O que é mongoDB?

O mongoDB é um tipo de base de dados orientado a documentos, diferente das bases de dados tradicionais que seguem o modelo relacional. É o mais famoso *NoSQL* do mercado *open source* e é escrito em *C++*.

Este modelo de dados foi construído para suportar alto rendimento em processos de leitura e escrita e grande velocidade no processamento de queries de dados. No entanto, quer seja necessário apenas um nó de base de dados ou dezenas deles, o mongoDB consegue providenciar uma boa performance na sua escalabilidade. No entanto, o brilho desta ferramenta encontra-se no facto de ser um modelo de dados intuitivo que consegue lidar com grandes estruturas de dados. Para além disso, o facto de ser orientado para o armazenamento de dados em documentos *BSON* com esquema dinâmico, significa que o administrador pode guardar os seus registos sem ter a preocupação de organizar as estruturas de dados (como o número de campos ou tipos de campos para armazenar valores).

**Porquê MongoDB?**

Ao contrário do gestor de bases de dados relacionais SQL que estrutura os seus dados em tabelas interligadas, o mongoDB guarda os seus dados em documentos que podem ser alterados simplesmente adicionando novos campos ou excluindo existentes, tornando o mongoDB útil para representar relações hierárquicas, armazenamento de matrizes e outras estruturas mais complexas, de uma maneira mais simples e intuitiva.

Outra diferença fundamental é que bases de dados orientados a documentos não fornecem relacionamentos estritos entre documentos, o que ajuda a manter o seu design sem esquemas. Isso difere muito de bases de dados relacionais, que dependem fortemente de relacionamentos para normalizar o armazenamento de dados. Em vez de armazenar dados relacionados numa área de armazenamento separada, em bases de dados de documentos, estas são integradas ao próprio documento. Isto é muito mais rápido do que armazenar uma referência a um outro documento onde os dados relacionados se encontram armazenados, uma vez que cada referência exigiria uma consulta adicional. Contudo, se for preciso armazenar dados relacionados separadamente, é possível faze-lo facilmente no mongoDB usando uma coleção separada.

**Migração do modelo relacional SQL para o modelo não relaciona mongoDB: modelação dos dados**

O mongoDB agrupa documentos em várias coleções, não impondo um esquema restrito: *Schema-less.* Na prática, significa que os documentos de uma mesma coleção poderão ter estruturas completamente diferentes uns dos outros, e os campos que são comuns entre esses documentos também poderão conter diferentes tipos de dados. Em contrapartida, já foi visto que num modelo relacional isto não acontece, sendo que todos os registos que figuram numa mesma tabela deverão ter a mesma estrutura e conter o mesmo tipo de dados.

A título exemplificativo, a inserção de documentos com a estrutura seguinte numa mesma coleção é completamente possível:

****

Figure 11- Diferentes estruturas para documentos semelhantes

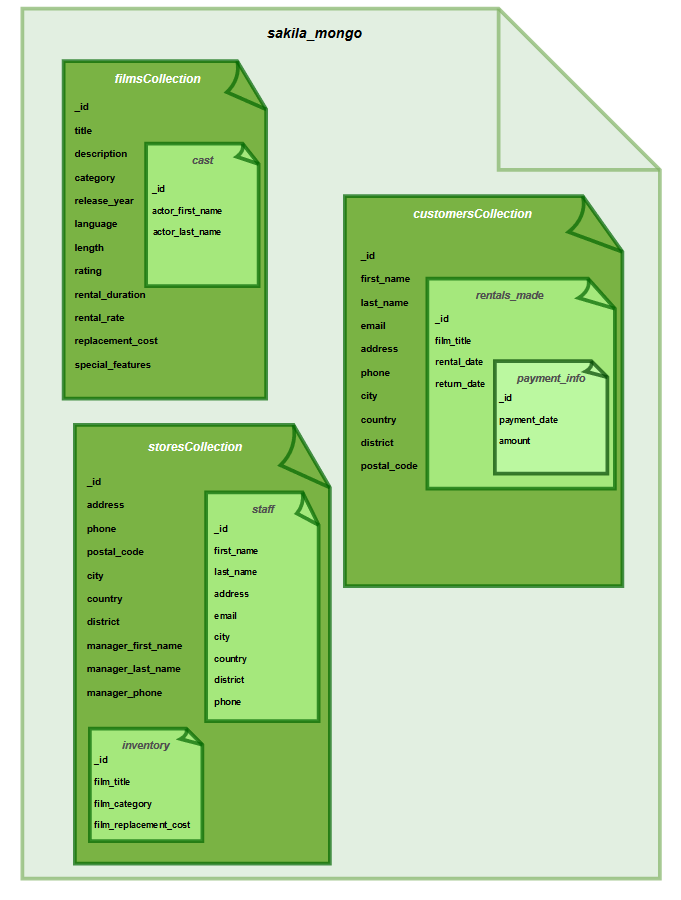
Este fator de “não imposição” de um esquema para a inserção dos dados em mongoDB confere-lhe algumas vantagens, entre as quais destacam-se:

* a estrutura dos dados é determinada pela codificação da aplicação que se suporta na base de dados, e não pela base de dados em si (ao contrário do que acontece numa base de dados relacional).
* a fase inicial do desenvolvimento de uma aplicação que se suporta numa base de dados mongoDB é acelerada, pois neste momento os esquemas mudam frequentemente e o mongoDB lida bem com isso.
* os esquemas flexíveis para os dados permitem modelá-los genuinamente nas suas propriedades, e não apenas adaptá-los a um esquema que tem que ser seguido.

Posto isto, o primeiro passo na migração da base de dados relacional SQL para este novo modelo de bases de dados consiste em delinear todas as coleções que serão criadas, tendo em conta que:

* o esquema para este novo modelo segue-se pelas tabelas existentes do modelo SQL e deve satisfazer os mesmos requisitos funcionais.
* se certos objetos serão usados em conjunto, a melhor solução será juntá-los no mesmo documento (garantindo que não existe necessidade de efetuar *joins*).
* é aceitável a duplicação de dados (de maneira contenciosa) uma vez que espaço em disco é mais barato comparado com tempo de computação - ao existirem dados duplicados é mais rápido e menos custoso efetuar pesquisas/alterações na base de dados.

**Esquematização das coleções da base de dados mongoDB**



Á primeira vista, um olhar sobre as tabelas existentes na base de dados do modelo relacional SQL e sobre as coleções criadas na base de dados não relacional mongoDB evidencia claramente que, neste segundo modelo, os dados se encontram mais aglutinados/agrupados. Tal se deve ao facto de no mongoDB não existirem *joins* e, como tal, os dados que pretendemos relacionar devem-se encontrar juntos no mesmo documento.

Após algum tempo despendido na análise e estrutura da base de dados relacional SQL fornecida, tornou-se evidente e claro a existência de 3 grandes unidades fundamentais de dados: a unidade de dados dos filmes que compõe o inventário de uma loja e toda a informação relacionada com eles (como a informação dos atores de cada filme), a unidade dos clientes que alugam filmes e informação relacionada com eles (como dados acerca dos alugueres efetuados e respetivos pagamentos de cada um deles), e também a unidade de dados das próprias lojas que contêm um inventário de filmes e um conjunto de funcionários que nelas trabalham.

Sendo assim, idealizaram-se, por isso, 3 grandes coleções constituintes da base de dados não relacional *sakila\_mongo*:

* filmsCollection
* customersCollection
* storesCollection
* **filmsCollection**

Esta coleção procura representar todas as informações úteis e necessárias de cada filme, e por isso contém os seguintes campos:

* **title:** indica o título do filme
* **description:** indica uma breve descrição do assunto do filme
* **category:** indica o tipo de categoria na qual o filme se insere (action, comedy, horror, etc.)
* **release\_year:** indica o ano de lançamento do filme
* **length:** indica a duração do filme
* **rating:** indica a classificação do filme
* **rental\_duration:** indica o tempo de aluguer previsto para o filme
* **rental\_rate:** indica o rácio de aluguer do filme
* **replacement\_cost:** indica o custo de substituição de um filme
* **cast:** documento embebido que contém informação relevante acerca dos atores que figuram no filme
  + **actor\_first\_name:** indica o primeiro nome do ator
  + **actor\_last\_name:** indica o apelido do ator
* **customersCollection**

