Database Technologies (TBD)
2021/2022

NoSQL Assignment
Teaching Service

Henrique Reis Sendim Rodrigues (UP201606462)
João Carlos Machado Rocha Pires (UP201806079)
Mariana Catarina Pereira Soares (UP201605775)

U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Contents

1. Problem description

Our problem consists on a database with information regarding the distribution of teaching service in a faculty.

There are courses (table XUCS), described by a code (codigo), a designation (designacao), an acronym (sigla_uc) and a program (curso). Courses have occurrences in several years. Each occurrence is recorded by a row in the table XOCORRENCIAS, with information on the course code (codigo), academic year (ano_letivo), period of classes (periodo, that may be A-annual, 1S- first semester, 1T- first trimester, etc.), number of enrolled students (inscritos), students with distributed assessment (com_frequencia), number of approved (aprovados), course goals (objetivos) and content (conteudo), and department in charge (departamento).

Each occurrence may have one or more class types (T-theoretic, P-practical, L-laboratory, TP- theoretic/practical, OT- tutorial guidance). Each class type for an occurrence is recorded on table XTIPOSAULA with the number of similar classes (turnos), the number of week hours for each class (horas_turno), and in some cases the number of weekly classes (n_aulas).

The table XDSD records the teaching service distribution, in each semester, for each professor. More specifically, it records, for each class type of an occurrence, how many weekly hours are assigned to that professor. If a professor is teaching, in a single class, more than one course at the same time, for example from different programs, the weight of that course, in the perspective of the professor, may be less than 1 and recorded in attribute fator. Otherwise, the attribute fator will be 1. From the program perspective, the attribute fator is ignored. The attribute ordem enables listing the set of professors of a specific course occurrence in a specific order.

The professors are recorded in the table XDOCENTES with a number (nr), a name (nome), an acronym (sigla), a category code (categoria), a given name (proprio), a family name (apelido), and a status (estado: A-ativo, NA-não ativo, R-reformado).

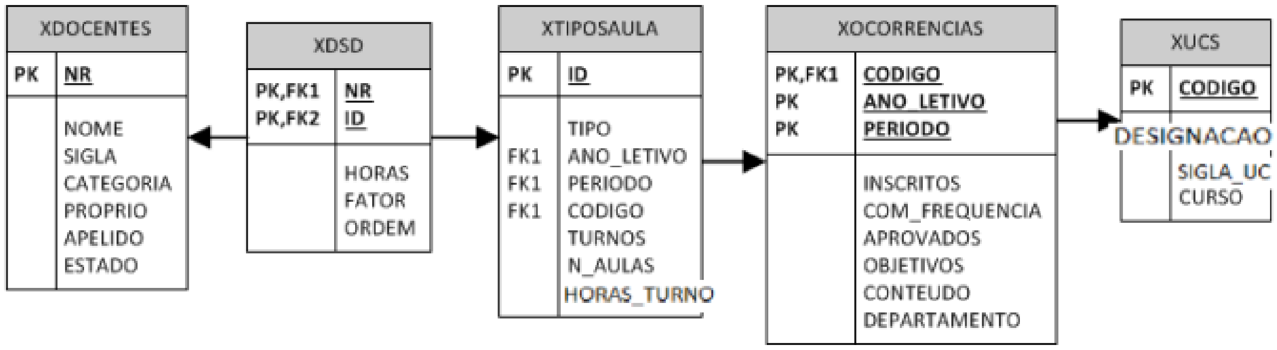
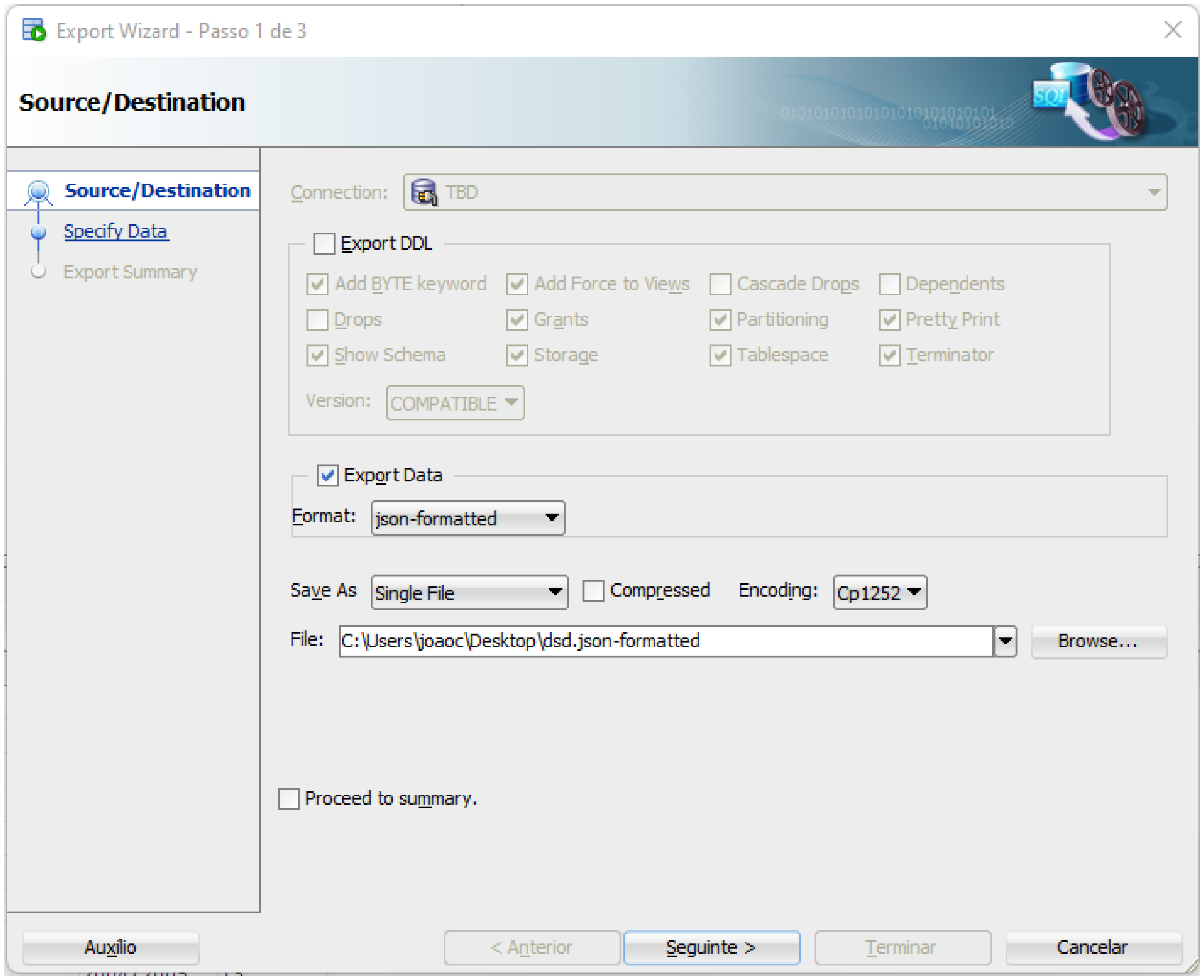


Figure 1.1: Relational model for the case Teaching Service.

