

# Banco de Dados II

Material de Aula

**MySql**

**Prof. Enildo**

E-mail: [professor.enildo@hotmail.com](mailto:professor.enildo@hotmail.com)

# Sumário

Introdução.....	3
Linguagem SQL.....	3
Banco de dados relacional.....	3
SQL é igual em todos os bancos de dados.....	3
Subconjuntos da linguagem SQL.....	4
Leitura complementar.....	4
Implementação de banco de dados.....	5
Comentários na query.....	5
Criação e exclusão de banco de dados.....	5
Create - Criando uma base de dados.....	5
Símbolos em sintaxe.....	6
Drop - Excluindo uma base de dados.....	6
Show – Mostrando as bases de dados.....	6
Tipos de Dados.....	7
Comandos SQL.....	9
Sub-Linguagem (DDL).....	9
CREATE.....	9
Describe.....	10
ALTER.....	11
SQL Constraints (Restrições).....	12
NOT NULL.....	12
UNIQUE.....	12
DEFAULT.....	13
CHECK.....	13
PRIMARY KEY.....	13
FOREIGN KEY.....	14
AUTO_INCREMENT.....	15
INDEX.....	16
RENAME.....	16
FUNÇÕES.....	17
Linguagem de manipulação de dados – DML.....	18
INSERT.....	18
DELETE.....	19
SQL_SAFE_UPDATES.....	19

UPDATE .....	20
Linguagem de consulta de dados – DQL.....	20
SELECT .....	20
Blocos de linguagem de consulta estruturada (SQL) .....	21
Stored Procedures.....	21
Funções de string do MySQL .....	23
Funções Numéricas do MySQL .....	24
Funções de data do MySQL .....	25
Funções avançadas do MySQL.....	26
Variáveis e constantes.....	27
Variáveis .....	27
Constantes.....	28
IF-THEN-ELSE .....	29
Os operadores AND, OR e NOT do MySQL .....	30
O operador MySQL EXISTS.....	31
Order by .....	31
LIMIT .....	33
Função MySQL STR_TO_DATE().....	34
Triggers em MySQL.....	36
NEW e .OLD .....	37
Tabelas Históricas.....	38
Exemplos de gatilhos no MySQL.....	39
JOIN .....	45
INNER JOIN .....	45
LEFT JOIN .....	47
RIGHT JOIN .....	48
FULL JOIN.....	49
UNION .....	50
CROSS JOIN.....	52
Exceções (tratamentos de erros).....	54
PROXIMOS CAPITULOS .....	61

## Introdução

Este material foi produzido com a finalidade de auxiliar nas aulas de banco de dados. Aqui irei apresentar comandos da linguagem Sql padrão ANSI, por utilizar o SGBD MySql com a interface gráfica no MySql Workbench, aparecera comandos que só irão funcionar no MySql. Mas não se preocupe, os comandos que não fazem parte da linguagem Sql padrão ANSI, você pode pesquisar por equivalentes em qualquer outro SGBD, ou seja, procure a sintaxe equivalente no SGBD que estiver utilizando.

## Linguagem SQL

Existem inúmeras linguagens no mercado de TI, linguagens de programação orientadas a objeto, estruturadas, de marcação de texto entre outras. A linguagem voltada para banco de dados é a SQL (Linguagem de consulta estruturada, em português) é uma linguagem que todo programador, técnico ou administrador de banco de dados deve conhecer. Sua aplicação é extremamente ampla no mercado de banco de dados e programação.

A linguagem SQL (Structure query Language - Linguagem de Consulta Estruturada) é a linguagem padrão ANSI (American National Standards Institute - Instituto Nacional de Padronização Americano) para a operação em bancos de dados relacionais. A linguagem SQL foi criada para atender a todos os bancos de dados relacionais e permitir que usuários possam acessar qualquer banco usando a mesma base de conhecimento.

## Banco de dados relacional

É um tipo de banco onde a sua estrutura ou objetos relacionam-se entre si. Este é o ponto alto dos bancos relacionais, pois ao atrelar um objeto a outro ele se torna muito mais consistente, evitando acidentes com os dados e garantindo a integridade.

O termo muito comum neste tipo de banco é a chamada integridade relacional onde um objeto A relaciona-se com um objeto B atrelando uma chave primária a uma chave estrangeira.

## SQL é igual em todos os bancos de dados

Inicialmente imagina-se que a linguagem SQL seria a mesma para todos os bancos de dados, porém, algumas empresas desenvolvedoras de banco de dados, entre elas a Microsoft e a Oracle, fizeram aperfeiçoamentos na linguagem SQL e acabaram criando versões próprias de acesso ao seu banco de dados.

Geralmente o padrão é chamado de SQL ANSI que é padronizado e serve para qualquer banco de dados. Já os específicos podem ser o PL SQL adotado pela Oracle ou o Transact-SQL adotado pela Microsoft para o seu principal banco de dados. <sup>1</sup>

---

<sup>1</sup> <https://www.luis.blog.br/o-que-e-sql.html>

## Subconjuntos da linguagem SQL



2

## Leitura complementar

Veja a seguir alguns links com arquivos e publicações interessante que falam sobre este assunto:

<https://www.luis.blog.br/o-que-e-sql.html>

<https://pt.wikipedia.org/wiki/SQL>

<https://dicasdeprogramacao.com.br/o-que-e-sql/>

<https://www.devmedia.com.br/guia/guia-completo-de-sql/38314>

<https://blog.betrybe.com/sql/>

---

<sup>2</sup> <https://blog.betrybe.com/sql/>

## Implementação de banco de dados

### Comentários na query

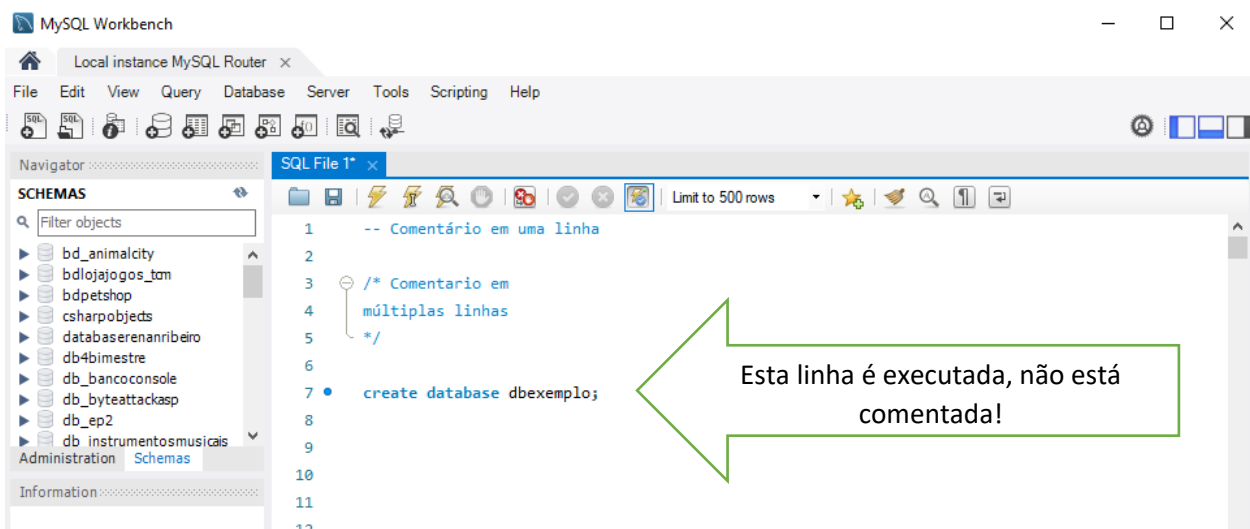
Antes de começarmos, vamos aprender como fazer comentários no arquivo do tipo .sql que nada mais é que um arquivo texto. As que estiverem comentadas não são executadas!

-- Comentário em uma linha

/\* Comentário em

múltiplas linhas

\*/



## Criação e exclusão de banco de dados

### Create - Criando uma base de dados

Sintaxe para criar uma base de dados:

**CREATE DATABASE <Nome da Base> [;]**

- CREATE: é um comando SQL para criar objetos.
- DATABASE: é o objeto do tipo base de dados.
- <Nome da Base>: é o nome que damos a base de dados.
- [;]: ponto e vírgula indica o término do comando.

Exemplo: **CREATE DATABASE dbExemplo;**

## Símbolos em sintaxe

Para conseguirmos ler e compreender as sintaxes com mais facilidade precisamos entender o que significa alguns símbolos:

- < > substituir por nome
- [] opcional
- | separador de opções

## Drop - Excluindo uma base de dados

Sintaxe para excluir uma base de dados:

**DROP DATABASE** <Nome da Base> [;]

- DROP: é um comando SQL para excluir objetos.

Exemplo: **DROP DATABASE** dbExemplo;

O comando drop pode ser entendido como excluir, apagar, eliminar, e em banco de dados a palavra dropar é a mais utilizada. Todas querem dizer a mesma coisa, retirar o objeto em questão do SGBD.

## Show – Mostrando as bases de dados

O comando show mostra todos os bancos de dados existentes em um SGBD, mas este comando não é padrão Sql ANSI! Mas conforme já comentado, o nosso SGBD em estudo é o MySql, e o MySql ele é utilizado para visualizar objetos existente.

Sintaxe:

**show databases;**

**Dica:** Certifique-se de ter privilégios de administrador antes de criar qualquer banco de dados. Uma vez que um banco de dados é criado, você pode verificá-lo na lista de bancos de dados com o seguinte comando SQL: **SHOW DATABASES;**

## Indicação de leitura complementar

[https://www.w3schools.com/sql/sql\\_create\\_db.asp](https://www.w3schools.com/sql/sql_create_db.asp)

<https://www.tutorialspoint.com/sql/sql-create-database.htm>

<http://www.bosontreinamentos.com.br/mysql/mysql-comandos-show-describe-e-mysqlshow-39/>

## Tipos de Dados

Os tipos de dados apresentados aqui têm como objetivo atender ao SGBD MySQL, mesmo que baseados no padrão ANSI, algumas particularidades podem ocorrer.

Um desenvolvedor SQL deve decidir que tipo de dados serão armazenados dentro de cada coluna ao criar uma tabela. O tipo de dados é uma diretriz para o SQL entender que tipo de dados é esperado dentro de cada coluna e identifica como o SQL irá interagir com os dados armazenados.<sup>3</sup>

### Tipos de dados de string

Tipo de Dado	Descrição
CHAR(tamanho)	Uma string de comprimento fixo (pode conter letras, números e caracteres especiais). O parâmetro “tamanho” especifica o comprimento da coluna em caracteres - pode ser de 0 a 255. O padrão é 1
VARCHAR(tamanho)	Uma string de comprimento VARIABLE (pode conter letras, números e caracteres especiais). O parâmetro “tamanho” especifica o comprimento máximo da coluna em caracteres - pode ser de 0 a 65535
TEXT(tamanho)	Contém uma string com comprimento máximo de 65.535 bytes
NCHAR(tamanho)	Dados Unicode de comprimento fixo. Máximo 4000 caracteres
NVARCHAR(tamanho)	Dados Unicode de comprimento variável. Máximo 4000 caracteres
BINARY(tamanho)	Igual a CHAR(), mas armazena strings de bytes binários. O parâmetro “tamanho” especifica o comprimento da coluna em bytes. O padrão é 1
VARBINARY(tamanho)	Igual a VARCHAR(), mas armazena strings de bytes binários. O parâmetro “tamanho” especifica o comprimento máximo da coluna em bytes.
TINYBLOB	Para BLOBs (objetos binários grandes). Comprimento máximo: 255 bytes
TINYTEXT	Contém uma string com um comprimento máximo de 255 caracteres
BLOB(tamanho)	Para BLOBs (Binary Large Objects). Armazena até 65.535 bytes de dados
MEDIUMTEXT	Contém uma string com comprimento máximo de 16.777.215 caracteres
MEDIUMBLOB	Para BLOBs (objetos binários grandes). Armazena até 16.777.215 bytes de dados
LONGTEXT	Contém uma string com um comprimento máximo de 4.294.967.295 caracteres
LOBLOB	Para BLOBs (objetos binários grandes). Armazena até 4.294.967.295 bytes de dados

---

<sup>3</sup> <http://www.w3big.com/pt/mysql/mysql-data-types.html>



### Tipos de dados numéricos inteiros

Tipo de Dado	Descrição
BIT	Um tipo de valor de bit. O número de bits por valor é especificado em tamanho.
TINYINT	Um número inteiro muito pequeno. O intervalo assinado é de -128 a 127. O intervalo não assinado é de 0 a 255.
BOOLEAN ou BOOL	Zero é considerado falso, valores diferentes de zero são considerados verdadeiros.
SMALLINT	Um inteiro pequeno. O intervalo assinado é de -32768 a 32767. O intervalo não assinado é de 0 a 65535.
MEDIUMINT	Um inteiro médio. O intervalo assinado é de -8388608 a 8388607. O intervalo não assinado é de 0 a 16777215.
INT ou INTEGER	Um inteiro médio. O intervalo assinado é de -2147483648 a 2147483647. O intervalo não assinado é de 0 a 4294967295.
BIGINT	Um número inteiro grande. O intervalo assinado é de -9223372036854775808 a 9223372036854775807.

### Tipos de dados ponto flutuante

Tipo de Dado	Descrição
FLOAT(tamanho, d)	Um número de ponto flutuante. O número total de dígitos é especificado em tamanho. O número de dígitos após o ponto decimal é especificado no parâmetro d.
DOUBLE(tamanho, d)	Um número de ponto flutuante de tamanho normal. O número total de dígitos é especificado em tamanho. O número de dígitos após o ponto decimal é especificado no parâmetro d.
DECIMAL(tamanho, d)	Um número de ponto fixo exato. O número total de dígitos é especificado em tamanho. O número de dígitos após o ponto decimal é especificado no parâmetro d. O número máximo para tamanho é 65. O número máximo para d é 30. O valor padrão para tamanho é 10. O valor padrão para d é 0.

### Tipos de dados de data e hora

Tipo de Dado	Descrição
DATE	DATA Uma data. Formato: AAAA-MM-DD. O intervalo suportado é de '1000-01-01' a '9999-12-31'
TIME	Hora. Formato: hh:mm:ss. O intervalo suportado é de '00:00:00' a '23:59:59'.
DATETIME	Uma combinação de data e hora. Formato: AAAA-MM-DD hh:mm:ss. O intervalo suportado é de '1000-01-01 00:00:00' a '9999-12-31 23:59:59'.

**Nota:** Todos os tipos de dados numéricos podem ter uma opção extra: UNSIGNED ou ZEROFILL. Se você adicionar a opção UNSIGNED, o MySQL não permitirá valores negativos para a coluna. Se você adicionar a opção ZEROFILL, o MySQL automaticamente também adiciona o atributo UNSIGNED à coluna, alguns tipos podem ser determinados em tipos como INT e outros, lembre-se que estas características são do MySQL.

## Comandos SQL

### Sub-Linguagem (DDL)

Linguagem de definição de dados – DDL, são utilizados quando falamos de objetos, sendo eles base de dados, tabelas, stored procedures, ...

