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

Unidade Curricular de Bases de Dados

Ano Lectivo de 2019/2020

Trabalho Prático

Fábio Silva, Helena Martins, Pedro Medeiros, Carlos Gomes

Novembro, 2019

BD

Data de Recepção	
Responsável	
Avaliação	
Observações	

cLEInics

Fábio Silva, Helena Martins, Pedro Medeiros, Carlos Gomes

Novembro, 2019

Resumo

Área de Aplicação: Desenho e arquitetura de Sistemas de Bases de Dados.

Este documento retrata de forma aprofundada e completa a implementação de uma base de dados, tal como a sua organização. Esta suporta o funcionamento de uma aplicação que permite reservar testes clínicos para atletas de atletismo, organizadas por escalão e modalidade.
Numa primeira parte do relatório serão retratadas as fases iniciais antes da própria implementação da base de dados.
Posteriormente serão retratadas toas as entidades envolvidas assim como os atributos e a forma como as entidades estão relacionadas entre si. Este será o modelo lógico da nossa base de dados. Serão, também nesta fase inicial, identificadas as chaves primárias e estrangeiras.
De seguida, será apresentado o modelo lógico onde toda a estrutura da base de dados será implementada.
Por fim, os dados do modelo lógico serão analisados e validados através da normalização das relações existentes.

Palavras-Chave: Bases de Dados Relacionais, modelo conceptual e lógico, entidades, atributos, relacionamentos, chaves primaria e estrangeira.

Índice

Resumo	iii
Índice	iv
Índice de Figuras	vi
Índice de Tabelas	vii
1 - Introdução	1
1.1 - Contextualização	2
1.2 - Apresentação do Caso de Estudo	2
1.3 - Motivação e Objetivos	3
1.4 - Estrutura do Relatório	3
2 - Análise e Levantamento de Requisitos	5
2.1 - Requisitos de Descrição	5
2.1.1 - Requisitos de Exploração	7
2.1.2 - Requisitos de Controlo	8
3 - Modelação Conceptual	9
3.1 - Apresentação da abordagem de modelação realizada	9
3.2 - Identificação e caracterização das entidades	10
3.3 - Identificação e caracterização dos relacionamentos	12
3.4 - Identificação e caracterização das associações dos atributos com as entidades e relacionamentos	13
3.5 - Apresentação e explicação do diagrama ER	15
3.6 - Validação do modelo de dados com o utilizador	16
4 - Modelação Lógica	17
4.1. Construção e validação do modelo de dados lógico	17
4.3 - Validação do modelo através da normalização	19
4.4. Validação do modelo com interrogações do utilizador	20
4.5. Validação do modelo com as transações estabelecidas	21
4.6. Revisão do modelo lógico com o utilizador	21
5 - Implementação Física	22
5.1 - Seleção do sistema de gestão de base de dados	22
5.2 - Tradução do esquema lógico para o sistema de gestão de bases de dados escolhido em SQL	22
5.3 - Tradução das interrogações do utilizador para SQL	27
5.4 - Tradução das transações estabelecidas para SQL	30
5.5 - Escolha, definição e caracterização de índices em SQL	30
5.6 - Estimativa do espaço em disco da base de dados e taxa de crescimento anual	31
5.7 - Revisão do sistema implementado com o utilizador	34
6 - Abordagem Não Relacional	35

6.1 - Introdução 35
6.2 - Base de Dados NoSQL 35
6.2.1 - Principais características 36
6.2.2 - SQL vs NoSQL 37
6.2.3 - Vantagens 38
6.3 - Migração de dados 39
6.4 - Grafo resultante 40
6.5 - Tradução das interrogações do utilizador para Cypher 41
6.6 - Revisão do sistema implementado 44
6.7. Conclusões 44
Referências 45
Lista de Siglas e Acrónimos 46
Anexos 47
I. Anexo 48

Índice de Figuras

Figure 1 Modelo Conceptual 15
Figure 2 Modelo Lógico 19
Figure 3 Criação da tabela Modalidade 24
Figure 4 Criação da tabela Categoria 24
Figure 5 Criação da tabela Atleta 25
Figure 6 Criação da tabela Medico 25
Figure 7 Criação da tabela Consulta 26
Figure 8 Query 1 27
Figure 9 Query 2 27
Figure 10 Query 3 27
Figure 11 Query 4 28
Figure 12 Query 5 28
Figure 13 Query 6 28
Figure 14 Query 7 29
Figure 15 Query 8 29
Figure 16 Query 9 29
Figure 17 Transação 30
Figure 18 Grafo da Base de Dados Neo4j 40
Figure 19 Query 1 Neo4j 41
Figure 20 Query 2 Neo4j 41
Figure 21 Query 3 Neo4j 41
Figure 22 Query 4 Neo4j 42
Figure 23 Query 5 Neo4j 42
Figure 24 Query 6 Neo4j 42
Figure 25 Query 7 Neo4j 42
Figure 26 Query 8 Neo4j 43
Figure 27 Query 9 Neo4j 43

Índice de Tabelas

Tabela 1 Dicionário de relacionamentos do modelo conceptual 13

1 - Introdução

Neste trabalho é nos pedido que façamos uma análise e planeamento, seguidos da estruturação e implementação de um sistema de base de dados relacional, que sirva de suporte e otimize uma clínica de testes de atletas. Antes de tudo, já sabemos que a nossa base de dados será um sistema que gere informação de modo a guardá-la eficazmente e permite que o utilizador consiga extrair informação de forma rápida e concreta. A base de dados é constituída por tabelas e associações entre as mesmas. As linhas representam instâncias de entidades e as colunas os dados. Neste caso, a clínica vai centrar-se na base de dados, daí ser extremamente importante e fundamental haver um bom planeamento e estruturação desta, de modo a ter a eficácia desejada.

Posto isto, neste capítulo vamos introduzir o projeto em mente e a contextualização do caso em estudo.

No resto do relatório iremos apresentar o modelo conceptual e lógico e aprofundar os dois modelos de forma a explicá-los pormenorizadamente. Por fim, implementamos fisicamente os modelos apresentados através das relações existentes neles.

Por fim, apresentaremos neste relatório a abordagem não relacional deste trabalho, bem como a sua implementação e explicação.

1.1 - Contextualização

Um dos problemas de algumas clínicas que fazem testes a atletas de Atletismo é o facto de, como estas são consultas especializadas para atletas, terem de ser registadas separadas do resto de todas as consultas de uma clínica.

A maioria das clínicas que fazem este tipo de testes armazena toda a informação sobre os mesmos em arquivos físicos, o que não só requere muito mais tempo de procura, como também exige mais trabalho por parte dos funcionários da clínica. Com um sistema de base de dados que armazene a informação e a possa mostrar ao utilizador este problema pode ser resolvido. Assim, a câmara municipal de Braga decidiu investir numa mudança nas clínicas onde este problema estava presente e optar por contratar a nossa empresa no âmbito de criar um sistema de base de dados que suporte todos os requisitos solicitados.

1.2 - Apresentação do Caso de Estudo

A câmara pretende desenvolver um sistema de base de dados permita armazenar todos os testes clínicos efetuados nas clínicas de Braga. Com esta implementação, consegue minimizar-se quaisquer problemas de organização e em termos de eficiência no tempo de procura de informação é muito melhor.

Atualmente, o tempo é fundamental para tudo no nosso quotidiano e a falta dele é um problema recorrente na maioria das pessoas. Assim, decidimos criar e implementar um sistema que reduza principalmente o tempo despendido na organização e procura da informação acerca destes testes.

Com esta implementação, será possível efetuar uma gestão eficiente e rígida das consultas, dando uma visão geral e objetiva da informação, dando ao funcionário um conjunto de informações úteis para que tire o maior proveito dela.

1.3 - Motivação e Objetivos

Todas as equipas de qualquer desporto federado têm, pelo menos, uma consulta com fins de testes clínicos por época. Ora sendo tão grande a variedade de desportos, há, portanto, uma grande quantidade de consultas com esses fins em qualquer cidade. Desta forma, Braga não é exceção, daí fornecermos às clínicas um sistema que poupa bastante tempo e esforço para que estas possam ter mais tempo para outro tipo de atividades mais importantes.

Assim, a empresa vai proporcionar uma melhor qualidade de serviço nas clínicas de Braga, bem como uma melhor qualidade em termos de organização da própria clínica.

1.4 - Estrutura do Relatório

O conteúdo deste relatório vai conter uma descrição detalhada do caso de estudo e a elaboração e implementação da base de dados utilizada pela clínica que nos contactou.

No primeiro capítulo é feita uma introdução deste trabalho apresentando-se o caso de estudo, bem como os objetivos e motivação da clínica pelo desenvolvimento deste projeto.

No capítulo 2 é feita uma recolha de todos os requisitos, consistindo em apresentar e descrever todos os requisitos do sistema e de todas as entidades da base de dados em caso.

