

NoSQL – Teaching Service

Assignment 3 Report



Integrated Masters in Informatics and Computing Engineering

Database Technologies

Grupo A:

Carlos Albuquerque - up201706735@edu.fe.up.pt

Gonçalo Marantes - up201706917@edu.fe.up.pt

Tito Griné - up201706732@edu.fe.up.pt

Faculdade de Engenharia da Universidade do Porto
Rua Roberto Frias, sn, 4200-465 Porto, Portugal

June 5, 2021

Summary

Relational databases have been around for a very long time. Some of them are still at the top of most used database management systems (DBMS). However, more recently there has been a rise in popularity of non relational databases. There are many kinds of different NoSQL databases. Some of the more common include document-oriented databases, from which MongoDB is the most popular one, and graph databases, where Neo4j stands out as one of the most used ones.

These new kinds of databases will not replace relational databases. Each has its pros, cons and specific use cases. While it is easier to represent someone's Facebook friendships as edges in a graph, it would be more practical to store big amounts of information to train a Machine Learning model in a relational database.

Therefore, in this report we explore the features and capabilities of MongoDB and Neo4j. We start by designing the model for each of these databases, then we export the data from OracleSQL's database into each of them, and, finally, we develop some queries to get a sense of how easy and performant the two DBMS are.

Overall, we enjoyed the different ways of querying data and understand why these NoSQL databases are becoming increasingly popular. We also measured the used space and the time it took each kind of DBMS to run all queries in order to compare them and to take some conclusions regarding our model design choices. At the end of the report, we present some of our conclusions and opinions regarding these databases.

Contents

1	MongoDB	5
1.1	Document Model	5
1.2	Data Migration	6
1.3	Queries	8
1.3.1	Question a)	8
1.3.2	Question b)	9
1.3.3	Question c)	10
1.3.4	Question d)	11
1.3.5	Question e)	12
1.3.6	Question f)	13
2	Neo4j	14
2.1	Graph Model	14
2.2	Data Migration	15
2.3	Queries	17
2.3.1	Question a)	17
2.3.2	Question b)	18
2.3.3	Question c)	19
2.3.4	Question d)	19
2.3.5	Question e)	20
2.3.6	Question f)	20
3	MongoDB, Neo4j and OracleSQL Comparison	21
4	Conclusion	22
	References	23

List of Figures

1	MongoDB document model diagram	5
2	Oracle XML export results	7
3	Mongoimport command used to load the data into MongoDB	8
4	Question a) MongoDB answer	9
5	Question b) MongoDB answer	10
6	Question c) MongoDB answer	11
7	Question d) MongoDB answer	12
8	Question e) MongoDB answer	13
9	Question f) MongoDB answer	14
10	Neo4j graph model	15
11	Oracle CSV export results	15
12	Question a) Neo4j answer	18
13	Question b) Neo4j answer	18
14	Question c) Neo4j answer	19
15	Question d) Neo4j answer	20
16	Question e) Neo4j answer	20
17	Question f) Neo4j answer	21

Listings

1	SQL JSON extraction script	6
2	Example JSON object extracted from the XML file	7
3	Question a) MongoDB Script	8
4	Question b) MongoDB Script	9
5	Question c) MongoDB Script	10
6	Question d) MongoDB Script	11
7	Question e) MongoDB Script	12

8	Question f) MongoDB Script	13
9	SQL CSV extraction script	15
10	UC node creation from exported CSV file	16
11	Ocorrencias node and relations creation from exported CSV file	16
12	Tiposaula node and relations creation from exported CSV file	16
13	Docentes node and relations creation from exported CSV file	17
14	Dsd relation creation from exported CSV file	17
15	Question a) Neo4j Cypher Script	17
16	Question b) Neo4j Cypher Script	18
17	Question c) Neo4j Cypher Script	19
18	Question d) Neo4j Cypher Script	19
19	Question e) Neo4j Cypher Script	20
20	Question f) Neo4j Cypher Script	21

List of Tables

1	Different metrics comparison between the three database models	21
---	--	----

1 MongoDB

1.1 Document Model

To come up with a suitable MongoDB model, that would make good use of its capabilities, we started by analyzing the existing tables and their relations. We noticed that most of the data is related through the *tiposaula* table, which lead us to use it as the starting point of our information model.

Starting from there it becomes really easy to adapt the data. Every *tiposaula* has an array of *dsd*, which we called *docentes*. Every entry of said array contains the information of both the *dsd* and *docente* tables. Every *tiposaula* is also related to one *ocorrencia*, so there is a nested object to contain all the occurrence information. This includes a field with the course's information inside yet another object, called *uc*, just like the SQL table.

The resulting model structure is as follows:

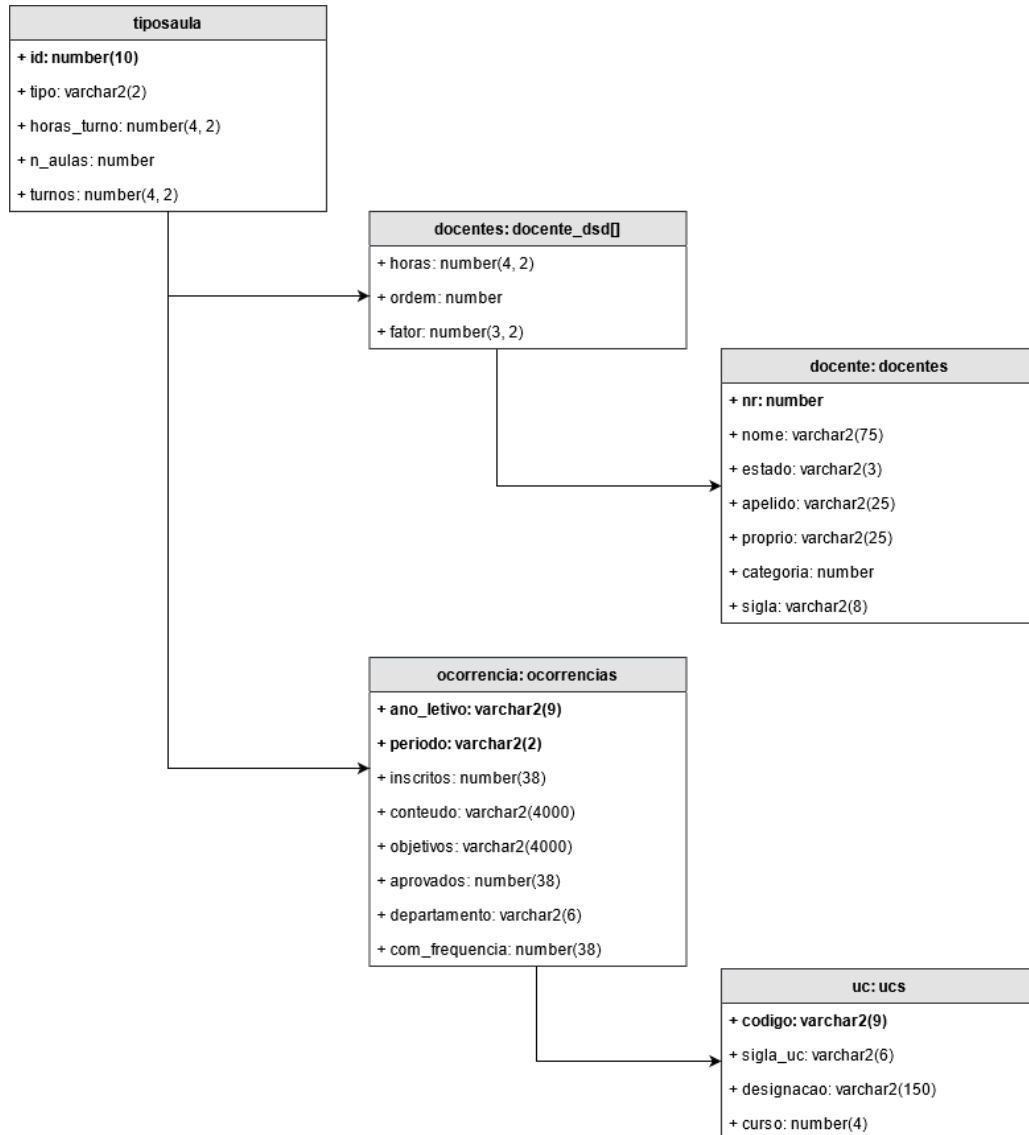


Figure 1: MongoDB document model diagram

We considered other ways to organize the information, but we deemed the use of a single collection the best option for our case. That way, we do not need to perform any join to get information, which would later reveal to make the queries very efficient. We could have also started our model from other tables. For example, if we started with the *uc* table instead of *tiposaula*, we would be reducing data redundancy as each course is being replicated for each year, occurrence and type of class. Nevertheless, that would make the model harder to query since it would have a greater depth, i.e. every object would be nested inside the object above like a Matryoshka doll.

1.2 Data Migration

To migrate the data from the SQL tables into MongoDB, we wrote the query shown in listing 1. We made use of the Oracle's *json_object()* method to retrieve the values in JSON format, which can directly be interpreted by MongoDB.

```
1 select
2     json_object(
3         '_id' value tp.id,
4         'tipo' value tp.tipo,
5         'horas_turno' value tp.horas_turno,
6         'n_aulas' value tp.n_aulas,
7         'turnos' value tp.turnos,
8         'docentes' value (select
9             json_arrayagg(
10                 json_object(
11                     'horas' value dsd.horas,
12                     'ordem' value dsd.ordem,
13                     'fator' value dsd.fator,
14                     'docente' value (select
15                         json_object(
16                             'nr' value d.nr,
17                             'nome' value d.nome,
18                             'estado' value d.estado,
19                             'apelido' value d.apelido,
20                             'proprio' value d.proprio,
21                             'categoria' value d.categoria,
22                             'sigla' value d.sigla
23                         )
24                     from xdocentes d
25                     where d.nr = dsd.nr) returning clob
26                 ) returning clob
27             )
28         from xdsd dsd
29         where dsd.id = tp.id),
30         'ocorrencia' value (select
31             json_object(
32                 'ano_letivo' value o.ano_letivo,
33                 'periodo' value o.periodo,
34                 'inscritos' value o.inscritos,
35                 'conteudo' value o.conteudo,
36                 'objetivos' value o.objetivos,
37                 'aprovados' value o.aprovados,
38                 'departamento' value o.departamento,
39                 'com_frequencia' value o.com_frequencia,
40                 'uc' value (select
41                     json_object(
42                         'codigo' value u.codigo,
43                         'sigla_uc' value u.sigla_uc,
44                         'designacao' value u.designacao,
45                         'curso' value u.curso
46                     )
47                     from xucs u
48                     where u.codigo = o.codigo) returning clob
49                 )
50             from xocorrencias o
51             where o.ano_letivo = tp.ano_letivo
52                 and o.codigo = tp.codigo
53                 and o.periodo = tp.periodo) returning clob
54         ) as all_data
55 from xtuposaula tp;
```

Listing 1: SQL JSON extraction script

The above query already nests and names all the fields according to our model, so the result is a single column and each row represents one document to be inserted in Mongo. Afterwards, we exported the data to a XML file as shown in figure 2. We chose XML instead of JSON because the latter was producing weird results. Sometimes it would export an empty array, other times it did not export every row in the results.

