRESS (ID,LINE,CITY,ZIPCODE,STATE,COUNTRY) values
 (:1,:2,:3,:4,:5,:6)"""
    oracle_cursor.executemany(sql, address_data) #good

    sql = """Insert into GRADUATION (SEMESTER,YEAR) values (:1,:2)"""
    oracle_cursor.executemany(sql, graduation_data) #good

    sql = """Insert into MAJOR (NAME) values (:1)"""
    oracle_cursor.executemany(sql, major_data) #good

    for batch in batches(student_data, 50000):
        sql = """Insert into STUDENT (ID,FIRSTNAME,LASTNAME,GENDER,RACE,DOB,GR
ADLEVEL,GRADUATED,MAJOR_NAME,SEMESTER,YEAR) values (:1,:2,:3,:4,:5,to_date(:
6,'mm/dd/yyyy'),:7,:8,:9,:10,:11)"""
        oracle_cursor.executemany(sql, batch)
#         break

    for batch in batches(live_data, 50000):
        sql = """Insert into LIVE (ADDRESS_ID,STUDENT_ID,TYPE) values (:1,:2,:
3)"""
        oracle_cursor.executemany(sql, batch) #
#         break

    sql = """Insert into INSTRUCTOR (ID,FIRSTNAME,LASTNAME) values (:1, :2, :
3)"""
    oracle_cursor.executemany(sql, instructor_data) #good

    sql = """Insert into COURSE (ID,NAME,CREDIT) values (:1,:2,:3)"""
    oracle_cursor.executemany(sql, course_data) #good

    for batch in batches(enrollment_data, 50000):
        sql = """Insert into ENROLLMENT (STUDENT_ID,COURSE_ID,INSTRUCTOR_ID,GR
ADE,SCORE,ABSENCERATE,SEMESTER,YEAR) values (:1,:2,:3,:4,:5,:6,:7,:8)"""
        oracle_cursor.executemany(sql, batch)
#         break

    oracle_connection.commit()

    return []
```

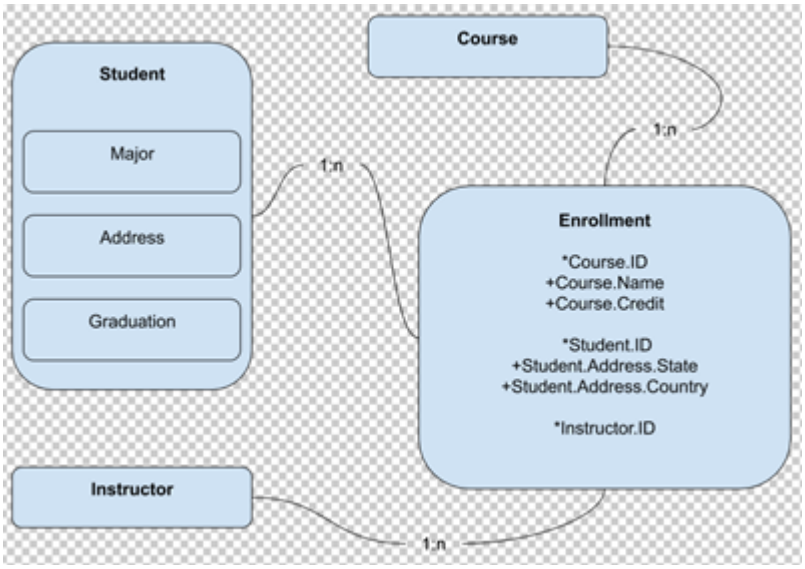
In [103]:
```
%%time
reset_oracle()
dataload_oracle()
```

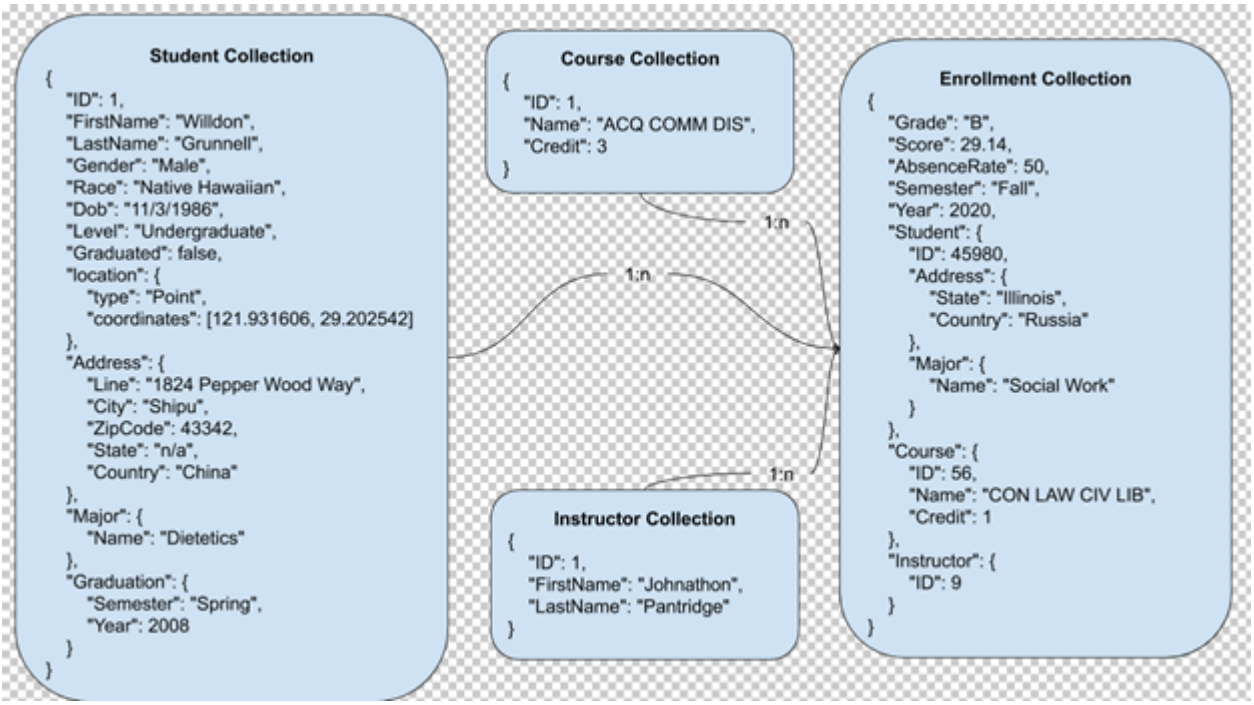
```
Wall time: 13.2 s
```

Out[103]: []

MongoDB

The Data Model:





Connect to the Database

Setup the Database:

Create a database in MongoDB Atlas, get your credentials and connection string.

Ref: https://docs.mongodb.com/manual/introduction/ (https://docs.mongodb.com/manual/introduction/)

In [389]:
```
username = "Jonalex210"
password = "6u3LvaTUPrqz9NL"

mongodb_client = pymongo.MongoClient("mongodb+srv://"+username+":"+password+"@jonathannjeunje-cluster.cgfhy.mongodb.net/UniversityEnrollment?retryWrites=true&w=majority")
```

Verify the Connection:

In [390]:
```
# dir(mongodb_client)
print("Version: " + mongodb_client.server_info()["version"])
```

```
Version: 4.2.11
```

Reset Database:

In [36]:
```
coll_student = mongodb_client.UniversityEnrollment.Student
coll_instructor = mongodb_client.UniversityEnrollment.Instructor
coll_course = mongodb_client.UniversityEnrollment.Course
coll_enrollment = mongodb_client.UniversityEnrollment.Enrollment
```

In [37]:
```
def reset_mongodb():
    coll_student.delete_many({})
    coll_instructor.delete_many({})
    coll_course.delete_many({})
    coll_enrollment.delete_many({})
```

Data Load:

In [17]:

```
def get_student_info(data, id):
    for record in data:
        if record["ID"] == id:
            return {
                "Address": {
                    "State": record["Address"]["State"],
                    "Country": record["Address"]["Country"]
                },
                "Major": record["Major"]
            }

def get_course_info(data, id):
    for record in data:
        if record["ID"] == id:
            return {
                "Name": record["Name"],
                "Credit": record["Credit"]
            }
```

In [18]:

```
%%time
UniversityEnrollment_Student = 'dataset/UniversityEnrollment_Student.json'
UniversityEnrollment_Instructor = 'dataset/UniversityEnrollment_Instructor.jso
n'
UniversityEnrollment_Course = 'dataset/UniversityEnrollment_Course.json'
UniversityEnrollment_Enrollment = 'dataset/UniversityEnrollment_Enrollment.jso
n'
UniversityEnrollment_Major = 'dataset/UniversityEnrollment_Major.json'
UniversityEnrollment_Graduation = 'dataset/UniversityEnrollment_Graduation.jso
n'

# Opening JSON file returns JSON object
f = open(UniversityEnrollment_Student,)
data = json.load(f)

coll_student_data = []
for record in data:
    temp = {}
    temp.update(record.pop("Student"))
    temp.update({
        "location": {
          "type": "Point",
          "coordinates": [record["Address"].pop("Longitude"), record["Address"
].pop("Latitude")]
        }
    })

    temp.update(record)
    coll_student_data.append(temp)
pprint.pprint(coll_student_data)

# Closing file
f.close()


# Opening JSON file returns JSON object
f = open(UniversityEnrollment_Instructor,)
data = json.load(f)

coll_instructor_data = []
for record in data: coll_instructor_data.append(record["Instructor"])
# pprint.pprint(coll_instructor_data)

# Closing file
f.close()


# Opening JSON file returns JSON object
f = open(UniversityEnrollment_Course,)
data = json.load(f)

coll_course_data = []
for record in data: coll_course_data.append(record["Course"])
# pprint.pprint(coll_course_data)

# Closing file
```

```
f.close()


# Opening JSON file returns JSON object
f = open(UniversityEnrollment_Enrollment,)
data = json.load(f)

coll_enrollment_data = []

# Iterating through the json list (batches of data)
for record in data:
    #Built the database collections based on designed schema
    temp = {}
    record["Student"].update(get_student_info(coll_student_data, record["Stude
nt"]["ID"]))
    record["Course"].update(get_course_info(coll_course_data, record["Course"]
["ID"]))
    temp.update(record.pop("Enrollment"))
    temp.update(record)
    coll_enrollment_data.append(temp)
# pprint.pprint(coll_enrollment_data)

# Closing file
f.close()
```

```
Wall time: 40min 36s
```

In [19]:
```
print("Records ~", len(coll_student_data) + len(coll_course_data) + len(coll_i
nstructor_data) + len(coll_enrollment_data))
```

```
Records ~ 300467
```

In [20]:
```
def dataload_mongodb():
    coll_student.insert_many(coll_student_data)
    coll_course.insert_many(coll_course_data)
    coll_instructor.insert_many(coll_instructor_data)
    coll_enrollment.insert_many(coll_enrollment_data)
```

In [21]:
```
%%time
reset_mongodb()
dataload_mongodb()
```

```
Wall time: 1min 15s
```

Neo4j

The Data Model:

Connect to the Database:

Setup the Database:

- Create a database in Neo4j Desktop
- Set your password
- The user is by default "neoj4"
- Start your database, and
- Get the uri connection string.

Ref: [https://neo4j.com/developer/python/ (https://neo4j.com/developer/python/)](https://neo4j.com/developer/python/)

In [391]:
```
# Create an instance of the driver object
uri = "bolt://localhost:7687"
username = "neo4j"
password = "UniversityEnrollment"
neo4j_driver = GraphDatabase.driver(uri, auth = (username, password))
```

Verify Connection:

In [392]:
```
print(neo4j_driver.verify_connectivity())
```

```
Neo4j/4.2.1

C:\Users\mvami\Anaconda3\lib\site-packages\ipykernel_launcher.py:1: Experimen
talWarning: The configuration may change in the future.
  """Entry point for launching an IPython kernel.
```

In [393]:
```
# Create an instance of the session object
neo4j_session = neo4j_driver.session()
```

Reset Database:

In [15]:
```
def reset_neo4j():
    neo4j_session.run("DROP INDEX student_id IF EXISTS");
    neo4j_session.run("DROP INDEX major_name IF EXISTS");
    neo4j_session.run("DROP INDEX instructor_id IF EXISTS");
    neo4j_session.run("DROP INDEX course_id IF EXISTS");
    neo4j_session.run("MATCH (n) DETACH DELETE n;")
```

Data Load:

In [16]:

```
%%time
UniversityEnrollment_Student = 'dataset/UniversityEnrollment_Student.json'
UniversityEnrollment_Instructor = 'dataset/UniversityEnrollment_Instructor.jso
n'
UniversityEnrollment_Course = 'dataset/UniversityEnrollment_Course.json'
UniversityEnrollment_Enrollment = 'dataset/UniversityEnrollment_Enrollment.jso
n'
UniversityEnrollment_Major = 'datasete/UniversityEnrollment_Major.json'
UniversityEnrollment_Graduation = 'dataset/UniversityEnrollment_Graduation.jso
n'

# Opening JSON file returns JSON object
f = open(UniversityEnrollment_Student,)
data = json.load(f)

neo4j_student_data = data
# pprint.pprint(neo4j_student_data)

# Closing file
f.close()


# Opening JSON file returns JSON object
f = open(UniversityEnrollment_Instructor,)
data = json.load(f)

neo4j_instructor_data = data
# pprint.pprint(neo4j_instructor_data)

# Closing file
f.close()


# Opening JSON file returns JSON object
f = open(UniversityEnrollment_Course,)
data = json.load(f)

neo4j_course_data = data
# pprint.pprint(neo4j_student_data)

# Closing file
f.close()


# Opening JSON file returns JSON object
f = open(UniversityEnrollment_Enrollment,)
data = json.load(f)

neo4j_enrollment_data = data
# pprint.pprint(neo4j_enrollment_data)

# Closing file
f.close()
```

```
Wall time: 2.53 s
```

In [17]:
```
def batches(data, number_of_batches):
# Yield successive n-sized batches from lst."""
    for idx in range(0, len(data), number_of_batches):
        yield data[idx:idx + number_of_batches]
```

In [18]:

```
def dataload_neo4j():

    neo4j_session.run("CREATE INDEX student_id FOR (n:Student) ON (n.ID)");
#     neo4j_session.run("CREATE INDEX address_id FOR (n:Address) ON (n.ID)");
    neo4j_session.run("CREATE INDEX major_name FOR (n:Major) ON (n.Name)");
    neo4j_session.run("CREATE INDEX instructor_id FOR (n:Instructor) ON (n.I
D)");
    neo4j_session.run("CREATE INDEX course_id FOR (n:Course) ON (n.ID)");

    for batch in batches(neo4j_student_data, 50000):
        cypherQP = """
        UNWIND $student_data AS row
        MERGE (std:Student {
                            ID: row.Student.ID,
                            FirstName: row.Student.FirstName,
                            LastName: row.Student.LastName,
                            Gender: row.Student.Gender,
                            Dob: row.Student.Dob,
                            Race: row.Student.Race,
                            Level: row.Student.Level
                            })
        MERGE (adr:Address {
                            Line: row.Address.Line,
                            City: row.Address.City,
                            State: row.Address.State,
                            ZipCode: row.Address.ZipCode,
                            Country: row.Address.Country,
                            Latitude: row.Address.Latitude,
                            Longitude: row.Address.Longitude
                            })
        MERGE (mjr:Major {Name: row.Major.Name})
        MERGE (grd:Graduation {
                                Semester: row.Graduation.Semester,
                                Year: row.Graduation.Year
                                })
        MERGE (std)-[rel1:MajorsIn]->(mjr)
        MERGE (std)-[rel2:LivesAt]->(adr)
        MERGE (std)-[rel3:GraduatesIn {Graduated: row.Student.Graduated}]->(gr
d)
        """
        neo4j_session.run(cypherQP, parameters={"student_data": batch});
#         break

    cypherQP = """
    UNWIND $instructor_data AS row
    MERGE (ins:Instructor {
                        ID: row.Instructor.ID,
                        FirstName: row.Instructor.FirstName,
                        LastName: row.Instructor.LastName
                        })
    """
    neo4j_session.run(cypherQP, parameters={"instructor_data": neo4j_instructo
r_data});

    cypherQP = """
    UNWIND $course_data AS row
```

```
    MERGE (crs:Course {
                        ID: row.Course.ID,
                        Name: row.Course.Name,
                        Credit: row.Course.Credit
                        })
    """
    neo4j_session.run(cypherQP, parameters={"course_data": neo4j_course_data
});

    for batch in batches(neo4j_enrollment_data, 50000):
        cypherQP = """
        UNWIND $enrollment_data AS row
        MATCH (crs:Course {ID: row.Course.ID})
        MATCH (ins:Instructor {ID: row.Instructor.ID})
        MATCH (std:Student {ID: row.Student.ID})
        MERGE (std)-[enl:Enrolls {
                                    AbsenceRate: row.Enrollment.AbsenceRate,
                                    Grade: row.Enrollment.Grade,
                                    Score: row.Enrollment.Score,
                                    Semester: row.Enrollment.Semester,
                                    Year: row.Enrollment.Year
                                    }]->(crs)
        MERGE (ins)-[tea:Teaches {
                                    Semester: row.Enrollment.Semester,
                                    Year: row.Enrollment.Year,
                                    Effectiveness: (.80 * row.Enrollment.Score
+ .20 * row.Enrollment.AbsenceRate)
                                    }]->(crs)
        """
        neo4j_session.run(cypherQP, parameters={"enrollment_data": batch});
#         break
```

In [19]:
```
# %%time
reset_neo4j()
dataload_neo4j()
```

Benchmark Data Load

In [296]:
```
# Ref: https://www.geeksforgeeks.org/graph-plotting-python-set-1/
# Ref: https://www.geeksforgeeks.org/graph-plotting-python-set-2/
# Ref: https://www.geeksforgeeks.org/graph-plotting-python-set-3/

# Select Plot Style
#plt.style.use('fivethirtyeight')
plt.style.use('ggplot')### Benchmark Data Load:

################################################################################
######################
def init_benchmark_data(title):
    benchmark_data = {
        "title": title,
        "label": ["Oracle", "MongoDB", "Neo4j"],
        "runtime": [],
        "result": []
    }
    return benchmark_data

# Ref: https://www.kite.com/python/answers/how-to-round-a-number-to-significan
t-digits-in-python#:~:text=Use%20round()%20to%20round,)))%20%2D%201)%20.
def round_sig(a_number, significant_digits = 4):
    rounded_number =  round(a_number, significant_digits - int(math.floor(math
.log10(abs(a_number)))) - 1)
    return rounded_number

def plot_benchmark_data(benchmark_data):

    # Calculate average
    for runtime in benchmark_data["runtime"]:
        sum = 0
        for val in runtime:
            sum += val
        runtime.append(sum/len(runtime))

    plt.subplots(figsize=(15,5))
    # init position
    pos = np.arange(len(benchmark_data["runtime"][1]))
    barWidth = 0.25

    for idx in range(len(benchmark_data["runtime"])):
        # Set position of bar on X axis
        pos = [x + barWidth for x in pos]
        # Make the plot
        rects = plt.bar(pos, benchmark_data["runtime"][idx], width = barWidth,
edgecolor ='white', label = benchmark_data["label"][idx])
        # Ref: https://matplotlib.org/3.3.3/gallery/lines_bars_and_markers/bar
chart.html#sphx-glr-gallery-lines-bars-and-markers-barchart-py
        """Attach a text label above each bar in *rects*, displaying its heigh
t."""
        for rect in rects:
            height = rect.get_height()
            plt.annotate('{}'.format(round_sig(height)),
                        xy=(rect.get_x() + rect.get_width() / 2, height),
                        xytext=(0, 3),  # 3 points vertical offset
                        textcoords="offset points",
```

```
                        ha='center', va='bottom')

    # Adding Xticks
    plt.xlabel('Execution', fontweight ='bold')
    plt.ylabel('Runtime in seconds', fontweight ='bold')
    plt.xticks([r + barWidth*2 for r in range(len(benchmark_data["runtime"][1
]))], ['first', 'Second', 'Third', 'Average'])
    plt.legend(benchmark_data["label"])
    plt.title(benchmark_data["title"])



    plt.show()
def print_benchmark_data(benchmark_data):
    results = benchmark_data.pop("result")
    runtimes = benchmark_data.pop("runtime")
    label = benchmark_data.pop("label")

    pprint.pprint(benchmark_data)
    print(pds.DataFrame(label))
    print(pds.DataFrame(runtimes))

    data = []
    for result in results: data.append(list(result))
    display = pds.DataFrame(data)
    return display

def benchmark_run(func, num_exe = 3):
    # Collects data for the benchmark
    runtime = []
    for idx in range(num_exe):
        begin = time.time() # store starting time
        result = func()
        end = time.time() # store end time
        runtime.append(end - begin)
    benchmark_data["runtime"].append(runtime)
    benchmark_data["result"].append(result)
##############################################################################
######################
```

In [297]:

```
# def test():
#     time.sleep(.1)
#     return [1000]
# ############################################################################
########################
# benchmark_data = init_benchmark_data("Data Load")

# benchmark_run(test, 3)
# benchmark_run(test, 3)
# benchmark_run(test, 3)

# plot_benchmark_data(benchmark_data)
# print_benchmark_data(benchmark_data)
# ############################################################################
########################
```

In [298]:
```
# ##############################################################################
########################
# benchmark_data = init_benchmark_data("Data Load")

# # datadel_oracle()
# # benchmark_run(dataload_oracle, 1)
# # datadel_mongodb()
# # benchmark_run(dataload_mongodb, 1)
# # datadel_neo4j()
# # benchmark_run(dataload_neo4j, 1)

# plot_benchmark_data(benchmark_data)
# print_benchmark_data(benchmark_data)
# ##############################################################################
########################
```

Queries Composition and Comparison

Query 1 - Which major has the most students retained after the first year?

Oracle:

In [394]:
```
def query_oracle():
    sql = """
        SELECT
            major_name,
            COUNT(student_id) AS "Count_STUDENT_ID"
        FROM
            (
                SELECT DISTINCT
                    COUNT(DISTINCT enrollment.year) AS "count_year",
                    enrollment.student_id,
                    student.major_name
                FROM
                         enrollment
                    INNER JOIN student ON student.id = enrollment.student_id
                GROUP BY
                    enrollment.student_id,
                    student.major_name
                HAVING
                    COUNT(DISTINCT enrollment.year) > 1
            )
        GROUP BY
            major_name
        ORDER BY
            "Count_STUDENT_ID" DESC
        FETCH FIRST 5 ROWS ONLY
    """
    result = oracle_cursor.execute(sql)
    return result
# print(list(query_oracle()))
```

MongoDB:

In [395]:

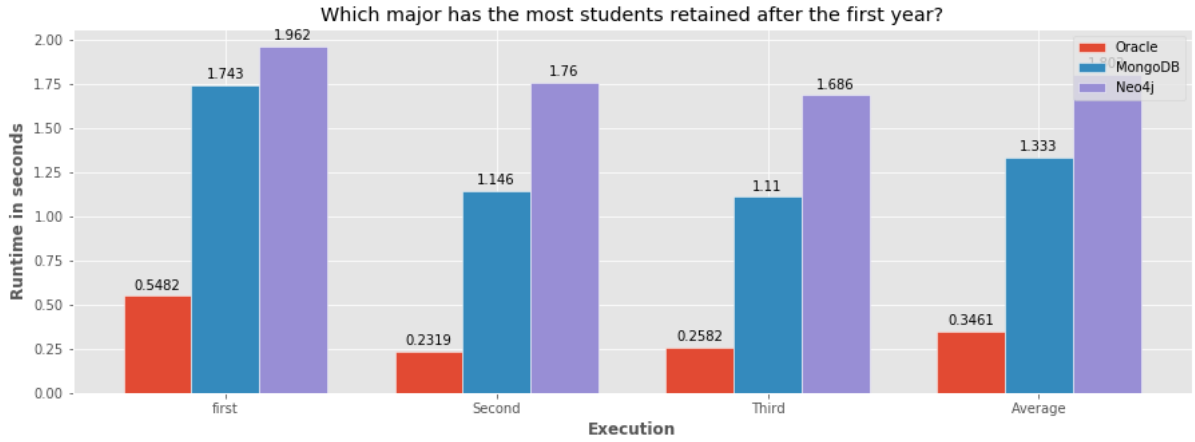
```
def query_mongodb():
    result = coll_enrollment.aggregate([
        {
            '$group': {
                '_id': {
                    'major': '$Student.Major.Name',
                    'student': '$Student.ID'
                },
                'years': {
                    '$addToSet': '$Year'
                }
            }
        }, {
            '$project': {
                'count_years': {
                    '$size': '$years'
                }
            }
        }, {
            '$match': {
                'count_years': {
                    '$gte': 2
                }
            }
        }, {
            '$group': {
                '_id': {
                    'major': '$_id.major'
                },
                'count_student': {
                    '$sum': 1
                }
            }
        }, {
            '$sort': {
                'count_student': -1
            }
        }, {
            '$limit': 5
        }
    ])
    return result
# print(list(query_mongodb()))
```

Neo4j:

In [396]:
```
def query_neo4j():
    cypherQ = """
        MATCH (mjr:Major)<-[mjn:MajorsIn]-(std:Student)-[enr:Enrolls]->(crs:Co
urse)
        WITH mjr, std, collect(DISTINCT enr.Year) as years
        WHERE size(years) > 1
        WITH mjr, count(std) AS student_count
        ORDER BY student_count DESC
        RETURN mjr.Name, student_count
        LIMIT 5
    """
    result = neo4j_session.run(cypherQ)
    return result
# pprint.pprint(list(query_neo4j()))
```

Comparison:

In [397]:
```
##########################################################################
######################
benchmark_data = init_benchmark_data("Which major has the most students retain
ed after the first year?")

