4/23/2022

F21BD Course Work -2 IMDB

F21BD Group 5

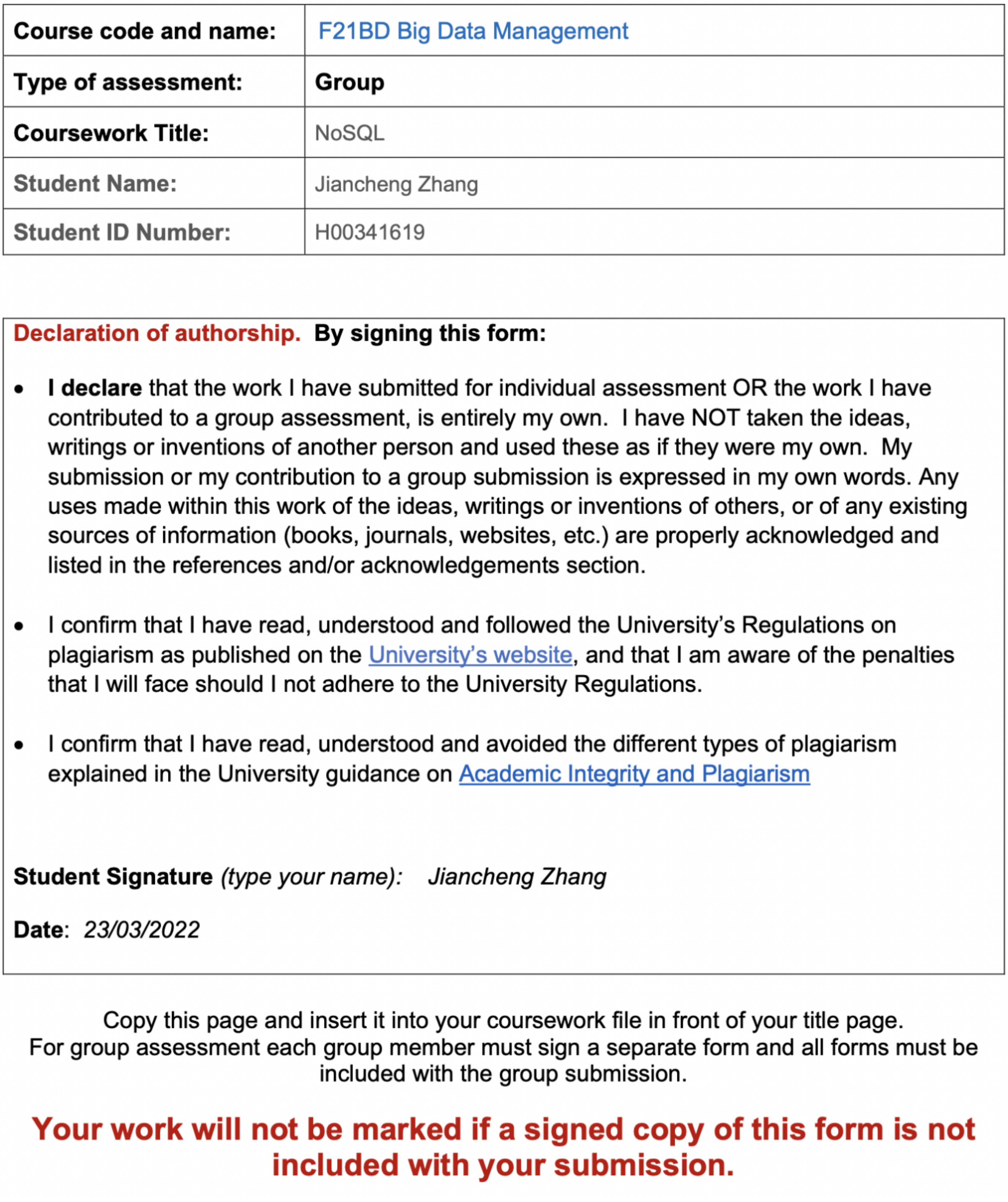
Group Members:

Jiancheng Zhang	H0034619
Nandita Baniya	H00385371
Siddhesh Pangam	H00382943
Shonan Gomes	H00383160
Shreyas Arunesh	H00357095

Table of Contents

Standard Declaration of Student Authorship

Jiancheng Zhang



Course code and name:	F21BD Big Data Management
Type of assessment:	**Group**
Coursework Title:	NoSQL
Student Name:	Jiancheng Zhang
Student ID Number:	H00341619

Declaration of authorship. By signing this form:

- **I declare** that the work I have submitted for individual assessment OR the work I have contributed to a group assessment, is entirely my own. I have NOT taken the ideas, writings or inventions of another person and used these as if they were my own. My submission or my contribution to a group submission is expressed in my own words. Any uses made within this work of the ideas, writings or inventions of others, or of any existing sources of information (books, journals, websites, etc.) are properly acknowledged and listed in the references and/or acknowledgements section.

- I confirm that I have read, understood and followed the University's Regulations on plagiarism as published on the University's website, and that I am aware of the penalties that I will face should I not adhere to the University Regulations.

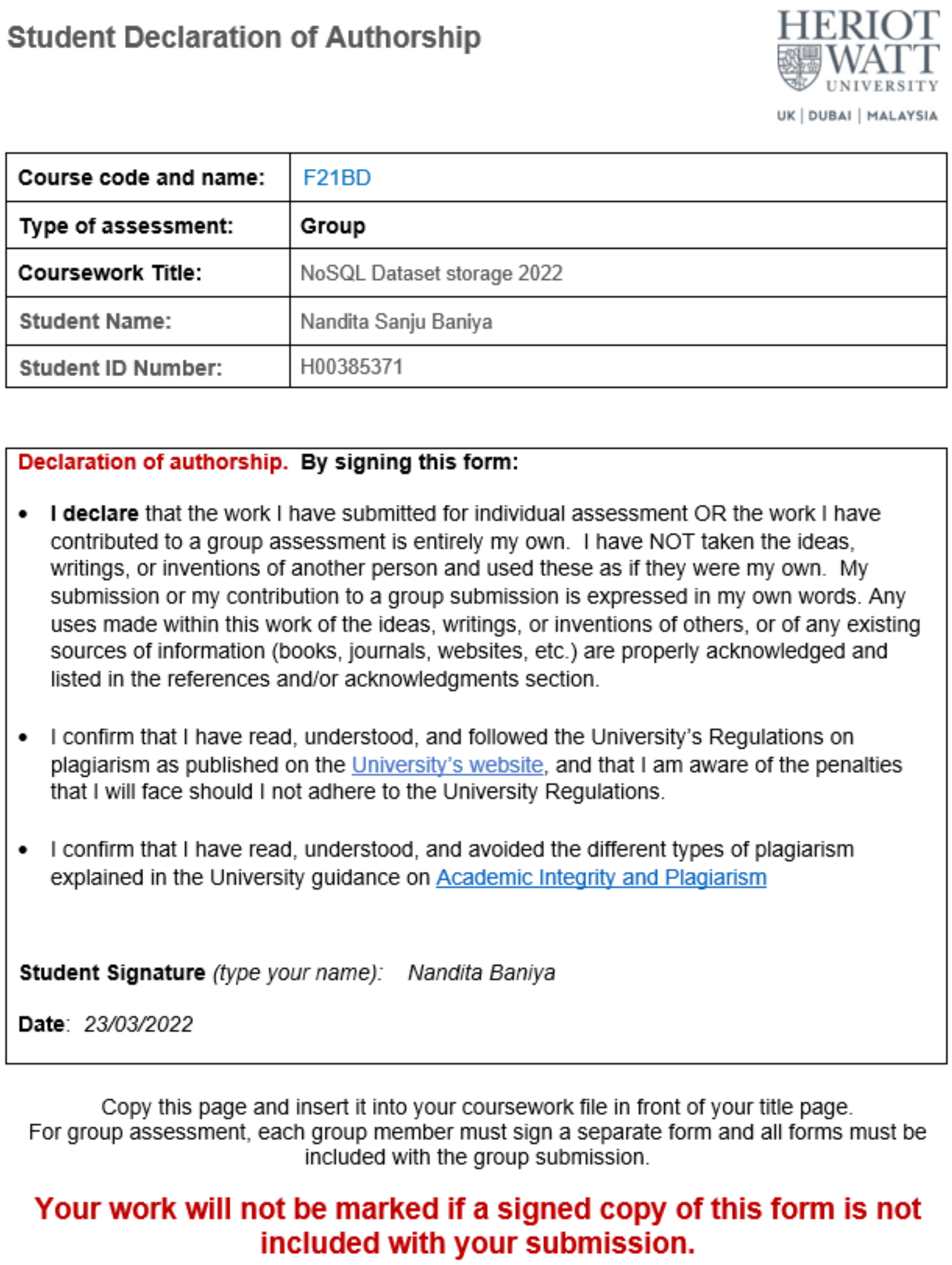
- I confirm that I have read, understood and avoided the different types of plagiarism explained in the University guidance on Academic Integrity and Plagiarism

Student Signature *(type your name):* *Jiancheng Zhang*

Date: *23/03/2022*

Copy this page and insert it into your coursework file in front of your title page.
For group assessment each group member must sign a separate form and all forms must be included with the group submission.

Your work will not be marked if a signed copy of this form is not included with your submission.



Student Declaration of Authorship

Course code and name:	F21BD
Type of assessment:	**Group**
Coursework Title:	NoSQL Dataset storage 2022
Student Name:	Nandita Sanju Baniya
Student ID Number:	H00385371

Declaration of authorship. By signing this form:

- **I declare** that the work I have submitted for individual assessment OR the work I have contributed to a group assessment is entirely my own. I have NOT taken the ideas, writings, or inventions of another person and used these as if they were my own. My submission or my contribution to a group submission is expressed in my own words. Any uses made within this work of the ideas, writings, or inventions of others, or of any existing sources of information (books, journals, websites, etc.) are properly acknowledged and listed in the references and/or acknowledgments section.