Esta coleção procura representar todas as informações úteis e necessárias de cada cliente das lojas de filmes, e por isso contém os seguintes campos:

* **first\_name:** indica o primeiro nome do cliente
* **last\_name:** indica o apelido do cliente
* **email:** indica o email do cliente
* **address:** indica a rua de residência do cliente
* **phone:** indica o contacto telefónico do cliente
* **city:** indica a cidade de residência do cliente
* **country:** indica o país de residência do cliente
* **district:** indica o distrito de residência do cliente
* **postal\_code:** indica o código postal de residência do cliente
* **rentals\_made:** documento embebido que contém informação relevante acerca dos alugueres feitos pelo cliente
  + **film\_title:** indica o título do filme alugado pelo cliente
  + **rental\_date:** indica a data de aluguer do filme pelo cliente
  + **return\_date:** indica a data de devolução do filme alugado pelo cliente
  + **payment\_info:** documento embebido que contém informação relevante acerca do pagamento de cada aluguer efetuado pelo cliente (útil caso o pagamento do aluguer fosse repartido pelo momento de requisição e devolução do filme)
    - **payment\_date:** indica a data de pagamento do aluguer do filme
    - **amount:** indica o valor do pagamento do aluguer do filme
* **storesCollection**

Esta coleção procura representar todas as informações úteis e necessárias de cada loja de aluguer de filmes, e por isso contém os seguintes campos:

* **address:** indica a rua da loja de aluguer de filmes
* **phone:** indica o contacto telefónico da loja de aluguer de filmes
* **city:** indica a cidade da loja de aluguer de filmes
* **country:** indica o país da loja de aluguer de filmes
* **district:** indica o distrito da loja de aluguer de filmes
* **manager\_first\_name:** indica o primeiro nome do gerente da loja de aluguer de filmes
* **manager\_last\_name:** indica o apelido do gerente da loja de aluguer de filmes
* **manager\_phone:** indica o contacto telefónico do gerente da loja de aluguer de filmes
* **staff:** documento embebido que contém informação relevante acerca dos trabalhadores da loja de aluguer de filmes
  + **first\_name:** indica o primeiro nome do trabalhador da loja
  + **last\_name:** indica o apelido do trabalhador da loja
  + **address:** indica a rua de residência do trabalhador da loja
  + **email:** indica o email do trabalhador da loja
  + **city:** indica a cidade de residência do trabalhador da loja
  + **country:** indica o país de residência do trabalhador da loja
  + **district:** indica o distrito de residência do trabalhador da loja
  + **phone:** indica o contacto telefónico do trabalhador da loja
* **inventory:** documento embebido que contém informação relevante acerca do inventário de filmes presente na loja de aluguer de filmes
  + **film\_title:** indica o título do filme que faz parte do inventário da loja
  + **film\_category:** indica a categoria do filme que faz parte do inventário da loja
  + **film\_replacement\_cost:** indica o custo de substituição do filme que faz parte do inventário da loja

Note-se que não existe uma configuração ótima acerca do número de coleções e também dos campos de cada coleção, uma vez que a base de dados é construída mediante o tipo de queries que poderão ser feitas. Desta forma, um esquema pode ser ótimo e suficiente para um determinado tipo de interrogações feitas à base de dados, mas, no entanto, poderá ser falível e não suficiente para suportar outro tipo de questões.

No desenho do esquema proposto, procurou-se que a base de dados criada fosse generalizada quanto às questões a que permite responder e fosse o mais completa possível face ao modelo relacional na qual se baseia.

Ainda assim, mesmo que, eventualmente, haja uma interrogação ou outra que para a configuração proposta não seja possível dar uma resposta, o problema não é grande uma vez que neste tipo de base de dados é muito fácil e “imediato” adaptá-la de forma a conseguir dar uma resposta à interrogação apresentada.

**Migração do modelo relacional SQL para o modelo não relacional mongoDB: migração dos dados**

Para realizar esta etapa crucial na transição de um modelo de bases de dados para o outro utilizou-se um *script* na linguagem de programação *Python* que nos permitiu migrar os dados/tabelas que se encontravam na base de dados SQL (previamente povoada) para o mongoDB.

De uma forma simplista, pode-se dizer que o *script* basicamente “pega” nos dados do modelo SQL, converte-os numa sintax aceite pelo mongoDB e insere-os.

