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

Unidade Curricular de Bases de Dados

Ano Lectivo de 2018/2019

Ginásio 100calorias

Ana Filipa Vilela Pereira - A81712

Ana Marta Santos Ribeiro - A82474

Carla Isabel Novais da Cruz - A80564

Jéssica Andreia Fernandes Lemos – A82061

Janeiro, 2019

BD

Data de Recepção	
Responsável	
Avaliação	
Observações	

Ginásio 100calorias

Ana Filipa Vilela Pereira - A81712

Ana Marta Santos Ribeiro - A82474

Carla Isabel Novais da Cruz - A80564

Jéssica Andreia Fernandes Lemos – A82061

Janeiro, 2019

Resumo

O trabalho realizado no âmbito da unidade curricular de Base de Dados, consistiu na criação de uma base de dados relacional, e posteriormente a migração para uma não relacional, que tinha como objetivo suportar as necessidades do ginásio do Sr. João, “Ginásio 100calorias”.

Numa fase inicial foi retratada a contextualização deste tema, qual a motivação e objetivos deste projeto, bem com a análise de viabilidade. Começamos por identificar quais as necessidades do Sr. João para este projeto, efetuando assim o levantamento de requisitos. Posteriormente foi realizada a análise dos mesmos de modo a que esta base de dados responda ao que é pretendido.

De seguida, é feito o Modelo Conceptual onde são apresentadas as entidades relevantes ao nosso projeto, bem como os seus atributos e de que forma estas se relacionam. Tendo em conta os atributos que constituem cada entidade foram identificadas as chaves candidatas de cada, de modo a determinar as chaves primárias e as chaves alternativas. Após a validação do modelo com o utilizador passamos à fase seguinte.

O Modelo Lógico foi elaborado através da derivação do Modelo Conceptual, sendo abordado de forma mais profunda as entidades existentes bem como os relacionamentos estabelecidos entre estas, e o domínio dos seus atributos de modo a se identificar as chaves primárias e estrangeiras. Além da derivação realizou-se a validação do modelo tendo em conta as regras de normalização e transações, garantindo a integridade dos dados. Validado o Modelo Lógico obtemos a aprovação do trabalho realizado até ao momento por parte do Sr. João, pelo que podemos avançar para a fase final do projeto.

Esta última fase da elaboração da base de dados relacional teve como objetivo a criação do Modelo Físico. Para tal, realizamos a tradução do modelo elaborado na fase anterior para um SGBS, começando por gerar o script de criação e povoamento em SQL. De seguida, tornou-se imperativo a concretização dos requisitos de exploração elaborados numa fase inicial, realizando-se queries de modo a satisfazer os pedidos dos utilizadores e as funcionalidades da aplicação. De seguida, efetuamos uma estimativa do tamanho da base de dados em termos de ocupação de espaço no disco. Por fim, definimos as vistas de utilização e regras de acesso.

Na segunda fase do projeto, começamos por exportar os dados elaborando um script em Java no qual extraímos os dados de cada tabela para ficheiros csv. Para a migração da base de dados implementada anteriormente para uma NoSQL, utilizando os ficheiros anteriormente criados e recorrendo à linguagem Cypher criamos os nodos e relacionamentos bem como realizamos as interrogações definidas previamente.

Terminamos com a apresentação das conclusões do trabalho realizado, elaborando uma análise crítica de todo o processo realizado anteriormente bem como sugestões para trabalho futuro.

Área de Aplicação: Desenho e Arquitetura de Sistemas de Base de Dados no âmbito de uma aplicação responsável por registar todos os elementos do ginásio

Palavras-Chave: Base de Dados Relacional, Base de Dados Não Relacional, Análise de Requisitos, Modelo Conceptual, Modelo Lógico, Modelo Físico, Normalização, Validação, brModelo, MySQL Workbench, SQL, NoSQL, Neo4j, Migração, Cypher, Java

Índice

Anexos

Índice de Figuras

Índice de Tabelas

1 Introdução

1.1 Contextualização

O Sr. João Silva passou por graves problemas de saúde causados por uma vida sedentária. Quando conseguiu ultrapassá-los decidiu investir num ginásio, de modo a poder proporcionar uma oportunidade às pessoas que passaram pela mesma situação e desta forma mudar o seu estilo de vida.

Com as suas poucas capacidades económicas a construção do ginásio irá surgir com muito esforço e dedicação do próprio. Assim, inaugurou o “Ginásio 100calorias” em 2015, dois anos após ter idealizado o projeto.

Devido à sua reduzida capacidade de investimento, as instalações adquiridas para o ginásio eram pequenas, possuíam escassos equipamentos e as aulas que existiam eram dadas pelo próprio. Assim, numa fase inicial, este era apenas frequentado por familiares e amigos próximos que o queriam ajudar, o que facilitava na sua gestão.

Apesar disto, com os passar dos primeiros meses começou a constatar-se melhorias nas vidas dos clientes. Desta forma, o Sr. João aproveitou tais casos de sucesso para promover o seu negócio. Assim, verificou-se um aumento na afluência tornando o projeto rentável. Esta evolução chegou ao ponto em que o número de clientes era excessivo para as condições verificadas. Deste modo, tornou-se imperativo a expansão das instalações, a compra de novos equipamentos e a contratação de professores.

Com o intuito de proporcionar diversas experiências aos seus clientes o Sr. João decidiu implementar diversas atividades de fitness lecionadas pelos novos professores. Este crescimento originou diversos problemas ao nível da gestão, uma vez que conciliar vários alunos em diferentes atividades lecionadas por diversos professores tornou-se impossível. Acrescido a este problema, como o Sr. João pretende fornecer aos seus clientes um acompanhamento personalizado notou que era necessária uma forma de organizar toda a informação. Assim sendo, recorreu aos nossos serviços para a elaboração de um Sistema de Base de Dados capaz de reverter a situação em que se encontra o ginásio.

1.2 Apresentação do Caso de Estudo

O “Ginásio 100calorias” é uma empresa em desenvolvimento que procura melhorar o estilo de vida dos seus clientes. Para tal, pretende-se que todos os dias existam diferentes aulas de fitness lecionadas por diferentes professores especializados.

Este ginásio distingue-se por um ambiente acolhedor e familiar, como tal espera-se que exista um acompanhamento personalizado a cada um dos clientes. Desta forma, numa fase inicial é idealizado um plano de treinos para cada um dos clientes que poderá ser modificado consoante o feedback destes. Ao fim de cada mês é verificado os desenvolvimentos de cada aluno, de modo a perceber os objetivos alcançados e a definir novos.

Com o aumento da afluência ao ginásio verificou-se que a procura era superior à oferta havendo a necessidade de uma evolução tecnológica do sistema.

Tendo em conta os aspetos referidos anteriormente, o ginásio considera que o seu modelo de negócio é viável e pretende avançar com o desenvolvimento do mesmo analisando e implementando uma base de dados que servirá de suporte a todo o sistema.

1.3 Motivação e Objetivos

Dado que o “Ginásio 100calorias” se encontra numa fase de desenvolvimento e ainda não possui um armazenamento de dados implementado, o Sr. João sentiu a necessidade de reverter esta situação, para tal recorreu aos nossos serviços para a implementação de um Sistema de Base de Dados. O grupo aceitou esta proposta por querer participar no desenvolvimento deste projeto inovador.

Esta base de dados implementada no projeto do Sr. João assenta-se nos seguintes pressupostos:

- Maior facilidade em obter a informação de um cliente de modo a conhecer as suas limitações físicas e todo o historial de treinos;
- Obter os professores que lecionam as diversas atividades fitness;
- Conhecer o número de equipamentos disponíveis;

O desenvolvimento do sistema proposto constitui uma grande responsabilidade, uma vez que qualquer falha da nossa parte pode comprometer o funcionamento do ginásio. No entanto, é um desafio interessante e atrativo.

1.4 Análise da viabilidade do projeto

A implementação de um sistema de base de dados relacional que sustenta o projeto “Ginásio 100calorias” tem como principais benefícios a garantia da integridade dos dados, controlo de redundância e consistência de dados. Uma base de dados bem idealizada e

elaborada garante a integridade dos dados através da inclusão de restrições, que têm de ser cumpridas. É importante existir um armazenamento bem estruturado, de modo a evitar ter os mesmo dados guardados de forma diferente, reduzindo a ocupação desnecessária de espaço. Este controlo permite diminuir a probabilidade de os dados se tornaram inconsistentes.

1.5 Estrutura do Relatório

No relatório elaborado, foi realizada a divisão em cinco capítulos:

- Definição do Sistema (capítulo atual)
- Levantamento e Análise de Requisitos
- Implementação da Base de Dados Relacional
  - Modelação Conceptual
  - Modelação Lógica
  - Implementação Física
- Implementação da Base de Dados Não Relacional
- Conclusões

Na fase inicial, como já foi previamente apresentado, existiu a contextualização do problema, a apresentação dos objetivos e motivação para a realização do projeto, bem como, a análise da viabilidade do mesmo.

No segundo capítulo, é realizada uma entrevista ao Sr. João de modo a saber quais as necessidades da sua aplicação, resultando isto, num levantamento de requisitos aos quais a base de dados implementada terá de responder e satisfazer.

No terceiro capítulo, é demonstrado todo o processo a realizar para a implementação da nossa base de dados relacional. Neste capítulo, é assim apresentada toda a metodologia, desde a modelação conceptual até à modelação lógica e física. Na conceção da modelação conceptual, são identificadas as entidades, os relacionamentos entre estas, os atributos das entidades e dos relacionamentos. É ainda realizada a validação deste modelo com o utilizador. Numa segunda fase é apresentado o modelo lógico, nomeadamente: derivação de relações do modelo lógico e validação destas através da normalização e verificação das restrições de integridade. Por fim, é descrito o modelo físico particularmente a resposta a queries com o sistema SQL, implementação de transações e triggers.

No quarto capítulo, é apresentado todo o processo da elaboração da base de dados não relacional que teve como base a implementada no capítulo anterior. Para tal, mostramos como realizamos a migração dos dados para o Neo4j. É demonstrado, também, a realização das queries definidas previamente.

Para terminar o nosso relatório, são apresentadas uma série de conclusões, bem como, uma análise crítica do que fora realizado e uma avaliação do projeto.

2 Levantamento e análise de requisitos

2.1 Método de levantamento e de análise de requisitos adotado

Para obter a informação necessária e tomar conhecimento dos objetivos pretendidos ao desenvolvimento da base de dados, foi realizada uma entrevista ao Sr. João. A esta seguiram-se diversas reuniões, para discutir alguns detalhes acerca dos objetivos previamente estabelecidos.

2.2 Especificação de Requisitos

Com base na entrevista feita ao Sr. João, identificaram-se os seguintes requisitos:

2.2.1 Requisitos de descrição

1. Os alunos devem ser registados sendo-lhes atribuído um ID sequencial pela aplicação e deve ser fornecido nome, data de nascimento, morada (endereço e localidade), os contactos telefónicos (tipo e número de telemóvel) e as suas limitações físicas.
2. Os professores são registados atribuindo-lhes um ID sequencial e um estado (ativo e inativo), fornecendo o nome, morada (endereço e localidade), data de nascimento e contacto (email e número de telemóvel).
3. As atividades fitness existentes devem ser registadas sendo-lhes atribuído um ID sequencial e é necessário indicar o número máximo de participantes, nome da atividade, duração das aulas, a sala e o número de inscritos.
4. As máquinas são identificadas por um ID sequencial e devem conter o tipo e a quantidade existente do mesmo.
5. Cada plano realizado inclui um ID sequencial, o preço, a data do início deste e o estado que indica se este está a ser realizado pelo cliente.
6. Um cliente enquanto frequenta o ginásio poderá ver definido um novo plano quando os objetivos do anterior forem cumpridos.

7. Um professor pode lecionar uma ou mais atividades fitness, contudo, uma atividade apenas pode ser lecionada por um professor.
8. Um professor, pode ainda, criar diversos planos que serão atribuídos aos clientes, no entanto um plano é elaborado por apenas um professor.
9. Um plano é constituído por diversas atividades fitness e uma atividade está presente em vários planos.
10. No momento em que o professor adiciona uma atividade ao plano, deve indicar o número de aulas a frequentar.
11. As máquinas são distribuídas por atividades fitness, sendo que cada atividade pode recorrer à utilização de várias máquinas e uma máquina pode estar presente em diversas atividades.
12. Sempre que uma máquina é associada a uma atividade fitness, é necessário indicar a quantidade pretendida.

2.2.2 Requisitos de exploração

13. Obter o número de clientes do ginásio.
14. Verificar o número de professores que lecionam no ginásio.
15. Saber para determinada atividade o professor que a leciona.
16. Conhecer a sala onde a atividade é lecionada.
17. Consultar os planos realizados por um dado cliente.
18. Identificar o Top 3 de clientes com maior número de planos associados.
19. Reconhecer o professor que leciona mais atividades.
20. Obter o Top 3 das atividades com mais alunos inscritos.
21. Identificar o número de planos elaborados por cada professor.
22. Verificar o Top 3 dos alunos com maior número de aulas de uma determinada atividade fitness.
23. Obter a atividade fitness mais frequentada por um determinado aluno.
24. Conhecer os planos elaborados por um professor ordenados pelo preço, bem como o cliente ao qual o plano foi atribuído.
25. Identificar os planos realizados por um determinado cliente com um determinado estado.
26. Obter atividades fitness lecionadas por um professor.
27. Apresentar os nomes e contactos dos alunos que frequentam uma dada atividade.
28. Obter os clientes de uma dada localidade.

2.2.3 Requisitos de controlo

29. O Sr. João tem permissões de consulta, inserção e alteração de alunos, professores, máquinas, planos e atividades do sistema.
30. Os professores têm permissões de consulta das informações relativas aos clientes, às atividades, aos planos, às máquinas e de alteração e inserção de planos.
31. Os alunos têm permissões para consultas as informações das atividades fitness existentes no ginásio.

2.3 Análise geral dos requisitos

Após a elaboração final dos requisitos, reunimos novamente com o Sr. João para apresentar os mesmos. Nesta o nosso cliente deu a sua aprovação pelo que podemos prosseguir com o nosso trabalho. Assim, começamos a construir o Modelo Conceptual suportado pelos requisitos apresentados.

2.4 Caraterização dos Perfis de Utilização

Atendendo aos requisitos apresentados, é possível identificar três perfis de utilização: o de cliente, de professor e o Sr. João. O cliente pode consultar a descrição relativa às diversas atividades fitness do ginásio. O professor pode consultar os planos, os alunos, as máquinas e atividades existentes no ginásio. No perfil do Sr. João, é possível fazer alterações diretas no sistema no que toca à atualização, inserção e consulta de dados.

3 Modelação conceptual

3.1 Apresentação da abordagem de modelação realizada

Para este projeto optamos por utilizar uma vista de análise de desenvolvimento, assim, abordamos a questão de uma vez só. Como este problema é de complexidade reduzida, consideramos que esta é a melhor solução, dado que o dividir e utilizar mais do que uma vista seria desnecessário.

3.2 Identificação e caracterização das entidades

Para a elaboração do Modelo Conceptual começamos por identificar as entidades do problema. É essencial uma observação cuidadosa dos requisitos, de modo a não fazer uma seleção incorreta destes, pois, por vezes, existem objetos com importância para o modelo, mas que não são necessariamente uma entidade.

Assim sendo, iremos ilustrar as entidades escolhidas, bem como o seu significado e o porquê da sua ocorrência.

Cliente

Termo geral que descreve todos os clientes do ginásio registados na base de dados. Cada cliente possui atributos próprios, tem uma existência autónoma e pode ser identificado univocamente, sendo assim, uma entidade.

Professor

Entidade que representa todos os professores do ginásio registados na base de dados, responsável por criar os planos dos diversos alunos, bem como lecionar várias atividades de fitness.

Plano

Entidade que indica as atividades que o aluno deve frequentar, elaborados por um professor. Todos os alunos têm o seu plano, de modo a frequentar o ginásio.

Atividade Fitness

Termo geral para todas as atividades existentes no “Ginásio 100calorias”, que serão lecionadas por apenas um professor. Cada aula irá conter a sua duração e nome, para que o cliente saiba as informações da atividade a participar.

Máquina

Entidade que representa todas as máquinas utilizadas nas diversas aulas de fitness. Esta contém a quantidade de máquinas disponíveis para gerir o número dos equipamentos utilizados nas atividades.

3.2.1 Dicionário de dados das entidades

Entidade	Descrição	Alcunha	Ocorrência
Cliente	Entidade que recorre aos serviços oferecidos pelo ginásio.	Aluno	O cliente é uma entidade fundamental uma vez que é este que faz uso do ginásio.
Professor	Entidade representativa do responsável por lecionar as diversas atividades e criação de planos	-	O professor é a base do funcionamento das atividades lecionadas no ginásio.
Plano	Entidade que contém as informações referentes às aulas que cada aluno deverá frequentar.	-	O plano define o percurso dos nossos clientes no ginásio.
Atividade Fitness	Entidade representativa das aulas lecionadas no ginásio pelos diferentes professores.	Aula	A atividade fitness é o motivo pelo qual os clientes frequentam o “Ginásio 100calorias”.
Máquina	Entidade que contém toda a informação referente aos equipamentos necessários para as atividades.	Equipamento	A máquina é uma peça chave dado que torna as atividades mais dinâmicas.

Tabela 1 - Dicionário de dados das entidades

3.3 Identificação e caracterização dos relacionamentos

Nesta secção procedemos à identificação dos relacionamentos estabelecidos entre as entidades ilustradas anteriormente. É também apresentada informação adicional relativa a cada relacionamento.

Cliente e Plano

O relacionamento “possui” entre a entidade *Cliente* e a entidade *Plano* caracteriza-se por uma cardinalidade de 1 para N, uma vez que um cliente pode possuir vários planos e um plano corresponde a apenas um aluno.

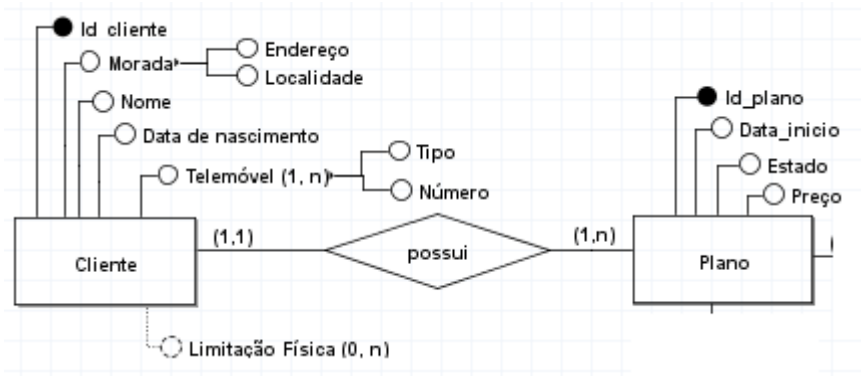


Figura 1 - Relacionamento Cliente e Plano

Plano e Professor

O relacionamento “criado” entre a entidade *Plano* e a entidade *Professor* caracteriza-se por uma cardinalidade de N para 1, dado que um professor pode elaborar diversos planos e um plano é criado por apenas um professor.

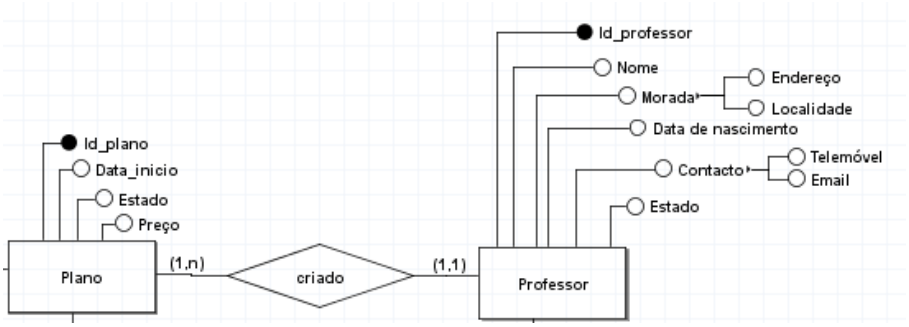


Figura 2 - Relacionamento Plano e Professor

Plano e Atividade fitness

O relacionamento “constituído” entre a entidade *Plano* e a entidade *Atividade Fitness* caracteriza-se por uma cardinalidade de N para N, porque um plano pode ser constituído por várias atividades fitness e uma atividade pode fazer parte de diversos planos.

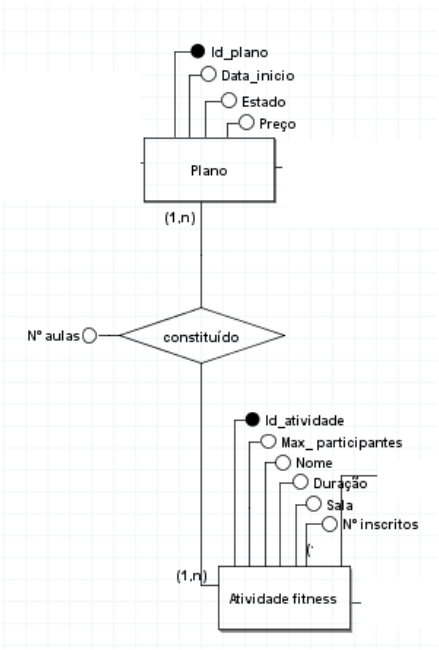


Figura 3 - Relacionamento Plano e Atividade fitness

Professor e Atividade fitness

O relacionamento “leciona” entre a entidade *Professor* e a entidade *Atividade Fitness* caracteriza-se por uma cardinalidade de 1 para N, uma vez que um professor pode lecionar várias atividades fitness, mas uma aula só é lecionada por um professor.

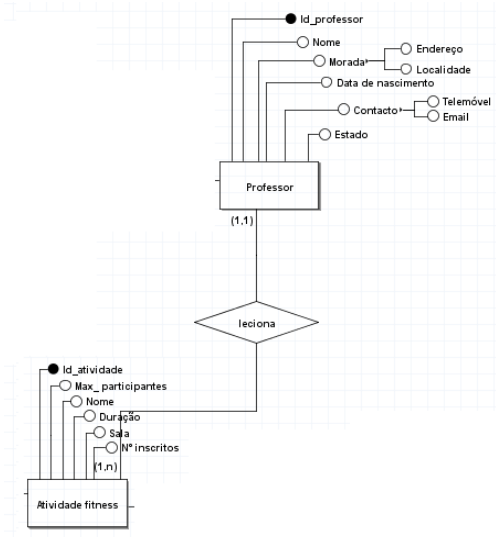


Figura 4 - Relacionamento Professor e Atividade fitness

Atividade fitness e Máquina

O relacionamento “contém” entre a entidade *Atividade Fitness* e a entidade *Máquina* caracteriza-se por uma cardinalidade de N para N, visto que uma atividade fitness pode necessitar de várias máquinas e estas podem estar presente em diversas atividades fitness.

