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

Unidade Curricular de Bases de Dados

Ano Lectivo de 2020/2021

Base de Dados da Escola de Condução

GRUPO 18

Gonçalo Nogueira(a86617),
Joana Sousa(a83614),
João André Freitas (a83782),
Tiago Gomes(a78141)

Dezembro,2020

BD

Data de Recepção	
Responsável	
Avaliação	
Observações	

Base de Dados da Escola de Condução

Grupo 18
Gonçalo Nogueira(a86617),
Joana Sousa(a83614),
João André Freitas (a83782),
Tiago Gomes(a78141)

Dezembro,2020

Resumo

Este relatório diz respeito ao projeto realizado na Unidade Curricular de Base de Dados, lecionada no 3º ano do curso de Engenharia Informática na Universidade do Minho, que consiste no agendamento e planeamento de aulas práticas numa escola de condução.

No decorrer deste relatório, será possível verificar algumas decisões feitas pelo nosso grupo em relação ao Sistema de Base de Dados, desde a sua análise até uma parte mais demonstrativa.

Nos próximos capítulos, é evidente a necessidade de análise de requisitos e uma contextualização da nossa futura base de dados, definindo também as condições e os requisitos aos quais deverá ser capaz de estar apta.

Primeiramente, começamos por desenvolver o modelo conceptual, e de seguida a conceção dos modelos, tanto lógico como físico. Em cada uma destas etapas, fizemos a sua validação, de maneira a termos uma base de dados mais eficiente e robusta.

Para além de concebermos, descrevermos e validarmos o Sistema de Base de Dados, apresentamos as *queries* que julgamos ser de maior interesse e às quais a nossa Base de Dados tem que responder.

Finalmente, apresentamos as nossas conclusões e as dificuldades que sentimos durante todo este trabalho, indicando também alternativas e correções que fizemos ao longo de todo este processo, bem como uma análise crítica da nossa abordagem.

Área de Aplicação: Sistemas de Base de Dados.
Palavras-Chave: MySQL, Base de Dados Relacional, Modelo Conceptual e Modelo Lógico, Modelo Físico, Queries,Entidades, Relacionamentos.

Índice

1. Introdução

1.1. Contextualização

Desde cedo que o António Gonçalves adorava brincar com carros e sempre sonhou ter a sua própria escola de condução, ainda por cima seria a primeira na sua cidade. Foi assim que a escola de condução “Drive It” nasceu e começou a sua atividade há 1 ano, sendo o Sr. Gonçalves um dos instrutores. Com a abertura para o mercado, a escola pretendia captar a atenção do público com residência ou local laboral próximo à sua localização física.

Nos últimos meses verificou-se um aumento elevado do número de clientes interessados em frequentar a escola devido à satisfação dos primeiros clientes, sendo conhecida nacionalmente pela simpatia e atendimento.

1.2. Apresentação do Caso de Estudo

Dado o aumento da procura e da adesão por parte das pessoas da cidade, a administração da escola de condução “Drive It” decidiu implementar um sistema de base de dados para gerir os seus alunos, instrutores, funcionários e aulas. Esta ideia surgiu com o objetivo de tornar o processo mais rápido e cómodo, apresentando para isso uma criteriosa análise de requisitos, seguida da sua modelação, e terminando com a implementação física da base de dados.

O propósito deste sistema é que a escola seja capaz de observar as diversas funcionalidades a que pode estar sujeita, como a listagem de aulas livres para que o aluno possa marcar, o carro e o instrutor a que está associado, aulas a decorrer no momento, entre outras.

1.3. Motivação e Objetivos

Com a crescente necessidade de armazenar dados, tornou-se inviável o tratamento manual da gestão de todas as necessidades que a escola de condução necessitava. O Sr. António sabia que durante muitos anos nas escolas de condução, na maior parte dos casos, era utilizado o papel que envolvia processos demorados e dispendiosos e que estavam associados a possíveis erros.

Este problema levou ao surgimento de um novo sistema de base de dados, que permitisse que o acesso aos dados fosse rápido, fácil e seguro.

Com a realização deste projeto, é ambicionado alcançar dois objetivos distintos: Objetivos de aprendizagem e Objetivos de estudo.

Estes objetivos tornam-se característicos pois, de uma forma, queremos enriquecer o nosso conhecimento em relação à modelação de bases de dados e, além disso, desenvolver críticas de análise. Em relação aos objetivos de estudo esperamos, de certa forma, aplicar os conceitos abordados nas aulas teóricas e práticas, permitindo a sua contextualização em desafios hipoteticamente reais e, ademais, a consolidação dos objetivos de conhecimento referidos.

1.4. Estrutura do Relatório

Na Estrutura do Relatório será descrito um breve resumo do que se irá tratar todo este documento, desde a sua contextualização até ao seu modelo físico.

Neste primeiro capítulo, são apresentados todos os aspetos gerais deste projecto prático, indo desde a sua contextualização, passando nos seus objetivos até à sua estrutura, de modo a que sintetizar o que vai ser encontrado nas próximas secções.

O segundo capítulo é referente ao levantamento de requisitos, identificando o método utilizado, os tipos de requisitos levantados e uma análise dos mesmos.

No terceiro capítulo, foi tratado de toda a metodologia para a construção do modelo conceptual, desde a identificação das entidades, dos relacionamentos, dos atributos, determinação das multiplicidades, apresentação das definições de chave primária e alternativa e por último a verificação e validação do Modelo Conceptual.

2. Levantamento e Análise de Requisitos

2.1. Método de levantamento e de análise de requisitos adotado

Em relação ao método para o levantamento dos requisitos necessários, decidimos que, uma forma para podermos agrupar todos os que dizem respeito a cada perfil de utilização, tendo assim um único conjunto de requisitos que correspondem à totalidade da base de dados. Assim, iremos recorrer a várias metodologias de *fact-finding* tais como:

- ➔ **Entrevista:** Foram realizadas várias entrevistas com a administração da escola de condução Drive IT para que se perceba as várias vistas/necessidades que estas pessoas têm sobre o novo sistema de base de dados e como é que estas interagem. Destas conversações retirou-se um conjunto de entidades requeridas.
- ➔ **Análise do Sistema de Informação:** Feita análise a todo o sistema de informação exigido, foi-nos possível recolher detalhes do sistema, tais como os tipos de dados armazenados, a maneira como se interligam e também a forma como serão importados os dados que já existem atualmente no sistema. Após este levantamento de requisitos, fomos estruturando todo o processo e apresentamos à administração da escola de condução, de forma a garantir que o problema apresentado é o que estes pretendiam ver resolvido.

De acordo com a pesquisa efetuada verificou-se que a base de dados deverá suportar os seguintes requisitos:

- ➔ A base de dados deve permitir guardar informações:
  - ◆ Nome, número de telemóvel, email, morada, número de CC e NIF sobre os alunos.
  - ◆ Nome, número de telemóvel, email, morada, número de CC e morada dos instrutores e funcionários.
  - ◆ Ano, marca e modelo do carro.
  - ◆ Data, hora, estado, instrutor, carro, funcionário e aluno das aulas.

- ➔ A base de dados deve permitir fazer marcações de aulas, tal como identificar diversos tipos de aulas (livres, marcadas, decorrer e realizadas).

- ➔ Cada aluno, instrutor e funcionário deve ter um número identificativo.

- ➔ A duração de cada aula é de exatamente de uma hora.

- ➔ As aulas decorrem entre as dez e as doze da manhã e entre as quinze e dezasseis da tarde.

- ➔ A escola pode possuir um número variável de instrutores e de funcionários.

- ➔ A base de dados deve permitir guardar todos os espaços de aulas futuras, quer estejam marcadas ou não, para determinados dias.

- ➔ A base de dados deve permitir aceder a uma lista de moradas a qualquer altura, para facilmente associar aos alunos, funcionários e instrutores.

- ➔ Consultar todas as aulas disponíveis para um certo dia.

- ➔ Consultar qual o carro utilizado numa aula.

2.2. Análise e validação geral dos requisitos

O sistema de base de dados a ser concebido deve ser utilizado na escola de condução, para gerir as marcações dos alunos, bem como a atribuição de aulas aos mesmos.

O funcionário deve ter permissão para adicionar/remover alunos. A pedido do aluno, o funcionário deve fazer uma ou mais marcações, na qual deve efetuar a inscrição do aluno, caso ainda não exista no sistema.

Posteriormente, o funcionário atribui uma aula a cada marcação, com o seu respetivo instrutor. Nessa aula é indicado o carro que se vai utilizar e o seu id de funcionário, para efeitos estatísticos e para futuro contacto caso exista algum problema ou conflito de marcações de aulas, mudando o estado da aula de livre para marcada .

Também, deverá ser possível, tanto para o funcionário como para o instrutor, atualizar uma aula que esteja a decorrer para realizada. Desta forma, irá permitir que o instrutor possa indicar que uma aula foi concluída sem ter de contactar um funcionário.

3. Metodologia para a construção do modelo conceptual

3.1. Apresentação da abordagem de modelação realizada

A fase da modelação conceptual é uma das fases mais importantes neste projecto, é sobre o resultado desta fase que vão assentar todos os desenvolvimentos futuros e por isso, qualquer irá refletir-se mais á frente, podendo levar à própria inviabilidade do projeto.

