Universidade do Minho
Escola de Engenharia

BASES DE DADOS NOSQL

Trabalho Prático

SAKILA *Database*

Diogo Araújo A78485; Diogo Nogueira A78957; Duarte Freitas A63129

Conteúdo

Resumo

Este relatório pertence ao trabalho prática da Unidade Curricular **Bases de Dados NoSQL**, e tem como principal objetivo instruir a prática de análise, planeamento e implementação de um Sistema de Bases de Dados Relacional e dois Não Relacionais, tendo como base a Base de Dados **Sakila** disponibilizada em MySQL.

Área de Aplicação: Planeamento e Implementação de diferentes Sistema de Bases de Dados para a database Sakila.

Palavras-Chave: Sakila, MySQL, Oracle, MongoDB, Neo4J, Organização, Planeamento, Análise, Rapidez, Atributos, Relacionamentos, Nodos, Grafo, NoSQL.

1. Contextualização

A ideia base deste trabalho prático passa por agrupar toda a aprendizagem obtida até então, na tentativa de modelar três diferentes tipos de Sistema de Bases de Dados, tendo como raiz a Sakila, uma database totalmente preparada para casos de estudo, dado que traz consigo uma ideia mais real daquilo que pode ser uma Base de Dados massiva em termos de informação.

Para que se conseguisse estabelecer cada SBD, foi necessário primeiro compreender como a Sakila funcionava, estudando as relações existentes entre as várias tabelas e de que modo se poderia eliminar este número excessivo, tornando depois toda a migração mais simples no que toca essencialmente aos Modelos de Dados Não Relacionais.

Dessa forma, criou-se um pensamento inicial para cada modelo a desenvolver e anotou-se a base principal que deveria existir para cada um deles:

- Oracle: Dado que se trata por si só de um modelo também ele Relacional, o processo de migração é o mais simples dos três. Apenas algumas mudanças no povoamento e preparação do *schema* foram precisas.
- MongoDB: Quando falamos em MongoDB, sabemos logo à partida que não é preciso uma preocupação tão grande no que toca à redundância da informação. Assim, pode-se repetir informação para as várias coleções, simplificando-se o processo de extração de dados e respetiva análise.
- Neo4J: Para se construir os vários nodos e respetivas relações, teve-se por base as ligações no modelo MySQL. Algumas relações acabaram por ser evitadas, dado que poderiam ser agregadas na informação do nodo em si.

Estes princípios foram essenciais para se começar a planear cada esquema e com isso se proceder à migração dos dados em bruto para todos os novos sistemas. O relatório vai explicar esses processos, justificando as mudanças mais importantes e o porquê de terem sido tomadas pelos elementos do grupo.

2. Base de Dados SAKILA

2.1 SBD Oracle

Como primeira parte do relatório temos a modelação e transformação da Base de Dados Sakila originalmente feita para a Base de Dados *MySQL*. Assim desta forma existem algumas nuances quando se pensa na Base de Dados (também) relacional *Oracle*. Desde a mudanças no povoamento, até à preparação inicial do *schema*, existem variantes que serão estranhas ao PL/SQL que possam advir do povoamento e *schema* em forma de *MySQL*.

2.1.1. Esquema da Base de Dados

Através da criação dum ficheiro *.ddl* foi possível a criação dum *schema* completo para a colocação do projeto em mão. Para a criação do(s) *tablespace(s)* foi criado a pequena porção de código demonstrada abaixo que nos tornou possível o local onde armazenar fisicamente os dados que serão agregados da Base de Dados *Sakila*. Foi criado um *tablespace* principal intitulado de *TABLESAKILA* e o seu respetivo *datafile* de 5MB que se pode estender até a um máximo de 250MB, criando assim um espaço dinâmico conforme a necessidade de alocação de dados.

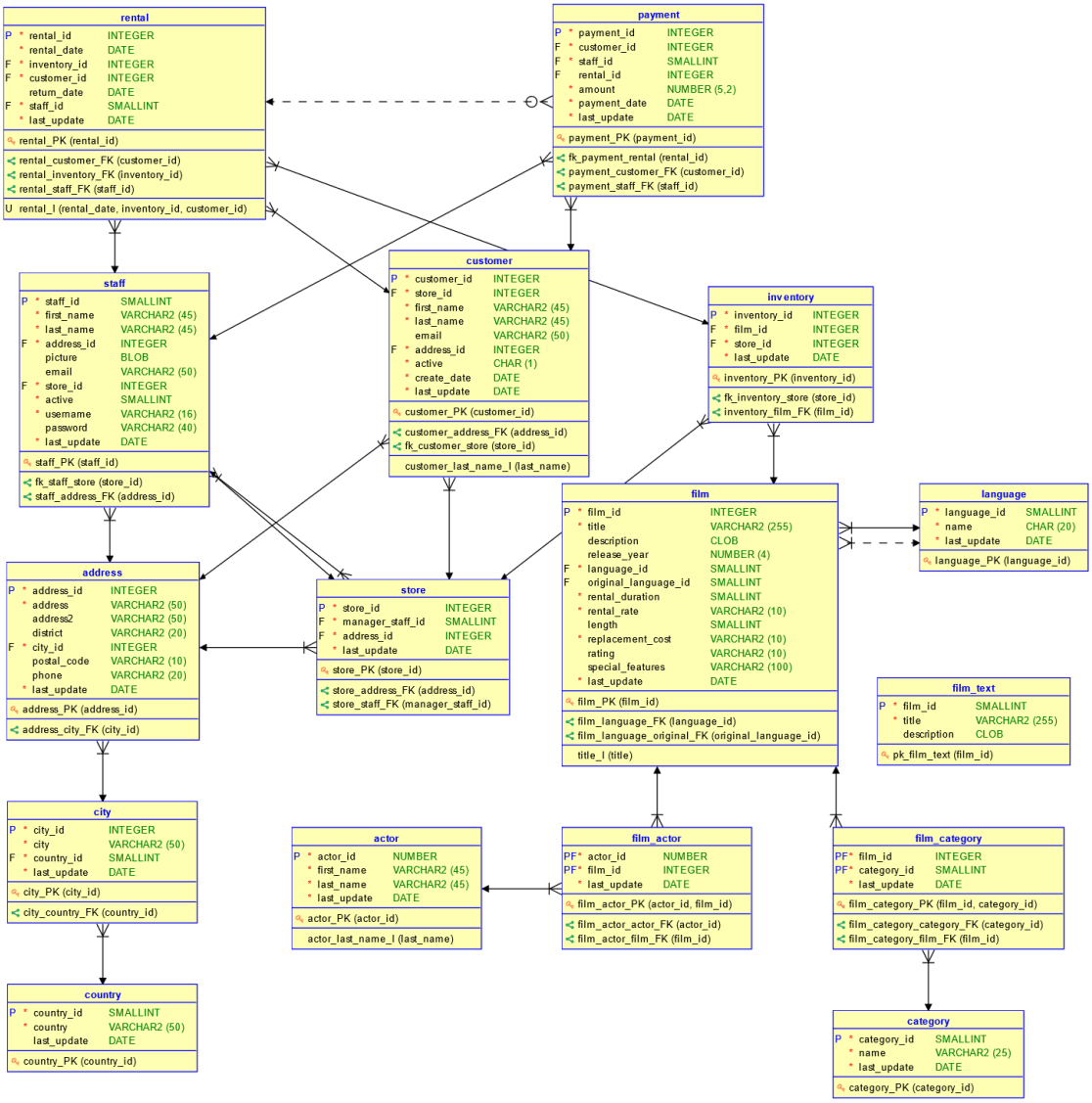
Também foi criado um *temporary tablespace* para melhor alocação e *cache* dos dados para a sessão que o utilizador esteja e assim melhorar o desempenho.

```
CREATE TABLESPACE TABLESAKILA
    DATAFILE
        'DATAFILESAKILA1' SIZE 5M AUTOEXTEND ON NEXT 5M MAXSIZE 250M;

CREATE TEMPORARY TABLESPACE TABLESAKILATEMP
    TEMPFILE
        'DATAFILESAKILATEMP' SIZE 5M;
```