Posteriormente à análise dos requisitos, no terceiro capítulo apresentamos o modelo conceptual do caso. Aqui são identificadas as entidades e os atributos de cada entidade. São também analisados os relacionamentos e determinadas as chaves candidatas, primárias e alternadas. Por fim, são apresentadas as queries, perguntas feitas pelo utilizador, e é verificada a não existência de redundância.

Quanto ao modelo lógico, este é apresentado no capítulo 4. Inicialmente, dedicamo-nos à conversão do modelo conceptual num modelo lógico e à sua apresentação. Nos tópicos seguintes, é feita a análise das entidades e dos relacionamentos contidos no modelo. É também, por fim, feita a normalização, assim como a validação do modelo através de queries e a revisão do modelo.

No quinto capítulo, é apresentada a tradução do modelo lógico para o sistema escolhido, sendo esta a implementação da base de dados. São descritas, no início, as relações base, seguidas da tradução das queries para SQL. Finalmente, é feita a análise das transações, definidos os índices em SQL, caraterização das vistas e revisão de todo o sistema implementado.

No sexto capítulo, iremos apresentar as conclusões deste relatório, no que toca à parte relacional.

O sétimo e último capítulo conterá tudo o que diz respeito à parte não relacional do projeto. Isto é, apresentar a estrutura base para o sistema, assim como a identificação dos objetos de dados seguida da conversão e migração dos dados. Por fim, apresentamos as conclusões desta parte.

2 - Análise e Levantamento de Requisitos

Neste capítulo, são apresentados e definidos os requisitos necessários para o bom funcionamento do sistema.

Para uma base de dados ser bem implementada, é necessário conhecer na integra a aplicação e o resto, isto é, as suas funcionalidades e tudo aquilo com que interagem.

Além disto, o cliente pretende que a base de dados tenha as seguintes funcionalidades:

- Procurar todos os atletas com consultas marcadas
- Procurar todas as consultas pagas
    - Consulta mais lucrativa
    - Médico com mais consultas dadas
- Top 3 das modalidades com mais atletas
    - Modalidade com mais categorias
    - Número total de consultas existentes
- Atleta com mais consultas
- Modalidade com mais consultas

2.1 - Requisitos de Descrição

Após uma análise cuidadosa e criteriosa, decidimos reunir toda a informação que se mostrou relevante para a posterior construção da base de dados. A abordagem desses dados será feita tendo em conta os principais elementos deste sistema: o **atleta**, a **categoria**, a **consulta**, o **médico** e finalmente as **modalidades**.

I. **Atleta / Cliente**

É o grande motivo pelo qual uma clínica existe. As informações do atleta são absolutamente necessárias para que se lhe possa associar uma consulta. Por essa razão, decidimos que um atleta terá que fornecer o seu **nome**, o **escalão** a que pertence e finalmente o seu **contacto** (que será o seu telemóvel e e-mail).

I. Modalidade

No atletismo, tal como noutros desportos, temos modalidades pelas quais se organiza o desporto. Assim, há necessidade de termos a informação de modalidade de cada atleta. Estas podem ser: corrida, natação, lançamento e salto.
Esta entidade possui um identificador que permite identificar uma modalidade em específico.

- **Categoria**

Dentro de cada desporto existem modalidades e dentro de cada modalidade existem categorias. Com o fim de ter uma organização em relação a uma categoria em especifico, é criada esta entidade, que possui um identificador de categoria e um nome da categoria em si.

II. Consulta

Para uma clínica funcionar em condições, é preciso marcar uma consulta. Assim, dessa forma todos os atletas tem uma oportunidade de serem atendidos pelo médico e analisar o seu respectivo caso. Ora, para se marcar uma consulta, é preciso fornecer uma breve **descrição**, escolher um **horário** para ser atendido. Consequentemente, para uma consulta também é necessário saber a **duração** de uma consulta e o respectivo **custo**, e finalmente, saber se foi **pago** ou não.

III. Médico

O elemento crúcial para o funcionamento de uma clínica, sem um médico, não havia nenhum elemento com o conhecimento suficiente para realizar uma consulta.
Ora, mais uma vez, é preciso saber o respectivo **nome** do médico e a sua **especialidade**. Também decidimos adicionar a sua **reputação**.

2.1.1 - Requisitos de Exploração

Um dos objectivos de uma base de dados é ter acesso a informação requerida, nem mais, nem menos. Por exemplo, se um médico quiser obter a informação do seu doente, então deverá ser-lhe facultada a informação necessária como a descrição do seu caso e o nome do cliente, sem qualquer outra informação adicional.

Por conseguinte, em baixo, expomos alguns requisitos necessários à manipulação da base de dados.

- Para criarmos uma consulta, é necessário fornecer obrigatoriamente uma breve descrição e a hora de início da mesma
- Cada consulta só pode ser efetuada por um médico
- Para o atleta marcar uma consulta, tem de estar registado no sistema
- A base de dados tem de ter capacidade para suportar todos os dados das consultas e atletas e organizá-los de forma eficiente
- Tem de ser possível verificar se uma hora está disponível para a marcação de consultas, assim como verificar se uma consulta está paga
- As consultas de uma clínica são organizadas juntas para aumentar a eficiência de procura

2.1.2 - Requisitos de Controlo

Uma base de dados é, por definição, uma ferramenta que permite guardar e gerir da forma mais eficaz, toda a informação que um dado sistema informático possui. É importante que todos os utilizadores da aplicação possam aceder aos dados que lhe dizem ao respeito, mas, mais importante é garantir que esses utilizadores não acedam a informações alheias e desnecessárias.

Por essa razão, é imprescindível definir várias formas de monitorizar e restringir o acesso a vários dados por parte dos utilizadores.

- O atleta não poderá aceder à informação do médico da sua consulta excepto a informação relativa ao nome deste
- O médico não poderá aceder à informação relativa ao contacto e ao nif do atleta
- Apenas o administrador tem acesso a toda a informação da base de dados

3 - Modelação Conceptual

Depois de termos definidos os requisitos e ter sido feita uma análise detalhada dos mesmos, segue-se a modelação conceptual do sistema de base de dados.

Neste modelo, procuramos representar de forma fidedigna o modelo de informação usado pela empresa.

Para facilitar a interpretação do modelo, apresentamos o diagrama ER e um dicionário de dados sobre as entidades, relações e atributos identificados.

3.1 - Apresentação da abordagem de modelação realizada

O procedimento da modelação utilizada na realização do modelo seguiu os seguintes passos:

- Identificação dos tipos das entidades existente
- Identificação dos tipos de relacionamento
- Identificação e associação dos atributos a cada tipo de entidades ou relacionamentos
- Determinação do domínio dos atributos
- Determinação das chaves candidatas, primárias e estrangeiras
- Consideração do uso de detalhe ou generalização de entidades
- Verificação do suporte do modelo conceptual quanto às transações necessárias
- Validação do modelo de dados com o utilizador

Posto isto, foi possível definir o modelo conceptual.

3.2 - Identificação e caracterização das entidades

Para a realização do modelo conceptual, seguimos alguns passos fundamentais. O primeiro passo foi a definição dos tipos de entidades existentes.

Através da análise dos requisitos levantados, conseguimos identificar as entidades necessárias.

As entidades identificadas foram:

- **Atleta:**
  Cada atleta pode agendar várias consultas. Para tal, terá que estar registado na aplicação, inserindo os seus dados e as suas credenciais, que serão usadas para a sua autenticação. Cada Atleta tem o seu próprio número de identificação único (ID)

- **Consulta:**
  Cada consulta tem um conjunto de dados associados, tais como horário, duração, descrição, tipo e localidade.
  Também tem associado um custo fixo, a sua data prevista e um boolean que indica se esta foi paga ou não. Cada consulta tem um número de identificação (ID) associado.

- **Médico:**
  Cada médico realiza uma ou mais consultas, tendo a ele associado uma dada especialidade e reputação (avaliação dada pelos seus utentes). Também tema associado o seu nome e um número de identificação (ID).

- **Categoria:**
  Uma categoria tem associada a ela um número de identificação (ID) e um nome.

- **Modalidades:**
  Dentro de uma categoria, ela pode ser associada a uma modalidade. Esta tem um número de identificação (ID) e uma modalidade.

3.3 - Identificação e caracterização dos relacionamentos

O passo seguinte do processo de construção do modelo conceptual foi a identificação dos relacionamentos existentes entre as entidades das quais falamos anteriormente.

Os relacionamentos identificados foram os seguintes:

Atleta-Consulta:

Este relacionamento representa o agendamento de uma consulta por parte de um atleta.

Um atleta pode ter uma ou várias consultas agendadas. Cada consulta realizada está associada a um único atleta. Assim, estamos perante um relacionamento de 1 para N.