Para a realização do Modelo Conceptual são utilizadas técnicas de modelização de dados que permitem criar uma visão abstrata da estrutura da base de dados que suportará os dados reais, ou seja, tem como objetivo representar os dados presentes nos requisitos do utilizador. Aquando a construção deste modelo, é necessário levantar toda a informação sobre as entidades, relacionamentos e atributos e todas estas deverão estar devidamente documentadas. Por isso, que o Modelo Conceptual, reflete a perceção que os utilizadores têm dos dados e deve ser um processo de construção onde é fundamental haver várias fases de teste e validação contínuas para que fique assegurado que a futura base de dados seja coerente e que efetivamente resolva um problema

3.2. Identificação das Entidades

A partir da leitura dos requisitos, existem vários substantivos que se destacam, nomeadamente, funcionário, aluno, aula, instrutor, carro e morada.

Entidade	Descrição
Morada	Termo genérico que representa a localização onde alguém reside.
Funcionário	Termo genérico que representa o trabalhador que está a interagir com a base de dados.
Aluno	Termo genérico que representa a pessoa que frequenta a escola.

Aula	Termo que representa o leccionamento de uma aula pelo instrutor ao aluno.
Instrutor	Termo genérico que representa a pessoa que leciona uma aula.
Carro	Termo genérico que representa o veículo onde decorrem as aulas.

3.3 Identificação dos Relacionamentos

Tendo em conta o levantamento de requisitos realizado e as entidades previamente identificadas, foram selecionados alguns verbos considerados chave para descrever as relações entre entidades. Assim, obtiveram-se os seguintes relacionamentos:

- Aluno faz marcação de Aula com Funcionário
- Aula dada por Instrutor
- Aula dada com Carro
- Aluno reside em Morada
- Instrutor reside em Morada
- Funcionário reside em Morada

3.4. Determinação das Multiplicidades

Uma Aula apenas é dada por um Instrutor e dada num Carro, porém várias Aulas podem ser dadas por um Instrutor e dadas num Carro, daí temos que a relação Aula e Instrutor e Aula e Carro são relações de 1 para muitos.

Um Funcionário, Aluno ou Instrutor residem numa Morada e uma Morada pode ser de um ou mais Funcionários, Alunos ou Instrutores, devido ao facto de ser possível existirem entidades diferentes no sistema do mesmo agregado familiar, implicando terem a mesma morada, sendo assim estas relações são de 1 para muitos.

Entidade	Multiplicidade	Relacionamento	Multiplicidade	Entidade
Funcionário	N	reside em	1	Morada
Aluno	N	reside em	1	
Instrutor	N	reside em	1	
Aula	N	dada por	1	Instrutor
	N	dado no	1	Carro

Entidade	Multiplicidade	Relacionamento	Multiplicidade	Entidade	Multiplicidade	Entidade
Aluno	1	marcação	N	Aula	1	Funcionário

3.5. Identificação dos Atributos

Existem certos atributos que consideramos essenciais nas entidades mais importantes: Nome, Email, Número de telemóvel, Número de Cartão de Cidadão e Morada. As entidades Aluno, Funcionário e Instrutor possuem todas os atributos mencionados, realçando que a Morada **é opcional** no Funcionário e no Instrutor, **sendo obrigatório** no Aluno caso seja preciso ir buscar o Aluno à sua residência.

O funcionário e o Instrutor além dos atributos já mencionados devem ter também um identificador único como atributo, o id..

Os alunos devem deixar também mais uma informação aquando da inscrição na escola para ser guardada na base de dados, o NIF, que servirá para questões de pagamento e ainda é assinalado se efetuou o pagamento sendo assim preciso um atributo para guardar essa informação (‘S’ e ’N’ são as únicas opções possíveis).

Todas as aulas possuem uma data e hora como atributo e um estado que serve para classificar a aula podendo apenas assumir um carácter da lista seguinte:

- ‘L’ se a aula não estiver ainda marcada por ninguém.
- ‘R’ se estiver realizada;
- ‘M’ se estiver marcada;
- ‘D’ se estiver a decorrer.

O carro possui uma matrícula que atua como identificador e um atributo composto (Descrição), onde são mencionados o ano, modelo e a sua marca.

Por fim, a morada tem como atributos um identificador único, uma rua, localidade e código postal.

3.6. Definição dos Domínios dos Atributos

Todos os atributos foram definidos como VARCHAR, de tamanho variável dependendo do atributo em questão, com exceções da data que foi marcada como DATE, o ano que foi marcado como YEAR e hora como TIME, o estado e pagamento como ENUM.

3.7. Definição de Chaves Candidatas, Primárias e Alternativas

Esta verificação consiste em três passos, sendo o primeiro ignorado devido à ausência de relações 1 para 1 no nosso modelo.

No segundo, não existem também relações redundantes e por isso não é necessária a remoção de nenhuma relação do nosso modelo não havendo ciclos.

Por fim, no terceiro passo, como não ocorreu nenhuma alteração no segundo, não é necessária uma nova verificação do modelo, continuando com a inexistência de ciclos.

Para o caso da entidade Aluno, cinco atributos irão ter valor único, o ID, Email, número de telemóvel, número de cartão de cidadão e o número de identificação fiscal, visto que se podem registar duas pessoas com o mesmo Nome, o que nos dá as cinco chaves candidatas para o caso. Destas, o Email possui alguma probabilidade de ser alterado, uma pessoa pode atualizar o email com o qual se registou e o mesmo acontece com o número de telemóvel. Como tal, escolhemos o ID como chave primária em vez do número de cartão de cidadão e NIF por motivos práticos, deixando os outros quatro atributos únicos como chaves alternativas.

No caso do funcionário temos quatro atributos que irão ter valor único, o ID, Email, o número de telemóvel e o número de cartão de cidadão, a escolha foi feita tal como no Aluno, ficando o ID a chave primária e os números de telemóvel e NIF como chaves alternativas.

Em relação à entidade Aula, apenas temos o ID como chave candidata, pois data, hora e estado não são atributos únicos devido a possibilidade de existirem diferentes aulas a decorrer ao mesmo tempo, o que torna o ID a chave primária.

Sobre o Instrutor é também semelhante às outras entidades. Temos quatro chaves candidatas, o ID, o Email, o número de telemóvel e número de identificação fiscal, e novamente selecionamos o ID como chave primária, devido à possibilidade de alteração dos outros atributos, deixando assim o NIF como chave alternativa.

Relativamente à morada, temos apenas o ID como chave candidata, visto que rua, localidade e código-postal não são únicos para identificar uma morada, tornando-se assim chave primária.

Por fim, relativamente ao carro, temos apenas a matrícula como chave candidata e ,por isso chave primária, uma vez que podem existir carros do mesmo ano, modelo ou da mesma marca.

Entidade	**Atributo**	**Descrição**	**Domínio**	**Nulo**	**Derivado**	**Multivalorado**
Aluno	ID (Chave Primária)	Número Identificativo de cada Aluno registado no sistema	Inteiro	Não	Não	Não
	Nome	Nome do aluno	*String* até 45 caracteres	Não	Não	Não
	Email (Chave Alternativa)	Email do aluno, utilizado para contacto com o mesmo	*String* até 45 caracteres	Não	Não	Não
	Nrº_Telemovel (Chave Alternativa)	Número de telemóvel de cada aluno	*String* até 9 caracteres	Não	Não	Não
	Nrº_CC (Chave Alternativa)	Número do cartão de cidadão do aluno	*String* até 45 caracteres	Não	Não	Não

	NIF (Chave Alternativa)	Número de Identificação fiscal do aluno	9 caracteres	Não	Não	Não
	Pagamento	Indica se o aluno já efetuou o pagamento	*Enum* (‘S’,’N’)	Não	Não	Não
Morada	idMorada (Chave Primária)	Número Identificativo da Morada	Inteiro	Não	Não	Não
	Rua	Indica a Rua a que pertence	*String* até 45 caracteres	Não	Não	Não
	Localidade	Indica a Localidade a que está associada	*String* até 45 caracteres	Não	Não	Não
	Código-Postal	Código Postal que identifica a morada	*String* até 45 caracteres	Não	Não	Não
Funcionario	ID (Chave Primária)	Número identificativo do bilhete	Inteiro	Não	Não	Não
	Nome	Preço do bilhete	*String* até 45 caracteres	Não	Sim	Não
	Email (Chave Alternativa)	Classe do portador do bilhete	*String* até 45 caracteres	Não	Não	Não

	NrºTelemovel (Chave Alternativa)	Número de telemóvel de cada aluno	*String* até 9 caracteres	Não	Não	Não
	Nrº CC (Chave Alternativa)	Número do cartão de cidadão do aluno	*String* até 45 caracteres	Não	Não	Não
Aula	ID (Chave Primária)	Número Identificativo da aula	Inteiro	Não	Não	Não
	Data	Altura do começo da aula	Data	Não	Não	Não
	Hora	Tempo que começa a aula	Tempo	Não	Não	Não
	Estado	Define o estado em que a aula se encontra	*Enum* (‘L’,’M’,’D’,’R’)	Não	Não	Não
Instrutor	ID (Chave Primária)	Número identificativo do comboio	Inteiro	Não	Não	Não
	Email(Chave Alternativa)	Representa os lugares que o comboio disponibiliza	*String* até 45 caracteres	Não	Não	Sim

	Nrº_Telemóvel(Chave Alternativa)	Indica a capacidade máxima do comboio	*String* até 45 caracteres	Não	Não	Não
	NIF (Chave Alternativa)	Número de Identificação fiscal do aluno	S*tring* até 45 caracteres	Não	Não	Não
	Nome	Nome do aluno	*String* até 45 caracteres	Não	Não	Não
Carro	Matricula (Chave Primária)	Matrícula do carro	*String* até 45 caracteres	Não	Não	Não
	Marca	Marca do carro	*String* até 45 caracteres	Não	Não	Não
	Modelo	Modelo do carro	*String* até 45 caracteres	Não	Não	Não
	Ano	Ano do carro	*YEAR* com 4 números	Não	Não	Não

3.8. Verificação de Redundâncias no Modelo

Esta verificação consiste em três passos, sendo o primeiro ignorado devido à ausência de relações 1 para 1 no nosso modelo.

