

Banco de Dados com MySQL

Introdução ao Banco de Dados

Segundo Korth, um **banco de dados** “é uma coleção de dados inter-relacionados, representando informações sobre um domínio específico”, ou seja, sempre que for possível agrupar informações que se relacionam e tratam de um mesmo assunto, posso dizer que tenho um banco de dados.

Por exemplo as informações armazenadas de uma lista telefônica, um registros de usuários em um sistema de e-commerce, um sistema de controle de RH de uma empresa.

Já um sistema de gerenciamento de **banco de dados (SGBD)** é um software que possui recursos capazes de manipular as informações do banco de dados e interagir com o usuário. Exemplos de SGBDs são: [Oracle](#), [SQL Server](#), DB2, [PostgreSQL](#), [MySQL](#), o próprio Access ou Paradox, entre outros.

Por último, temos que entender que um sistema de banco de dados é como o conjunto de quatro componentes básicos: dados, hardware, software e usuários.

O objetivo de um **sistema de banco de dados** são o de isolar o usuário dos detalhes internos do banco de dados (promover a abstração de dados) e promover a independência dos dados em relação às aplicações, ou seja, tornar independente da aplicação, a estratégia de acesso e a forma de armazenamento.

Tipo de Dados

Bom agora que entendemos o conceito de banco de dados, também, precisamos entender que assim como tudo que manipula dados precisamos definir a “**Tipagem**” dos dados, vamos ver algumas:

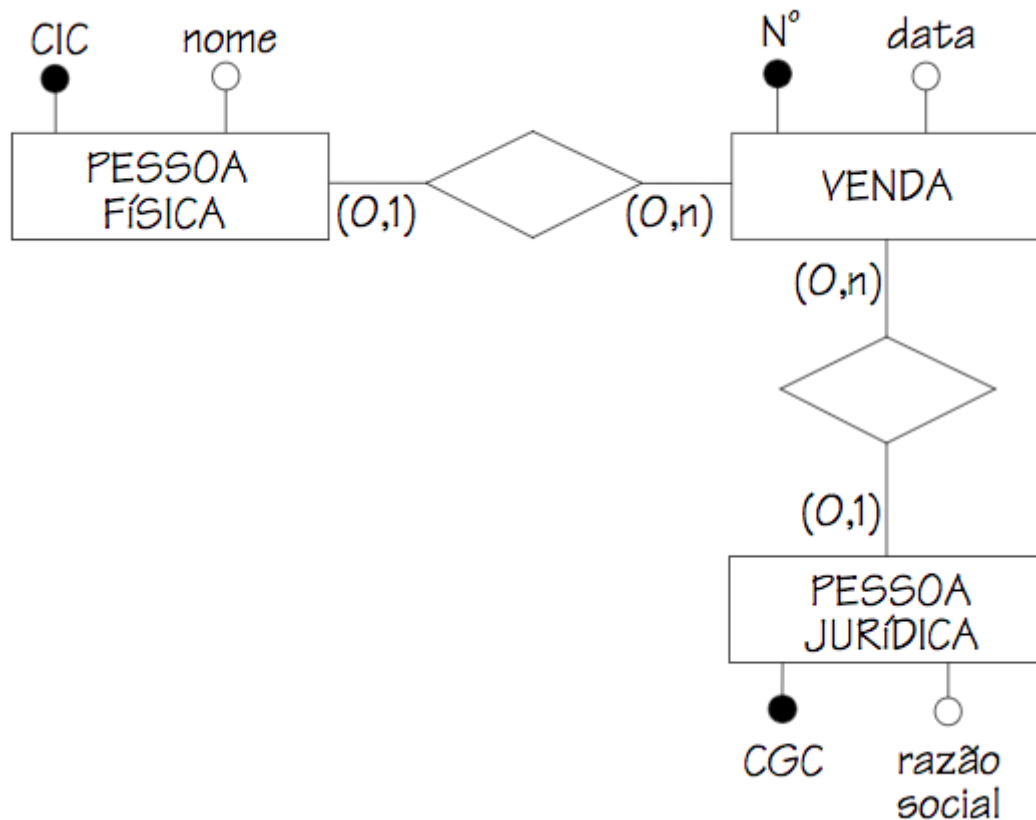
Dado	Tipo
Texto	VARCHAR(n)
Carácter	CHAR(n)

Dado	Tipo
Data	DATE
Data e Hora	DATETIME
Inteiro	INT
Decimal	DECIMAL
Ponto Flutuante	FLOAT
Ponto Flutuante mais casas decimais	DOUBLE
Numero decimal alterável padrão (10,0)	DECIMAL
Booleano (Verdadeiro ou Falso)	BOOLEAN

[Documentação tipo de dados W3Schools](#)

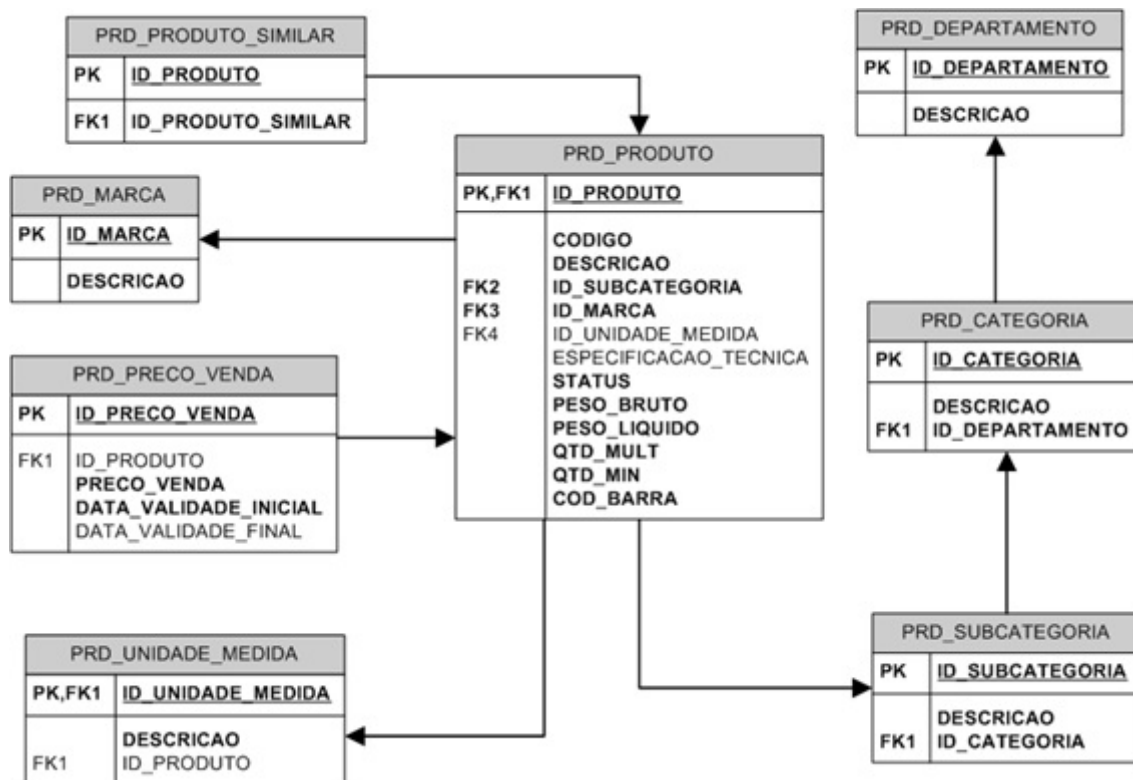
MER - Modelo Entidade Relacional

Assim como todos os projetos, devemos iniciar por um “esboço” do que de fato será nossa base de dados, quando desenhamos esta estrutura, a chamamos de **MER - Modelo Entidade Relacional** onde com apenas um papel e uma caneta podemos desenhar o modelo do nosso banco de dados expondo todos os relacionamentos entre as tabelas.