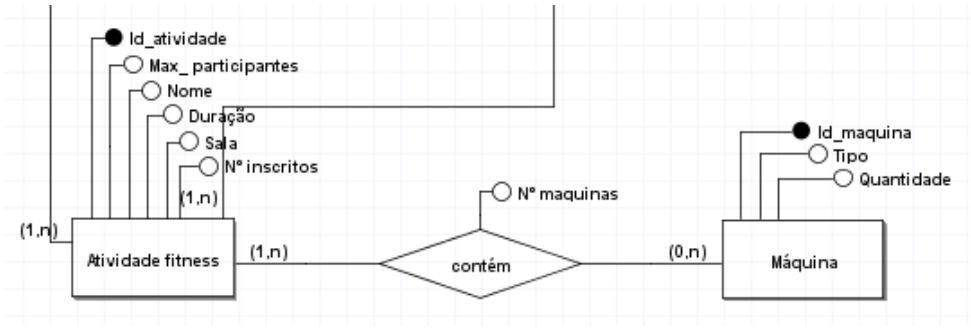


Figura 5 - Relacionamento Atividade fitness e Máquina

3.3.1 Dicionário de relacionamento

Entidade	Multiplicidade	Relacionamento	Multiplicidade	Entidade
Cliente	1..1	possui	1..N	Plano
Plano	1..N	criado	1..1	Professor
Plano	1..N	constituído	1..N	Atividade fitness
Atividade fitness	1..N	contém	0..N	Máquina
Professor	1..1	leciona	1..N	Atividade fitness

Tabela 2 - Dicionário de relacionamento

3.4 Identificação e Associação de Atributos aos Tipos de Entidades e de Relacionamentos

Entidade/Relacionamento	Atributos	Descrição	Tipo do atributo	Tipo de dados e tamanho
Cliente (Entidade)	**Id_cliente**	Identificador do cliente	Chave primária	INT
	Morada Endereço	Endereço postal do cliente	Composto	VARCHAR(45)
	Localidade	Localidade do cliente		VARCHAR(45)
	Nome	Nome do cliente	Simples	VARCHAR(45)
	Data de nascimento	Data de nascimento do cliente	Simples	DATE
	Limitação Física	Limitações de um cliente	Multivalorado	VARCHAR(45)

	Telemóvel Tipo Número	Tipo de contacto associado Contacto telefónico	Composto, Multivalorado	VARCHAR(45) CHAR(9)
Plano (Entidade)	**Id_plano**	Identificador do plano	Chave primária	INT
	Preço	Preço a pagar pelo plano	Simples	DOUBLE
	Data de início	Data de início do plano	Simples	DATE
	Estado	Indica se o plano está a ser realizado pelo cliente	Simples	VARCHAR(7)
Professor (Entidade)	**Id_professor**	Identificador do professor	Chave primária	INT
	Nome	Nome do professor	Simples	VARCHAR(45)
	Morada Endereço Localidade	Endereço de postal do professor Código postal do professor	Composto	VARCHAR(45) VARCHAR(45)
	Data de nascimento	Data de nascimento do professor	Simples	DATE
	Estado	Indica se o professor ainda trabalha no ginásio	Simples	VARCHAR(7)
	Contacto Email Telemóvel	Email do cliente Número de telemóvel do cliente	Composto	VARCHAR(45) CHAR(9)
Atividade fitness (Entidade)	**Id_atividade**	Identificador da atividade	Chave primária	INT
	Nº Inscritos	Número de participantes	Simples	INT
	Max_participantes	Indica o número máximo de participantes na atividade	Simples	INT
	Nome	Nome da atividade	Simples	VARCHAR(45)
	Duração	Duração da atividade	Simples	TIME
	Sala	Sala da atividade	Simples	INT
	Nome	Nome da limitação	Simples	VARCHAR(45)
Máquina (Entidade)	**Id_máquina**	Identificador da máquina	Chave primária	INT
	Tipo	Tipo da máquina	Simples	VARCHAR(45)
	Quantidade	Número de máquinas	Simples	INT

Constituído (Relacionamento)	Nº aulas	Número de aulas a frequentar	Simples	INT
Contém (Relacionamento)	Nº máquinas	Número de máquinas utilizadas	Simples	INT

Tabela 3 - Dicionário de dados dos atributos das entidades e relacionamentos

3.5 Determinação das chaves candidatas, chaves primárias e chaves alternadas

Uma vez identificadas as entidades, os seus atributos e os relacionamentos estabelecidos, podemos agora verificar quais desses atributos poderão ser chaves primárias. É importante referir que uma chave primária corresponde a um atributo, que pode identificar a identidade inequivocamente.

As chaves candidatas são todos os atributos da entidade que a identificam univocamente e os que não forem escolhidos para chave primária denominam-se chaves alternadas. Assim, serão apresentadas de seguida as chaves de cada entidade do Modelo Conceptual.

Cliente
Chaves candidatas: Id_cliente
Chave primária: Id_cliente

Plano
Chaves candidatas: Id_plano
Chave primária: Id_plano

Professor
Chaves candidatas: Id_professor, Telemovel
Chave primária: Id_professor
Chaves alternadas: Telemovel

Atividade_Fitness
Chaves candidatas: Id_atividade, Nome
Chave primária: Id_atividade
Chaves alternadas: Nome

Máquina
Chaves candidatas: Id_maquina, Tipo
Chave primária: Id_maquina

Chaves alternadas: Tipo

3.6 Detalhe ou generalização de entidades

No nosso projeto não utilizamos generalização nem especialização de entidades no Modelo Conceptual.

3.7 Apresentação e explicação do diagrama ER

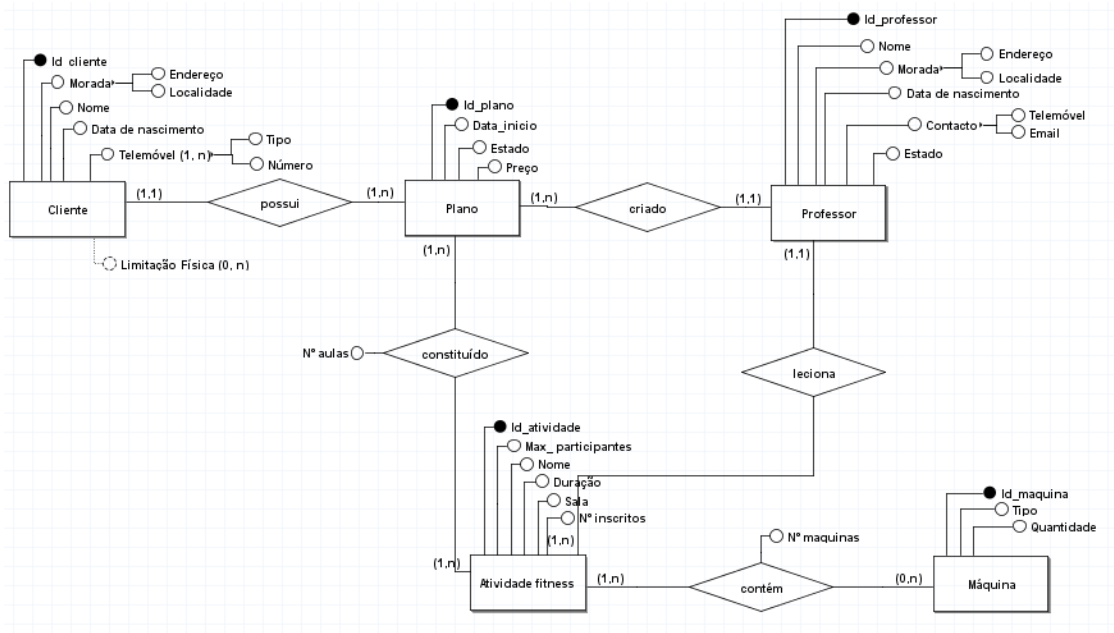


Figura 6 - Modelo Conceptual dos dados

3.8 Validação do modelo de dados com o utilizador

Uma vez terminado o Modelo Conceptual, é necessário a validação do mesmo. Para tal, é fundamental ser possível responder a todas as perguntas que o utilizador possa elaborar. Assim, escolhemos as questões posteriormente expostas, de modo a verificar a viabilidade com o Modelo Conceptual.

1. Qual a atividade fitness mais frequentada por um determinado aluno? **(RE 23)**

   Para obter a informação necessária para a resposta, é fundamental recorrer às entidades *Aluno*, *Plano* e *Atividade Fitness*. Tendo em conta o relacionamento entre o *Aluno* e o *Plano* é possível ter acesso a todos os planos realizados pelo aluno. Através do relacionamento entre o *Plano* e a *Atividade Fitness* obtemos o número de aulas de cada atividade. Assim, é possível determinar qual a mais frequentada.

2. Quais os planos elaborados por um professor ordenados pelo preço, bem como o cliente ao qual o plano foi atribuído? **(RE 24)**

   Serão necessárias as entidades *Professor, Plano e Cliente*. Através do relacionamento entre o *Professor* e o *Plano*, obtemos todos os planos elaborados por este. De seguida, obtemos o cliente cujo plano foi atribuído pelo relacionamento entre *Plano e Cliente.*

3. Quais os planos realizados por um dado cliente com um determinado estado? **(RE 25)**

   De modo a responder a esta pergunta, torna-se imperativo obter a informação das entidades *Cliente* e *Plano*. Através do relacionamento entre estas será possível obter todos os planos realizados pelo aluno. Assim, conseguimos obter os planos com o estado pretendido.

4. Quais as atividades fitness lecionadas por um professor? **(RE 26)**

   Para responder a esta questão será necessário obter a informação das entidades *Professor* e *Atividade Fitness*, dado que através do relacionamento entre estas entidades conseguimos verificar as atividades lecionadas por cada professor.

5. Quais os nomes e contactos dos alunos que frequentam uma dada atividade? **(RE 27)**

   Serão necessárias as informações das entidades *Cliente*, *Plano* e *Atividade Fitness*. Através do relacionamento entre a *Atividade Fitness* e o *Plano* conseguimos obter todos os planos cuja a atividade está incluída. De seguida, obtemos os clientes a que os planos correspondem com o relacionamento *Cliente* e *Plano*. Assim, conseguimos aceder ao contacto desses clientes.

4 Modelo lógico

Terminada a conceptualização do problema apresentado através do modelo anteriormente elaborado, torna-se agora necessário a construção do mesmo num Modelo Lógico. Esta será uma etapa fundamental para o desenvolvimento do nosso sistema de gestão da base de dados uma vez que nos permitirá derivar os relacionamentos. Assim, quando o Modelo Lógico estiver validado, tendo em conta as formas normais, este irá suportar o nosso problema.

4.1 Construção e validação do modelo de dados lógico

Para a elaboração do Modelo Lógico, começamos por derivar/criar todas as relações que retratam as entidades, os relacionamentos e os atributos previamente identificados no Modelo Conceptual.

4.1.1 Entidades fortes

Uma entidade forte define-se por possuir uma chave primária, que a identifica e não depende de outras entidades. No Modelo Lógico, cada entidade forte será representada numa tabela, em que cada um dos seus atributos constitui uma coluna. Para os atributos compostos, na tabela apenas serão apresentados os atributos simples que o constituem.

Cliente (Id_cliente, Nome, Data_nascimento, Endereco)
Chave primária: Id_cliente

Id_cliente	Nome	Data_nascimento	Endereco
1	João Sousa	1970-06-12	Rua das Lajes nº52 4700-500
(…)	(…)	(…)	(…)

Tabela 4 – Representação da entidade Cliente

Plano (Id_plano, Preco, Data_inicio, Estado)
Chave primária: Id_plano

Id_plano	Estado	Data_inicio	Preco
1	Ativo	2017-10-01	50
(…)		(…)	(…)

Tabela 5 - Representação da entidade Plano

Professor (Id_professor, Nome, Endereco, Data_nascimento, Telemovel, Email, Estado)

Chave primária: Id_professor

Id_professor	Nome	Endereco	Data_nascimento	Telemovel	Email	Estado
23	Ana Dias	Rua da Capela nº114 4700-234	1980-08-05	912567883	anad@gmail.com	Ativo
(…)	(…)	(…)	(…)	(…)	(…)	(…)

Tabela 6 - Representação da entidade Professor

Atividade_Fitness (Id_atividade, Max_participantes, Nr_inscritos, Nome, Duracao, Sala)

Chave primária: Id_atividade

Id_atividade	Max_participantes	Nr_inscritos	Nome	Duracao	Sala
57	20	15	Cycling	40	3
(…)	(…)	(…)	(…)	(…)	(…)

Tabela 7 - Representação da entidade Atividade_Fitness

Maquina (Id_maquina, Tipo, Quantidade)

Chave primária: Id_maquina

Id_maquina	Tipo	Quantidade
6	Passadeira	30
(…)	(…)	(…)

Tabela 8 - Representação da entidade Máquina

4.1.2 Relacionamentos de um para muitos (1:N)

Neste tipo de relacionamento, a entidade que apresenta multiplicidade N possui como atributo a chave primária da entidade com multiplicidade 1. Este atributo é considerado uma chave estrangeira.

Plano (Id_plano, Estado, Preco, Data_inicio)
Chave Primária: Id_plano
Chave Estrangeira: Id_cliente, Id_professor

Atividade_Fitness (Id_atividade, Max_participantes, Nr_inscritos, Nome, Duracao, Sala)
Chave Primária: Id_atividade
Chave Estrangeira: Id_professor

Como os clientes e os professores do ginásio são maioritariamente moradores locais, o atributo localidade irá se repetir várias vezes, pelo que sentimos a necessidade de criar uma tabela *Localidade*. Atendendo ao facto de que vários clientes e professores terão a mesma localidade, a nova tabela terá como atributos o Nome e o Id_Localidade que corresponde à chave primária e que será uma chave estrangeira em Cliente e Professor, como se pode verificar a seguir.

Cliente (Id_cliente, Nome, Data_nascimento, Endereço)
Chave Primária: Id_cliente
Chave Estrangeira: Id_localidade

Professor (Id_professor, Nome, Endereço, Data_nascimento, Telemovel, Email, Estado)
Chave primária: Id_professor
Chave Estrangeira: Id_localidade

4.1.3 Relacionamentos de muitos para muitos (N:M)

Este tipo de multiplicidade gera um novo relacionamento, onde a chave primária é composta pelas chaves primárias de cada uma das entidades envolvidas.

Plano_Atividade_Fitness (Nr_aulas)
Chave primária: Id_plano, Id_atividade

Atividade_Fitness_Maquina (Nr_maquinas)
Chave primária: Id_atividade, Id_maquina

4.1.1 Atributos multivalorados

Este tipo de atributo é utilizado, quando para a mesma entidade, pode assumir diversos valores. Um atributo multivalorado cria um novo relacionamento, em que a chave primária é composta pela chave primária da entidade e o próprio atributo.

No Modelo Conceptual, consideramos o atributo Telemovel da entidade *Cliente* como multivalorado, dado que este pode ter mais do que um contacto telefónico, nomeadamente, o pessoal e um para o qual se possa ligar em caso de acidente.

Telemóvel (Tipo, Numero)
Chave primária: Numero
Chave Estrangeira: Id_cliente

Outro atributo multivalorado presente no nosso Modelo Conceptual é a limitação física, contudo verificamos que esta estabelece com o cliente um relacionamento de muitos para muitos e não apenas de um para muitos como no telefone. Assim, sentimos a necessidade de criar dois relacionamentos, uma tabela *Limitacao_Fisica* pelo facto de ser um atributo multivalorado e a tabela *Cliente_Limitacao_Fisica*, proveniente do relacionamento de muitos para muitos estabelecido.

Cliente_Limitacao_Fisica()
Chave primária: Id_cliente, Id_limitacao

Limitacao_Fisica (Id_limitacao, Nome)
Chave primária: Id_limitacao

4.2 Desenho do modelo lógico

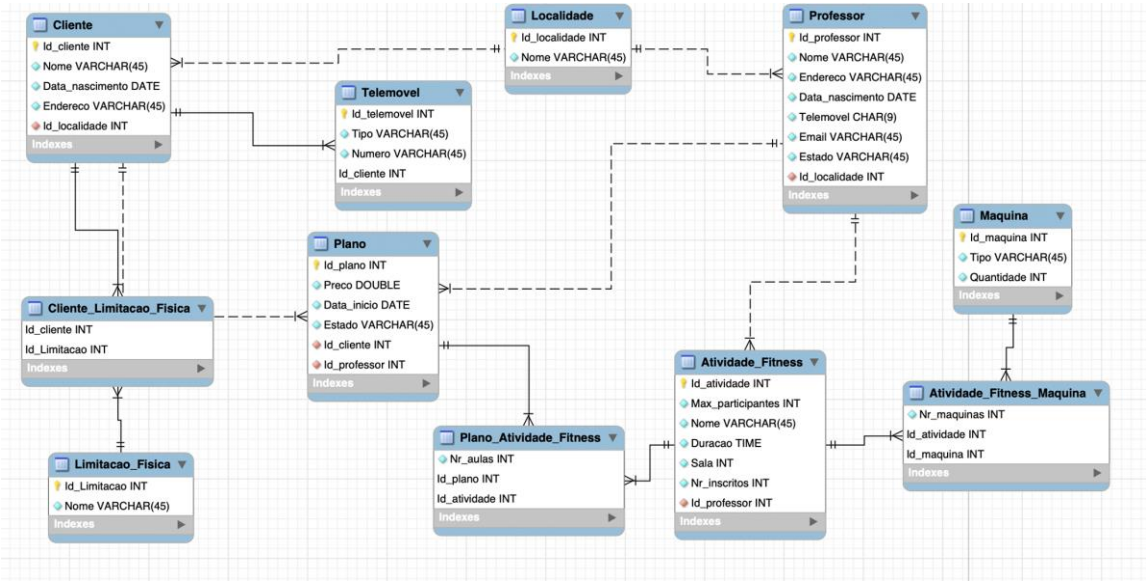


Figura 7 - Modelo Lógico

4.3 Validação das relações através da normalização

No modelo relacional de uma base de dados o objetivo é agrupar os atributos em relações, uma vez que existem dependências entre estes. Desta forma, é possível obter um Modelo Lógico estruturalmente consistente e com a mínima redundância.

Tendo em conta que a normalização tem como objetivo assegurar que as relações têm um número mínimo, mas suficiente de atributos necessários para suportar as exigências relativas aos dados a guardar, podemos perceber melhor cada atributo e o que este representa na base de dados.

Para proceder à validação do modelo através de normalização, começamos por identificar as dependências funcionais entre os atributos.

Dependências Funcionais de Cliente

Id_cliente – Nome, Data_nascimento, Endereco, Id_localidade

Dependências Funcionais de Plano

Id_plano – Data_inicio, Preco, Estado, Id_cliente, Id_professor

Dependências Funcionais de Professor

Id_professor – Nome, Endereco, Data_nascimento, Telemovel, Email, Estado, Id_localidade

Dependências Funcionais de Atividade Fitness

Id_atividade – Max_participantes, Nome, Duracao, Sala, Nr_inscritos, Id_professor

Dependências Funcionais de Maquina

Id_maquina – Tipo, Quantidade

Dependências Funcionais de Limitacao_fisica

Id_Limitacao – Nome

Dependências Funcionais de Plano_Atividade_Fitness

Id_plano, Id_atividade – Nr_aulas

Dependências Funcionais de Atividade_Fitness_Maquina

Id_atividade, Id_maquina – Nr_maquinas

4.3.1 Primeira Forma Normal (1FN)

Um relacionamento encontra-se na Primeira Forma Normal se todos os atributos forem atómicos, ou seja, não ser possível decompô-los em subconjuntos de atributos e não existirem diferentes atributos que descrevem o mesmo.

No nosso esquema conceptual temos atributos compostos, os atributos Morada e Contacto das entidades *Cliente* e *Professor*.

No caso dos atributos compostos, houve a necessidade de separar os sub-atributos pelas diferentes colunas das tabelas correspondentes. Assim, nas tabelas exclui-se a *Morada* e o *Contacto*, de modo a não violar a primeira forma normal.

Além de atributos compostos, temos também atributos multivalorados, a Limitação Física e o Telemóvel nos quais se tornou imperativo a criação de novas tabelas a *Limitacao_Fisica* e a *Telemovel*, respetivamente.

Podemos, assim, concluir que o nosso modelo está de acordo com a 1FN.

4.3.2 Segunda Forma Normal (2FN)

Uma relação está na Segunda Forma Normal se pertencer à 1FN e se todos os atributos não chave dependerem totalmente da chave primária. Como podemos constatar pela verificação das dependências funcionais anteriormente realizada, não existe dependência funcional total em relação à chave primária. Assim, o nosso modelo está de acordo com a 2FN.

4.3.3 Terceira Forma Normal (3FN)

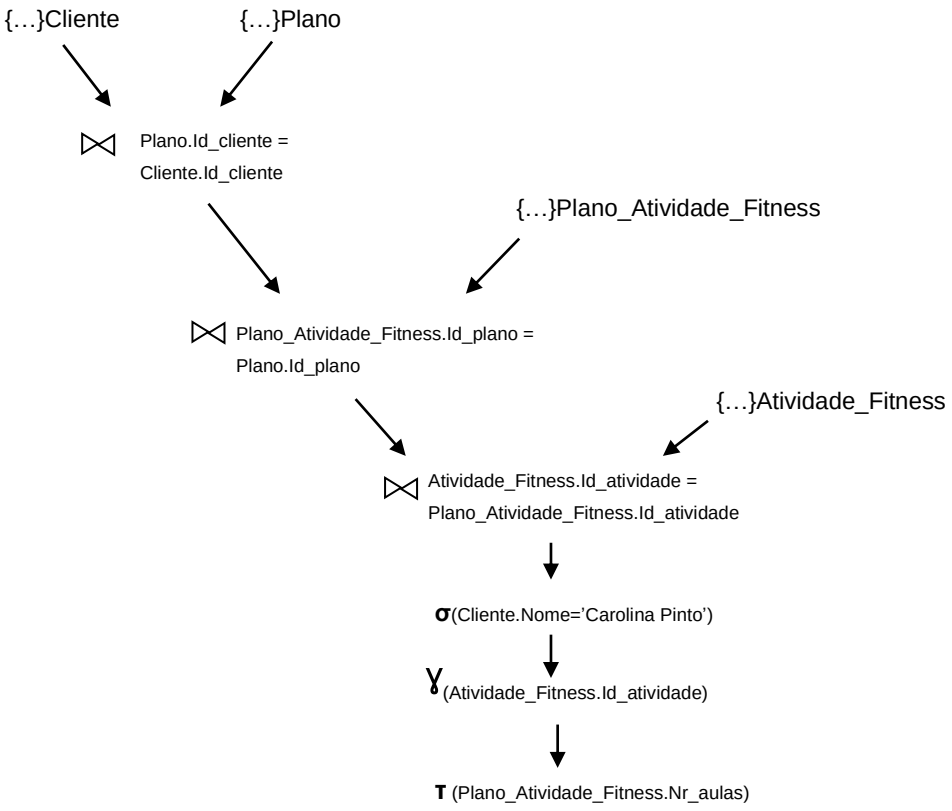
Um relacionamento encontra-se na Terceira Forma Normal se está na 2FN e se todos os atributos não chave só dependerem unicamente da chave primária e não dependerem de um outro atributo que por sua vez dependesse da chave primária. Como podemos verificar o nosso modelo não apresenta dependências transitivas então está de acordo com a 3FN.

