

**Universidade do Minho**

Mestrado Integrado em Engenharia Informática

**Unidade Curricular de**

**Bases de Dados NOSQL**

Ano Letivo de 2019/2020

**Migração *Sakila***

**Ana Ribeiro - A82474**

**Jéssica Lemos - A82061**

**Pedro Freitas - A80975**

Janeiro, 2020

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

**Migração *Sakila***

**Ana Ribeiro - A82474**

**Jéssica Lemos - A82061**

**Pedro Freitas - A80975**

Janeiro, 2020

**Resumo**

Este trabalho que foi realizado no âmbito da unidade curricular de Base de Dados NoSQL, consistiu na migração da base de dados *Sakila* para uma outra base de dados relacional, em *Oracle* e duas não relacionais, uma orientada a documentos, *Mongo*, e outra a grafos *Neo4j*.

Inicialmente foi criado um esquema para cada um dos novos sistemas, equivalente ao esquema da base de dados relacional fornecido, sendo posteriormente explicado os processos para fazer a migração dos dados contidos na base de dados fornecida. Foi ainda implementado um conjunto de *queries* com o intuito de demonstrar a operacionalidade dos sistemas.

Por fim realizamos uma análise crítica do trabalho realizado, comparando os modelos e as funcionalidades agora implementadas.

**Palavras-Chave:** Bases de Dados Relacional, Bases de Dados Relacional, MongoDB, Neo4j, Sakila, SQL, noSQL, Oracle, Migração de dados

**Índice**

[**Resumo** i](#_Toc29723567)

[**Índice** ii](#_Toc29723568)

[**Índice de Figuras** iii](#_Toc29723569)

[**Índice de Tabelas** iv](#_Toc29723570)

[1 Introdução 1](#_Toc29723571)

[1.1 Contextualização 1](#_Toc29723572)

[1.2 Apresentação do Caso de Estudo 1](#_Toc29723573)

[1.3 Motivação e Objectivos 1](#_Toc29723574)

[1.4 Estrutura do Relatório 1](#_Toc29723575)

[2 Queries 2](#_Toc29723576)

[3 Base de Dados Relacional - Oracle 3](#_Toc29723577)

[3.1 Modelo 3](#_Toc29723578)

[3.2 Migração 4](#_Toc29723579)

[3.3 Queries 4](#_Toc29723580)

[3.4 Resultados 6](#_Toc29723581)

[4 Base de Dados Orientada a Grafos - Neo4j 8](#_Toc29723582)

[4.1 Modelo 8](#_Toc29723583)

[4.2 Migração 9](#_Toc29723584)

[4.3 Queries 10](#_Toc29723585)

[4.4 Resultados 12](#_Toc29723586)

[4.5 Comparação com o Modelo Relacional 14](#_Toc29723587)

[5 Conclusões e Trabalho Futuro 15](#_Toc29723588)

[8 Anexos 18](#_Toc29723589)

**Anexos**

[I. Anexo 1 – Criação das tabelas MySQL 19](#_Toc29739733)

[II. Anexo 2 – Foreign keys MySQL 28](#_Toc29739734)

[III. Anexo 3 – Povoamento MySQL 31](#_Toc29739735)

[IV. Anexo 4 – Exportação de Dados Neo4j 35](#_Toc29739736)

[V. Anexo 5 – Importação de Dados Neo4j 38](#_Toc29739737)

**Índice de Figuras**

[Figura 1 - Modelo lógico Sakila 4](#_Toc29755867)

[Figura 2 - Resutado da query 1 em *MySQL* 9](#_Toc29755868)

[Figura 3 - Resultado da query 4 em *MySQL* 10](#_Toc29755869)

[Figura 4 - Resultado da query 5 em *MySQL* 10](#_Toc29755870)

[Figura 5 - Esquema Modelo Neo4j 12](#_Toc29755871)

[Figura 6 - Resultado da *query* 1 em *Neo4j* 15](#_Toc29755872)

[Figura 7 - Resultado da *query* 4 em *Neo4j* 16](#_Toc29755873)

[Figura 8 – Resultado da *query* 5 em *Neo4j* 16](#_Toc29755874)

**Índice de Tabelas**

[Tabela 1 - Tabelas Sakila 3](#_Toc29755852)

# Introdução

<<Este primeiro capítulo deverá ter obrigatoriamente as subsecções abaixo apresentadas.>>

## Contextualização

<<Nesta secção deverá ser apresentado o contexto no qual se desenvolve o caso de estudo seleccionado.>>

## Apresentação do Caso de Estudo

<<Esta secção acolherá uma descrição concisa do caso de estudo seleccionado.>>

## Motivação e Objectivos

<<Esta secção acolherá os diversos motivos, acompanhados por uma breve descrição, que conduziram à proposta e ao desenvolvimento do trabalho, assim como a apresentação detalhada dos diversos objectivos a alcançar com a sua realização.>>

## Estrutura do Relatório

<<Após a leitura da introdução de um relatório é "simpático" apresentar uma breve descrição daquilo que se vai encontrar nos demais capítulos do relatório.>>

# Sakila

De forma a facilitar o processo de migração da base de dados Sakila para os modelos pretendidos, começamos por realizar uma análise detalhada. Assim sendo, de seguida é possível observar todas as tabelas existentes no modelo fornecido, bem como uma breve descrição para perceber a sua função

|  |  |
| --- | --- |
| Tabela | Descrição |
| Ator | Lista a informação de todos os atores. Encontra-se ligada à tabela film através da tabela film\_actor |
| Address | Contém informações de endereços para clientes, funcionários e lojas. |
| Category | Lista as categorias que podem ser atribuídas a um filme. Está ligada a tabela film através da tabela film\_actor |
| City | Contém uma lista de cidades |
| Country | Possui uma lista de países |
| Customer | Dispões uma lista de todos os clientes |
| Film | A tabela dos filmes tem uma lista de todos os filmes que podem existir em stock nas lojas. As que existem de facto estão na tabela inventário. |
| Film\_actor | Esta tabela é utilizada para supostar o relacionamento de muitos para muitos existente entre as tabelas fim e actor. Para cada ator em um determinado filme haverá uma linha neste tabela |
| Film\_category | É usada para apoiar um relacionamento de muitos para muitos entre as tabelas film e category. Assim, para cada categoria associada a um film exitirá uma linha nesta tabela |
| Film\_text | Utilizada para guardar a descrição associada ao film. Esta é mantida em coerência com a tabela film através de triggers |
| Inventory | Contém um registo para cada film exitente numa determinada loja |
| Language | Lista os possíveis idiomas que os filmes podem ter |
| Payment | Regista cada pagamento feito por um cliente, com informações como o valor e o aluguer em questão. |
| Rental | Contém uma linha para cada aluguer de um item do inventário. Esta tem informações sobre quem alugou quando o fez e a quando devolveu |
| Staff | Lista as informações relativas a todos os trabalhadores |
| Store | Lista todas as lojas do sistema. Todo o stock é atribuído a lojas específicas. |

Tabela 1 - Tabelas Sakila

Uma vez estudadas as tabelas, foi necessário verificar as relações estabelecidas de modo a perceber como toda a informação se encontra associada bem como identificar todas as chaves primárias e estrangeiras contida nas mesmas. Este é uma etapa fundamental já que para conseguirmos elaborar o esquema de dados para os novos sistemas é essencial entender como tudo se procede no *MySQL.* Deste modo, de seguida ilustramos o modelo lógico apenas como as chaves primárias e secundárias de cada tabela para que seja mais intuitivo e percetível.