2. MongoDB version and data migration

In order to create a MongoDB version of the previously mentioned database, we started by exporting each table from the Oracle Database into a .json file, using the export function of Oracle, as shown bellow:



Export Wizard - Passo 1 de 3

Source/Destination

Connection: TBD

☐ Export DDL

☑ Add BYTE keyword ☑ Add Force to Views ☐ Cascade Drops ☐ Dependents
☐ Drops ☑ Grants ☑ Partitioning ☑ Pretty Print
☑ Show Schema ☑ Storage ☑ Tablespace ☑ Terminator

Version: COMPATIBLE

☑ Export Data

Format: json-formatted

Save As Single File ☐ Compressed Encoding: Cp1252

File: C:\Users\joaoc\Desktop\dsd.json-formatted

☐ Proceed to summary.

Figure 2.1: Example of export for the table 'dsd'.

Note: Please note that the exported format is .json-formatted and not .json. However, we converted the exported files to .json just by changing the file extension name.

Then, we created a Python script that reads the content of each .json file (one per table) and stores the information in a nested-document format, obtaining just a single .json file with 27385 documents.

Each one of this documents results of storing the information of the database in a nested format, as explained in the following steps:

- The 'dsd' table served as the base table.
- The attribute 'nr' in the 'dsd' table was replaced by a nested structure containing the information about the professor, joined by the attribute with the same name in the table 'docentes'.
- The attribute 'codigo' in the table 'ocorrencias' was replaced by the entire structure of the 'ucs' table correspondent to the same 'codigo' entry.
- The attributes 'ano_letivo', 'periodo' and 'codigo' in the table 'tiposaula' were replaced by the corresponding information of the table 'ocorrencias'
- Last but not least, the 'id' attribute at the table 'dsd' was replaced by the correspondent structure of the table 'tiposaula'

After the steps above, each document of the 27385 has the following structure:

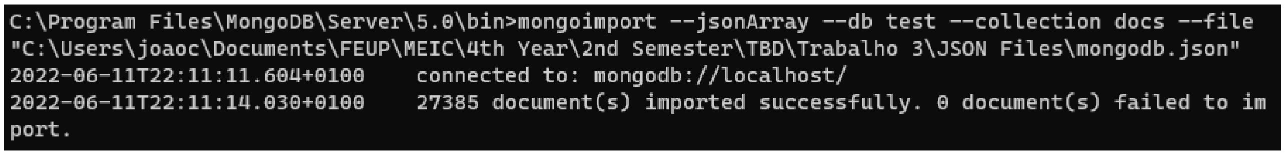
```json
{
    "horas": 5,
    "fator": 1,
    "ordem": 0,
    "docente": {
        "nr": 208963,
        "nome": "Alberto Manuel Carneiro Sereno",
        "sigla": "AMS",
        "categoria": 110,
        "proprio": "Alberto Manuel Carneiro",
        "apelido": "Sereno",
        "estado": "R"
    },
    "tipoaula": {
        "id": 3529,
        "tipo": "T",
        "turnos": 1,
        "n_aulas": "",
        "horas_turno": 1,
        "ocorrencia": {
            "ano_letivo": "1998/1999",
            "periodo": "A",
            "inscritos": 63,
            "com_frequencia": 55,
            "aprovados": 55,
                "objetivos": "",
                "conteudo": "",
                "departamento": "DEQ",
                "uc": {
                    "codigo": "EQ500",
                    "designacao": "Anteprojecto(anual)",
                    "sigla_uc": "AP",
                    "curso": 331
                }
            }
        }
}
```

As we can see from above, the information regarding the 'uc' is nested in a field with the same name, and the same happens to 'ocorrencia', 'tipoaula' and 'docente'.

This leads to a redundancy in the 'docente' and 'tipoaula' information (as well as 'uc' and 'ocorrencias' info since they're nested inside 'tipoaula'). This redundancy comes from having as base table the 'dsd' one, which means that, for the same pair 'dsd'-'tipoaula', the information will be stored the same number of times as the number of corresponding relations with 'docente'. This has an impact on the database querying process since most of the times we have to check for unique/distinct values, but it was the best solution we found to design the MongoDB version.

Finally, to upload the documents to MongoDB, we used the tool 'mongoimport' and the following command, that gets the file previously generated by the Python script and uploads to the 'docs' collection to the 'test' database with all the documents.

```
mongoimport --jsonArray --db test --collection docs --file "mongodb.json"
```

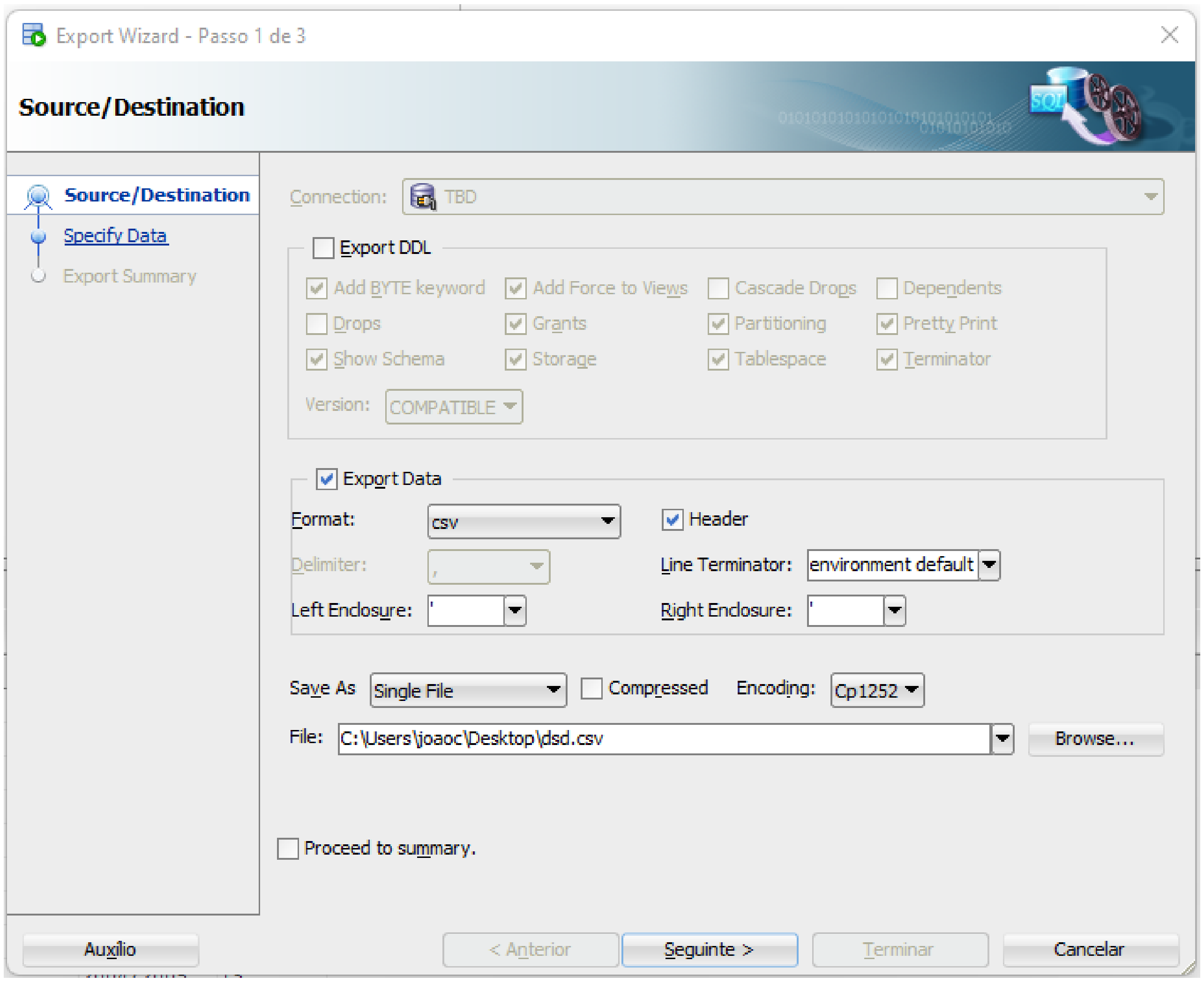


```
C:\Program Files\MongoDB\Server\5.0\bin>mongoimport --jsonArray --db test --collection docs --file
"C:\Users\joaoc\Documents\FEUP\MEIC\4th Year\2nd Semester\TBD\Trabalho 3\JSON Files\mongodb.json"
2022-06-11T22:11:11.604+0100    connected to: mongodb://localhost/
2022-06-11T22:11:14.030+0100    27385 document(s) imported successfully. 0 document(s) failed to im
port.
```

