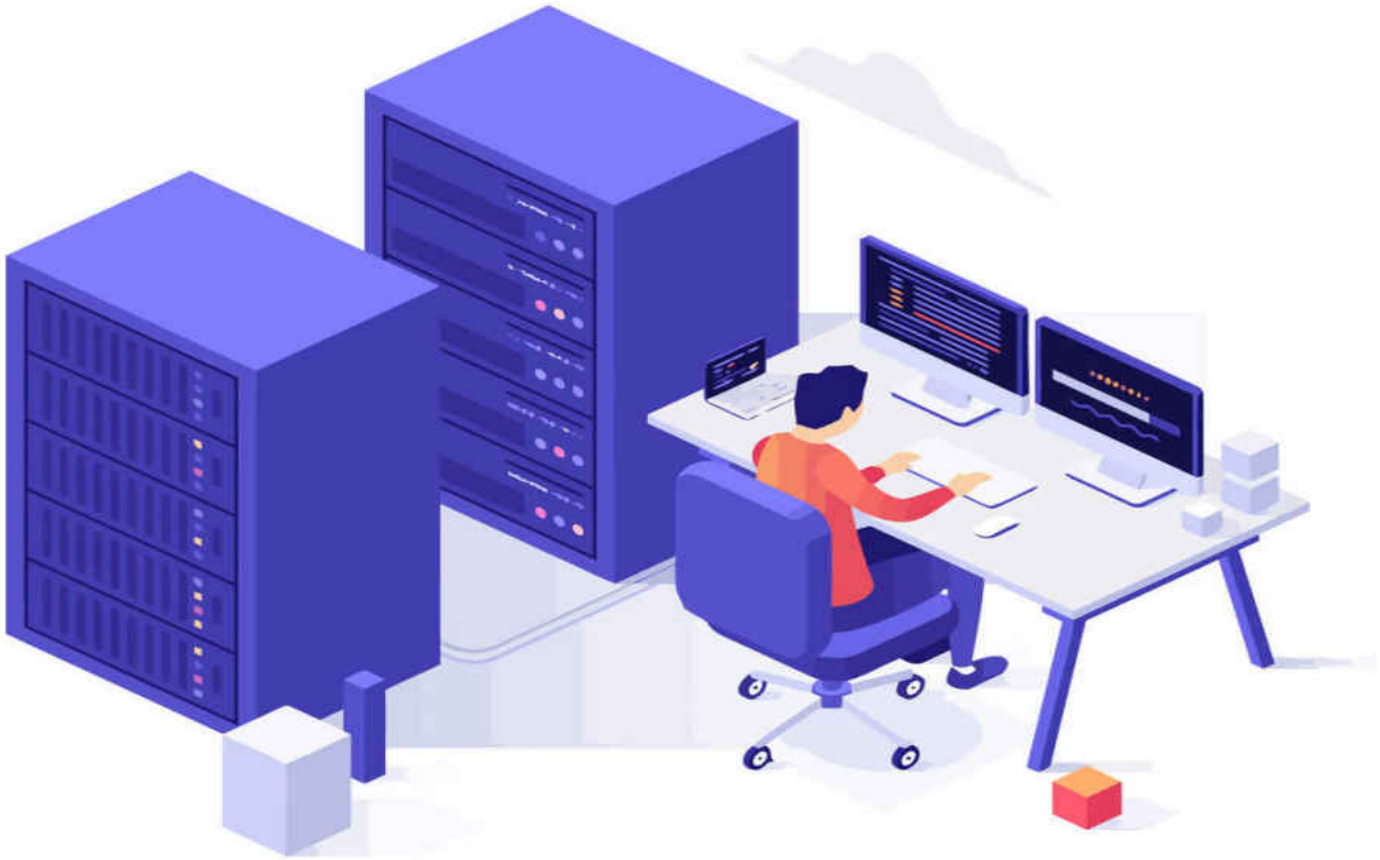
Tiago P. Ribeiro



O MÍNIMO QUE VOCÊ PRECISA SABER SOBRE BANCO DE DADOS E SQL

Uma abordagem prática e muito simples – para verdadeiros iniciantes.

PRODUTIVIDADE PROGRAMADA

produtividadeprogramada.com.br

[2022]
Produtividade Programada – O Mínimo que Você Precisa Saber sobre Banco de Dados
produtividadeprogramada.com.br/banco-de-dados

TREINAMENTO COM VÍDEO-AULAS

Prove por 07 dias – sem qualquer compromisso – o nosso treinamento em vídeo.

Você aprenderá tudo o que você encontra neste material, mas em vídeo-aulas cuidadosamente elaboradas pensando em seu completo entendimento sobre o tema Banco de Dados e SQL.

Você aprenderá tudo o que precisa saber sobre os Fundamentos de Banco de Dados Relacional, Modelo de Dados e consultas SQL.

E para você fixar e desenvolver o que foi estudado, elaboramos mais de 50 exercícios resolvidos de SQL para você fazer, aprender e se desenvolver.

Saiba mais sobre o treinamento agora mesmo, acessando o seguinte link:

produtividadeprogramada.com.br/banco-de-dados

SUMÁRIO

BEM-VINDO

Seja muito bem-vindo, meu caro! Que prazer te ver por aqui.

Você acabou de adquirir o e-book O Mínimo que você precisa saber sobre Banco de Dados, da série Produtividade Programada.

Foi uma ótima escolha! Parabéns pelo investimento.

Vemos nessa primeira parte do e-book as diretrizes gerais do treinamento, como funciona e o que deve ser feito para ter um excelente aproveitamento com resultados reais.

Você aprenderá:

- O que é um Banco de Dados;
- Estrutura de um Banco de Dados Relacional (o mais utilizado);
- O que é e como criar um Modelo de Dados;
- O que é SQL – A linguagem de programação utilizada em Bancos de Dados;
- SQLite – O Banco de Dados relacional mais simples e fácil de manusear que existe hoje, ideal para nosso aprendizado;
- (Bônus) Como interagir SQLite com Python;

Vamos nessa!

MAPA DE BORDO (ROADMAP)

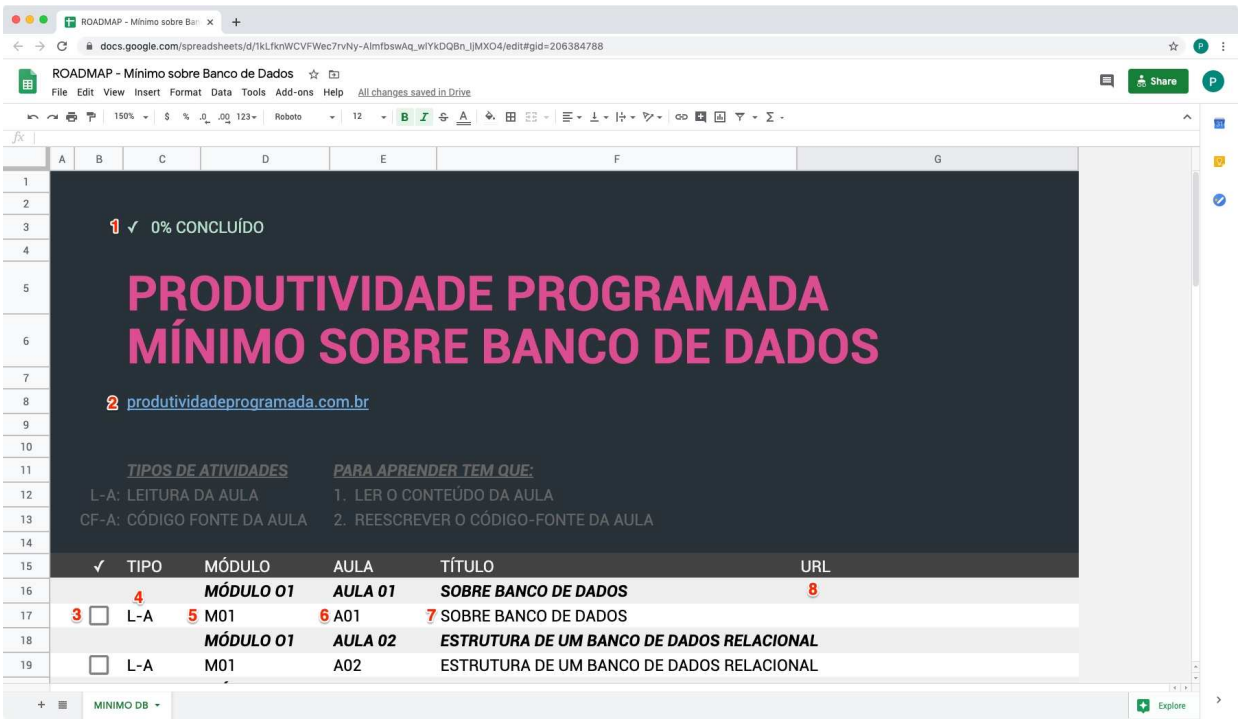
Preparamos um Mapa de Bordo (ROADMAP) para você acompanhar o seu progresso em cada aula estudada e cada código fonte reescrito.

Clique no link para copiar o guia (ROADMAP) para o seu GDrive: https://bit.ly/ROADMAP-MINIMO-DB

OU para versão em PDF: https://bit.ly/3c6OcWw

Temos a seguinte estrutura:

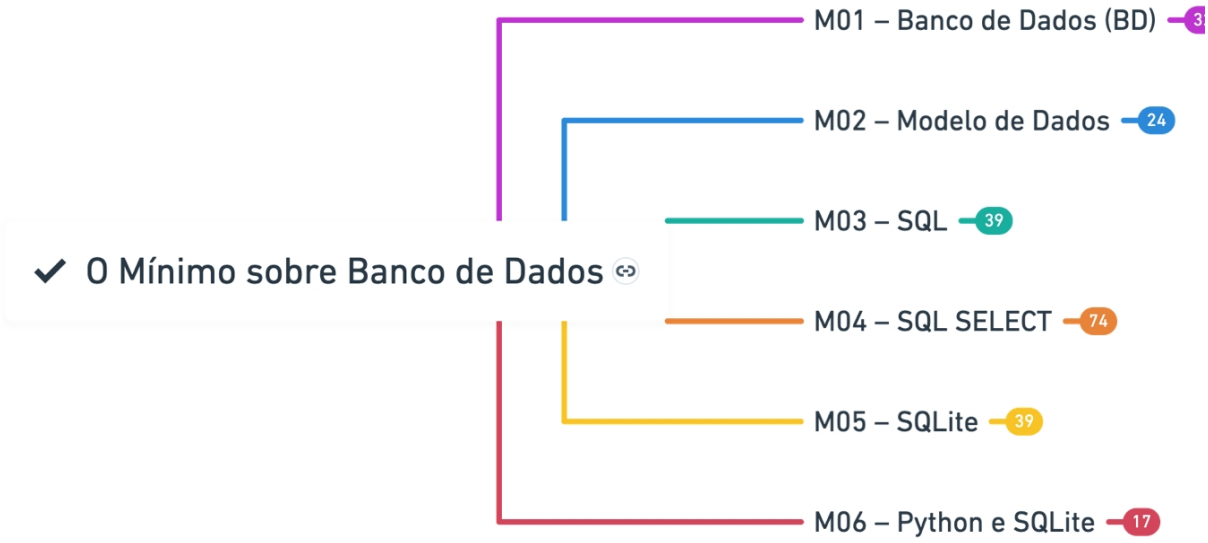
1. Percentual concluído do curso;
2. Link para a página de vendas do curso Produtividade Programada;
3. Checkbox – Deve ser marcado ao completar a atividade (tarefa);
4. Tipos de atividades:
    1. L-A: Leitura da Aula;
    2. CF-A: Código fonte da Aula;
5. Código do módulo do curso;
6. Código da aula do módulo;
7. Título da aula;
8. URL: Link para o código fonte apresentado em aula;


ROADMAP - Mínimo sobre Banco de Dados
File Edit View Insert Format Data Tools Add-ons Help
All changes saved in Drive
Share
1 ✓ 0% CONCLUÍDO
PRODUTIVIDADE PROGRAMADA
MÍNIMO SOBRE BANCO DE DADOS
2 produtividadeprogramada.com.br
TIPOS DE ATIVIDADES
L-A: LEITURA DA AULA
CF-A: CÓDIGO FONTE DA AULA
PARA APRENDER TEM QUE:
1. LER O CONTEÚDO DA AULA
2. REESCREVER O CÓDIGO-FONTE DA AULA
✓ TIPO MÓDULO AULA TÍTULO URL
MÓDULO 01 AULA 01 SOBRE BANCO DE DADOS 8
3 L-A 5 M01 6 A01 7 SOBRE BANCO DE DADOS
MÓDULO 01 AULA 02 ESTRUTURA DE UM BANCO DE DADOS RELACIONAL
L-A M01 A02 ESTRUTURA DE UM BANCO DE DADOS RELACIONAL
MINIMO DB
Explore

MAPA MENTAL (CONTEÚDO RESUMIDO)

Preparamos um mapa mental com todo o conteúdo do treinamento para você fixar o aprendizado de uma forma leve e descomplicada.

Clique no link para acessar: https://bit.ly/mindmap-minimo-db



CÓDIGO FONTE

Todos os códigos fonte deste livro se encontram na url abaixo:
https://bit.ly/3c43Lye

MÓDULO 01 – BANCO DE DADOS

Neste curso aprenderemos o mínimo necessário sobre Banco de Dados e a sua conexão com Python.

AULA 01 – SOBRE BANCO DE DADOS

O que é?

Podemos definir um Banco de Dados como sendo um conjunto de informações organizadas em um formato onde os dados possam ser facilmente armazenados, gerenciados, atualizados e recuperados.

Tipos de Banco de Dados

Os sistemas de Banco de Dados são divididos em diversas categorias, mas podemos dizer que a maioria se enquadra em duas categorias, Banco de Dados Relacional ou Banco de Dados Não Relacional, comumente chamado de NoSQL.

Vamos trabalhar neste treinamento com o modelo de Banco de Dados Relacional, o mais utilizado no mercado atualmente.

AULA 02 – ESTRUTURA DE UM BANCO DE DADOS RELACIONAL

Banco de Dados Relacional

O Banco de Dados Relacional é caracterizado pela sua forma de armazenamento dos dados em tabelas, da relação dessas tabelas umas com as outras e do seu *schema* fixo e rígido que determina como cada tabela deve ser.

Tabela ou Entidade

PESSOAS

CPF	***NOME***	***IDADE***
011.801.211-01	Maria Oliveira	30
011.801.211-02	João da Silva	40

A grosso modo, veja uma Tabela (ou Entidade) como sendo uma aba de uma planilha, ou seja, uma estrutura de armazenamento onde os dados são organizados em Colunas e Linhas.

Coluna ou Atributo

NOME
Maria Oliveira
João da Silva

A Coluna (ou Atributo) é uma unidade que armazena um tipo específico de valor.

Temos:

- Nome da Coluna – ex: *NOME*
- Valor da Coluna – ex: *Maria Oliveira*

Linha ou Tupla

011.801.211-01	Maria Oliveira	30

Uma Linha (ou Tupla) representa todos os dados de uma ocorrência na Tabela.

Por exemplo, na linha 1 temos todos os dados referentes a *Pessoa* (Tabela) que se chama *Maria*.

Tipos de Dados

Cada coluna/atributo da Tabela deve obrigatoriamente ter e aceitar um e somente um tipo de dado.

Por exemplo, a Coluna *NOME* deve ser do tipo TEXT – só poderá armazenar textos.

Já a Coluna *IDADE* deve ser do tipo INTEGER – número inteiro.

Os principais tipos são:

- **TEXT:** Textos;
- **INTEGER:** Números inteiros;
- **REAL:** Números decimais.
- **NULL:**Nulo.

Para saber mais sobre todos os tipos de dados suportados pelo Banco de Dados SQLite, acesse o seguinte link: https://www.sqlite.org/datatype3.html

Chave Primária (PK)

Uma Chave Primária (PK – Primary Key) serve para identificar exclusivamente cada linha em uma tabela, sendo um identificador único, um valor que não pode se repetir.

