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

Unidade Curricular de Bases de Dados

Ano Letivo de 2019/2020

Testes Clínicos De Atletas de Atletismo

João Manuel Silva Antunes (A81663),

Francisco Domingos Martins Oliveira (A82066),

Maria Luísa Faria Silva (A82124)

Grupo 5

Janeiro, 2020

BD

Data de Receção	
Responsável	
Avaliação	
Observações	

Testes Clínicos de Atletas de Atletismo

Francisco Domingos Martins Oliveira (A82066),

João Manuel Silva Antunes (A81663),

Maria Luísa Faria Silva (A82124)

Dezembro, 2019

Resumo

Foi-nos proposta a realização de uma Base de Dados cujo tema é agendamento e realização de Testes Clínicos de atletas de atletismo de diferentes modalidades e categorias.
A nossa Base de Dados é capaz de guardar informação relativa a Atletas, Modalidades, Categoria, Testes Clínicos e Profissionais de Saúde e tem como principal função facilitar a consulta de informação relativa aos mesmos.

Área de Aplicação: *Bases de Dados* (BD)
Palavras-Chave: Sistema de Gestão de Bases de Dados, mySQL, mongoDB, atletas, profissionais de saúde, testes clínicos, NoSQL, Migração de Dados

Índice:

Índice de Figuras

Índice de Tabelas

1. Introdução

1.1. Contextualização

A Federação Portuguesa de Atletismo é responsável pelos testes clínicos realizados nos diversos atletas que participam em provas. Para melhor gestão e armazenamento dos testes clínicos contactaram-nos para criarmos um sistema que pudesse realizar o que é pretendido.

1.2. Apresentação do Caso de Estudo

Após terem nos dado este problema foi decidido pelo grupo que a melhor forma de armazenar toda esta informação será através de uma base de dados, devido ao fácil armazenamento, acesso e manipulação dos dados tal como a de criar aplicações para esses mesmos.

1.3. Motivação e Objetivos

Queremos com este trabalho aprofundar o nosso conhecimento sobre a matéria ao criar uma base dados que consiga resolver um problema real de forma eficiente.
Tendo em conta o vasto uso de sistemas de base de dados e a sua importância para a sociedade hoje em dia é necessário que estas sejam eficientes e bem estruturadas para os problemas de cada empresa. E é exatamente isso mesmo que pretendemos com a que criamos.

1.4. Estrutura do Relatório

Neste relatório vai ser abordado os requisitos feita pela federação portuguesa de atletismo, modelação conceptual, modelação lógica e a resposta aos requisitos feitos em SQL e NoSQL, tal como todas as suas etapas.

2. Levantamento e Análise de Requisitos

2.1. Requisitos Levantados

Após consultar com a Federação Portuguesa de Atletismo concluímos que os requisitos são os seguintes:

2.1.1. Requisitos de Descrição

O sistema terá vários testes e cada **Teste** refere-se a um **Atleta** e a um **Profissional de Saúde (PFS).**

Cada **Teste** regista a data da realização, o local da realização, o tipo de teste e o resultado.

A um **Atleta** está associado o seu nome, data de nascimento **(DoB)**, o seu contacto, email e a sua condição médica.

A um Profissionais de saúde **(pfs)** estará associado o seu nome, contacto, e-mail e especialidade.

A cada **Categoria** e **Modalidade** estão associados o seu respetivo nome e uma pequena descrição da mesma.

2.1.2. Requisitos de Exploração

- Obter todos os testes clínicos e os atletas num certo dia.
- Obter todas os testes clínicos agendados.
- Mostrar as informações de um atleta dado o seu identificador.
- Mostrar as informações de um profissional de saúde dado o seu identificador.
- Obter os testes agendados de um atleta dado o seu identificador.
- Obter o resultado de testes passados de um atleta dado o seu identificador.

2.1.3. Requisitos de Controlo

- Existe um utilizador da base de dados chamado administrador que tem acesso total a todas as tabelas, podendo manipulá-las como bem entender.
- Existe outro utilizador, visualizador, no qual o seu acesso é exclusivamente de observação, não podendo modificar o estado da base de dados.

3. Modelação Conceptual

Como o problema tem, de certa forma, dimensões pequenas, decidimos abordá-lo todo de uma só vez, ou seja, de forma centralizada. O que significa que os requisitos para cada utilizador foram fundidos num único conjunto de requisitos e a partir daí realizámos a modelação da base de dados.