Figure 2.2: Result of the MongoDB import.

3. Neo4j version and data migration

For the Neo4j version, the first step was similar to the one made for the MongoDB. We started by exporting the data from the Oracle tables using the export functionality, but this time in the .csv format, as shown bellow:


Export Wizard - Passo 1 de 3

Source/Destination

Connection: TBD

☐ Export DDL

✓ Add BYTE keyword	✓ Add Force to Views	☐ Cascade Drops	☐ Dependents
☐ Drops	✓ Grants	✓ Partitioning	✓ Pretty Print
✓ Show Schema	✓ Storage	✓ Tablespace	✓ Terminator

Version: COMPATIBLE

✓ Export Data

Format: csv ✓ Header

Delimiter: , Line Terminator: environment default

Left Enclosure: ' Right Enclosure: '

Save As Single File ☐ Compressed Encoding: Cp1252

File: C:\Users\joaoc\Desktop\dsd.csv

☐ Proceed to summary.

Figure 3.1: Example of export for the table 'dsd'.

Note: For the file 'ocorrencias.csv', exceptionally, we needed to do an extra step. Since on the Neo4j data migration process we had a problem with the attributes 'objetivo' and 'conteudo', we had to remove those two columns from all the entries in the .csv file. This did not affect the queries, since no one uses it, but we recognize it's not a good solution. However, due to time constraints, we didn't find any other solution to this problem and this was the only way around we found.

Then, inside Neo4j, we created a project and a local DBMS inside of it. We moved the 5 .csv files to the project folder and then changed the following two lines in the project settings:

```
dbms.memory.heap.initial_size=3000m
dbms.memory.heap.max_size=5000m
```

Last but not least, we executed, in order, the following instructions:

```
LOAD CSV WITH HEADERS FROM 'file:///UCS_DATA_TABLE.csv' AS line
CREATE (:UC {CODIGO: line.CODIGO, DESIGNACAO: line.DESIGNACAO, SIGLA_UC: line.SIGLA_UC,
    CURSO: toInteger(line.CURSO)});

:auto USING PERIODIC COMMIT 500
LOAD CSV WITH HEADERS FROM 'file:///OCORRENCIAS_DATA_TABLE.csv' AS line
CREATE (p:Ocorrencias {CODIGO : line.CODIGO, ANO_LETIVO: line.ANO_LETIVO, PERIODO: line.
    PERIODO,
INSCRITOS: toInteger(line.INSCRITOS), COM_FREQUENCIA: toInteger(line.COM_FREQUENCIA),
APROVADOS: toInteger(line.APROVADOS), DEPARTAMENTO: line.DEPARTAMENTO});

MATCH
  (u:UC),
  (p:Ocorrencias)
WHERE u.CODIGO = p.CODIGO
CREATE (p)-[r:RELTYPE]->(u)
return type(r);

:auto USING PERIODIC COMMIT 500
LOAD CSV WITH HEADERS FROM 'file:///TIPOSAULA_DATA_TABLE.csv' AS line
CREATE (t:TiposAula {ID: toInteger(line.ID), CODIGO: line.CODIGO, ANO_LETIVO : line.
    ANO_LETIVO, PERIODO : line.PERIODO,
TIPO: line.TIPO, TURNOS: toInteger(line.TURNOS), N_AULAS: line.N_AULAS, HORAS_TURNO:
    toInteger(line.HORAS_TURNO)});

MATCH
  (t:TiposAula),
  (p:Ocorrencias)
WHERE t.ANO_LETIVO = p.ANO_LETIVO and t.PERIODO = p.PERIODO and p.CODIGO = t.CODIGO
CREATE (t)   [:Occurs]->(p);

MATCH
  (t:TiposAula),
  (u:UC)
WHERE t.CODIGO = u.CODIGO
CREATE (t)   [r:Occur]->(u)
RETURN type(r);

LOAD CSV WITH HEADERS FROM 'file:///DOCENTES_DATA_TABLE.csv' AS line
CREATE (:Docentes { NR: toInteger(line.NR), NOME: line.NOME, SIGLA: line.SIGLA,
CATEGORIA: toInteger(line.CATEGORIA),PROPRIO: line.PROPRIO, APELIDO: line.apelido,ESTADO
    : line.ESTADO});

:auto USING PERIODIC COMMIT 500
```

```
LOAD CSV WITH HEADERS FROM 'file:///DSD_DATA_TABLE.csv' AS line
CREATE (x:Dsd { ID : toInteger(line.ID), NR: toInteger(line.NR), HORAS: toInteger(line.
    HORAS), FATOR: toInteger(line.FATOR), ORDEM: toInteger(line.ORDEM)});

MATCH
  (x:Dsd),
  (t:TiposAula)
WHERE t.ID = x.ID
CREATE (x)   [r:Type]->(t)
return type(r);

MATCH
  (x:Dsd),
  (d:Docentes)
WHERE d.NR = x.NR
CREATE (x)   [r:Records]->(d)
return type(r);
```

Each one of these instructions loaded the contents of the .csv files to the database, creating a list of nodes for each one with the corresponding fields and establishing the necessary relations between nodes, following the same logic presented in the relational model.

4. Queries

In this chapter, in each section, we present the written query, the MongoDB and Neo4j queries, the answer to the query and a brief explanation of the procedure.

4.1 Query a)

How many class hours of each type did the program 233 got in year 2004/2005?

4.1.1 MongoDB

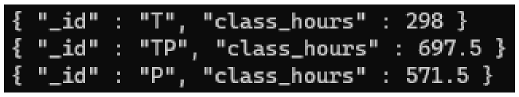
For this first query, since there's a redundancy in the information of 'tipoaula' (as explained in the chapter 2), we first needed to create a collection named 'tiposaula' where we stored as many documents as the distinct combinations of class type, number of classes, number of hours per class and id, considering only the entries corresponding to the academic year '2004/2005' and to the course 233.

