



# NoSQL Assignment

## Group C

Telmo Alexandre Espírito Santo Baptista up201806554@edu.fe.up.pt  
Telmo Alexandre Espírito Santo Baptista up201806554@edu.fe.up.pt

13th June 2022

**TBD Project - 2021/22 - MEIC**

## Teachers

Gabriel de Sousa Torcato David gtd@fe.up.pt

# Index

<b>1</b>	<b>Introduction</b>	<b>3</b>
<b>2</b>	<b>SQL Baseline</b>	<b>3</b>
2.1	Data Model . . . . .	3
2.2	Exporting Data . . . . .	3
2.3	Queries . . . . .	3
<b>3</b>	<b>NoSQL</b>	<b>6</b>
3.1	Mongo Document Model . . . . .	6
3.2	Data Migration . . . . .	7
3.3	Queries . . . . .	8
<b>4</b>	<b>Neo4J</b>	<b>13</b>
4.1	Graph Model . . . . .	13
4.2	Data Migration . . . . .	14
4.3	Queries . . . . .	15
<b>5</b>	<b>TechnologiesComparison</b>	<b>17</b>
<b>6</b>	<b>Conclusion</b>	<b>18</b>

## 1 Introduction

Within the scope of the Databases Technologies course of Masters in Informatics and Computing Engineering, this project was developed with the objective of implementing MongoDB and Neo4J databases to represent the same data as a given SQL database populated with real data. As such, the data had to be migrated from SQL Developer into these technologies, afterwards implementing a set of queries based on relevant data needs to validate the new models.

## 2 SQL Baseline

### 2.1 Data Model

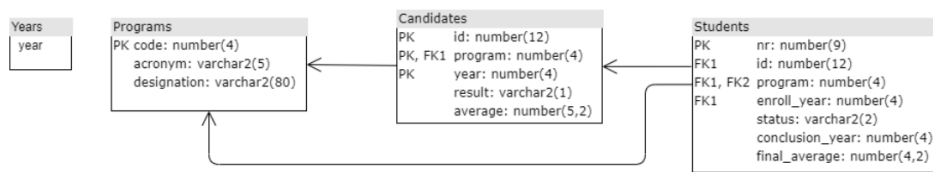


Figure 1: SQL Schema

### 2.2 Exporting Data

Our first approach towards migrating the database data was to simply export directly from SQL Developer. Although prone to many formatting errors when exporting comma separated files, this tool proved effective. Other formats were also supported, such as JSON, but the file generated only contained the direct rows from each table, which proved to be problematic when migrating into MongoDB.

### 2.3 Queries

#### Query A

‘Obtain the name of the program where the candidate 12147897 was enrolled.’

Listing 2.1: SQL Query A

```

-- Obtain the name of the program where the candidate 12147897 was enrolled
SELECT Programs.designation FROM GTD12.Students Students
INNER JOIN GTD12.Programs Programs ON Students.program = Programs.code
WHERE Students.id = 12147897;
  
```

	DESIGNATION
1	Engenharia Informática e Computação

Figure 2: Results of Query A

## Query B

‘Calculate the total number of students enrolled in each program, in each year, after 1991.’

Listing 2.2: SQL Query B

```
-- Calculate the total number of students enrolled in each program, in each year,
  after 1991
SELECT Programs.code, Programs.designation, Students.enroll_year,
       count(Students.nr) FROM GTD12.Students Students
INNER JOIN GTD12.Programs Programs ON Students.program = Programs.code
WHERE Students.enroll_year > 1991
GROUP BY Programs.code, Programs.designation, Students.enroll_year
ORDER BY Programs.designation, Students.enroll_year;
```

	CODE	DESIGNATION	ENROLL_YEAR	COUNT(STUDENTS.NR)
1	1093	Ciência da Informação	2002	30
2	233	Engenharia Civil	1992	133
3	233	Engenharia Civil	1993	135
4	233	Engenharia Civil	1994	129
5	233	Engenharia Civil	1995	163
6	233	Engenharia Civil	1996	156
7	233	Engenharia Civil	1997	158
8	233	Engenharia Civil	1998	168
9	233	Engenharia Civil	1999	201
10	233	Engenharia Civil	2000	211
11	233	Engenharia Civil	2001	222
12	233	Engenharia Civil	2002	222
13	318	Engenharia de Minas	1992	18
14	318	Engenharia de Minas	1993	9
15	318	Engenharia de Minas	1994	9
16	318	Engenharia de Minas	1995	10

Figure 3: Results of Query B

## Query C

‘Obtain the BI and the student number of the students with a final grade (med\_final) higher than the application grade (media).’