4.4 Validação do modelo com interrogações do utilizador

Seguidamente serão ilustradas as resoluções de algumas interrogações, que correspondem a requisitos de exploração, em álgebra relacional. Tal procedimento permite-nos validar o modelo.

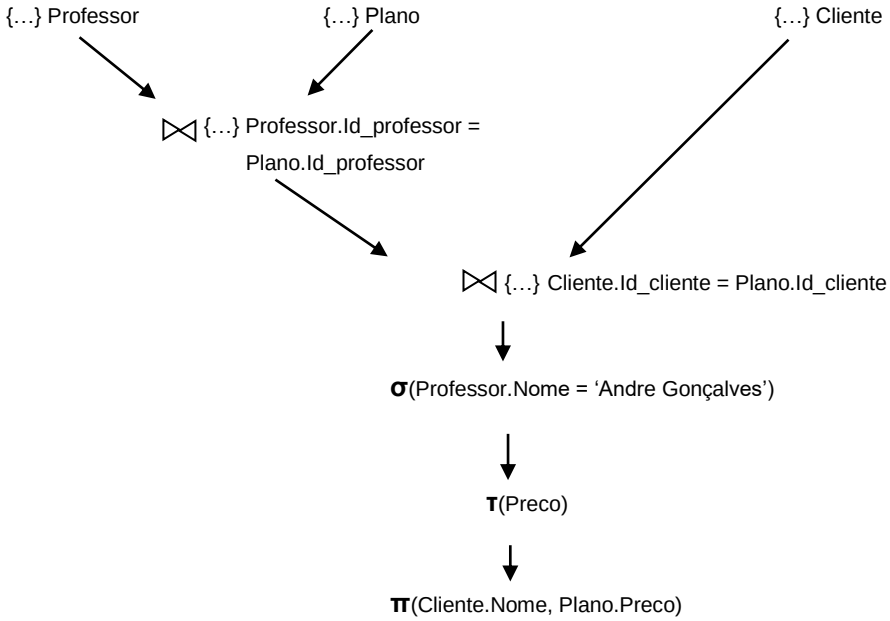
1. Qual a atividade fitness mais frequentada por um determinado aluno? **(RE 23)**

LIMIT1(**τ**(Plano_Atividade_Fitness.Nr_aulas)ɣ(Atividade_Fitness.Id_atividade)**σ**(Client e.Nome='Carolina Pinto') ((Atividade_Fitness)⋈ Atividade_Fitness.Id_atividade = Plano_Atividade_Fitness.Id_atividade(Plano_Atividade_Fitness) ⋈ Plano_Atividade_Fitness.Id_plano = Plano.Id_plano (Plano) ⋈ Plano.Id_cliente = Cliente.Id_cliente (Cliente)))



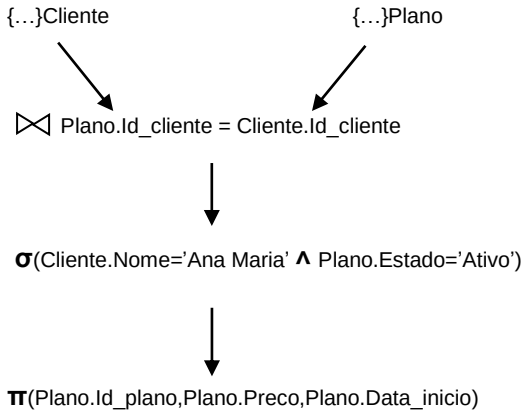
2. Quais os planos elaborados por um professor ordenados pelo preço, bem como o cliente ao qual o plano foi atribuído? **(RE 24)**

π (Cliente.Nome,Plano.Preco)(**τ** (Preco)(**σ** (Professor.Nome='Andre Gonçalves') ((Cliente)
⋈ Cliente.Id_cliente=Plano.Id_cliente(Plano)
⋈ Professor.Id_professor = Plano.Id_professor(Professor))



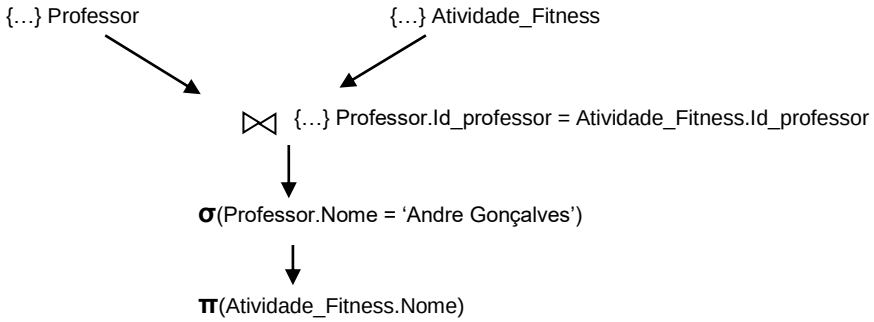
3. Quais os planos realizados por um dado cliente com um determinado estado? **(RE 25)**

π (Plano.Id_plano,Plano.Preco,Plano.Data_inicio)(**σ** (Cliente.Nome='Ana Maria'
^ Plano.Estado='Ativo') ((Cliente)⋈Plano.Id_cliente=Cliente.Id_cliente (Plano)))



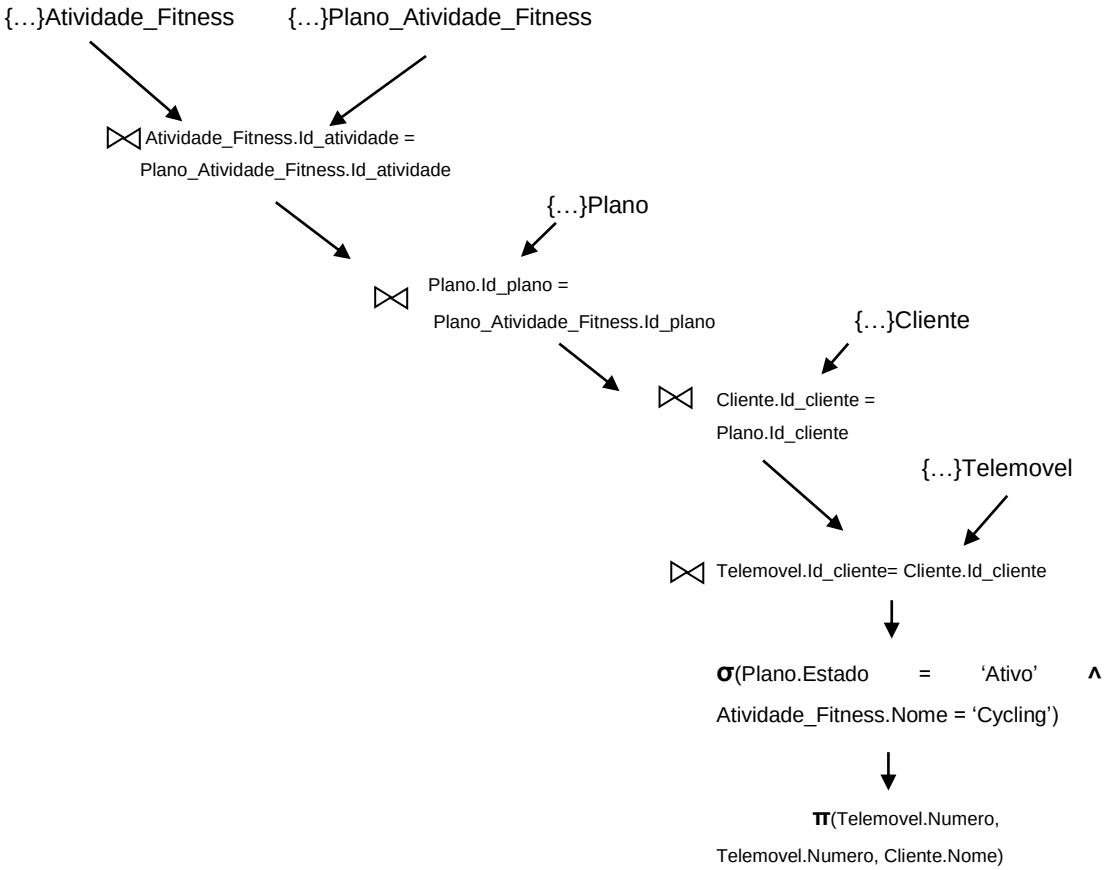
4. Quais as atividades fitness lecionadas por um professor? **(RE 26)**

π (Atividade_Fitness.Nome)(**σ** (Professor.Nome='Andre Gonçalves')((Professor) ⋈ Professor.Id_professor = Atividade_Fitness.Id_professor(Atividade_Fitness)))



5. Quais os nomes e contactos dos alunos que frequentam uma dada atividade? **(RE 27)**

π (Telemovel.Numero, Telemovel.Numero, Cliente.Nome) **σ**(Plano.Estado = 'Ativo' **^** Atividade_Fitness.Nome = 'Cycling') ((Telemovel) ⋈ Telemovel.Id_cliente = Cliente.Id_cliente (Cliente) ⋈ Cliente.Id_cliente = Plano.Id_cliente(Plano) ⋈ Plano.Id_plano=Plano_Atividade_Fitness.Id_plano(Plano_Atividde_Fitness) ⋈ Plano_Atividade_Fitness.Id_atividade (Atividade_Fitness))



4.5 Validação do modelo com as transações estabelecidas

De seguida iremos apresentar algumas das transações que consideramos importantes para o nosso projeto.

- **Registar Cliente**

  Para adicionar um cliente começa-se por elaborar um novo registo na tabela *Cliente,* adicionando a localidade na tabela *Localidade* caso essa ainda não exista. De seguida, torna-se imperativo inserir os contactos telefónicos do mesmo, efetuando novo registo na tabela *Telemovel*. Devem ser também registadas as limitações físicas do cliente caso seja necessário, inserindo na tabela *Limitacao_Fisica* a limitação se esta ainda não se encontrar na base de dados de modo a poder indicar as limitações do cliente na tabela *Cliente_Limitacao_Fisica*.

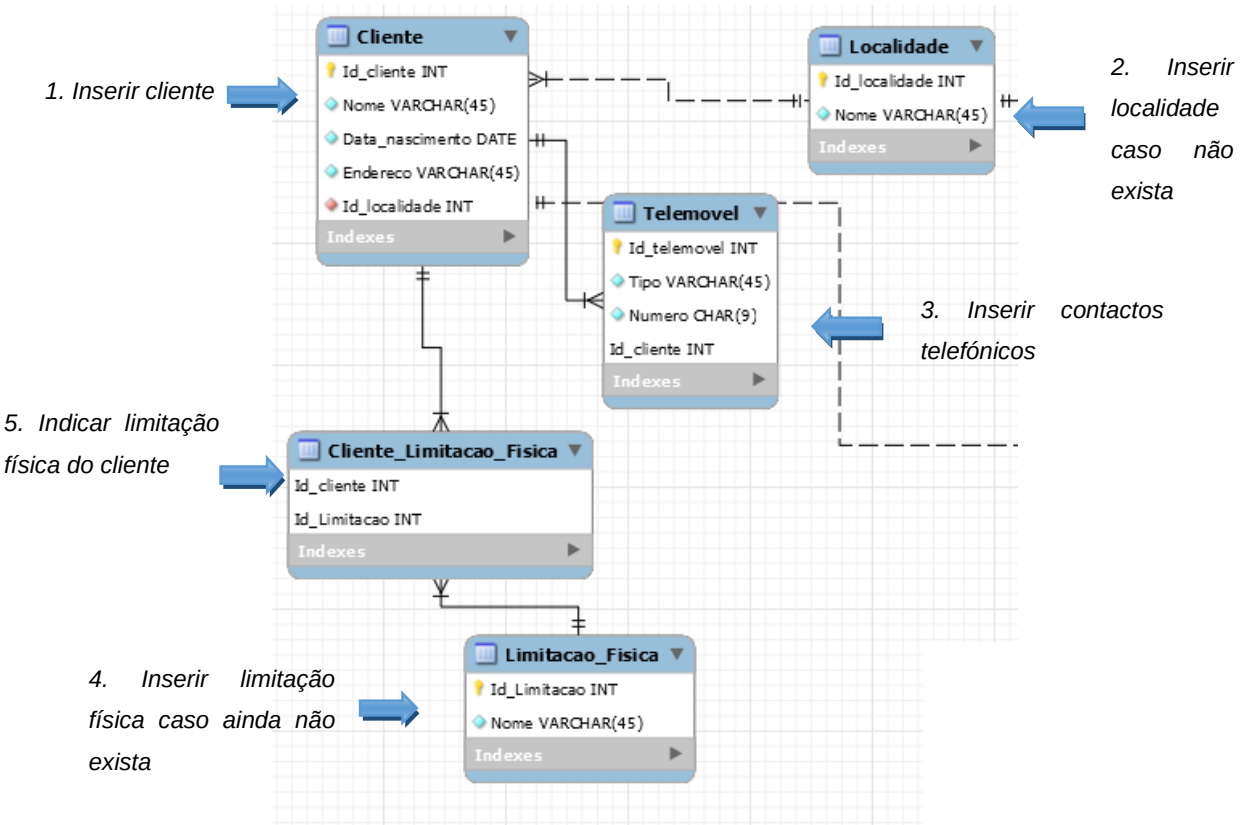


Figura 8 - Mapa de Transações de registar cliente

- **Registar Plano**

  Para adicionar um plano efetua-se primeiro o registo do plano fazendo um novo registo na tabela *Plano*. De seguida, é necessário indicar as atividades que constituem o plano, efetuando um novo registo na tabela *Plano_Atividade_Fitness*. Desta forma, será necessário

atualizar o número de participantes nessa atividade. É de notar que apenas é adicionada uma atividade ao plano caso esta ainda não se encontre com lotação máxima.

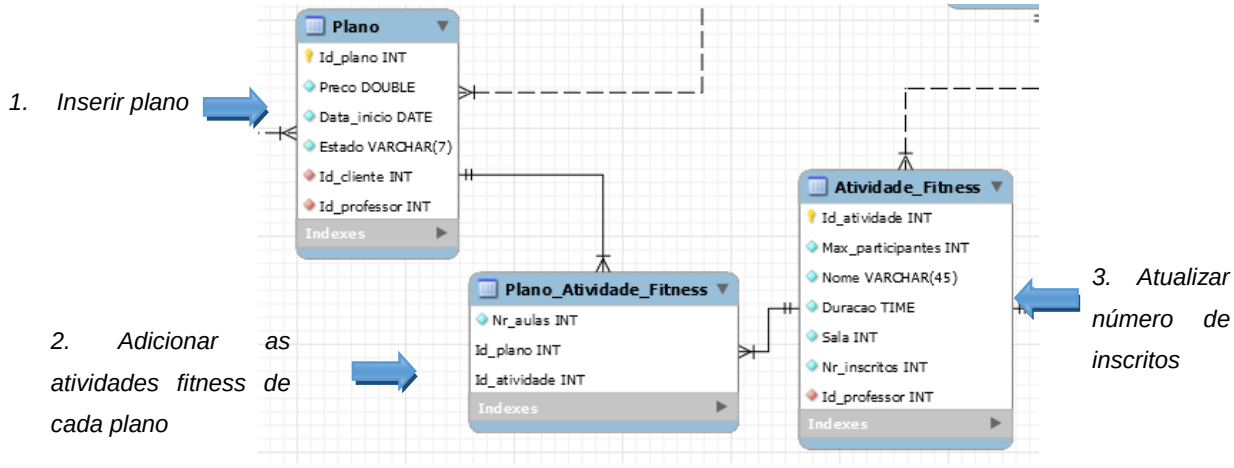


Figura 9 - Mapa de Transações de registar plano

- **Arquivar plano**

Para arquivar um plano, quando este deixa de estar em execução, é necessário alterar o estado do mesmo na tabela *Plano*. É de realçar que será necessário reduzir o número de inscritos nas atividades associadas a este, na tabela *Atividade_Fitness*.

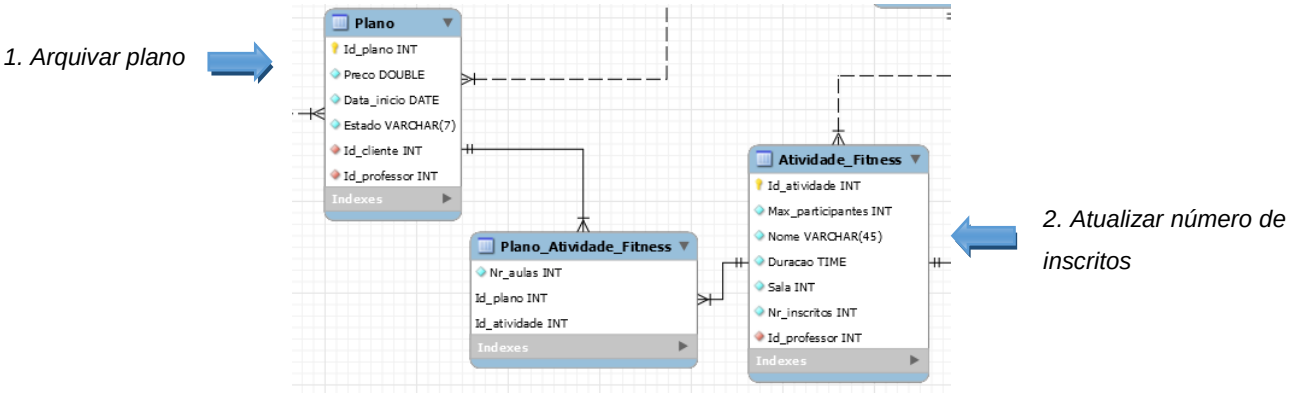


Figura 10 - Mapa de Transações de arquivar plano

4.6 Verificação das Restrições de Integridade

De forma a garantir que o nosso Modelo Lógico represente os dados de forma fiável, é necessária a aplicação de regras, nomeadamente, restrições de integridade, sendo possível

assim garantir, que a nossa base de dados seja consistente e completa. As restrições aplicadas são as seguintes:

- **Dados necessários**

  Numa Base de Dados existem campos de preenchimento obrigatório e que devem possuir um valor válido, como é o caso das chaves primárias. No SGBD que estamos a implementar, todos os campos são de preenchimento obrigatório.

- **Integridade de Domínio:**

  Esta restrição refere-se aos atributos, dado que todos possuem um domínio de valores admissíveis. É possível observar o tipo de dados de cada atributo no dicionário de dados dos atributos das entidades e relacionamentos.

- **Integridade da Entidade:**

  De modo a que todas as entidades sejam identificadas e que não exista repetição das mesmas, é necessário garantir a existência de uma chave primária, tendo esta de ser única e não nula. Como as chaves primárias definidas no Modelo Conceptual transitam para o Modelo Lógico, é possível verificar as que identificam cada entidade na secção *Determinação das chaves candidatas, chaves primárias e chaves alternadas*.

- **Restrições de Multiplicidade:**

  As multiplicidades são definidas na criação do Modelo Concetual, e como tal devem transitar para o Modelo Lógico. No entanto, existem casos particulares como os atributos multivalorados que devem ser analisados, como é retratado na secção *Construção e validação do modelo de dados lógicos*, onde também é possível encontrar os relacionamentos estabelecidos.

- **Integridade Referencial:**

  Esta restrição tem por base a relação entre chaves estrangeiras e primárias, de modo a salvaguardar os relacionamentos entre as tabelas quando é efetuado uma inserção, atualização ou remoção de um registo. Esta garante que os dados permanecem consistentes em todas as tabelas. É importante referenciar, que qualquer UPDATE que ocorra numa tabela cuja chave primária é estrangeira noutra é possível garantir que as tabelas que a contêm como chave estrangeira são também atualizadas, dado que seguem a opção CASCADE.

4.7 Revisão do modelo lógico com o utilizador

Após todo o processo realizado reunimos com o Sr. João, de modo a apresentar todo o trabalho desenvolvido, revendo os requisitos através das interrogações e transações. Como recebemos a aprovação do Sr. João podemos começar a elaborar o Modelo Físico.

5 Implementação Física

5.1 Seleção do sistema de gestão de bases de dados

Para o nosso projeto utilizamos como sistema de gestão de bases de dados o *MySQL*. Esta escolha baseou-se essencialmente no facto ser de fácil utilização, e de ter um bom desempenho. Um fator que também influenciou a nossa decisão foi o do software ser gratuito. Tendo em conta todas as vantagens enumeradas anteriormente, consideramos esta a melhor opção, pois a nossa base dados é de tamanho reduzido e as funcionalidades do *MySQL* permitem implementar todas as tarefas pretendidas de forma eficaz.

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

Terminado o Modelo Lógico podemos passar à elaboração do Modelo Físico, de modo a que as relações base e as suas restrições sejam sustentadas pelo sistema de gestão de bases de dados. Para tal será necessário realizar o desenho das relações base e das restrições gerais. É importante referir que para gerar o Modelo Físico através do Modelo Lógico, recorremos à ferramenta do *MySQL Workbench*, *Forward Engineer*.