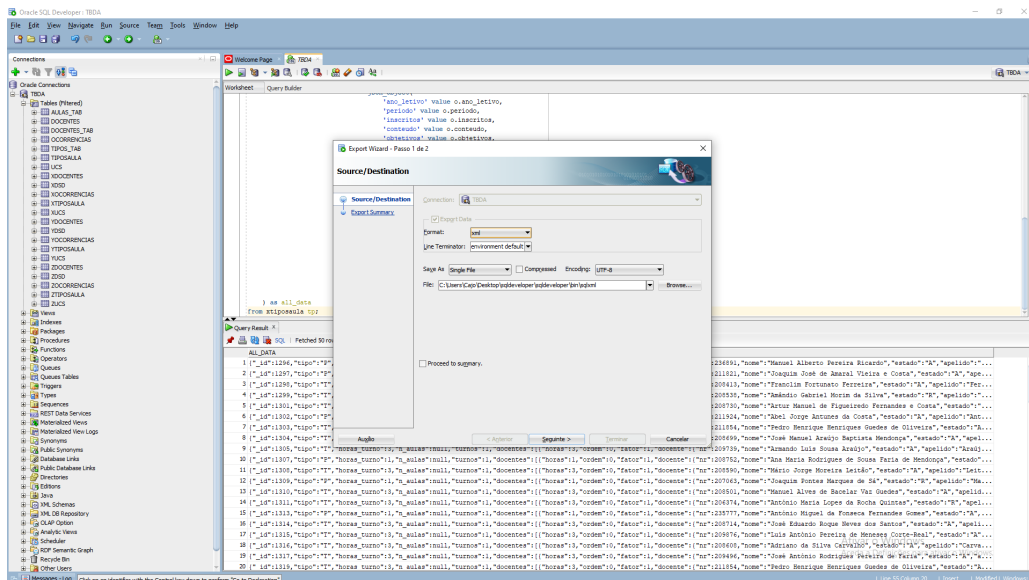


Figure 2: Oracle XML export results

After exporting to XML, we had to trim the file to extract the JSON objects inside each tag. The result is an array of objects similar to the one depicted in listing 2. With this steps we successfully exported the data into a JSON file. There might be other alternatives that require less intervention from the programmer, but we found this approach simple enough to follow.

```
[
  {
    "_id": 1296,
    "tipo": "P",
    "horas_turno": 2,
    "n_aulas": null,
    "turnos": 5,
    "docentes": [
      {
        "nr": 236891,
        "nome": "Manuel Alberto Pereira Ricardo",
        "estado": "A",
        "apellido": "Pereira Ricardo",
        "proprio": "Manuel",
        "categoria": 110,
        "sigla": "MPR"
      },
      {
        "nr": 901030,
        "nome": "Carlos Alberto Rodrigues e Silva",
        "estado": "NA",
        "apellido": "Silva",
        "proprio": "Carlos",
        "categoria": 11007,
        "sigla": "CRS"
      }
    ]
  },
  {
    "ocorrencia": {
      "ano_letivo": "1998/1999",
      "periodo": "2S",
      "inscritos": 92,
      "conteudo": "Introdução à Comunicação de Dados - técnicas de comunicação, modos de operação, serviços. Modelo OSI. Data Link - LAPB, LAPD, LAPF (Frame Relay). LANs / interligação de LANs. WANs (X.25). Internet (arquitetura TCP / IP).",
    }
  }
]
```

```

43     "objetivos": null,
44     "aprovados": 69,
45     "departamento": "DEEC",
46     "com_frequencia": 70,
47     "uc": {
48         "codigo": "EEC4261",
49         "sigla_uc": "CDRC1",
50         "designacao": "Comunica  o de Dados e Redes de Computadores I",
51         "curso": 255
52     }
53 }
54 }
55 ]

```

Listing 2: Example JSON object extracted from the XML file

Finally, to load the JSON data file into our MongoDB instance, we used the *mongoimport* tool provided by Mongo's creators. This executable is very easy to use and, with just one command, we were able to populate the whole database.

The command connects into the database endpoint with the provided credentials, reads the JSON file from the provided path and, finally, creates every document in the specified collection. The command used is shown below, in figure 3. After executing it, the data migration process is concluded.

```

$ mongoimport --host vdbase.inesctec.pt:27017 --db tbda --collection tiposaula --authenticationDatabase admin
--username tbda --password grupoa --file <path-to-json-file> --jsonArray

```

Figure 3: Mongoimport command used to load the data into MongoDB

1.3 Queries

1.3.1 Question a)

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

Using the aggregate function, the query just needs to first match on the desired *program* and *ano_letivo* and then group the results by *tipo*, summing for each group the *horas_turno* multiplied by the number of *turnos*. The *course* and *year* are selected just to confirm that the original match worked. Finally the projection is just to return the results with more appropriate names.

```

1 db.tiposaula.aggregate(
2   [
3     {
4       $match:{
5         "ocorrencia.uc.curso": 233,
6         "ocorrencia.ano_letivo": "2004/2005"
7       }
8     },
9     {
10      $group: {
11        _id: "$tipo",
12        "class_hours": { $sum: { $multiply: ["$horas_turno", "$turnos"] }},
13        "course": { $first: "$ocorrencia.uc.curso" },
14        "year": { $first: "$ocorrencia.ano_letivo" }
15      }
16    },
17    {
18      $project: {
19        _id: 0,
20        "type": "$_id",
21        "class_hours": 1,
22        "course": 1,
23        "year": 1
24      }
25    }
26  ]
27 );

```

Listing 3: Question a) MongoDB Script

	class_hours	course	year	type
1	697.5	233	2004/2005	TP
2	581.5	233	2004/2005	P
3	308	233	2004/2005	T

Figure 4: Question a) MongoDB answer

1.3.2 Question 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?”

For this query, the aggregation pipeline starts by doing a match on the desired `ano_letivo`, after which, in order to get the hours assigned to teachers of each **tiposaula** we must add a new field to each document that holds the result of reducing the array of `docentes` by adding each `horas` assigned. After adding this field, the pipeline groups the results by `codigo`, summing all the `hours_assigned` and, like the previous query, sum the product of `horas_turno` by the number of `turnos` to get the total hours required.

The following projection adds a new boolean field that is true if the `total_hours_assigned` are different from the `total_hours_required`. This field is used by the match that follows, to filter the results that, as the question requests, have a difference between the total hours required and the total hours assigned. The `$ne` operator couldn't be used directly in the match, due to MongoDB's restrictions on this pipeline stage.

Lastly, a projection is done to keep only the desired fields and an ordering on `code` is performed.

```

1 db.tiposaula.aggregate(
2   [
3     {
4       $match: {
5         "ocorrencia.ano_letivo": "2003/2004"
6       }
7     },
8     {
9       $addFields: {
10        "hours_assigned": {
11          $reduce: {
12            input: "$docentes",
13            initialValue: 0,
14            in: { $add: ["$$value", "$$this.horas"]}
15          }
16        }
17      },
18    },
19    {
20      $group: {
21        _id: "$ocorrencia.uc.codigo",
22        "total_hours_required": { $sum: { $multiply: ["$horas_turno", "$turnos"] } },
23        "total_hours_assigned": { $sum: "$hours_assigned" }
24      }
25    },
26    {
27      $project: {
28        code: "$_id",
29        hours_required: "$total_hours_required",
30        hours_assigned: "$total_hours_assigned",
31        diff: { $ne: ["$total_hours_assigned", "$total_hours_required"] }
32      }
33    },
34    {
35      $match: { "diff": true }
36    },
37    {
38      $project: {
39        "_id": 0,
40        "code": 1,
41        "hours_required": 1,
42        "hours_assigned": 1,
43      }
44    }
45  ]
46 )

```

```

44     },
45     {
46         $sort: { "code": 1 }
47     }
48 ]
49 );

```

Listing 4: Question b) MongoDB Script

tiposaula 0.065 sec.			
	codigo	hours_required	hours_assigned
1	CI002	4	0
2	CI003	4	0
3	CI004	4	0
4	CI007	4	0
5	CI008	4	0
6	CI009	4	0

Figure 5: Question b) MongoDB answer

1.3.3 Question 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.”

The approach to this question is a bit unorthodox due to some constraints on the way aggregation works in MongoDB.

The first few stages are straightforward, it starts by matching the results that have the desired `ano_letivo`, followed by the `unwind` operator, that separates every entry in the `docentes` array to its own document. This means that each `docente` is now a separate result, keeping the information from the original document. Next, the results are grouped by both the `docente`’s number and the `tipo`, keeping the `nome` and summing all the `horas` associated with that `docente` for that `tipo`. At this point, the natural next step would be to group the results by only `type` and have the `max` aggregation operator on `hours` to get the answer. This would work to an extent, because, although the results for each `type` would be correct, by having to group by `type` only, we lose the association to the `docente`. As far as we were able to gather, there isn’t a way to get the maximum of a field in a grouping stage and retrieve another field associated with that entry that isn’t a part of the grouping key.