Listing 2.3: SQL Query C

```
-- Obtain the BI and the student number of the students with a final grade
-- (med_final) higher than the application grade (media).
SELECT Students.id, Students.nr FROM GTD12.Students Students
INNER JOIN GTD12.Candidates Candidates ON Students.id = Candidates.id and
       Students.program = Candidates.program and Students.enroll_year = Candidates.year
WHERE Students.final_average > Candidates.average;
```

## Query D

‘Find the average of the final grades of all the students finishing their program in a certain number of years, 5 years, 6 years, ...’

Listing 2.4: SQL Query D

	ID	NR
1	11040554	980503066
2	10346293	980503004
3	10769332	980503027
4	11232377	960506011
5	11235285	960506008

Figure 4: Results of Query C

```
-- Find the average of the final grades of all the students finishing their program
  in
-- a certain number of years, 5 years, 6 years, ...
SELECT Students.conclusion_year - Students.enroll_year AS program_duration,
       avg(Students.final_average) FROM GTD12.Students Students
GROUP BY Students.conclusion_year - Students.enroll_year
ORDER BY program_duration;
```

[illegible]

Figure 5: Results of Query D

### Query E

‘Which students applied multiple times to the program they enrolled in.’

Listing 2.5: SQL Query E

```
-- The students that applied multiple times to the same program
```

```

SELECT Students.id, Programs.code, Programs.designation, count(Candidates.year) AS
    applications_per_student FROM GTD12.Students Students
INNER JOIN GTD12.Candidates Candidates ON Students.id = Candidates.id and
    Students.program = Candidates.program
INNER JOIN GTD12.Programs Programs ON Candidates.program = Programs.code
GROUP BY Students.id, Candidates.program, Programs.code, Programs.designation
HAVING count(Candidates.year) > 1
ORDER BY applications_per_student DESC;

```

ID	CODE	DESIGNATION	APPLICATIONS_PER_STUDENT
1	10646064	255 Engenharia Electrotécnica e de Computadores	4
2	2720758	275 Engenharia Informática e Computação	4
3	7062064	304 Engenharia Mecânica	4
4	16159993	304 Engenharia Mecânica	4
5	10669452	304 Engenharia Mecânica	3
6	11518128	331 Engenharia Química	3
7	7864001	233 Engenharia Civil	3
8	6247957	255 Engenharia Electrotécnica e de Computadores	3
9	4465916	255 Engenharia Electrotécnica e de Computadores	3
10	7648131	233 Engenharia Civil	3
11	10047051	233 Engenharia Civil	3
12	3328591	233 Engenharia Civil	3
13	10139259	255 Engenharia Electrotécnica e de Computadores	3
14	9469304	331 Engenharia Química	3
15	11050745	233 Engenharia Civil	3
16	10539164	255 Engenharia Electrotécnica e de Computadores	3

Figure 6: Results of Query E

## 3 NoSQL

### 3.1 Mongo Document Model

To represent the same data as in the relational database, we decided to create two collections, one for Programs and another one for Students. Their specific structures can be found below.

Each program in its collection contains its candidates and the students enrolled in it. Each student in its collection contains which programs it applied and enrolled to.

This structure was chosen as we believe it strikes a good balance between query performance and data duplication. We believe this balance stays true to NoSQL's ideology of performing joins on insert instead of during queries.

