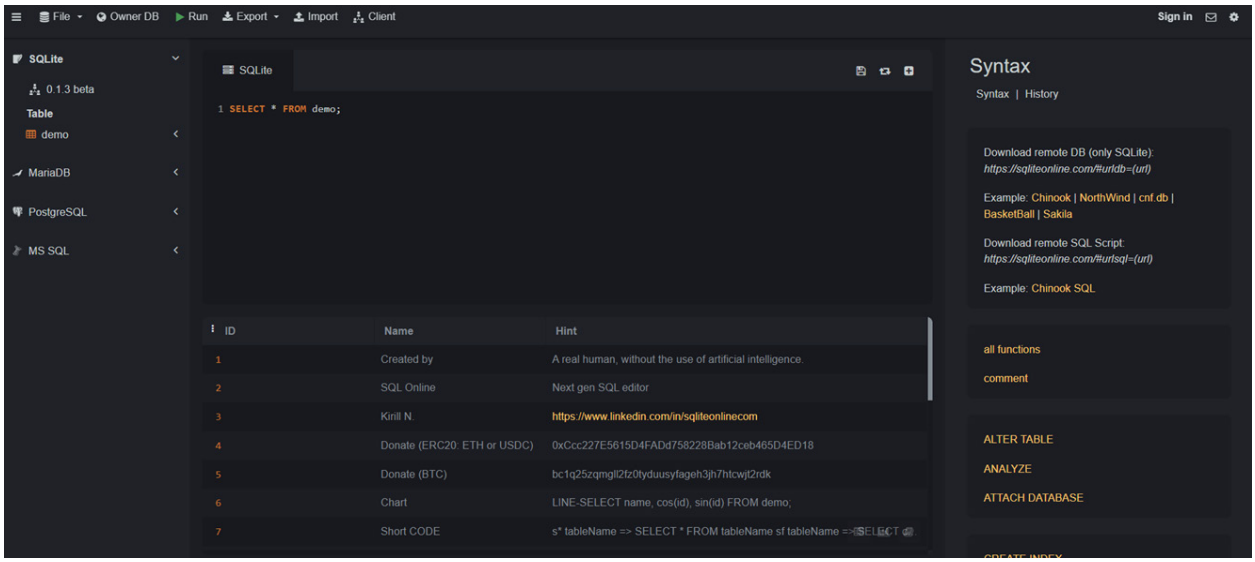
# TUTORIAL: SQL – APLICANDO O CONHECIMENTO

