Universidade do Minho

Escola de Engenharia

Bases de Dados NoSQL

EC- Engenharia de Conhecimento
Mestrado em Engenharia Informática

Online Eletronics Store

PG50229	PG50499	PG51246
		
António Fernandes	João Torres	Mário Correia

Agosto, 2023

Conteúdo

1 Introdução **2**
- 1.1 Contextualização . . . 2
- 1.2 Principais objetivos . . . 2
- 1.3 Estrutura do relatório . . . 3

2 Dados Fornecidos **4**

3 Migração dos Dados **6**
- 3.1 MongoDB . . . 6
  - 3.1.1 Estrutura . . . 6
  - 3.1.2 Triggers . . . 7
  - 3.1.3 Testes . . . 8
- 3.2 Neo4j . . . 11
  - 3.2.1 Nós criados . . . 11
  - 3.2.2 Relacionamentos . . . 11
  - 3.2.3 Trigger . . . 13

4 Queries **15**
- 4.1 Queries MongoDB . . . 15
- 4.2 Queries Neo4j . . . 17

5 Trabalho Futuro **19**

6 Conclusão **20**

1. Introdução

Este trabalho prático surge no âmbito da unidade curricular de Bases de Dados NoSQL, pertencente ao perfil de Engenharia de Conhecimento do Mestrado em Engenharia Informática da Universidade do Minho. O projeto desenvolvido teve como motivação criar competências, nomeadamente na utilização de diferentes paradigmas de bases de dados e a sua aplicação na conceção e implementação de sistemas.

Deste modo, foi-nos proposto realizar um trabalho de análise, planeamento e implementação de um SGBD relacional e dois não relacionais. Para isso, tivemos de utilizar a base de dados relacional Online Electronics Store, cujo script customizado foi disponibilizado pela equipa docente.

1.1 Contextualização

O setor de comércio eletrónico apresenta desafios específicos em termos de armazenamento e gestão eficiente de grandes volumes de dados relacionados a produtos, clientes, transações e outras informações relevantes. A escolha adequada de modelos e paradigmas de bases de dados pode ter um impacto significativo no desempenho, na escalabilidade e na experiência do utilizador de uma loja online.

Com o avanço da tecnologia e a crescente procura por armazenamento e processamento eficiente de dados, os sistemas de bases de dados evoluíram ao longo do tempo, indo além do tradicional modelo relacional. Surgiram os sistemas de bases de dados NoSQL, que oferecem alternativas aos modelos relacionais, atendendo a diferentes necessidades e tipos de dados.

Assim, a base de dados fornecida, desenvolvida em Oracle, representa uma loja online de eletrónica fictícia e inclui várias entidades. Essas, por sua vez, estão relacionadas e fornecem informações sobre utilizadores registados, produtos, categorias, promoções, carrinho de compras, detalhes do pedido, detalhes de pagamento, funcionários, departamentos, moradas de clientes, histórico de funcionários arquivados e stock de produtos.

Para realizar o trabalho, foi necessário utilizar um modelo relacional e dois modelos não relacionais de bases de dados. Os modelos não relacionais escolhidos são baseados no paradigma de documentos (MongoDB) e no paradigma de grafos (Neo4j).

1.2 Principais objetivos

A primeira tarefa consiste em definir e explicar os processos necessários para migrar os dados fornecidos no modelo relacional para sistemas não relacionais, usando o MongoDB e o Neo4j, com o objetivo de maximizar as características de cada um desses paradigmas.

Além disso, definimos e implementamos um conjunto de *queries* que demonstram o funcionamento dos sistemas implementados. Estas serão responsáveis por testar a operacionalidade dos sistemas e mostrar como lidam com as necessidades de manipulação e recuperação dos dados.

Por fim, pretendemos comparar os modelos e funcionalidades implementadas nos sistemas não relacionais com aquelas disponibilizadas no sistema relacional fornecido. Isso irá permitir uma avaliação das diferenças entre os paradigmas utilizados, de forma a destacar as vantagens e desvantagens de cada abordagem.

1.3 Estrutura do relatório

Este relatório começa com uma introdução, onde são apresentados a contextualização do problema e os principais objetivos deste trabalho. De seguida, é fornecida a estrutura geral do relatório, que indica como as informações serão organizadas.

Dito isto, a secção "Dados Fornecidos"descreve os dados que foram disponibilizados para pela equipa docente.

A secção "Migração dos Dados"é subdividida em duas partes: MongoDB e Neo4j. Cada parte aborda a forma como foi feita a migração para ambas as bases de dados não relacionais: uma orientada aos documentos e a outra aos grafos, respetivamente.

Na secção "Queries"serão ilustrados exemplos de queries de consulta sobre as nossas base de dados não relacionais.

Por fim são apresentadas as secções "Trabalho Futuro"e "Conclusão"onde são discutidas possíveis melhorias, aprofundamentos ou desenvolvimentos futuros relacionados à migração dos dados e conclusões obtidas durante a realização do relatório.