Listing 3.1: Program Document

```

{
  "code": "",
  "acronym": "",
  "designation": "",
  "candidates": [
    {
      "id": 123,
      "year": 2000,
      "result": "C",
      "average": 14.6
    }
  ]
},

```

```
"students": [  
  {  
    "id": 123,  
    "nr": 20,  
    "enroll_year": 2001,  
    "status": "C",  
    "conclusion_year": 2006,  
    "final_average": 13.6  
  }  
]
```

Listing 3.2: Student Document

```
{  
  "id": 123,  
  "candidate_to": [  
    {  
      "program": {  
        "code": "",  
        "acronym": "",  
        "designation": ""  
      },  
      "year": 2000,  
      "result": "C",  
      "average": 14.6  
    }  
  ],  
  "enrolled_in": [  
    {  
      "program": {  
        "code": "",  
        "acronym": "",  
        "designation": ""  
      },  
      "nr": 20,  
      "enroll_year": 2001,  
      "status": "C",  
      "conclusion_year": 2006,  
      "final_average": 13.6  
    }  
  ]  
}
```

## 3.2 Data Migration

To import the data into Mongo, we first considered using Studio 3T's wizard. However, our trial accounts had already expired, so this option was no longer available for us. As an alternative, we resorted to using Python to first upload the data.

For this effect, we used the *pymongo* package to help interface with the Mongo database running on INESC's servers. Some problems arose regarding version mismatch, as the cluster is running Mongo v3.4 and the package had to be rolled back to an older version for compatibility.

The python script then takes the .csv files that were easily generated from SQL Developer and loaded them into Pandas Dataframes. The code is very inneficient as it was a quick and dirty solution. For a better alternative, the script would directly query the SQL database for the data before creating the json uploaded to MongoDB.

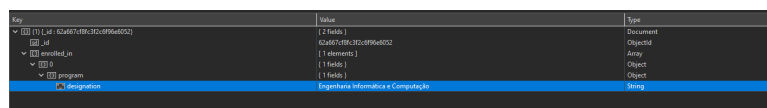
### 3.3 Queries

#### Query A

‘Obtain the name of the program where the candidate 12147897 was enrolled.’

Listing 3.3: Mongo Query A

```
db.students.find(
  {
    "id": 12147897,
  }, {
    "enrolled_in.program.designation": 1,
  })
```



Key	Value	Type
id	12147897	Objectid
enrolled_in	[{"program": "Engenharia Informatica e Computacao", "year": 2019, "enrollment": 1}]	Array
enrolled_in.program	Engenharia Informatica e Computacao	String

Figure 7: Results of Query A

#### Query B

‘Calculate the total number of students enrolled in each program, in each year, after 1991.’

Listing 3.4: Mongo Query B

```
db.getCollection("students").aggregate([
  {
    "$project": {
      "enrolled_in": {
        "$filter": {
          input: "$enrolled_in",
          as: "enrollment",
          cond: { $gt: ["$$enrollment.enroll_year", 1991] }
        }
      }
    }
  },
  { $unwind: "$enrolled_in" },
  {
    $group: {
      "_id": {
        "year": "$enrolled_in.enroll_year",
        "program": "$enrolled_in.program.code",
        "designation": "$enrolled_in.program.designation"
      }
    }
  }
])
```



```

    },
    "n_students": { $sum: 1 }
  }
},
{ $sort: { "_id.designation": 1, "_id.year": 1 } }
])