Médico-Consulta:

Uma consulta é realizada por apenas um médico. Porém, um médico pode realizar nenhuma ou várias consultas. Estamos, portanto, perante um relacionamento de 1 para N.

Atleta-Categoria:

Um atleta tem apenas uma categoria. No entanto, uma categoria é constituída por nenhum ou vários atletas. Assim, podemos considerar este relacionamento um relacionamento de N para 1.

Categoria-Modalidades:

Uma modalidade apenas pode ter uma categoria. Sendo assim, estamos perante um relacionamento de 1 para 1.

Dicionário de Relacionamentos do Modelo:

Entidade	Multiplicidade	Relacionamento	Multiplicidade	Entidade
Atleta	1	Agenda	N	**Consulta**
Médico	1	Realiza	N	**Consulta**
Atleta	1	Pertence	1	**Categoria**
Modalidade	1	Contém	1	**Categorias**

Tabela 1 Dicionário de relacionamentos do modelo conceptual

3.4 - Identificação e caracterização das associações dos atributos com as entidades e relacionamentos

Depois de identificadas todas as entidades envolvidas no nosso modelo, da relação entre elas e a sua multiplicidade, identificamos agora os atributos que sejam mais relevantes para a próxima etapa de desenvolvimento da nossa Base de Dados.

Entidade	Atributo	Descrição	Data Type	Null	Tipo de Atributo
Atleta	ID	Identificador do atleta	INT	Não	Chave Primária
	Nome	Nome do atleta	VARCHAR	Não	Simples
	Contacto Telemóvel Email	 Número de telemóvel Email do atleta	 INT VARCHAR(45)	 Não Não	Composto Simples Simples

	Escalão	Escalão etário do atleta	VARCHAR(15)	Não	Simples
Consulta	ID	Identificador da consulta	INT	Não	Chave Primária
	Descrição	Breve descrição da consulta	VARCHAR(45)	Não	Simples
	Duração	Duração da consulta	TIME	Não	Simples
	Horario	Horario de início da consulta	DATETIME	Não	Simples
	Custo	Custo da consulta	DECIMAL(5,2)	Não	Simples
	Pago	Variável binária que verifica se uma consulta está paga	BINARY(0)	Não	Simples
Médico	ID	Identificador do médico	INT	Não	Chave Primária
	Nome	Nome do médico	VARCHAR(32)	Não	Simples
	Especialidade	Especialidade do médico	VARCHAR(32)	Não	Simples
	Reputação	Reputação em termos qualitativos do médico	VARCHAR(10)	Não	Simples
Categoria	ID	Identificador de categoria	INT	Não	Chave Primária
	Nome	Nome da categoria	VARCHAR(45)	Não	Simples
Modalidade	ID	Identificador da modalidade	INT	Não	Chave Primária
	Modalidade	Nome da modalidade	VARCHAR(321	Não	Simples

3.5 - Apresentação e explicação do diagrama ER

Depois de se ter definido as entidades, os atributos e os relacionamentos necessários, foi possível chegarmos ao seguinte diagrama:

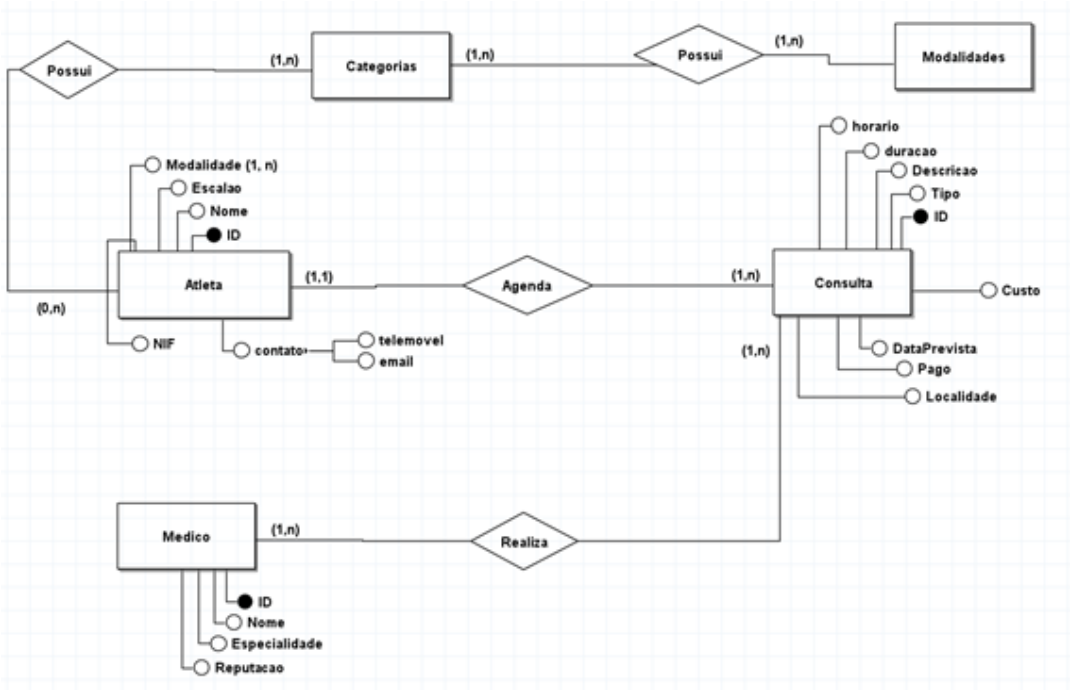

Figure 1 Modelo Conceptual

Como se pode observar através do diagrama, na entidade Atleta verifica-se a existência de três chaves candidatas: “**idAtleta**”, “**telemovel**” e “**email**”. Um atleta poderia ser identificado pelo seu **telemovel**, que é um valor único. No entanto, comparativamente ao atributo “**idAtleta**”, o seu valor numérico é muito maior. Tal acontece com o atributo “email”. Assim sendo, definimos o “**idAtleta**” como chave primária.

Na entidade Consulta, apenas o seu identificador, “idConsulta” serve como chave candidate, sendo automaticamente selecionado como chave primária. Tal também acontece com as restantes entidades: a chave primária da entidade Médico é o seu identificador “idMedico”, a chave primária da entidade Categoria é o seu identificador “idCategoria” e a chave primária da entidade Modalidades é o seu identificador “idModalidades”.

Assim sendo, temos o nosso modelo conceptual do projeto a desenvolver.

3.6 - Validação do modelo de dados com o utilizador

Depois de uma revisão feita pelo grupo de todos os passos deste capítulo, incluindo também a identificação e a caracterização das entidades e relacionamentos e respetivos atributos, não foram encontradas incoerências no modelo conceptual. Assim concluímos que o modelo representa o projeto que pretendemos modular.

4 - Modelação Lógica

Após o modelo conceptual ter sido validado pelo cliente, prosseguimos à definição do modelo lógico do sistema baseando através do modelo de dados relacional.

4.1. Construção e validação do modelo de dados lógico

Após a interpretação do modelo conceptual, realizado na secção anterior, convertemo-lo para o respetivo modelo lógico.

Em primeiro foram identificadas tanto as relações que representam entidades e relacionamentos, como os atributos das mesmas. Para isso foi usada DDL.

4.1.1 - Entidades Fortes

Durante a realização do modelo conceptual, concluiu-se que todas as entidades existentes são entidades fortes. Para além dos atributos simples e da chave primária, foram também identificadas algumas chaves alternativas para algumas entidades e os atributos compostos presentes no modelo, para que a fragmentação fosse possível. Assim, foram identificadas as seguintes estruturas de cada entidade:

- **Atleta**(idAtleta, Nome, Escalao, Telemovel, Email, Categoria_idCategoria)
  - **Chave Primária**: idAtleta
  - **Chave Estrangeira:** Categoria_idCategoria

- **Médic**o(idMedico, Nome, Especialidade, Reputação)
  - **Chave Primária:** idMedico
  - **Chave Alternativa**: Nome

- **Consulta**(idConsulta, Descrição, Horário, Duração, Custo, Pago, Medico_idMedico, Atleta_idAtleta)
  - **Chave Primária:** idConsulta
  - **Chave Estrangeira:** Medico_idMedico, Atleta_idAtleta

- **Categoria**(idCategoria, nome, Modalidade_idModalidade)
    - **Chave Primária**: idCategoria
    - **Chave Estrangeira:** Modalidade_idModalidade

- **Modalidade**(idModalidade, Modalidade)
    - **Chave Primária:** idModalidade
    - **Chave Alternativa:** Modalidade

4.1.2 - Relacionamentos binários

Relacionamentos de 1 para N

Neste tipo de relacionamento, a entidade de multiplicidade N tem um dos atributos que é considerado uma chave estrangeira, sendo esta a chave primária na entidade de multiplicidade 1. Temos assim relacionamentos deste tipo entre as entidades seguintes:

- Atleta (idAtleta,Nome, Escalao, Telemovel, Email, Categoria_idCategoria)
    Chave Primária (idAtleta)
    Chave Estrangeira (Categoria_idCategoria)

- Categoria (id_Categoria, Nome, Modalidades_idModalidades)
    Chave Primária (id_Categoria)
    Chave Estrangeira (Modalidades_idModalidades)

- Consulta (idConsulta, Descricao, Horario, Duracao, Custo, Pago, idAtleta, idMedico)
    Chave Primária (idConsulta)
    Chave Estrangeira (idAtleta)
    Chave Estrangeira (idMedico)

Relacionamentos de 1 para 1

Este tipo de relacionamento pode ser de participação dos dois lados ou apenas de um. No modelo lógico apresentado no próximo ponto do relatório, verificamos que

existia um relacionamento destes entre as entidades categoria e modalidades. Neste caso a entidade atleta fica com a chave estrangeira Modalidades_idModalidades que identifica a modalidade dentro de uma certa categoria de atletismo.