5.2.1 Desenho das relações base

- **Relação Cliente**

  Domínio Id_cliente Inteiro
  Domínio Nome String tamanho variável
  Domínio Data_nascimento Data
  Domínio Endereço String tamanho variável
  Domínio Id_localidade Inteiro

  Cliente(
  Id_cliente INT NOT NULL,

Nome	VARCHAR(45)	NOT NULL
Data_nascimento	DATE	NOT NULL
Endereço	VARCHAR(45)	NOT NULL
Id_localidade	INT	NOT NULL

PRIMARY KEY(Id_cliente),

FOREIGN KEY (Id_localidade) **REFERENCES** Localidade(Id_localidade));

- **Relação Plano**

Domínio Id_plano	Inteiro
Domínio Preco	Número real
Domínio Data_inicio	Data
Domínio Estado	String de tamanho variável
Domínio Id_professor	Inteiro
Domínio Id_cliente	Inteiro

Plano(

Id_plano	INT	NOT NULL
Preco	DOUBLE	NOT NULL
Data_inicio	DATE	NOT NULL
Estado	VARCHAR(7)	NOT NULL
Id_professor	INT	NOT NULL
Id_cliente	INT	NOT NULL

PRIMARY KEY (Id_plano),

FOREIGN KEY (Id_professor) **REFERENCES** Professor(Id_professor),

FOREIGN KEY (Id_cliente) **REFERENCES** Cliente(Id_professor));

- **Relação Professor**

Domínio Id_professor	Inteiro
Domínio Nome	String tamanho variável
Domínio Data_nascimento	Data
Domínio Endereço	String tamanho variável
Domínio Id_localidade	Inteiro
Domínio Telemovel	String
Domínio Email	String tamanho variável
Domínio Estado	String tamanho variável

Professor (

Id_professor	INT	NOT NULL,
Nome	VARCHAR(45)	NOT NULL
Data_nascimento	DATE	NOT NULL

Endereço	VARCHAR(45)	NOT NULL
Id_localidade	INT	NOT NULL
Telemovel	CHAR(9)	NOT NULL
Email	VARCHAR(45)	NOT NULL
Estado	VARCHAR(7)	NOT NULL

PRIMARY KEY(Id_professor),

FOREIGN KEY (Id_localidade) **REFERENCES** Localidade(Id_localidade));

- **Relação Atividade_Fitness**

Domínio Id_atividade	Inteiro
Domínio Max_participantes	Inteiro
Domínio Nr_inscritos	Inteiro
Domínio Nome	String tamanho variável
Domínio Duração	Tempo
Domínio Sala	Inteiro
Domínio Id_professor	Inteiro

Atividade_Fitness(

Id_atividade	INT	NOT NULL
Max_participantes	INT	NOT NULL
Nr_inscritos	INT	NOT NULL
Nome	VARCHAR(45)	NOT NULL
Duracao	TIME	NOT NULL
Sala	INT	NOT NULL
Id_professor	INT	NOT NULL

PRIMARY KEY (Id_atividade),

FOREIGN KEY (Id_professor) **REFERENCES** Professor(Id_professor));

- **Relação Maquina**

Domínio Id_maquina	Inteiro
Domínio Tipo	String tamanho variável
Domínio Quantidade	Inteiro

Maquina (

Id_maquina	INT	NOT NULL
Tipo	VARCHAR(45)	NOT NULL
Quantidade	INT	NOT NULL

PRIMARY KEY (Id_maquina),

FOREIGN KEY (Id_atividade) **REFERENCES** Atividade_fitness(Id_atividade));

- **Relação Limitacao_Fisica**

  Domínio Id_Limitacao Inteiro

  Domínio Nome String tamanho variável

  Limitacao_Fisica (

  Id_Limitacao INT NOT NULL

  Nome VARCHAR(45) NOT NULL

  PRIMARY KEY (Id_Limitacao));

- **Relação Telemóvel**

  Domínio Id_telemovel Inteiro

  Domínio Número String

  Domínio Tipo String tamanho variável

  Domínio Id_cliente Inteiro

  Telemóvel (

  Id_telemovel INT NOT NULL

  Número CHAR(9) NOT NULL

  Tipo VARCHAR(45) NOT NULL

  Id_cliente INT NOT NULL

  PRIMARY KEY (Id_telemovel),

  FOREIGN KEY (Id_cliente) **REFERENCES** Cliente (Id_cliente));

- **Relação Localidade**

  Domínio Id_localidade Inteiro

  Domínio Nome String tamanho variável

  Localidade (

  Id_localidade INT NOT NULL

  Nome VARCHAR(45) NOT NULL

  PRIMARY KEY (Id_localidade));

- **Relação Plano_Atividade_Fitness**

  Domínio Nr_aulas Inteiro

  Domínio Id_plano Inteiro

  Domínio Id_atividade Inteiro

  Plano_Atividade_Fitness(

  Nr_aulas INT NOT NULL

  Id_plano INT NOT NULL

Id_atividade INT NOT NULL
PRIMARY KEY (Id_plano, Id_atividade),
FOREIGN KEY (Id_plano) **REFERENCES** Plano(Id_plano),
FOREIGN KEY (Id_atividade) **REFERENCES** Atividade_Fitness(Id_atividade));

- **Atividade_Fitness_Maquina**

Domínio Nr_maquinas Inteiro
Domínio Id_atividade Inteiro
Domínio Id_maquina Inteiro

Atividade_Fitness_Maquina(
Nr_maquinas INT NOT NULL
Id_atividade INT NOT NULL
Id_maquina INT NOT NULL
PRIMARY KEY (Id_atividade, Id_maquina),
FOREIGN KEY (Id_atividade) **REFERENCES** Atividade_Fitness(Id_atividade),
FOREIGN KEY (Id_maquina) **REFERENCES** Maquina (Id_maquina));

- **Cliente_Limitacao_Fisica**

Domínio Id_cliente Inteiro
Domínio Id_Limitacao Inteiro

Cliente_Limitacao_Fisica(
Id_cliente INT NOT NULL
Id_Limitacao INT NOT NULL
PRIMARY KEY (Id_cliente, Id_Limitacao),
FOREIGN KEY (Id_cliente) **REFERENCES** Cliente (Id_cliente),
FOREIGN KEY (Id_Limitacao) **REFERENCES** Limitacao_Fisica (Id_Limitacao));

5.2.2 Desenho das restrições

Nesta parte serão expostas as restrições gerais do problema agregadas por relação. Deste modo, serão apresentadas de seguida os scripts de criação.

- **Cliente**

```
-- -----------------------------------------------------
-- Table `ginasio`.`Cliente`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `ginasio`.`Cliente` (
  `Id_cliente` INT NOT NULL AUTO_INCREMENT,
  `Nome` VARCHAR(45) NOT NULL,
  `Data_nascimento` DATE NOT NULL,
  `Endereco` VARCHAR(45) NOT NULL,
  `Id_localidade` INT NOT NULL,
  PRIMARY KEY (`Id_cliente`),
  INDEX `fk_Cliente_Localidade1_idx` (`Id_localidade` ASC),
  CONSTRAINT `fk_Cliente_Localidade1`
    FOREIGN KEY (`Id_localidade`)
    REFERENCES `ginasio`.`Localidade` (`Id_localidade`)
    ON DELETE NO ACTION
    ON UPDATE CASCADE)
ENGINE = InnoDB;
```

Figura 11 - Criação da tabela Cliente

- **Plano**

```
-- -----------------------------------------------------
-- Table `ginasio`.`Plano`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `ginasio`.`Plano` (
  `Id_plano` INT NOT NULL AUTO_INCREMENT,
  `Preco` DOUBLE NOT NULL,
  `Data_inicio` DATE NOT NULL,
  `Estado` VARCHAR(7) NOT NULL,
  `Id_cliente` INT NOT NULL,
  `Id_professor` INT NOT NULL,
  PRIMARY KEY (`Id_plano`),
  INDEX `fk_Plano_Cliente1_idx` (`Id_cliente` ASC),
  INDEX `fk_Plano_Professor1_idx` (`Id_professor` ASC),
  CONSTRAINT `fk_Plano_Cliente1`
    FOREIGN KEY (`Id_cliente`)
    REFERENCES `ginasio`.`Cliente` (`Id_cliente`)
    ON DELETE NO ACTION
    ON UPDATE CASCADE,
  CONSTRAINT `fk_Plano_Professor1`
    FOREIGN KEY (`Id_professor`)
    REFERENCES `ginasio`.`Professor` (`Id_professor`)
    ON DELETE NO ACTION
    ON UPDATE CASCADE)
ENGINE = InnoDB;
```

Figura 12 - Criação da tabela Plano

- **Professor**

```
-- -----------------------------------------------------
-- Table `ginasio`.`Professor`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `ginasio`.`Professor` (
  `Id_professor` INT NOT NULL AUTO_INCREMENT,
  `Nome` VARCHAR(45) NOT NULL,
  `Endereco` VARCHAR(45) NOT NULL,
  `Data_nascimento` DATE NOT NULL,
  `Telemovel` CHAR(9) NOT NULL,
  `Email` VARCHAR(45) NOT NULL,
  `Estado` VARCHAR(7) NOT NULL,
  `Id_localidade` INT NOT NULL,
  PRIMARY KEY (`Id_professor`),
  INDEX `fk_Professor_Localidade1_idx` (`Id_localidade` ASC),
  CONSTRAINT `fk_Professor_Localidade1`
    FOREIGN KEY (`Id_localidade`)
    REFERENCES `ginasio`.`Localidade` (`Id_localidade`)
    ON DELETE NO ACTION
    ON UPDATE CASCADE)
ENGINE = InnoDB;
```

Figura 13 - Criação da tabela Professor

- **Atividade_Fitness**

```
-- -----------------------------------------------------
-- Table `ginasio`.`Atividade_Fitness`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `ginasio`.`Atividade_Fitness` (
  `Id_atividade` INT NOT NULL AUTO_INCREMENT,
  `Max_participantes` INT NOT NULL,
  `Nome` VARCHAR(45) NOT NULL,
  `Duracao` TIME NOT NULL,
  `Sala` INT NOT NULL,
  `Nr_inscritos` INT NOT NULL,
  `Id_professor` INT NOT NULL,
  PRIMARY KEY (`Id_atividade`),
  INDEX `fk_Atividade_Fitness_Professor1_idx` (`Id_professor` ASC),
  CONSTRAINT `fk_Atividade_Fitness_Professor1`
    FOREIGN KEY (`Id_professor`)
    REFERENCES `ginasio`.`Professor` (`Id_professor`)
    ON DELETE NO ACTION
    ON UPDATE CASCADE)
ENGINE = InnoDB;
```

Figura 14 - Criação da tabela Atividade_Fitness

- **Maquina**

```
-- -----------------------------------------------------
-- Table `ginasio`.`Maquina`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `ginasio`.`Maquina` (
  `Id_maquina` INT NOT NULL AUTO_INCREMENT,
  `Tipo` VARCHAR(45) NOT NULL,
  `Quantidade` INT NOT NULL,
  PRIMARY KEY (`Id_maquina`))
ENGINE = InnoDB;
```

Figura 15 - Criação da tabela Maquina

- **Limitacao_Fisica**

```
-- -----------------------------------------------------
-- Table `ginasio`.`Limitacao_Fisica`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `ginasio`.`Limitacao_Fisica` (
  `Id_Limitacao` INT NOT NULL AUTO_INCREMENT,
  `Nome` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`Id_Limitacao`))
ENGINE = InnoDB;
```

Figura 16 - Criação da tabela Limitacao_Fisica

- **Telemovel**

```
-- -----------------------------------------------------
-- Table `ginasio`.`Telemovel`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `ginasio`.`Telemovel` (
  `Id_telemovel` INT NOT NULL AUTO_INCREMENT,
  `Tipo` VARCHAR(45) NOT NULL,
  `Numero` CHAR(9) NOT NULL,
  `Id_cliente` INT NOT NULL,
  PRIMARY KEY (`Id_telemovel`, `Id_cliente`),
  INDEX `fk_Telemovel_Cliente1_idx` (`Id_cliente` ASC),
  CONSTRAINT `fk_Telemovel_Cliente1`
    FOREIGN KEY (`Id_cliente`)
    REFERENCES `ginasio`.`Cliente` (`Id_cliente`)
    ON DELETE NO ACTION
    ON UPDATE CASCADE)
ENGINE = InnoDB;
```

Figura 17 - Criação da tabela Telemovel

- **Localidade**

```
-- -----------------------------------------------------
-- Table `ginasio`.`Localidade`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `ginasio`.`Localidade` (
  `Id_localidade` INT NOT NULL AUTO_INCREMENT,
  `Nome` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`Id_localidade`))
ENGINE = InnoDB;
```

Figura 18 - Criação da tabela Localidade

- **Plano_Atividade_Fitness**

```
-- -----------------------------------------------------
-- Table `ginasio`.`Plano_Atividade_Fitness`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `ginasio`.`Plano_Atividade_Fitness` (
  `Nr_aulas` INT NOT NULL,
  `Id_plano` INT NOT NULL,
  `Id_atividade` INT NOT NULL,
  PRIMARY KEY (`Id_plano`, `Id_atividade`),
  INDEX `fk_Plano_has_Atividade_Fitness_Atividade_Fitness1_idx` (`Id_atividade` ASC),
  INDEX `fk_Plano_has_Atividade_Fitness_Plano_idx` (`Id_plano` ASC),
  CONSTRAINT `fk_Plano_has_Atividade_Fitness_Plano`
    FOREIGN KEY (`Id_plano`)
    REFERENCES `ginasio`.`Plano` (`Id_plano`)
    ON DELETE NO ACTION
    ON UPDATE CASCADE,
  CONSTRAINT `fk_Plano_has_Atividade_Fitness_Atividade_Fitness1`
    FOREIGN KEY (`Id_atividade`)
    REFERENCES `ginasio`.`Atividade_Fitness` (`Id_atividade`)
    ON DELETE NO ACTION
    ON UPDATE CASCADE)
ENGINE = InnoDB;
```

Figura 19 - Criação da tabela Plano_Atividade_Fitness

- **Atividade_Fitness_Maquina**

```
-- -----------------------------------------------------
-- Table `ginasio`.`Atividade_Fitness_Maquina`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `ginasio`.`Atividade_Fitness_Maquina` (
  `Nr_maquinas` INT NOT NULL,
  `Id_atividade` INT NOT NULL,
  `Id_maquina` INT NOT NULL,
  PRIMARY KEY (`Id_atividade`, `Id_maquina`),
  INDEX `fk_Atividade_Fitness_has_Maquina_Maquina1_idx` (`Id_maquina` ASC),
  INDEX `fk_Atividade_Fitness_has_Maquina_Atividade_Fitness1_idx` (`Id_atividade` ASC),
  CONSTRAINT `fk_Atividade_Fitness_has_Maquina_Atividade_Fitness1`
    FOREIGN KEY (`Id_atividade`)
    REFERENCES `ginasio`.`Atividade_Fitness` (`Id_atividade`)
    ON DELETE NO ACTION
    ON UPDATE CASCADE,
  CONSTRAINT `fk_Atividade_Fitness_has_Maquina_Maquina1`
    FOREIGN KEY (`Id_maquina`)
    REFERENCES `ginasio`.`Maquina` (`Id_maquina`)
    ON DELETE NO ACTION
    ON UPDATE CASCADE)
ENGINE = InnoDB;
```

Figura 20 - Criação da tabela Atividade_Fitness_Maquina

- **Cliente_Limitacao_Fisica**

```
-- -----------------------------------------------------
-- Table `ginasio`.`Cliente_Limitacao_Fisica`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `ginasio`.`Cliente_Limitacao_Fisica` (
  `Id_cliente` INT NOT NULL AUTO_INCREMENT,
  `Id_Limitacao` INT NOT NULL,
  PRIMARY KEY (`Id_cliente`, `Id_Limitacao`),
  INDEX `fk_Cliente_has_Limitacao_Fisica_Limitacao_Fisica1_idx` (`Id_Limitacao` ASC),
  INDEX `fk_Cliente_has_Limitacao_Fisica_Cliente1_idx` (`Id_cliente` ASC),
  CONSTRAINT `fk_Cliente_has_Limitacao_Fisica_Cliente1`
    FOREIGN KEY (`Id_cliente`)
    REFERENCES `ginasio`.`Cliente` (`Id_cliente`)
    ON DELETE NO ACTION
    ON UPDATE CASCADE,
  CONSTRAINT `fk_Cliente_has_Limitacao_Fisica_Limitacao_Fisica1`
    FOREIGN KEY (`Id_Limitacao`)
    REFERENCES `ginasio`.`Limitacao_Fisica` (`Id_Limitacao`)
    ON DELETE NO ACTION
    ON UPDATE CASCADE)
ENGINE = InnoDB;
```

Figura 21 – Criação da tabela Cliente_Limitacao_Fisica

5.3 Tradução das interrogações do utilizador para SQL

De seguida serão apresentadas as respostas a algumas das interrogações realizadas pelo utilizador, que se encontram nos requisitos de exploração.

1. Qual a atividade fitness mais frequentada por um determinado aluno?

```
SELECT sum(PA.Nr_aulas) AS Nr_Aulas, A.Nome AS Atividade FROM Cliente AS C
INNER JOIN Plano AS P ON P.Id_cliente = C.Id_cliente
INNER JOIN Plano_Atividade_Fitness AS PA ON PA.Id_plano = P.Id_plano
INNER JOIN Atividade_Fitness AS A ON PA.Id_atividade = A.Id_atividade
WHERE C.Nome = 'Carolina Pinto'
GROUP BY A.Id_atividade
ORDER BY PA.Nr_aulas DESC
LIMIT 1;
```

Figura 22 - Resolução do requisito de exploração 23

2. Quais os planos elaborados por um professor ordenados pelo preço, bem como o cliente ao qual o plano foi atribuído?

```
DROP PROCEDURE IF EXISTS planos_professor;

DELIMITER $$
CREATE PROCEDURE planos_professor (IN nome_professor VARCHAR(45))
BEGIN
SELECT PL.Preco , C.Nome AS Cliente FROM Professor AS P
INNER JOIN Plano AS PL ON PL.Id_professor = P.Id_professor
INNER JOIN Cliente AS C ON C.Id_cliente = PL.Id_cliente
WHERE P.Nome = nome_professor
ORDER BY PL.preco DESC;
END $$
DELIMITER ;
```

Figura 23 – Resolução do requisito de exploração 24

3. Quais os planos realizados por um dado cliente com um determinado estado?

```
DROP PROCEDURE IF EXISTS estado_planos

DELIMITER $$
CREATE PROCEDURE estado_planos (IN nome_cliente VARCHAR(45), IN estado VARCHAR(45))
BEGIN
SELECT P.Id_plano, P.Preco, P.Data_inicio AS Inicio FROM Cliente AS C
INNER JOIN Plano AS P ON P.Id_cliente = C.Id_cliente
WHERE C.Nome = nome_cliente
AND P.Estado = estado;
END $$
DELIMITER ;
```

Figura 24 - Resolução do requisito de exploração 25

4. Quais as atividades fitness lecionadas por um professor?

```
DROP PROCEDURE IF EXISTS atividades_professor;

DELIMITER $$
CREATE PROCEDURE atividades_professor (IN nome_professor VARCHAR(45))
BEGIN
SELECT A.Nome FROM Professor AS P
INNER JOIN Atividade_Fitness AS A ON A.Id_professor = P.Id_professor
WHERE P.Nome = nome_professor;
END $$
DELIMITER ;
```

Figura 25 – Resolução do requisito de exploração 26

5. Quais os nomes e contactos dos alunos que frequentam uma dada atividade?

```
DROP PROCEDURE IF EXISTS alunos_atividade

DELIMITER $$
CREATE PROCEDURE alunos_atividade (IN nome_atividade VARCHAR(45))
BEGIN
SELECT C.Nome AS Cliente, T.Numero, T.Tipo FROM Atividade_Fitness AS A
INNER JOIN Plano_Atividade_Fitness AS PA ON PA.Id_atividade = A.Id_atividade
INNER JOIN Plano AS P ON P.Id_plano = PA.Id_plano
INNER JOIN Cliente AS C ON C.Id_cliente = P.Id_cliente
INNER JOIN Telemovel AS T ON C.Id_cliente = T.Id_cliente
WHERE P.Estado = 'Ativo'
AND A.Nome = nome_atividade;
END $$
DELIMITER ;
```

Figura 26 - Resolução do requisito 27

5.4 Tradução das transações estabelecidas para SQL

De seguida será exposta a implementação de todas as transações consideradas importantes para o projeto.

