Universidade do Minho
Mestrado Integrado em Engenharia Informática
Licenciatura em Ciências da Computação

Unidade Curricular de Bases de Dados

Ano Letivo de 2019/2020

Agendamento e realização de Testes Clínicos de atletas de Atletismos de diferentes modalidades e categorias

Grupo 18

João Pedro Rodrigues Azevedo, a85227
Paulo Jorge da Silva Araújo, a85729
Paulo Jorge Moreira Lima, a89983
Pedro Filipe Costa Machado, a83719

Dezembro, 2019

BD

Resumo

Este trabalho consiste na análise, planeamento, desenvolvimento e implementação de um SGBD para uma clínica que realiza agendamento e realização de testes clínicos para os Atletas da BD juntamente com os médicos da clínica.

Numa primeira fase, foi desenvolvido um sistema relacional no qual foram feitos vários esquemas (conceptual, lógico e físico) e implementadas várias *queries* para garantir a operacionalidade da Base de Dados.

Foram definidos os vários recursos (físicos e humanos), sistemas de informação e serviços que seriam necessários para garantir o funcionamento da clínica e do seu SGBD.

Na segunda fase, procedemos à migração de dados do antigo sistema relacional para um não relacional (Neo4j). Neste novo sistema é também garantida a sua operacionalidade através de *queries* equivalentes às anteriormente desenvolvidas.

Em suma, o trabalho cumpre os objetivos propostos na sua totalidade. Contudo, assumimos que não é perfeito, pelo que podemos melhorá-lo em certos aspetos, tais como, os identificadores das entidades dos dois sistemas, pois poderíamos ter usado outras propriedades como identificadores e, assim, poupar espaço a nível de memória que é um recurso essencial numa Base de Dados. Achamos que temos espaço para evolução a nível da migração de dados e nas *queries* de ambos sistemas e que com mais algum tempo poderíamos ter um trabalho mais estruturado e aplicável ao mundo real.

Área de Aplicação: Desenho e arquitectura de Sistemas de Bases de Dados
Palavras-Chave: Bases de Dados Relacionais, Base de Dados Gráfico, Neo4j, Clínica, Atletismo, Modalidades, Atletas, Médicos

Índice

Índice de Figuras

1. Introdução

Aqui iremos fazer uma breve introdução ao tema do projeto, o propósito da sua realização, os objetivos que pretendemos ter obtido durante a sua realização e uma descrição breve da estrutura do resto do relatório.

1.1. Contextualização

Este trabalho foi realizado no âmbito da unidade curricular de Base de Dados de Mestrado Integrado em Engenharia Informática (MIEI) e serviu para assimilar conceitos como sistemas relacionais e não relacionais introduzidos nas aulas teóricas e um pouco explorados nas aulas práticas.

1.2. Apresentação do Caso de Estudo

O projeto desenvolvido consistia na criação de uma Base de Dados para uso de uma certa Clínica. Para isso, a base de dados teria de ser capaz de suportar vários atletas de diferentes modalidades de atletismo e os seus testes clínicos.

Para além disso, teria de ser possível o agendamento de testes e existirem várias modalidades e respetivas categorias associadas a cada atleta. De realçar que estes testes clínicos podem ser usufruídos pelos vários atletas, tendo para isso médicos especializados em cada área especifica.

1.3. Motivação e Objetivos

Com a realização deste trabalho prático foi pretendido que desenvolvêssemos capacidades nos conteúdos da unidade curricular em Bases de Dados, principalmente no planeamento e execução de projetos de Sistemas Bases de Dados, (SBD), nas duas componentes estudadas, modelo relacional e não relacional (NoSQL). Com particular ênfase na análise, planeamento, modelação, arquitetura e implementação deste tipo de sistemas (Connolly and Begg, 2004).

Após a realização deste trabalho era pretendido que tivéssemos uma melhor perceção de como funciona o mercado de trabalho na área de Base de Dados, com vista a proporcionar-nos competências para conseguirmos sucesso nesta área da Informática. Realçar que o nosso grupo usou como modelo não relacional uma base de dados gráfica (*Graph Databases*).

1.4. Estrutura do Relatório

No resto do relatório vamos demonstrar como procedemos para planear, modelar e implementar os sistemas de base de dados para este tema.

Para isso, começamos por implementar uma Base de Dados relacional que depois servirá de migração para uma nova Base de Dados não relacional que cumprirá os mesmos requisitos que a anterior. Iremos também demonstrar as *queries* que desenvolvemos para a operacionalidade desta Base de Dados e o que implicará a nível de recursos humanos, sistemas de informação e serviços. No final iremos fazer uma conclusão do trabalho realizado.

2. Desenvolvimento do SGBD

Em primeiro lugar começamos por desenvolver uma Base de Dados relacional devido à sua simplicidade e estrutura lógica que, no começo do desenvolvimento, revelaram-se fundamentais para estruturar os dados consoante os nossos requisitos.

Outro fator que nos levou à utilização de um SBD relacional foi por causa da similaridade que esta base de dados tem perante um esquema lógico e concetual, esquemas que começamos por estruturar depois da caracterização dos perfis de utilização e dos requisitos envolvidos por parte do tema proposto.

2.1. Análise e justificação da viabilidade de cada um dos SGBD relacional e não relacional

A implementação de um SGBD relacional garante que os valores nulos sejam suportados para representar informação não disponível ou não aplicável e todos os dados são representados da mesma maneira: tabelas bidimensionais. Para além disso, existe uma independência física e lógica no sistema e suporta a propriedade ACID. Assim, o sistema relacional permite a criação de um modelo lógico consistente da informação a ser armazenada (Pereira, J. L., 2006).

Contudo, este sistema perde muito em performance quando o número de dados começa a aumentar (*Big Data*) e é aí que começam as vantagens do sistema não relacional.

Um sistema não relacional tem a característica de não ter os dados estruturados, pelo que não tem um esquema fixo. Enquanto que nos sistemas relacionais é preciso o esforço de modelação, este não necessita desse requisito. Logo, o SGBD não

relacional tende a ser mais flexível com os dados que pode aceitar, tornando o processo de desenvolvimento muito ágil.

De realçar que a maioria das bases de dados não relacionais não seguem os princípios do ACID.

2.2. Planificação do levantamento de requisitos do problema

Nesta secção enumeramos os vários requisitos para permitir à Base de Dados um correto funcionamento na gestão dos dados clínicos da Clínica em causa.

Os requisitos levados em conta foram os seguintes:

- ➔ Um teste clínico é dado por apenas um médico, mas um médico pode ter vários testes;
- ➔ Um atleta pertence a uma equipa apenas, a um escalão, tem apenas um código postal e pode ter uma ou várias Modalidades;
- ➔ Apenas existe um exame para cada teste clínico realizado;
- ➔ Um atleta pode ter várias consultas realizadas e por realizar;
- ➔ Uma Modalidade pode ter várias categorias;

Na nossa Base de Dados consideramos que um atleta poderia ter mais do que um teste clínico por ano e, por consequente, mais do que um exame por ano.