Then, we queried that newly created collection to get the total number of hours per class type. To obtain the total number of hours, we multiplied for each entry the number of classes for the number of hours per class, summing then the result grouped by class type.

```
db.tiposaula.insert(db.docs.aggregate([
    { "$match" : { "tipoaula.ocorrencia.uc.curso" : 233, "tipoaula.
   ocorrencia.ano_letivo": "2004/2005"} } ,
    { "$group" : { "_id" : {"tipo":"$tipoaula.tipo", "turnos":"$tipoaula.
   turnos", "horas_turno":"$tipoaula.horas_turno", "id":"$tipoaula.id"}}
   },
]).toArray())

db.tiposaula.aggregate([
    { "$group" : { "_id" : "$_id.tipo", "class_hours" : { "$sum" : { "
   $multiply" : ["$_id.turnos", "$_id.horas_turno"] }}} },
    { "$project" : { "_id": 1, "class_hours" : 1 } },
])
```

The result of the above query is the following:



```
{ "_id" : "T", "class_hours" : 298 }
{ "_id" : "TP", "class_hours" : 697.5 }
{ "_id" : "P", "class_hours" : 571.5 }
```

Figure 4.1: Query a) - MongoDB Answer.

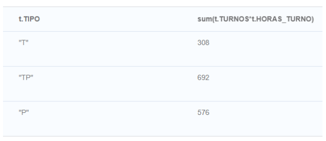
Note: There's a difference between this answer and the answer obtained for the same query in the SQL and Neo4j databases. The reason for that is that, in the MongoDB version, the entries of the table 'tiposaula' were only kept if there's a match with an entry in the table 'dsd' (on the attribute 'id). Otherwise, the entries that standalone without a connection to 'dsd' were ignored, which means they're not stored in MongoDB.

4.1.2 Neo4j

For the Neo4j query, the logic was pretty much the same as if it was in SQL. After performing a match between 'TiposAula' and 'UC' and applying the constraints of 'CURSO' equal to 233 and 'ANO_LETIVO' equal to '2004/2005', we returned the type of class and the sum of the number of classes times the number of hours per class.

```
Match (t:TiposAula)-[:Occur]->(u:UC)
Where u.CURSO = 233 AND t.ANO_LETIVO = '2004/2005'
Return t.TIPO, sum(t.turnos*t.horas_turno)
```

The result of the above query is the following:



t.TIPO	sum(t.TURNOS*t.HORAS_TURNO)
"T"	308
"TP"	692
"P"	576

Figure 4.2: Query a) - Neo4j Answer.

4.2 Query b)

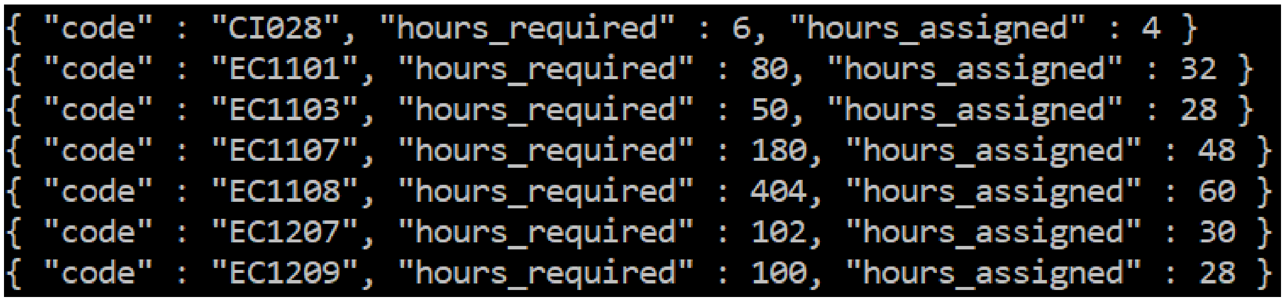
Which courses (show the code, total class hours required, total classes assigned) have a difference between total class hours required and the service actually assigned in year 2003/2004?

4.2.1 MongoDB

The aggregate pipeline for this query begins with a match on the required ano letivo 2003/2004. Then the pipeline groups the data by codigo and sums the product of horas_turno by the number of turnos (if the number of horas_turno or turnos is equal to "", i.e. no value defined, we considered 0) to get the total hours required and sums all the hours assigned. The following projection adds a new boolean field that is true if the total hours assigned and total hours required are different, which will be used by the match to filter the true results. The final projection is only used to keep the questions' required fields.

```
db.docs.aggregate([{
 $match: {
  'tipoaula.ocorrencia.ano_letivo': '2003/2004'
 }
}, {
 $group: {
  _id: '$tipoaula.ocorrencia.uc.codigo',
  total_hours_required: {
   $sum: {
    $cond: [
     {
      $eq: [
       '$tipoaula.horas_turno',
       ''
      ]
     },
     0,
     {
      $cond: [
       {
        $eq: [
         '$tipoaula.turnos',
         ''
        ]
       },
       0,
       {
        $multiply: [
         '$tipoaula.horas_turno',
         '$tipoaula.turnos'
        ]
       }
      ]
     }
    ]
   }
  },
  total_hours_assigned: {
   $sum: '$horas'
  }
 }
}, {
 $project: {
  code: '$_id',
  hours_required: '$total_hours_required',
  hours_assigned: '$total_hours_assigned',
  diff: {
   $ne: [
    '$total_hours_assigned',
    '$total_hours_required'
   ]
  }
 }
}, {
 $match: {
  diff: true
 }
}, {
 $project: {
  _id: 0,
  code: 1,
  hours_required: 1,
  hours_assigned: 1
 }
}, {
 $sort: {
  code: 1
 }
}]);
```

Some of the results of the above query are the following:



```
{ "code" : "CI028", "hours_required" : 6, "hours_assigned" : 4 }
{ "code" : "EC1101", "hours_required" : 80, "hours_assigned" : 32 }
{ "code" : "EC1103", "hours_required" : 50, "hours_assigned" : 28 }
{ "code" : "EC1107", "hours_required" : 180, "hours_assigned" : 48 }
{ "code" : "EC1108", "hours_required" : 404, "hours_assigned" : 60 }
{ "code" : "EC1207", "hours_required" : 102, "hours_assigned" : 30 }
{ "code" : "EC1209", "hours_required" : 100, "hours_assigned" : 28 }
```