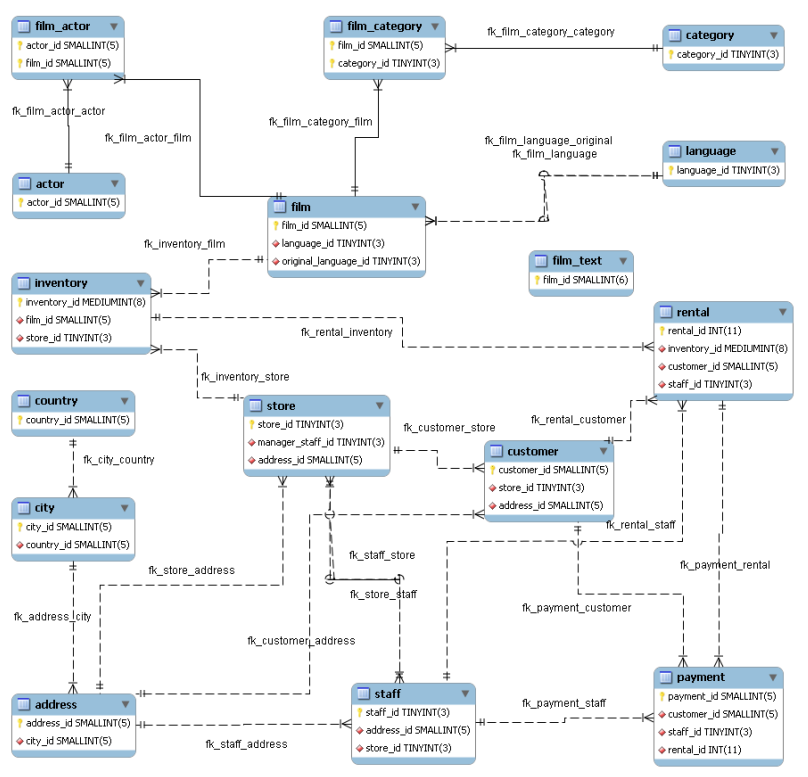


Figura 1 - Modelo lógico Sakila

# Queries

De modo a comparar os resultados obtidos nas implementações da Base de Dados *Sakila* em *Oracle*, *Mongo* e *Neo4j* bem como testar as mesmas, e desta forma demonstrar a operacionalidade dos sistemas implementado, decidimos definir 7 queries que são apresentadas de seguida. Após o processo de definição do modelo e migração da base de dados é expectável que os resultados obtidos das queries no modelo relacional e nos dois não relacionais sejam iguais aos obtidos em *MySQL*.

1. Top 5 dos filmes mais alugados
2. Top 3 das lojas que mais faturam
3. Top 3 dos alugueres de filmes mais longos
4. Top 3 dos filmes que tiveram alugados por mais tempo
5. Top 5 das categorias mais alugadas
6. Top 3 dos atores que participou em mais filmes
7. Top 5 dos distritos com mais clientes
8. Top 5 dos funcionários que alugaram mais filmes

# Base de Dados Relacional - Oracle

Tal como o *MySQL* a *Oracle* também é um Sistema de Gestão de Base de Dados relacional, pelo que o esquema de dados utilizado foi o mesmo. Contudo, verificamos que os tipos de dados existentes variam e como tal foi necessário modificar os que não eram reconhecidos pela *Oracle*. Para além disso, ainda foi preciso ter em atenção o processo de migração dos dados, já que alguns tipos foram alterados e como tal não vão corresponder. Assim sendo, de seguida iremos explicar todo o processo bem como a execução das queries propostas anteriormente.

## Modelo

Como optamos por manter o esquema geral da base de dados e apenas alterar as funcionalidades que não se encontram disponíveis na Oracle, na conversão das tabelas o mais complicado foi perceber quais não existiam e para que tipo deveriam migrar. Após uma análise verificamos que essencialmente existiam quatro problemas diferentes. Primeiro deparamo-nos com a incompatibilidade de dados, por exemplo o *BOOLEAN* não é um tipo definido em *Oracle* pelo que foi necessário substituí-lo por *NUMBER(1,0)*. De forma a realizar esta conversão do modo mais correto possível recorremos a um guia de conversão. Em *MySQL* no ficheiro de criação das tabelas era introduzida uma regra que basicamente atualizava o *timestamp* da tabela sempre que nesta fosse alterado algum dos seus campos. No entanto, verificamos que em *Oracle* não era permitido efetuar da mesma forma. A solução encontrada passou pela criação de um *trigger* para cada tabela que sempre que a mesma é atualizada altera o seu *timestamp* para a data atual como pretendido. Outra das diferenças encontradas foi o modo como é realizada a autoincrementação de uma variável. No *MySQL* utiliza-se o *AUTO\_INCREMENT* enquanto que no *Oracle* se cria uma sequência indicando-lhe em que valor deve começar e quanto deve aumentar de cada vez. Finalmente constatamos que a sintaxe de criação de *primary keys* em *Oracle* não é igual, pelo que mais uma vez tivemos de adptar.

Assim sendo, de seguida apresentamos a título de exemplo a criação da tabela *Country* em ambos os SGBD, onde é possível verificar as principais alterações enumeradas anteriormente inclusive o *trigger* para atualizar o timestamp.

* **Criação da tabela *Country MySQL***

**CREATE TABLE country (**

**country\_id SMALLINT UNSIGNED NOT NULL AUTO\_INCREMENT,**

**country VARCHAR(50) NOT NULL,**

**last\_update TIMESTAMP NOT NULL DEFAULT CURRENT\_TIMESTAMP ON UPDATE CURRENT\_TIMESTAMP,**

**PRIMARY KEY (country\_id)**

**)ENGINE=InnoDB DEFAULT CHARSET=utf8;**

* **Criação da tabela *Country Oracle***

**CREATE TABLE country (**

**country\_id SMALLINT GENERATED BY DEFAULT AS IDENTITY (START WITH 1 INCREMENT BY 1),**

**country VARCHAR(50) NOT NULL,**

**last\_update TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,**

**CONSTRAINT pk\_country PRIMARY KEY (country\_id));**

## Migração

De forma a realizar a migração dos dados optamos por utilizar a linguagem Python. Assim sendo recorremos ao módulo de extensão cx\_Oracle do Python para estabelecer a conexão com a base de dados Oracle. Desta forma, apenas tivemos de percorrer o ficheiro de povoamento da sakila e inserir os dados na base de dados Oracle. Contudo foi necessário proceder a algumas alterações devido à mudança efetuada em alguns tipos de dados.

De forma a facilitar este processo decidimos inserir as chaves estrangeiras apenas depois do povoamento, já que verificamos que existem bastantes relações entre tabelas. Esta alteração apenas foi possível porque é nos assegurado que os dados se encontram consistentes, já que provêm do MySQL. Contudo criamos um ficheiro com todas as chaves estrangeiras, de forma a manter todas as verificações necessárias.

Deste modo após efetuarmos o povoamento da base de dados Oracle, executamos o script que contém todas as Constraints. Assim, na eventualidade de os dados não serem consistentes é possível detetar. É importante referir que tanto o ficheiro de povoamento como o de inserção das chaves estrangeiras se encontram em anexo.

## Queries

Nesta secção iremos apresentar a implementação de todas as queries planeadas anteriormente em *MySQL*.

