**CS5600: Programming Assignment 2 (50 points)**

In this assignment, you will learn and implement CRUD(Create, Read, Update, Delete). These are the basic APIs to perform at most functions on web applications. You will develop the backend web framework using any programming languages such as Python, Java, C#, …, etc. The backend application must connect to Neo4j-a graph DBMS developed by Neo4j, inc. (https://neo4j.com). To test the backend web framework, you must use API platforms such as Postman API Platform(<https://www.postman.com>).

The CSV file (netflix\_titles.csv-Movie and Show on Netflix) is provided in this assignment with the column names including:

**show\_id**

**type**

**title**

**director**

**cast**

**country**

**date\_added**

**release\_year**

**rating**

**duration**

**description**

First, the instructor will guide you how to generate Neo4j Database on local and cloud using Neo4j Desktop and Neo4j Aura(https://console.neo4j.io). You will import this csv file to create graph database on Neo4j DBMS.

This is a database model for the assignment:

**Person**

[ :ACTED\_IN ]

**Movie**

[ :DIRECTED ]

[ :DISTRIBUTED\_IN ]

**Country**

**Nodes:**

Movie [ show\_id, type, title, release\_year, rating, duration, description ]

load csv with headers from 'file:///netflix1.csv' as row

with row.show\_id as show\_id, row.type as movie\_type, row.title as title, row.release\_year as release\_year, row.rating as rating, row.duration as duration, row.description as description

merge(:Movie{show\_id:show\_id,type:movie\_type,title:title,release\_year:release\_year, rating:rating, duration:duration, description:description })

optimize:

load csv with headers from 'file:///netflix\_titles.csv' as row

merge(m:Movie{title:row.title}) set m.show\_id=row.show\_id,m.type=row.type,m.release\_year=row.release\_year, m.rating=row.rating, m.duration=row.duration, m.description=row.description

Person [ name ]

load csv with headers from 'file:///netflix1.csv' as row

with  SPLIT(row.cast,',') as cast

unwind cast as cast1

with \*,trim(cast1) as cast2

merge(:Person{name:cast2})

optimize

load csv with headers from 'file:///netflix\_titles.csv' as row

unwind SPLIT(row.cast,',') as cast

merge(:Person{name:trim(cast)})

Relationship

load csv with headers from 'file:///netflix1.csv' as row

with row.title as title, SPLIT(row.cast,',') as cast

unwind cast as cast1

with \*,trim(cast1) as cast2

match(m:Movie{title:title}),(p:Person{name:cast2})

merge(p)-[:Acted\_IN]->(m)

return m,p

optimize

load csv with headers from 'file:///netflix\_titles.csv' as row

unwind row.cast as cast

match(m:Movie{title:row.title}),(p:Person{name:trim(cast)})

merge(p)-[:Acted\_IN]->(m)

return m,p

Director:

load csv with headers from 'file:///netflix1.csv' as row

with  SPLIT(row.director,',') as director

unwind director as director1

with \*,trim(director1) as director2

merge(:Person{name:director2})

optimize:

load csv with headers from 'file:///netflix\_titles.csv' as row

unwind SPLIT(row.director,',') as director

merge(:Person{name:trim(director)})

Relationship:

load csv with headers from 'file:///netflix1.csv' as row

with row.title as title, SPLIT(row.director,',') as director

unwind director as director1

with \*,trim(director1) as director2

match(m:Movie{title:title}),(p:Person{name:director2})

merge(p)-[:DIRECTED\_IN]->(m)

return m,p

optimize:

load csv with headers from 'file:///netflix\_titles.csv' as row

unwind SPLIT(row.director,',') as director

match(m:Movie{title:row,title}),(p:Person{name:trim(director)})

merge(p)-[:DIRECTED\_IN]->(m)

return m,p

Country [ name ]

load csv with headers from 'file:///netflix1.csv' as row

with SPLIT(row.country,',') as country

unwind country as country1

with \*,trim(country1) as country2

merge(c:Country{name:country2})

return c;

load csv with headers from 'file:///netflix\_titles.csv' as row

unwind SPLIT(row.country,',') as country

merge(c:Country{name:trim(country)})

return c;

Relationship

load csv with headers from 'file:///netflix1.csv' as row

with row.title as title, SPLIT(row.country,',') as country

unwind country as country1

with \*,trim(country1) as country2

match(m:Movie{title:title}),(c:Country{name:country2})

merge(m)-[:DISTRIBUTED\_IN]->(c)

return m,c

optimized:

load csv with headers from 'file:///netflix\_titles.csv' as row

unwind SPLIT(row.country,',') as country

match(m:Movie{title:row.title}),(c:Country{name:trim(country)})

merge(m)-[:DISTRIBUTED\_IN]->(c)

return m,c

**Relationships:**

ACTED\_IN

DIRECTED

DISTRIBUTED\_IN

The Neo4j Database Configuration:

url = “bolt://localhost:7687” for local database

usr = “neo4j” //default username

pwd = “Admin1234” //the password

…

This is an example of configuration in Python Flask:

Text

Description automatically generated

These are the example of API interfaces in Python file:

In xxxx.py file, you must perform the following main functions:

1. Insert the new movie and show.

Merge(m:movie{title:title}) set m.show\_id=row.show\_id,m.type=row.type,m.release\_year=row.release\_year, m.rating=row.rating, m.duration=row.duration, m.description=row.description

Merge(m:Movie{title:"dil hai hindustani"}) set m.show\_id=210496,m.type="comedy",m.release\_year=2001, m.rating=5

match(m:Movie{title:"Dil hai hindustani"})

merge (m)-[:DISTRIBUTED\_IN]->(p:Country{name:"India"})

return p,m

@app.route('/title', methods=['POST'])

1. Update the movie and show information using title. (By update only title, description, and rating)

Match(m:movie{title:title}) set …..

Match(m:Movie{title:"dil hai hindustani"}) set m.title="Dil hai hindustani",m.description="very good movie, comedy at hight"

@app.route('/title/<string:fname>', methods=['PATCH'])

1. Delete the movie and show information using title.

Match(m:movie)<-[:r]-(:Person)

Where movie.title=””

Delete r

Match(m:movie{title:title}) delete m

match(c:Country)<-[d]-(m:Movie{title: "Dil hai hindustani"})<-[r]-(p:Person)

delete d,m,r,p

@app.route('/title/<string:fname>', methods=['DELETE'])

1. Retrieve all the movies and shows in database.

Match(m:movie{title:””})

Return m

match(m:Movie{title:"Dil hai hindustani"}) return m

@app.route('/title', methods=['GET'])

1. Display the movie and show’s detail includes actors, directors and distributed country using title.

Match(m:Movie)<-[:r]-(p:Person)

Where m.title=””

Return m,p;

match(c:Country)<-[d]-(m:Movie{title: "Keith Richards: Under the Influence"})<-[r]-(p:Person)

return m,p,c

@app.route('/title/<string:fname>', methods=['GET'])

**Submission:**

**Turn in your codes in folder name “PROG\_ASSIGN2\_XXXXX\_YYYYYY” where XXXXX is course number (CRN) and YYYYYY is 700# student id and zip the folder before submitting your assignment.**

**Notes:**

**If you want to use other web frameworks such as Java, JavaScript, PHP, or Microsoft Visual Studio .NET, etc. You must contact me beforehand and let me know how to execute your web framework. You can use the internet cloud application platforms such as Heroku (**[**https://www.heroku.com**](https://www.heroku.com)**). However, in your assignment submission, you must provide me the links for execute the web application and source codes.**