2. Dados Fornecidos

Como já foi dito anteriormente, a base de dados Oracle fornecida representa uma loja fictícia de eletrónicos online e inclui diversos objetos de base de dados, como tabelas, views, sequências, índices, triggers, funções e procedimentos.

A bases de dados fornecida inclui 14 tabelas:

- STORE_USERS – Tabela que contém os utilizadores registados no site da loja;
- PRODUCT_CATEGORIES – Tabela que contém as categorias dos produtos;
- PRODUCT – Tabela que contém os produtos que a loja vende;
- DISCOUNT – Tabela de promoções ativas e expiradas na loja;
- CART_ITEM – Tabela que contém os produtos adicionados ao carrinho pelo cliente numa determinada sessão;
- SHOPPING_SESSION – Tabela que contém as sessões criadas pelos utilizadores;
- ORDER_DETAILS – Tabela com detalhes do pedido do utilizador;
- ORDER_ITEMS – Tabela que contém os produtos do pedido feito pelo utilizador;
- PAYMENT_DETAILS – Tabela que contém os detalhes de pagamento do pedido;
- EMPLOYEES – Tabela que contém os funcionários da loja;
- DEPARTMENTS – Tabela com os departamentos dos funcionários internos da loja;
- ADRESSES – Tabela coma as moradas dos clientes;
- EMPLOYEES_ARCHIVE – Tabela que contém os dados de funcionários arquivados – novas linhas adicionadas, linhas atualizadas e linhas excluídas, juntamente com informações sobre a hora da modificação e também sobre o utilizador que fez as alterações;
- STOCK – Tabela com o stock de produtos.

De seguida apresentamos o modelo físico da base de dados.

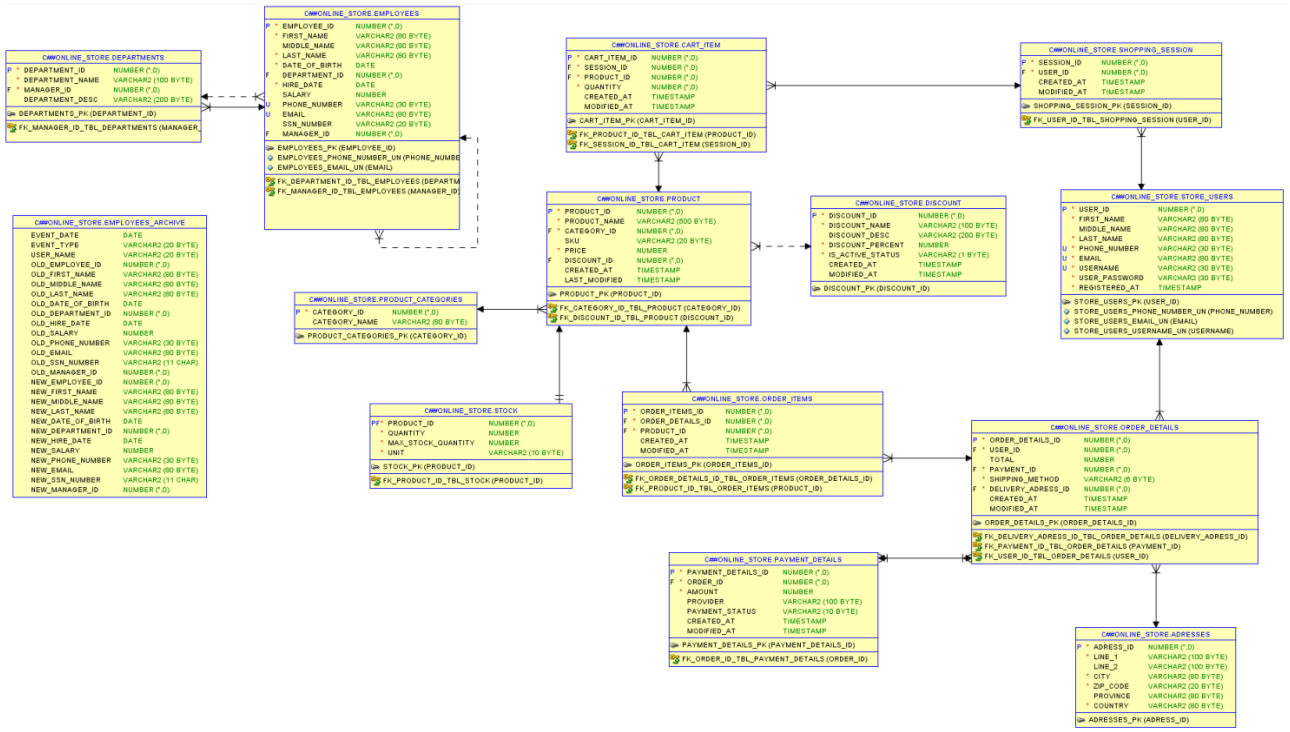

Figura 2.1: Modelo lógico da base de dados relacional fornecida

3. Migração dos Dados

Neste capítulo vamos explicar, detalhadamente, a nossa abordagem para fazer a migração dos dados fornecidos para os novos sistemas não relacionais - MongoDB e Neo4j.