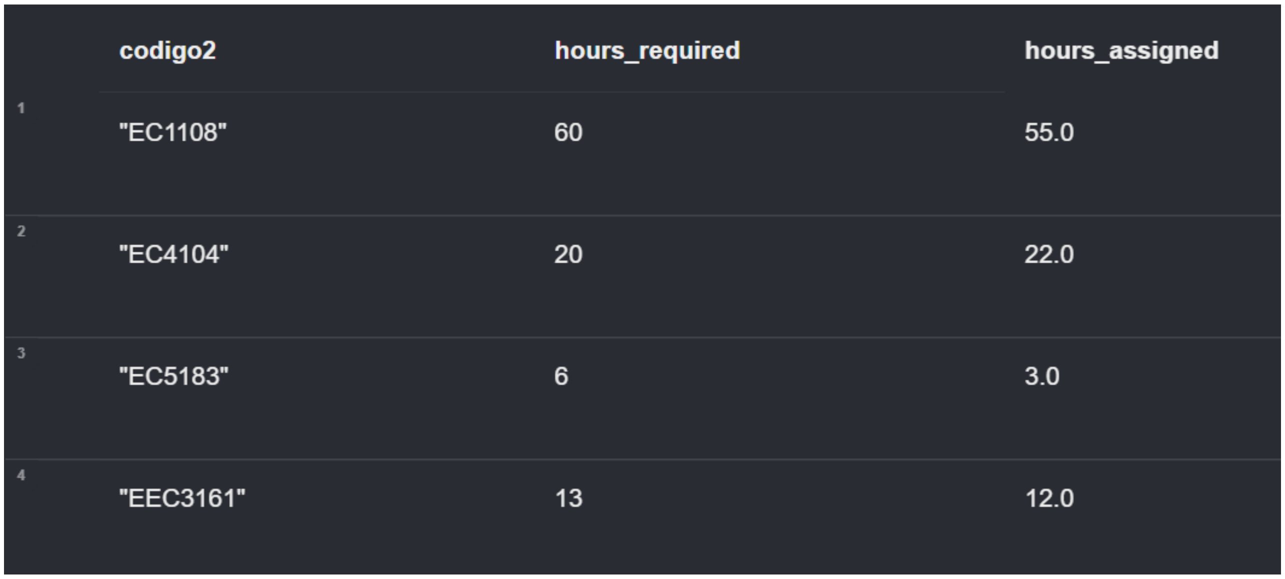
Figure 4.3: Query b) - MongoDB Answer.

4.2.2 Neo4j

The query below has two matches. The first one is used to get the required hours by summing the multiplication of horas_turno and turnos for 2003/2004. The second one gets the hours_assigned by summing all the hours for 2003/2004. Then we check the courses that have the hours required and the hours assigned differently.

```
Match (t:TiposAula) -[:Occur] - >(u1:UC)
Where t.ANO_LETIVO = '2003/2004'
WITH u1.CODIGO as codigo1, sum(t.HORAS_TURNO * t.TURNOS) as hours_required
Match (d:Dsd)-[:Type]->(t:TiposAula)-[:Occur] - >(u2:UC)
Where t.ANO_LETIVO = '2003/2004'
WITH u2.CODIGO as codigo2, codigo1, round(sum(d.HORAS), 10) as hours_assigned,
    hours_required
WHERE codigo2 = codigo1 AND hours_required <> hours_assigned
RETURN codigo2, hours_required, hours_assigned
ORDER BY codigo1;
```

Some of the results of the above query is the following:



	codigo2	hours_required	hours_assigned
1	"EC1108"	60	55.0
2	"EC4104"	20	22.0
3	"EC5183"	6	3.0
4	"EEC3161"	13	12.0

Figure 4.4: Query b) - Neo4jAnswer.

4.3 Query c)

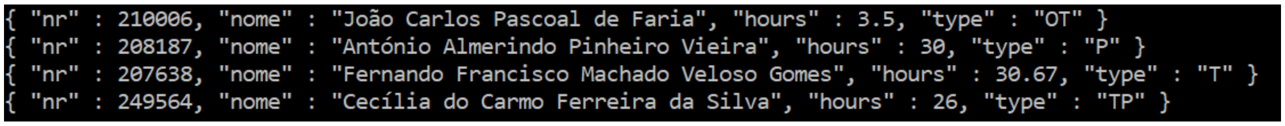
Who is the professor with more class hours for each type of class, in the academic year 2003/2004? Show the number and name of the professor, the type of class and the total of class hours times the factor.

4.3.1 MongoDB

This pipeline starts by matching the results that have the wanted ano_letivo 2003/2004. Then, the results are grouped by the docente's nr and the tipo, keeping the nome and summing all the horas associated with that docente for that tipo. After that, we use a sort to put the data in decreasing order by hours, which means the first result for each type will be the professor with the most class hours. To acquire the desired response, we group the results by tipo and keep the first entry of each result. The final projection is only used to keep the questions' required fields.

```
db.docs.aggregate([{
 $match: {
  'tipoaula.ocorrencia.ano_letivo': '2003/2004'
 }
}, {
 $group: {
  _id: {
   docente: '$docente.nr',
   type: '$tipoaula.tipo'
  },
  nome: {
   $first: '$docente.nome'
  },
  hours: {
   $sum: '$horas'
  }
 }
}, {
 $sort: {
  hours: -1
 }
}, {
 $group: {
  _id: '$_id.type',
  nr: {
   $first: '$_id.docente'
  },
  nome: {
   $first: '$nome'
  },
  hours: {
   $first: '$hours'
  }
 }
}, {
 $project: {
  _id: 0,
  type: '$_id',
  nr: 1,
  nome: 1,
  hours: 1
 }
}]);
```

The result of the above query is the following:



```
{ "nr" : 210006, "nome" : "João Carlos Pascoal de Faria", "hours" : 3.5, "type" : "OT" }
{ "nr" : 208187, "nome" : "António Almerindo Pinheiro Vieira", "hours" : 30, "type" : "P" }
{ "nr" : 207638, "nome" : "Fernando Francisco Machado Veloso Gomes", "hours" : 30.67, "type" : "T" }
{ "nr" : 249564, "nome" : "Cecília do Carmo Ferreira da Silva", "hours" : 26, "type" : "TP" }
```

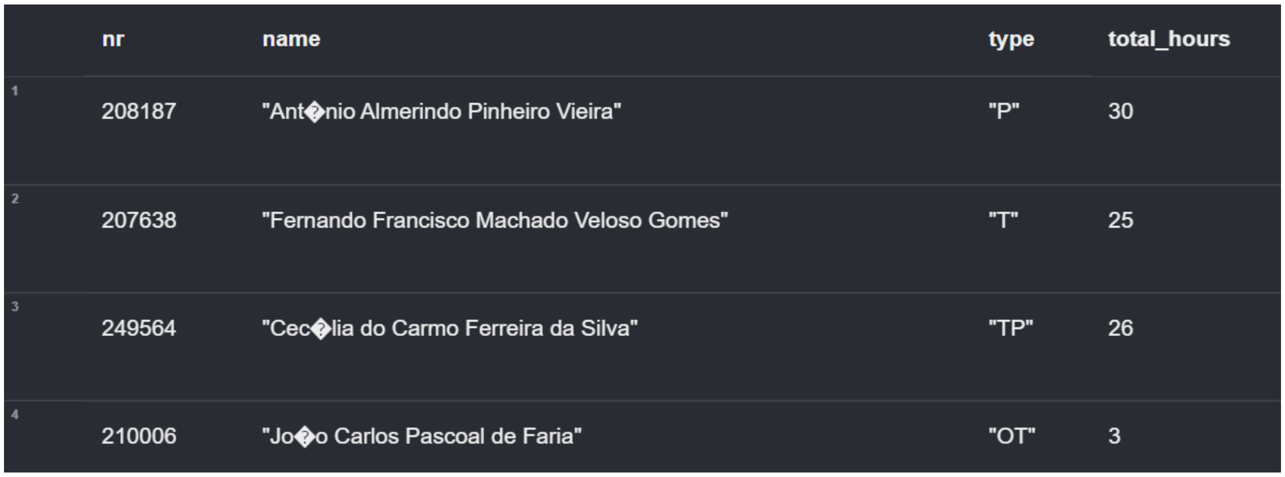
Figure 4.5: Query c) - MongoDB Answer.