- I confirm that I have read, understood, and followed the University's Regulations on plagiarism as published on the University's website, and that I am aware of the penalties that I will face should I not adhere to the University Regulations.

- I confirm that I have read, understood, and avoided the different types of plagiarism explained in the University guidance on Academic Integrity and Plagiarism

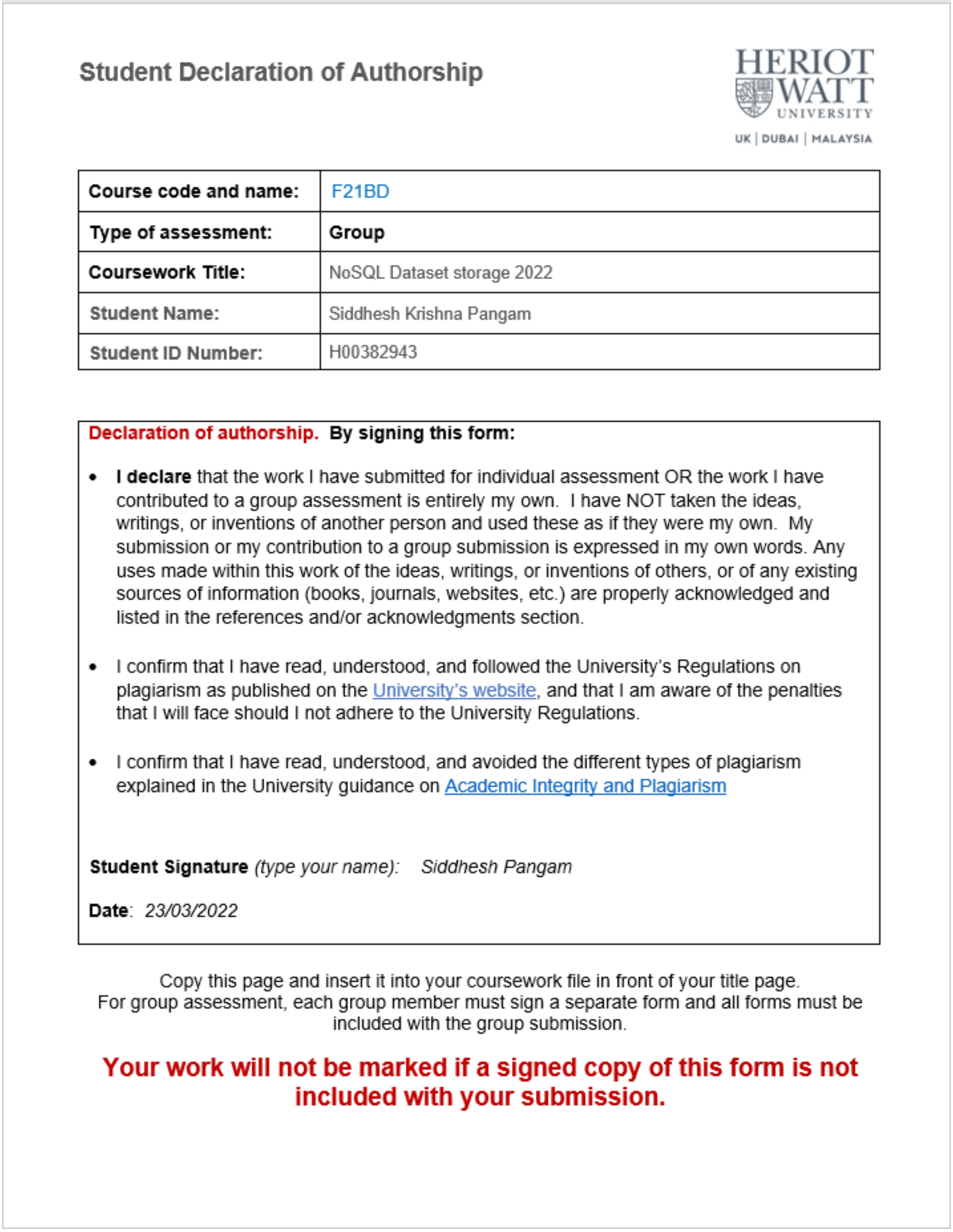
Student Signature *(type your name):* *Nandita Baniya*

Date: *23/03/2022*

Copy this page and insert it into your coursework file in front of your title page.
For group assessment, each group member must sign a separate form and all forms must be included with the group submission.

Your work will not be marked if a signed copy of this form is not included with your submission.

Siddhesh Pangam



Student Declaration of Authorship

Course code and name:	F21BD
Type of assessment:	**Group**
Coursework Title:	NoSQL Dataset storage 2022
Student Name:	Siddhesh Krishna Pangam
Student ID Number:	H00382943

Declaration of authorship. By signing this form:

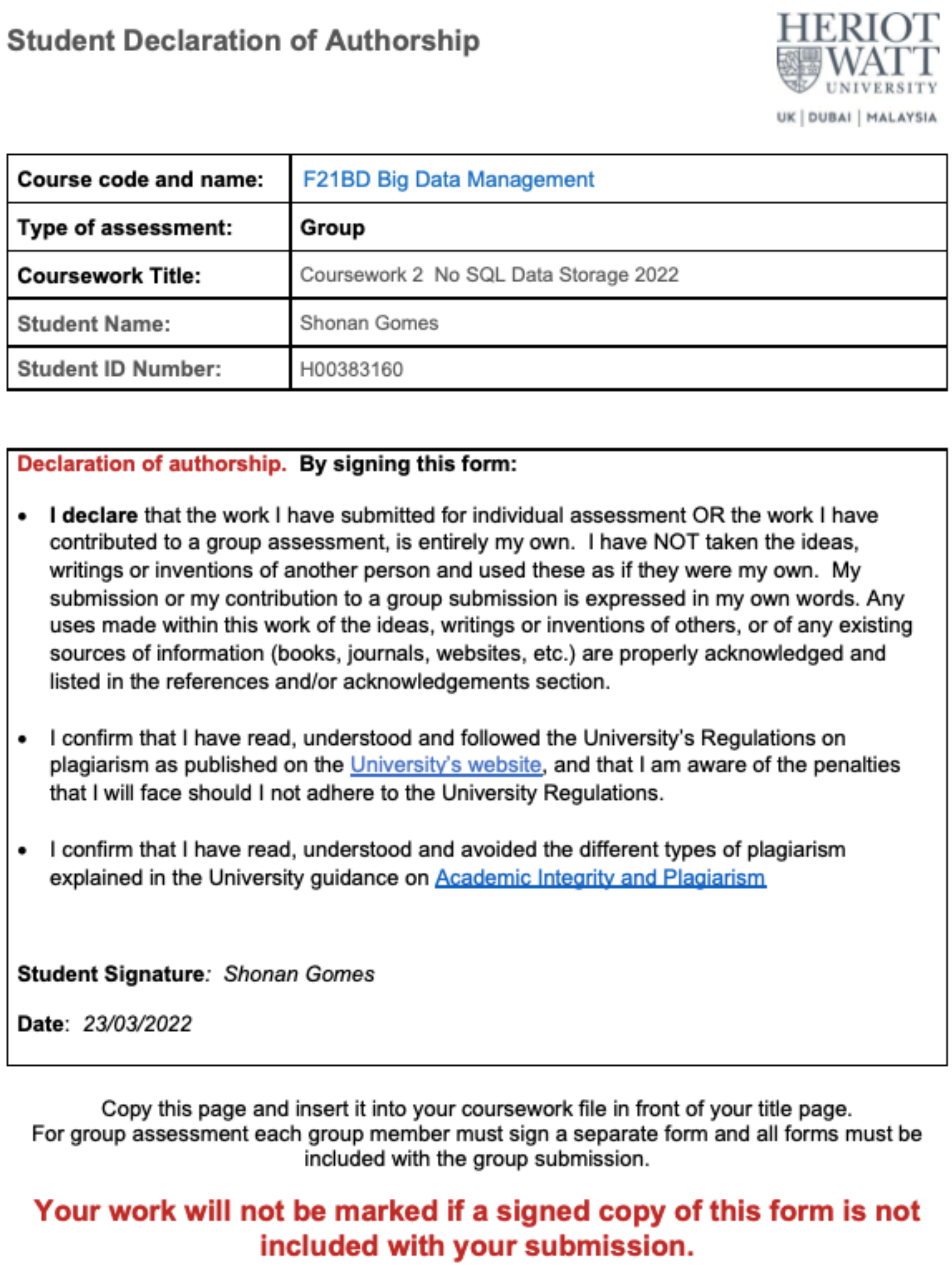
- **I declare** that the work I have submitted for individual assessment OR the work I have contributed to a group assessment is entirely my own. I have NOT taken the ideas, writings, or inventions of another person and used these as if they were my own. My submission or my contribution to a group submission is expressed in my own words. Any uses made within this work of the ideas, writings, or inventions of others, or of any existing sources of information (books, journals, websites, etc.) are properly acknowledged and listed in the references and/or acknowledgments section.
- I confirm that I have read, understood, and followed the University's Regulations on plagiarism as published on the University's website, and that I am aware of the penalties that I will face should I not adhere to the University Regulations.
- I confirm that I have read, understood, and avoided the different types of plagiarism explained in the University guidance on Academic Integrity and Plagiarism

Student Signature *(type your name)*: *Siddhesh Pangam*

Date: *23/03/2022*

Copy this page and insert it into your coursework file in front of your title page.
For group assessment, each group member must sign a separate form and all forms must be included with the group submission.

Your work will not be marked if a signed copy of this form is not included with your submission.



