Universidade do Minho
Mestrado Integrado em Engenharia Informática

## Unidade Curricular de
### Bases de Dados NoSQL

Ano Letivo de 2019/2020

# FlixNet: Transformações em Bases de Dados

**Bruno Martins A80410, Catarina Machado A81047**

**Daniel Tinoco A78828, Eduardo Barbosa A83344**

**Grupo 4**

Janeiro, 2020

| | |
|---|---|
| Data de Recepção | |
| Responsável | |
| Avaliação | |
| Observações | |

# FlixNet: Transformações em Bases de Dados

**Bruno Martins A80410, Catarina Machado A81047**

**Daniel Tinoco A78828, Eduardo Barbosa A83344**

**Grupo 4**

Janeiro, 2020

# Resumo

O presente trabalho prático foi elaborado no âmbito do desenvolvimento dos novos requisitos do software FlixNet e consistiu na análise, planeamento e implementação de um Sistema de Bases de Dados Relacional (Oracle) e dois Não Relacionais (MongoDB e Neo4j), tendo como base o modelo da base de dados Sakila do software FlixNet, em MySQL.

Desta forma, ao longo deste relatório, encontra-se descrito de forma detalhada o trabalho efetuado, começando pela descrição da base de dados Sakila e, em seguida, apresentando os Sistemas de Bases de Dados implementados e as diferentes estratégias adotadas ao longo do seu processo de desenvolvimento. Por fim, apresenta-se descrito o conjunto de queries desenvolvido que permite demonstrar a operacionalidade dos sistemas implementados.

Os principais objetivos foram a familiarização com cada uma das ferramentas de implementação do Sistema de Bases de Dados e a perceção das principais diferenças entre cada uma delas.

**Área de Aplicação:** Desenho, arquitetura, desenvolvimento e implementação de Sistemas de Bases de Dados.

**Palavras-Chave:** Bases de Dados, Bases de Dados Relacionais, Bases de Dados Não Relacionais, SQL, NoSQL, Sakila, MongoDB, Neo4j, Oracle, Migração de Dados, Modelo Lógico.

# Índice

*Resumo* ..... *i*

*Índice* ..... *ii*

*Índice de Figuras* ..... *iv*

*Índice de Excertos de Código* ..... *v*

*1. Introdução* ..... *1*

**1.1. Apresentação do Caso de Estudo** ..... **1**

**1.2. Contextualização da Solução** ..... **2**

**1.3. Estrutura do Relatório** ..... **2**

*2. Sakila* ..... *3*

**2.1. *Customer Data*** ..... **3**

**2.2. *Inventory*** ..... **4**

**2.3. *Business*** ..... **5**

*3. Queries* ..... *6*

*4. Base de Dados Relacional - Oracle* ..... *7*

**4.1. Modelo de Dados** ..... **7**

**4.2. Migração de Dados** ..... **7**

**4.3. *Views*** ..... **7**

4.3.1 *View 1* ..... 8

4.3.2 *View 2* ..... 8

4.3.3 *View 3* ..... 9

*5. Base de Dados Não Relacionais* ..... *10*

*6. Base de Dados Orientada a Documentos - MongoDB* ..... *11*

**6.1. Modelo de Dados** ..... **11**

**6.2. Migração de Dados** ..... **12**

**6.3. *Queries*** ..... **12**

6.3.1 *Query 1* ..... 13
6.3.2 *Query 2* ..... 13
6.3.3 *Query 3* ..... 13
6.3.4 *Query 4* ..... 13
6.3.5 *Query 5* ..... 13

***7. Base de Dados Orientada a Grafos - Neo4j ..... 14***

**7.1. Modelo de Dados ..... 14**

**7.2. Migração de Dados ..... 15**

**7.3. *Queries* ..... 16**

7.3.1 *Query 1* ..... 16
7.3.2 *Query 2* ..... 17
7.3.3 *Query 3* ..... 17
7.3.4 *Query 4* ..... 17
7.3.5 *Query 5* ..... 17

***8. Conclusão ..... 18***

***Lista de Siglas e Acrónimos ..... 19***

***Anexos ..... 20***

***I. Anexo 1: Modelo Representativo MongoDB ..... 21***

# Índice de Figuras

Figura 1 - Tabelas da camada *Customer Data*. 3
Figura 2 - Tabelas da camada *Inventory*. 4
Figura 3 - Tabelas da camada *Business*. 5
Figura 4 - Representação grafos Neo4j. 15

# Índice de Excertos de Código


Excerto 1 - *View* 1 (Oracle). 8

Excerto 2 - *View* 2 (Oracle). 8

Excerto 3 - *View* 3 (Oracle). 9

Excerto 4 - Representação dos dados do SGBD MySQL em Ruby. 10

Excerto 5 - Migração de Dados para MongoDB. 12

Excerto 6 - *Query 1* (MongoDB). 13

Excerto 7 - *Query 2* (MongoDB). 13

Excerto 8 - *Query 3* (MongoDB). 13

Excerto 9 - *Query 4* (MongoDB). 13

Excerto 10 - *Query 5* (MongoDB). 13

Excerto 11 - Migração de Dados para Neo4j (método *collect_models*). 15

Excerto 12 - Migração de Dados para Neo4j (método *create_edges*). 16

Excerto 13 - *Query 1* (Neo4j). 16

Excerto 14 - *Query* 2 (Neo4j). 17

Excerto 15 - *Query 3* (Neo4j). 17

Excerto 16 - *Query 4* (Neo4j). 17

Excerto 17 - *Query 5* (Neo4j). 17

# 1. Introdução

## 1.1. Apresentação do Caso de Estudo

O FlixNet é um software de gestão de alugueres de filmes que disponibiliza várias funcionalidades, servindo também como um sistema de recomendação de filmes com base em personalizações por parte dos clientes.