Para ser alcançado tal feito, o *script* trata de realizar 3 funções essenciais:

* Conexão à base de dados (ao servidor) MySQL e mongoDB

Esta parte é responsável por estabelecer a conexão a ambas as bases de dados*.*

Por um lado, à base de dados MySQL (que se deve encontrar povoada) que será a fonte dos dados para a base de dados mongoDB a criar, e por outro lado estabelecer a conexão à base de dados mongoDB para que efetivamente se consiga estruturá-la, preenche-la e implementá-la.

* Traduzir e transferir os dados do modelo SQL para o modelo mongoDB

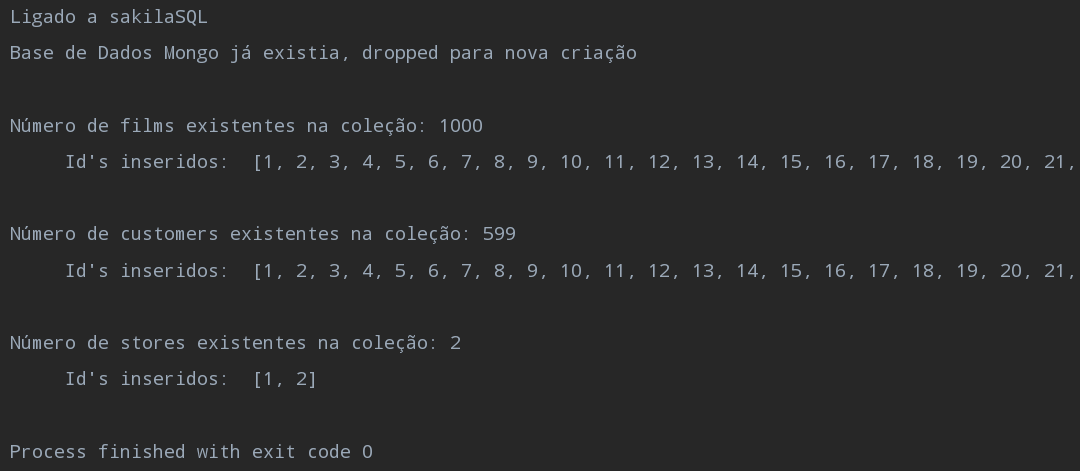
Esta é parte fulcral do *script* de migração, na medida em que é nela que faz a “tradução” e a inserção dos dados. Durante este processo, várias queries SQL são executadas de forma a que os dados pretendidos sejam extraídos da base de dados relacional. À medida que este processo de extração dos dados é feito, os documentos das diferentes coleções vão sendo criados, estruturados e povoados com os dados que vão sendo extraídos.

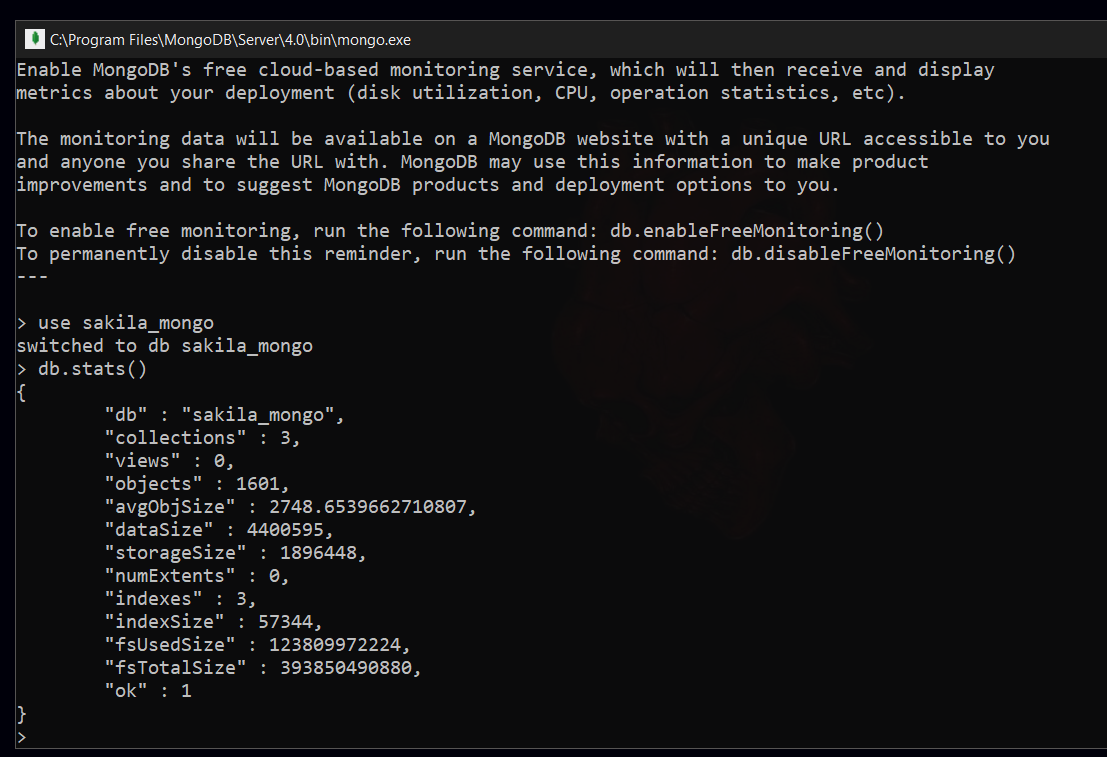
* Inserir os documentos *BSON* criados no modelo mongoDB

