



# SISTEMAS DE BANCO DE DADOS 1

## AULA 6

### Linguagem SQL - Inicial MySQL



Vandor Roberto Vilardi Rissoli



# APRESENTAÇÃO

- Linguagem SQL
- Fundamentos SQL
- Instruções de Definição de Dados
- Instruções de Manutenção de Dados
- Referências



# Linguagem em Banco de Dados

De forma geral, os Banco de Dados Relacionais implementam o padrão SQL e acrescentam características específicas ao padrão.

Cada SGBD procura incluir elementos que o possam diferenciar dos concorrentes e implementam também algumas possibilidades procedurais da SQL, por exemplo, a *Transact SQL* para o **SQL SERVER** e o *PL/SQL* para a **ORACLE** entre outros SGBDs.

Este material apresentará a SQL tão portável quanto possível, mas as instruções aqui apresentadas poderão ser implementadas no SGBD **MySQL**.



# Structured Query Language - SQL

A SQL surgiu no início da década de 70, por uma **iniciativa da IBM**. Ela tornou-se a linguagem mais popular para acesso aos bancos de dados, juntamente com a difusão dos SGBDs Relacionais.

Vários “dialetos” surgiram rapidamente e a ANSI padronizou o SQL, surgindo em 1986 **SQL-86** e outros padrões que seguem 89, 92 e SQL-101...



# Structured Query Language - SQL

## CARACTERÍSTICAS DAS LINGUAGENS

A linguagem SQL é **declarativa** e os SGBDs Relacionais a têm como padrão. Suas características são originárias da **Álgebra Relacional**.

- Linguagem Declarativa (ou **Não** Procedural): Linguagem que descreve O QUE se deseja realizar, SEM se preocupar em dizer COMO será feito. O sistema é que deverá determinar a forma de como fazer (não se descreve ou conhece como o sistema realizará tal processamento);
- Linguagem Procedural: Linguagem que fornece uma descrição detalhada de COMO um processamento será realizado, operando sobre um registro ou uma unidade de dados de cada vez.



# Structured Query Language - SQL

A expressão **Programação Declarativa** é usada para discernir da programação praticada por meio das Linguagens de Programação Imperativas, sendo estas últimas sinônimos das **Linguagens Procedurais**.

Assim, em uma comparação simples, tem-se que:

*“um programa é declarativo se descreve o que ele faz e NÃO como seus procedimentos funcionam”.*

**ORACLE®**  
**PL/SQL**

procedural



não procedural



# Structured Query Language - SQL

## LINGUAGEM

Um sistema de banco de dados relacional, geralmente proporciona dois tipos principais de recursos em sua linguagem SQL:

- a) uma específica para manipular as estruturas do BD (DDL);
- b) outra para expressar consultas e atualizações sobre o que é armazenado nessas estruturas (DML).

- **DDL** - *Data Definition Language*;
- **DML** - *Data Manipulation Language*.



# Structured Query Language - SQL

- Linguagem de Definição de Dados (**DDL** - *Data Definition Language*) – uma estrutura de dados é representada por um conjunto de definições expressas por uma linguagem específica.
    - O resultado no uso da DDL constitui em um arquivo especial chamado de dicionário ou diretório de dados;
    - Um dicionário de dados é um arquivo de **metadados**.
- **METADADOS** são dados a respeito de dados. Em um sistema de Banco de Dados, esse arquivo ou diretório é consultado antes que o dado real seja modificado (manipulado).