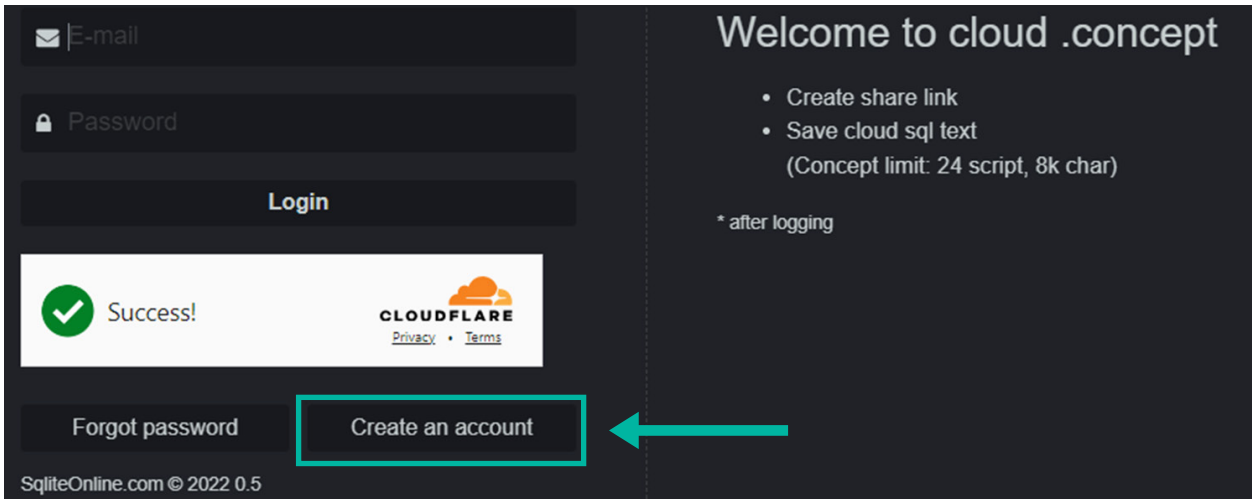
## MÓDULO I

## VAMOS AO PASSO A PASSO – PARTE I:

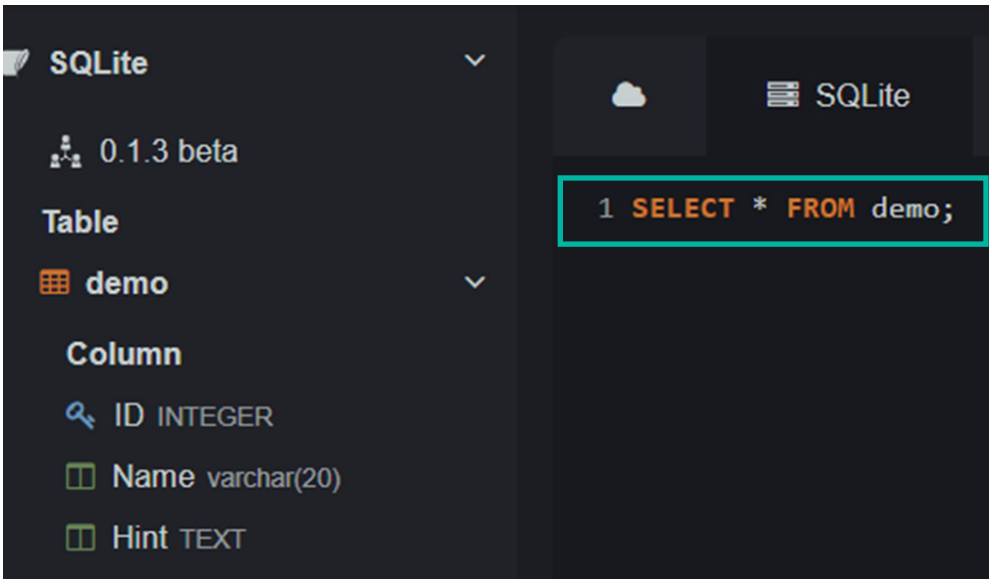
**A.** Acesse o link https://sqliteonline.com/. Você será transferido para uma tela como a abaixo:



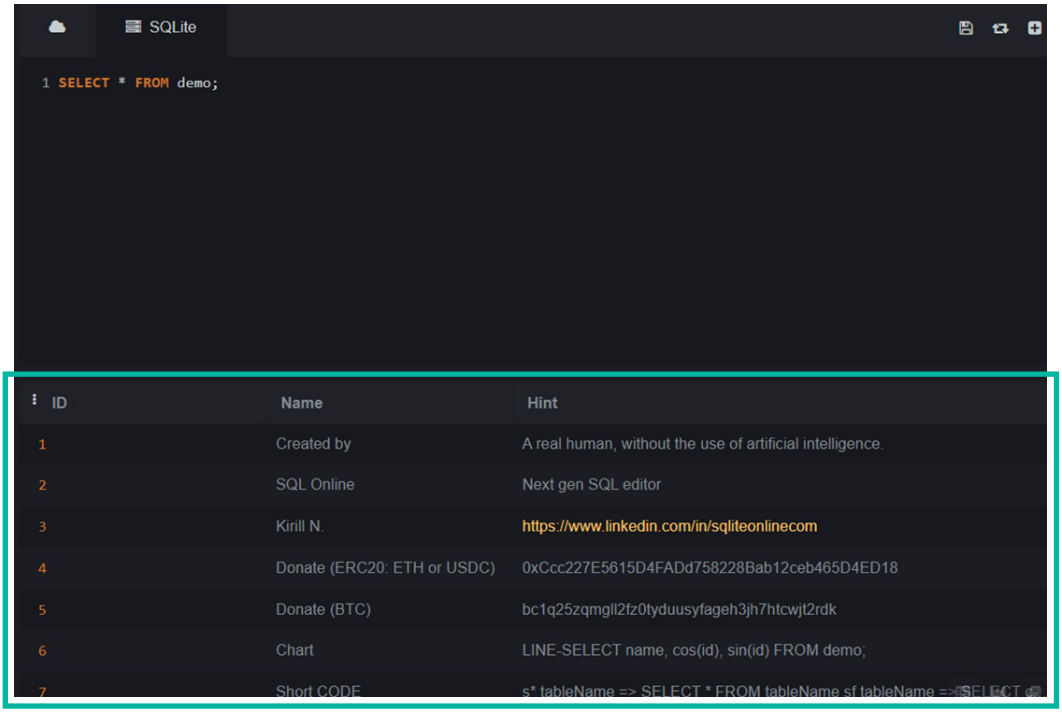

**B.** Crie uma conta de usuário, para que scripts simples não sejam perdidos. Para isso, vá ao canto direito superior da tela, e clique em "sign in". Uma tela de pop-up se abrirá, e você poderá criar a sua conta.



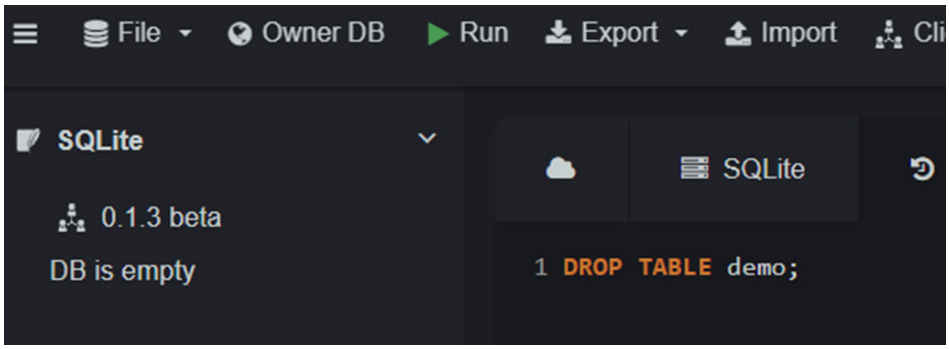
**C.** Agora, vamos aproveitar e já utilizar um comando em SQL: o Drop. Como este site é para pequenos experimentos e aprendizados, uma tabela chamada demo já está criada, como podemos ver a seguir. Como é necessário aplicar o conhecimento desde o zero, vamos deletar essa tabela. E nada melhor que utilizar o comando DROP. Lembrete: o comando DROP em SQL é utilizado para excluir um objeto do banco de dados, como uma tabela, índice, visão ou procedimento armazenado. Sendo assim, vamos à nossa tela de inserção de scripts. Na parte que está escrito SELECT * FROM demo; será onde escreveremos nossos comandos, tanto de inserção de dados quanto de criação de tabelas e consultas de registros.




**D.** Na parte logo abaixo, quando realizamos consultas, é possível ver os registros. Recorda-se que tínhamos os campos ID, NAME e HINT? Verifique na figura os registros que já tínhamos.

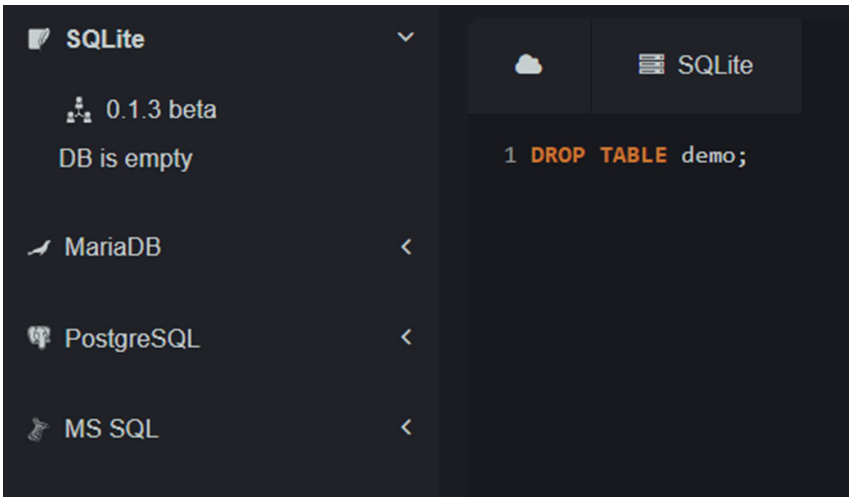


**E.** Agora, onde está escrito o comando de SELECT, apague e insira o comando DROP demo, devendo ficar como a figura abaixo mostra. Para o comando ser executado, precisamos clicar em RUN, no menu superior da nossa tela preta.