Os principais comando desta linguagem são:

- **CREATE** - comando para criar objeto
- **DROP** - comando para apagar objeto
- **ALTER** - comando para modificar objeto

### CREATE

Comando CREATE é utilizado para criar objeto.

**Sintaxe:** **create** <tipo\_objeto> <nome\_objeto> [;]

Vamos relembrar o comando para criar uma base de dados:

- **create database** dbNome;

Uma base de dados não requer parâmetros obrigatórios, diferentes da criação de uma tabela.

**Sintaxe:** **create table** <nome\_tabela> (<parâmetro1>, <parâmetro1>,...) [;]

- **Create** - comando para criar objeto
- **Table** - objeto do tipo tabela
- <nome\_tabela> - aqui colocamos o nome da tabela
- () – entre os parênteses colocamos os parâmetros da tabela
- <parâmetro> - os parâmetros de uma tabela são os campos “colunas/atributos”
- ; - o ponto e vírgula marca o término do comando, no MySQL, são obrigatórios

Falando sobre parâmetros, um campo em uma tabela, vejamos como declarar um atributo:

**Sintaxe:** <nome\_campo> <tipo> <restrições>

Onde:

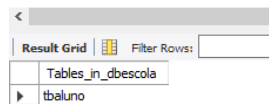
- <nome\_campo> - é o nome do campo propriamente dito, exemplo: “Id, Nome, ...”
- <tipo> - são os tipos de dados que serão armazenados neste campo, exemplo: “int, numeric, varchar”
- <restrições> - são as regras para inserir um dado ou se ele pode ficar sem preenchimento, exemplo: “null, not null, ...”

## Exemplo:

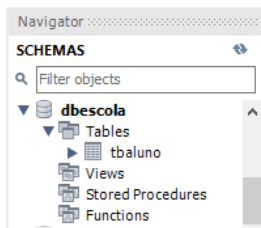
```
create table tbAluno(  
AlunoID int not null,  
Nome varchar(50) null,  
Endereco varchar(50) null,  
Estado char(2) not null  
);
```

Comando para visualizar as tabelas existente na base de dados em uso é o comando show, veja exemplo a seguir:

```
show tables;
```



No MySQL Workbench podemos visualizar na janela navigator, veja exemplo a seguir:



## Describe

Retorna as informações básicas de metadados de uma tabela. As informações de metadados incluem o nome da coluna, o tipo de coluna e o comentário da coluna. Opcionalmente, você pode especificar uma especificação de partição ou um nome de coluna para retornar os metadados pertencentes a uma partição ou coluna, respectivamente.

### Sintaxe:

```
describe tbAluno;
```

	Field	Type	Null	Key	Default	Extra
▶	AlunoID	int	NO		NULL	
	Nome	varchar(50)	YES		NULL	
	Endereco	varchar(50)	YES		NULL	
	Estado	char(2)	YES		NULL	

## ALTER

Comandos da sub-linguagem SQL DDL, o comando ALTER é utilizado para alterar/modificar objeto.

**Sintaxe:** **alter table** <nome\_tabela> <tipo\_alteração> [**column**] <coluna\_p\_alteração> [<tipo> <restrições>] [;]

- **<nome\_tabela>:** aqui colocamos o nome da tabela a qual deve ser alterada, não coloque os sinais de < e >.
  - **<tipo\_alteração>** [**DROP** | **ADD** | **MODIFY**]
  - [**column**]: comando que indica que é uma coluna.
  - **<coluna\_p\_alteração>:** nome da coluna que será alterada.
  - **<tipo>:** são os tipos de dados que serão armazenados neste campo, exemplo: “int, numeric, varchar”
  - **<restrições>:** são as regras para inserir um dado ou se ele pode ficar sem preenchimento, exemplo: “null, not null, ...”
  - [;]: determina o termino do comando sql.
- 
- **DROP** : apaga um objeto/atributo
  - **ADD** : adiciona um objeto/atributo
  - **MODIFY**: altera um objeto/atributo

Exemplo excluindo uma coluna existente na tabela:

```
alter table tbAluno drop Estado;
```

Exemplo incluindo uma coluna na tabela:

```
alter table tbAluno add Estado char (2) null;
```

Exemplo (1) alterando uma coluna na tabela:

```
alter table tbAluno modify Estado int;
```

-- Exemplo (2) alterando uma coluna na tabela:

```
alter table tbAluno modify column Estado char (2) null;
```

## SQL Constraints (Restrições)

As restrições são regras aplicadas nas colunas de uma tabela, usadas para limitar os tipos de dados que são inseridos.

Podem ser especificadas no momento de criação da tabela (CREATE) ou após a tabela ter sido criada (ALTER).

As principais constraints são as seguintes:

- NOT NULL
- UNIQUE
- DEFAULT
- CHECK
- PRIMARY KEY
- FOREIGN KEY

### NOT NULL

A constraint NOT NULL impõe a uma coluna a NÃO aceitar valores NULL, ou seja, obriga um campo a sempre possuir um valor. Deste modo, não é possível inserir um registro (ou atualizar) sem entrar com um valor neste campo.

**Exemplo de sintaxe utilizando a restrição not null em uma coluna:**

```
create table tbconstraint(
```

```
NomeRestricao varchar (60) not null
```

```
);
```



### UNIQUE

A restrição UNIQUE identifica de forma única cada registro em uma tabela de um banco de dados.


As constraints UNIQUE e PRIMARY KEY garantem a unicidade em uma coluna ou conjunto de colunas.

Uma constraint PRIMARY KEY automaticamente possui uma restrição UNIQUE definida, portanto não é necessário especificar essa constraint neste caso.

É possível termos várias constraints UNIQUE em uma mesma tabela, mas apenas uma Chave Primária por tabela (lembrando que uma PK pode ser composta, ou seja, constituída por mais de uma coluna – mas ainda assim, será uma única chave primária).

### Exemplo de sintaxe utilizando a restrição unique em uma coluna:

```
create table tbconstrant(  
NomeRestricao varchar (60) not null  
Cnpj bigint unique  
);
```




### DEFAULT

A restrição DEFAULT é usada para inserir um valor padrão especificado em uma coluna.

O valor padrão será adicionado a todos os novos registros caso nenhum outro valor seja especificado na hora de inserir dados.

### Exemplo de sintaxe utilizando a restrição default em uma coluna:

```
CREATE TABLE Pessoa (  
Cidade varchar(255) DEFAULT 'São Paulo'  
);
```




### CHECK

A CHECK restrição limita o valor que pode ser colocado em uma coluna.

### Exemplo de sintaxe utilizando a restrição check em uma coluna:

```
CREATE TABLE Pessoa (  
Idade int,  
CHECK (Idade >= 18)  
);
```



### PRIMARY KEY

A restrição PRIMARY KEY (Chave Primária) identifica de forma única cada registro em uma tabela de banco de dados.

As Chaves Primárias devem sempre conter valores únicos.

Uma coluna de chave primária não pode conter valores NULL

Cada tabela deve ter uma chave primária e apenas uma chave primária.

**Exemplo de sintaxe utilizando a restrição primary key em uma coluna:**

```
CREATE TABLE Pessoa (  
  ID int PRIMARY KEY  
);
```



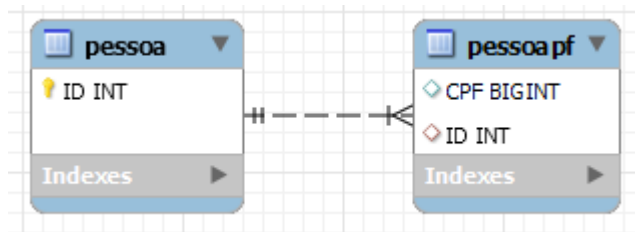
## FOREIGN KEY

Uma FOREIGN KEY (Chave Estrangeira) em uma tabela é um campo que aponta para uma chave primária em outra tabela. Desta forma, é usada para criar os relacionamentos entre as tabelas no banco de dados.<sup>4</sup>

**Exemplo de sintaxe utilizando a restrição foreign key em uma coluna:**

```
CREATE TABLE Pessoa (  
  ID int PRIMARY KEY  
);
```

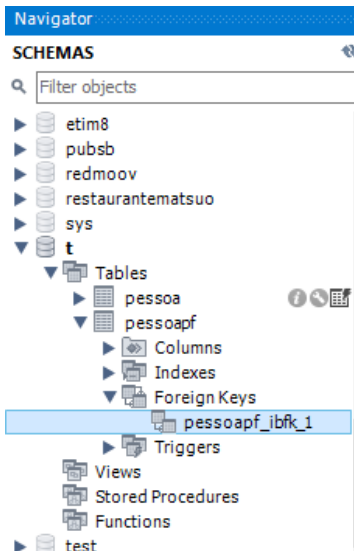
```
CREATE TABLE PessoaPF (  
  CPF bigint,  
  ID int,  
  foreign key (ID) references Pessoa(ID)  
);
```



**Vídeo sobre Restrições:**

<https://www.youtube.com/watch?v=10Gu4XZBMdo&t=3s>

<sup>4</sup> [http://www.bosontreinamentos.com.br/mysql/mysql-constraints-restricoes-primary-key-fk-default-etc-06/#:~:text=SQL%20Constraints%20\(Restri%C3%A7%C3%B5es\)%20no%20MySQL,de%20dados%20que%20s%C3%A3o%20inseridos.](http://www.bosontreinamentos.com.br/mysql/mysql-constraints-restricoes-primary-key-fk-default-etc-06/#:~:text=SQL%20Constraints%20(Restri%C3%A7%C3%B5es)%20no%20MySQL,de%20dados%20que%20s%C3%A3o%20inseridos.)

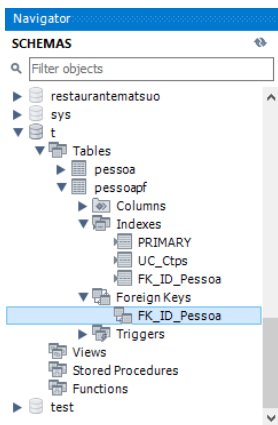


Como podemos ver na aba de navegação em schemas, que as restrições são objetos da tabela nomeados automaticamente!

Mas é possível nomeá-los? Sim, é possível!

Para permitir a nomeação de uma restrição podemos utilizar o comando **CONSTRAINT**.

## . Exemplo de sintaxe utilizando restrições nomeadas:



```
CREATE TABLE PessoaPF (
  ID int not null,
  CPF bigint,
  CTPS int,
  Idade smallint,
  CONSTRAINT CHK_Idade CHECK (Idade>=18),
  CONSTRAINT UC_Ctps UNIQUE (CTPS),
  CONSTRAINT PK_PessoaPF PRIMARY KEY (CPF),
  CONSTRAINT FK_ID_Pessoa foreign key (ID) references Pessoa(ID)
);
```

## Veja um exemplo apagando uma FOREIGN KEY:

```
ALTER TABLE PessoaPF DROP FOREIGN KEY FK_ID_Pessoa;
```

## AUTO\_INCREMENT

O incremento automático “**AUTO\_INCREMENT**” permite que um número único seja gerado automaticamente quando um novo registro é inserido em uma tabela.

Muitas vezes este é o campo de chave primária que gostaríamos que fosse criado automaticamente toda vez que um novo registro fosse inserido.



**Exemplo de sintaxe utilizando a restrição primary key e o auto\_increment:**

```
CREATE TABLE Pessoa (  
    PessoaId int NOT NULL AUTO_INCREMENT,  
    primary key (PessoaId)  
);
```



## INDEX

Os índices são usados para recuperar dados do banco de dados mais rapidamente do que de outra forma. Os usuários não podem ver os índices, eles são usados apenas para acelerar as buscas/consultas.

**Nota:** Atualizar uma tabela com índices leva mais tempo do que atualizar uma tabela sem (porque os índices também precisam de uma atualização). Portanto, crie índices apenas em colunas que serão pesquisadas com frequência.

A CREATE INDEX instrução é usada para criar índices em tabelas. <sup>5</sup>

### Sintaxe:

```
CREATE INDEX <nome_do_index> ON <tabela> (Coluna_da_tabela);
```

**Exemplo de sintaxe criando um index em uma tabela.**

```
CREATE INDEX idx_Nome ON Pessoa (Name);
```

## RENAME

Comando para renomear uma tabela no MySQL é o RENAME, a sintaxe é bastante simples:

```
RENAME TABLE Pessoa TO tbPessoa;
```

Com o comando RENAME podemos também renomear as colunas de uma tabela.

### Sintaxe:

```
ALTER TABLE nome_da_tabela RENAME COLUMN nome_do_campo_atual TO  
nome_do_campo_novo;
```

**Exemplo de sintaxe renomeando uma coluna de uma tabela.**

```
ALTER TABLE tbPessoa RENAME COLUMN ID TO PessoaID;
```

---

<sup>5</sup> [https://www.w3schools.com/sql/sql\\_create\\_index.asp](https://www.w3schools.com/sql/sql_create_index.asp)

## FUNÇÕES

Vejamos algumas funções uteis:

A função `CURRENT_DATE()` retorna a data atual.

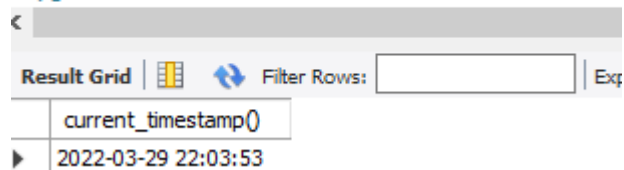
A função `CURRENT_TIME()` retorna a hora atual.

A função `CURRENT_TIMESTAMP()` retorna a data e hora atual.

A função `CURRENT_USER()` retorna o usuário logado.

**Veja um exemplo com a função `current_timestamp`:**

```
77 • select current_timestamp();
78
```



The screenshot shows a SQL query result in a table format. The table has one column named `current_timestamp()` and one row with the value `2022-03-29 22:03:53`. The interface includes a 'Result Grid' tab, a 'Filter Rows' input field, and an 'Exp' button.

current_timestamp()
2022-03-29 22:03:53

## Linguagem de manipulação de dados – DML

A manipulação de dados significa:

- a busca da informação armazenada no BD;
- a inserção de novas informações no BD;
- a eliminação de informações no BD;
- a modificação de dados armazenados no BD.

Linguagem de Manipulação de Dados ( DML, de Data Manipulation Language) é uma família de linguagens de computador utilizada para a recuperação, inclusão, remoção e modificação de informações em bancos de dados. Pode ser procedural, que especifica como os dados devem ser obtidos do banco; pode também ser declarativa (não procedural), em que os usuários não necessitam especificar o caminho de acesso, isto é, como os dados serão obtidos. O padrão SQL é não procedural. DMLs foram utilizadas inicialmente apenas por programas de computador, porém (com o surgimento da SQL) também têm sido utilizadas por pessoas.