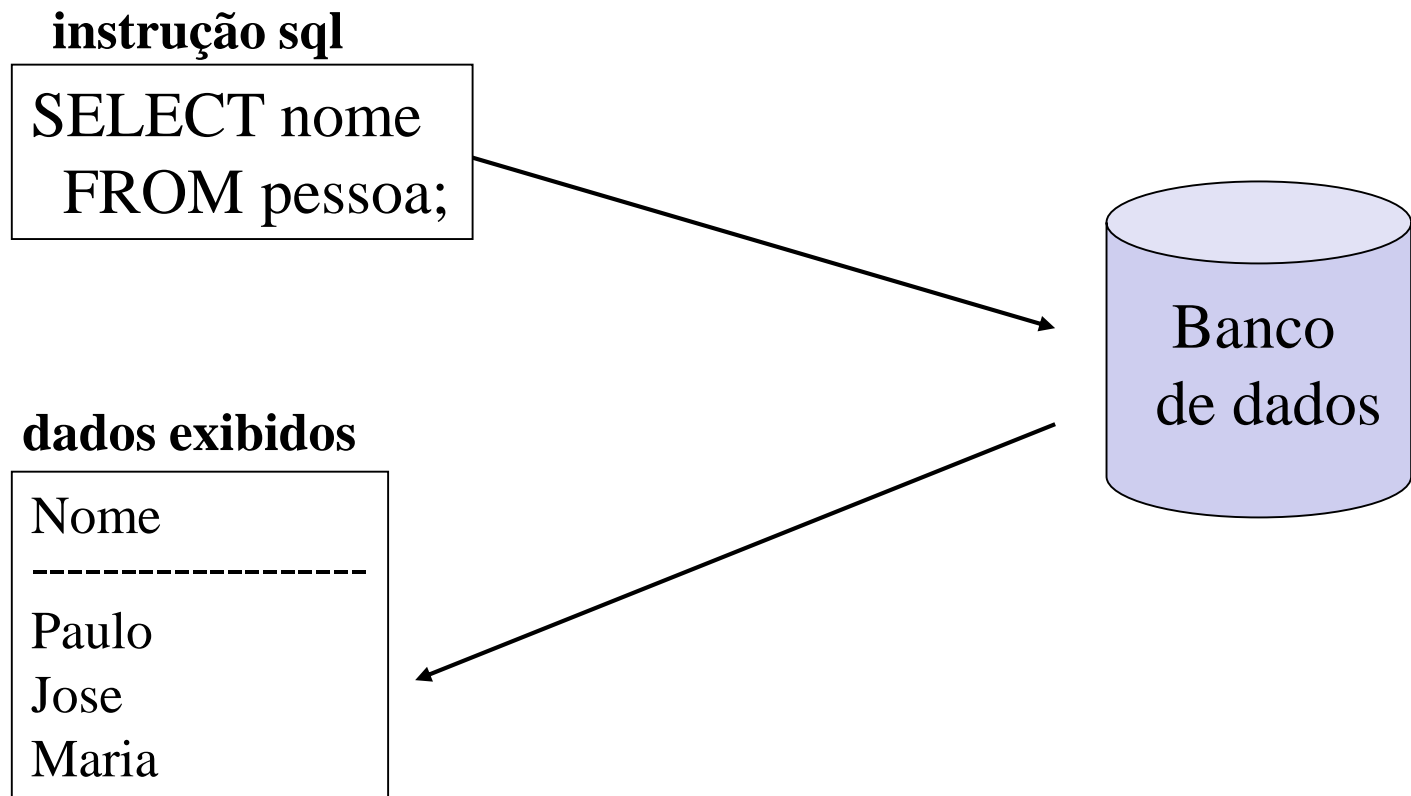


# Structured Query Language - SQL

- Linguagem de Manipulação dos Dados (DML - *Data Manipulation Language*) – é a linguagem que viabiliza o acesso ou a manipulação dos dados de forma compatível ao modelo de dados apropriado. Por manipulação de dados entende-se:
    - Recuperação dos dados armazenados no BD;
    - Inserção de novos dados no BD;
    - Remoção e modificação de dados do BD.
  - Linguagem de Consulta dos Dados (SQL - *Strutured Query Language*) – é parte de uma DML responsável pela recuperação de dados (pesquisa ou consulta).
- Apesar da SQL indicar uma linguagem de consulta, ela possui recursos para definição de estruturas de dados, de modificação de dados no BD e de especificação de segurança.