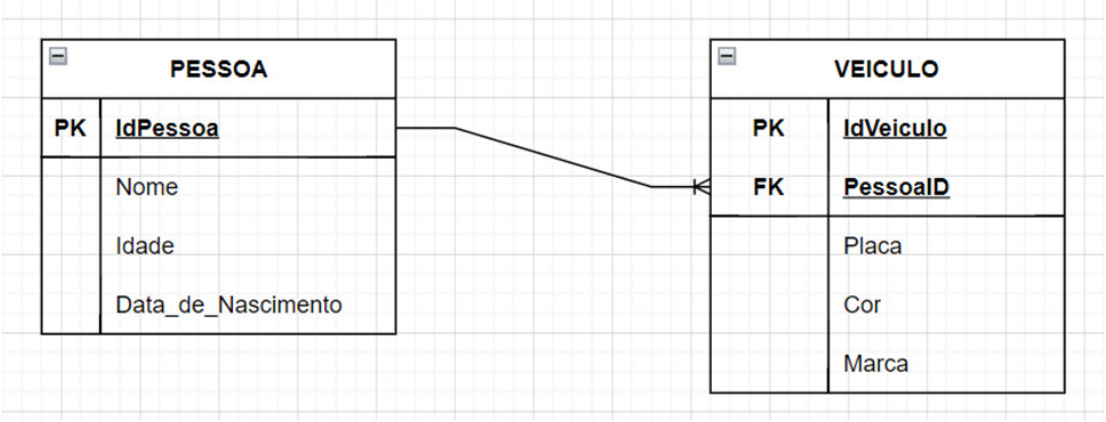



**F.** O DB está sem tabela alguma. Ótimo!

Primeiro comando seu executado com sucesso. Conseguimos realizar um **DROP** na tabela de demonstração do nosso pseudo sistema gerenciador de banco de dados. Disse pseudo porque esse sistema online possui cache temporário e é utilizado apenas para situações de aprendizado mesmo.




Como disse anteriormente, você irá utilizar o diagrama entidade-relacionamento PESSOA-VEÍCULO que você criou anteriormente.



A primeira etapa que precisará realizar será a de transformarmos nossas entidades em tabelas reais no banco de dados.

## VAMOS AO PASSO A PASSO – PARTE 2:

**A.** Continuando dos passos anteriores, utilize o comando CREATE para criar a tabela CLIENTE. A tabela possui os atributos NOME, IDADE e DATA_DE_NASCIMENTO. Ou seja, a primeira linha será CREATE TABLE pessoa. Aqui temos detalhes importantes:

- Os parênteses informam que dentro deles deverá haver entradas, que seriam quais colunas vamos inserir nas nossas tabelas.
- O ponto-e-vírgula no final é crucial. Esse ponto é muito esquecido pelos desenvolvedores iniciantes, e muitas vezes os compiladores mencionam que são outros erros, mas não o ponto-e--vírgula. Por isso, sugiro prestar atenção (mais ainda!).

```
1 CREATE TABLE pessoa (
2
3 );
```

**B.** Agora você irá inserir as colunas, ou seja, os atributos da nossa entidade pessoa, que está para se tornar uma tabela de banco de dados. Para isso, peço que copie o código abaixo na sua tela de script, do site que está aberto, e que a seguir vou lhe explicar:

```
1 CREATE TABLE PESSOA (
2     nome VARCHAR(100),
3     idade INT,
4     data_de_nascimento DATE
5 );
```

