

MYSQL





01

MySQL

02

Instalação

03

Exemplo

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 - Sistema Gerenciador de Banco de Dados) 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.

Introdução ao Banco de Dados

O objetivo de um sistema de banco de dados é 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(100)
carácter	CHAR
data	DATE
data e hora	DATETIME
inteiro	INT
decimal	DECIMAL

DADO	TIPO
ponto flutuante	FLOAT, DOUBLE
booleano	BOOLEAN

MER - Modelo Entidade Relacional

Assim como todo os projetos, devemos iniciar por um “esboço” do que de fato será nossa base de dados, quando desenhamos esta estrutura, á 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 relacionamento entre as tabelas.

Características do Modelo Entidade Relacionamento (MER):

- Foi desenvolvido para facilitar o projeto lógico do BD;
- Permite a representação da estrutura lógica global do BD;
- É um dos modelos de dados com maior capacidade semântica;
- Representa um problema como um conjunto de entidades e relacionamentos entre estas entidades;

MER - Modelo Entidade Relacional

Entidade

linha ou tupla



marca	modelo	ano	cor	...
dado01	dado02	dado03	dado04	dado05



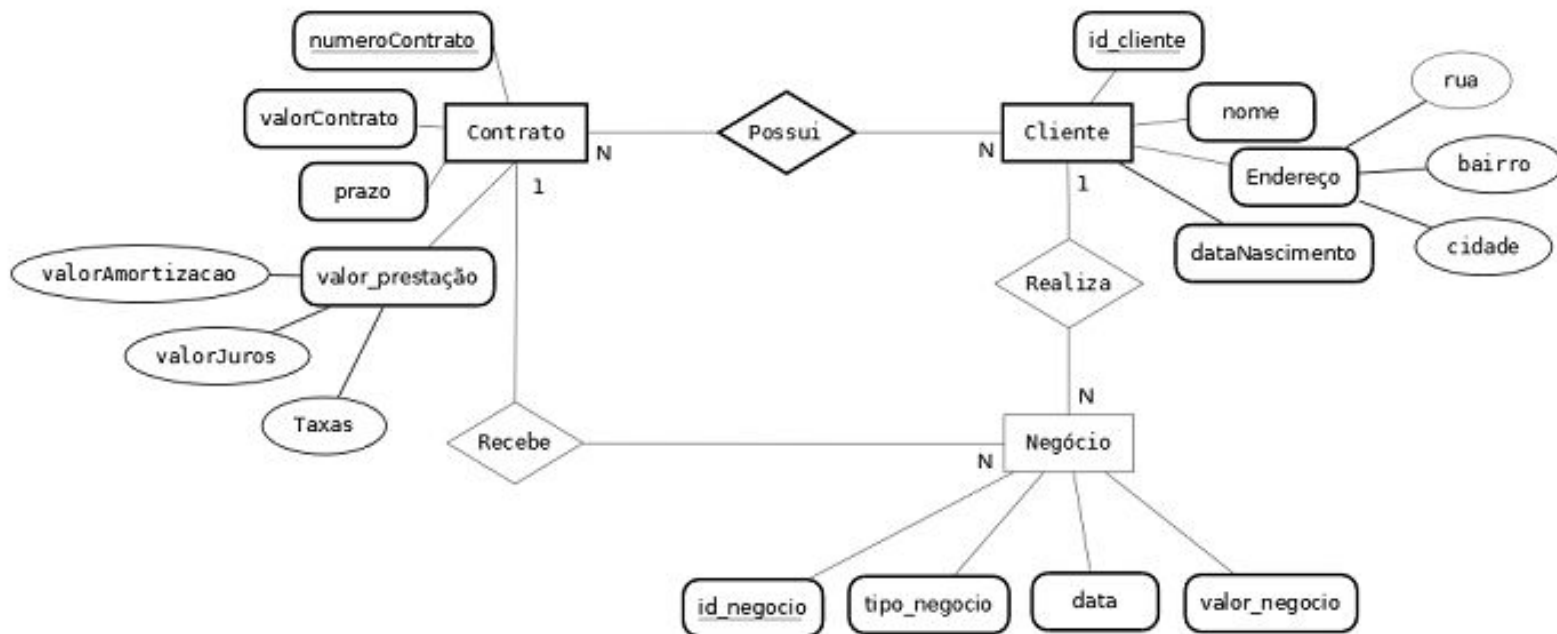
coluna

MER - Modelo Entidade Relacional

Schemas

Os Schemas são uma coleção de objetos dentro de um determinado database (banco de dados), servem para agrupar objetos no nível de aplicação como também para simplesmente fazer divisões departamentais. Schemas são bastante utilizados em padrões de sistema de banco de dados. São muito importantes para a performance e segurança.

MER - Modelo Entidade Relacional



RETÂNGULO: Entidade;

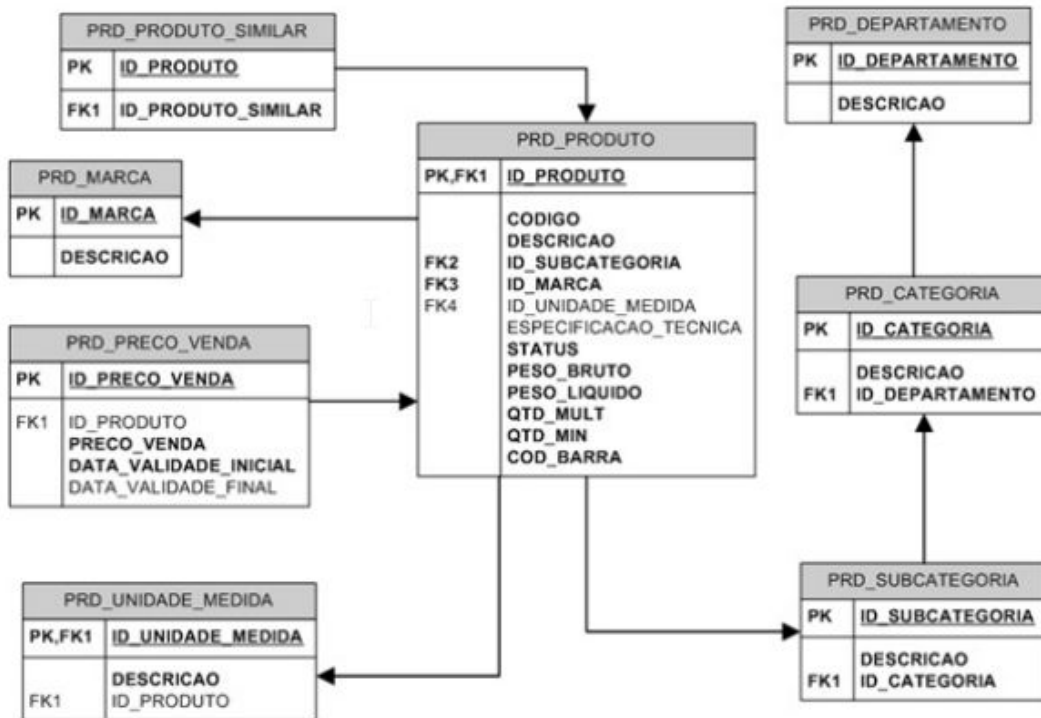
ELIPSE: Atributo;

LOSANGO: Relacionamento.