3.1 MongoDB

3.1.1 Estrutura

Como sabemos que as bases de dados não relacionais não têm um esquema definido, para o **MongoDB** decidímos juntar as tabelas, de acordo com a seguinte estrutura:

- Employess - Department;

  Decidimos agregar estas duas tabelas num só documento, de forma a maximizar a pesquisa do departamento de um *Employee*. Ao manter estas duas tabelas juntas conseguimos obter de forma direta o departamento do empregado.

  Nesta agregação, as informações do Departamento são contidas na coleção de Funcionários. Cada documento de funcionário contém um objeto *Department* incluído que armazena detalhes sobre o departamento ao qual o empregado pertence. Isto permite uma representação mais eficiente e organizada do relacionamento entre funcionários e seus respetivos departamentos. Por exemplo, esta integração facilita alguns casos práticos, como por exemplo: consultar um funcionário específico e aceder às suas informações de departamento diretamente dentro do documento do funcionário; recuperar o nome do departamento, a localização ou outros atributos específicos do mesmo.

- Employees_Archive;

  Decidimos deixar esta tabela isolada como uma coleção com o objetivo de manter a ideia de "arquivo"mais concreta. Servirá de apoio essencial ao *trigger* criado, pois aqui será feito o armazenamento de todas as informações relativas a alterações à coleção *Employee*.

  Essa separação permite melhorar a gestão e a recuperação de informações históricas dos funcionários. Por exemplo, ao consultar a coleção *Employees_Archive* podemos recuperar registos antigos de alterações nos funcionários.

- Order_Details - Addresses - Payment_Details - Order_Items;

  Aqui consideramos toda a informação relativa à *Order*. Decidimos agregar toda esta informação (*Order_Details*, *Address*, *Payment_Details* e *Order_Items*). Deste modo, maximizamos a pesquisa destes campos devido à sua integração numa só coleção.

  Dito isto, os detalhes do pedido incluem informações como o número do pedido, a data de criação e o cliente. Os objetos *Address* e *Payment_Details* fornecem os detalhes do endereço de entrega e informações de pagamento, respetivamente. Através do *Order_Items* são representados os artigos individuais encomendados. Esta agregação permite uma representação completa e organizada das informações relacionadas a um pedido. Ao consultar um pedido específico, é possível aceder facilmente a todos os detalhes e pormenores associados a uma encomenda.

  Esta abordagem permite uma visão abrangente das informações relacionadas a um pedido num único documento, o que facilita a manipulação e a análise dos dados, tornado-os mais eficientes.

- Product - Discount - Category - Stock;

  Neste caso, criamos uma estrutura de uma coleção denominada produto onde integramos as tabelas *Discount*, *Category* e *Stock*. Esta agregação tem como objetivo melhorar a estrutura e permitir pesquisas eficientes sobre os dados de um determinado Produto. Algumas das possíveis aplicações práticas podem ser, por exemplo, saber se um produto possui um desconto, em que categoria se encontra e se ainda se encontra em stock. A resposta a estas perguntas será "fácil", uma vez que permitimos uma pesquisa direta sobre o Produto e as suas características (sem a necessidade de executar consultas separadas em coleções diferentes).

- Store_Users;

  Decidimos manter esta tabela como uma só coleção (independente), visto que esta apenas conter informações relativas aos *Users*.

  Ao manter essa coleção separada, permite a gestão independente de dados relacionados ao utilizador e simplifica operações relacionadas ao mesmo. Por exemplo, a consulta desta coleção permite recuperar perfis de utilizadores, atualizar informações dos mesmos ou autenticar credenciais destes para login num cenário funcional.

- Shopping_Session - Cart_Item.

  Por último, agregamos estas duas tabelas, de forma a conseguir manter conta de quais *cart items* é que pertencem à Shopping Session, consequentemente melhorando esta pesquisa.

  Nesta agregação, a tabela *Cart_Item* está integrada na *Shopping_Session*. De facto, cada documento sobre uma sessão de compras inclui informações sobre os artigos adicionados ao carrinho de compras do utilizador durante a mesma. Esta agregação facilita a gestão e o rastreio do carrinho de cada cliente. Assim, podemos, por exemplo, consultar uma sessão de compras e aceder aos artigos que estão no carrinho; realizar operações como adicionar ou remover itens.

Desta forma conseguimos obter documentos com informações relativas apenas aos Products, Employees, Employees_Archive, Orders, Users e Shopping_Sessions.