DER - Diagrama Entidade Relacional

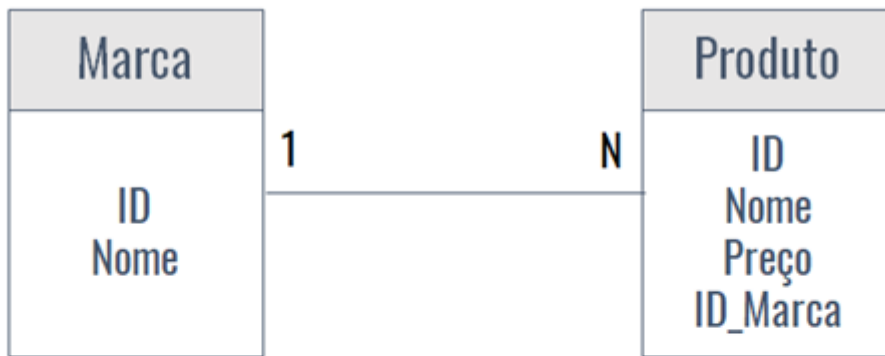
Assim como o **MER**, o **DER - Diagrama Entidade Relacional** também pode ser visto como um “esboço” de uma forma mais detalhada em comparação ao **MER** da nossa base d, porém ele é feito através de uma ferramenta específica, onde podemos compartilhar de uma forma segura com os demais membros de nossa equipe, e também extrair as o nosso **DER** em forma de query (o código sql usado para manipular o banco de dados).



Relacionamento Entre Tabelas

Uma vez que terminamos o nosso MER e o nosso DER, precisamos definir o relacionamento entre as tabelas, De acordo com a quantidade de objetos envolvidos em cada lado do relacionamento, podemos classifica-los de três formas:

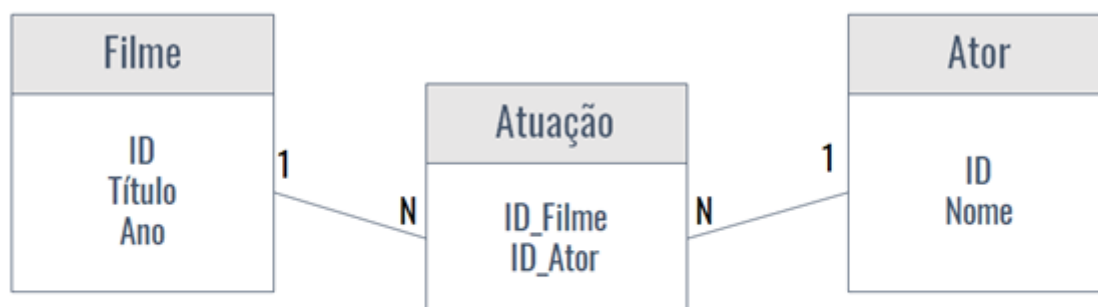
- **Relacionamento 1...1 (um para um):** cada uma das duas entidades envolvidas referenciam obrigatoriamente apenas uma unidade da outra. Por exemplo, em um banco de dados de currículos, cada usuário cadastrado pode possuir apenas um currículo na base, ao mesmo tempo em que cada currículo só pertence a um único usuário cadastrado.
- **Relacionamento 1...n (um para muitos):** uma das entidades envolvidas pode referenciar várias unidades da outra, porém, do outro lado cada uma das várias unidades referenciadas só pode estar ligada uma unidade da outra entidade. Por exemplo, em um sistema de plano de saúde, um usuário pode ter vários dependentes, mas cada dependente só pode estar ligado a um usuário principal. Note que temos apenas duas entidades envolvidas: usuário e dependente. O que muda é a quantidade de unidades/exemplares envolvidas de cada lado.



Uma marca possui vários produtos, mas um produto só pertence a uma marca.

- **Relacionamento n...n ou ... (muitos para muitos):** neste tipo de relacionamento cada entidade, de ambos os lados, podem referenciar múltiplas unidades da outra. Por exemplo, em um sistema de biblioteca, um título pode ser escrito por vários autores, ao mesmo tempo em que um autor pode escrever vários títulos. Assim, um objeto do tipo autor pode referenciar múltiplos objetos do tipo título, e vice versa.

Importante: Uma boa pratica é nomearmos os relacionamentos com verbos ou expressões que representa a forma como as entidades (tabelas) interagem, ou a ação que uma exerce sobre a outra, este nome pode mudar de acordo com com a direção que se lê o relacionamento, Ex uma sala de aula pode ter vários alunos, enquanto um aluno pode ter varias salas.



Um filme possui vários atores e um ator pode participar de vários filmes.

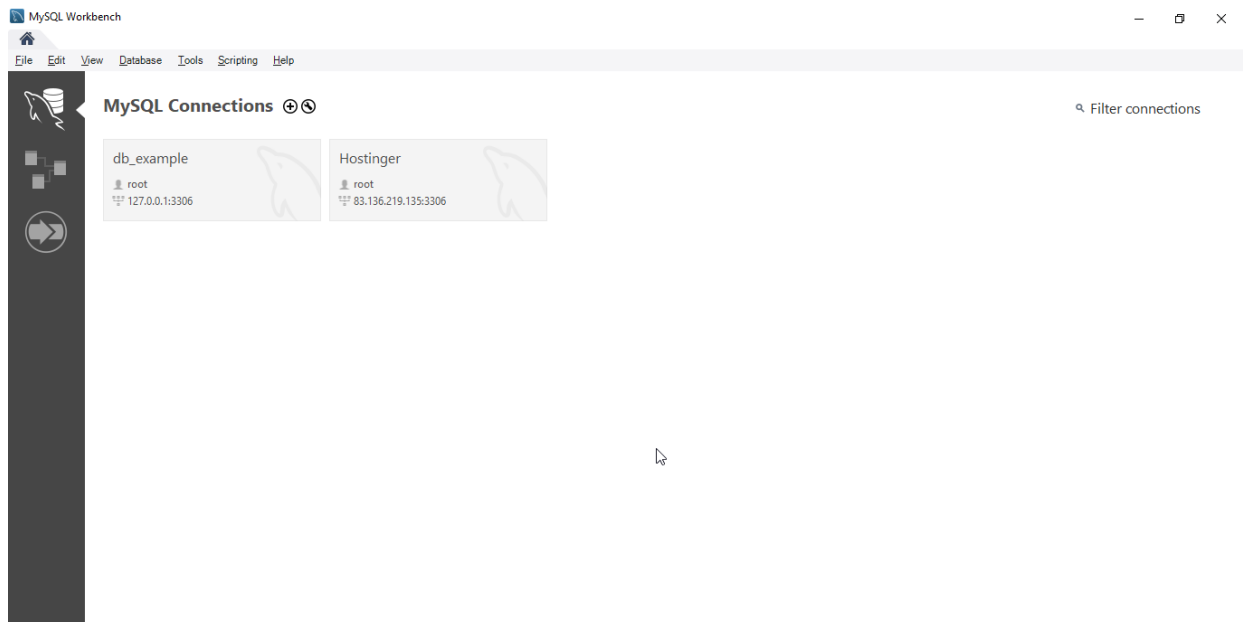
Bom agora que já definimos a estrutura do nosso banco de dados e o relacionamento entre as tabelas, Vamos por a mão na massa!