# Structured Query Language - SQL

A SQL consiste em uma linguagem não procedural (declarativa) que permite a interface básica para comunicação com o banco de dados.



# Structured Query Language - SQL

Um SGBD realiza alguns processos que podem ser efetuados por meio da linguagem SQL

- DEFINIÇÃO: criação descritiva do esquema(s) que atenderá as necessidades no BD;
- CONSTRUÇÃO: inserção das instâncias iniciais no banco de dados;
- MANIPULAÇÃO: realização de consultas e atualizações sobre os dados decorrentes do dia a dia que usa este BD.



# Structured Query Language - SQL

Será usada algumas ferramentas SQL para manipular o BD. As respectivas ferramentas são baseadas no SQL padrão, mas possuem algumas diferenças, por exemplo, recursos específicos da **ORACLE** que podem ser usados para se escrever relatórios e controlar o modo como a saída de tela ou papel é formatada difere da saída do **MySQL**.

Para começar a usar o BD é necessário estabelecer uma conexão com o SGBD que usa a SQL.

Nas características de um SGBD, lembre-se que existe uma preocupação com a restrição de acesso **NÃO autorizado**.



# Structured Query Language - SQL

## Criando uma Base de Dados

O estabelecimento de uma conexão com o BD **MySQL** só pode acontecer com o usuário que esteja cadastrado no BD.

Usando o BD **MySQL** será criada uma base de dados específica para a implementação do projeto. Essa base de dados receberá o nome de: *baseDados*.

**DATABASE** (base de dados): Conjunto de registros de dados dispostos em estrutura regular que possibilita o seu armazenamento organizado produzindo informação.

**CREATE DATABASE** <nome da base de dados>;

**DROP DATABASE** <nome da base de dados>;

**SHOW DATABASES;** -- lista bases de dados (só **MySQL**)

# Structured Query Language - SQL

## Estabelecendo a Conexão

Para se conectar ao BD **MySQL** disponível no computador local do laboratório use as opções do Sistema Operacional instalado para localizar o servidor **MySQL** disponível.

Procure a opção que indica o **MySQL** com acesso por linha de (*Command Prompt*) e a execute fornecendo os dados solicitados para um usuário já cadastrado nesse SGBD:

- confirme com seu professor qual o **usuário** disponível;
- solicite ao professor a **senha** individual desse usuário;
- pergunte ao professor qual será a base de dados (*database*) da atividade a ser realizada e execute a instrução abaixo:

use <nome da database>; (instrução só do MySQL)

use baseDados;

# Structured Query Language - SQL

Para iniciar a definição de um BD é necessário conhecer os seus tipos de dados válidos.

Os tipos usados na Descrição de Esquemas (DE) têm seus correspondentes em cada BD. Alguns do **MySQL** são:

- **int** – conjunto dos números inteiros (ver variações)
- **decimal** ( $n,d$ ) – conjunto dos números reais
- **char**( $n$ ) – de 1 até 255 caracteres
- **varchar**( $n$ ) – caracteres variáveis até 4000
- **date** – data com ano, mês e dia no padrão
- **time** – horário com hora, minutos e segundos
  - $n$  corresponde ao comprimento ou tamanho
  - $d$  quantidade de dígitos decimais depois da vír

Visite o sítio virtual abaixo e confira todos os tipos do **MySQL**

<https://dev.mysql.com/doc/refman/5.7/en/data-types.html>

# Structured Query Language - SQL

A primeira fase em qualquer BD começa com o uso de instruções **DDL** (*Linguagem de Definição de Dados*), pois serão por meio delas que se criam recursos no BD.

Exemplo:

→ Como seria a relação que armazenaria todas as unidades da federação brasileira (os estados)?

Supondo esta necessidade elabora-se o esquema coerente para armazenar e manipular essas Unidades da Federação.

