



Universidade do Minho

Mestrado Integrado em Engenharia Informática
Licenciatura em Ciências da Computação

Unidade Curricular de Bases de Dados

Ano Letivo de 2017/2018

Troca-Turnos da UM

Diana Costa A78985

Marcos Pereira A79116

Sérgio Oliveira A77730

Vítor Castro A77870

Janeiro de 2018

BD

Data de Recepção	
Responsável	
Avaliação	
Observações	

Troca-Turnos da UM

Diana Costa A78985

Marcos Pereira A79116

Sérgio Oliveira A77730

Vítor Castro A77870

Janeiro de 2018

Olhem para nós

*Frente ao computador te escrevo,
Não fosse eu estudante de informática.
Penso no tempo que passou,
E em como ainda ontem, em desespero,
Chorava e pensava em como tudo mudou.*

*Eu era livre e inconsciente,
Vim para a universidade e cresci.
Hoje sou barbudo e mal-humorado,
Fui para casa e dormi.*

*Acordei atrapalhado,
Faltavam horas para entregar o trabalho.
É melhor correr, nem tomar banho,
Que o limite está aí!*

*Sou inteligente e perspicaz,
Um trabalho nunca ficou para trás.
Este está muito bem feito,
A falha aqui jaz.*

*Olho para a frente com confiança,
Olho o futuro com dúvida,
Olho-me com a certeza
De que de tudo sou capaz.*

Resumo

Perante a proposta de dar continuidade ao primeiro projeto, e criar uma segunda base de dados compatível com a primeira, desta vez não relacional, surgiu desde logo um impasse, uma vez que seria necessário elaborar todo um plano para converter os esquemas existentes da BD implementada em SQL para um esquema diferente, NoSQL, baseado em documentos – MongoDB.

Posto isto, numa primeira parte, é feita uma contextualização do problema, apresentando-se, depois, as decisões que levaram o grupo a optar por uma BD NoSQL.

De seguida, todo o esforço e trabalho incidiram em idealizar e planear como seriam armazenados os dados num modelo baseado em documentos. Assim, foi elaborado um pequeno esquema para melhor elucidar a linha de pensamento que a equipa teve em consideração.

Como terceira tarefa, era necessário implementar um programa que fizesse a migração da Base de Dados, isto é, que a partir do modelo lógico do Troca-Turnos, fossem criados documentos e coleções equivalentes que permitissem o seu posterior povoamento, da forma mais eficaz possível.

Depois, converteram-se também as *queries* existentes em SQL para a primeira Base de Dados, para outras compatíveis, em JSON.

Por fim, foi possível, depois de algum trabalho e tempo, uma base de dados bem estruturada, manipulável e passível de ser usada em contexto real. Elaborou-se, posteriormente, uma análise crítica, tendo em conta os dois modelos estudados e as duas Bases de Dados implementadas, focando os pontos positivos e negativos de cada uma.

Área de Aplicação: Desenho e arquitetura de Sistemas de Bases de Dados para controlo de turnos do DI-UM.

Palavras-Chave: Turno, UC, Aluno, Docente, Entidade, Atributo, Relacionamento, Base de Dados, NoSQL, SQL, MongoDB, Modelo Relacional, Modelo não relacional, NoSQL.

Índice

Resumo	iv
Índice	v
Índice de Figuras	vi
1. Introdução	1
1.1. Contextualização	1
1.2. Apresentação do caso em estudo, motivação e objetivos	2
2. MongoDB e paradigma não relacional	3
2.1. Justificação da mudança de paradigma	3
2.2. Porquê MongoDB	4
3. Processo de migração	5
3.1. Diagrama do Modelo Lógico do projeto anterior	5
3.2. Esquema das Coleções	6
3.3. Estrutura dos Documentos	8
3.4. Migração dos dados de MySQL para MongoDB	10
3.5. Migração do esquema de MySQL para MongoDB	11
4. Conclusões e Trabalho Futuro	13
Referências Bibliográficas	14
Lista de Siglas e Acrónimos	15
Anexos	16
I. Anexo 1 – Algumas <i>queries</i> de SQL para MongoDB	17
II. Anexo 2 – Script de migração dos dados	21

Índice de Figuras

Figura 1 - Modelo Lógico da BD relacional	5
Figura 2 - Esquema representativo das coleções Docente e Aluno	6
Figura 3 - Esquema representativo da coleção Alunos_UC	7
Figura 4 - Esquema de um documento para uma aluna	8
Figura 5 - Esquema de um documento para uma docente	9
Figura 6 - Esquema de um documento da coleção Alunos_UC	9
Figura 7 - Query 1 (SQL e correspondente em MongoDB)	17
Figura 10 - Query 4 (SQL e correspondente em MongoDB)	19
Figura 11 - Classe Aluno	21
Figura 12 - Classe Docente	22
Figura 13 - Classe Turno	22
Figura 14 - Classe UC	23
Figura 15 - Classe AlunosUC	23
Figura 16 - Classe SQLReader (1)	24
Figura 17 - Classe SQLReader (2)	25
Figura 18 - Classe SQLReader (3)	26
Figura 19 - Classe SQLReader (4)	27
Figura 20 - Classe SQLReader (5)	28
Figura 21 - Classe SQLReader (6)	29
Figura 22 - Classe SQLReader (7)	30
Figura 23 - Classe SQLReader (8)	30
Figura 24 - Classe Migrator (1)	31
Figura 25 - Classe Migrator (2)	32
Figura 26 - Classe Migrator (3)	32

1. Introdução

1.1. Contextualização

O curso de Engenharia Informática, na Universidade do Minho, recebe anualmente cerca de 150 pessoas. A licenciatura tem a duração de três anos e, todos os anos, há a necessidade de atribuir horários aos alunos inscritos.