4.2 - Desenho do modelo lógico

Com as relações derivadas no ponto anterior, foi possível criar o seguinte modelo lógico:

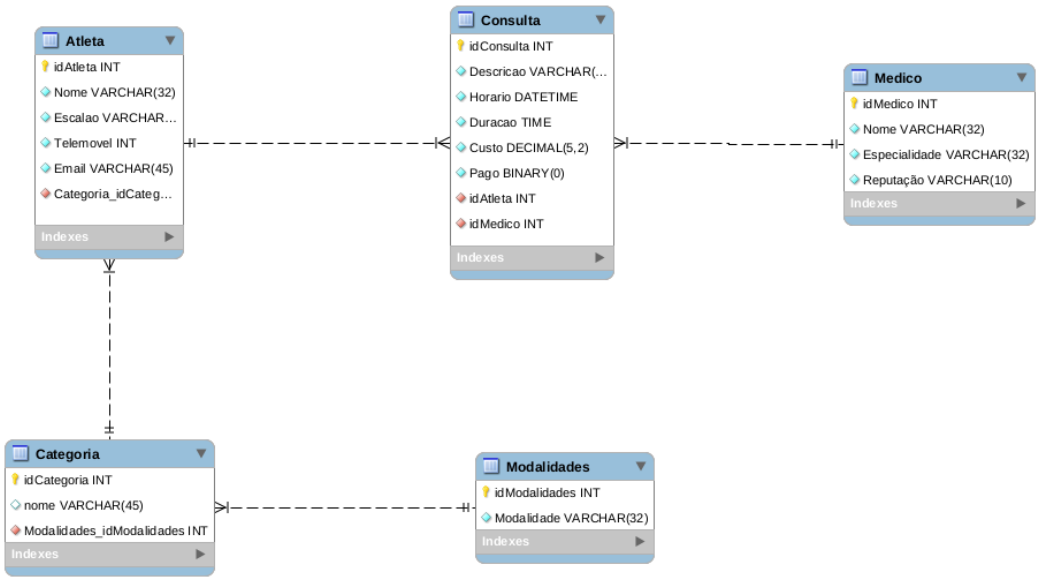

Figure 2 Modelo Lógico

4.3 - Validação do modelo através da normalização

A normalização de um modelo relacional é muito importante uma vez que garante que as relações tenham um número mínimo, mas necessário, de atributos, de modo a que todos os requisitos de dados da empresa sejam suportados. Além disso, permite que a redundância de dados seja a mínima possível.

Tendo isto na mente, procedemos à verificação do modelo através da normalização, ou seja, verificamos se o modelo estava de acordo com a 1ª Forma Normal, a 2ª Forma Normal e a 3ª Forma Normal:

- Todas as relações estão na 1ª Forma Normal, pois nenhuma apresenta atributos multivalor nem grupos repetidos, ou seja, a interseção de cada linha e cada coluna apresenta apenas um único valor.
- Todas as relações estão na 1ª Forma Normal e todos os seus atributos são totalmente dependentes da chave primária. Como não existem chaves compostas e não existem dependências parciais, o modelo respeita a 2a Forma Formal.
- Todas as relações estão na 1ª e 2ª Forma Normal e nenhum dos seus atributos apresenta dependências transitivas. Como nenhum atributo de nenhuma relação depende de outro atributo que não seja a chave primária, o modelo respeita a 3ª Forma Normal.

Posto isto, como todas as relações estão de acordo com a 3ª Forma Norma, podemos concluir que o modelo está normalizado.

4.4. Validação do modelo com interrogações do utilizador

Um dos requisitos que é necessário para que o modelo possa ser considerado válido é responder a todas as perguntas que o utilizador possa fazer. Sendo assim, foram selecionadas pertinentes cuja viabilidade através vai ser verificada através da comparação com o modelo lógico definido:

- **Médico com mais consultas dadas**

Para ter acesso a esta informação, é necessário as tabelas consulta e medico. Após reunirmos toda a informação é feita uma soma das quantidades, que são posteriormente ordenadas por ordem decrescente e é retirado a primeira linha da tabela resultante que corresponde ao médico com mais consultas dadas.

- **Modalidade com mais consultas**

Neste caso, reunindo informações das tabelas consulta, atleta, categoria e modalidades e seguindo o mesmo raciocínio acima, é possível chegar à modalidade com mais consultas

4.5. Validação do modelo com as transações estabelecidas

4.6. Revisão do modelo lógico com o utilizador

Para dar como terminada esta fase, o modelo lógico deverá ser visto na perspetiva do utilizador. Este processo é importante pois o utilizador deve ter a capacidade de reconhecer que o modelo lógico idealizado é uma representação dos requisitos da clinica. Por isso, foi necessário avaliar toda a documentação associada ao modelo lógico, desde das entidades até aos relacionamentos entre elas. Ao fazermos o modelo, foi-nos possível identificar todas as tabelas necessárias no modelo e os seus diversos relacionamentos.

5 - Implementação Física

5.1 - Seleção do sistema de gestão de base de dados

Para construirmos a Base de Dados proposta , utilizamos como sistema de gestão de base dados, o MySQL. Esta decisão deve-se ao facto de este sistema ter sido utilizado nesta unidade curricular, o que de certa forma facilitou a nossa implementação da Base de Dados.

5.2 - Tradução do esquema lógico para o sistema de gestão de bases de dados escolhido em SQL

O processo de modelação de um esquema físico de uma base de dados envolve a tradução das relações base, definidas previamente no modelo lógico, de maneira a que estas e as suas restrições sejam suportadas pelo SGDB. Assim, para concretizar de forma correcta este tópico, é necessária a conclusão de alguns passos, nomeadamente a descrição das relações base e o desenho das restrições gerais.

5.2.1 - Descrição das relações base

- **Relação Modalidade**

Domínio idModalidades -> Integer
Domínio Modalidade -> String de tamanho variável, de tamanho 32

- **Relação Categoria**

Domínio idCategoria-> Integer
Domínio nome -> String de tamanho variável, de tamanho 45
Domínio Modalidades_idModalidades -> Integer

- **Relação Atleta**

Domínio idAtleta -> Integer
Domínio Nome -> String de tamanho variável, de tamanho 32
Domínio Escalao -> String de tamanho variável, de tamanho 10
Domínio Telemovel -> Integer
Domínio Email -> String de tamanho variável, de tamanho 45
Domínio Categoria_idCategoria -> Integer

- **Relação Consulta**

Domínio idConsulta-> Integer
Domínio Descricao -> String de tamanho variável, de tamanho 32
Domínio Horario -> DATETIME
Domínio Duracao -> TIME
Domínio Custo -> Decimal
Domínio Pago -> Binary
Domínio idAtleta -> Integer
Domínio idMedico -> Integer

- **Relação Medico**

Domínio idMedico-> Integer
Domínio Nome -> String de tamanho variável, de tamanho 32
Domínio Especialidade -> String de tamanho variável, de tamanho 32
Domínio Reputacao -> String de tamanho variável, de tamanho 10

5.2.2 - Desenho das restrições gerais

Nesta fase é necessário apresentar as condições restritivas gerais que dizem respeito ao problema. Sendo assim, vão ser apresentadas todas as restrições agrupadas por relação.