Student Declaration of Authorship

Course code and name:	F21BD Big Data Management
Type of assessment:	**Group**
Coursework Title:	Coursework 2 No SQL Data Storage 2022
Student Name:	Shonan Gomes
Student ID Number:	H00383160

Declaration of authorship. By signing this form:

- **I declare** that the work I have submitted for individual assessment OR the work I have contributed to a group assessment, is entirely my own. I have NOT taken the ideas, writings or inventions of another person and used these as if they were my own. My submission or my contribution to a group submission is expressed in my own words. Any uses made within this work of the ideas, writings or inventions of others, or of any existing sources of information (books, journals, websites, etc.) are properly acknowledged and listed in the references and/or acknowledgements section.
- I confirm that I have read, understood and followed the University's Regulations on plagiarism as published on the University's website, and that I am aware of the penalties that I will face should I not adhere to the University Regulations.
- I confirm that I have read, understood and avoided the different types of plagiarism explained in the University guidance on Academic Integrity and Plagiarism

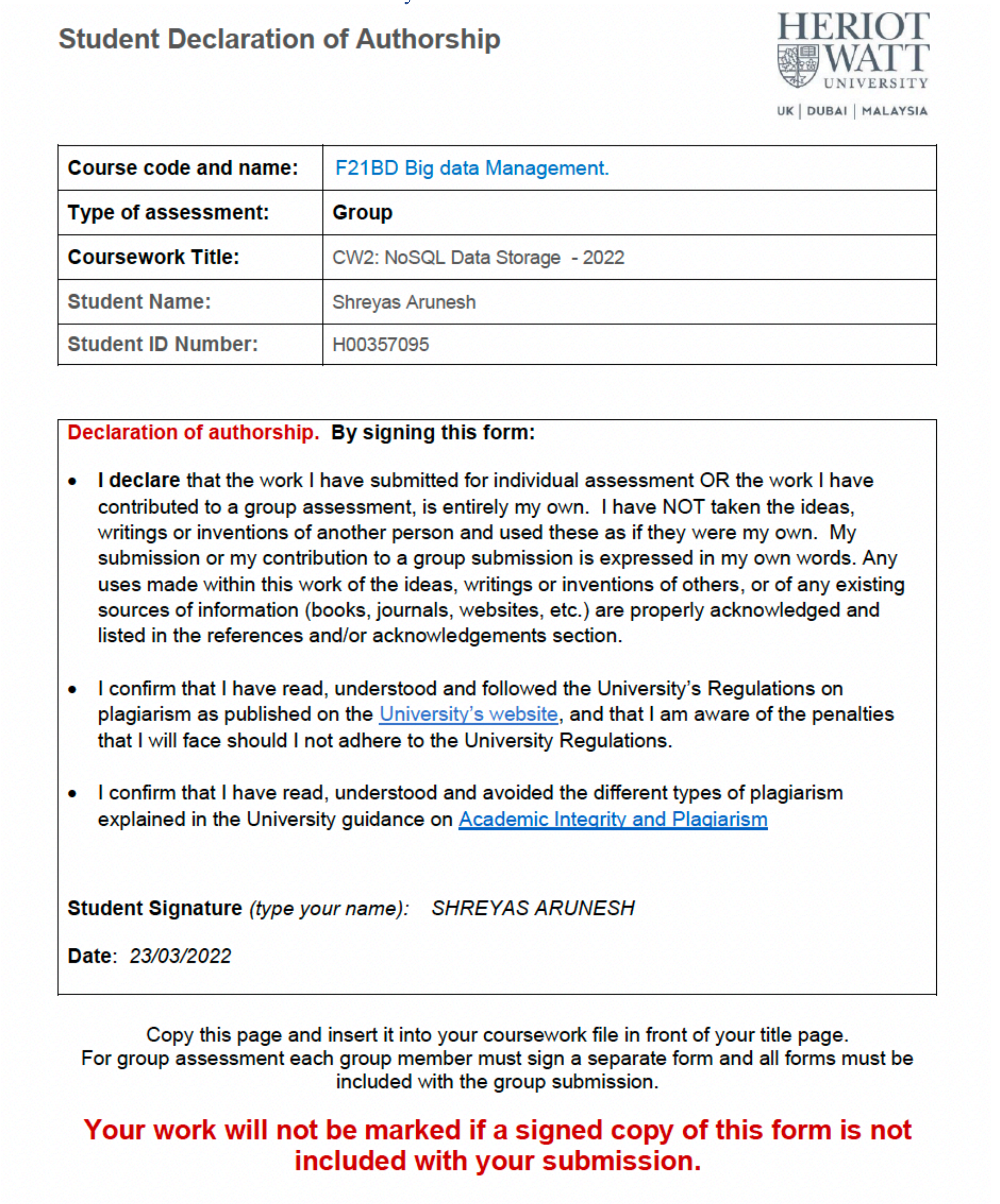
Student Signature: *Shonan Gomes*

Date: *23/03/2022*

Copy this page and insert it into your coursework file in front of your title page.
For group assessment each group member must sign a separate form and all forms must be included with the group submission.

Your work will not be marked if a signed copy of this form is not included with your submission.

Shreyas Arunesh



Student Declaration of Authorship

HERIOT
WATT
UNIVERSITY
UK | DUBAI | MALAYSIA

Course code and name:	F21BD Big data Management.
Type of assessment:	**Group**
Coursework Title:	CW2: NoSQL Data Storage - 2022
Student Name:	Shreyas Arunesh
Student ID Number:	H00357095

Declaration of authorship. By signing this form:

- **I declare** that the work I have submitted for individual assessment OR the work I have contributed to a group assessment, is entirely my own. I have NOT taken the ideas, writings or inventions of another person and used these as if they were my own. My submission or my contribution to a group submission is expressed in my own words. Any uses made within this work of the ideas, writings or inventions of others, or of any existing sources of information (books, journals, websites, etc.) are properly acknowledged and listed in the references and/or acknowledgements section.

- I confirm that I have read, understood and followed the University's Regulations on plagiarism as published on the University's website, and that I am aware of the penalties that I will face should I not adhere to the University Regulations.

- I confirm that I have read, understood and avoided the different types of plagiarism explained in the University guidance on Academic Integrity and Plagiarism

Student Signature *(type your name):* *SHREYAS ARUNESH*

Date: *23/03/2022*

Copy this page and insert it into your coursework file in front of your title page.
For group assessment each group member must sign a separate form and all forms must be included with the group submission.

Your work will not be marked if a signed copy of this form is not included with your submission.

Group Contribution List

Everybody in the group contributed equally in all the sections of this coursework and all the Queries and output results obtained in this report are the results of mutual collaboration and knowledge sharing among the team members.

HW Number	Name	Tasks worked	Contribution Level (%)	Signature
H0034619	Jiancheng Zhang	Task 1: Data Analysis. Task-2: Data cleaning. Task-3: Neo4j- Query 7, 10, 11, 12, 13, 14, 15 Task 4: Proof reading Report: Formatting and Structuring.	100	Jiancheng Zhang
H00385371	Nandita Baniya	Task-1: Relations in ER diagram. Task-2: Proof Reading and cross-checking outputs. Task-3: Cross checking Outputs. Task-4: Sustainability, MongoDB: 1, 2, 3, Report: Writing and formatting.	100	Nandita baniya
H00382943	Siddhesh Pangam	Task-1: ER diagram. Task-2: Proof Reading and cross-checking outputs. Task-3: Cross checking Outputs. Task-4: MongoDB: 5, 6, 7 Comparison of MongoDB and Neo4J Report: Writing and Structuring.	100	Siddhesh Pangam
H00383160	Shonan Gomes	Task 1: Relations in ER diagram. Task 2: Data loading Task 3: Neo4j: Query 1, 2, 3, 4, 5,6,8 Task 4: Differences. Report: Writing and structuring.	100	Shonan Gomes
H00357095	Shreyas Arunesh	Task-1: ER diagram. Task-2 Data cleaning and loading. Task 3: Neo4j- Query 9 , MongoDB- Task 4: Query- 10. Report: Writing, formatting, and Structuring.	100	ShreyasArunesh