Esta última parte do *script* insere os documentos *BSON* provenientes do passo anterior e insere-os na base de dados mongoDB, sendo este o momento em que as diferentes coleções são criadas e povoadas.

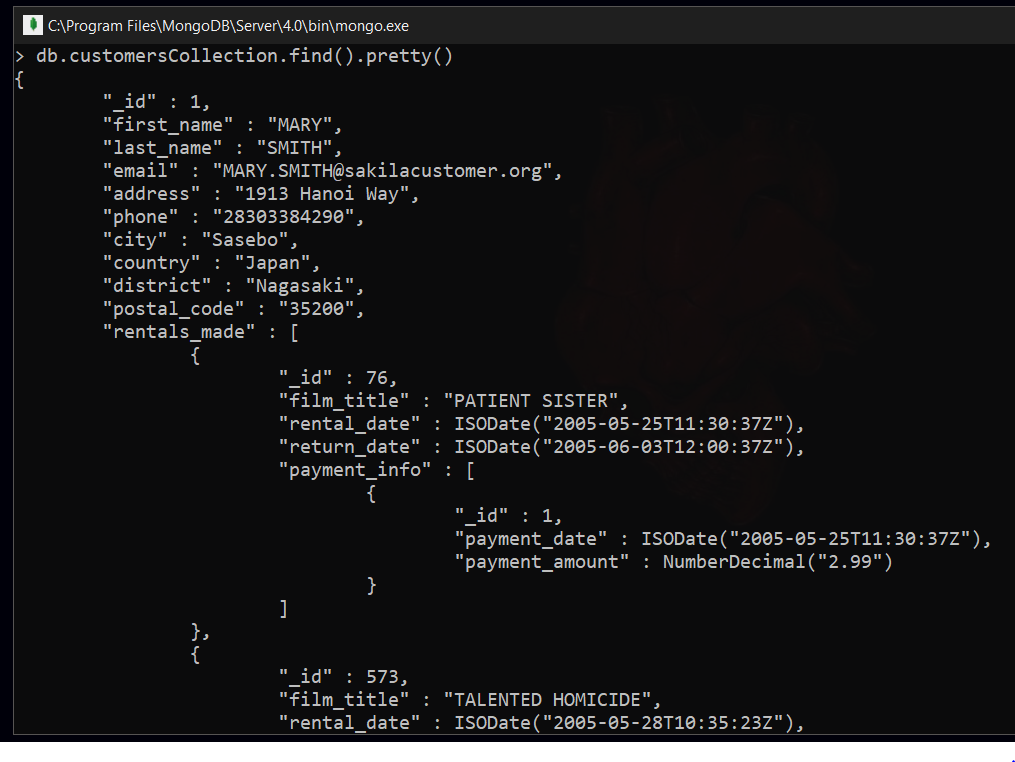
**Execução**

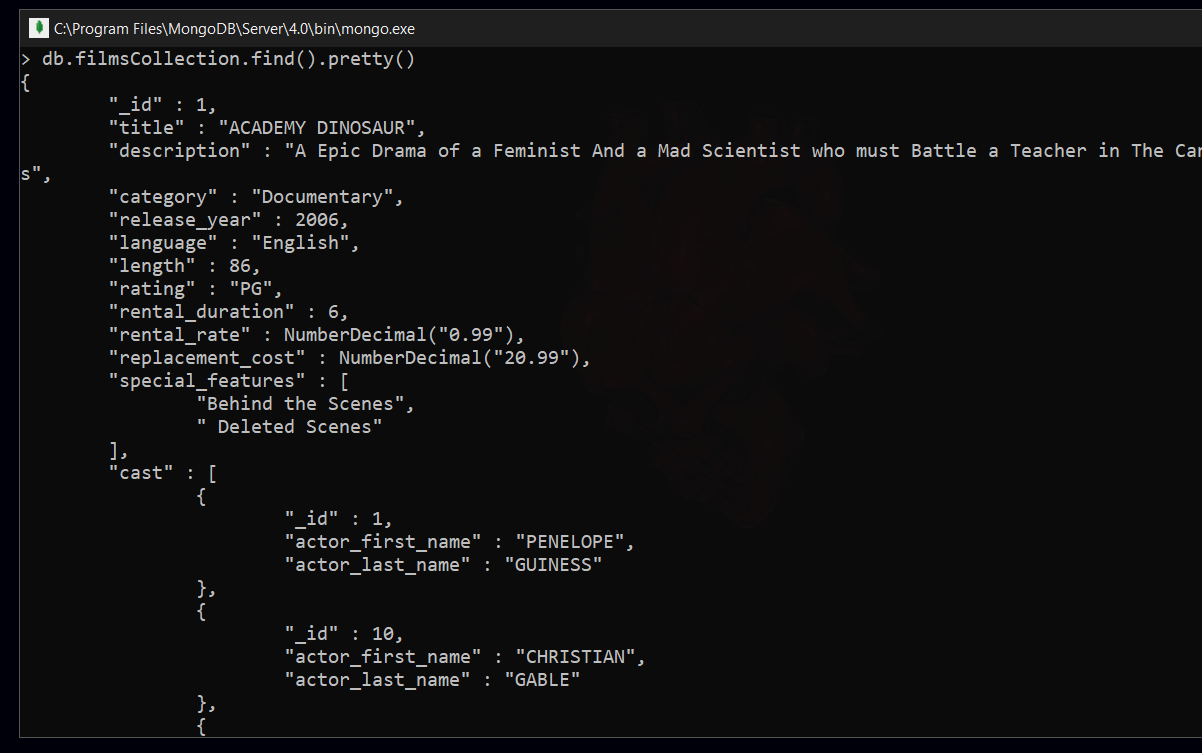
De seguida encontram-se os passos necessários para a implementação da base de dados mongoDB

* Estabelecer a conexão à base de dados *MySQL* (ao servidor)
* Certificar que a base de dados *MySQL* se encontra povoada
* Inicializar tanto o cliente mongo (*MongoClient)* como a base de dados/servidor (*MongoDatabase)*
* Executar o *script:* num terminal ou num programa adequado