```

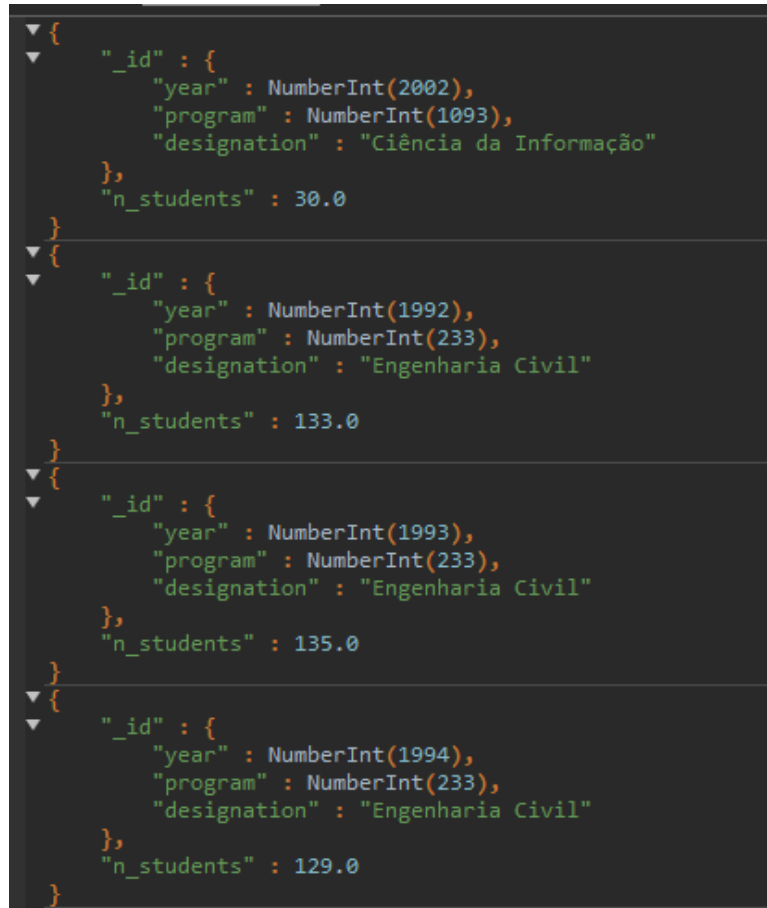


Figure 8: Results of Query B

## Query C

‘Obtain the BI and the student number of the students with a final grade (med\_final) higher than the application grade (media).’

Listing 3.5: Mongo Query C

```

db.getCollection("students").aggregate([
  {
    "$project": {
      "enrolled_in": {
        "$filter": {
          input: "$enrolled_in",
          as: "enrollment",
          cond: { $ifNull: ["$$enrollment.conclusion_year", false] }
        }
      }
    }
  }
])

```

```

    },
    "candidate_to": 1,
    "id": 1,
  }
},
{ $unwind: "$enrolled_in" },
{
  "$project": {
    "candidate_to": {
      "$filter": {
        input: "$candidate_to",
        as: "candidature",
        cond: {
          $and: [
            { $eq: ["$$candidature.program.code",
              "$enrolled_in.program.code" ] },
            { $ifNull: ["$$candidature.average", false] },
            { $gt: ["$enrolled_in.final_average",
              "$$candidature.average" ] }
          ]
        }
      }
    },
    "nr": "$enrolled_in.nr",
    "id": 1
  }
},
{ $unwind: "$candidate_to" },
{
  "$project": {
    "id": 1,
    "nr": 1
  }
}
])

```

## Query D

‘Find the average of the final grades of all the students finishing their program in a certain number of years, 5 years, 6 years, ...’

Listing 3.6: Mongo Query D

```

db.getCollection("students").aggregate([
  {
    "$project": {
      "enrolled_in": {
        "$filter": {
          input: "$enrolled_in",
          as: "enrollment",
          cond: { $ifNull: ["$$enrollment.conclusion_year", false] }
        }
      },
      "candidate_to": 1
    }
  }
])

```



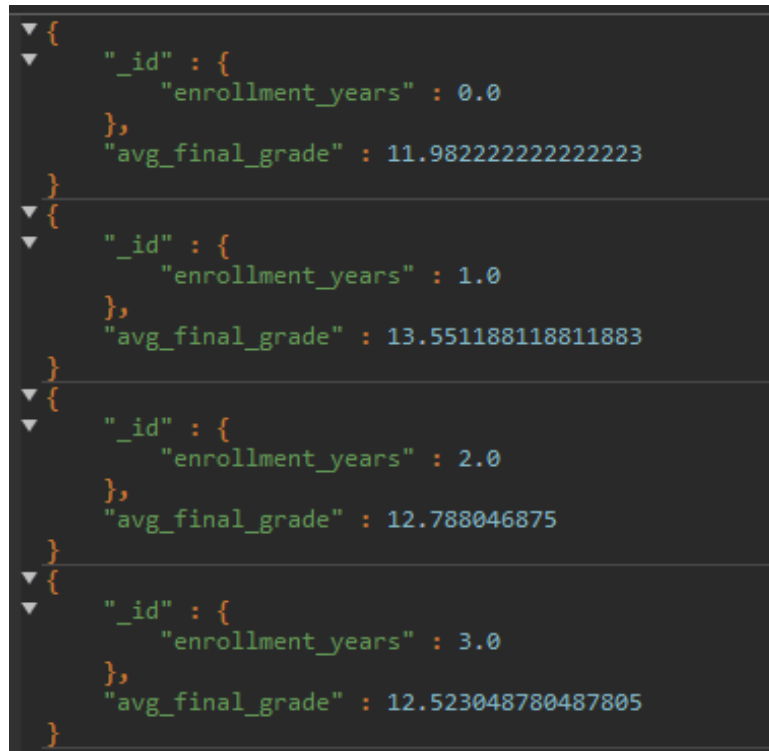
Figure 9: Results of Query C

```

    }
  },
  { $unwind: "$enrolled_in" },
  {
    "$project": {
      "candidate_to": {
        "$filter": {
          input: "$candidate_to",
          as: "candidature",
          cond: { $eq: ["$$candidature.program.code",
            "$enrolled_in.program.code"] }
        }
      },
      "enrolled_in": 1,
    }
  },
  {
    $addFields: {
      enrollment_years: { $subtract: ["$enrolled_in.conclusion_year",
        "$enrolled_in.enroll_year"] }
    }
  },
  {
    $group: {
      "_id": { "enrollment_years": "$enrollment_years" },
      "avg_final_grade": { $avg: "$enrolled_in.final_average" }
    }
  },
  { $sort: { "_id.enrollment_years": 1 } }
}

