Task 3

MongoDB:

1.First of all we will create the data base:

-use myDB

Second we will create the collection:

- db.createCollection("Books")

An example of inserting to the collection:

db.Books.insert({"BookName":"Life by Gabi","authors":"Gabi Dunaevsky","Publisher":"Tzomet Sfarim","PublishedYear":1996,"Content":"I am 25 living in Jerusalem Student At Ariel University"})

2. var mapper =function() {

'$addFields': {

'sizes': {

'$map': {

'input': {

'$filter': {

'input': {

'$regexFindAll': {

'input': '$the\_book',

'regex': '[a-z]\*',

'options': 'i'

}

},

'as': 'val',

'cond': {

'$ne': [

'$$val.match', ''

]

}

}

},

'as': 'val',

'in': {

'$strLenCP': '$$val.match'

}

}

}

}

emit(sizes,1);

}

Varreducer(sizes,count){

Return Array.sum(count);

db.sourceData.mapReduce(

mapper,

reducer,

{

out : "amountOfWordsLength"

}

);

db. amountOfWordsLength.find()

// in this part We got some help from this site: https://www.mongodb.com/community/forums/t/mapreducer-returning-the-number-of-words-of-each-size-from-a-document-value/109422

Neo4j

match(p:Person{name:'Dani'}) with p AS Dani match(person:Person) where person.age > Dani.age and (Dani)<-[:friend\*..3]-(person) with distinct collect(person)as DaniesFriends match(post:Post),(person:Person) where(person:Person{name:'Dani'})-[:publish]->(post)with collect(post) As daniesPosts match(post:daniesPosts) where all(x IN daniesPosts where(person)-[:likes]->(x)) return person

X-path:

//Country[@count(/city/num)>1000000]/name

Stream:

1.This is the code that we wrote in java on the university files for the first part.

students.stream().forEach(s->{  
 double avg = s.getGrades().stream().mapToDouble(g->g.getValue()).average().getAsDouble();  
 System.*out*.println(  
 s.getName() + ": " + avg  
 );  
});

This is the results on the file for the first part:

Text

Description automatically generated

2.This is the code to the second part:

System.*out*.println("-------- Part B----------------");  
 courses.stream().forEach(c->{  
 List<Double> GradesInCourse = new LinkedList<Double>();  
 students.stream().forEach(s->{  
 double gr = s.getGrades().stream().filter(name -> name.getCourse().getId() == c.getId()).mapToDouble(g -> g.getValue()).average().orElse(-1);  
 if(gr !=-1) {  
 GradesInCourse.add(s.getGrades().stream().filter(name -> name.getCourse().getId() == c.getId()).mapToDouble(g -> g.getValue()).average().orElseThrow());  
 }  
 });  
 System.*out*.println(  
 c.getName() + ": " + GradesInCourse.stream().mapToDouble(d->d).average().orElse(0.0)  
 );  
  
// Our id's - GabiDunaevsky - 208407379,Idan Kaminetzky - 208588392  
 });

And this is the results:

Text

Description automatically generated

This is the main directory with the code:



SPARQL, RDF:

1.

|  |  |  |
| --- | --- | --- |
| Object | Predicate | Subject |
| Dani | F\_name | 123 |
| Cohen | L\_name | 123 |
| 24 | Age | 123 |
| 333 | Father\_id | 123 |
| Michal | F\_name | 333 |
| Levi | L\_name | 333 |
| 56 | Age | 333 |
| 444 | Father\_id | 333 |
| Reuven | F\_name | 444 |
| Levi | L\_name | 444 |
| 80 | Age | 444 |
| 111 | Father\_id | 444 |

2.

SELECT ?grandson WHERE{

?grandson Age 25.

?grandson Father\_id ?son.

?son Father\_id 444.

}

Second option:

SELECT ?grandson WHERE{

?grandson Father\_id/ Father\_id 444.

?grandson Age 25.}

Note: if we don't know Reuven's id we will do:

SELECT ?grandson WHERE{

?grandson Age 25.

?grandson Father\_id ?son.

?son Father\_id ?id.

?id F\_name Reuven.

}

Or a second option:

SELECT ?grandson WHERE{

?grandson Father\_id/ Father\_id/ F\_name Reuven.

?grandson Age 25.}

**\*There are another several options for this query!!**

TF-IDF:

Q: what day is today

1: Very sunny day outside.

Score: 1/4\*log(4/2) =**0.07525**

2: Today is Thursday.

Score: 1/3\*log(4/3) + 1/3\*log(4/2) = **0.14198**

3: What a pleasant day today!

Score: 1/5\*log(4/1)+1/5\*log(4/2)+1/5\*log(4/3)=**0.2056**

4: Today is Jonathan's birthday.

Score: 1/4\*log(4/3)+1/4\*log(4/2)=**0.10649**

**There for the answer is:**

**3>2>4>1**