Criando nosso primeiro Banco de Dados

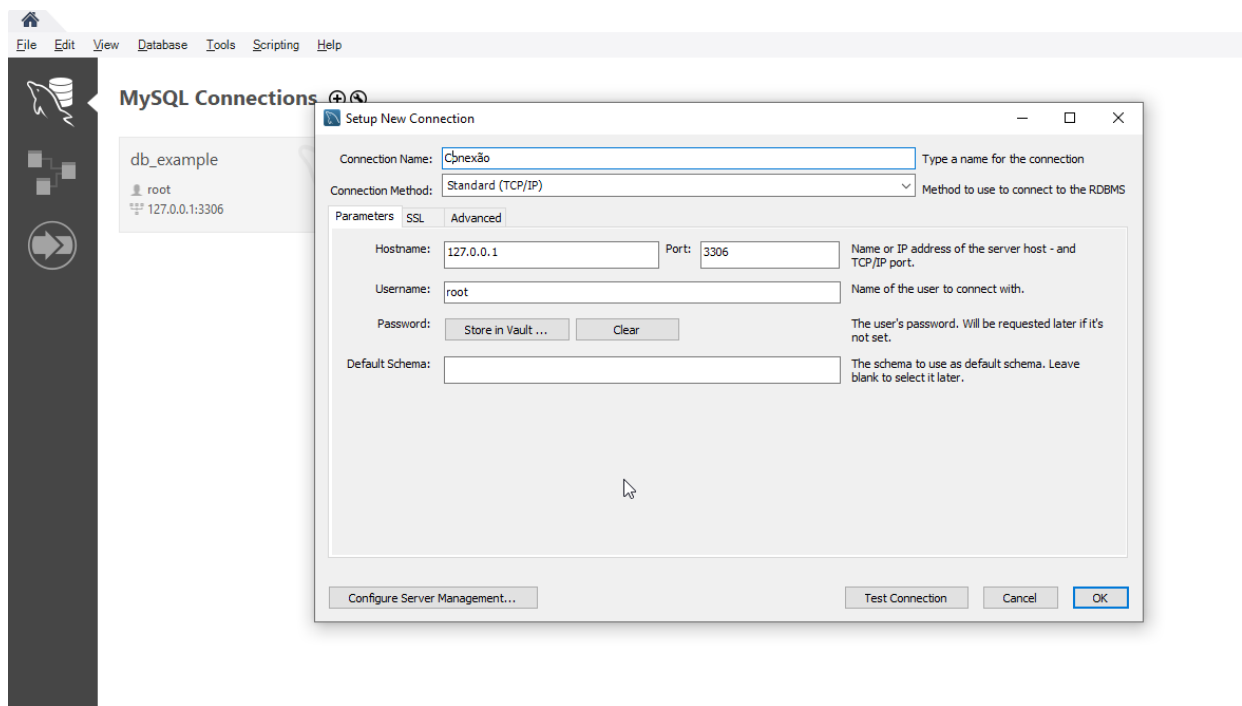
A IDE que utilizaremos para criar e manipular o nosso banco de dados será o MySQL Workbench



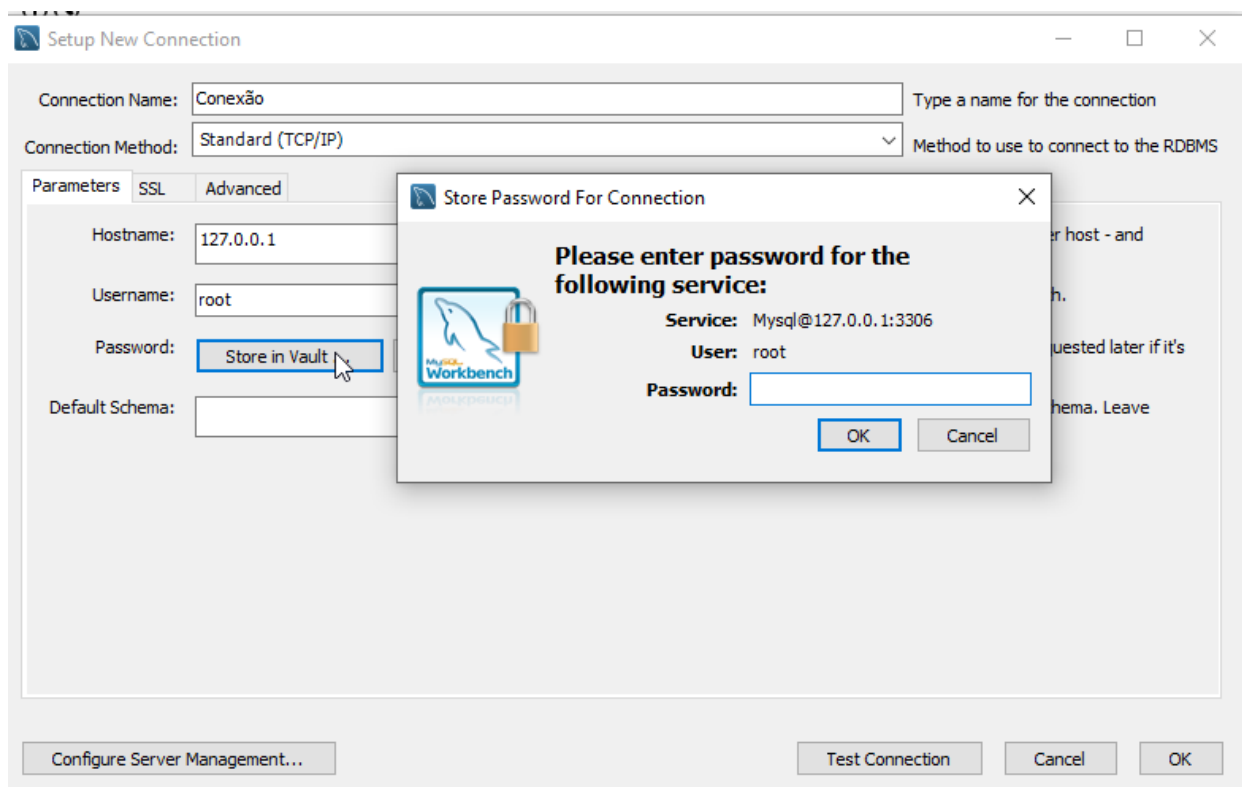
Uma vez que estamos o MySQL Workbench aberto, precisamos criar uma conexão local “+”.