Por exemplo, o CPF de uma pessoa pode ser utilizado como Chave Primária, pois cada pessoa só possui um valor único de CPF e cada CPF pertence a uma única pessoa.

A Chave Primária pode ser definida em uma única coluna ou uma combinação de várias colunas – Chave Primária Composta.

Por convenção, geralmente usamos em cada Tabela a Coluna/Atributo *ID* (abreviação de identificação) sequencial como Chave Primária.

Por sequencial entende-se valores de números inteiros em sequência, por exemplo 1, 2, 3 e assim por diante – gerados automaticamente pelo Banco de Dados.

Regras para Chave Primária:

- Deve ser única;
- Não pode mudar nunca;
- Não pode ser nula;

Chave Estrangeira (FK)

PESSOAS

CPF	***NOME***	***CIDADE_ID***
011.801.211-01	Maria Oliveira	1

CIDADES

ID	***NOME***
1	São Paulo

Uma Chave Estrangeira (FK – Foreign Key) serve de referência para uma Chave Primária (PK) de uma outra tabela – na qual ela se relaciona.

Por exemplo, uma Tabela *Pessoas* pode ter como Coluna/Atributo *CIDADE_ID*, onde faz referência a Chave Primária *CIDADE_ID* da Tabela *Cidades*.

AULA 03 – RELACIONAMENTOS ENTRE TABELAS / ENTIDADES

Relacionamentos entre Tabelas / Entidades

O relacionamento é a associação entre as Tabelas, que são conectadas por Chaves Primárias e Chaves Estrangeiras.

São quatro tipos de relacionamentos:

- 1:1 – Relacionamento UM para UM;
- 1:N – Relacionamento UM para MUITOS;
- M:N – Relacionamento MUITOS para MUITOS;
- Auto Relacionamento;

1:1 – Relacionamento UM para UM

Abaixo:

- Um Gerente gerencia um (e somente um) Departamento*;*
- Um Departamento só possui um (e somente um) Gerente*;*
- *João só gerencia o RH e o RH só tem um Gerente, o João.*
- *Maria só gerencia TI e TI só tem uma gerente, a Maria.*

GERENTES

ID	***NOME***
1	João
2	Maria

DEPARTAMENTOS

ID	NOME	GERENTE_ID
1	Recursos Humanos	1
2	Tecnologia da Informação	2

O relacionamento **1:1** ocorre quando um registro da Tabela A corresponde com somente um registro da Tabela B.

No exemplo acima não teria problema a Chave Estrangeira estar na Tabela *GERENTES*. É uma questão de escolha somente.

1:N – Relacionamento UM para MUITOS

Abaixo:

- Uma Marca possui vários Produtos;
- Um Produto pertence a uma Marca (somente);
- *A Marca Apple tem vários produtos e o Produto iPhone pertence (somente) a Marca Apple.*

MARCAS

ID	NOME
1	Apple
2	Samsung

PRODUTOS

ID	NOME	MARCA_ID
1	iPhone	1
2	iMac	1
3	Galaxy	2

O relacionamento **1:N** ocorre quando um registro da Tabela A se relaciona com vários registros da Tabela B, mas cada registro da Tabela B só corresponde com um registro da Tabela A.

M:N – Relacionamento MUITOS para MUITOS

Abaixo:

- Uma Música possui um ou vários Autores;
- Um Autor possui a autoria de uma ou várias Músicas;
- *A Música "Eu Sei Que Vou te Amar" é de autoria de Tom Jobim e Vinicius de Morais;*
- *Tom Jobim é autor de várias músicas, dentre elas "Eu Sei Que Vou te Amar" e "Águas de Março";*

MUSICA

ID	***NOME***
1	Eu Sei Que Vou te Amar
2	Águas de Março

AUTORES

ID	***NOME***
1	Tom Jobim
2	Vinicius de Moraes

MUSICAS_AUTORES

MUSICA_ID	***AUTOR_ID***

1	1
1	2
2	1

O relacionamento **M:N** ocorre quando um registro da Tabela A se relaciona com vários registros da Tabela B e vice-versa.

Auto Relacionamento

Abaixo:

- Uma Pessoa possui um pai e uma mãe;
- *João é pai de José;*
- *Maria é mãe de José.*
- *José é filho de João e Maria.*

PESSOAS

ID	***NOME***	***PAI_ID***	***MAE_ID***
1	João		
2	Maria		
3	José	1	2

O **Auto Relacionamento** ocorre quando um registro da Tabela A se relaciona com um ou vários registros da mesma Tabela A.

MÓDULO 02 – MODELO DE DADOS

AULA 01 – SOBRE MODELO DE DADOS

O que é?

Existe uma quantidade massiva de informações em um Banco de Dados e é muito difícil entender como esses dados separados estão organizados e como eles se relacionam olhando para uma ou mais tabelas – como já vimos, bem similar a uma planilha.

Para este desafio, foram desenvolvidas formas visuais de organização dos dados.

Podemos dizer então que um Modelo de Dados é uma forma visual da organização dos dados e do relacionamento entre eles.

Existem diversos diagramas e modelos que podem ser utilizados em diversas fases de um projeto, mas para o nosso propósito, vamos estudar somente o Diagrama de Entidade e Relacionamento (DER).

App para manipulação do Modelo de Dados – Quick Database Diagrams

Podemos muito bem usar papel e caneta para desenhar os diagramas, mas é muito melhor e mais fácil usar uma ferramenta.

Sendo assim, usaremos o app Quick Database Diagrams.

https://www.quickdatabasediagrams.com/

AULA 02 – INTRODUÇÃO AO DIAGRAMA DE ENTIDADE E RELACIONAMENTO

Sobre Diagrama de Entidade e Relacionamento (DER)

O Diagrama de Entidade e Relacionamento (DER) é um modelo visual de organização dos dados e relacionamentos entre eles.

O Diagrama de Entidade e Relacionamento (DER) é composto de três categorias principais:

- Entidades (Tabelas)
- Atributos (Colunas)
- Relacionamentos

Entidades (Tabelas) são objetos ou conceitos associados a dados importantes – exemplo: *Cliente*.

Atributos (Colunas) representam as características de uma Entidade – exemplo: *Nome do Cliente*.

Relacionamentos demonstram os vínculos entre Entidades.

Entidade (Tabela), Atributos (Colunas) e Tipo do Atributo

Primeiramente, vamos estudar como uma tabela é representada graficamente em um DER.

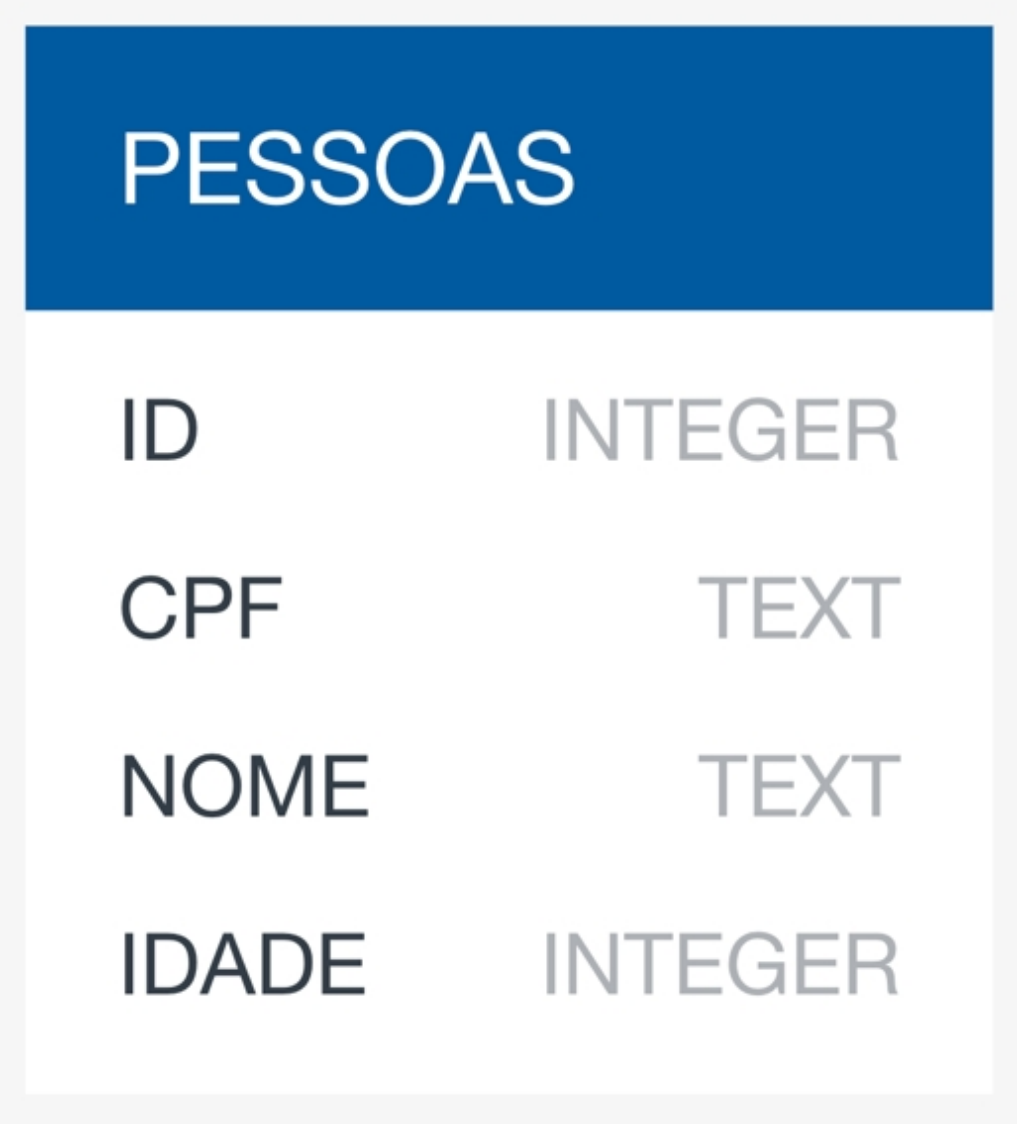
Como já vimos anteriormente, temos uma tabela com os dados das Pessoas, com os mesmos organizados em linhas e colunas.

PESSOAS

ID	***CPF***	***NOME***	***IDADE***
1	011.801.211-01	Maria Oliveira	30

Representamos no DER somente a estrutura da Tabela / Entidade, quais são os Atributos / Colunas e o Tipo do Atributo.

Sendo assim, vemos abaixo a representação gráfica do "esqueleto", da estrutura de uma Tabela / Entidade.



Sendo:

- PESSOAS: Nome da Entidade / Tabela
- ID, CPF, NOME, IDADE: Nome do Atributo / Coluna
- INTEGER, TEXT: Tipo do Atributo

Chave Primária – PK

Como vimos anteriormente, cada Entidade precisa de uma Chave Primária (PK – Primary Key) única para que cada registro seja facilmente identificado.

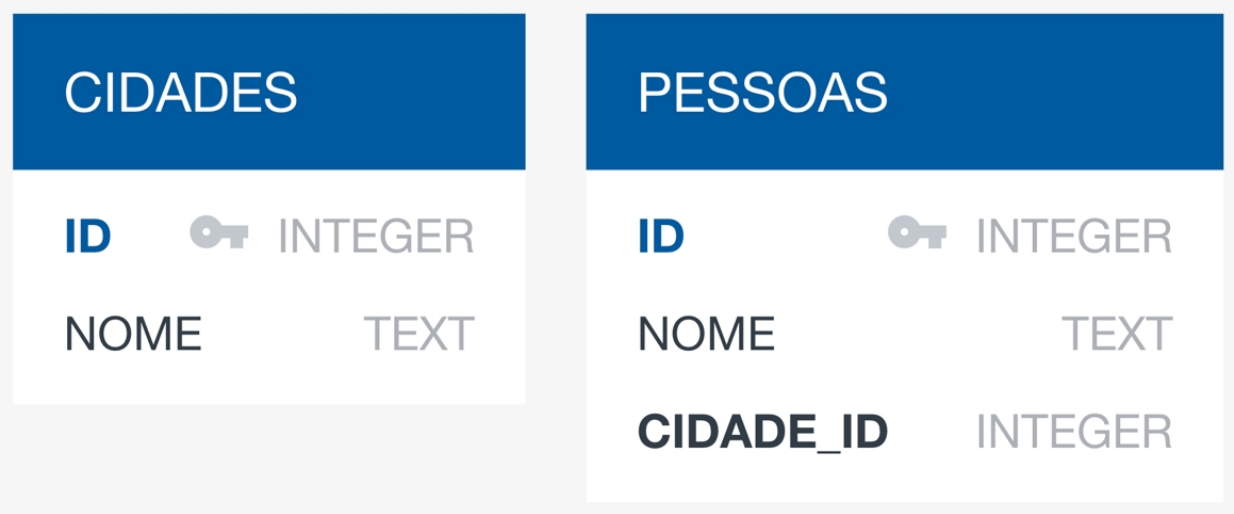
Para representar graficamente uma PK, comumente usamos o desenho de uma chave ��, mas vemos também a grafia "(PK)" antes ou depois do Tipo do Atributo – ex: ID INTEGER (PK).

No DER abaixo, vemos o Atributo ***CIDADES.ID*** ou ***ID*** da Entidade ***CIDADES*** com uma chave identificando que é uma PK e também o Atributo ***PESSOAS.ID*** ou ***ID*** da Entidade ***PESSOAS***.

Chave Estrangeira – FK

A Chave Estrangeira (FK) está representada no DER abaixo com o Atributo em negrito.

Temos como FK o Atributo ***PESSOAS.CIDADE_ID*** ou ***CIDADE_ID*** da Entidade ***PESSOAS***.



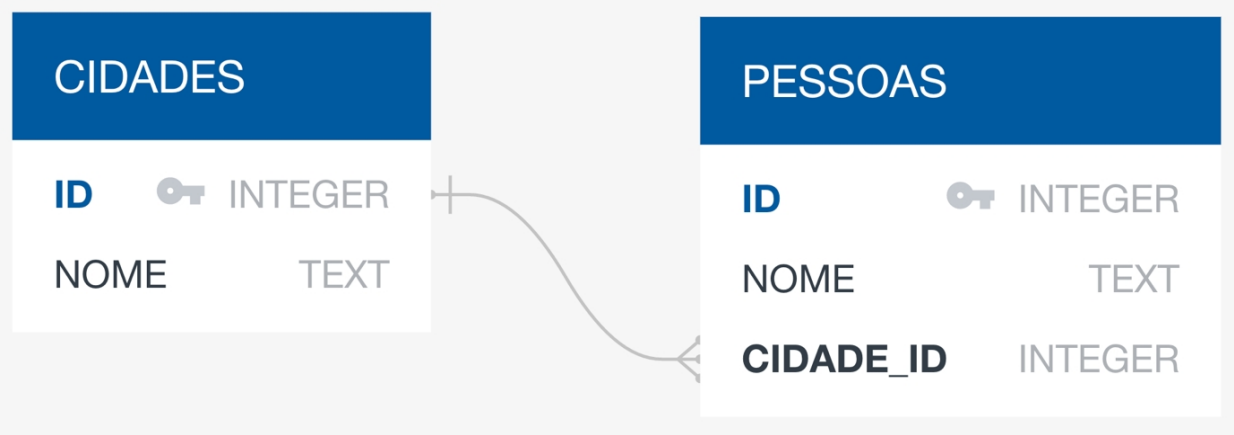
AULA 03 – [DER]
RELACIONAMENTOS

Sobre Relacionamentos

Vimos no DER acima duas Entidades, sendo que uma delas com um Atributo FK, entretanto sem nenhuma representação gráfica da ligação entre as duas Entidades.

Para representar um relacionamento, **usamos uma linha com uma cardinalidade nas pontas** – falaremos sobre cardinalidade em seguida.