Há uns meses, por sugestão do novo Diretor de Curso, e com a colaboração do CESIUM, foi desenvolvida uma aplicação que gere a distribuição dos alunos pelos turnos, de forma a que possam ir a todas as cadeiras em que estão inscritos, e com o objetivo de acabar com o processo de inscrição manual feito pelos docentes. O Departamento de Informática contactou este grupo de trabalho, para que fosse feita uma Base de Dados que suportasse esta aplicação, isto é, que guardasse toda a informação relativa aos docentes, unidades curriculares, turnos e alunos neles inscritos. Na altura, optou-se por um SGBD relacional, dado que oferece uma grande organização e clareza dos dados, integridade dos mesmos e resposta rápida aos pedidos de informação.

Com o passar do tempo, verificou-se que, para além da quantidade de dados aumentar progressivamente, os elementos do grupo, sendo administradores da Base de Dados, não tinham disponibilidade suficiente para a gerir, nem fundos monetários para contratar alguém externo que realizasse este serviço. Assim, depois de uma reunião com todos os responsáveis, chegou-se à conclusão que seria necessário alterar o paradigma, e mudar a BD de SQL para noSQL, utilizando-se MongoDB para responder às necessidades da equipa.

1.2. Apresentação do caso em estudo, motivação e objetivos

A BD desenvolvida pela equipa permite uma gestão dos alunos, nos diversos anos letivos, nos turnos a que estão inscritos nas diversas UCs. Assim, depois da aplicação desenvolvida pelo CESIUM atribuir todos os alunos aos diferentes turnos, o armazenamento é automático para a Base de Dados.

É importante que cada aluno, após efetuar o registo na aplicação, tenha associado o seu número de aluno, nome e curso que frequenta. Após a atribuição dos turnos, a cada aluno vai corresponder uma lista de turnos, e em cada um deles deverá constar o código da UC que o turno se refere, o nome da UC, o turno e o ano letivo atual.

Cada docente terá também o seu registo presente na Base de Dados, em que deverá consignar o seu número, nome, escola de ensino, e uma lista de UCs lecionadas pelo mesmo. Em cada ocorrência da lista deverá constar o código da UC, o nome da mesma, o ano em que a UC é lecionada, e os ECTS.

Haverá também uma outra presença na BD, relativa à UC, que visa o armazenamento do código da mesma, o nome, ano em que é lecionada, ECTS e os alunos inscritos naquela UC. Esta informação poderá parecer redundante, já que já estavam planeadas estas informações estarem associadas a um aluno, mas tal será explicado com detalhe no decorrer do relatório.

Assim sendo, o principal objetivo da implementação da base de dados é a manutenção de dados, para que seja possível ao Departamento de Informática fazer o controlo dos alunos em cada ano, bem como conseguir obter dados rapidamente para fazer estatísticas de qualidade.

Esta preocupação com a qualidade do ensino tem sido crescente, principalmente depois da mudança para o modelo Bolonha, em que se registou uma aparente redução da qualidade do curso. Para além disso, têm surgido várias queixas por parte dos alunos de desadequação das UCs lecionadas às necessidades reais no mundo do trabalho. Ainda mais, a percentagem de repetentes a certas cadeiras tem sido muito elevada e a Direção admitiu que necessita de mais dados estatísticos fiáveis e mais específicos do que os fornecidos pelos relatórios internos da Universidade.

Surge, então, a necessidade de implementar uma Base de Dados forte e flexível, que possibilite o armazenamento de toda a informação crescente relativa aos estudantes, cadeiras e UCs, semestre após semestre. É de extrema importância construir uma BD compreensível, permitindo assim uma estrutura que facilite a consulta e atualização rápida da base de dados. Como este sistema vai ser usado em importantes estatísticas, é mandatório que a solução a implementar seja segura, garantindo a integridade dos dados e, obviamente, confidencialidade de informação.

2. MongoDB e paradigma não relacional

2.1. Justificação da mudança de paradigma

Como já fora supracitado, este projeto consiste em alterar a BD relacional que o grupo desenvolveu inicialmente para outra equivalente em NoSQL, que responda às mesmas *queries* e que facilite o acesso aos dados da maneira mais eficiente possível. Assim, é necessário que sejam revistas as vantagens que uma Base de Dados não relacional traz, em relação aos SGBDR.

Um dos principais pontos que levou a equipa a uma mudança de paradigma foi a quantidade ascendente de dados que, com o passar dos meses, se fez notar. Dado que a liberdade de esquemas promove alta disponibilidade e escalabilidade, a performance da nova BD melhoraria consideravelmente. Foi também alvo de críticas, na reunião com os administradores da BD e com os responsáveis da aplicação, a falta de tempo que os primeiros tinham para administrar a BD. Assim, como um modelo não relacional não exige tanto tempo e esforço de manutenção, este seria mais um ponto a favor para a alteração do paradigma.

Assim sendo, e comparando friamente as duas abordagens, enquanto que nos SGBDR prevalecem as características transacionais ACID (Atomicity, Consistency, Isolation e Durability), o que é bom e visa a que este modelo permaneça no topo, os modelos não relacionais seguem as propriedades BASE (basically available, soft state, eventually consistent). Esta última é uma abordagem mais otimista, que aceita que a consistência da base de dados esteja num “estado de fluxo”. Embora isto pareça impossível de lidar, na realidade é bastante gerenciável, e leva a níveis de escalabilidade que não podem ser obtidos com ACID.