Introduction

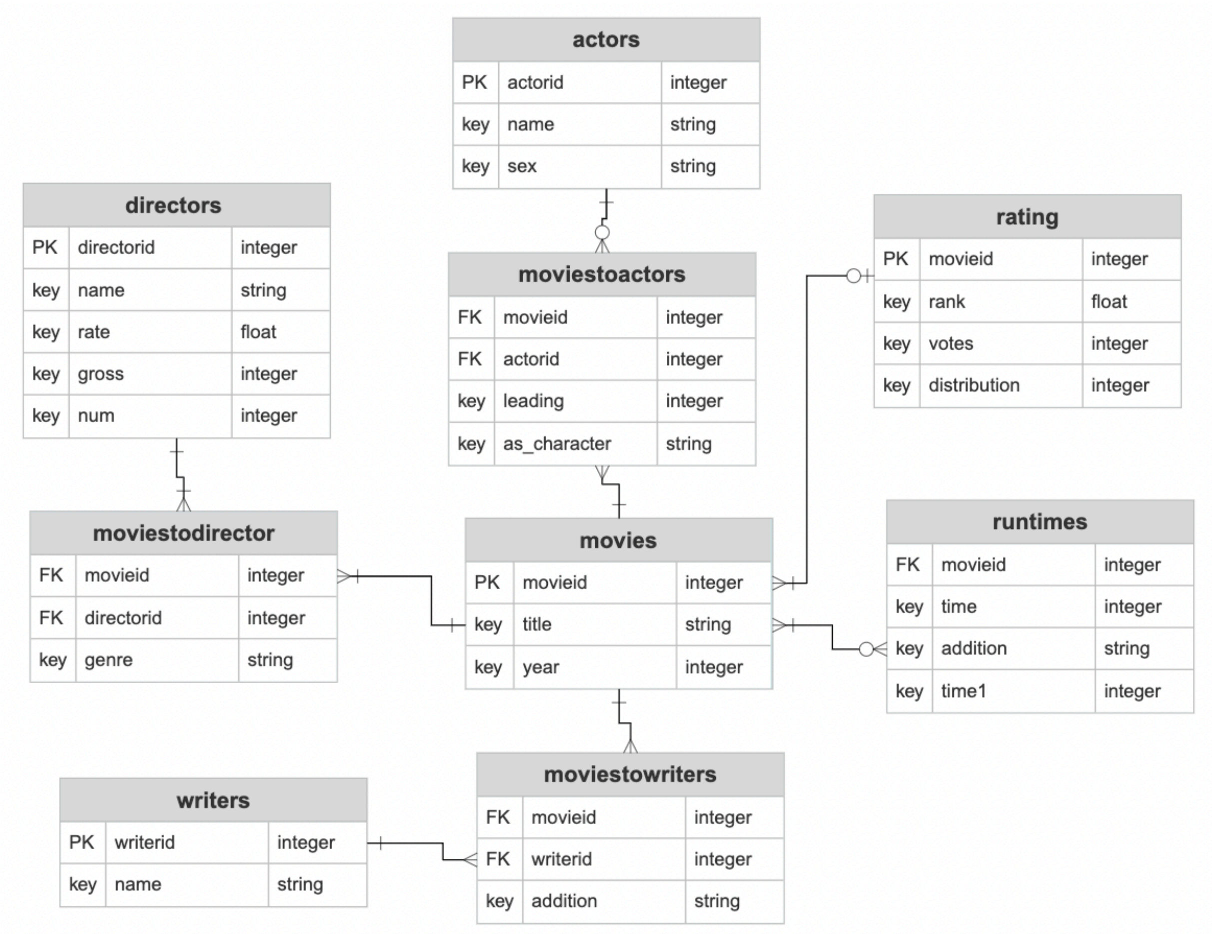
This Coursework focuses on the NoSQL database systems where the primary focus is on Neo4j and MongoDB querying language to retrieve the task specified information's. The dataset provided is a Internet movie dataset which is the website of information about movies. We are provided with the subset of IMDB in the form of CSV files.

This document provides the database Schema illustrated through Entity Relationship diagram, detailed explanation of the data pre-processing and the instructions to load the datasets in neo4j and MonogoDB. Followed by Cypher/ Commands to answer the coursework questions along with the comments and explanation of the approaches taken. At the end, the suitability of Neo4j and MongoDB is being discussed including strengths/ Weaknesses, challenges and illustrated the difference in the answers between Neo4j and MongoDB.

Task – 1: Entity Relationship Diagram of IMDB Dataset

This section discusses about the IMDB dataset provided in the coursework. It is observed that all the records in the database have a well-defined schema to maintain consistency and ordering. Physical schema provides the logical representation of the information about each entity, relations. This logical structure is provided using Entity- Relationship diagram (ER Diagram). The Entity relationship diagram of IMDB database is illustrated in the Figure 1 bellow. The database includes information about movies, actors, directors, and writers in general and is normalised as seen in the table movie, ratings and running times. In the figure 1 below, the relations are illustrated as:

1. One Movie can have many Actors.
2. One Movie can have one or many Directors.
3. One Movie can have one or many Writers.
4. One Movie can have zero or many Runtimes.
5. One Movie can have zero or one Ratings.
6. One Writer can have one or many Movies.
7. One Director can have one or many Movies.
8. One Actor can have zero or many Movies.



actors

PK	actorid	integer
key	name	string
key	sex	string

directors

PK	directorid	integer
key	name	string
key	rate	float
key	gross	integer
key	num	integer

moviestoactors

FK	movieid	integer
FK	actorid	integer
key	leading	integer
key	as_character	string

rating

PK	movieid	integer
key	rank	float
key	votes	integer
key	distribution	integer

moviestodirector

FK	movieid	integer
FK	directorid	integer
key	genre	string

movies

PK	movieid	integer
key	title	string
key	year	integer

runtimes

FK	movieid	integer
key	time	integer
key	addition	string
key	time1	integer

writers

PK	writerid	integer
key	name	string

moviestowriters

FK	movieid	integer
FK	writerid	integer
key	addition	string

Figure 1: Entity Relationship diagram of IMDB database.

Task -2: Data Pre-processing

Data Cleaning:

This section describes the data-pre-processing required before loading the dataset for querying. The data cleaning was completed using Python using the libraries Pandas, regular expression, and Unicode. The below table contains the discovered noises, the approaches to clean the discovered noise and additional column for the name.

Importing Libraries

```
import pandas as pd
import NumPy as np
from unidecode import unidecode
import re
import os
```

Reading all the data:

```
"""Load the CSV life provided """

actors = pd.read_csv(os.getcwd()+"/imdb/actors.csv", delimiter=';')
directors = pd.read_csv(os.getcwd()+ "/imdb/directors.csv", delimiter=';')
moviestodirectors = pd.read_csv(os.getcwd()+ "/imdb/moviestodirectors.csv",
delimiter=';')
ratings = pd.read_csv(os.getcwd()+ "/imdb/ratings.csv", delimiter=';')
writers = pd.read_csv(os.getcwd()+ "/imdb/writers.csv", delimiter=';')
movies = pd.read_csv(os.getcwd()+ "/imdb/movies.csv", delimiter=';')
moviestoactors = pd.read_csv(os.getcwd()+ "/imdb/moviestoactors.csv", delimiter=';')
moviestowriters =  pd.read_csv(os.getcwd()+ "/imdb/moviestowriters.csv", delimiter=';')
runningtimes = pd.read_csv(os.getcwd()+ "/imdb/runningtimes.csv", delimiter=';')
```

Actors Data Cleaning:

Some of the Noises cleaned are:

1. Removed special characters from the name using Regular expression (Eg: (‘, (), $-.).
2. Splitting actors name into first name and last name.
3. Removed noises (Ex: (I), Jr etc…)
4. Checked for any duplicated Actor id and removed unnecessary spaces.

Python Code for cleaning the data:

```
"""Cleaning the Actors data"""
"""Apply unicode"""
actors["name"] = actors["name"].apply(unidecode)

""" removing special characters"""
def removeSpecialCharecter(item):
    item = item.replace("Jr", "")
    item = re.sub(r'[-]',r' ',item)
    item = re.sub(r'[^a-zA-Z0-9 ]',r'',item)
    return item

"""split name into first name and last name"""
def splitName(name):
    f_name = name
    l_name = ""
    if(',' in name):
        l_name, f_name = name.split(', ')
        if( ' (' in f_name):
            f_name = f_name.split(' (')[0]
    elif( ' (' in name):
        f_name = name.split(' (')[0]
    f_name = removeSpecialCharecter(f_name)
    l_name = removeSpecialCharecter(l_name)
    if(f_name == ""):
        return [l_name,f_name]
    else:
        return [f_name,l_name]
```

```
"""Remove duplicated values and splitting the name to f_name and l_name"""
actorsnames = (actors.drop_duplicates(keep= "first")).name.apply(splitName)

""" Creating fname and lname column"""
actors['fname'] = np.array([i[0] for i in actorsnames]).T
actors['lname'] = np.array([i[1] for i in actorsnames]).T

"""Exporting the cleaned data to new CSV file"""
actors.to_csv(os.getcwd()+ "/Cleaned_Dataset/actors.csv",index=False)
```

Director Data cleaning

Some of the noises cleaned are:

1. Removed special characters from the name using Regular expression (Ex: (‘,()$-.).
2. Splitting actors name into first name and last name.
3. Removed noises (Ex: (I), Jr etc…)
4. Checked for any duplicated Actor id

The python functions for the data cleaning :

```
"""Cleaning the Director data"""
"""Apply unicode"""
directors["name"] = directors["name"].apply(unidecode)

""" removing special characters"""
def removeSpecialCharecter(item):
    item = item.replace("Jr","")
    item = re.sub(r'[-]',r' ',item)
    item = re.sub(r'[^a-zA-Z0-9 ]',r'',item)
    return item

"""split name into first name and last name"""
def splitName(name):
    f_name = name
    l_name = ""
    if(',' in name):
        l_name, f_name = name.split(', ')
    if( ' (' in f_name):
        f_name = f_name.split(' (')[0]

    f_name = removeSpecialCharecter(f_name)
    l_name = removeSpecialCharecter(l_name)

    if(f_name == ""):
        return [l_name,f_name]
    else:
        return [f_name,l_name]
```

Calling the functions:

```
"""Remove duplicated values and splitting the name to f_name and l_name"""
names = directors.name.apply(splitName)

""" Creating fname and lname column"""
directors['fname'] = np.array([i[0] for i in names]).T
directors['lname'] = np.array([i[1] for i in names]).T

"""Exporting the cleaned data to new CSV file"""
directors.to_csv(os.getcwd()+ "/Cleaned_Dataset/directors.csv",index=False)
```

Movie Data cleaning:

Some of the noises cleaned are:

1. Extracting only movie titles.
2. Check for any duplicate values.

The python code has same functions of actors:

```
"""Applying unicode"""
movies["title"] = movies["title"].apply(unidecode)

"""Extracting only the movie titles and remove all  other special character"""
movies["title"] = movies["title"].apply(lambda item: item.split(" (")[0]) # remove ' in the title
movies["title"] = movies["title"].apply(lambda item: item.replace("'","") if item.count("'") == 2 else item)

"""Exporting the cleaned data to new CSV file"""
movies.to_csv(os.getcwd()+ "/Cleaned_Dataset/movies.csv",index=False)
```

Writers’ data Cleaning:

Some of the noises cleaned are:

1. Removed special characters in writers name.
2. Checked for duplicated writers id.

Python code for cleaning the writers data.

```
"""Cleaning Writers data"""
"""split name into first name and last name"""
def splitName(name):
   name = unidecode(name)
   f_name = l_name = ""
   if "," in name :
      f_name, l_name = name.split(",")
   l_name = re.sub(r"[.']",r' ',l_name.split(' (')[0])
   f_name = f_name.replace("Jr.","")
   return [f_name,l_name]

"""Remove duplicated values and splitting the name to f_name and l_name"""
names = (writers.drop_duplicates(keep= "first")).name.apply(splitName)

""" Creating fname and lname column"""
writers['fname'] = np.array([i[0] for i in names]).T
writers['lname'] = np.array([i[1] for i in names]).T

"""Exporting the cleaned data to new CSV file"""
writers.to_csv(os.getcwd()+ "/Cleaned_Dataset/writers.csv",index=False)
```

Movie-to-writers Data cleaning:

Some of the noises cleaned are:

1. Removing special characters
2. Removing additional information like <0-9,0-9,0-9>.

Python code for the cleaning:

```
"""Cleaning Movie to writter data"""
""" removing special characters"""
def removeSpecialCharecters(item):
    item = unidecode(str(item))
    if "(" in item :
        item = (item.split("(")[1]).split(')')[0]
    if '" ' in item :
        item = item.split('" ')[1]
    if ' "' in item :
        item = item.split(' "')[0]
    if ': ' in item:
        item = item.split(': ')[0]
    if ',' in item:
        item = item.split(',')[0]
    if "as " in item or item == 'by' or item == 'nan':
        item = ""
    if re.search(r'^<', item):
        item = ""
    item = re.sub(r'[^a-zA-Z ]', r'', item)
    return item


moviestowriters["additions"] = moviestowriters.addition.apply(removeSpecialCharecters)

"""Exporting the cleaned data to new CSV file"""
moviestowriters.to_csv(os.getcwd()+ "/Cleaned_Dataset/moviestowriters.csv", index=False)
```

Movies-to-actors data cleaning:

Some of the noise cleaned are:

1. as_character had appended leading
2. as_character info was extracted from embedding in []
3. as_character having unwanted special characters, number and null values that are removed

Python code for cleaning movie-to-actor data:

```
"""Cleaning Movie to Actor"""
""" removing special characters"""
def getRole(item):
    item = str(item)
    if ", Age" in item:
        item = item.split(',')[0]
    if ", age" in item:
        item = item.split(',')[0]
    if('[' in item):
        item = (item.split('[')[1]).split(']')[0]
    if 'age ' in item:
        item = item.split('age 0-9')[0]

    if '(' in item:
        item = item.split('(')[0]
    return item


moviestoactors["played_character"] = moviestoactors["as_character"].apply(getRole)
"""replace all nan values by ' ' """
moviestoactors.loc[moviestoactors['as_character'].isna(),'as_character'] = ""
"""remove ' from from the played_character column"""
moviestoactors["played_character"] = moviestoactors["played_character"].apply(lambda
item: item.replace("'","") if item.count("'") == 2 else item)
"""unidecode data"""
moviestoactors["played_character"] =
moviestoactors["played_character"].apply(unidecode)
"""Exporting the cleaned data to new CSV file"""
moviestoactors.to_csv(os.getcwd()+ "/Cleaned_Dataset/moviestoactors.csv",index=False)
```

Movie-to-directors data cleaning:

There was no noise discovered in this dataset

Python code for cleaning movie-to-directors

```
"""cleaning movie-to-directors"""
"""Exporting the cleaned data to new CSV file"""
moviestodirectors.to_csv(os.getcwd()+"/Cleaned_Dataset/moviestodirectors.csv",index=False)
```

Running times data cleaning:

Some of the noises cleaned are:

1. Removing special characters from time column.
2. Removing (():) characters.
3. Extracting only time from the formats Country:time.

Python code for data cleaning:

```
"""cleaning runningtimes data"""

"""Remove special Characters from time column"""
runningtimes.time = runningtimes.time.apply(lambda item: re.sub(r'[A-Za-z:]',r'',item))

"""Remove special Characters from addition1 column and store it in new column"""
runningtimes['addition1'] = runningtimes.addition.apply(lambda item: unidecode(re.sub(r'[):(]',r'',item)) if str(item) != 'nan' else "")

"""Exporting the cleaned data to new CSV file"""
runningtimes.to_csv(os.getcwd()+ "/Cleaned_Dataset/runningtimes.csv",index=False)
```

Ratings data cleaning:

Some of the noises cleaned are:

1. Proceedings zero in distribution column
2. Periods in between distribution column

Python code for cleaning Ratings data

```
"""cleaning rating Data"""

"""Remove special Characters and formating the number"""
def formatNumber(number):
    if len(str(number)) != 0 and str(number) != 'nan':
        return int(re.sub(r'[.]',r'',str(number)))
    else:
        return 0

ratings['distributionNumber'] = ratings.distribution.apply(formatNumber)

"""Exporting the cleaned data to new CSV file"""
ratings.to_csv(os.getcwd()+ "/Cleaned_Dataset/ratings.csv",index=False)
```

Commands required to load the CSV file in Neo4j:

```
# Create constraint for actorid

CREATE CONSTRAINT ON (a:Actors) ASSERT a.actorid IS UNIQUE;

# Loading actor CSV file in Neo4j

:auto USING PERIODIC COMMIT
LOAD CSV WITH HEADERS FROM "http://localhost:11001/project-82664212-95dd-
4e3d-87a0-b543e47741c5/actors.csv" AS csv
MERGE(a:Actors{actorid:toInteger(csv.actorid)})
ON CREATE SET a.name = csv.name,a.sex = csv.sex,a.fname = csv.fname,a.lname =
csv.lname;
```

```
# Create constraint for directorid

CREATE CONSTRAINT ON (d:Directors) ASSERT d.directorid IS UNIQUE;

# Loading actor CSV file in Neo4j

:auto USING PERIODIC COMMIT
LOAD CSV WITH HEADERS FROM "http://localhost:11001/project-82664212-95dd-
4e3d-87a0-b543e47741c5/directors.csv" AS csv
MERGE(d:Directors{directorid:toInteger(csv.directorid)})
ON CREATE SET d.name = csv.name, d.rate = toFloat(csv.rate),
d.gross = toInteger(csv.gross), d.num = toInteger(csv.num), d.fname = csv.fname, d.lname
= csv.lname;
```

```
# Create constraint for writerid

CREATE CONSTRAINT ON (w:Writers) ASSERT w.writerid IS UNIQUE;

# Loading actor CSV file in Neo4j

:auto USING PERIODIC COMMIT
LOAD CSV WITH HEADERS FROM "http://localhost:11001/project-82664212-95dd-
4e3d-87a0-b543e47741c5/writers.csv" AS csv
MERGE(w:Writers{writerid:toInteger(csv.writerid)})
ON CREATE SET w.name = csv.name,w.fname = csv.fname,w.lname = csv.lname;
```

```
# Create constraint for movieid
CREATE CONSTRAINT ON (m:Movies) ASSERT m.movieid IS UNIQUE;

# Loading actor CSV file in Neo4j
:auto USING PERIODIC COMMIT
LOAD CSV WITH HEADERS FROM "http://localhost:11001/project-82664212-95dd-
4e3d-87a0-b543e47741c5/movies.csv" AS csv
MERGE(m:Movies{movieid:toInteger(csv.movieid)})
ON CREATE SET m.title = csv.title, m.year = toInteger(csv.year)
```

```
# Loading actor CSV file and create a relationship between them in Neo4j

LOAD CSV WITH HEADERS FROM "http://localhost:11001/project-82664212-95dd-
4e3d-87a0-b543e47741c5/moviestoactors.csv" AS csv
MATCH (a:Actors {actorid: toInteger(csv.actorid)})
MATCH (m:Movies {movieid: toInteger(csv.movieid)})
CREATE (a)-[r:ActedIn{as_character : csv.played_character,leading :
toInteger(csv.leading)}]->(m)
```

```
# Loading actor CSV file and create a relationship between them in Neo4j

LOAD CSV WITH HEADERS FROM "http://localhost:11001/project-82664212-95dd-
4e3d-87a0-b543e47741c5/moviestodirectors.csv" AS csv
MATCH (d:Directors {directorid: toInteger(csv.directorid)})
MATCH (m:Movies {movieid: toInteger(csv.movieid)})
CREATE (d)-[r:Directed{genre:csv.genre}]->(m)
```

```
# Loading actor CSV file and create a relationship between them in Neo4j

LOAD CSV WITH HEADERS FROM "http://localhost:11001/project-82664212-95dd-
4e3d-87a0-b543e47741c5/moviestowriters.csv" AS csv
MATCH (w:Writers {writerid: toInteger(csv.writerid)})
MATCH (m:Movies {movieid: toInteger(csv.movieid)})
CREATE (w)-[r:Written{addition : csv.additions}]->(m)
```

```
# Loading actor CSV file and create a relationship between them in Neo4j

LOAD CSV WITH HEADERS FROM "http://localhost:11001/project-82664212-95dd-
4e3d-87a0-b543e47741c5/ratings.csv" AS csv
MATCH (m:Movies{movieid: toInteger(csv.movieid)})
CREATE(r:Ratings{movieid:toInteger(csv.movieid),rank:toFloat(csv.rank),votes:toInteger
(csv.votes),distribution:toInteger(csv.distributionNumber)})
CREATE (r)-[:Rated]->(m);
```

```
# Loading actor CSV file and create a relationship between them in Neo4j

LOAD CSV WITH HEADERS FROM "http://localhost:11001/project-82664212-95dd-
4e3d-87a0-b543e47741c5/runningtimes.csv" AS csv
MATCH (m:Movies{movieid: toInteger(csv.movieid)})
CREATE(r:RunningTimes{movieid:toInteger(csv.movieid),time:toInteger(csv.time),additi
on:csv.addition1})
CREATE (r)-[:Running]->(m)
```

Commands required to load the CSV file in MongoDB

Start the MonoDB server in the terminal and go to the cleaned dataset directory and run the following command to load the csv file and create all the collections in the database.

```
# Go to cleaned dataset directory

cd Cleaned_dataset

 # Load all files to the database

for i in *.csv; do mongoimport -d Bigdata -c ${i%.*} --type csv --file $i --headerline ; done
```

Task-3: Neo4j

Question 1	**How many of movies have been directed by Ron Howard?**

Query	// get all directors who directed the movies
	MATCH (d:Directors) -[:Directed]->(m:Movies)
	// filter directors who has first name “Ron” and last name “Howard”
	WHERE d.fname = "Ron" AND d.lname ="Howard"
	// Returns the movie count
	RETURN count(m)

Output	"count(m)" 12
Explanation	First, we get all the directors that have directed movies and then filter them with first name = “Ron” and last name = “Howard”. Finally, we return the count of the movie.