No segundo, não existem também relações redundantes e por isso não é necessária a remoção de nenhuma relação do nosso modelo não havendo ciclos.

Por fim, no terceiro passo, como não ocorreu nenhuma alteração no segundo, não é necessária uma nova verificação do modelo, continuando com a inexistência de ciclos.

3.9. Validação do Modelo Conceptual

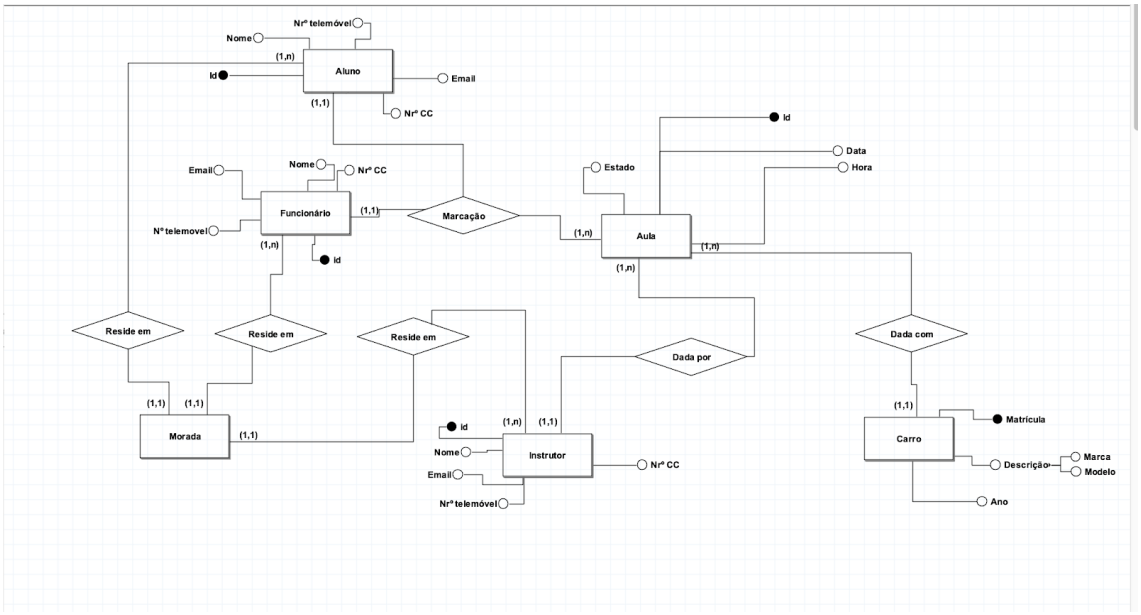
Tendo em conta que o sistema de base de dados a implementar pretende gerir as marcações e respetivas aulas de uma escola de condução, este deve conseguir executar operações fundamentais, de modo a permitir o seu bom funcionamento e o cumprimento dos seus objetivos.

Sendo assim, deve permitir visualizar o calendário de aulas, as aulas que já estão marcadas para um determinado dia e as aulas que estão livres podendo ser assim marcadas e ocupadas pelo aluno que a pretende frequentar, tornando o ato de marcação da aula como operação essencial também.

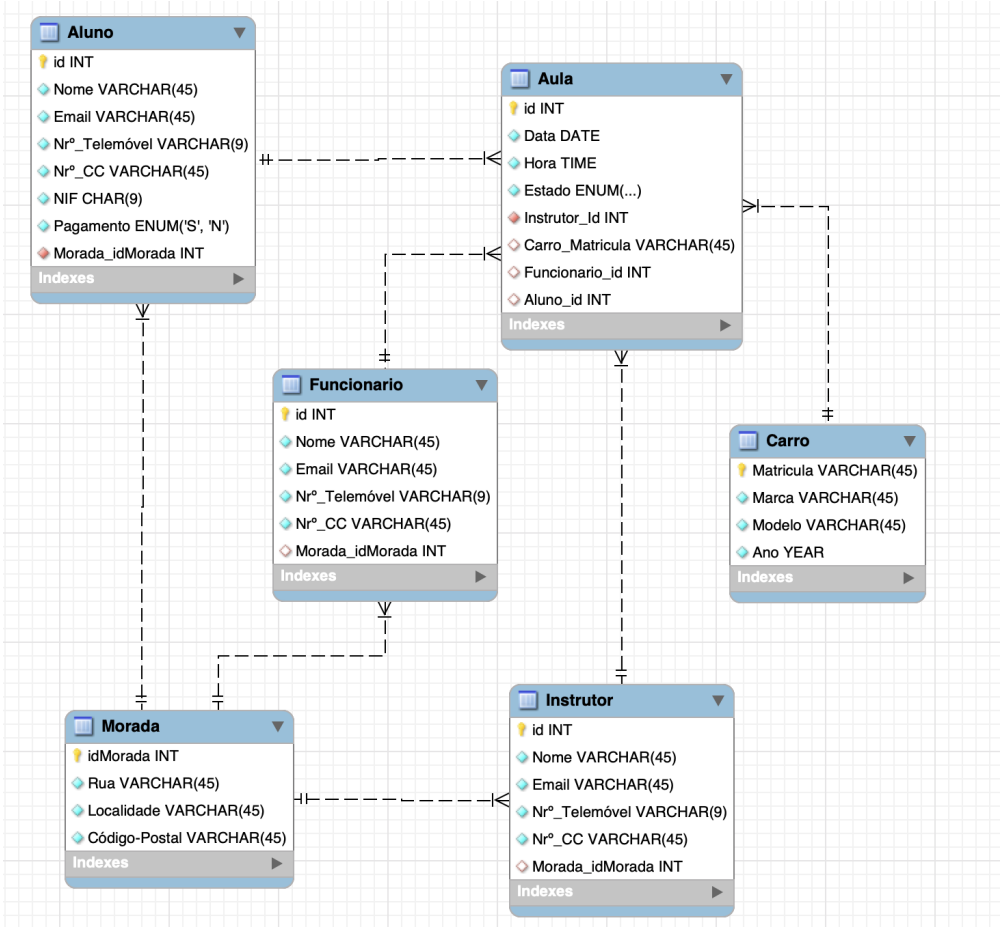
Através do atributo **estado** da Aula bem como a sua data e hora consegue-se determinar todo o calendário de aulas bem como as que já estão marcadas.

3.10. Apresentação e explicação do diagrama ER

Na imagem abaixo, é apresentado o Modelo Conceptual da nossa Base de Dados, nele conseguimos ver mais generalizadas todas as relações, entidades e atributos, fazendo assim um culminar de tudo o que foi apresentado e explicado durante este capítulo.



4. Construção e validação do modelo de dados lógico



Na imagem é apresentado o modelo Lógico, de acordo com a nossa ideia inicial. Neste modelo são apresentadas de forma mais concreta a forma como os atributos interagem com as entidades e são apresentadas pela primeira vez, o conceito de chave primária e chave estrangeira.

4.1 Chave Primária

A chave primária pode ser definida como a chave que numa coluna de uma tabela identifica de forma única a respetiva identidade. Esta não pode assumir nunca o valor NULL.

4.2. Chave Estrangeira

A chave estrangeira identifica a tabela que representa a relação entre duas tabelas. Esta permite garantir a integridade entre duas instâncias da mesma entidade. É também a chave primária da entidade associada.

4.3 Derivar relações a partir do modelo conceptual

A partir do modelo conceptual, conseguimos estruturar o modelo lógico da base de dados requerida para a escola de condução Drive IT. Este modelo tem como objetivo percepcionar melhor as relações entre as entidades e de alguma forma, apurar se a base de dados a criar é capaz de suportar todas as finalidades estabelecidas.

Para iniciar a construção do modelo lógico começa-se por derivar as relações presentes no modelo conceptual, de forma a apresentar as entidades, relações e atributos. Atribui-se um nome da relação, seguida pela lista dos seus atributos, identificando as chaves primárias e estrangeiras . Na transição do modelo conceptual para o modelo lógico começa-se por escolher as chaves primárias para cada entidade.

Em relacionamentos do tipo 1:N a chave primária da entidade “pai” (1), é utilizada como chave estrangeira na entidade “filho” (N). Por exemplo, na entidade Aluno a chave primária é o *ID* e na entidade aula é utilizado como chave estrangeira. Quando se trata de relacionamentos de 1:1, 0:0, 1:0,

existem três tipos de tratamentos. Se a relação é mandatória nos dois sentidos, faz-se a junção numa só entidade.