As such, to work around this problem, the pipeline instead orders the results by `hours`, in descending order, which means that now the first result for each `type` will be the professor with the most class hours. This means that we can then group the results by `type` and keep the first entry of each result to get the desired answer. The final projection is just to return the results with better field names.

```

1 db.tiposaula.aggregate(
2   [
3     {
4       $match: { "ocorrencia.ano_letivo": "2003/2004" }
5     },
6     {
7       $unwind: "$docentes"
8     },
9     {
10      $group: {
11        _id: {
12          "docente": "$docentes.docente.nr",
13          "type": "$tipo"
14        },
15        "nome": { $first: "$docentes.docente.nome" },
16        "hours": { $sum: "$docentes.horas" }
17      }
18    },
19    {
20      $sort: { "hours": -1 }
21    }
22  ]
23 );

```

```

20 {
21   $group: {
22     _id: "$_id.type",
23     "nr": { $first: "$_id.docente" },
24     "nome": { $first: "$nome" },
25     "hours": { $first: "$hours" }
26   }
27 },
28 {
29   $project: {
30     "_id": 0,
31     "type": "$_id",
32     "nr": 1,
33     "nome": 1,
34     "hours": 1
35   }
36 }
37 ]
38 );

```

Listing 5: Question c) MongoDB Script

tiposaula 0.062 sec.			
	nr	nome	hours
1	210006	João Carlos Pascoal de Faria	3.5
2	249564	Cecília do Carmo Ferreira da Silva	26
3	208187	António Almerindo Pinheiro Vieira	30
4	207638	Fernando Francisco Machado Veloso Gomes	30.67

Figure 6: Question c) MongoDB answer

1.3.4 Question d)

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

For this query, the pipeline begins by matching only the documents that have the `ano_letivo` in the desired years by using the `$in` operator. This is followed by the unwind stage that, as explained in the previous question, creates a document for each entry in the `docentes` array.

The following match is used to remove results that have `categoria` as `NULL` since we believe this shouldn't be considered as a separate category.

Next in the pipeline, it groups the results by `docente's` number, category and `ano_letivo`. For each group we calculate the average of the `horas`. It's important to note that we get the first `nome` of each group because they will all be the same since it's also grouped by `docente.nr`.

Finally, a projection is used to have better field names for the results with them also being sorted by `name` and `year`.

```

1 db.tiposaula.aggregate(
2   [
3     {
4       $match: {
5         "ocorrencia.ano_letivo": {
6           $in: ["2001/2002", "2002/2003", "2003/2004", "2004/2005"]
7         }
8       }
9     },
10    { $unwind: "$docentes" },
11    {
12      $match: { "docentes.docente.categoria": { $exists: true, $ne: null } }
13    },
14    {
15      $group: {
16        _id: {
17          "docente": "$docentes.docente.nr",
18          "category": "$docentes.docente.categoria",

```

```

19         "year": "$ocorrencia.ano_letivo"
20     },
21     "name": { $first: "$docentes.docente.nome" },
22     "avg_hours": { $avg: "$docentes.horas" }
23 }
24 },
25 {
26     $project: {
27         _id: 0,
28         "category": "$_id.category",
29         "year": "$_id.year",
30         "name": 1,
31         "avg_hours": 1
32     }
33 },
34 {
35     $sort: { "name": 1, "year": 1 }
36 }
37 ]
38 );

```

Listing 6: Question d) MongoDB Script

tiposaula 0.097 sec.				
	name	avg_hours	category	year
1	Abel Dias dos Santos	3.16666666666667	116	2002/2003
2	Abel Dias dos Santos	2.40909090909091	116	2003/2004
3	Abel Dias dos Santos	3.0	116	2004/2005
4	Abel Jorge Antunes da Costa	2.88833333333333	116	2001/2002
5	Abel Jorge Antunes da Costa	3.5	116	2002/2003
6	Abel Jorge Antunes da Costa	1.83333333333333	116	2003/2004

Figure 7: Question d) MongoDB answer

1.3.5 Question e)

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

In relation to the previous queries, this one doesn't introduce anything new. It starts by matching `ano_letivo` to "2009/2010", although any other year would also work, `periodo` to be either "1S" or "2S" given that we want the total hours per work "on each semester" and finally, to avoid courses that are NULL, it removes these from the results.

After this stage, it groups by `curso` and `periodo` calculating the total hours per week by summing the `horas_turno` of each group.

Much like the previous queries, the pipeline ends by making a projection to make the results more readable and, in this case, also sorts them by `course` and `semester`.

```

1 db.tiposaula.aggregate(
2   [
3     {
4       $match: {
5         "ocorrencia.ano_letivo": "2009/2010",
6         "ocorrencia.periodo": { $in: ['1S', '2S'] },
7         "ocorrencia.uc.curso": {
8           $exists: true,
9           $ne: null
10        }
11      }
12    },
13    { $group: {
14      _id: {
15        "course": "$ocorrencia.uc.curso",
16        "semester": "$ocorrencia.periodo"
17      }

```

```

18     "total_hours": { $sum: "$horas_turno" }
19   },
20 },
21 {
22   $project: {
23     _id: 0,
24     "course": "$_id.course",
25     "semester": "$_id.semester",
26     "total_hours": 1
27   }
28 },
29 {
30   $sort: { "course": 1, "semester": 1 }
31 }
32 ]
33 );

```

Listing 7: Question e) MongoDB Script

tiposaula 0.043 sec.			
	total_hours	course	semester
1	34	2001	1S
2	31	2001	2S
3	21	2004	1S
4	36	2004	2S
5	28.0	2006	1S
6	26.0	2006	2S

Figure 8: Question e) MongoDB answer

1.3.6 Question f)

“Ask the database a query you think is interesting.”

To explore more of MongoDB’s functionalities, we aimed to create a query that would use operators that hadn’t been used previously. With this in mind, the query that was chosen finds all the curricular units that Professor Gabriel David has been assigned to and the respective frequency, with each curricular unit being distinguished by its code.

Analyzing the query, the pipeline first has the unwind stage to separate each `docente` in its own document, after which it matches the name to be that of Professor Gabriel David.

Following this it uses the `sortByCount` stage that will count the number of entries for each `codigo` and sort the results in descending order. However, due to the way this stage operator works, the results only have the `codigo`, because it was the id used, and the respective count. In order to get the `sigla` and `designacao` of the respective curricular units, the next stage in the pipeline performs a lookup on all occurrences to find the documents that have the `ocorrencia.uc.codigo` as the result ids.