Question 2	**Write a single query that shows both the number of female actors and the number of male actors in the dataset**

Query	//selects actors
	MATCH (n:Actors)
	WITH count(n.sex) AS number, n.sex AS sex
	//returns the male and female count
	RETURN sex, number

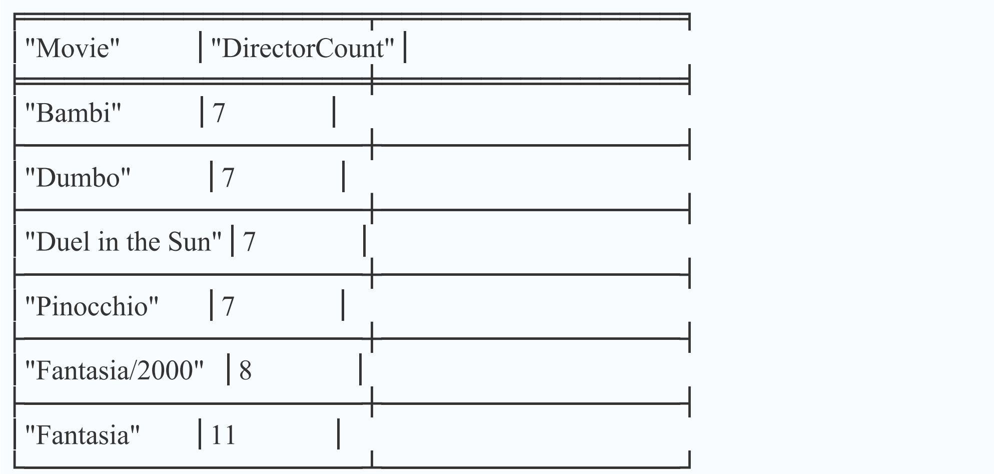
Output	"sex" "number" "M" 65794 "F" 32896
Explanation	The number of male and and female actors is calculated using the count aggregate function. The sex and the count for male and female is returned.

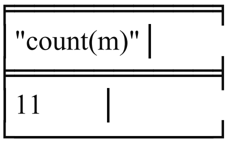
Question 3	**What is the year of the oldest movie listed in the database?**

Query	MATCH (m:Movies)
	// returns movie title and year
	RETURN (m.title) as MovieTitle, (m.year) as Year
	// orders it in asc order
	ORDER BY Year ASC
	// Limit by 1
	LIMIT 1

Output	"MovieTitle" "Year" "The Lodger" 1898

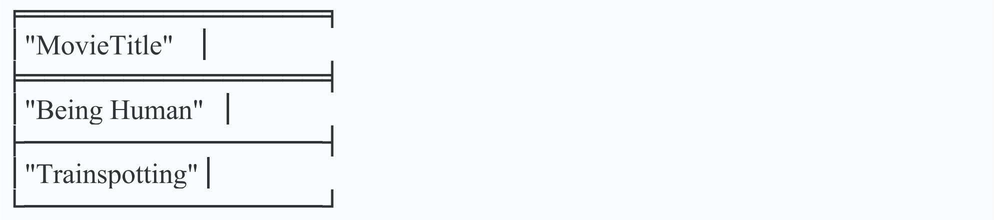
<table>
<tr><td><b>Explanation</b></td><td>To find the oldest movie, the movie title and year is returned as a list. ORDER BY is used to get the list in ascending order. LIMIT 1 returns only the first record which in this case is the oldest year.</td></tr>
</table>

<table>
<tr><td><b>Question 4</b></td><td><b>List the movie titles and number of directors involved for movies with more than 6 directors.</b></td></tr>
<tr><td><b>Query</b></td><td>MATCH (d:Directors) - [r:Directed] -> (m:Movies)
//checks director count
WITH m, count (d) as Count
//Filter the director count to 6
WHERE Count >6
// returns by asc order of director count
RETURN m.title as Movie, Count as DirectorCount
ORDER BY Count</td></tr>
<tr><td><b>Output</b></td><td>
"Movie" | "DirectorCount" |
"Bambi" | 7 |
"Dumbo" | 7 |
"Duel in the Sun" | 7 |
"Pinocchio" | 7 |
"Fantasia/2000" | 8 |
"Fantasia" | 11 |</td></tr>
<tr><td><b>Explanation</b></td><td>The condition for director count is checked whether it is greater than 6. The movie title and director count are returned in ascending order of count by using ORDER BY.</td></tr>
</table>

<table>
<tr><td><b>Question 5</b></td><td><b>Number of movies with a running time of less than 10 minutes
Note: Count all versions of a movie, not just the originals; use the time1 column for movie lengths</b></td></tr>
<tr><td><b>Query</b></td><td>MATCH (r:RunningTimes) –[: Running] -> (m:Movies)
//filters movies with time1 less than 10 minutes
WHERE r.time1 < 10
//returns the movie count
RETURN count(m);</td></tr>
<tr><td><b>Output</b></td><td>
"count(m)" |
11 |</td></tr>
<tr><td><b>Explanation</b></td><td>Selects the running times that are associated with the movies. The movies having running time less than 10 minutes are filtered and the movie count is returned.</td></tr>
</table>

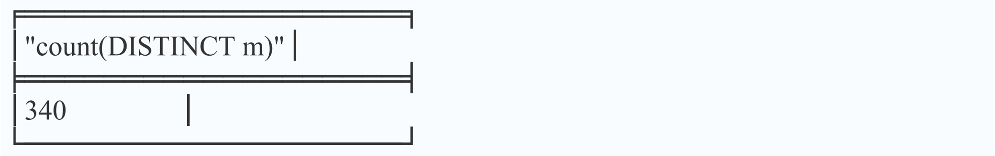
Question 6	**The movie titles which star both ‘Ewan McGregor’ and ‘Robert Carlyle’ (i.e. both actors were in the same film)**

Query	// filters movies where actors first name and last name is Ewan McGregor and Robert Carlyle
	MATCH (a1:Actors{fname:"Ewan",lname:"McGregor"})-[:ActedIn]->(m:Movies)<-[:ActedIn]-(a2:Actors{fname:"Robert",lname:"Carlyle"})
	//Returns movie list
	RETURN m.title as MovieTitle

Output	 "MovieTitle" "Being Human" "Trainspotting"
Explanation	Finds actors Ewan McGregor and Robert Carlyle that acted in same the movies and returns the movie title list.

Question 7	**How many movies have fewer male actors than female actors?**

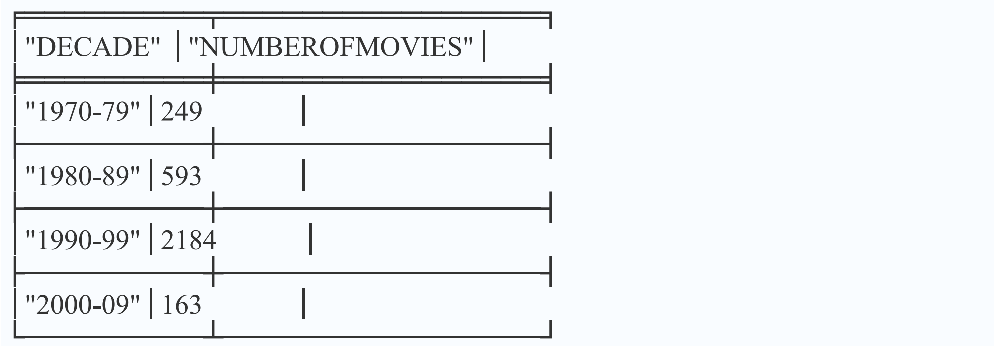
Query	MATCH (a:Actors)-[:ActedIn]->(m:Movies)
	// Filters female actors
	WHERE a.sex="F"
	OPTIONAL MATCH (b:Actors)-[:ActedIn]->(m:Movies)
	// Filters male actors
	WHERE b.sex= "M"
	WITH count(DISTINCT a) AS female, count(DISTINCT b) AS male, m
	//checks where male actor count is less than female actor count
	WHERE male < female
	//returns movie count
	RETURN count(DISTINCT m)

Output	 "count(DISTINCT m)" 340
Explanation	First select actors that are male and acted in movies. Selects actors that are female and acted in movies. Count the male actors and female actors. Check the condition where the male actor count is less than female actor count and return the movie count.