3.1. Identificação e caracterização das entidades

Depois de identificados os requisitos, decidimos que era necessária a criação das cinco seguintes entidades: Atleta, Categoria. Modalidade, Teste e PFS.

Entidade	Caracterização
Atleta	Descreve cada atleta do sistema.
Categoria	Descreve as categorias a que pertencem os atletas.
Modalidade	Descreve as modalidades em que os atletas estão integrados.
Teste	Descreve os testes clínicos dos vários atletas.
PFS	Descreve os profissionais de saúde que realizam os testes clínicos.

Tabela 1 - Tabela de caracterização das entidades

3.2. Identificação e caracterização dos relacionamentos

Tal como nas entidades, depois de identificados os requisitos, achamos que devíamos inserir os seguintes relacionamentos.

Relacionamento	Entidade 1	Entidade 2	Cardinalidade
Realiza	Atletas	Testes	1 : N
Faz	PfS	Testes	1 : N
Contém	Modalidade	Categoria	1 : N
Pertence	Atletas	Categoria	1 : N

Tabela 2 - Tabela de caracterização dos relacionamentos

3.3. Identificação e Caracterização dos Atributos

Entidade / Relacionamento	Atributo	Descrição
Atletas	IDCivil	Número de identificação civil do atleta
	Nome	Nome do atleta
	DoB	Data de nascimento do atleta
	Contacto	Contacto do atleta
	Email	Email do atleta
Pfs	ID	Número de identificação profissional de saúde
	Nome	Nome do profissional de saúde
	Contacto	Contacto do profissional de saúde
	Email	E-Mail do profissional de saúde
	Especialidade	Especialidade do profissional de saúde
Modalidade	ID	Identificador da Modalidade
	Nome	Nome da modalidade
Categoria	ID	Identificador da categoria
	Nome	Nome do funcionário
	Descrição	Idade do funcionário
Testes	ID	Identificador dos testes
	DataTestes	Data a que os testes irão\foram realizados
	Local	Local onde os testes foram realizados
	TipoTestes	Tipo de testes a que os atletas foram sujeitos
	Resultado	Resultado do teste

Tabela 3 - Tabela de caracterização dos atributos

3.4. Apresentação do Diagrama ER

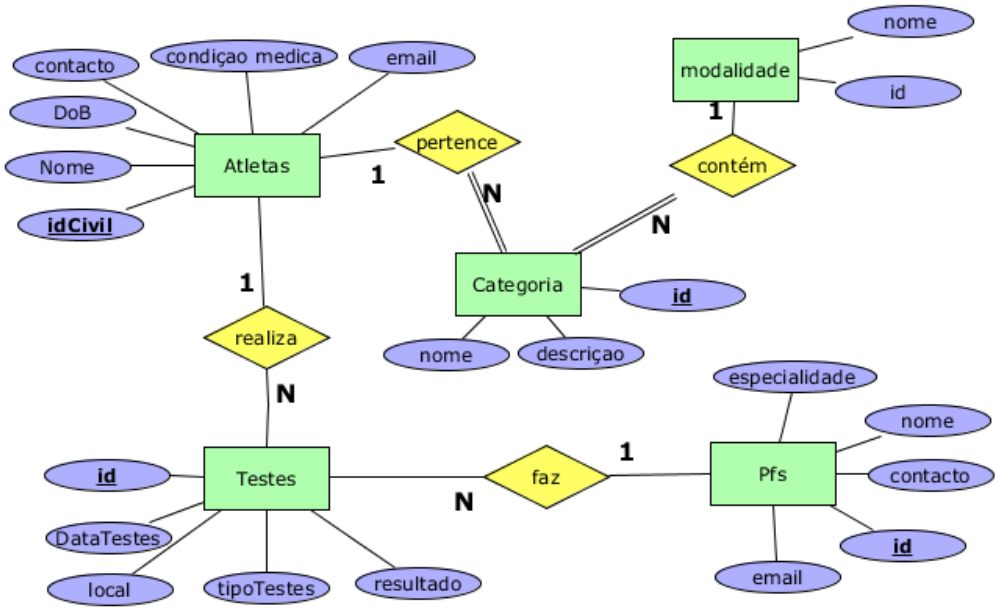

Figura 1 - Modelo Conceptual da Base de Dados

4. Modelação Lógica

4.1. Construção do Modelo Lógico

Para criarmos o modelo de dados lógicos usando o modelo conceptual seguimos um conjunto de regras:

- Cada **entidade** gera uma tabela, com os seus atributos como colunas dessa mesma tabela. Atributos **identificadores** dão origem a colunas que são **chaves candidatas**.
- **Relacionamentos** de cardinalidade 1 : N dão origem a uma **chave externa** na tabela com cardinalidade N referente à **chave primária** da tabela com cardinalidade 1.
- **Relacionamentos** de cardinalidade N : M dão origem a uma nova tabela cuja **chave primária** é a chave conjunta das **chaves externas** das entidades que estão envolvidas neste mesmo relacionamento.
  Tais chaves externas são referentes às chaves primárias das tabelas das entidades do relacionamento.

Depois disto restava-nos apenas *tipar* todos os atributos conforme os requisitos e as suas descrições consolidadas anteriormente (3.3).
Assim concluímos a construção do modelo de dados lógico.

4.2. Desenho do Modelo Lógico

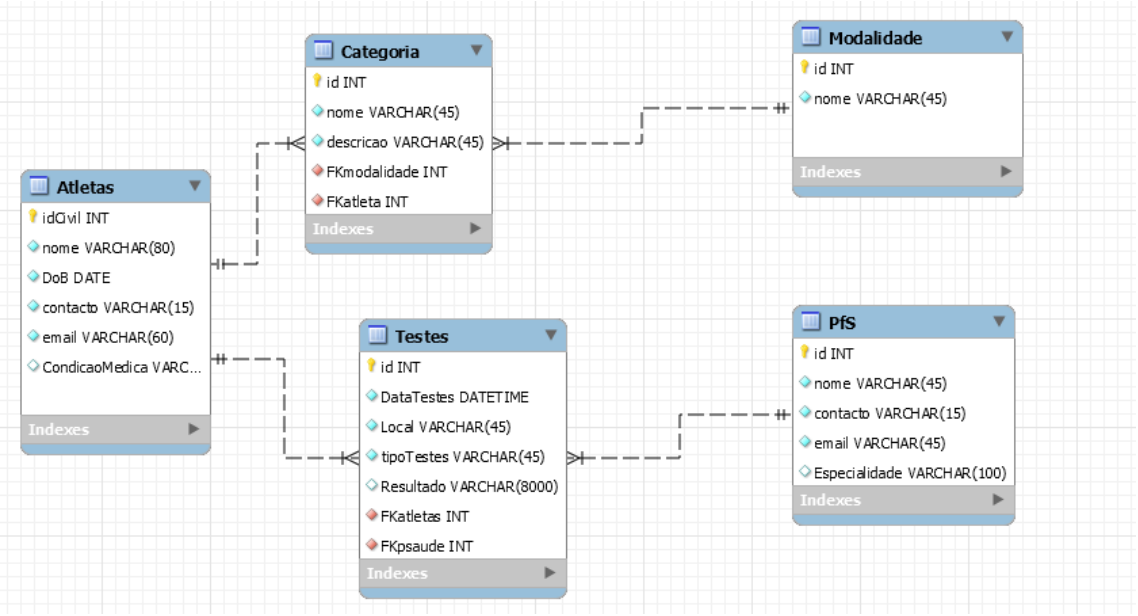

Figura 2 - Modelo Lógico da Base de Dados

5. Implementação Física em SQL

Após muita ponderação, optamos por escolher o MySQL como gestor da nossa base de dados. Vários fatores que nos fez tomar este rumo foram: outras empresas, com bases de dados "densas", conhecidas o fazerem (ex: Facebook, Google, Adobe, etc), custos reduzidos em comparação com outros "gestores", entre outros que passaremos a explicar. Com perspetiva no futuro, procurávamos um sistema que fosse prático e barato, mas que mantivesse a eficiência necessária. O MySQL revelou-se bastante interessante nesse ponto de vista uma vez que se virmos a taxa de custo anual, esta revela-se bastante reduzida em comparação com outros sistemas. Em termos de performance podemos verificar que é perto do ótimo, a nível de tempo de respostas de tabelas é bastante rápido e posto isto, foi um fator preponderante à nossa escolha.