Neste projeto será utilizado, de entre muitos, o SGBD MongoDB, baseado em documentos, em que prevalecem as propriedades C&P do teorema CAP (Consistency, Availability, Partition Tolerance).

2.2. Porquê MongoDB

O MongoDB é, atualmente, uma das BD mais populares, e um dos modelos open-source mais desenvolvidos e explorados no mercado. A estrutura de armazenamento de dados em forma de documentos (JSON) é bastante intuitiva e torna mais suave a típica rigidez dos esquemas relacionais.

O facto de obedecer às propriedades C&P do teorema CAP (Consistency, Availability, Partition Tolerance), torna esta Base de Dados tolerante à partição - ótimo para sistemas distribuídos e alta escalabilidade - e consistente. A redundância dos dados, à partida vista como uma desvantagem, oferece velocidade no acesso a dados e respostas a *queries* mais rápidas. Também a existência de drivers e ODM (object document mapper), para várias linguagens, tornam esta ferramenta atrativa.

3. Processo de migração

3.1. Diagrama do Modelo Lógico do projeto anterior

Para facilitar o processo de migração, antes de mais, o grupo analisou o Modelo Lógico que se segue, realizado para a Base de Dados não relacional do primeiro projeto. Daqui se tentou idealizar como seria organizada a informação, numa abordagem NoSQL orientada a documentos, de forma a que a essência da BD não se perdesse.

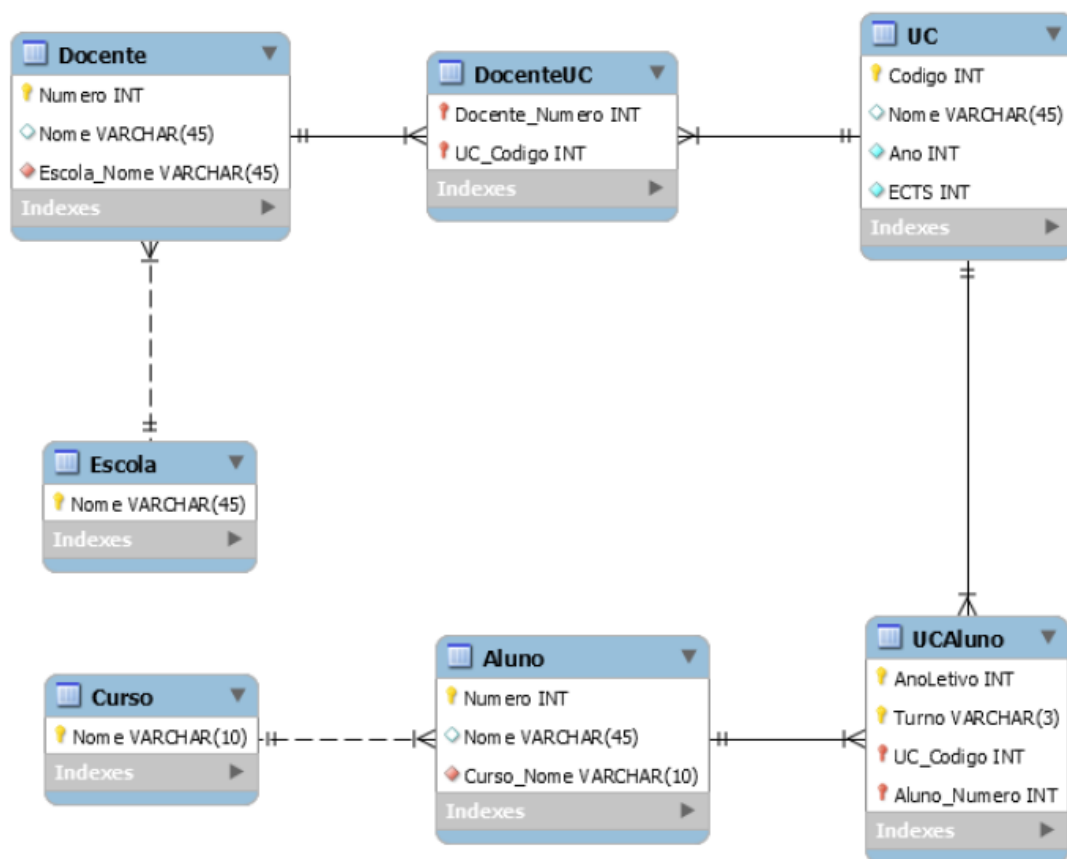


Figura 1 - Modelo Lógico da BD relacional

3.2. Esquema das Coleções

Depois de algum tempo e trabalho, e numa primeira versão, chegou-se ao seguinte esquema representativo das coleções.



Figura 2 - Esquema representativo das coleções Docente e Aluno

No presente esquema, as UCS estão “*embedded*” nas outras duas coleções, em vez de referenciar, para assim permitir um acesso mais rápido aos dados. Uma vez que as UCS são alvo de muito poucas, ou quase nenhuma, modificações, prosseguiu-se assim para a implementação da nova Base de Dados.

Algum tempo depois de avançar com este esquema, o grupo, depois de verificar os objetivos da BD e as *queries* que seriam necessárias pelos requisitos do utilizador, chegou à conclusão que iria precisar de outra coleção, intitulada “Alunos_UC”. Deste modo, embora a informação esteja redundante, a velocidade de acesso aos dados aumenta consideravelmente, e, conseqüentemente, a resposta às *queries* torna-se mais eficiente.