Este software utiliza o motor de base de dados MySQL que já se encontrava escalado verticalmente ao máximo computacionalmente possível. Devido à crescente popularidade deste software e às perspetivas da continuação de um crescimento exponencial, o FlixNet viu-se obrigado a expandir para uma infraestrutura mais escalável horizontal.

Em consequência disso, foi necessário proceder à exploração de diferentes motores de bases de dados, tanto relacionais, como não relacionais, como forma de perceber a que melhor se adequa aos novos requisitos.

Após várias pesquisas e análises das bases de dados mais utilizadas e das respetivas características, foram escolhidos três diferentes motores de bases de dados, cada um deles com o objetivo de ser utilizado em diferentes partes da aplicação. Desta forma, recorreu-se à base de dados não relacional orientada a documentos MongoDB para consultas, à base de dados não relacional orientada a grafos Neo4j para pesquisas relacionais e, por fim à base de dados relacional Oracle, por ser uma base de dados Enterprise, de modo a obtermos suporte comercial.

Como tal, foi necessário migrar todos os dados existentes na base de dados MySQL, com o nome de Sakila, para os três sistemas de gestão de bases de dados mencionados, adequando o respetivo esquema da Sakila às novas bases de dados.

## 1.2. Contextualização da Solução

A possibilidade da migração de dados entre diferentes paradigmas de bases de dados é uma mais-valia no mercado atual devido à quantidade de diferentes sistemas de bases de dados que existem e às mudanças repentinas dos requisitos do software, tal como aconteceu ao FlixNet.

Para responder aos novos requisitos deste sistema, foi necessário pôr em prática o conceito de migração, tirando partido da sua base de dados, Sakila, e dos respetivos scripts de criação e povoamento.

Desta forma, a partir da base de dados do FlixNet, o objetivo foi efetuar a migração dos dados para um outro Sistema de Base de Dados Relacional, Oracle, e para dois Sistemas de Bases de Dados Não Relacionais, MongoDB e Neo4j.

Assim, numa fase inicial foi crucial conhecer bem estes sistemas e o modelo Sakila, para assim, posteriormente, sermos capazes de desenvolver novos esquemas, equivalentes à base de dados Sakila, mas tendo em consideração as especificidades dos sistemas de bases de dados Oracle, MongoDB e Neoj4J.

Como forma de destacar as diferenças entre cada um destes motores, implementamos um conjunto de queries, que permitem comparar os diferentes modelos e funcionalidades desenvolvidas.

## 1.3. Estrutura do Relatório

O presente relatório é composto por sete capítulos:

No primeiro capítulo, **Introdução**, é feito um enquadramento e contextualização do trabalho prático, sendo feita a descrição do problema.

Em seguida, em **Sakila**, encontra-se uma descrição detalhada da base de dados Sakila.

No capítulo **Queries** são apresentadas as queries que foram desenvolvidas com o objetivo de testar a operacionalidade dos sistemas implementados.

Posteriormente, nos capítulos **Base de Dados Relacional - Oracle**, **Base de Dados Orientada a Documentos - MongoDB** e **Base de Dados Orientada a Grafos - Neo4j**, encontram-se descritas as diferentes estratégias adotadas no desenvolvimento de cada um dos modelos, explicando o raciocínio aplicado na migração dos dados e comparando os diferentes sistemas e a respetiva implementação das queries.

Por fim, na **Conclusão**, termina-se o relatório com uma síntese e análise do trabalho realizado.

# 2. Sakila

A Sakila é a base de dados do software FlixNet, que serve de esquema padrão para gestão de alugueres fictícios, contendo filmes, atores, clientes, alugueres e funcionários. Também poderá ser usada para livros, tutoriais, artigos, entre outros. Um exemplo simples de utilização desta base de dados seria o aluguer e devolução de DVDs e encontrar DVDs em atraso.

De uma forma geral, a base de dados Sakila está organizada em três diferentes camadas: ***Customer Data***, ***Inventory*** e ***Business***, cada uma delas com várias tabelas. A base de dados contém também algumas *Views*, *Stored Procedures*, *Stored Functions* e *Triggers*.

## 2.1. *Customer Data*

A camada *Customer Data* contém quatro diferentes tabelas: ***Customer***, ***Address***, ***Country*** e ***City***. Estas tabelas englobam os dados dos clientes.

A tabela ***customer*** contém a lista de todos os clientes, incluindo informações como o primeiro e último nome do cliente, o seu e-mail, se é um cliente ativo e quando é que essa entrada foi criada. Esta tabela possui também como chave estrangeira informação do *address*, que contém a morada do cliente, e da *store*, que se trata da sua *"home store"*, isto é, a loja onde habitualmente o cliente compra (mas pode comprar noutras também).

A tabela ***address*** contém informações da morada tanto de clientes, como de *staff* e de lojas. Esta tabela por sua vez recorre à tabela ***country*** e à tabela ***city*** para armazenar o país e a cidade, respetivamente.