Tivemos em conta também que o exame relativo ao teste clínico indicaria se o atleta em causa estaria ou não apto para a prática da modalidade, sendo o resultado deste declarado pelo médico do exame. O médico que executa o teste clínico não tem necessariamente de ser o médico que executa o exame associado a este teste, pois poderá ser outro médico que consoante os resultados do teste clínico e do exame realizado determina o resultado deste. De realçar que um teste clínico pode ter mais do que um exame.

Assim, um teste clínico comporta-se como uma entidade que armazena os perímetros e restantes medições feitas ao atleta, enquanto que o exame armazena o resultado, o médico desse exame e outros atributos.

Deste modo, foram considerados as seguintes entidades do problema:

2.2.1. Atleta

- nif_Atleta: Identificador do Atleta;
- nome: Primeiro e último nome do Atleta;
- email: Email do Atleta;
- data_nascimento: Data de nascimento do Atleta;
- pais: País que o Atleta representa;
- genero: Género do Atleta;
- idEquipa: Identifica a entidade “Equipa”;
- idCodigo_Postal: Identifica a entidade “Codigo_Postal”;
- idEscalao: Identifica a entidade “Escalao”;

2.2.2. Categorias_Modalidade

- descricao_Categorias: Descreve a Categoria;
- nif_Atleta: Identificador da entidade “Atleta”;
- idModalidade: Identificador da entidade “Modalidade”;

2.2.3. Codigo_Postal

- idCodigo_Postal: Identificador do Código Postal;
- Localidade: Localidade do Código Postal;
- codigo: Código Postal;

2.2.4. Equipa

- idEquipa: Identificador da Equipa;
- nome: Nome da Equipa;

2.2.5. Escalao

- idEscalao: Identificador do Escalão;
- descricao: Descrição do Escalão;

2.2.6. Especialidade

- idEspecialidade: Identificador da Especialidade;
- nome: Nome da Especialidade;

2.2.7. Exame

- idExame: Identificador do Exame;
- nome: Nome do Exame;
- descricao: Descrição do Exame;
- preco: Preço do Exame;
- idTeste_Clinico: Identificador da entidade “Teste_Clinico”;
- nif_Medico: Identificador da entidade “Medico”;

2.2.8. Marcacao

- idMarcacao: Identificador da marcação;
- data_Agendada: Data que a marcação está agendada;
- nif_Atleta: Identificador da entidade “Atleta”;
- nif_Medico: Identificador da entidade “Medico”;
- idTeste_Clinico: Identificador da entidade “Teste_Clinico”;

2.2.9. Medico

- nif_Medico: Identificador do Médico;
- nome: Nome do Médico;
- data_nascimento: Data de nascimento do Médico;
- email: Email do Médico;
- idCodigo_Postal: Identificador da entidade “Codigo_Postal”;
- idEspecialidade: Identificador da entidade “Especialidade”;

2.2.10. Modalidade

- idModalidade: Identificador da Modalidade;
- nome: Nome da Modalidade;

2.2.11. Teste_Clinico

- idTeste_Clinico: Identificador do Teste Clínico;
- altura: Altura medida no Teste;
- peso: Peso medido no Teste;
- pressao_arterial: Pressão arterial medida no Teste;
- freq_cardiaca: Frequência Cardíaca medida no Teste;
- indice_massa_corporal: IMC medido no Teste;
- data: Data em que o Teste foi realizado;

2.3. Relacionamento entre as Entidades

Relação entre as entidades Atleta e Equipa. Um atleta apenas pode ter uma equipa, mas uma equipa pode ter vários atletas;

Relação entre as entidades Atleta e Escalao. Um atleta apenas pode ter um escalão, porém um escalão tem vários atletas;

Relação entre entidades Atleta e Codigo_Postal. Um atleta tem apenas um código postal e um código postal tem vários atletas;

Relação entre entidades Atleta e Modalidade. Um atleta pode ter várias modalidades e uma modalidade pode ter vários atletas. Como é uma relação de n para n no modelo relacional podemos meter uma entidade intermédia, neste caso a Categorias_Modalidade;

Relação entre entidades Atleta e Medico. Um atleta pode ter vários médicos e um médico pode ter vários atletas. Temos novamente uma relação de n para n, logo usamos como entidade intermédia Marcacao;

Relação entre entidades Medico e Codigo_Postal. Um médico tem um código postal e um código postal tem vários médicos;

Relação entre entidades Medico e Especialidade. Um médico tem uma especialidade, enquanto que uma especialidade tem vários médicos.

Relação entre entidades Teste_Clinico e Marcacao. Um teste clínico tem uma marcação e uma marcação tem um teste clínico.

Relação entre entidades Teste_Clinico e Exame. Um teste clínico pode ter vários exames, mas um exame tem apenas um teste clínico.

2.4. Construção dos diferentes esquemas

Nesta secção pretende-se desenvolver, validar e documentar todas as etapas definidas na construção dos diferentes esquemas (conceptual, lógico e físico).

Depois de já termos as entidades definidas e a maior parte das relações estabelecidas, podemos começar a definir os esquemas antecessores à introdução do SGBD relacional (Elmasri and Navathe, 2010).

Assim, definimos o seguinte esquema conceptual:

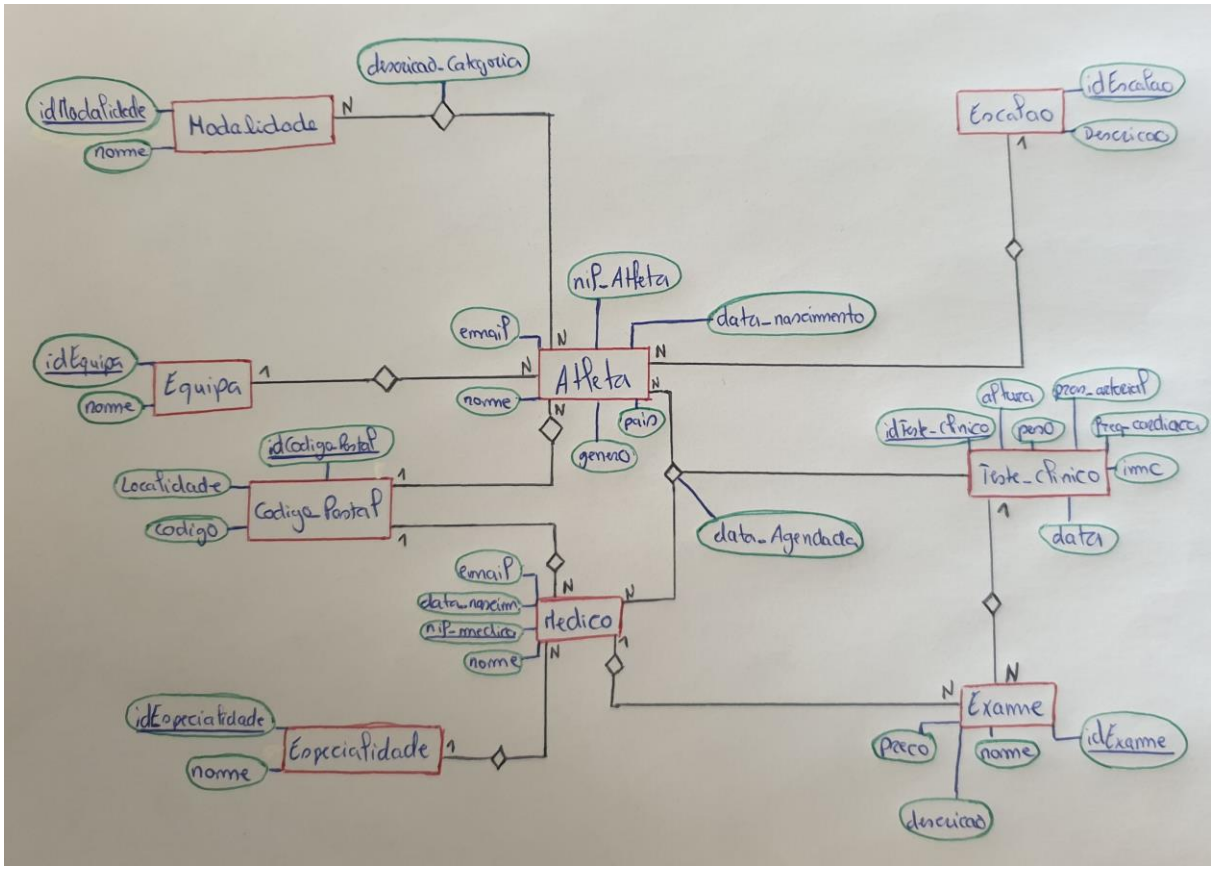

Figura 1 - Esquema Conceptual.

Neste esquema começamos a ter uma noção de como a nossa Base de Dados vai ficar estruturada, pois conseguimos identificar as entidades, as suas relações e os atributos das próprias entidades ou das relações destas.

De realçar que temos duas relações com atributos: a relação de Atleta para Modalidade e a relação entre Medico, Teste_Clinico e Atleta. Estes dois atributos de relação irão se transformar em entidades no esquema lógico, como se irá ver.

De seguida, já estamos em posição para fazer o esquema lógico através do *MySQL Workbench:*

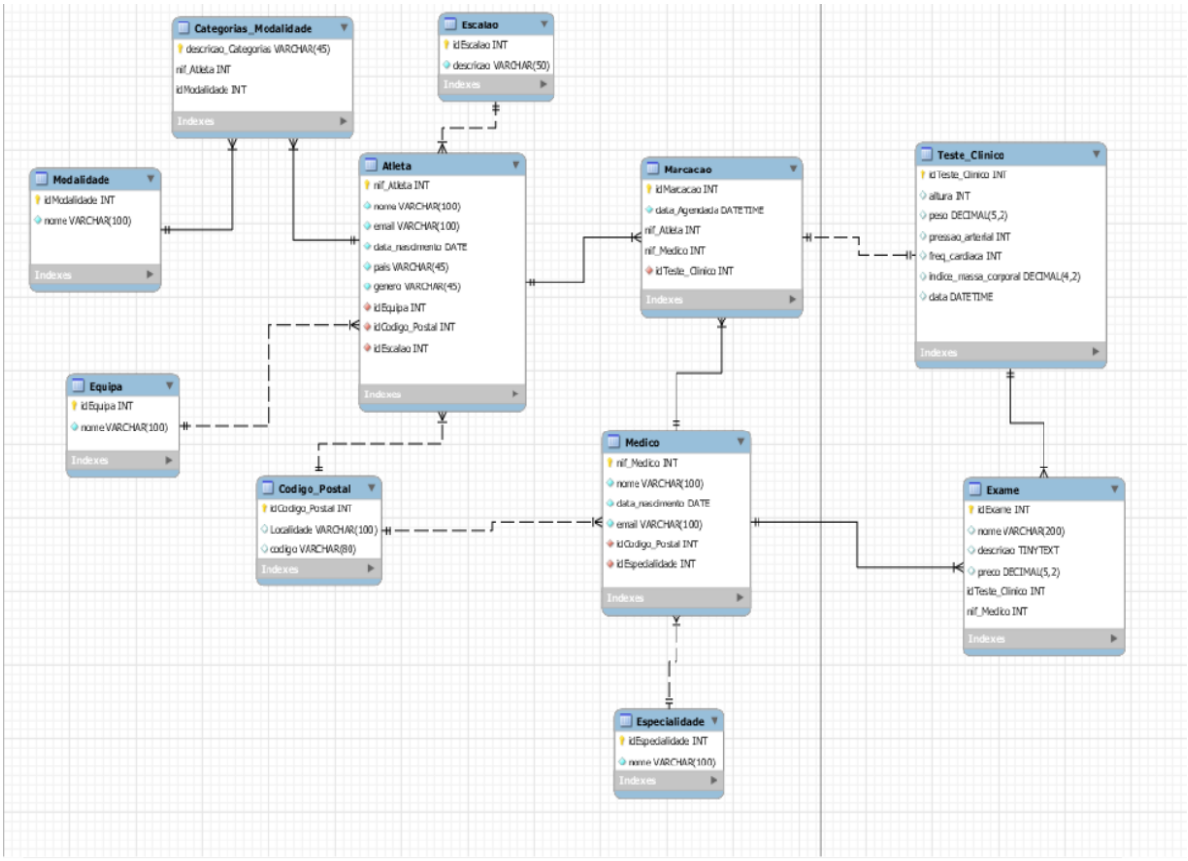

Figura 2 - Esquema Lógico.

Neste esquema notamos que existem algumas relações como *non-identifying* e outras como *identifying*, sendo as primeiras a tracejado. Isto deriva da dependência de algumas entidades em relação a outras, ou seja, entidades com relações *identifying* devem-se ao facto de uma tabela depender da outra para existir e vice-versa. Temos como exemplo o Exame e o Médico, em que um exame não pode existir sem um Médico.

Por último através do *Forward Engineer* da WorkBench podemos obter o esquema físico seguinte, que representa os anteriores modelos em código SQL. Aqui definimos os atributos que se vão comportar como *foreign keys* e *primary keys*, bem como criar as tabelas e os seus restantes atributos. Notar que aqui apenas mostramos uma parte deste esquema.

```
-- -----------------------------------------------------
-- Table `Testes_Clinicos_Atletismo`.`Equipa`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `Testes_Clinicos_Atletismo`.`Equipa` (
  `idEquipa` INT NOT NULL AUTO_INCREMENT,
  `nome` VARCHAR(100) NOT NULL,
  PRIMARY KEY (`idEquipa`))
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `Testes_Clinicos_Atletismo`.`Codigo_Postal`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `Testes_Clinicos_Atletismo`.`Codigo_Postal` (
  `idCodigo_Postal` INT NOT NULL AUTO_INCREMENT,
  `Localidade` VARCHAR(100) NULL,
  `codigo` VARCHAR(80) NULL,
  PRIMARY KEY (`idCodigo_Postal`))
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `Testes_Clinicos_Atletismo`.`Escalao`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `Testes_Clinicos_Atletismo`.`Escalao` (
  `idEscalao` INT NOT NULL AUTO_INCREMENT,
  `descricao` VARCHAR(50) NOT NULL,
  PRIMARY KEY (`idEscalao`))
```

Figura 3 - Esquema Físico.