DER - Diagrama Entidade Relacionamento

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, 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).

DER - Diagrama Entidade Relacional

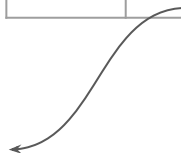
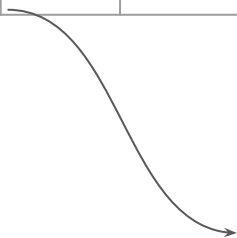


Relacionamento Entre Tabelas

CLIENTE

PRODUTO

PEDIDO



Relacionamento Entre Tabelas

Chave primária / primary key

Coluna que identifica unicamente um único registro/tupla dentro de uma tabela

- Única dentro de uma tabela
- Não vazia
- Não muda, é muito trabalhoso
- Pode ser uma ou mais colunas

COD	NOME	EMAIL	CPF
1	MARIA		
2	JOAO		
3	JOSE		

SURROGATE KEY

NATURAL KEY

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 classificá-los de três formas:

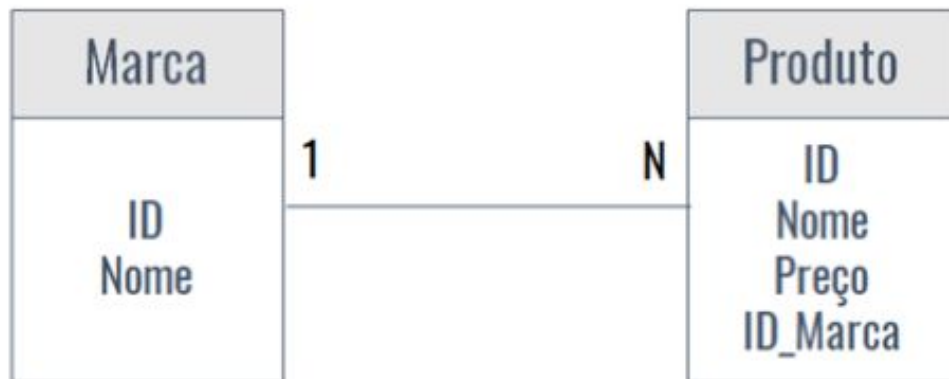
Relacionamento 1...1 (um para um): cada uma das duas entidades envolvidas referenciam obrigatoriamente apenas uma unidade da outra. Por exemplo, em um banco de dados de currículos, cada usuário cadastrado pode possuir apenas um currículo na base, ao mesmo tempo em que cada currículo só pertence a um único usuário cadastrado.

Relacionamento Entre Tabelas

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.

Relacionamento Entre Tabelas



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

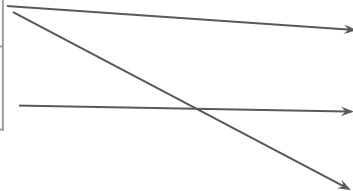
Relacionamento Entre Tabelas

MÃE

cod	nome
1	Maria
2	Helena

FILHO

cod	nome
1	João
2	José
3	Francisco



Relacionamento Entre Tabelas

MÃE

cod	nome
1	Maria
2	Helena

FILHO **FK**



cod	nome	mae_cod
1	João	1
2	José	2
3	Francisco	1

Chave estrangeira (Foreign key - FK)

No contexto dos banco de dados, o conceito de chave estrangeira ou chave externa se refere ao tipo de relacionamento entre distintas tabelas de dados do banco de dados. Uma chave estrangeira é chamada quando há o relacionamento entre duas tabelas.

Relacionamento Entre Tabelas

MÃE

cod	nome
1	Maria
2	Helena
3	Andreia

FILHO **FK**

cod	nome	mae_cod
1	João	3
2	José	2
3	Francisco	1

A restrição UNIQUE garante que todos os valores em uma coluna sejam diferentes.

UNIQUE

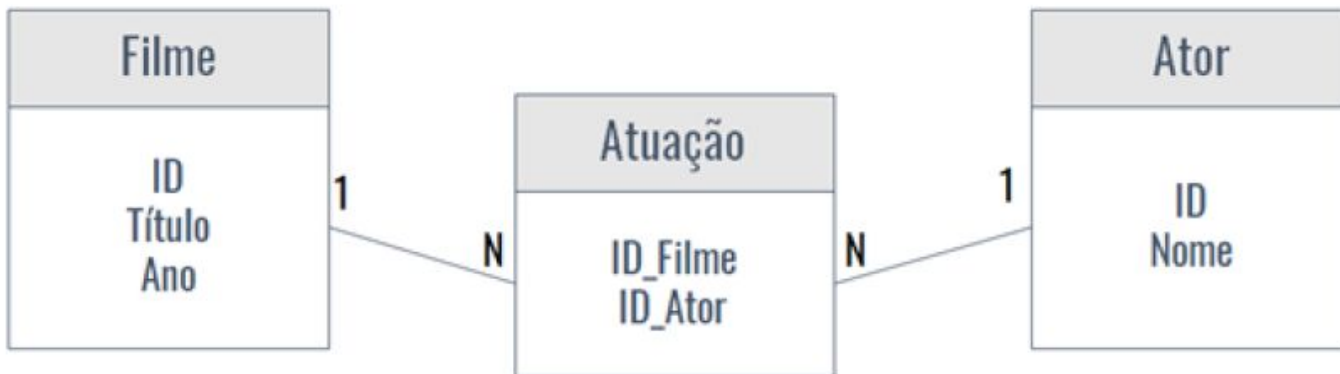
Relacionamento Entre Tabelas

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.

Relacionamento Entre Tabelas

Importante: Uma boa prática 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 a direção que se lê o relacionamento, Ex uma sala de aula pode ter vários alunos, enquanto um aluno pode ter várias salas.



Relacionamento Entre Tabelas

TIA

id	nome
1	
2	

FK

TIA_SOB

FK

tia_id	sobrinha_id
1	1
1	2
2	1
2	1

SOBRINHA

id	nome
1	
2	

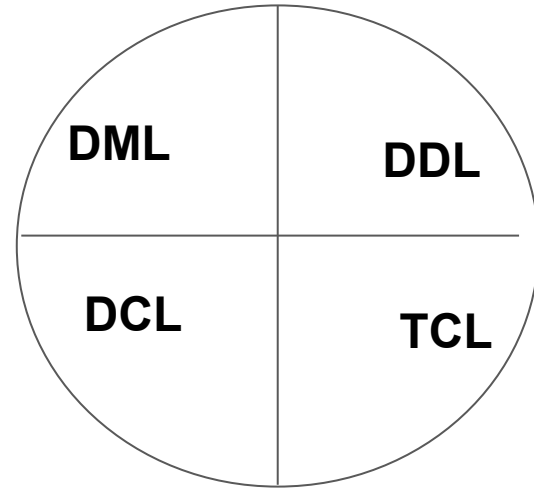
Diferentes tipos: Categorias de instruções (Transact-SQL)

Data Manipulation Language (DML)

Data Definition Language (DDL)

Data Control Language (DCL)

Transactional Control Language (TCL)



Diferentes tipos: Categorias de instruções (Transact-SQL)

DML (Linguagem de Manipulação de Dados) – É um conjunto de instruções usada nas consultas e modificações dos dados armazenados nas tabelas do banco de dados. Alguns Exemplos:

SELECT -> Recupera linhas do banco de dados e permite a seleção de uma ou várias linhas ou colunas de uma ou várias tabelas

INSERT -> Instrução utilizada para inserir dados a uma ou mais tabelas no banco de dados

UPDATE -> Instrução utilizada para atualizar dados de uma ou mais tabelas no banco de dados

DELETE -> Instrução utilizada para excluir dados de uma ou mais tabelas no banco de dados

MERGE -> Realiza operações de inserção, atualização ou exclusão em uma tabela de destino com base nos resultados da junção com a tabela de origem

Diferentes tipos: Categorias de instruções (Transact-SQL)

DDL (Linguagem de Definição de Dados) – É um conjunto de instruções usado para criar e modificar as estruturas dos objetos armazenados no banco de dados.