**C.** Todo atributo necessita ser catalogado como um tipo. Como parâmetros de entrada temos:

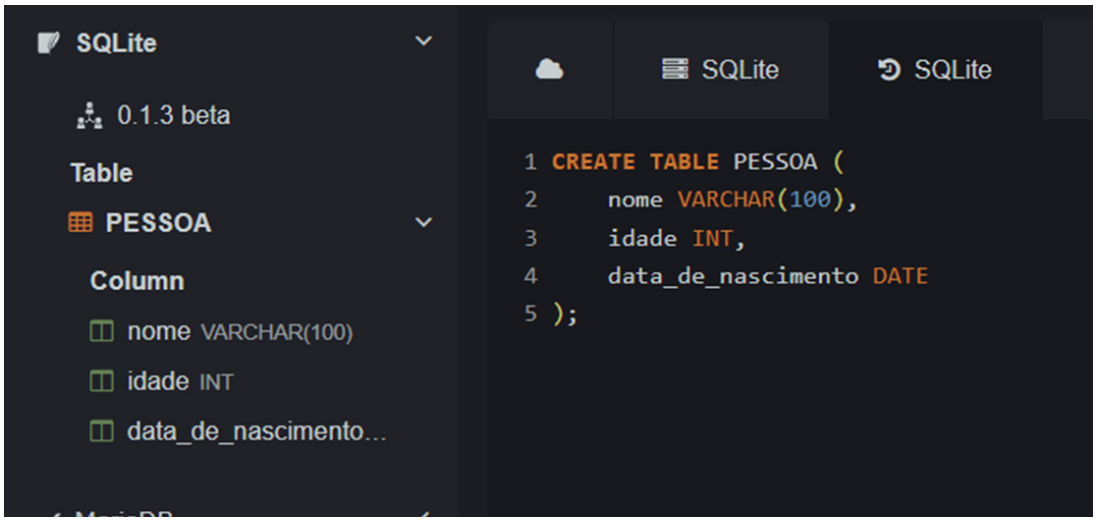
- Nome, que é um texto, certo? No SQL chamamos de VARCHAR. E necessitamos colocar um número de caracteres máximo para esse tipo, para evitar utilização excessiva de memória. Lembre-se, de seus conhecimentos anteriores, alocar memória e não a utilizar faz com que seus programas ou aplicações se tornem lentos e inviáveis de serem utilizados.
- Idade, que basicamente é um numeral. Como não trabalharemos com números racionais, apenas inteiros, vamos declarar esse atributo como INT.

- Por último, mas não menos importante, temos DATA_DE_NASCIMENTO, em que temos um tipo específico chamado DATE.

Como já feito anteriormente, reavalie se o código foi escrito corretamente e clique em RUN, na sua tela do site em que você está rodando o SQL script.

**D. IMPORTANTE:** caso você já tenha conhecimento em SQL e em IDEs (programas que utilizamos para gerenciar os bancos de dados), pode utilizá-los também. Fica a seu critério!

Feito isso, tem-se de resultado a tabela PESSOA criada:




**E.** Antes de inserir registros nessa tabela, você se recorda do nosso diagrama entidade-relacionamento, em que havia um campo de identificação chamado IdPessoa?

Não se preocupe. Não pedi para você inserir de propósito. Ele é de extrema importância para garantirmos a integridade dos relacionamentos do nosso banco de dados. Não é interesse seu que os VEÍCULOS sejam vinculados a mais de uma pessoa, correto?

Deve-se respeitar o conceito de **imutabilidade** – uma vez definida, a chave primária não deve ser alterada. Isso garante a integridade dos dados e a consistência das relações entre as tabelas.

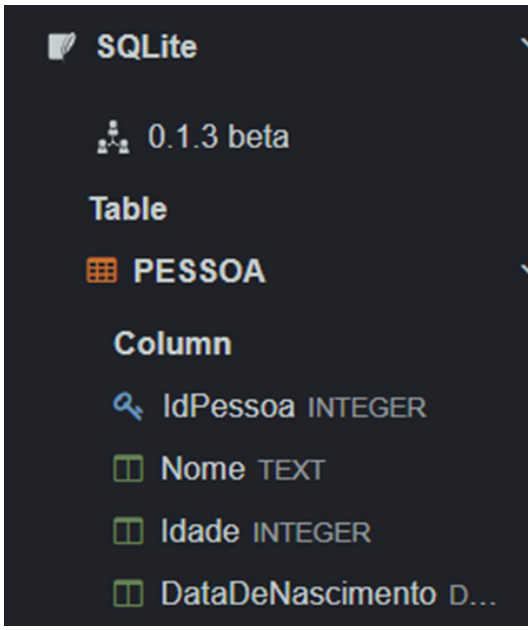
Então, vamos utilizar o DROP novamente nessa tabela e vamos criar agora com a chave primária!