5.1. Tradução dos Procedimentos do utilizador para SQL (alguns exemplos)

Obter todos os atletas que vão ter testes num certo dia:

```
DELIMITER //
CREATE PROCEDURE AtletasComConsultasNoDia(dia datetime)
BEGIN
SELECT atletas.nome as Nome_Do_Atleta, testes.Local , PfS.nome as Profissional_De_Saúde,
testes.tipoTestes as Tipo_De_Consulta, testes.DataTestes as Marcado_no_Dia  ,
categoria.nome as Categoria, modalidade.nome as Modalidade
from atletas
JOIN testes on atletas.idCivil = testes.FKatletas
JOIN categoria on atletas.idCivil= categoria.FKatleta
JOIN modalidade on categoria.FKmodalidade = modalidade.id
join PfS on PfS.id=testes.fkpsaude
where day(dia) = day(testes.DataTestes) and month(dia) = month(testes.DataTestes) and year(dia) = year(testes.DataTestes)
ORDER BY testes.DataTestes ASC;
end
// DELIMITER ;
```

Figura 3 - Resolução do primeiro procedimento

Obter todas as consultas agendadas:

```
DELIMITER //
CREATE PROCEDURE TestesAgendados()
BEGIN
SELECT atletas.nome as Nome_Do_Atleta, testes.Local , PfS.nome as Profissional_De_Saúde,
testes.DataTestes as Marcado_no_Dia  , categoria.nome as Categoria, modalidade.nome as Modalidade
from atletas
JOIN testes on atletas.idCivil = testes.FKatletas
JOIN categoria on atletas.idCivil= categoria.FKatleta
JOIN modalidade on categoria.FKmodalidade = modalidade.id
join PfS on PfS.id=testes.fkpsaude
where year(CURRENT_TIMESTAMP) < year(testes.DataTestes)
or (year(CURRENT_TIMESTAMP) < year(testes.DataTestes) and month(CURRENT_TIMESTAMP) < month(testes.DataTestes))
or  (day(CURRENT_TIMESTAMP) <= day(testes.DataTestes) and month(CURRENT_TIMESTAMP) <= month(testes.DataTestes)
and year(CURRENT_TIMESTAMP) <= year(testes.DataTestes))
ORDER BY testes.DataTestes ASC;
end
// DELIMITER ;
```

Figura 4 - Resolução do segundo procedimento.

Mostrar as informações de um atleta dado o seu identificador:

```
DELIMITER //
CREATE PROCEDURE InfoAtleta (numatleta int(11))
BEGIN
SELECT atletas.idCivil as Numero_De_Identificação_Do_Atleta , atletas.nome as Nome ,
atletas.DoB as Data_De_Nascimento , atletas.email, atletas.contacto , atletas.CondicaoMedica ,
categoria.nome as Categoria , modalidade.nome as Modalidade
from atletas
JOIN categoria on atletas.idCivil = categoria.FKatleta
JOIN modalidade on categoria.FKmodalidade = modalidade.id
where atletas.idCivil =  numatleta ;
END
// DELIMITER ;
```

Figura 1 - Resolução do terceiro procedimento.

Mostrar as informações de um profissional de saúde dado o seu identificador:

```
DELIMITER //
CREATE PROCEDURE infoPfS (idPfS int(11))
BEGIN
SELECT PfS.id as Numero_De_Identificação_Do_PfS , PfS.nome as Nome ,
PfS.email, PfS.contacto , PfS.especialidade as Especialidade
from PfS
where PfS.id =  idPfS ;

END
// DELIMITER ;
```

Figura 6 - Resolução do quarto procedimento.

Mostrar os testes agendados de um atleta dado o seu identificador:

```
DELIMITER //
CREATE PROCEDURE TestesAtleta (numatleta int(11))
BEGIN
SELECT PfS.nome as Profissional_De_Saúde, testes.tipoTestes as Tipo_De_Consulta,
testes.DataTestes as Marcado_no_Dia , testes.id as Código_do_Teste, testes.Local
from atletas
JOIN testes on atletas.idCivil = testes.FKatletas
join PfS on PfS.id=testes.fkpsaude
where (year(CURRENT_TIMESTAMP) < year(testes.DataTestes)
or (year(CURRENT_TIMESTAMP) < year(testes.DataTestes) and month(CURRENT_TIMESTAMP) < month(testes.DataTestes))
or  (day(CURRENT_TIMESTAMP) <= day(testes.DataTestes) and month(CURRENT_TIMESTAMP) <= month(testes.DataTestes)
and year(CURRENT_TIMESTAMP) <= year(testes.DataTestes))) and atletas.idCivil =  numatleta ;

END
// DELIMITER ;
```

Figura 7 - Resolução do quinto procedimento.