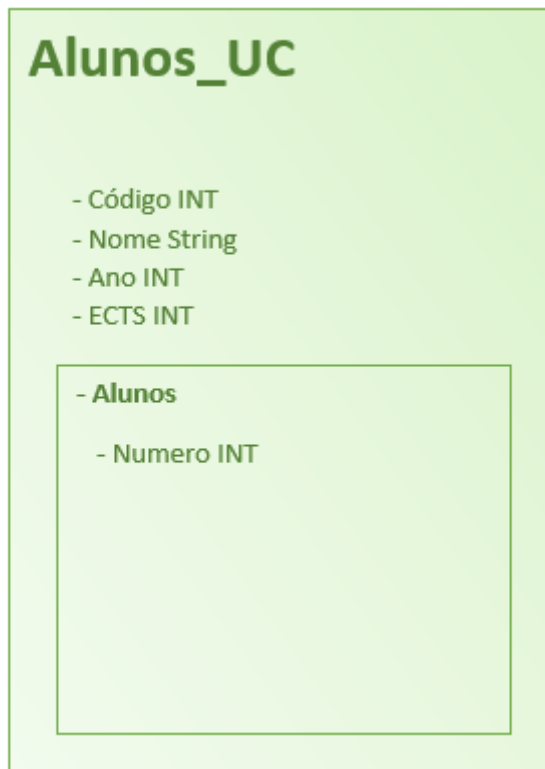


Figura 3 - Esquema representativo da coleção Alunos_UC

3.3. Estrutura dos Documentos

Seguem-se três exemplos representativos do que será o esquema dos documentos da nova Base de Dados, já com os dados referentes a uma aluna, docente e também da coleção Alunos_UC.

```
{
  "Numero": 78415,
  "Nome": "Sara Sampaio",
  "Curso": "MIEI",
  "Turnos": [
    {
      "UC_Codigo": 365,
      "UC_Nome": "Álgebra Linear",
      "Turno": "PL2",
      "AnoLetivo": 1617
    },
    {
      "UC_Codigo": 423,
      "UC_Nome": "Cálculo",
      "Turno": "PL2",
      "AnoLetivo": 1617
    },
    {
      "UC_Codigo": 122,
      "UC_Nome": "Algoritmos e Complexidade",
      "Turno": "PL1",
      "AnoLetivo": 1617
    },
    {
      "UC_Codigo": 050,
      "UC_Nome": "Programação Orientada aos Objetos",
      "Turno": "PL3",
      "AnoLetivo": 1617
    }
  ]
}
```

Figura 4 - Esquema de um documento para uma aluna

```

{
  "Numero": 64850,
  "Nome": "Dulce Oliveira",
  "Escola": "Escola de Engenharia",
  "UCS": [
    {
      "Codigo": 323,
      "Nome": "Programação Funcional",
      "Ano": 1,
      "ECTS": 5
    },
    {
      "Codigo": 604,
      "Nome": "Computação Gráfica",
      "Ano": 3,
      "ECTS": 5
    },
    {
      "Codigo": 455,
      "Nome": "Programação Imperativa",
      "Ano": 1,
      "ECTS": 5
    }
  ]
}

```

Figura 5 - Esquema de um documento para uma docente

```

{
  "Codigo" : 112,
  "Nome" : "Análise",
  "Ano" : 1,
  "ECTS" : 5,
  "Alunos" : [ 78790, 81230, 70608, 80828, 69634, 72111 ]
}

```

Figura 6 - Esquema de um documento da coleção Alunos_UC

3.4. Migração dos dados de MySQL para MongoDB

Para o processo de migração dos dados da Base de Dados relacional para uma não relacional foram consideradas várias opções, sendo que se verificou que o desenvolvimento de uma aplicação que fizesse isso tornaria o processo muito mais flexível, simples e intuitivo. Assim, codificou-se em Java um pequeno programa que, fundamentalmente, acede à base de dados de MySQL e obtém o seu conteúdo. De seguida, transfere a informação para ficheiros em formato JSON que são posteriormente importados para coleções em MongoDB.

Para a ligação ao SGBD relacional, usaram-se os drivers que permitem a conexão a partir de Java e, deste modo, tendo em conta o esquema delineado para as duas bases de dados, criaram-se classes que pudessem resultar em documentos no MongoDB numa estrutura tal que tornasse possível responder com sucesso às *queries* que eram sucedidamente respondidas em MySQL.

Portanto, a recolha dos dados de MySQL consistiu em 3 operações responsáveis por obter os dados necessários para formar os documentos, sendo que cada operação corresponde a uma coleção em Mongo e, por isso, obtém-se as coleções Aluno, Docente e UC.

Usámos, por fim, a biblioteca Jackson que trata de exportar objetos em Java para ficheiros em formato JSON. Foram obtidos, assim, três ficheiros JSON relativos a três coleções e, uma vez que, este formato é aceite pelo MongoDBImport, todo o processo de importação dos documentos na base de dados não relacional foi feito de forma quase imediata e a base de dados foi criada com sucesso.

3.5. Migração do esquema de MySQL para MongoDB

No Modelo Lógico da primeira fase, é possível encontrar três tabelas principais: Docente, UC e Aluno. Destas e dos seus relacionamentos, surgem as relações DocenteUC e UCAluno, tal como se pode verificar na Figura 1. As relações Escola e Curso servem apenas o propósito da normalização.