4.3.2 Neo4j

The query below performs two matches. The first one finds the maximum total hours, by type, a professor has been assigned to in the academic year 2003/2004. The second one gets the total hours by the professor by each type. Then, we match the types and the hours to find out the numbers and names of the professors with the most class hours for each type of class.

```
Match(e1:Docentes)<-[z:Records]-(d1:Dsd)-[y:Type]->(t1:TiposAula)
WHERE t1.ANO_LETIVO = '2003/2004'
WITH d1.NR as nr, e1.NOME as nome, t1.TIPO as tipo1, sum(d1.HORAS) as total_time
WITH tipo1, max(total_time) as max_total
Match(e2:Docentes)<-[z:Records]-(d2:Dsd)-[y:Type]->(t2:TiposAula)
WHERE t2.ANO_LETIVO = '2003/2004'
WITH d2.NR as nr, e2.NOME as nome, tipo1, t2.TIPO as tipo2, sum(d2.HORAS) as total_horas
    , max_total
WHERE tipo1 = tipo2 AND total_horas = max_total
RETURN nr, nome as name, tipo1 as type, total_horas as total_hours;
```

The result of the above query is the following:



	nr	name	type	total_hours
1	208187	"Ant�nio Almerindo Pinheiro Vieira"	"P"	30
2	207638	"Fernando Francisco Machado Veloso Gomes"	"T"	25
3	249564	"Cec�lia do Carmo Ferreira da Silva"	"TP"	26
4	210006	"Jo�o Carlos Pascoal de Faria"	"OT"	3

Figure 4.6: Query c) - Neo4jAnswer.

4.4 Query d)

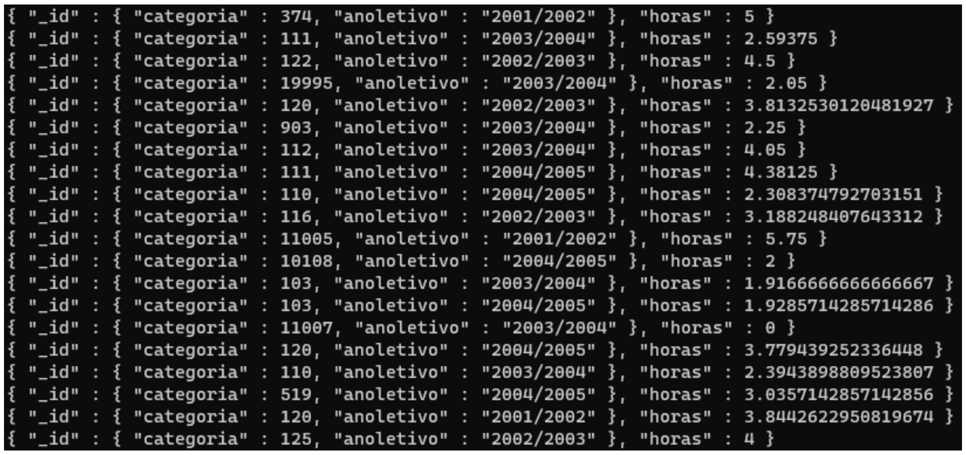
Which is the average number of hours by professor by year in each category, in the years between 2001/2002 and 2004/2005?

4.4.1 MongoDB

For this query, firstly, we restrict the pipeline only to return the required years, then we use the group function with the category, the year, and the average of the hours worked, and finally, using the project, we show the required fields for the answer.

```
db.docs.aggregate([
    { "$match" : {$or: [{ "tipoaula.ocorrencia.ano_letivo": {$eq: "2001/2002"}}, { "tipoaula.ocorrencia.ano_letivo": {$eq: "2002/2003"}}, { "tipoaula.ocorrencia.ano_letivo": {$eq: "2003/2004"}}, { "tipoaula.ocorrencia.ano_letivo": {$eq: "2004/2005"}}]}},
    { "$group" : { "_id" : {"categoria": "$docente.categoria", "anoletivo": "$tipoaula.ocorrencia.ano_letivo"}, "horas": { $avg: "$horas" }}},
    { "$project" : {"horas": 1, "categoria": 1, "anoletivo": 1} },
])
```

The result of the above query is the following:



```
{ "_id" : { "categoria" : 374, "anoletivo" : "2001/2002" }, "horas" : 5 }
{ "_id" : { "categoria" : 111, "anoletivo" : "2003/2004" }, "horas" : 2.59375 }
{ "_id" : { "categoria" : 122, "anoletivo" : "2002/2003" }, "horas" : 4.5 }
{ "_id" : { "categoria" : 19995, "anoletivo" : "2003/2004" }, "horas" : 2.05 }
{ "_id" : { "categoria" : 120, "anoletivo" : "2002/2003" }, "horas" : 3.8132530120481927 }
{ "_id" : { "categoria" : 903, "anoletivo" : "2003/2004" }, "horas" : 2.25 }
{ "_id" : { "categoria" : 112, "anoletivo" : "2003/2004" }, "horas" : 4.05 }
{ "_id" : { "categoria" : 111, "anoletivo" : "2004/2005" }, "horas" : 4.38125 }
{ "_id" : { "categoria" : 110, "anoletivo" : "2004/2005" }, "horas" : 2.308374792703151 }
{ "_id" : { "categoria" : 116, "anoletivo" : "2002/2003" }, "horas" : 3.188248407643312 }
{ "_id" : { "categoria" : 11005, "anoletivo" : "2001/2002" }, "horas" : 5.75 }
{ "_id" : { "categoria" : 10108, "anoletivo" : "2004/2005" }, "horas" : 2 }
{ "_id" : { "categoria" : 103, "anoletivo" : "2003/2004" }, "horas" : 1.9166666666666667 }
{ "_id" : { "categoria" : 103, "anoletivo" : "2004/2005" }, "horas" : 1.9285714285714286 }
{ "_id" : { "categoria" : 11007, "anoletivo" : "2003/2004" }, "horas" : 0 }
{ "_id" : { "categoria" : 120, "anoletivo" : "2004/2005" }, "horas" : 3.779439252336448 }
{ "_id" : { "categoria" : 110, "anoletivo" : "2003/2004" }, "horas" : 2.3943898809523807 }
{ "_id" : { "categoria" : 519, "anoletivo" : "2004/2005" }, "horas" : 3.0357142857142856 }
{ "_id" : { "categoria" : 120, "anoletivo" : "2001/2002" }, "horas" : 3.8442622950819674 }
{ "_id" : { "categoria" : 125, "anoletivo" : "2002/2003" }, "horas" : 4 }
```

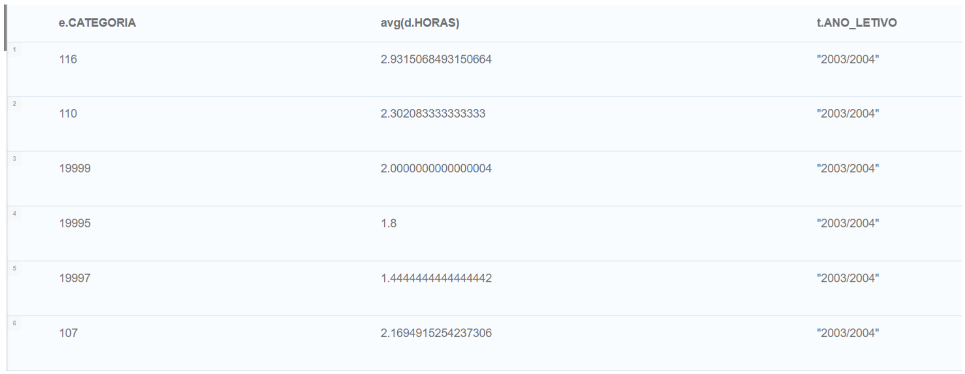
Figure 4.7: Query d) - Subset of the MongoDB Answer.