* Verificação da base de dados criada

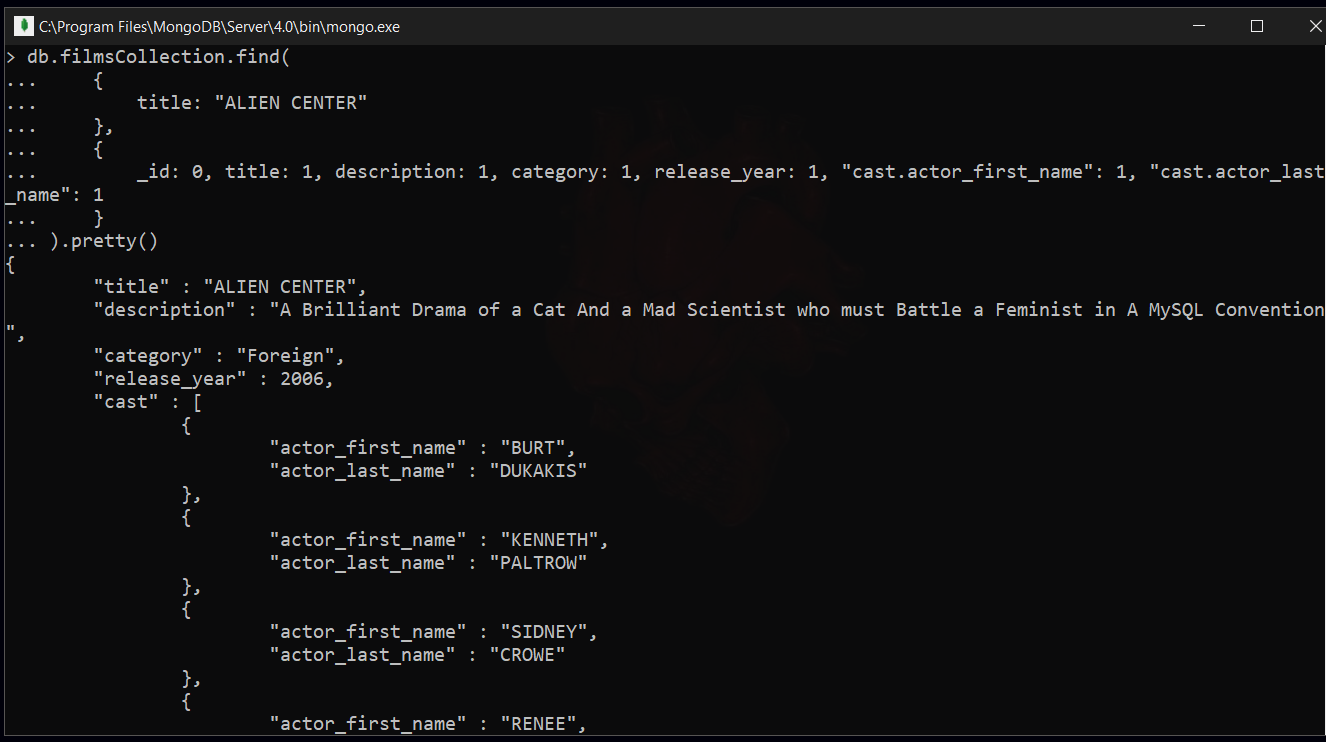
* Exemplo de um documento da coleção *customersCollection* na base de dados



*  Exemplo de um documento da coleção *filmsCollection* na base de dados

**Validação do modelo (queries)**

Na primeira parte deste trabalho prático, foi necessária a validação do modelo com perguntas ao utilizador para garantir que a base de dados cumpria todos os requisitos exigidos pelo cliente. Desta maneira, após essa validação, é necessário a certificação da mesma, agora num modelo não relacional. Sendo assim, delineamos as seguintes questões, e sua respetiva codificação e resultado:

*  **Nome dos filmes e pagamentos dos alugueres do cliente "MARIA MILLER**
* **Total de dinheiro gasto pela cliente "MARIA MILLER”**
* **Listar os filmes com "Deleted Scenes" nas special features**
* **Listar os filmes da categoria "Action" presentes na loja 1**
* **Elenco do filme "ALIEN CENTER"**

**…**

**Resultado**

**Análise crítica sobre o modelo de base de dados mongoDB**

Após a conclusão deste modelo de base de dados, chegou-se ao fim com uma base de dados NoSQL baseada em *Document Stores* – o mongoDB – capaz de responder e suportar aquilo que idealizado para ela, que é uma representação, segundo o modelo de base de dados orientadas a documentos, da base de dados relacional SQL proposta. À medida que se trabalha com este tipo de base de dados, depressa é percetível a relativa simplicidade e fácil implementação íntrinsecas a este tipo de base de dados não relacional. É também evidente a facilidade com que este tipo de base de dados se consegue mutar, ou seja, facilmente adaptar/alterar para cumprir novos requisitos, assim como este tipo de base de dados não exige uma preocupação tão grande em aspetos como tipos de dados e coerência na estrutura dos documentos, destacando-se por isso como uma base de dados de implementação mais ou menos rápida e com uma boa escalabilidade na lida com quantidades de dados maiores.