### Principais comandos

As DMLs têm sua capacidade funcional organizada pela palavra inicial em uma declaração, a qual é quase sempre um verbo. No caso da SQL, estes verbos são:

- Insert
- Update
- Delete

Cada declaração SQL é um comando declarativo. As declarações individuais da SQL são declarativas, em oposição às imperativas, na qual descrevem o que o programa deveria realizar, em vez de descrever como ele deveria realizar. Muitas implementações de banco de dados SQL estendem suas capacidades SQL fornecendo linguagens imperativas, isto é, procedurais. Exemplos destas implementações são o PL/SQL, da Oracle, e o SQL PL, da DB2. As linguagens de manipulação de dados tendem a ter muitos tipos diferentes e capacidades entre distribuidores de banco de dados. Há um padrão estabelecido para a SQL pela ANSI, porém os distribuidores ainda fornecem suas próprias extensões ao padrão enquanto não implementam o padrão por completo. <sup>6</sup>

### INSERT

A instrução SQL INSERT INTO é usada para inserir novos registros em uma tabela.

É possível escrever a INSERT INTO declaração de duas maneiras:

1. Especifique os nomes das colunas e os valores a serem inseridos:

```
INSERT INTO table_name (column1, column2, column3, ...)
VALUES (value1, value2, value3, ...);
```

2. Se você estiver adicionando valores para todas as colunas da tabela, não precisará especificar os nomes das colunas na consulta SQL. No entanto, certifique-se de que a ordem dos valores esteja na mesma ordem das colunas na tabela.

A sintaxe seria a seguinte:

```
INSERT INTO table_name
VALUES (value1, value2, value3, ...);
```

---

<sup>6</sup> [https://pt.wikipedia.org/wiki/Linguagem\\_de\\_manipula%C3%A7%C3%A3o\\_de\\_dados](https://pt.wikipedia.org/wiki/Linguagem_de_manipula%C3%A7%C3%A3o_de_dados)

## DELETE

A instrução DELETE é usada para excluir registros existentes em uma tabela.

### Sintaxe:

```
DELETE FROM table_name WHERE condição;
```

**Nota:** Tenha cuidado ao excluir registros em uma tabela! Observe a WHERE cláusula na declaração DELETE. A WHERE cláusula especifica quais registros devem ser excluídos. Se você omitir a cláusula WHERE, todos os registros da tabela serão excluídos!

### Excluir todos os registros

É possível excluir todas as linhas de uma tabela sem excluir a tabela. Isso significa que a estrutura, os atributos e os índices da tabela estarão intactos:

### Sintaxe:

```
DELETE FROM table_name;
```

**Nota:** “No Mysql” caso utilize a declaração DELETE sem a cláusula WHERE, ocorrerá um erro por conta do modo de atualização segura, código de ERRO MySQL 1175, veja a seguir como funciona o `sql_safe_updates`

## SQL\_SAFE\_UPDATES


O MySQL ERROR code 1175 é acionado quando você tenta atualizar ou excluir dados de uma tabela sem usar uma WHERE cláusula.

O MySQL tem um modo de atualização seguro para evitar que os administradores emitam uma instrução UPDATE ou DELETE sem uma cláusula WHERE.

Você pode ver se o modo de atualização segura está habilitado em seu servidor MySQL verificando a variável global `sql_safe_updates`.

Verifique a variável global usando a SHOW VARIABLES instrução da seguinte forma:

```
13 • SHOW VARIABLES LIKE "sql_safe_updates";
14
```



Variable_name	Value
sql_safe_updates	ON

O exemplo acima mostra que `sql_safe_updates` é ON, então uma instrução UPDATE ou DELETE sem a cláusula WHERE causará o erro 1175.

Para corrigir o erro, você pode desabilitar o modo de atualização segura ou seguir a descrição do erro adicionando uma cláusula WHERE que usa uma coluna KEY.

Você pode usar a SET instrução para desabilitar a atualização segura conforme mostrado abaixo:

```
set sql_safe_updates = 0;
```

Agora você deve ser capaz de executar uma instrução UPDATE ou DELETE sem uma cláusula WHERE.<sup>7</sup>

Se você quiser ativar o modo de atualização segura novamente, você pode SET usar a variável global como mostrado abaixo:

```
set sql_safe_updates = 1;
```

## UPDATE

A instrução UPDATE é usada para modificar os registros existentes em uma tabela.

### Sintaxe

```
UPDATE table_name  
SET column1 = value1, column2 = value2, ...  
WHERE condição;
```

**Nota:** Tenha cuidado ao atualizar registros em uma tabela! Observe a cláusula WHERE na declaração UPDATE. A cláusula WHERE especifica quais registros devem ser atualizados. Se você omitir a cláusula WHERE, todos os registros da tabela serão atualizados!

## Linguagem de consulta de dados – DQL

Structured Query Language, ou Linguagem de Consulta Estruturada ou SQL, é a linguagem de pesquisa declarativa padrão para banco de dados relacional (base de dados relacional). Muitas das características originais do SQL foram inspiradas na álgebra relacional.

## SELECT

A instrução SELECT do MySQL é usada para selecionar dados de um banco de dados.

Os dados retornados são armazenados em uma tabela de resultados, chamada de conjunto de resultados.

### Sintaxe SELECT

```
SELECT column1, column2, ...  
FROM table_name;
```

Aqui, coluna1, coluna2, ... são os nomes dos campos da tabela da qual você deseja selecionar os dados. Se você deseja selecionar todos os campos disponíveis na tabela, use a seguinte sintaxe:

```
SELECT * FROM table_name;
```

---

<sup>7</sup> <https://sebastian.com/mysql-error-code-1175/>

## Blocos de linguagem de consulta estruturada (SQL)

### Stored Procedures

Quando desenvolvemos aplicações que acessam banco de dados (boa parte delas), é comum executarmos rotinas complexas de manipulação desses dados a partir da linguagem/ferramenta utilizada. Para isso, utilizamos várias instruções SQL em sequência para obter o resultado esperado. Dependendo da rotina a ser executada, isso pode requerer várias consultas e atualizações na base, o que acarreta um maior consumo de recursos pela aplicação.

No caso de aplicações web, isso se torna ainda mais visível, devido a maior quantidade de informações que precisam trafegar pela rede e de requisições ao servidor.

Uma stored procedure (procedimento armazenado) é um conjunto de instruções desenvolvidas em linguagem Sql que, quando armazenadas ou salvas, ficam dentro do servidor de forma pré-compilada. Esse procedimento armazenado no banco de dados (Servidor) pode ser chamado a qualquer momento, tanto pelo sistema gerenciador de banco de dados (SGBD) quanto por uma aplicação.

### Sintaxe

Delimiter \$\$

**CREATE PROCEDURE** nome\_procedimento ([parâmetros])

**BEGIN**

/\*CORPO DO PROCEDIMENTO\*/

**END \$\$**

Exemplo de procedure sem parâmetros:

Criando uma stored procedure para selecionar todos os fornecedores.

```
144 Delimiter $$
145 CREATE PROCEDURE spSelectFornecedor()
146 BEGIN
147 select * from tbfornecedor;
148 END $$
149
```

Executando a procedure

```
call spSelectFornecedor();
```

	Codigo	CNPJ	Nome	Telefone
▶	1	1245678937123	Revenda Chico Loco	11934567897
	2	1345678937123	José Faz Tudo S/A	11934567898
	3	1445678937123	Vadalto Entregas	11934567899
	4	1545678937123	Astrogildo das Estrelas	11934567800
	5	1645678937123	Amoroso e Doce	11934567801

Exemplo de procedure com parâmetros:

```
describe tbCidade;
```

```
delimiter $$
```

```
CREATE procedure spInsertFornecedor(vNome varchar(200),vCNPJ decimal(14,0),vTelefone decimal(11,0))
```

```
begin
```

```
insert into tbfornecedor (Nome,CNPJ,Telefone)
```

```
values(vNome,vCNPJ,vTelefone);
```

```
end $$
```

```
call spInsertFornecedor('Revenda Chico Loco', 1245678937123,11934567897);
```

## Funções de string do MySQL

Função	Descrição da Função
<a href="#"><u>ASCII</u></a>	Retorna o valor ASCII para o caractere específico
<a href="#"><u>CHAR_LENGTH</u></a>	Retorna o comprimento de uma string (em caracteres)
<a href="#"><u>CHARACTER_LENGTH</u></a>	Retorna o comprimento de uma string (em caracteres)
<a href="#"><u>CONCAT</u></a>	Adiciona duas ou mais expressões juntas
<a href="#"><u>CONCAT_WS</u></a>	Adiciona duas ou mais expressões junto com um separador
<a href="#"><u>FIELD</u></a>	Retorna a posição do índice de um valor em uma lista de valores
<a href="#"><u>FIND_IN_SET</u></a>	Retorna a posição de uma string dentro de uma lista de strings
<a href="#"><u>FORMAT</u></a>	Formata um número para um formato como "#,###,###.##", arredondado para um número especificado de casas decimais
<a href="#"><u>INSERT</u></a>	Insere uma string dentro de uma string na posição especificada e para um certo número de caracteres
<a href="#"><u>INSTR</u></a>	Retorna a posição da primeira ocorrência de uma string em outra string
<a href="#"><u>LCASE</u></a>	Converte uma string para minúscula
<a href="#"><u>LEFT</u></a>	Extraí um número de caracteres de uma string (começando da esquerda)
<a href="#"><u>LENGTH</u></a>	Retorna o comprimento de uma string (em bytes)
<a href="#"><u>LOCATE</u></a>	Retorna a posição da primeira ocorrência de uma substring em uma string
<a href="#"><u>LOWER</u></a>	Converte uma string para minúscula
<a href="#"><u>LPAD</u></a>	Preenche uma string com outra string, até um certo comprimento
<a href="#"><u>LTRIM</u></a>	Remove espaços à esquerda de uma string
<a href="#"><u>MID</u></a>	Extraí uma substring de uma string (começando em qualquer posição)
<a href="#"><u>POSITION</u></a>	Retorna a posição da primeira ocorrência de uma substring em uma string
<a href="#"><u>REPEAT</u></a>	Repete uma string quantas vezes forem especificadas
<a href="#"><u>REPLACE</u></a>	Substitui todas as ocorrências de uma substring dentro de uma string por uma nova substring
<a href="#"><u>REVERSE</u></a>	Inverte uma string e retorna o resultado
<a href="#"><u>RIGHT</u></a>	Extraí um número de caracteres de uma string (começando da direita)
<a href="#"><u>RPAD</u></a>	Preenche uma string com outra string, até um certo comprimento
<a href="#"><u>RTRIM</u></a>	Remove espaços à direita de uma string
<a href="#"><u>SPACE</u></a>	Retorna uma string do número especificado de caracteres de espaço
<a href="#"><u>STRCMP</u></a>	Compara duas strings
<a href="#"><u>SUBSTR</u></a>	Extraí uma substring de uma string (começando em qualquer posição)
<a href="#"><u>SUBSTRING</u></a>	Extraí uma substring de uma string (começando em qualquer posição)
<a href="#"><u>SUBSTRING_INDEX</u></a>	Retorna uma substring de uma string antes que ocorra um número especificado de delimitador
<a href="#"><u>TRIM</u></a>	Remove espaços à esquerda e à direita de uma string
<a href="#"><u>UCASE</u></a>	Converte uma string para maiúscula
<a href="#"><u>UPPER</u></a>	Converte uma string para maiúscula



## Funções Numéricas do MySQL

Função	Descrição da Função
<a href="#">ABS</a>	Retorna o valor absoluto de um número
<a href="#">ACOS</a>	Retorna o arco cosseno de um número
<a href="#">ASIN</a>	Retorna o arco seno de um número
<a href="#">ATAN</a>	Retorna o arco tangente de um ou dois números
<a href="#">ATAN2</a>	Retorna o arco tangente de dois números
<a href="#">AVG</a>	Retorna o valor médio de uma expressão
<a href="#">CEIL</a>	Retorna o menor valor inteiro que é $\geq$ para um número
<a href="#">CEILING</a>	Retorna o menor valor inteiro que é $\geq$ para um número
<a href="#">COS</a>	Retorna o cosseno de um número
<a href="#">COT</a>	Retorna a cotangente de um número
<a href="#">COUNT</a>	Retorna o número de registros retornados por uma consulta selecionada
<a href="#">DEGREES</a>	Converte um valor em radianos para graus
<a href="#">DIV</a>	Usado para divisão inteira
<a href="#">EXP</a>	Retorna e elevado à potência de um número especificado
<a href="#">FLOOR</a>	Retorna o maior valor inteiro que é $\leq$ para um número
<a href="#">GREATEST</a>	Retorna o maior valor da lista de argumentos
<a href="#">LEAST</a>	Retorna o menor valor da lista de argumentos
<a href="#">LN</a>	Retorna o logaritmo natural de um número
<a href="#">LOG</a>	Retorna o logaritmo natural de um número ou o logaritmo de um número para uma base especificada
<a href="#">LOG10</a>	Retorna o logaritmo natural de um número na base 10
<a href="#">LOG2</a>	Retorna o logaritmo natural de um número na base 2
<a href="#">MAX</a>	Retorna o valor máximo em um conjunto de valores
<a href="#">MIN</a>	Retorna o valor mínimo em um conjunto de valores
<a href="#">MOD</a>	Retorna o resto de um número dividido por outro número
<a href="#">PI</a>	Retorna o valor de PI
<a href="#">POW</a>	Retorna o valor de um número elevado à potência de outro número
<a href="#">POWER</a>	Retorna o valor de um número elevado à potência de outro número
<a href="#">RADIANS</a>	Converte um valor de grau em radianos
<a href="#">RAND</a>	Retorna um número aleatório
<a href="#">ROUND</a>	Arredonda um número para um número especificado de casas decimais
<a href="#">SIGN</a>	Retorna o sinal de um número
<a href="#">SIN</a>	Retorna o seno de um número
<a href="#">SQRT</a>	Retorna a raiz quadrada de um número
<a href="#">SUM</a>	Calcula a soma de um conjunto de valores
<a href="#">TAN</a>	Retorna a tangente de um número
<a href="#">TRUNCATE</a>	Trunca um número para o número especificado de casas decimais