4.4.2 Neo4j

For the neo4j, the query is very similar to the above and to the SQL one, there is a match to get the required information than a where to restrict the selected years and finally a return to show the CATEGORIA, the average of HORAS and the year.

```
Match(e:Docentes)<-[z:Records]-(d:Dsd)-[y:Type]->(t:TiposAula)
WHERE t.ANO_LETIVO >= '2001/2002' AND t.ANO_LETIVO <= '2004/2005'
RETURN e.CATEGORIA, avg(d.HORAS), t.ANO_LETIVO
```

The result of the above query is the following:



	e.CATEGORIA	avg(d.HORAS)	t.ANO_LETIVO
1	116	2.9315068493150664	"2003/2004"
2	110	2.302083333333333	"2003/2004"
3	19999	2.0000000000000004	"2003/2004"
4	19995	1.8	"2003/2004"
5	19997	1.4444444444444442	"2003/2004"
6	107	2.1694915254237306	"2003/2004"

Figure 4.8: Query d) - Subset of Neo4jAnswer.

4.5 Query e)

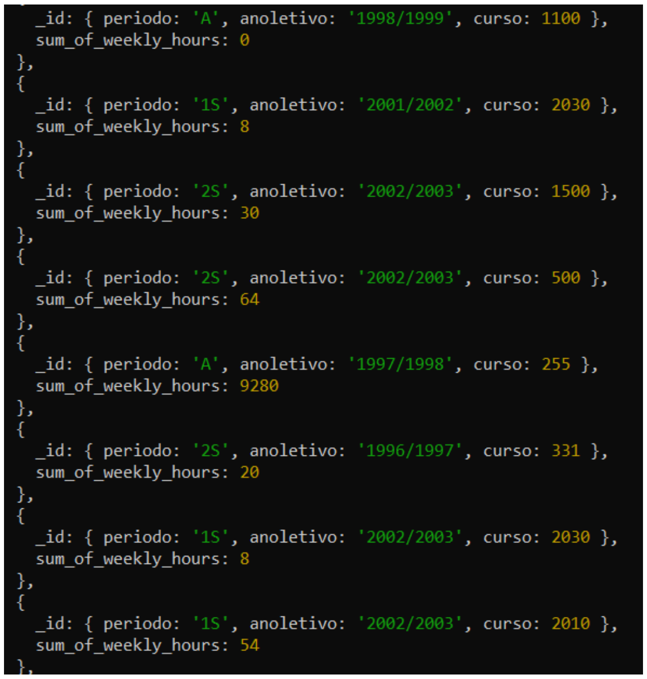
Which is the total hours per week, on each semester, that a hypothetical student enrolled in every course of a single curricular year from each program would get.

4.5.1 MongoDB

For this one, we start with a group in which there is a sum of a multiplication of the hours and the hours per turn, replacing the ones that have missing fields with a 0; there is also a group with the year and the course; finally, we use the project to return the required fields.

```
db.docs.aggregate([
    { "$group" : {"sum_of_weekly_hours": {
        $sum: { $cond: [{ $eq: ['$tipoaula.horas_turno',''] },0,
          {$cond: [{$eq: ['$tipoaula.turnos','']},0,
            {
              $multiply: [
               '$tipoaula.horas_turno',
               '$tipoaula.turnos']}]}]}}
       },"_id" : {"periodo": "$tipoaula.ocorrencia.periodo", "anoletivo":
    "$tipoaula.ocorrencia.ano_letivo", "curso": "$tipoaula.ocorrencia.uc.
   curso"}}},
    { "$project" : {"anoletivo": 1, "periodo": 1, "curso": 1, "
   sum_of_weekly_hours":1} },
])
```

The result of the above query is the following:



```
  _id: { periodo: 'A', anoletivo: '1998/1999', curso: 1100 },
  sum_of_weekly_hours: 0
},
{
  _id: { periodo: '1S', anoletivo: '2001/2002', curso: 2030 },
  sum_of_weekly_hours: 8
},
{
  _id: { periodo: '2S', anoletivo: '2002/2003', curso: 1500 },
  sum_of_weekly_hours: 30
},
{
  _id: { periodo: '2S', anoletivo: '2002/2003', curso: 500 },
  sum_of_weekly_hours: 64
},
{
  _id: { periodo: 'A', anoletivo: '1997/1998', curso: 255 },
  sum_of_weekly_hours: 9280
},
{
  _id: { periodo: '2S', anoletivo: '1996/1997', curso: 331 },
  sum_of_weekly_hours: 20
},
{
  _id: { periodo: '1S', anoletivo: '2002/2003', curso: 2030 },
  sum_of_weekly_hours: 8
},
{
  _id: { periodo: '1S', anoletivo: '2002/2003', curso: 2010 },
  sum_of_weekly_hours: 54
},
```

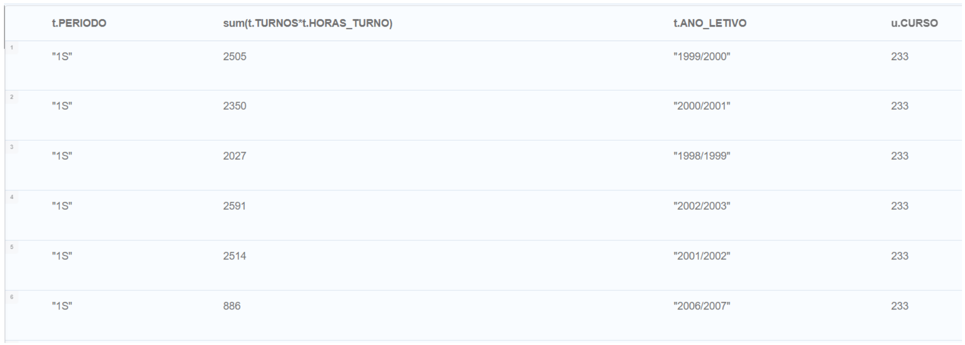
Figure 4.9: Query e) - Subset of MongoDB Answer.

4.5.2 Neo4j

On the neo4j the reasoning is the same as the MongoDB and the SQL, first there is a match to get the required information and then a return with the period, the course, the year and then the sum of the hours worked per week.

```
MATCH (e:Docentes)<-[z:Records]-(d:Dsd)-[y:Type]->(t:TiposAula)-[:Occurs]->(o:
    Ocorrencias)-[:RELTYPE]->(u:UC)
RETURN t.PERIODO, sum(t.TURNOS*t.HORAS_TURNO), t.ANO_LETIVO,u.CURSO
```

The result of the above query is the following:



	t.PERIODO	sum(t.TURNOS*t.HORAS_TURNO)	t.ANO_LETIVO	u.CURSO
1	"1S"	2505	"1999/2000"	233
2	"1S"	2350	"2000/2001"	233
3	"1S"	2027	"1998/1999"	233
4	"1S"	2591	"2002/2003"	233
5	"1S"	2514	"2001/2002"	233
6	"1S"	886	"2006/2007"	233

Figure 4.10: Query e) - Subset of Neo4jAnswer.