1. **Top 5 dos filmes mais alugados**

**select f.title as film, count(r.rental\_id) as num\_films**

**from film f**

**inner join inventory i on (i.film\_id = f.film\_id)**

**inner join rental r on (r.inventory\_id = i.inventory\_id)**

**group by f.title**

**order by num\_films DESC**

**fetch first 5 rows only;**

1. **Top 3 das lojas que mais faturam**

**select s.store\_id as Store, sum(p.amount) as Faturado**

**from store s**

**inner join staff st on (st.store\_id = s.store\_id)**

**inner join payment p on (st.staff\_id = p.staff\_id)**

**group by s.store\_id**

**order by Faturado DESC**

**fetch first 3 rows only;**

1. **Top 3 dos alugueres de filmes mais longos**

**select f.title as Title, sum(r.return\_date-r.rental\_date) as TotalDays**

**from film f**

**inner join inventory i on (i.film\_id = f.film\_id)**

**inner join rental r on (r.inventory\_id = i.inventory\_id)**

**group by f.title**

**order by TotalDays DESC**

**fetch first 3 rows only;**

1. **Top 5 das categorias mais alugadas**

**select count(r.rental\_id) as num\_films, c.name as category**

**from category c**

**inner join film\_category fc on (fc.category\_id = c.category\_id)**

**inner join film f on (f.film\_id = fc.film\_id)**

**INNER JOIN inventory i ON (i.film\_id=f.film\_id)**

**INNER JOIN rental r ON (r.inventory\_id=i.inventory\_id)**

**group by c.name**

**order by num\_films DESC**

**fetch first 5 rows only;**

1. **Top 3 dos atores que participaram em mais filmes**

**select a.first\_name, a.last\_name, count(fa.film\_id) as num\_films**

**from actor a**

**inner join film\_actor fa on (fa.actor\_id = a.actor\_id)**

**group by a.actor\_id,a.first\_name, a.last\_name**

**order by num\_films DESC**

**fetch first 3 rows only;**

1. **Top 5 dos distritos com mais clientes**

**select a.district as District, count(c.customer\_id) as Total**

**from address a**

**inner join customer c on (c.address\_id = a.address\_id)**

**group by a.district**

**order by Total DESC**

**fetch first 5 rows only;**

1. **Top 5 dos funcionários que alugaram mais filmes**

**select s.staff\_id as ID, s.first\_name as FirstName, s.last\_name as LastName, count(p.payment\_id) as Total**

**from staff s**

**inner join payment p on (p.staff\_id = s.staff\_id)**

**inner join rental r on (r.rental\_id = p.rental\_id)**

**group by s.staff\_id, s.first\_name, s.last\_name**

**order by Total DESC**

**fetch first 5 rows only;**

## Resultados

De seguida iremos expor os resultados obtidos de algumas das queries definidas previamente, nomeadamente a 1, 4 e 5.

1. **Top 5 dos filmes mais alugados**

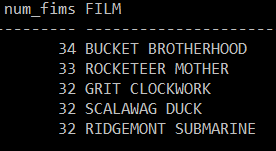


Figura 2 - Resutado da query 1 em *MySQL*

1. **Top 5 das categorias mais alugadas**

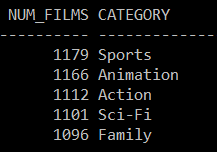


Figura 3 - Resultado da query 4 em *MySQL*

1. **Top 3 dos atores que participaram em mais filmes**

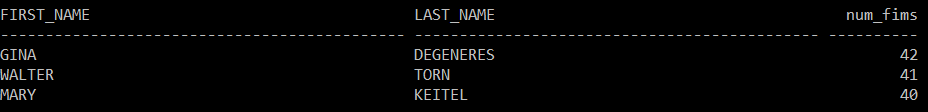


Figura 4 - Resultado da query 5 em *MySQL*

# Base de Dados Orientada a Grafos - Neo4j

## Modelo

De modo a obtermos o modelo apropriado para o *Neo4j* da Base de Dados *Sakila* tornou-se necessário realizarmos algumas alterações no modelo da mesma, entre as quais:

* **Retirar a Identificação –** No Modelo Relacional todas as entidades possuem uma chave estrangeira, que se caracterizam por um ID. Uma vez que no *Neo4j*, por ser uma base de dados orientada a grafos, não são necessários estes mesmos identificadores para criarmos as entidades e os correspondentes relacionamentos decidimos remover os mesmos. Contudo, constatamos que a *Store* não possuía mais nenhum elemento que a caracteriza-se pelo que esta é a única entidade com respetivo identificador.
* **Retirar tabelas desnecessárias –** Ao longo deste processo tornou-se percetível que existiam algumas tabelas que não eram necessárias serem convertidas, sendo estas as correspondentes aos relacionamentos muitos para muitos, ou seja, as tabelas*film\_category*, *film\_actor* e *inventory*. No entanto, nesta última existia um indentificador como chave estrangeira, que permitia obtermos o *stock* de filmes existente em cada *store*. Para armazenar este valor optamos por criar uma propriedade no relacionamento entre a *Store* e o *Film* com o *stock* correspondente àquela loja. Por último, optamos por não converter a tabela *film\_text*, tendo em conta que consideramos que esta não acrescentava informação útil nem necessária às seguintes fases do projeto.

Terminada a fase de tomada de decisão, podemos então esboçar o esquema que pretendemos para o nosso modelo, que é apresentado na Figura 1.

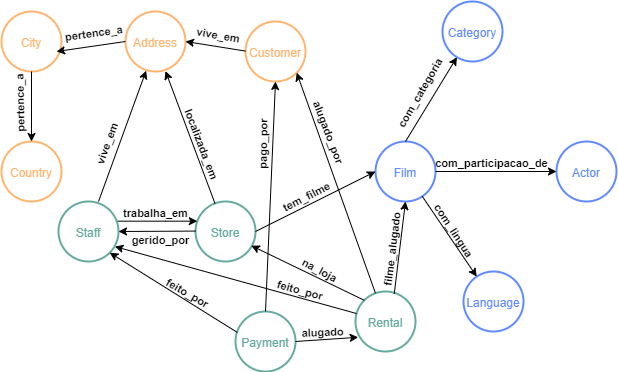


Figura 5 - Esquema Modelo Neo4j

## Migração

Nesta fase passamos para a exportação dos dados da *Sakila* para ficheiros *CSV* de modo a importá-los posteriormente para o *Neo4j*. Tendo em conta as decisões indicadas anteriormente criamos um ficheiro *sql*, que se encontra em anexo, para a extração destes mesmos dados. Basicamente o processo que adotamos para a migração foi o seguinte:

* **Exportação –** nesta fase recorrendo ao *sql* extraímos todos os dados pretendidos tendo em conta as decisões tomadas. Tal como foi indicado anteriormente, decidimos remover os identificadores uma vez que não eram necessários, contudo devido à existência de algumas entidades, como é o caso do *Payment* e da *Rental*, que não tinham nenhum atributo que as pudessem identificar houve a necessidade de migrar estas com os seus IDs para a posterior criação dos relacionamentos. Para além disso, uma vez que constatamos que existia um ator repetido optamos também por migrar o seu ID. A título exemplificativo apresentamos de seguida um excerto referente à exportação do *Country* e da *City.*

**select \* from country**

**into outfile '/var/lib/mysql-files/country.csv'**

**FIELDS TERMINATED BY ','**

**LINES TERMINATED BY '\n';**

**select city, country from city**

**inner join country on city.country\_id = country.country\_id**

**into outfile '/var/lib/mysql-files/city.csv'**

**FIELDS TERMINATED BY ','**

**LINES TERMINATED BY '\n';**

* **Importação –** realizada a exportação podemos agora importar os dados para *Neo4j*. Nesta fase, tivemos o cuidado de eliminar os identificadores das entidades *Payment*, *Rental* e *Actor,* que foram migrados tal com explicado anteriormente, após a criação dos relacionamentos. De seguida, é exemplificado a importação para os mesmos casos, em que para o *Country* são criados os nós e de seguida um índice no nome do mesmo. Enquanto que para a *City* para além da criação dos nós é criado o relacionamento entre o anterior e esta, sendo por fim criado o índice no nome da mesma.