3. Funcionamento do SBD Relacional e *Queries*

Após o desenvolvimento do SBD e dos seus esquemas podemos começar a fazer povoamento da nossa Base de Dados e começar a testar o seu funcionamento. Para isso, precisamos de inserir dados nas tabelas criadas (atletas, equipas, testes clínicos, …). Antes da inserção o nosso grupo investigou na área de Atletismo vários atletas de renome nacional e mundial, as suas atuais equipas, localidades onde moram, etc., com vista a dar um valor real aos dados inseridos na nossa Base de Dados e tornar a nosso trabalho mais fiel, pois esta BD tem como finalidade a sua utilização por parte de uma Clínica, os seus atletas e médicos.

Assim, seguem-se alguns exemplos de inserções que efetuamos:

```
insert into Codigo_Postal (Localidade, codigo) values("Martinique", "0034"); -- 38
insert into Codigo_Postal (Localidade, codigo) values("Port of Spain", "0035"); -- 39
insert into Codigo_Postal (Localidade, codigo) values("Oliveira de Azeméis", "3720"); -- 40
insert into Codigo_Postal (Localidade, codigo) values("Brazzaville", "0037"); -- 41
insert into Codigo_Postal (Localidade, codigo) values("Beja", "7800"); -- 42
insert into Codigo_Postal (Localidade, codigo) values("Guimarães", "4800"); -- 43
insert into Codigo_Postal (Localidade, codigo) values("Setúbal", "2900"); -- 44
insert into Codigo_Postal (Localidade, codigo) values("São Jorge de Arroios", "1072"); -- 45
insert into Codigo_Postal (Localidade, codigo) values("Laikipia County", "0038"); -- 46
insert into Codigo_Postal (Localidade, codigo) values("Havana", "0039"); -- 47


-- Povoamento Escalao
insert into Escalao (descricao) values("Senior");
insert into Escalao (descricao) values("Juniores(Sub20)");
insert into Escalao (descricao) values("Juvenil(Sub18)");


-- Povoamento Atleta
insert into Atleta values(999, "Usain Bolt", "usainbolt@bolt.com","1986-08-21", "Jamaica", "Masculino", 6, 1, 1);
insert into Atleta values(0, "Sergio Vieira", "serpor@outlook.com", "1976-02-20", "Portugal", "Masculino", 1, 3, 1);
insert into Atleta values(1, "Pedro Isidro", "50kmped@gmail.com", "1985-07-17", "Portugal", "Masculino", 3, 3, 1);
insert into Atleta values(2, "Ricardo Ribas", "ricpor@outlook.com", "1997-10-08", "Portugal", "Masculino", 4, 2, 1);
insert into Atleta values(3, "Tsanko Arnaudov", "tsa14@hotmail.com", "1992-03-14", "Portugal", "Masculino", 3, 3, 1);
```

Figura 4 - Povoamento da Base de Dados.

De seguida, já com a base de dados povoada, conseguimos demonstrar algumas das funcionalidades que a nossa BD proporciona com as seguintes *queries*:

Funções implementadas:

- ➔ **Função que indica o total faturado pelo medico X nos seus exames.** Esta *query* pode ser útil para a clínica no que conta a determinar os médicos que geram mais receitas.

```
delimiter //
create function fatMedico(medico varchar(100)) returns float
begin
return(
select coalesce(sum(e.preco),0) as Total_Faturado from exame e, medico
m
where m.nif_Medico = e.nif_Medico and m.nome = medico
);
end//
delimiter ;
```

Triggers implementados:

- ➔ ***Trigger* que faz um desconto para 10 euros caso o preço do exame seja superior a 25€.** Possibilita à clínica efetuar mudanças no preço dos exames através de descontos.

```
create table consultasDesconto (descricao varchar(100), idExame int,
preco int, dataDesconto datetime);

delimiter //
create trigger DescontoExame after update on exame
      for each row
      begin
            if(OLD.preco > 25 and new.preco = 10) then
                  insert into consultasDesconto values(new.descricao,
new.idExame, new.preco, now());
            end if;
      end //
```

```
delimiter ;
```

➔ ***Trigger*** **que cria uma tabela com os atletas que foram aprovados.** Permite à clinica aceder facilmente aos atletas aprovados pelos exames.

```
create table atletasAprovados (nome varchar(100), genero varchar(45),
equipa varchar(100));

delimiter //
create trigger AprovaAtleta after insert on exame
      for each row
    begin
      if(new.descricao = "Aprovado") then
            insert into atletasAprovados
                  select atleta.nome, atleta.genero, equipa.nome
                  from atleta, equipa, marcacao, teste_clinico
                  where atleta.nif_Atleta = marcacao.nif_Atleta
                        and       teste_clinico.idTeste_Clinico       =
marcacao.idTeste_Clinico
                        and       teste_clinico.idTeste_Clinico       =
new.idTeste_Clinico
                        and equipa.idEquipa = atleta.idEquipa
                        and new.descricao = "Aprovado";
            end if;
    end //

delimiter ;
```

Procedures implementados:

- ➔ ***Procedure* que dado um médico, indica os atletas que tiveram consultas com ele.** Garante a lista de atletas que tiveram consultas com um determinado médico à escolha.

```
delimiter //
create procedure AtletasComMedico(in medicoX varchar(100))
begin
SELECT
    atleta.nome AS NomeAtleta,
    marcacao.data_Agendada AS DataMarcacao
FROM
    medico
        INNER JOIN
    marcacao ON marcacao.nif_Medico = medico.nif_Medico
        AND medico.nome = medicoX
        INNER JOIN
    atleta ON atleta.nif_Atleta = marcacao.nif_Atleta;
end//

delimiter ;
```

- ➔ ***Procedure* com cursor que agenda uma marcação em atletas de uma Equipa.** Possibilita o agendamento de testes a vários atletas pertencentes à mesma equipa, com uma data e médico à escolha.

```
delimiter //
create procedure marcaEquipa(in nome_equipa varchar(100), nMedico
varchar(100), dataM varchar(100))
begin
   declare finished integer default 0;
    declare nifA int;
    declare atletaCursor cursor for
         select a.nif_Atleta from atleta a, equipa e
        where a.idEquipa = e.idEquipa and e.nome = nome_equipa;

    declare continue handler for not found set finished = 1;
```

```
    open atletaCursor;
    getAtleta: loop
         fetch atletaCursor into nifA;
        if finished = 1 then leave getAtleta;
         end if;
        insert into teste_clinico values();
        set @lastIdTeste = last_insert_id();
        insert    into    marcacao    (data_Agendada,    nif_Atleta,
nif_Medico,    idTeste_Clinico)    values(dataM,    nifA,    nMedico,
@lastIdTeste);
   end loop getAtleta;
    close atletaCursor;
end //
delimiter ;
```