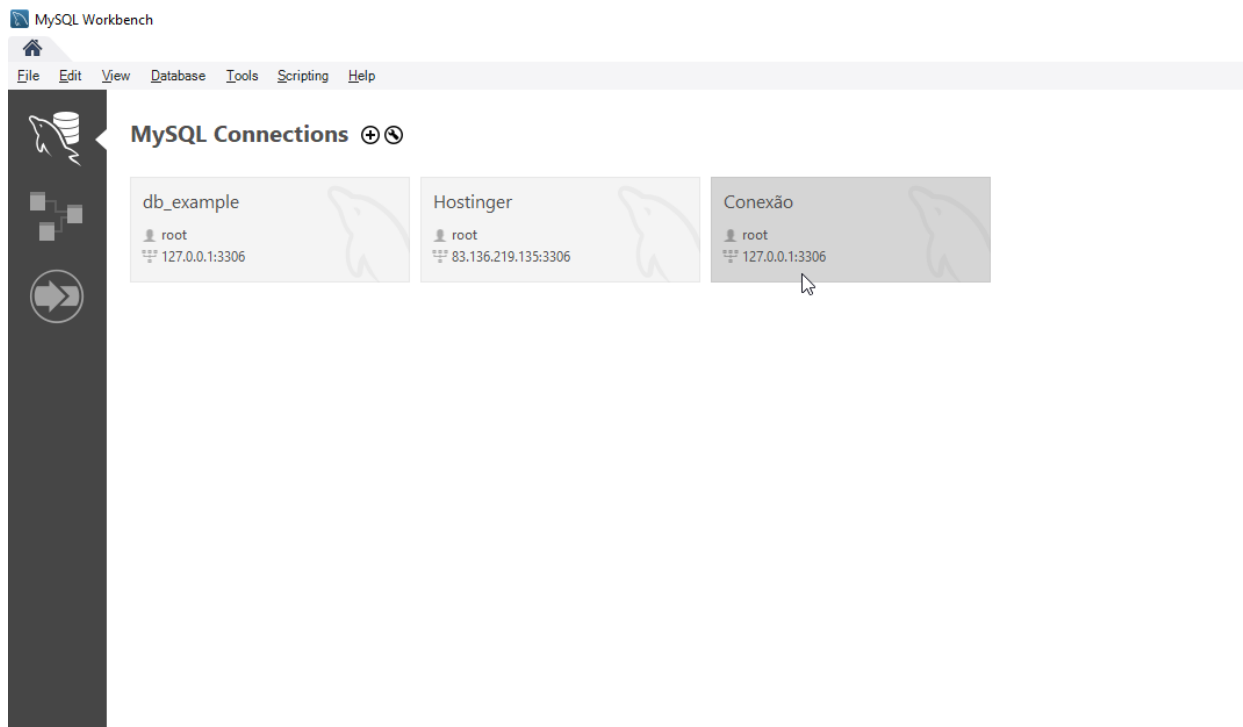
2. Defina um nome para a sua conexão.



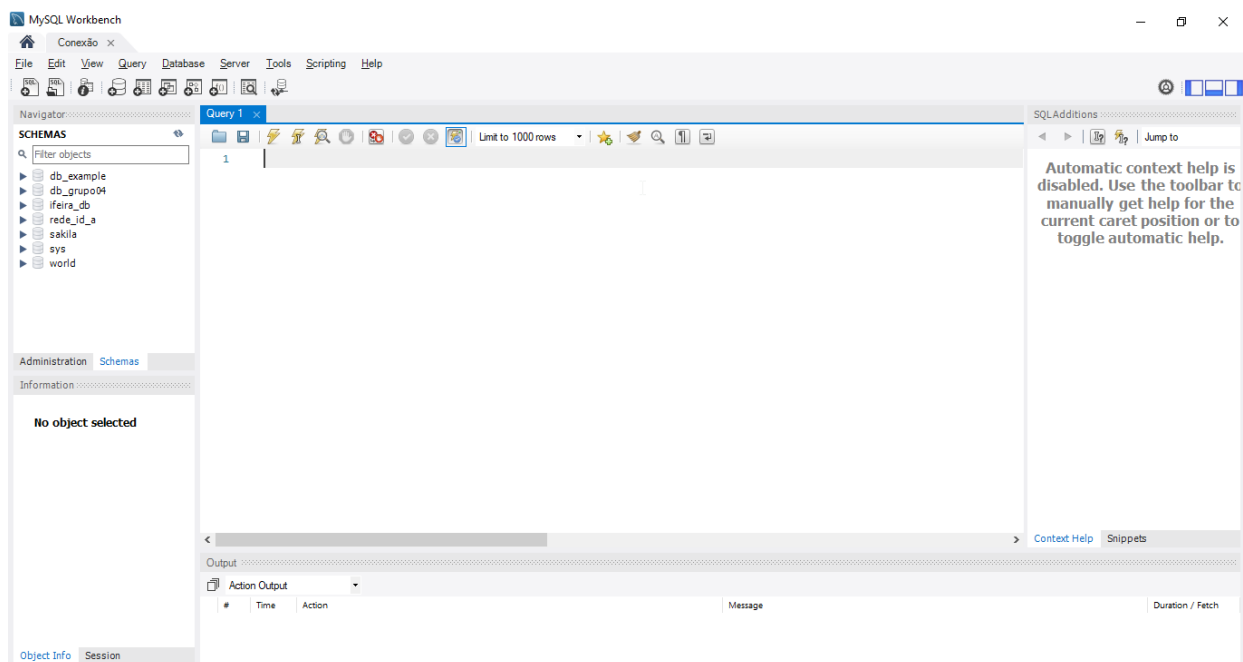
3. Defina uma senha para esta conexão clicando em Store in Vault..., e clique em OK.



4. Uma vez feito os passos acima, já temos nossa conexão criada.

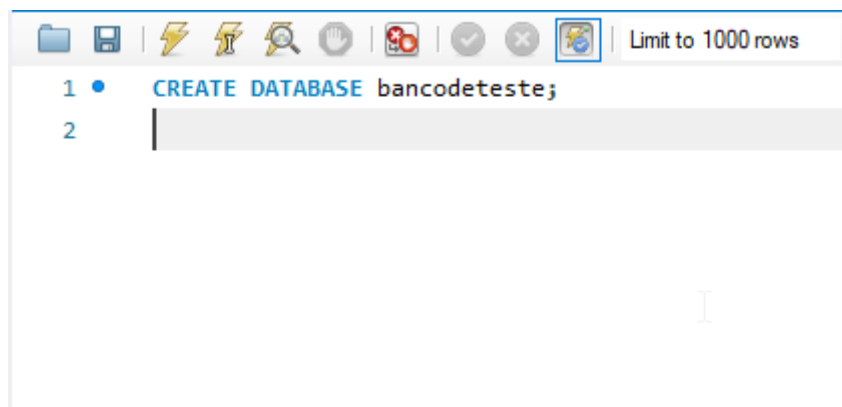


5. Após acessar a conexão criada teremos um ambiente onde vamos criar nossas tabelas e relaciona-las.

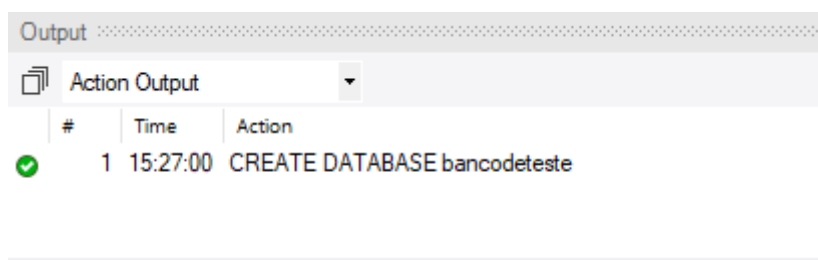


6. Agora Vamos criar o nosso primeiro banco de dados, para fazer isso basta ir na aba **Query** digitar e executar a query abaixo

```
CREATE DATABASE bancodeteste;
```

caso dê certo, no campo de Output aparecerá o log informando que a operação foi efetuada.