**load csv from "file:///country.csv" as line**

**create (co :Country {country : line[1]});**

**create index on :Country(country);**

**load csv from "file:///city.csv" as line**

**match (co: Country)**

**where co.country = line[1]**

**create (ci: City {city: line[0]}),**

**(ci)-[r:pertence\_a]->(co);**

**create index on :City(city);**

## Queries

1. **Top 5 dos filmes mais alugados**

**match(f:Film)<-[a:filme\_alugado]-(r:Rental)**

**return f, count(a) as num\_films**

**order by num\_films desc**

**limit 5**

1. **Top 3 das lojas que mais faturam**

**match(s:Store)<-[:trabalha\_em]-(st:Staff)<-[:feito\_por]-(p:Payment)**

**return s, sum(p.amount) as num\_stores**

**order by num\_stores desc**

**limit 3**

1. **Top 3 dos alugueres de filmes mais longos**

**match(r:Rental)-[:filme\_alugado]->(f:Film)**

**where exists(r.return\_date)**

**return f.title, avg(duration.between(datetime(r.rental\_date), datetime(r.return\_date))) as dur\_avg**

**order by dur\_avg desc**

**limit 3**

1. **Top 5 das categorias mais alugadas**

**match(r:Rental)-[:filme\_alugado]->(f:Film)-[:com\_categoria]->(c:Category)**

**return count(f) as num\_films,c**

**order by num\_films desc**

**limit 5**

1. **Top 3 dos atores que participaram em mais filmes**

**match(f:Film)-[:com\_participacao\_de]->(a:Actor)**

**return count(f) as num\_films,a**

**order by num\_films desc**

**limit 3**

1. **Top 5 dos distritos com mais clientes**

**MATCH (c:Customer)-[r:vive\_em]->(a:Address)**

**return count(c) as num\_customer, a.district**

**order by num\_customer desc**

**limit 5**

1. **Top 5 dos funcionários que alugaram mais filmes**

**match(r:Rental)<-[:alugado]-(p:Payment)-[:feito\_por]->(s:Staff)**

**return count(r) as num\_films, s**

**order by num\_films desc**

**limit 5**

## Resultados

Através das *queries* definidas pelo grupo tornou-se assim possível obter várias informações relevantes da Base de Dados *Sakila*. De seguida são apresentados os resultados obtidos em algumas das apresentadas anteriormente.

1. **Top 5 dos filmes mais alugados**

Através desta *query* é possível obter os 5 filmes mais alugados, sendo possível observar neste caso em específico o nome dos filmes bem como o número de vezes que foi alugado. Nesta optamos por devolver os filmes, ou seja, todas as informações referentes aos mesmos, contudo no resultado seguinte apenas apresentamos o título, dado que se tornaria muita informação para ser apresentada.

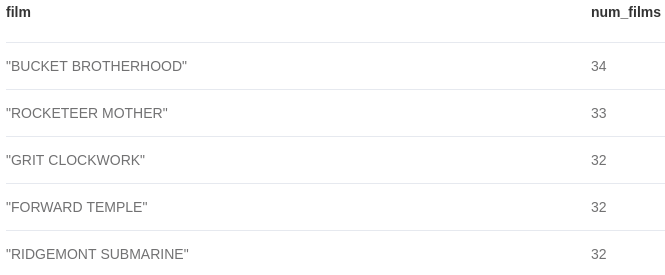


Figura 6 - Resultado da *query* 1 em *Neo4j*

1. **Top 5 das categorias mais alugadas**

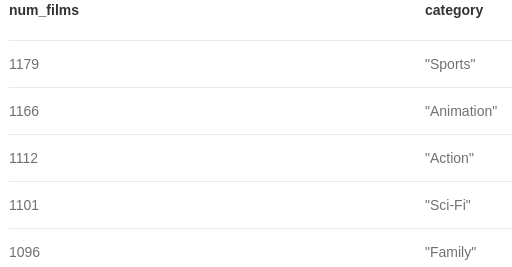


Figura 7 - Resultado da *query* 4 em *Neo4j*

1. **Top 3 dos atores que participaram em mais filmes**



Figura 8 – Resultado da *query* 5 em *Neo4j*

## Comparação com o Modelo Relacional

Como era expectável a migração da Base de Dados Relacional *Sakila* para *Neo4j* permitiu a implementação de um modelo bastante mais simples, em parte porque os relacionamentos de muitos para muitos podem ser tratados uma forma eficiente, e ainda foi possível eliminar as chaves estrangeiras. Para além disso, a implementação das *queries* é bastante mais percetível e rápida sendo que a performance é melhorada uma vez que em *MySQL* as mesmas teriam uma resolução mais complexa com vários *JOINs.*

# Conclusões e Trabalho Futuro

Com a realização deste trabalho foi possível ficar a conhecer mais detalhadamente cada um do SGBD implementados. Ao longo do seu desenvolvimento verificamos que cada um possui as suas características próprias que o torna mais adequado para determinados contextos. Desta forma, no futuro ser-nos-á mais fácil escolher o sistema indicado já que conhecemos melhor cada um.

Assim como o *MySQl* a *Oracle* também é um SGBD relacional, pelo que a migração foi bastante simples sendo necessário apenas algumas alterações principalmente relacionadas com a sintaxe de ambos os modelos. Tendo a *Sakila* já apresenta uma complexidade considerável constatamos na resolução das queries que é necessário realizar vários *Join* o que não e muito bom a nível de desempenho. //ESCREVAM QUALQUER COISA SOBRE O VOSSO

Em útlima instância, concluímos que o trabalho foi realizado com sucesso, uma vez que conseguimos migrar a Sakila para todos os modelos de base de dados propostos.

Referências

1. Ian Robinson, Jim Webber and Emil Eifrem 2015. Graph Databases New Opportunities for Connected Data. Second Edition / O’Reilly Media.

Lista de Siglas e Acrónimos

**BD** Base de Dados

**SGBD** Sistema de Gestão de Base de Dados

**SQL** Structured Query Language

**SBD** Sistema de Base de Dados

**SBDR** Sistema de Base de Dados Relacional

**NoSQL** Not only SQL

# Anexos

**1. Criação das tabelas MySQL**

**2. Foreign keys MySQL**

**3. Povoamento MySQl**

**4. Importação dos Dados Neo4j**

**5. Exportação dos Dados Neo4j**

1. Anexo 1 – Criação das tabelas MySQL

CREATE TABLE actor (

actor\_id SMALLINT GENERATED BY DEFAULT AS IDENTITY (START WITH 1 INCREMENT BY 1),

first\_name VARCHAR(45) NOT NULL,

last\_name VARCHAR(45) NOT NULL,

last\_update TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

CONSTRAINT pk\_actor PRIMARY KEY (actor\_id));

CREATE INDEX idx\_actor\_last\_name ON actor (last\_name);

CREATE OR REPLACE TRIGGER actor\_timestamp\_update\_trigger

BEFORE UPDATE ON actor

FOR EACH ROW

BEGIN

:new.last\_update := current\_timestamp;