➔ ***Procedure*** **que indica se um Atleta se encontra apto pelo resultado da última marcação.** Visto que o resultado de um teste clínico, na nossa base de dados, é um conjunto de resultados dos exames feitos, um atleta está apto para a prática da modalidade caso passe a todos exames.

```
delimiter //
create procedure isApto(in nome_Atleta varchar(100))
begin
   select  e.descricao  as  "Status",  t.data  as  "Data  do  Teste
Clínico"
   from Exame e, Teste_Clinico t, Atleta a, Marcacao m
   where a.nome = "Tsanko Arnaudov" and m.nif_Atleta = a.nif_Atleta
   and  e.idTeste_Clinico  =  t.idTeste_Clinico  and  t.idTeste_Clinico
= m.idTeste_Clinico
   order by t.data desc, e.descricao desc
    limit 1;
end //
delimiter ;
```

➔ ***Procedure*** **que muda a Equipa de um atleta.** Permite à clinica fazer a atualização de dados do atleta, neste caso atualiza a equipa a que o atleta pertence.

```
delimiter //
create procedure mudarEquipaAtleta (in nome_atleta varchar(45),
                                    in nova_equipa int)
begin
   declare v_error bool default 0;
   declare continue handler for sqlexception set v_error = 1;

   set autocommit = OFF;
   start transaction;

    update
         Atleta
   set idEquipa = nova_equipa
    where nome = nome_atleta;

   if (v_error) then rollback;
    end if;

    commit;
end //
delimiter ;
```

➔ ***Procedure*** **que adiciona dados ao teste clínico agendado.** Depois do agendamento de testes clínicos, é necessário poder atualizar os testes criados nesse agendamento.

```
delimiter //
create procedure atualizaTeste (idTeste int,
                            naltura int, npeso int, press int, freq
int, imc int, ndata datetime)
begin
      declare v_error bool default 0;
      declare continue handler for sqlexception set v_error = 1;
```

```
      set autocommit = OFF;
      start transaction;

    update
            teste_clinico
      set altura = naltura, peso = npeso, pressao_arterial = press,
    freq_cardiaca = freq, indice_massa_corporal = imc, data = ndata
    where idTeste_Clinico = idTeste;

      if (v_error) then rollback;
    end if;
    commit;

end //
delimiter ;
```

Views desenvolvidas:

➔ **View que apresenta os atletas da modalidade "Corrida de pista".**

```
CREATE VIEW AtletasCorridaPista AS
    SELECT
        a.nome AS Nome,
        c.descricao_Categorias AS Categoria,
        m.nome AS Modalidade
    FROM
        atleta a,
        categorias_modalidade c,
        modalidade m
    WHERE
        a.nif_Atleta = c.nif_Atleta
            AND c.idModalidade = m.idModalidade
            AND m.nome = 'Corrida de Pista';
```

➔ ***View* que indica os Atletas da Localidade "Lisboa".** Permite à clínica obter todos os Atletas que moram em Lisboa.

```
CREATE VIEW AtletasLisboa AS
    SELECT
```

```
        a.nome AS Nome, e.nome AS Equipa, c.codigo AS Codigo_Postal
    FROM
        atleta a,
        codigo_postal c,
        equipa e
    WHERE
        a.idCodigo_Postal = c.idCodigo_Postal
            AND c.Localidade = 'Lisboa'
            AND e.idEquipa = a.idEquipa;
```

➔ ***View* que conta o número de atletas e médicos em cada Localidade.** Enumera o número de Atletas e Médicos para cada Localidade da Base de Dados. Permite à clinica ter uma noção da distribuição de utentes (atletas) e médicos a nível geográfico.

```
CREATE VIEW NLocalidade AS
    SELECT
        c.Localidade AS Localidade,
        QTSMEDICOSLOCALIDADE(c.Localidade) AS Medicos,
        QTSATLETASLOCALIDADE(c.Localidade) AS Atletas
    FROM
        atleta a,
        codigo_postal c,
        medico m
    GROUP BY c.Localidade;
```

➔ ***View* que apresenta os Médicos com mais Marcações.** Possibilita à Clínica saber qual o Médico mais requisitado.

```
CREATE VIEW MedicoMaisMarcacoes AS
    SELECT
        m.nome AS Nome, IDADE(m.data_nascimento) AS Idade,
        count(ma.idTeste_Clinico) as NumeroConsultas
    FROM
        medico m,
        marcacao ma
    WHERE
        ma.nif_Medico = m.nif_Medico
```

```
    GROUP BY ma.nif_Medico
    ORDER BY ma.nif_Medico ASC
    LIMIT 1;
```

➔ ***View* que indica os Atletas da Equipa "Nike".** Permite à clínica conhecer todos atletas que pertencem à equipa “Nike”, uma das que tem mais atletas pertencentes à Base de Dados.

```
CREATE VIEW AtletasNike AS
    SELECT
        Atleta.nome AS Nome,
        Atleta.pais AS País,
        Atleta.genero AS Género
    FROM
        Atleta
            INNER JOIN
        Equipa ON Atleta.idEquipa = Equipa.idEquipa
            AND Equipa.nome = 'Nike'
    ORDER BY Atleta.pais ASC;
```

➔ ***View* que seleciona todos atletas com marcações futuras.** Dá a conhecer à clínica todos atletas que têm um teste clínico agendado.

```
create view MarcacaoFutura
as
SELECT
    a.nome AS 'Nome do Atleta',
    m.data_Agendada AS 'Teste agendado, mas não realizado'
FROM
    Marcacao m
        INNER JOIN
    Atleta a ON a.nif_Atleta = m.nif_Atleta
        INNER JOIN
    Teste_Clinico t ON m.idTeste_Clinico = t.idTeste_Clinico
        LEFT JOIN
    Exame e ON e.idTeste_Clinico = t.idTeste_Clinico
WHERE
    e.idExame IS NULL;
```

Usamos também um conjunto de *queries* auxiliares e alguns exemplos de consultas básicas, como vemos de seguida:

➔ **Função que calcula a idade de uma pessoa (atleta ou médico).**

```
delimiter //
create function idade(dat date) returns int
begin
return (TIMESTAMPDIFF(YEAR, dat, CURDATE()));
end //
delimiter ;
```

➔ **Função que devolve o número de atletas que vivem na localidade X.**

```
create function qtsAtletasLocalidade(loc varchar(100)) returns int
begin
return(
SELECT
    COUNT(*)
FROM
    atleta a,
    codigo_postal c
WHERE
    a.idCodigo_Postal = c.idCodigo_Postal
        AND c.Localidade = loc
);
end//
delimiter ;
```

➔ **Função que diz o número de médicos na localidade X.**

```
create function qtsMedicosLocalidade(loc varchar(100)) returns int
begin
return(
SELECT
    COUNT(*)
```

```
FROM
    medico m,
    codigo_postal c
WHERE
    m.idCodigo_Postal = c.idCodigo_Postal
        AND c.Localidade = loc
);
end//
delimiter ;
```

➔ ***Query*** **que seleciona alguns dados relativos a um Atleta.** Uma consulta de onde se obtem alguns dados relativos ao atleta “Tomás Azevedo”

```
select a.nome as Nome, e.nome as Equipa from atleta a, equipa e
where a.nome = "Tomás Azevedo" and a.idEquipa = e.idEquipa;
```