3.1.2 Triggers

Introdução

Para migrar todas as informações dos *scripts* fornecidos de forma eficiente, decidimos criar dois *"triggers"* nos serviços do Atlas. Esses *"triggers"* serão responsáveis por atualizar os dados de um arquivo de um funcionário sempre que ocorrer um insert, um update ou um delete e de verificar se aquando um update na coleção de um Product a quantidade inserida não excede o stock máximo relativo ao mesmo.

Estrutura das coleções

Para facilitar o acesso ao arquivo, combinamos as tabelas "Employees", "Department"numa coleção chamada "Employees"e deixamos a tabela "Employees_Archive"isolada como uma coleção de modo a podermos ter todos estes eventos registados. Por outro lado para facilitar o acesso ao stock de um produto decidimos agregar esta informação diretamente na coleção "Product". Assim desta forma conseguimos aceder aos valores do "Stock"muito mais diretamente e eficientemente.

Funcionamento do "trigger_archiving_employees"

Quando ocorrer uma inserção (*insert*) na coleção Employees, os novos dados serão automaticamente inseridos no arquivo correspondente, chamada de "new"(ou seja, são novos valores). Os valores antigos ("old") serão definidos como nulos, já que é a primeira inserção. No caso de uma atualização (*update*) na coleção Employees, os dados atualizados serão considerados os novos valores, substituindo os dados anteriores, que serão movidos para a parte "old". Aquando a ocorrência de um (*delete*) na coleção Employees, os dados removidos serão movidos para a parte "old"dos valores do arquivo e os valores "new"serão nulos uma vez que já não existe o documento relativo a estes valores na coleção Employees.

Funcionamento do trigger "update_product_stock"

Quando ocorrer um update na coleção Product, o trigger vai verificar se o atributo quantity do Stock alterou, e se isto se confirmar vai efetuar mais uma verificação de modo a confirmar se este valor é menor do que o Max_Stock. Se for menor, o update é efetuado com sucesso. por outro lado, se for maior, o trigger vai usar o Document Pre Image para reverter as alterações.

Métodos fornecidos pelo Atlas e sua importância

Estas operações foram conseguidas através de métodos providenciados pelo Atlas, sendo estes o Full Document, que neste caso servia para obter os dados criados ou alterados no "Change Event", e o Document Preimage que nos permitia receber o documento antes de ele ser alterado ou removido. Estas duas funcionalidades tornaram-se indispensáveis na criação dos triggers.

3.1.3 Testes

Vamos agora apresentar um exemplos de inserções, atualizações e deletes para o *MongoDB*, com o objetivo de mostrar como estão a ser a realizadas.

Primeiramente, vamos ilustrar a inserção de um documento na coleção *employees* e o respetivo resultado. Neste caso, inserimos um empregado chamado John Doe Smith e todos os dados relativos ao mesmo.


```
_id: ObjectId('64726cdddd78bb4c471b426a')
employee_id: 20
first_name: "John"
middle_name: "Doe"
last_name: "Smith"
date_of_birth: "1990-05-15T00:00:00.000+00:00"
▾ department: Object
    department_id: 2
    manager_id: null
    department_name: "Sales"
    department_desc: "Responsible for sales operations"
hire_date: "2021-01-10T00:00:00.000+00:00"
salary: 5000
phone_number: "123-456-7890"
email: "johndoe@example.com"
ssn_number: "123-45-6789"
manager_id: null
```


Figura 3.1: *Insert* de um documento em employees

```
_id: ObjectId('647545929ddb8d29b7417f77')
EVENT_DATE: 2023-05-30T00:38:42.266+00:00
EVENT_TYPE: "insert"
USER_NAME: "STORE"
OLD_EMPLOYEE_ID: null
OLD_FIRST_NAME: null
OLD_MIDDLE_NAME: null
OLD_LAST_NAME: null
OLD_DATE_OF_BIRTH: null
OLD_DEPARTMENT_ID: null
OLD_MANAGER_ID: null
OLD_HIRE_DATE: null
OLD_SALARY: null
OLD_PHONE_NUMBER: null
OLD_EMAIL: null
OLD_SSN_NUMBER: null
NEW_EMPLOYEE_ID: 20
NEW_FIRST_NAME: "John"
NEW_MIDDLE_NAME: "Doe"
NEW_LAST_NAME: "Smith"
NEW_DATE_OF_BIRTH: "1990-05-15T00:00:00.000+00:00"
NEW_DEPARTMENT_ID: 2
NEW_MANAGER_ID: null
NEW_HIRE_DATE: "2021-01-10T00:00:00.000+00:00"
NEW_SALARY: 5000
NEW_PHONE_NUMBER: "123-456-7890"
NEW_EMAIL: "johndoe@example.com"
NEW_SSN_NUMBER: "123-45-6789"
```