Os relacionamentos N:N originam uma nova entidade que inclui as chaves primárias de ambas as entidades.

Por fim, os atributos derivados, caso existam, o tratamento é feito como se de uma relação do tipo 1:N se tratasse.

Morada(**idMorada**, Rua,Localidade,Código-Postal) **Chave Primária:** idMorada
Funcionario(**id**, Nome, Email, Nrº_Telemóvel,Nrº_CC, Morada) **Chave Primária:** id **Chave Estrangeira:** Morada **referencia** Morada (idMorada)
Instrutor(**id**, Nome, Email, Nrº_Telemóvel, Nrº_CC,Morada) **Chave Primária:** id **Chave Estrangeira** Morada **referencia** Morada (idMorada)
Aluno(**id**, Nome, Email, Nrº_Telemóvel, Nrº_CC, NIF, Pagamento, Morada) **Chave Primária** id **Chave Estrangeira** Morada **referencia** Morada (idMorada)
Carro(**Matricula**, Marca, Modelo, Ano) **Chave Primária** Matricula
Aula(**id**, Data, Hora, Estado, Instrutor, Carro, Funcionario, Aluno) **Chave Primária** id **Chave Estrangeira** Instrutor **referencia** Instrutor(id)

Chave Estrangeira Carro **referencia** Carro(Matricula)
Chave Estrangeira Funcionário **referencia** Funcionario(id)
Chave Estrangeira Aluno **referencia** Aluno(id)

De salientar ainda, que a relação ternária **Marcação** foi incluída na relação aula, que contém o id do aluno que reservou a aula, bem como o id do funcionário que tratou a marcação, sendo assim a relação Aula contém toda a informação necessária sobre as aulas.

4.4 Entidade

Num modelo lógico as entidades são representadas por tabelas dentro das quais deve ser colocada a sua designação além dos seus atributos.

Uma entidade é um objeto ou um conceito que está claramente identificado numa organização como tendo uma existência independente.

Cada entidade deve representar uma única “coisa” presente no mundo real, caso fosse representada por diversas “coisas” não seria possível estabelecer as relações entre apenas uma parte da entidade com outra entidade qualquer do modelo.

A existência de entidades, por si só, não permite que a informação existente em cada uma delas se possa relacionar com a informação presente noutras entidades para permitir esta conexão de informação são estabelecidas relações/relacionamentos.

4.5. Relacionamentos

A nossa base de dados contém 7 relacionamentos de 1 para N.

Este relacionamento significa que 1 entidade está relacionada com N elementos de outra entidade. Posto isto, exemplos destes relacionamentos de 1:N, na nossa base de dados da escola de condução são:

Entidade	Descrição
Morada: Instrutor	Um Instrutor tem opção de ter uma morada. Uma Morada poderá ter vários instrutores
Morada: Funcionário	Um Funcionário tem opção de ter uma morada. Uma Morada poderá ter vários funcionários.
Morada: Aluno	Um Aluno obrigatoriamente tem uma morada. Uma Morada poderá ter vários alunos.
Funcionário: Aula	Uma Aula pode ser marcada por um Funcionário. Um funcionário poderá marcar várias aulas.
Aluno: Aula	Uma Aula pode estar associada a um Aluno. Um Aluno poderá estar associado a várias aulas.
Instrutor: Aula*	Uma Aula está obrigatoriamente associada a um Instrutor. Um Instrutor poderá lecionar várias aulas.
Carro: Aula	Uma Aula pode estar associada a um carro. Um Carro poderá ter sido utilizado em várias aulas.

*Neste ponto da tabela, achamos bem explicar esta associação entre um Instrutor e uma Aula. Os estados de uma Aula são diversos e, independentemente destes estados, um instrutor estará associado obrigatoriamente a uma Aula. Por exemplo, no momento do registo de uma aula, seja ela como livre, um Instrutor estará imediatamente associado. Isto irá nos permitir que seja verificado quais os horários de aulas disponíveis para cada instrutor.

4.6. Validar relações usando a normalização

O modelo lógico encontra-se normalizado até à segunda forma normal. Um exemplo do uso desta regra é nas três tabelas Instrutor, Aluno e Funcionário. Todos os atributos não primários destas tabelas devem ser (totalmente) funcionalmente dependentes da chave primária dessa tabela.

4.7. Validar relações com interrogações do utilizador

De forma a deixar a nossa base de dados viável, foram realizadas diversas queries de forma a que seja possível permitir obter respostas de interrogações feitas pelo utilizador à base de dados.

A seguir, mostramos alguns exemplos desenvolvidos e iremos falar um bocado sobre a lógica por trás do desenvolvimento delas.

- ***Exemplo de interrogação número 1:*** Obter toda a informação do registo das aulas na base de dados.

```
-- Querie 0: Lista de todas as aulas
Select * From aula;
```

Trata-se de uma interrogação direta ao sistema de base de dados, isto é, apenas se verifica a utilização de um operador de projeção.

- ***Exemplo de interrogação número 2:*** Obter a informação do registo das aulas em estado livre da base de dados.

```
-- Querie 1: Lista de slots das aulas livres com info relevante, sem identificar o nome do instrutor
Select Data, Hora, Estado, Instrutor_id AS `ID do Instrutor` From aula
    where Estado IN ('L');
```

Trata-se de uma interrogação direta ao sistema de base de dados, isto é, apenas se verifica a utilização de um operador de projeção. Porém, é utilizado uma filtragem dessa projeção dos dados, isto é, é feita uma seleção dos dados projetados (neste caso pelo estado da aula).

- ***Exemplo de interrogação número 3:*** Obter a informação do registo das aulas em estado livre da base de dados, identificando o nome do instrutor.

```
-- Querie 2: Lista de slots das aulas livres com info relevante, indentificando o nome do instrutor
Select Instrutor_id AS `ID do Instrutor`, Instrutor.Nome AS `Nome do Instrutor`, Data, Hora, Estado  From aula
    JOIN Instrutor ON Instrutor.id = aula.Instrutor_id
        where Estado IN ('L');
```

A diferença desta interrogação, comparada com as anteriores, é que de facto é necessário informação que está em duas tabelas distintas. Para isso, foi necessário utilizar um operador de junção. Assim, é utilizado operadores de projeção, de seleção e de junção.

- ***Exemplo de interrogação número 4:*** Obter a informação do registo das aulas marcadas na base de dados, identificando o nome do aluno, ordenados ascendentemente, e a sua morada.

```
-- Querie 3: Lista de aulas marcadas por Aluno, ordenados ascendentemente pelo nome do aluno (mostra morada também)
Select aula.id AS `ID da Aula`, Aluno.id AS `ID do Aluno`, Aluno.Nome, aula.Data, aula.Hora,
    aula.Carro_Matricula AS `Matrícula do Carro`, aula.Instrutor_id AS `ID do Instrutor`, Morada.Rua, Morada.Localidade,
    Morada.`Código-Postal` FROM Aluno
    JOIN aula ON Aluno.id = aula.Aluno_id
    JOIN Morada ON Morada.idMorada = Aluno.Morada_idMorada
        order by Aluno.Nome ASC;
```

Mais uma vez, verifica-se a necessidade de obter informações proveniente de múltiplas tabelas (de Aluno, de Aula e da Morada). Deste modo, de forma semelhante, é utilizado dois operadores de junção, um de projeção e neste caso, um de ordenação.

- ***Exemplo de interrogação número 5:*** Obter um ranking de interações de funcionários (sempre que marca uma aula, o funcionário é relacionado a essa marcação).

```
-- Querie 7: Ranking de funcionário, consoante as aulas com que interagiu (exclui-se as que estão livres) por ordem
-- decrescente
Select Nome, Email, Nrº_Telemóvel AS `Nº de Telemóvel`, count(Funcionario.id) AS `Total de Aulas` From Funcionario
    JOIN aula ON aula.Funcionario_id = Funcionario.id
        where aula.Estado IN ('M','D','R')
        group by Funcionario.id
        order by `Total de Aulas` DESC;
```

Para obter esta informação, além dos operadores relacionais demonstrados anteriormente, também foi necessário utilizar um operador de agrupamento e uma função para contabilizar esse agrupamento.

- ***Exemplo de interrogação número 6:*** Atualizar o estado de uma aula para concluída (uma aula que está a decorrer e que acaba).

```
-- Querie 14: Exemplo de uma atualização de uma aula, passar para 'realizada'. Só passa para realizada se o estado
-- for a decorrer e os dados introduzidos derem match
-- Se for preciso apagar a procedure:
DROP PROCEDURE update_aula_realizada;
delimiter //
CREATE PROCEDURE update_aula_realizada (IN idInstrutor INT, idAula INT)
        BEGIN
         Update aula
             Set Estado = 'R'
             -- é dado o estado da aula, id do Instrutor, o dia e a hora da aula
             where Estado = 'D' AND aula.id = idAula;
        END//
delimiter ;
```

Para generalizar esta interrogação, foram utilizados procedimentos armazenados. O procedimento irá receber dados de entrada (neste caso, o identificador do instrutor e o identificador da aula) e através desses dados, realizar uma atualização dessa aula, utilizando um operador de seleção.