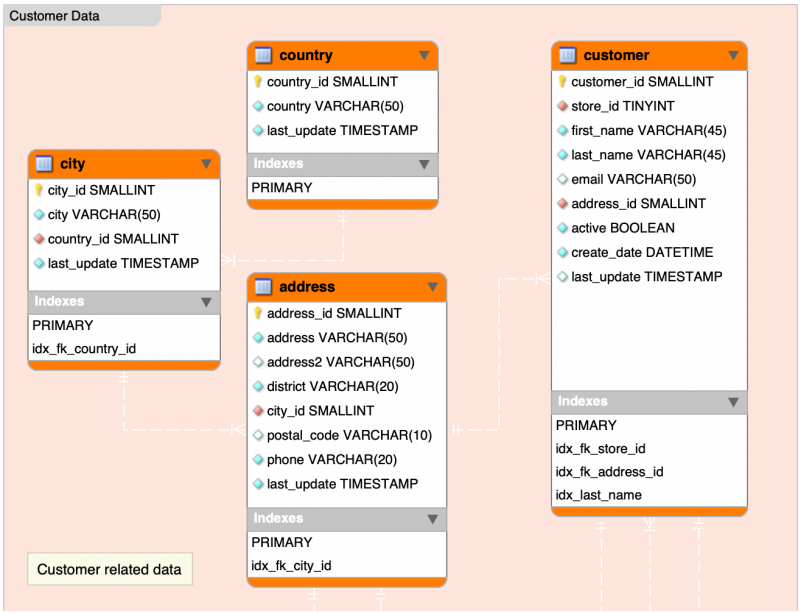


Figura 1 - Tabelas da camada *Customer Data*.

## 2.2. *Inventory*

A camada *Inventory* representa os filmes, sendo constituída por oito diferentes tabelas: ***film***, ***fim_category***, ***language***, ***category***, ***actor***, ***film_actor***, ***film_text*** e ***inventory***.

A tabela ***film*** é uma lista de todos os filmes potencialmente em stock nas lojas. Os filmes que estão realmente em stock encontram-se na tabela ***inventory***. O ***film*** contém informações como o título, a descrição, o ano de lançamento, a classificação, a duração (em minutos), entre outros dados relevantes relativos a um filme. A tabela ***language*** armazena o idioma do filme.

Na tabela ***film_category***, com recurso à tabela ***category***, encontra-se a informação relativa à categoria do filme, e, na tabela ***film_actor*** ficam armazenados os atores que participaram num determinado filme. A tabela ***actor*** guarda o primeiro e último nome de todos os atores existentes.

A tabela ***inventory*** contém uma linha para cada cópia de um determinado filme numa determinada loja.

Por fim, a tabela ***film_text*** indica o título e a descrição do filme. Esses dados são sempre iguais aos contidos na tabela ***film*** devido a *triggers* que já se encontram implementados.

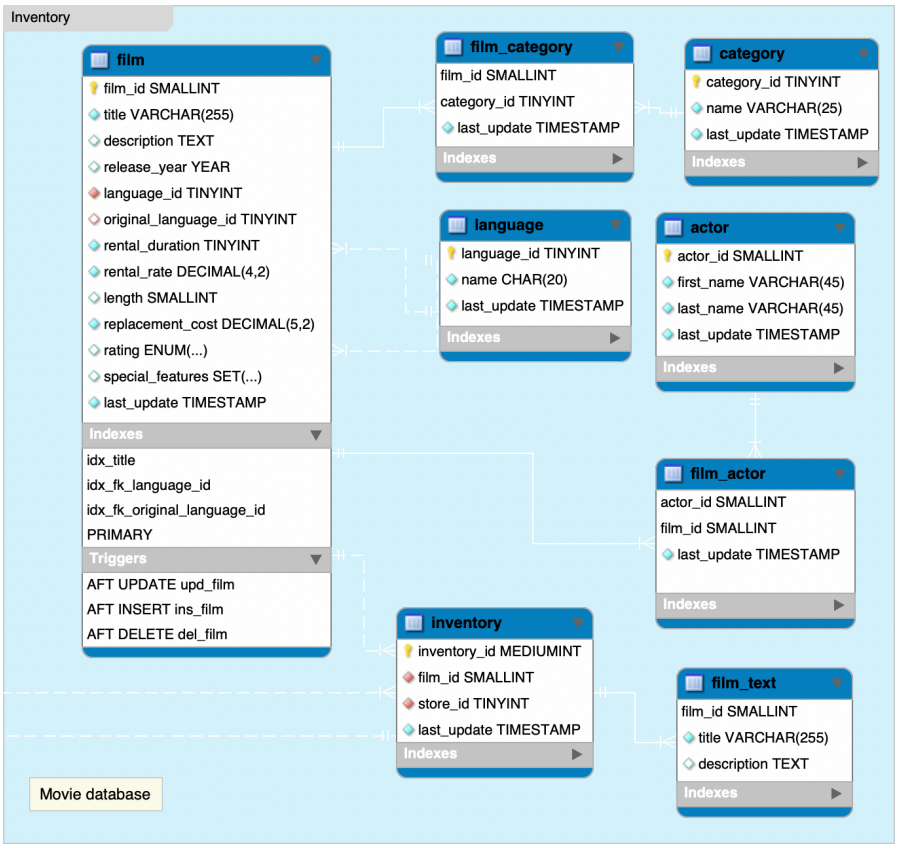


Figura 2 - Tabelas da camada *Inventory*.

## 2.3. *Business*

Por fim, na camada *Business* encontram-se os dados do negócio através das tabelas ***staff***, ***payment***, ***rental*** e ***store***.

A tabela ***staff*** contém a lista de todos os membros do *staff*, incluindo informações como o primeiro e último nome, o e-mail, dados para o *login* e a fotografia do funcionário. Contém também a informação da loja onde o funcionário trabalha, com recurso à tabela *store*, e a sua morada, com recurso à tabela *address*.

A tabela ***store*** armazena a lista de todas as lojas existentes no sistema. Cada loja possui um *manager* (funcionário responsável) e morada. A loja é referida no inventário, nos clientes e nos funcionários, tal como já mencionado.