➔ ***Query*** **que seleciona os atletas com mais de 1.80 metros de altura.** Esta *query* garante uma consulta simples a nível de Atletas e os seus dados de testes clínicos;

```
SELECT
    a.nome AS Nome, t.altura AS Altura
FROM
    atleta a,
    marcacao m,
    teste_clinico t
WHERE
    m.idTeste_Clinico = t.idTeste_Clinico
        AND m.nif_Atleta = a.nif_Atleta
        AND t.altura > 180;
```

➔ ***Query* que consulta os preços pagos por cada Atleta nos seus testes clínicos.** Esta *query* poderá ser útil a nível de estatísticas monetárias, tomadas de decisão a nível da própria clínica, …

```
select a.nome as "Nome do Atleta",
    m.data_Agendada as "Data agendada para o Teste Clínico",
    coalesce(sum(e.preco),0) as "Total pago"
from Marcacao m
inner join Atleta a on a.nif_Atleta = m.nif_Atleta
inner join Teste_Clinico t on m.idTeste_Clinico = t.idTeste_Clinico
left join Exame e on e.idTeste_Clinico = t.idTeste_Clinico
group by a.nome, m.data_Agendada
order by sum(e.preco);
```

4. Elementos de Informação, Recursos e Serviços

Primeiramente para a implementação da nossa Base de Dados é necessário a criação de uma *app* com uma interface *user friendly* que permita aos seus utilizadores interagirem com a mesma sem grandes dificuldades.

Esta base de dados teria de estar armazenada num servidor ao qual os utilizadores da BD iriam aceder para manipular esses mesmos dados, claro que com validação do acesso destes para garantir uma maior segurança e persistência dos dados.

Em termos de recursos físicos, para além do servidor, seria necessário um edifício (clínica) no qual operavam todos os intervenientes da base de dados, onde seriam realizados os vários testes clínicos e a recolha das informações pertinentes dos vários atletas. Uma estrutura destas envolveria ainda assim uma grande quantidade de recursos humanos, tais como médicos, rececionistas, um administrador e os próprios atletas.

No que toca aos rececionistas teriam de ter ao seu dispor um computador que lhes permitisse aceder à base de dados e credenciais de acesso à mesma. Estes poderiam marcar ou desmarcar a realização de testes clínicos e inserção de novos atletas no que toca ao seu acesso à base de dados. O administrador, como principal gestor da base de dados, tem como serviços a sua manutenção e garantia da utilidade dos dados inseridos e existentes. Seria ainda capaz de remover e inserir médicos, atletas e acesso total a todas as suas informações. Relativamente aos médicos estes iriam realizar os testes clínicos e os exames correspondentes aos mesmos. Seria assim responsável por toda a recolha de dados referentes aos atletas e respetiva inserção dos mesmos na base dados. Para tal estes teriam de ter ao seu dispor aparelhos que permitissem a realização dos exames. Os atletas seriam assim o recurso essencial à

existência da base de dados visto que estes fornecem toda a sua informação útil e ainda assim aqueles que apenas podem requisitar um teste clínico.

5. Implementação do SBD não relacional

À medida que quantidade de dados vai crescendo um SGDB relacional começa a piorar a nível de performance devido à sua estrutura. Por isto mesmo é necessário, por vezes, mudar o sistema da base de dados para uma não relacional (NoSQL) para armazenar e recuperar esta grande quantidade de dados. Este novo SGBD não relacional proporciona, assim, a vantagem de consultas rápidas em relação ao anterior sistema.

Nesta parte do projeto tivemos de proceder a uma migração de dados do nosso SDB relacional para uma não relacional, o Neo4j.

5.1. Processo de migração de dados para o novo sistema não relacional

Para procedermos à migração de dados, precisávamos de guardar os dados da BD anterior para depois os descarregarmos na nova BD.

Para tal, efetuamos um conjunto de *queries* em SQL para o selecionamento desses dados e de seguida guardamo-los em ficheiros “csv”, ou seja, os dados relativos aos Atletas foram guardados num ficheiro “atleta.csv” através da *query* “`select * from atleta;`” e o mesmo para as restantes entidades.

Assim, podemos começar a fazer a migração de dados para o novo sistema não relacional. Para isso usamos *queries* em *cypher*, linguagem do neo4j, como nos exemplos a seguir:

Carregamento dos dados dos Atletas:

```
load csv with headers from "file:///atleta.csv" as row
create (:Atleta {nif_Atleta: toInteger(row.nif_Atleta), nome: row.nome, data_nascimento: row.data_nascimento, pais: row.pais, genero: row.genero, idEquipa: toInteger(row.idEquipa), idCodigo_Postal: toInteger(row.idCodigo_Postal), idEscalao: toInteger(row.idEscalao)});
```

Carregamento dos dados das Equipas:

```
load csv with headers from "file:///equipa.csv" as row
create (:Equipa {idEquipa: toInteger(row.idEquipa), nome: row.nome});
```

Após o carregamento dos dados de todas entidades e a criação dos nodos correspondentes, criamos *constraints* nos identificadores de cada nodo (entidade) para que cada um deles seja único:

```
create constraint on (a:Atleta) assert a.nif_Atleta is unique;
create constraint on (e:Equipa) assert e.idEquipa is unique;
…
```

De seguida, já estamos em condições de criar as relações entre nodos tendo em conta o esquema lógico definido anteriormente no sistema relacional.

Assim sendo, pegamos nesse esquema lógico e a entidade Categorias_Modalidade deixa de existir passando a ser uma relação com atributos do Atleta para Modalidade. O mesmo acontece para a entidade “Marcacao” que é agora uma relação com atributos de Atleta para Teste_Clinico. O resto das relações mantêm-se, ficando agora apenas com um nome a identificá-las.