- **Inserir Cliente**

  Para inserir um cliente é necessário fazer primeiro o registo do mesmo na tabela *Cliente*. Note-se que a localidade do cliente pode ainda não se encontrar registada, caso tal aconteça é necessário fazer um novo registo na tabela *Localidade*.

```
DROP PROCEDURE IF EXISTS inserir_cliente;

DELIMITER $$
CREATE PROCEDURE inserir_cliente(IN nome VARCHAR(45), IN datanascimento DATE, IN endereco VARCHAR(45),
        IN localidade VARCHAR(45))

        BEGIN
        Declare erro Bool default 0;
        declare continue handler for SQLEXCEPTION set erro = 1;
        start transaction;
        set @A:=0;
        SELECT @A:= L.Id_localidade FROM Localidade AS L where L.Nome = localidade;
        IF(@A = 0) THEN
        INSERT INTO Localidade(Nome) VALUE (localidade);
        SELECT @A:= L.Id_localidade FROM Localidade AS L where L.Nome = localidade;
        END IF;
        INSERT INTO Cliente(Nome, Data_nascimento, Endereco, Id_localidade)
                VALUE (nome, datanascimento, endereco, @A);
            if erro then ROLLBACK;
    else commit;

    END IF;

END $$
DELIMITER ;
```

Figura 27 – Inserir Cliente

De seguida é efetuado o registo dos seus contactos na tabela *Telemovel*.

```
DROP PROCEDURE IF EXISTS inserir_telemovel;

DELIMITER $$
CREATE PROCEDURE inserir_telemovel(IN tipo VARCHAR(45), IN numero CHAR(9), IN nome VARCHAR(45))

        BEGIN
        Declare erro Bool default 0;
        declare continue handler for SQLEXCEPTION set erro = 1;
        start transaction;
        set @A:=0;
        SELECT @A:= C.Id_cliente FROM Cliente AS C where C.Nome = nome;
        INSERT INTO Telemovel(Tipo, Numero, Id_cliente)
                VALUE (tipo, numero, @A);
            if erro then ROLLBACK;
    else commit;

    END IF;

END $$
DELIMITER ;
```

Figura 28 - Inserir contacto telefónico

Por fim, se o cliente possuir limitações físicas estas deverão ser registadas na tabela *Cliente_Limitação_Física*, verificando-se previamente se tal limitação já se encontra registada, em caso negativo é efetuado também o registo da limitação na tabela *Limitacao_Fisica*.

```
DROP PROCEDURE IF EXISTS inserir_limitacao;

DELIMITER $$
CREATE PROCEDURE inserir_limitacao(IN nomeLimitacao VARCHAR(45), IN nomeCliente VARCHAR(45))
        BEGIN
        Declare erro Bool default 0;
        declare continue handler for SQLEXCEPTION set erro = 1;
        start transaction;
        set @A:=0;
        SELECT @A:= L.Id_Limitacao FROM Limitacao_fisica AS L where L.Nome = nomeLimitacao;
        IF(@A = 0) THEN
        INSERT INTO Limitacao_Fisica(Nome)
                VALUE (nomeLimitacao);
        SELECT @A:= L.Id_Limitacao FROM Limitacao_fisica AS L where L.Nome = nomeLimitacao;
        END IF;
        SELECT @B:= C.Id_cliente FROM Cliente AS C where C.Nome = nomeCliente;
        INSERT INTO Cliente_Limitacao_Fisica(Id_cliente, Id_Limitacao)
                VALUE (@B,@A);
            if erro then ROLLBACK;
    else commit;

    END IF;

END $$
DELIMITER ;
```

Figura 29 - Inserir limitações do cliente

- **Inserir professor**

  Para inserir um novo professor no ginásio deverá se fazer um novo registo na tabela *Professor*. Caso a localidade indicada ainda não esteja registada, será então efetuado um novo registo na tabela *Localidade*.

```
DROP PROCEDURE IF EXISTS inserir_professor;

DELIMITER $$
CREATE PROCEDURE inserir_professor(IN nome VARCHAR(45), IN datanascimento DATE, IN Endereco VARCHAR(45),
        IN telemovel CHAR(9), IN email VARCHAR(45), IN localidade VARCHAR(45))

        BEGIN
        Declare erro Bool default 0;
        declare continue handler for SQLEXCEPTION set erro = 1;
        start transaction;
        set @A:=0;
        SELECT @A:= L.Id_localidade FROM Localidade AS L where L.Nome = localidade;
        IF(@A = 0) THEN
        INSERT INTO Localidade(Nome) VALUE (localidade);
        SELECT @A:= L.Id_localidade FROM Localidade AS L where L.Nome = localidade;
        END IF;
        INSERT INTO Professor(Nome, Endereco, Data_nascimento, Telemovel, Email, Estado, Id_localidade)
                VALUE (nome, endereco, datanascimento, telemovel, email, 'Ativo', @A);
            if erro then ROLLBACK;
    else commit;

    END IF;

END $$
DELIMITER ;
```

Figura 30 - Transação inserir Professor

- **Inserir Plano**

  Para inserir um plano começa-se por efetuar o registo do mesmo na tabela *Plano*.

```
DROP PROCEDURE IF EXISTS inserir_plano;

DELIMITER $$
CREATE PROCEDURE inserir_plano(IN preco DOUBLE, IN datainicio DATE, IN nomeprof VARCHAR(45),
                                IN nome VARCHAR(45))
        BEGIN
        Declare erro Bool default 0;
        declare continue handler for SQLEXCEPTION set erro = 1;
        start transaction;
        set @A:=0;
        SELECT @A:= P.Id_professor FROM Professor AS P where P.Nome = nomeprof;
        set @B:=0;
        set @E:=0;
        SELECT @B:= C.Id_cliente, @E:= P.Id_plano FROM Cliente AS C
        INNER JOIN Plano AS P ON P.Id_cliente = C.Id_cliente
        where C.Nome = nome AND P.Estado = 'Ativo';
        INSERT INTO Plano(Preco, Data_inicio, Estado, Id_professor, Id_cliente)
                VALUE (preco, datainicio, 'Ativo', @A, @B);
            if erro then ROLLBACK;
    else commit;

    END IF;

END $$
DELIMITER ;
```

Figura 31 - Inserir plano

De seguida, deverá se registar as atividades fitness que esse plano irá ter, fazendo um novo registo na tabela *Plano_Atividade_Fitness*. É de destacar que apenas será possível inserir no plano uma atividade que ainda não tenha atingido o número máximo de participantes.

```
DROP PROCEDURE IF EXISTS inserir_atividade_plano;

DELIMITER $$
CREATE PROCEDURE inserir_atividade_plano(IN numAulas INT, IN id_plano INT, IN nome VARCHAR(45))
        BEGIN
        Declare erro Bool default 0;
        declare continue handler for SQLEXCEPTION set erro = 1;
        start transaction;
        set @A:=0;
        set @B:=0;
        set @C:=0;
        SELECT @A:= AF.Id_atividade, @B:= AF.Max_participantes, @C:= AF.Nr_inscritos FROM Atividade_Fitness AS AF where AF.Nome = nome;
        IF(@B > @C) THEN
            INSERT INTO Plano_Atividade_Fitness(Nr_aulas, Id_plano, Id_atividade)
                VALUE (numAulas, id_plano, @A);
        END IF;
            if erro then ROLLBACK;
            else commit;
    END IF;

END $$
DELIMITER ;
```

Figura 32 - Inserir atividade do plano

A elaboração do plano conduz a que o número de inscritos nas atividades contidas neste aumente. Assim, tornou-se imperativo a criação de um *trigger* que será executado aquando da inserção de cada atividade no plano, ou seja, na criação de um novo registo na tabela *Plano_Atividade_Fitness.*

```
DROP trigger IF EXISTS update_participantesIncrementa

DELIMITER $$
CREATE TRIGGER update_participantesIncrementa
AFTER INSERT ON plano_atividade_fitness
FOR EACH ROW
BEGIN
    UPDATE Atividade_fitness
    SET Nr_inscritos = Nr_inscritos + 1
    WHERE Id_atividade = @A;
END
$$
DELIMITER ;
```

Figura 33 - Atualizar número de inscritos

- **Arquivar Plano**

  Quando um plano deixa de estar em execução será necessário arquivar o mesmo, para tal será atualizado o Estado para Inativo na tabela *Plano*.

```
DELIMITER $$
CREATE PROCEDURE arquivar_plano(IN id INT)
        BEGIN
        Declare erro Bool default 0;
        declare continue handler for SQLEXCEPTION set erro = 1;
        start transaction;
        set @K = id;
        UPDATE Plano
        SET Plano.Estado = 'Inativo'
        Where Plano.Id_plano = id AND Plano.Estado = 'Ativo';
            if erro then ROLLBACK;
    else commit;

    END IF;

END $$
DELIMITER ;
```

Figura 34 - Arquivar plano

Dado que o cliente com esse plano irá deixar de realizar as aulas associadas ao plano será necessário decrementar o número de participantes nas respetivas atividades. Assim, tornou-se necessário a criação de um *trigger*.

```
DROP trigger IF EXISTS update_participantesDecrementa;

DELIMITER $$
CREATE TRIGGER update_participantesDecrementa
AFTER UPDATE ON Plano
for each row
BEGIN
     CALL decrementaInscritos(@K);
END
$$
DELIMITER ;
```

Figura 35 - Atualizar o número de inscritos

```
DROP PROCEDURE IF EXISTS decrementaInscritos;

DELIMITER $$
CREATE PROCEDURE decrementaInscritos (id_plano INT)
BEGIN
    declare contador INT;
    SET @I := 1;
    while(@I <= (SELECT MAX(Id_atividade) FROM Plano_Atividade_Fitness)) DO
        SELECT count(PA.Id_atividade) into contador FROM Plano_Atividade_Fitness AS PA
            where PA.Id_plano = id_plano AND PA.Id_atividade = @I;
        IF (contador>0) THEN
            UPDATE Atividade_Fitness
            SET Nr_inscritos = Nr_inscritos - 1
            WHERE Id_atividade =@I ;
        END IF;
        SET @I := @I +1;
        END WHILE;
END
$$
DELIMITER ;
```

Figura 36 – Decrementar o número de inscritos

- **Arquivar Professor**

Quando um professor deixar de exercer funções no ginásio será necessário alterar o seu *Estado* para *Inativo* na tabela *Professor*.

```
DROP PROCEDURE arquivar_professor;

DELIMITER $$
CREATE PROCEDURE arquivar_professor(IN nome VARCHAR(45))
        BEGIN
        Declare erro Bool default 0;
        declare continue handler for SQLEXCEPTION set erro = 1;
        start transaction;
        set @H:=0;
        SELECT @H:= P.Id_professor FROM Professor AS P where P.Nome = nome;
        UPDATE Professor
        SET Professor.Estado = 'Inativo'
        Where Professor.Id_professor = @H;
            if erro then ROLLBACK;
    else commit;

    END IF;

END $$
DELIMITER ;
```

Figura 37 - Arquivar um professor

- **Alterar Quantidade Máquinas**

Uma vez que o Sr. João pretendia poder alterar o stock de cada tipo de máquina no ginásio, tornou-se necessário dar a possibilidade de alterar a *Quantidade* na tabela *Maquina*.

```
DROP PROCEDURE IF EXISTS aumentarQt_maquina;

DELIMITER $$
CREATE PROCEDURE aumentarQt_maquina(IN tipo VARCHAR(45), IN qt INT)
        BEGIN
        SELECT @F:= Maquina.Id_maquina FROM Maquina WHERE Maquina.Tipo = tipo;
        UPDATE Maquina
        SET Maquina.Quantidade = Maquina.Quantidade + qt
        WHERE Maquina.Id_maquina = @F;

END $$
DELIMITER ;
```

Figura 38 - Aumentar quantidade de máquinas no ginásio

```
DROP PROCEDURE IF EXISTS diminuirQt_maquina;

DELIMITER $$
CREATE PROCEDURE diminuirQt_maquina(IN tipo VARCHAR(45), IN qt INT)
        BEGIN
        SELECT @F:= Maquina.Id_maquina FROM Maquina WHERE Maquina.Tipo = tipo;
        UPDATE Maquina
        SET Maquina.Quantidade = Maquina.Quantidade - qt
        WHERE Maquina.Id_maquina = @F;

END $$
DELIMITER ;
```

Figura 39 - Diminuir o número de máquinas no ginásio

- **Alterar Quantidade Máquinas**

  Poderá haver a necessidade de alterar o número de um determinado tipo de máquina necessária para uma determinada atividade.

```
DROP PROCEDURE IF EXISTS alterarQt_maquinaAtividade;

DELIMITER $$
CREATE PROCEDURE alterarQt_maquinaAtividade(IN atividade VARCHAR(45), IN maquina VARCHAR(45), IN qt INT)
    BEGIN
    SELECT @G:= Atividade_Fitness.Id_atividade, @J:= Maquina.Id_maquina FROM Atividade_Fitness
    INNER JOIN Atividade_Fitness_Maquina ON Atividade_Fitness_Maquina.Id_atividade=Atividade_Fitness.Id_atividade
    INNER JOIN Maquina ON Maquina.Id_maquina = Atividade_Fitness_Maquina.Id_maquina
    WHERE Atividade_Fitness.Nome = atividade AND  Maquina.Tipo = maquina;
    UPDATE Atividade_Fitness_Maquina
    SET Atividade_Fitness_Maquina.Nr_maquinas = qt
    WHERE Atividade_Fitness_Maquina.Id_atividade = @G
    AND Atividade_Fitness_Maquina.Id_maquina = @J;

END $$
DELIMITER ;
```

Figura 40 – Alterar o número de máquinas utilizadas numa atividade

5.5 Escolha, definição e caracterização de índices em SQL

No nosso sistema de bases de dados optamos por não implementar índices, dado que consideramos que este não dispõe de uma dimensão e complexidade significativa. Além disso na fase em que o projeto se encontra não existe uma quantidade de dados considerável.

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

Ao elaborar a base de dados tornou-se imperativo ter em atenção o espaço que ocupará em disco. Com o intuito de prever o espaço ocupado em disco, começamos por identificar o tamanho de cada tipo de dados que utilizamos. De seguida, para cada tabela do Modelo Lógico verificamos o espaço ocupado por esta.

Tipo de dados	Tamanho (bytes)
INT	4
VARCHAR(N)	N+1
CHAR(N)	N
DATE	3
TIME	3
DOUBLE	8

Tabela 9 - Espaço ocupado no disco por cada tipo de dados

	Atributos	Tipo de dados	Espaço no disco
Cliente	Id_cliente	INT	4
	Nome	VARCHAR(45)	46
	Data_nascimento	DATE	3
	Endereço	VARCHAR(45)	46
	Id_localidade	INT	4
Total			103

Tabela 10 - Espaço ocupado no disco por Cliente

	Atributo	Tipo de dados	Espaço no disco
Telemovel	Id_telemovel	INT	4
	Tipo	VARCHAR(45)	46
	Numero	CHAR(9)	9

	Id_cliente	INT	4
Total			63

Tabela 11 - Espaço ocupado no disco por Telemovel

	Atributo	**Tipo de dados**	**Espaço no disco**
Cliente_Limitacao_Fisica	Id_cliente	INT	4
	Id_Limitacao	INT	4
Total			8

Tabela 12 - Espaço ocupado no disco por Cliente_Limitacao_Fisica

	Atributo	**Tipo de dados**	**Espaço no disco**
Limitacao_Fisica	Id_Limitacao	INT	4
	Nome	VARCHAR(45)	46
Total			50

Tabela 13 - Espaço ocupado no disco por cada limitação de um cliente

	Atributo	**Tipo de dados**	**Espaço no disco**
Localidade	Id_localidade	INT	4
	Nome	VARCHAR(45)	46
Total			50

Tabela 14 - Espaço ocupado no disco por localidade

	Atributo	**Tipo de dados**	**Espaço no disco**
Plano	Id_plano	INT	4
	Estado	VARCHAR(7)	8
	Preco	DOUBLE	8
	Data_inicio	DATE	3
	Id_professor	INT	4
	Id_cliente	INT	4
Total			31

Tabela 15 - Espaço ocupado no disco por Plano

	Atributo	**Tipo de dados**	**Espaço no disco**
Professor	Id_professor	INT	4
	Nome	VARCHAR(45)	46
	Endereço	VARCHAR(45)	46

	Id_localidade	INT	4
	Data_nascimento	DATE	3
	Telemovel	CHAR(9)	9
	Email	VARCHAR(45)	46
	Estado	VARCHAR(7)	8
Total			166

Tabela 16 - Espaço ocupado no disco por Professor

	Atributo	**Tipo de dados**	**Espaço no disco**
Plano_Atividade_Fitness	Nr_aulas	INT	4
	Id_plano	INT	4
	Id_atividade	INT	4
Total			12

Tabela 17 - Espaço ocupado no disco pelo relacionamento entre a Atividade e o Plano

	Atributo	**Tipo de dados**	**Espaço no disco**
Atividade_Fitness	Id_atividade	INT	4
	Max_participantes	INT	4
	Nr_Inscritos	INT	4
	Nome	VARCHAR(45)	46
	Duracao	TIME	3
	Sala	INT	4
	Id_professor	INT	4
Total			69

Tabela 18 - Espaço ocupado no disco por uma Atividade Fitness

	Atributo	**Tipo de dados**	**Espaço no disco**
Atividade_Fitness_Maquina	Nr_maquinas	INT	4
	Id_atividade	INT	4
	Id_maquina	INT	4
Total			12

Tabela 19 - Espaço ocupado no disco pelo relacionamento entre a Atividade e a Máquina

	Atributo	**Tipo de dados**	**Espaço no disco**
Maquina	Id_maquina	INT	4

	Tipo	VARCHAR(45)	46
	Quantidade	INT	4
Total			54

Tabela 20 - Espaço ocupado no disco por cada Máquina

Considerando a dimensão da população presente na nossa base de dados, em seguida será ilustrado o espaço ocupado no disco por esta.

Tabela	**Espaço no disco**
Cliente	103*50 = 5150
Telemovel	63*93 = 5859
Cliente_Limitacao_Fisica	8*11 = 88
Limitacao_Fisica	50*8 = 400
Localidade	50*4 = 200
Plano	31*93 = 2883
Professor	166*9 = 1494
Plano_atividade_Fitness	12*176 = 2112
Atividade_Fitness	69*10 = 690
Atividade_Fitness_Maquina	12*10 = 120
Maquina	54*10 = 540
Total	19536

Tabela 21 - Espaço ocupado em disco pela população atual

Como podemos observar na tabela acima, a estimativa para o tamanho da nossa base de dados com o povoamento final é de 19536 bytes. No entanto, é fundamental verificar como se comportará com uma quantidade de dados significante, ou seja, com uma escala real. Atendendo ao crescimento do ginásio nos últimos meses é expectável que dentro de algum tempo estejam inscritos cerca de 100 alunos conduzindo também para uma maior diversidade de limitações físicas e localidades. Assim passarão a existir 20 limitações físicas, 10 localidades e 250 telefones. De modo a não sobrecarregar tanto os professores, passaram a trabalhar no ginásio 10. Com o aumento do número de clientes e a conclusão de planos dos clientes que já frequentaram o ginásio existirão na base de dados 400 planos. Relativamente às relações *Plano_Atividade_Fitness* e *Cliente_Limitacao_Fisica* espera-se que existam 1500 e 40, respetivamente.

Tabela	**Espaço no disco**
Cliente	103*100 = 10300
Telemovel	63*250 = 15750
Cliente_Limitacao_Fisica	8*40 = 320

Limitacao_Fisica	50*20 = 1000
Localidade	50*10 = 500
Plano	31*400 = 1240
Professor	166*10 = 1660
Plano_atividade_Fitness	12*1500 = 18000
Atividade_Fitness	69*10 = 690
Atividade_Fitness_Maquina	12*10 = 120
Maquina	54*10 = 540
Total	50120

Tabela 22 - Espaço ocupado em disco pela população no futuro

Assim sendo, o espaço ocupado em disco será de 50120 bytes.

5.7 Definição e caracterização das vistas de utilização em SQL

Nesta secção, são apresentadas as views que consideramos fundamentais para os utilizadores da nossa base de dados.

Na Figura 46, encontra-se a view das limitações físicas de cada aluno que pode ser acedida pelos professores e pelo Sr. João.

```
DROP VIEW IF EXISTS view_limitacao_fisica;

DELIMITER $$
CREATE VIEW view_limitacao_fisica
AS
    SELECT C.Nome, L.Nome AS Limitação FROM Cliente AS C
    INNER JOIN Cliente_Limitacao_Fisica AS CL ON CL.Id_cliente = C.Id_cliente
    INNER JOIN Limitacao_Fisica AS L ON L.Id_limitacao = CL.Id_limitacao
$$
DELIMITER ;
```

Figura 41 - View limitações físicas

Em seguida, podemos observar a *view* com as informações das máquinas utilizadas em cada atividade, bem como a quantidade utilizada e disponível, que se encontra disponível para os professores e para o Sr. João.

```
DROP VIEW IF EXISTS view_maquinas;

DELIMITER $$
CREATE VIEW view_maquinas
AS
    SELECT A.Nome AS Atividade, M.Tipo, AM.Nr_maquinas AS QtUtilizada,
        M.Quantidade AS QtDisponivel FROM Maquina AS M
    INNER JOIN Atividade_Fitness_Maquina AS AM ON AM.Id_maquina = M.Id_maquina
    INNER JOIN Atividade_Fitness AS A ON A.Id_atividade = AM.Id_atividade;
$$
DELIMITER ;
```

Figura 42 - View máquinas

Na seguinte figura, encontra-se a view com os contactos de todos os clientes que frequentam ou já frequentaram o ginásio, visíveis para os professores e para o Sr. João.

```
DROP VIEW IF EXISTS view_alunos;

DELIMITER $$
CREATE VIEW view_alunos
AS
    SELECT C.Nome, T.Numero,T.Tipo AS TipoContacto, C.Data_nascimento AS DataNascimento,
    L.Nome AS Localidade, C.Endereco FROM Cliente AS C
    INNER JOIN Telemovel AS T ON T.Id_cliente = C.Id_cliente
    INNER JOIN Localidade AS L ON L.Id_localidade = C.Id_localidade
$$
DELIMITER ;
```

Figura 43 - View alunos

Na Figura 49, pode-se observar a view com todas as informações referentes às atividades fitness acessível para todos os utilizadores da aplicação.

```
DROP VIEW IF EXISTS view_atividades;

DELIMITER $$
CREATE VIEW view_atividades
AS
    SELECT A.Nr_inscritos AS NumeroParticipantes, A.Max_participantes AS MaxParticipantes, A.Nome AS Atividade,
    A.Duracao, A.Sala FROM Atividade_Fitness AS A
    INNER JOIN Professor AS P ON P.Id_professor = A.Id_professor
    ORDER BY Atividade
$$
DELIMITER ;
```

Figura 44 - View atividades

De seguida, podemos observar todos os planos elaborados no ginásio que estará disponível para os professores e para o Sr. João.

```
DROP VIEW IF EXISTS view_plano;

DELIMITER $$
CREATE VIEW view_plano
AS
    SELECT P.Id_plano AS ID, P.Preco, P.Data_inicio AS Inicio, P.Estado,
    PR.Nome AS Professor, C.Nome AS Cliente FROM Plano AS P
    INNER JOIN Professor AS PR ON PR.Id_professor = P.Id_professor
    INNER JOIN Cliente AS C ON C.Id_cliente = P.Id_cliente
    ORDER BY ID ASC
$$
DELIMITER ;
```

Figura 45 - View plano

Na seguinte view é possível saber quais os professores que lecionam cada atividade fitness, que será acessível pelo Sr. João.

```
DROP VIEW IF EXISTS view_atividade_professor;

DELIMITER $$
CREATE VIEW view_atividade_professor
AS
    SELECT A.Nome AS Atividade, P.Nome AS Professor FROM Atividade_Fitness AS A
    INNER JOIN Professor AS P ON P.Id_professor = A.Id_professor
    ORDER BY Atividade
$$
DELIMITER ;
```