- ***Exemplo de interrogação número 7:*** Cancelamento de uma aula.

```
-- Querie 15: Exemplo de um cancelamento de uma aula, passar para 'livre'. Só passa para livre se o estado
-- for de marcada e os dados introduzidos derem match
-- Se for preciso apagar a procedure:
DROP PROCEDURE cancelar_aula;
delimiter //
CREATE PROCEDURE cancelar_aula (IN idFuncionario INT, idAula INT)
       BEGIN
        Update aula
            -- atualiza slot para Livre
            -- atualiza a matricula do carro para null
            -- apaga o funcionário
            -- apaga o id do aluno
            Set Estado = 'L', Carro_Matricula = null, Funcionario_id = null, Aluno_id = null
            -- Aluno_id = 0,
                -- é dado o estado da aula, id do Funcionario (só funcionários é que cancelam aulas) e o id da aula
                where aula.Estado = 'M' AND aula.id = idAula AND aula.Funcionario_id = idFuncionario;
       END//
delimiter ;
```

Comparando com a interrogação interior, trata-se de um procedimento que segue exatamente a mesma estruturação.

- ***Exemplo de interrogação número 8:*** Marcação de uma aula.

```
-- Querie 16: Exemplo de uma marcação de aula, passar para 'marcada'. Só passa para marcada se o estado
-- for a livre e os dados introduzidos derem match
-- Se for preciso apagar a procedure:
DROP PROCEDURE marcar_aula;
delimiter //
CREATE PROCEDURE marcar_aula (IN idAula INT, idFuncionario INT, idAluno INT, matricula VARCHAR(45))
       BEGIN
        Update aula
            Set Estado = 'M', `Carro_Matricula` = matricula, Funcionario_Id = idFuncionario, Aluno_id = idAluno
            -- é dado o id do aluno, o dia, hora da aula e confirmar que se tratava de uma aula a decorrer
            where Estado = 'L' AND aula.id = idAula;
       END//
delimiter ;
```

Mais uma vez, o processo é o mesmo que as duas interrogações interiores. Como se tratam de interrogações bastante importantes no nosso sistema, achamos por bem demonstrar estes três procedimentos anteriores.

- ***Exemplo de interrogação número 9:*** Marcação de todos os slots para o dia de trabalho de um instrutor.

```
-- Querie 17: Marcar slots de todas as aulas de um instrutor para um dia
DROP PROCEDURE marcar_slots_aula;
delimiter //
CREATE PROCEDURE marcar_slots_aula (IN idInstrutor INT, dataC DATE)
	BEGIN
	 INSERT INTO aula (`Data`,`Hora`,`Estado`,`Instrutor_Id`)
		VALUES
			(dataC,"10:00:00","L",idInstrutor),
			(dataC,"11:00:00","L",idInstrutor),
			(dataC,"15:00:00","L",idInstrutor),
			(dataC,"16:00:00","L",idInstrutor),
			(dataC,"17:00:00","L",idInstrutor);
	END//
delimiter ;
```

Esta interrogação foi realizada para procurar automatizar este processo. Uma vez que será necessário criar slots de aulas livres para um dia de cada instrutor, foi criado este procedimento que recebe o identificador do instrutor e um certo dia. Assim, irão ser inseridas as 5 horas pré definidas de um dia de trabalho de um instrutor.

- ***Exemplo de interrogação número 10:*** Verificar todas as aulas no dia atual

```
-- Querie 18: Verificar todas as aula no dia atual (utilizando views)
CREATE VIEW vwAulasHoje AS
Select * from aula
    where Data = date(now());

Select * from vwAulasHoje;
```

Mais uma vez, foi procurado utilizar um processo que seja mais conveniente para este tipo de tarefas pedidas pelo utilizador. Para esse fim, foi utilizado a criação de uma vista. A vista irá chamar um operador de projeção filtrado por um operador de seleção.

- ***Exemplo de interrogação número 11:*** Verificar o nome de um instrutor dado o id dele

```
-- Querie 19: Retorna o nome de um instrutor dado o id dele (FUNCTION)
DROP FUNCTION nomeInstrutor;
delimiter //
CREATE FUNCTION `nomeInstrutor` (idInstrutor int)
    returns VARCHAR(45)
    READS SQL DATA
    DETERMINISTIC
    Begin
        declare nomeInstrutor VARCHAR(45);
        set nomeInstrutor = (select Nome from Instrutor where id = idInstrutor);
        return nomeInstrutor;
    End//
```

Como numa aula um instrutor está identificado por um valor numérico, achamos conveniente utilizar um processo que dado um identificador numérico de um instrutor, irá retornar o seu nome. Para tal, foi escolhido o método de criar funções.
É necessário declarar uma variável local (*'nomeInstrutor'*) e alterar o seu valor. Para alterar o seu valor, são utilizados um operador de projeção e um de seleção, retornando, assim, a informação requerida.

Foram realizadas mais interrogações para que seja possível o utilizador aceder a mais dados relevantes. Se necessário, poderá verificar as restantes interrogações no *'anexo 2'*.

4.8.Verificar integridade das restrições

4.8.1 Integridade referencial

De facto, as nossas tabelas estão relacionadas entre si, sendo por isso necessário que, ao fazer-se uma nova inserção numa tabela, que os dados referentes a outras tabelas já existam nessas mesmas tabelas.
Seja o caso de marcar uma aula: esta terá de referenciar diversos ID's de várias tabelas correspondentes às diferentes entidades relacionadas e será necessário que estas também existam nas respectivas tabelas.
Do mesmo modo, quando se atualizam ID's que são referenciados, é também importante que haja consistência referencial.

4.9. Rever modelo lógico com utilizador

O objetivo principal deste modelo é permitir efetuar marcação de aulas. Como se pode verificar em 4.3, é possível fazê-lo utilizando todas as entidades presentes neste modelo. Ou seja, o nosso modelo cumpre aquilo que lhe é pedido.

4.10. Verificar futuro crescimento

O modelo lógico criado para o cumprimento dos requisitos essenciais no nosso caso, encontra-se preparado para um eventual crescimento no futuro, visto que é capaz de suportar alterações futuras, como por exemplo um aumento do número de alunos, funcionários, carros e instrutores.

Acrescentando a estes aspectos, o nosso modelo lógico conseguirá também suportar a introdução de aulas teóricas visto que no nosso caso apenas abordamos as aulas práticas, conseguirá também propor alunos a exames depois de X aulas frequentadas.

Além disso, o sistema está normalizado e sem redundâncias, facilitando a manutenção e desenvolvimento futuros.

5. Metodologia para construção do modelo físico

5.1. Seleção do Sistema de Gestão de Bases de Dados

Depois de termos o modelo de dados especificado, já podemos fazer a implementação física da base de dados com a estrutura especificada no modelo entidade associação e pelo modelo de dados respetivo.

Teremos de selecionar uma das Bases de Dados ou, mais corretamente, um dos Sistemas Gestores de Bases de Dados existentes no mercado e criar aí uma base de dados para suportar estes dados. A criação de uma Base de Dados pode ser feita através de utilitários que alguns sistemas gestores de base de dados disponibilizam para facilitar a execução deste procedimento por parte dos utilizadores ou através de um comando.Ora esse comando é um dos muitos que fazem parte da linguagem SQL (CREATE DATABASE).

SQL é uma linguagem de alto nível, pois apenas dizemos ao sistema o que fazer e não lhe dizemos nem como, nem onde o tem que fazer.

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

Tendo em vista um melhor aproveitamento dos mecanismos que o MySQL dispõe, o nosso grupo de trabalho decidiu que ao longo de todo o processo de desenho, desenvolvimento e administração de base dados, iria ser utilizado o MySQL Workbench. Após a criação do modelo lógico, usamos a ferramenta de Forward Engineering, tal como o seu nome indica, permite a realização do processo top-down de construção de componentes de baixo nível, neste caso a implementação de um sistema de gestão de base dados, com base numa abstração de alto nível que é o nosso modelo lógico. Depois de termos realizado este processo, bastou-nos executar o código obtido para que a nossa base de dados fosse criada.