benchmark_run(query_oracle)
benchmark_run(query_mongodb)
benchmark_run(query_neo4j)

plot_benchmark_data(benchmark_data)
print_benchmark_data(benchmark_data)
##########################################################################
######################
```




```
{'title': 'Which major has the most students retained after the first year?'}
        0
0  Oracle
1  MongoDB
2  Neo4j
          0         1         2         3
0  0.548161  0.231867  0.258194  0.346074
1  1.743407  1.146028  1.109854  1.333096
2  1.962085  1.759998  1.685862  1.802648
```

Out[397]:

	0	1	2	3	4
0	(Spanish Education, 1998)	(Athletic Training, 1944)	(Philosophy, 1941)	(Elementary Education, 1928)	(Physical Education, 1925)
1	{'_id': {'major': 'Spanish Education'}, 'count_student': 1998}	{'_id': {'major': 'Athletic Training'}, 'count_student': 1944}	{'_id': {'major': 'Philosophy'}, 'count_student': 1941}	{'_id': {'major': 'Elementary Education'}, 'count_student': 1928}	{'_id': {'major': 'Physical Education'}, 'count_student': 1925}
2	(Spanish Education, 1998)	(Athletic Training, 1944)	(Philosophy, 1941)	(Elementary Education, 1928)	(Physical Education, 1925)

Query 2 - What course has the highest percent of people with grade W?

Oracle:

In [398]:
```
def query_oracle():
    sql = """
SELECT
    course.name,
    ( COUNT(DISTINCT enrollment.student_id) / (
        SELECT
            COUNT(DISTINCT enrollment.student_id) AS "total"
        FROM
                  enrollment
            INNER JOIN course ON enrollment.course_id = course.id
        WHERE
            enrollment.grade LIKE 'W'
    ) ) * 100 AS "percentage"
FROM
         enrollment
    INNER JOIN course ON enrollment.course_id = course.id
WHERE
    enrollment.grade LIKE 'W'
GROUP BY
    course.name
ORDER BY
    "percentage" DESC
FETCH FIRST 5 ROWS ONLY
    """
    result = oracle_cursor.execute(sql)
    return result
```

MongoDB:

In [399]:

```
def query_mongodb():
    result = coll_enrollment.aggregate([
    {
        '$match': {
            'Grade': {
                '$eq': 'W'
            }
        }
    }, {
        '$facet': {
            'courses': [
                {
                    '$group': {
                        '_id': {
                            'course': '$Course.Name'
                        },
                        'count': {
                            '$sum': 1
                        }
                    }
                }, {
                    '$sort': {
                        'count': -1
                    }
                }, {
                    '$limit': 5
                }
            ],
            'totals': [
                {
                    '$group': {
                        '_id': None,
                        'total': {
                            '$sum': 1
                        }
                    }
                }
            ]
        }
    }, {
        '$project': {
            'courses': 1,
            'total': {
                '$arrayElemAt': [
                    '$totals', 0
                ]
            }
        }
    }, {
        '$project': {
            'courses': {
                '$map': {
                    'input': '$courses',
                    'as': 'course',
                    'in': [
                        {
```

```
                            'state': '$$course._id.course'
                        }, {
                            'percentage': {
                                '$multiply': [
                                    {
                                        '$divide': [
                                            '$$course.count', '$total.total'
                                        ]
                                    }, 100
                                ]
                            }
                        }
                    ]
                }
            }
        }
    }
])
    return result
```

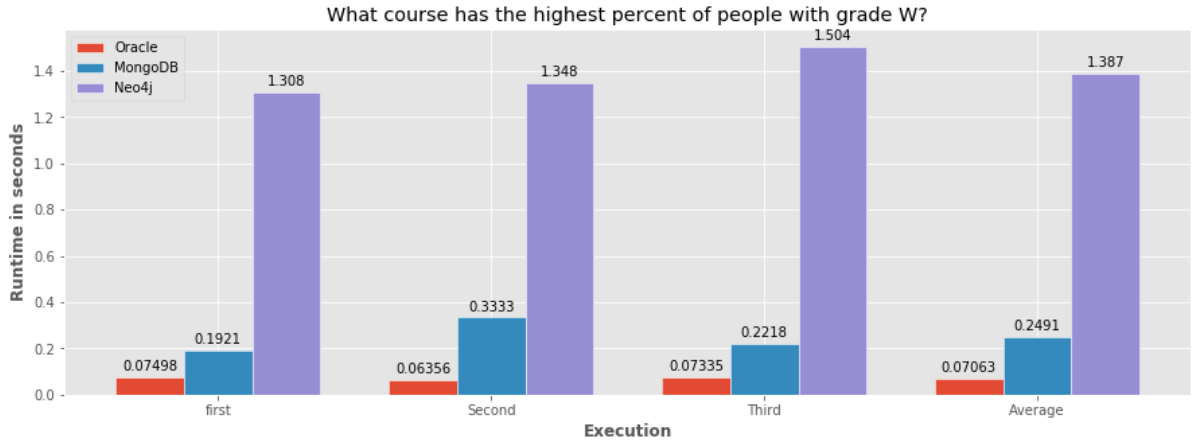
Neo4j:

In [400]:
```
def query_neo4j():
    cypherQ = """
        MATCH path = (std:Student)-[enr:Enrolls {Grade:"W"}]->(crs:Course)
        WITH count(path) AS total

        MATCH path = (std:Student)-[enr:Enrolls {Grade:"W"}]->(crs:Course)
        WITH crs, total, count(path) AS count
        ORDER BY count DESC
        LIMIT 5

        RETURN crs.Name, (toFloat(count)/total)*100 AS percentage
    """
    result = neo4j_session.run(cypherQ)
    return result
```

Comparison:

In [401]:

```
################################################################################
######################
benchmark_data = init_benchmark_data("What course has the highest percent of p
eople with grade W?")

benchmark_run(query_oracle)
benchmark_run(query_mongodb)
benchmark_run(query_neo4j)

plot_benchmark_data(benchmark_data)
print_benchmark_data(benchmark_data)
################################################################################
######################
```




```
{'title': 'What course has the highest percent of people with grade W?'}
          0
0  Oracle
1  MongoDB
2  Neo4j
          0         1         2         3
0  0.074977  0.063559  0.073347  0.070628
1  0.192114  0.333345  0.221769  0.249076
2  1.308122  1.347869  1.504276  1.386756
```

Out[401]:

	0	1	2	3	
0	(CLIN ASSESS II, 0.5625935969563282)	(NUM ALG GEO T&L, 0.5504512891083498)	(AGRI FINANCE, 0.5383089812603715)	(SERV PROD MKTG, 0.5261666734123932)	(INT 0.5
1	{'courses': [[{'state': 'CLIN ASSESS II'}, {'percentage': 0.48693337069992293}], [{'state': 'NUM ALG GEO T&L'}, {'percentage': 0.47642401737546414}], [{'state': 'AGRI FINANCE'}, {'percentage': 0.469417781825825}], [{'state': 'SERV PROD MKTG'}, {'percentage': 0.4554053107265466}], [{'state': 'STRUCTURAL GEOL'}, {'percentage': 0.4483990751769074}]]}	None	None	None	
2	(CLIN ASSESS II, 0.48693337069992293)	(NUM ALG GEO T&L, 0.47642401737546414)	(AGRI FINANCE, 0.469417781825825)	(SERV PROD MKTG, 0.4554053107265466)	(INT 0.44

Query 3 - What's a 4-year graduation rate among the female students, grouped by majors?

Oracle:

In [402]:
```
def query_oracle():
    sql = """
SELECT
    student.major_name,
    COUNT(student.id) AS "Count_ID"
FROM
    student
WHERE
    student.gender LIKE 'Female'
    AND student.year BETWEEN ( 2020 - 3 ) AND 2020
    AND student.graduated LIKE '1'
GROUP BY
    student.major_name
ORDER BY
    "Count_ID" DESC

    """
    result = oracle_cursor.execute(sql)
    return result
```

MongoDB:

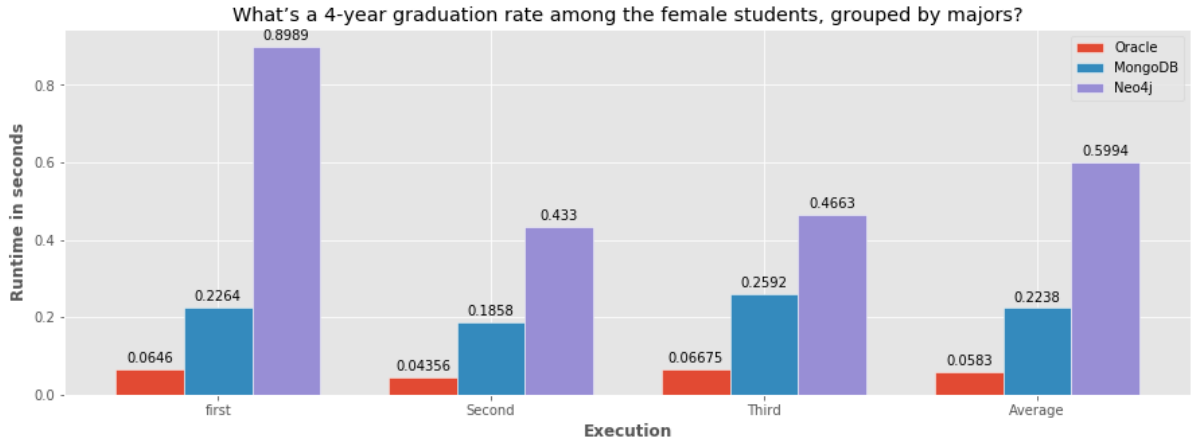
In [403]:
```
def query_mongodb():
    result = coll_student.aggregate([
    {
        '$match': {
            'Gender': 'Female',
            'Graduated': True
        }
    }, {
        '$project': {
            'Graduation': 1,
            'Major': 1,
            'match': {
                '$in': [
                    '$Graduation.Year', {
                        '$range': [
                            2020 - 3, 2020, 1
                        ]
                    }
                ]
            }
        }
    }, {
        '$match': {
            'match': True
        }
    }, {
        '$group': {
            '_id': {
                'major': '$Major.Name'
            },
            'count_student': {
                '$sum': 1
            }
        }
    }, {
        '$sort': {
            'count_student': -1
        }
    }
])
    return result
```

Neo4j:

In [404]:
```
def query_neo4j():
    cypherQ = """
        WITH 2020 AS late_grd_year
        MATCH (grd:Graduation)
        WHERE grd.Year IN range(late_grd_year-3, late_grd_year, 1)
        MATCH (mjr:Major)<-[mjn:MajorsIn]-(std:Student {Gender: "Female"})-[gr
n:GraduatesIn {Graduated: true}]->(grd)
        WITH mjr.Name AS major, count(std) AS student_count
        ORDER BY student_count DESC
        RETURN major, student_count
    """
    result = neo4j_session.run(cypherQ)
    return result
```

Comparison:

In [405]:
```
##########################################################################
######################
benchmark_data = init_benchmark_data("What’s a 4-year graduation rate among th
e female students, grouped by majors?")