```

1)



{	"_id" : { "enrollment_years" : 0.0 }, "avg_final_grade" : 11.982222222222223 }
{	"_id" : { "enrollment_years" : 1.0 }, "avg_final_grade" : 13.551188118811883 }
{	"_id" : { "enrollment_years" : 2.0 }, "avg_final_grade" : 12.788046875 }
{	"_id" : { "enrollment_years" : 3.0 }, "avg_final_grade" : 12.523048780487805 }

Figure 10: Results of Query D

## Query E

‘Which students applied multiple times to the program they enrolled in.’

Listing 3.7: Mongo Query E

```
db.getCollection("students").aggregate([
  { $unwind: "$enrolled_in" },
  {
    "$project": {
      "candidate_to": {
        "$filter": {
          input: "$candidate_to",
          as: "candidature",
          cond: { $eq: [ "$$candidature.program.code",
            "$enrolled_in.program.code" ] }
        }
      },
      "id": 1
    }
  },
  { $unwind: "$candidate_to" },
  {
    $group: {
      "_id": {
        "id": "$id",
```

```

        "code": "$candidate_to.program.code",
        "designation": "$candidate_to.program.designation",
    },
    "n_applications": { $sum: 1 }
  }
},
{ $sort: { "n_applications": -1 } }
])

```



```

[
  {
    "_id": {
      "id": NumberInt(7062064),
      "code": NumberInt(304),
      "designation": "Engenharia Mecânica"
    },
    "n_applications": 4.0
  },
  {
    "_id": {
      "id": NumberInt(16159993),
      "code": NumberInt(304),
      "designation": "Engenharia Mecânica"
    },
    "n_applications": 4.0
  },
  {
    "_id": {
      "id": NumberInt(10646064),
      "code": NumberInt(255),
      "designation": "Engenharia Electrotécnica e de Computadores"
    },
    "n_applications": 4.0
  },
  {
    "_id": {
      "id": NumberInt(2720758),
      "code": NumberInt(275),
      "designation": "Engenharia Informática e Computação"
    },
    "n_applications": 4.0
  },
  {
    "_id": {
      "id": NumberInt(10255006),
      "code": NumberInt(315),
      "designation": "Engenharia Metalúrgica e de Materiais"
    },
    "n_applications": 3.0
  },
  {
    "_id": {
      "id": NumberInt(11050745),
      "code": NumberInt(233),
      "designation": "Engenharia Civil"
    },
    "n_applications": 3.0
  }
]

```

Figure 11: Results of Query E

## 4 Neo4J

### 4.1 Graph Model

The translation of the SQL schema into Neo4J results in a very simple model, consisting of only four labels. As such, we believe that Neo4J's mantra of 'relationships are as important as the data they are connecting' was accurate, as applying or enrolling into a program is as important as the person or the program by themselves.

In the resulting model, Whenever someone applies for a program, a node is created with their ID and is given the `:PERSON` label and a `:CANDIDATE_TO` relationship is added connecting it to the respective program.