Na tabela ***payment*** encontram-se todos os pagamentos efetuados pelos clientes, com informações como a quantia, a data do pagamento, o funcionário que processou o pagamento e o aluguer a que se refere (se aplicável). Por fim, a tabela ***rental*** contém uma linha por cada aluguer de cada item do inventário, com os dados de quem alugou o item, quando foi alugado e quando foi devolvido.

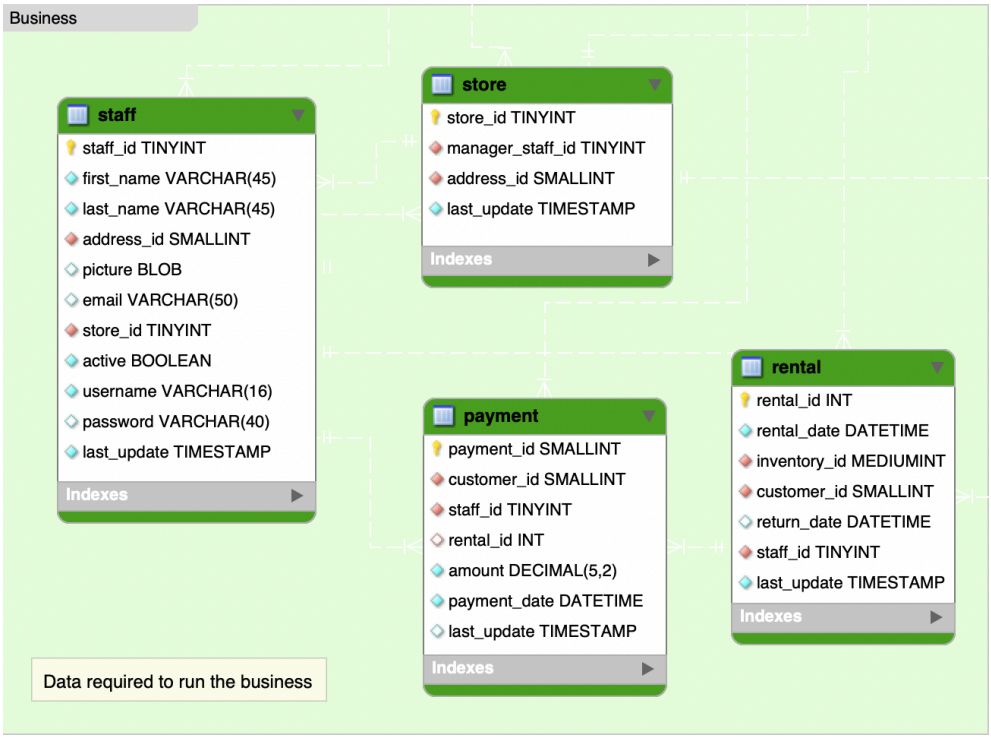


Figura 3 - Tabelas da camada *Business*.

# 3. Queries

A implementação de um conjunto de queries que permitam demonstrar a operacionalidade dos sistemas de bases de dados é um dos requisitos do presente trabalho prático.

Desta forma, uma vez que cada um dos motores utilizados, Oracle, MongoDB e Neo4j, têm diferentes características e diferentes finalidades, foram definidas diferentes *queries* para um deles, que se encontram na Secção 4.3. , 6.3. e 7.3. respetivamente.

No caso do sistema de gestão de bases de dados **Oracle**, todos os seus dados são guardados em tabelas, que se encontram relacionadas com outras tabelas através de chaves estrangeiras e de tabelas intermédias.

O **MongoDB** foi concebido especialmente para quando os dados podem ser representados em vários documentos, não sendo adequado para quando temos que interconectar múltiplas vezes os dados. Desta forma, o MongoDB é mais apropriado para realizar **consultas.**

O **Neo4j** é focado nos relacionamentos (arestas) que existem nos dados, funcionando bastante bem quando existem vários relacionamentos entre várias entidades, ao contrário de grande parte dos restantes motores de bases de dados. Como tal, as queries que melhor demonstram a operacionalidade deste motor são as alusivas a **pesquisas relacionais**.

# 4. Base de Dados Relacional - Oracle

## 4.1. Modelo de Dados

O esquema da base de dados Sakila desenvolvido para Oracle foi idêntico ao original em MySQL, uma vez ambas seguem o modelo relacional. Desta forma, não foi necessário alterar nenhuma tabela, atributo ou relacionamento, mantendo toda a estrutura inicial. Foram também criadas várias *views* e *triggers*.

No entanto, foi necessário proceder a algumas alterações devido ao facto do motor de base de dados Oracle não fornecer algumas funcionalidades que são facultadas pela MySQL. Essas alterações apresentam-se em seguida.

- Os **Inline Index** deixaram de existir, passando a primeiro ser criada a tabela e, só em seguida, o index;
- Os tipos **numéricos** (floats, entre outros) passaram a ser **integers**, exceto o preço que devido ao facto de ter que ser obrigatoriamente um decimal ficou com o tipo **number**;
- Os **defaults** foram todos praticamente excluídos, principalmente os das chaves estrangeiras.

## 4.2. Migração de Dados

Visto serem motores de bases de dados equivalentes, suportando a mesma linguagem de interrogação (SQL), não foi necessário proceder a alterações do *script* de criação dos dados, sendo apenas necessário executá-lo.

## 4.3. *Views*

As *queries* em Oracle são as *views*.

### 4.3.1 *View 1*

Vendas por categoria de filme:

```
CREATE VIEW sales_by_film_category AS
        SELECT c.name AS category,
                SUM(p.amount) AS total_sales
        FROM payment p
        INNER JOIN rental r ON (p.rental_id = r.rental_id)
        INNER JOIN inventory i ON (r.inventory_id = i.inventory_id)
        INNER JOIN film f ON (i.film_id = f.film_id)
        INNER JOIN film_category fc ON (f.film_id = fc.film_id)
        INNER JOIN category c ON (fc.category_id = c.category_id)
        GROUP BY c.name
        ORDER BY total_sales;
```

Excerto 1 - *View* 1 (Oracle).