Na DER abaixo temos um relacionamento da Entidade ***CIDADES*** com a Entidade ***PESSOAS*** através da FK ***PESSOAS.CIDADE_ID***, que por sua vez faz referência a PK ***CIDADES.ID*** – e para tanto temos a linha representando esse relacionamento.



Cardinalidades

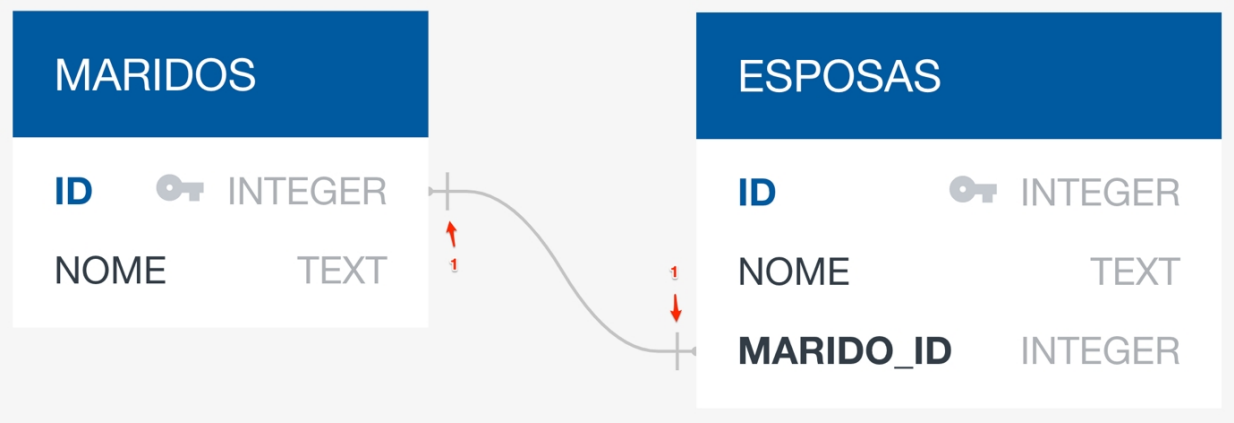
A Cardinalidade serve para definir o grau de relação entre duas Entidades.

Temos os seguintes níveis de relacionamento.

- 1:1 – Relacionamento UM para UM;
- 1:N – Relacionamento UM para MUITOS;
- M:N – Relacionamento MUITOS para MUITOS;

1:1 – Relacionamento UM para UM

Abaixo um relacionamento 1:1 de UM (1) Marido para com a sua única (1) Esposa e vice-versa.

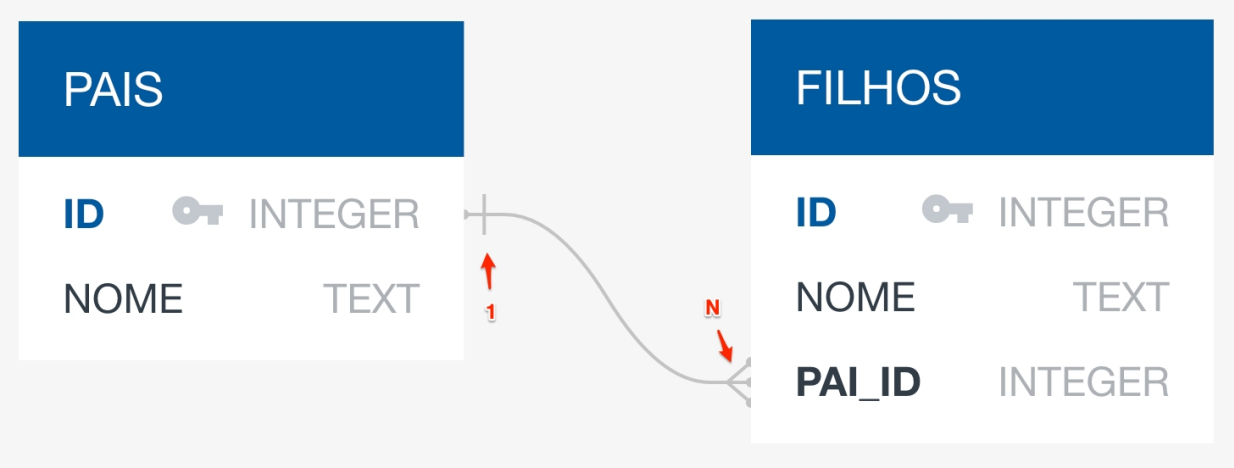


- UM (1) Marido possui UMA (1) Esposa*;*
- UMA (1) Esposa possui UM (1) Marido*;*

Veja que graficamente usamos um traço para representar o relacionamento UM (1).

1:N – Relacionamento UM para MUITOS

Abaixo um relacionamento 1:N de UM (1) Pai e seus MUITOS Filhos.



- UM (1) Pai pode ter ZERO (0), UM (1) ou MUITOS (N) Filhos*;*
- UM (1) Filho têm somente UM (1) Pai*;*

Veja que graficamente usamos um "pé de galinha" para representar o relacionamento MUITOS (N).

M:N – Relacionamento MUITOS para MUITOS

Abaixo um relacionamento entre MUITOS (N) Tios e MUITOS (N) Sobrinhos.

- UM (1) Tio pode ter ZERO (0), UM (1) ou MUITOS (N) Sobrinhos*;*
- UM (1) Sobrinho pode ter ZERO (0), UM (1) ou MUITOS (N) Tios*;*

MÓDULO 03 – SQL

AULA 01 – SOBRE SQL

O que é?

Linguagem de Consulta Estruturada ou *Structured Query Language* (SQL) é uma linguagem desenvolvida para "conversarmos" com o Banco de Dados.

Com ela podemos:

- Criar e modificar Bancos de Dados;
- Criar e modificar Tabelas;
- Criar e modificar Permissões de acesso;
- Criar e modificar Registros de Dados;
- Gerenciar Transações – ex: Confirmar ou Desfazer;

Organização da SQL

A SQL é organizada em subconjuntos com propósitos muito bem definidos.

São eles:

- **DDL – Data Definition Language ou Linguagem de Definição de Dados:** Define os comandos utilizados para criação, manutenção e exclusão das Tabelas e outras estruturas, como Índices e Views;
  - **CREATE:** Cria uma Tabela;
  - **ALTER:** Altera a Tabela;
  - **DROP:** Exclui a Tabela;

- **DCL – Data Control Language ou Linguagem de Controle de Dados:** Define os comandos utilizados para controlar o acesso aos dados do Banco de Dados;
  - **GRANT:** Concede o acesso;
  - **REVOKE:** Revoga (remove) o acesso;

- **DTL – Data Transaction Language ou Linguagem de Transação de Dados:** Define os comandos utilizados para gerenciar as transações executadas no Banco de Dados;
  - **BEGIN:** Inicia uma TRANSACTION (Transação);
  - **COMMIT:** Confirma uma TRANSACTION (Transação);
  - **ROLLBACK:** Desfaz uma TRANSACTION (Transação);

- **DML – Data Manipulation Language ou Linguagem de Manipulação de Dados:** Define os comandos utilizados para manipulação dos dados;
  - **INSERT:** Insere dados na Tabela;
  - **UPDATE:** Atualiza dados da Tabela;
  - **DELETE:** Exclui dados da tabela;

- **DQL – Data Query Language ou Linguagem de Consulta de Dados:** Define o comando utilizado para a realização das consultas dos dados;
  - **SELECT:** Consulta os dados em uma Tabela;

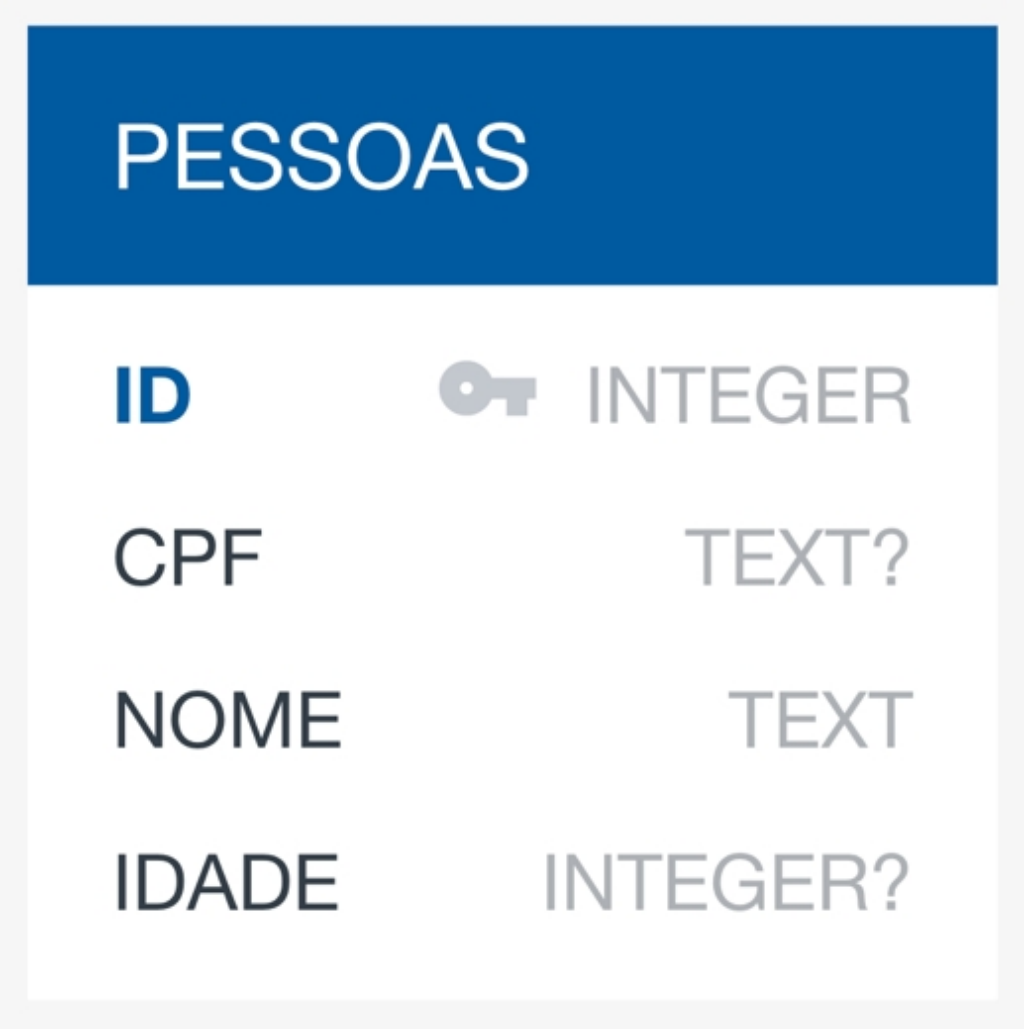
Tudo o que foi apresentado acima é de extrema importância, mas iremos focar em seguida no que é mais comumente usado no nosso dia a dia:

- Criação da Tabela;
- Inserção e manutenção de dados;
- Seleção de dados;

AULA 02 – [CREATE TABLE]
CRIANDO UMA TABELA

Para criar uma Tabela no Banco de Dados, usamos a instrução SQL CREATE TABLE.

Vamos criar com base na Entidade *PESSOAS* abaixo.



```sql
CREATE TABLE PESSOAS (
    ID INTEGER PRIMARY KEY AUTOINCREMENT,
    CPF TEXT,
    NOME TEXT NOT NULL,
    IDADE INTEGER
);
```

PP-MDB-M03-A02-C01: https://bit.ly/3c3pPso

Vamos examinar a Query (código fonte) acima com mais detalhes.

- **1:** Cria a Tabela *PESSOAS* utilizando a declaração CREATE TABLE.
- **2-5:** Define os Atributos da Tabela.
  - **2:** Cria o Atributo *ID* do Tipo *INTEGER*, *PRIMARY KEY* (PK, Chave Primária) e *AUTOINCREMENT* (Auto Incremental), ou seja, o Banco de Dados gera o valor do *ID* **automaticamente** somando +1 ao valor do último *ID*.
  - **3:** Cria o Atributo *CPF* do Tipo *TEXT*. Aceita registro nulo – por padrão, o Atributo aceita valores nulos.
  - **4:** Cria o Atributo *NOME* do Tipo *TEXT* e *NOT NULL* (não aceita registro nulo). Reforçando o fato de que é necessário definir o Atributo como sendo *NOT NULL* (não nulo), quando assim o quiser, pois o padrão do Banco de Dados é criar o Atributo como sendo *NULL*, Atributo que aceita registro de valor nulo.
  - **5:** Cria o Atributo *IDADE* do Tipo *INTEGER*.

AULA 03 – [INSERT] INSERINDO DADOS

Para inserir novos registros na Tabela *PESSOAS*, usamos a instrução SQL INSERT INTO.

Criamos com base na nos registros abaixo:

CPF	***NOME***	***IDADE***
011.801.211-01	Maria Oliveira	30
011.801.211-02	João da Silva	40

```sql
INSERT INTO PESSOAS(CPF, NOME, IDADE)
VALUES
  ('011.801.211-01', 'Maria Oliveira', 30),
  ('011.801.211-02', 'João da Silva', 40);
```

PP-MDB-M03-A03-C01: https://bit.ly/2JQ4Xca

Vamos examinar a instrução INSERT INTO com mais detalhes:

Primeiramente vemos a instrução INSERT INTO seguida do nome da Tabela, *PESSOAS* juntamente com os nomes das colunas entre parênteses.

Sobre as colunas:

- Podem vir em qualquer ordem;
- Somente as colunas *NOT NULL* são obrigatórias;

Nas linhas seguintes (2-4) temos os novos registros que serão inseridos.

Atente-se ao detalhe de que os novos registros devem ser inseridos na mesma ordem definida para as colunas.

AULA 04 – [UPDATE] ATUALIZANDO DADOS

Para atualizar algum registro na Tabela *PESSOAS*, usamos a instrução SQL UPDATE.

Perceba que na inserção da passada (AULA 03), inserimos o CPF, NOME e IDADE e o Banco de Dados gerou automaticamente o ID para cada linha – ID é Primary Key.

O ID será usado para identificarmos a linha e alterar o dado que queremos.

ID	***CPF***	***NOME***	***IDADE***
1	011.801.211-01	Maria Oliveira	30
2	011.801.211-02	João da Silva	40

```
1 UPDATE PESSOAS
2 SET IDADE = 45
3 WHERE ID = 2;
```

PP-MDB-M03-A04-C01: https://bit.ly/2Roag6z

Vamos examinar a instrução UPDATE com mais detalhes:

- **1:** Definimos através da instrução UPDATE a atualização da Tabela *PESSOAS*.
- **2:** Definimos que a coluna *IDADE* receberá como novo valor, o valor *45*.

- **3:** Definimos através do WHERE um filtro estabelecendo que somente a linha que possuir o *ID* igual a (=) *2* será atualizada.

ATENÇÃO! Se não usarmos a cláusula WHERE, toda a tabela PESSOAS será atualizada. Todas as linhas receberão um novo valor.

WHERE: É uma cláusula SQL utilizada para filtrar os dados – como vimos na Query acima.

Veremos com mais detalhes sobre a cláusula WHERE mais a frente.

AULA 05 – [DELETE] EXCLUINDO DADOS

Para excluir um ou mais registros na Tabela *PESSOAS*, usamos a instrução SQL DELETE.

```
1 DELETE FROM PESSOAS
2 WHERE ID = 2;
```

PP-MDB-M03-A05-C01: https://bit.ly/2y5pVRs

Vamos examinar a instrução DELETE:

- **1:** Definimos através da instrução DELETE a exclusão do(s) registro(s) na Tabela *PESSOAS*.
- **2:** Definimos através do WHERE um filtro estabelecendo que somente a linha que possuir o *ID* igual a (=) *2* será excluída.

ATENÇÃO! Se não usarmos a cláusula WHERE, todos os registros da Tabela *PESSOAS* serão excluídos.