When a student is enrolled in a program, we give it another label (`:STUDENT`) and add a `:ENROLLED_IN` relationship to the respective program. This system of having a person and student label allow for simpler queries, as selecting applicants or students becomes trivial.

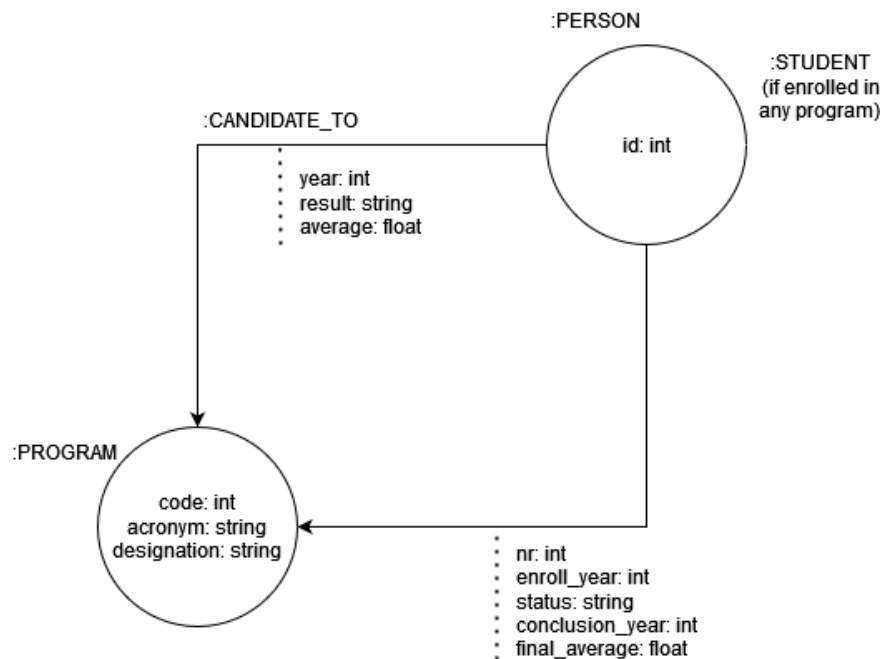


Figure 12: Graph Model

## 4.2 Data Migration

The data migration process was simple and followed the examples set by the course materials. Without the creation of indexes, this process took over 10 seconds, however, after creating indexes for the main labels, it was in the order of milliseconds.

A point that confused us was that the syntax for more fine-grained control over the indexes was considered deprecated, although the documentation referred to it as "the correct method". Deprecating features whose replacement has not yet been released is an odd tactic that caught us by surprise.

We also planned on setting the relevant unique constraints, however, some of them were better suited to be node key constraints, a feature locked behind the enterprise edition. Both of these constraints would implicitly create indexes, however, we decided to only apply the indexes manually.

In the script, we used `MERGE` instead of `Match` to ensure that if a `:Person` node with the respective ID didn't exist, it would be created and used for the query. The documentation warns us that this doesn't ensure uniqueness, however, as, for example, sharding, could result in multiple nodes with the same data being created. Fortunately, this did not occur anytime during our tests.

Listing 4.1: Neo4J Data Import

```

MATCH (x) DETACH DELETE x;

CREATE INDEX IF NOT EXISTS FOR (p:Program) ON (p.code);
CREATE INDEX IF NOT EXISTS FOR (p:Person) ON (p.id);
CREATE INDEX IF NOT EXISTS FOR (s:Student) ON (s.nr);

LOAD CSV WITH HEADERS FROM 'file:///year.csv' AS line
CREATE (:Year { year: toInteger(line.YEAR) });

LOAD CSV WITH HEADERS FROM 'file:///programs.csv' AS line
CREATE ( :Program {
    code: toInteger(line.CODE),
    acronym: line.ACRONYM,
    designation: line.DESIGNATION
});

LOAD CSV WITH HEADERS FROM 'file:///candidates.csv' AS line
MATCH (prog:Program {code: toInteger(line.PROGRAM)})
MERGE (candidate:Person { id: toInteger(line.ID) })
CREATE (candidate) - [:CANDIDATE_TO {
    year: toInteger(line.YEAR),
    result: line.RESULT,
    average: toFloat(line.AVERAGE)
}] -> (prog);

LOAD CSV WITH HEADERS FROM 'file:///students.csv' AS line
MATCH (s:Person {id: toInteger(line.ID)} )
MATCH (prog: Program { code: toInteger(line.PROGRAM) } )
SET s:Student
CREATE (s) - [:ENROLLED_IN {
    nr: toInteger(line.NR),
    enroll_year: toInteger(line.ENROLL_YEAR),
    status: line.STATUS,
    conclusion_year: toInteger(line.CONCLUSION_YEAR),
    final_average: toFloat(line.FINAL_AVERAGE)
} ] -> (prog);

