

Faculdade de Engenharia da Universidade do Porto



## Relatório TBDA

### *3.º Projeto*

### *NOSQL ASSIGNMENT 3 - Cultural Facilities*

Turma 1

João Augusto dos Santos Lima, up201605314@fe.up.pt

Susana Maria de Sousa Lima, up201603634@fe.up.pt

# Índice

<b>Introdução</b>	<b>3</b>
<b>Modelo NoSQL</b>	<b>4</b>
Mongo	4
Neo4j	5
<b>Migração de dados</b>	<b>8</b>
Mongo	8
Neo4j	9
<b>Perguntas</b>	<b>13</b>
Pergunta a	13
Mongo	13
Interrogação	13
Resultados	13
Neo4j	15
Interrogação	15
Resultados	15
Pergunta b	16
Mongo	16
Interrogação	16
Resultados	16
Neo4j	17
Interrogação	17
Resultados	17
Pergunta c	17
Mongo	18
Interrogação	18
Resultados	18
Neo4j	18
Interrogação	18
Resultados	18
Pergunta d	19
Mongo	19
Interrogação	19
Resultados	19
Neo4j	20
Interrogação	20
Resultados	21
Pergunta e	21
Mongo	21

Interrogação	21
Resultados	22
Neo4j	22
Interrogação	22
Resultados	22
Pergunta f	23
Mongo	23
Interrogação	23
Resultados	23
Neo4j	24
Interrogação	24
Resultados	25
<b>Comparação Mongo, Neo4j e Oracle</b>	<b>26</b>
Tempos de execução	26
Tamanho da informação	27
Facilidade da Query	29
Comentários Adicionais	29
<b>Conclusão</b>	<b>30</b>

# Introdução

O projeto, realizado no âmbito da unidade curricular de Tecnologia de Bases de Dados do Mestrado Integrado em Engenharia Informática e Computação, consiste na implementação de um modelo de dados usando duas abordagens distintas do paradigma não relacional (NoSQL), um banco de dados de documentos (MongoDB) e um banco de dados de gráficos (Neo4j). No fim é feita uma comparação destas duas abordagens entre si e com a abordagem relacional.

# Modelo NoSQL

## Mongo

Num momento inicial foram consideradas duas abordagens distintas para implementação do modelo em *Mongo*. A primeira passa por criar diversas coleções, mantendo uma coleção principal com a informação essencial e diversas secundárias com a restante informação relevante. Por outro lado, a segunda abordagem consiste em manter um único documento cujas associações estão presentes em subdocumentos embutidos no documento principal. A primeira abordagem permite reduzir a redundância de informação no modelo, no entanto, não existindo operações nativas de *join*, as interrogações podem se tornar bastante complexas. Deste modo, e visto não existir grande redundância no modelo para o problema em questão, o grupo optou pela segunda abordagem.

O modelo implementado representa uma *municipality*. Cada *municipality* é constituída por um código oficial, uma designação e está localizada num distrito e numa região. De forma a preservar as relações *Municipalities/Districts* e *Municipalities/Regions* presentes no modelo relacional, essas informações foram adicionadas em dois subdocumentos (*DISTRICT* e *REGION*, respetivamente) embutidos no documento principal. É de referir que cada *DISTRICT* possui um campo *REGION* que representa o código da região correspondente caso exista e toma o valor *null* caso contrário, de modo a manter a relação *Districts/Regions*. O mesmo raciocínio se aplica à relação *Municipalities/Facilities*, sendo neste caso, adicionada uma lista de subdocumentos *facilities*. De igual modo, as relações *Facilities/Uses/Activities* e *Facilities/Roomtypes* são preservadas adicionando os respetivos subdocumentos (*ACTIVITIES* e *ROOMTYPES*) a cada subdocumento *facility*.

A coleção criada (*municipalities*) tem, por conseguinte, 308 documentos (correspondendo às 308 municipalities), com a seguinte estrutura:

```
{
  "_id" : ObjectId("5ed4340ce9c7b10884c3f927"),
  "COD" : NumberLong(203),
  "DESIGNATION" : "Alvito",
  "REGION" : {
    "COD" : NumberLong(4),
    "DESIGNATION" : "Alentejo",
    "NUT1" : "Continente"
  },
  "DISTRICT" : {
    "COD" : NumberLong(2),
    "DESIGNATION" : "Beja",
    "REGION" : NumberLong(4)
  },
}
```

```

"FACILITIES" : [
  {
    "ID" : NumberLong(974),
    "NAME" : "SALA DE ESPECTÁCULOS DO CENTRO CULTURAL DO ALVITO",
    "CAPACITY" : NumberLong(100),
    "ADDRESS" : "LG DO RELÓGIO-EDIF DA CÂMARA MUNICIPAL",
    "ROOMTYPE" : {
      "ROOMTYPE" : NumberLong(8),
      "DESCRIPTION" : "Outras salas de espetáculo"
    },
    "ACTIVITIES" : [
      {
        "REF" : "1",
        "ACTIVITY" : "cinema"
      },
      {
        "REF" : "6",
        "ACTIVITY" : "teatro"
      }
    ]
  }
]
}

```

## Neo4j

A idealização de um modelo em *Neo4j*, análogo ao modelo relacional proposto, revelou-se relativamente simples uma vez que apenas foram criados nós para as tabelas mais relevantes e estabelecidas as relações entre eles necessárias para preservar a informação presente no modelo SQL. Em cada nó são guardadas como propriedades as informações presentes na tabela correspondente.