## Funções de data do MySQL

Função	Descrição da Função
<a href="#"><u>ADDDATE</u></a>	Adiciona um intervalo de data/hora a uma data e retorna a data
<a href="#"><u>ADDTIME</u></a>	Adiciona um intervalo de tempo a uma hora/datahora e, em seguida, retorna a hora/datahora
<a href="#"><u>CURDATE</u></a>	Retorna a data atual
<a href="#"><u>CURRENT_DATE</u></a>	Retorna a data atual
<a href="#"><u>CURRENT_TIME</u></a>	Retorna a hora atual
<a href="#"><u>CURRENT_TIMESTAMP</u></a>	Retorna a data e hora atuais
<a href="#"><u>CURTIME</u></a>	Retorna a hora atual
<a href="#"><u>DATE</u></a>	Extraí a parte de data de uma expressão de data e hora
<a href="#"><u>DATEDIFF</u></a>	Retorna o número de dias entre dois valores de data
<a href="#"><u>DATE_ADD</u></a>	Adiciona um intervalo de data/hora a uma data e retorna a data
<a href="#"><u>DATE_FORMAT</u></a>	Formata uma data
<a href="#"><u>DATE_SUB</u></a>	Subtrai um intervalo de data/hora de uma data e retorna a data
<a href="#"><u>DAY</u></a>	Retorna o dia do mês para uma determinada data
<a href="#"><u>DAYNAME</u></a>	Retorna o nome do dia da semana para uma determinada data
<a href="#"><u>DAYOFMONTH</u></a>	Retorna o dia do mês para uma determinada data
<a href="#"><u>DAYOFWEEK</u></a>	Retorna o índice de dias da semana para uma determinada data
<a href="#"><u>DAYOFYEAR</u></a>	Retorna o dia do ano para uma determinada data
<a href="#"><u>EXTRACT</u></a>	Extraí uma parte de uma determinada data
<a href="#"><u>FROM_DAYS</u></a>	Retorna uma data de um valor numérico de data
<a href="#"><u> HOUR</u></a>	Retorna a parte da hora para uma determinada data
<a href="#"><u>LAST_DAY</u></a>	Extraí o último dia do mês de uma determinada data
<a href="#"><u>LOCALTIME</u></a>	Retorna a data e hora atuais
<a href="#"><u>LOCALTIMESTAMP</u></a>	Retorna a data e hora atuais
<a href="#"><u>MAKEDATE</u></a>	Cria e retorna uma data com base em um valor de ano e número de dias
<a href="#"><u>MAKETIME</u></a>	Cria e retorna uma hora com base em um valor de hora, minuto e segundo
<a href="#"><u>MICROSECOND</u></a>	Retorna a parte de microssegundo de uma hora/datahora
<a href="#"><u>MINUTE</u></a>	Retorna a parte minuto de uma hora/datahora
<a href="#"><u>MONTH</u></a>	Retorna a parte do mês para uma determinada data
<a href="#"><u>MONTHNAME</u></a>	Retorna o nome do mês de uma determinada data
<a href="#"><u>NOW</u></a>	Retorna a data e hora atuais
<a href="#"><u>PERIOD_ADD</u></a>	Adiciona um número especificado de meses a um período
<a href="#"><u>PERIOD_DIFF</u></a>	Retorna a diferença entre dois períodos
<a href="#"><u>QUARTER</u></a>	Retorna o trimestre do ano para um determinado valor de data
<a href="#"><u>SECOND</u></a>	Retorna a parte dos segundos de uma hora/datahora
<a href="#"><u>SEC_TO_TIME</u></a>	Retorna um valor de tempo com base nos segundos especificados
<a href="#"><u>STR_TO_DATE</u></a>	Retorna uma data baseada em uma string e um formato
<a href="#"><u>SUBDATE</u></a>	Subtrai um intervalo de data/hora de uma data e retorna a data
<a href="#"><u>SUBTIME</u></a>	Subtrai um intervalo de tempo de um datetime e, em seguida, retorna o time/datetime
<a href="#"><u>SYSDATE</u></a>	Retorna a data e hora atuais

<a href="#"><u>TIME</u></a>	Extrai a parte do tempo de uma determinada hora/datahora
<a href="#"><u>TIME_FORMAT</u></a>	Formata uma hora por um formato especificado
<a href="#"><u>TIME_TO_SEC</u></a>	Converte um valor de tempo em segundos
<a href="#"><u>TIMEDIFF</u></a>	Retorna a diferença entre duas expressões de hora/data e hora
<a href="#"><u>TIMESTAMP</u></a>	Retorna um valor de data e hora com base em um valor de data ou data e hora
<a href="#"><u>TO_DAYS</u></a>	Retorna o número de dias entre uma data e a data "0000-00-00"
<a href="#"><u>WEEK</u></a>	Retorna o número da semana para uma determinada data
<a href="#"><u>WEEKDAY</u></a>	Retorna o número do dia da semana para uma determinada data
<a href="#"><u>WEEKOFYEAR</u></a>	Retorna o número da semana para uma determinada data
<a href="#"><u>YEAR</u></a>	Retorna a parte do ano para uma determinada data
<a href="#"><u>YEARWEEK</u></a>	Retorna o ano e o número da semana para uma determinada data

## Funções avançadas do MySQL

<a href="#"><u>BIN</u></a>	Retorna uma representação binária de um número
<a href="#"><u>BINARY</u></a>	Converte um valor em uma string binária
<a href="#"><u>CASE</u></a>	Passa por condições e retorna um valor quando a primeira condição é atendida
<a href="#"><u>CAST</u></a>	Converte um valor (de qualquer tipo) em um tipo de dados especificado
<a href="#"><u>COALESCE</u></a>	Retorna o primeiro valor não nulo em uma lista
<a href="#"><u>CONNECTION_ID</u></a>	Retorna o ID de conexão exclusivo para a conexão atual
<a href="#"><u>CONV</u></a>	Converte um número de um sistema de base numérica para outro
<a href="#"><u>CONVERT</u></a>	Converte um valor no tipo de dados ou conjunto de caracteres especificado
<a href="#"><u>CURRENT_USER</u></a>	Retorna o nome do usuário e o nome do host para a conta MySQL que o servidor usou para autenticar o cliente atual
<a href="#"><u>DATABASE</u></a>	Retorna o nome do banco de dados atual
<a href="#"><u>IF</u></a>	Retorna um valor se uma condição for TRUE, ou outro valor se uma condição for FALSE
<a href="#"><u>IF NULL</u></a>	Retorna um valor especificado se a expressão for NULL, caso contrário, retorna a expressão
<a href="#"><u>IS NULL</u></a>	Retorna 1 ou 0 dependendo se uma expressão é NULL
<a href="#"><u>LAST_INSERT_ID</u></a>	Retorna o ID AUTO_INCREMENT da última linha que foi inserida ou atualizada em uma tabela
<a href="#"><u>NULLIF</u></a>	Compara duas expressões e retorna NULL se forem iguais. Caso contrário, a primeira expressão é retornada
<a href="#"><u>SESSION_USER</u></a>	Retorna o nome de usuário e o nome do host MySQL atual
<a href="#"><u>SYSTEM_USER</u></a>	Retorna o nome de usuário e o nome do host MySQL atual
<a href="#"><u>USER</u></a>	Retorna o nome de usuário e o nome do host MySQL atual
<a href="#"><u>VERSION</u></a>	Retorna a versão atual do banco de dados MySQL

## Variáveis e constantes

### Variáveis

Quando declaramos uma variável em um programa, estamos na verdade definindo e reservando um espaço na memória para armazenar o valor que aquela variável conterà em determinado tempo de execução do programa.

Suponha que você precise fazer um programa que solicite ao usuário dois números inteiros, some esses dois números e apresente o resultado da soma para o usuário, esses valores são armazenados em variáveis.

O MySQL reconhece diferentes tipos de variáveis. O primeiro tipo são as variáveis definidas pelo utilizador, identificadas, normalmente, por um símbolo @ utilizado como um prefixo. No MySQL, é possível aceder a variáveis definidas pelo utilizador sem as declarar ou inicializar previamente. Se o fizer, um valor NULL é atribuído à variável quando é inicializada. Por exemplo, se utilizar SELECT com uma variável sem lhe atribuir um valor, como neste caso:

```
SELECT @SomeVariable;
```

Para inicializar uma variável definida pelo utilizador, é necessário utilizar uma declaração SET ou SELECT. É possível inicializar muitas variáveis ao mesmo tempo, separando cada declaração de atribuição com uma vírgula:

```
SET @FirstVar=1, @SecondVar=2;
```

Uma vez atribuído um valor a uma variável, esta terá um tipo de acordo com o valor dado. Nos exemplos anteriores, @FirstVar e @SecondVar são do tipo int.

A duração de uma variável definida pelo utilizador dura enquanto a sessão estiver ativa, e é invisível para outras sessões. Uma vez encerrada a sessão, a variável desaparece.

Há 5 tipos de dados que pode atribuir a uma variável definida pelo utilizador:

- caracteres
- inteiro
- ponto flutuante
- NULL, que pode ser associado a qualquer tipo

Para atribuir um valor a uma variável, pode utilizar o símbolo = ou :=. As duas afirmações seguintes têm o mesmo efeito:

```
SET @MyIntVar = 1;
```

```
SET @MyIntVar := 1;
```

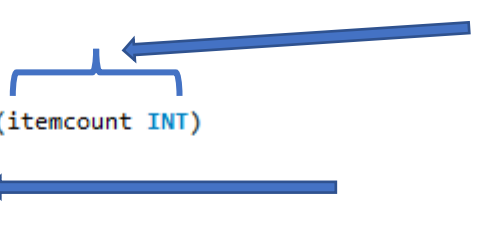
## Declare Variáveis Locais no MySQL

As variáveis locais não precisam do prefixo @ nos seus nomes, mas devem ser declaradas antes de poderem ser utilizadas. Para declarar uma variável local, pode utilizar a declaração DECLARE ou utilizá-la como parâmetro dentro de uma declaração STORED PROCEDURE.

Cada variável vive dentro de um âmbito, delimitado pelo bloco BEGIN ... END que contém a sua declaração.<sup>8</sup>

O exemplo seguinte ilustra duas formas diferentes de utilizar variáveis locais: como parâmetro de procedimento e como variável interna ao procedimento:

```
DELIMITER $$  
  
CREATE PROCEDURE GetUpdatedPrices(itemcount INT)  
BEGIN  
    DECLARE factor DECIMAL(5, 2);  
    SET factor:=3.45;  
    SELECT PartNo, Description, itemcount * factor * ListPrice FROM Catalogue;  
END  
$$  
  
DELIMITER ;
```



## Constantes

Uma constante é uma variável no sentido de que uma constante também reserva um espaço de memória para o tipo de dado que manipulará. Entretanto, uma constante armazenará um valor ÚNICO, um valor que NÃO mudará com o tempo de execução do programa.

O exemplo seguinte ilustra uma constante após a condição “Where”, o “2” é uma constante.

```
select * from tbAluno where AlunoID >= 2;
```

---

<sup>8</sup> <https://www.delftstack.com/pt/howto/mysql/mysql-declare-variable/>

## IF-THEN-ELSE

No MySQL, a instrução IF-THEN-ELSE é usada para executar código quando uma condição for TRUE, ou executar um código diferente se a condição for avaliada como FALSE.

### Sintaxe

A sintaxe para a instrução IF-THEN-ELSE no MySQL é:

```
IF condição 1 THEN
{ ... instruções a serem executadas quando condição 1 for TRUE...}
[ ELSEIF condição 2 THEN
{ ... instruções a serem executadas quando a condição 1 for FALSE e a condição 2 for TRUE...} ]
[ ELSE
{ ...instruções a serem executadas quando condição 1 e condição 2 forem FALSE...} ]
END IF;
```

### ELSEIF

Opcional. Você usaria a condição ELSEIF quando desejasse executar um conjunto de instruções quando uma segunda condição (ou seja: condição2) fosse TRUE.

### ELSE

Opcional. Você usaria a condição ELSE quando desejasse executar um conjunto de instruções quando nenhuma das condições IF ou ELSEIF fosse avaliada como TRUE.

### Observação

Assim que uma condição for considerada TRUE, a instrução IF-THEN-ELSE executará o código correspondente e não avaliará mais as condições.

Se nenhuma condição for atendida, a parte ELSE da instrução IF-THEN-ELSE será executada.

É importante observar que as partes ELSEIF e ELSE são opcionais. <sup>9</sup>

### Exemplo

O exemplo usando a instrução IF-THEN-ELSE em uma procedure no MySQL:

```
DELIMITER $$
CREATE procedure Nota( Nota INT )
BEGIN
declare vResultado varchar(250);
IF Nota >= 7 THEN
SET vResultado = 'Aprovado';
ELSEIF Nota >= 5 THEN
SET vResultado = 'Recuperação';
ELSE
SET vResultado = 'Reprovado';
END IF;
select vResultado as 'Situação';
END; $$
```

---

<sup>9</sup> [https://www.techonthenet.com/mysql/loops/if\\_then.php](https://www.techonthenet.com/mysql/loops/if_then.php)

## Os operadores AND, OR e NOT do MySQL

A WHERE cláusula pode ser combinada com os operadores AND, OR e NOT.

Os operadores AND e OR são usados para filtrar registros com base em mais de uma condição:

O AND operador exibe um registro se todas as condições separadas por AND forem TRUE.

O OR operador exibe um registro se alguma das condições separadas por OR for TRUE.

O NOT operador exibe um registro se a(s) condição(ões) NÃO for VERDADEIRA.

### Exemplo sintaxe AND

```
SELECT column1, column2, ...  
FROM table_name  
WHERE condition1 AND condition2 AND condition3 ...;
```

### Exemplo sintaxe OR

```
SELECT column1, column2, ...  
FROM table_name  
WHERE condition1 OR condition2 OR condition3 ...;
```

### Exemplo Sintaxe NOT

```
SELECT column1, column2, ...  
FROM table_name  
WHERE NOT condition;
```

## O operador MySQL EXISTS

O EXISTS operador é usado para testar a existência de qualquer registro em uma subconsulta, retornará TRUE se a subconsulta retornar um ou mais registros.

### Exemplo sintaxe EXISTS

```
SELECT column_name(s)
FROM table_name
WHERE EXISTS (SELECT column_name FROM table_name WHERE condition);
```

### Exemplo sintaxe EXISTS apagando uma base de dados caso ela exista

```
DROP DATABASE IF EXISTS dbDISTRIBUIDORA;
```

### Exemplo sintaxe EXISTS criando uma base de dados caso ela exista

```
CREATE DATABASE IF NOT EXISTS dbDISTRIBUIDORA;
```

## Order by

A ORDER BY palavra-chave é usada para classificar o conjunto de resultados em ordem crescente ou decrescente.

A palavra- ORDER BY chave classifica os registros em ordem crescente por padrão, será crescente (ASC). Para classificar os registros em ordem decrescente, use a palavra- DESC chave.

### Sintaxe Order By

```
SELECT column1, column2, ...
FROM table_name
ORDER BY column1, column2, ... ASC|DESC;
```

Podemos configurar a ordenação como ascendente (crescente) ou descendente (decrescente) com o uso das palavras ASC ou DESC:

**ASC** – Ordem ascendente

**DESC** – Ordem descendente (inversa)

Ordem de Classificação por Tipo de Dados



A ordem padrão de classificação do ORDER BY (ASC) para os diversos tipos de dados é a seguinte:

- Valores numéricos são exibidos com os menores valores primeiro (do menor para o maior).
- Valores de data são mostrados com os valores menores primeiro, o que significa as datas mais antigas (do mais antigo para o mais recente).
- Valores de caracteres são exibidos em ordem alfabética.
- Quando houver valores nulos (NULL), eles serão mostrados por último (em sequências descendentes, com DESC, são mostrados primeiro)

É possível classificar os dados da consulta usando uma coluna que não esteja presente na lista de colunas da declaração SELECT.

Também é possível ordenar os resultados da consulta usando mais de uma coluna. Para isso, basta listar as colunas na cláusula ORDER BY, separadas por vírgulas. Os resultados serão ordenados pela primeira coluna na lista, e então pela segunda coluna, e assim por diante. Pode-se ordenar qualquer das colunas listadas em qualquer ordem, por exemplo colunas em ordem ascendente e colunas em ordem descendente, bastando para isso suceder o nome da coluna que será classificada em ordem inversa com a palavra-chave DESC.