Morada

Domínio ID	Inteiro
Domínio da Rua	Variável conjunto de caracteres únicos com tamanho
Domínio da Localidade	Variável conjunto de caracteres com tamanho 45
Domínio do Código-Postal	Variável conjunto de caracteres com tamanho 45

```
Morada(
  `idMorada` INT NOT NULL AUTO_INCREMENT,
  `Rua` VARCHAR(45) NOT NULL,
  `Localidade` VARCHAR(45) NOT NULL,
  `Código-Postal` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`idMorada`))

);
```

Funcionario

Domínio ID	Inteiro
Domínio Nome	Variável conjunto de caracteres com tamanho 45
Domínio Email	Variável conjunto de caracteres com tamanho 45
Domínio Nrº_Telemóvel	Variável conjunto de caracteres com tamanho 9

Domínio Nrº_CC	Variável conjunto de caracteres com tamanho 45

```
`Funcionario` (
  `id` INT NOT NULL AUTO_INCREMENT,
  `Nome` VARCHAR(45) NOT NULL,
  `Email` VARCHAR(45) NOT NULL,
  `Nrº_Telemóvel` VARCHAR(9) NOT NULL,
  `Nrº_CC` VARCHAR(45) NOT NULL,
  `Morada_idMorada` INT NULL,
  PRIMARY KEY (`id`),
  INDEX `fk_Funcionario_Morada1_idx` (`Morada_idMorada` ASC) VISIBLE,
  CONSTRAINT `fk_Funcionario_Morada1`
        FOREIGN KEY (`Morada_idMorada`)
        REFERENCES `driveIt`.`Morada` (`idMorada`)
        ON DELETE NO ACTION
        ON UPDATE NO ACTION
);
```

Instrutor

Domínio ID	Inteiro
Domínio Nome	Variável conjunto de caracteres com tamanho 45
Domínio Email	Variável conjunto de caracteres com tamanho 45
Domínio Nrº_Telemóvel	Variável conjunto de caracteres com tamanho 9
Domínio Nrº_CC	Variável conjunto de caracteres com tamanho 45
Domínio Morada_idMorada	Inteiro

```
`Instrutor` (
  `id` INT NOT NULL AUTO_INCREMENT,
  `Nome` VARCHAR(45) NOT NULL,
  `Email` VARCHAR(45) NOT NULL,
  `Nrº_Telemóvel` VARCHAR(9) NOT NULL,
  `Nrº_CC` VARCHAR(45) NOT NULL,
  `Morada_idMorada` INT NULL,
  PRIMARY KEY (`id`),
  INDEX `fk_Instrutor_Morada1_idx` (`Morada_idMorada` ASC) VISIBLE,
  CONSTRAINT `fk_Instrutor_Morada1`
        FOREIGN KEY (`Morada_idMorada`)
        REFERENCES `driveIt`.`Morada` (`idMorada`)
        ON DELETE NO ACTION
        ON UPDATE NO ACTION
);
```

Aluno

Domínio ID	Inteiro
Domínio Nome	Variável conjunto de caracteres com tamanho 45
Domínio Email	Variável conjunto de caracteres com tamanho 45
Domínio Nrº_Telemóvel	Variável conjunto de caracteres com tamanho 9
Domínio Nrº_CC	Variável conjunto de caracteres com tamanho 45
Domínio NIF	Inteiro
Domínio Pagamento	Apenas caracteres entre ‘S’ ou ’N’
Domínio Morada_idMorada	Inteiro

```
`Aluno` (
  `id` INT NOT NULL AUTO_INCREMENT,
  `Nome` VARCHAR(45) NOT NULL,
  `Email` VARCHAR(45) NOT NULL,
  `Nrº_Telemóvel` VARCHAR(9) NOT NULL,
  `Nrº_CC` VARCHAR(45) NOT NULL,
  `NIF` CHAR(9) NOT NULL,
  `Pagamento` ENUM('S', 'N') NOT NULL,
  `Morada_idMorada` INT NOT NULL,
  PRIMARY KEY (`id`),
  INDEX `fk_Aluno_Morada1_idx` (`Morada_idMorada` ASC) VISIBLE,
  CONSTRAINT `fk_Aluno_Morada1`
        FOREIGN KEY (`Morada_idMorada`)
        REFERENCES `driveIt`.`Morada` (`idMorada`)
        ON DELETE NO ACTION
        ON UPDATE NO ACTION
);
```

Carro

Domínio Matricula	Variável conjunto de caracteres com tamanho 45
Domínio Marca	Variável conjunto de caracteres com tamanho 45
Domínio Modelo	Variável conjunto de caracteres com tamanho 45
Domínio Ano	Ano

```
`Carro` (
  `Matricula` VARCHAR(45) NOT NULL,
  `Marca` VARCHAR(45) NOT NULL,
  `Modelo` VARCHAR(45) NOT NULL,
  `Ano` YEAR(4) NOT NULL,
  PRIMARY KEY (`Matricula`)
);
```

Aula

Domínio ID	Inteiro
Domínio Data	Data
Domínio Hora	Time
Domínio Estado	Caracteres, entre 'L', 'R', 'M', 'D'
Domínio Instrutor_id	Inteiro
Domínio Carro_Matricula	Variável conjunto de caracteres com tamanho 45
Domínio Funcionario_id	Inteiro
Domínio Aluno_id	Inteiro

```
 `Aula` (
`id` INT NOT NULL AUTO_INCREMENT,
`Data` DATE NOT NULL,
`Hora` TIME NOT NULL,
-- L : livre; R: realizada; M: marcada; D : decorrer
`Estado` ENUM('L', 'R', 'M', 'D') NOT NULL,
`Instrutor_Id` INT DEFAULT 0 NOT NULL,
`Carro_Matricula` VARCHAR(45) NULL,
`Funcionario_id` INT NULL,
`Aluno_id` INT NULL,
PRIMARY KEY (`id`),
INDEX `Instrutor_Id_idx` (`Instrutor_Id` ASC) VISIBLE,
INDEX `fk_Aula_Carro1_idx` (`Carro_Matricula` ASC) VISIBLE,
INDEX `fk_Aula_Funcionario1_idx` (`Funcionario_id` ASC) VISIBLE,
INDEX `fk_Aula_Aluno1_idx` (`Aluno_id` ASC) VISIBLE,
CONSTRAINT `Instrutor_Id`
      FOREIGN KEY (`Instrutor_Id`)
      REFERENCES `driveIt`.`Instrutor` (`id`)
      ON DELETE NO ACTION
      ON UPDATE NO ACTION,
CONSTRAINT `fk_Aula_Carro1`
```

```
        FOREIGN KEY (`Carro_Matricula`)
        REFERENCES `driveIt`.`Carro` (`Matricula`)
        ON DELETE NO ACTION
        ON UPDATE NO ACTION,
  CONSTRAINT `fk_Aula_Funcionario1`
        FOREIGN KEY (`Funcionario_id`)
        REFERENCES `driveIt`.`Funcionario` (`id`)
        ON DELETE NO ACTION
        ON UPDATE NO ACTION,
  CONSTRAINT `fk_Aula_Aluno1`
        FOREIGN KEY (`Aluno_id`)
        REFERENCES `driveIt`.`Aluno` (`id`)
        ON DELETE NO ACTION
        ON UPDATE NO ACTION
);
```

5.3. Escolha, definição e caracterização de índices em SQL

Quando se trabalha diariamente com um sistema de base de dados é necessária a consulta de tabelas com grandes quantidades de registos. Com vista a tornar estas consultas mais rápidas e eficientes é aconselhada a criação de índices. Com a sua utilização passa a ser possível o uso de procuras binárias que permitem reduzir de forma drástica o tempo de consulta em tabelas com muitas colunas e muitos registos. No entanto, a utilização desta estratégia aumenta a quantidade de memória gasta, é por isso, importante estudar bem o caso de estudo para tomar uma decisão informada.

Além disso, as chaves estrangeiras utilizam-se como indexação para facilitar futuras junções.Tendo em conta que temos queries de consulta e de estatísticas em que vamos buscar informação sobre as diversas entidades onde é apresentada a sua morada, acreditamos que seja uma mais valia a criação do índice Morada.

Todos os índices criados estão presentes nas tabelas apresentadas no ponto anterior.

5.4. Estimativa de memória gasta

Utilizando como exemplo a povoação da base de dados, em que a escola de condução tem 5 instrutores, 5 funcionários, 40 alunos inscritos, 50 moradas, 10 carros , 100 aulas livres e 75 aulas marcadas. Em média por dia existem 5 novos alunos e são criados todos os novos espaços de aulas livres para todos os instrutores, sendo 5 instrutores com 5 horários para o início das aulas, são 25 as aulas livres criadas.

Tabela Morada

Atributo	Total em Bytes
ID (INT)	4
Rua(VARCHAR(45))	45
Localidade (VARCHAR(45))	45
Código-postal(VARCHAR(45))	45
Total por linha	139 Bytes

Tabela Funcionario

Atributo	Total em Bytes
ID (INT)	4
Nome(VARCHAR(45))	45
Email (VARCHAR(45))	45
Nrº_Telemovel(VARCHAR(9))	9
Nrº_CC(VARCHAR(45))	45
Morada(INT)	4
Total por linha	152 Bytes

Tabela Aluno

Atributo	Total em Bytes
ID (INT)	4
Nome(VARCHAR(45))	45
Email (VARCHAR(45))	45
Nrº_Telemovel(VARCHAR(9))	9
Nrº_CC(VARCHAR(45))	45
NIF(CHAR(9))	9
Pagamento(ENUM('S,'N'))	1
Morada(INT)	4
Total por linha	162 Bytes

Tabela Instrutor

Atributo	Total em Bytes
ID (INT)	4
Nome(VARCHAR(45))	45
Email (VARCHAR(45))	45
Nrº_Telemovel(VARCHAR(9))	9
Nrº_CC(VARCHAR(45))	45
Morada(INT)	4
Total por linha	152 Bytes

Tabela Carro

Atributo	Total em Bytes
Matricula(VARCHAR(45))	45
Marca(VARCHAR(45))	45
Modelo(VARCHAR(45))	45
Ano(YEAR(4))	4
Total por linha	139 Bytes

Tabela Aula

Atributo	Total em Bytes
ID (INT)	4
Data(DATE)	3
Hora(TIME(7))	5
Estado(ENUM('L', 'R', 'M', 'D'))	1
Instrutor(INT)	4
Carro_Matricula(VARCHAR(45))	45
Funcionario(INT)	4
Aluno(INT)	4
Total por linha	70 Bytes

Nas condições indicadas anteriormente, a base de dados diariamente aumenta o gasto de memória em:

162*5(5 novos alunos)+ 70*25(espaços de aulas)= 2560 bytes (média diária)

De acordo com a povoação realizada pelo grupo a base de dados ocupa:

Total estimado de memória = 152*5(instrutores) + 152*5(funcionários) + 162*40 (alunos) + 139*50(moradas), 139*10(carros) + 70*175(aulas) = 28590 bytes =28.590 Kb=0.028590 Mb.