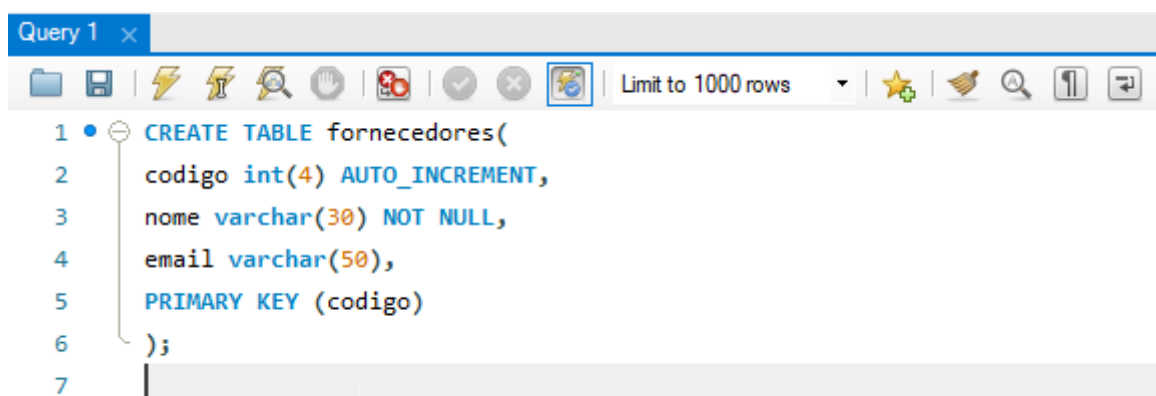


Criando nossa primeira tabela

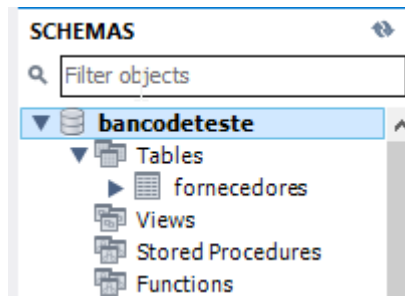
CREATE TABLE

1.O processo para criação de de uma tabela é bem simples, basta inserir as query, e em seguida clicar em executar conforme abaixo:.

```
`CREATE TABLE fornecedores(  
    codigo int(4) AUTO_INCREMENT,  
    nome varchar(30) NOT NULL,  
    email varchar(50),  
    PRIMARY KEY (codigo)  
);`
```



Caso a tudo ocorrer corretamente no canto esquerdo na aba das tabelas aparecerá a tabela criada.

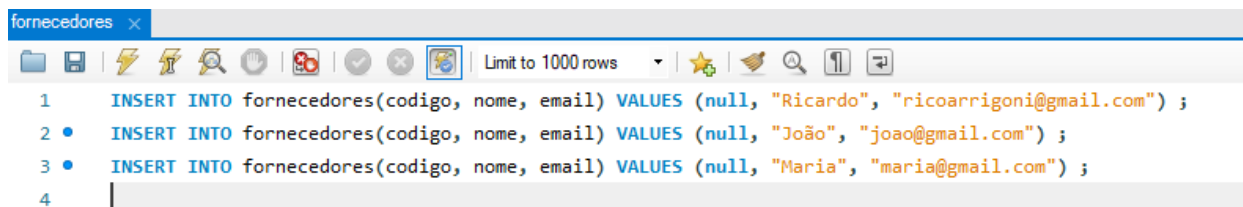


[Documentação create table W3Schools](#)

INSERT INTO

1. Uma vez que já temos a tabela criada, precisamos popula-la ou seja, inserir dados

```
INSERT INTO fornecedores(codigo, nome, email) VALUES (null, "Ricardo",  
"ricoarrigoni@gmail.com") ;  
INSERT INTO fornecedores(codigo, nome, email) VALUES (null, "João",  
"joao@gmail.com") ;  
INSERT INTO fornecedores(codigo, nome, email) VALUES (null, "Maria",  
"maria@gmail.com") ;
```



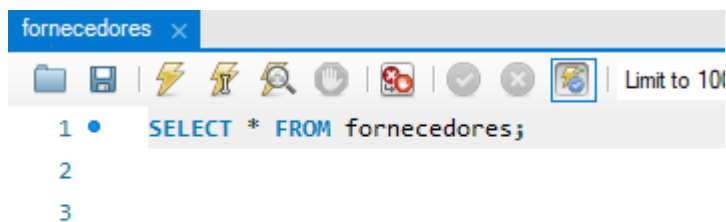
feito isto os dados do Ricardo, João e Maria serão inseridos na tabela, o primeiro valor antes do nome, será inserido como null porque o campo codigo esta definido com chave primária (PRIMARY KEY, AUTO_INCREMENT).

[Documentação Insert Into W3Schools](#)

SELECT

1. Agora vamos fazer a nossa primeira consulta no bando para ver se os dados foram devidamente inseridos na tabela. Para isto basta fazer um Select na tabela.

```
SELECT * FROM fornecedores;
```

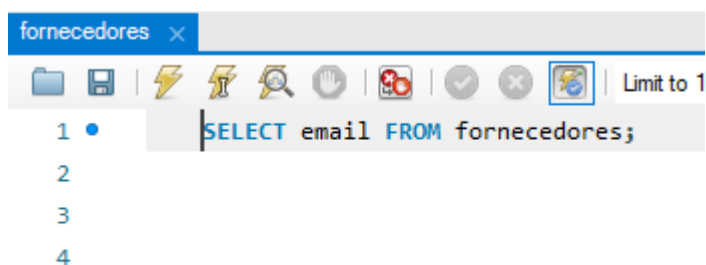


Feito isto aparecerá no log de Output a seguinte tabela

	codigo	nome	email
1		Ricardo	ricoarrigoni@gmail.com
2		João	joao@gmail.com
3		Maria	maria@gmail.com
*	NULL	NULL	NULL

2. Caso seja necessário seleccionar apenas um dado como por exemplo o email, basta substituir o (*) asterisco pelo nome do cmpo desejado, caso quisermos mostrar mais campo, debes separa-los por virgula ex (e mail, nome).