Figura 46 - View professores por atividade

5.8 Definição e caracterização dos mecanismos de segurança em SQL

Atendendo aos requisitos de controlo apresentados na secção 2, podemos identificar três tipos de utilizadores. O Sr. João, que é o proprietário do “Ginásio 100calorias” e como tal pode consultar, inserir e alterar os dados dos clientes, professores, máquinas, planos e atividades. Existem os professores, que têm a possibilidade de consultar as informações relativas aos alunos, às atividades e aos planos e de alterar e inserir planos. Os alunos podemos consultar as informações das atividades.

```
USE ginasio;
CREATE USER 'SrJoao'@'localhost' IDENTIFIED BY 'Joao1';
CREATE USER 'Professor'@'localhost' IDENTIFIED BY 'Professor2';
CREATE USER 'Cliente'@'localhost' IDENTIFIED BY 'Cliente3';

GRANT SELECT, INSERT, UPDATE ON * TO 'SrJoao'@'localhost';
GRANT SELECT ON Cliente TO 'Professor'@'localhost';
GRANT SELECT ON Limitacao_Fisica TO 'Professor'@'localhost';
GRANT SELECT ON Maquina To 'Professor'@'localhost';
GRANT SELECT ON Atividade_Fitness TO 'Professor'@'localhost';
GRANT SELECT, INSERT, UPDATE ON Plano TO 'Professor'@'localhost';
GRANT SELECT ON Atividade_Fitness TO 'Cliente'@'localhost';

GRANT EXECUTE ON PROCEDURE clientes_localidade TO 'SrJoao'@'localhost';
GRANT EXECUTE ON PROCEDURE inserir_professor TO 'SrJoao'@'localhost';
GRANT EXECUTE ON PROCEDURE arquivar_professor TO 'SrJoao'@'localhost';
GRANT EXECUTE ON PROCEDURE inserir_cliente TO 'SrJoao'@'localhost';
GRANT EXECUTE ON PROCEDURE alterarQt_maquinaAtividade TO 'SrJoao'@'localhost';
GRANT EXECUTE ON PROCEDURE diminuirQt_maquina TO 'Professor'@'localhost';
GRANT EXECUTE ON PROCEDURE aumentarQt_maquina TO 'SrJoao'@'localhost';
GRANT EXECUTE ON PROCEDURE arquivar_plano TO 'SrJoao'@'localhost';
GRANT EXECUTE ON PROCEDURE inserir_telemovel TO 'SrJoao'@'localhost';
GRANT EXECUTE ON PROCEDURE inserir_limitacao TO 'SrJoao'@'localhost';
GRANT EXECUTE ON PROCEDURE inserir_atividade_plano TO 'SrJoao'@'localhost';
GRANT EXECUTE ON PROCEDURE inserir_plano TO 'SrJoao'@'localhost';
GRANT EXECUTE ON PROCEDURE arquivar_plano TO 'SrJoao'@'localhost';

GRANT EXECUTE ON PROCEDURE inserir_atividade_plano TO 'Professor'@'localhost';
GRANT EXECUTE ON PROCEDURE inserir_plano TO 'Professor'@'localhost';
GRANT EXECUTE ON PROCEDURE arquivar_plano TO 'Professor'@'localhost';


GRANT SELECT ON view_maquinas TO 'SrJoao' @'localhost';
GRANT SELECT ON view_maquinas TO 'Professor' @'localhost';
GRANT SELECT ON view_limitacao_fisica TO 'SrJoao' @'localhost';
GRANT SELECT ON view_limitacao_fisica TO 'Professor' @'localhost';
GRANT SELECT ON view_alunos TO 'SrJoao' @'localhost';
GRANT SELECT ON view_alunos TO 'Professor' @'localhost';
GRANT SELECT ON view_atividades TO 'SrJoao' @'localhost';
GRANT SELECT ON view_atividades TO 'Professor' @'localhost';
GRANT SELECT ON view_atividades TO 'Cliente' @'localhost';
GRANT SELECT ON view_plano TO 'SrJoao' @'localhost';
GRANT SELECT ON view_plano TO 'Professor' @'localhost';
GRANT SELECT ON view_atividade_professor TO 'SrJoao' @'localhost';


FLUSH privileges;
```

Figura 47 – Permissões aos diferentes utilizadores

5.9 Revisão do sistema implementado com o utilizador

Após a realização de todos os objetivos inicialmente estabelecidos, reunimos novamente com o Sr. João para mostrar todo o trabalho realizado de modo a recebermos feedback. Junto deste foram analisados todos os requisitos estabelecidos e foi revisto novamente todas as fases do trabalho já realizado, desde a elaboração do Modelo Conceptual até ao Modelo Físico,

6 Projeto de um Sistema de Base de Dados não Relacional

6.1 Utilização de um sistema NoSQL

Os sistemas de base de dados relacionais assumem uma posição de predominância no mercado. Tal deve-se principalmente à eficácia do armazenamento dos dados estruturados. É importante referir que este tipo de sistemas preservam a consistência, integridade, isolamento dos dados e durabilidade. Contudo, surgiram problemas que conduziram ao aparecimento de sistemas de base de dados não relacionais (NoSQL) que foram idealizados atendendo às lacunas que as base de dados tradicionais demonstraram, como alta performance e capacidade de expansão. Estas em vez de armazenarem os dados em tabelas utilizam estruturas dependendo do tipo da base de dados. Existem quatro tipos que são chave-valor, grafos, colunas e documentos.

Dado que o negócio do Sr. João tem vindo a crescer exponencialmente, o sistema anteriormente implementado tem vindo a tornar-se mais lento pelo que em reunião com o mesmo ficou decido alterar o sistema de base de dados para um não relacional.

6.1.1 Neo4j

Esta é uma base de dados NoSQL baseada em grafos, constituída por três componentes básicas, sendo estas os nodos, os relacionamentos entre estes e as suas propriedades.

As principais vantagens deste tipo de base de dados são: performance, flexibilidade e agilidade. Com o aumento dos relacionamentos e da quantidade de dados a processar as consultas na base de dados relacional tornam-se bastante difíceis dado que o sistema se torna lento. Tal não acontece numa base de dados baseada em grafos, visto que o tempo permanece constante com o aumento do volume de dados. Este tipo de base de dados é bastante flexível uma vez que permite ajustar a estrutura e o esquema de um modelo às alterações verificadas. As bases de dados NoSQL possibilitam um desenvolvimento correto e sistemas de manutenção acessível.

Atendendo às vantagens apresentadas anteriormente, o Sr. João concordou em implementar uma base de dados não relacional baseada em grafos, nomeadamente o Neo4j.

6.2 Objetivos da Base de Dados

Ao longo do tempo o “Ginásio 100calorias” foi ganhando popularidade, devido aos excelentes resultados obtidos pelos seus clientes e pelo atendimento personalizado prestado a estes. Assim, deu-se um crescimento exponencial pelo que a base de dados implementada se tornou inadequada, uma vez que as interrogações demoravam bastante tempo a retornar o resultado.

Deste modo, fomos contactados pelo Sr. João para tentarmos resolver o problema em questão. A solução encontrada consistiu em migrar a antiga base de dados para um sistema de base de dados não relacional.

6.3 Identificação das Queries realizadas sobre o sistema

Com o sistema Neo4j será dada resposta às mesmas questões previamente elaboradas. Assim sendo, de seguida apresentamos as queries realizadas:

- Obter o número total de clientes do ginásio
- Verificar o número de professores que lecionam no ginásio
- Saber que professor leciona certa atividade física
- Conhecer a sala onde a atividade é lecionada
- Consultar os planos realizados por dado cliente
- Identificar o top 3 de clientes com maior número de planos associados
- Reconhecer o professor que leciona mais atividades
- Obter o top 3 das atividades com mais alunos inscritos
- Identificar o número de planos elaborados por cada professor
- Verificar o top 3 dos alunos com maior número de aulas de uma determinada atividade fitness
- Atividade fitness mais frequentada por um determinado aluno
- Devolver os planos elaborados por um professor ordenados pelo preço, bem como o cliente ao qual o plano foi atribuído
- Ver quais os planos que já foram realizados por um dado cliente com determinado estado (ativo ou inativo)
- As atividades fitness lecionadas por um professor
- Nomes e contactos dos alunos que frequentam certa atividade
- Obter os alunos de uma localidade

6.4 Estrutura base para o sistema de dados NoSQL

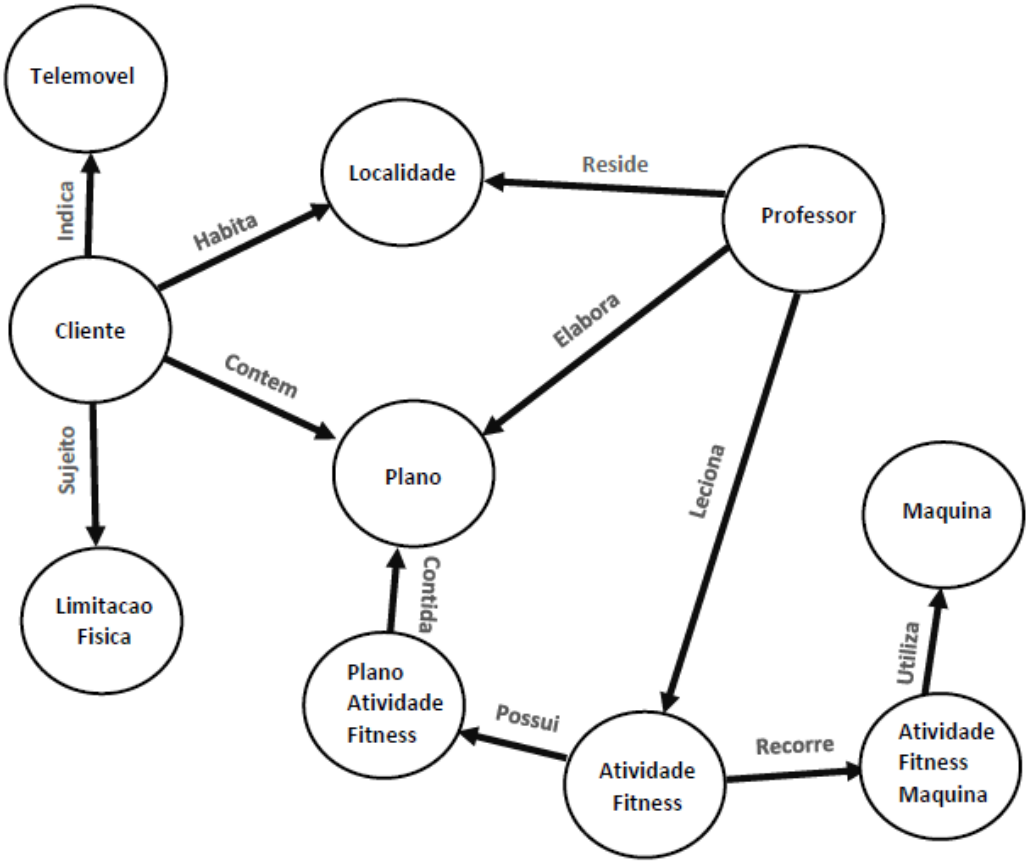


Figura 48 - Estrutura adotada para o sistema NoSQL

6.4.1 Objetos de dados no sistema NoSQL

De modo a satisfazer o novo sistema de dados, foi criada a estrutura acima apresentada, em que, a partir da Base de Dados não relacional, cada nodo retratado representa as tabelas existentes no modelo lógico do MySQL. Assim, é representado um grafo, em que as ligações entre nodos correspondem aos relacionamentos existentes entre as entidades. Esta estrutura adotada será o suporte para a nossa base de dados no Neo4j, servindo esta de esboço para simplificar a compreensão do conteúdo existente.

6.5 Migração de Dados

6.5.1 Exportação dos Dados

Para efetuar o processo de migração de uma base de dados relacional para uma não relacional, é necessária a elaboração de um script (encontrado em anexo). Este foi

implementado na linguagem Java, com o intuito de extrair os dados de cada tabela para um ficheiro .csv, em que as informações extraídas são armazenadas em ficheiros distintos.

Tendo estes ficheiros criados, prosseguimos para a importação dos mesmos no sistema de dados Neo4j, para a possível criação de nodos e relacionamentos entre eles, como será apresentado nas secções seguintes.

6.5.2 Criação dos Nodos

Após a exportação dos dados e consequente importação dos mesmos para o Neo4j, avançamos para a criação dos nodos com toda a informação presente nos respetivos ficheiros .csv. Assim é elaborado um conjunto de operações que são guardadas num ficheiro .cql (Cypher Query Language), gerando os nodos pretendidos. É importante realçar que na criação de alguns nodos tornou-se imperativo criar Unique Constraints. Assim, garantimos que determinados valores não se repetem para uma determinada propriedade, por exemplo não pode existir duas localidades com o mesmo identificador.

Nodo Localidade

Este nodo irá conter o id da localidade e o seu nome.

```
CREATE CONSTRAINT ON (l:Localidade) ASSERT l.IdLocalidade IS UNIQUE;
USING PERIODIC COMMIT
LOAD CSV WITH HEADERS FROM "file:///localidade.csv" AS line
CREATE (l:Localidade {IdLocalidade: TOINTEGER(line.IdLocalidade),
Nome: line.Nome});
```

Figura 49 - Criação do Nodo Localidade

Nodo Cliente

Este nodo irá conter o seu id, o seu nome, data de nascimento, endereço e ainda o id da localidade a que pertence.

```
CREATE CONSTRAINT ON (c:Cliente) ASSERT c.IdCliente IS UNIQUE;
USING PERIODIC COMMIT
LOAD CSV WITH HEADERS FROM "file:///cliente.csv" AS line
CREATE (c:Cliente { IdCliente: TOINTEGER(line.IdCliente),
Nome: (line.Nome),
DataNascimento: (line.DataNascimento),
Endereco: (line.Endereco),
IdLocalidade: TOINTEGER(line.IdLocalidade)});
```

Figura 50 - Criação do Nodo Cliente

Nodo Telemovel

Este nodo irá conter o seu id, o tipo (pessoal ou emergência), o número e ainda o id do cliente ao qual este telemóvel pertence.

```
CREATE CONSTRAINT ON (t:Telemovel) ASSERT t.IdTelemovel IS UNIQUE;
USING PERIODIC COMMIT
LOAD CSV WITH HEADERS FROM "file:///telemovel.csv" AS line
CREATE (t:Telemovel { IdTelemovel: TOINTEGER(line.IdTelemovel),
Tipo: (line.Tipo),
Numero: (line.Numero),
IdCliente: TOINTEGER(line.IdCliente)});
```

Figura 51 - Criação do Nodo Telemóvel

Nodo Limitacao_Fisica

Este nodo irá conter o seu id, bem como o seu nome.

```
CREATE CONSTRAINT ON (lf:Limitacao_Fisica) ASSERT lf.IdLimitacao IS UNIQUE;
USING PERIODIC COMMIT
LOAD CSV WITH HEADERS FROM "file:///limitacao_fisica.csv" AS line
CREATE (lf: Limitacao_Fisica { IdLimitacao: TOINTEGER(line.IdLimitacao),
Nome: (line.Nome)});
```

Figura 52 - Criação do Nodo Limitacao_Fisica

Nodo Plano

Este nodo irá conter o seu id, o seu preço, a data de início, o estado do plano e ainda o id do Cliente ao qual este plano pertence e qual o professor que o efetuou.

```
CREATE CONSTRAINT ON (pl:Plano) ASSERT pl.IdPlano IS UNIQUE;
USING PERIODIC COMMIT
LOAD CSV WITH HEADERS FROM "file:///plano.csv" AS line
CREATE (pl:Plano { IdPlano:TOINTEGER (line.IdPlano),
Preco: TOFLOAT(line.Preco),
DataInicio: (line.DataInicio),
Estado: (line.Estado),
IdCliente: TOINTEGER(line.IdCliente),
IdProfessor: TOINTEGER(line.IdProfessor)});
```

Figura 53 - Criação do Nodo Plano

Nodo Professor

Este nodo irá conter o seu id, o seu nome, endereço, a data de nascimento, telemóvel, email, o estado que indica se este se encontra a lecionar no ginásio e ainda o id da localidade onde reside.

```
CREATE CONSTRAINT ON (p:Professor) ASSERT p.IdProfessor IS UNIQUE;
USING PERIODIC COMMIT
LOAD CSV WITH HEADERS FROM "file:///professor.csv" AS line
CREATE (p:Professor { IdProfessor:TOINTEGER (line.IdProfessor),
Nome: (line.Nome),
Endereco: (line.Endereco),
DataNascimento: (line.DataNascimento),
Telemovel: (line.Telemovel),
Email:(line.Email),
Estado: (line.Estado),
IdLocalidade: TOINTEGER(line.IdLocalidade)});
```

Figura 54 - Criação do Nodo Professor

Nodo Atividade_Fitness

Este nodo irá conter o seu id, o máximo de participantes, o seu nome, duração, a sala onde é lecionada, o número de inscritos, bem como o id do professor que leciona esta atividade.

```
CREATE CONSTRAINT ON (a:Atividade_Fitness) ASSERT a.IdAtividade IS UNIQUE;
USING PERIODIC COMMIT
LOAD CSV WITH HEADERS FROM "file:///atividade_fitness.csv" AS line
CREATE (af: Atividade_Fitness { IdAtividade: TOINTEGER (line.IdAtividade),
MaxParticipantes: TOINTEGER(line.MaxParticipantes),
Nome: (line.Nome),
Duracao: (line.Duracao),
Sala: TOINTEGER(line.Sala),
NrInscritos: TOINTEGER(line.NrInscritos),
IdProfessor: TOINTEGER(line.IdProfessor)});
```

Figura 55 - Criação do Nodo Atividade_Fitness

Nodo Plano_Atividade_Fitness

Este nodo irá conter o id do plano, o id da atividade e ainda o número de aulas. A criação deste nodo deveu-se ao facto de um plano ter associado o número de aulas de cada atividade.

```
USING PERIODIC COMMIT
LOAD CSV WITH HEADERS FROM "file:///plano_atividade_fitness.csv" AS line
CREATE (pa:Plano_Atividade_Fitness { IdPlano:TOINTEGER (line.IdPlano),
IdAtividade:TOINTEGER (line.IdAtividade),
NrAulas: TOINTEGER(line.NrAulas)});
```

Figura 56 - Criação do Nodo Plano_Atividade_Fitness

Nodo Maquina

Este nodo irá conter o id, o tipo da máquina e ainda a quantidade de máquinas existentes deste tipo.

```
CREATE CONSTRAINT ON (m:Maquina) ASSERT m.IdMaquina IS UNIQUE;
USING PERIODIC COMMIT
LOAD CSV WITH HEADERS FROM "file:///maquina.csv" AS line
CREATE (m:Maquina { IdMaquina: TOINTEGER(line.IdMaquina),
Tipo: (line.Tipo),
Quantidade: TOINTEGER(line.Quantidade)});
```

Figura 57 - Criação do Nodo Maquina

Nodo Atividade_Fitness_Máquina

Este nodo irá conter o id da atividade fitness, o id da máquina e o número de máquinas. A criação deste nodo deveu-se a que cada atividade utiliza um determinado número de máquinas de cada tipo.

```
USING PERIODIC COMMIT
LOAD CSV WITH HEADERS FROM "file:///atividade_fitness_maquina.csv" AS line
CREATE (am:Atividade_Fitness_Maquina { IdAtividade:TOINTEGER (line.IdAtividade),
IdMaquina:TOINTEGER (line.IdMaquina),
NrMaquinas:TOINTEGER (line.Nr_Maquinas)});
```

Figura 58 - Criação do Nodo Atividade_Fitness_Máquina

6.5.3 Criação dos Relacionamentos

Concluída a criação dos nodos, é necessário gerar os relacionamentos existentes entre estes. Para tal, mais uma vez, é utilizada a linguagem *Cypher*, combinando um conjunto de operações que permitem a elaboração dos respetivos relacionamentos.

Assim, os relacionamentos existentes entre os nodos são:

Cliente – Localidade

Este relacionamento representa que um dado cliente “Habita” numa determinada localidade.

```
MATCH (c:Cliente),(l:Localidade)
WHERE c.IdLocalidade = l.IdLocalidade
CREATE(c)-[r:Habita]->(l);
```

Figura 59 - Criação do Relacionamento Cliente – Localidade

Cliente – Telemovel

Este relacionamento representa que um dado cliente “Indica” os números de telemóvel que possui.

```
MATCH (c:Cliente),(t:Telemovel)
WHERE c.IdCliente = t.IdCliente
CREATE(c)-[r:Indica]->(t);
```

Figura 60 - Criação do Relacionamento Cliente – Telemovel

Professor – Localidade

Bem como os clientes, os professores devem indicar onde moram. Assim, o professor “Reside” numa determinada localidade.

```
MATCH (p:Professor),(l:Localidade)
WHERE p.IdLocalidade = l.IdLocalidade
CREATE(p)-[r:Reside]->(l);
```

Figura 61 - Criação do Relacionamento Professor – Localidade

Professor – Atividade_Fitness

Os professores do ginásio lecionam várias aulas na instituição. Assim, cria-se o relacionamento em que Professor “Leciona” Atividade_Fitness.

```
MATCH (p:Professor),(a:Atividade_Fitness)
WHERE p.IdProfessor = a.IdProfessor
CREATE(p)-[r:Leciona]->(a);
```

Figura 62 - Criação do Relacionamento Professor – Atividade_Fitness

Professor – Plano

Para satisfazer as necessidades e objetivos de um cliente, um professor “Elabora” planos para o mesmo.

```
MATCH (p:Professor),(pl:Plano)
WHERE p.IdProfessor = pl.IdProfessor
CREATE(p)-[r:Elabora]->(pl);
```

Figura 63 - Criação do Relacionamento Professor – Plano

Cliente – Limitacao_Fisica

Para que a elaboração do plano seja o mais coerente possível, é necessário saber as limitações físicas de um cliente. Assim, um cliente está “Sujeito” a limitações físicas.

```
USING PERIODIC COMMIT
LOAD CSV WITH HEADERS FROM "file:///cliente_limitacao_fisica.csv" AS line
MATCH (c:Cliente {IdCliente: TOINTEGER(line.IdCliente)})
MATCH (l:Limitacao_Fisica {IdLimitacao: TOINTEGER(line.IdLimitacao)})
MERGE (c)-[r:Sujeito]->(l);
```

Figura 64 - Criação do Relacionamento Cliente – Limitacao_Fisica

Atividade_Fitness_Maquina – Maquina

Para o decorrer das aulas no ginásio, muitas requerem um suporte extra, nomeadamente de máquinas, assim Atividade_Fitness_Maquina “Utiliza” Maquina. É importante referir que a Atividade_Fitness_Maquina contém o número de máquinas de um dado tipo a utilizar numa determinada aula.

```
MATCH (am: Atividade_Fitness_Maquina),(m:Maquina)
WHERE am.IdMaquina = m.IdMaquina
CREATE(am)-[r:Utiliza]->(m);
```

Figura 65 - Criação do Relacionamento Atividade_Fitness_Maquina – Maquina

Atividade_Fitness – Atividade_Fitness_Maquina

Semelhante ao relacionamento anterior e com o mesmo intuito, uma Atividade_Fitness “Recorre” Atividade_Fitness_Maquina, dado que as máquinas são utilizadas nas atividades fitness.

```
MATCH (a: Atividade_Fitness),(am: Atividade_Fitness_Maquina)
WHERE a.IdAtividade = am.IdAtividade
CREATE(a)-[r:Recorre]->(am);
```

Figura 66 - Criação do Relacionamento Atividade_Fitness – Atividade_Fitness_Maquina

Plano_Atividade_Fitness – Plano

Dado que um plano é constituído por atividades fitness, é necessário a existência de um relacionamento entre ambos. Assim, o Plano_Atividade_Fitness está “Contida” no Plano. É importante realçar que o Plano_Atividade_Fitness contém o número de aulas de uma dada atividade num determinado plano.

```
MATCH (pa: Plano_Atividade_Fitness),(pl:Plano)
WHERE pa.IdPlano = pl.IdPlano
CREATE(pa)-[r:Contida]->(pl);
```

Figura 67 - Criação do Relacionamento Plano_Atividade_Fitness – Plano

Atividade_Fitness – Plano_Atividade_Fitness

Semelhante ao relacionamento anterior, uma Atividade_Fitness "Possui" um Plano_Atividade_Fitness, uma vez que uma atividade pode estar incluída em planos.

```
MATCH (a: Atividade_Fitness),(pa: Plano_Atividade_Fitness)
WHERE a.IdAtividade = pa.IdAtividade
CREATE(a)-[r:Possui]->(pa);
```