Relação: ESTADOS  
sigla literal (2)  
nome literal (20)

Unidades da Federação	
sigla	nome
AC	Acre
AM	Amazonas
:	:
DF	Distrito Federal



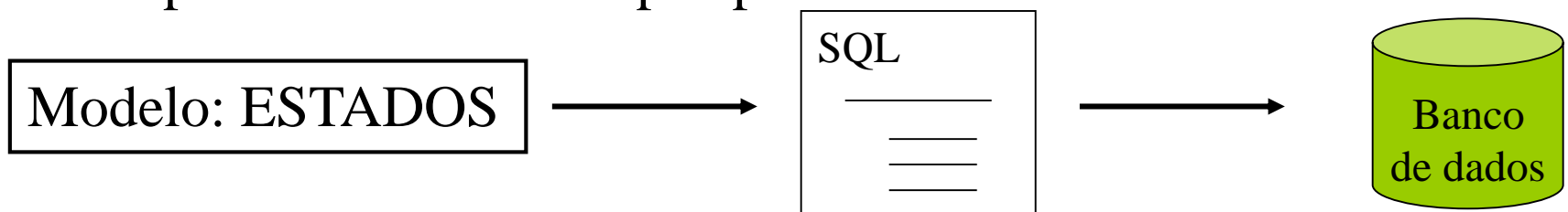
# Structured Query Language - SQL

Após modelar e confirmar qual a relação que atende as necessidades, uma tabela será criada no BD.

Para criar esta relação (tabela) será usada a declaração SQL (DDL) que efetiva essa tabela no BD.

```
CREATE TABLE <nome da tabela> (  
  nome do atributo_1 <tipo dado>, -- tipo do atributo_1  
  nome do atributo_2 <tipo dado> ) ;
```

- Na linha de comando do SQL os dois traços seguidos (--) indicam um comentário a partir deles até o final da linha;
- O ponto e vírgula encerra uma instrução deixando-a pronta para ser executada após pressionada a tecla <ENTER>.



# Structured Query Language - SQL

Todos os Estados possuem uma *sigla*, portanto sempre que se for cadastrar um novo estado ele deverá possuir uma *sigla*. Para que o SGBD controle esta restrição na tabela, não permitindo que seja informado um estado sem uma *sigla*, é necessário que o projeto da relação possua uma confirmação que a **obrigue**.

Caso nada identifique esta obrigatoriedade, será possível o armazenamento de um estado sem *sigla*.

A obrigatoriedade de um atributo é representada pela expressão **NOT NULL** (não nulo) que deve estar vinculada a todos os atributos que são obrigatórios em uma tabela.

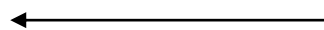


# Structured Query Language - SQL

A criação da tabela que representará as Unidades da Federação Brasileira será feita com a seguinte instrução:

```
CREATE TABLE ESTADOS (  
    sigla char(2) NOT NULL,  
    nome varchar(20) ); -- pressionando <ENTER>
```

Tabela criada.



resultado da operação

Por meio deste comando foi criada a tabela **ESTADOS** que possui como domínio todas as Unidades da Federação Brasileira.



# Structured Query Language - SQL

A expressão **NULL** significa que não existe valor, ou seja, não contém dados. O **erro** mais comum que se comete é a atribuição de um valor nulo em um atributo numérico.

O valor **nulo** é diferente de zero, pois zero é um valor, enquanto que nulo é a ausência de valor. Se ocorrer a operação  $1 + \text{NULL}$  o resultado vai ser NULL.

*“Nunca use um **NULL** para representar um valor igual a zero em um atributo numérico. Se este atributo for usado para operações aritméticas dê a ele o valor de zero e não **NULL**”*



# Structured Query Language - SQL

No **MySQL** a instrução *DESCRIBE* e *SHOW* são importantes. As duas instruções oferecem um resumo de dados sobre a tabela, também chamada de esquema, com suas colunas (atributos), tipos de dados e outras informações relevantes. O primeiro comando pode ser usado de maneira abreviada (**4 letras iniciais**), tendo o mesmo efeito.

```
mysql> desc ESTADOS;
```

Field	Type	Null	Key	Default	Extra
sigla	char(2)	No		NULL	
nome	varchar(20)	Yes		NULL	