SELECT email FROM fornecedores;

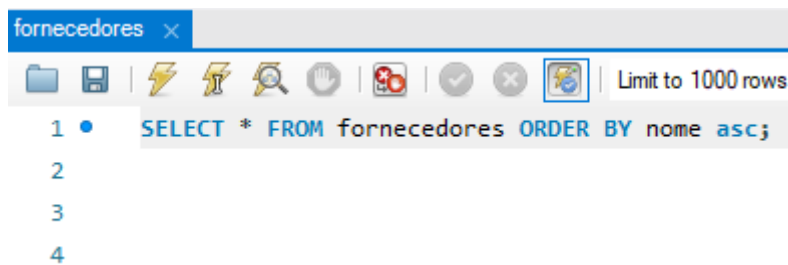


`com o código acima, teremos apenas as informações referente a os e-mails que contém na tabela de fornecedores.

	email
1	ricoarrigoni@gmail.com
2	joao@gmail.com
3	maria@gmail.com

3. Também podemos ordenar de acordo com o campo desejado, exemplo se eu quiser ordenar por nome, basta adicionar o comando `ORDER BY` e o campo que eu queira ordenar e em seguida por `ASC` para crescente ou `DESC` para decrescente.

```
SELECT * FROM fornecedores ORDER BY nome ASC;
```



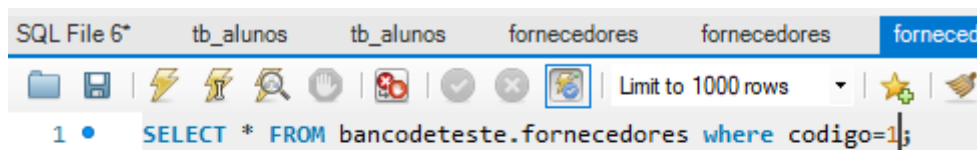
e retornaremos a tabela completa, porem ordenada por nome de forma crescente.

The screenshot shows a 'Result Grid' window. It contains a table with the following data:

	codigo	nome	email
▶	2	João	joao@gmail.com
	3	Maria	maria@gmail.com
	1	Ricardo	ricoarrigoni@gmail.com
*	NULL	NULL	NULL

4. Caso haja a necessidade de retornar apenas uma linha especifica, exemplo apenas as informações pertinentes ao Ricardo, basta utilizarmos o `WHERE` + o campo que queremos filtrar (`codigo`) + o sinal de igual e o numero refente ao código do Ricardo, neste caso será o numero 1. Ficará assim:

```
SELECT * FROM fornecedores WHERE codigo = 1;
```



aparecerá na tabela abaixo, apenas as informações pertinente a codigo 1:

Result Grid	Filter Rows:	Edit
codigo	nome	email
1	Ricardo Arrigoni	ricoarrigoni@gmail.com
NULL	NULL	NULL

[Documentação Select W3Schools](#)

UPDATE

1. Para atualizar os dados devemos dar um Update no da dado que queremos atualizar.

Importante devemos utilizar o *Where* para indicarmos ao Workbanche exatamente o dado/linha que queremos atualizar/alterar

```
UPDATE fornecedores SET nome = "Ricardo Arrigoni" WHERE codigo = 1;
```



na tela de Output veremos o log indicando que houve sucesso ao executar a query

Output		
Action Output		
#	Time	Action
11	15:51:28	SELECT * FROM fornecedores ORDER BY nome asc LIMIT 0, 1000
12	15:55:50	UPDATE fornecedores SET nome="Ricardo Arrigoni" WHERE codigo=1

E ao consultar com o `SELECT` poderemos verificar as mudanças feita no fornecedor Ricardo.

Result Grid			
	codigo	nome	email
▶	1	Ricardo Arrigoni	ricoarrigoni@gmail.com
	2	João	joao@gmail.com
	3	Maria	maria@gmail.com
*	NULL	NULL	NULL

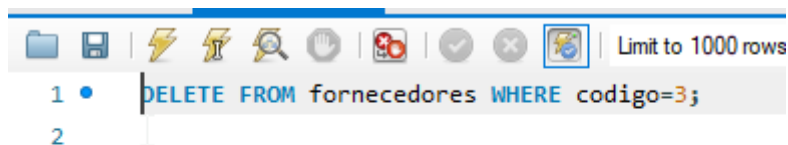
[Documentação Update W3Schools](#)

DELETE

1. Para deletar um fornecedor da nossa tabela, basta utilizar o comando `DELETE`

Importante, assim como o `UPDATE` no `DELETE` também devemos mostrar para o Workbanche qual dado ele irá deletar, para isto devemos utilizar o comando `WHERE`, caso isto não seja feito, todos os dados da tabela serão alterados ou removidos.

```
DELETE FROM fornecedores WHERE codigo = 3;
```



feito isto, o Output apresentará o log informando que o dado foi deletado.

Output		
Action Output		
#	Time	Action
✓ 13	15:57:17	SELECT * FROM bancodeteste.fornecedores LIMIT 0, 1000
✓ 14	16:04:15	DELETE FROM fornecedores WHERE codigo=3

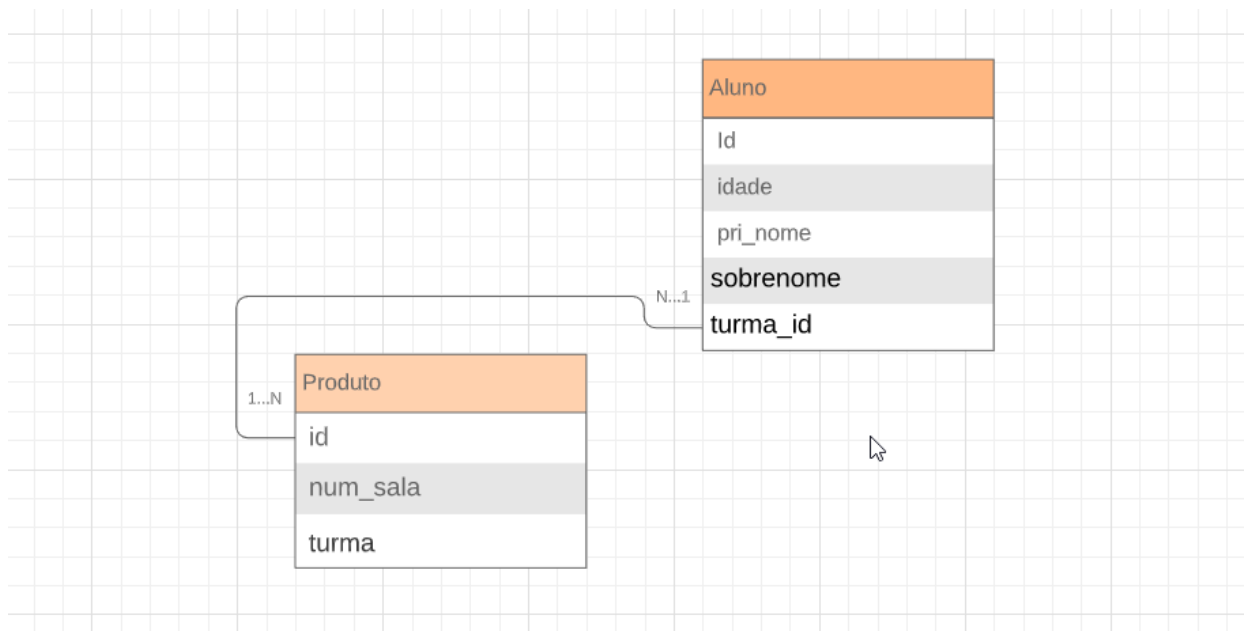
a o fazer o a consulta através de um `SELECT` podemos verificar o as informações pertinente a Maria não constarão mais na tabela.