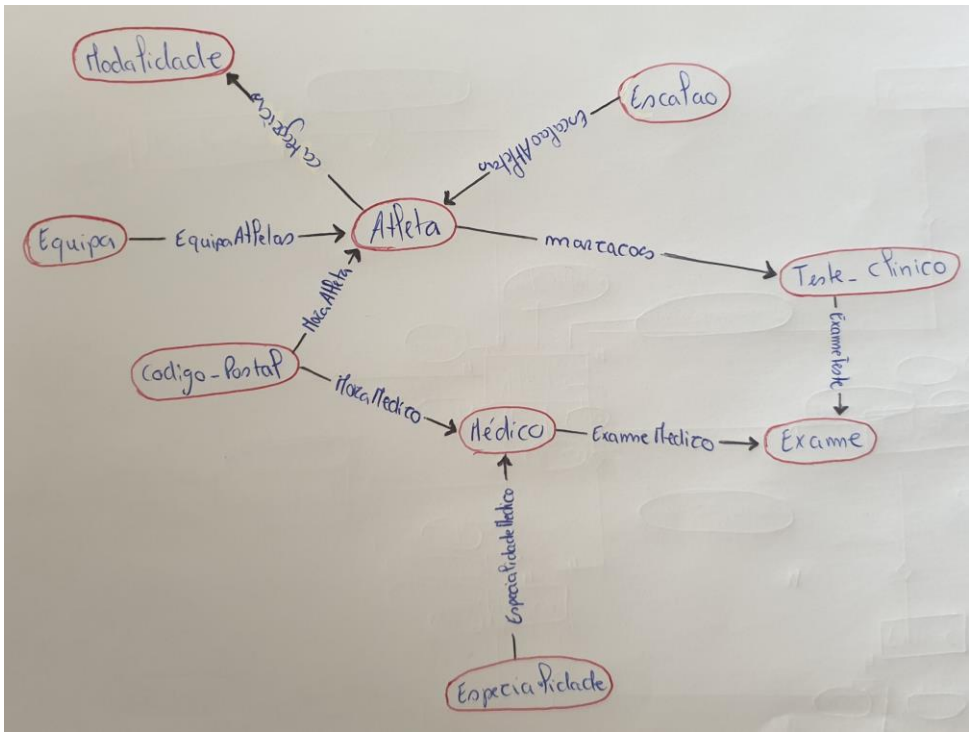

Figura 5 - Esquema do Sistema não Relacional.

Em suma, temos tudo necessário para a criação das relações a nível de código (*cypher*), sendo esta correspondente aos seguintes comandos:

- **Relação entre o Atleta e a Equipa**

```
match (a:Atleta)
match (e:Equipa)
where a.idEquipa = e.idEquipa
merge (e)-[r:EquipaAtletas]-(a);
```

- **Relação entre o Atleta e o Escalão**

```
match (a:Atleta)
match (e:Escalao)
where a.idEscalao = e.idEscalao
merge (e)-[r:EscalaoAtletas]-(a);
```

- **Relação entre o Atleta e a Modalidade**

```
match (a:Atleta)
match (m:Modalidade)
match (c:Categorias_Modalidade)
where c.idModalidade = m.idModalidade
and a.nif_Atleta = c.nif_Atleta
merge (a)-[r:Categorias]-(m)
on create set r.descricao = c.descricao_Categorias;
```

- **Relação entre o Atleta e o Código Postal**

```
match (a:Atleta)
match (c:Codigo_Postal)
where a.idCodigo_Postal = c.idCodigo_Postal
merge (c)-[r:MoraAtleta]-(a);
```

- **Relação entre o Atleta e o Teste Clínico**

```
match (a:Atleta)
match (me:Medico)
match (t:Teste_Clinico)
match (m:Marcacao)
where a.nif_Atleta = m.nif_Atleta and me.nif_Medico = m.nif_Medico
and t.idTeste_Clinico = m.idTeste_Clinico
merge (a)-[ma:Marcacoes]-(t)
on create set ma.data_Agendada = m.data_Agendada,
ma.nif_Medico = me.nif_Medico;
```

- **Relação entre o Médico e o Código Postal**

```
match (m:Medico)
match (c:Codigo_Postal)
where m.idCodigo_Postal = c.idCodigo_Postal
merge (c)-[r:MoraMedico]-(m);
```

- **Relação entre o Médico e o Exame**

```
match (e:Exame)
match (m:Medico)
```

```
where m.nif_Medico = e.nif_Medico
merge (m)-[r:ExameMedico]-(e);
```

- **Relação entre o Médico e a Especialidade**

```
match (m:Medico)
match (e:Especialidade)
where m.idEspecialidade = e.idEspecialidade
merge (e)-[r:EspecialidadeMedico]-(m);
```

- **Relação entre o Teste Clínico e o Exame**

```
match (t:Teste_Clinico)
match (e:Exame)
match (m:Medico)
where t.idTeste_Clinico = e.idTeste_Clinico and m.nif_Medico =
e.nif_Medico
merge (t)-[ex:ExameTeste]-(e);
```

Após a criação das relações encontramo-nos com o SGBD não relacional estruturado consoante a arquitetura do SGBD relacional anteriormente desenvolvido. Para finalizar a implementação deste sistema não relacional, falta-nos então o desenvolvimento de *queries* capazes de demonstrar a funcionalidade do nosso SGBD.

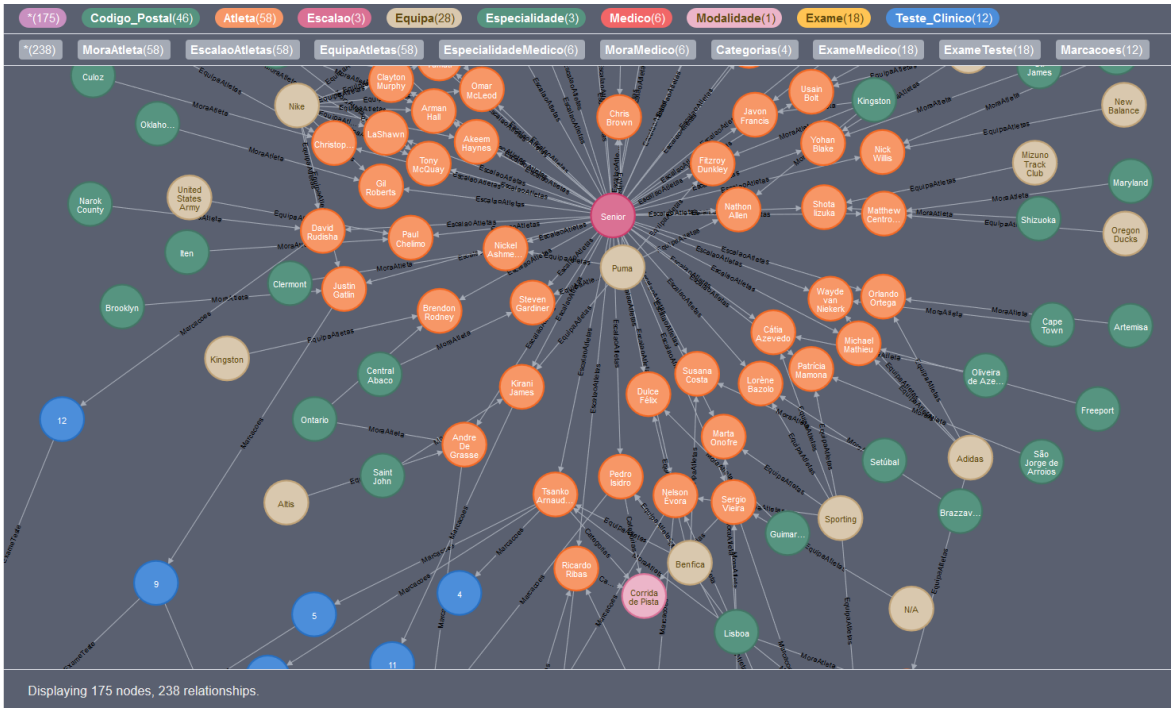

Figura 6 - Grafo do Sistema não Relacional

5.2. Demonstração da operacionalidade do sistema não relacional implementado

Depois de demonstradas as várias funcionalidades que o nosso sistema relacional permite obter, temos agora de provar o mesmo para este novo SGBD após a migração de dados.