Mostrar os resultados de testes realizados de um atleta dado o seu identificador:

```
DELIMITER //
CREATE PROCEDURE ResultadoAtleta (numatleta int(11))
BEGIN
SELECT testes.id as Código_do_Teste , PfS.nome as Profissional_De_Saúde,
testes.tipoTestes as Tipo_De_Consulta, testes.DataTestes as Marcado_no_Dia , testes.Resultado
from atletas
JOIN testes on atletas.idCivil = testes.FKatletas
join PfS on PfS.id=testes.fkpsaude
where (year(CURRENT_TIMESTAMP) > year(testes.DataTestes)
or (year(CURRENT_TIMESTAMP) > year(testes.DataTestes) and month(CURRENT_TIMESTAMP) > month(testes.DataTestes))
or  (day(CURRENT_TIMESTAMP) > day(testes.DataTestes) and month(CURRENT_TIMESTAMP) >= month(testes.DataTestes)
and year(CURRENT_TIMESTAMP) >= year(testes.DataTestes))) and atletas.idCivil =  numatleta ;

END
// DELIMITER ;
```

Figura 8 - Resolução do sexto procedimento.

5.2. Definição e caracterização dos mecanismos de segurança em SQL (alguns exemplos)

O mecanismo de segurança da nossa base de dados são os dois utilizadores e as suas permissões.

```
CREATE USER 'admin' IDENTIFIED BY 'administrador';
GRANT ALL ON trabalhonovo.* TO 'admin';

CREATE USER 'visualizador' IDENTIFIED BY 'visualizar';
GRANT ALL ON trabalhonovo.* TO 'visualizador';
```

6. Implementação NoSql

6.1. Justificação do uso NoSql

Com o aumento constante da utilização da base de dados criada para Testes Clínicos irá ser necessário aumentar a eficiência de acesso e registo na mesma. Visto isto, optamos por um modelo não relacional, devido às suas vantagens a nível de escalabilidade, que se refere à capacidade do sistema lidar com um aumento de informação, que é o que acontece neste caso, tornado assim mais eficiente.
Iremos utilizar o MongoDB, que é um programa de base de dados orientada a documentos.

6.2. Identificação das interrogações feitas pelo utilizador

As interrogações serão as mesmas utilizadas no modelo relacional.

- Obter todos os testes clínicos e os atletas num certo dia.
- Obter todas os testes clínicos agendados.
- Mostrar as informações de um atleta dado o seu identificador.
- Mostrar as informações de um profissional de saúde dado o seu identificador.
- Obter os testes agendados de um atleta dado o seu identificador.
- Obter o resultado de testes passados de um atleta dado o seu identificador.

6.3. Descrição do processo de migração

O processo de migraçao de dados consiste na conversão de toda a informação do sistema SQL para o sistema MongoDB.

A extração de dados é realizada utilizando MySQL.

Utilizamos Python para transformar os dados extraídos para converter os dados extraídos em formato JSON.

Por fim estes dados são carregados no MongoDB.

6.4. Estrutura da base de dados NoSQL

No modelo NoSQL optamos por criar apenas uma coleção, Testes, que irá conter todas as informações pessoais do atleta tal como a categoria e modalidade a que se insere, mas também a informação do profissional de saúde que realizou o teste clínico.