<http://www.bosontreinamentos.com.br/mysql/mysql-order-by-consultas-com-ordenacao-13/>

<https://www.devmedia.com.br/sql-order-by/41225>

<https://www.diegomacedo.com.br/como-ordenar-o-resultado-de-uma-consulta-no-mysql-order-by/>

## LIMIT

A cláusula LIMIT é usada para especificar o número de registros a serem retornados, é útil em tabelas grandes com milhares de registros, é usada para recuperar registros de uma ou mais tabelas no MySQL e limitar o número de registros retornados com base em um valor limite.

### Sintaxe LIMIT

```
SELECT column_name(s)  
FROM table_name  
WHERE condition  
LIMIT number;
```

Exemplo da sintaxe com [Where], [Order By] e [Limit]:

```
SELECT expressions  
FROM tables  
[WHERE conditions]  
[ORDER BY expression [ ASC | DESC ]]  
LIMIT row_count;
```

[https://www.w3schools.com/mysql/mysql\\_limit.asp](https://www.w3schools.com/mysql/mysql_limit.asp)

[https://www.techonthenet.com/mysql/select\\_limit.php](https://www.techonthenet.com/mysql/select_limit.php)

## Função MySQL STR\_TO\_DATE()

A função STR\_TO\_DATE() retorna uma data baseada em uma string e um formato.

### Sintaxe

STR\_TO\_DATE(string, format)

Valores de parâmetro

Parametro	Descrição																																																															
<i>string</i>	Sequência Necessário. A string a ser formatada para uma data																																																															
<i>formato</i>	Formato Obrigatório. O formato a ser usado. Pode ser um ou uma combinação dos seguintes valores:																																																															
	<table><tr><th>Formato</th><th>Descrição</th></tr><tr><td>%a</td><td>Nome abreviado do dia da semana (Dom a Sáb)</td></tr><tr><td>%b</td><td>Nome do mês abreviado (jan a dezembro)</td></tr><tr><td>%c</td><td>Nome numérico do mês (0 a 12)</td></tr><tr><td>%D</td><td>Dia do mês como valor numérico, seguido de sufixo (1º, 2º, 3º, ...)</td></tr><tr><td>%d</td><td>Dia do mês como valor numérico (01 a 31)</td></tr><tr><td>%e</td><td>Dia do mês como valor numérico (0 a 31)</td></tr><tr><td>%f</td><td>Microsegundos (000000 a 999999)</td></tr><tr><td>%H</td><td>Hora (00 a 23)</td></tr><tr><td>%h</td><td>Hora (00 a 12)</td></tr><tr><td>%I</td><td>Hora (00 a 12)</td></tr><tr><td>%i</td><td>Minutos (00 a 59)</td></tr><tr><td>%j</td><td>Dia do ano (001 a 366)</td></tr><tr><td>%k</td><td>Hora (0 a 23)</td></tr><tr><td>%l</td><td>hora (1 a 12)</td></tr><tr><td>%M</td><td>Nome do mês completo (janeiro a dezembro)</td></tr><tr><td>%m</td><td>Nome do mês como valor numérico (01 a 12)</td></tr><tr><td>%p</td><td>AM ou PM</td></tr><tr><td>%r</td><td>Hora no formato de 12 horas AM ou PM (hh:mm:ss AM/PM)</td></tr><tr><td>%S</td><td>Segundos (00 a 59)</td></tr><tr><td>%s</td><td>segundos (00 a 59)</td></tr><tr><td>%T</td><td>Hora no formato de 24 horas (hh:mm:ss)</td></tr><tr><td>%U</td><td>Semana onde domingo é o primeiro dia da semana (00 a 53)</td></tr><tr><td>%u</td><td>Semana onde segunda-feira é o primeiro dia da semana (00 a 53)</td></tr><tr><td>%V</td><td>Semana onde domingo é o primeiro dia da semana (01 a 53). Usado com %X</td></tr><tr><td>%v</td><td>Semana onde segunda-feira é o primeiro dia da semana (01 a 53). Usado com %X</td></tr><tr><td>%W</td><td>Nome completo do dia da semana (domingo a sábado)</td></tr><tr><td>%w</td><td>Dia da semana em que Domingo=0 e Sábado=6</td></tr><tr><td>%X</td><td>Ano para a semana em que domingo é o primeiro dia da semana. Usado com %V</td></tr><tr><td>%x</td><td>Ano para a semana em que segunda-feira é o primeiro dia da semana. Usado com %V</td></tr><tr><td>%Y</td><td>Ano como um valor numérico de 4 dígitos</td></tr><tr><td>%y</td><td>Ano como um valor numérico de 2 dígitos</td></tr></table>	Formato	Descrição	%a	Nome abreviado do dia da semana (Dom a Sáb)	%b	Nome do mês abreviado (jan a dezembro)	%c	Nome numérico do mês (0 a 12)	%D	Dia do mês como valor numérico, seguido de sufixo (1º, 2º, 3º, ...)	%d	Dia do mês como valor numérico (01 a 31)	%e	Dia do mês como valor numérico (0 a 31)	%f	Microsegundos (000000 a 999999)	%H	Hora (00 a 23)	%h	Hora (00 a 12)	%I	Hora (00 a 12)	%i	Minutos (00 a 59)	%j	Dia do ano (001 a 366)	%k	Hora (0 a 23)	%l	hora (1 a 12)	%M	Nome do mês completo (janeiro a dezembro)	%m	Nome do mês como valor numérico (01 a 12)	%p	AM ou PM	%r	Hora no formato de 12 horas AM ou PM (hh:mm:ss AM/PM)	%S	Segundos (00 a 59)	%s	segundos (00 a 59)	%T	Hora no formato de 24 horas (hh:mm:ss)	%U	Semana onde domingo é o primeiro dia da semana (00 a 53)	%u	Semana onde segunda-feira é o primeiro dia da semana (00 a 53)	%V	Semana onde domingo é o primeiro dia da semana (01 a 53). Usado com %X	%v	Semana onde segunda-feira é o primeiro dia da semana (01 a 53). Usado com %X	%W	Nome completo do dia da semana (domingo a sábado)	%w	Dia da semana em que Domingo=0 e Sábado=6	%X	Ano para a semana em que domingo é o primeiro dia da semana. Usado com %V	%x	Ano para a semana em que segunda-feira é o primeiro dia da semana. Usado com %V	%Y	Ano como um valor numérico de 4 dígitos	%y
Formato	Descrição																																																															
%a	Nome abreviado do dia da semana (Dom a Sáb)																																																															
%b	Nome do mês abreviado (jan a dezembro)																																																															
%c	Nome numérico do mês (0 a 12)																																																															
%D	Dia do mês como valor numérico, seguido de sufixo (1º, 2º, 3º, ...)																																																															
%d	Dia do mês como valor numérico (01 a 31)																																																															
%e	Dia do mês como valor numérico (0 a 31)																																																															
%f	Microsegundos (000000 a 999999)																																																															
%H	Hora (00 a 23)																																																															
%h	Hora (00 a 12)																																																															
%I	Hora (00 a 12)																																																															
%i	Minutos (00 a 59)																																																															
%j	Dia do ano (001 a 366)																																																															
%k	Hora (0 a 23)																																																															
%l	hora (1 a 12)																																																															
%M	Nome do mês completo (janeiro a dezembro)																																																															
%m	Nome do mês como valor numérico (01 a 12)																																																															
%p	AM ou PM																																																															
%r	Hora no formato de 12 horas AM ou PM (hh:mm:ss AM/PM)																																																															
%S	Segundos (00 a 59)																																																															
%s	segundos (00 a 59)																																																															
%T	Hora no formato de 24 horas (hh:mm:ss)																																																															
%U	Semana onde domingo é o primeiro dia da semana (00 a 53)																																																															
%u	Semana onde segunda-feira é o primeiro dia da semana (00 a 53)																																																															
%V	Semana onde domingo é o primeiro dia da semana (01 a 53). Usado com %X																																																															
%v	Semana onde segunda-feira é o primeiro dia da semana (01 a 53). Usado com %X																																																															
%W	Nome completo do dia da semana (domingo a sábado)																																																															
%w	Dia da semana em que Domingo=0 e Sábado=6																																																															
%X	Ano para a semana em que domingo é o primeiro dia da semana. Usado com %V																																																															
%x	Ano para a semana em que segunda-feira é o primeiro dia da semana. Usado com %V																																																															
%Y	Ano como um valor numérico de 4 dígitos																																																															
%y	Ano como um valor numérico de 2 dígitos																																																															

O STR\_TO\_DATE() converte a string em um valor de data com base na string de formato. A STR\_TO\_DATE() função pode retornar um DATE, TIME, ou DATETIME valor com base nas strings de entrada e formato. Se a string de entrada for ilegal, a STR\_TO\_DATE()função retorna NULL.

A STR\_TO\_DATE()função verifica a string de entrada para corresponder à string de formato. A string de formato pode conter caracteres literais e especificadores de formato que começam com o caractere de porcentagem (%).

A STR\_TO\_DATE() função é muito útil na migração de dados que envolve a conversão de dados temporais de um formato externo para o formato de dados temporais do MySQL.

## Exemplos do MySQL STR\_TO\_DATE

A instrução a seguir converte uma string em um DATE valor.

```
SELECT STR_TO_DATE('21,5,2013','%d,%m,%Y');
```

STR_TO_DATE('21,5,2013','%d,%m,%Y')
2013-05-21

O STR\_TO\_DATE() define todos os valores de data incompletos, que não são fornecidos pela string de entrada, como zero. Veja o seguinte exemplo:

```
SELECT STR_TO_DATE('2013','%Y');
```

STR_TO_DATE('2013','%Y')
2013-00-00

Como a string de entrada fornece apenas o valor do ano, a STR\_TO\_DATE() função retorna um valor de data que tem mês e dia definidos como zero.

O exemplo a seguir converte uma string de tempo em um TIME valor:

```
SELECT STR_TO_DATE('113005','%h%i%s');
```

STR_TO_DATE('113005','%h%i%s')
11:30:05

Semelhante à parte de data não especificada, a STR\_TO\_DATE() função define a parte de hora não especificada como zero, veja o exemplo a seguir:

```
SELECT STR_TO_DATE('11','%h');
```

STR_TO_DATE('11','%h')
11:00:00

O exemplo a seguir converte a string em um DATETIME valor porque a string de entrada fornece partes de data e hora.

```
SELECT STR_TO_DATE('20130101 1130','%Y%m%d %h%i');
```

STR_TO_DATE('20130101 1130','%Y%m%d %h%i')
2013-01-01 11:30:00

[https://www.w3schools.com/sql/func\\_mysql\\_str\\_to\\_date.asp](https://www.w3schools.com/sql/func_mysql_str_to_date.asp)

[https://www.mysqltutorial.org/mysql-str\\_to\\_date/](https://www.mysqltutorial.org/mysql-str_to_date/)

<https://dev.mysql.com/doc/refman/8.0/en/date-and-time-functions.html>

## Triggers em MySQL

O que é um Trigger SQL?

Um trigger (“gatilho”) é um objeto programável do banco de dados associado a uma tabela. Trata-se de um procedimento que é invocado automaticamente quando um comando DML é executado na tabela, sendo executado para cada linha afetada. Desta forma, as operações que podem disparar um trigger são:

- INSERT
- UPDATE
- DELETE

Geralmente, os triggers são empregados para verificar integridade dos dados, fazer validação dos dados e outras operações relacionadas.

### Diferença entre Trigger e Procedimento Armazenado

Tanto os triggers quanto as stored procedures são objetos programáveis de um banco de dados. Porém, eles possuem diferenças importantes entre si, que afetam o modo como são aplicados. Algumas das principais diferenças entre trigger e procedimentos armazenados são:

- Um Trigger é associado a uma tabela.
- Os triggers são armazenados no próprio banco de dados como arquivos separados.
- Triggers não são chamados diretamente, sendo invocados automaticamente, ao contrário dos procedimentos armazenados.
- Procedimentos armazenados podem trabalhar com parâmetros; não passamos parâmetros aos triggers.
- Os triggers não retornam um conjunto de resultados (resultset) ao cliente chamador.

### Aplicações dos triggers

As principais aplicações dos triggers em bancos de dados são:

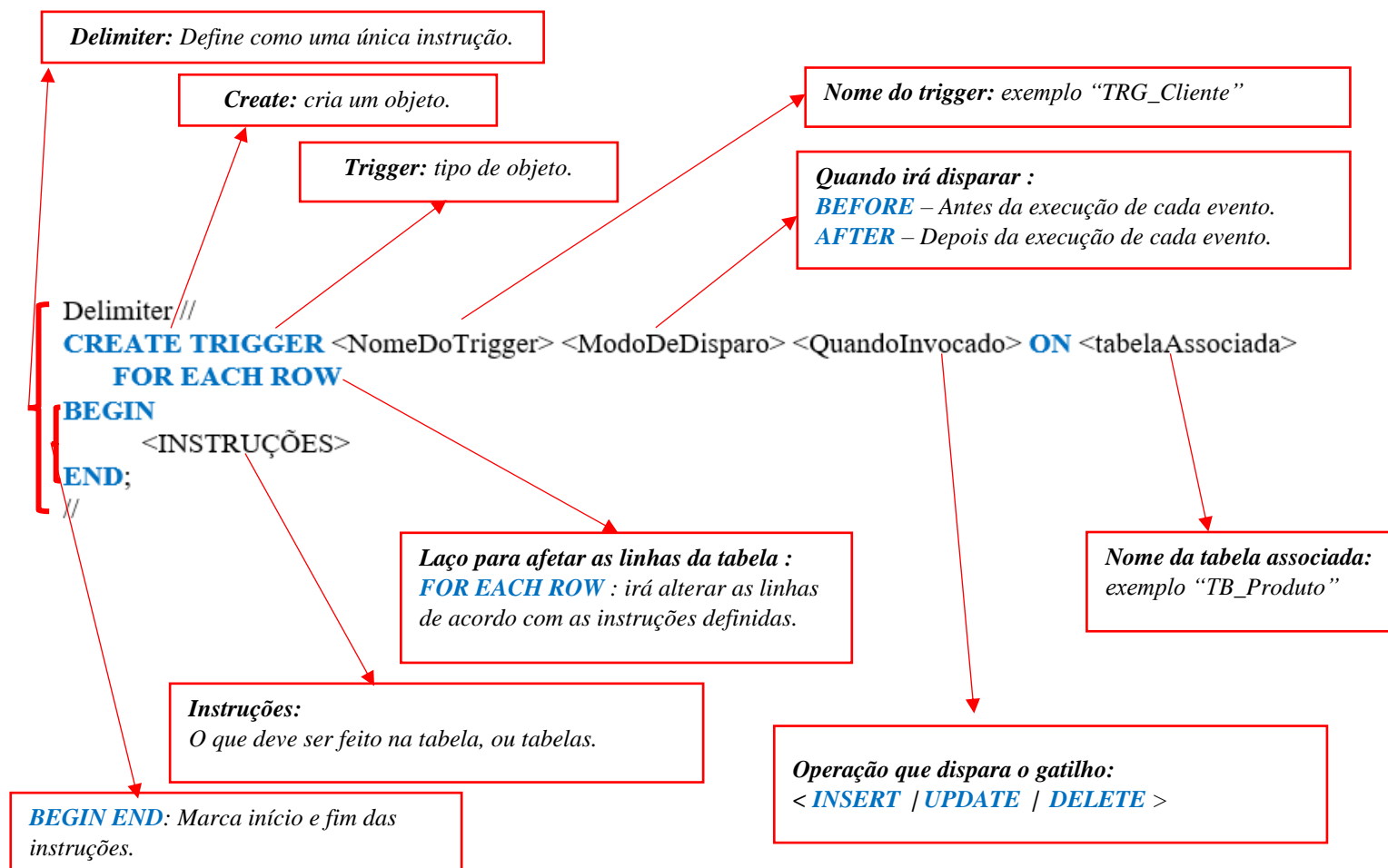
- Validação de Dados (tipos de dados, faixas de valores etc.).
- Rastreamento e registro de logs de atividades em tabelas.
- Verificação de integridade de dados e consistência
- Arquivamento de registros excluídos.