### 4.3.2 *View 2*

Lista de *staff*:

```
CREATE VIEW staff_list AS
    SELECT s.staff_id AS ID,
           (s.first_name || ' ' || s.last_name) AS name,
           a.address AS address,
           a.postal_code AS zip_code,
           a.phone AS phone,
           city.city AS city,
           country.country AS country,
           s.store_id AS SID
    FROM staff s
    JOIN address a ON (s.address_id = a.address_id)
    JOIN city ON (a.city_id = city.city_id)
    JOIN country ON (city.country_id = country.country_id);
```

Excerto 2 - *View* 2 (Oracle).

### 4.3.3 *View 3*

Lista de *customers*:

```
CREATE VIEW customer_list AS
    SELECT cu.customer_id AS ID,
           (cu.first_name || ' ' || cu.last_name) AS name,
           a.address AS address, a.postal_code AS zip_code,
            a.phone AS phone,
           city.city AS city,
           country.country AS country,
           cu.active AS notes,
           cu.store_id AS SID
    FROM customer cu
    JOIN address a ON cu.address_id = a.address_id
    JOIN city ON a.city_id = city.city_id
    JOIN country ON city.country_id = country.country_id;
```

Excerto 3 - *View* 3 (Oracle).

# 5. Base de Dados Não Relacionais

Para a migração dos dados de MySQL (Base de Dados Relacional) para Bases de Dados Não Relacionais, recorremos à linguagem de programação Ruby e a bibliotecas de MongoDB e Neo4j. Para acedermos ao MySQL usamos o *ActiveRecord*.

O *Active Record* é a camada do sistema responsável por representar dados e lógica de negócios (é o M, modelo, no MVC). O *Active Record* facilita a criação e o uso de objetos cujos dados requerem armazenamento persistente numa base de dados.

O *Active Record* oferece várias funcionalidades, sendo as mais importantes mencionadas em seguida.

- Representar modelos e os seus dados;
- Representar associações entre esses modelos;
- Representar hierarquias de herança através de modelos que estão relacionados;
- Validar os modelos antes que eles persistam na base de dados;
- Executar operações de bases de dados de uma forma orientada a objetos.

Desta forma, foi criado um *ActiveRecord::Base* para cada uma das tabelas presentes em MySQL. Em seguida, apresenta-se o código resultante, por exemplo, para a tabela *Store*:

```ruby
module SqlSakila
  class Store < ActiveRecord::Base
    self.table_name = "store"
    self.primary_key = "store_id"

    belongs_to :address, class_name: "Address"
    belongs_to :manager_staff, class_name: "Staff"

    has_many :staff, class_name: "Staff", foreign_key: :store_id
    has_many :inventory, class_name: "Inventory", foreign_key: :store_id
    has_many :customer, class_name: "Customer", foreign_key: :store_id
  end
end
```

Excerto 4 - Representação dos dados do SGBD MySQL em Ruby.

Tal como se pode ver, encontram-se representadas todas as informações da tabela *store* original em MySQL: o nome da tabela é ***store***, tem como chave primária o ***store_id***, possui duas chaves estrangeiras, pertencentes à tabela ***Address*** e à tabela ***Staff*** e, por fim, é chave estrangeira de três diferentes tabelas, ***Staff***, ***Inventory*** e ***Customer***.

# 6. Base de Dados Orientada a Documentos - MongoDB

## 6.1. Modelo de Dados

O modelo de dados desenvolvido para MongoDB teve como principal objetivo eliminar todas as relações, transformando-as em documentos alinhados.

Desta forma, passaram a existir apenas os quatro seguintes documentos:

***Actor:***

Este documento contém o **primeiro** e **último nome** do ator e a **data da última atualização**. Em seguida, tem uma lista de **filmes**, com os respetivos atributos também presentes em MySQL, que por sua vez contém ainda uma lista de **categorias** e uma lista de **linguagens**.

***Store:***

O documento loja contém a **data da última atualização**, a **morada** da loja com os respetivos atributos, o **manager** responsável, a lista de todos os membros do **staff** e, por fim, os dados dos **clientes** da loja.

***Film:***

Esta coleção contém o **título** do filme, a **descrição**, o **ano de lançamento**, a **duração do aluguer**, o **custo** do aluguer do filme para a duração especificada, a **duração** do filme, a **classificação** do filme, o **valor cobrado** ao cliente se o filme não for devolvido ou se for devolvido danificado, lista de **funcionalidades especiais** incluídas e a **data da última atualização**, informações que também se encontram contidas como atributos na tabela original. Em seguida, contém também uma lista de **categorias**, de **linguagens** e de **atores**, cada um com as respetivas informações.

***Customer:***

O cliente contém o respetivo **id**, o **id da store** a que pertece, o seu **primeiro** e **último nome**, o **e-mail**, uma flag que indica se é um cliente **ativo** ou não, a **data da criação** da ficha do cliente, a **data da última atualização**, a sua **morada**, com os respetivos dados e, por fim, uma lista dos **filmes** alugados, com os respetivos atributos.

Assim, toda a conversão pensada para o MongoDB esteve relacionada com a transformação dos relacionamentos originais em documentos embebidos no documento de cada uma das coleções explicadas anteriormente.

No Anexo 1 encontra-se o modelo de dados representativo do nosso JSON *schema*.