Para a criação do utilizador foi essencial o *script* seguinte que coloca a sua identificação/palavra-passe, a sua quota e os *tablespaces* associados e, por conseguinte, a concessão de vários privilégios necessários para a o funcionamento adequado como a possibilidade da criação de tabelas, *triggers*, sequências e também

procedures e *views*. Assim, com a junção do privilégio para a sua possibilidade de conexão e criação de sessão e assim poder-se conectar e manipular o seu *schema*.

```
CREATE USER sakilanosql IDENTIFIED BY nosql
    DEFAULT TABLESPACE TABLESAKILA
    TEMPORARY TABLESPACE TABLESAKILATEMP
    QUOTA UNLIMITED ON TABLESAKILA
    ACCOUNT UNLOCK;

GRANT connect,
    CREATE SESSION,
    CREATE ANY TABLE,
    CREATE ANY TRIGGER,
    CREATE ANY VIEW,
    CREATE ANY SEQUENCE,
    CREATE ANY PROCEDURE
TO sakilanosql;
```



Por conseguinte, tivemos o pensamento análogo ao modelo lógico do Sakila fornecido pelo *MySQL* e assim criamos uma ideia igual utilizando a ferramenta *Oracle Data Modeler* que se torna um hub para toda o design da Base de Dados e como visto na figura acima existiu a criação das tabelas da mesma maneira que a outra Base de Dados está modelada.

Assim apenas houve a criação das *constraints* e *foreign keys* mas também alguns extras que têm de ser feitos de forma mais manual como as sequências e *triggers* para os IDs serem auto incrementáveis e também a colocação do *timestamp* atual sempre que se introduz ou se altera uma linha da tabela associada, como é demonstrado em baixo um dos exemplos.

```
- Criação do "autoincrement" do country através da sequência e do seu trigger
.
CREATE SEQUENCE country_sq
START WITH 110;

CREATE OR REPLACE TRIGGER country_bi_trigger
BEFORE INSERT ON country FOR EACH ROW
BEGIN
    IF (:NEW.country_id IS NULL) THEN
        SELECT country_sq.nextval INTO :NEW.country_id
        FROM DUAL;
    END IF;
    -- aproveitar o trigger para adicionar o timestamp em modo ORACLE
    :NEW.last_update:=current_date;
END;
```

2.1.2. Migração dos Dados

Para a migração dos dados para a Base de Dados *Oracle* foi analisado as melhores maneiras para tal, sendo que se experimentou a passagem para um ficheiro *.csv* e depois fazer a importação utilizando uma ferramenta presente na administração e exploração da BD *Oracle* chamada de ***SQL*Loader*** mas concluímos que seria demasiado complicado fazer a conversão para poupar alguns segundos de povoamento.

Com tal, foi pensado na forma intermédia que trouxesse a velocidade de correr um *script* SQL mas em modo *batch*, algo também permitido pela Base de Dados Oracle colocando assim cerca de 1000 *inserts* por segundo, o que facilitou o povoamento e o tempo que demorou tornou-se significativamente menor.

Desta forma foi feita a transformação do povoamento dado pelos docentes que é utilizado pelo *MySQL* mas apenas trocando algumas situações utilizando o poder de ferramentas de manipulação de grandes textos com **expressões regulares**. Por exemplo para a colocação das datas, o *MySQL* deixa fazer os *inserts* encarregando-se ele de transformar para DATETIME, mas no caso do tipo DATE do Oracle, temos de utilizar a função *to_date(...,...)* para fazer a conversão correta, logo utilizando a expressão regular `,'\d{4}-\d{2}-\d{2} \d{2}:\d{2}:\d{2}'\)` para encontrar os *timestamps* e assim alterar ou eliminar consoante o caso. Outras nuances alteradas foi o caso do Oracle não dar para fazer um insert múltiplo, mas sim o comando INSERT só insere uma linha nova de valores, logo, teve de se fazer essa conversão também para funcionar no povoamento em Oracle. Com estas transformações foi possível aproveitar o *script* de povoamento original advindo do *MySQL* e colocar as alterações necessárias para povoar a nova Base de Dados relacional agora em *Oracle*.

2.1.3. *Queries* Experimentais

De forma a testar o funcionamento correto da Base de Dados foram selecionadas umas 10 *queries* que serão feitas de forma análogas nos três sistemas de Base de Dados presentes neste projeto. Desta forma temos as seguintes ideias para a pesquisa de informação útil em SQL:

1. Nome e Sobrenome de todos os Atores existentes.

```
SELECT
    first_name AS FirstName,
    last_name AS LastName
FROM actor
```

2. Lista dos Títulos dos Filme e respetivo Número de Atores que dele fazem parte.

```
SELECT
    film.title AS Title,
    count(*) AS NumberActors
```

```
FROM film

INNER JOIN film_actor ON film.film_id = film_actor.film_id

GROUP BY film.title
ORDER BY NumberActors DESC
```

3. Lista dos Nomes (Primeiro e Último) dos Atores que aparecem no Filme de nome "African Egg".

```
SELECT
    first_name AS FirstName,
    last_name AS LastName
FROM actor

INNER JOIN film_actor ON film_actor.actor_id = actor.actor_id
INNER JOIN film ON film.film_id = film_actor.film_id

WHERE film.title = 'AFRICAN EGG';
```