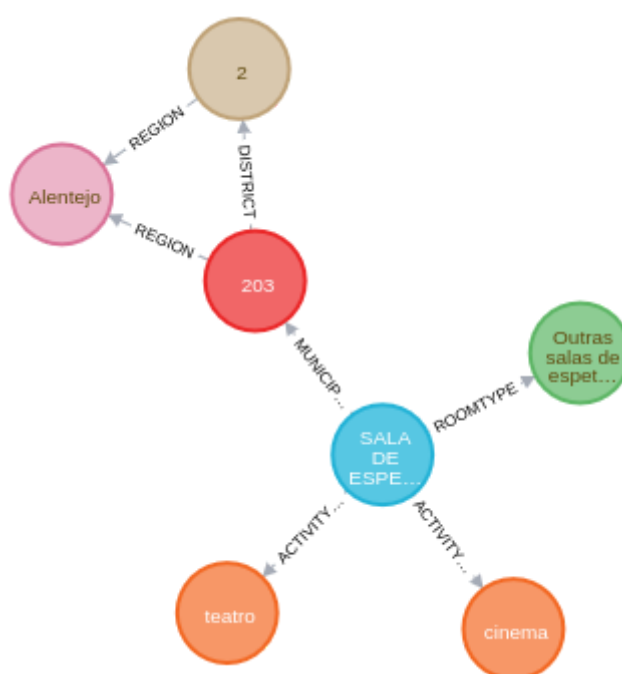
Deste modo, foram criados os seguintes tipos de nós:

- **Activities**
- **Facilities**
- **Roomtypes**
- **Municipalities**
- **Districts**
- **Regions**

É de notar que não foi criado um nó para a tabela *Uses* uma vez que esta apenas serve para estabelecer as ligações entre *Facilities* e *Activities* (uma *facility* pode ter zero ou mais *activities*), sendo esta informação representada através de relações entre os nós *Facilities* e *Activities* no grafo. Desta forma foram estabelecidas as seguintes relações:

- **ACTIVITY\_TYPE**: relação de *Facilities* para *Activities*
- **ROOMTYPE**: relação de *Facilities* para *Roomtypes*
- **MUNICIPALITY**: relação de *Facilities* para *Municipalities*
- **DISTRICT**: relação de *Municipalities* para *Districts*
- **REGION**: relação de *Municipalities* para *Regions* e relação de *Districts* para *Regions*

A figura seguinte representa um fragmento do grafo obtido, no qual estão presentes todos os tipos de nós e relações estabelecidas entre eles.



Os números totais de nós e de relações do grafo final podem ser observados nas seguintes tabelas:

Tipo de nó	Número de nós
<i>Activities</i>	6
<i>Districts</i>	20
<i>Facilities</i>	1084
<i>Municipalities</i>	308
<i>Regions</i>	7

<i>Roomtypes</i>	18
------------------	----

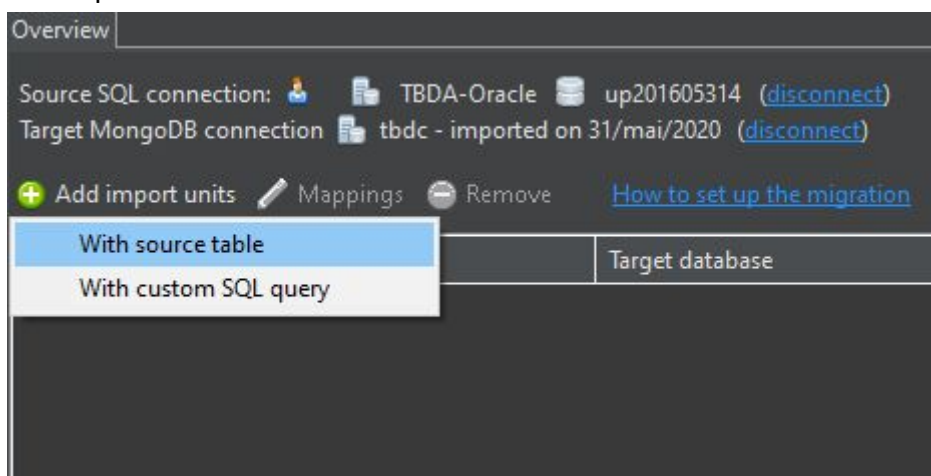
<b>Tipo de relação</b>	<b>Número de relações</b>
<i>ACTIVITY_TYPE</i>	2059
<i>ROOMTYPE</i>	1084
<i>MUNICIPALITY</i>	1084
<i>DISTRICT</i>	308
<i>REGION</i>	322