6.5. Processo de migração dos dados

```
import sys
import mysql.connector
import json
import io
from datetime import datetime

cnx = mysql.connector.connect(user='root', password='dev',
                              host='127.0.0.1',
                              database='trabalhonovo')

cursor = cnx.cursor()
cursor.execute("SET NAMES 'utf8';")

cursor.execute("SELECT id FROM Testes;")

testes = []
for (idTeste,) in cursor:
    testes.append(int(idTeste))

cursor.execute("SELECT id FROM PfS;")
pfs = []
for (idPFS,) in cursor:
    pfs.append(int(idPFS))

cursor.execute("SELECT idCivil FROM Atletas;")
atletas = []
for (idAtleta,) in cursor:
    atletas.append(int(idAtleta))

cursor.close()
cursor = cnx.cursor()

def data_to_string(data):
    return data.strftime("%Y-%m-%d %H:%M:%S")

def obter_modalidade(idModalidade):
    modalidade = {}
    cursor.execute("SELECT id, nome FROM Modalidade WHERE id = %u" %
idModalidade)
```

```
    for (idMod, nome) in cursor:
        modalidade['id'] = idMod
        modalidade['nome'] = nome

    return modalidade

def obter_categorias(idAtleta):
    categorias = []
    cursor.execute("SELECT id, nome, descricao, FKModalidade FROM
Categoria WHERE FKatleta = %u" % idAtleta)
    for (idCat, nome, descricao, fkModalidade) in cursor:
        categoria = {}
        categoria['idCat'] = idCat
        categoria['nome'] = nome
        categoria['descricao'] = descricao
        categoria['modalidade'] = fkModalidade
        categorias.append(categoria)

    for categoria in categorias:
        categoria['modalidade']                                    =
obter_modalidade(categoria['modalidade'])

    return categorias

def obter_atleta(id_):
    atleta = {}
    cursor.execute("SELECT idCivil, nome, DoB, contacto, email,
CondicaoMedica FROM Atletas WHERE idCivil = %u" % id_)
    for (idAtleta, nome, data, contacto, email, CondicaoMedica) in
cursor:
        atleta['idCivil'] = idAtleta
        atleta['nome'] = nome.encode("utf8")
        atleta['DoB'] = data_to_string(data)
        atleta['contacto'] = contacto
        atleta['email'] = email
        atleta['CondicaoMedica'] = CondicaoMedica
        atleta['categorias'] = obter_categorias(id_)

    return atleta

def obter_pfs(idPfs):
```

```
    pfs = {}
    cursor.execute("SELECT id, nome, contacto, email, Especialidade
FROM PfS WHERE id = %u" % idPFS)
    for (idPfs, nome, contacto, email, especialidade) in cursor:
        pfs['idPfs'] = idPfs
        pfs['nome'] = nome
        pfs['contacto'] = contacto
        pfs['email'] = email
        pfs['especialidade'] = especialidade

    return pfs


testesJson = []
for idTeste in testes:
    testeJson = json.loads('{}')
    cursor.execute("SELECT * FROM Testes WHERE id = %u" % idTeste)
    for (idTeste, DataTeste, Local, tipoTestes, Resultado, atleta,
pfs) in cursor:
        testeJson['id'] = idTeste
        testeJson['data'] = data_to_string(DataTeste)
        testeJson['local'] = Local
        testeJson['tipo'] = tipoTestes
        testeJson['resultado'] = Resultado
        testeJson['atleta'] = obter_atleta(atleta)
        testeJson['pfs'] = obter_pfs(pfs)

        testesJson.append(testeJson)

with io.open('out.json', 'w', encoding='utf8') as json_file:
    data = json.dumps(testesJson, ensure_ascii=False, indent=4 ,
encoding='utf8')
    json_file.write(unicode(data) + u"\n")


cursor.close()
```

6.6. Interrogações

- Obter todos os testes clínicos e os atletas num certo dia:

```
db.trabalho.find({"data": {"$gte": ISODate("2016-07-01"),"$lt":
ISODate("2016-07-02")}}).pretty()
```

- Obter todas os testes clínicos agendados.

```
db.trabalho.find({"data": {"$gte": ISODate("2019-12-26")}}).pretty()
```

- Mostrar as informações de um atleta dado o seu identificador.

```
db.trabalho.findOne({"atleta.idCivil": 4142}, {atleta: 1, _id: 0})
```

- Mostrar as informações de um profissional de saúde dado o seu identificador.

```
db.trabalho.findOne({"pfs.idPfs": 2134}, {pfs: 1, _id: 0})
```

- Obter os testes agendados de um atleta dado o seu identificador.

```
db.trabalho.find({ $and: [{"data": {"$gte": ISODate("2019-12-26")}},
{"atleta.idCivil": 4142}]}).pretty()
```

- Obter o resultado de testes passados de um atleta dado o seu identificador.

```
db.trabalho.find({ $and: [{"resultado" : "sucesso"},
{"atleta.idCivil": 4142}]}).pretty()
```

- Foi necessário também colocar a string vinda da migração em formato data:

```
db.trabalho.updateMany({},[{"$set": {"data": {"$toDate": "$data"}}}])
```

Lista de Siglas e Acrónimos

BD Base de Dados

PfS Profissional de saúde

Conclusão

A realização deste trabalho permitiu-nos perceber melhor o processo de desenvolvimento de um projeto, mais concretamente, as fases inicias de especificação e modelação.