Figura 3.2: Resultado do *insert* de um documento

De seguida, mostramos um exemplo da atualização de um documento da coleção employess. Aqui atualizamos o *LAST_NAME* do empregado previamente adicionado e o seu respetivo *EMAIL*.


```
_id: ObjectId('64726cdddd78bb4c471b426a')
employee_id: 20
first_name: "John"
middle_name: "Doe"
last_name: "Wick"
date_of_birth: "1990-05-15T00:00:00.000+00:00"
▾ department: Object
    department_id: 2
    manager_id: null
    department_name: "Sales"
    department_desc: "Responsible for sales operations"
hire_date: "2021-01-10T00:00:00.000+00:00"
salary: 5000
phone_number: "123-456-7890"
email: "johnwick@example.com"
ssn_number: "123-45-6789"
manager_id: null
```


Figura 3.3: *Update* de um documento em employees

```
_id: ObjectId('647545ce9ddb8d29b741b5a4')
EVENT_DATE: 2023-05-30T00:39:42.016+00:00
EVENT_TYPE: "update"
USER_NAME: "STORE"
OLD_EMPLOYEE_ID: 20
OLD_FIRST_NAME: "John"
OLD_MIDDLE_NAME: "Doe"
OLD_LAST_NAME: "Smith"
OLD_DATE_OF_BIRTH: "1990-05-15T00:00:00.000+00:00"
OLD_DEPARTMENT_ID: 2
OLD_MANAGER_ID: null
OLD_HIRE_DATE: "2021-01-10T00:00:00.000+00:00"
OLD_SALARY: 5000
OLD_PHONE_NUMBER: "123-456-7890"
OLD_EMAIL: "johndoe@example.com"
OLD_SSN_NUMBER: "123-45-6789"
NEW_EMPLOYEE_ID: 20
NEW_FIRST_NAME: "John"
NEW_MIDDLE_NAME: "Doe"
NEW_LAST_NAME: "Wick"
NEW_DATE_OF_BIRTH: "1990-05-15T00:00:00.000+00:00"
NEW_DEPARTMENT_ID: 2
NEW_MANAGER_ID: null
NEW_HIRE_DATE: "2021-01-10T00:00:00.000+00:00"
NEW_SALARY: 5000
NEW_PHONE_NUMBER: "123-456-7890"
NEW_EMAIL: "johnwick@example.com"
NEW_SSN_NUMBER: "123-45-6789"
```


Figura 3.4: Resultado do *update* de um documento

Por fim, apresentamos o *delete* do empregado previamente adicionado.


```
_id: ObjectId('647545eb9ddb8d29b741d4ea')
EVENT_DATE: 2023-05-30T00:40:11.276+00:00
EVENT_TYPE: "delete"
USER_NAME: "STORE"
OLD_EMPLOYEE_ID: 20
OLD_FIRST_NAME: "John"
OLD_MIDDLE_NAME: "Doe"
OLD_LAST_NAME: "Wick"
OLD_DATE_OF_BIRTH: "1990-05-15T00:00:00.000+00:00"
OLD_DEPARTMENT_ID: 2
OLD_MANAGER_ID: null
OLD_HIRE_DATE: "2021-01-10T00:00:00.000+00:00"
OLD_SALARY: 5000
OLD_PHONE_NUMBER: "123-456-7890"
OLD_EMAIL: "johnwick@example.com"
OLD_SSN_NUMBER: "123-45-6789"
NEW_EMPLOYEE_ID: null
NEW_FIRST_NAME: null
NEW_MIDDLE_NAME: null
NEW_LAST_NAME: null
NEW_DATE_OF_BIRTH: null
NEW_DEPARTMENT_ID: null
NEW_MANAGER_ID: null
NEW_HIRE_DATE: null
NEW_SALARY: null
NEW_PHONE_NUMBER: null
NEW_EMAIL: null
NEW_SSN_NUMBER: null
```


Figura 3.5: Resultado de um *delete* de um documento

3.2 Neo4j

Como sabemos que os bancos de dados não relacionais, como o **Neo4j**, não possuem tabelas, mas sim grafos, decidimos criar e, de seguida, unir os nós, de acordo com a seguinte estrutura.

3.2.1 Nós criados

Nesta subsecção vamos apresentar os nós criados. Criamos um nó por cada tabela da base de dados relacional fornecida.

- User;
- Session;
- Cart;
- Address;
- Product;
- Category;
- Discount;
- Payment;
- Employee;
- Employee_Archive;
- Department;
- Stock;
- OrderDetails;
- OrderItems.

3.2.2 Relacionamentos

Nesta subsecção vamos apresentar os relacionamentos estabelecidos entre os nós, após a criação dos mesmos.

- Criar relacionamento BELONGS_TO entre Product e Category;

  Fizemos este relacionamento para poder saber, de uma forma mais eficiente, a qual categoria é que um produto está associado, como por exemplo, um Iphone 13 PRO está associado à categoria Smartphones.

- Criar relacionamento DELIVER_AT entre OrderDetails e Address;

  Fizemos este relacionamento, com o objetivo de saber em qual endereço uma certa encomenda deve ser entregue.