MÓDULO 04 – SQL SELECT

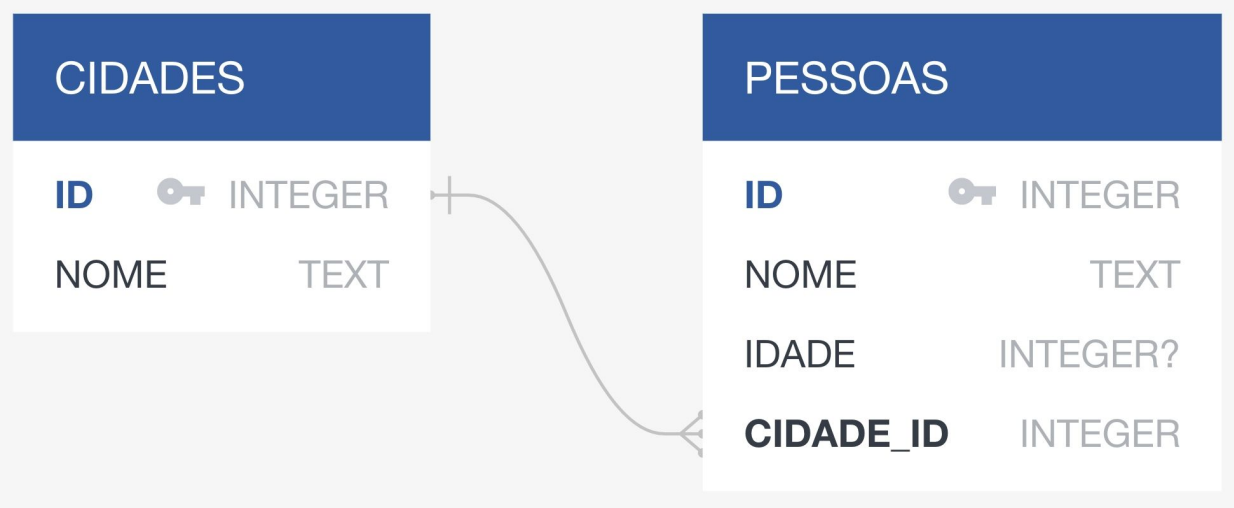
AULA 01 – INTRODUÇÃO

A declaração SELECT é usada para selecionar dados.

Os dados retornados são armazenados em uma Tabela de resultados, chamada *result-set* ou *Conjunto de Dados*.

Para selecionar os dados, usaremos as Tabelas *PESSOAS* e *CIDADES* usando o seguinte DER e dados definidos nas tabelas abaixo.

Criamos as seguintes Tabelas.




```sql
CREATE TABLE CIDADES (
    ID INTEGER PRIMARY KEY AUTOINCREMENT,
    NOME TEXT NOT NULL
);
```

PP-MDB-M04-A01-C01: https://bit.ly/3a3zptM

```sql
CREATE TABLE PESSOAS (
    ID INTEGER PRIMARY KEY AUTOINCREMENT,
    NOME TEXT NOT NULL,
    IDADE INTEGER,
    CIDADE_ID INTEGER NOT NULL
);
```

PP-MDB-M04-A01-C01: https://bit.ly/3a3zptM

Inserimos os seguintes dados.

PESSOAS

ID	NOME	IDADE	CIDADE_ID
1	Maria Oliveira	30	1
2	Tiago da Silva	30	2
3	Raquel da Cruz	39	3
4	Ana Joaquina Silveira	34	2
5	José Dias	35	2
6	Ana Carolina Santos	36	1
7	João do Carmo	80	2
8	Marcos Alves	90	1
9	Maria das Dores	83	2
10	Priscila da Silva	79	1

CIDADES

ID	NOME

1	Brasília
2	São Paulo
3	São Joaquim

```sql
INSERT INTO CIDADES(NOME) VALUES
   ('Brasília'),
   ('São Paulo'),
   ('São Joaquim');
```

PP-MDB-M04-A01-C01: https://bit.ly/3a3zptM

```sql
INSERT INTO PESSOAS(NOME, IDADE, CIDADE_ID) VALUES
   ('Maria Oliveira',   30, 1),
   ('Tiago da Silva',   30, 2),
   ('Raquel da Cruz',   39, 3),
   ('Ana Joaquina Silveira',    34, 2),
   ('José Dias',    35, 2),
   ('Ana Carolina Santos',  36, 1),
   ('João do Carmo',    80, 2),
   ('Marcos Alves', 90, 1),
   ('Maria das Dores',  83, 2),
   ('Priscila da Silva',    79, 1);
```

PP-MDB-M04-A01-C01: https://bit.ly/3a3zptM

AULA 02 – SELECIONANDO OS DADOS

Nesta aula aprenderemos a estrutura de uma declaração SELECT e aprenderemos a selecionar os dados de três formas diferentes.

Seleção de todas as Colunas

```
1 SELECT *
2 FROM PESSOAS;
```

PP-MDB-M04-A02-C01: https://bit.ly/2xmvXgK

Uma declaração SELECT tem a seguinte estrutura:

- **SELECT:** Declaração de consulta dos dados;
- ***:** Define que serão selecionadas todas as Colunas da Tabela;
- **FROM:** Define qual a Tabela que será usada na seleção dos dados;
- **PESSOAS:** Nome da Tabela.

Conjunto de Dados retornado:

PESSOAS

ID	*NOME*	*IDADE*	*CIDADE_ID*
1	Maria Oliveira	30	1
2	Tiago da Silva	30	2
3	Raquel da Cruz	39	3
4	Ana Joaquina Silveira	34	2
5	José Dias	35	2
6	Ana Carolina Santos	36	1
7	João do Carmo	80	2
8	Marcos Alves	90	1
9	Maria das Dores	83	2

10	Priscila da Silva	79	1

Seleção de algumas colunas

```
1 SELECT NOME, IDADE
2 FROM PESSOAS;
```

PP-MDB-M04-A02-C01: https://bit.ly/2xmvXgK

A declaração SELECT acima retorna somente o Conjunto de Dados presente nas Colunas *NOME* e *IDADE* da Tabela *PESSOAS*.

Conjunto de Dados retornado:

PESSOAS

NOME	***IDADE***
Maria Oliveira	30
Tiago da Silva	30
Raquel da Cruz	39
Ana Joaquina Silveira	34
José Dias	35
Ana Carolina Santos	36
João do Carmo	80
Marcos Alves	90
Maria das Dores	83
Priscila da Silva	79

Seleção única dos dados de uma Coluna

```
1 SELECT DISTINCT(CIDADE_ID)
2 FROM PESSOAS;
```

PP-MDB-M04-A02-C01: https://bit.ly/2xmvXgK

Com a declaração acima, filtramos e temos como resultado um Conjunto de Dados com os valores da *PESSOAS.CIDADE_ID*, mas sem dados repetidos.

Conjunto de Dados retornado:

PESSOAS

CIDADE_ID
1
2
3

AULA 03 – [WHERE]
SELECIONANDO COM FILTROS

Usamos a cláusula WHERE para filtrar os dados.

Para isso, usamos Operadores de Comparação e Operadores Lógicos, como veremos a seguir.

Operadores de Comparação

Um Operador de Comparação testa se duas expressões são verdadeiras.

Sendo assim, é correto afirmar que:

- 1 = 1
- 1 <> 2 ou 1 != 2
- 1 < 2
- 2 > 1

Abaixo todos os Operadores de Comparação.

Operador	***Significado***
=	Igual a
<> ou !=	Não igual a
<	Menor que
>	Maior que
<=	Menor ou igual a
>=	Maior ou igual a

(=) Igual a – com teste de Números

```
1 SELECT *
2 FROM PESSOAS
3 WHERE IDADE = 30;
```

PP-MDB-M04-A03-C01: https://bit.ly/2xpPUTV

Conjunto de Dados retornado:

PESSOAS

ID	NOME	IDADE	CIDADE_ID
1	Maria Oliveira	30	1
2	Tiago da Silva	30	2

(=) Igual a – com teste de String

```sql
SELECT *
FROM PESSOAS
WHERE NOME = 'Marcos Alves';
```

PP-MDB-M04-A03-C01: https://bit.ly/2xpPUTV

Conjunto de Dados retornado:

PESSOAS

ID	NOME	IDADE	CIDADE_ID
8	Marcos Alves	90	1

(>) Maior que

```sql
SELECT *
FROM PESSOAS
WHERE IDADE > 70;
```

PP-MDB-M04-A03-C01: https://bit.ly/2xpPUTV

Conjunto de Dados retornado:

PESSOAS

ID	*NOME*	*IDADE*	*CIDADE_ID*
7	João do Carmo	80	2
8	Marcos Alves	90	1
9	Maria das Dores	83	2
10	Priscila da Silva	79	1

Operadores Lógicos

O Banco de Dados SQLite não fornece Tipo de Valor *BOOLEAN*, portanto *1* significa *TRUE* e *0* significa *FALSE*.

Sendo assim, temos abaixo os seguintes Operadores Lógicos:

Operador	***Significado***
IN	Retorna 1 se o valor testado estiver na lista
LIKE	Retorna 1 se o valor testado coincidir com o padrão passado
AND	Retorna 1 se as duas expressões testadas forem 1
OR	Retorna 1 se ao menos uma das expressões testadas forem 1
BETWEEN	Retorna 1 se o valor testado estiver no intervalo passado

IN

No exemplo abaixo temos o retorno de um Conjunto de Dados onde *PESSOAS.IDADE* são iguais a (=) 30 ou 79.

```
1 SELECT *
2 FROM PESSOAS
3 WHERE IDADE IN(30, 79);
```

PP-MDB-M04-A03-C01: https://bit.ly/2xpPUTV

Conjunto de Dados retornado:

PESSOAS

ID	***NOME***	***IDADE***	***CIDADE_ID***
1	Maria Oliveira	30	1
2	Tiago da Silva	30	2
10	Priscila da Silva	79	1

LIKE com Sinal Curinga (%)

No exemplo abaixo temos o retorno de um Conjunto de Dados onde *PESSOAS.NOME* possuem o valor 'Silva' em algum lugar do *NOME*.

Só é possível fazer este tipo de comparação porque usamos o Sinal Curinga (%) no começo e final da String – ***'%Silva%'***, sendo assim, os nomes abaixo retornariam no Conjunto de Dados:

- João da Silva
- Silva Maria
- José Silva Maria

Se quiséssemos um Conjunto de Dados com (somente) os nomes que terminam com 'Silva', teríamos que testar com a String ***'%Silva'*** e se quiséssemos quem tivesse o primeiro nome 'Silva', testaríamos com **'Silva%'**.

```sql
SELECT *
FROM PESSOAS
WHERE NOME LIKE '%Silva%';
```

PP-MDB-M04-A03-C01: https://bit.ly/2xpPUTV

Conjunto de Dados retornado:

PESSOAS

ID	NOME	IDADE	CIDADE_ID
2	Tiago da Silva	30	2
10	Priscila da Silva	79	1

AND

No exemplo abaixo temos o retorno de um Conjunto de Dados onde *PESSOAS.IDADE* deve ser maior que *30* e (AND) *PESSOAS.CIDADE_ID* igual a *1*.

```
1 SELECT *
2 FROM PESSOAS
3 WHERE IDADE > 30
4 AND CIDADE_ID = 1;
```

PP-MDB-M04-A03-C01: https://bit.ly/2xpPUTV

Conjunto de Dados retornado:

PESSOAS

ID	NOME	IDADE	CIDADE_ID
6	Ana Carolina Santos	36	1
8	Marcos Alves	90	1
10	Priscila da Silva	79	1

OR

No exemplo abaixo temos o retorno de um Conjunto de Dados onde *PESSOAS.IDADE* deve ser igual a *30* ou (OR) *PESSOAS.CIDADE_ID* igual a *3*.

```
1 SELECT *
2 FROM PESSOAS
3 WHERE IDADE = 30
4 OR CIDADE_ID = 3;
```

PP-MDB-M04-A03-C01: https://bit.ly/2xpPUTV

Conjunto de Dados retornado:

PESSOAS

ID	NOME	IDADE	CIDADE_ID
1	Maria Oliveira	30	1
2	Tiago da Silva	30	2
3	Raquel da Cruz	39	3

BETWEEN

No exemplo abaixo temos o retorno de um Conjunto de Dados onde *PESSOAS.IDADE* deve estar entre 30 e 39.

```
1 SELECT *
2 FROM PESSOAS
3 WHERE IDADE BETWEEN 30 AND 39;
```

PP-MDB-M04-A03-C01: https://bit.ly/2xpPUTV

Conjunto de Dados retornado:

PESSOAS

ID	*NOME*	*IDADE*	*CIDADE_ID*
1	Maria Oliveira	30	1
2	Tiago da Silva	30	2
3	Raquel da Cruz	39	3
4	Ana Joaquina Silveira	34	2
5	José Dias	35	2
6	Ana Carolina Santos	36	1

AULA 04 – [ORDER BY]
ORDENANDO OS RESULTADOS

Por ordem crescente (ASC)

No exemplo abaixo temos o retorno de um Conjunto de Dados ordenado (ORDER BY) por ordem crescente de *PESSOAS.IDADE*.

```
1 SELECT *
2 FROM PESSOAS
3 ORDER BY IDADE;
```

PP-MDB-M04-A04-C01: https://bit.ly/2VrMatc

Conjunto de Dados retornado:

PESSOAS

ID	NOME	IDADE	CIDADE_ID
1	Maria Oliveira	30	1
2	Tiago da Silva	30	2
4	Ana Joaquina Silveira	34	2
5	José Dias	35	2
6	Ana Carolina Santos	36	1
3	Raquel da Cruz	39	3
10	Priscila da Silva	79	1
7	João do Carmo	80	2
9	Maria das Dores	83	2
8	Marcos Alves	90	1

Por ordem decrescente (DESC)

No exemplo abaixo temos o retorno de um Conjunto de Dados ordenado (ORDER BY) por ordem decrescente de *PESSOAS.IDADE*.

```
1 SELECT *
2 FROM PESSOAS
3 ORDER BY IDADE DESC;
```

PP-MDB-M04-A04-C01: https://bit.ly/2VrMatc

Conjunto de Dados retornado:

PESSOAS

ID	***NOME***	***IDADE***	***CIDADE_ID***
8	Marcos Alves	90	1
9	Maria das Dores	83	2
7	João do Carmo	80	2
10	Priscila da Silva	79	1
3	Raquel da Cruz	39	3
6	Ana Carolina Santos	36	1
5	José Dias	35	2
4	Ana Joaquina Silveira	34	2
1	Maria Oliveira	30	1
2	Tiago da Silva	30	2

Por múltiplas colunas

No exemplo abaixo temos o retorno de um Conjunto de Dados ordenado (ORDER BY) por ordem crescente de *PESSOAS.IDADE* e *PESSOAS.NOME*.

```
1 SELECT *
2 FROM PESSOAS
3 ORDER BY IDADE, NOME;
```

PP-MDB-M04-A04-C01: https://bit.ly/2VrMatc

Conjunto de Dados retornado:

PESSOAS

ID	***NOME***	***IDADE***	***CIDADE_ID***
1	Maria Oliveira	30	1
2	Tiago da Silva	30	2
4	Ana Joaquina Silveira	34	2
5	José Dias	35	2
6	Ana Carolina Santos	36	1
3	Raquel da Cruz	39	3
10	Priscila da Silva	79	1
7	João do Carmo	80	2
9	Maria das Dores	83	2
8	Marcos Alves	90	1

AULA 05 – FUNÇÕES

Temos no SQLite funções embutidas que realizam algum tipo de processamento.