Alguns Exemplos:

ALTER -> Use as instruções ALTER para modificar a definição de entidades existentes. Use ALTER TABLE para adicionar uma nova coluna a uma tabela ou use ALTER DATABASE para definir opções do banco de dados.

CREATE -> Use instruções CREATE para definir novas entidades. Use CREATE TABLE para adicionar uma nova tabela em um banco de dados.

DROP -> Use instruções DROP para remover entidades existentes. Use DROP TABLE para remover uma tabela de um banco de dados.

Diferentes tipos: Categorias de instruções (Transact-SQL)

DCL (Linguagem de Controle de Dados) – São usados para controle de acesso e gerenciamento de permissões para usuários em no banco de dados. Com eles, pode facilmente permitir ou negar algumas ações para usuários nas tabelas ou registros (segurança de nível de linha). Alguns Exemplos:

GRANT -> Atribui privilégios de acesso do usuário a objetos do banco de dados.

REVOKE -> Remove os privilégios de acesso aos objetos obtidos com o comando **GRANT**.

DENY -> O comando é usado para impedir explicitamente que um usuário receba uma permissão específica.

Diferentes tipos: Categorias de instruções (Transact-SQL)

TCL (Linguagem de Controle de Transações) – São usados para gerenciar as mudanças feitas por instruções DML . Ele permite que as declarações a serem agrupadas em transações lógicas. Alguns Exemplos:

COMMIT -> É usado para salvar permanentemente qualquer transação no banco de dados.

ROLLBACK -> Este comando restaura o banco de dados para o último estado committed.

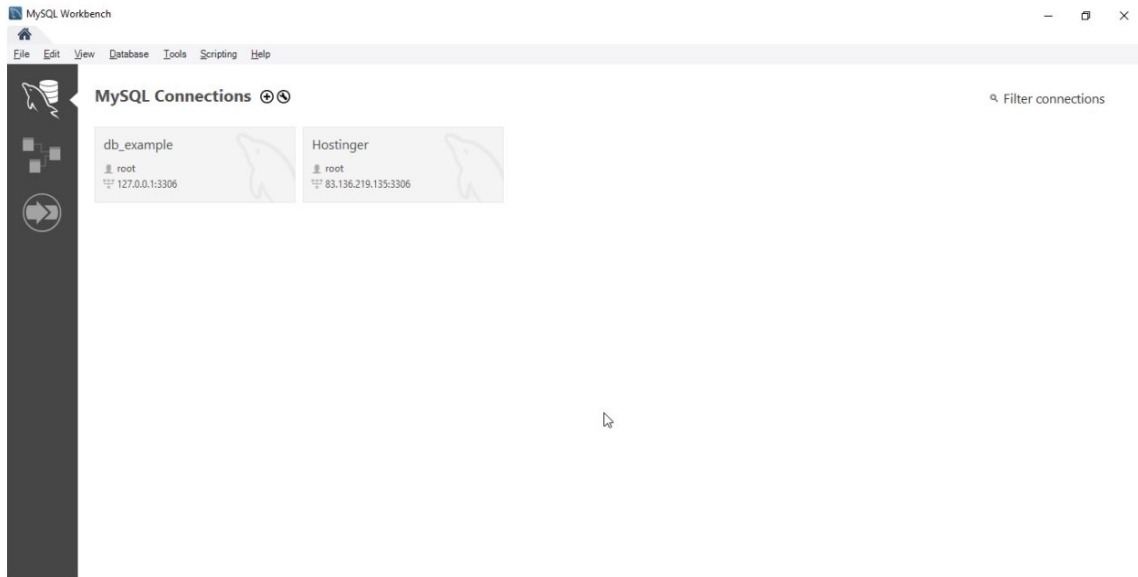
BEGIN TRANSACTION -> Marca o ponto inicial de uma transação local explícita. As transações explícitas começam com a instrução **BEGIN TRANSACTION** e terminam com a instrução **COMMIT** ou **ROLLBACK**.

VAMOS PRATICAR?