### Modos de Disparo de um Trigger

Um Trigger em MySQL pode ser disparado de dois modos diferentes:

- **BEFORE** – O trigger é disparado e seu código executado ANTES da execução de cada evento – por exemplo, antes de cada inserção de registros na tabela.
- **AFTER** – O código presente no trigger é executado após todas as ações terem sido completadas na tabela especificada.

## Sintaxe para criação de um trigger em MySQL



## NEW e .OLD

Para distinguir entre o valor das colunas BEFORE e AFTER o DML disparado, você usa os “pseudo registros” NEW e .OLD

Por exemplo, se você atualizar a descrição da coluna, no corpo do gatilho, poderá acessar o valor da descrição antes da atualização OLD.Coluna o novo valor NEW.Coluna.

A tabela a seguir ilustra a disponibilidade dos modificadores OLD e NEW

Evento acionador	OLD	NEW
<b>INSERT</b>	<b>Não</b>	<b>Sim</b>
<b>UPDATE</b>	<b>Sim</b>	<b>Sim</b>
<b>DELETE</b>	<b>Sim</b>	<b>Não</b>

## Tabelas Históricas

Cada tabela temporal de período do sistema está ligada a uma tabela de históricos. Quando uma linha é atualizada ou excluída de uma tabela temporal de período do sistema, o gerenciador de banco de dados insere uma cópia da linha antiga em sua tabela de históricos associada, com um registro de data e hora do tempo final atualizado. Este armazenamento de dados anteriores da tabela temporal de período do sistema fornece a capacidade de recuperar dados de momentos passados.

As colunas na tabela de históricos e tabela temporal de período do sistema devem ter os mesmos exatos nomes, ordem e tipos de dados. É possível criar uma tabela de históricos com os mesmos nomes e definições como as colunas da tabela temporal de período do sistema, usando a cláusula **LIKE** da instrução **CREATE TABLE**.<sup>10</sup>

### Exemplo:

```
CREATE TABLE <NomeDaNovaTabela> LIKE <TabelaBase>;
```

Uma tabela de históricos será sujeita às seguintes regras e restrições quando o versionamento estiver ativado:

- Uma tabela de históricos não pode ser explicitamente eliminada. Ela só pode ser implicitamente eliminada quando a tabela temporal do período do sistema associada for eliminada.
- As colunas da tabela de históricos não podem ser explicitamente incluídas, eliminadas ou mudadas.
- Uma tabela de históricos não pode ser definida como pai, filho ou de autorreferência em uma restrição de referência.
- Quando o versionamento é estabelecido, uma mudança para uma tabela temporal de período do sistema causa uma mudança correspondente implícita na tabela de históricos. Por exemplo, se uma tabela temporal de período do sistema é alterada para incluir uma coluna, a mesma coluna é incluída na tabela de históricos.
- Como excluir dados em uma tabela de históricos pode prejudicar sua habilidade de auditar o histórico de dados da tabela temporal de período do sistema, deve-se restringir o acesso a uma tabela de históricos para proteger seus dados.

---

<sup>10</sup> <https://www.ibm.com/docs/pt-br/i/7.3?topic=tables-history>

## Exemplos de gatilhos no MySQL

Vamos começar criando uma base de dados no MySQL, uma tabela, e inserir alguns registros:

```
Create database db_Exemplo;
use db_Exemplo;
Create table tb_Funcionario(
FuncId int primary key auto_increment,
FuncNome varchar(150) not null,
FuncEmail varchar(150) not null
);

insert into tb_Funcionario (FuncNome, FuncEmail)
values('José Mario','jose@escola.com'),
      ('Antonio Pedro','ant@escola.com'),
      ('Monica Cascão','moc@escola.com');

select * from tb_Funcionario;
```

Em seguida, crie a tabela temporária a partir da tabela tb\_Funcionario:

```
CREATE TABLE tb_FuncionarioHistorico LIKE tb_Funcionario;
```

Agora façamos as alterações necessárias na tabela tb\_FuncionarioHistorico:

```
describe tb_FuncionarioHistorico;
```

Field	Type	Null	Key	Default	Extra
FuncId	int	NO	PRI	NULL	auto_increment
FuncNome	varchar(150)	NO		NULL	
FuncEmail	varchar(150)	NO		NULL	

Para excluir a PK, você deve primeiro remover a propriedade de AUTO\_INCREMENT e depois remover a chave primária.

```
ALTER TABLE tb_FuncionarioHistorico MODIFY FuncId INT NOT NULL;
ALTER TABLE tb_FuncionarioHistorico DROP PRIMARY KEY;
```



Agora vamos incluir os campos para informar a data do evento e o evento realizado, vamos também incluir uma PK compostas por três campos.

```
ALTER TABLE tb_FuncionarioHistorico ADD Atualizacao datetime;
```

```
ALTER TABLE tb_FuncionarioHistorico ADD Situacao varchar(20);
```

```
ALTER TABLE tb_FuncionarioHistorico
```

```
ADD Constraint PK_Id_FuncHistorico primary key(FuncId,Atualizacao,Situacao);
```

Vejamos como ficou a estrutura da tabela:

```
describe tb_FuncionarioHistorico;
```

Field	Type	Null	Key	Default	Extra
FuncId	int	NO	PRI	NULL	
FuncNome	varchar(150)	NO		NULL	
FuncEmail	varchar(150)	NO		NULL	
Atualizacao	datetime	NO	PRI	NULL	
Situacao	varchar(20)	NO	PRI	NULL	

Agora o trigger para armazenar na tabela histórico do registro novo.

```
Delimiter //
```

```
CREATE TRIGGER TRG_FuncHistoricoInsert AFTER INSERT ON tb_Funcionario  
FOR EACH ROW
```

```
BEGIN
```

```
INSERT INTO tb_FuncionarioHistorico
```

```
SET FuncId = NEW.FuncId,
```

```
FuncNome = NEW.FuncNome,
```

```
FuncEmail = NEW.FuncEmail,
```

```
Atualizacao = current_timestamp(),
```

```
Situacao = "Novo";
```

```
END;
```

```
//
```

**NEW** – Pega o novo valor NEW.Coluna, o valor que está sendo inserido.

Para verificar o funcionamento o trigger, vamos inserir um novo funcionário.

```
insert into tb_Funcionario (FuncNome, FuncEmail)
values('Will Jr','willj@escola.com');
```

```
select * from tb_Funcionario;
```

FuncId	FuncNome	FuncEmail
1	José Mario	jose@escola.com
2	Antonio Pedro	ant@escola.com
3	Monica Cascão	moc@escola.com
4	Will Jr	willj@escola.com

*Registro inserido*

```
select * from tb_FuncionarioHistorico;
```

FuncId	FuncNome	FuncEmail	Atualizacao	Situacao
4	Will Jr	willj@escola.com	2022-09-02 14:38:35	Novo

*O registro também foi inserido na tabela histórico! O gatilho sempre irá acionar sempre que a tabela funcionária sofrer um insert!*

Agora o trigger para armazenar na tabela histórico do registro excluído.

```
Delimiter //
```

```
CREATE TRIGGER TRG_FuncHistoricoDelete BEFORE DELETE ON tb_Funcionario
FOR EACH ROW
```

```
3 BEGIN
```

```
    INSERT INTO tb_FuncionarioHistorico
    SET      FuncId = Old.FuncId,
            FuncNome = Old.FuncNome,
            FuncEmail = Old.FuncEmail,
            Atualizacao = current_timestamp(),
            Situacao = "Excluído";
```

```
4 END;
```

```
//
```

Para verificar o funcionamento o trigger, vamos excluir o registro de um funcionário.

```
delete from tb_Funcionario where FuncId = 3;
```

```
select * from tb_Funcionario;
```

FuncId	FuncNome	FuncEmail
1	José Mario	jose@escola.com
2	Antonio Pedro	ant@escola.com
4	Will Jr	willj@escola.com

*Registro de número 3 excluído!*

```
select * from tb_FuncionarioHistorico;
```

FuncId	FuncNome	FuncEmail	Atualizacao	Situacao
3	Monica Cascão	moc@escola.com	2022-09-02 16:59:51	Excluído
4	Will Jr	willj@escola.com	2022-09-02 14:38:35	Novo

*O registro foi inserido na tabela histórico! O gatilho sempre irá acionar sempre que a tabela funcionária sofrer um delete!*

Veja o trigger para armazenar na tabela histórico o registro que foi alterado.

Delimiter //

```
CREATE TRIGGER TRG_FuncHistoricoUpdate AFTER UPDATE ON tb_Funcionario
FOR EACH ROW
BEGIN
    INSERT INTO tb_FuncionarioHistorico
        SET    FuncId = NEW.FuncId,
              FuncNome = New.FuncNome,
              FuncEmail = New.FuncEmail,
              Atualizacao = current_timestamp(),
              Situacao = "Depois";
    INSERT INTO tb_FuncionarioHistorico
        SET    FuncId = Old.FuncId,
              FuncNome = Old.FuncNome,
              FuncEmail = Old.FuncEmail,
              Atualizacao = current_timestamp(),
              Situacao = "Antes";
END;
//
```

Para verificar o funcionamento o trigger, vamos alterar o registro do funcionário de código 1.

```
update tb_Funcionario set FuncNome = 'Maria', FuncEmail = 'maria@teste.com' where FuncId =1;
```

```
select * from tb_Funcionario;
```

FuncId	FuncNome	FuncEmail
1	Maria	maria@teste.com
2	Antonio Pedro	ant@escola.com
3	Monica Cascão	moc@escola.com

*Registro alterado código 1!*

```
select * from tb_FuncionarioHistorico;
```

FuncId	FuncNome	FuncEmail	Atualizacao	Situacao
1	José Mario	jose@escola.com	2022-09-02 20:30:57	Antes
1	Maria	maria@teste.com	2022-09-02 20:30:57	Depois
3	Monica Cascão	moc@escola.com	2022-09-02 20:30:38	Excluído
4	Will Jr	willj@escola.com	2022-09-02 20:30:27	Novo

*O registro que foi alterado foi inserido na tabela histórico, como esta antes e como ficou depois da alteração!*

## Prós e Contras das Triggers

Os principais pontos positivos sobre os triggers são:

- Parte do processamento que seria executado na aplicação passa para o banco, poupando recursos da máquina cliente.
- Facilita a manutenção, sem que seja necessário alterar o código fonte da aplicação.

Já contra sua utilização existem as seguintes considerações:

- Alguém que tenha acesso não autorizado ao banco de dados poderá visualizar e alterar o processamento realizado pelos gatilhos.
- Requer maior conhecimento de manipulação do banco de dados (SQL) para realizar as operações internamente.<sup>11</sup>

<sup>11</sup> <https://www.devmedia.com.br/mysql-basico-triggers/37462>  
<https://www.mysqltutorial.org/create-the-first-trigger-in-mysql.aspx>



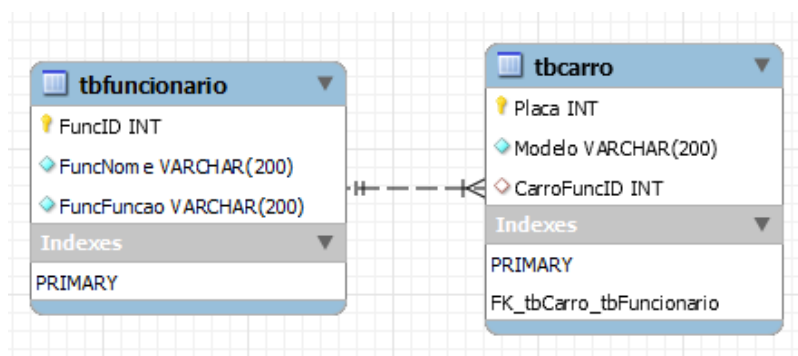
## JOIN

Uma cláusula JOIN em SQL, correspondente a uma operação de junção em álgebra relacional, combina colunas de uma ou mais tabelas em um banco de dados relacional. Ela cria um conjunto que pode ser salvo como uma tabela ou usado da forma como está.

Um JOIN é um meio de combinar colunas de uma (auto-junção) ou mais tabelas, usando valores comuns a cada uma delas. O SQL padrão ANSI especifica cinco tipos de JOIN: INNER JOIN, LEFT JOIN, RIGHT JOIN, FULL JOIN e CROSS JOIN. Sendo que o FULL JOIN não é padrão SQL ANSI.

### Banco exemplo

Para exemplificar os comandos como os JOIN vamos utilizar a base de dados dbExemplo a seguir:



Veja os dados nas tabelas de funcionários e os carros.

FuncID	FuncNome	FuncFuncao
1	Roberta	Diretor Executivo (CEO)
2	João	Diretor de Operações (COO)
3	Maria	Diretor Financeiro (CFO)
4	Antonio	Diretor de marketing (CMO)
5	Carla	Diretor de TI (CIO)
6	Daniel	Diretor de Receita (CRO)

Placa	Modelo	CarroFuncID
12345	AUDI - A5 SPORTBACK	4
12346	AUDI - Q5 PRESTIGE	3
12347	JAGUAR - E-PACE P250 R-DYNAMIC S	2
12348	MERCEDES-BENZ - GLA 250 SPORT	2
12349	VOLKSWAGEN - TAOS 250 TSI	1
12350	VOLVO - XC40 T5	5
12351	BMW Série 3	NULL
12352	BMW X1	NULL
12353	Mercedes-Benz GLB	NULL
12354	Volvo XC60	NULL
12355	Volvo XC40	NULL
12356	Audi Q5	NULL
12357	Mercedes-Benz Classe C	NULL
12358	Land Rover Discovery	NULL

### INNER JOIN

A cláusula INNER JOIN compara cada linha da tabela **A** com as linhas da tabela **B** para encontrar todos os pares de linhas que satisfazem a condição de junção. Se a condição de junção for avaliada como TRUE, os valores da coluna das linhas correspondentes das tabelas A e B serão combinados em uma nova linha e incluídos no conjunto de resultados.

## Sintaxe:

**SELECT** <campos>

**FROM** <Tabela A>

**INNER JOIN** <Tabela B>

**ON** <A.Key> = <B.Key> [;]

Veja exemplo com o INNER JOIN com os dados nas tabelas de funcionários e carros.