- **Modalidade**

```
CREATE TABLE IF NOT EXISTS `cLEInics`.`Modalidades` (
  `idModalidades` INT NOT NULL AUTO_INCREMENT,
  `Nome` VARCHAR(32) NOT NULL,
  PRIMARY KEY (`idModalidades`))
ENGINE = InnoDB;
```

Figure 3 Criação da tabela Modalidade

- **Categoria**

```
CREATE TABLE IF NOT EXISTS `cLEInics`.`Categoria` (
  `idCategoria` INT NOT NULL AUTO_INCREMENT,
  `Nome` VARCHAR(45) NULL,
  `Modalidades_idModalidades` INT NOT NULL,
  PRIMARY KEY (`idCategoria`),
  INDEX `fk_Categoria_Modalidades1_idx` (`Modalidades_idModalidades` ASC) VISIBLE,
  CONSTRAINT `fk_Categoria_Modalidades1`
    FOREIGN KEY (`Modalidades_idModalidades`)
    REFERENCES `cLEInics`.`Modalidades` (`idModalidades`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;
```

Figure 4 Criação da tabela Categoria

- **Atleta**

```
CREATE TABLE IF NOT EXISTS `cLEInics`.`Atleta` (
  `idAtleta` INT NOT NULL AUTO_INCREMENT,
  `Nome` VARCHAR(32) NOT NULL,
  `Escalao` VARCHAR(10) NOT NULL,
  `Telemovel` INT NOT NULL,
  `Email` VARCHAR(45) NOT NULL,
  `Categoria_idCategoria` INT NOT NULL,
  PRIMARY KEY (`idAtleta`),
  INDEX `fk_Atleta_Categoria1_idx` (`Categoria_idCategoria` ASC) VISIBLE,
  CONSTRAINT `fk_Atleta_Categoria1`
    FOREIGN KEY (`Categoria_idCategoria`)
    REFERENCES `cLEInics`.`Categoria` (`idCategoria`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;
```

Figure 5 Criação da tabela Atleta

- **Medico**

```
CREATE TABLE IF NOT EXISTS `cLEInics`.`Medico` (
  `idMedico` INT NOT NULL AUTO_INCREMENT,
  `Nome` VARCHAR(32) NOT NULL,
  `Especialidade` VARCHAR(32) NOT NULL,
  `Reputacao` VARCHAR(12) NOT NULL,
  PRIMARY KEY (`idMedico`))
ENGINE = InnoDB;
```

Figure 6 Criação da tabela Medico

- **Consulta**

```
CREATE TABLE IF NOT EXISTS `cLEInics`.`Consulta` (
  `idConsulta` INT NOT NULL AUTO_INCREMENT,
  `Descricao` VARCHAR(45) NOT NULL,
  `Horario` DATETIME NOT NULL,
  `Duracao` TIME NOT NULL,
  `Custo` DECIMAL(5,2) NOT NULL,
  `Pago` BINARY(1) NOT NULL,
  `Atleta_idAtleta` INT NOT NULL,
  `Medico_idMedico` INT NOT NULL,
  PRIMARY KEY (`idConsulta`),
  INDEX `fk_Consulta_Atleta1_idx` (`Atleta_idAtleta` ASC) VISIBLE,
  INDEX `fk_Consulta_Medico1_idx` (`Medico_idMedico` ASC) VISIBLE,
  CONSTRAINT `fk_Consulta_Atleta1`
    FOREIGN KEY (`Atleta_idAtleta`)
    REFERENCES `cLEInics`.`Atleta` (`idAtleta`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_Consulta_Medico1`
    FOREIGN KEY (`Medico_idMedico`)
    REFERENCES `cLEInics`.`Medico` (`idMedico`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;
```

Figure 7 Criação da tabela Consulta

5.3 - Tradução das interrogações do utilizador para SQL

1ª Querie - Procurar todos os atletas com consultas marcadas

```
-- Procurar todos os atletas com consultas marcadas
select a.nome, a.escalao
from atleta a
inner join consulta c
on a.idAtleta = c.Atleta_idAtleta;
```

Figure 8 Query 1

2ª Querie - Procurar todas as consultas que foram pagas

```
-- Procurar todas as consultas que foram pagas
select a.nome, a.escalao, c.custo, c.descrição
from atleta a
inner join consulta c
on a.idAtleta = c.Atleta_idAtleta
and c.pago = 1;
```

Figure 9 Query 2

3ª Querie - Consulta mais lucrativa

```
-- Consulta mais lucrativa
select c.custo, c.pago, m.Nome, m.Especialidade, m.Reputação
from consulta c
inner join medico m
on c.Medico_idMedico = m.idMedico
order by c.custo desc
limit 1;
```

Figure 10 Query 3

4ª Querie - Médico com mais consultas dadas

```
-- Médico com mais consultas dadas
select count(*), m.nome, m.especialidade, m.reputação
from consulta c
inner join medico m
on c.Medico_idMedico = m.idMedico
group by c.Medico_idMedico
limit 1;
```

Figure 11 Query 4

5ª Querie - Top 3 das modalidades com mais atletas

```
-- Devolve o top 3 das modalidades com mais atletas
select count(*), m.modalidade
from categoria c
inner join modalidades m
on c.Modalidades_idModalidades = m.idModalidades
inner join atleta a
on c.idCategoria = a.Categoria_idCategoria
group by m.modalidade
limit 3;
```

Figure 12 Query 5

6ª Querie - Modalidade com mais categorias

```
-- Devolve a modalidade com mais categorias
select count(*), m.modalidade
from categoria c
inner join modalidades m
on c.Modalidades_idModalidades = m.idModalidades
group by m.modalidade
order by count(*) desc
limit 1;
```

Figure 13 Query 6

7ª Querie - Número total de consultas existente

```
-- Número total de consultas que existe
select count(*)
from consulta;
```

Figure 14 Query 7

8ª Querie - Atleta com mais consultas

```
-- Atleta com mais consultas
select count(*), a.nome, a.escalao
from atleta a
inner join consulta c
on a.idAtleta = c.Atleta_idAtleta
group by a.nome
order by count(*) desc
limit 1;
```

Figure 15 Query 8

9ª Querie - Modalidade com mais consultas

```
-- Modalidade com mais consultas
select count(*), m.modalidade
from consulta c
inner join atleta a
on c.Atleta_idAtleta = a.idAtleta
inner join categoria cat
on a.Categoria_idCategoria = cat.idCategoria
inner join modalidades m
on cat.Modalidades_idModalidades = m.idModalidades
group by a.nome
order by count(*) desc
limit 1;
```

Figure 16 Query 9

5.4 - Tradução das transações estabelecidas para SQL

```
-- Modalidade com mais consultas
select count(*), m.modalidade
from consulta c
inner join atleta a
on c.Atleta_idAtleta = a.idAtleta
inner join categoria cat
on a.Categoria_idCategoria = cat.idCategoria
inner join modalidades m
on cat.Modalidades_idModalidades = m.idModalidades
group by a.nome
order by count(*) desc
limit 1;
```

Figure 17 Transação

5.5 - Escolha, definição e caracterização de índices em SQL

Todo o conteúdo na base de dados está agrupado em tabelas, existindo vários registos relacionados. Estes são arquivados em ficheiros ou na memória onde, normalmente, cada arquivo tem os registos relativos a uma tabela.

Quando é executada uma query que utiliza a primitiva “WHERE”, a base de dados percorre todas as linhas de uma tabela até encontrar o que procura.

Em SQL os dados de uma coluna de uma tabela são organizados por um índice. Deste modo a procura torna-se mais eficiente, pois só é necessário ler as linhas referentes aos dados em questão.

Quando criamos um índice em SQL temos de ter o cuidado dar prioridade as colunas que se relacionam entre si, colunas que são pesquisadas muitas vezes e as colunas com maior número registos.

Em contrapartida os índices são responsáveis por atrasar as inserções e atualização nas tabelas, pelo que não é aconselhável usá-los em tabelas que sejam

atualizadas frequentemente. Contudo estes tornam as cláusulas “WHERE” e “ORDERBY” mais eficientes.

Aquando da definição de uma chave primária, o InnoDB utiliza-a como clustured index, significando que existem tantas chaves primárias para a nossa tabela como quantos clustered index foram definidos.

No final acabamos por não criar mais índices do que os que foram definidos automaticamente, uma vez que a nossa base não é suficientemente complexa para que compensasse fazê-lo.

5.6 - Estimativa do espaço em disco da base de dados e taxa de crescimento anual

Como em qualquer implementação de uma Base de Dados, é imprescindível considerar o espaço que esta irá ocupar em disco para podermos disponibilizar espaço suficiente para que seja possível guardar todos os dados. Podemos ver que cada tipo de registo terá um tamanho específico de acordo com o tipo de dados que lhe estão associados.