benchmark_run(query_oracle)
benchmark_run(query_mongodb)
benchmark_run(query_neo4j)

plot_benchmark_data(benchmark_data)
print_benchmark_data(benchmark_data)
##########################################################################
######################
```




```
{'title': 'What’s a 4-year graduation rate among the female students, grouped
'
          'by majors?'}
          0
0  Oracle
1  MongoDB
2  Neo4j
          0         1         2         3
0  0.064601  0.043560  0.066747  0.058302
1  0.226381  0.185796  0.259163  0.223780
2  0.898915  0.432986  0.466252  0.599384
```

Out[405]:

	0	1	2	3	4	5
0	(Middle Level Education, 196)	(Physical Education, 192)	(History, 187)	(Spanish Education, 186)	(Dietetics, 183)	(Philosophy, 180)
1	{'_id': {'major': 'Physical Education'}, 'count_student': 160}	{'_id': {'major': 'Middle Level Education'}, 'count_student': 155}	{'_id': {'major': 'Information technology'}, 'count_student': 142}	{'_id': {'major': 'Exercise Science'}, 'count_student': 141}	{'_id': {'major': 'Spanish Education'}, 'count_student': 139}	{'_id': {'major': 'Biology'}, 'count_student': 138} '
2	(Middle Level Education, 196)	(Physical Education, 192)	(History, 187)	(Spanish Education, 186)	(Dietetics, 183)	(Philosophy, 180)

3 rows × 30 columns

Query 4 - Rank all courses in terms of their popularity (cumulated number enrollments).

Oracle:

In [406]:
```
def query_oracle():
    sql = """
SELECT
    course.name,
    COUNT(*) AS "Count_Enrollments"
FROM
         course
    INNER JOIN enrollment ON course.id = enrollment.course_id
GROUP BY
    course.name
ORDER BY
    "Count_Enrollments" DESC

    """
    result = oracle_cursor.execute(sql)
    return result
```

MongoDB:

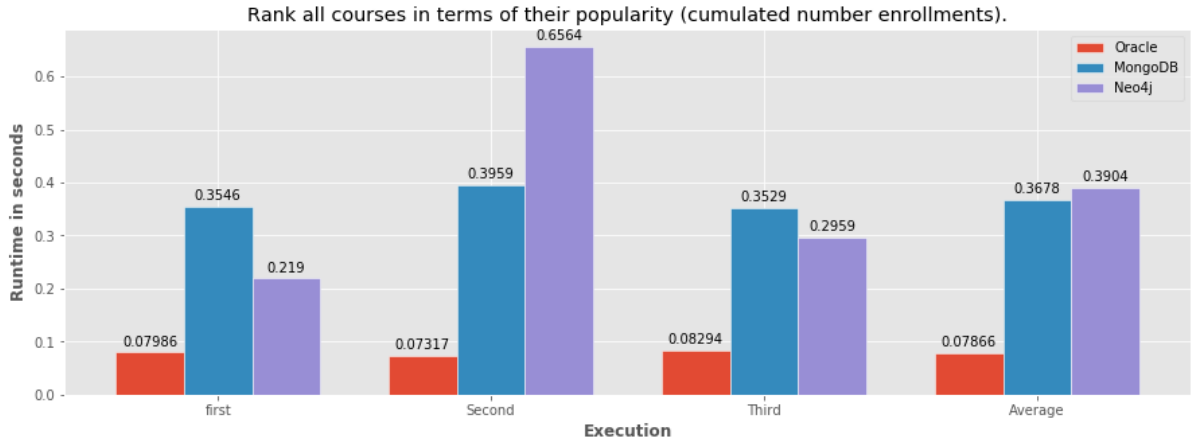
In [407]:
```
def query_mongodb():
    result = coll_enrollment.aggregate([
    {
        '$group': {
            '_id': {
                'course': '$Course.Name'
            },
            'count_enrollment': {
                '$sum': 1
            }
        }
    }, {
        '$sort': {
            'count_enrollment': -1
        }
    }
])

    return result
```

Neo4j:

In [408]:
```
def query_neo4j():
    cypherQ = """
        MATCH (std:Student)-[enr:Enrolls]->(crs:Course)
WITH crs.Name AS name_crs, count(enr) As num_enr
ORDER BY num_enr DESC
RETURN name_crs, num_enr

    """
    result = neo4j_session.run(cypherQ)
    return result
```

Comparison:

In [409]:
```
##########################################################################
######################
benchmark_data = init_benchmark_data("Rank all courses in terms of their popul
arity (cumulated number enrollments).")

benchmark_run(query_oracle)
benchmark_run(query_mongodb)
benchmark_run(query_neo4j)

plot_benchmark_data(benchmark_data)
print_benchmark_data(benchmark_data)
##########################################################################
######################
```




```
{'title': 'Rank all courses in terms of their popularity (cumulated number '
          'enrollments).'}
          0
0  Oracle
1  MongoDB
2  Neo4j
          0         1         2         3
0  0.079857  0.073166  0.082944  0.078656
1  0.354577  0.395923  0.352933  0.367811
2  0.218989  0.656381  0.295936  0.390435
```

Out[409]:

	0	1	2	3	4	
0	(CLIN ASSESS II, 828)	(UNIV PHYS II, 826)	(STRUCTURAL GEOL, 820)	(SYNOPTIC MET II, 814)	(CALC AN GEOM II, 807)	(TH
1	{'_id': {'course': 'CLIN ASSESS II'}, 'count_enrollment': 828}	{'_id': {'course': 'UNIV PHYS II'}, 'count_enrollment': 826}	{'_id': {'course': 'STRUCTURAL GEOL'}, 'count_enrollment': 820}	{'_id': {'course': 'SYNOPTIC MET II'}, 'count_enrollment': 814}	{'_id': {'course': 'CALC AN GEOM II'}, 'count_enrollment': 807}	'cou
2	(CLIN ASSESS II, 828)	(UNIV PHYS II, 826)	(STRUCTURAL GEOL, 820)	(SYNOPTIC MET II, 814)	(CALC AN GEOM II, 807)	(TH

3 rows × 267 columns

Query 5 - What is the trend of enrollment over the years?

Oracle:

In [410]:
```
def query_oracle():
    sql = """
SELECT
    COUNT(*) AS "Count_Enrollments",
    enrollment.year
FROM
    enrollment
GROUP BY
    enrollment.year
ORDER BY
    "Count_Enrollments" DESC

    """
    result = oracle_cursor.execute(sql)
    return result
```

MongoDB:

In [411]:
```
def query_mongodb():
    result = coll_enrollment.aggregate([
    {
        '$group': {
            '_id': {
                'year': '$Year'
            },
            'count_enrollment': {
                '$sum': 1
            }
        }
    }, {
        '$sort': {
            'count_enrollment': -1
        }
    }
])

    return result
```

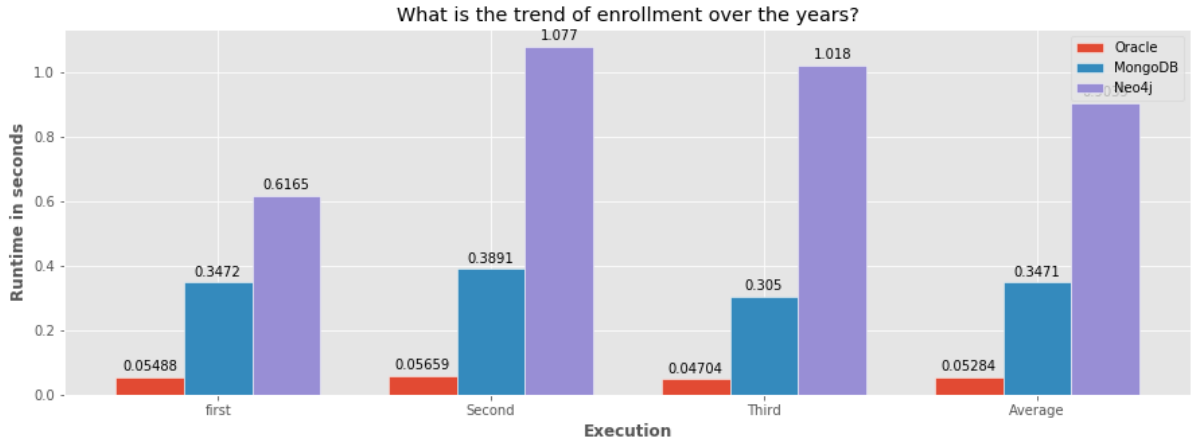
Neo4j:

In [412]:

```
def query_neo4j():
    cypherQ = """
        MATCH (std:Student)-[enr:Enrolls]->(crs:Course)
With enr.Year as year, count(enr) AS enr_count
ORDER BY enr_count DESC
RETURN year, enr_count

    """
    result = neo4j_session.run(cypherQ)
    return result
```

Comparison:

In [413]:
```
##########################################################################
######################
benchmark_data = init_benchmark_data("What is the trend of enrollment over the
years?")

benchmark_run(query_oracle)
benchmark_run(query_mongodb)
benchmark_run(query_neo4j)

plot_benchmark_data(benchmark_data)
print_benchmark_data(benchmark_data)
##########################################################################
######################
```




```
{'title': 'What is the trend of enrollment over the years?'}
         0
0  Oracle
1  MongoDB
2  Neo4j
          0         1         2         3
0  0.054877  0.056593  0.047038  0.052836
1  0.347185  0.389113  0.304992  0.347097
2  0.616533  1.077328  1.017729  0.903863
```

Out[413]:

	0	1	2	3	4
0	(10106, 2013)	(10085, 2012)	(10034, 2020)	(10019, 2019)	(10013, 2014)
1	{'_id': {'year': 2013}, 'count_enrollment': 10106}	{'_id': {'year': 2012}, 'count_enrollment': 10085}	{'_id': {'year': 2020}, 'count_enrollment': 10034}	{'_id': {'year': 2019}, 'count_enrollment': 10019}	{'_id': {'year': 2014}, 'count_enrollment': 10013}
2	(2013, 10106)	(2012, 10085)	(2020, 10034)	(2019, 10019)	(2014, 10013)

3 rows × 21 columns

Query 6 - How does the number of full-time students vs part-time students compare for both graduate and undergraduate students?

Oracle:

In [414]:

```
def query_oracle():
    sql = """
SELECT
    'Full-time'             AS type,
    gradlevel,
    COUNT("Sum_CREDIT")     AS "Count_Sum_CREDIT"
FROM
    (
        SELECT
            student.id,
            SUM(course.credit) AS "Sum_CREDIT",
            student.gradlevel
        FROM
                 student
            INNER JOIN enrollment ON student.id = enrollment.student_id
            INNER JOIN course ON enrollment.course_id = course.id
        WHERE
                enrollment.year = 2020
            AND enrollment.semester LIKE 'Spring'
        GROUP BY
            student.id,
            student.gradlevel
        HAVING
            SUM(course.credit) >= 9
    )
GROUP BY
    'Full-time',
    gradlevel
UNION
SELECT
    'Part-time'             AS type,
    gradlevel,
    COUNT("Sum_CREDIT")     AS "Count_Sum_CREDIT"
FROM
    (
        SELECT
            student.id,
            SUM(course.credit) AS "Sum_CREDIT",
            student.gradlevel
        FROM
                 student
            INNER JOIN enrollment ON student.id = enrollment.student_id
            INNER JOIN course ON enrollment.course_id = course.id
        WHERE
                enrollment.year = 2020
            AND enrollment.semester LIKE 'Spring'
        GROUP BY
            student.id,
            student.gradlevel
        HAVING
            SUM(course.credit) < 9
    )
GROUP BY
    'Part-time',
    gradlevel
```

```
    """
    result = oracle_cursor.execute(sql)
    return result
```

MongoDB:

In [415]:

```
def query_mongodb():
    result = coll_enrollment.aggregate([
    {
        '$match': {
            'Year': 2020,
            'Semester': 'Spring'
        }
    }, {
        '$group': {
            '_id': '$Student.ID',
            'total_credit': {
                '$sum': '$Course.Credit'
            }
        }
    }, {
        '$facet': {
            'full_time': [
                {
                    '$match': {
                        'total_credit': {
                            '$gte': 9
                        }
                    }
                }, {
                    '$lookup': {
                        'from': 'Student',
                        'localField': '_id',
                        'foreignField': 'ID',
                        'as': 'student'
                    }
                }, {
                    '$project': {
                        'type': 'Full-time',
                        'total_creddit': 1,
                        'student': {
                            '$arrayElemAt': [
                                '$student', 0
                            ]
                        }
                    }
                }, {
                    '$group': {
                        '_id': {
                            'level': '$student.Level'
                        },
                        'count_student': {
                            '$sum': 1
                        }
                    }
                }
            ],
            'part_time': [
                {
                    '$match': {
                        'total_credit': {
                            '$lt': 9
```

```
                        }
                    }
                }, {
                    '$lookup': {
                        'from': 'Student',
                        'localField': '_id',
                        'foreignField': 'ID',
                        'as': 'student'
                    }
                }, {
                    '$project': {
                        'type': 'Full-time',
                        'total_creddit': 1,
                        'student': {
                            '$arrayElemAt': [
                                '$student', 0
                            ]
                        }
                    }
                }, {
                    '$group': {
                        '_id': {
                            'level': '$student.Level'
                        },
                        'count_student': {
                            '$sum': 1
                        }
                    }
                }
            ]
        }
    }
])

    return result
```

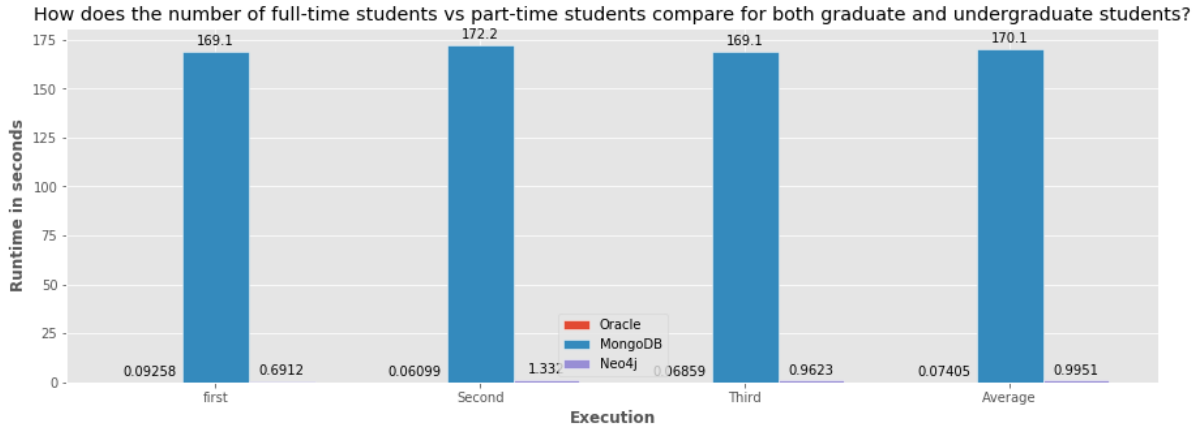
Neo4j:

In [416]:

```
def query_neo4j():
    cypherQ = """
        WITH 2020 AS enr_year, 'Spring' AS enr_semester
        MATCH (std:Student)-[enr:Enrolls {Year: enr_year, Semester: enr_semest
er}]->(crs:Course)
        WITH std, sum(crs.Credit) AS total_credit
        WHERE total_credit >= 9
        WITH std.Level AS student_level, count(*) AS student_count
        RETURN "Full-time" AS type, student_level, student_count
    """
    result = neo4j_session.run(cypherQ)

    cypherQ = """
        WITH 2020 AS enr_year, 'Spring' AS enr_semester
        MATCH (std:Student)-[enr:Enrolls {Year: enr_year, Semester: enr_semest
er}]->(crs:Course)
        WITH std, sum(crs.Credit) AS total_credit
        WHERE total_credit < 9
        WITH std.Level AS student_level, count(*) AS student_count
        RETURN "Part-time" AS type, student_level, student_count
    """
    result = neo4j_session.run(cypherQ)
    return result
```

Comparison:

In [417]:
```
##########################################################################
######################
benchmark_data = init_benchmark_data("How does the number of full-time student
s vs part-time students compare for both graduate and undergraduate students?"
)

benchmark_run(query_oracle)
benchmark_run(query_mongodb)
benchmark_run(query_neo4j)

plot_benchmark_data(benchmark_data)
print_benchmark_data(benchmark_data)
##########################################################################
######################
```




```
{'title': 'How does the number of full-time students vs part-time students '
          'compare for both graduate and undergraduate students?'}
         0
0  Oracle
1  MongoDB
2  Neo4j
             0           1           2           3
0  0.092578    0.060987    0.068591    0.074052
1  169.069020  172.194228  169.139583  170.134277
2  0.691152    1.331744    0.962342    0.995079
```

Out[417]:

	0	1	2	3
0	(Full-time, Graduate, 4)	(Full-time, Undergraduate, 5)	(Part-time, Graduate, 1601)	(Part-time, Undergraduate, 1692)
1	{'full_time': [{'_id': {'level': 'Undergraduate'}, 'count_student': 5}, {'_id': {'level': 'Graduate'}, 'count_student': 4}], 'part_time': [{'_id': {'level': 'Graduate'}, 'count_student': 1601}, {'_id': {'level': 'Undergraduate'}, 'count_student': 1692}]}	None	None	None
2	(Part-time, Graduate, 1601)	(Part-time, Undergraduate, 1692)	None	None

Query 7 - What are the percentages of female vs male students in STEM programs?

Oracle:

In [418]:
```
def query_oracle():
    sql = """
SELECT
    student.gender,
    (COUNT(student.id) / (
        SELECT
            COUNT(student.id) AS "total"
        FROM
            student
        WHERE
            student.major_name IN (
                'Astronomy',
                'Biology',
                'Chemistry',
                'Computer science',
                'Engineering',
                'Earth sciences',
                'Health sciences',
                'Information technology',
                'Mathematics',
                'Physics'
            )
    ))*100 AS "percentage"
FROM
    student
WHERE
    student.major_name IN (
        'Astronomy',
        'Biology',
        'Chemistry',
        'Computer science',
        'Engineering',
        'Earth sciences',
        'Health sciences',
        'Information technology',
        'Mathematics',
        'Physics'
    )
GROUP BY
    student.gender

    """
    result = oracle_cursor.execute(sql)
    return result
```

MongoDB:

In [419]:

```
def query_mongodb():
    result = coll_student.aggregate([
    {
        '$project': {
            'Gender': 1,
            'stem': {
                '$in': [
                    '$Major.Name', [
                        'Astronomy', 'Biology', 'Chemistry', 'Computer scienc
e', 'Engineering', 'Earth sciences', 'Health sciences', 'Information technolog
y', 'Mathematics', 'Physics'
                    ]
                ]
            }
        }
    }, {
        '$match': {
            'stem': True
        }
    }, {
        '$facet': {
            'genders': [
                {
                    '$group': {
                        '_id': {
                            'gender': '$Gender'
                        },
                        'count_student': {
                            '$sum': 1
                        }
                    }
                }
            ],
            'total': [
                {
                    '$group': {
                        '_id': None,
                        'total': {
                            '$sum': 1
                        }
                    }
                }
            ]
        }
    }, {
        '$project': {
            'gender1': {
                '$arrayElemAt': [
                    '$genders', 0
                ]
            },
            'gender2': {
                '$arrayElemAt': [
                    '$genders', 1
                ]
            },
```

```
            'total': {
                '$arrayElemAt': [
                    '$total', 0
                ]
            }
        }
    }, {
        '$project': {
            'result': [
                {
                    'gender': '$gender1._id.gender',
                    'percentage': {
                        '$multiply': [
                            {
                                '$divide': [
                                    '$gender1.count_student', '$total.total'
                                ]
                            }, 100
                        ]
                    }
                }, {
                    'gender': '$gender2._id.gender',
                    'percentage': {
                        '$multiply': [
                            {
                                '$divide': [
                                    '$gender2.count_student', '$total.total'
                                ]
                            }, 100
                        ]
                    }
                }
            ]
        }
    }
])

    return result
```

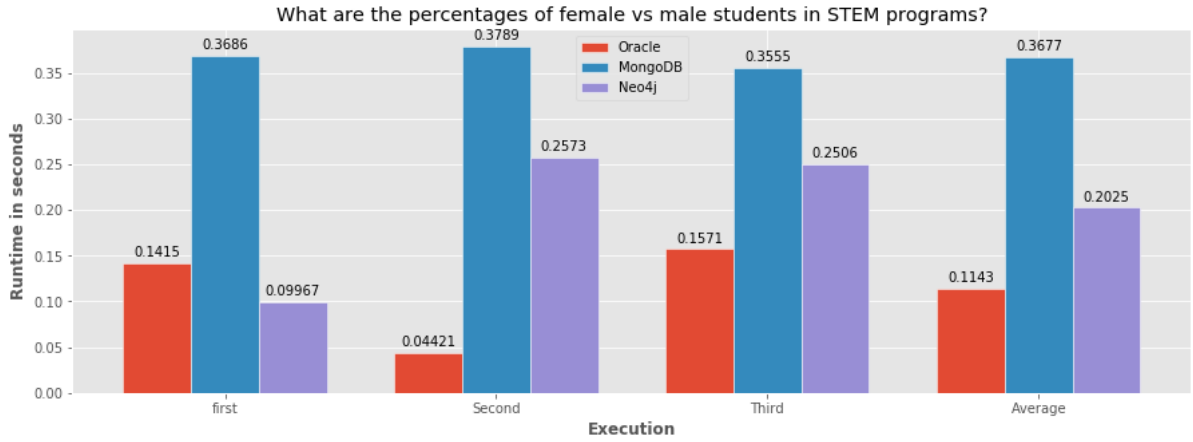
Neo4j:

In [420]:

```
def query_neo4j():
    cypherQ = """
        WITH ["Astronomy","Biology","Chemistry","Computer science","Engineerin
g","Earth sciences","Health sciences","Information technology","Mathematic
s","Physics"
] AS stem_majors
MATCH (mjr:Major)
WHERE mjr.Name IN stem_majors

MATCH (std:Student)-[mjn:MajorsIn]->(mjr)
WITH count(std) AS total

WITH ["Astronomy","Biology","Chemistry","Computer science","Engineering","Eart
h sciences","Health sciences","Information technology","Mathematics","Physics"
] AS stem_majors, total
MATCH (mjr:Major)
WHERE mjr.Name IN stem_majors

MATCH (std:Student)-[mjn:MajorsIn]->(mjr)
WITH std.Gender AS student_gender, (toFloat(count(std))/total)*100 AS percenta
ge

RETURN student_gender, percentage
    """
    result = neo4j_session.run(cypherQ)
    return result
```

Comparison:

In [421]:
```
##########################################################################
######################
benchmark_data = init_benchmark_data("What are the percentages of female vs ma
le students in STEM programs?")

benchmark_run(query_oracle)
benchmark_run(query_mongodb)
benchmark_run(query_neo4j)

plot_benchmark_data(benchmark_data)
print_benchmark_data(benchmark_data)
##########################################################################
######################
```




```
{'title': 'What are the percentages of female vs male students in STEM '
          'programs?'}
          0
0  Oracle
1  MongoDB
2  Neo4j
          0         1         2         3
0  0.141470  0.044206  0.157077  0.114251
1  0.368642  0.378913  0.355508  0.367688
2  0.099666  0.257316  0.250568  0.202517
```

Out[421]:

	0	1
0	(Male, 49.903649283391545)	(Female, 50.096350716608455)
1	{'result': [{'gender': 'Female', 'percentage': 50.09635071660845}, {'gender': 'Male', 'percentage': 49.903649283391545}]}	None
2	(Male, 49.903649283391545)	(Female, 50.09635071660845)

Query 8 - What is the distribution of students with respect to their race?

Oracle:

In [422]:
```
def query_oracle():
    sql = """
SELECT
    student.race,
    ( COUNT(student.id) / (
        SELECT
            COUNT(student.id) AS "total"
        FROM
            student
    ) ) * 100 AS "percentage"
FROM
    student
GROUP BY
    student.race
ORDER BY
    "percentage" DESC

    """
    result = oracle_cursor.execute(sql)
    return result
```

MongoDB:

In [423]:

```
def query_mongodb():
    result = coll_student.aggregate([
    {
        '$facet': {
            'races': [
                {
                    '$group': {
                        '_id': {
                            'race': '$Race'
                        },
                        'count_student': {
                            '$sum': 1
                        }
                    }
                }
            ],
            'total': [
                {
                    '$group': {
                        '_id': None,
                        'total': {
                            '$sum': 1
                        }
                    }
                }, {
                    '$project': {
                        '_id': 0,
                        'total': 1
                    }
                }
            ]
        }
    }, {
        '$project': {
            'races': 1,
            'total': {
                '$arrayElemAt': [
                    '$total', 0
                ]
            }
        }
    }, {
        '$project': {
            'races': {
                '$map': {
                    'input': '$races',
                    'as': 'race',
                    'in': [
                        {
                            'race': '$$race._id.race'
                        }, {
                            'percentage': {
                                '$multiply': [
                                    {
                                        '$divide': [
                                            '$$race.count_student', '$total.to
```

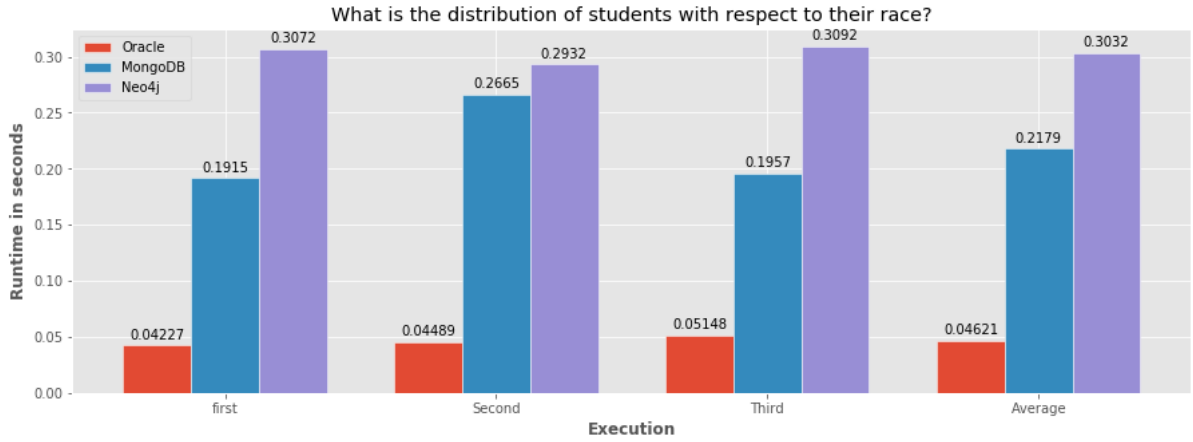
```
tal'
                                        ]
                                    }, 100
                                ]
                            }
                        }
                    ]
                }
            }
        }
    }
])

    return result
```

Neo4j:

In [424]:
```
def query_neo4j():
    cypherQ = """
        MATCH (std: Student)
WITH count(*) AS total
MATCH (std: Student)
WITH std.Race as race, (toFloat(count(std))/total)*100 AS per
ORDER BY per DESC
RETURN race, per

    """
    result = neo4j_session.run(cypherQ)
    return result
```

Comparison:

In [425]:
```
##########################################################################
######################
benchmark_data = init_benchmark_data("What is the distribution of students wit
h respect to their race?")

benchmark_run(query_oracle)
benchmark_run(query_mongodb)
benchmark_run(query_neo4j)

plot_benchmark_data(benchmark_data)
print_benchmark_data(benchmark_data)
##########################################################################
######################
```




```
{'title': 'What is the distribution of students with respect to their race?'}
          0
0  Oracle
1  MongoDB
2  Neo4j
          0         1         2         3
0  0.042268  0.044887  0.051484  0.046213
1  0.191484  0.266478  0.195736  0.217899
2  0.307173  0.293214  0.309173  0.303187
```

Out[425]:

	0	1	2	3	4
0	(Native Hawaiian, 20.205)	(Black, 19.986)	(Asian, 19.957)	(American Indian, 19.938)	(White, 19.914)
1	{'races': [[{'race': 'American Indian'}, {'percentage': 19.938}], [{'race': 'Black'}, {'percentage': 19.986}], [{'race': 'Asian'}, {'percentage': 19.957}], [{'race': 'Native Hawaiian'}, {'percentage': 20.205000000000002}], [{'race': 'White'}, {'percentage': 19.914}]]}	None	None	None	None
2	(Native Hawaiian, 20.205000000000002)	(Black, 19.986)	(Asian, 19.957)	(American Indian, 19.938)	(White, 19.914)

Query 9 - What is the percentage of enrollment per state in the USA?

Oracle:

In [426]:
```
def query_oracle():
    sql = """
SELECT
    address.state,
    ( COUNT(*) / (
        SELECT
            COUNT(*) AS "total"
        FROM
                  student
            INNER JOIN live ON student.id = live.student_id
            INNER JOIN address ON live.address_id = address.id
        WHERE
            address.country LIKE 'United States'
    ) ) * 100 AS "percentage"
FROM
         student
    INNER JOIN live ON student.id = live.student_id
    INNER JOIN address ON live.address_id = address.id
WHERE
    address.country LIKE 'United States'
GROUP BY
    address.state
ORDER BY
    "percentage" DESC

    """
    result = oracle_cursor.execute(sql)
    return result
```

MongoDB:

In [427]:

```
def query_mongodb():
    result = coll_student.aggregate([
    {
        '$match': {
            'Address.Country': 'United States'
        }
    }, {
        '$facet': {
            'states': [
                {
                    '$group': {
                        '_id': {
                            'state': '$Address.State'
                        },
                        'count': {
                            '$sum': 1
                        }
                    }
                }
            ],
            'total': [
                {
                    '$group': {
                        '_id': None,
                        'total': {
                            '$sum': 1
                        }
                    }
                }
            ]
        }
    }, {
        '$project': {
            'states': 1,
            'total': {
                '$arrayElemAt': [
                    '$total', 0
                ]
            }
        }
    }, {
        '$project': {
            'states': {
                '$map': {
                    'input': '$states',
                    'as': 'state',
                    'in': [
                        {
                            'state': '$$state._id.state'
                        }, {
                            'percentage': {
                                '$multiply': [
                                    {
                                        '$divide': [
                                            '$$state.count', '$total.total'
                                        ]
```

```
                                    }, 100
                                ]
                            }
                        }
                    ]
                }
            }
        }
    }
])

    return result
```

Neo4j:

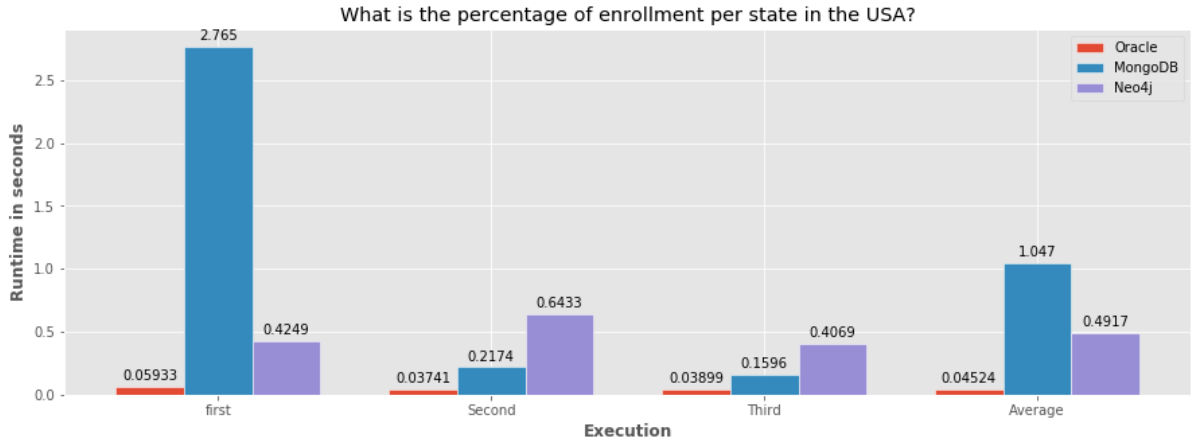
In [428]:

```
def query_neo4j():
    cypherQ = """
        MATCH (adr:Address {Country:"United States"})<-[lvt:LivesAt]-(std:Stud
ent)
        WITH count(std) AS std_total
        MATCH (adr:Address {Country:"United States"})<-[lvt:LivesAt]-(std:Stud
ent)
        WITH adr.State AS state, (toFloat(count(std))/std_total)*100 As std_pe
r
        RETURN state, std_per
    """
    result = neo4j_session.run(cypherQ)
    return result
```

Comparison:

In [429]:

```
##########################################################################
######################
benchmark_data = init_benchmark_data("What is the percentage of enrollment per
state in the USA?")

benchmark_run(query_oracle)
benchmark_run(query_mongodb)
benchmark_run(query_neo4j)

plot_benchmark_data(benchmark_data)
print_benchmark_data(benchmark_data)
##########################################################################
######################
```




```
{'title': 'What is the percentage of enrollment per state in the USA?'}
         0
0  Oracle
1  MongoDB
2  Neo4j
          0         1         2         3
0  0.059329  0.037415  0.038989  0.045244
1  2.764507  0.217422  0.159571  1.047167
2  0.424864  0.643278  0.406911  0.491684
```

Out[429]:

	0	1	2	3	
0	(California, 10.939627780299592)	(Texas, 10.894235133908307)	(Florida, 7.671357240127099)	(New York, 5.038583749432592)	4.1307
1	{'states': [[{'state': 'Nebraska'}, {'percentage': 0.9532455742169769}], [{'state': 'Arizona'}, {'percentage': 2.0880617339990923}], [{'state': 'Iowa'}, {'percentage': 1.0894235133908308}], [{'state': 'Alabama'}, {'percentage': 2.3604176123468}], [{'state': 'Georgia'}, {'percentage': 2.678166137085792}], [{'state': 'California'}, {'percentage': 10.939627780299592}], [{'state': 'Louisiana'}, {'percentage': 2.3150249659555153}], [{'state': 'Maryland'}, {'percentage': 0.9078529278256923}], [{'state': 'North Carolina'}, {'percentage': 1.7703132092601}], [{'state': 'Kentucky'}, {'percentage': 1.8157058556513845}], [{'state': 'Ohio'}, {'percentage': 4.1307308216069}], [{'state': 'Tennessee'}, {'percentage': 2.3150249659555153}], [{'state': 'Oklahoma'}, {'percentage': 1.9518837948252383}], [{'state': 'Alaska'}, {'percentage': 0.5447117566954154}], [{'state': 'South Dakota'}, {'percentage': 0.18157058556513844}], [{'state': 'Delaware'}, {'percentage': 0.45392646391284613}], [{'state': 'Connecticut'}, {'percentage': 1.1802088061734}], [{'state': 'Florida'}, {'percentage': 7.6713572401271}], [{'state': 'Texas'}, {'percentage': 10.894235133908307}], [{'state': 'Nevada'}, {'percentage': 1.6341352700862462}], [{'state': 'Mississippi'}, {'percentage':	None	None	None	

0	1	2	3
0.6808896958692692}], [{'state': 'Colorado'}, {'percentage': 2.0880617339990923}], [{'state': 'Wisconsin'}, {'percentage': 0.8624602814344077}], [{'state': 'Michigan'}, {'percentage': 1.7703132092601}], [{'state': 'West Virginia'}, {'percentage': 0.9532455742169769}], [{'state': 'Hawaii'}, {'percentage': 0.5901044030867}], [{'state': 'Arkansas'}, {'percentage': 0.6354970494779846}], [{'state': 'South Carolina'}, {'percentage': 0.7716749886518384}], [{'state': 'New Jersey'}, {'percentage': 1.0894235133908308}], [{'state': 'Missouri'}, {'percentage': 2.451202905129369}], [{'state': 'Idaho'}, {'percentage': 0.13617793917385385}], [{'state': 'Montana'}, {'percentage': 0.13617793917385385}], [{'state': 'Rhode Island'}, {'percentage': 0.13617793917385385}], [{'state': 'Utah'}, {'percentage': 0.5447117566954154}], [{'state': 'Virginia'}, {'percentage': 4.0853381752156155}], [{'state': 'Washington'}, {'percentage': 2.0426690876078077}], [{'state': 'District of Columbia'}, {'percentage': 3.313663186563777}], [{'state': 'New York'}, {'percentage': 5.038583749432592}], [{'state': 'Wyoming'}, {'percentage': 0.22696323195642307}], [{'state': 'North Dakota'}, {'percentage': 0.13617793917385385}], [{'state': 'Kansas'}, {'percentage': 1.4979573309123921}], [{'state': 'Minnesota'}, {'percentage': 1.8610985020426691}], [{'state': 'New Mexico'}, {'percentage': 0.8170676350431231}],			

	0	1	2	3
	[{'state': 'Illinois'}, {'percentage': 2.3150249659555153}], [{'state': 'Indiana'}, {'percentage': 1.8610985020426691}], [{'state': 'Oregon'}, {'percentage': 1.1802088061734}], [{'state': 'Pennsylvania'}, {'percentage': 3.4498411257376307}], [{'state': 'Massachusetts'}, {'percentage': 1.4525646845211075}]]}			
2	(New York, 5.038583749432592)	(Utah, 0.5447117566954154)	(Missouri, 2.451202905129369)	(Kansas, 1.4979573309123921)	7.6713

3 rows × 48 columns

Query 10 - Which instructor is most "effective" in the long run? – Average(80%*score*+*20%*absenceRate)

Oracle:

In [430]:
```
def query_oracle():
    sql = """
    SELECT
        instructor.id,
        instructor.firstname,
        instructor.lastname,
        AVG((enrollment.score * 0.8 + enrollment.absencerate * 0.2)) AS "eff"
    FROM
             instructor
        INNER JOIN enrollment ON instructor.id = enrollment.instructor_id
    GROUP BY
        instructor.id,
        instructor.firstname,
        instructor.lastname
    ORDER BY
        "eff" DESC
    FETCH FIRST 5 ROWS ONLY
    """
    result = oracle_cursor.execute(sql)
    return result
```

MongoDB:

In [431]:

```
def query_mongodb():
    result = coll_enrollment.aggregate([
    {
        '$project': {
            'Instructor': 1,
            'effectiveness': {
                '$add': [
                    {
                        '$multiply': [
                            '$Score', .80
                        ]
                    }, {
                        '$multiply': [
                            '$AbsenceRate', .20
                        ]
                    }
                ]
            }
        }
    }, {
        '$group': {
            '_id': '$Instructor.ID',
            'effectiveness': {
                '$avg': '$effectiveness'
            }
        }
    }, {
        '$lookup': {
            'from': 'Instructor',
            'localField': '_id',
            'foreignField': 'ID',
            'as': 'instructor'
        }
    }, {
        '$sort': {
            'effectiveness': -1
        }
    }, {
        '$limit': 5
    }
])

    return result
```

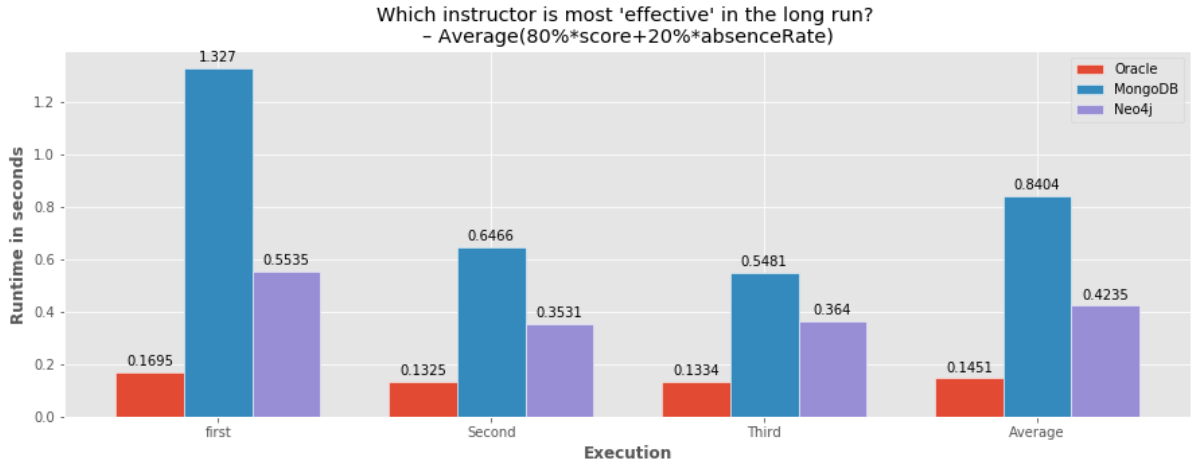
Neo4j:

In [432]:
```
def query_neo4j():
    cypherQ = """
        MATCH (ins:Instructor)-[tea:Teaches]->(crs:Course)
        WITH ins, avg(tea.Effectiveness) AS effectiveness
        ORDER BY effectiveness DESC
        RETURN ins, effectiveness
        LIMIT 5
    """
    result = neo4j_session.run(cypherQ)
    return result
```

Comparison:

In [433]:

```
##########################################################################
######################
benchmark_data = init_benchmark_data("""Which instructor is most 'effective' i
n the long run?
- Average(80%*score+20%*absenceRate)""")

benchmark_run(query_oracle)
benchmark_run(query_mongodb)
benchmark_run(query_neo4j)

plot_benchmark_data(benchmark_data)
print_benchmark_data(benchmark_data)
##########################################################################
######################
```