# Migração de dados

## Mongo

De forma a migrar os dados para a base de dados *Mongo*, recorreu-se ao seguinte método:

1. Usar a aplicação [Studio 3T](#)
2. Migração SQL (SQL -> Migração MongoDB)
3. Selecionar a conexão na *overview*. (Source SQL connection e Target MongoDB connection)
4. Add import units -> With source table



5. Selecione *GTD8.table.municipalities*
6. Com *GTD8.municipalities* selecionado -> mappings
7. Adicionar 2 relações *one-to-one* para *district* e *region*
8. Adicionar um *array* a *facilities*
9. Nas *facilities*, adicionar relação *one-to-one* para *roomtype* e um *array* para *activities* (para eliminar a tabela de ligação *uses*: criar um array para *uses* e de *uses* uma relação *one-to-one* para *activities*; através da interface gráfica do studio 3T *drag-and-drop* o objeto de *activities* para dentro do objeto de *uses*, eliminar os atributos desnecessários e mudar o nome do array *uses* para *activities*)

The screenshot shows the Neo4j Studio interface with the 'Overview' tab selected. The main window is divided into three sections:

- Schema view:** A tree view showing the database schema. The root is 'Municipalities + 2', which contains 'Regions' and 'Districts'. 'Regions' contains 'Regions' and 'Regions'. 'Districts' contains 'Districts' and 'Districts'. 'Facilities' is also shown, containing 'Facilities' and 'Facilities'.
- Datasets view:** A table showing the datasets and their relationships. The table has columns: Table/Query, Alias, and Joined on. The datasets are:
 

Table/Query	Alias	Joined on
GTD8.Municipalities	Municipalities	
GTD8.Regions	Regions	Regions.COD = Municipalities.Region
GTD8.Districts	Districts	Districts.COD = Municipalities.District
GTD8.Facilities	Facilities	Facilities.Municipality = Municipalities.COD
GTD8.Roomtypes	Roomtypes	Roomtypes.Roomtype = Facilities.Roomtype
GTD8.Uses	Uses	Uses.ID = Facilities.ID
GTD8.Activities	Activities	Activities.Ref = Uses.Ref
- JSON output preview:** A preview of the JSON output for the selected dataset. It shows a list of objects, each representing a facility. The first object is for 'SALA DE ESPECTÁCULOS CASA DO POVO VALONGO DO VOUGA' and the second is for 'CINE TEATRO SÃO PEDRO DE ÁGUEDA'.

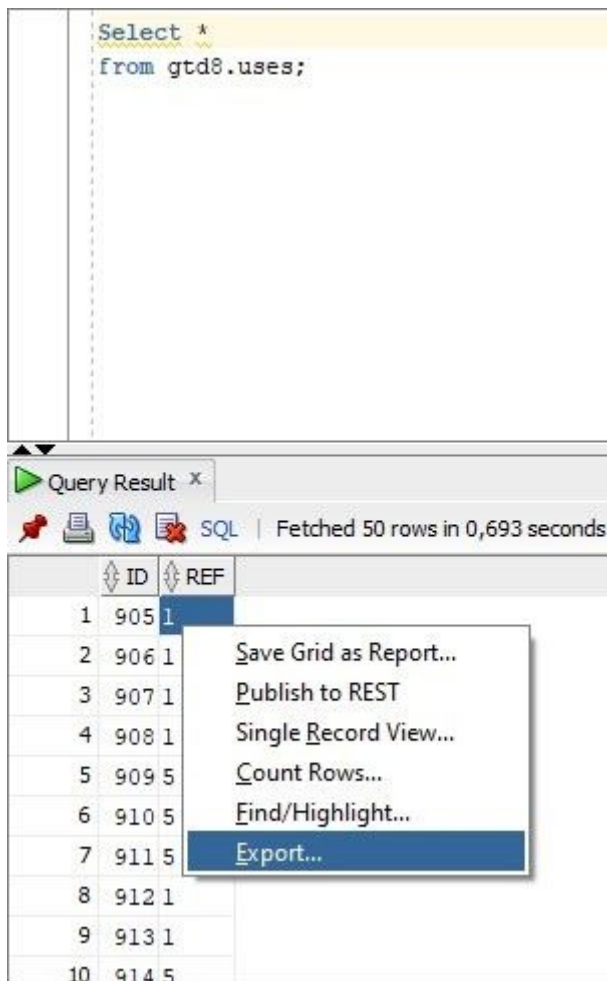
## Neo4j

De forma a migrar os dados para a base de dados *Neo4j*, recorreu-se ao seguinte método:

1. Obter o ficheiro csv correspondente a cada tabela. Para tal, executar a seguinte query no *sql developer*:

```
Select *
From gtd8.<table>;
```

em que *<table>* corresponde ao nome da tabela (*uses*, *activities*, *facilities*, *roomtypes*, *municipalities*, *districts* e *regions*) e exportar os resultados obtidos como ficheiro csv.