Lastly, because now for each result there is an array of documents associated, the pipeline must do a projection to only get the fields that are necessary, namely `sigla` and `designacao`. Although they are the same for each document, because there are multiple, we must use the `$arrayElemAt` operator to retrieve only the first entry.

```

1 db.tiposaula.aggregate(
2   [
3     { $unwind: "$docentes" },
4     { $match: { "docentes.docente.nome": "Gabriel de Sousa Torcato David" } },
5     { $sortByCount: "$ocorrencia.uc.codigo" },
6     {
7       $lookup: {
8         from: "tiposaula",
9         localField: "_id",

```

```

10         foreignField: "ocorrencia.uc.codigo",
11         as: "tipoaula"
12     },
13 },
14 { $project: {
15     "_id": 0,
16     "code": "$_id",
17     "sigla": { $arrayElemAt: ["$tipoaula.ocorrencia.uc.sigla_uc", 0] },
18     "designacion": { $arrayElemAt: ["$tipoaula.ocorrencia.uc.designacao", 0] },
19     "count": 1
20 }
21 },
22 ]
23 );

```

Listing 8: Question f) MongoDB Script

tiposaula 0.749 sec.				
	count	code	sigla	designacion
1	8	CI036	BASDA	Bases de Dados
2	6	MGI1211	BD	Bases de Dados
3	6	EI1106	AD	Armazéns de Dados
4	6	EI1105	TBD	Tecnologias de Bases de Dados
5	5	EEC431	BD	Bases de Dados
6	5	EIC3206	CLF	Computabilidade e Linguagens Formais

Figure 9: Question f) MongoDB answer

2 Neo4j

2.1 Graph Model

Designing a graph model from the original schema was relatively straightforward since most tables made sense to be represented as nodes. The only exception was the `asa` table, that in the original schema worked as an intermediate table between `docentes` and `tiposaula`. It made more sense to represent it as an edge with the information instead of having it as a node as well.

The resulting model structure is as follows:

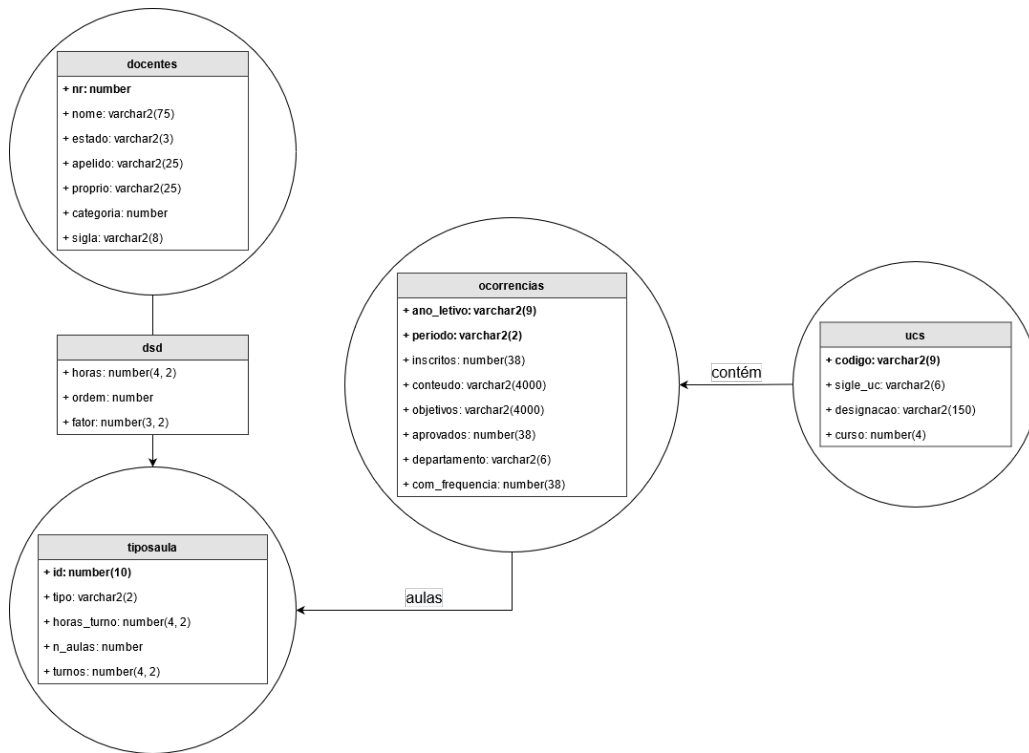


Figure 10: Neo4j graph model

2.2 Data Migration

In order to migrate the data from the Oracle SQL Server, we first needed to extract the tables to *CSV* files. This can be easily done by selecting everything from the 5 tables and then exporting the result.

```

1 SELECT * FROM XDOCENTES;
2 SELECT * FROM XDSD;
3 SELECT * FROM XTIPOSAULA;
4 SELECT * FROM XOCORRENCIAS;
5 SELECT * FROM XUCS;

```

Listing 9: SQL CSV extraction script

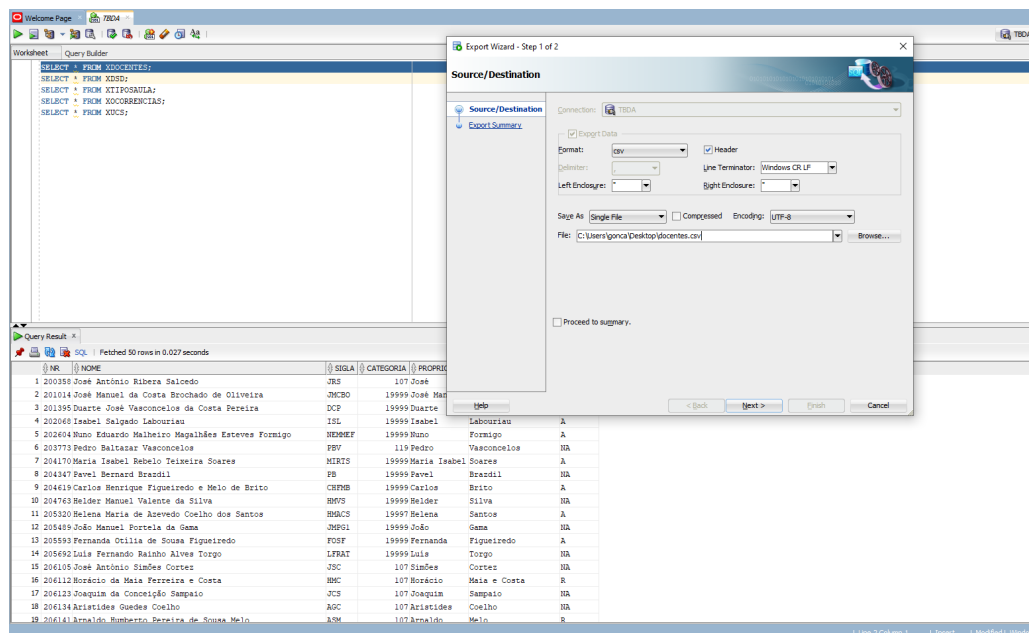


Figure 11: Oracle CSV export results

After running the queries in listing 9 and exporting the results such as described in figure 11, we should now

have 5 *CSV* files.