- Criar relacionamento HAS_CART entre Session e Cart;

  Fizemos este relacionamento porque, durante uma sessão de compras de um utilizador, existem items que são adicionados pelo mesmo, mantendo assim a consistência na relação entre Session - Cart. Desta forma conseguimos saber eficientemente todos os Cart Items que estão naquela sessão.

- Criar relacionamento HAS_DISCOUNT entre Product e Discount;

  Fizemos este relacionamento, para poder saber se um produto possui um desconto e os valores correspondentes ao mesmo.

- Criar relacionamento HAS_SESSION entre User e Session;

  Fizemos este relacionamento, com o objetivo de saber a qual/quais sessão/ões de compras é que um utilizador está associado.

- Criar relacionamento HAS_STOCK entre Product e Stock;

  Fizemos este relacionamento, devido ao facto de ser necessário verificar se um determinado produto está em stock na loja, para poder efetuar compras com dados consistentes.

- Criar relacionamento IS_IN_CART_ITEM entre Product e Cart;

  Fizemos este relacionamento, para que, durante uma sessão, seja possível verificar eficientemente qual é o produto que está ou não associado ao Cart Item.

- Criar relacionamento IS_IN_ORDER_ITEM entre Product e OrderItem;

  Fizemos este relacionamento para poder saber qual o produto identificado no OrderItem. Como o OrderItem irá possuir uma relação com o produto esta permite saber qual é o produto daquele OrderItem e as suas características.

- Criar relacionamento IS_PAID entre OrderDetails e Payment;

  Fizemos este relacionamento para conseguir verificar de forma eficaz se uma determinada order está paga, quais foram os detalhes da transacção.

- Criar relacionamento MANAGES entre Employee e Employee;

  Fizemos este relacionamento para conseguir identificar quais os funcionários que um determinado funcionário gere.

- Criar relacionamento ORDERED entre User e OrderDetails;

  Fizemos este relacionamento para verificar efetivamente quais as Orders que um User efetuou e as suas características.

- Criar relacionamento ORDERED_IN entre OrderDetails e OrderItem;

  Fizemos este relacionamento para identificar quais os itens que estão inseridos dentro de uma Order, e consequentemente saber as suas características de uma forma eficiente.

- Criar relacionamento WORKS_IN entre Employee e Department;

  Fizemos este relacionamento para identificar em qual departamento é que um certo Employee trabalha, e consequentemente saber todos os detalhes do seu departamento eficazmente.

3.2.3 Trigger

Nesta parte do trabalho tínhamos como objetivo criar ambos os triggers que migramos para MongoDB a partir da Base de Dados Relacional Oracle, usando o APOC.

O APOC (Awesome Procedures on Cypher) oferece uma vasta gama de procedimentos e funções adicionais para efectuar operações complexas de forma mais eficaz, expandindo assim as capacidades do Cypher.

Com a ajuda do APOC, os utilizadores do Neo4j podem realizar tarefas complexas que não se enquadram nas perguntas simples do Cypher.

O APOC elimina a necessidade de processamento físico, permitindo que os desenvolvedores realizem tarefas complexas de manipulação e transformação de dados diretamente no banco de dados.

Com isto em mente tentamos algumas abordagens para fazer os *triggers* para o **Neo4j**, mas devido à quantidade baixa de documentação externa á documentação oficial do APOC, não nos foi possível concluir este objetivo, no entanto, vamos demonstrar os nossos esforços para criar o trigger que tinha o objetivo de criar nós no nó do arquivo dos Employees:

```
CALL apoc.trigger.add('updateEmployeesArchive', '
  UNWIND apoc.trigger.nodesByLabel($assignedLabels, "after") AS node
  WITH node, apoc.date.format(apoc.date.currentTimestamp(), "s", "yyyy-MM-dd
      HH:mm:ss") AS eventDate
  CALL apoc.create.node(["Employees_Archive"]) YIELD node AS archiveNode
  SET archiveNode += {
    event_date: eventDate,
    event_type: $eventType,
    old_date_of_birth: node.date_of_birth,
    old_department_id: node.department_id,
    old_email: node.email,
    old_employee_id: node.employee_id,
    old_first_name: node.first_name,
    old_hire_date: node.hire_date,
    old_last_name: node.last_name,
    old_middle_name: node.middle_name,
    old_phone_number: node.phone_number,
    old_salary: node.salary,
    old_ssn_number: node.ssn_number
  }
  SET archiveNode += {
    new_date_of_birth: CASE $eventType WHEN "DELETE" THEN NULL ELSE
        node.date_of_birth END,
    new_department_id: CASE $eventType WHEN "DELETE" THEN NULL ELSE
        node.department_id END,
    new_email: CASE $eventType WHEN "DELETE" THEN NULL ELSE node.email END,
    new_employee_id: CASE $eventType WHEN "DELETE" THEN NULL ELSE node.employee_id
        END,
    new_first_name: CASE $eventType WHEN "DELETE" THEN NULL ELSE node.first_name
        END,
    new_hire_date: CASE $eventType WHEN "DELETE" THEN NULL ELSE node.hire_date END,
    new_last_name: CASE $eventType WHEN "DELETE" THEN NULL ELSE node.last_name END,
    new_middle_name: CASE $eventType WHEN "DELETE" THEN NULL ELSE node.middle_name
        END,
    new_phone_number: CASE $eventType WHEN "DELETE" THEN NULL ELSE
        node.phone_number END,
```

```
    new_salary: CASE $eventType WHEN "DELETE" THEN NULL ELSE node.salary END,
    new_ssn_number: CASE $eventType WHEN "DELETE" THEN NULL ELSE node.ssn_number END
  }
  RETURN archiveNode
', {phase: 'after', eventType: "INSERT"})
```