## 6.2. Migração de Dados

Uma vez determinado o modelo de dados, procedemos à migração dos dados, recorrendo à linguagem de programação Ruby e à biblioteca Mongo Ruby Driver. Como referido anteriormente, para aceder ao MySQL usamos o *ActiveRecord*.

Para migrarmos os dados, desenvolvemos quatro diferentes métodos, cada um para migrar cada uma das quatro *collections* criadas e apresentadas na secção anterior. O método **collect_actors**, tal como o nome indica, trata dos atores, o método **collect_films** está responsável pelos filmes, **collect_stores** pelas lojas e, por fim, **collect_customers** pelos clientes.

```
  def collect_actors
    collection = @mongo_client[:actor]
    actors = SqlSakila::Actor.includes(film: :category).as_json(include: {film:
{include: [:category]}})
    actors.each do |actor|
      actor["film"].each do |film|
        film["category"].each do |category|
          category.delete("category_id")
        end
        film["language"] = SqlSakila::Film.find(film["film_id"]).
                           language.as_json(except: [:language_id])
        ol = SqlSakila::Film.find(film["film_id"]).original_language
        film["original_language"] = ol.as_json(except: [:language_id]) if ol
        film.delete("film_id")
        film.delete("language_id")
        film.delete("original_language_id")
      end
      actor.delete("actor_id")
      collection.insert_one(actor)
    end
  end
```

Excerto 5 - Migração de Dados para MongoDB.

## 6.3. *Queries*

Tal como referido na Secção 3. , decidimos implementar diferentes queries para cada um dos motores de bases de dados uma vez que todos eles têm diferentes características e diferentes propósitos de utilização.

Desta forma, tendo em consideração que o MongoDB é adequado para consultas, as queries implementadas seguem esse propósito e apresentam-se em seguida.

### 6.3.1 *Query 1*

Título dos filmes que o utilizador "MARY SMITH" alugou (alterando o nome pode pesquisar-se por qualquer outro utilizador):

```
db.customers.find({"first_name":"MARY", "last_name": "SMITH"},
                  {"_id":0,"film.title":1}).pretty();
```

Excerto 6 - *Query 1* (MongoDB).

### 6.3.2 *Query 2*

Quantidade de filmes que o utilizador "MARY SMITH" alugou:

```
db.customers.aggregate(
    {$match: {"first_name": "MARY", "last_name":"SMITH"}},
    {$unwind: "$film"},
    {$group: {_id:null, number:{$sum:1}}
});
```

Excerto 7 - *Query 2* (MongoDB).

### 6.3.3 *Query 3*

Todos os filmes em que o ator “PENELOPE GUINESS” participou:

```
db.actor.find({"first_name":"PENELOPE", "last_name": "GUINESS"},
               {"_id":0,"film.title":1}).pretty();
```

Excerto 8 - *Query 3* (MongoDB).

### 6.3.4 *Query 4*

Morada da loja a que o utilizador "MARY SMITH" pertence:

```
db.store.find({"customer.first_name":"MARY", "customer.last_name": "SMITH"},
              {"address":1}).pretty();
```

Excerto 9 - *Query 4* (MongoDB).

### 6.3.5 *Query 5*

Todos os atores do filme:

```
db.film.find({"title":"ACADEMY DINOSAUR"},
             {"_id":0, "actor.first_name":1, "actor.last_name":1}).pretty();
```

Excerto 10 - *Query 5* (MongoDB).

# 7. Base de Dados Orientada a Grafos - Neo4j

## 7.1. Modelo de Dados

Para migrar a Base de Dados Sakila de um modelo relacional para um modelo não relacional, neste caso orientado a grafos, foi necessário repensar o esquema da base de dados de modo a que fosse implementado tendo em consideração as características do Neo4j.

Desta forma, as alterações efetuadas foram as seguintes:

**Atributo *store_id* da tabela *Staff*:**

No modelo original da base de dados Sakila, a tabela *Staff* tem como chave estrangeira o *store_id* da Tabela *Store*. No entanto, uma vez que é o membro do *staff* que pertence a uma *store*, faz mais sentido que seja a Tabela *Staff* a ter como chave estrangeira o *store_id*. Pensamos que o modelo de dados original está organizado desta forma apenas por questões de normalização, o que não é uma premissa das bases de dados não relacionais.

**Tabela *Inventory*:**

A tabela *Inventory* foi substituída por um relacionamento entre a tabela *Film* e *Rental*, que tem como atributo a *store_id*. Em MySQL apenas existia esta tabela por questões de normalização, no entanto, em Neo4j essa tabela pode traduzir-se num simples relacionamento entre as tabelas *Film* e *Rental*, mantendo assim os requisitos da base de dados Sakila e tirando proveito das características de Neo4j.

**Tabela *film_text*:**

Após uma análise dos dados presentes na base de dados, optamos por eliminar a tabela *film*_text pois as informações que esta contém já se encontram presentes na tabela *film*.

**Tabela *film_category* e *film_actor*:**

Tendo em consideração as características de Neo4j, tanto a tabela *film_category*, como a tabela *film_actor*, foram substituídas por dois relacionamentos, de *film* para *category* e de *film* para *actor*, respetivamente.

No que diz respeito às restantes tabelas, atributos e relacionamentos, a conversão foi direta para os vértices e arestas de Neo4j.