Veremos a seguir as seguintes funções:

FUNÇÃO	***DESCRIÇÃO***
AVG	Retorna o valor da Média do grupo
MIN	Retorna o Menor valor
MAX	Retorna o Maior valor
SUM	Retorna a Soma
COUNT	Retorna a Quantidade de linhas

AVG

No exemplo abaixo temos o retorno do valor da Média de *PESSOAS.IDADE*.

```
1 SELECT AVG(IDADE) FROM PESSOAS;
```

PP-MDB-M04-A05-C01: https://bit.ly/34w2Fbr

Conjunto de Dados retornado:

AVG(IDADE)
53.6

MIN

No exemplo abaixo temos o retorno do Menor valor da *PESSOAS.IDADE*.

```
1 SELECT MIN(IDADE) FROM PESSOAS;
```

PP-MDB-M04-A05-C01: https://bit.ly/34w2Fbr

Conjunto de Dados retornado:

MIN(IDADE)
30

MAX

No exemplo abaixo temos o retorno do Maior valor da *PESSOAS.IDADE*.

```
1 SELECT MAX(IDADE) FROM PESSOAS;
```

PP-MDB-M04-A05-C01: https://bit.ly/34w2Fbr

Conjunto de Dados retornado:

MAX(IDADE)
90

SUM

No exemplo abaixo temos o retorno da Soma dos valores da *PESSOAS.IDADE*.

```
1 SELECT SUM(IDADE) FROM PESSOAS;
```

PP-MDB-M04-A05-C01: https://bit.ly/34w2Fbr

Conjunto de Dados retornado:

SUM(IDADE)
536

COUNT

No exemplo abaixo temos o retorno da Quantidade de linhas na Tabela *PESSOAS*.

```
1 SELECT COUNT(*) FROM PESSOAS;
```

PP-MDB-M04-A05-C01: https://bit.ly/34w2Fbr

Conjunto de Dados retornado:

COUNT(*)
10

AULA 06 – [GROUP BY] AGRUPAMENTO

No exemplo abaixo temos o retorno de um Conjunto de Dados da Quantidade (COUNT) de Linhas agrupadas (GROUP BY) por Cidade.

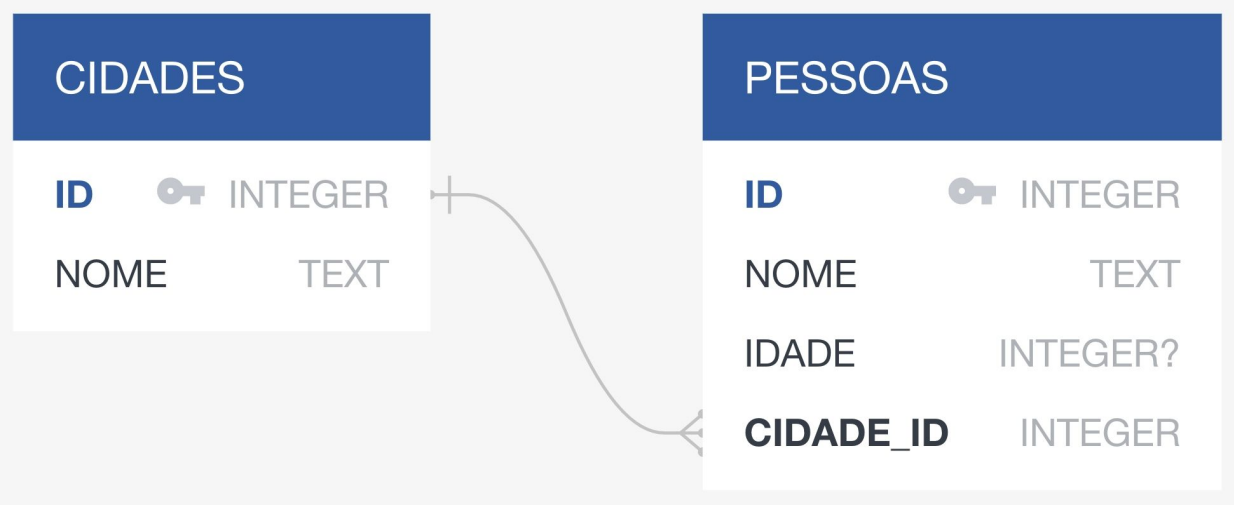
```
1 SELECT CIDADE_ID, COUNT(NOME)
2 FROM PESSOAS
3 GROUP BY CIDADE_ID;
```

PP-MDB-M04-A06-C01: https://bit.ly/2Rsd0zR

Conjunto de Dados retornado:

CIDADE_ID	***COUNT(NOME)***
1	4
2	5
3	1

AULA 07 – JUNÇÃO DE TABELAS



Até então vimos somente os dados da Tabela *PESSOAS* e vimos que no resultado da Tabela *PESSOAS* temos a Coluna *PESSOAS.CIDADE_ID*.

E se quisermos o valor do Atributo *CIDADES.NOME* no Conjunto de Dados de retorno, de acordo com o Nome da Cidade que a Pessoa pertence?

O conceito da solução é bem simples. Basta selecionar os valores das duas Tabelas na cláusula FROM e "ligar" as duas através da cláusula WHERE, onde *PESSOAS.CIDADE_ID* devem ser iguais a *CIDADES.ID*.

Veja o exemplo abaixo:

```
1 SELECT CIDADES.NOME, PESSOAS.NOME, PESSOAS.IDADE
2 FROM PESSOAS, CIDADES
3 WHERE PESSOAS.CIDADE_ID = CIDADES.ID
```

PP-MDB-M04-A07-C01: https://bit.ly/2XpgQOh

Conjunto de Dados retornado:

NOME (CIDADES)	***NOME (PESSOAS)***	***IDADE***
Brasília	Maria Oliveira	30
São Paulo	Tiago da Silva	30
São Joaquim	Raquel da Cruz	39
São Paulo	Ana Joaquina Silveira	34
São Paulo	José Dias	35
Brasília	Ana Carolina Santos	36
São Paulo	João do Carmo	80
Brasília	Marcos Alves	90
São Paulo	Maria das Dores	83
Brasília	Priscila da Silva	79

MÓDULO 05 – SQLITE

AULA 01 – SOBRE O SQLITE

SQLite é um Banco de Dados SQL, relacional, pequeno, rápido, eficiente, independente, de alta confiabilidade e completo, sendo o Banco de Dados mais utilizado no mundo hoje.

O SQLite está embutido em todos os telefones celulares e na maioria dos computadores e aplicativos.

O uso do SQLite é recomendado onde a simplicidade da administração, implementação e manutenção são mais importantes que incontáveis recursos que Sistema de Gerenciamento de Banco de Dados (SGBD) tradicionais, mais voltados para aplicações complexas, comumente implementam.

As situações onde a simplicidade é a melhor escolha são muito mais frequentes do que podemos imaginar.

O SQLite:

- É Software Livre e multiplataforma;
- Não necessita de instalação, configuração ou administração;
- Permite guardar o banco de dados em um único arquivo;
- Suporta bases de dados abaixo de 2TB;
- Não tem dependências externas;

Exemplos de uso do SQLite:

- Sites com menos de cem mil requisições por dia;
- Dispositivos e sistemas embarcados;
- Aplicações desktop;

- Ferramentas estatísticas e de análise;
- Aprendizado de banco de dados;

Não se recomenda o uso do SQLite para sites com:

- Muitos acessos
- Grande quantidade de dados (talvez maior que algumas dúzias de gigabytes)
- Sistemas com grande concorrência

AULA 02 – SQLITE ONLINE

Para diversas finalidades podemos usar um Banco de Dados online, sem nenhuma instalação, configuração ou cadastro.

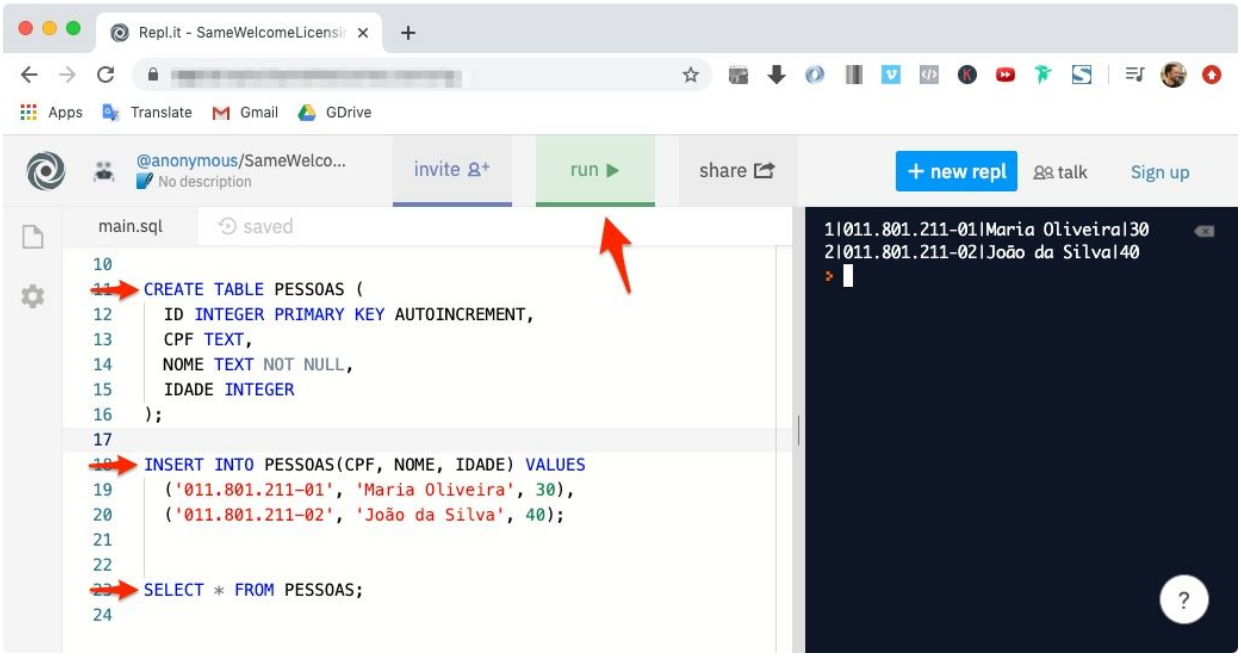
Temos diversas opções de Banco de Dados SQLite online, bastando pesquisar no Google por "SQLite Online" para encontrar várias opções.

Dentre as opções disponíveis, optamos pelo serviço Repl.it.

Acesse clicando no link seguinte: https://repl.it/languages/sqlite

O Banco de Dados SQLite no Repl.it executa em memória, ou seja, os dados não são salvos e para tanto precisamos criar as tabelas e inserir os dados todas as vezes que executarmos o script SQL.

Veja o exemplo abaixo.



https://bit.ly/2Roag6z

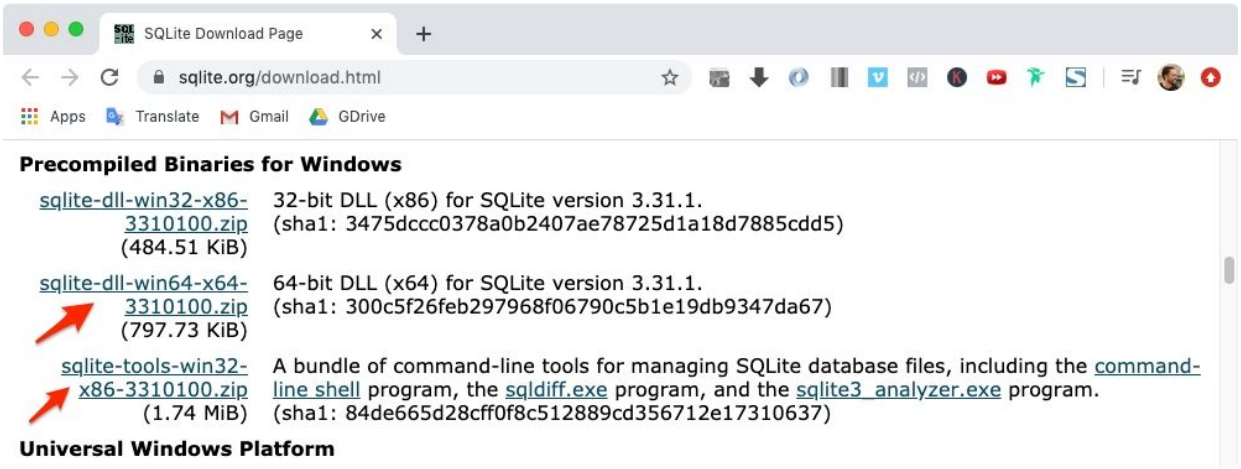
AULA 03 – INSTALANDO NO WINDOWS

Para instalar o SQLite no Microsoft Windows siga os seguintes passos.

Passo 01 – Download dos Arquivos

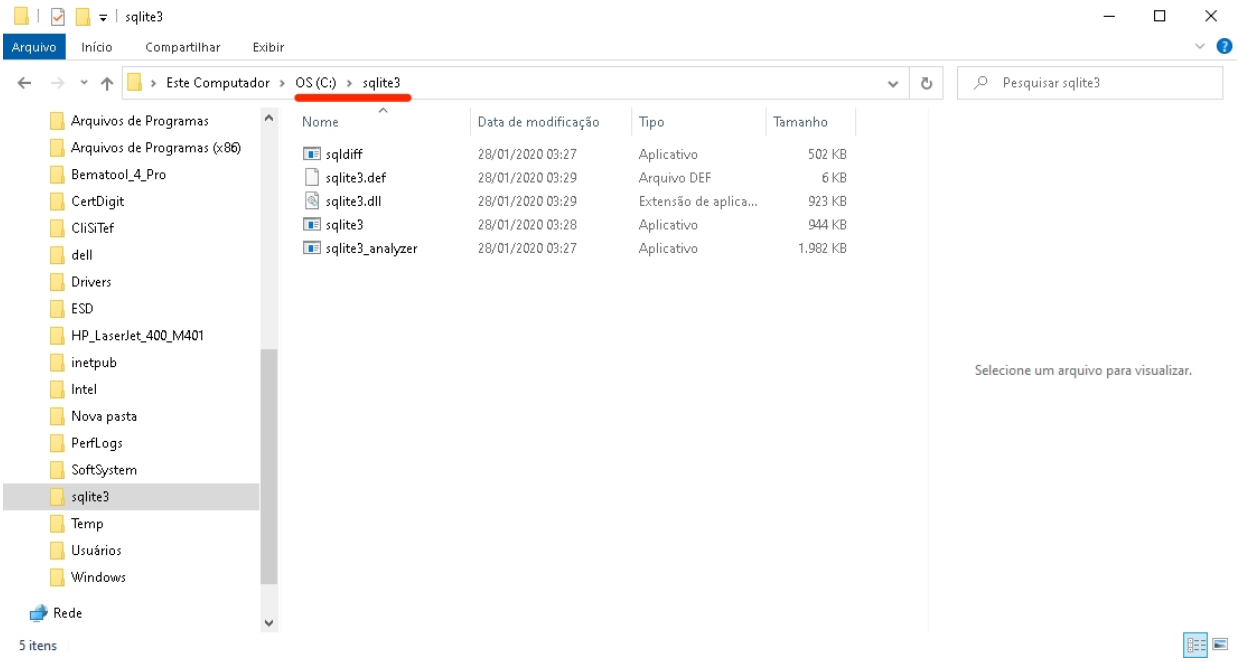
Vá para a página de downloads: https://www.sqlite.org/download.html

Na seção **Precompiled Binaries for Windows** baixe os arquivos sqlite-dll-win64-x64-xxxxxxx.zip e sqlite-tools-win32-x86-xxxxxxx.zip.