After creating a new Neo4j project, it is necessary to move the 5 *CSV* files to the project's import folder. Depending on the Neo4j version, it can usually be found in `path/to/.Neo4jDesktop/relate-data/dbmss/dbms-[new-project-hash]/import/`. After the fact, the scripts below can be executed in the Neo4j browser, in order to create the graph database as well as populate it.

A few simple indexes were also created in order to speed up the database population process.

```
1 // create ucs
2 LOAD CSV WITH HEADERS FROM 'file:///ucs.csv' AS line
3 CREATE (:uc
4   {
5     codigo: line.CODIGO,
6     designacao: line.DESIGNACAO,
7     sigla_uc: line.SIGLA_UC,
8     curso: toInteger(line.CURSO)
9   }
10 );
11 // create UC index for faster joins
12 CREATE INDEX uc_codigo_idx IF NOT EXISTS
13 FOR (u:uc)
14 ON (u.codigo);
```

Listing 10: UC node creation from exported CSV file

```
1 // create ocorrencias
2 LOAD CSV WITH HEADERS FROM 'file:///ocorrencias.csv' AS line
3 MATCH (u:uc // match foreign key
4   {
5     codigo: line.CODIGO
6   }
7 )
8 CREATE (o:ocorrencia
9   {
10    codigo: line.CODIGO,
11    ano_letivo: line.ANO_LETIVO,
12    periodo: line.PERIODO,
13    inscritos: toInteger(line.INSCRITOS),
14    com_frequencia: toInteger(line.COM_FREQUENCIA),
15    aprovados: toInteger(line.APROVADOS),
16    objetivos: line.OBJETIVOS,
17    conteudo: line.CONTEUDO,
18    departamento: line.DEPARTAMENTO
19  }
20 )
21 CREATE (u)-[:contem]->(o);
22 // create ocorrencia index for faster joins
23 CREATE INDEX ocorrencia_composite_idx IF NOT EXISTS
24 FOR (o:ocorrencia)
25 ON (o.codigo, o.ano_letivo, o.periodo);
```

Listing 11: Ocorrencias node and relations creation from exported CSV file

```
1 // create tiposaula
2 LOAD CSV WITH HEADERS FROM 'file:///tiposaula.csv' AS line
3 MATCH (o:ocorrencia // match foreign keys
4   {
5     codigo: line.CODIGO,
6     ano_letivo: line.ANO_LETIVO,
7     periodo: line.PERIODO
8   }
9 )
10 CREATE (t:tiposaula
11   {
12     id: toInteger(line.ID),
13     tipo: line.TIPO,
14     ano_letivo: line.ANO_LETIVO,
15     periodo: line.PERIODO,
16     codigo: line.CODIGO,
17     turnos: toFloat(line.TURNOS),
18     n_aulas: toInteger(line.N_AULAS),
19     horas_turno: toFloat(line.HORAS_TURNOS)
20   }
21 )
22 CREATE (o)-[:aulas]->(t);
23 // create tiposaula index for faster joins
```



```

24 CREATE INDEX tipoaula_id_idx IF NOT EXISTS
25 FOR (t:tipoaula)
26 ON (t.id);

```

Listing 12: Tiposaula node and relations creation from exported CSV file

```

1 // create docentes
2 LOAD CSV WITH HEADERS FROM 'file:///docentes.csv' AS line
3 CREATE (:docente
4 {
5     nr: toInteger(line.NR),
6     nome: line.NOME,
7     sigla: line.SIGLA,
8     categoria: line.CATEGORIA,
9     proprio: line.PROPRIO,
10    apelido: line.APELIDO,
11    estado: line.ESTADO
12 }
13 );
14 // create tipoaula index for faster joins
15 CREATE INDEX docente_nr_idx IF NOT EXISTS
16 FOR (d:docente)
17 ON (d.nr);

```

Listing 13: Docentes node and relations creation from exported CSV file

```

1 // create dsd
2 LOAD CSV WITH HEADERS FROM 'file:///dsd.csv' AS line
3 MATCH (d:docente // match foreign keys
4 {
5     nr: toInteger(line.NR)
6 }
7 )
8 MATCH (t:tipoaula
9 {
10    id: toInteger(line.ID)
11 }
12 )
13 CREATE (d)-[:dsd
14 {
15     horas: toFloat(line.HORAS),
16     fator: toFloat(line.FATOR),
17     ordem: toInteger(line.ORDEM)
18 }
19 ]->(t);

```

Listing 14: Dsd relation creation from exported CSV file

2.3 Queries

2.3.1 Question a)

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

This query is rather simple, all we need to do is match the graph pattern starting from the *uc* node with the program (*curso*) 233, in which the *ocorrencia* node has the *ano_letivo* attribute equal to 2004/2005. After that, end the pattern in the *tipoaula* node.

Finally, we simply need to aggregate the sum of class hours by each type of class.

```

1 MATCH (u:uc {curso: 233})-[:contem]->(o:ocorrencia {ano_letivo: '2004/2005'})-[:aulas]->(t:
    tipoaula)
2 WITH u.curso as curso, o.ano_letivo as ano_letivo, t.tipo as tipo, sum(t.horas_turno * t.
    turnos) as sumHoras
3 RETURN curso, ano_letivo, tipo, sumHoras;

```

Listing 15: Question a) Neo4j Cypher Script

	curso	ano_letivo	tipo	sumHoras
1	233	"2004/2005"	"TP"	697.5
2	233	"2004/2005"	"T"	308.0
3	233	"2004/2005"	"P"	581.5

Started streaming 3 records after 1 ms and completed after 97 ms.

Figure 12: Question a) Neo4j answer

2.3.2 Question 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?”

This query demands a bit more thought. The problem here is that, if we match everything in the same *Match* block, the hours for each *tipoaula* will be replicated depending on the number of professors (*docente*) assigned to it through the *dsd* edge. This will make all the hour calculations wrong.

Therefore, we start by calculating the total amount of hours for each *tipoaula*. This is achieved in the first *Match* block, from where we save the total amount of hours required in an alias (*total_required*).

Afterwards, we need to sum the total amount of service assigned to each *tipoaula*. This could be achieved by simply adding the *dsd* edge and the *docente* node. However, some of the *tipoaula* nodes have no *dsd* edges, which means they would not appear in the results, even though, conceptually, they have 0 service assigned.