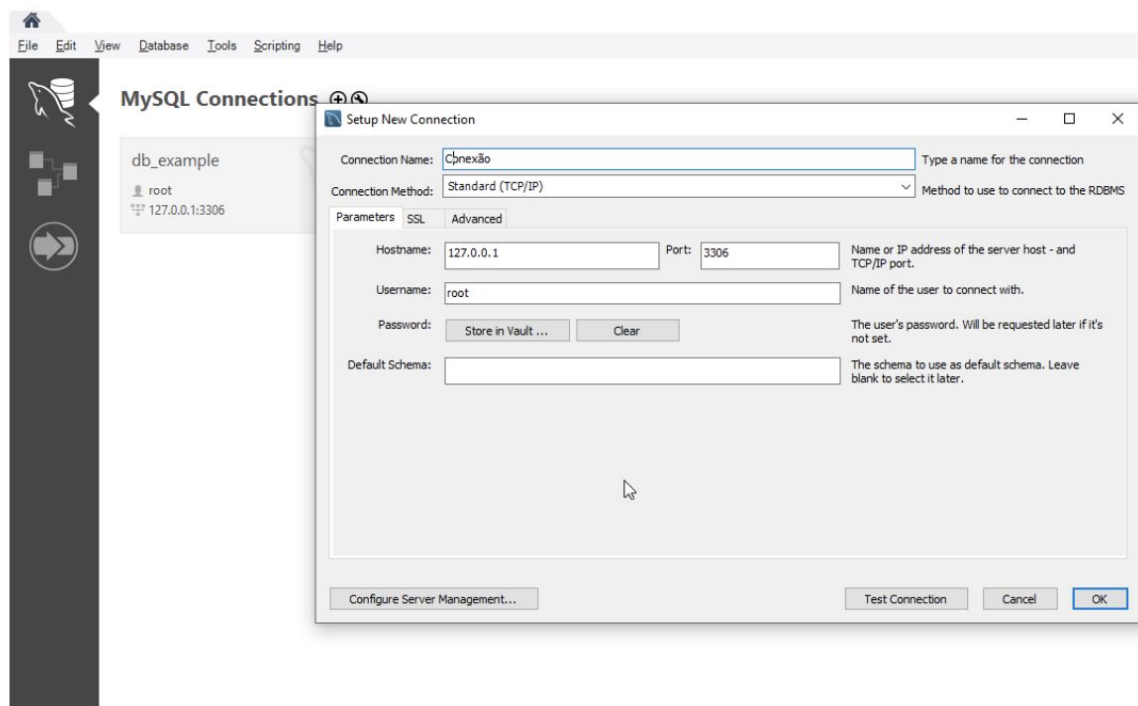
Criando nosso primeiro Banco de Dados

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

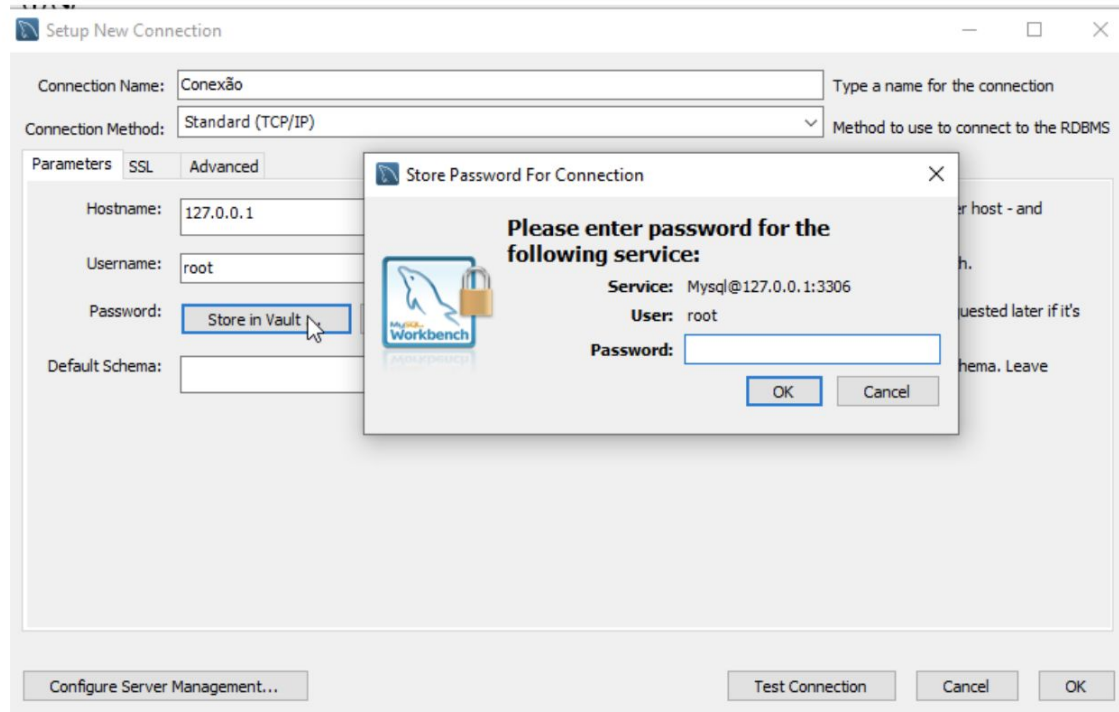
1. Uma vez que estamos com o MySql Workbench aberto, precisamos criar uma conexão local “+”.



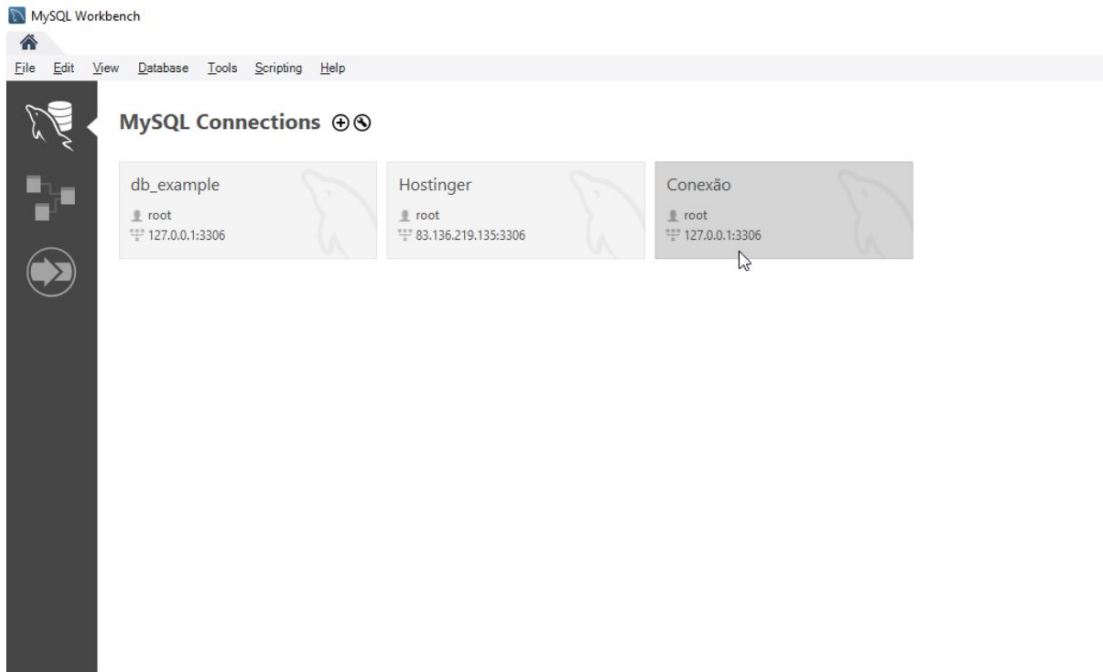
2. Defina um nome para a sua conexão.