2. Mover os ficheiros csv obtidos para o diretório:

```
/path/to/folder/.Neo4jDesktop/neo4jDatabases/[database]/installation-4.0.3/import
```

3. Executar as seguintes *queries*:

```
// Create activities
LOAD CSV WITH HEADERS FROM 'file:///activities.csv' AS row
MERGE (activities:Activities {activityREF: toInteger(row.REF)})
  ON CREATE SET activities.activity = row.ACTIVITY;

// Create facilities
LOAD CSV WITH HEADERS FROM 'file:///facilities.csv' AS row
MERGE (fac:Facilities {facilityID: toInteger(row.ID)})
  ON CREATE SET fac.name = row.NAME, fac.capacity = toInteger(row.CAPACITY),
  fac.roomtype = toInteger(row.ROOMTYPE), fac.address = row.ADDRESS,
  fac.municipality = toInteger(row.MUNICIPALITY);

// Create roomtype
LOAD CSV WITH HEADERS FROM 'file:///roomtypes.csv' AS row
```

```

MERGE (room:Roomtypes {roomtypeID: toInteger(row.ROOMTYPE)})
  ON CREATE SET room.description = row.DESCRPTION;

// Create municipality
LOAD CSV WITH HEADERS FROM 'file:///municipalities.csv' AS row
MERGE (mun:Municipalities {municipalityCOD: toInteger(row.COD)})
  ON CREATE SET mun.designation = row.DESIGNATION, mun.district =
toInteger(row.DISTRICT), mun.region = toInteger(row.REGION);

// Create districts
LOAD CSV WITH HEADERS FROM 'file:///districts.csv' AS row
MERGE (dis:Districts {districtCOD: toInteger(row.COD)})
  ON CREATE SET dis.designation = row.DESIGNATION, dis.region =
toInteger(row.REGION);

// Create regions
LOAD CSV WITH HEADERS FROM 'file:///regions.csv' AS row
MERGE (reg:Regions {regionCOD: toInteger(row.COD)})
  ON CREATE SET reg.description = row.DESIGNATION, reg.nut1 = row.NUT1;

CREATE INDEX uses_id_ref FOR (u:Uses) ON (u.usesID, u.usesREF);
CREATE INDEX activity_ref FOR (act:Activities) ON (act.activityREF);
CREATE INDEX facility_id FOR (fac:Facilities) ON (fac.facilityID);
CREATE INDEX roomtype_id FOR (room:Roomtypes) ON (room.roomtypeID);
CREATE INDEX municipalities_cod FOR (mun:Municipalities) ON
(mun.municipalityCOD);
CREATE INDEX districts_cod FOR (dis:Districts) ON (dis.districtCOD);
CREATE INDEX regions_cod FOR (reg:Regions) ON (reg.regionCOD);

// Create relation between facilities and activities
LOAD CSV WITH HEADERS FROM 'file:///uses.csv' AS row
MATCH (fac:Facilities {facilityID: toInteger(row.ID)})
MATCH (act:Activities {activityREF: toInteger(row.REF)})
CREATE (fac)-[:ACTIVITY_TYPE]->(act);

// Create relation between facilities and roomtype
LOAD CSV WITH HEADERS FROM 'file:///facilities.csv' AS row
MATCH (fac:Facilities {facilityID: toInteger(row.ID)})
MATCH (room:Roomtypes {roomtypeID: toInteger(row.ROOMTYPE)})
MERGE (fac)-[:ROOMTYPE {type: room.description}]->(room);

// Create relation between facilities and municipalitites
LOAD CSV WITH HEADERS FROM 'file:///facilities.csv' AS row
MATCH (fac:Facilities {facilityID: toInteger(row.ID)})
MATCH (mun:Municipalities {municipalityCOD: toInteger(row.MUNICIPALITY)})
MERGE (fac)-[:MUNICIPALITY]->(mun);

// Create relation between municipalities and districts
LOAD CSV WITH HEADERS FROM 'file:///municipalities.csv' AS row
MATCH (mun:Municipalities {municipalityCOD: toInteger(row.COD)})

```

```
MATCH (dist:Districts {districtCOD: toInteger(row.DISTRICT)})
MERGE (mun)-[:DISTRICT]->(dist);

// Create relation between municipalities and regions
LOAD CSV WITH HEADERS FROM 'file:///municipalities.csv' AS row
MATCH (mun:Municipalities {municipalityCOD: toInteger(row.COD)})
MATCH (reg:Regions {regionCOD: toInteger(row.REGION)})
MERGE (mun)-[:REGION]->(reg);

// Create relation between districts and regions
LOAD CSV WITH HEADERS FROM 'file:///districts.csv' AS row
MATCH (dist:Districts {districtCOD: toInteger(row.COD)})
MATCH (reg:Regions {regionCOD: toInteger(row.REGION)})
MERGE (dist)-[:REGION]->(reg);
```

# Perguntas

## Pergunta a

*Which are the facilities where the room type description contains 'touros' and have 'teatro' as one of their activities? Show the id, name, description and activity.*