**F.** O SQLite que você está utilizando agora não consegue fazer a alteração diretamente. Mais adiante vamos trabalhar com SGDBs mais poderosos que lhe deixarão fazer isso. Com o código abaixo, conseguimos deletar a tabela pessoa, que não possuía chave primária. Segundo ponto, criaremos novamente a tabela pessoa, agora com IdPessoa sendo um número inteiro, e agora definido como PRIMARY KEY, ou seja, chave primária, além de ser auto-increment, que fará com que se inicie a contagem em 1, e não seja necessário um controle das próximas inserções de registros na tabela pessoa.

```sql
-- Drop tabela pessoa se existir
DROP TABLE IF EXISTS pessoa;

-- Criação da tabela pessoa com a nova estrutura
CREATE TABLE PESSOA (
    IdPessoa INTEGER PRIMARY KEY AUTOINCREMENT,
    Nome TEXT,
    Idade INTEGER,
    DataDeNascimento DATE
);
```

**G.** Como resultado, temos a tabela PESSOA criada. Agora podemos inserir registros do tipo da entidade PESSOA em nossa tabela.



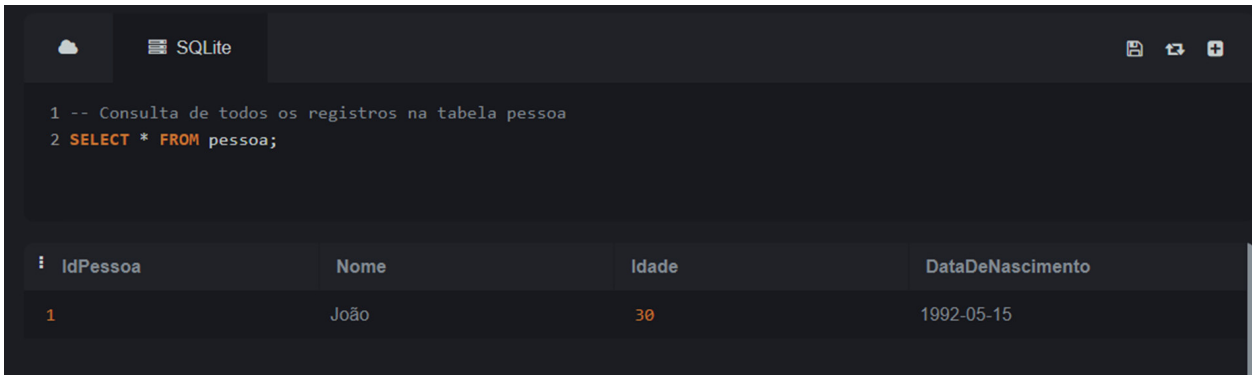

**H.** Como será feito isso? Utilizaremos os comandos INSERT. A operação INSERT serve para inserirmos registros em nossa tabela. O conector INTO relaciona qual tabela, no caso PESSOA.

**Importante:** quando definimos um IdPessoa como auto-increment, ele não precisa vir como atributo, pois será definido automaticamente. Agora você pode consultar todos os registros inseridos na sua tabela PESSOA.

```sql
-- Inserção de um registro na tabela pessoa
INSERT INTO pessoa (Nome, Idade, DataDeNascimento)
VALUES ('João', 30, '1992-05-15');
```

**I.** Execute o comando SELECT e vamos explicar detalhadamente o que ocorre:

- O comando SELECT realiza consultas em sua tabela.
- Quando citamos * , significa que queremos todos os registros existentes.
- From traduzimos literalmente do inglês: qual será a origem?
- E por último, qual tabela, no caso, PESSOA.



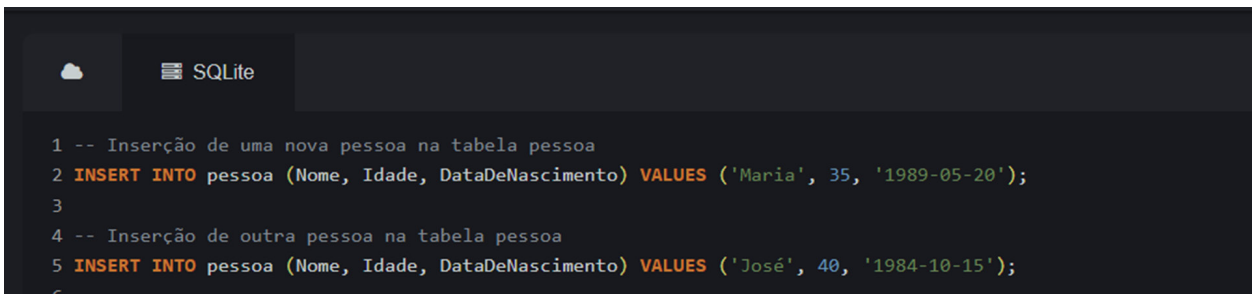

**J.** No prompt inferior, aparecem as colunas IdPessoa, Nome, Idade e DataDeNascimento e abaixo todos os registros, divididos por colunas.