Sendo agora necessário gerar um modelo para a utilização de MongoDB, num paradigma não-relacional, é importante desde logo realizar que os princípios de construção variam com o paradigma.

Depois de algum confronto inicial, sobre o dever de representar as tabelas intermédias DocenteUC e UCAluno, e de alguma pesquisa, decidiu-se que estas deviam ser fundidas nas coleções Docente, UC e Aluno.

De facto, a decisão de não manter a estrutura original foi a mais sensata, uma vez que o objetivo é fazer uma migração inteligente e funcional, respeitadora do paradigma, e não uma simples migração 1:1, em que a única coisa que muda é o formato de armazenamento de dados, degradando qualquer vantagem ou camuflando qualquer desvantagem da utilização de MongoDB.

Durante o planeamento deste modelo, surgiu a hipótese de criar uma coleção UC, em que todas as UCs seriam armazenadas, sendo que teriam embebidos os seus turnos. Depois de confrontar esta implementação com as *queries* exigidas, verificou-se que, por exemplo, como cada aluno está inscrito em um ou mais turnos de uma UC, então os turnos não poderiam estar embebidos nas UCs, de forma a que o (eventual) documento UC passaria a representar e repetir todas as informações sobre determinada UC, com exceção de um único campo "Turno". Posto isto, decidiu-se refletir sobre quais os dados mais importantes a representar, quer para docentes, quer para alunos. Identificou-se, para os docentes, face às *queries* impostas, a necessidade de guardar o Código e Nome, bem como o Ano em que são lecionadas e o seu valor, em ECTS. Relativamente aos alunos, seria importante guardar o Código da UC e o respetivo Nome, bem como o Turno em que estão inscritos nessas UCs e em que Ano Letivo. Desta forma, não se guardava toda a informação (nem se repetia, como consequência), mas sim apenas aquela necessária.

Por fim, numa fase final de confronto do modelo com as *queries* exigidas, foi necessário criar a coleção AlunosUC, apesar de todos os esforços feitos para repetir o menos possível os dados. Esta coleção foi criada com o objetivo de ser possível conhecer o número de alunos inscritos em cada UC e, por isso, numa tentativa de reduzir ao máximo o volume de dados utilizado, cada documento teria embebido apenas as informações relativas àquela UC, bem como uma lista com os identificadores de cada aluno inscrito.

Assim, viu-se finalizado o processo de planeamento e construção, verificado pela resposta a todas as *queries*, que inclui as coleções: Docente, Aluno, AlunosUC

4. Conclusões e Trabalho Futuro

Findo este trabalho, conclui-se que foi desafiante e bastante instrutivo. Por um lado, a mudança do paradigma relacional para não relacional originou várias dúvidas, mas permitiu que o grupo pudesse lidar com diferentes abordagens e crescesse intelectualmente.

Assim sendo, inicialmente pensou-se que a migração da Base de Dados de MySQL para MongoDB seria tarefa fácil, dada a natureza simplista da segunda BD, uma vez que a falta de esquemas rígidos e a introdução de redundância de dados permite uma liberdade que não foi concedida em SQL. Na verdade, esta tarefa foi mais complicada que o esperado: foi necessário desenvolver todo um programa que fizesse a migração dos dados, que envolveu bastante pesquisa e esforço por parte do grupo. Para além disso, a “falta de esquemas rígidos” e a falta de regras de normalização e integridade, pôs em questão demasiadas opções para representar os novos esquemas dos documentos, pelo que a equipa teve que optar pela alternativa que melhorasse as respostas às *queries* pretendidas e o acesso aos dados lidos com mais frequência pelos utilizadores

De seguida, estando a nova Base de Dados operacional, o grupo achou por bem converter algumas das interrogações elaboradas para o primeiro projeto, em SQL, para JSON. Esta tarefa revelou-se bastante interessante e mostrou as grandes vantagens que se podem tirar de um modelo não relacional, como a simplicidade de acesso e a velocidade de inserção de dados.

É de salientar que, num contexto real, o projeto seria mais desafiante, e iria expor muito mais as vantagens e desvantagens deste novo paradigma. No entanto, o grupo sente que conseguiu tirar partido de NoSQL e Bases de Dados orientadas a documentos, e que pretende usufruir desta “ferramenta” no futuro, sabendo o que a mesma implica a nível de desempenho e organização.

Com toda a certeza, este foi um projeto trabalhoso e interessante, não tanto como o primeiro, mas que permitiu explorar uma parte da informática que o grupo não tinha ainda experienciado.

Referências Bibliográficas

[1] Thomas Connolly, Carolyn Begg 2005. Database Systems: A Practical Approach to Design, Implementation, and Management, 4ª edição, Adisson Wesley.