END;

/

CREATE TABLE country (

country\_id SMALLINT GENERATED BY DEFAULT AS IDENTITY (START WITH 1 INCREMENT BY 1),

country VARCHAR(50) NOT NULL,

last\_update TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

CONSTRAINT pk\_country PRIMARY KEY (country\_id));

CREATE OR REPLACE TRIGGER country\_timestamp\_update\_trigger

BEFORE UPDATE ON country

FOR EACH ROW

BEGIN

:new.last\_update := current\_timestamp;

END;

/

CREATE TABLE city (

city\_id SMALLINT GENERATED BY DEFAULT AS IDENTITY (START WITH 1 INCREMENT BY 1),

city VARCHAR(50) NOT NULL,

country\_id SMALLINT NOT NULL,

last\_update TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

CONSTRAINT pk\_city PRIMARY KEY (city\_id),

CHECK(country\_id>0));

CREATE INDEX idx\_fk\_city\_country\_id ON city (country\_id);

CREATE OR REPLACE TRIGGER city\_timestamp\_update\_trigger

BEFORE UPDATE ON city

FOR EACH ROW

BEGIN

:new.last\_update := current\_timestamp;

END;

/

CREATE TABLE address (

address\_id SMALLINT GENERATED BY DEFAULT AS IDENTITY (START WITH 1 INCREMENT BY 1),

address VARCHAR(50) NOT NULL,

address2 VARCHAR(50) DEFAULT NULL,

district VARCHAR(20) NOT NULL,

city\_id SMALLINT NOT NULL,

postal\_code VARCHAR(10) DEFAULT NULL,

phone VARCHAR(20) NOT NULL,

last\_update TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

CONSTRAINT pk\_address PRIMARY KEY (address\_id),

CHECK(city\_id>0));

CREATE INDEX idx\_fk\_address\_city\_id ON address (city\_id);

CREATE OR REPLACE TRIGGER address\_timestamp\_update\_trigger

BEFORE UPDATE ON address

FOR EACH ROW

BEGIN

:new.last\_update := current\_timestamp;

END;

/

CREATE TABLE category (

category\_id SMALLINT GENERATED BY DEFAULT AS IDENTITY (START WITH 1 INCREMENT BY 1),

name VARCHAR(25) NOT NULL,

last\_update TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

CONSTRAINT pk\_category PRIMARY KEY (category\_id));

CREATE OR REPLACE TRIGGER category\_timestamp\_update\_trigger

BEFORE UPDATE ON category

FOR EACH ROW

BEGIN

:new.last\_update := current\_timestamp;

END;

/

CREATE TABLE staff (

staff\_id SMALLINT GENERATED BY DEFAULT AS IDENTITY (START WITH 1 INCREMENT BY 1),

first\_name VARCHAR(45) NOT NULL,

last\_name VARCHAR(45) NOT NULL,

address\_id SMALLINT NOT NULL,

picture BLOB DEFAULT NULL,

email VARCHAR(50) DEFAULT NULL,

store\_id SMALLINT NOT NULL,

active NUMBER(1,0) DEFAULT 1,

username VARCHAR(16) NOT NULL,

password varchar(45) DEFAULT NULL,

last\_update TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

CONSTRAINT pk\_staff PRIMARY KEY (staff\_id),

CHECK(address\_id > 0),

CHECK(store\_id > 0));

CREATE INDEX idx\_fk\_staff\_store\_id ON staff (store\_id);

CREATE INDEX idx\_fk\_staff\_address\_id ON staff (address\_id);

CREATE OR REPLACE TRIGGER staff\_timestamp\_update\_trigger

BEFORE UPDATE ON staff

FOR EACH ROW

BEGIN

:new.last\_update := current\_timestamp;

END;

/

CREATE TABLE store (

store\_id SMALLINT GENERATED BY DEFAULT AS IDENTITY (START WITH 1 INCREMENT BY 1),

manager\_staff\_id SMALLINT NOT NULL,

address\_id SMALLINT NOT NULL,

last\_update TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

CONSTRAINT pk\_store PRIMARY KEY (store\_id),

CHECK(manager\_staff\_id > 0),

CHECK(address\_id > 0));

CREATE INDEX idx\_fk\_store\_address\_id ON store (address\_id);

CREATE INDEX idx\_store\_manager\_staff\_id ON store(manager\_staff\_id);

CREATE OR REPLACE TRIGGER store\_timestamp\_update\_trigger

BEFORE UPDATE ON store

FOR EACH ROW

BEGIN

:new.last\_update := current\_timestamp;

END;

/

CREATE TABLE customer (

customer\_id SMALLINT GENERATED BY DEFAULT AS IDENTITY (START WITH 1 INCREMENT BY 1),

store\_id SMALLINT NOT NULL,

first\_name VARCHAR(45) NOT NULL,

last\_name VARCHAR(45) NOT NULL,

email VARCHAR(50) DEFAULT NULL,

address\_id SMALLINT NOT NULL,

active NUMBER(1,0) DEFAULT 1,

create\_date DATE NOT NULL,

last\_update TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

CONSTRAINT pk\_customer PRIMARY KEY (customer\_id),

CHECK(address\_id > 0),

CHECK(store\_id > 0));

CREATE INDEX idx\_fk\_customer\_store\_id ON customer (store\_id);

CREATE INDEX idx\_fk\_customer\_address\_id ON customer (address\_id);

CREATE INDEX idx\_fk\_customer\_last\_name ON customer (last\_name);

CREATE OR REPLACE TRIGGER customer\_timestamp\_update\_trigger

BEFORE UPDATE ON customer

FOR EACH ROW

BEGIN

:new.last\_update := current\_timestamp;

END;

/

CREATE TABLE language (

language\_id SMALLINT GENERATED BY DEFAULT AS IDENTITY (START WITH 1 INCREMENT BY 1),

name CHAR(20) NOT NULL,

last\_update TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

CONSTRAINT pk\_language PRIMARY KEY (language\_id),

CHECK(language\_id>0));

CREATE OR REPLACE TRIGGER language\_timestamp\_update\_trigger

BEFORE UPDATE ON language

FOR EACH ROW

BEGIN

:new.last\_update := current\_timestamp;

END;

/

CREATE TABLE film (

film\_id SMALLINT GENERATED BY DEFAULT AS IDENTITY (START WITH 1 INCREMENT BY 1),

title VARCHAR(255) NOT NULL,

description CLOB DEFAULT NULL,

release\_year NUMBER DEFAULT NULL,

language\_id SMALLINT NOT NULL,

original\_language\_id SMALLINT DEFAULT NULL,

rental\_duration SMALLINT DEFAULT 3 NOT NULL,

rental\_rate DECIMAL(4,2) DEFAULT 4.99 NOT NULL,

length SMALLINT DEFAULT NULL,

replacement\_cost DECIMAL(5,2) DEFAULT 19.99 NOT NULL,

rating VARCHAR(10) DEFAULT 'G',

special\_features VARCHAR(64) DEFAULT NULL,

last\_update TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

CONSTRAINT pk\_film PRIMARY KEY (film\_id),

CHECK(rating IN ('G','PG','PG-13','R','NC-17')),

CHECK(language\_id > 0),

CHECK(original\_language\_id > 0),

CHECK(rental\_duration > 0),

CHECK(length > 0));

CREATE INDEX idx\_fk\_film\_title ON film (title);

CREATE INDEX idx\_fk\_film\_language\_id ON film (language\_id);

CREATE INDEX idx\_fk\_film\_original\_language\_id ON film (original\_language\_id);