Question 8	**List the actors (male/female) that have worked together on more than 10 films, include their names and number of films they’ve co-starred in**
Query	
Output	
Explanation	

Question 9	**List the number of movies released per decade in this dataset, as listed below: (1970-79,1980-89,1990-99,2000-2009)**

Query	// Select The Movies Released In 1970 – 1979 Decade
	MATCH (m:Movies)
	WHERE m.year >= 1970 AND m.year <= 1979
	RETURN '1970-79' AS DECADE, count (m) AS NUMBEROFMOVIES
	
	// Combine It With the next decade movie
	UNION ALL
	
	// Select The Movies Released In 1970 – 1979 Decade
	MATCH (m:Movies)
	WHERE m.year >= 1980 AND m.year <= 1989
	RETURN '1980-89' AS DECADE, count(m) AS NUMBEROFMOVIES
	
	// Combine It With the next decade movie
	UNION ALL
	
	// Select The Movies Released In 1980 – 1989 Decade
	MATCH (m:Movies)
	WHERE m.year >= 1990 AND m.year <= 1999
	RETURN '1990-99' AS DECADE, count(m) AS NUMBEROFMOVIES
	
	// Combine It With the next decade movie
	UNION ALL
	
	// Select The Movies Released In 2000 – 2009 Decade
	MATCH (m:Movies)
	WHERE m.year >= 2000 AND m.year <= 2009
	RETURN '2000-09' AS DECADE, count(m) AS NUMBEROFMOVIES

Output	 │"DECADE" │"NUMBEROFMOVIES"│ │"1970-79"│249 │ │"1980-89"│593 │ │"1990-99"│2184 │ │"2000-09"│163 │
Explanation	Fist we select the movie that was released in that particular decade and then print the count of the movies followed by combine all the results using UNION ALL command.

Question 10	**Question: How many movies did Tom Hanks act in between 1993 and 1998 (inclusive)?**

Query	// find actor name is Tom Hanks
	MATCH (a:Actors {fname:"Tom", lname:"Hanks"})-[:ActedIn]->(m:Movies)
	// between 1993 and 1998(inclusive)
	WHERE 1993 <= m.year <= 1998
	// count how many movies
	RETURN count(DISTINCT m)

Output	╒═══════════════════╕ │"count(DISTINCT m)"│ ╞═══════════════════╡ │10 │ └───────────────────┘
Explanation	Since many queries need to match string, we can build an index for name first which can decrease the execution time. In this query, we need to filter actor name is Tom Hanks first, and find out all the relationship which start from him and end with a movie. Then, filter movie's released year and count how many movies left.

Question 11	**Question: Based on the average rank per movie genre, which are the bottom 3 scoring genres which have received 1000 or more votes. Give the genre and score.** **(Note: where a higher value for rank is considered a better movie)**

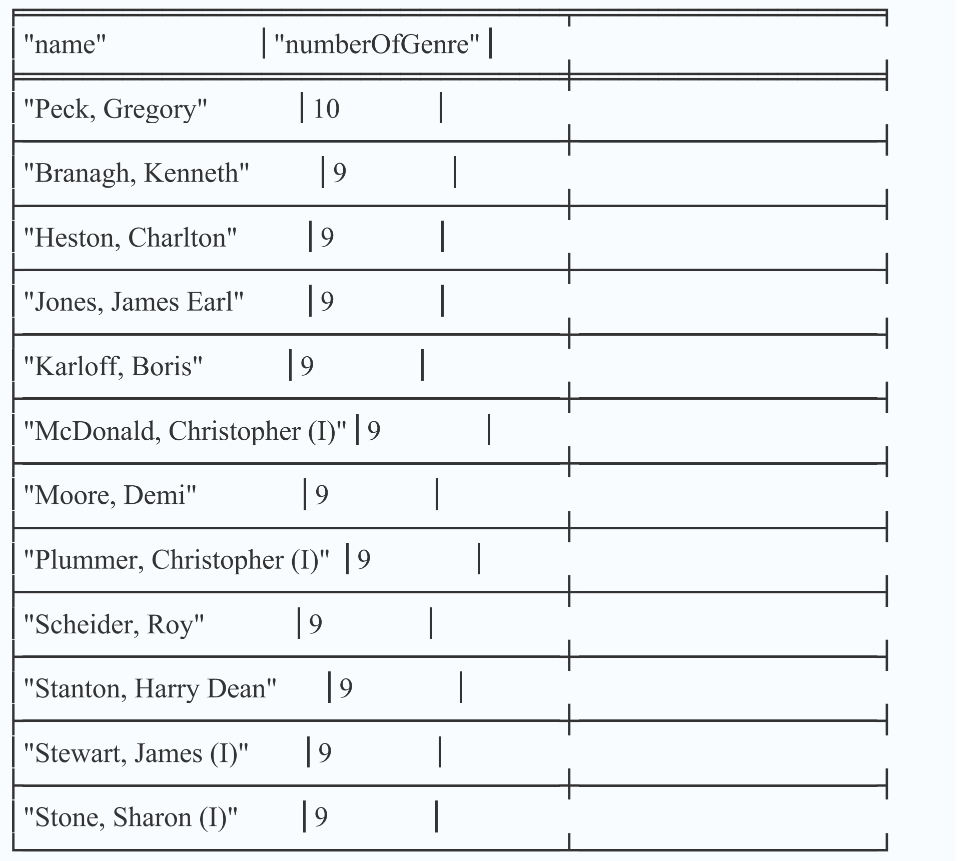
Query	MATCH (d:Directors)-[g:Directed]->(m:Movies)<-[:Rated]-(r:Ratings)
	WITH r.rank AS total, g.genre AS genre, r.votes AS vote
	// received 1000 or more
	WHERE vote >= 1000
	// auto grouped by genre and return the average score
	RETURN avg(total) AS average, genre
	// arrange in aesc order
	ORDER BY average
	// get the bottom 3
	LIMIT 3

Output	╒═════════════════╤═══════════╕ │"average" │"genre" │ ╞═════════════════╪═══════════╡ │5.744444444444444│"Sci-Fi" │ ├─────────────────┼───────────┤ │5.876470588235294│"Fantasy" │ ├─────────────────┼───────────┤ │5.9 │"Romance" │ └─────────────────┴───────────┘
Explanation	Genre attribute is stored in Directed relationship, rank and vote are stored in Ratings class. First, we need to pick out these three attributes, and then, filter by number of votes. Finally, using avg() method to get average rank per genre, since it will automictically group by the genre.

Question 12	**Show the shortest path between actors ‘Ewan McGregor’ and ‘Mark Hamill’ from the IMDB data subset. Include nodes and edges – answer can be shown as an image or text description in form (a)-[ ]->(b)-[ ]->(c)...**
Query	MATCH (u:Actors {fname:"Ewan", lname:"McGregor"}), (v:Actors {fname:"Mark", lname:"Hamill"}), p=shortestPath((u)-[*]-(v)) RETURN p
Output	(Hamill, Mark (I))-[:ActedIn]->(Star Wars: Episode VI - Return of the Jedi)<-[:ActedIn]-(Oz, Frank)-[:ActedIn]->(Star Wars: Episode I - The Phantom Menace)<-[:ActedIn]-(McGregor, Ewan)
Explanation	find out actor’s name is Ewan McGregor and actor’s name is Mark Hamill first, and using shortestPath() method to get the path between the two actors.

Quesiton 13	List all actors (male/female) that have starred in 9 or more different film genres (show names, and number of genres) and sorting the results by the number of genres (most to least) and actor name (A-Z).

Query	MATCH (a:Actors)-[:ActedIn]->(m:Movies)<-[d:Directed]-(x:Directors)
	// distinct movie genre
	WITH a.name AS name, count(DISTINCT d.genre) AS numberOfGenre
	// 9 or more different genre
	WHERE numberOfGenre >= 9
	RETURN name, numberOfGenre
	// for different number of genre, order by number of genre for same number of genre, order by the name
	ORDER BY numberOfGenre DESC, name

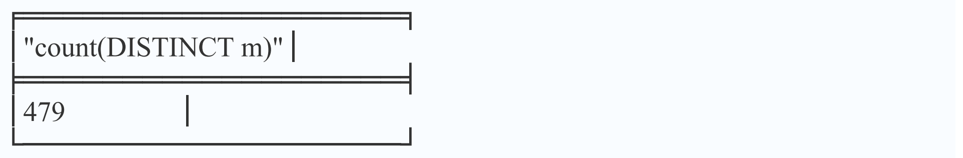
Output	

```
│"name"                    │"numberOfGenre"│
│"Peck, Gregory"           │10             │
│"Branagh, Kenneth"        │9              │
│"Heston, Charlton"        │9              │
│"Jones, James Earl"       │9              │
│"Karloff, Boris"          │9              │
│"McDonald, Christopher (I)"│9             │
│"Moore, Demi"             │9              │
│"Plummer, Christopher (I)"│9              │
│"Scheider, Roy"           │9              │
│"Stanton, Harry Dean"     │9              │
│"Stewart, James (I)"      │9              │
│"Stone, Sharon (I)"       │9              │
```

Explanation	when counting the number of different genre, we need to add DISTINCT keyword, otherwise it will return the total number of genre. When ordering the result, we can add multiple keys, in this case, the first key is ordering result by the number of different genre first, and then, by the name of actor.

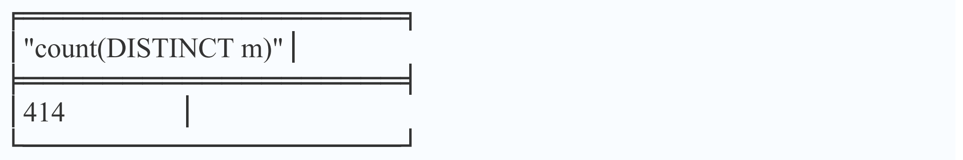
Question 14	How many movies have an actor (male/female) that also directed the movie?

Query	MATCH (a:Actors)-[:ActedIn]->(m:Movies)<-[:Directed]-(d:Directors)
	WHERE a.name=d.name
	RETURN count(DISTINCT m)

Output	 "count(DISTINCT m)" 479
Explanation	since we want to find an actor starred in also directed a movie, both Actors and Directors point at a same Movies. And filter by their name is equal.