FuncID	FuncNome	FuncFuncao	Placa	Modelo	CarroFuncID
1	Roberta	Diretor Executivo (CEO)	12349	VOLKSWAGEN - TAOS 250 TSI	1
2	João	Diretor de Operações (COO)	12347	JAGUAR - E-PACE P250 R-DYNAMIC S	2
2	João	Diretor de Operações (COO)	12348	MERCEDES-BENZ - GLA 250 SPORT	2
3	Maria	Diretor Financeiro (CFO)	12346	AUDI - Q5 PRESTIGE	3
4	Antonio	Diretor de marketing (CMO)	12345	AUDI - A5 SPORTBACK	4
5	Carla	Diretor de TI (CIO)	12350	VOLVO - XC40 T5	5
6	Daniel	Diretor de Receita (CRO)			

**SELECT** \*

**FROM** tbFuncionario

**INNER JOIN** tbCarro

**ON** tbFuncionario.FuncID = tbCarro.CarroFuncID;

FuncID	FuncNome	FuncFuncao	Placa	Modelo	CarroFuncID
1	Roberta	Diretor Executivo (CEO)	12349	VOLKSWAGEN - TAOS 250 TSI	1
2	João	Diretor de Operações (COO)	12347	JAGUAR - E-PACE P250 R-DYNAMIC S	2
2	João	Diretor de Operações (COO)	12348	MERCEDES-BENZ - GLA 250 SPORT	2
3	Maria	Diretor Financeiro (CFO)	12346	AUDI - Q5 PRESTIGE	3
4	Antonio	Diretor de marketing (CMO)	12345	AUDI - A5 SPORTBACK	4
5	Carla	Diretor de TI (CIO)	12350	VOLVO - XC40 T5	5

Observe que o funcionário Daniel não aparece na recuperação de dados, porque ele não tem um correspondente na tabela carro! Existem registro de carros que também não aparecem por não ter um correspondente na tabela de funcionários.

## LEFT JOIN

Retorna todos os registros da tabela esquerda e os registros correspondentes da tabela direita.

Para cada linha da tabela A, a consulta a compara com todas as linhas da tabela B. Se um par de linhas fizerem com que a condição de junção seja avaliada como TRUE, os valores da coluna dessas linhas serão combinados para formar uma nova linha que será incluída no conjunto de resultados.

Se uma linha da tabela “esquerda” A não tiver nenhuma linha correspondente da tabela “direita” B, a consulta irá combinar os valores da coluna da linha da tabela “esquerda” A com NULL para cada valor da coluna da tabela da “direita” B que não satisfaça a condição de junto (FALSE).

Em resumo, a cláusula LEFT JOIN retorna todas as linhas da tabela “esquerda” A e as linhas correspondentes ou valores NULL da tabela “direita” B.

### Sintaxe:

**SELECT** <campos>

**FROM** <Tabela A>

**LEFT JOIN** <Tabela B>

**ON** <A.Key> = <B.Key> [;]

Veja exemplo com o LEFT JOIN com os dados nas tabelas de funcionários e carros.

FuncID	FuncNome	FuncFuncao
1	Roberta	Diretor Executivo (CEO)
2	João	Diretor de Operações (COO)
3	Maria	Diretor Financeiro (CFO)
4	Antonio	Diretor de marketing (CMO)
5	Carla	Diretor de TI (CIO)
6	Daniel	Diretor de Receita (CRO)

Placa	Modelo	CarroFuncID
12345	AUDI - A5 SPORTBACK	4
12346	AUDI - Q5 PRESTIGE	3
12347	JAGUAR - E-PACE P250 R-DYNAMIC S	2
12348	MERCEDES-BENZ - GLA 250 SPORT	2
12349	VOLKSWAGEN - TAOS 250 TSI	1
12350	VOLVO - XC40 T5	5
12351	BMW Série 3	NULL
12352	BMW X1	NULL
12353	Mercedes-Benz GLB	NULL
12354	Volvo XC60	NULL
12355	Volvo XC40	NULL
12356	Audi Q5	NULL
12357	Mercedes-Benz Classe C	NULL
12358	Land Rover Discovery	NULL

```
SELECT *  
FROM tbFuncionario  
LEFT JOIN tbCarro  
ON tbFuncionario.FuncID = tbCarro.CarroFuncID;
```

FuncID	FuncNome	FuncFuncao	Placa	Modelo	CarroFuncID
1	Roberta	Diretor Executivo (CEO)	12349	VOLKSWAGEN - TAOS 250 TSI	1
2	João	Diretor de Operações (COO)	12348	MERCEDES-BENZ - GLA 250 SPORT	2
2	João	Diretor de Operações (COO)	12347	JAGUAR - E-PACE P250 R-DYNAMIC S	2
3	Maria	Diretor Financeiro (CFO)	12346	AUDI - Q5 PRESTIGE	3
4	Antonio	Diretor de marketing (CMO)	12345	AUDI - A5 SPORTBACK	4
5	Carla	Diretor de TI (CIO)	12350	VOLVO - XC40 T5	5
6	Daniel	Diretor de Receita (CRO)	NULL	NULL	NULL



# RIGHT JOIN

Retorna todos os registros da tabela direita e os registros correspondentes da tabela esquerda.

A RIGHT JOIN combina dados de duas ou mais tabelas. A RIGHT JOIN começa a selecionar dados da tabela “direita” B e a corresponder às linhas da tabela “esquerda” A.

A RIGHT JOIN retorna um conjunto de resultados que inclui todas as linhas da tabela “direita” B, com ou sem linhas correspondentes na tabela “esquerda” A. Se uma linha na tabela direita B não tiver nenhuma linha correspondente da tabela “esquerda” A, a coluna da tabela “esquerda” A no conjunto de resultados será nula igualmente ao que acontece no LEFT JOIN.

## Sintaxe:

```
SELECT <campos>
FROM <Tabela A>
RIGHT JOIN <Tabela B>
ON <A.Key> = <B.Key> [;]
```

Veja exemplo com o RIGHT JOIN com os dados nas tabelas de funcionários e carros.

FuncID	FuncNome	FuncFuncao	Placa	Modelo	CarroFuncID
1	Roberta	Diretor Executivo (CEO)	12345	AUDI - A5 SPORTBACK	4
2	João	Diretor de Operações (COO)	12346	AUDI - Q5 PRESTIGE	3
3	Maria	Diretor Financeiro (CFO)	12347	JAGUAR - E-PACE P250 R-DYNAMIC S	2
4	Antonio	Diretor de marketing (CMO)	12348	MERCEDES-BENZ - GLA 250 SPORT	2
5	Carla	Diretor de TI (CIO)	12349	VOLKSWAGEN - TAOS 250 TSI	1
6	Daniel	Diretor de Receita (CRO)	12350	VOLVO - XC40 T5	5
			12351	BMW Série 3	NULL
			12352	BMW X1	NULL
			12353	Mercedes-Benz GLB	NULL
			12354	Volvo XC60	NULL
			12355	Volvo XC40	NULL
			12356	Audi Q5	NULL
			12357	Mercedes-Benz Classe C	NULL
			12358	Land Rover Discovery	NULL

```
SELECT *
FROM tbFuncionario
RIGHT JOIN tbCarro
ON tbFuncionario.FuncID = tbCarro.CarroFuncID;
```

FuncID	FuncNome	FuncFuncao	Placa	Modelo	CarroFuncID
4	Antonio	Diretor de marketing (CMO)	12345	AUDI - A5 SPORTBACK	4
3	Maria	Diretor Financeiro (CFO)	12346	AUDI - Q5 PRESTIGE	3
2	João	Diretor de Operações (COO)	12347	JAGUAR - E-PACE P250 R-DYNAMIC S	2
2	João	Diretor de Operações (COO)	12348	MERCEDES-BENZ - GLA 250 SPORT	2
1	Roberta	Diretor Executivo (CEO)	12349	VOLKSWAGEN - TAOS 250 TSI	1
5	Carla	Diretor de TI (CIO)	12350	VOLVO - XC40 T5	5
NULL	NULL	NULL	12351	BMW Série 3	NULL
NULL	NULL	NULL	12352	BMW X1	NULL
NULL	NULL	NULL	12353	Mercedes-Benz GLB	NULL
NULL	NULL	NULL	12354	Volvo XC60	NULL
NULL	NULL	NULL	12355	Volvo XC40	NULL
NULL	NULL	NULL	12356	Audi Q5	NULL
NULL	NULL	NULL	12357	Mercedes-Benz Classe C	NULL
NULL	NULL	NULL	12358	Land Rover Discovery	NULL

## FULL JOIN

O “Full Join”, “Full Outer Join” ou “Outer Join”, retorna todos os registros da tabela direita e todos os registros da tabela esquerda.

A sintaxe a seguir mostra como resultado todos os registros que estão na tabela A e todos os registros da tabela B, se uma linha na tabela direita B não tiver nenhuma linha correspondente da tabela A, a coluna da tabela A no conjunto de resultados será nula, e vice versa.

### Sintaxe:

**SELECT** <campos>

**FROM** <Tabela A>

**FULL JOIN** <Tabela B>

**ON** <A.Key> = <B.Key> [;]

Veja exemplo com o FULL JOIN com os dados nas tabelas de funcionários e carros do nosso banco dbExemplo.

```
SELECT * FROM tbcarro FULL JOIN tbfuncionario ON CarroFuncID = FuncID;
```

Placa	Modelo	CarroFuncID	FuncID	FuncNome	FuncFuncao
12349	VOLKSWAGEN - TAOS 250 TSI	1	1	Roberta	Diretor Executivo (CEO)
12347	JAGUAR - E-PACE P250 R-DYNAMIC S	2	2	João	Diretor de Operações (COO)
12348	MERCEDES-BENZ - GLA 250 SPORT	2	2	João	Diretor de Operações (COO)
12346	AUDI - Q5 PRESTIGE	3	3	Maria	Diretor Financeiro (CFO)
12345	AUDI - A5 SPORTBACK	4	4	Antonio	Diretor de marketing (CMO)
12350	VOLVO - XC40 T5	5	5	Carla	Diretor de TI (CIO)

Como podemos observar a seguir, no exemplo com o INNER JOIN o resultado é o mesmo! Assim podemos concluir que o MySql não suporta o FULL JOIN!

```
SELECT * FROM tbcarro INNER JOIN tbfuncionario ON CarroFuncID = FuncID;
```

Placa	Modelo	CarroFuncID	FuncID	FuncNome	FuncFuncao
12349	VOLKSWAGEN - TAOS 250 TSI	1	1	Roberta	Diretor Executivo (CEO)
12347	JAGUAR - E-PACE P250 R-DYNAMIC S	2	2	João	Diretor de Operações (COO)
12348	MERCEDES-BENZ - GLA 250 SPORT	2	2	João	Diretor de Operações (COO)
12346	AUDI - Q5 PRESTIGE	3	3	Maria	Diretor Financeiro (CFO)
12345	AUDI - A5 SPORTBACK	4	4	Antonio	Diretor de marketing (CMO)
12350	VOLVO - XC40 T5	5	5	Carla	Diretor de TI (CIO)

Neste caso, podemos utilizar outro comando que irá realizar a tarefa, este operador é o UNION que veremos a seguir.

## UNION

Assim como o JOIN, uma operação UNION permite combinar dados provenientes de duas ou mais tabelas (ou da mesma tabela, com condições diferentes). Porém, em vez de combinar as colunas dessas tabelas, a UNION combina as linhas de dois ou mais conjuntos de resultados. Imagine que um INNER JOIN é uma operação de intersecção entre conjuntos, e que o UNION é uma operação de soma de conjuntos.

Assim, uma UNION combina (“une”) duas ou mais declarações SELECT. O resultado de cada SELECT deve possuir o mesmo número de colunas, e o tipo de dado de cada coluna correspondente deve ser compatível.<sup>12</sup>

### Sintaxe:

**SELECT** declaração\_1

**UNION** [ALL]

**SELECT** declaração\_2

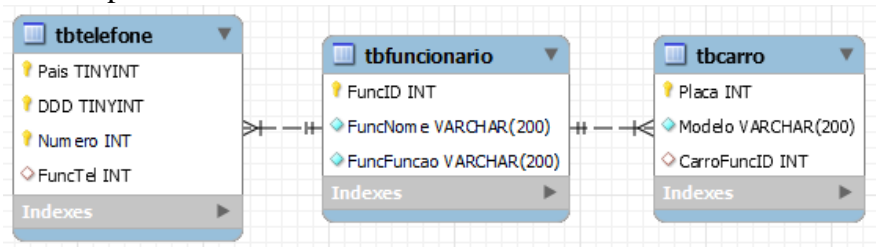
**UNION** [ALL]

**SELECT** declaração\_3 ...

[**ORDER BY** colunas]

### Exemplo:

Para exemplificar o operador UNION acrescentamos mais uma tabela no nosso banco exemplo:



Pais	DDD	Numero	FuncTel	FuncID	FuncNome	FuncFuncao	Placa	Modelo	CarroFuncID
55	11	911112222	1	1	Roberta	Diretor Executivo (CEO)	12345	AUDI - A5 SPORTBACK	4
55	11	911112223	1	2	João	Diretor de Operações (COO)	12346	AUDI - Q5 PRESTIGE	3
55	11	911112227	2	3	Maria	Diretor Financeiro (CFO)	12347	JAGUAR - E-PACE P250 R-DYNAMIC S	2
55	11	911112226	3	4	Antonio	Diretor de marketing (CMO)	12348	MERCEDES-BENZ - GLA 250 SPORT	2
55	11	911112225	4	5	Carla	Diretor de TI (CIO)	12349	VOLKSWAGEN - TAOS 250 TSI	1
55	11	911112224	5	6	Daniel	Diretor de Receita (CRO)	12350	VOLVO - XC40 T5	5
							12351	BMW Série 3	NULL
							12352	BMW X1	NULL
							12353	Mercedes-Benz GLB	NULL
							12354	Volvo XC60	NULL
							12355	Volvo XC40	NULL
							12356	Audi Q5	NULL
							12357	Mercedes-Benz Classe C	NULL
							12358	Land Rover Discovery	NULL

<sup>12</sup> <http://www.bosontreinamentos.com.br/mysql/operador-union-em-mysql-combinar-resultados-de-duas-ou-mais-consultas/>

## Exemplo:

Veja o exemplo com o operador union, uni as duas tabelas colocando uma embaixo da outra.

```
select * from tbfuncionario
```

union

```
select * from tbcarro;
```

FuncID	FuncNome	FuncFuncao
1	Roberta	Diretor Executivo (CEO)
2	João	Diretor de Operações (COO)
3	Maria	Diretor Financeiro (CFO)
4	Antonio	Diretor de marketing (CMO)
5	Carla	Diretor de TI (CIO)
6	Daniel	Diretor de Receita (CRO)
12345	AUDI - A5 SPORTBACK	4
12346	AUDI - Q5 PRESTIGE	3
12347	JAGUAR - E-PACE P250 R-DYNAMIC S	2
12348	MERCEDES-BENZ - GLA 250 SPORT	2
12349	VOLKSWAGEN - TAOS 250 TSI	1
12350	VOLVO - XC40 T5	5
12351	BMW Série 3	NULL
12352	BMW X1	NULL
12353	Mercedes-Benz GLB	NULL
12354	Volvo XC60	NULL
12355	Volvo XC40	NULL