Na tabela seguinte apresentamos cada tipo de dados usados na nossa Base de Dados e o tamanho ocupado, respetivamente:

Tipo de Dados	**Tamanho (Bytes)**
Integer	**4**
VarChar(N)	**N+1**
DateTime	**3**
Time	**3**
Decimal	**5**
Binary	**1**

5.6.1 - Modalidades

Tabela	Atributo	Tipo de dados	Espaço Ocupado
Cliente	**IdModalide** **Modalidade**	**INTEGER** **VARCHAR(32)**	**4 bytes** **32 bytes**

Total(N): Número de modalidades * Tamanho dos dados para uma modalidade = N*36 bytes

5.6.2 - Categoria

Tabela	Atributo	Tipo de dados	Espaço Ocupado
Categoria	**idCategoria** **nome** **Modalidades_idModalidades**	**INTEGER** **VARCHAR(45)** **INTEGER**	**4 bytes** **32 bytes** **4 bytes**

Total(N): Número de categorias * Tamanho dos dados para uma categoria = N*40 bytes

5.6.3 - Atleta

Tabela	Atributo	Tipo de dados	Espaço Ocupado
Atleta	**idAtleta** **nome** **Escalao** **Telemovel** **Email** **Categoria_idCategoria**	**INTEGER** **VARCHAR(45)** **INTEGER** **INTEGER** **VARCHAR(45)** **INTEGER**	**4 bytes** **32 bytes** **4 bytes** **4 bytes** **45 bytes** **4 bytes**

Total(N): Número de atletas* Tamanho dos dados para um atleta = N*93 bytes

5.6.4 - Consulta

Tabela	Atributo	Tipo de dados	Espaço Ocupado
Consulta	**idConsulta**	**INTEGER**	**4 bytes**
	Descricao	**VARCHAR(45)**	**32 bytes**
	Horario	**DATETIME**	**8 bytes**
	Duracao	**TIME**	**3 bytes**
	Custo	**DECIMAL(5,2)**	**5 bytes**
	Pago	**BINARY(1)**	**1 bytes**

Total(N): Número de consultas * Tamanho dos dados para uma consulta = N * 53 bytes

5.6.5 - Médico

Tabela	Atributo	Tipo de dados	Espaço Ocupado
Consulta	**idMedico**	**INTEGER**	**4 bytes**
	Nome	**VARCHAR(32)**	**32 bytes**
	Especialidade	**VARCHAR(32)**	**32bytes**
	Reputacao	**VARCHAR(10)**	**10 bytes**

Total (N) : Número de médicos* Tamanho dos dados para um médico = N * 78 bytes

5.7 - Revisão do sistema implementado com o utilizador

Para dar como terminada esta fase, o modelo físico deverá ser visto na perspetiva do utilizador. Este processo tem um papel importante pois o utilizador tem que ter a capacidade de reconhecer que o modelo físico apresentado representa a situação real da clínica que está a ser modelada.

Sendo assim, este modelo é aprovado uma vez que é visto como uma possível solução que responde a todas as necessidades pretendidas.

6 - Abordagem Não Relacional

6.1 - Introdução

No mercado atual tem se verificado um crescimento de criação de base de dados não relacional. A principal razão para o tal acontecimento é a resolução do problema de escalabilidade das bases de dados tradicionais.

O outro motivo é a enorme quantidade de dados associados a algumas áreas de negócio e os problems que daí surgem. Estes modelos, para além de facilitar a manipulação de dados, permite também um armazenamento sustentável.

A ferramenta utilizada para implementar a base de dados não relacional foi o Neo4j que é baseado em grafos. Este tipo de base de dados utiliza o modelo de grafos para representar o esquema. Podem ser observados três componentes básicas:

I. **Nodos** que representam os vértices do grafo
II. **Relacionamento** entre nodos que são arestas do grafo
III. **Propriedades** relativamente aos nodos ou aos relacionamentos entre nodos

6.2 - Base de Dados NoSQL

Durante vários anos, o modelo de dados predominante usado para o desenvolvimento de aplicações foi o modelo usado por base de dados relacionais, como por exemplo, Oracle, DB2, entre outros. Apenas em meados de 2000 os modelos de dados começaram a ser adotados e a ter um uso mais significativo. Para diferenciar e categorizar essas novas classes de bases e modelos, foi criado o termo NoSQL.

NoSQL é um termo genérico que representa base de dados não relacionais sendo open source e completamente distinta do modelo relacional tradicional, isto porque as bases de dados NoSQL foram projetadas através das necessidades que as bases de dados tradicionais relacionais não satisfaziam, como por exemplo, a alta performance e a capacidade de expansão.

Sendo assim, é possível afirmar que existem quatro tipos diferentes de base de dados NoSQL:

I. **Grafos** – baseados na teoria de grafos, armazena os seus dados na forma de um grafo com vértices e arestas, por exemplo, Neo4j
II. **Documentos** – armazena os seus dados como documentos, em que cada um é uma coleção de pares chave-valor e permite a realização de consultas mais elaboradas, envolvendo filtros por atributos e a possibilidade de uso de índices, por exemplo, MongoDB.
III. **Colunas** – armazena os seus dados em linhas particulares de uma tabela no disco, por exemplo, Cassandra.
IV. **Chave-Valor** – armazena os seus dados num padrão chave-valor, num estilo de tabelas de hash, por exemplo, Riak.

6.2.1 - Principais características

1. **Alta performance e escabilidade horizontal**

Enquanto vários servidores de base de dados possuiam a adição de um maior número de recursos ao servidor como a memória e o disco para conseguirem suportar mais dados, as bases de dados NoSQL redistribuem-se horizontalmente, ou seja, funciona como um sistema fracamente acoplado, o que torna mais económica e escalável. Graças a esse mecanismo, a sua performance provém de não possuir um servidor central, mas sim de vários servidores que não necessitam de ser alta performance, conectados e trabalhando em conjunto.

2. **Raízes Open Source**

Muitas bases de dados têm raízes na comunidade Open Source, o que terá ajudado bastante no seu crescimento. É de notar que há companhias que oferecem versões comerciais de base de dados NoSQL com uma forte estrutura de suporte e serviços.

3. **Baixo custo operacional**

Devido ao peso do open source no NoSql, o custo para inicializar a utilização desses base de dados é quase nulo. É comum dizer que a transição relacional para NoSQL diminuiu muito os custos e obtendo um desempenho igual ou melhor ao anterior.

4. **Ausência de esquema**

Como já foi referido, uma das principais caracteristicas do NoSQL é a ausência total ou quase total de esquema que define a estrutura de dados. Devido a essa ausência, existe uma fácil aplicação de escabilidade e também um aumento na disponibilidade. Também é de referir que possui uma API simples para fácil acesso de dados.

6.2.2 - SQL vs NoSQL

1 **Linguagem**

As bases de dados SQL são estruturadas em linguagem de consulta (SQL) para definição e controlo de dados. É possível dizer que graças a isso, é extremamente poderoso, o SQL é uma das opções mais versáveis e mais utilizadas, sendo uma escolha segura e ótima para consultas complexas. Mas por outro lado, apesar de ser extremamente poderoso, pode ser restritivo, o SQL exige o uso de esquemas pré-definidos para determinar a estrutura dos seus dados antes de trabalhar com eles.

- **Escabilidade**

Na maioria das situações, as bases de dados SQL são verticalmente escaláveis, o que significa que se pode aumentar o carregamento num servidor melhorando coisas como por exemplo, o CPU, RAM ou SSD.

As bases de dados NoSQL, por sua vez, são horizontalmente escaláveis. Isso quer dizer que suporta muito mais tráfego por sharding (particionamento de dados), ou 50 seja, adicionando mais servidores na sua base de dados NoSQL.

- **Estrutura**

Enquanto as bases de dados SQL são baseadas em tabelas, as bases de dados NoSQL podem ser baseadas em documentos, pares de valor-chave, grafos ou orientados a colunas. Isto torna as bases de dados SQL relacionais opções melhores para aplicações que requerem transações retornando várias colunas – como por exemplo, um sistema de contabilidade.

6.2.3 - Vantagens

- Dados sempre disponiveis
- Custo reduzido
- Base de dados orientado a objectos flexíveis
- Facilidade em inserir novos dados
- Excelente opção de lidar com o problema de dados em massa
- Performance superior na consulta de grandes ou pequenas quantidades de dados

6.2.4 - Desvantagens

- Mais recentes que as bases de dados relacionais, ou seja, não existe tanta alternativa
- Não resolve o problema de escabilidade de um website
- Situações onde o modo como os dados estão estruturados é importante, como o exemplo de contas bancárias, uma base de dados relacional será mais adequada
- Neo4j apenas possui hash indexes, faltando um index de alcance

6.3 - Migração de dados

A migração de dados de um modelo relacional para um não relacional, nem sempre é considerado uma tarefa fácil, principalmente quando se trata de base de dados com quantidade de informação muito superior do que aquela que por nós foi tratada. A decisão de fazer este tipo de migração passa, entre outras razões expostas anteriormente, pela falta de flexibilidade dos modelos relacionais.

Para migrar dados de e para um sistema de base de dados, implica retirar a informação de uma base de dados e colocar essa informação noutra, algo que apesar de parecer simples, tem algumas dificuldades.

Tendo isso em conta, a nossa equipa definiu que o primeiro ponto a ter em consideração, seria a **exportação de informação** do sistema de base de dados para ficheiros do tipo .csv . Feito isso, no Neo4j, teríamos que **importar a informação** guardada nesses mesmos ficheiros. A importação seria feita quando da **criação dos nodos,** o que nos faz popuar bastante na escrita de código. Com estes, já tinhamos toda a informação necessária para **criar todos os relacionamentos existentes** de uma forma automática. Finalmente, a última fase seria a **remoção de dados** de certos modos. A partir da importação, todo o código foi escrito na linguagem Cypher, linguagem disponível no programa Neo4j.