4. Lista dos Nomes (Primeiro e Último) e Email dos Clientes originais da Argentina

```
SELECT
    customer.first_name AS FirstName,
    customer.last_name AS LastName,
    customer.email AS Email
FROM customer

INNER JOIN address ON customer.address_id = address.address_id
INNER JOIN city ON address.city_id = city.city_id
INNER JOIN country ON city.country_id = country.country_id

WHERE country.country = 'Argentina'
```

5. Lista dos 5 primeiros Géneros/Categorias e sua respetiva Receita Bruta, por ordem descendente

```
SELECT category.name AS Category, SUM(payment.amount) AS TotalVendas
FROM category
```

```
INNER JOIN film_category ON category.category_id = film_category.catego
ry_id
INNER JOIN film ON film_category.film_id = film.film_id
INNER JOIN inventory ON film.film_id = inventory.film_id
INNER JOIN rental ON inventory.inventory_id = rental.inventory_id
INNER JOIN payment ON rental.rental_id = payment.rental_id

GROUP BY category.name
ORDER BY TotalVendas DESC
FETCH FIRST 5 ROWS ONLY;
```

6. Lista dos Filmes alugados com mais frequência, por ordem decrescente.

```
SELECT film.title AS Title, Rental.NumberRented AS CountRented
FROM film

INNER JOIN (
    SELECT inventory.film_id AS idFilm, count(rental.rental_id) AS Numb
erRented
    FROM rental

    INNER JOIN inventory ON rental.inventory_id = inventory.inventory_i
d

    GROUP BY inventory.film_id
) Rental
ON film.film_id = Rental.idFilm

ORDER BY Rental.NumberRented DESC
```

7. Total de cópias em Inventário do filme de nome "Connecticut Tramp".

```
SELECT film.title AS Title, count(*) TotalCopys
FROM film

INNER JOIN inventory ON film.film_id = inventory.film_id

WHERE film.title = 'CONNECTICUT TRAMP'
GROUP BY film.title;
```

8. Lista dos Nomes (Primeiro e Último) dos Clientes e o total pago por cada um deles ao sistema em si, ordenados alfabeticamente consoante o Primeiro Nome.

```
SELECT customer.first_name AS FirstName, customer.last_name AS LastName
, sum(payment.amount) AS TotalPago
FROM payment

INNER JOIN customer ON payment.customer_id = customer.customer_id

GROUP BY customer.first_name, customer.last_name
ORDER BY customer.FIRST_NAME;
```

9. Lista de todos os Filmes da Categoria "Action", bem como seu Ano de Lançamento e Rating.

```
SELECT title AS Title, release_year AS ReleaseYear, rating AS Rental
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

WHERE category.name = 'Action'
ORDER BY Title;
```

10. Lista das Cidades mais populares em termos de pagamentos por parte dos seus moradores.

```
SELECT country.country AS Name, count(payment.payment_id) AS TotalPayme
nts
FROM country

INNER JOIN city ON city.country_id = country.country_id
INNER JOIN address ON address.city_id = city.city_id
INNER JOIN customer ON customer.address_id = address.address_id
INNER JOIN payment ON payment.customer_id = customer.customer_id

GROUP BY country.country
ORDER BY TotalPayments DESC;
```

2.2 SBD MongoDB

No que diz respeito ao Sistema Não Relacional MongoDB, pensaram-se em três coleções principais, que representariam a base de povoamento deste novo modelo. Dessa forma, inseriu-se o máximo de informação possível em cada *collection*, eliminando-se as ligações em termos de tabelas MySQL.

2.2.1 Esquema da Base de Dados

O esquema da Base de Dados é apresentação abaixo sob forma de imagem, que permitem validar os *fields* criados para cada coleção de documentos. As imagens correspondem apenas a um protótipo da organização da informação, não tenho sido usadas para se efetuar a inserção em si.

- Modelo pensado para a *Collection* Customer:

- Modelo pensado para a *Collection* Store e Film:

```
{
    "Manager First Name": String,
    "Manager Last Name": String,
    "Manager Id": ObjectId(),
    "Address": String,
    "City": String,
    "Country": String,
    "Postal Code": String,
    "Phone": Number,
    "Staff":[
        {
            "Id": Number,
            "First Name": String,
            "Last Name": String,
            "Address": String,
            "City": String,
            "Country": String,
            "Postal Code": String,
            "Phone": Number
        }
    ],
    "Inventory":[
        {
            "Id": ObjectId(),
            "Filme Id": Number,
            "Film Title": String
        }
    ]
}
```

```
{
    "Film Id": ObjectId(),
    "Title": String,
    "Description": String,
    "Release Year": Date(),
    "Original Language": String,
    "Language": String,
    "Rental Duration": Number,
    "Rental Rate": Number,
    "Rating": String,
    "Length": Number,
    "Replacement Cost": Number,
    "Special Features": String,
    "Category": String,
    "Actors":[
        {
            "Id": ObjectId(),
            "First Name": String,
            "Last Name": String,
        }
    ]
}
```

2.2.2 Migração dos Dados

Para efetuar a migração de MySQL para MongoDB foram criados dois *scripts* em NodeJS, um para fazer a extração dos campos necessários do MySQL em formato .json e outra para fazer a inserção no MongoDB seguindo então os modelos definidos anteriormente.

A ideia destes dois ficheiros funcionais foi tornar mais prático todo o processo de estruturação/povoação do modelo de Base de Dados, criando uma ligação ao servidor do MongoDB praticamente instantânea desde o processo de extração de informação vinda do MySQL até à parte em que se insere efetivamente toda a informação pelo uso dos ficheiros JSON criados.

2.2.3 *Queries* Experimentais

1. Nome e Sobrenome de todos os Atores existentes.

```
db.getCollection('customers').find(
    {},
    {
        "_id": 0,
        "First Name": 1,
        "Last Name": 1
    }
)
```

2. Lista dos Títulos dos Filme e respetivo Número de Atores que dele fazem parte.

```
db.getCollection("films").aggregate(
    [
        {
            "$project" : {
                "_id" : 0.0,
                "Title" : 1.0,
```

```
                "NumberActors" : {"$size" : "$Actors"}
            }
        }
    ]
)
```

3. Lista dos Nomes (Primeiro e Último) dos Atores que aparecem no Filme de nome "African Egg".

```
db.getCollection("films").find(
    {"Title" : "AFRICAN EGG"},
    {
        "_id" : 0,
        "Actors.First Name" : 1,
        "Actors.Last Name" : 1
    }
)
```

4. Lista dos Nomes (Primeiro e Último) e Email dos Clientes originais da Argentina.

```
db.getCollection("customers").find(
    {"Country" : "Argentina"},
    {
        "First Name" : 1,
        "Last Name" : 1,
        "Email" : 1
    }
)
```