Lista de Siglas e Acrónimos

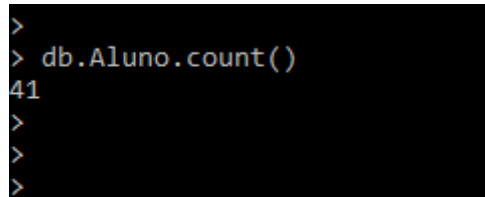
BD	Base de Dados
SBD	Sistema de Base de Dados
SGBD	Sistema de Gestão de Base de Dados
ER	Entidade-Relacionamento
SQL	Structured Query Language
NoSQL	Not Only SQL

Anexos

I. Anexo 1 – Algumas *queries* de SQL para MongoDB

```
/*  
O diretor de curso deve conseguir saber o total de Alunos  
que o curso tem atualmente.  
*/  
SELECT COUNT(Numero) AS numero_alunos FROM Aluno;  
  
-- db.Aluno.count()
```

Figura 7 - Query 1 (SQL e correspondente em MongoDB)



```
>  
> db.Aluno.count()  
41  
>  
>  
>
```

```

/*
  O diretor de curso deve ser capaz de obter a listagem de todos os
  alunos do curso.
*/
SELECT Aluno.Numero, Aluno.Nome FROM Aluno;

-- db.Aluno.find({}, {Numero: 1, Nome: 1}).pretty()

```

```

> db.Aluno.find({}, {Numero:1, Nome:1}).pretty()
{
  "_id" : ObjectId("5a5f628456d3a00a25d6a44a"),
  "Numero" : 51146,
  "Nome" : "Andr  Azevedo"
}
{
  "_id" : ObjectId("5a5f628456d3a00a25d6a44b"),
  "Numero" : 68892,
  "Nome" : "Gil Sampaio"
}
{
  "_id" : ObjectId("5a5f628456d3a00a25d6a44c"),
  "Numero" : 68503,
  "Nome" : "Duarte Machado"
}
{
  "_id" : ObjectId("5a5f628456d3a00a25d6a44d"),
  "Numero" : 69545,
  "Nome" : "Gon alo Oliveira"
}
{
  "_id" : ObjectId("5a5f628456d3a00a25d6a44e"),
  "Numero" : 69308,
  "Nome" : "Alexandre Marques"
}
{
  "_id" : ObjectId("5a5f628456d3a00a25d6a44f"),
  "Numero" : 69634,
  "Nome" : "Daniela Ribeiro"
}
{
  "_id" : ObjectId("5a5f628456d3a00a25d6a450"),
  "Numero" : 70608,
  "Nome" : "Beatriz Sousa"
}
{
  "_id" : ObjectId("5a5f628456d3a00a25d6a451"),
  "Numero" : 66906,
  "Nome" : "Hugo Gramacho"
}

```

(...)


```

/*
O diretor de curso deve conseguir saber o total de UC's que
estão a ser lecionadas no curso.
*/
SELECT COUNT(Codigo) AS numero_ucs FROM UC;

-- db.Docente.distinct("UCS").length
-- OU
-- db.UC.count()

```

```

>
> db.UC.count()
27
>
>

```

```

/*
O diretor de curso deve ser capaz de aceder à lista
de alunos e respectivas cadeiras.
*/
SELECT Aluno.Numero, Aluno.Nome, UC.Nome FROM Aluno
INNER JOIN UCaluno ON UCaluno.Aluno_Numero = Aluno.Numero
INNER JOIN UC ON UC.Codigo = UCaluno.UC_Codigo;

-- db.Aluno.find({Curso:"MIEI"},{Numero:1, Nome:1, Turnos:1}).pretty()

```

Figura 8 - Query 4 (SQL e correspondente em MongoDB)

```

> db.Aluno.find({Curso: "MIEI"},{Numero:1, Nome:1, Turnos:1}).pretty()
{
  "_id" : ObjectId("5a5f628456d3a00a25d6a44a"),
  "Numero" : 51146,
  "Nome" : "Andr  Azevedo",
  "Turnos" : [
    {
      "UC_Codigo" : 312,
      "UC_Nome" : "Estat stica Aplicada",
      "Turno" : "PL2",
      "AnoLetivo" : 1617
    },
    {
      "UC_Codigo" : 455,
      "UC_Nome" : "Programac o Imperativa",
      "Turno" : "PL2",
      "AnoLetivo" : 1617
    }
  ]
}

{
  "_id" : ObjectId("5a5f628456d3a00a25d6a44b"),
  "Numero" : 68892,
  "Nome" : "Gil Sampaio",
  "Turnos" : [
    {
      "UC_Codigo" : 10,
      "UC_Nome" : "Laborat rios de Inform tica IV",
      "Turno" : "PL3",
      "AnoLetivo" : 1617
    },
    {
      "UC_Codigo" : 667,
      "UC_Nome" : "Bases de Dados",
      "Turno" : "PL4",
      "AnoLetivo" : 1617
    },
    {
      "UC_Codigo" : 702,
      "UC_Nome" : "Comunica es por Computador",
      "Turno" : "PL4",
      "AnoLetivo" : 1617
    }
  ]
}

```

(...)

II. Anexo 2 – Script de migração dos dados

```
package migrator;

import java.util.ArrayList;

public class Aluno {

    private int Numero;
    private String Nome;
    private String Curso;
    private ArrayList<Turno> Turnos;

    public Aluno(int numero, String nome, String curso, ArrayList<Turno> turnos) {

        Numero = numero;
        Nome = nome;
        Curso = curso;
        Turnos = turnos;
    }

}
```

Figura 9 - Classe Aluno

```

package migrator;

import java.util.ArrayList;

public class Docente {

    private int Numero;
    private String Nome;
    private String Escola;
    private ArrayList<UC> UCS;

    public Docente(int numero, String nome, String escola, ArrayList<UC> ucs) {

        Numero = numero;
        Nome = nome;
        Escola = escola;
        UCS = ucs;
    }

}

```

Figura 10 - Classe Docente

```

package migrator;

public class Turno {

    private int UC_Codigo;
    private String UC_Nome;
    private String Turno;
    private int AnoLetivo;

    public Turno(int codigoUC, String nomeUC, String turno, int anoLetivo) {

        UC_Codigo = codigoUC;
        UC_Nome = nomeUC;
        Turno = turno;
        AnoLetivo = anoLetivo;
    }

}