```

## 4.3 Queries

### Query A

‘Obtain the name of the program where the candidate 12147897 was enrolled.’

Listing 4.2: Neo4J QueryAB

```

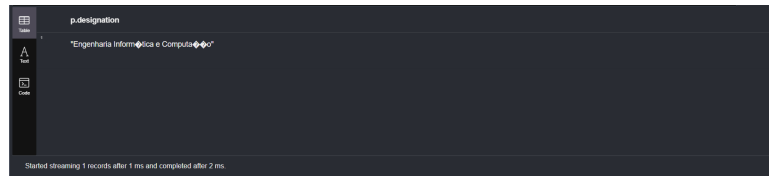
MATCH (:Person{id: 12147897 }) - [:CANDIDATE_TO] -> (p:Program)
RETURN p.designation;

```

### Query B

‘Calculate the total number of students enrolled in each program, in each year, after 1991.’

Listing 4.3: Neo4J Query B



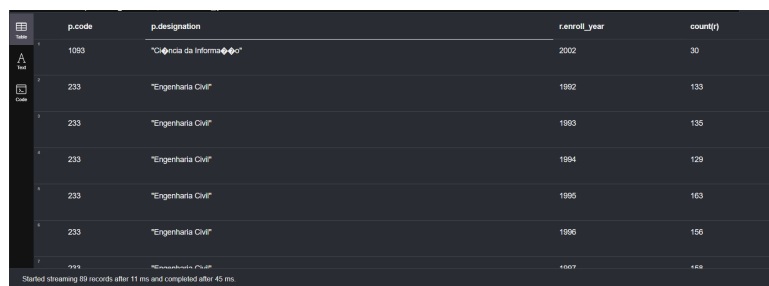
p.code	p.designation	r.enroll_year	count(r)
1083	"Engenharia Informatica e Computacao"	2002	30

Figure 13: Results of Query A

```

MATCH (:Student) - [r:ENROLLED_IN ] -> (p:Program)
WHERE r.enroll_year > 1991
RETURN p.code, p.designation, r.enroll_year, count(r)
ORDER BY p.designation, r.enroll_year

```



p.code	p.designation	r.enroll_year	count(r)
1083	"Engenharia Informatica e Computacao"	2002	30
233	"Engenharia Civil"	1992	133
233	"Engenharia Civil"	1993	135
233	"Engenharia Civil"	1994	129
233	"Engenharia Civil"	1995	163
233	"Engenharia Civil"	1996	156
1993	"Engenharia Civil"	1997	148

Figure 14: Results of Query B

## Query C

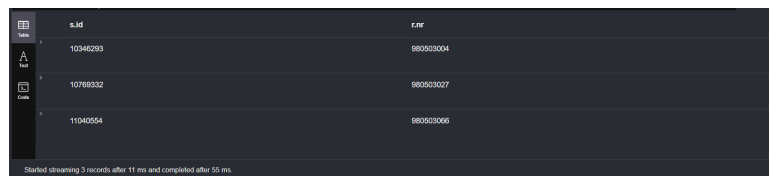
‘Obtain the BI and the student number of the students with a final grade (med\_final) higher than the application grade (media).’

Listing 4.4: Neo4J Query C

```

MATCH (s:Student) - [r:ENROLLED_IN] -> (p:Program)
MATCH (s) - [c:CANDIDATE_TO] -> (p)
WHERE c.average < r.final_average
RETURN s.id, r.nr

```



s.id	r.nr
10346293	980503004
10769332	980503027
11040954	980503066

Figure 15: Results of Query C

## Query D

‘Find the average of the final grades of all the students finishing their program in a certain number of years, 5 years, 6 years, ...’