```
{'title': "Which instructor is most 'effective' in the long run? \n"
          '- Average(80%*score+20%*absenceRate)'}
          0
0  Oracle
1  MongoDB
2  Neo4j
          0         1         2         3
0  0.169549  0.132480  0.133399  0.145143
1  1.326696  0.646551  0.548093  0.840447
2  0.553522  0.353087  0.363992  0.423534
```

Out[433]:

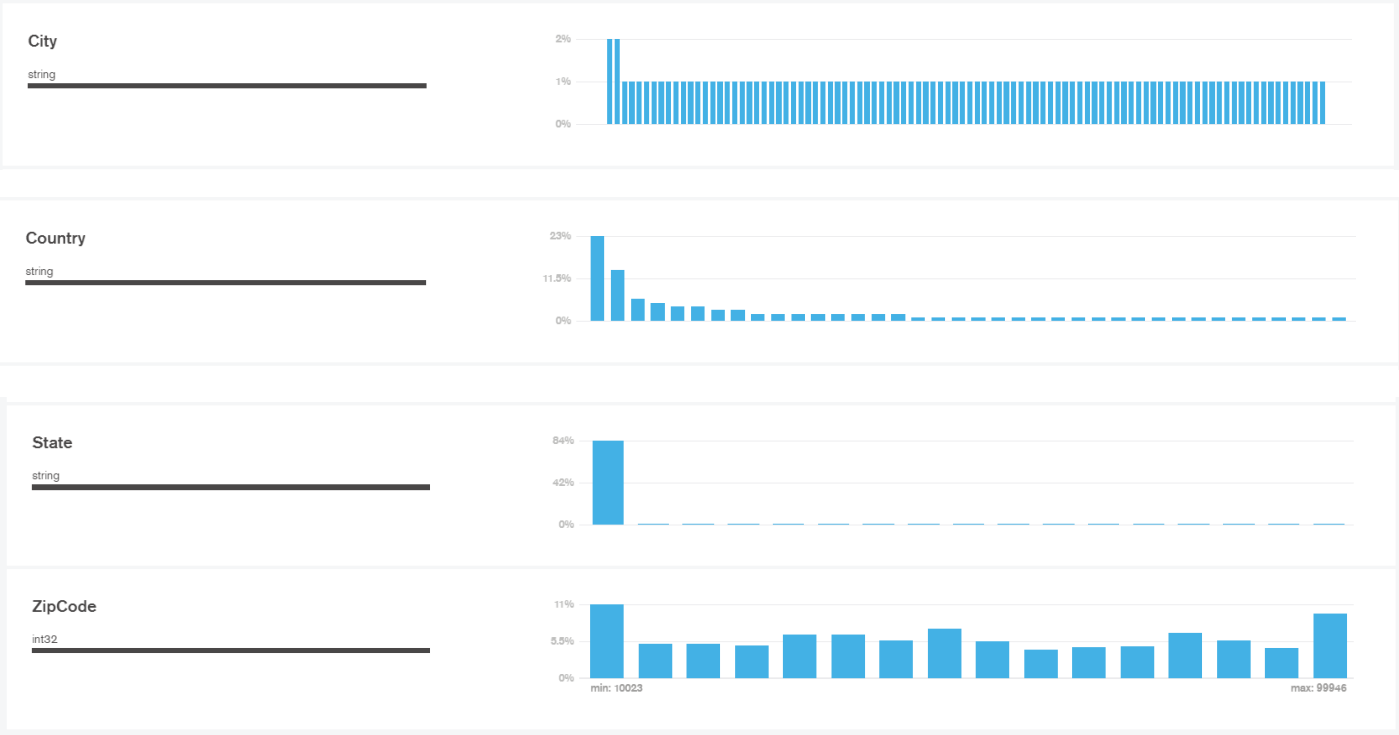
	0	1	2	
0	(8, Isiahi, Assad, 52.66811023622047)	(110, Estevan, Bompas, 52.00999000999001)	(126, Melania, Sweynson, 51.76083499005964)	(64, J 51.39
1	{'_id': 8, 'effectiveness': 52.66264566929134, 'instructor': [{'_id': 5fd52300d058deb9114e17fb, 'ID': 8, 'FirstName': 'Isiahi', 'LastName': 'Assad'}]}	{'_id': 110, 'effectiveness': 52.00517882117882, 'instructor': [{'_id': 5fd52300d058deb9114e1861, 'ID': 110, 'FirstName': 'Estevan', 'LastName': 'Bompas'}]}	{'_id': 126, 'effectiveness': 51.74983697813121, 'instructor': [{'_id': 5fd52300d058deb9114e1871, 'ID': 126, 'FirstName': 'Melania', 'LastName': 'Sweynson'}]}	{'_id': 6 51.4 5fd52300d05 'ID': 64, 'Firs 'LastNa
2	((LastName, ID, FirstName), 52.66264566929134)	((LastName, ID, FirstName), 52.005178821178845)	((LastName, ID, FirstName), 51.749836978131256)	((LastName 51.4

Observations:

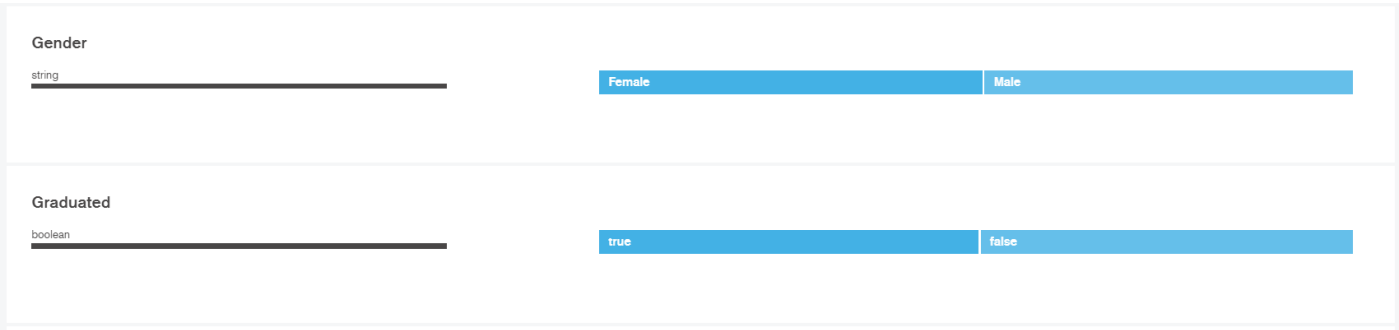
- It is important to note that all results are quite biased by many subjective factors such as my humble coding skills, modeling skills, and the fact that no personal optimizations at the index level were done. Also, the fact that all these databases are running on different build environments with varied performances.
- The population of the database took the longest on Neo4j, then MongoDB, and the least on Oracle. This result could be greatly influenced by the environment of the database and the fact MongoDB had to transit through a VPN setup needed for the Oracle database.
- The relatively better performance of MongoDB compared to Neo4j had to do with the flexibility of the schema allowing some collections to be embedded into others for faster queries. However, there is still some impact from the different environments we used, Neo4j being greatly affected.
- MongoDB aggregations turned out to be very adequate in streamlining the dataset from one stage to another but the faceted stage and lookup within MongoDB turned out to have poor performance for the runtime.
- The relational database, Oracle, overall had a better performance which can be to the environment as well, and the quite none complex join and single table queries that were performed.

Extra Features

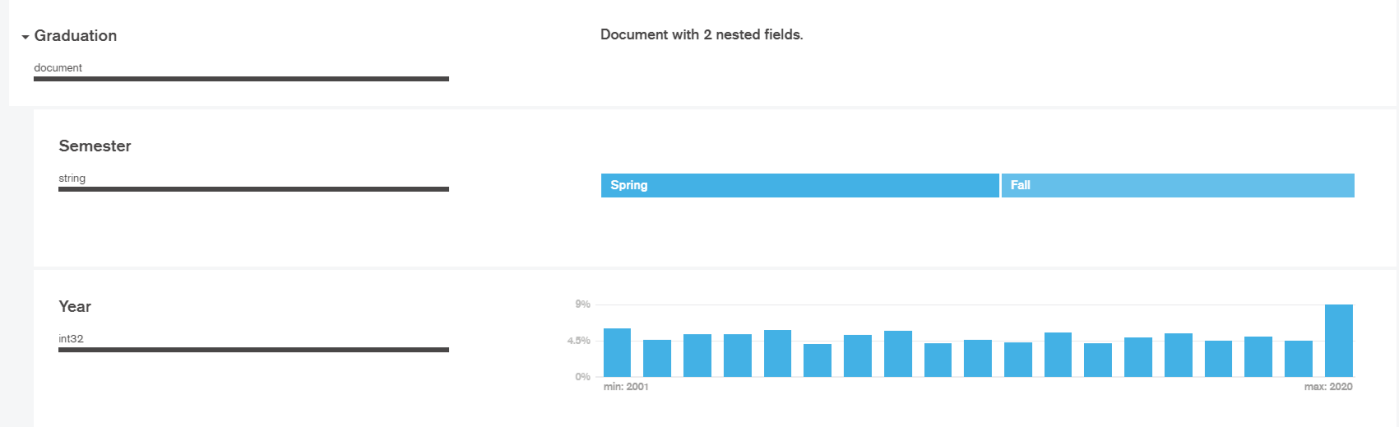
For this section, the most adequate panel of features that really stood out were the ones offered by MongoDB Compass on the Schema Analysis section. Here MongoDB through some of its aggregation features answered quite all of the main queries we considered for this implementation. Some of the visuals are:



Represeting the number of students per city, country, state and Zipcode

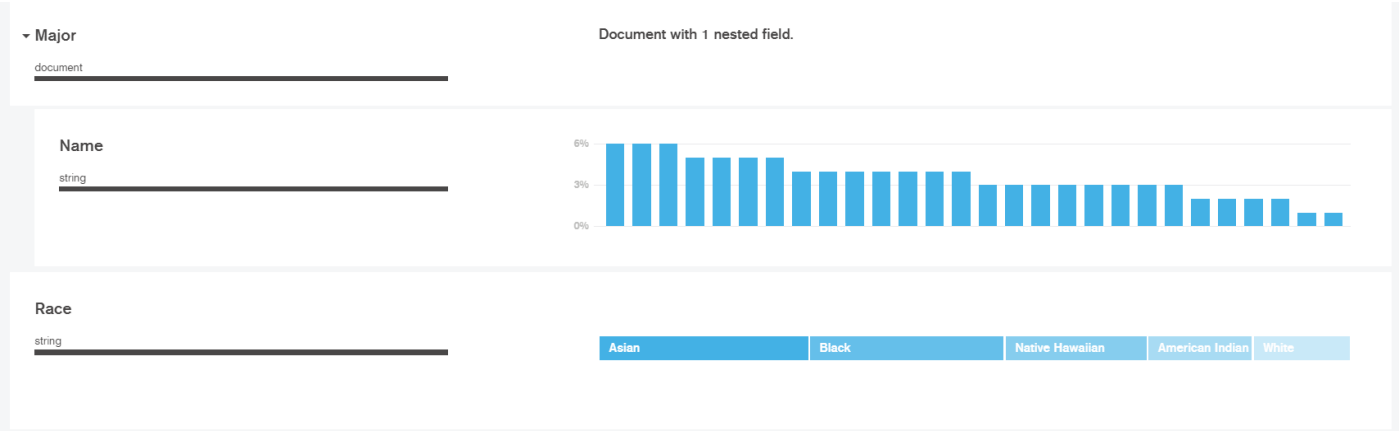


Representing the student distribution

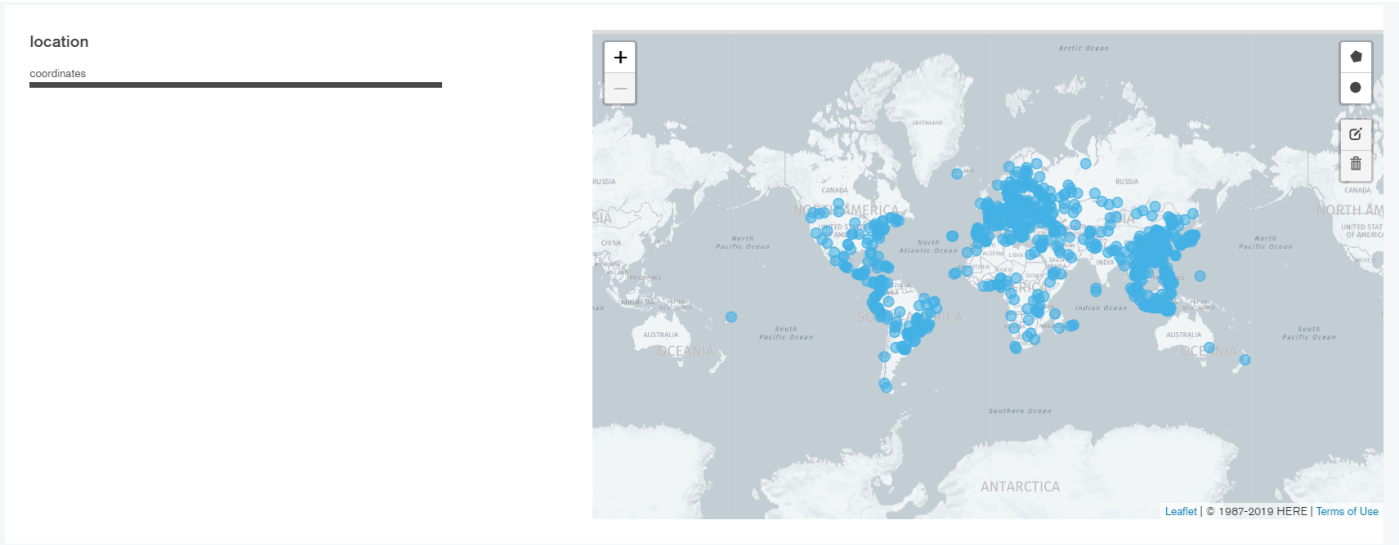


The Student distribution per graduation criteria such as year or semester

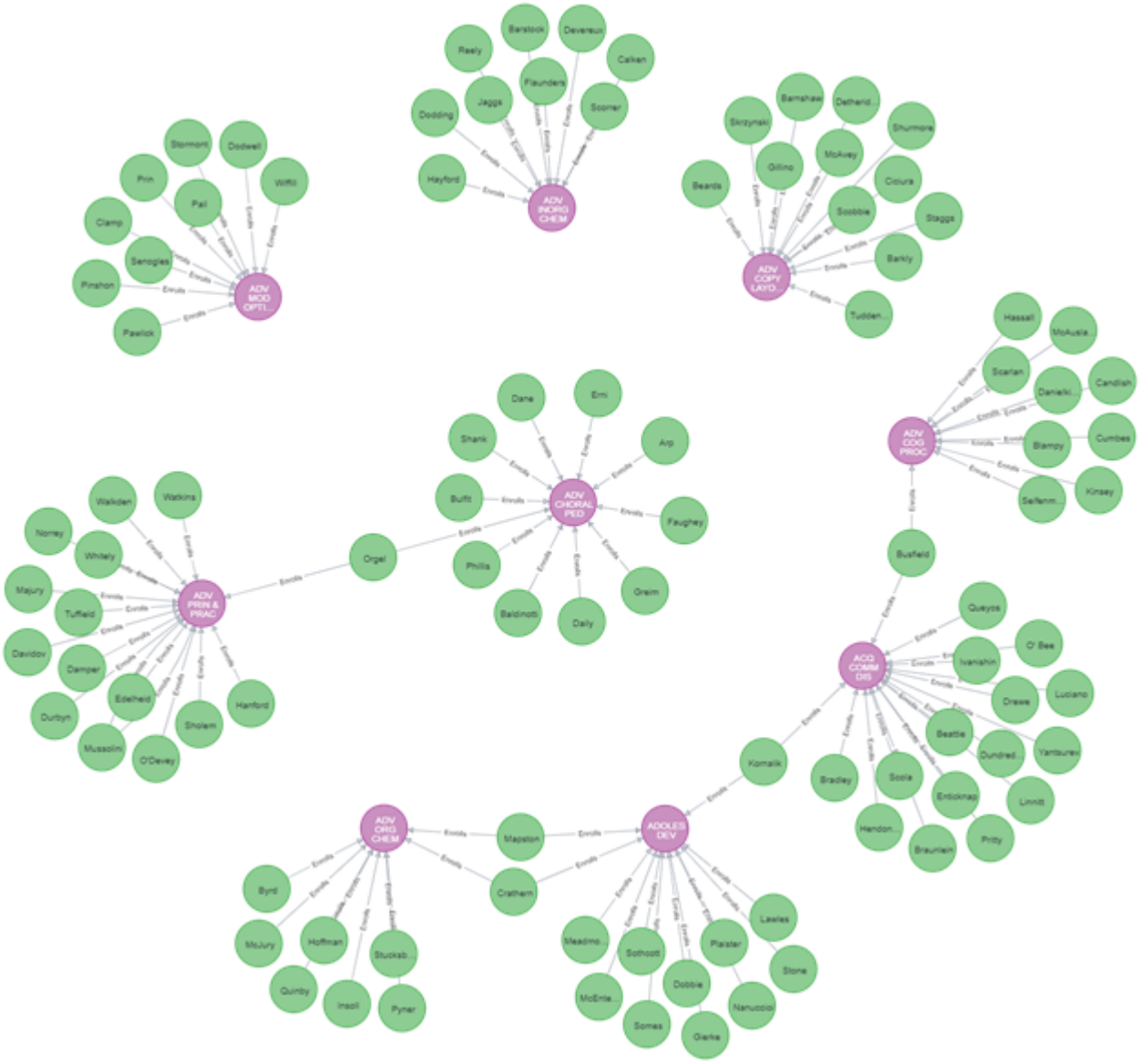
and much more:



Above all the features was these other cool one for spatial queries. MongoDB offered a visual interaction for the spatial query based on the location information build into the document.



On the other side, Neo4j did really offered lots of handy/exploitable features with regard to our scenario and predefined queries. Nonetheless, the fact Neo4j had graphical representation gave cool visuals that could stand when we want to see all related entities of a particular enrollment. For example:



This could be further enhanced with more relationship and graph theory to find a student with similar ambitions or career paths.

Finally, coming to oracle we didn't find much features to exploit to better the production stage of such a University enrollment database implementation.

Conclusion:

On a concluding note, one will choose the oracle database for this implementation especially if every bit of design details are clearly established in advance and if the focus is placed on the performance as judged from the results.

Nonetheless, on a personal note, it will be preferable to do the implementation with MongoDB as this database covered relatively good performances as the SQL database did. Plus, if some of the percentages calculations are move to the front-end, then it's down performance due to the facet stage can be overcome. Furthermore, MongoDB won the extra adequate feature list and has a cool Geospatial indexing that means a lot when trying to study spatial distributions from the coordinates made available. Last but not the least, MongoDB offers that flexibility and scalability that would make any future improvements less costly (compared to Oracle) and also will quicken any present implementation (as oracle took more time to be implemented with regards to this project).

Clean UP:

In [373]:
```
if mongodb_client: mongodb_client.close()
```

In [375]:
```
if neo4j_session: neo4j_session.close()
if neo4j_driver: neo4j_driver.close()
```

In [434]:
```
if oracle_cursor: oracle_cursor.close()
if oracle_connection: oracle_connection.close()
```

The End.