Mas, se realizarmos o mesmo comando com a tabela de telefone, ocorreria um erro! Conforme falado anteriormente, as tabelas devem ter o mesmo número de colunas! Vimos no exemplo anterior que o union coloca uma tabela em cima da outra.

```
select * from tbfuncionario
```

union

```
select * from tbtelefone;
```

#	Time	Action	Message
1	16:32:39	select * from tbfuncionario union select * from tbtelefone	Error Code: 1222. The used SELECT statements have a different number of columns

Vejam os o mesmo código, mas desta vez com o mesmo número de colunas:

```
select * from tbfuncionario
```

union

```
select Pais,DDD,Numero from tbtelefone;
```

FuncID	FuncNome	FuncFuncao
1	Roberta	Diretor Executivo (CEO)
2	João	Diretor de Operações (COO)
3	Maria	Diretor Financeiro (CFO)
4	Antonio	Diretor de marketing (CMO)
5	Carla	Diretor de TI (CIO)
6	Daniel	Diretor de Receita (CRO)
55	11	911112222
55	11	911112223
55	11	911112227
55	11	911112226
55	11	911112225
55	11	911112224

Agora podemos ver como obtermos o resultado do “FULL JOIN” no MySql, para isso vamos utilizar o “UNION” vejamos o exemplo a seguir:

**SELECT \* FROM tbcarro LEFT JOIN tbfuncionario ON CarroFuncID = FuncID**

**UNION**

**SELECT \* FROM tbcarro RIGHT JOIN tbfuncionario ON CarroFuncID = FuncID;**

Placa	Modelo	CarroFuncID	FuncID	FuncNome	FuncFuncao
12345	AUDI - A5 SPORTBACK	4	4	Antonio	Diretor de marketing (CMO)
12346	AUDI - Q5 PRESTIGE	3	3	Maria	Diretor Financeiro (CFO)
12347	JAGUAR - E-PACE P250 R-DYNAMIC S	2	2	João	Diretor de Operações (COO)
12348	MERCEDES-BENZ - GLA 250 SPORT	2	2	João	Diretor de Operações (COO)
12349	VOLKSWAGEN - TAOS 250 TSI	1	1	Roberta	Diretor Executivo (CEO)
12350	VOLVO - XC40 T5	5	5	Carla	Diretor de TI (CIO)
12351	BMW Série 3	NULL	NULL	NULL	NULL
12352	BMW X1	NULL	NULL	NULL	NULL
12353	Mercedes-Benz GLB	NULL	NULL	NULL	NULL
12354	Volvo XC60	NULL	NULL	NULL	NULL
12355	Volvo XC40	NULL	NULL	NULL	NULL
12356	Audi Q5	NULL	NULL	NULL	NULL
12357	Mercedes-Benz Classe C	NULL	NULL	NULL	NULL
12358	Land Rover Discovery	NULL	NULL	NULL	NULL
NULL	NULL	NULL	6	Daniel	Diretor de Receita (CRO)

## CROSS JOIN

O “Cross Join”, junção cruzada é um tipo de junção que retorna o produto cartesiano de linhas das tabelas na junção. Em outras palavras, ele combina cada linha da primeira tabela com cada linha da segunda tabela. <sup>13</sup>

### Sintaxe:

**SELECT** <campos>

**FROM** <Tabela A>

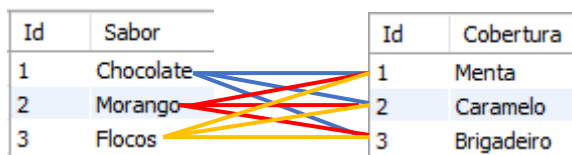
**CROSS JOIN** <Tabela B> [;]

Para o exemplo criamos duas tabelas no nosso banco dbExemplo.

<b>tbcobertura</b> Id INT Cobertura VARCHAR(50) Indexes	<b>tbsabor</b> Id INT Sabor VARCHAR(50) Indexes
--	--

<sup>13</sup> <https://learn.microsoft.com/en-us/power-query/cross-join>

Veja os dados nas tabelas de sabores e coberturas:



### Exemplo:

Exemplo com cross Join.

```
select * from tbSabor cross join tbCobertura;
```

Id	Sabor	Id	Cobertura
3	Flocos	1	Menta
2	Morango	1	Menta
1	Chocolate	1	Menta
3	Flocos	2	Caramelo
2	Morango	2	Caramelo
1	Chocolate	2	Caramelo
3	Flocos	3	Brigadeiro
2	Morango	3	Brigadeiro
1	Chocolate	3	Brigadeiro

## Exceções (tratamentos de erros)

Vamos começar dizendo que não existem “TRY CATCH” declarações tradicionais no MySQL, mas existem condições que são quase sinônimos de exceções. As condições contêm um código e uma descrição.

Em vez de envolver todo o seu código em um bloco try catch, você basicamente adiciona o que é chamado de manipulador ao seu procedimento.<sup>14</sup>

### Sintaxe:

**DECLARE** action **HANDLER FOR** condition\_value statemente;

Se uma condição cujo valor corresponder ao condition\_value “Condição\_valor”, o MySQL executará o statemente “instrução” continuará ou sairá do bloco de código atual com base no action “ação”.

- **action** aceita um dos seguintes valores:
  - CONTINUE: a execução do bloco de código anexo ( BEGIN... END) continua.
  - EXIT: a execução do bloco de código anexo, onde o manipulador é declarado, termina.
- **condition\_value** especifica uma condição específica ou uma classe de condições que ativam o manipulador. A condition\_value aceita um dos seguintes valores, um código de erro do MySQL.
  - SQLSTATE valor padrão, ou pode ser um SQLWARNING, NOTFOUND ou SQLEXCEPTION condição, que é uma abreviação para a classe de SQLSTATE valores.
  - NOTFOUND condição é usada para um cursor ou SELECT INTO variable\_listinstrução.

Uma condição nomeada associada a um código ou SQLSTATE valor de erro do MySQL.

- **statement** pode ser uma instrução simples ou uma instrução composta delimitada pelas palavras-chave BEGIN e END.<sup>15</sup>

### Exemplo:

Vamos criar uma procedure para inserir registro na tabela sabor:

```
delimiter $$
create procedure spInsertSabor (vSabor varchar(50))
begin
insert into tbSabor (Sabor) values (vSabor);
end $$
delimiter ;
```

---

<sup>14</sup>

<https://medium.com/startupward/how-to-try-and-catch-errors-in-mysql-stored-procedures-56604ffa17df>

<https://www.mysqltutorial.org/mysql-error-handling-in-stored-procedures/>

[https://www.youtube.com/watch?v=Fn6W7\\_EraHg](https://www.youtube.com/watch?v=Fn6W7_EraHg)

<sup>15</sup> <https://dev.mysql.com/doc/refman/8.0/en/declare-handler.html>

Para testar a procedure, vamos inserir o novo sabor:

```
call spInsertSabor ('Creme');  
select * from tbSabor;
```

Id	Sabor
1	Chocolate
4	Creme
3	Flocos
2	Morango

Se tentarmos inserir o mesmo sabor, acontecerá um erro:

```
call spInsertSabor ('Creme');
```

Output				
Action Output				
#	Time	Action	Message	
1	14:18:00	call spInsertSabor ('Creme')	Error Code: 1062. Duplicate entry 'Creme' for key 'tbsabor.Sabor'	

```
create table tbSabor(  
Id int auto_increment primary key  
Sabor varchar(50) not null unique  
);
```

O erro ocorreu porque o campo tem uma restrição "unique", não pode existir outro dado igual na mesma coluna.

Vamos então tratar este erro, primeiro apagando a procedure:

```
drop procedure spInsertSabor;
```

Agora inserimos o código para tratar o erro e criamos a procedure novamente:

```
delimiter $$  
create procedure spInsertSabor (vSabor varchar(50))  
begin  
DECLARE exit HANDLER FOR SQLEXCEPTION SELECT 'NÃO FOI POSSÍVEL REALIZAR O CADASTRO' AS MENSAGEM;  
insert into tbSabor (Sabor) values (vSabor);  
end $$  
delimiter ;
```



Veja que se tentarmos inserir um registro o qual já tenha na tabela, o erro não acontece, aparecerá a mensagem de erro programada.

```
call spInsertSabor ('Creme');
```

MENSAGEM	
▶	NÃO FOI POSSÍVEL REALIZAR O CADASTRO

Result 17 ×

Output

Action Output

#	Time	Action	Message
✓ 1	15:36:43	call spInsertSabor ('Creme')	1 row(s) returned

Agora vamos inserir um sabor que não está cadastrado:

```
call spInsertSabor ('MiLho-verde');
```

Output

Action Output

	#	Time	Action	Message
✓	1	11:27:07	call spInsertSabor ('MiLho-verde')	1 row(s) affected

Como podemos ver, se o sabor não existir ele é inserido, caso exista a mensagem é apresentada e não acontece o erro.

```
select * from tbSabor;
```

Id	Sabor
1	Chocolate
4	Creme
3	Flocos
5	Minho-verde
2	Morango

## Visões Controladas (Views)

Uma View é um objeto que pertence a um banco de dados, definida baseada em declarações SELECT's, retornando uma determinada visualização de dados de uma ou mais tabelas. Esses objetos são chamados por vezes de “virtual tables”, formada a partir de outras tabelas que por sua vez são chamadas de “base tables” ou ainda Views. E alguns casos, as Views são atualizáveis e podem ser alvos de declaração INSERT, UPDATE e DELETE, que na verdade modificam sua “base tables”.

Os benefícios da utilização de Views, além dos já salientados, são:

- Uma View pode ser utilizada, por exemplo, para retornar um valor baseado em um identificador de registro;
- Pode ser utilizada para promover restrições em dados para aumentar a segurança dos mesmos e definir políticas de acesso em nível de tabela e coluna.
- Podem ser configurados para mostrar colunas diferentes para diferentes usuários do banco de dados;
- Pode ser utilizada com um conjunto de tabelas que podem ser unidas a outros conjuntos de tabelas com a utilização de JOIN's ou UNION. <sup>16</sup>

### Sintaxe:

```
CREATE VIEW view_nome AS
```

```
SELECT ...
```

### Sintaxe exemplo:

```
CREATE VIEW view_nome AS
```

```
SELECT coluna1, coluna2, ...
```

```
FROM tabela_nome
```

```
WHERE condição
```

### Exemplo:

Para exemplificar fizemos uma alteração na tabela de funcionário adicionando uma coluna com o salário:

FuncID	FuncNome	FuncFuncao	Salario
1	Roberta	Diretor Executivo (CEO)	12000.00
2	João	Diretor de Operações (COO)	22000.00
3	Maria	Diretor Financeiro (CFO)	17000.00
4	Antonio	Diretor de marketing (CMO)	19000.00
5	Carla	Diretor de TI (CIO)	14000.00
6	Daniel	Diretor de Receita (CRO)	15500.00

---

<sup>16</sup> <https://dev.mysql.com/doc/refman/8.0/en/create-view.html>

<https://www.devmedia.com.br/mysql-trabalhando-com-views/8724>

Vamos imaginar que não é qualquer perfil de usuário que pode visualizar os salários dos funcionários. Para solucionar a questão, podemos criar visualizações “Views” e dar acesso somente as views que não vão constar o campo salário.

```
create view vwtbFuncionario as
select FuncID, FuncNome, FuncFuncao
from tbfuncionario;
```

Imaginando que o usuário só tem acesso as views, então quando quer ver os registros dos funcionários, ele só pode realizar uma consulta “Select” na View, e não na tabela.

```
select * from vwtbFuncionario;
```

Veja que a coluna salário não existe na view “tabela virtual”.

FuncID	FuncNome	FuncFuncao
1	Roberta	Diretor Executivo (CEO)
2	João	Diretor de Operações (COO)
3	Maria	Diretor Financeiro (CFO)
4	Antonio	Diretor de marketing (CMO)
5	Carla	Diretor de TI (CIO)
6	Daniel	Diretor de Receita (CRO)

Podemos observar a seguir que podemos trabalhar com qualquer coluna ou operador na consulta:

```
select FuncNome from vwtbFuncionario;
```

FuncNome
Roberta
João
Maria
Antonio
Carla
Daniel

Mas se tentar acessar a coluna salário, uma mensagem de erro é apresentada informando que a coluna não existe!

```
select Salario from vwtbFuncionario;
```

Output				
Action Output				
#	Time	Action	Message	
❌ 1	12:03:54	select Salario from vwtbFuncionario LIMIT 0, 500	Error Code: 1054. Unknown column 'Salario' in 'field list'	

Conforme dissemos anteriormente, uma view pode sofrer operações do tipo DML “INSERT/ UPDATE/ DELETE”, é claro que temos que atender as restrições de cada tabela. Vejamos um exemplo com uma atualização na vwtbFuncionario.

Primeiro vamos verificar a tbFuncionario:

```
select * from tbFuncionario;
```

FuncID	FuncNome	FuncFuncao	Salario
1	Roberta	Diretor Executivo (CEO)	12000.00
2	João	Diretor de Operações (COO)	22000.00
3	Maria	Diretor Financeiro (CFO)	17000.00
4	Antonio	Diretor de marketing (CMO)	19000.00
5	Carla	Diretor de TI (CIO)	14000.00
6	Daniel	Diretor de Receita (CRO)	15500.00

Agora vamos realizar a atualização na vwtbFuncionario:

```
update vwtbFuncionario set FuncNome = 'Maria Bonita' where FuncID = 3;
```

```
select * from vwtbFuncionario;
```

FuncID	FuncNome	FuncFuncao
1	Roberta	Diretor Executivo (CEO)
2	João	Diretor de Operações (COO)
3	Maria Bonita	Diretor Financeiro (CFO)
4	Antonio	Diretor de marketing (CMO)
5	Carla	Diretor de TI (CIO)
6	Daniel	Diretor de Receita (CRO)

```
select * from tbFuncionario;
```

FuncID	FuncNome	FuncFuncao	Salario
1	Roberta	Diretor Executivo (CEO)	12000.00
2	João	Diretor de Operações (COO)	22000.00
3	Maria Bonita	Diretor Financeiro (CFO)	17000.00
4	Antonio	Diretor de marketing (CMO)	19000.00
5	Carla	Diretor de TI (CIO)	14000.00
6	Daniel	Diretor de Receita (CRO)	15500.00

Uma visão pode ser atualizada com a create or replace view.<sup>17</sup>

### Sintaxe:

```
CREATE OR REPLACE VIEW view_nome AS
```

```
SELECT ...
```

---

<sup>17</sup> [https://www.w3schools.com/mysql/mysql\\_view.asp](https://www.w3schools.com/mysql/mysql_view.asp)

Para excluir uma View, basta utilizar o comando drop. <sup>18</sup>

**Sintaxe:**

**DROP VIEW** view\_nome [;]

---

<sup>18</sup> <https://www.devmedia.com.br/mysql-trabalhando-com-views/8724>

## PROXIMOS CAPITULOS

### 1. Merge e Permissões

<https://manifestotecnologico.wordpress.com/dba-banco-de-dados/mysql/tipos-de-tabelas/tabelas-merge/>

<https://www.dirceuresende.com/blog/sql-server-como-utilizar-o-comando-merge-para-inserir-atualizar-e-apagar-dados-com-apenas-1-comando/>

<https://imasters.com.br/banco-de-dados/vamos-combinar-insert-delete-e-update-em-um-unico-comando-conheca-o-comando-merge>