6. Conclusões e Trabalho Futuro

Durante a execução deste trabalho conseguimos colocar em prática os conhecimentos adquiridos durante as aulas da unidade Curricular de Bases de Dados, porém num contexto específico do mundo real.

Na fase de definição do caso de estudo, a inclusão de aulas teóricas além das práticas foi também ponderado, contudo optamos por não abordar essa questão devido ao aumento da complexidade que isso traria ao sistema.

Durante a análise das interrogações ao sistema, foi dada mais ênfase à criação de uma slot de aula livre e a sua respetiva ocupação por um aluno que a pretende frequentar, visto nós considerarmos essas como sendo das mais relevantes, tendo em conta o modelo que desenvolvemos.

Posto isto, fazemos uma avaliação positiva do nosso desempenho, tendo – na medida das nossas capacidades – feito a melhor base de dados possível, tendo sempre em vista o caso de estudo que definimos e os requisitos que foram sendo levantados. Contudo, admitimos que numa aplicação de um sistema de base de dados no mundo real, existiriam outras exigências, tais como, a possibilidade de resposta num curto período de tempo a *queries* que manipulam um enorme volume de dados e, como já foi referido, a inclusão de aulas teóricas.

7.Referências

- Damas,L.(2005), "*SQL Structured Query Language",* 13ª Edição, FCA-Editora de Informática, Lisboa

- Universidade de Lisboa(2016/2017) "*Fact-finding - Resumo Bases de Dados*",https://www.studocu.com/pt/document/universidade-nova-de-lisboa/bases-de-dados/resumos/fact-finding-resumo-bases-de-dados/2058192/view (acedido em 30 Novembro 2020)

I. Anexo 1: Modelo em Linguagem SQL

```sql
-- 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,NO_ZERO_IN_DATE,NO_ZERO_DATE,ERROR_FOR_DIVISION_BY_ZERO,NO_ENGINE_SUBSTITUTION';

-- -----------------------------------------------------
-- Schema driveIt
-- -----------------------------------------------------

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

-- -----------------------------------------------------
-- Table `driveIt`.`Morada`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `driveIt`.`Morada` ;
CREATE TABLE IF NOT EXISTS `driveIt`.`Morada` (
  `idMorada` INT NOT NULL AUTO_INCREMENT,
  `Rua` VARCHAR(45) NOT NULL,
  `Localidade` VARCHAR(45) NOT NULL,
  `Código-Postal` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`idMorada`))
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `driveIt`.`Funcionario`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `driveIt`.`Funcionario` ;
CREATE TABLE IF NOT EXISTS `driveIt`.`Funcionario` (
  `id` INT NOT NULL AUTO_INCREMENT,
  `Nome` VARCHAR(45) NOT NULL,
  `Email` VARCHAR(45) NOT NULL,
  `Nrº_Telemóvel` VARCHAR(9) NOT NULL,
  `Nrº_CC` VARCHAR(45) NOT NULL,
  `Morada_idMorada` INT NULL,
  PRIMARY KEY (`id`),
  INDEX `fk_Funcionario_Morada1_idx` (`Morada_idMorada` ASC) VISIBLE,
  CONSTRAINT `fk_Funcionario_Morada1`
    FOREIGN KEY (`Morada_idMorada`)
    REFERENCES `driveIt`.`Morada` (`idMorada`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `driveIt`.`Aluno`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `driveIt`.`Aluno` ;
CREATE TABLE IF NOT EXISTS `driveIt`.`Aluno` (
  `id` INT NOT NULL AUTO_INCREMENT,
  `Nome` VARCHAR(45) NOT NULL,
  `Email` VARCHAR(45) NOT NULL,
  `Nrº_Telemóvel` VARCHAR(9) NOT NULL,
  `Nrº_CC` VARCHAR(45) NOT NULL,
  `NIF` CHAR(9) NOT NULL,
  `Pagamento` ENUM('S', 'N') NOT NULL,
  `Morada_idMorada` INT NOT NULL,
  PRIMARY KEY (`id`),
  INDEX `fk_Aluno_Morada1_idx` (`Morada_idMorada` ASC) VISIBLE,
  CONSTRAINT `fk_Aluno_Morada1`
    FOREIGN KEY (`Morada_idMorada`)
    REFERENCES `driveIt`.`Morada` (`idMorada`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `driveIt`.`Instrutor`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `driveIt`.`Instrutor` ;
CREATE TABLE IF NOT EXISTS `driveIt`.`Instrutor` (
  `id` INT NOT NULL AUTO_INCREMENT,
  `Nome` VARCHAR(45) NOT NULL,
  `Email` VARCHAR(45) NOT NULL,
  `Nrº_Telemóvel` VARCHAR(9) NOT NULL,
  `Nrº_CC` VARCHAR(45) NOT NULL,
  `Morada_idMorada` INT NULL,
  PRIMARY KEY (`id`),
  INDEX `fk_Instrutor_Morada1_idx` (`Morada_idMorada` ASC) VISIBLE,
  CONSTRAINT `fk_Instrutor_Morada1`
    FOREIGN KEY (`Morada_idMorada`)
    REFERENCES `driveIt`.`Morada` (`idMorada`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;

-- -----------------------------------------------------
-- Table `driveIt`.`Carro`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `driveIt`.`Carro` ;
CREATE TABLE IF NOT EXISTS `driveIt`.`Carro` (
  `Matricula` VARCHAR(45) NOT NULL,
  `Marca` VARCHAR(45) NOT NULL,
  `Modelo` VARCHAR(45) NOT NULL,
  `Ano` YEAR(4) NOT NULL,
  PRIMARY KEY (`Matricula`))
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `driveIt`.`Aula`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `driveIt`.`Aula` ;
CREATE TABLE IF NOT EXISTS `driveIt`.`Aula` (
  `id` INT NOT NULL AUTO_INCREMENT,
  `Data` DATE NOT NULL,
  `Hora` TIME NOT NULL,
  -- L : livre; R: realizada; M: marcada; D : decorrer
  `Estado` ENUM('L', 'R', 'M', 'D') NOT NULL,
  `Instrutor_Id` INT DEFAULT 0 NOT NULL,
  `Carro_Matricula` VARCHAR(45) NULL,
  `Funcionario_id` INT NULL,
  `Aluno_id` INT NULL,
  PRIMARY KEY (`id`),
  INDEX `Instrutor_Id_idx` (`Instrutor_Id` ASC) VISIBLE,
  INDEX `fk_Aula_Carro1_idx` (`Carro_Matricula` ASC) VISIBLE,
  INDEX `fk_Aula_Funcionario1_idx` (`Funcionario_id` ASC) VISIBLE,
  INDEX `fk_Aula_Aluno1_idx` (`Aluno_id` ASC) VISIBLE,
  CONSTRAINT `Instrutor_Id`
    FOREIGN KEY (`Instrutor_Id`)
    REFERENCES `driveIt`.`Instrutor` (`id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_Aula_Carro1`
    FOREIGN KEY (`Carro_Matricula`)
    REFERENCES `driveIt`.`Carro` (`Matricula`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_Aula_Funcionario1`
    FOREIGN KEY (`Funcionario_id`)
    REFERENCES `driveIt`.`Funcionario` (`id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_Aula_Aluno1`
    FOREIGN KEY (`Aluno_id`)
    REFERENCES `driveIt`.`Aluno` (`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;
```