5. Total de cópias em Inventário do filme de nome "Connecticut Tramp".

```
db.stores.aggregate([
    { $unwind: "$Inventory" },
    { $match: {"Inventory.Title":"CONNECTICUT TRAMP"} },
    { $group: {_id: "$Inventory" , TotalCopys:  {$sum: 1}} }
])
```

6. Lista dos Nomes (Primeiro e Último) dos Clientes e o total pago por cada um deles ao sistema em si, ordenados alfabeticamente consoante o Primeiro Nome.

```
db.customers.aggregate([{ "$project": {"First Name": 1,"TotalSpent"
: {"$sum": {$toInt: {"$Payments.Amount"}}}}])
```

7. Lista de todos os Filmes da Categoria "Action", bem como seu Ano de Lançamento e Rating.

```
db.films.find({Category: "Action"}, {"Title": 1,"Release year": 1 ,
 Rating: 1, _id:0})
```

2.3 SBD em Grafos Neo4J

Estando modelada a Base de Dados Sakila em modo *Oracle* e em MongoDB, é hora de começar a planear a fase final do projeto. Fala-se das Bases de Dados orientadas a Grafos, um sistema totalmente avançado e prático de se usar, que traz consigo uma forma diferente de trabalhar com os dados.

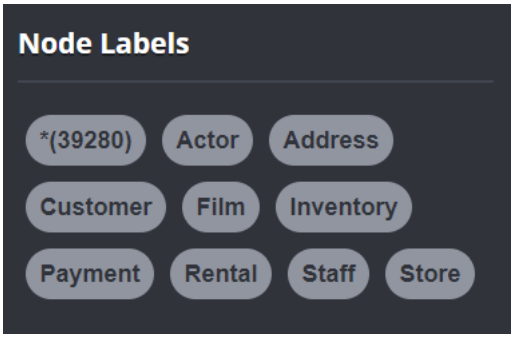
Esta nova forma de visualização/manipulação oferece muitas aplicações práticas e isso ficou claro no desenvolvimento deste último esquema de Base de Dados, que será explicado nas subalíneas que se seguem.

2.3.1 Esquema da Base de Dados

Através da análise profunda do modelo lógico fornecido pelo MySQL, criou-se logo um esboço inicial daquilo que poderiam representar os grafos e as suas respetivas ligações. De modo a se entender melhor as escolhas tomadas, vai ser feita uma explicação de cada *Nodo Label* e *Relationship Type* existentes para o modelo.

- Lista de Nodos criados:

Note-se que apenas as tabelas **city** e **country** foram ignoradas como candidatas a nodos. Todas as restantes tabelas (à exceção das tabelas **film_actor**, **film_category**, **film_text**) foram "convertidas" em nodos.

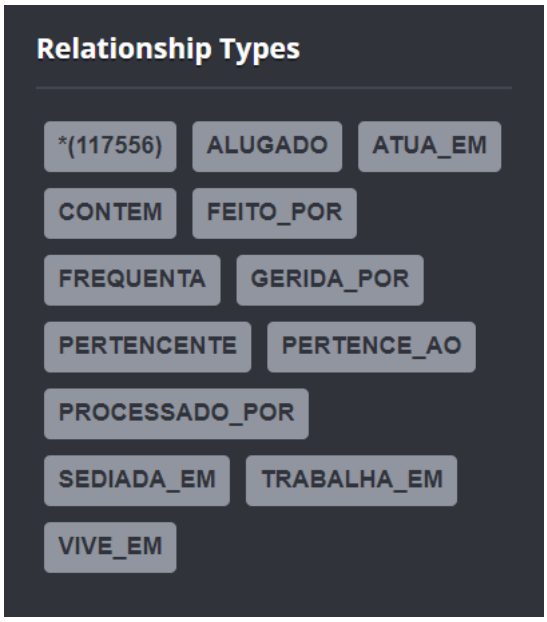


Dessa forma, colocou-se a informação relativa à City e ao Country no nodo Address, já que a única informação em tabela MySQL era o nome de ambos e que em termos de Neo4J não faria sentido estar-se a criar mais dois nodos e mais um tanto de relacionamentos para cada Customer. Outra decisão tomada foi colocar a informação da Language e da Category dentro do próprio Film, poupando-se também ao nível dos MATCH e da quantidade de nodos existentes no Grafo em geral.

Ao fazer-se isso poupa-se em termos de tempo de migração de dados e queries que necessitem desta informação, que fica logo acessível apenas pela recorrência ao nodo Address.

- Lista de Relações criadas:

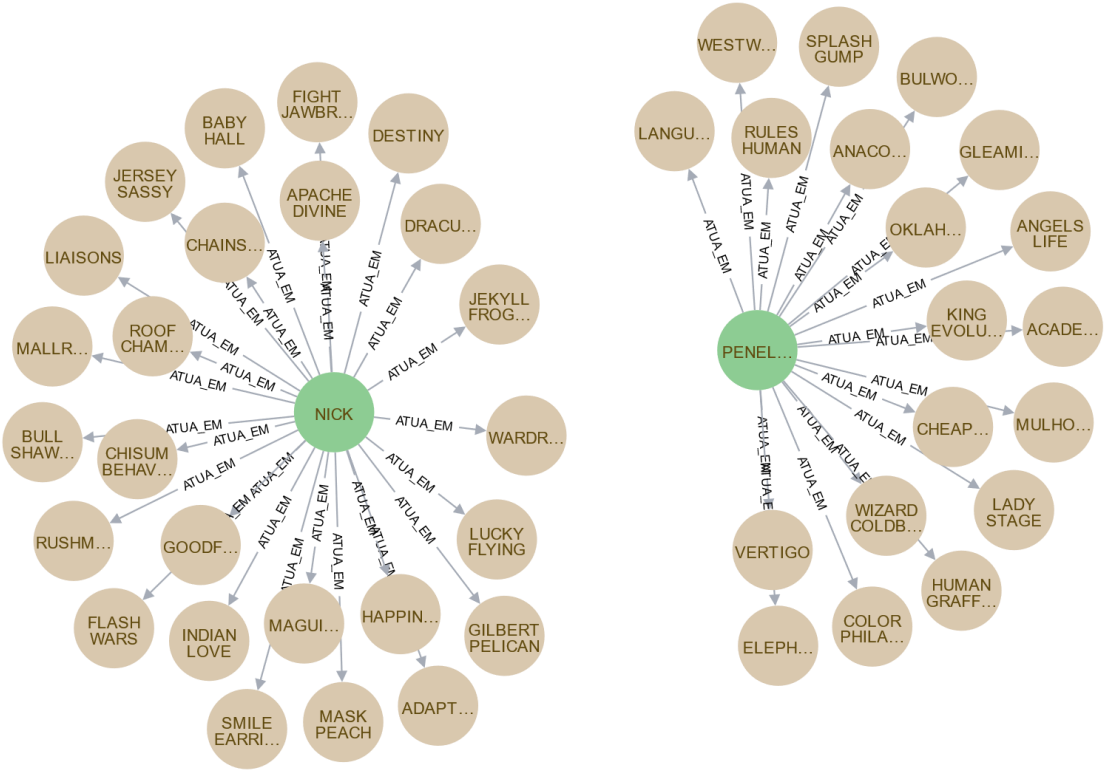
A criação de relacionamentos entre os nodos abordados anteriormente, teve um processo de pensamento muito próprio. Com a Base de Dados MySQL a correr e com o seu *schema* mentalizado, a ideia passou por verificar cada tabela e registar as ligações nela existentes através das chaves primárias e chaves estrangeiras.