Listing 4.5: Neo4J Query D



```

MATCH (s:Student) - [r:ENROLLED_IN {status: 'C'} ] -> (p:Program)
WITH *, r.conclusion_year - r.enroll_year AS duration
RETURN duration, avg(r.final_average)
ORDER BY duration

```

	duration	avg(r.final_average)
0		11.888888888888889
1		13.489049504950495
2		12.710837499999999
3		12.42682926829268
4		13.48738827620235
5		12.351415094338629

Started streaming 24 records after 1 ms and completed after 10 ms.

Figure 16: Results of Query D

## Query E

‘Which students applied multiple times to the program they enrolled in.’

Listing 4.6: Neo4J Query E

```

MATCH (s:Student) -[:ENROLLED_IN]-> (p:Program)
WITH *

MATCH (s) -[:CANDIDATE_TO]-> (p)
WITH s AS student, p AS program, count(p) AS applications_per_student

WHERE applications_per_student > 1
RETURN student.id, program.code, program.designation, applications_per_student
ORDER BY applications_per_student DESC

```

student.id	program.code	program.designation	applications_per_student
10648064	255	"Engenharia Eletrotônica e de Computadores"	4
7062084	304	"Engenharia Mecânica"	4
16158983	304	"Engenharia Mecânica"	4
2720798	275	"Engenharia Informática e Computação"	4
10138259	255	"Engenharia Eletrotônica e de Computadores"	3
6247867	255	"Engenharia Eletrotônica e de Computadores"	3
44444444	914	"Engenharia Eletrotônica e de Computadores"	3

Started streaming 209 records after 14 ms and completed after 73 ms.

Figure 17: Results of Query E

## 5 Technologies Comparison

Given that all database management systems are running on different hardware under different circumstances, it is high impossible (and unfair) to empirically determine which one is the fastest. Another factor that could affect the comparison is the data size. As the data required to process increases, vertical solutions such as SQL tend to become less efficient, whilst NoSQL solutions are able to outscale the former.

	Data Size
SQL	0,94 MiB
MongoDB	19,2 MiB
Neo4J	8,24 MiB

If we were to compare performance, however, Neo4J has a relevant aspect exclusive to its enterprise edition, which is the ability to run queries in PIPELINE mode instead of INTERPRETED. According to Neo4J's developer blog posts, the same Cypher query execution time dropped from 1847 ms to 744 ms by adding a relevant index, but by simply running the query in PIPELINE mode, its execution time was further lowered to 507 ms. In a nutshell, this PIPELINE mode can be described as SQL's query optimizer, so it greatly improves query times to any project without any intervention from a database administrator.

Regarding database size, theoretically, Neo4J should have some extra 80MB somewhere, but we did not find them.

MongoDB ended up occupying the most space, as it was also expected, due to the high redundancy model we chose to follow through, however, it has a trade-off of high performance to query speeds, making it a powerful specially for big data.

In terms of query easiness, SQL and Cypher come out on top. SQL is so widely used that its syntax is already ingrained in our minds. Cypher is very simple and intuitive by design, thanks to the graph structure. Nevertheless, it has some hidden pitfalls and problems with deprecated syntax that can complicate things. NoSQL's syntax is in equal parts convoluted and powerful, resulting in great query flexibility, but at the cost of easiness to query.

## 6 Conclusion

All three database systems are powerful and usable, each having their own specializations. SQL, was the easiest to query on, due to also its strict structured data model, it is a very powerful tool to model databases and is also highly scalable. However, the query performance can be affected really heavily if the database and the query itself aren't properly optimized, especially when dealing with big data. On the opposite spectrum, NoSQL had complex yet powerful querying capabilities, its high redundant model leads to data being always available "everywhere and anywhere", trading off the database size for a highly performant query speed. NoSQL is therefore a powerful database when dealing with large amounts of data and naturally unstructured data. Finally, Neo4j also proved to be a powerful tool when the database schema permits so, this is, when our data has inherent relationships between them, such as our case, Neo4j becomes an easy and powerful tool to model on. However, on data without inherent relationships, such as time series data, this database system can be quite hard to manage, and often not the best choice.