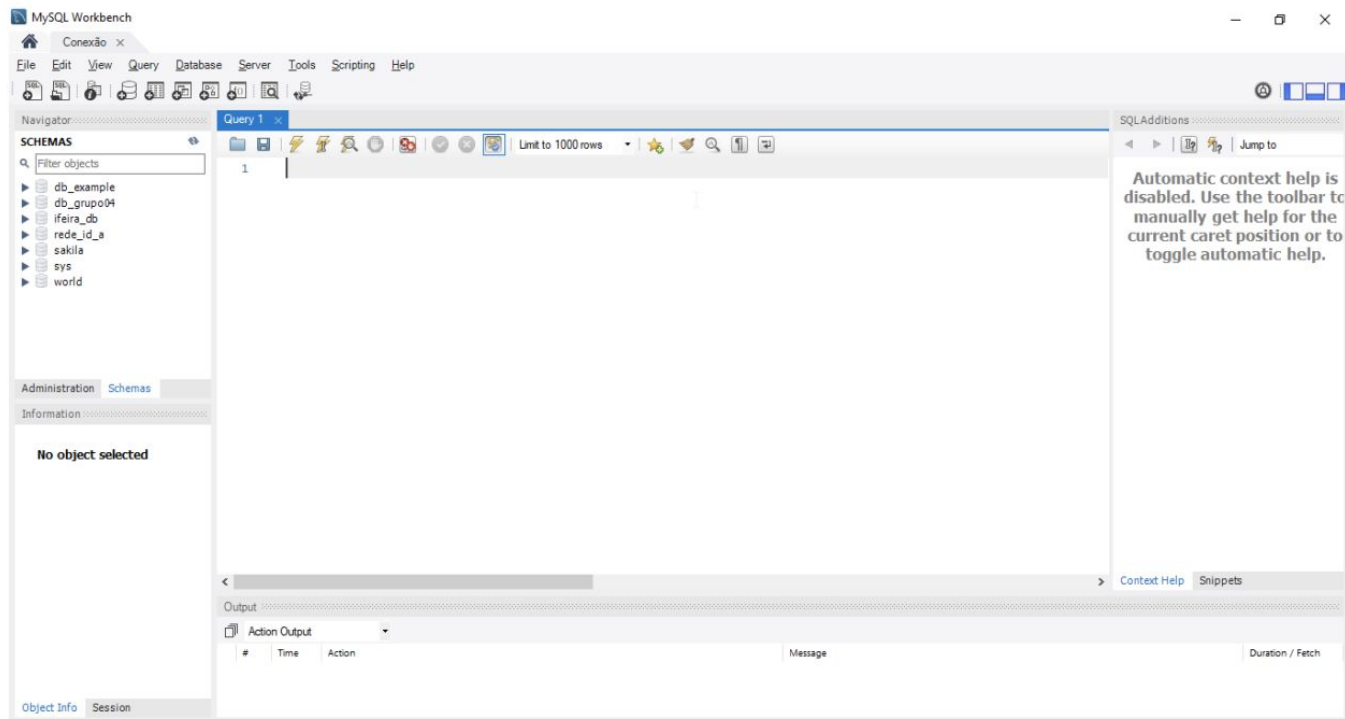
3. Defina uma senha para esta conexão clicando em Store in Vault..., e clique em OK.



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.



Criando nosso primeiro Banco de Dados

```
CREATE DATABASE bancodeteste;
```

```
CREATE TABLE
```

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

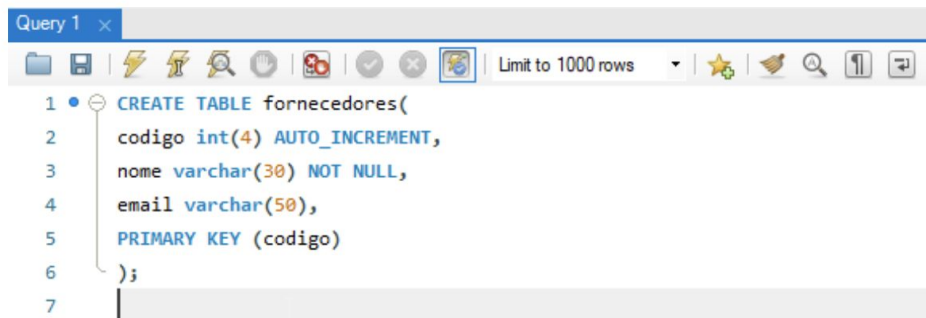
```
`CREATE TABLE fornecedores(  
codigo int(4) AUTO_INCREMENT,  
nome varchar(30) NOT NULL,  
email varchar(50),  
PRIMARY KEY (codigo)  
);
```

Criando nossa primeira tabela

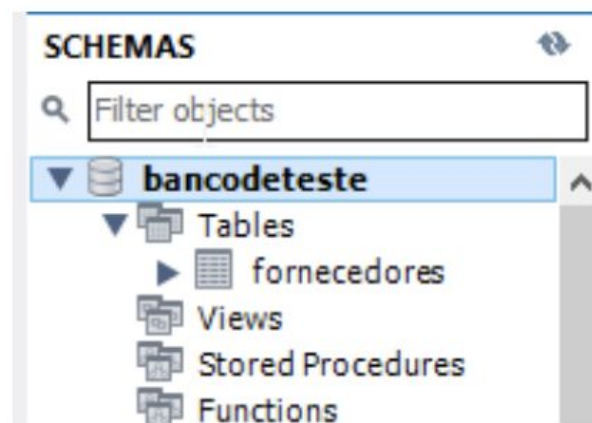
CREATE TABLE

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)  
);`
```



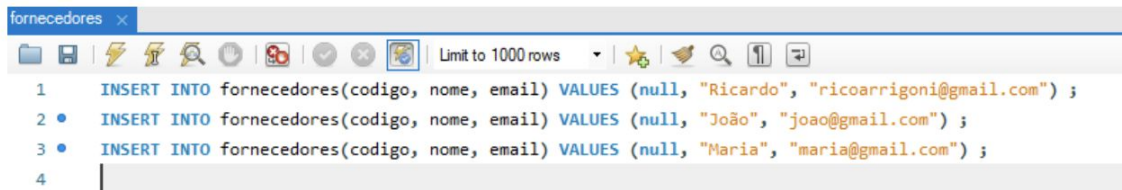
Caso a tudo ocorrer corretamente no canto esquerdo na aba das tabelas aparecerá a tabela criada.



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") ;
```



The screenshot shows a database client window with the title 'fornecedores'. The window contains a list of three SQL INSERT statements, each on a new line and numbered 1, 2, and 3. The statements are: 1. INSERT INTO fornecedores(codigo, nome, email) VALUES (null, "Ricardo", "ricoarrigoni@gmail.com") ; 2. INSERT INTO fornecedores(codigo, nome, email) VALUES (null, "João", "joao@gmail.com") ; 3. INSERT INTO fornecedores(codigo, nome, email) VALUES (null, "Maria", "maria@gmail.com") ;. The window also has a toolbar with various icons and a 'Limit to 1000 rows' dropdown menu.

```
fornecedores x  
Limit to 1000 rows  
1 INSERT INTO fornecedores(codigo, nome, email) VALUES (null, "Ricardo", "ricoarrigoni@gmail.com") ;  
2 INSERT INTO fornecedores(codigo, nome, email) VALUES (null, "João", "joao@gmail.com") ;  
3 INSERT INTO fornecedores(codigo, nome, email) VALUES (null, "Maria", "maria@gmail.com") ;  
4
```

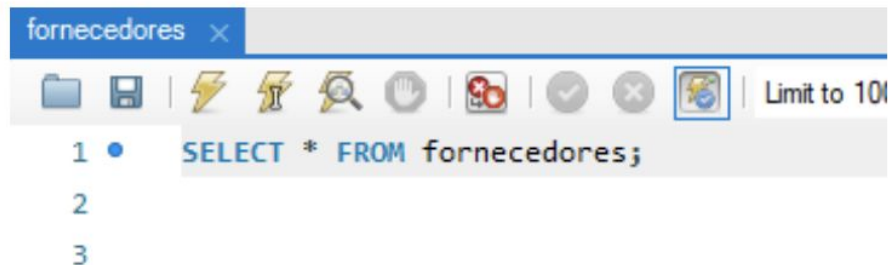
Feito isto os dados do Ricardo, João e Maria serão inseridos na tabela, o primeiro valor antes do nome, será inserido como null porque o campo código está definido com chave primária (PRIMARY KEY, AUTO_INCREMENT).

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.