Para cada uma das relações vai ser anexada uma imagem ilustrativa. Para obter cada um desses resultados limitou-se a quantidade de dados a 50 (menor em certos casos), uma vez que apenas é necessário compreender a relação em si entre os nodos.

1. Relacionamento Actor-Film

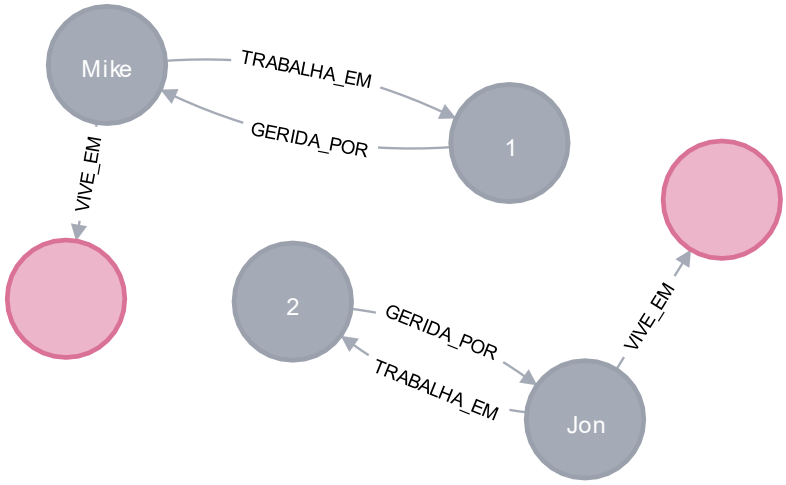
Uma vez que há uma única relação existente entres estes dois nodos, é necessário estabelecer os Atores que atuam num determinado Filme. Por essa razão existe o relacionamento **ATUA_EM** que permite de imediato obter a lista de Atores para cada um dos nodos **Film**.



2. Relacionamento Staff-Address e Staff-Store

Correspondem aos tipos **VIVE_EM** e **TRABALHA_EM**. Este relacionamento permite facilitar o acesso ao **Staff** de cada **Store** no que toca ao **Address** de ambos esses nodos.

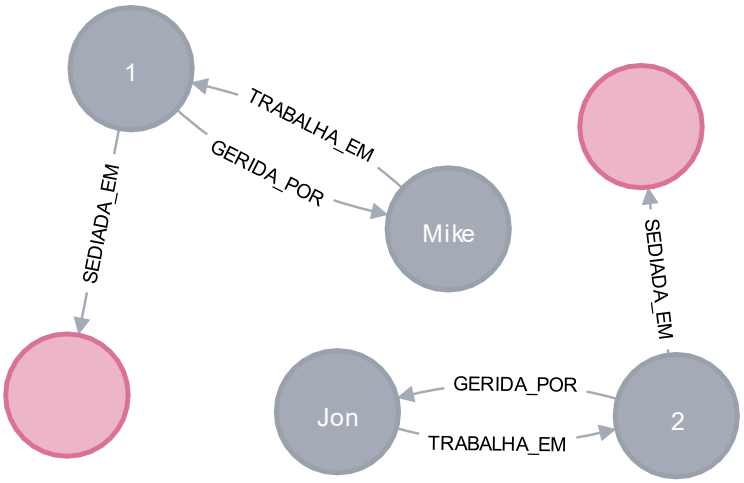
Ainda que para a Sakila apenas existam apenas 2 membros de **Staff**, é necessário pensar-se sem termos futuros, já que pode existir a inserção de novos membros e com isso possíveis queries para extrair informações acerca destes dois nodos.



3. Relacionamento Store-Address e Store-Staff

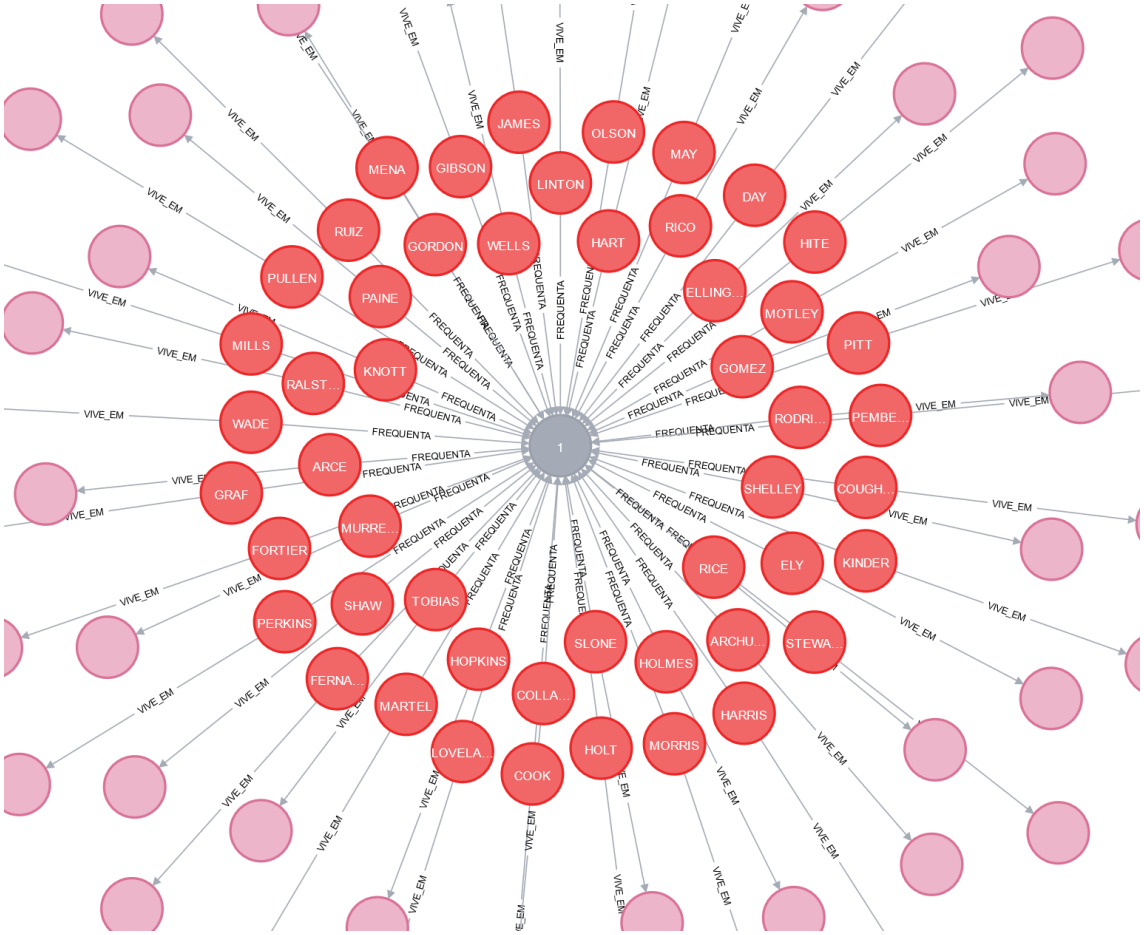
Correspondem aos tipos **SEDIADA_EM** e **GERIDA_POR**. Facilmente se entende a necessidade deste relacionamento pela observação da tabela MySQL referente à **Store**. Nela verificamos que existe uma ligação mantida com a tabela **address** e **staff**, pelo uso das chaves estrangeiras **manager_staff_id** e **address_id**.