CREATE OR REPLACE TRIGGER film\_timestamp\_update\_trigger

BEFORE UPDATE ON film

FOR EACH ROW

BEGIN

:new.last\_update := current\_timestamp;

END;

/

CREATE TABLE film\_actor (

actor\_id SMALLINT NOT NULL,

film\_id SMALLINT NOT NULL,

last\_update TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

CONSTRAINT pk\_film\_actor PRIMARY KEY (actor\_id,film\_id),

CHECK(actor\_id > 0),

CHECK(film\_id > 0));

CREATE INDEX idx\_fk\_film\_actor\_film ON film\_actor (film\_id);

CREATE OR REPLACE TRIGGER film\_actor\_timestamp\_update\_trigger

BEFORE UPDATE ON film\_actor

FOR EACH ROW

BEGIN

:new.last\_update := current\_timestamp;

END;

/

CREATE TABLE film\_category (

film\_id SMALLINT NOT NULL,

category\_id SMALLINT NOT NULL,

last\_update TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

CONSTRAINT pk\_film\_category PRIMARY KEY (film\_id, category\_id),

CHECK(film\_id > 0),

CHECK(category\_id > 0));

CREATE OR REPLACE TRIGGER film\_category\_timestamp\_update\_trigger

BEFORE UPDATE ON film\_category

FOR EACH ROW

BEGIN

:new.last\_update := current\_timestamp;

END;

/

CREATE TABLE film\_text (

film\_id SMALLINT NOT NULL,

title VARCHAR(255) NOT NULL,

description CLOB,

CONSTRAINT pk\_film\_text PRIMARY KEY (film\_id));

CREATE TABLE inventory (

inventory\_id NUMBER(7,0) GENERATED BY DEFAULT AS IDENTITY (START WITH 1 INCREMENT BY 1),

film\_id SMALLINT NOT NULL,

store\_id SMALLINT NOT NULL,

last\_update TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

CONSTRAINT pk\_inventory PRIMARY KEY (inventory\_id),

CHECK(film\_id > 0),

CHECK(store\_id > 0));

CREATE INDEX idx\_fk\_inventory\_film\_id ON inventory (film\_id);

CREATE INDEX idx\_fk\_inventory\_store\_id\_film\_id ON inventory (store\_id,film\_id);

CREATE OR REPLACE TRIGGER inventory\_timestamp\_update\_trigger

BEFORE UPDATE ON inventory

FOR EACH ROW

BEGIN

:new.last\_update := current\_timestamp;

END;

/

CREATE TABLE rental (

rental\_id INT GENERATED BY DEFAULT AS IDENTITY (START WITH 1 INCREMENT BY 1),

rental\_date DATE NOT NULL,

inventory\_id NUMBER(7,0) NOT NULL,

customer\_id SMALLINT NOT NULL,

return\_date DATE DEFAULT NULL,

staff\_id SMALLINT NOT NULL,

last\_update TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

CONSTRAINT pk\_rental PRIMARY KEY (rental\_id),

CHECK(inventory\_id > 0),

CHECK(customer\_id > 0),

CHECK(staff\_id > 0));

CREATE UNIQUE INDEX idx\_rental\_date\_inventory\_customer ON rental (rental\_date,inventory\_id,customer\_id);

CREATE INDEX idx\_fk\_rental\_inventory\_id ON rental (inventory\_id);

CREATE INDEX idx\_fk\_rental\_customer\_id ON rental (customer\_id);

CREATE INDEX idx\_fk\_rental\_staff\_id ON rental (staff\_id);

CREATE OR REPLACE TRIGGER rental\_timestamp\_update\_trigger

BEFORE UPDATE ON rental

FOR EACH ROW

BEGIN

:new.last\_update := current\_timestamp;

END;

/

CREATE TABLE payment (

payment\_id SMALLINT GENERATED BY DEFAULT AS IDENTITY (START WITH 1 INCREMENT BY 1),

customer\_id SMALLINT NOT NULL,

staff\_id SMALLINT NOT NULL,

rental\_id INT DEFAULT NULL,

amount DECIMAL(5,2) NOT NULL,

payment\_date DATE NOT NULL,

last\_update TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

CONSTRAINT pk\_payment PRIMARY KEY (payment\_id),

CHECK(customer\_id > 0),

CHECK(staff\_id > 0));

CREATE INDEX idx\_fk\_payment\_staff\_id ON payment (staff\_id);

CREATE INDEX idx\_fk\_payment\_customer\_id ON payment (customer\_id);

CREATE OR REPLACE TRIGGER payment\_timestamp\_update\_trigger

BEFORE UPDATE ON payment

FOR EACH ROW

BEGIN

:new.last\_update := current\_timestamp;

END;

/

1. Anexo 2 – Foreign keys MySQL

ALTER TABLE city

ADD CONSTRAINT fk\_city\_country FOREIGN KEY (country\_id) REFERENCES country(country\_id);

ALTER TABLE address

ADD CONSTRAINT fk\_address\_city FOREIGN KEY (city\_id) REFERENCES city(city\_id);

ALTER TABLE staff

ADD CONSTRAINT fk\_staff\_address FOREIGN KEY (address\_id) REFERENCES address (address\_id);

ALTER TABLE store

ADD CONSTRAINT fk\_store\_staff FOREIGN KEY (manager\_staff\_id) REFERENCES staff (staff\_id);

ALTER TABLE store

ADD CONSTRAINT fk\_store\_address FOREIGN KEY (address\_id) REFERENCES address (address\_id);

ALTER TABLE staff

ADD CONSTRAINT fk\_staff\_store FOREIGN KEY (store\_id) REFERENCES store (store\_id);

ALTER TABLE customer

ADD CONSTRAINT fk\_customer\_address FOREIGN KEY (address\_id) REFERENCES address (address\_id);

ALTER TABLE customer

ADD CONSTRAINT fk\_customer\_store FOREIGN KEY (store\_id) REFERENCES store (store\_id);

ALTER TABLE film

ADD CONSTRAINT fk\_film\_language FOREIGN KEY (language\_id) REFERENCES language (language\_id);

ALTER TABLE film

ADD CONSTRAINT fk\_film\_language\_original FOREIGN KEY (original\_language\_id) REFERENCES language (language\_id);

ALTER TABLE film\_actor

ADD CONSTRAINT fk\_film\_actor\_actor FOREIGN KEY (actor\_id) REFERENCES actor (actor\_id);

ALTER TABLE film\_actor

ADD CONSTRAINT fk\_film\_actor\_film FOREIGN KEY (film\_id) REFERENCES film (film\_id);

ALTER TABLE film\_category

ADD CONSTRAINT fk\_film\_category\_film FOREIGN KEY (film\_id) REFERENCES film (film\_id);

ALTER TABLE film\_category

ADD CONSTRAINT fk\_film\_category\_category FOREIGN KEY (category\_id) REFERENCES category (category\_id);

ALTER TABLE inventory

ADD CONSTRAINT fk\_inventory\_store FOREIGN KEY (store\_id) REFERENCES store (store\_id);

ALTER TABLE inventory

ADD CONSTRAINT fk\_inventory\_film FOREIGN KEY (film\_id) REFERENCES film (film\_id);

ALTER TABLE rental

ADD CONSTRAINT fk\_rental\_staff FOREIGN KEY (staff\_id) REFERENCES staff (staff\_id);

ALTER TABLE rental

ADD CONSTRAINT fk\_rental\_inventory FOREIGN KEY (inventory\_id) REFERENCES inventory (inventory\_id);