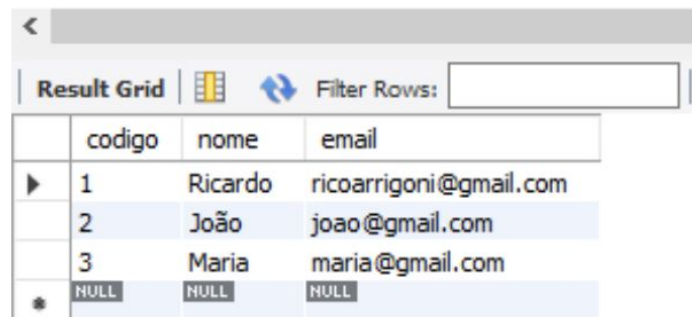
SELECT * FROM fornecedores;



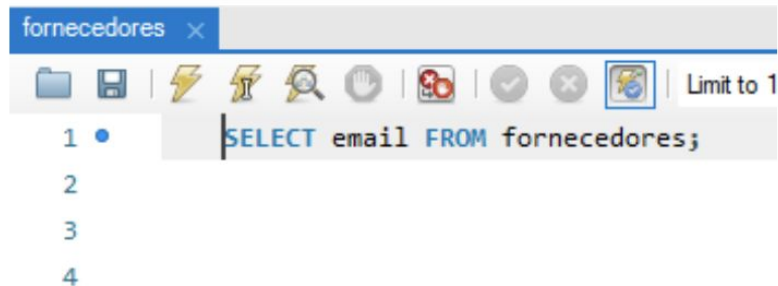
Feito isto aparecerá no log de Output a seguinte tabela

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

SELECT email FROM fornecedores;

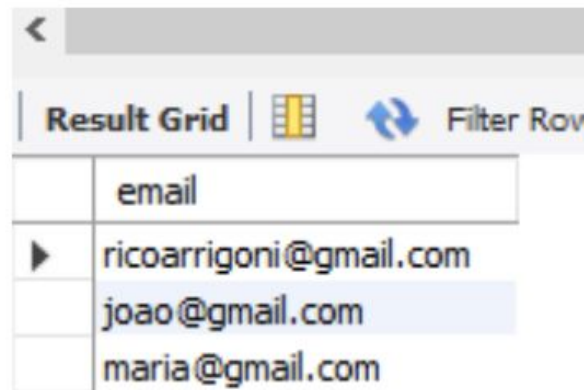


	codigo	nome	email
▶	1	Ricardo	ricoarrigoni@gmail.com
	2	João	joao@gmail.com
	3	Maria	maria@gmail.com
*	NULL	NULL	NULL



```
1 • | SELECT email FROM fornecedores;  
2  
3  
4
```

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

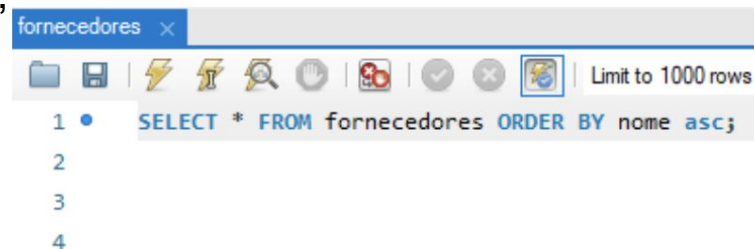


A screenshot of a database application's 'Result Grid'. The grid has a single column labeled 'email'. It contains three rows of email addresses: 'ricoarrigoni@gmail.com', 'joao@gmail.com', and 'maria@gmail.com'. The second row is highlighted with a blue background. Above the grid, there are icons for 'Result Grid', a table icon, and a 'Filter Rows' button.

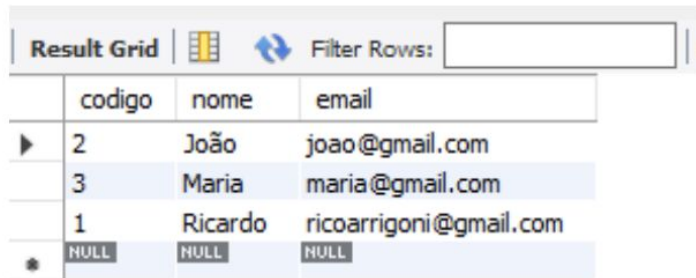
email
ricoarrigoni@gmail.com
joao@gmail.com
maria@gmail.com

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;



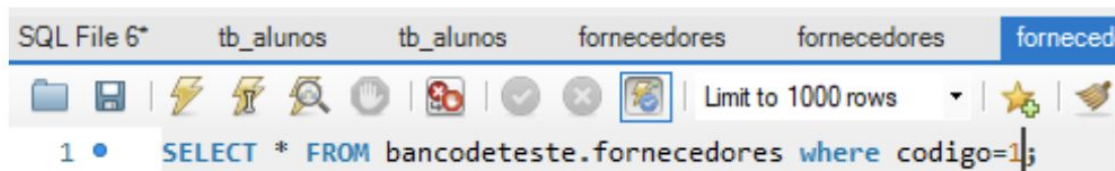
e retornaremos a tabela completa, porém ordenada por nome de forma crescente.



	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 específica, exemplo apenas as informações pertinentes ao Ricardo, basta utilizarmos o WHERE + o campo que queremos filtrar (código) + o sinal de igual e o número referente ao código do Ricardo, neste caso será o número 1. Ficará assim:

SELECT * FROM fornecedores WHERE codigo = 1;



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

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

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 formecedores 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.

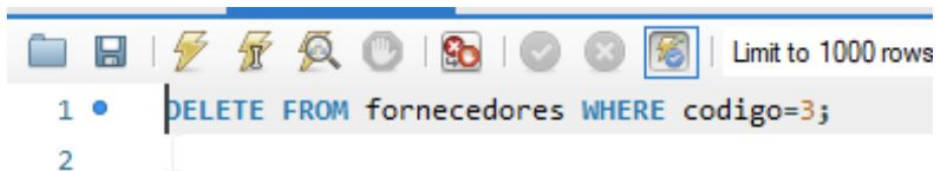
<			
Result Grid			
Filter Rows: <input type="text"/>			
	codigo	nome	email
▶	1	Ricardo Arrigoni	ricoarrigoni@gmail.com
	2	João	joao@gmail.com
	3	Maria	maria@gmail.com
*	NULL	NULL	NULL

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.fomecedores LIMIT 0, 1000
✓ 14	16:04:15	DELETE FROM fomecedores 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.