2 rows in set (**0.01 sec**) => tempo de resposta do SGBD

# Structured Query Language - SQL

A instrução que modifica o nome de uma relação é:

**RENAME** <nome\_atual> **TO** <novo\_nome>;

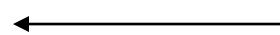
A instrução que remove uma relação (será detalhada a frente) consiste no comando a seguir:

**DROP** TABLE <nome\_da\_tabela>;

Exemplo:

**RENAME TABLE ESTADOS TO UNIDADES FEDERAIS;**

Tabela renomeada.



resultado

**DROP TABLE ESTADOS;**

**ERRO 1017: Arquivo não encontrado.**



resultado

Código  
do erro



Número do erro



# Structured Query Language - SQL

## Remover uma Tabela

Para se remover uma relação (tabela) do SGBD deve ser usada a instrução **DROP TABLE**.

DROP TABLE <nome da tabela>;

Por meio desta instrução a tabela deixará de existir no BD, sendo todos os dados guardados nessa tabela apagados. Similar a instrução **DROP DATABASE**, que apaga a base de dados com todas as tabelas e recursos existentes no SGBD.

Exemplo:

DROP TABLE UNIDADES FEDERAIS;

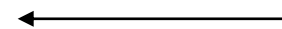
Tabela eliminada.



resultado

DROP TABLE CIDADES;

ERRO 1051: tabela desconhecida.



resultado

# Structured Query Language - SQL

É possível assim manipular relações (tabelas) com DML, após conhecer os procedimentos básicos para a criação e remoção das relações com comandos DDL.

Identificando o domínio das unidades da federação, pode-se armazenar seus dados, em que serão guardados os valores de todos os estados da Federação Brasileira.

Com isso se inicia o processo de construção do BD, pois a definição projetou o BD mais coerente para o armazenamento necessário dos dados do problema.

## Estados Brasileiros

SIGLA	NOME
AM	Amazonas
AP	Amapa
CE	Ceara
:	:





# Structured Query Language - SQL

## Instrução INSERT

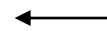
Esta instrução é usada para inserir tuplas em uma relação.

**INSERT INTO <tabela>[(coluna\_1,...,coluna\_n)] VALUES (valor\_1,...,valor\_n);**

Exemplo:

INSERT INTO ESTADOS(sigla, nome) VALUES('GO', 'Goiás');

**ERRO 1146: tabela não existe.**



resultado

A tabela desejada **não existe mais**, pois foi apagada pelo comando DROP. Portanto, crie a tabela **ESTADOS** novamente e depois insira os estados.

INSERT INTO ESTADOS(sigla, nome) VALUES('GO', 'Goiás');

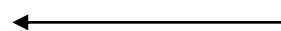
1 linha criada.



Resultado da 1ª forma

INSERT INTO ESTADOS(nome, sigla) VALUES('ACRE', 'AC');

1 linha criada.

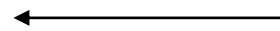


Resultado da 2ª forma

# Structured Query Language - SQL

```
INSERT INTO ESTADOS VALUES('SP', 'São Paulo');
```

1 linha criada.



Resultado da forma geral

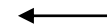
A instrução **INSERT** deve respeitar as características de cada tipo de dado para inserir dados na base de dados corretamente nas formas exigidas para cada tipo de dado em **SQL**.

Um exemplo relevante é a manipulação correta de caracter(es) ou *String* em **SQL** que deve ser definida usando **APÓSTROFE** e **NÃO ASPAS**. Usa-se **ASPAS** somente em situações muito específicas e bem definidas em **SQL**.

```
INSERT INTO ESTADOS VALUEX("SE", "Sergipe");
```

```
INSERT INTO ESTADOS VALUES('SE', 'Sergipe');
```

1 linha criada.



resultado

# Structured Query Language - SQL

## Instrução DELETE

Esta instrução é usada para remover uma ou mais tuplas da relação, possuindo duas formas básicas:

DELETE FROM <tabela>;

ou

DELETE FROM <tabela>

**WHERE** <condição>;

A primeira forma é obrigatória e apaga todos os dados da relação, enquanto que a segunda possui uma parte opcional, a partir do **WHERE**, que apaga somente os dados da tabela que atendem a uma condição (ou condições) imposta pela cláusula **WHERE**.