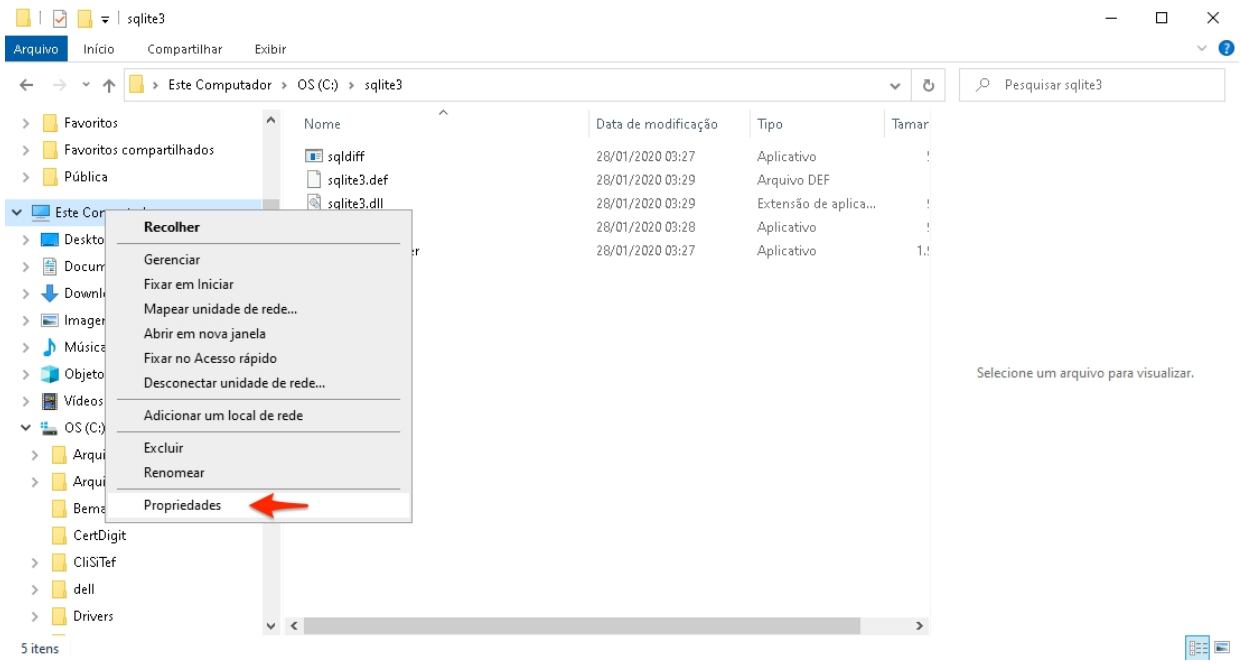
Passo 02 – Descompacte os Arquivos

Descompacte os arquivos dos dois arquivos .zip na mesma pasta sqlite3 e copie a pasta para a raiz, C:\sqlite3.



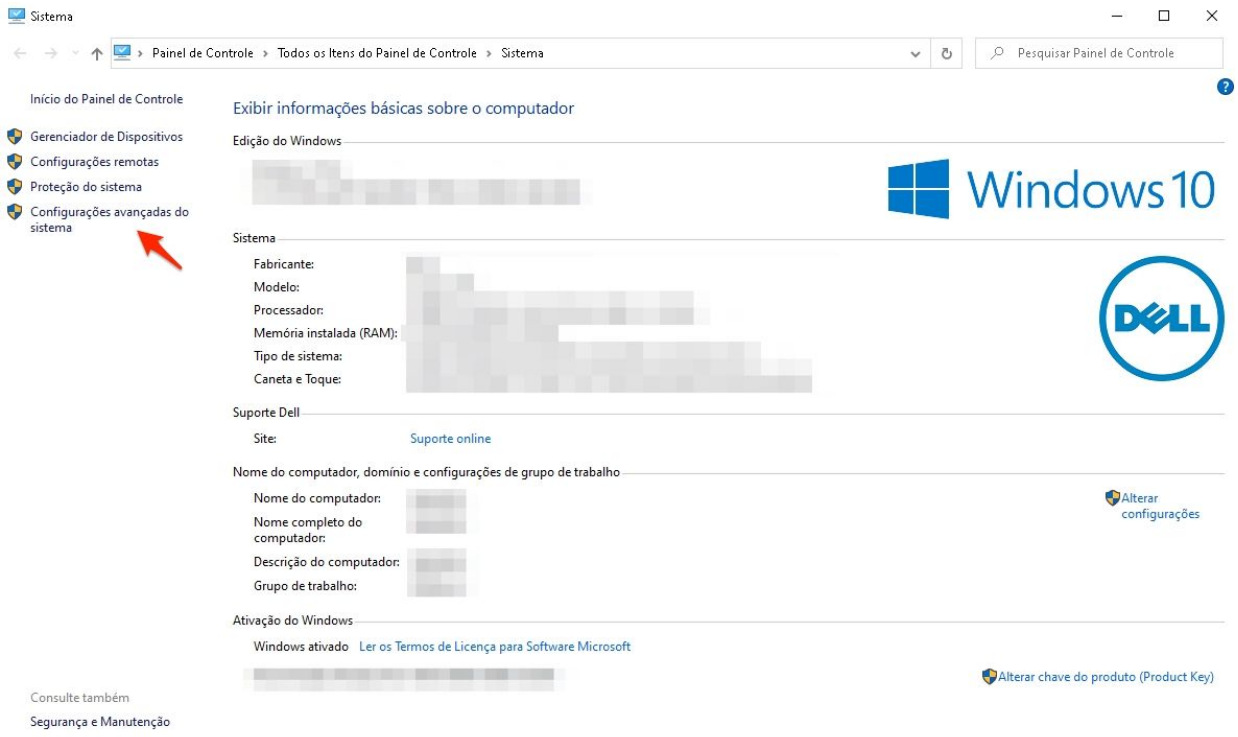
Passo 03 – Abra Painel de Controle > Sistema

Clique com o botão direito do mouse em Este Computador e em seguida em Propriedades.



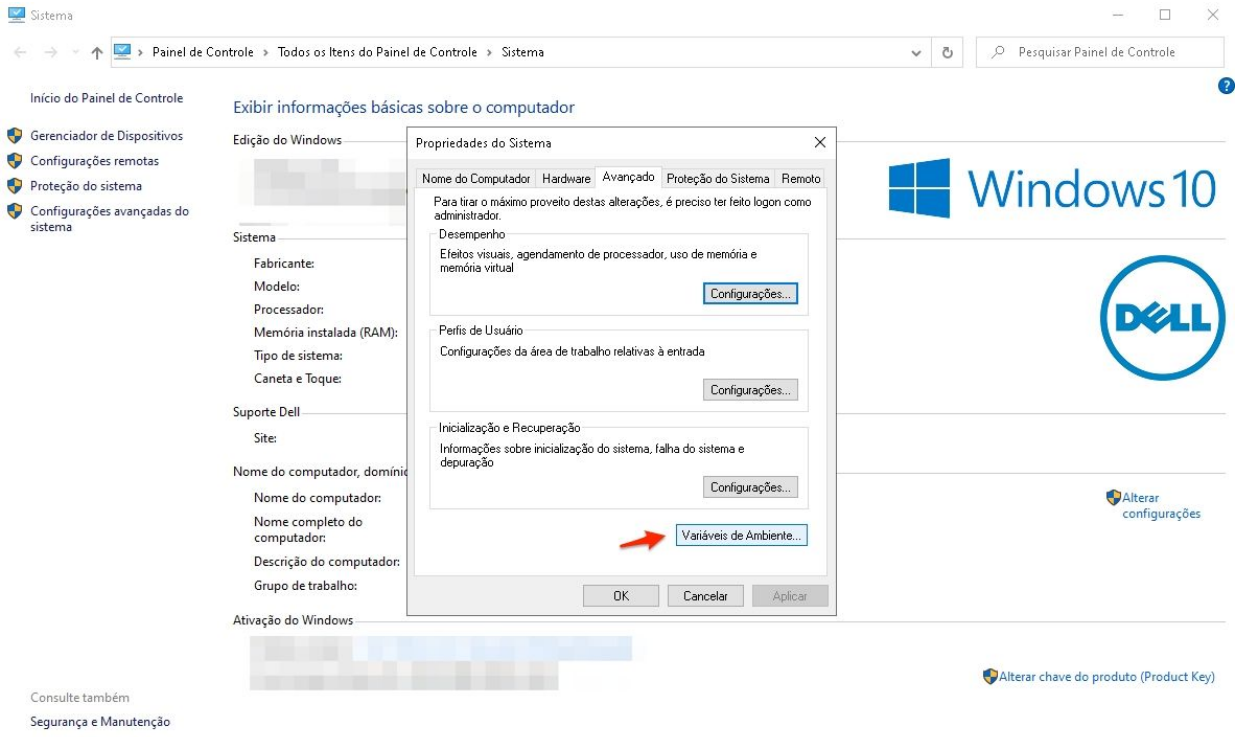
Passo 04 – Abra Configurações avançadas do sistema

Clique em Configurações avançadas do sistema.



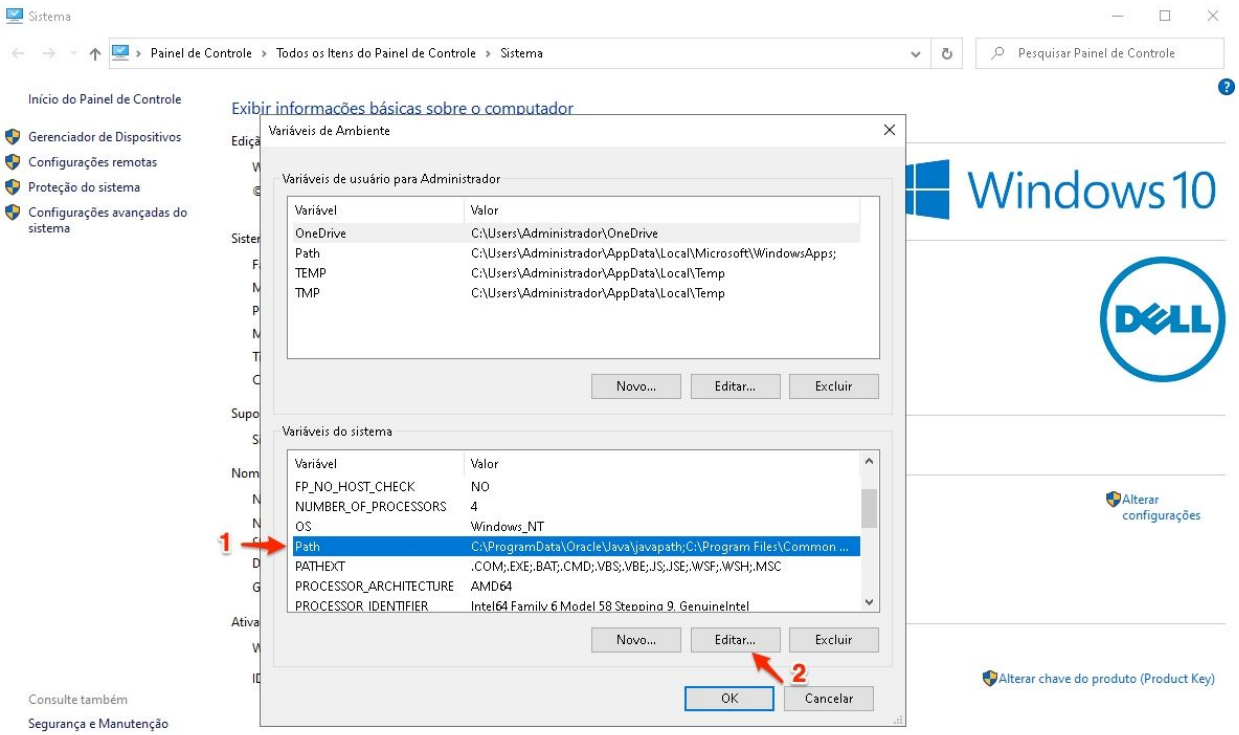
Passo 05 – Abra Variáveis de Ambiente

Clique em Variáveis de Ambiente.



Passo 06 – Edite o Path

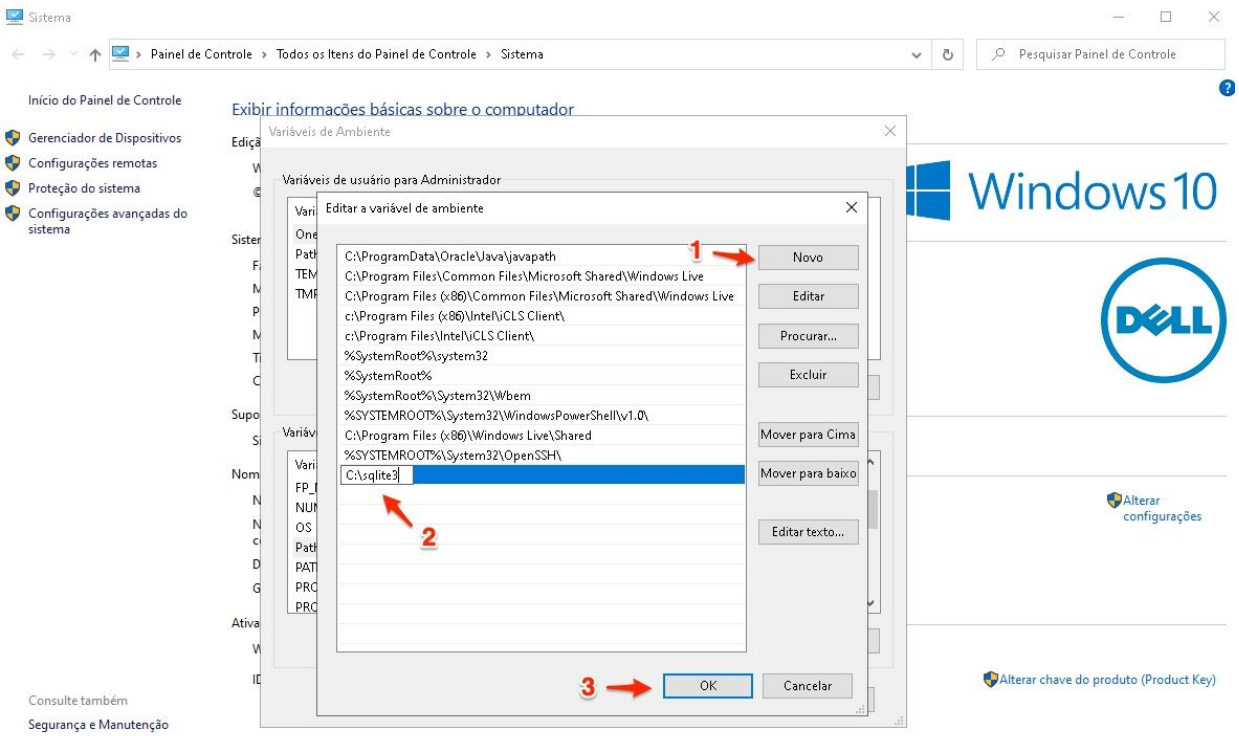
Clique em Path e logo após no botão Editar.



Passo 07 – Nova Variável de Ambiente

Adicione uma nova Variável de ambiente apontando para o caminho da pasta onde foram descompactados os arquivos do sqlite3, C:\sqlite3 provavelmente.

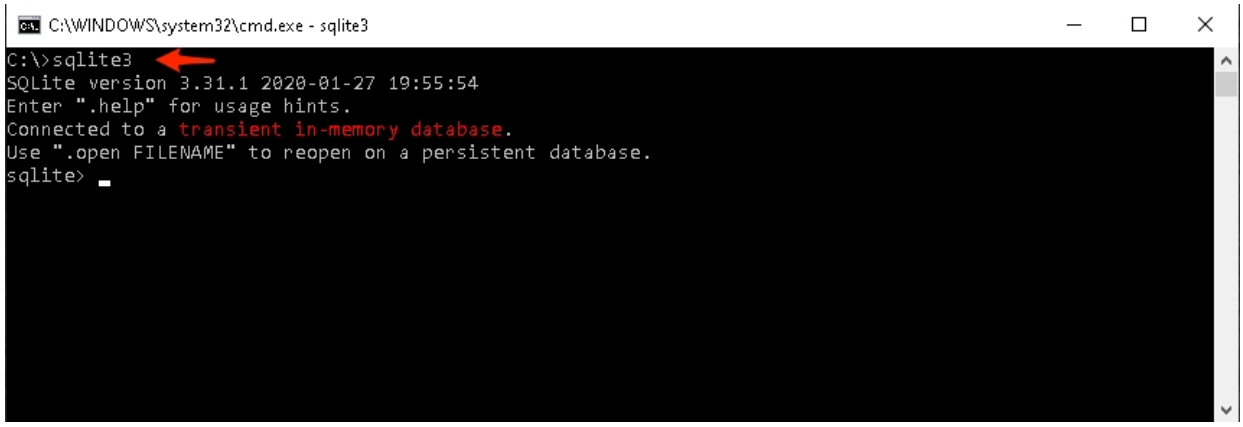
1. Clique no botão Novo.
2. Adicione o caminho da pasta sqlite3, **por exemplo** "C:\sqlite3".