Quanto às bases de dados aqui criadas, futuramente poderíamos continuar o seu desenvolvimento, por exemplo, acrescentar um histórico de faturas relacionadas com os testes clínicos, entre outras coisas.

Concluindo, foi um projeto onde pudemos utilizar e melhorar as competências e os conhecimentos relativos à UC de Bases de Dados, e do qual tiramos um balanço positivo do seu desenvolvimento.

Anexo 1 Modelo físico

```
-- MySQL Workbench Forward Engineering

SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0;
SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS, FOREIGN_KEY_CHECKS=0;
SET @OLD_SQL_MODE=@@SQL_MODE, SQL_MODE='ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO_ZERO_DATE,ERROR_FOR_DIVISION_BY_ZERO,NO_ENGINE_SUBSTITUTION';

-- -----------------------------------------------------
-- Schema TestesClinicos
-- -----------------------------------------------------
--
--

-- -----------------------------------------------------
-- Schema TestesClinicos
--
--
--
-- -----------------------------------------------------
CREATE SCHEMA IF NOT EXISTS `TestesClinicos` DEFAULT CHARACTER SET utf8 COLLATE utf8_bin ;
USE `TestesClinicos` ;

-- -----------------------------------------------------
-- Table `TestesClinicos`.`Atletas`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `TestesClinicos`.`Atletas` (
  `idCivil` INT NOT NULL,
  `nome` VARCHAR(80) NOT NULL,
  `DoB` DATE NOT NULL,
```

```
  `contacto` VARCHAR(15) NOT NULL,
  `email` VARCHAR(60) NOT NULL,
  `CondicaoMedica` VARCHAR(800) NULL,
  PRIMARY KEY (`idCivil`))
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `TestesClinicos`.`Modalidade`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `TestesClinicos`.`Modalidade` (
  `id` INT NOT NULL,
  `nome` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`id`))
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `TestesClinicos`.`Categoria`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `TestesClinicos`.`Categoria` (
  `id` INT NOT NULL,
  `nome` VARCHAR(45) NOT NULL,
  `descricao` VARCHAR(45) NOT NULL,
  `FKmodalidade` INT NOT NULL,
  `FKatleta` INT NOT NULL,
  PRIMARY KEY (`id`),
  INDEX `IDmodalidades_idx` (`FKmodalidade` ASC) VISIBLE,
  INDEX `IDatleta_idx` (`FKatleta` ASC) VISIBLE,
  CONSTRAINT `IDmodalidade`
    FOREIGN KEY (`FKmodalidade`)
    REFERENCES `TestesClinicos`.`Modalidade` (`id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `IDatleta`
    FOREIGN KEY (`FKatleta`)
    REFERENCES `TestesClinicos`.`Atletas` (`idCivil`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;
```

```
-- -----------------------------------------------------
-- Table `TestesClinicos`.`PfS`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `TestesClinicos`.`PfS` (
  `id` INT NOT NULL,
  `nome` VARCHAR(45) NOT NULL,
  `contacto` VARCHAR(15) NOT NULL,
  `email` VARCHAR(45) NOT NULL,
  `Especialidade` VARCHAR(100) NULL,
  PRIMARY KEY (`id`))
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `TestesClinicos`.`Testes`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `TestesClinicos`.`Testes` (
  `id` INT NOT NULL,
  `DataTestes` DATETIME NOT NULL,
  `Local` VARCHAR(45) NOT NULL,
  `tipoTestes` VARCHAR(45) NOT NULL,
  `Resultado` VARCHAR(8000) NULL,
  `FKatletas` INT NOT NULL,
  `FKpsaude` INT NOT NULL,
  PRIMARY KEY (`id`),
  INDEX `atletasfk_idx` (`FKatletas` ASC) VISIBLE,
  INDEX `FKmedico_idx` (`FKpsaude` ASC) VISIBLE,
  CONSTRAINT `atletasfk`
    FOREIGN KEY (`FKatletas`)
    REFERENCES `TestesClinicos`.`Atletas` (`idCivil`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `FKmedico`
    FOREIGN KEY (`FKpsaude`)
    REFERENCES `TestesClinicos`.`PfS` (`id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;
```

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