Anexo 2: Queries realizadas neste projeto

```
USE driveIt;

-- Querie 0: Lista de todas as aulas
Select * From aula;

-- Querie 1: Lista de slots das aulas livres com info relevante, sem identificar o nome do instrutor
Select Data, Hora, Estado, Instrutor_id AS `ID do Instrutor` From aula
    where Estado IN ('L');


-- Querie 2: Lista de slots das aulas livres com info relevante, indentificando o nome do instrutor
Select Instrutor_id AS `ID do Instrutor`, Instrutor.Nome AS `Nome do Instrutor`, Data, Hora, Estado  From aula
    JOIN Instrutor ON Instrutor.id = aula.Instrutor_id
        where Estado IN ('L');


-- Querie 3: Lista de aulas marcadas por Aluno, ordenados ascendentemente pelo nome do aluno (mostra morada também)
Select aula.id AS `ID da Aula`, Aluno.id AS `ID do Aluno`, Aluno.Nome, aula.Data, aula.Hora,
    aula.Carro_Matricula AS `Matrícula do Carro`, aula.Instrutor_id AS `ID do Instrutor`, Morada.Rua, Morada.Localidade,
    Morada.`Código-Postal` FROM Aluno
    JOIN aula ON Aluno.id = aula.Aluno_id
    JOIN Morada ON Morada.idMorada = Aluno.Morada_idMorada
        order by Aluno.Nome ASC;


-- Querie 4: Lista de alunos com pagamentos em FALTA (mostra morada)
Select Pagamento AS `Estado de Pagamento`, id AS `ID do Aluno`, Nome, Email, Nrº_Telemóvel AS `Nº de Telemóvel`,
    Nrº_CC AS `Nº de CC`, NIF, Morada.Rua, Morada.Localidade, Morada.`Código-Postal` From Aluno
    JOIN Morada ON Morada.idMorada = Aluno.Morada_idMorada
        where Aluno.Pagamento = 'N'
        order by Nome ASC;


-- Querie 5: Lista de alunos com pagamentos FEITOS (mostra morada)
Select Pagamento AS `Estado de Pagamento`, id AS `ID do Aluno`, Nome, Email, Nrº_Telemóvel AS `Nº de Telemóvel`,
    Nrº_CC AS `Nº de CC`, NIF, Morada.Rua, Morada.Localidade, Morada.`Código-Postal` From Aluno
    JOIN Morada ON Morada.idMorada = Aluno.Morada_idMorada
        where Aluno.Pagamento = 'S'
        order by Nome ASC;


-- Querie 6: Lista de aulas dadas por instrutor
Select aula.id AS `ID da aula`, instrutor.id AS `ID do Instrutor`, Nome, Email, Nrº_Telemóvel, aula.Data, aula.Hora,
    aula.Carro_Matricula, aula.Aluno_id FROM instrutor
    JOIN aula ON instrutor.id = aula.Instrutor_id
        where aula.Estado = 'M'
         order by instrutor.Nome;


-- Querie 7: Ranking de funcionário, consoante as aulas com que interagiu (exclui-se as que estão livres) por ordem
-- decrescente
Select Nome, Email, Nrº_Telemóvel AS `Nº de Telemóvel`, count(Funcionario.id) AS `Total de Aulas` From Funcionario
    JOIN aula ON aula.Funcionario_id = Funcionario.id
        where aula.Estado IN ('M','D','R')
        group by Funcionario.id
        order by `Total de Aulas` DESC;


-- Querie 8: Ranking de instrutor, consoante as aulas dadas (contabiliza também as que estão a decorrer, caso
-- algum esteja durante uma aula
Select Nome, Email, Nrº_Telemóvel AS `Nº de Telemóvel`, count(Instrutor.id) AS `Total de Aulas` From Instrutor
    JOIN aula ON aula.Instrutor_id = Instrutor.id
        -- escolher se é só realizadas ou a decorrer também
        where aula.Estado IN ('R','D')
        group by Instrutor.id
        order by `Total de Aulas` DESC;


-- Querie 9: Ranking de carros, consoante as aulas em que foram utilizados
Select Matricula, count(Carro.Matricula) AS `Total de Aulas` From Carro
    JOIN aula ON aula.Carro_matricula = Carro.Matricula
        -- escolher se é só realizadas ou a decorrer também
        where aula.Estado IN ('R','D')
        group by Carro.Matricula
        order by `Total de Aulas` DESC;


-- Querie 10: Lista de alunos na nossa base de dados, com a sua morada, ordenados ascendentemente pelo nome
Select Aluno.id AS `ID do Aluno`, Nome, Email, Nrº_Telemóvel AS `Nº de Telemóvel`, Nrº_CC AS `Nº de CC`, NIF, Pagamento,
    Morada.Rua, Morada.Localidade, Morada.`Código-Postal` From Aluno
    JOIN Morada ON Morada.idMorada = Aluno.Morada_idMorada
        order by Nome ASC;


-- Querie 11: Lista de funcionários com a sua morada, ordenados ascendentemente pelo nome
Select Funcionario.id AS `ID do Funcionário`, Nome, Email, Nrº_Telemóvel AS `Nº de Telemóvel`, Nrº_CC AS `Nº de CC`,
    Morada.Rua, Morada.Localidade, Morada.`Código-Postal` From Funcionario
    JOIN Morada ON Morada.idMorada = Funcionario.Morada_idMorada
        order by Nome ASC;


-- Querie 12: Lista de instrutores com a sua morada
Select Instrutor.id AS `ID do Instrutor`, Nome, Email, Nrº_Telemóvel AS `Nº de Telemóvel`, Nrº_CC AS `Nº de CC`, Morada.Rua,
    Morada.Localidade, Morada.`Código-Postal` From Instrutor
    JOIN Morada ON Morada.idMorada = Instrutor.Morada_idMorada
        order by Nome ASC;


-- Querie 13: Listagem das aulas todas realizadas, identificando pelo aluno
SELECT aula.id AS `ID da aula`, Data, Hora, Estado AS `Estado da Aula`, Carro_Matricula,
    Instrutor.Nome AS `Nome do Instrutor`, Aluno.Nome AS `Nome do Aluno`, Aluno.Email FROM aula
    JOIN Aluno ON Aluno.id = aula.Aluno_id
    JOIN Instrutor ON Instrutor.id = aula.Instrutor_id
        where Estado IN ('R')
        order by Aluno.Nome;


-- Querie 14: Exemplo de uma atualização de uma aula, passar para 'realizada'. Só passa para realizada se o estado
-- for a decorrer e os dados introduzidos derem match
-- Se for preciso apagar a procedure:
DROP PROCEDURE update_aula_realizada;
delimiter //
CREATE PROCEDURE update_aula_realizada (IN idInstrutor INT, idAula INT)
       BEGIN
        Update aula
            Set Estado = 'R'
            -- é dado o estado da aula, id do Instrutor, o dia e a hora da aula
            where Estado = 'D' AND aula.id = idAula;
       END//
delimiter ;


-- Querie 15: Exemplo de um cancelamento de uma aula, passar para 'livre'. Só passa para livre se o estado
-- for de marcada e os dados introduzidos derem match
-- Se for preciso apagar a procedure:
DROP PROCEDURE cancelar_aula;
delimiter //
CREATE PROCEDURE cancelar_aula (IN idFuncionario INT, idAula INT)
       BEGIN
        Update aula
            -- atualiza slot para Livre
            -- atualiza a matricula do carro para null
            -- apaga o funcionário
            -- apaga o id do aluno
            Set Estado = 'L', Carro_Matricula = null, Funcionario_id = null, Aluno_id = null
            -- Aluno_id = 0,
                -- é dado o estado da aula, id do Funcionario (só funcionários é que cancelam aulas) e o id da aula
                where aula.Estado = 'M' AND aula.id = idAula AND aula.Funcionario_id = idFuncionario;
       END//
delimiter ;


-- Querie 16: Exemplo de uma marcação de aula, passar para 'marcada'. Só passa para marcada se o estado
-- for a livre e os dados introduzidos derem match
-- Se for preciso apagar a procedure:
DROP PROCEDURE marcar_aula;
delimiter //
CREATE PROCEDURE marcar_aula (IN idAula INT, idFuncionario INT, idAluno INT, matricula VARCHAR(45))
       BEGIN
        Update aula
            Set Estado = 'M', `Carro_Matricula` = matricula, Funcionario_id = idFuncionario, Aluno_id = idAluno
            -- é dado o id do aluno, o dia, hora da aula e confirmar que se tratava de uma aula a decorrer
            where Estado = 'L' AND aula.id = idAula;
       END//
delimiter ;

CALL marcar_aula(1,1,1,"QA-18-QL");


-- Querie 17: Marcar slots de todas as aulas de um instrutor para um dia
DROP PROCEDURE marcar_slots_aula;
delimiter //
CREATE PROCEDURE marcar_slots_aula (IN idInstrutor INT, dataC DATE)
       BEGIN
        INSERT INTO aula (`Data`,`Hora`,`Estado`,`Instrutor_Id`)
            VALUES
                (dataC,"10:00:00","L",idInstrutor),
                (dataC,"11:00:00","L",idInstrutor),
                (dataC,"15:00:00","L",idInstrutor),
                (dataC,"16:00:00","L",idInstrutor),
                (dataC,"17:00:00","L",idInstrutor);
       END//
delimiter ;

CALL marcar_slots_aula(1, date(now()));


-- Querie 18: Verificar todas as aula no dia atual (utilizando views)
CREATE VIEW vwAulasHoje AS
Select * from aula
    where Data = date(now());

Select * from vwAulasHoje;

-- Querie 19: Retorna o nome de um instrutor dado o id dele (FUNCTION)
DROP FUNCTION nomeInstrutor;
delimiter //
CREATE FUNCTION `nomeInstrutor` (idInstrutor int)
    returns VARCHAR(45)
    READS SQL DATA
    DETERMINISTIC
    Begin
        declare nomeInstrutor VARCHAR(45);
        set nomeInstrutor = (select Nome from Instrutor where id = idInstrutor);
        return nomeInstrutor;
    End//

Select nomeInstrutor(1);

-- Querie 20: A cada instrutor, inserir aulas dele do dia de hoje. Não funciona!
delimiter //
CREATE TRIGGER inserir_aulas AFTER INSERT ON Instrutor
FOR EACH ROW
BEGIN
    CALL marcar_slots_aula(NEW.id, date(now()));
END;
delimiter ;
```