Para isso, iremos implementar um conjunto de *queries* equivalentes àquelas já desenvolvidas e descritas. De realçar que neste GDBMS as *queries* funcionam como consultas à Base de Dados e, ao contrário de um SGBD Relacional, não possibilita *triggers*, *functions*, *procedures*, …

Por isso mesmo, apenas realizámos *queries* que no nosso entender fariam sentido neste novo sistema não relacional.

a. Marcações do atleta com NIF ‘3’

```
match (a:Atleta {nif_Atleta: 3})-[m:Marcacoes]-()
return a.nome as Nome, m.data_Agendada as Data;
```

b. Todos atletas que foram aprovados em Exames

```
match (a:Atleta)-[m:Marcacoes]-(t:Teste_Clinico)-[r:ExameTeste]-
(e:Exame {descricao: "Aprovado"})
return a.nome as Nome, t.data as Data, e.nome as Exame;
```

c. Lista de Atletas da Equipa Nike

```
match (e:Equipa)-[ea:EquipaAtletas]-(a:Atleta)
where e.nome = "Nike"
return a.nome as Atleta;
```

d. Atletas com mais de 180 cm

```
match (a:Atleta)-[:Marcacoes]-(t:Teste_Clinico)
where t.altura > 180
return distinct a.nome as Atleta;
```

e. Dado um médico, indica os atletas que tiveram consultas com ele

```
match (a:Atleta)-[:Marcacoes {nif_Medico: 1111111111}]-
(t:Teste_Clinico)
return distinct a.nome as Atleta;
```

f. Seleciona todos atletas com marcações futuras

```
match (t:Teste_Clinico)<-[:Marcacoes]-(a:Atleta)-[:Categorias]-
>(m:Modalidade)
where (t.altura is null)
return a.nome as Atleta, m.nome as Modalidade,
duration.between(date(a.data_nascimento), date()).years as Idade;
```

g. Apresenta os atletas que são da modalidade "Corrida de Pista"

```
match (a:Atleta)-[c:Categorias]-(m:Modalidade {nome: "Corrida de
Pista"})
return a.nome as Atleta, c.descricao as Categoria;
```

h. Conta o número de atletas e médicos em cada Localidade

```
match (m:Medico)-[:MoraMedico]-(c:Codigo_Postal)
return c.localidade as Localidade, count(m.nome) as Medicos;

match (a:Atleta)-[:MoraAtleta]-(c:Codigo_Postal)
return c.localidade as Localidade, count(a.nome) as Atletas;
```

i. Apresenta o Médico com mais Marcações

```
match ()-[m:Marcacoes]-(t:Teste_Clinico)
match (me:Medico {nif_Medico: m.nif_Medico})
return me.nome as Medico, count(t.idTeste_Clinico) as NumeroConsultas
order by NumeroConsultas desc
limit 1;
```

j. Indica se um Atleta se encontra apto pelo resultado da última marcação

```
match (a:Atleta {nome: "Tsanko Arnaudov"})-[ma:Marcacoes]-
>(t:Teste_Clinico)-[et:ExameTeste]->(e:Exame)
return a.nome as Nome, t.data as Data, e.descricao as Resultado
order by t.data desc, e.descricao desc
limit 1;
```

k. Agenda uma marcação para atletas de uma Equipa

```
match (e:Equipa {nome: "Sporting"})-[eq:EquipaAtletas]-(a:Atleta)
merge (a)-[m:Marcacoes {data_Agendada: "2020-02-10", nif_Medico:
1111111113}]->(t:Teste_Clinico);
```

l. Apresenta o total pago por teste clínico (soma dos exames) por atleta, ordenado pelo total pago

```
match (a:Atleta)-[m:Marcacoes]-(t:Teste_Clinico)-[:ExameTeste]-
(e:Exame)
return a.nome, sum(e.preco) as TotalPago
order by sum(e.preco);
```

m. Atualiza os valores de um teste clínico agendado

```
match (t:Teste_Clinico {idTeste_Clinico: 4})
set t.altura = 178, t.peso = 78, t.pressao_arterial = 10,
t.freq_cardiaca = 80, t.indice_massa_corporal = 20, t.data = "2019-12-
31T10:00:00";
```

Juntamente a estas *queries* implementadas, usámos algumas consultas auxiliares, tais como:

i. Calcular o número de atletas que moram na localidade “Lisboa”

```
match (c:Codigo_Postal {localidade: "Lisboa"})-[:MoraAtleta]-
(a:Atleta)
return count(a);
```

ii. Calcular o número de médicos que moram na localidade “Famalicão”

```
match (c:Codigo_Postal)-[:MoraMedico]-(m:Medico)
where c.localidade = "Famalicão"
return count(m);
```

iii. Calcula a idade de uma pessoa

```
return duration.between(date("1999-10-15"), date()).years;
```

Na maior parte das *queries* apresentadas não generalizamos os argumentos embutidos, pois a linguagem *cypher* não proporciona tal funcionalidade. Assim, as *queries* são exemplos da funcionalidade e não um caso geral de uso.

Com isto, garantimos a operacionalidade do novo SGBD não relacional tal como já o tínhamos garantido para o antigo SGBD.

Constatámos que o novo sistema apresenta uma maior simplicidade na consulta de dados em relação ao anterior, resultado este já esperado devido às diferenças dos dois sistemas anteriormente discutidas. Contudo, de realçar que o sistema relacional se destaca na parte de inserção de dados e estruturação destes.

6. Conclusões e Trabalho Futuro

Em suma, achamos que o trabalho cumpre os objetivos propostos na sua totalidade. Contudo, assumimos que não é perfeito, pelo que podemos melhorá-lo em certos aspetos, tais como, os identificadores das entidades dos dois sistemas, pois poderíamos ter usado outras propriedades como identificadores e, assim, poupar espaço a nível de memória que é um recurso essencial numa Base de Dados. Achamos que temos espaço para evolução a nível da migração de dados e nas *queries* de ambos sistemas e que com mais algum tempo poderíamos ter um trabalho mais estruturado e aplicável ao mundo real.

A nível de trabalho futuro seria necessário limar algumas limitações de performance e, por fim, ligar a Base de Dados a uma aplicação para os utentes da clínica a usarem e usufruírem da finalidade desta BD.

Referências

Connolly, T., Begg, C., 2004. Database Systems, A Practical Approach to Design, Implementation, and Management, Addison - Wesley, 4aEdição.

Pereira, J. L., 2006. Tecnologia de Bases de Dados, FCA - Editora de Informática 4ª Ed.

Elmasri, R. & Navathe, S.B., 2010- Fundamentals of Database Systems, Addison-Wesley, 6ªEd.

Neo4j. Import Relational Data Into Neo4j. [online] Available at: <https://neo4j.com/developer/guide-importing-data-and-etl/> [Accessed 20 December 2019].

Lista de Siglas e Acrónimos

BD	**Base de Dados**
GDBMS	*Graph Database Management Systems*
SGBD	Sistema Gestão de Base de Dados
SBD	Sistema de Base de Dados
IMC	Índice de Massa Corporal
ACID	Atomicidade, Consistência, Isolamento e Durabilidade