No Neo4J isso transpõe-se para um relacionamento que parte do nodo **Store** e vai em direção tanto ao nodo **Address** como também ao nodo **Staff** (duas ligações em separado).



4. Relacionamento Customer-Address e Customer-Store

Correspondem aos tipos **VIVE_EM** e **FREQUENTE**. Esta ligação é mais do que necessária, tendo em conta que só assim estabelecemos o contacto direito do **Customer** com as **Stores** que frequenta, conseguindo com isso estabelecer *queries* no que diz respeito aos vários pagamentos e alugers feitos pelo **Customer** numa determinada **Store**.

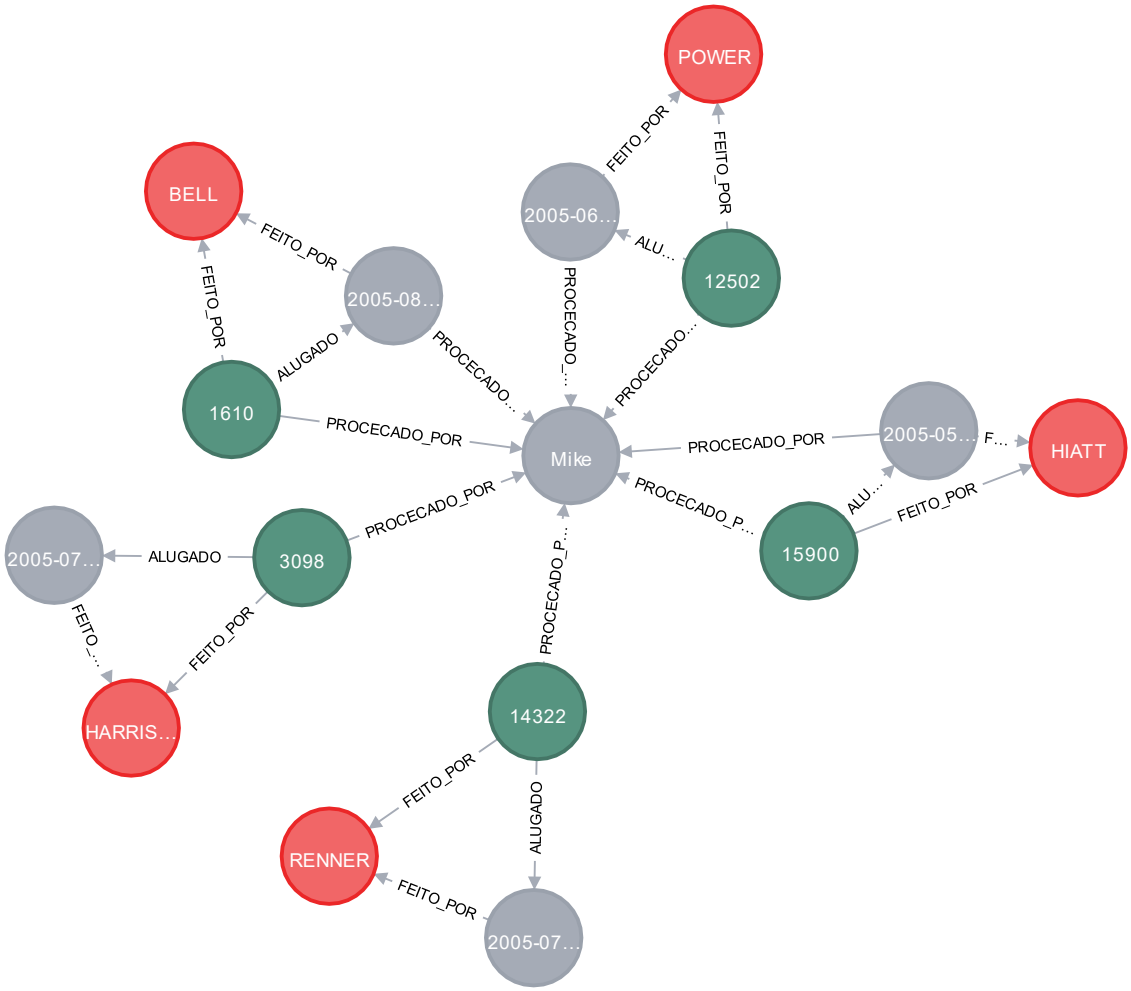


Na imagem supracitada notamos a essência por detrás destes dois relacionamento. Neste caso, refere-se a um **Store** das duas existentes, demonstrando perfeitamente a ideia de que cada **Customer** possui o seu **Address** e frequenta a **Store** em causa.

5. Relacionamento Payment-Customer, Payment-Staff, Payment-Rental

Correspondem aos tipos **FEITO_POR**, **PROCESSADO_POR** e **ALUGADO**. Estas três ligações são as mais demoradas em termos de *insert* da informação. Isso deve-se ao facto de que se definiu o **Payment** como sendo um nodo e dessa forma cria-se uma grande quantidade de informação quando se está a inserir os pagamentos referentes a todos os clientes.