Figura 68 - Criação do Relacionamento Atividade_Fitness - Plano_Atividade_Fitness

Cliente – Plano

Um cliente que frequenta o ginásio realiza vários planos. Assim, cria-se o relacionamento em que o Cliente "Contem" Plano.

```
MATCH (c:Cliente),(pl:Plano)
WHERE c.IdCliente = pl.IdCliente
CREATE(c)-[r:Contem]->(pl);
```

Figura 69 - Criação do Relacionamento Cliente - Plano

6.6 Resolução das Queries Propostas

De forma a cumprir os requisitos estabelecidos efetuamos a resolução das queries previamente definidas utilizando a linguagem *Cypher*. De seguida apresentamos a resolução e explicação destas:

```
MATCH(c:Cliente)
RETURN count(*) AS NrClientes
```

Figura 70 - Obter número total de clientes do ginásio

Com esta query, é possível obter o número de clientes que frequentam o ginásio.

```
MATCH(p:Professor)
RETURN count(*) AS NrProfessores
```

Figura 71 - Verificar o número de professores que lecionam no ginásio

Nesta questão é apresentado o número de professores existentes no ginásio.

```
MATCH (p:Professor)-[r:Leciona]->(a:Atividade_Fitness{Nome:"Step"})
RETURN p.Nome AS Nome
```

Figura 72 – Saber que professor leciona certa atividade física

Esta query devolve-nos o nome do professor que leciona a atividade fitness “Step”.

```
MATCH(a:Atividade_Fitness{IdAtividade:7})
RETURN a.Sala AS Sala
```

Figura 73 - Conhecer a sala onde a atividade é lecionada

É obtida a sala onde se realiza a atividade fitness com o id 7.

```
MATCH(c:Cliente{Nome:"Ana Maria"})-[:Contem]->(p:Plano)
RETURN p.IdPlano AS Id, p.Preco AS Preco, p.DataInicio AS DataInicio, p.Estado AS
Estado
```

Figura 74 - Consultar os planos realizados por dado cliente

Através desta questão obtemos os planos associados ao cliente com o nome “Ana Maria”.

```
MATCH(c:Cliente)
WITH c, size((c)-[:Contem]->()) as NrPlanos
RETURN c.Nome AS Nome, NrPlanos
ORDER BY NrPlanos DESC
LIMIT 3
```

Figura 75 - Identificar o Top 3 de clientes com maior número de planos associados

Com esta questão é possível identificar o top 3 dos clientes com o maior número de planos.

```
MATCH(p:Professor)
WITH p, size((p)-[:Leciona]->()) as NrAtividades
RETURN p.Nome AS Nome, NrAtividades
ORDER BY NrAtividades DESC
LIMIT 1
```

Figura 76 - Reconhecer o professor que lecionou mais atividades

É indicado o professor que leciona mais atividades fitness no Ginásio ”100calorias”.

```
MATCH(a:Atividade_Fitness)
RETURN a.Nome AS Nome, a.NrInscritos AS NrInscritos
ORDER BY a.NrInscritos DESC
LIMIT 3
```

Figura 77 - Obter o Top 3 das atividades com mais alunos inscritos

São obtidas as três atividades fitness com maior número de inscritos.

```
MATCH(p:Professor)
WITH p, size((p)-[:Elabora]->()) as NrPlanos
RETURN p.Nome AS Nome, NrPlanos
ORDER BY NrPlanos DESC
```

Figura 78 - Identificar o número de planos elaborados por cada professor

Para uma melhor gestão do ginásio, com esta query é possível obter o número de planos elaborados por cada professor.

```
MATCH(a:Atividade_Fitness{Nome:"Run"})-[:Possui]->(pa:Plano_Atividade_Fitness)-
[:Contida]->(p:Plano)<-[:Contem]-(c:Cliente)
WITH c, sum(pa.NrAulas) as NrAulas
RETURN c.Nome AS Nome, NrAulas
ORDER BY NrAulas DESC
LIMIT 3
```

Figura 79 - Verificar o Top 3 dos alunos com maior número de aulas de uma determinada atividade fitness

É obtido o top 3 de clientes que frequentam mais aulas da atividade fitness “Run”.

```
MATCH(a:Atividade_Fitness)-[:Possui]->(pa:Plano_Atividade_Fitness)-[:Contida]->
(p:Plano)<-[:Contem]-(c:Cliente{Nome:"Carolina Pinto"})
WITH a, sum(pa.NrAulas) as NrAulas
RETURN a.Nome AS Nome, NrAulas
ORDER BY NrAulas DESC
LIMIT 1
```

Figura 80 - Atividade Fitness mais frequentada por determinado aluno

Nesta questão é possível determinar a atividade mais frequentada pela aluna “Carolina Pinto”.

```
MATCH(p:Professor{Nome:"Andre Gonçalves"})-[:Elabora]->(pl:Plano)<-[:Contem]-
(c:Cliente)
RETURN pl.Preco AS Preco, c.Nome AS Cliente
ORDER BY pl.Preco DESC
```

Figura 81 - Devolver os planos elaborados por um professor ordenados pelo preço

São devolvidos os planos elaborados pelo professor “Andre Gonçalves”, estando estes ordenados pelo preço respetivo.

```
MATCH(c:Cliente{Nome:"Marco Paulo"})-[:Contem]->(p:Plano{Estado:"Inativo"})
RETURN p.IdPlano AS Id, p.Preco AS Preco, p.DataInicio AS DataInicio
ORDER BY p.IdPlano
```

Figura 82 - Ver quais os planos que já foram realizados por um dado cliente

Com esta query, conseguimos determinar quais os planos que o cliente “Marco Paulo” já realizou, desde a sua entrada no Ginásio “100 calorias”.

```
MATCH(p:Professor{Nome:"Andre Gonçalves"})-[:Leciona]->(a: Atividade_Fitness)
RETURN a.Nome AS Nome
```

Figura 83 - Atividades fitness lecionada por um professor

São apresentadas todas as atividades fitness que o professor “Andre Gonçalves” leciona no ginásio.

```
MATCH(t:Telemovel)<-[:Indica]-(c:Cliente)-[:Contem]->(p:Plano{Estado:"Ativo"})<-[:Contida]-(pa:Plano_Atividade_Fitness)
<-[:Possui]-(a:Atividade_Fitness{Nome:"Cycling"})
RETURN c.Nome AS Nome, t.Numero AS Numero, t.Tipo AS Tipo
```

Figura 84 - Nomes e contactos dos alunos que frequentam certa atividade

Com a resolução desta questão, fica a ser conhecida a informação correspondente aos clientes que frequentam a atividade fitness “Cycling”, nomeadamente o nome e o contacto respetivo.

```
MATCH(c:Cliente)-[:Habita]->(l:Localidade{Nome:"Souto"})
RETURN c.Nome AS Nome
```

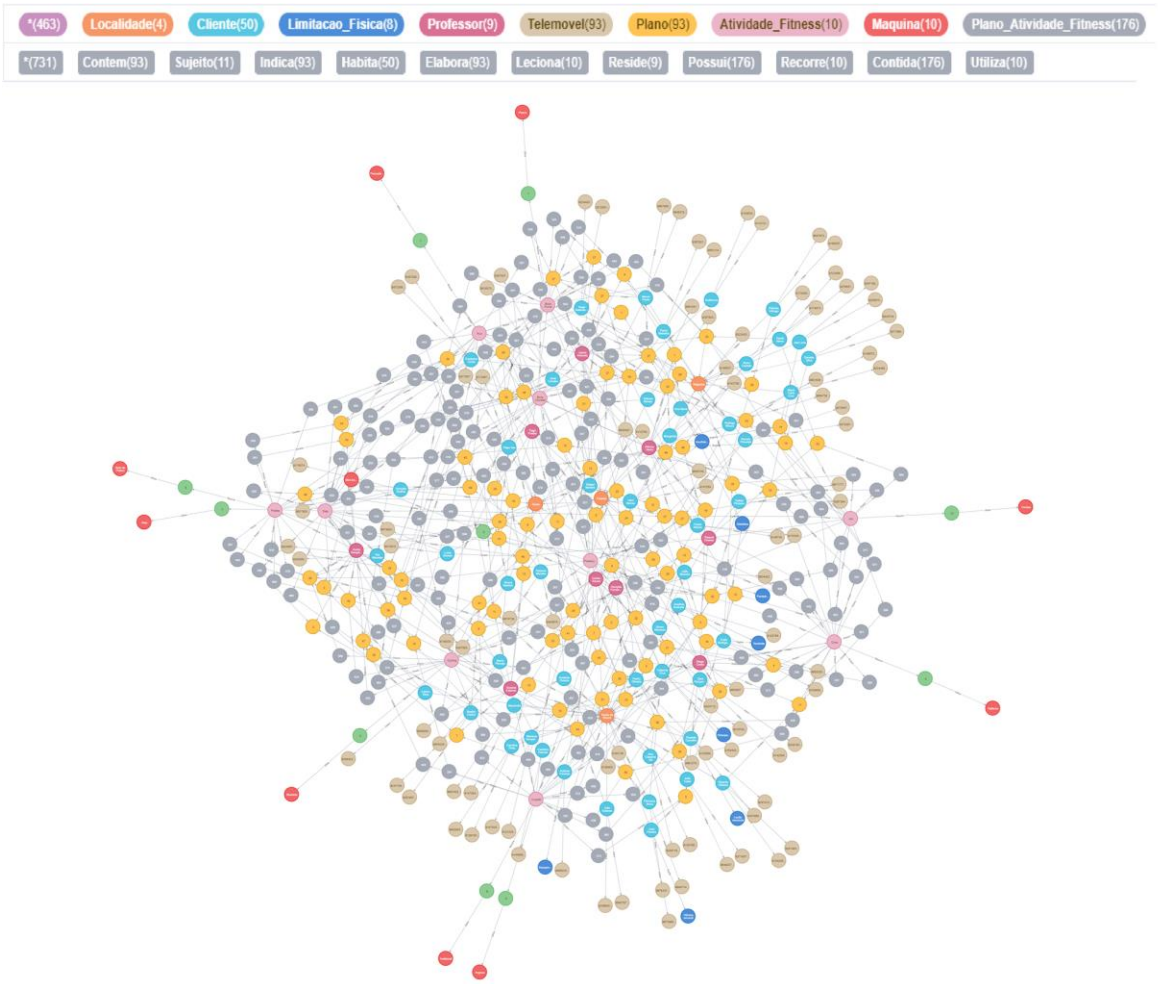
Figura 85 – Obter os alunos de uma localidade

Por fim, é possível determinar quais os clientes que habitam na localidade “Souto”.

6.7 Base de Dados em Neo4j

Uma das funcionalidades/características do Neo4j é a apresentação da base de dados através de um grafo, atribuindo a cada entidade (nodo) uma cor específica, permitindo assim uma melhor perceção dos relacionamentos existentes entre cada uma destas.

É possível observar na figura seguinte o grafo final da nossa base de dados que indica todos os tipos de nodos e as suas respetivas cores, bem como os relacionamentos existentes entre estes.


*(463)
Localidade(4)
Cliente(50)
Limitacao_Fisica(8)
Professor(9)
Telemovel(93)
Plano(93)
Atividade_Fitness(10)
Maquina(10)
Plano_Atividade_Fitness(176)
*(731)
Contem(93)
Sujeito(11)
Indica(93)
Habita(50)
Elabora(93)
Leciona(10)
Reside(9)
Possui(176)
Recorre(10)
Contida(176)
Utiliza(10)

7 Conclusões e Trabalho Futuro

Este projeto consistia na elaboração de um sistema de base de dados para o ginásio do Sr. João. Para tal, começamos por efetuar o levantamento de requisitos, seguido da construção do Modelo Conceptual, Modelo Lógico e implementação do Modelo Físico. Ao longo deste processo foram realizadas diversas reuniões com o Sr. João de modo a validar o trabalho executado e se necessário proceder a alterações.

As nossas maiores dificuldades ao nível do Modelo Conceptual surgiram numa fase mais avançada e não no momento da sua construção, uma vez que na fase de elaboração do Modelo Lógico foram detetadas diversas falhas que implicaram a alteração deste. Contudo houve ainda lacunas que não foram possíveis corrigir, mas que se deverá ter em atenção num trabalho futuro. Como a maior parte dos atributos presentes no Cliente se encontram no Professor, deveríamos recorrer à generalização e detalhe de entidades para reverter esta situação. Outro aspeto a rever seria o atributo *Endereço*, que contém a rua. Dado que esta é comum a vários clientes e professores, deveria ser separada da restante informação que é distinta.

De modo a tornar o SGBD implementado mais completo, poderíamos permitir uma gestão de máquinas mais detalhada ao Sr. João, isto é, permitir que este distribua cada máquina para uma aula e conhecer as máquinas avariadas. Para tal seria necessário tratar cada máquina individualmente, atribuindo-lhes um identificador, e não agrupadas por tipo como procedemos.

A realização deste projeto, possibilitou-nos seguir detalhadamente todos as fases de desenvolvimento de uma base de dados relacional. Durante este processo tornou-se percetível que para desenvolver um sistema de base de dados ideal, é necessário estruturar bem todos os requisitos com o cliente, visto que uma modificação nestes implicará diversas alterações no trabalho já efetuado, o que poderá ter custo elevados.

Numa segunda fase do projeto implementamos uma base de dados não relacional, nomeadamente, o Neo4j que é baseada em grafos, tendo em conta a elaborada numa primeira fase. Assim, foi necessário perceber as diferentes etapas da passagem de uma base de dados relacional para uma não relacional.

Com o auxílio da linguagem Cypher implementamos a nova base dados, criando os nodos e os respetivos relacionamentos, bem como a elaboração das queries definidas numa fase anterior do projeto.

Neste trabalho verificamos o modo como cada um dos sistemas funciona, bem como o papel que desempenham no mercado de trabalho. Os sistemas de base de dados relacionais

assumem uma posição de predominância no mercado, contudo as não relacionais encontram-se numa fase de crescimento dadas as características apresentadas em secções anteriores.

Apesar de todas as dificuldades encontradas e dos aspetos menos bem conseguidos, consideramos que cumprimos os objetivos previamente estabelecidos. Ao longo da sua elaboração foi possível desenvolver o sentido crítico para projetos futuros relativamente a qual o modelo de base de dados a implementar.

8 Referências

1. Thomas Connolly, Carolyn Begg 2015. *Database Systems A Pratical Approach to Design, Implementation and Management*. Sixth Edition / Global Edition ed. Harlow: Pearson Education Limited.

2. Feliz Gouveia 2014. *Fundamentos de Bases de Dados*, FCA.

3. Ian Robinson, Jim Webber and Emil Eifrem 2015. *Graph Databases New Opportunities for Connected Data*. Second Edition / O'Reilly Media.

9 Lista de Siglas e Acrónimos

BD Base de Dados

SGBD Sistema de Gestão de Base de Dados

SQL Structured Query Language

SBD Sistema de Base de Dados

SBDR Sistema de Base de Dados Relacional

NoSQL Not only SQL

10 Anexos

1. **Script Completo de Criação**
2. **Script Parcial do Povoamento**
3. **Queries em SQL**
4. **Exportação de Dados**

I. Anexo 1 – Script Completo de Criação

```
-- 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_ZE
RO_DATE,ERROR_FOR_DIVISION_BY_ZERO,NO_ENGINE_SUBSTITUTION';

-- -----------------------------------------------------
-- Schema ginasio
-- -----------------------------------------------------

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

-- -----------------------------------------------------
-- Table `ginasio`.`Localidade`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `ginasio`.`Localidade` (
  `Id_localidade` INT NOT NULL AUTO_INCREMENT,
  `Nome` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`Id_localidade`))
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `ginasio`.`Cliente`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `ginasio`.`Cliente` (
  `Id_cliente` INT NOT NULL AUTO_INCREMENT,
```

```
  `Nome` VARCHAR(45) NOT NULL,
  `Data_nascimento` DATE NOT NULL,
  `Endereco` VARCHAR(45) NOT NULL,
  `Id_localidade` INT NOT NULL,
  PRIMARY KEY (`Id_cliente`),
  INDEX `fk_Cliente_Localidade1_idx` (`Id_localidade` ASC),
  CONSTRAINT `fk_Cliente_Localidade1`
    FOREIGN KEY (`Id_localidade`)
    REFERENCES `ginasio`.`Localidade` (`Id_localidade`)
    ON DELETE NO ACTION
    ON UPDATE CASCADE)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `ginasio`.`Professor`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `ginasio`.`Professor` (
  `Id_professor` INT NOT NULL AUTO_INCREMENT,
  `Nome` VARCHAR(45) NOT NULL,
  `Endereco` VARCHAR(45) NOT NULL,
  `Data_nascimento` DATE NOT NULL,
  `Telemovel` CHAR(9) NOT NULL,
  `Email` VARCHAR(45) NOT NULL,
  `Estado` VARCHAR(7) NOT NULL,
  `Id_localidade` INT NOT NULL,
  PRIMARY KEY (`Id_professor`),
  INDEX `fk_Professor_Localidade1_idx` (`Id_localidade` ASC),
  CONSTRAINT `fk_Professor_Localidade1`
    FOREIGN KEY (`Id_localidade`)
    REFERENCES `ginasio`.`Localidade` (`Id_localidade`)
    ON DELETE NO ACTION
    ON UPDATE CASCADE)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `ginasio`.`Plano`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `ginasio`.`Plano` (
  `Id_plano` INT NOT NULL AUTO_INCREMENT,
```

```
  `Preco` DOUBLE NOT NULL,
  `Data_inicio` DATE NOT NULL,
  `Estado` VARCHAR(7) NOT NULL,
  `Id_cliente` INT NOT NULL,
  `Id_professor` INT NOT NULL,
  PRIMARY KEY (`Id_plano`),
  INDEX `fk_Plano_Cliente1_idx` (`Id_cliente` ASC),
  INDEX `fk_Plano_Professor1_idx` (`Id_professor` ASC),
  CONSTRAINT `fk_Plano_Cliente1`
    FOREIGN KEY (`Id_cliente`)
    REFERENCES `ginasio`.`Cliente` (`Id_cliente`)
    ON DELETE NO ACTION
    ON UPDATE CASCADE,
  CONSTRAINT `fk_Plano_Professor1`
    FOREIGN KEY (`Id_professor`)
    REFERENCES `ginasio`.`Professor` (`Id_professor`)
    ON DELETE NO ACTION
    ON UPDATE CASCADE)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `ginasio`.`Atividade_Fitness`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `ginasio`.`Atividade_Fitness` (
  `Id_atividade` INT NOT NULL AUTO_INCREMENT,
  `Max_participantes` INT NOT NULL,
  `Nome` VARCHAR(45) NOT NULL,
  `Duracao` TIME NOT NULL,
  `Sala` INT NOT NULL,
  `Nr_inscritos` INT NOT NULL,
  `Id_professor` INT NOT NULL,
  PRIMARY KEY (`Id_atividade`),
  INDEX `fk_Atividade_Fitness_Professor1_idx` (`Id_professor` ASC),
  CONSTRAINT `fk_Atividade_Fitness_Professor1`
    FOREIGN KEY (`Id_professor`)
    REFERENCES `ginasio`.`Professor` (`Id_professor`)
    ON DELETE NO ACTION
    ON UPDATE CASCADE)
ENGINE = InnoDB;
```

```
-- -----------------------------------------------------
-- Table `ginasio`.`Maquina`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `ginasio`.`Maquina` (
  `Id_maquina` INT NOT NULL AUTO_INCREMENT,
  `Tipo` VARCHAR(45) NOT NULL,
  `Quantidade` INT NOT NULL,
  PRIMARY KEY (`Id_maquina`))
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `ginasio`.`Limitacao_Fisica`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `ginasio`.`Limitacao_Fisica` (
  `Id_Limitacao` INT NOT NULL AUTO_INCREMENT,
  `Nome` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`Id_Limitacao`))
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `ginasio`.`Plano_Atividade_Fitness`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `ginasio`.`Plano_Atividade_Fitness` (
  `Nr_aulas` INT NOT NULL,
  `Id_plano` INT NOT NULL,
  `Id_atividade` INT NOT NULL,
  PRIMARY KEY (`Id_plano`, `Id_atividade`),
  INDEX          `fk_Plano_has_Atividade_Fitness_Atividade_Fitness1_idx`
(`Id_atividade` ASC),
  INDEX `fk_Plano_has_Atividade_Fitness_Plano_idx` (`Id_plano` ASC),
  CONSTRAINT `fk_Plano_has_Atividade_Fitness_Plano`
    FOREIGN KEY (`Id_plano`)
    REFERENCES `ginasio`.`Plano` (`Id_plano`)
    ON DELETE NO ACTION
    ON UPDATE CASCADE,
  CONSTRAINT `fk_Plano_has_Atividade_Fitness_Atividade_Fitness1`
    FOREIGN KEY (`Id_atividade`)
    REFERENCES `ginasio`.`Atividade_Fitness` (`Id_atividade`)
```

```
    ON DELETE NO ACTION
    ON UPDATE CASCADE)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `ginasio`.`Atividade_Fitness_Maquina`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `ginasio`.`Atividade_Fitness_Maquina` (
  `Nr_maquinas` INT NOT NULL,
  `Id_atividade` INT NOT NULL,
  `Id_maquina` INT NOT NULL,
  PRIMARY KEY (`Id_atividade`, `Id_maquina`),
  INDEX  `fk_Atividade_Fitness_has_Maquina_Maquina1_idx`  (`Id_maquina`
ASC),
  INDEX        `fk_Atividade_Fitness_has_Maquina_Atividade_Fitness1_idx`
(`Id_atividade` ASC),
  CONSTRAINT `fk_Atividade_Fitness_has_Maquina_Atividade_Fitness1`
    FOREIGN KEY (`Id_atividade`)
    REFERENCES `ginasio`.`Atividade_Fitness` (`Id_atividade`)
    ON DELETE NO ACTION
    ON UPDATE CASCADE,
  CONSTRAINT `fk_Atividade_Fitness_has_Maquina_Maquina1`
    FOREIGN KEY (`Id_maquina`)
    REFERENCES `ginasio`.`Maquina` (`Id_maquina`)
    ON DELETE NO ACTION
    ON UPDATE CASCADE)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `ginasio`.`Telemovel`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `ginasio`.`Telemovel` (
  `Id_telemovel` INT NOT NULL AUTO_INCREMENT,
  `Tipo` VARCHAR(45) NOT NULL,
  `Numero` CHAR(9) NOT NULL,
  `Id_cliente` INT NOT NULL,
  PRIMARY KEY (`Id_telemovel`, `Id_cliente`),
  INDEX `fk_Telemovel_Cliente1_idx` (`Id_cliente` ASC),
  CONSTRAINT `fk_Telemovel_Cliente1`
```

```
    FOREIGN KEY (`Id_cliente`)
    REFERENCES `ginasio`.`Cliente` (`Id_cliente`)
    ON DELETE NO ACTION
    ON UPDATE CASCADE)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `ginasio`.`Cliente_Limitacao_Fisica`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `ginasio`.`Cliente_Limitacao_Fisica` (
  `Id_cliente` INT NOT NULL AUTO_INCREMENT,
  `Id_Limitacao` INT NOT NULL,
  PRIMARY KEY (`Id_cliente`, `Id_Limitacao`),
  INDEX          `fk_Cliente_has_Limitacao_Fisica_Limitacao_Fisica1_idx`
(`Id_Limitacao` ASC),
  INDEX   `fk_Cliente_has_Limitacao_Fisica_Cliente1_idx`   (`Id_cliente`
ASC),
  CONSTRAINT `fk_Cliente_has_Limitacao_Fisica_Cliente1`
    FOREIGN KEY (`Id_cliente`)
    REFERENCES `ginasio`.`Cliente` (`Id_cliente`)
    ON DELETE NO ACTION
    ON UPDATE CASCADE,
  CONSTRAINT `fk_Cliente_has_Limitacao_Fisica_Limitacao_Fisica1`
    FOREIGN KEY (`Id_Limitacao`)
    REFERENCES `ginasio`.`Limitacao_Fisica` (`Id_Limitacao`)
    ON DELETE NO ACTION
    ON UPDATE CASCADE)
ENGINE = InnoDB;


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

II. Anexo 2 – Script Parcial do Povoamento

```
INSERT INTO Localidade
      (Nome)
    VALUES
    ('Terras de Bouro'),
    ('Fontelo'),
    ('Souto'),
    ('Nogueira');


INSERT INTO Cliente
      (Nome, Data_nascimento, Endereco, Id_localidade)
    VALUES
    ('Ana Maria', '1980-02-13', 'Rua da Capela nº12 4700-800', 2),
    ('João Moreira', '1988-04-24', 'Rua da Travessa nº41 4700-801', 2),
    ('Carlos Silva', '1990-08-30', 'Rua 25 de abril nº92 4700-770', 1),
  ('Josefina Andrade', '1999-12-12', 'Avenida Central nº15 4700-540', 3),
    ('José Pereira', '1992-01-05', 'Rua das Veigas nº26 4700-700', 1),
                                    .
                                    .
                                    .