	codigo	nome	email
1		Ricardo Arrigoni	ricoarrigoni@gmail.com
2		João	joao@gmail.com
*	NULL	NULL	NULL

[Documentação Delete W3Schools](#)

Associações entre tabelas

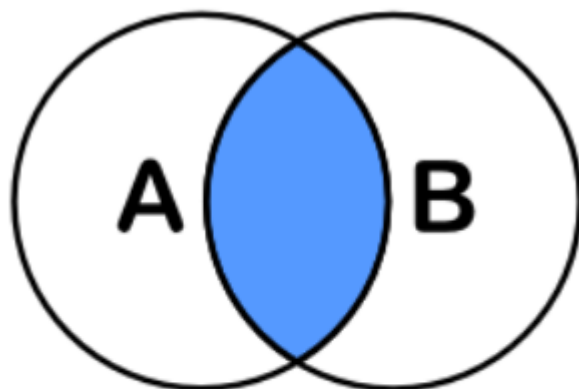
Como vimos no capítulo de Relacionamento Entre Tabelas, podemos definir relacionamentos entre tabelas de N:1 e N:N, mas podemos precisar fazer buscas relacionadas entre estas tabelas, para isso precisamos de um mecanismo de busca de Associações entre tabelas.



Associações de tabelas ou busca por **JOIN** podem ser utilizadas para diversas finalidades, como converter em informação os dados encontrados em duas ou mais tabelas.

A cláusula **JOIN** é usada para combinar dados provenientes de duas ou mais tabelas do banco de dados, baseado em um relacionamento entre colunas destas tabelas. há três categorias principais de joins:

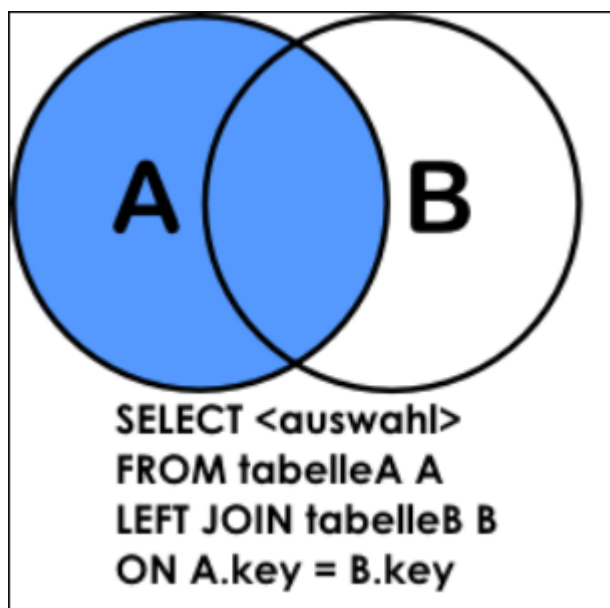
INNER JOIN



```
SELECT <auswahl>  
FROM tabelleA A  
INNER JOIN tabelleB B  
ON A.key = B.key
```

Nas pesquisas com **INNER JOIN** resultado trará somente as linhas que sejam comum nas 2 tabelas, ligadas pelos campos das tabelas em questão na pesquisa.

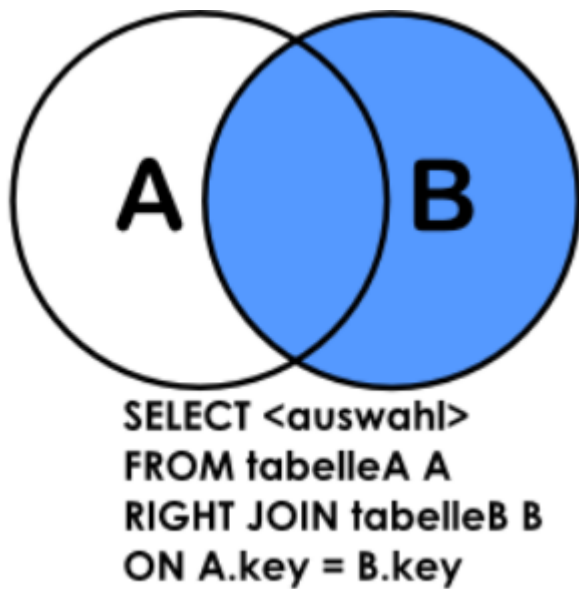
LEFT JOIN



```
SELECT <auswahl>  
FROM tabelleA A  
LEFT JOIN tabelleB B  
ON A.key = B.key
```

A cláusula **LEFT JOIN** permite obter não apenas os dados relacionados de duas tabelas**, mas também os dados não relacionados encontrados na tabela à esquerda da cláusula JOIN. Caso não existam dados relacionados entre as tabelas à esquerda e a direita do JOIN, os valores resultantes de todas as colunas da lista de seleção da tabela à direita serão nulos.

RIGHT JOIN



Ao contrário do **LEFT JOIN**, a cláusula **RIGHT JOIN** retorna todos os dados encontrados na tabela à direita de JOIN. Caso não existam dados associados entre as tabelas à esquerda e à direita de JOIN, serão retornados valores nulos.

Exemplos

Veremos o exemplo abaixo.

Abra o arquivo de query e execute no MySQL Workbench.

1. Crie uma bancotabela com o nome de turma

```
1 • Use baseteste;
2
3 • CREATE TABLE `turma` (
4     `id` bigint NOT NULL AUTO_INCREMENT,
5     `num_sala` int NOT NULL,
6     `turma` varchar(255) DEFAULT NULL,
7     PRIMARY KEY (`id`)
8 )
```

#	Time	Action	Message
1	12:58:28	Create database baseteste	1 row(s) affected
2	13:01:56	Use baseteste	0 row(s) affected
3	13:01:57	CREATE TABLE `turma` (`id` bigint NOT NULL AUTO_INCREMENT, `num_sala` int NOT NULL, `turma` varc...	0 row(s) affected

```
CREATE TABLE `turma` (  
    `id` bigint NOT NULL AUTO_INCREMENT,  
    `num_sala` int NOT NULL,
```

```

    `turma` varchar(255) DEFAULT NULL,
    PRIMARY KEY (`id`)
);

```

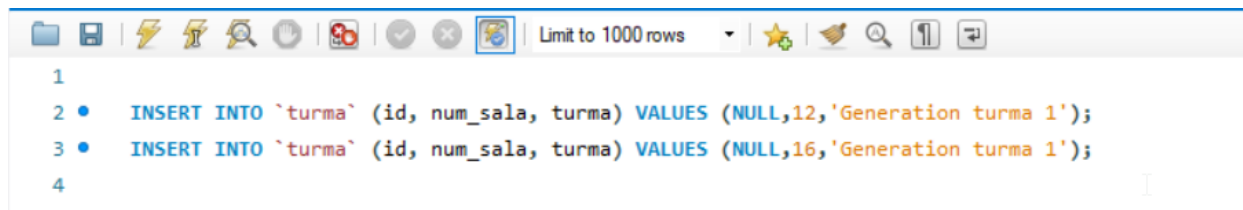
2. crie uma com o nome de aluno

3. Insira os dados nesta tabela atraves do **INSERT**

```

INSERT INTO `turma` (id, num_sala, turma) VALUES (NULL,12,'Generat:
INSERT INTO `turma` (id, num_sala, turma) VALUES (NULL,16,'Generat:

```



```

1
2 • INSERT INTO `turma` (id, num_sala, turma) VALUES (NULL,12,'Generation turma 1');
3 • INSERT INTO `turma` (id, num_sala, turma) VALUES (NULL,16,'Generation turma 1');
4

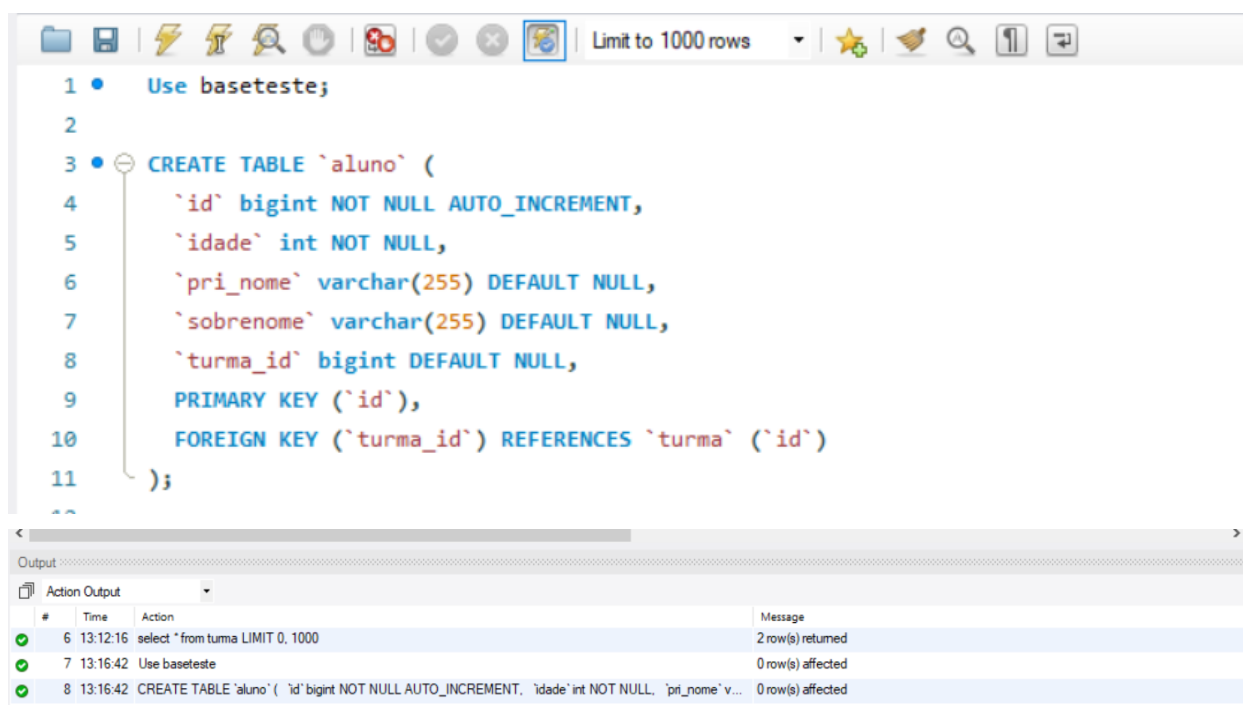
```

3. crie uma tabela com o nome de aluno

```

CREATE TABLE `aluno` (
    `id` bigint NOT NULL AUTO_INCREMENT,
    `idade` int NOT NULL,
    `pri_nome` varchar(255) DEFAULT NULL,
    `sobrenome` varchar(255) DEFAULT NULL,
    `turma_id` bigint DEFAULT NULL,
    PRIMARY KEY (`id`),
    FOREIGN KEY (`turma_id`) REFERENCES `turma` (`id`)
);

```



```

1 • Use baseteste;
2
3 • CREATE TABLE `aluno` (
4     `id` bigint NOT NULL AUTO_INCREMENT,
5     `idade` int NOT NULL,
6     `pri_nome` varchar(255) DEFAULT NULL,
7     `sobrenome` varchar(255) DEFAULT NULL,
8     `turma_id` bigint DEFAULT NULL,
9     PRIMARY KEY (`id`),
10    FOREIGN KEY (`turma_id`) REFERENCES `turma` (`id`)
11 );

```

#	Time	Action	Message
6	13:12:16	select * from turma LIMIT 0, 1000	2 row(s) returned
7	13:16:42	Use baseteste	0 row(s) affected
8	13:16:42	CREATE TABLE `aluno` (`id` bigint NOT NULL AUTO_INCREMENT, `idade` int NOT NULL, `pri_nome` v...	0 row(s) affected

4. Insira dados a ela.

```

INSERT INTO `aluno` (id, idade, pri_nome, sobrenome, turma_id ) VALUES
INSERT INTO `aluno` (id, idade, pri_nome, sobrenome, turma_id ) VALUES
INSERT INTO `aluno` (id, idade, pri_nome, sobrenome, turma_id ) VALUES
INSERT INTO `aluno` (id, idade, pri_nome, sobrenome, turma_id ) VALUES
INSERT INTO `aluno` (id, idade, pri_nome, sobrenome, turma_id ) VALUES
INSERT INTO `aluno` (id, idade, pri_nome, sobrenome, turma_id ) VALUES

```

```

1 • INSERT INTO `aluno` (id, idade, pri_nome, sobrenome, turma_id ) VALUES (Null,29,'Marcelo','Barboza',2);
2 • INSERT INTO `aluno` (id, idade, pri_nome, sobrenome, turma_id ) VALUES (Null,26,'Lucas','Capelloto',1);
3 • INSERT INTO `aluno` (id, idade, pri_nome, sobrenome, turma_id ) VALUES (Null,19,'Maria','Silva',1);
4 • INSERT INTO `aluno` (id, idade, pri_nome, sobrenome, turma_id ) VALUES (Null,32,'Madona','Barretos',2);
5 • INSERT INTO `aluno` (id, idade, pri_nome, sobrenome, turma_id ) VALUES (Null,34,'Creonilta','Santos',1);
6 • INSERT INTO `aluno` (id, idade, pri_nome, sobrenome, turma_id ) VALUES (Null,34,'João','Santos',2);
7
8

```

5. Digite a seguinte query para para **INNER JOIN**

```

Select idade, pri_nome, turma.turma from aluno
inner join turma on turma.id = aluno.turma_id

```

	idade	pri_nome	turma
▶	26	Lucas	Generation turma 1
	19	Maria	Generation turma 1
	34	Creonilta	Generation turma 1
	29	Marcelo	Generation turma 2
	32	Madona	Generation turma 2
	34	João	Generation turma 2

[Documentação Inner Join W3Schools](#)

6. Digite a seguinte query para **LEFT JOIN**

```

Select turma.turma, aluno.idade, aluno.pri_nome from turma
Left join aluno on turma.id = aluno.turma_id

```