## Mongo

### Interrogação

```
db.municipalities.aggregate([
  {$unwind: "$FACILITIES"},
  {$project: {
    "_id": 0,
    "FACILITIES.ID": 1,
    "FACILITIES.NAME": 1,
    "FACILITIES.ROOMTYPE.DESRIPTION": 1,
    "FACILITIES.ACTIVITIES.ACTIVITY": 1
  }},
  {$match: {
    "FACILITIES.ACTIVITIES.ACTIVITY": "teatro",
    "FACILITIES.ROOMTYPE.DESRIPTION": /touros/,
  }},
],
)
```

## Resultados

```
/* 1 */
{
  "FACILITIES" : {
    "ID" : NumberLong(940),
    "NAME" : "ARENA DE ÉVORA - EX PRAÇA DE TOIROS",
    "ROOMTYPE" : {
      "DESCRIPTION" : "Praça de touros multiusos"
    },
    "ACTIVITIES" : [
      {
```

```

        "ACTIVITY" : "dança"
    },
    {
        "ACTIVITY" : "música"
    },
    {
        "ACTIVITY" : "tauromaquia"
    },
    {
        "ACTIVITY" : "teatro"
    }
]
}
}

/* 2 */
{
    "FACILITIES" : {
        "ID" : NumberLong(957),
        "NAME" : "COLISEU DE REDONDO - EX PRAÇA DE TOIROS",
        "ROOMTYPE" : {
            "DESCRIPTION" : "Praça de touros multiusos"
        },
        "ACTIVITIES" : [
            {
                "ACTIVITY" : "dança"
            },
            {
                "ACTIVITY" : "música"
            },
            {
                "ACTIVITY" : "tauromaquia"
            },
            {
                "ACTIVITY" : "teatro"
            }
        ]
    }
}

/* 3 */
{
    "FACILITIES" : {
        "ID" : NumberLong(916),
        "NAME" : "COLISEU JOSÉ RONDÃO DE ALMEIDA-EX PRAÇA DE TOIROS",

```

```

    "ROOMTYPE" : {
        "DESCRIPTION" : "Praça de touros multiusos"
    },
    "ACTIVITIES" : [
        {
            "ACTIVITY" : "dança"
        },
        {
            "ACTIVITY" : "música"
        },
        {
            "ACTIVITY" : "tauromaquia"
        },
        {
            "ACTIVITY" : "teatro"
        }
    ]
}

```

## Neo4j

### Interrogação

```

MATCH (a:Activities)<-[:ACTIVITY_TYPE]-(f:Facilities)-[:ROOMTYPE]->(r:Roomtypes)
WHERE r.description CONTAINS 'touros' and a.activity='teatro'
RETURN f.facilityID,f.name, r.description, a.activity

```

### Resultados

f.facilityID	f.name	r.description	a.activity
957	"COLISEU DE REDONDO - EX PRAÇA DE TOIROS"	"Praça de touros multiusos"	"teatro"
916	"COLISEU JOSÉ RONDÓ DE ALMEIDA-EX PRAÇA DE TOIROS"	"Praça de touros multiusos"	"teatro"
940	"ARENA DE VORA - EX PRAÇA DE TOIROS"	"Praça de touros multiusos"	"teatro"

## Pergunta b

*How many facilities with 'touros' in the room type description are there in each region?*

## Mongo

### Interrogação

```
db.municipalities.aggregate([
  { $unwind: "$FACILITIES" },
  { $match:
    {
      "FACILITIES.ROOMTYPE.DESCRPTION": /touros/,
    }
  },
  { $group : { _id : "$REGION.DESIGNATION", count : { $sum : 1 } } }
])
```

### Resultados

```
/* 1 */
{
  "_id" : "Lisboa",
  "count" : 6.0
}

/* 2 */
{
  "_id" : "Alentejo",
  "count" : 43.0
}

/* 3 */
{
  "_id" : "Norte",
  "count" : 3.0
}

/* 4 */
{
```

```

    "_id" : "Centro",
    "count" : 11.0
  }

/* 5 */
{
  "_id" : "Algarve",
  "count" : 1.0
}

```

## Neo4j

### Interrogação

```

MATCH
  (reg:Regions)-[:REGION]-(m:Municipalities)-[:MUNICIPALITY]-(f:Facilities)-[:ROOMTYPE]-(r:Roomtypes)
WHERE r.description CONTAINS 'touros'
RETURN reg.description, COUNT(f) AS nr
ORDER BY nr DESC

```

### Resultados

reg.description	nr
"Alentejo"	43
"Centro"	11
"Lisboa"	6
"Norte"	3
"Algarve"	1

## Pergunta c

*How many municipalities do not have any facility with an activity of 'cinema'?*

# Mongo

## Interrogação

```
db.municipalities.aggregate([
  { $match:
    { "FACILITIES.ACTIVITIES.ACTIVITY": { $ne: "cinema" } }
  },
  { $count : "count" }
])
```