INSERT INTO Telemovel
      (Tipo, Numero, Id_Cliente)
    VALUES
    ('Pessoal', '932343333', 1),
    ('Pessoal', '963442211', 2),
    ('Pessoal', '926843322', 3),
    ('Pessoal', '912376551', 4),
    ('Pessoal', '967389000', 5),
               .
               .
               .
```

```
INSERT INTO Limitacao_Fisica
      (Nome)
    VALUES
    ( 'Insuficiencia cardiaca'),
    ( 'Paraplegico'),
    ( 'Tendinite'),
    ( 'Hérnea cervical'),
    ( 'Rompimento da cartilagem'),
    ( 'Osteoperose'),
    ( 'Lesão meniscal'),
    ( 'Diabético');

INSERT INTO Cliente_Limitacao_Fisica
      (Id_Cliente, Id_Limitacao)
    VALUES
    ( 1, 1),
    ( 1, 8),
    ( 2, 2),
    ( 4, 3),
    ( 5, 4),
    ( 7, 5),
    ( 8, 6),
    ( 10, 7),
    ( 25, 1),
    ( 30, 3),
    ( 30, 8);

INSERT INTO Professor
      (Nome, Data_nascimento, Endereco, Telemovel, Email, Estado,
Id_localidade)
    VALUES
    ('Joana Antunes', '2000-02-13', 'Rua da Cruz nº 20 4700-880',
'916060454', 'joana@gmail.com', 'Ativo', 2),
    ('Carlos Sousa', '1970-06-13', 'Avenida da Liberdade nº67 4700-370',
'968571332', 'carlosmsousa@gmail.com', 'Ativo', 4),
    ('Andre Gonçalves', '1990-03-02', 'Rua da Laje nº68 4700-870',
'933423322', 'andreg@gmail.com', 'Ativo', 2),
                                    .
                                    .
                                    .
```

```
INSERT INTO Atividade_Fitness
      (Max_participantes,    Nome,    Duracao,    Sala,    Nr_inscritos,
Id_professor)
    VALUES
    (20, 'Run', 4500, 1, 9, 1),
    (15, 'Cycling', 4000, 2, 9, 2),
    (25, 'Step', 4500, 3, 12, 3),
    (15, 'Core', 4000, 4, 11, 4),
    (20, 'Pilates', 4500, 5, 9, 3),
    (20, 'Crossfit', 4000, 6, 11, 5),
    (25, 'Body Pump', 4500, 2, 12, 9),
    (20, 'Powerjump', 5000, 4, 11, 6),
    (25, 'Body Combat', 4000, 3, 9, 7),
    (20, 'Hiit', 4500, 5, 6, 8);

INSERT INTO Maquina
      (Tipo, Quantidade)
    VALUES
    ('Passadeira', 20),
    ('Bicicleta', 25),
    ('Step', 15),
    ('Halteres', 20),
    ('Bola de Pilates', 15),
    ('Cordas', 15),
    ('Mini-trampolim', 15),
    ('Pesos', 20),
    ('Kettlebell', 20),
    ('Argolas', 16);

INSERT INTO Atividade_Fitness_Maquina
      (Nr_maquinas, Id_atividade, Id_maquina)
    VALUES
    (15,1,1),
    (20,2,2),
    (10,3,3),
    (10,4,4),
    (10,5,5),
    (10,10,6),
    (15,8,7),
    (15,7,8),
    (15,6,9),
```

```
    (15,6,10);

INSERT INTO Plano
      (Preco, Data_inicio, Estado, Id_professor, Id_cliente)
    VALUES
      (30, '2018-01-01', 'Inativo', 1, 1),
      (40, '2018-02-01', 'Inativo', 2, 2),
      (50, '2018-03-01', 'Inativo', 3, 3),
      (40, '2018-04-01', 'Inativo', 4, 4),
      (35, '2018-10-25', 'Inativo', 3, 18),
      (52, '2017-10-28', 'Inativo', 2, 12),
      (45, '2018-07-12', 'Ativo', 2, 20),
      (37, '2017-09-06', 'Inativo', 1, 13),
      (40, '2018-08-14', 'Ativo', 4, 17),
      (45, '2018-03-12', 'Inativo', 7, 15),
                        .
                        .
                        .


INSERT INTO Plano_Atividade_Fitness
      (Nr_aulas, Id_plano, Id_atividade)
    VALUES
    (15, 1, 1),
    (10, 2, 2),
    (10, 3, 3),
    (8, 4, 4),
    (10, 8, 2),
    (7, 10, 1),
    (10, 6, 4),
    (12, 5, 7),
    (5, 6, 7),
    (15, 7, 8),
         .
         .
         .
```

III. Anexo 3 – Queries em SQL

```
-- RE 13 -> Obter o numero total de clientes do ginasio
SELECT count(C.Id_cliente) AS Nr_Clientes FROM Cliente AS C;

-- RE 14 -> Verificar o numero de professores que lecionam no ginasio
SELECT count(P.Id_professor) AS Nr_Professores FROM Professor AS P;

-- RE 15 -> Saber que professor leciona certa atividade fisica
SELECT P.Nome AS Nome_Professor FROM Professor AS P
INNER JOIN Atividade_Fitness AS A ON A.Id_professor = P.Id_Professor
WHERE A.nome = 'Step';

-- RE 16 -> Conhecer a sala onde a atividade é lecionada
SELECT A.Sala FROM Atividade_Fitness AS A
WHERE A.Id_atividade = 7;

-- RE 17 -> Consultar os planos realizados por um dado cliente
DROP PROCEDURE IF EXISTS planos_cliente;

DELIMITER $$
CREATE PROCEDURE planos_cliente (IN nome_cliente VARCHAR(45))
BEGIN
SELECT P.Id_plano AS ID, P.Preco, P.Data_inicio AS Inicio, P.Estado FROM
Cliente AS C
INNER JOIN Plano AS P ON P.Id_cliente = C.Id_cliente
WHERE C.Nome = nome_cliente;
END $$
DELIMITER ;

CALL planos_cliente ('Ana Maria');

-- RE 18 -> Identificar o Top 3 de clientes com maior número de planos
associados
```

```
SELECT count(P.Id_plano) AS Nr_Planos, C.Nome FROM Plano AS P
INNER JOIN Cliente AS C ON C.Id_cliente = P.Id_cliente
GROUP BY C.Nome
ORDER BY Nr_planos DESC
LIMIT 3;

-- RE 19 -> Reconhecer o professor que lecionou mais atividades
SELECT count(A.Id_professor) AS Nr_Atividades, P.Nome FROM
Atividade_Fitness AS A
INNER JOIN Professor AS P ON P.Id_professor = A.Id_professor
GROUP BY A.Id_professor
ORDER BY Nr_atividades DESC
LIMIT 1;

-- RE 20 -> Obter o Top 3 das atividades com mais alunos inscritos
SELECT  A.Nr_inscritos, A.Nome FROM Atividade_Fitness AS A
ORDER BY Nr_inscritos DESC
LIMIT 3;

-- RE 21 -> Identificar o número de planos elaborados por cada professor
SELECT count(P.Id_plano) AS Nr_Planos, PR.Nome FROM Plano AS P
INNER JOIN Professor AS PR ON PR.Id_professor = P.Id_professor
GROUP BY PR.Nome
ORDER BY Nr_planos DESC;

-- RE 22 -> Verificar o Top 3 dos alunos com maior número de aulas de
uma determinada atividade fitness
SELECT sum(PA.Nr_aulas) AS Nr_Aulas, C.Nome AS Cliente FROM
Plano_Atividade_Fitness AS PA
INNER JOIN Plano AS P ON P.Id_plano = PA.Id_plano
INNER JOIN Atividade_Fitness AS A ON A.Id_atividade = PA.Id_atividade
INNER JOIN Cliente AS C ON C.Id_cliente = P.Id_cliente
WHERE A.Nome = 'Run'
GROUP BY P.Id_cliente
ORDER BY Nr_aulas DESC
LIMIT 3;

-- RE 23 -> Atividade fitness mais frequentada por um determinado aluno
SELECT sum(PA.Nr_aulas) AS Nr_Aulas, A.Nome AS Atividade FROM Cliente AS
C
INNER JOIN Plano AS P ON P.Id_cliente = C.Id_cliente
```

```
INNER JOIN Plano_Atividade_Fitness AS PA ON PA.Id_plano = P.Id_plano
INNER JOIN Atividade_Fitness AS A ON PA.Id_atividade = A.Id_atividade
WHERE C.Nome = 'Carolina Pinto'
GROUP BY A.Id_atividade
ORDER BY PA.Nr_aulas DESC
LIMIT 1;

-- RE 24 -> Devolver os planos elaborados por um professor ordenados pelo preço, bem como o cliente ao qual o plano foi atribuído
DROP PROCEDURE IF EXISTS planos_professor;

DELIMITER $$
CREATE PROCEDURE planos_professor (IN nome_professor VARCHAR(45))
BEGIN
SELECT PL.Preco , C.Nome AS Cliente FROM Professor AS P
INNER JOIN Plano AS PL ON PL.Id_professor = P.Id_professor
INNER JOIN Cliente AS C ON C.Id_cliente = PL.Id_cliente
WHERE P.Nome = nome_professor
ORDER BY PL.preco DESC;
END $$
DELIMITER ;

CALL planos_professor ('Andre Gonçalves');

-- RE 25 -> Ver quais os planos que ja foram realizados por um dado cliente com determinado estado (ativo ou inativo)
DROP PROCEDURE IF EXISTS estado_planos;

DELIMITER $$
CREATE PROCEDURE estado_planos (IN nome_cliente VARCHAR(45), IN estado VARCHAR(45))
BEGIN
SELECT P.Id_plano, P.Preco, P.Data_inicio AS Inicio FROM Cliente AS C
INNER JOIN Plano AS P ON P.Id_cliente = C.Id_cliente
WHERE C.Nome = nome_cliente
AND P.Estado = estado;
END $$
DELIMITER ;

CALL estado_planos('Marco Paulo', 'Inativo');
```

```
-- RE 26 -> As atividades fitness lecionadas por um professor
DROP PROCEDURE IF EXISTS atividades_professor;

DELIMITER $$
CREATE PROCEDURE atividades_professor (IN nome_professor VARCHAR(45))
BEGIN
SELECT A.Nome FROM Professor AS P
INNER JOIN Atividade_Fitness AS A ON A.Id_professor = P.Id_professor
WHERE P.Nome = nome_professor;
END $$
DELIMITER ;

CALL atividades_professor ('Andre Gonçalves');

-- RE 27 -> Nomes e contactos dos alunos que frequentam certa atividade
DROP PROCEDURE IF EXISTS alunos_atividade;

DELIMITER $$
CREATE PROCEDURE alunos_atividade (IN nome_atividade VARCHAR(45))
BEGIN
SELECT C.Nome AS Cliente, T.Numero, T.Tipo FROM Atividade_Fitness AS A
INNER JOIN Plano_Atividade_Fitness AS PA ON PA.Id_atividade =
A.Id_atividade
INNER JOIN Plano AS P ON P.Id_plano = PA.Id_plano
INNER JOIN Cliente AS C ON C.Id_cliente = P.Id_cliente
INNER JOIN Telemovel AS T ON C.Id_cliente = T.Id_cliente
WHERE P.Estado = 'Ativo'
AND A.Nome = nome_atividade;
END $$
DELIMITER ;

CALL alunos_atividade('Cycling');

-- RE 28 -> Alunos de uma localidade
DROP PROCEDURE IF EXISTS clientes_localidade;

DELIMITER $$
CREATE PROCEDURE clientes_localidade (IN nome_localidade VARCHAR(45))
BEGIN
SELECT C.Nome AS Cliente FROM Localidade AS L
INNER JOIN Cliente AS C ON C.Id_localidade = L.Id_localidade
```

```
WHERE L.Nome = nome_localidade
ORDER BY Cliente;
END $$
DELIMITER ;

CALL clientes_localidade('Souto');
```

IV. Anexo 4 - Exportação de Dados

```
import java.io.FileWriter;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.Statement;
import java.sql.ResultSet;
import java.sql.SQLException;

public class export {
public static void main(String[] args) {

try {
Class.forName("com.mysql.jdbc.Driver");
} catch (Exception e) {
e.printStackTrace();
}

try {
String URL = "localhost";
String SCHEMA = "ginasio";
String USERNAME = "root";
String PASSWORD = "password";
Connection conn =
DriverManager.getConnection("jdbc:mysql://"+URL+"/"+SCHEMA+"?autoReconnect=true&useSS

String filename = "C:/Users/Utilizador/Desktop/csv/maquina.csv";
FileWriter f = new FileWriter(filename);
String query = "select * from Maquina";
Statement stm = conn.createStatement();
ResultSet rs = stm.executeQuery(query);
f.append("IdMaquina,Tipo,Quantidade");
f.append('\n');
```

```
while(rs.next()){
f.append(String.valueOf(rs.getInt(1)));
f.append(',');
f.append(rs.getString(2));
f.append(',');
f.append(String.valueOf(rs.getInt(3)));
f.append('\n');
}
f.flush();
f.close();
System.out.println("CSV File is created!");

filename = "C:/Users/Utilizador/Desktop/csv/cliente.csv";
f = new FileWriter(filename);
query = "select * from Cliente";
stm = conn.createStatement();
rs = stm.executeQuery(query);
f.append("IdCliente,Nome,DataNascimento,Endereco,IdLocalidade");
f.append('\n');
while(rs.next()){
f.append(String.valueOf(rs.getInt(1)));
f.append(',');
f.append(rs.getString(2));
f.append(',');
f.append(rs.getString(3));
f.append(',');
f.append(rs.getString(4));
f.append(',');
f.append(String.valueOf(rs.getInt(5)));
f.append('\n');
}
f.flush();
f.close();
System.out.println("CSV File is created!");

filename = "C:/Users/Utilizador/Desktop/csv/atividade_fitness.csv";
f = new FileWriter(filename);
query = "select * from Atividade_Fitness";
```

```
stm = conn.createStatement();
rs = stm.executeQuery(query);
f.append("IdAtividade,MaxParticipantes,Nome,Duracao,Sala,NrInscritos,IdProfessor");
f.append('\n');
while(rs.next()){
f.append(String.valueOf(rs.getInt(1)));
f.append(',');
f.append(String.valueOf(rs.getInt(2)));
f.append(',');
f.append(rs.getString(3));
f.append(',');
f.append(rs.getString(4));
f.append(',');
f.append(String.valueOf(rs.getInt(5)));
f.append(',');
f.append(String.valueOf(rs.getInt(6)));
f.append(',');
f.append(String.valueOf(rs.getInt(7)));
f.append('\n');
}
f.flush();
f.close();
System.out.println("CSV File is created!");

filename = "C:/Users/Utilizador/Desktop/csv/limitacao_fisica.csv";
f = new FileWriter(filename);
query = "select * from Limitacao_Fisica";
stm = conn.createStatement();
rs = stm.executeQuery(query);
f.append("IdLimitacao,Nome");
f.append('\n');
while(rs.next()){
f.append(String.valueOf(rs.getInt(1)));
f.append(',');
f.append(rs.getString(2));
f.append('\n');
}
```

```
f.flush();
f.close();
System.out.println("CSV File is created!");

filename = "C:/Users/Utilizador/Desktop/csv/localidade.csv";
f = new FileWriter(filename);
query = "select * from Localidade";
stm = conn.createStatement();
rs = stm.executeQuery(query);
f.append("IdLocalidade,Nome");
f.append('\n');
while(rs.next()){
f.append(String.valueOf(rs.getInt(1)));
f.append(',');
f.append(rs.getString(2));
f.append('\n');
}
f.flush();
f.close();
System.out.println("CSV File is created!");

filename = "C:/Users/Utilizador/Desktop/csv/plano.csv";
f = new FileWriter(filename);
query = "select * from Plano";
stm = conn.createStatement();
rs = stm.executeQuery(query);
f.append("IdPlano,Preco,DataInicio,Estado,IdCliente,IdProfessor");
f.append('\n');
while(rs.next()){
f.append(String.valueOf(rs.getInt(1)));
f.append(',');
f.append(rs.getString(2));
f.append(',');
f.append(rs.getString(3));
f.append(',');
f.append(rs.getString(4));
f.append(',');
f.append(String.valueOf(rs.getInt(5)));
```

```
f.append(',');
f.append(String.valueOf(rs.getInt(6)));
f.append('\n');
}
f.flush();
f.close();
System.out.println("CSV File is created!");

filename="C:/Users/Utilizador/Desktop/csv/telemovel.csv";
f = new FileWriter(filename);
query = "select * from Telemovel";
stm = conn.createStatement();
rs = stm.executeQuery(query);
f.append("IdTelemovel,Tipo,Numero,IdCliente");
f.append('\n');
while(rs.next()){
f.append(String.valueOf(rs.getInt(1)));
f.append(',');
f.append(rs.getString(2));
f.append(',');
f.append(rs.getString(3));
f.append(',');
f.append(String.valueOf(rs.getInt(4)));
f.append('\n');
}
f.flush();
f.close();
System.out.println("CSV File is created!");

filename = " C:/Users/Utilizador/Desktop/csv/professor.csv";
f = new FileWriter(filename);
query = "select * from Professor";
stm = conn.createStatement();
rs = stm.executeQuery(query);
f.append("IdProfessor,Nome,Endereco,DataNascimento,Telemovel,Email,Estado,IdLocalidad
f.append('\n');
while(rs.next()){
f.append(String.valueOf(rs.getInt(1)));
```

```
f.append(',');
f.append(rs.getString(2));
f.append(',');
f.append(rs.getString(3));
f.append(',');
f.append(rs.getString(4));
f.append(',');
f.append(rs.getString(5));
f.append(',');
f.append(rs.getString(6));
f.append(',');
f.append(rs.getString(7));
f.append(',');
f.append(String.valueOf(rs.getInt(8)));
f.append('\n');
}
f.flush();
f.close();
System.out.println("CSV File is created!");

filename = "
C:/Users/Utilizador/Desktop/csv/cliente_limitacao_fisica.csv";
f = new FileWriter(filename);
query = "select * from Cliente_Limitacao_Fisica";
stm = conn.createStatement();
rs = stm.executeQuery(query);
f.append("IdCliente,IdLimitacao");
f.append('\n');
while(rs.next()){
f.append(String.valueOf(rs.getInt(1)));
f.append(',');
f.append(String.valueOf(rs.getInt(2)));
f.append('\n');
}
f.flush();
f.close();
System.out.println("CSV File is created!");
```

```
filename = "
C:/Users/Utilizador/Desktop/csv/plano_atividade_fitness.csv";
f = new FileWriter(filename);
query = "select * from Plano_Atividade_Fitness";
stm = conn.createStatement();
rs = stm.executeQuery(query);
f.append("NrAulas,IdPlano,IdAtividade");
f.append('\n');
while(rs.next()){
f.append(String.valueOf(rs.getInt(1)));
f.append(',');
f.append(String.valueOf(rs.getInt(2)));
f.append(',');
f.append(String.valueOf(rs.getInt(3)));
f.append('\n');
}
f.flush();
f.close();
System.out.println("CSV File is created!");

filename = "
C:/Users/Utilizador/Desktop/csv/atividade_fitness_maquina.csv";
f = new FileWriter(filename);
query = "select * from Atividade_Fitness_Maquina";
stm = conn.createStatement();
rs = stm.executeQuery(query);
f.append("Nr_Maquinas,IdAtividade,IdMaquina");
f.append('\n');
while(rs.next()){
f.append(String.valueOf(rs.getInt(1)));
f.append(',');
f.append(String.valueOf(rs.getInt(2)));
f.append(',');
f.append(String.valueOf(rs.getInt(3)));
f.append('\n');
}
f.flush();
f.close();
```

```
System.out.println("CSV File is created!");

filename =
"C:/Users/Utilizador/Desktop/csv/plano_atividade_fitness.csv";
f = new FileWriter(filename);
query = "select * from Plano_Atividade_Fitness";
stm = conn.createStatement();
rs = stm.executeQuery(query);
f.append("NrAulas,IdPlano,IdAtividade");
f.append('\n');
while(rs.next()){
f.append(String.valueOf(rs.getInt(1)));
f.append(',');
f.append(String.valueOf(rs.getInt(2)));
f.append(',');
f.append(String.valueOf(rs.getInt(3)));
f.append('\n');
}
f.flush();
f.close();
conn.close();
System.out.println("CSV File is created!");
} catch (Exception e) {
e.printStackTrace();
}
}
}
```