For this reason, we make use of an *Optional Match* to match the *tipoaula* node with the *dsd* edges. This operation will always return the *tipoaula*, either with an actual *dsd* or with a *null* edge. This way, when we sum the hours of each *dsd* grouped by the professor, the result will be 0 if the edge is *null*. The *round* method is being used to cut off rounding errors.

Finally, with the results of both matches, we can match the *ucs* by their *codigo* field and filter the ones where the total hours required do not match the assigned service hours.

```

1 MATCH (u1:uc)-[:contem]->(:ocorrencia {ano_letivo: '2003/2004'})-[:aulas]->(t:tipoaula)
2 WITH u1.codigo as codigo1, sum(t.horas_turno * t.turnos) as total_required
3 MATCH (u2:uc)-[:contem]->(:ocorrencia {ano_letivo: '2003/2004'})-[:aulas]->(t:tipoaula)
4 OPTIONAL MATCH (t)-[:dsd]->(d:docente)
5 WITH u2.codigo as codigo2, codigo1, round(sum(d.horas), 10) as service_assigned,
   total_required
6 WHERE codigo2 = codigo1 AND total_required <> service_assigned
7 RETURN codigo2, total_required, service_assigned
8 ORDER BY codigo2;

```

Listing 16: Question b) Neo4j Cypher Script

	codigo2	total_required	service_assigned
1	"CI002"	4.0	0.0
2	"CI003"	4.0	0.0
3	"CI004"	4.0	0.0
4	"CI007"	4.0	0.0
5	"CI008"	4.0	0.0
6	"CI009"	4.0	0.0
7	"CI010"	4.0	0.0

Started streaming 88 records after 1 ms and completed after 10395 ms.

Figure 13: Question b) Neo4j answer

2.3.3 Question c)

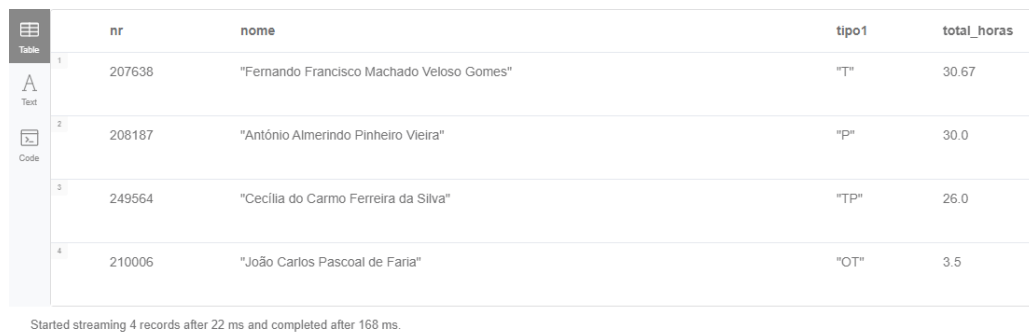
“Who is the professor with more class hours for each type of class, in the academic year 2003/2004? Show the number and name of the professor, the type of class and the total of class hours times the factor.”

This query was developed in two parts. The first one, depicted in the first *Match* block, is responsible for finding the maximum total hours, by type, a professor has been assigned to in the academic year 2003/2004. That way, although we do not yet know which professor corresponds to each type, we already know the max amount of hours for each type of class (*max_total*).

The second part, which is computed in the second *Match* block, will get the total hours by professor by each type. Then, it is just a matter of matching the types (*tipoaula* nodes) and the hours to find out the names and numbers of the *docentes* that have the most class hours for each type of class.

```
1 MATCH (:ocorrencia {ano_letivo: '2003/2004'})-[:aulas]->(t1:tipoaula)-[ds1:dsd]-(d1:docente)
2 WITH d1.nr as nr, d1.nome as nome, t1.tipo as tipo1, sum(ds1.horas) as temp_total
3 WITH tipo1, max(temp_total) as max_total
4 MATCH (:ocorrencia {ano_letivo: '2003/2004'})-[:aulas]->(t2:tipoaula)-[ds2:dsd]-(d2:docente)
5 WITH d2.nr as nr, d2.nome as nome, tipo1, t2.tipo as tipo2, sum(ds2.horas) as total_horas,
6 max_total
7 WHERE tipo1 = tipo2 AND total_horas = max_total
8 RETURN nr, nome, tipo1, total_horas;
```

Listing 17: Question c) Neo4j Cypher Script



	nr	nome	tipo1	total_horas
1	207638	"Fernando Francisco Machado Veloso Gomes"	"T"	30.67
2	208187	"Antônio Almerindo Pinheiro Vieira"	"P"	30.0
3	249564	"Cecília do Carmo Ferreira da Silva"	"TP"	26.0
4	210006	"João Carlos Pascoal de Faria"	"OT"	3.5

Started streaming 4 records after 22 ms and completed after 168 ms.

Figure 14: Question c) Neo4j answer

2.3.4 Question d)

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

For this query, we only need to filter the occurrences (*ocorrencia*) by the according years, remove the professors without a category and then calculate the average hours those professors have assigned to a specific class.

```
1 MATCH (o:ocorrencia)-[:aulas]->(t:tipoaula)-[ds:dsd]-(d:docente)
2 WHERE o.ano_letivo IN ['2001/2002', '2002/2003', '2003/2004', '2004/2005'] AND d.categoria IS
3 NOT NULL
4 WITH d.categoria as categoria, d.nr as nr, d.nome as nome, o.ano_letivo as ano_letivo, avg(ds.
5 horas) as media_horas
6 ORDER BY nome, ano_letivo
7 RETURN categoria, nome, ano_letivo, media_horas;
```

Listing 18: Question d) Neo4j Cypher Script

	categoria	nome	ano_letivo	media_horas
1	"116"	"Abel Dias dos Santos"	"2002/2003"	3.1666666666666665
2	"116"	"Abel Dias dos Santos"	"2003/2004"	2.409090909090909
3	"116"	"Abel Dias dos Santos"	"2004/2005"	3.0
4	"116"	"Abel Jorge Antunes da Costa"	"2001/2002"	2.888333333333333
5	"116"	"Abel Jorge Antunes da Costa"	"2002/2003"	3.5
6	"116"	"Abel Jorge Antunes da Costa"	"2003/2004"	1.8333333333333333
7

Started streaming 1897 records after 15 ms and completed after 23 ms, displaying first 1000 rows.