Nesta versão de código, existe a utilização de funções "deprecated"nas novas versões do APOC, como por exemplo o *"apoc.trigger.add"*.

Para além disto, o trigger seria criado com objetivo de capturar eventos de inserção no nó Employees e criar um novo nó no nó Employees_Archive com os detalhes do evento e os valores antigos e novos dos campos relevantes.

De seguida, seria suposto fazer UNWIND dos nós com base no rótulo especificado numa variável (assignedLabels) e capturar os nós atualizados. Depois, deveria criar um novo nó no nó Employees_Archive usando apoc.create.node. As propriedades do novo nó seriam definidas com base nos valores dos nós atualizados.

O trigger usaria uma variável eventType para poder especificar o tipo de evento a ser capturado.

Neste segundo trigger, tinhamos como objetivo de criar um outro que verificasse o valor da quantidade de Stock de um Produto quando ocorresse um *insert* ou *update* no nó Stock.

```
CALL apoc.trigger.add('checkStockQuantity', '
  UNWIND $createdNodes + $updatedNodes AS stock
  WITH stock
  WHERE stock.quantity > stock.max_stock_quantity
  SET stock.quantity = stock.max_stock_quantity
', {phase: 'before'})
```

Este trigger fornecia uma camada de garantia que a quantidade de Stock de um nó não excedesse a quantidade máxima especificada (max_stock_quantity). Se a quantidade for maior que o máximo permitido, seria ajustada automaticamente para o valor máximo de stock durante a criação ou atualização do nó.

4. Queries

Neste capítulo definimos um conjunto de queries para demonstrar a operacionalidade dos sistemas que implementamos.

4.1 Queries MongoDB

Nesta subseção vamos apresentar as queries criadas para o MongoDB

- **Consulta de produtos com desconto ativo**

```
db.Product.find({
  "discount.is_active_status": "Y"
})
```

- **Consulta de sessão de compra de um utilizador com os itens do carrinho**

```
db.Shopping_Session.aggregate([
  {
    $match: {
      user_id: 3
    }
  },
  {
    $lookup: {
      from: "Product",
      localField: "cart_items.product_id",
      foreignField: "product_id",
      as: "cart_items_details"
    }
  },
  {
    $project: {
      session_id: 1,
      user_id: 1,
      "cart_items_details.product_id": 1,
      "cart_items_details.product_name": 1,
      "cart_items_details.price": 1,
      "cart_items.quantity": 1
    }
  }
])
```

- **Consulta para obter o histórico de alterações de um funcionário específico**

```
db.Employees_Archive.find({
  "NEW_EMPLOYEE_ID": 1
})
```

- **Consulta para obter a quantidade de produtos em stock de uma determinada categoria**

```
db.Product.aggregate([
  {
    $match: {
      "category.category_id": 1
    }
  },
  {
    $group: {
      _id: "$category.category_name",
      total_stock: { $sum: "$stock.quantity" }
    }
  }
])
```

- **Consulta para obter o valor total das vendas processadas através do PayPal**

```
db.Order.aggregate([
  {
    $match: {
      "payment.provider": "PayPal",
      "payment.payment_status": "PROCESSED"
    }
  },
  {
    $group: {
      _id: null,
      total_sales: { $sum: "$total" }
    }
  }
])
```

- **Consulta para obter a lista de utilizadores registados depois de uma determinada data**

```
db.Store_Users.find({
  "REGISTERED_AT": {
    $gt: ISODate("2021-01-01T00:00:00.000Z")
  }
})
```

- **Consulta para obter a média salarial por departamento**

```
db.Employees.aggregate([
  {
    $group: {
      _id: "$department.department_name",
      average_salary: { $avg: "$salary" }
    }
  }
])
```

- **Consulta para obter os produtos mais populares com base no número de pedidos**

```
db.Order.aggregate([
  {
    $unwind: "$order_items"
  },
  {
    $group: {
      _id: "$order_items.product_id",
      total_orders: { $sum: 1 }
    }
  },
  {
    $lookup: {
      from: "Product",
      localField: "_id",
      foreignField: "product_id",
      as: "product"
    }
  },
  {
    $sort: {
      total_orders: -1
    }
  },
  {
    $limit: 5
  },
  {
    $project: {
      _id: 0,
      product_id: "$product.product_id",
      product_name: "$product.product_name",
      total_orders: 1
    }
  }
])
```