## Resultados

```
/* 1 */
{
  "count" : 100
}
```

# Neo4j

## Interrogação

```
MATCH (mun:Municipalities) with count(mun) as totalMun
MATCH
(a:Activities)<-[:ACTIVITY_TYPE]-(f:Facilities)-[:MUNICIPALITY]->(m:Municipalities)
WHERE a.activity="cinema"
WITH DISTINCT m, totalMun
RETURN totalMun - count(m) as nrMunicipalities
```

## Resultados

nrMunicipalities
100

## Pergunta d

*Which is the municipality with more facilities engaged in each of the six kinds of activities?  
Show the activity, the municipality name and the corresponding number of facilities.*

## Mongo

### Interrogação

```
db.municipalities.aggregate([
  { $unwind: "$FACILITIES" },
  { $unwind: "$FACILITIES.ACTIVITIES" },
  { $group: { _id: { "municipality": "$DESIGNATION", "activity":
"$FACILITIES.ACTIVITIES.ACTIVITY" },
    count: { $sum: 1 } } },
  { $sort: { "count": -1 } },
  { $group: { _id: "$_id.activity", "municipality": { "$first":
"$_id.municipality" },
    count: { $max: "$count" } } },
  ]
})
```

## Resultados

```
/* 1 */
{
  "_id" : "circo",
  "municipality" : "Lisboa",
  "count" : 2.0
}
```

```
/* 2 */
{
  "_id" : "cinema",
  "municipality" : "Lisboa",
  "count" : 96.0
}

/* 3 */
{
  "_id" : "música",
  "municipality" : "Lisboa",
  "count" : 77.0
}

/* 4 */
{
  "_id" : "taurismaquia",
  "municipality" : "Moura",
  "count" : 4.0
}

/* 5 */
{
  "_id" : "teatro",
  "municipality" : "Lisboa",
  "count" : 66.0
}

/* 6 */
{
  "_id" : "dança",
  "municipality" : "Lisboa",
  "count" : 47.0
}
```

## Neo4j

### Interrogação

```
MATCH
(a:Activities)-[:ACTIVITY_TYPE]-(f:Facilities)-[:MUNICIPALITY]->(m:Municipalities)
with a,m,count(f) as nrFacilities
with a, collect(m) as mun, collect(nrFacilities) as counts
with a, mun, counts, reduce(x=[0,0], idx in range(0,size(counts)-1) | case when
counts[idx] > x[1] then [idx,counts[idx]] else x end)[0] as index
return a.activity AS Activity, mun[index].designation AS Municipality,
counts[index] AS Nr ORDER BY a.activity
```

### Resultados

Activity	Municipality	Nr
"cinema"	"Lisboa"	96
"circo"	"Lisboa"	2
"dança"	"Lisboa"	47
"música"	"Lisboa"	77
"tauromaquia"	"Moura"	4
"teatro"	"Lisboa"	66

## Pergunta e

*Which are the codes and designations of the districts with facilities in all the municipalities?*

## Mongo

### Interrogação

```
db.municipalities.aggregate([
```

```

    {$group:{
      _id: {"_id":"$DISTRICT.COD", designation:"$DISTRICT.DESIGNATION"},
      municipalities: { $push: { hasFacilities: {$gt: [{ $size: "$FACILITIES"
    },0]}, nome: "$designation"}}
    },},
    {$match:{
      "municipalities" : {"$not":{"$elemMatch":{"hasFacilities":false}}}
    }},
    {$project: {_id:0, "Code": "$_id._id", "Designation": "$_id.designation"}},
  ]
)

```

## Resultados

```

/* 1 */
{
  "Code" : NumberLong(15),
  "Designation" : "Setúbal"
}

/* 2 */
{
  "Code" : NumberLong(12),
  "Designation" : "Portalegre"
}

/* 3 */
{
  "Code" : NumberLong(7),
  "Designation" : "Évora"
}

/* 4 */
{
  "Code" : NumberLong(11),
  "Designation" : "Lisboa"
}


```

## Neo4j

### Interrogação

```
MATCH (m:Municipalities)
WHERE NOT ()-[:MUNICIPALITY]->(m)
WITH collect(m) as muns
MATCH (m1:Municipalities)-[:DISTRICT]->(d:Districts)
WHERE ALL(x in muns WHERE NOT (x)--(d)) with distinct d
RETURN d.districtCOD, d.designation
```

### Resultados

d.districtCOD	d.designation
7	"  vora"
11	"Lisboa"
12	"Portalegre"
15	"Set <del>ú</del> bal"

## Pergunta f

*Ask the database a query you think is interesting.*

*Municipalities que estão no centro do continente, que tem mais de 7 facilities.*

## Mongo

### Interrogação

```
db.municipalities.aggregate([
  { $match:
    { "REGION.NUT1": { $eq: "Continente" },
      "REGION.DESIGNATION": { $eq: "Centro" },
```

```
    }  
  },  
  {$project: { _id: 0, "DESIGNATION": 1, count: { $size:"$FACILITIES" }}}},  
  {$match:{count:{$gt:7}}},  
  {$sort: {"count": -1}},  
  ]  
)  
)
```