Question 15	**How many movies have been written and directed by an actor (male or female) that they didn’t star in? (i.e. the person who wrote and directed the movie is a film star but didn’t appear in the movie)**

Query	// for a movie, find it's writer and director
	MATCH (a:Writers)-[:Written]->(m:Movies)<-[:Directed]-(d:Directors)
	// the writer's name has to be equal to director's name
	WHERE a.name=d.name
	// this person must be an actor
	MATCH (w:Actors {name: d.name})
	// those who didn't acted in the movie
	WHERE NOT (w)-[:ActedIn]->(m)
	RETURN count(DISTINCT m)

Output	 "count(DISTINCT m)" 414
Explanation	first need to find a writer written and directed a movie, and then, check whether this person is an actor. If he/she is actor, filter the actor who did not acted in the movie. Finally, counting how many actors.

Task-4. A: MongoDB

Question 1	How many of movies have been directed by Ron Howard ?

Query	db.directors.find({'fname':'Ron','lname':'Howard'},{'directorid':1,'fname':1,'lname':1,'_id':0})
	db.moviestodirectors.aggregate([
	  {
	   $match: {'directorid':'121794'}
	  },
	  {
	   $count: 'TotalMovies'
	  }
	  ])

Output	{ directorid: '121794', fname: 'Ron', lname: 'Howard' }
	{ TotalMovies: 12 }

Explanation	First, the director id is retrieved from the director's dataset using his first and last name and used to retrieve all the movies directed by him, which are 12 as per the dataset.

Question 2	Write a single query that shows both the number of female actors and the number of male actors in the dataset

Query	db.actors.aggregate([
	  {$facet: {
	   'TotalMaleActors':[
	    {$match: {'sex':'M'}},
	    {$count: 'TotalMaleActors'}
	    ],
	   'TotalFemaleActresses':[
	    {$match: {'sex':'F'}},
	    {$count: 'TotalFemaleActresses'}
	    ]
	  }},
	  {'$project': {
	   'TotalMaleActors':
	{'$arrayElemAt':['$TotalMaleActors.TotalMaleActors',0]},
	   'TotalFemaleActresses':
	{'$arrayElemAt':['$TotalFemaleActresses.TotalFemaleActresses',0]}
	  }}
	  ])

Output	{ TotalMaleActors: 65794, TotalFemaleActresses: 32896 }
Explanation	Total male and female actors counts are retrieved from the actors dataset and stored in TotalMaleActors and TotalFemaleActresses respectively. Then both are printed (projected).

Question 3	**What is the year of the oldest movie listed in the database ?**
Query	db.movies.find({},{_id:0,movieid:0}).sort({year:1}).limit(1)
Output	{ title: 'The Lodger', year: '1898' }
Explanation	The movies datasetis sorted in ascending order by year and limited to show only the first document, which retrieves the the oldest movie in dataset

Question 4	**Number of movies with a running time of less than 10 minutes. Note: Count all versions of a movie, not just the originals; use the time1 column for movie lengths**

Query	db.runningtimes.aggregate([
	{
	$project: {newTime: { $toInt: '$time1' } }
	},
	{
	$match: {newTime: { $lt:10 } }
	},
	{
	$count: 'Movies with runtime less than 10 minutes'
	}
	])

Output	{ 'Movies with runtime less than 10 minutes': 11 }
Explanation	The values of the time1 column in the movie ‘runningtimes’ dataset is converted to integer values from string. These values are then filtered by less than operator with value of 10 to retrieve the count of all the movies with runtime less than 10 minutes.

Question 5	**The movie titles which star both ‘Ewan McGregor’ and ‘Robert Carlyle’ (i.e. both actors were in the same film)**

Query	db.actors.find({$or:[{fname:'Ewan',lname:'McGregor'},{fname:'Robert',lname:'Carlyle'}]},{actorid:1,name:1,_id:0})
	db.moviestoactors.aggregate([
	{ $group:
	{
	_id: '$movieid',
	movieActors: { $addToSet: '$actorid'}
	}
	},
	{ $project:
	{
	actorMatches: { $setIsSubset: [['1035771','244663'], '$movieActors']}
	}
	},
	{ $match:
	{
	actorMatches: true
	}
	},
	{ $project:
	{
	matchedActorsWithMovie: '$_id', _id:0
	}
	}
	])
	db.movies.find({ $or: [{movieid:'2513237'}, { movieid:'1762161' }] }, { title:1, _id:0 })

Output	{ actorid: '244663', name: 'Carlyle, Robert (I)' }
	{ actorid: '1035771', name: 'McGregor, Ewan' }
	{ matchedActorsWithMovie: '2513237' }
	{ matchedActorsWithMovie: '1762161' }
	{ title: 'Being Human' }
	{ title: 'Trainspotting' }

Explanation	Actor IDs are retrieved for Ewan McGregor and Robert Carlyle from the actor’s dataset using their first and last name. Movie IDs from the moviestoactor dataset are grouped into a set called movieActors as per their actor ids. A subset with both the mentioned actor’s id is printed against the movieActors set and then matched and printed only the true values obtained from them. Finally, the retrieved movie IDs are searched in the movie databases to retrieve the names of movies.

Question 6	How many movies have fewer male actors than female actors?
Query	
Output	
Explanation	

Question 7	**How many movies did Tom Hanks act in between 1993 and 1998 (inclusive)?**

Query	db.actors.aggregate([
	// select "Tom Hanks" actor
	{$match: {fname : "Tom",lname: "Hanks"}},
	// join collection moviestoactors to get actors details
	{$lookup: { from: 'moviestoactors', localField: "actorid", foreignField: "actorid", as: 'mta',}},
	// join collection movies to get movie details
	{$lookup: { from: 'movies',localField: 'mta.movieid',foreignField: 'movieid', as: 'movieActed'}},
	{$unwind: { path: "$movieActed",}},
	// select all the movies where years are between 1993 and 1998 (years included)
	{$match: {"movieActed.year" : {$gte: 1993, $lte: 1998}}},
	// display the total count of moviesActed
	{$count: 'movieActed'}])

Output	{ "movieActed" : 10 }
Explanation	First, we need to select the Tom hanks name from the Actor collections and then join with moviestoactor to get actor detail and keep that as an object. Then join the movies to get all the movie tom Hanks acted. Next filter only movies which has years between 1993 to 1998. At the end Project the output results.

Task 4.B: Suitability of Neo4j and MongoBD on these tasks

Neo4J

Strengths	Weaknesses.
• The dataset that is connected is manageable to depict. [3] • It's easy to retrieve, traverse and navigate in the large collection of data. • It effectively describes even the semi structured data in a human readable language. [1] • It does not involve complex joins to retrieve connected data unlike in RDBMS.[3] • Data visualisation can be efficiently performed as the relations can be easily addressed. [1]	• Its difficult to load the large dataset. • Date data type not supported. [3] • Poor scalability as it only supports vertical scaling in large datasets. • Neo4j offers only limited storage facilities. [3] • It's difficult the remove the labels, nodes and relations as they are linked with each other. [1]

MongoDB

Strengths	Weaknesses.
• Easy to load the large dataset with many collections and documents in database. [2] • The data alteration is easy as it doesn't require structured data to Query unlike RDBMS. • Better performance when renormalised the data into one document. • Deep querying feature can perform better for large information retrieval as it include full-text search, regular	• Underperforms for Relational data. • It supports document-level transactions only.[2] • Difficult to write complex query from other table as it doesn't support foreign keys. • The query time increases as the length of the query increases. [2] • Does not support joins, so the querying complex questions is longer than expected.

expressions, dynamic document queries, and aggregation. [2] • The special document data structure enables more flexible designs. • The feature of replica set, automatic failover process makes it configurable for the high-level database. [2]	• MonogDB does not support data transformation which has to be done while loading the data. [2]

Challenges faced on Neo4j and MongoDB

The challenges in Neo4J is to figure out where additional information needs to added. But after going through the Neo4J documentation [4], it was suggested that the property can be added onto relationships.

When querying MongoDB, it took a long time to run the complicated query. For instance, the 10th Query took 30 mins to get the output. After following MongoDB documentation [5], it was discovered that the MongoDB Atlas cloud service provides a fast server for running queries. The final two queries was deployed on MongoDB Atlas and generated results.

There was no difference in the answers when queried in Neo4J and MongoDB.

Sustainability:

1. Overall Neo4J performed better compared to MongoDB on Speed while querying extensive datasets.
2. While querying the complex queries, Neo4J provided better design while results analysis.
3. Neo4J provided better visualisation of the results while analysing the outputs.

References

1. Sharma, M., Sharma, V.D. and Bundele, M.M., 2018, November. Performance analysis of RDBMS and no SQL databases: PostgreSQL, MongoDB and Neo4j. In *2018 3rd International Conference and Workshops on Recent Advances and Innovations in Engineering (ICRAIE)* (pp. 1-5). IEEE.
   [Accessed on: 23/03/2022]
2. Parmar, R.R. and Roy, S., 2018. MongoDB as an efficient graph database: An application of document oriented NOSQL database. In *Data Intensive Computing Applications for Big Data* (pp. 331-358). IOS Press.
   [Accessed on: 23/03/2022]
3. Kamal, S.H., Elazhary, H.H. and Hassanein, E.E., 2019. A qualitative comparison of NoSQL data stores. *Int. J. Adv. Comput. Sci. Appl*, *10*(2), pp.330-338.
   [Accessed on: 23/03/2022]
4. https://neo4j.com/docs/
   [Accessed on: 23/03/2022]
5. https://www.mongodb.com/docs/
   [Accessed on: 23/03/2022]