6.4 - Grafo resultante

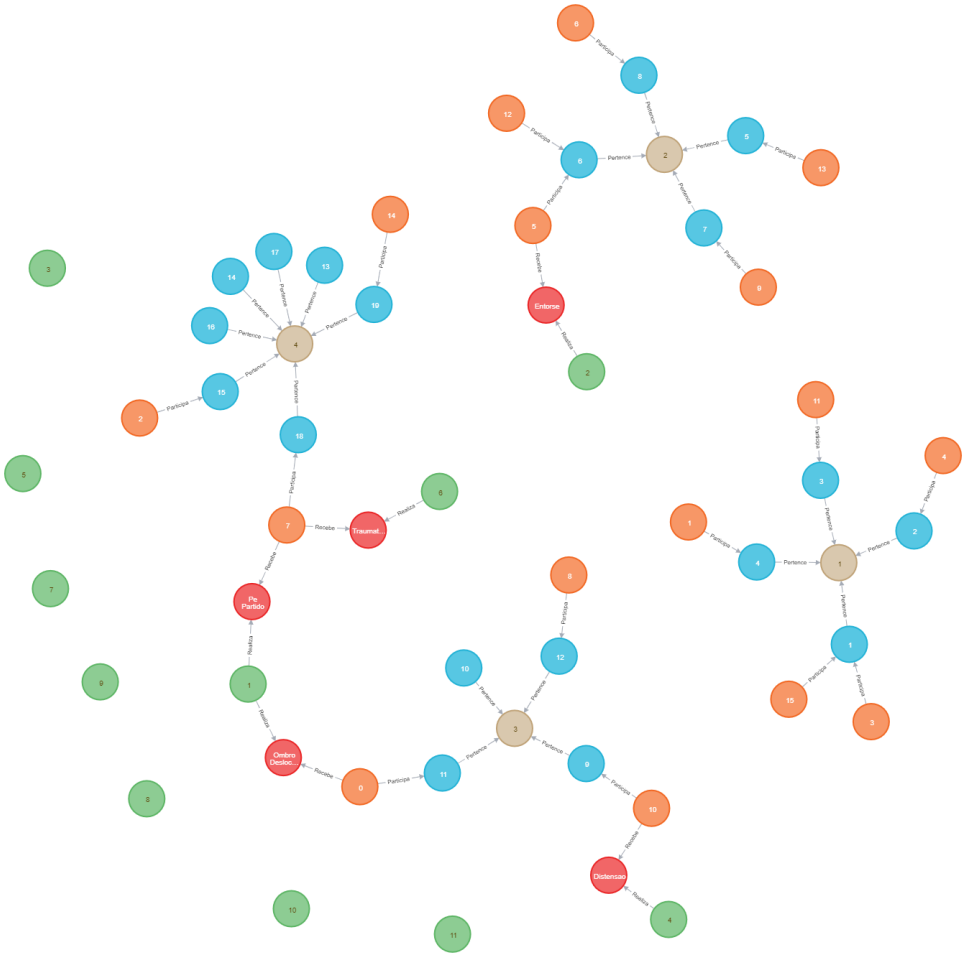

Figure 18 Grafo da Base de Dados Neo4j

6.5 - Tradução das interrogações do utilizador para Cypher

1ª querie - Procurar todos os atletas com consultas marcadas

```
//Procurar todos os atletas com consultas marcadas
MATCH (a:Atleta)-[r:Recebe]->(con:Consulta)
RETURN a.idAtleta
```

Figure 19 Query 1 Neo4j

2ª querie - Procurar todas as consultas que foram pagas

```
//Procurar todas as consultas que foram pagas
Match (con:Consulta{Pago: 1})
Return con
```

Figure 20 Query 2 Neo4j

3ª querie - Consulta mais lucrativa

```
//Consulta mais lucrativa
MATCH (con:Consulta)
RETURN con
ORDER BY con.Custo
LIMIT 1
```

Figure 21 Query 3 Neo4j

4ª querie - Médico com mais consultas dadas

```
//Médico com mais consultas dadas
MATCH(med:Medico)
WITH med, size((med)-[:Realiza]->()) as NrConsultas
RETURN med.Nome AS Nome, NrConsultas
ORDER BY NrConsultas DESC
LIMIT 1
```

Figure 22 Query 4 Neo4j

5ª querie - Top 3 das modalidades com mais atletas

```
//Devolve o top 3 das modalidades com mais atletas
MATCH (a:Atleta)-[:Participa]->(c:Categoria)-[:Pertence]->(m:Modalidade)
WITH m, collect(a.Nome) AS atls
RETURN m.Nome AS Modalidades,  size(atls)
ORDER BY size(atls) DESC
LIMIT 3
```

Figure 23 Query 5 Neo4j

6ª querie - Modalidade com mais categorias

```
//Devolve a modalidade com mais categorias
MATCH(m:Modalidade)
WITH m, size(()-[:Pertence]->(m)) AS NrCategorias
RETURN m.Nome AS Nome, NrCategorias
ORDER BY NrCategorias DESC
LIMIT 1
```

Figure 24 Query 6 Neo4j

7ª querie - Número total de consultas existente

```
//Número total de consultas que existe
MATCH(con:Consulta)
RETURN count(*) AS NrConsultas
```

Figure 25 Query 7 Neo4j

8ª querie - Atleta com mais consultas

```
//Atleta com mais consultas
MATCH(a:Atleta)
WITH a, size((a)-[:Recebe]->()) AS NrConsultas
RETURN a.Nome AS Nome, NrConsultas
ORDER BY NrConsultas DESC
LIMIT 1
```

Figure 26 Query 8 Neo4j

9ª querie - Modalidade com mais consultas

```
//Modalidade com mais consultas
MATCH (con:Consulta)<-[:Recebe]-(a:Atleta)-[:Participa]->(c:Categoria)-[:Pertence]->(m:Modalidade)
WITH m, collect(con.idConsulta) AS cons
RETURN m.Nome AS Modalidades,  size(cons)
ORDER BY size(cons) DESC
LIMIT 1
```

Figure 27 Query 9 Neo4j

6.6 - Revisão do sistema implementado

Tal como fizemos com a base de dados relacional, era necessário chegarmos a um resultado final do processo de migração de base de dados. Imediatamente obtivemos uma reação positiva, uma vez que ajuda a compreender melhor a informação existente e ao mesmo tempo um melhor manuseamento comparando com a outra.

6.7. Conclusões

Para o desenvolvimento de um Sistema de Gestão de Base de Dados, é necessário implementar vários passos fundamentais, tal como fizemos ao longo deste projeto, como por exemplo, a definição de entidades e relacionamentos que sejam capazes de otimizar a gerência e o armazenamento dos dados em causa, entre outros. Sendo assim, foi definido um modelo conceptual, com proximidade ao cliente, e, posteriormente, foi elaborado um modelo lógico, novamente validado pelo mesmo.

Também foi necessária a implementação da mesma Base de Dados, agora em NoSQL, utilizando assim Neo4j. Usando “Exports” em formato .csv do MySQL WorkBench, criamos então a base de dados em Neo4j, sendo que esta utiliza nodos para representação das suas entidades.

Posto isto, tendo em conta os resultados, podemos concluir que os objetivos foram cumpridos, pois conseguiu-se a elaboração de uma base de dados e funcional.