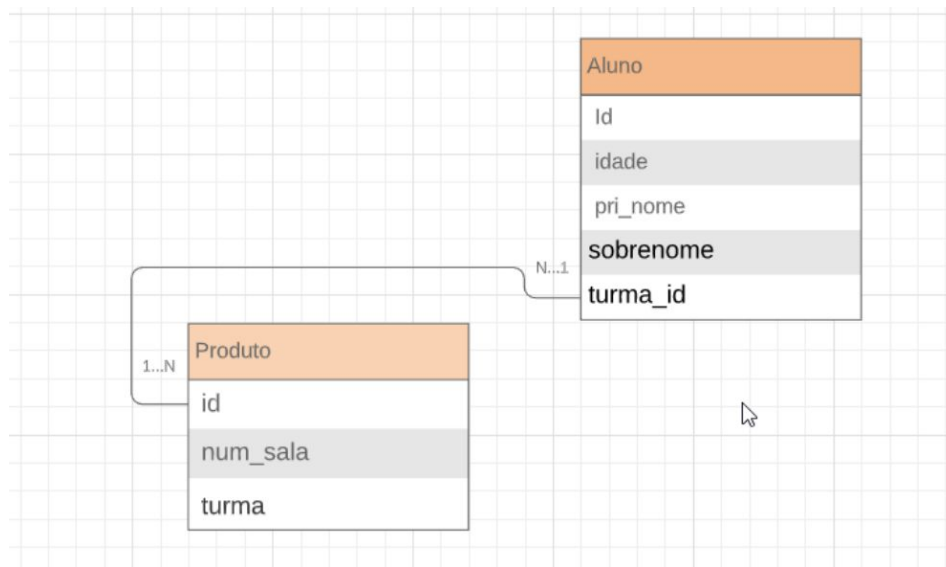
Result Grid

Filter Rows:

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

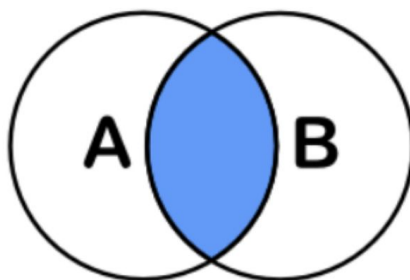
Associações entre tabelas

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.



Inner Join

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 duas categorias principais de joins:

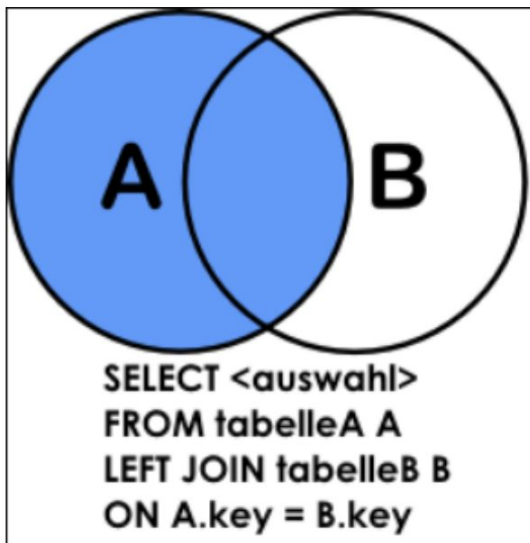


```
SELECT <auswahl>  
FROM tabelleA A  
INNER JOIN tabelleB B  
ON A.key = B.key
```

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.

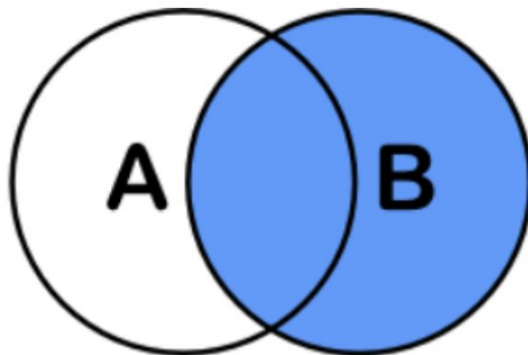
Left Join

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.



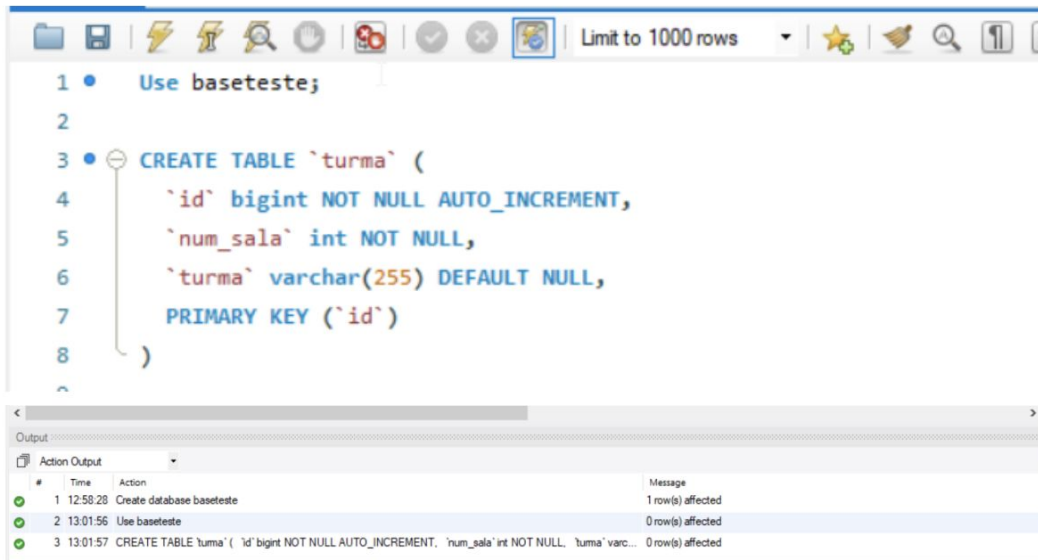
```
SELECT <auswahl>  
FROM tabelleA A  
RIGHT JOIN tabelleB B  
ON A.key = B.key
```

Exemplos

Veremos o exemplo abaixo.

Abra o arquivo de query e execute no MySQL Workbench.

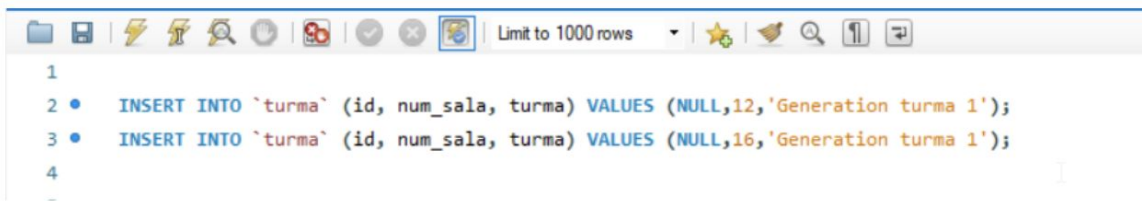
1. Crie uma bancotabela com o nome de turma



2. crie uma com o nome de aluno

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

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



The screenshot shows a SQL IDE interface with a toolbar at the top containing icons for file operations, execution, and search. Below the toolbar, a text area displays two SQL INSERT statements. The first statement inserts a row with id NULL, num_sala 12, and turma 'Generation turma 1'. The second statement inserts a row with id NULL, num_sala 16, and turma 'Generation turma 1'. The text area has a 'Limit to 1000 rows' dropdown and a search icon on the right.

```
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`)  
);
```

The screenshot shows a database IDE interface. The top toolbar includes icons for file operations, execution, and search. A dropdown menu is set to "Limit to 1000 rows". The main editor displays the following SQL code:

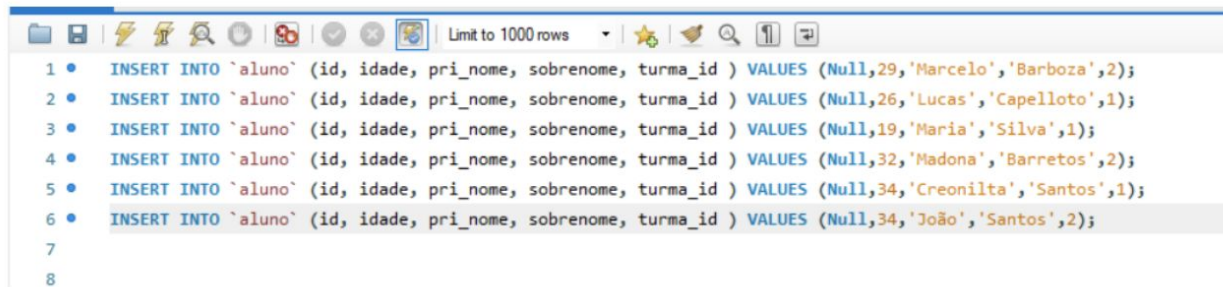
```
1 • Use baseteste;
2
3 • CREATE TABLE `aluno` (
4     `id` bigint NOT NULL AUTO_INCREMENT,
5     `idade` int NOT NULL,
6     `pri_nome` varchar(255) DEFAULT NULL,
7     `sobrenome` varchar(255) DEFAULT NULL,
8     `turma_id` bigint DEFAULT NULL,
9     PRIMARY KEY (`id`),
10    FOREIGN KEY (`turma_id`)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

Below the editor is the "Output" panel, which shows the execution results in a table:

#	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
```

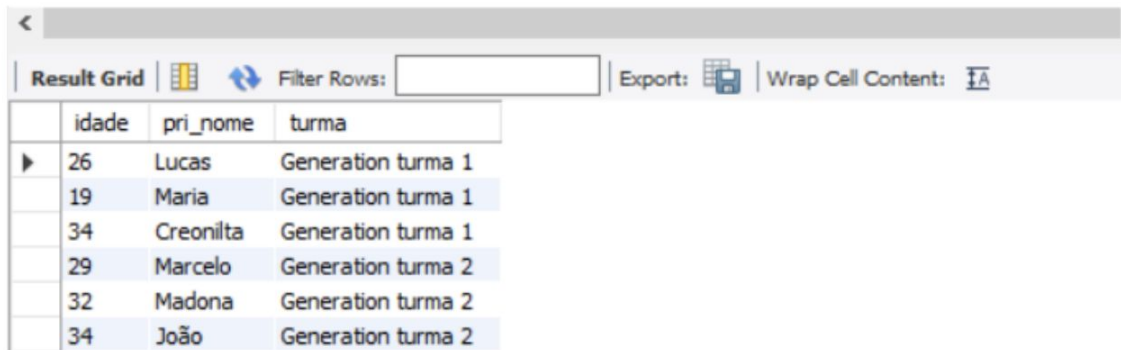


The screenshot shows a SQL IDE window with a toolbar at the top and a list of six SQL statements. The statements are numbered 1 through 6 on the left margin. Each statement is an INSERT INTO query for the 'aluno' table, with columns (id, idade, pri_nome, sobrenome, turma_id) and specific values in parentheses. The values for 'id' are all NULL, 'idade' values are 29, 26, 19, 32, 34, and 34 respectively. 'pri_nome' values are 'Marcelo', 'Lucas', 'Maria', 'Madona', 'Creonilta', and 'João'. 'sobrenome' values are 'Barboza', 'Capelloto', 'Silva', 'Barretos', 'Santos', and 'Santos'. 'turma_id' values are 2, 1, 1, 2, 1, and 2 respectively. The sixth statement is highlighted with a grey background.

```
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
```



The screenshot shows a database interface with a toolbar containing 'Result Grid', 'Filter Rows', 'Export', and 'Wrap Cell Content'. Below the toolbar is a table with 4 columns: 'idade', 'pri_nome', and 'turma'. The table contains 6 rows of data, representing the result of an INNER JOIN query. The first row is highlighted with a mouse cursor.

	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

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
```

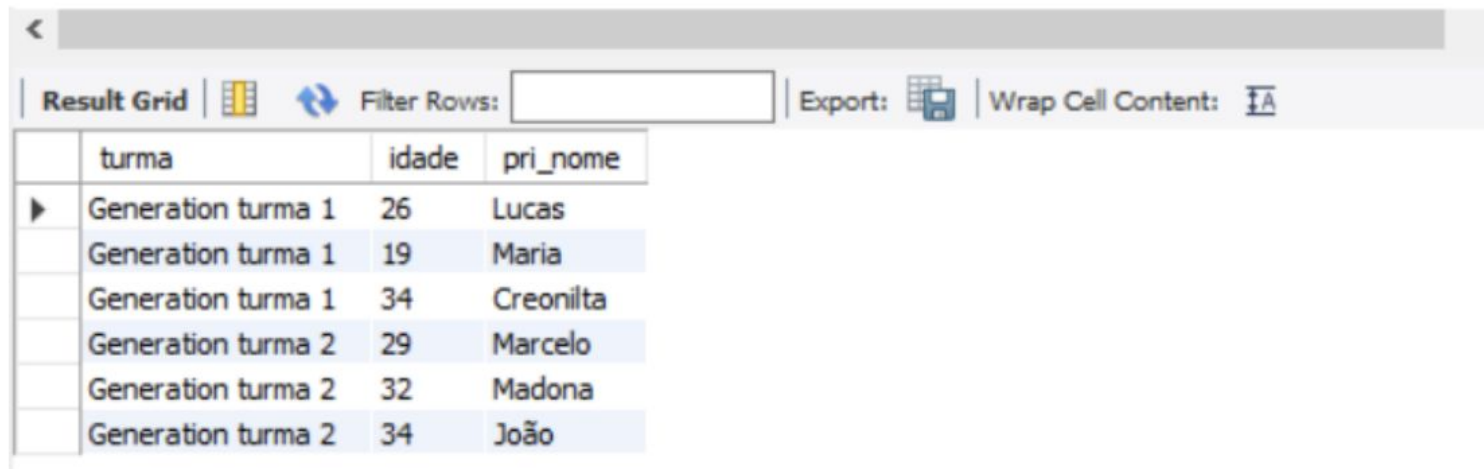


The screenshot shows a database interface with a toolbar containing 'Result Grid', 'Filter Rows', 'Export', and 'Wrap Cell Content'. Below the toolbar is a table with 4 columns: 'turma', 'idade', and 'pri_nome'. The table contains 6 rows of data, representing the result of a LEFT JOIN query. The first row is highlighted with a mouse cursor.

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

7. Digite a seguinte query digite a seguinte query para **RIGHT JOIN**

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



The screenshot shows a database interface with a query result grid. The grid has a toolbar at the top with options like 'Result Grid', 'Filter Rows', 'Export', and 'Wrap Cell Content'. The data is displayed in a table with four columns: 'turma', 'idade', and 'pri_nome'. The first column is expanded, showing a list of rows with a right-pointing arrow icon next to each row.

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