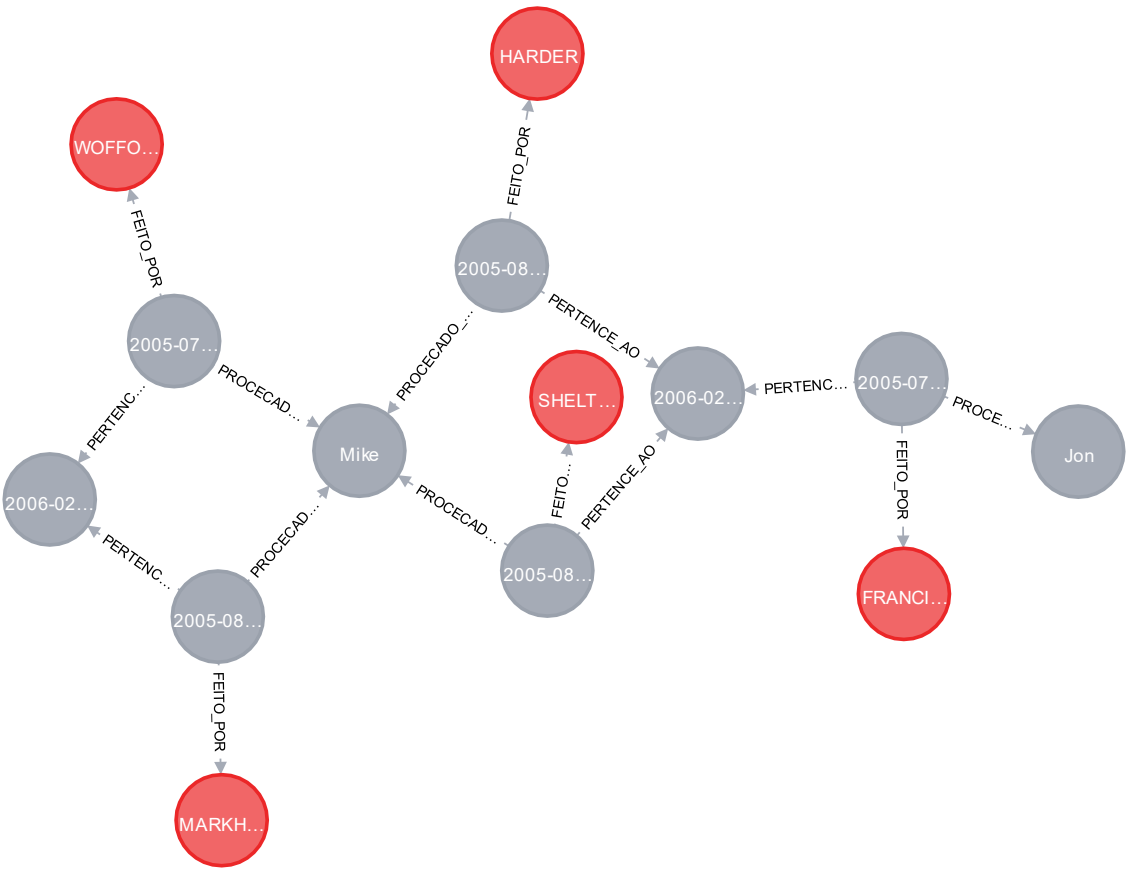
Apesar disso, o grupo achou ser a decisão mais acertada porque pensou no nível de facilidade em termos de futuras *queries* experimentais. Basta pensar no exemplo em que queremos retirar a quantidade de pagamentos efetuados por um determinado cliente - Com apenas um **MATCH** resolveríamos esta questão.



6. Relacionamento Rental-Customer, Rental-Staff, Rental-Inventory

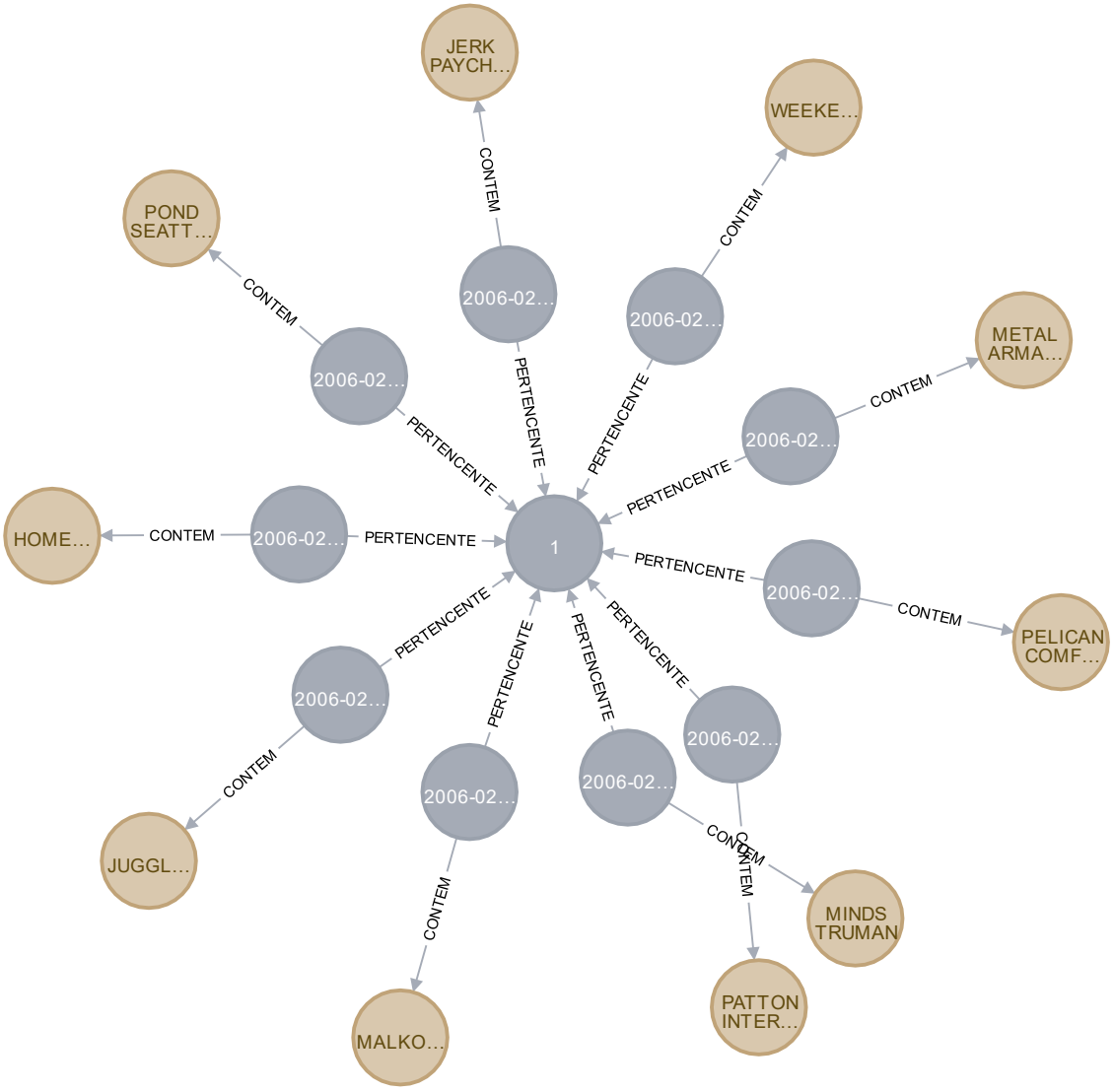
Igualmente ao que acontece para os dados relativos aos pagamentos, o estabelecimento destes três relacionamentos são igualmente demorados, também pelo facto de representarem o maior volume de informação. Correspondem aos tipos FEITO_POR, PROCESSADO_POR e PERTENCE_AO.