```

Figura 11 - Classe Turno

```

package migrator;

public class UC {

    private int Codigo;
    private String Nome;
    private int Ano;
    private int ECTS;

    public UC(int codigoUC, String nomeUC, int ano, int ects) {

        Codigo = codigoUC;
        Nome = nomeUC;
        Ano = ano;
        ECTS = ects;
    }

}

```

Figura 12 - Classe UC

```

package migrator;

import java.util.ArrayList;

public class AlunosUC {

    private int Codigo;
    private String Nome;
    private int Ano;
    private int ECTS;
    private ArrayList<Integer> Alunos;

    public AlunosUC(int codigoUC, String nomeUC, int ano, int ects, ArrayList<Integer> alunos) {

        Codigo = codigoUC;
        Nome = nomeUC;
        Ano = ano;
        ECTS = ects;
        Alunos = alunos;
    }

}

```

Figura 13 - Classe AlunosUC

```

package migrator;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.logging.Level;
import java.util.logging.Logger;
import jdk.nashorn.internal.runtime.Version;

public class SQLReader {

    private static String URL;
    private static String TABLE;
    private static String USERNAME;
    private static String PASSWORD;
    private static Connection cn;
    //private static final String PASSWORD = "admin1";

    public SQLReader() {

        URL = "localhost";
        TABLE = "TrocaTurnos";
        USERNAME = "admin";
        PASSWORD = "admin";
        cn = null;
    }
}

```

Figura 14 - Classe SQLReader (1)

```

// Iniciar a ligação à base de dados
public static void iniciarCon() {

    try {

        cn = DriverManager.getConnection("jdbc:mysql://" + URL + "/" +
            TABLE + "?user=" + USERNAME + "&password=" + PASSWORD);

    } catch (SQLException e) {

    }

}

// Fechar ligação à base de dados
public static void fecharCon() {

    try {

        cn.close();

    } catch (SQLException e) {

    }

}

public void migrateDocentes(ArrayList<Docente> d) {

    ResultSet docente = null;
    ResultSet uc = null;
    PreparedStatement stDocente = null;
    PreparedStatement stUcs = null;

```

Figura 15 - Classe SQLReader (2)

```

ArrayList<Docente> docentes = d;

try {
    stDocente = cn.prepareStatement("SELECT * FROM Docente;"); //alterar
    docente = stDocente.executeQuery();
    while (docente.next()) {
        int numero = docente.getInt("Numero");
        String nome = docente.getString("Nome");
        String escola = docente.getString("Escola_Nome");
        // Ler as UCs em que aquele docente está inscrito
        ArrayList<UC> ucs = new ArrayList<UC>();
        stUcs = cn.prepareStatement(
            "SELECT * FROM DocenteUC JOIN UC ON UC.Codigo = "
            + "DocenteUC.UC_Codigo WHERE DocenteUC.Docente_Numero"
            + " = " + numero);
        // obtem lista de ucs lecionadas pelo docente com o 'numero'
        uc = stUcs.executeQuery();
        while (uc.next()) {
            int codigo = uc.getInt("Codigo");
            String nomeUC = uc.getString("Nome");
            int ano = uc.getInt("Ano");
            int ects = uc.getInt("ECTS");
            ucs.add(new UC(codigo, nomeUC, ano, ects));
        }
        docentes.add(new Docente(numero, nome, escola, ucs));
    }
} catch (SQLException ex) {
    Logger lgr = Logger.getLogger(Version.class.getName());
    lgr.log(Level.SEVERE, ex.getMessage(), ex);
} finally {

```

Figura 16 - Classe SQLReader (3)


```

        try {
            if (uc != null) {
                uc.close();
            }
            if (docente != null) {
                docente.close();
            }
            if (stUcs != null) {
                stUcs.close();
            }
            if (stDocente != null) {
                stDocente.close();
            }
        } catch (SQLException ex) {
            Logger lgr = Logger.getLogger(Version.class.getName());
            lgr.log(Level.WARNING, ex.getMessage(), ex);
        }
    }

}

public void migrateAlunos(ArrayList<Aluno> a) {

    ResultSet aluno = null;
    ResultSet turno = null;
    PreparedStatement stAluno = null;
    PreparedStatement stTurno = null;
    ArrayList<Aluno> alunos = a;

    try {

```

Figura 17 - Classe SQLReader (4)

```

stAluno = cn.prepareStatement("SELECT * FROM Aluno;"); // alterar
aluno = stAluno.executeQuery();
while (aluno.next()) {
    int numero = aluno.getInt("Numero");
    String nome = aluno.getString("Nome");
    String curso = aluno.getString("Curso_Nome");
    // Ler os turnos em que aquele aluno está inscrito
    ArrayList<Turno> turnos = new ArrayList<Turno>();
    stTurno = cn.prepareStatement("SELECT * FROM UCaluno JOIN UC "
        + "ON UC.Codigo = UCaluno.UC_Codigo WHERE "
        + "UCAluno.Aluno_Numero = " + numero);
    // obtem lista de turnos para o aluno com aquele 'numero'
    turno = stTurno.executeQuery();
    while (turno.next()) {
        int codigo = turno.getInt("Codigo");
        String nomeUC = turno.getString("Nome");
        String turnoUC = turno.getString("Turno");
        int anoLetivo = turno.getInt("AnoLetivo");
        turnos.add(new Turno(codigo, nomeUC, turnoUC, anoLetivo));
    }
    alunos.add(new Aluno(numero, nome, curso, turnos));
}
} catch (SQLException ex) {
    Logger lgr = Logger.getLogger(Version.class
        .getName());
    lgr.log(Level.SEVERE, ex.getMessage(), ex);
} finally {
    try {
        if (turno != null) {
            turno.close();
        }
    }
}