## Resultados

```
/* 1 */  
{  
  "DESIGNATION" : "Coimbra",  
  "count" : 24  
}  
  
/* 2 */  
{  
  "DESIGNATION" : "Aveiro",  
  "count" : 20  
}  
  
/* 3 */  
{  
  "DESIGNATION" : "Leiria",  
  "count" : 19  
}  
  
/* 4 */  
{  
  "DESIGNATION" : "Viseu",  
  "count" : 14  
}  
  
/* 5 */  
{  
  "DESIGNATION" : "Guarda",  
  "count" : 13  
}  
  
/* 6 */  
{  
  "DESIGNATION" : "Figueira da Foz",
```

```
"count" : 12
}

/* 7 */
{
  "DESIGNATION" : "Caldas da Rainha",
  "count" : 10
}
```

## Neo4j

### Interrogação

```
MATCH (f:Facilities)-[:MUNICIPALITY]->(m:Municipalities)-[:REGION]->(r:Regions)
with r,m,count(f) as nrFacilities
WHERE r.nut1="Continente" and r.description="Centro" and nrFacilities > 7
return m.designation AS Municipality, nrFacilities
```

## Resultados

Municipality	nrFacilities
"Guarda"	13
"Aveiro"	20
"Figueira da Foz"	12
"Coimbra"	24
"Leiria"	19
"Caldas da Rainha"	10
"Viseu"	14

# Comparação *Mongo*, *Neo4j* e *Oracle*

## Tempos de execução

As médias dos tempos das primeiras 5 execuções de cada interrogação para os dois modelos não relacionais podem ser observadas na seguinte tabela:

Query	Tempo Mongo (s)	Tempo Neo4j (s)
<b>a</b>	0.043	0.018
<b>b</b>	0.029	0.026
<b>c</b>	0.023	0.037
<b>d</b>	0.033	0.110
<b>e</b>	0.029	0.060
<b>f</b>	0.024	0.012

Através da tabela apresentada é possível concluir que no geral as *queries* em *Mongo* são mais rápidas do que as implementadas em *Neo4j*. Esta diferença é mais acentuada em *queries* mais complexas, como é o caso em **d** e **e**. É de notar que após a primeira execução de uma *query* em *Neo4j* o tempo de execução tende a diminuir significativamente nas execuções seguintes da mesma *query*. Isto deve-se ao facto de quando em *cold boot* o servidor ainda não ter nada armazenado em *cache* pelo que necessita de ir ao disco para aceder aos registos. Caso o tempo da primeira execução não fosse contabilizado, provavelmente, na maioria das *queries* o *Neo4j* apresentaria melhores resultados que o *Mongo*, no entanto, tendo em conta que a *query* tem de ser executada uma primeira vez, o grupo não achou pertinente remover este tempo da comparação.

Embora o grupo não tenha elaborado as *queries* propostas para o modelo *SQL*, tendo em conta os resultados obtidos no primeiro trabalho da cadeira, em que *queries* de semelhante complexidade foram feitas a um modelo relativamente semelhante ao em questão, é possível deduzir que o tempo de execução do modelo em *Mongo* seja inferior ao do tempo de execução das mesmas *queries* num modelo *SQL*. A expectativa seria de que, para o tamanho do *dataset*, esta diferença fosse relativamente insignificante em *queries* de maior simplicidade, acentuando-se em *queries* mais complexas.

## Tamanho da informação

Para obter o tamanho da coleção *municipalities* implementada em *Mongo* recorreu-se ao seguinte comando:

```
db.municipalities.totalSize()
```

O resultado obtido foi de **688080 B**.

Por outro lado, foi calculado um valor aproximado do tamanho dos dados no *Neo4j* com base na seguinte informação disponibilizada em [neo4j](#):

Store File	Record size	Contents
neostore.nodestore.db	15 B	Nodes
neostore.relationshipstore.db	34 B	Relationships
neostore.propertystore.db	41 B	Properties for nodes and relationships
neostore.propertystore.db.strings	128 B	Values of string properties
neostore.propertystore.db.arrays	128 B	Values of array properties
Indexed Property	$1/3 * AVG(X)$	Each index entry is approximately 1/3 of the average property value size

- **Nodes:**
  - *Activities*: 6
  - *Districts*: 20
  - *Facilities*: 1084
  - *Municipalities*: 308
  - *Regions*: 7
  - *Roomtypes*: 18
  - **Total**:  $(6 + 20 + 1084 + 308 + 7 + 18) * 15 \text{ B} = \mathbf{21645 \text{ B}}$
- **Properties:**
  - *Activities*: 1
  - *Facilities*: 5
  - *Roomtypes*: 1
  - *Municipalities*: 3
  - *Districts*: 2
  - *Regions*: 2
  - **Total**:  $(1*6 + 5*20 + 1*1084 + 3*308 + 2*7 + 2*18) * 41 \text{ B} = \mathbf{88724 \text{ B}}$