Dois destes relacionamentos são análogos aos usados para o nodo do Payment.



7. Relacionamento Inventory-Film, Inventory-Store

Correspondem aos tipos **CONTEM** e **PERTENCENTE.** Permitem mapear o conjunto de filmes que fazem parte do inventário e associar um determinado inventário a cada uma das duas lojas existentes.



2.3.2 Migração dos Dados

Para proceder à migração de dados foi usado um pensamento similar ao Sistema de Base de Dados MongoDB, onde existe um script escrito na linguagem NodeJS, que trata de executar um conjunto de Queries que correm internamente em MySQL e devolvem a informação consoante aquilo que é pedido. A ideia passa então por obter toda a informação que depois é guardada em ficheiros *.csv* e exportada no próprio *software* do Neo4J Desktop.

Como seria de esperar, é necessário existir na fase inicial o *insert* de toda a informação referente aos nodos em si e só após isso ser feito, é que podem ser criadas todos os relacionamentos explicados anteriormente.

Assim, existe um ficheiro em modo .txt que contém todas as *queries* que devem ser corridas uma a uma para se conseguir obter o Sistema de Base de Dados Neo4J totalmente funcional.

2.3.3 *Queries* Experimentais

O resultado das *queries* criadas para teste permitem provar a operacionalidade do Sistema de Base de Dados em causa.

8. Nome e Sobrenome de todos os Atores existentes.

```
MATCH (actor:Actor)
RETURN actor.FirstName AS FirstName, actor.LastName AS LastName
```

9. Lista dos Títulos dos Filme e respetivo Número de Atores que dele fazem parte.

```
MATCH (actor:Actor)-[:ATUA_EM]->(film:Film)
RETURN film.Title AS Title, count(actor) AS NumberActors
```

10. Lista dos Nomes (Primeiro e Último) dos Atores que aparecem no Filme de nome "African Egg".

```
MATCH (actor:Actor)-[:ATUA_EM]->(film:Film)
WHERE film.Title = "AFRICAN EGG"
RETURN distinct(actor.FirstName) AS FirstName, actor.LastName
AS LastName
```

11. Lista dos Nomes (Primeiro e Último) e Email dos Clientes originais da Argentina.

```
MATCH (customer:Customer)-[:VIVE_EM]->(address:Address)
WHERE address.Country = "Argentina"
RETURN customer.FirstName AS FirstName, customer.LastName
AS LastName, customer.Email AS Email
```

12. Lista dos 5 primeiros Géneros/Categorias e sua respetiva Receita Bruta, por ordem descendente.

```
MATCH (film:Film)<-[:CONTEM]-(inventory:Inventory)<-[:PERTENCE_AO]-
(rental:Rental)<-[:ALUGADO]-(payment:Payment)

RETURN film.Category AS Category, sum(toFloat(payment.Amount)) AS T
otalVendas
ORDER BY TotalVendas DESC
LIMIT 5
```

13. Lista dos Filmes alugados com mais frequência, por ordem decrescente.

```
MATCH (film:Film)<-[:CONTEM]-(inventory:Inventory)<-[:PERTENCE_AO]-
(rental:Rental)

RETURN film.Title AS Title, count(rental.idRental) AS CountRented
ORDER BY CountRented DESC
```

14. Total de cópias em Inventário do filme de nome "Connecticut Tramp".

```
MATCH (inventory:Inventory)-[:CONTEM]->(film:Film)
WHERE film.Title = "CONNECTICUT TRAMP"

RETURN film.Title AS Title, count(film.idFilm) AS TotalCopys
```

15. Lista dos Nomes (Primeiro e Último) dos Clientes e o total pago por cada um deles ao sistema em si, ordenados alfabeticamente consoante o Primeiro Nome.

```
MATCH (customer:Customer)<-[:FEITO_POR]-(payment:Payment)

RETURN customer.FirstName AS FirstName, customer.LastName AS LastNa
me, sum(toFloat(payment.Amount)) AS TotalPago
ORDER BY FirstName
```

16. Lista de todos os Filmes da Categoria "Action", bem como seu Ano de Lançamento e Rating.

```
MATCH (film:Film)
WHERE film.Category = "Action"

RETURN film.Title AS Title, film.ReleaseYear AS ReleaseYear, film.R
ating AS Rating
ORDER BY Title
```

17. Lista dos Países mais populares em termos de pagamentos por parte dos seus moradores.

```
MATCH (address:Address)<-[:VIVE_EM]-(customer:Customer)<-
[:FEITO_POR]-(payment:Payment)
```

```
RETURN address.Country AS Name, count(payment.idPayment) AS TotalPa
yments
ORDER BY TotalPayments DESC
```

4. Conclusões e Críticas Finais

Estando toda a parte prática da Unidade Curricular totalmente compreendida e trabalhada, é hora de redigir as observações finais. Tendo em conta a dimensão deste trabalho prático, foi necessário ter um pensamento bem definido de como iriam ser estruturados os três diferentes modelos de Base de Dados, para que todos eles ficassem operacionais consoante a *database* Sakila fornecida pelo MySQL.

A maior dificuldade esteve no SDB MongoDB, dado que a forma como depois as *queries* teriam de ser feitas revelou-se bastante mais complexa comparativamente ao Oracle ou até mesmo Neo4J. Apesar da dificuldade inerente em passar as informações duma Base de Dados relacional para uma Base de Dados baseada em documentos JSON, acontece sempre a situação sobre que informação e dados são relevantes e quais podem ser redundantes para a melhor compreensão das futuras *queries* que possam existir. Por conseguinte as escolhas sobre o *design* e estrutura dos documentos JSON a alocar as informações advindas do *Sakila* foram feitas a pensar num maior leque de tipo de pesquisa e dividido nas três grandes componentes/coleções que são os filmes, clientes e lojas.

Contudo, a dificuldade foi ultrapassada e como este projeto demonstra e exemplifica, existiu um pormenor detalhado sobre como trazer da melhor forma a Base de Dados em estudo para estes sistemas díspares entre si, mas que nos ensinam por força da prática as suas grandes vantagens e principais limitações comparativamente aos outros estudados também, tomando assim esta aprendizagem como um teste sobre a importância da escolha do tipo de Base de Dados para os dados que se necessite de guardar.