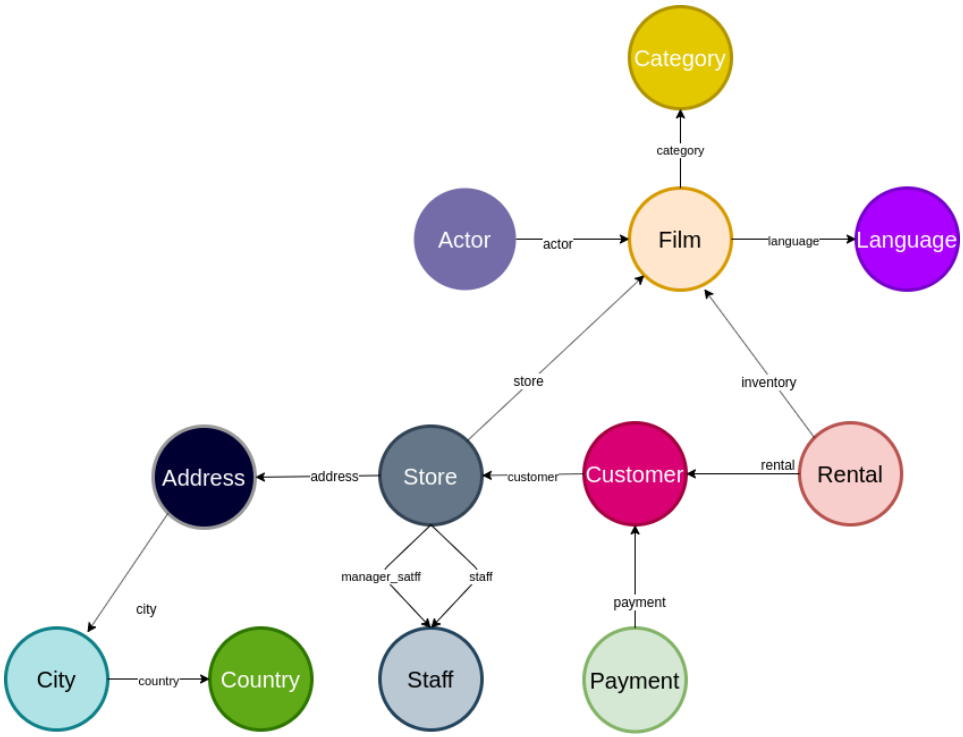


Figura 4 - Representação grafos Neo4j.

## 7.2. Migração de Dados

Decidido o modelo de dados, procedemos à migração dos dados, recorrendo à linguagem de programação Ruby e à biblioteca Neo4j.rb. Como já referido, para aceder ao MySQL usamos o *ActiveRecord*.

Para efetivamente migrarmos os dados, desenvolvemos dois diferentes métodos. O primeiro, chamado **collect_models**, tem como objetivo criar os nodos sem as ligações.

Em seguida, apresentamos um excerto do código implementado para esse efeito.

```
def collect_models
  SqlSakila::Actor.all.each do |actor|
    Actor.create(actor.as_json)
  end

  SqlSakila::Film.all.each do |film|
    Film.create(film.as_json(except: [:language_id, :original_language_id]))
  end
  ...
  ...
end
```

Excerto 11 - Migração de Dados para Neo4j (método *collect_models*).

O segundo método, chamado **create_edges**, foi implementado para criar as arestas/relações entre os nodos anteriormente criados. Os nodos foram criados inicialmente e, só numa fase seguinte, as relações, por simplicidade de garantia de não repetição dos nodos.

```
def create_edges
  SqlSakila::Actor.all.each do |actor|
    actor_n = Actor.find_by(actor_id: actor.actor_id)
    actor.film.each do |f|
      actor_n.film << Film.find_by(film_id: f.film_id)
    end
  end

  SqlSakila::Film.all.each do |film|
    film_n = Film.find_by(film_id: film.film_id)
    film.actor.each do |a|
      film_n.actor << Actor.find_by(actor_id: a.actor_id)
    end

    film.category.each do |c|
      cc = Category.find_by(category_id: c.category_id)
      film_n.category << cc
      cc.film << film_n
    end
    ...
  end
end
```

Excerto 12 - Migração de Dados para Neo4j (método *create_edges*).

## 7.3. *Queries*

Tal como já explicado, decidimos implementar diferentes queries para cada um dos motores de bases de dados visto que todos eles têm diferentes características e diferentes propósitos de utilização.

Desta forma, tendo em consideração que o Neo4j é adequado para pesquisas relacionais, as queries implementadas seguem esse intuito e apresentam-se em seguida.

### 7.3.1 *Query 1*

Top 5 dos clientes com mais alugueres:

```
match (c:`Neo4jSakila::Customer`)<-[:rental]-(r:`Neo4jSakila::Rental`)
return c, count(r)
order by count(r) desc
limit 5;
```

Excerto 13 - *Query 1* (Neo4j).

### 7.3.2 *Query 2*

Top 10 dos clientes que gastam mais dinheiro:

```
match (c:`Neo4jSakila::Customer`)<-[:payment]-(p:`Neo4jSakila::Payment`)
return c, sum(toInteger(p.amount))
order by sum(toInteger(p.amount)) desc
limit 10;
```

Excerto 14 - *Query* 2 (Neo4j).

### 7.3.3 *Query 3*

*Rank* mais alugueres por loja:

```
match (s:`Neo4jSakila::Store`)<-[:customer]-(c:`Neo4jSakila::Customer`)
                              <-[:payment]-(p:`Neo4jSakila::Payment`)
return s,count(p)
order by count(p) desc;
```

Excerto 15 - *Query 3* (Neo4j).

### 7.3.4 *Query 4*

*Rank* loja com mais faturamento:

```
match (s:`Neo4jSakila::Store`)<-[:customer]-(c:`Neo4jSakila::Customer`)
                               <-[:payment]-(p:`Neo4jSakila::Payment`)
return s,sum(toInteger(p.amount))
order by sum(toInteger(p.amount)) desc;
```

Excerto 16 - *Query 4* (Neo4j).