4.6 Query f)

The last query was up to us to define one that we thought interesting. In order to compare in the next chapter with the Oracle implementations, we've decided to use the same query we used in the previous work to illustrated the use of OR extensions. The query was/is the following:

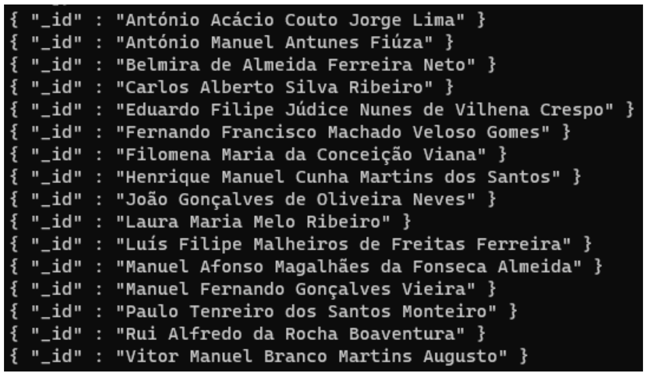
Which Professors gave lessons in a curricular unit where the number of students with frequency is no more than 10% of the number of students enrolled?

4.6.1 MongoDB

For this query, we started by getting the professor name, the number of students with frequency and 10% of the enrolled students (if the number of enrolled students is equal to "", i.e. no value defined, we considered 0) for each entry. Then, we got the professor name, an attribute called 'menor' of type boolean with the condition of the number of students with frequency being less or equal to the 10% number previously calculated and, finally, we applied a constraint to only show entries for which the attribute 'menor' was 'true', grouped by the professor name to only show distinct values and sorted by alphabetical order.

```
db.docs.aggregate([
    { "$project" : {"docente.nome" : 1, "tipoaula.ocorrencia.
   com_frequencia": 1, "percentageminscritos": {"$cond": [ { "$eq": [ "
   $tipoaula.ocorrencia.inscritos", "" ] }, 0, {"$multiply" : ["$tipoaula
   .ocorrencia.inscritos", 0.10]} ]}} },
    { "$project" : {"docente.nome" : 1, "menor": {$lte: ["$tipoaula.
   ocorrencia.com_frequencia", "$percentageminscritos"]}} },
    { "$match" : { "menor": true}},
    { "$group" : { "_id" : "$docente.nome"} },
    { "$sort" : { "_id" : 1}}
])
```

The result of the above query is the following:



```
{ "_id" : "António Acácio Couto Jorge Lima" }
{ "_id" : "António Manuel Antunes Fiúza" }
{ "_id" : "Belmira de Almeida Ferreira Neto" }
{ "_id" : "Carlos Alberto Silva Ribeiro" }
{ "_id" : "Eduardo Filipe Júdice Nunes de Vilhena Crespo" }
{ "_id" : "Fernando Francisco Machado Veloso Gomes" }
{ "_id" : "Filomena Maria da Conceição Viana" }
{ "_id" : "Henrique Manuel Cunha Martins dos Santos" }
{ "_id" : "João Gonçalves de Oliveira Neves" }
{ "_id" : "Laura Maria Melo Ribeiro" }
{ "_id" : "Luís Filipe Malheiros de Freitas Ferreira" }
{ "_id" : "Manuel Afonso Magalhães da Fonseca Almeida" }
{ "_id" : "Manuel Fernando Gonçalves Vieira" }
{ "_id" : "Paulo Tenreiro dos Santos Monteiro" }
{ "_id" : "Rui Alfredo da Rocha Boaventura" }
{ "_id" : "Vitor Manuel Branco Martins Augusto" }
```

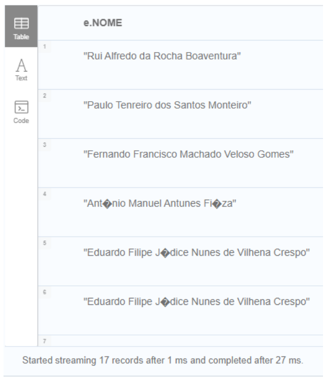
Figure 4.11: Query f) - MongoDB Answer.

4.6.2 Neo4j

This Neo4j query followed the same logic we would apply in an SQL query. After matching the nodes 'Docentes', 'Dsd', 'TiposAula' and 'Ocorrencias', we applied the constraint of the number of students with frequency being equal or less that 10% of the number of enrolled students, returning the name of the professor of those UC.

```
Match (e:Docentes)<-[z:Records]-(d:Dsd)-[y:Type]->(t:TiposAula)-[x:Occurs]->(o:
    Ocorrencias)
Where o.COM_FREQUENCIA <= 0.10*o.INSCRITOS
Return e.NOME
```

The result of the above query is the following:



	e.NOME
1	"Rui Alfredo da Rocha Boaventura"
2	"Paulo Tenreiro dos Santos Monteiro"
3	"Fernando Francisco Machado Veloso Gomes"
4	"Ant�nio Manuel Antunes Fi�za"
5	"Eduardo Filipe J�dice Nunes de Vilhena Crespo"
6	"Eduardo Filipe J�dice Nunes de Vilhena Crespo"
7	

Started streaming 17 records after 1 ms and completed after 27 ms.

Figure 4.12: Query f) - Neo4jAnswer.

Note: The picture above only shows part of the result, but the query returned 17 records, as it's written in the same picture.

5. MongoDB, Neo4j, Oracle - Comparison

5.1 Data size

In terms of data size, MongoDB was, of the three solutions, the one that occupied less space since, as explained above, the 'Dsd' served as the base table and all the entries for the tables 'Docentes' and 'TiposAula' without a match were discarded.

Then, comparing the Neo4j solution with the Oracle SQL one, in Neo4j we have 76.486 nodes and 118.555 relationships, which totalize 195.041 elements, while in SQL we have 76.486 records in the sum of all the tables.

5.2 Processing time

The average of the processing time of the queries for the Oracle SQL DB was 0.047 seconds.

The average of the processing time of the queries for the MongoDB was 0,4245 seconds.

The average of the processing time of the queries for the Neo4j was 0,1084 seconds.

5.3 Query easiness

Regarding the query easiness, the opinions were different among the group elements. For this reason, we include bellow the opinions for each group element:

- Henrique: Comparing MongoDB and Oracle, they are somewhat similar, and one can almost replace the oracle "select" with "project", the "where" with "match" and the "group by" with "group". So for these two, I found the Oracle easier to understand and read, which can also be attributed to already being used to it. For the Neo4j, everything seems more complicated; even the instructions to load the database are a more complex process than the others and, overall harder to read.

- João: In my opinion, the Neo4j queries are more similar to the SQL queries and thus easier to read when compared to MongoDB. However, even that I do not consider the MongoDB queries easier to read, I consider them easier to make comparing with Neo4j and SQL, but that opinion was formed based on the queries I did. Other queries might change my opinion. Nevertheless, I still prefer to use Oracle SQL.

- Mariana: My preference prevailed on Oracle SQL since the SQL syntax was more accessible to me because I was already familiar with the language from previous courses. Regarding Neo4j, although building the queries requires a slight change in perspective to think about the data in nodes and their connections as opposed to tables, the queries were more similar to SQL queries. For that reason, I find it easier to build the queries than MongoDB.