Passo 08 – Verificação da Instalação

Verifique se a instalação foi bem-sucedida da seguinte forma:

1. Abra o Prompt de Comando.
2. Digite sqlite3.
3. Se o resultado for como o apresentado na imagem abaixo, parabéns, o SQLite foi instalado com sucesso.



AULA 04 – INSTALANDO NO MACOS

Verificação de versão pré-instalada

O SQLite vem pré-instalado no macOS e está localizado no diretório /usr/bin/.

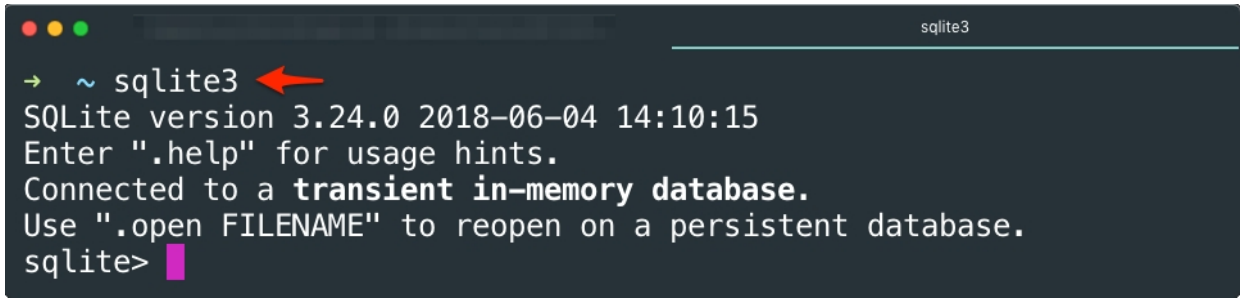
Vamos verificar se o SQLite está de fato instalado e funcionando como deveria.

- Abra o Terminal e digite “sqlite3”.

Se aparecer algo parecido com o que é exibido na imagem abaixo, o SQLite está devidamente instalado.

Pule para a próxima aula.

Observação: Para sair do sqlite digite “.quit”.

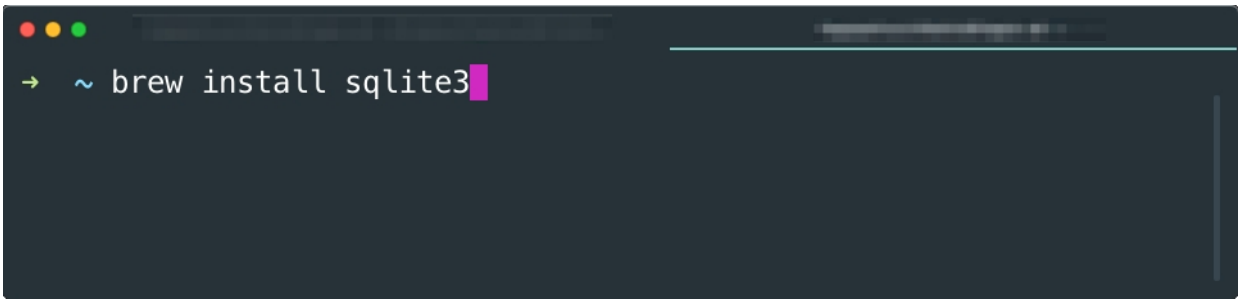


Instando o SQLite no macOS

Instale com o brew.

- brew install sqlite3

Caso não tenha o brew instalado, instale a partir do site: https://brew.sh/



AULA 05 – OPERAÇÕES BÁSICAS

Criando o Banco de Dados

Para criar um novo Banco de Dados, digite a seguinte linha, como mostra a imagem abaixo.

- sqlite3 banco.sqlite

banco.sqlite é o nome do arquivo com os dados – você pode usar o nome que quiser.

CREATE TABLE

Abaixo a Tabela *PESSOAS* sendo criada no Banco de Dados.

Observação: Digite a linha completa e aperte Enter ao final para ir para a próxima linha.

INSERT INTO

Abaixo temos a inserção de dados na Tabela *Pessoas*.

SELECT

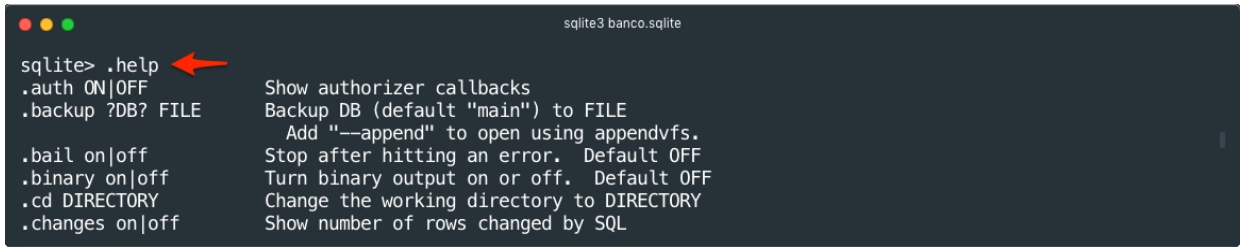
Abaixo a seleção dos dados da Tabela *PESSOAS*.

Comandos Especiais (dot-commands)

Linhas que começam com “.” são interpretadas com um comando especial dado ao Banco de Dados.

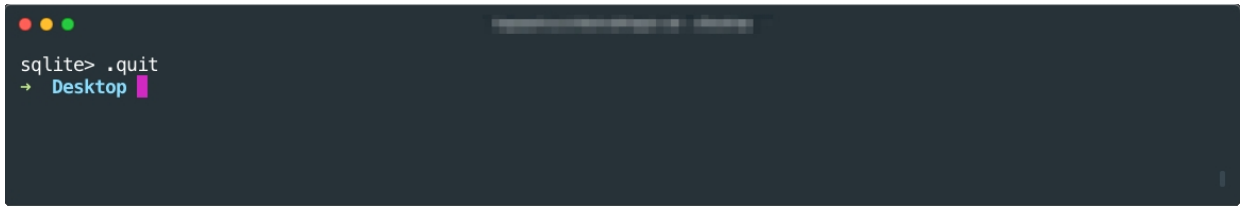
.help

Para ver a lista de comandos disponíveis, digite .help, como mostra a imagem abaixo.



.quit

Um dos comandos disponíveis é o comando .quit, usado para sair do Banco de Dados.



Export CSV

Abaixo temos um Conjunto de Resultados sendo exportado para um arquivo no formato .csv.

- .headers on: Define que o nome das colunas será escrito no arquivo;
- .mode csv: Define o tipo do arquivo;
- .once export.csv: Define que o arquivo será exportado uma única vez e o nome do arquivo será export.csv.
- SELECT: Define a seleção dos dados exportados.

Mais Informações

Para maiores informações consulte a documentação do SQLite.

https://sqlite.org/cli.html

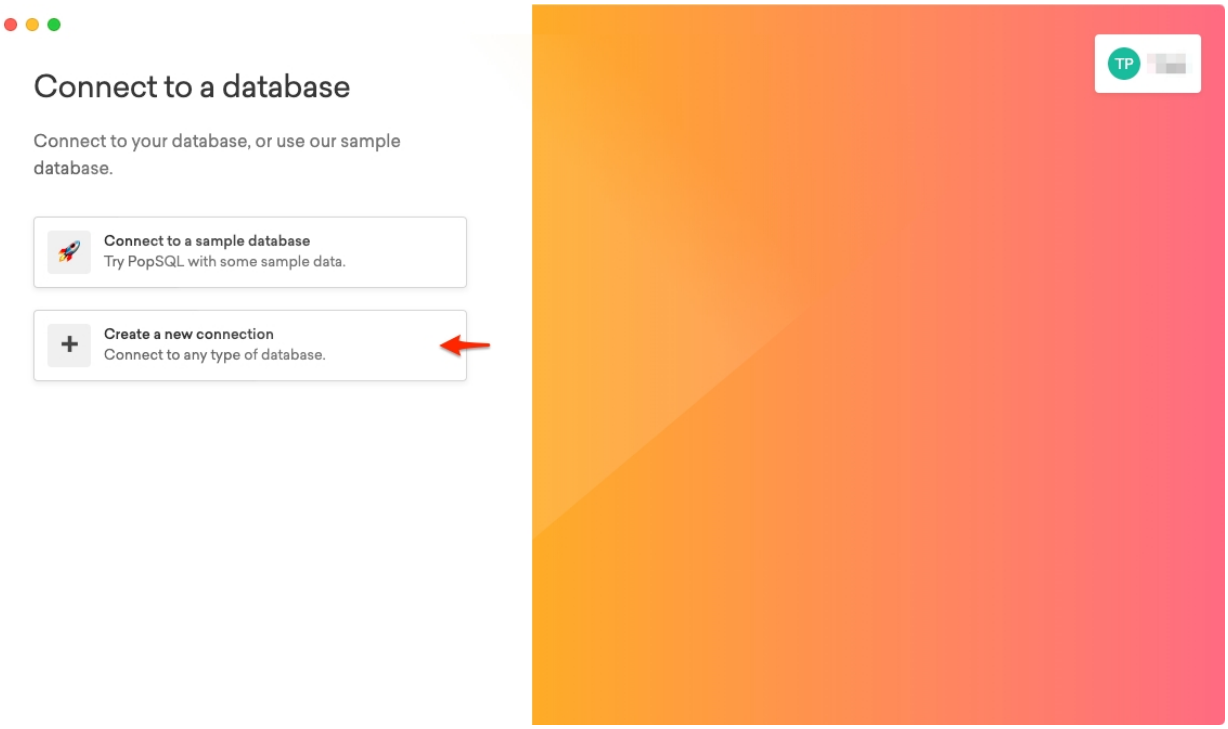
AULA 06 – APLICATIVOS

Temos diversos Apps que Gerenciam Banco de Dados SQLite, dentre eles, selecionamos três, o PopSQL, SQLiteBrowser e Beekeeper Studio (recomendado).

PopSQL

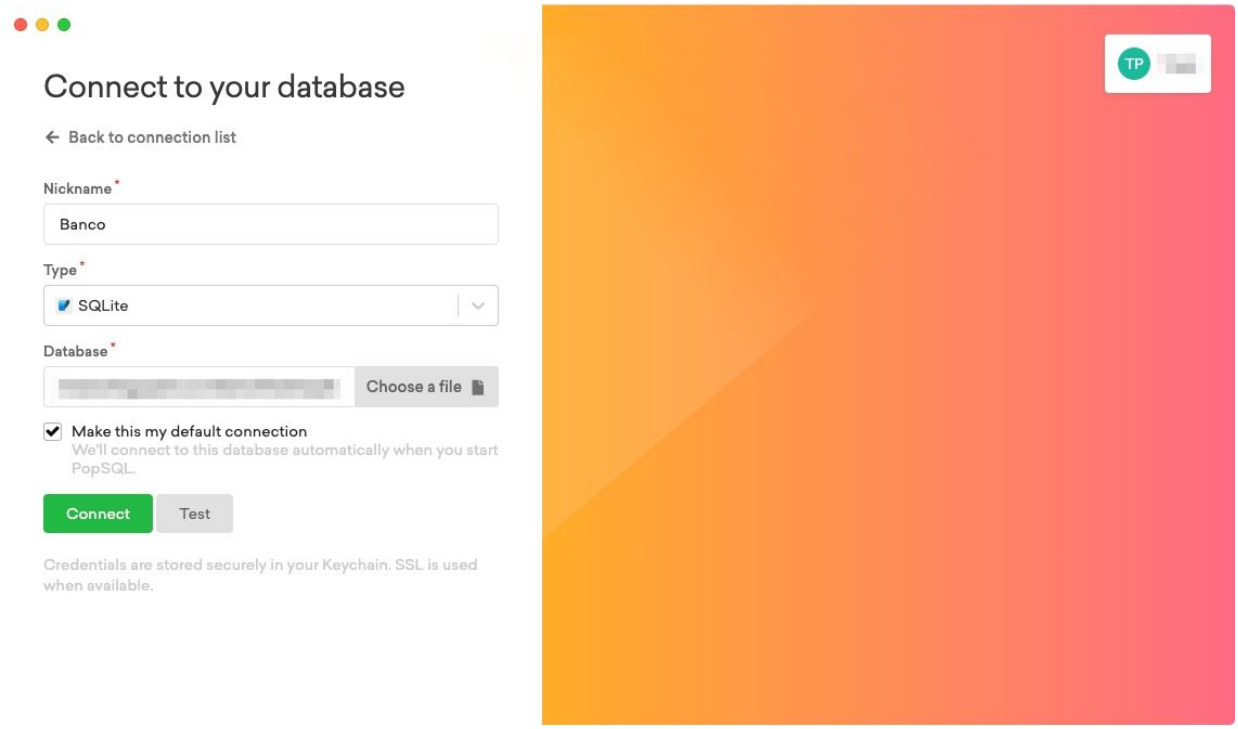
Passo 01 – Download e Instalação

- Faça o download: https://popsql.com/
- Instale;
- Abra o app e crie uma nova conexão com o Banco de Dados, conforme imagem;

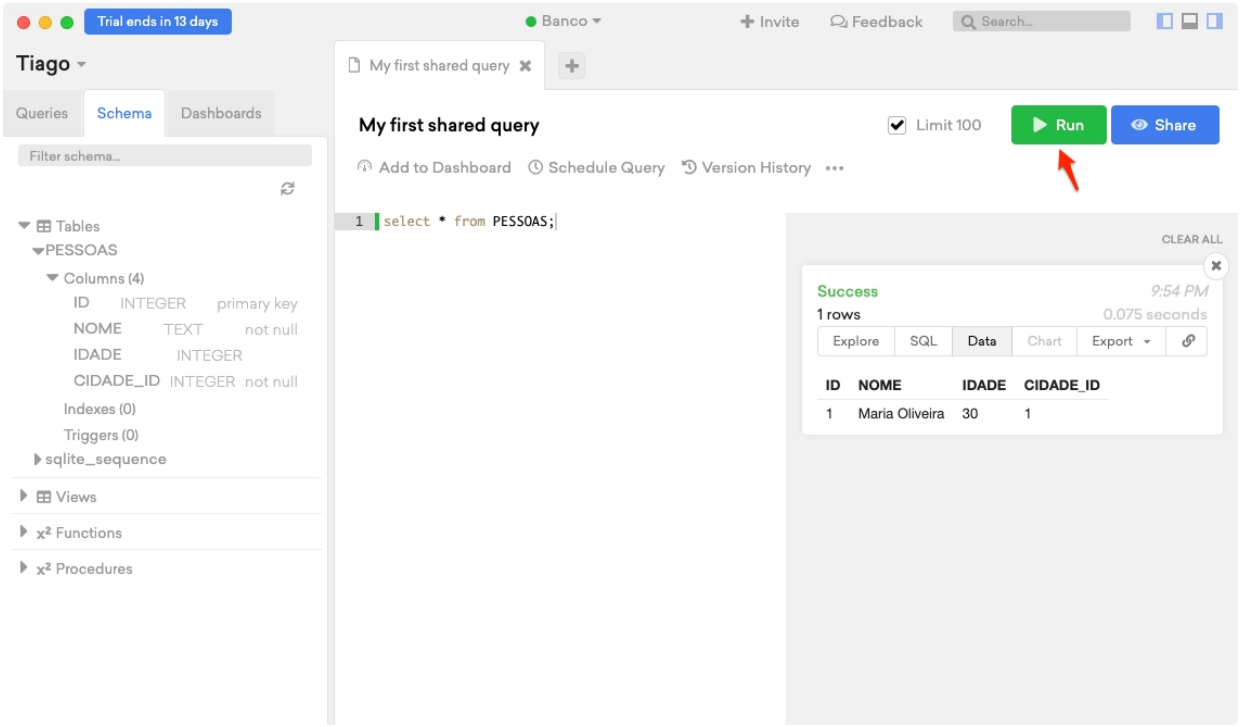


Passo 02 – Conexão com o Banco de Dados

Para criar uma conexão com o Banco de Dados informe os dados conforme imagem abaixo.