### 7.3.5 *Query 5*

*Rank* loja com mais filmes:

```
match (s:`Neo4jSakila::Store`)<-[:film]-(f:`Neo4jSakila::Film`)
return s, count(f)
order by count(f) desc;
```

Excerto 17 - *Query 5* (Neo4j).

# 8. Conclusão

A migração de dados entre diferentes sistemas de gestão de bases de dados é um processo tão custoso quão maior for a base de dados que se pretende migrar, sendo necessário adequar o respetivo esquema às características do motor de base de dados em questão.

Ao traduzir-se a base de dados Sakila em MySQL para Oracle, MongoDB e Neo4j foi evidente perceber que cada uma delas tem uma finalidade diferente.

No caso da Oracle, é um sistema praticamente idêntico ao MySQL, uma vez que ambos seguem o modelo relacional, apenas com a diferença do Oracle ser uma base de dados Enterprise, com suporte comercial. O Mongo é melhor para consultas de informações que estejam estruturadas num pequeno documento, sem coisas *nested* (aninhadas). O Neo4j é a melhor para pesquisas relacionais, sendo por esse motivo que as respetivas queries dizem respeito a *rankings*.

Desta forma, foi possível perceber que ao escolher-se um motor de base de dados tem que se ter atenção as características que se pretende privilegiar e quais vão ser as interrogações que serão mais frequentes para assim poder ser feita uma escolha consciente.

# Lista de Siglas e Acrónimos

**BD** Base de Dados

SGBD Sistema de Gestão de Bases de Dados

SQL *Structured Query Language*

# Anexos

# I. Anexo 1: Modelo Representativo MongoDB

**ACTOR**: FIRST NAME,
LAST NAME,
LAST UPDATED,
FILM [{
TITLE,
DESCRIPTION,
RELEASE YEAR,
RENTAL DURATION,
RENTAL_RATE,
LENGTH,
RATING,
REPLACEMENT COST,
SPECIAL FEATURES,
LAST_UPDATED,
CATEGORY[{
NAME
LAST_UPDATED
}],
LANGUAGE [{
NAME,
LAST_UPDATED
}]
}]

**FILM**: TITLE,
DESCRIPTION,
RELEASE YEAR,
RENTAL DURATION,
RENTAL_RATE,
LENGTH,
RATING,
REPLACEMENT COST,
SPECIAL FEATURES,
LAST_UPDATED,
CATEGORY[{
NAME
LAST_UPDATED
}],
LANGUAGE [{
NAME,
LAST_UPDATED
}]
ACTOR [{
FIRST NAME,
LAST NAME,
LAST UPDATED
}]

**STORE**: LAST_UPDATED,
ADDRESS {
ADDRESS,
ADDRESS2,
DISTRICT,
POSTAL CODE,
PHONE,
LAST_UPDATED,
CITY {
CITY,
LAST_UPDATED,,
COUNTRY {
COUNTRY,
LAST_UPDATED
}
}
}
MANAGER {
ADDRESS {
ADDRESS,
ADDRESS2,
DISTRICT,
POSTAL CODE,
PHONE,

```
      LAST_UPDATED,
      CITY {
        CITY,
        LAST_UPDATED,,
        COUNTRY {
          COUNTRY,
          LAST_UPDATED
        }
      }
}
},
STAFF [{ADDRESS {
        ADDRESS,
        ADDRESS2,
        DISTRICT,
        POSTAL CODE,
        PHONE,
        LAST_UPDATED,
        CITY {
          CITY,
          LAST_UPDATED,,
          COUNTRY {
            COUNTRY,
            LAST_UPDATED
          }
        }
}]
CUSTOMER_ID,
  STORE_ID,
  FIRST_NAME,
  LAST_NAME,
  EMAIL,
  ACTIVE,
  CREATE_DATE,
  LAST_UPDATED,
  ADDRESS {
    ADDRESS,
    ADDRESS2,
    DISTRICT,
    POSTAL CODE,
    PHONE,
    LAST_UPDATED,
    CITY {
      CITY,
      LAST_UPDATED,,
      COUNTRY {
        COUNTRY,
        LAST_UPDATED
      }
  }
  FILM [{ TITLE,
    DESCRIPTION,
    RELEASE YEAR,
    RENTAL DURATION,
    RENTAL_RATE,
    LENGTH,
    RATING,
    REPLACEMENT COST,
    SPECIAL FEATURES,
    LAST_UPDATED,
    CATEGORY[{
      NAME
      LAST_UPDATED
    }],
    LANGUAGE [{
      NAME,
      LAST_UPDATED
    }]
    ACTOR [{
      FIRST NAME,
     LAST NAME,
     LAST UPDATED
    }]
  }]
```

**CUSTOMERS**: CUSTOMER_ID,
STORE_ID,
FIRST_NAME,
LAST_NAME,
EMAIL,
ACTIVE,
CREATE_DATE,
LAST_UPDATED,
ADDRESS {
ADDRESS,
ADDRESS2,
DISTRICT,
POSTAL CODE,
PHONE,
LAST_UPDATED,
CITY {
CITY,
LAST_UPDATED,,
COUNTRY {
COUNTRY,
LAST_UPDATED
}
}
FILM [{ TITLE,
DESCRIPTION,
RELEASE YEAR,
RENTAL DURATION,
RENTAL_RATE,
LENGTH,
RATING,
REPLACEMENT COST,
SPECIAL FEATURES,
LAST_UPDATED,
CATEGORY[{
NAME
LAST_UPDATED
}],
LANGUAGE [{
NAME,
LAST_UPDATED
}]
ACTOR [{
FIRST NAME,
LAST NAME,
LAST UPDATED
}]
}]