Referências

<<Apresentar a lista de referências bibliográficas referidas ao longo do relatório; recomenda-se a utilização do formato Harvard - http://libweb.anglia.ac.uk/referencing/harvard.htm>>

Lista de Siglas e Acrónimos

<<Apresentar uma lista com todas as siglas e acrónimos utilizados durante a realização do trabalho. O formato base para esta lista deverá ser da forma como abaixo se apresenta.>>

BD	Base de Dados
DW	Data Warehouse
OLTP	*On-Line Analytical Processing*
…	*…*

Anexos

<<Os anexos deverão ser utilizados para a inclusão de informação adicional necessária para uma melhor compreensão do relatório o para complementar tópicos, secções ou assuntos abordados. Os anexos criados deverão ser numerados e possuir uma designação. Estes dados permitirão complementar o Índice geral do relatório relativamente à enumeração e apresentação dos diversos anexos.>>

I. Anexo

```
-- MySQL Script generated by MySQL Workbench
-- Mon Dec 30 18:24:01 2019
-- Model: New Model    Version: 1.0
-- MySQL Workbench Forward Engineering

SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0;
SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS,
FOREIGN_KEY_CHECKS=0;
SET @OLD_SQL_MODE=@@SQL_MODE,
SQL_MODE='ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO
_ZERO_DATE,ERROR_FOR_DIVISION_BY_ZERO,NO_ENGINE_SUBSTITUTION';

-- -----------------------------------------------------
-- Schema cLEInics
-- -----------------------------------------------------
DROP SCHEMA IF EXISTS `cLEInics` ;

-- -----------------------------------------------------
-- Schema cLEInics
-- -----------------------------------------------------
CREATE SCHEMA IF NOT EXISTS `cLEInics` DEFAULT CHARACTER SET utf8 ;
USE `cLEInics` ;

-- -----------------------------------------------------
-- Table `cLEInics`.`Modalidades`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `cLEInics`.`Modalidades` ;

CREATE TABLE IF NOT EXISTS `cLEInics`.`Modalidades` (
  `idModalidades` INT NOT NULL AUTO_INCREMENT,
  `Nome` VARCHAR(32) NOT NULL,
  PRIMARY KEY (`idModalidades`))
ENGINE = InnoDB;

INSERT INTO Modalidades
                (Nome)
      VALUES
```

```
        ('Corrida'),
        ('Salto'),
        ('Lancamento'),
        ('Natacao');

-- -----------------------------------------------------
-- Table `cLEInics`.`Categoria`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `cLEInics`.`Categoria` ;

CREATE TABLE IF NOT EXISTS `cLEInics`.`Categoria` (
  `idCategoria` INT NOT NULL AUTO_INCREMENT,
  `Nome` VARCHAR(45) NULL,
  `Modalidades_idModalidades` INT NOT NULL,
  PRIMARY KEY (`idCategoria`),
  INDEX `fk_Categoria_Modalidades1_idx` (`Modalidades_idModalidades` ASC) VISIBLE,
  CONSTRAINT `fk_Categoria_Modalidades1`
    FOREIGN KEY (`Modalidades_idModalidades`)
    REFERENCES `cLEInics`.`Modalidades` (`idModalidades`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;

INSERT INTO Categoria
                    (Nome, Modalidades_idModalidades)
        VALUES
        ('100 metros', 1),
        ('200 metros', 1),
        ('400 metros', 1),
        ('Obstáculos', 1),
        ('Altura', 2),
        ('Comprimento', 2),
        ('Triplo', 2),
        ('Vara', 2),
        ('Peso', 3),
        ('Dardo', 3),
        ('Martelo', 3),
        ('Disco', 3),
        ('Livres 50 metros', 4),
        ('Livres 100 metros', 4),
```

```
        ('Livres 800 metros', 4),
        ('Crawl 50 metros', 4),
        ('Crawl 100 metros', 4),
        ('Costas 50 metros', 4),
        ('Costas 100 metros', 4);

-- -----------------------------------------------------
-- Table `cLEInics`.`Atleta`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `cLEInics`.`Atleta` ;

CREATE TABLE IF NOT EXISTS `cLEInics`.`Atleta` (
  `idAtleta` INT NOT NULL AUTO_INCREMENT,
  `Nome` VARCHAR(32) NOT NULL,
  `Escalao` VARCHAR(10) NOT NULL,
  `Telemovel` INT NOT NULL,
  `Email` VARCHAR(45) NOT NULL,
  `Categoria_idCategoria` INT NOT NULL,
  PRIMARY KEY (`idAtleta`),
  INDEX `fk_Atleta_Categoria1_idx` (`Categoria_idCategoria` ASC) VISIBLE,
  CONSTRAINT `fk_Atleta_Categoria1`
    FOREIGN KEY (`Categoria_idCategoria`)
    REFERENCES `cLEInics`.`Categoria` (`idCategoria`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;

INSERT INTO Atleta
                    (Nome, Escalao, Telemovel, Email, Categoria_idCategoria)
        VALUES
        ('Andre Fonseca', 'Senior', 910000005, 'rasputin@gmail.com', 11),
        ('Mariana Pereira', 'Senior', 910000006, 'mulan@gmail.com', 4),
        ('Gonçalo Garcia', 'Senior', 910000011, 'golias@gmail.com', 15),
        ('Artur Ribeiro', 'Junior', 910000012, 'flash@gmail.com', 1),
        ('Davide Matos', 'Junior', 910000014, 'groot@gmail.com', 2),
        ('Pedro Medeiros', 'Junior', 910000024, 'preacher@gmail.com', 6),
        ('Nuno Silva', 'Juvenil', 910000025, 'mineiro@gmail.com', 8),
        ('Joao Duarte', 'Juvenil', 910000027, 'zet@gmail.com', 18),
        ('Pedro Lima', 'Juvenil', 910000032, 'eddy@gmail.com', 12),
        ('Francisco Freitas', 'Infantil', 910000045, 'cyborg@gmail.com', 7),
```

```
        ('Shahzod Yusupov', 'Infantil', 910000047, 'balboa@gmail.com', 9),
        ('Alexandre Pacheco', 'Infantil', 910000051, 'megamind@gmail.com', 3),
        ('Fabio Silva', 'Senior', 910000053, 'frota@gmail.com', 6),
        ('Diogo Sobral', 'Junior', 910000054, 'ambrosio@gmail.com', 5),
        ('Ines Alves', 'Juvenil', 910000059, 'sbentas@gmail.com', 19),
        ('Pedro Pinto', 'Infantil', 910000062, 'random@gmail.com', 1);

-- -----------------------------------------------------
-- Table `cLEInics`.`Medico`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `cLEInics`.`Medico` ;

CREATE TABLE IF NOT EXISTS `cLEInics`.`Medico` (
  `idMedico` INT NOT NULL AUTO_INCREMENT,
  `Nome` VARCHAR(32) NOT NULL,
  `Especialidade` VARCHAR(32) NOT NULL,
  `Reputacao` VARCHAR(12) NOT NULL,
  PRIMARY KEY (`idMedico`))
ENGINE = InnoDB;

INSERT INTO Medico
                (Nome, Especialidade, Reputacao)
        VALUES
        ('Adriana Meireles', 'Ortopedia', 'Famoso'),
        ('Helena Martins', 'Ortopedia', 'Conhecido'),
        ('Pedro Freitas', 'Ortopedia', 'Desconhecido'),
        ('Pedro Pinto', 'Ortopedia', 'Famoso'),
        ('Luis Moreira', 'Ortopedia', 'Desconhecido'),
        ('Andre Costa', 'Ortopedia', 'Conhecido'),
        ('Francisco Laço', 'Ortopedia', 'Famoso'),
        ('Filipa Pereira', 'Ortopedia', 'Conhecido'),
        ('Marta Ribeiro', 'Ortopedia', 'Desconhecido'),
        ('Maria Dias', 'Ortopedia', 'Conhecido'),
        ('Carla Cruz', 'Ortopedia', 'Famoso');

-- -----------------------------------------------------
-- Table `cLEInics`.`Consulta`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `cLEInics`.`Consulta` ;
```

```
CREATE TABLE IF NOT EXISTS `cLEInics`.`Consulta` (
  `idConsulta` INT NOT NULL AUTO_INCREMENT,
  `Descricao` VARCHAR(45) NOT NULL,
  `Horario` DATETIME NOT NULL,
  `Duracao` TIME NOT NULL,
  `Custo` DECIMAL(5,2) NOT NULL,
  `Pago` BINARY(1) NOT NULL,
  `Atleta_idAtleta` INT NOT NULL,
  `Medico_idMedico` INT NOT NULL,
  PRIMARY KEY (`idConsulta`),
  INDEX `fk_Consulta_Atleta1_idx` (`Atleta_idAtleta` ASC) VISIBLE,
  INDEX `fk_Consulta_Medico1_idx` (`Medico_idMedico` ASC) VISIBLE,
  CONSTRAINT `fk_Consulta_Atleta1`
    FOREIGN KEY (`Atleta_idAtleta`)
    REFERENCES `cLEInics`.`Atleta` (`idAtleta`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_Consulta_Medico1`
    FOREIGN KEY (`Medico_idMedico`)
    REFERENCES `cLEInics`.`Medico` (`idMedico`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;

INSERT INTO Consulta
                (Descricao, Horario, Duracao, Custo, Pago, Atleta_idAtleta, Medico_idMedico)
      VALUES
      ('Ombro Deslocado','2019-01-10 10:00:00', '00:20:00', 55.00, 1, 1, 1),
      ('Entorse','2019-02-24 14:30:00', '00:50:00', 25.90, 1, 6, 2),
      ('Traumatismo Craniano','2019-02-25 20:25:00', '01:20:00', 120.00, 1, 8, 6),
      ('Pe Partido','2019-07-01 08:00:00', '00:50:00', 45.00, 0, 8, 1),
      ('Distensao Muscular','2019-10-10 15:45:00', '00:45:00', 55.00, 1, 11, 4);

SET SQL_MODE=@OLD_SQL_MODE;
SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS;
SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS;
```