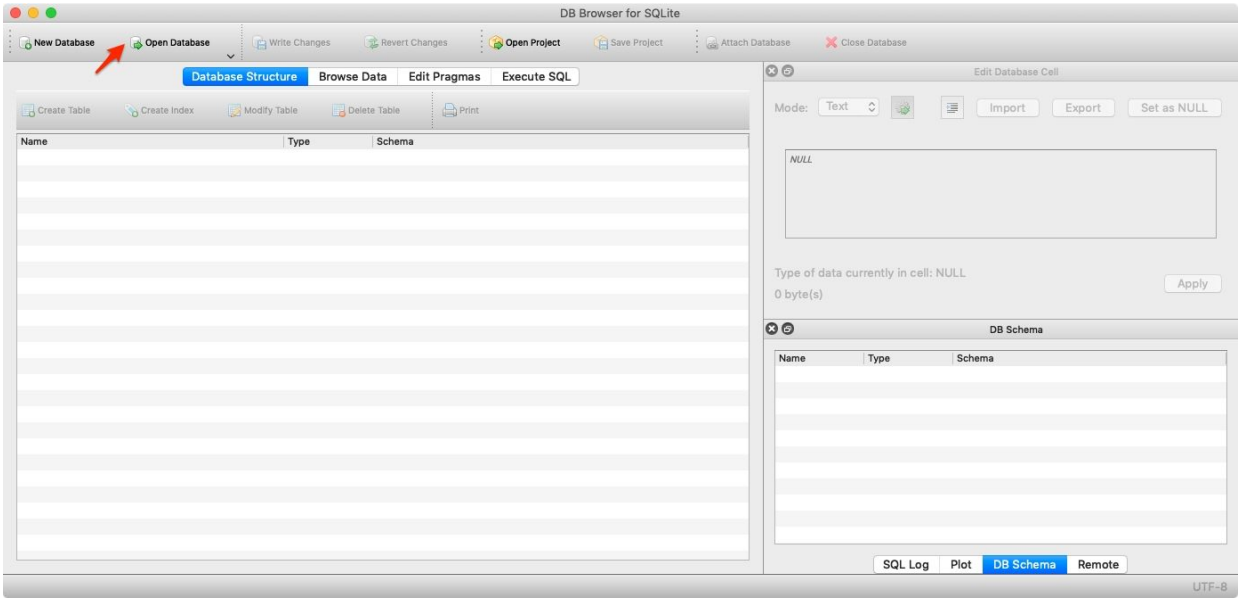
Passo 03 – Teste o App



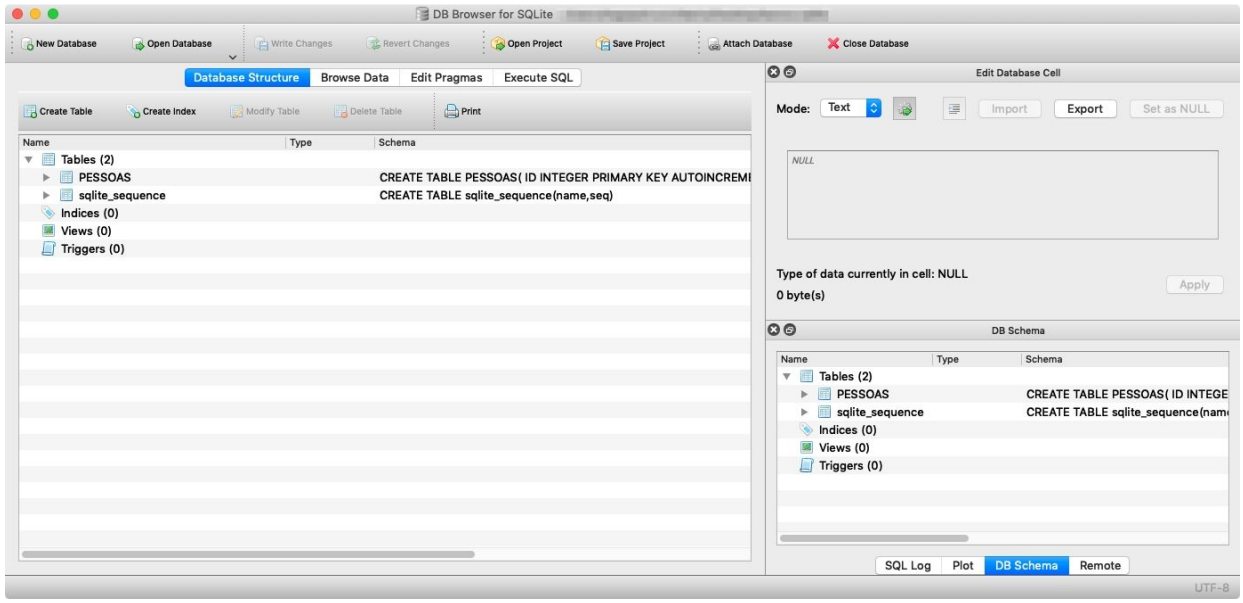
DB Browser for SQLite

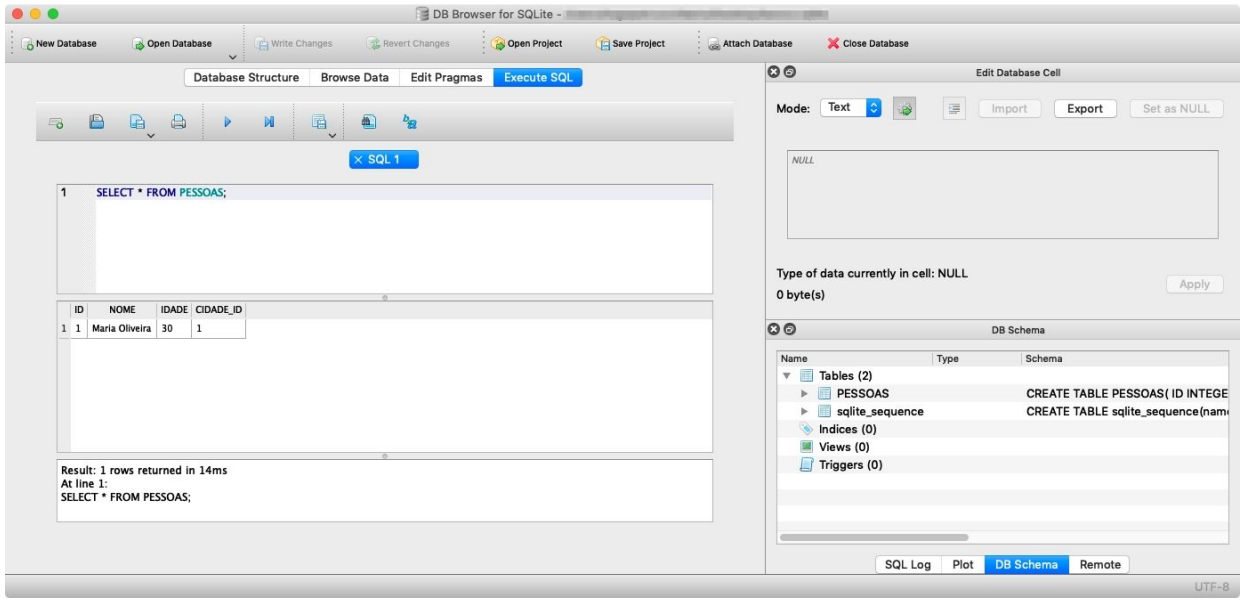
Passo 01 – Download e Instalação

- Faça o download: https://sqlitebrowser.org/
- Instale;
- Abra o app e crie uma nova conexão com o Banco de Dados, conforme imagem;



Passo 02 – Teste o App


DB Browser for SQLite
New Database
Open Database
Write Changes
Revert Changes
Open Project
Save Project
Attach Database
Close Database
Database Structure
Browse Data
Edit Pragmas
Execute SQL
Create Table
Create Index
Modify Table
Delete Table
Print
Name
Type
Schema
Tables (2)
PESSOAS
CREATE TABLE PESSOAS(ID INTEGER PRIMARY KEY AUTOINCREMI
sqlite_sequence
CREATE TABLE sqlite_sequence(name,seq)
Indices (0)
Views (0)
Triggers (0)
Edit Database Cell
Mode:
Text
Import
Export
Set as NULL
NULL
Type of data currently in cell: NULL
0 byte(s)
Apply
DB Schema
SQL Log
Plot
DB Schema
Remote
UTF-8


DB Browser for SQLite -
New Database
Open Database
Write Changes
Revert Changes
Open Project
Save Project
Attach Database
Close Database
Database Structure
Browse Data
Edit Pragmas
Execute SQL
SQL 1
1 SELECT * FROM PESSOAS;
ID
NOME
IDADE
CIDADE_ID
1 1 Maria Oliveira 30 1
Result: 1 rows returned in 14ms
At line 1:
SELECT * FROM PESSOAS;
Edit Database Cell
Mode:
Text
Import
Export
Set as NULL
NULL
Type of data currently in cell: NULL
0 byte(s)
Apply
DB Schema
SQL Log
Plot
DB Schema
Remote
UTF-8

Beekeeper Studio

Faça o download através da seguinte url: https://www.beekeeperstudio.io

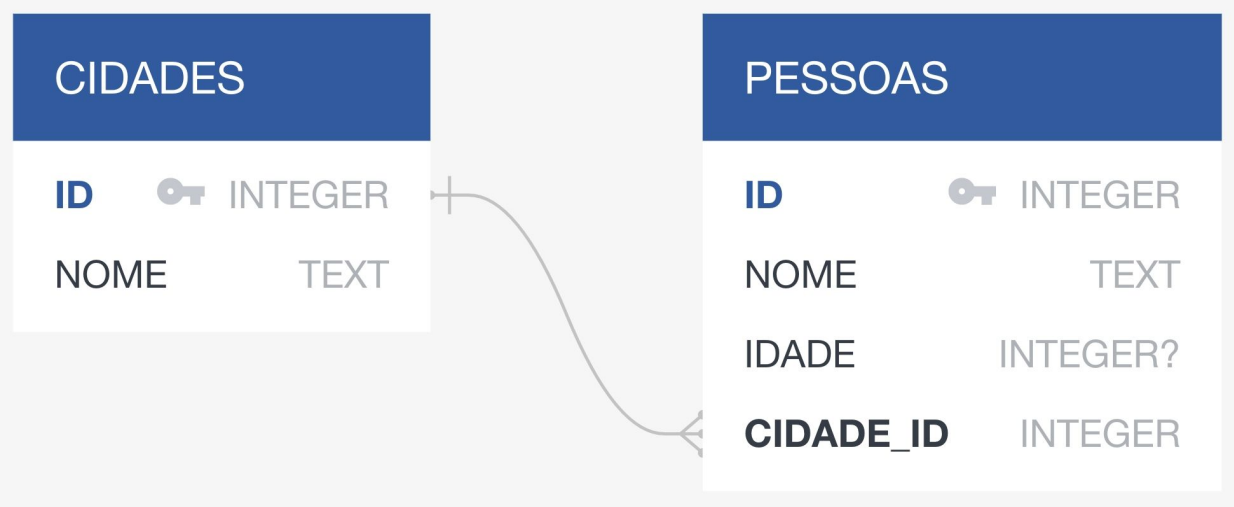
MÓDULO BÔNUS – PYTHON E SQLITE

AULA 01 – INTRODUÇÃO

Nesta aula veremos como interagir com o Banco de Dados SQLite através do Python.

- Como criar novo Banco de Dados, a partir do Python;
- Como se conectar ao Banco de Dados;
- Como criar tabelas;
- Como inserir dados;
- E como Selecionar os dados inseridos;

Criaremos as Tabelas descritas no Diagrama (DER) abaixo.



Vamos nessa!

AULA 02 – CRIANDO UM BANCO DE DADOS COM PYTHON

Para toda e qualquer conexão e interação com o SQLite é necessária a importação da biblioteca sqlite3. Ela já vem por padrão no Python, não sendo assim necessária a sua instalação.

Para criar o Banco de Dados, basta criar uma conexão através do código fonte descrito na linha 3 informando o nome que você deseja para o Banco de Dados, juntamente com a extensão .sqlite.

O Banco de Dados SQLite tem uma característica interessante que é a de salvar os dados em um arquivo "comum", como se fosse um arquivo .xls, como podemos ver abaixo, ao contrário de outros Bancos de Dados que mantém os dados em locais "obscuros".

Após criarmos o Banco de Dados e ter como retorno uma *connection*, criamos um cursor.

De uma forma simplificada, o cursor nos dará a possibilidade de executar scripts SQL.

```
1 import sqlite3
2
3 conn = sqlite3.connect('../db_m05.sqlite')
4 cur = conn.cursor()
```

https://bit.ly/3cfDnRN

AULA 03 – CRIANDO AS TABELAS

Nesta aula criamos duas Tabelas, a *CIDADES* e *PESSOAS*.

- **3:** Criamos uma conexão com um banco já criado, na aula passada;
- **4:** Criamos o cursor;
- **6-20:** Criamos as tabelas executando os devidos scripts SQL;
- **22:** Os comandos executados são "comitados". De uma forma simples, é confirmada a operação;

```python
import sqlite3

conn = sqlite3.connect('../db_m05.sqlite')
cur = conn.cursor()

cur.execute("""
  CREATE TABLE CIDADES (
    ID INTEGER PRIMARY KEY AUTOINCREMENT,
    NOME TEXT NOT NULL
  );
""")

cur.execute("""
  CREATE TABLE PESSOAS (
    ID INTEGER PRIMARY KEY AUTOINCREMENT,
    NOME TEXT NOT NULL,
    IDADE INTEGER,
    CIDADE_ID INTEGER NOT NULL
  );
""")

conn.commit()
```

https://bit.ly/3cfDnRN

AULA 04 – INSERINDO OS DADOS

Nesta aula inserimos os dados nas tabelas, executando os devidos scripts SQL.

```python
import sqlite3

conn = sqlite3.connect('../db_m05.sqlite')
cur = conn.cursor()

cur.execute("""
  INSERT INTO CIDADES(NOME) VALUES
    ('Brasília'),
    ('São Paulo'),
    ('São Joaquim');
""")

cur.execute("""
  INSERT INTO PESSOAS(NOME, IDADE, CIDADE_ID) VALUES
    ('Maria Oliveira',  30, 1),
    ('Priscila da Silva',   79, 1);
""")

conn.commit()
```

https://bit.ly/3cfDnRN

AULA 05 – SELECIONANDO OS DADOS

Nesta aula selecionamos os dados através de três métodos diferentes do cursor, sendo que o que difere um do outro é a quantidade de resultados que cada um retorna.

fetchone()

Executa o script SQL e retorna uma única linha de resultado.

fetchmany()

Executa o script SQL e retorna várias linhas, sendo a quantidade definida no parâmetro do método.

fetchall()

Executa o script SQL e retorna todas as linhas – sendo que a quantidade pode ser limitada no script.

```python
import sqlite3

conn = sqlite3.connect('../db_m05.sqlite')
cur = conn.cursor()

cur.execute("SELECT * FROM CIDADES;")
primeiro_resultado = cur.fetchone()
print(f'\nPrimeira linha: {primeiro_resultado}')

cur.execute("SELECT * FROM CIDADES;")
dois_primeiros = cur.fetchmany(2)
print(f'\nDuas primeiras linhas: {dois_primeiros}')

cur.execute("SELECT * FROM CIDADES;")
todos = cur.fetchall()
print(f'\nTodos os resultados: {todos}\n')
```

https://bit.ly/3cfDnRN

AGRADECIMENTO

Sou muito grato por você ter nos escolhidos em sua jornada do aprendizado de Banco de Dados.

Se você se interessa por programação, visite nosso site produtividadeprogramada.com.br

Que você tenha uma vida longa e próspera.