4.2 Queries Neo4j

- **Obter todos os produtos no carrinho de um User**

```
MATCH (u:User {user_id: 1})-[:HAS_SESSION]->(s:Session)-[:HAS_CART]->
(ci:Cart)-[:IS_IN_CART_ITEM]-(p:Product)
RETURN p.product_id, p.product_name, ci.quantity
```

- **Obter todos os funcionários de um departamento específico**

```
MATCH (e:Employee)-[:WORKS_IN]->(d:Department {department_id: 3})
RETURN e.employee_id, e.first_name, e.last_name, d.department_name
```

- **Retornar todos os produtos de uma determinada categoria**

```
MATCH (p:Product)-[:BELONGS_TO]->(c:Category {category_id: 1})
RETURN p.product_id, p.product_name
```

- **Retornar os produtos mais vendidos**

```
MATCH (p:Product)-[:IS_IN_ORDER_ITEM]->(:OrderItems)
WITH p, COUNT(*) AS total_orders
ORDER BY total_orders DESC
RETURN p.product_id, p.product_name, total_orders
```

- **Retornar os nomes completos de Employees que trabalham num departamento específico**

```
MATCH (e:Employee)-[:WORKS_IN]->(d:Department {department_id: 2})
RETURN e.first_name, e.last_name
```

- **Total gasto por um User específico em todas as suas Orders**

```
MATCH (u:User {user_id: 1})-[:ORDERED]->(o:OrderDetails)
RETURN u.user_id, sum(o.total) AS total_spent
```

- **Descobrir os produtos frequentemente comprados juntos**

```
MATCH
    (p1:Product)-[:IS_IN_ORDER_ITEM]->(oi:OrderItems)-[:ORDERED_IN]->(od:OrderDetails)
WHERE p1 <> p2
RETURN p1.product_name, p2.product_name, COUNT(DISTINCT
    od.order_details_id) AS co_occurrences
ORDER BY co_occurrences DESC
```

- **Retornar o valor total de vendas por categoria de produto**

```
MATCH (p:Product)-[:BELONGS_TO]->(c:Category)
OPTIONAL MATCH (p)-[:IS_IN_ORDER_ITEM]->(oi:OrderItems)
WITH p, c, COUNT(oi) AS total_sales
RETURN c.category_name, SUM(total_sales * p.price) AS total_sales
ORDER BY total_sales DESC
```

5. Trabalho Futuro

Como trabalho futuro considera-se a opção de expandir e explorar outras opções de sistemas de gestão de bancos de dados NoSQL como por exemplo, base de dados chave-valor. Assim como, explorar outras ferramentas de armazenamento de dados.

Para além disso, também fica em aberto possíveis otimizações do nosso código e a introdução de novas entradas e atributos para as nossas bases de dados. Sendo capaz de lidar com cenários de organizações e empresas reais. Isto é, poderíamos explorar em termos de performance como seria lidar com um maior volume e variedade de dados mas também outras opções de armazenamento e suas respetivas relações.

Em relação a base de dados NoSQL, podemos explorar técnicas de escalabilidade horizontal (sharding) e sua distribuição de dados para melhorar o desempenho em ambientes de grande escala.

Outro aspeto, seria investigar outras queries de consulta não mencionadas, de forma a explorar e analisar outros aspetos referentes ás nossas bases de dados e extrair outro conhecimento útil sobre os nossos dados.

Em suma, ao explorar sistemas adicionais, expandiria o nosso conhecimento em relação à área e permitia que avaliássemos quais as suas vantagens e desafios em comparação com os sistemas que foram implementados.

6. Conclusão

Em conclusão, o trabalho prático desenvolvido no âmbito da unidade curricular de Bases de Dados NoSQL fez-nos aprofundar os nossos conhecimentos adquiridos durante as aulas e obter novos conhecimentos relativos à utilização dos diferentes paradigmas de base de dados e como aplicá-los neste contexto.

Observámos que as base de dados NoSQL apresentam maior flexibilidade, escalabilidade e desempenho para diferentes tipos de dados e que estes paradigmas tornam-se adequados para aplicações modernas.

Considerámos que o maior desafio durante o desenvolvimento do projeto, foi a implementação dos triggers em Neo4j devido a sua complexidade. No entanto, acreditámos que foi realizado um bom trabalho e que foram conseguidos os objetivos do enunciado prático.

Para além disso, através da análise, planeamento e implementação das bases de dados utilizadas pudemos explorar melhor quais as vantagens e desvantagens de cada abordagem, bem como a sua aplicabilidade.

Por último, através da implementação deste trabalho prático conseguimos entender melhor quais os principais desafios ao implementar as base de dados não relacionais e assim aprofundar melhor o nosso conhecimento geral nesta área.