Figure 15: Question d) Neo4j answer

2.3.5 Question e)

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

In order to answer this question, we fetch all not *null* programs and aggregate the sum of weekly hours (*horas_turno*) by the program and semester. This is only applied to the *ocorrencias* in the academic year 2009/2010, which we chose to use for being the last registered year.

```

1 MATCH (u:uc)-[:contem]->(o:ocorrencia {ano_letivo: '2009/2010'})-[:aulas]->(t:tipoaula)
2 WHERE o.periodo in ['1S', '2S'] AND u.curso IS NOT NULL
3 WITH u.curso as curso, o.periodo as periodo, sum(t.horas_turno) as horas_semanais
4 ORDER BY curso, periodo
5 RETURN curso, periodo, horas_semanais;

```

Listing 19: Question e) Neo4j Cypher Script

	curso	periodo	horas_semanais
1	2001	"1S"	34.0
2	2001	"2S"	31.0
3	2004	"1S"	21.0
4	2004	"2S"	36.0
5	2006	"1S"	28.0
6	2006	"2S"	26.0
7

Started streaming 20 records after 15 ms and completed after 61 ms.

Figure 16: Question e) Neo4j answer

2.3.6 Question f)

“Ask the database a query you think is interesting.”

Since querying in Neo4j revolves around graph patterns, we wanted to try to find all the paths between two nodes. So we thought about finding all the relations between Professor Gabriel David and the course Tecnologias de Bases de Dados.

This is rather easy to do in graphql, we only need to find the two nodes we want to connect, the professor and the course, and then return the resulting pattern where anything can be in between. We limit the amount of edges it can traverse to 5, otherwise the query would take forever to return a result.

```

1 MATCH
2   (prof:docente {nome: 'Gabriel de Sousa Torcato David'}),
3   (tbda:uc {designacao: 'Tecnologias de Bases de Dados'}),
4   path = (prof)-[*..5]-(tbda)
5 RETURN path;

```

Listing 20: Question f) Neo4j Cypher Script

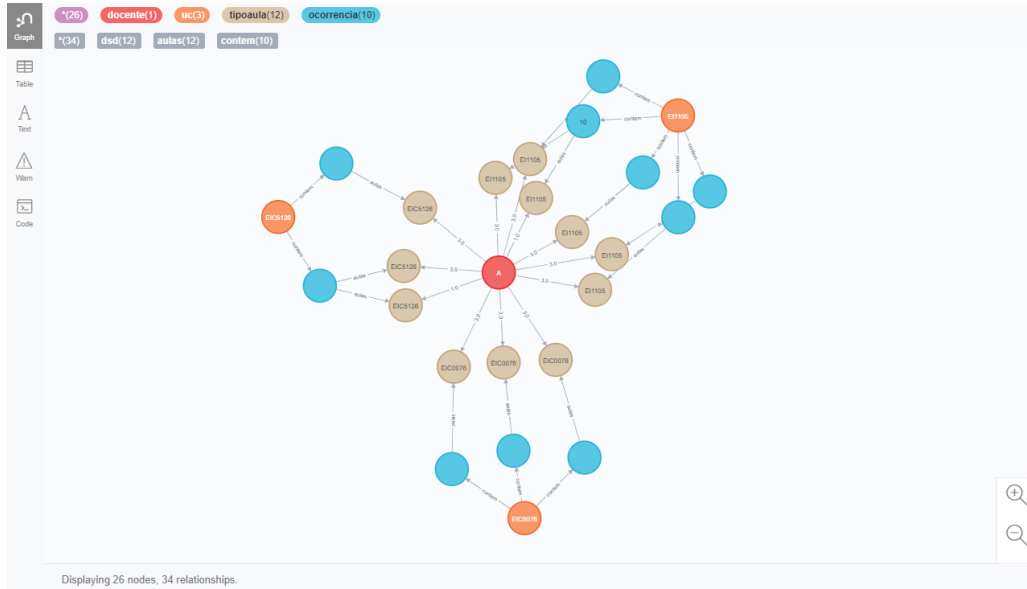


Figure 17: Question f) Neo4j answer

3 MongoDB, Neo4j and OracleSQL Comparison

In order to compare database performance, we decided to measure both the database size as well as average query processing time. To measure this processing time, we have ran the queries from a) to e) in all databases (Object Oriented Oracle SQL, MongoDB and Neo4j) and summed their execution times. The results can be found in table 1.

	OracleSQL	MongoDB	Neo4j
Data Size	20.25MB	47.55MB	26.0MB
Processing Time	8479ms	276ms	10363ms

Table 1: Different metrics comparison between the three database models

We can clearly see a difference in both data size and execution times. MongoDB is clearly a trade off between better execution times in exchange for data redundancy, which increases data size. Since all the information is contained in one document, rarely is there a need to perform join like operations which are usually the most expensive time wise. OracleSQL and Neo4j on the other hand, have a smaller data size with larger processing times. Besides one query, Neo4j actually had better results than OracleSQL, but the original data model is not the most appropriate for a graph like database so it makes sense that some queries designed for a relational database aren't as performant in Neo4j.

Regarding the queries, it's hard to be impartial when comparing the easiness to build the queries for all three models, since we were already familiar with the SQL syntax, but not as comfortable with the other languages. This being said, MongoDB was considerably easy to learn. The pipeline like query style is very intuitive which together with a good documentation made the process of building queries frictionless. When it comes to Neo4j cypher, building queries requires a slight change in perspective to think about the data in nodes and their connections as opposed to tables. However, since most queries weren't made for a graph like model, the results were mostly better represented as tables and as such many of the operators made specifically for graphs weren't explored. Nevertheless, the language shares a lot of common elements with SQL like languages and made it so the learning curve wasn't as steep. In summary, both model's languages were easy to learn and intuitive given the format with which they store the data.

4 Conclusion

By exploring these three different database models we were able to gather insight on their inner workings and appreciate their strengths when applied to the correct situations. After being very exposed to the standard relational database model, being introduced to different models was important to give some perspective on different possible solutions when designing databases. For example, if we are dealing with data that is not very interconnected, or if the highest priority is query speed, even if at the cost of memory (which is becoming an increasingly more common scenario nowadays), a document-oriented database like MongoDB might be a better choice. Alternatively, if we have data where the relationships closely resemble a graph like structure, naturally, a graph like database, such as Neo4j, could be worth exploring.

The main take away is that choosing a relational database model isn't necessarily always the best option. The data characteristics, how it will be used and the desired performance metrics should be taken into account in order to choose the model that will be the best fit.

Regarding our experience using the two models, we feel like some features of the NoSQL models weren't thoroughly explored, specially regarding Neo4j, because the original data was made for a relational database model. This meant that some of the relations and queries were somewhat awkward to translate into the other models and didn't fully take advantage of their characteristics. Nonetheless, it was still a useful experience to explore other NoSQL databases.

References

- [1] MongoDB. *MongoDB CRUD Operations — MongoDB Manual*. 2020. URL: <https://docs.mongodb.com/v4.4/crud/>.
- [2] Neo4j. *The Neo4j Cypher Manual v4.2 - Neo4j Cypher Manual*. 2020. URL: <https://neo4j.com/docs/cypher-manual/4.2/>.