ALTER TABLE rental

ADD CONSTRAINT fk\_rental\_customer FOREIGN KEY (customer\_id) REFERENCES customer (customer\_id);

ALTER TABLE payment

ADD CONSTRAINT fk\_payment\_rental FOREIGN KEY (rental\_id) REFERENCES rental (rental\_id);

ALTER TABLE payment

ADD CONSTRAINT fk\_payment\_customer FOREIGN KEY (customer\_id) REFERENCES customer (customer\_id);

ALTER TABLE payment

ADD CONSTRAINT fk\_payment\_staff FOREIGN KEY (staff\_id) REFERENCES staff (staff\_id);

1. Anexo 3 – Povoamento MySQL

#!/usr/bin/env python3

import re

import cx\_Oracle

import datetime

tabelas = {}

tabelas["rental"] = "insert into rental(rental\_id, rental\_date, inventory\_id, customer\_id, return\_date, staff\_id, last\_update) values"

tabelas["film\_category"] = "insert into film\_category(film\_id, category\_id, last\_update) values"

tabelas["language"] = "insert into language(language\_id, name, last\_update) values"

tabelas["inventory"] = "insert into inventory(inventory\_id, film\_id, store\_id, last\_update) values"

tabelas["actor"] = "insert into actor(actor\_id, first\_name, last\_name, last\_update) values"

tabelas["category"] = "insert into category(category\_id, name, last\_update) values"

tabelas["country"] = "insert into country(country\_id, country, last\_update) values"

tabelas["city"] = "insert into city(city\_id, city, country\_id, last\_update) values"

tabelas["address"] = "insert into address(address\_id, address, address2, district, city\_id, postal\_code, phone, last\_update) values"

tabelas["store"] = "insert into store(store\_id, manager\_staff\_id, address\_id, last\_update) values"

tabelas["film\_text"] = "insert into film\_text(film\_id, title, description) values"

tabelas["customer"] = "insert into customer(customer\_id, store\_id, first\_name, last\_name, email, address\_id, active, create\_date, last\_update) values"

tabelas["film"] = "insert into film(film\_id, title, description, release\_year, language\_id, original\_language\_id,rental\_duration,rental\_rate, length, replacement\_cost, rating, special\_features, last\_update) values"

tabelas["film\_actor"] = "insert into film\_actor(actor\_id, film\_id, last\_update) values"

tabelas["payment"] = "insert into payment(payment\_id, customer\_id, staff\_id, rental\_id, amount, payment\_date, last\_update) values"

tabelas["staff"] = "insert into staff(staff\_id,first\_name,last\_name,address\_id,email,store\_id,active,username,password,last\_update) values"

def leLinhas(inserir):

conn = cx\_Oracle.connect('final/123@localhost:1521/orcl')

mycursor = conn.cursor()

with open("sakila-data (1).sql") as file:

for line in file:

after = re.search("VALUES",line)

if(after is not None):

firstLine = line.split("VALUES")

tabela = line.split(" ")

chave = tabela[2].split("`")

if(len(chave)>1):

c = chave[1]

else:

c = chave[0]

inicio = tabelas[c]

inserir = 1 #indica que vamos começar a povoar

line = firstLine[1]

if(inserir==1):

if(re.search(";",line) is not None):

inserir = 0

l = line.split(';')

line = l[0]

else:

virgulas = line.split(",\n")

line = virgulas[0]

if(re.match("customer",c) is not None):

match = re.search(r'\'\d{4}\-\d{2}\-\d{2} \d{2}\:\d{2}\:\d{2}\'', line)

nova = match.group()

nData = 'TO\_DATE(' + match.group() + ",'" + 'yyyy/mm/dd hh24:mi:ss\')'

nData.lstrip("T")

line = line.replace(nova,nData)

run = inicio + line

elif(re.match("film",c) is not None and re.match("film\_",c) is None):

match = re.search(r'\d{1,2}\.\d{2}', line)

nova = match.group()

line = line.replace("'" + nova +"'",nova)

match2 = re.search(r'\'\d{1,2}\.\d{2}\'', line)

nova2 = match2.group()

n = nova2.split("'")

i = n[1]

line = line.replace(nova2, i)

run = inicio + line

elif(re.match("payment",c) is not None):

campos = line.split(",");

newN = re.search(r'\d+.\d+',campos[4]).group() #convertTonumber

newDate = 'TO\_DATE(' + campos[5] + ',' + "'yyyy/mm/dd hh24:mi:ss')"

run = inicio + campos[0] + "," + campos[1] + "," + campos[2] + "," + campos[3] +"," + newN +"," + newDate + "," + campos[6]

elif(re.match("rental",c) is not None):

campos = line.split(",");

newDate = 'TO\_DATE(' + campos[1] + ',' + "'yyyy/mm/dd hh24:mi:ss')"

newD = 'TO\_DATE(' + campos[4] + ',' + "'yyyy/mm/dd hh24:mi:ss')"

run = inicio + campos[0] + "," + newDate + "," + campos[2] + "," + campos[3] +"," + newD +"," + campos[5] + "," + campos[6]

elif(re.match("staff",c) is not None):

campos = line.split(",");

new10 = campos[10].split(")")

newT = 'TO\_TIMESTAMP(' + new10[0] + ',' + "'yyyy/mm/dd hh24:mi:ss.FF')"

run = inicio + campos[0] + "," + campos[1] + "," + campos[2] + "," + campos[3] +"," + campos[5] + "," + campos[6] + "," + campos[7] + "," + campos[8] + "," + campos[9] + "," + newT + ")"

else:

nLine = line.replace(r"''","' '")

run = inicio + nLine

print(run)

mycursor.execute(run)

conn.commit()

mycursor.close()

conn.close()

leLinhas(0)

1. Anexo 4 – Exportação de Dados Neo4j

select \* from country

into outfile '/var/lib/mysql-files/country.csv'

FIELDS TERMINATED BY ','

LINES TERMINATED BY '\n';

select city, country from city

inner join country on city.country\_id = country.country\_id

into outfile '/var/lib/mysql-files/city.csv'

FIELDS TERMINATED BY ','

LINES TERMINATED BY '\n';

select address, address2, district, city, postal\_code, phone, country from address

inner join city on address.city\_id = city.city\_id

inner join country on city.country\_id = country.country\_id

into outfile '/var/lib/mysql-files/address.csv'

FIELDS TERMINATED BY ','

LINES TERMINATED BY '\n';

select first\_name, last\_name, address, city, email, active, username, password, store\_id from staff

inner join address on address.address\_id = staff.address\_id

inner join city on city.city\_id = address.city\_id

into outfile '/var/lib/mysql-files/staff.csv'

FIELDS TERMINATED BY ','

LINES TERMINATED BY '\n';

select address, city, username, store.store\_id from store

inner join address on address.address\_id = store.address\_id

inner join city on city.city\_id = address.city\_id

inner join staff on staff.staff\_id = store.manager\_staff\_id

into outfile '/var/lib/mysql-files/store.csv'

FIELDS TERMINATED BY ','

LINES TERMINATED BY '\n';

select name from category

into outfile '/var/lib/mysql-files/category.csv'

FIELDS TERMINATED BY ','

LINES TERMINATED BY '\n';

select actor\_id, first\_name, last\_name from actor

into outfile '/var/lib/mysql-files/actor.csv'

FIELDS TERMINATED BY ','

LINES TERMINATED BY '\n';

select first\_name, last\_name, email, active, create\_date, store\_id, address, city from customer

inner join address on address.address\_id = customer.address\_id

inner join city on address.city\_id = city.city\_id

into outfile '/var/lib/mysql-files/customer.csv'