Parabéns! Você atingiu uma etapa fundamental aqui no seu curso de Desenvolvimento Back-End. Criou diagramas, tabelas e realizou consultas! Ótimo!

Agora, vamos inserir mais duas pessoas?

## PASSO A PASSO – PARTE 3:

**A.** Se fizer a consulta de SELECT * FROM Pessoa, você obterá:




**B.** E se você quiser apenas o segundo da lista, que seria a pessoa com nome Maria?

SQLite SQLite

```
1 -- Consulta de todos os registros na tabela pessoa
2 SELECT * FROM pessoa;
```

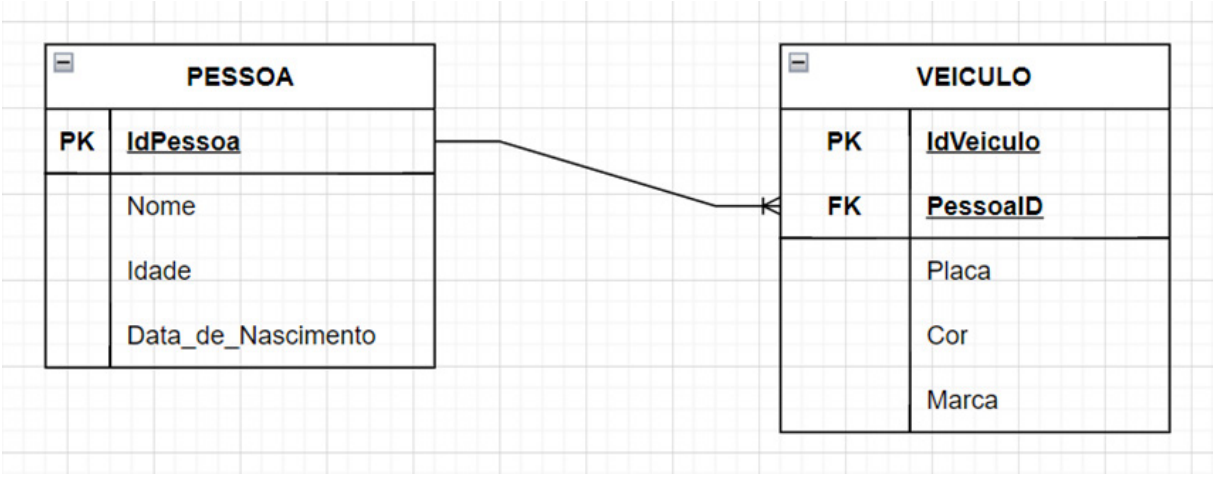
| IdPessoa | Nome | Idade | DataDeNascimento |
|---|---|---|---|
| 1 | João | 30 | 1992-05-15 |
| 2 | Maria | 35 | 1989-05-20 |
| 3 | José | 40 | 1984-10-15 |

**C.** Então, se insere a cláusula WHERE, informando qual Pessoa relacionada a uma específica IdPessoa você quer que apareça. Nesse caso, 2.

```
1 -- Consulta de todos os registros na tabela pessoa
2 SELECT * FROM pessoa WHERE IdPessoa = 2;
```

| IdPessoa | Nome | Idade | DataDeNascimento |
|---|---|---|---|
| 2 | Maria | 35 | 1989-05-20 |

**D.** Agora, para continuar nossa reflexão, precisará fazer o mesmo com a tabela veículos, seguindo nosso diagrama:



**E.** Para isso, teremos que criar uma tabela VEICULO, com a chave primária IdVeiculo, a chave estrangeira PessoaID, os atributos Placa, Cor e Marca. Para entender o script abaixo, a primeira linha, como já explicado, será a chave primária, IdVeiculo. Nossa chave exclusiva para cada veículo, do tipo inteiro e auto incrementável. Temos Placa, Cor, Modelo como texto – outro tipo que podemos inserir ao invés de VARCHAR.

```sql
-- Criação da tabela Veiculo
CREATE TABLE VEICULO (
    IdVeiculo INTEGER PRIMARY KEY AUTOINCREMENT,
    Placa TEXT,
    Cor TEXT,
    Modelo TEXT,
    PessoaId INTEGER,
    FOREIGN KEY (PessoaId) REFERENCES Pessoa(IdPessoa)
);

```

**F.** Por último, criamos o atributo PessoaId como inteiro também. A diferença é que posteriormente definimos essa mesma PessoaId como chave estrangeira, como está escrito FOREIGN KEY.