```

Figura 18 - Classe SQLReader (5)

```

    }
    if (aluno != null) {
        aluno.close();
    }
    if (stTurno != null) {
        stTurno.close();
    }
    if (stAluno != null) {
        stAluno.close();
    }
} catch (SQLException ex) {
    Logger lgr = Logger.getLogger(Version.class
        .getName());
    lgr.log(Level.WARNING, ex.getMessage(), ex);
}
}

}

void migrateUCS(ArrayList<AlunosUC> ucs) {

    ResultSet uc = null;
    ResultSet aluno = null;
    PreparedStatement stUc = null;
    PreparedStatement stAluno = null;
    ArrayList<AlunosUC> alunosUC = ucs;

    try {
        stUc = cn.prepareStatement("SELECT * FROM UC;"); // alterar
        uc = stUc.executeQuery();
    }

```

Figura 19 - Classe SQLReader (6)

```

while (uc.next()) {
    int codigo = uc.getInt("Codigo");
    String nome = uc.getString("Nome");
    int ano = uc.getInt("Ano");
    int ects = uc.getInt("ECTS");
    // Ler os alunos que estao inscritos na uc
    ArrayList<Integer> alunos = new ArrayList<Integer>();
    stAluno = cn.prepareStatement("SELECT * FROM UC JOIN UCAluno "
        + "ON UC.Codigo = UCAluno.UC_Codigo WHERE "
        + "UCAluno.UC_Codigo = " + codigo);
    // obtem lista de turnos para o aluno com aquele 'numero'
    aluno = stAluno.executeQuery();
    while (aluno.next()) {
        int numero = aluno.getInt("Aluno_Numero");
        alunos.add(new Integer(numero));
    }
    alunosUC.add(new AlunosUC(codigo, nome, ano, ects, alunos));
}
} catch (SQLException ex) {
    Logger lgr = Logger.getLogger(Version.class
        .getName());
    lgr.log(Level.SEVERE, ex.getMessage(), ex);
} finally {
    try {
        if (aluno != null) {
            aluno.close();
        }
        if (uc != null) {
            uc.close();
        }
    }
}
}

```

Figura 20 - Classe SQLReader (7)

```

        if (stAluno != null) {
            stAluno.close();
        }
        if (stUc != null) {
            stUc.close();
        }
    } catch (SQLException ex) {
        Logger lgr = Logger.getLogger(Version.class
            .getName());
        lgr.log(Level.WARNING, ex.getMessage(), ex);
    }
}
}
}

```

Figura 21 - Classe SQLReader (8)

```

package migrator;

import java.io.FileWriter;
import java.util.ArrayList;

import org.codehaus.jackson.annotate.JsonMethod;
import org.codehaus.jackson.map.*;
import org.codehaus.jackson.annotate.JsonAutoDetect.Visibility;

public class Migrator {

    public static void main(String[] args) throws Exception {

        SQLReader sql = new SQLReader(); // classe que tem os para ler

        ArrayList<Docente> docentes = new ArrayList<>();
        ArrayList<Aluno> alunos = new ArrayList<>();
        ArrayList<AlunosUC> ucs = new ArrayList<>();

        FileWriter docentesFile;
        FileWriter alunosFile;
        FileWriter ucsFile;

        ObjectMapper doc = new ObjectMapper();
        doc.setVisibility(JsonMethod.FIELD, Visibility.ANY);
        doc.enable(SerializationConfig.Feature.INDENT_OUTPUT);
        doc.writerWithDefaultPrettyPrinter();

        ObjectMapper alu = new ObjectMapper();
        alu.setVisibility(JsonMethod.FIELD, Visibility.ANY);
    }
}

```

Figura 22 - Classe Migrator (1)

```

alu.enable(SerializationConfig.Feature.INIDENT_OUTPUT);
alu.writerWithDefaultPrettyPrinter();

ObjectMapper aluucs = new ObjectMapper();
aluucs.setVisibility(JsonMethod.FIELD, Visibility.ANY);
aluucs.enable(SerializationConfig.Feature.INIDENT_OUTPUT);
aluucs.writerWithDefaultPrettyPrinter();

sql.iniciarCon(); // iniciar connection a base de dados

sql.migrateDocentes(docentes);
sql.migrateAlunos(alunos);
sql.migrateUCS(ucs);

docentesFile = new FileWriter("docentes.json");
doc.writeValue(docentesFile, docentes);
docentesFile.close();
System.out.println("Sucessfully made JSON Object to file docentes.");

alunosFile = new FileWriter("alunos.json");
doc.writeValue(alunosFile, alunos);
alunosFile.close();
System.out.println("Sucessfully made JSON Object to file alunos.");

ucsFile = new FileWriter("ucs.json");
aluucs.writeValue(ucsFile, ucs);
ucsFile.close();
System.out.println("Sucessfully made JSON Object to file ucs.");

sql.fecharCon();

System.out.println("Docentes: " + doc);

```

Figura 23 - Classe Migrator (2)

```

System.out.println("Alunos: " + alu);
System.out.println("UCS: " + aluucs);
}
}

```

Figura 24 - Classe Migrator (3)