FIELDS TERMINATED BY ','

LINES TERMINATED BY '\n';

select rental\_id, customer.first\_name, customer.last\_name, amount, payment\_date, staff.username, payment\_id from payment

inner join customer on customer.customer\_id = payment.customer\_id

inner join staff on staff.staff\_id = payment.staff\_id

into outfile '/var/lib/mysql-files/payment.csv'

FIELDS TERMINATED BY ','

LINES TERMINATED BY '\n';

select name from language

into outfile '/var/lib/mysql-files/language.csv'

FIELDS TERMINATED BY ','

LINES TERMINATED BY '\n';

select title, description, release\_year, rental\_duration, rental\_rate, film.length, replacement\_cost, rating, special\_features, language.name from film

inner join language on language.language\_id = film.language\_id

into outfile '/var/lib/mysql-files/film.csv'

FIELDS TERMINATED BY ',' ENCLOSED BY '"' LINES TERMINATED BY '\n';

select title, actor.actor\_id from film

inner join film\_actor on film.film\_id = film\_actor.film\_id

inner join actor on film\_actor.actor\_id = actor.actor\_id

into outfile '/var/lib/mysql-files/film\_actor.csv'

FIELDS TERMINATED BY ',' ENCLOSED BY '"' LINES TERMINATED BY '\n';

select film.title, store\_id, film\_text.description, count(\*) from inventory

inner join film on inventory.film\_id = film.film\_id

inner join film\_text on film\_text.film\_id = inventory.film\_id

group by film.film\_id, inventory.store\_id

into outfile '/var/lib/mysql-files/inventory.csv'

FIELDS TERMINATED BY ',' ENCLOSED BY '"' LINES TERMINATED BY '\n';

select title, category.name from film

inner join film\_category on film.film\_id = film\_category.film\_id

inner join category on category.category\_id = film\_category.category\_id

into outfile '/var/lib/mysql-files/film\_category.csv'

FIELDS TERMINATED BY ',' ENCLOSED BY '"' LINES TERMINATED BY '\n';

select rental\_id, rental\_date, ifnull(return\_date,""), customer.first\_name, customer.last\_name, staff.username, film.title, inventory.store\_id from rental

inner join inventory on inventory.inventory\_id = rental.inventory\_id

inner join customer on customer.customer\_id = rental.customer\_id

inner join staff on staff.staff\_id = rental.staff\_id

inner join film on inventory.film\_id = film.film\_id

into outfile '/var/lib/mysql-files/rental.csv'

FIELDS TERMINATED BY ',' ENCLOSED BY '"' LINES TERMINATED BY '\n';

select payment.payment\_id, rental.rental\_id from payment

inner join rental on payment.rental\_id = rental.rental\_id

into outfile '/var/lib/mysql-files/rental-payment.csv'

FIELDS TERMINATED BY ',' ENCLOSED BY '"' LINES TERMINATED BY '\n';

1. Anexo 5 – Importação de Dados Neo4j

load csv from "file:///country.csv" as line

create (co :Country {country : line[1]});

create index on :Country(country);

load csv from "file:///city.csv" as line

match (co: Country)

where co.country = line[1]

create (ci: City {city: line[0]}),

(ci)-[r:pertence\_a]->(co);

create index on :City(city);

using periodic commit 100

load csv from "file:///address.csv" as line

match (ci: City {city: line[3]})-[:pertence\_a]->(co: Country {country: line[6]})

create (adr: Address { address: line[0], address2: line[1], district: line[2], code: line[4], phone: line[5]}),

(adr)-[r:pertence\_a]->(ci);

create index on :Address(address);

load csv from "file:///staff.csv" as line

match (adr: Address {address: line[2]})-[:pertence\_a]->(ci: City {city : line[3]})

create (st: Staff { first\_name: line[0], last\_name: line[1], email: line[4], active: line[5], username: line[6], password: line[7]}),

(st)-[r:vive\_em]->(adr);

load csv from "file:///store.csv" as line

match (adr: Address {address: line[0]})-[:pertence\_a]->(ci: City {city : line[1]})

match (m: Staff {username: line[2]})

create (adr)<-[:localizada\_em]-(sto: Store {store\_id: line[3]})-[:gerido\_por]->(m);

create index on :Store(store\_id);

load csv from "file:///staff.csv" as line

match (staff: Staff {username: line[6]})

match (store: Store {store\_id: line[8]})

create (staff)-[:trabalha\_em]->(store);

create index on :Staff(username);

load csv from "file:///category.csv" as line

create (:Category{name:line[0]});

create index on :Category(name);

load csv from "file:///actor.csv" as line

create (:Actor{actor\_id: line[0], first\_name: line[1], last\_name: line[2]});

create index on :Actor(actor\_id);

create index on :Actor(first\_name, last\_name);

load csv from "file:///customer.csv" as line

match (adr: Address {address: line[6]})-[:pertence\_a]->(c: City {city: line[7]})

match (st: Store {store\_id: line[5]})

create (adr)<-[:vive\_em]-(:Customer {first\_name: line[0], last\_name: line[1], email: line[2], active: line[3], create\_date: replace(line[4],' ','T')})-[:membro\_em]->(st);

load csv from "file:///payment.csv" as line

match (c: Customer {first\_name: line[1], last\_name: line[2]})

match (s: Staff {username: line[5]})

create (c)<-[:pago\_por]-(p:Payment {amount: toFloat(line[3]), payment\_date: replace(line[4],' ','T'), payment\_id: line[6]})-[:feito\_por]->(s);

create index on :Payment(payment\_id);

load csv from "file:///language.csv" as line

create (:Language {name: line[0]});

create index on :Language(name);

load csv from "file:///film.csv" as line

match (l:Language {name: line[9]})

create (:Film {title: line[0], description: line[1], release\_year: line[2], rental\_duration: line[3], rental\_rate: line[4], length: line[5], replacement\_cost: line[6], rating: line[7], special\_features: line[8]})-[:com\_lingua]->(l);

create index on :Film(title);

load csv from "file:///film\_actor.csv" as line

match (f: Film {title: line[0]})

match (a: Actor {actor\_id: line[1]})

create (f)-[:com\_participacao\_de]->(a);

match (a:Actor)

remove a.actor\_id;

load csv from "file:///inventory.csv" as line

match (f: Film {title: line[0]})

match (s: Store {store\_id: line[1]})

create (s)-[:tem\_filme {numero: line[3]}]->(f);

load csv from "file:///film\_category.csv" as line

match (f: Film {title: line[0]})

match (c: Category {name: line[1]})

create (f)-[:com\_categoria]->(c);

load csv from "file:///rental.csv" as line

match (c: Customer {first\_name: line[3], last\_name: line[4]})

match (sta: Staff {username: line[5]})

match (f: Film {title: line[6]})

match (sto: Store {store\_id: line[7]})

create (sta)<-[:feito\_por]-(r:Rental {rental\_id: line[0], rental\_date: replace(line[1],' ','T'), return\_date: replace(line[2], " ", "T")})-[:filme\_alugado]->(f),

(sto)<-[:na\_loja]-(r)-[:alugado\_por]->(c);

create index on :Rental(rental\_id);

load csv from "file:///rental-payment.csv" as line

match (p: Payment {payment\_id: line[0]})

match (r: Rental {rental\_id: line[1]})

create (p)-[:alugado]->(r);

match (p:Payment)

remove p.payment\_id;

match (r:Rental)

remove r.rental\_id;