Referenciando esta, utilizando a cláusula REFERENCES Pessoa (IdPessoa). Ou seja, vinculando com a tabela Pessoa, e exclusivamente a chave primária IdPessoa desta tabela.

**G.** Para entender melhor a mágica, sugiro inserir alguns registros, como no exemplo abaixo. Pode-se inserir vários registros em um só INSERT. Porém, lembre-se que precisamos sempre inserir qual ID vamos referenciar na tabela IdPessoa.

```sql
-- Inserção de 6 veículos vinculados aos 3 clientes existentes
INSERT INTO Veiculo (Placa, Cor, Modelo, PessoaId)
VALUES
    ('ABC123', 'Preto', 'Sedan', 1),  -- Pessoa 1
    ('XYZ456', 'Branco', 'SUV', 2),   -- Pessoa 2
    ('DEF789', 'Prata', 'Hatchback', 3),  -- Pessoa 3
    ('GHI123', 'Azul', 'Picape', 1),   -- Pessoa 1
    ('JKL456', 'Vermelho', 'Caminhonete', 2),  -- Pessoa 2
    ('MNO789', 'Verde', 'Compacto', 3);  -- Pessoa 3
```

**H.** Se você fizer um SELECT * FROM Veiculo, você terá o seguinte resultado:

SQLite SQLite

```
1 SELECT * FROM VEICULO
```

| IdVeiculo | Placa | Cor | Modelo | PessoaId |
|---|---|---|---|---|
| 1 | ABC123 | Preto | Sedan | 1 |
| 2 | XYZ456 | Branco | SUV | 2 |
| 3 | DEF789 | Prata | Hatchback | 3 |
| 4 | GHI123 | Azul | Picape | 1 |
| 5 | JKL456 | Vermelho | Caminhonete | 2 |
| 6 | MNO789 | Verde | Compacto | 3 |

**I.** Não confunda o IdVeiculo com a PessoaId. Faça um select para ver quantos carros o IdPessoa "1" tem:

```
1 SELECT * FROM VEICULO WHERE PessoaId = 1
```

| IdVeiculo | Placa | Cor | Modelo | PessoaId |
|---|---|---|---|---|
| 1 | ABC123 | Preto | Sedan | 1 |
| 4 | GHI123 | Azul | Picape | 1 |

**J.** A pessoa com PessoaId "1" tem dois veículos. Mas você sabe qual é essa pessoa? Então precisaremos utilizar uma cláusula JOIN, que faz com que se consiga juntar duas tabelas e utilizar a cláusula de filtro WHERE:

SQLite SQLite

```
1 SELECT Pessoa.Nome, Veiculo.Placa, Veiculo.Cor, Veiculo.Modelo
2 FROM Pessoa
3 JOIN Veiculo ON Pessoa.IdPessoa = Veiculo.PessoaId
4 WHERE Pessoa.IdPessoa = 1;
```

| Nome | Placa | Cor | Modelo |
|---|---|---|---|
| João | ABC123 | Preto | Sedan |
| João | GHI123 | Azul | Picape |

**K.** Entenda por partes o que foi feito:

- Pessoa.Nome seleciona o nome da pessoa.
- Veiculo.Placa, Veiculo.Cor e Veiculo.Modelo selecionam os atributos do veículo.
- JOIN Pessoa ON Veiculo.PessoaId = Pessoa.IdPessoa combina as tabelas “Pessoa” e “Veiculo” usando a chave estrangeira “PessoaId” na tabela “Veiculo” e a chave primária “IdPessoa” na tabela “Pessoa”.
- WHERE Pessoa.IdPessoa = 1 filtra os resultados apenas para a Pessoa com ID igual a 1 (que é a Pessoa 1).