- **Relationships:**
  - *ACTIVITY\_TYPE*: 2059
  - *ROOMTYPE*: 1084
  - *MUNICIPALITY*: 1084
  - *DISTRICT*: 308
  - *REGION*: 322
  - **Total:**  $(2059 + 1084 + 1084 + 308 + 322) * 34 \text{ B} = 165138 \text{ B}$
- **Indexes:**
  - Activities: 1
  - Facilities: 1
  - Roomtypes: 1
  - Municipalities: 1
  - Districts: 1
  - Regions: 1
  - **Total:**  $(6 + 20 + 1084 + 308 + 7 + 18) * 128 \text{B} * 1/3 = 61568 \text{ B}$
- **Total:**  $21645 \text{ B} + 88724 \text{ B} + 165138 \text{ B} + 61568 \text{ B} = 337075 \text{ B}$

Por fim, para determinar o tamanho que cada uma das tabelas ocupa em disco, foi executada a seguinte *query SQL*:

```
select owner, table_name, round((num_rows*avg_row_len)) B
from all_tables
where num_rows > 0
and owner = 'GTD8';
```

Obtendo os seguintes valores:

Tabela	Tamanho (B)
<i>ACTIVITIES</i>	132
<i>DISTRICTS</i>	260
<i>FACILITIES</i>	87804
<i>MUNICIPALITIES</i>	10472
<i>REGIONS</i>	147
<i>ROOMTYPES</i>	378
<i>USES</i>	12354

Perfazendo um valor total de **111547 B**.

Deste modo, segundo os cálculo efetuados, como **111547 B < 337075 B < 688080 B**, é possível concluir que os dados em *SQL* são os que ocupam menos espaço, seguidos dos

dados em *Neo4j* e por fim, dos em *Mongo* que revelam um valor significativamente superior aos do seu modelo relacional análogo. Isto pode dever-se à existência de alguma redundância no modelo desenvolvido, embora tenha sido implementado de modo a ter o mínimo de redundância possível. O mesmo não se verifica no modelo em *Neo4j* nem em *SQL*.

## Facilidade da Query

Em relação a queries mais simples, a abordagem *SQL* seria mais fácil, uma vez que a estrutura ***SELECT FROM WHERE*** revela-se bastante intuitiva. No entanto, para queries mais complexas que resultem numa necessidade de recorrer a *subqueries*, tabelas auxiliares ou *views*, este modelo torna-se menos intuitivo e mais complexo.

Nos modelos não relacionais implementados a necessidade do uso de operações de *join* para obter informação é removida o que diminui, de uma forma geral, a complexidade das queries. Embora o modelo *Mongo* implementado seja relativamente simples, uma vez que apenas faz uso de uma coleção, o grupo notou que em queries mais complexas, como ***d*** e ***e***, a obtenção da informação não é necessariamente intuitiva ou simples, tornando-se mais complicado, não só de escrever a query, como também interpretá-la *a posteriori*. O mesmo não aconteceu para as queries implementadas em *Neo4j*. A maior dificuldade ocorreu na query ***d***, no entanto isto deveu-se mais a uma falta de conhecimento sobre o paradigma do que à complexidade da query em si, sendo que após alguma pesquisa esta dificuldade foi ultrapassada.

Para os modelos implementados, o modelo em *Neo4j* revelou-se mais simples e intuitivo na escrita das queries propostas. É de notar que o *Neo4j* não permite a criação de ligações bidirecionais, o que, se possível, poderia facilitar a implementação de algumas das queries (fazer duas ligações iria levar a um aumento do espaço ocupado).

## Comentários Adicionais

O *Mongo* é um banco de dados orientado a documentos e o *Neo4j* é baseado em gráficos, pelo que ambos apresentam, por conseguinte, esquemas flexíveis e facilmente adaptáveis aos requerimentos do utilizador (o que em geral se aplica aos modelos *NoSQL*). Esta flexibilidade constitui uma das grandes vantagens dos modelos não relacionais sobre os modelos relacionais, que se revelam bastante restritos neste aspeto. O *SQL*, por exigir o uso de esquemas predefinidos para os dados, que devem seguir sempre a mesma estrutura, implica uma maior dificuldade em lidar com alterações no modelo após a sua implementação. No entanto, é de referir que esta mesma flexibilidade dos modelos *NoSQL*, nomeadamente no modelo *Mongo* implementado, revelou-se como um aspeto não tão positivo na escrita das queries propostas, uma vez que não há uma estrutura bem definida para as interrogações como no modelo *SQL* (o mesmo não se aplica ao *Neo4j* que cujas interrogações, no geral, seguem sempre uma estrutura semelhante).

# Conclusão

Este trabalho permite concluir que os modelos não relacionais, se corretamente implementados, podem ter bastante potencial uma vez que representam soluções distribuídas e flexíveis capazes de lidar com grandes quantidades de informação.

No final, considerou-se que ambos os modelos *NoSQL* definidos são adequados ao problema em questão, sendo que a implementação dos mesmos e das *queries* propostas permitiu aprofundar o conhecimento, adquirido nas aulas, relativamente a estes tipos de representações não relacionais. Como expectável para todas as questões os resultados obtidos foram iguais nos dois modelos.