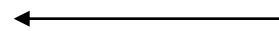


# Structured Query Language - SQL

## Exemplo:

DELETE FROM ESTADOS WHERE SIGLA = 'SP';

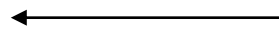
1 linha deletada.



resultado

DELETE FROM ESTADOS WHERE NOME = 'ACRE';

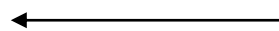
1 linha deletada.



resultado

DELETE FROM ESTADOS WHERE SIGLA = 'AC';

0 linhas deletadas.



resultado

INSERT INTO ESTADOS VALUES('DF', 'BRASÍLIA');

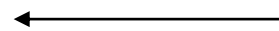
1 linha criada.



resultado

DELETE FROM ESTADOS;

3 linhas deletadas.



resultado



# Structured Query Language - SQL

## Instrução UPDATE

Esta instrução permite a atualização de um ou mais atributos em uma tupla, ou em um grupo de tuplas de uma relação (tabela). O conteúdo de cada atributo será ajustado com a cláusula **SET**. Como na instrução DELETE existem duas formas para UPDATE:

UPDATE <tabela> SET <atributo> = <valor>;

ou

UPDATE <tabela> SET <atributo> = <valor>

**WHERE** <condição>;

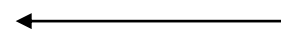
A primeira forma é obrigatória e atualiza todos os dados da relação, enquanto que a segunda possui uma parte **WHERE** opcional, que atualiza somente os dados da relação que atendem a uma condição (ou condições) imposta pela cláusula **WHERE**.

# Structured Query Language - SQL

## Exemplo:

```
UPDATE ESTADOS SET NOME = 'ACRE';
```

0 linhas atualizadas.

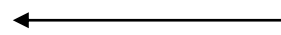


resultado

Com a remoção de todas as tuplas no último **DELETE**, não é possível alterar nenhum dado, pois não existem dados na tabela.

```
INSERT INTO ESTADOS VALUES('BA', 'BAIA');
```

1 linha criada.

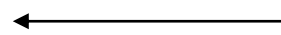


resultado

```
UPDATE ESTADOS SET NOME = 'Bahia'
```

```
WHERE SIGLA = 'BA';
```

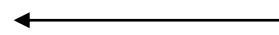
1 linha atualizada.



resultado

```
UPDATE ESTADOS SET NOME = 'ALAGOAS',  
SIGLA = 'AL' WHERE SIGLA = 'PA';
```

0 linhas atualizadas.



resultado

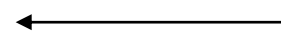


# Structured Query Language - SQL

## Exemplo:

```
INSERT INTO ESTADOS VALUES('PA', 'PARA');
```

1 linha criada.

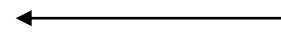


resultado

```
UPDATE ESTADOS SET NOME = 'ACRE'
```

```
WHERE SIGLA = 'PA';
```

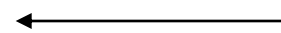
1 linha atualizada.



resultado

```
UPDATE ESTADOS SET NOME = 'ALAGOAS',  
SIGLA = 'AL' WHERE SIGLA = 'PA';
```

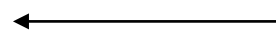
1 linha atualizada.



resultado

```
UPDATE CIDADES SET NOME = 'BRASÍLIA'
```

**ERRO 1146: tabela não existe.**



resultado



# Structured Query Language - SQL

## Instrução SELECT

Esta instrução é a essência do linguagem SQL. É por meio dela que se recupera (pesquisa) dados de um banco de dados. De modo simples, está se dizendo ao BD quais informações foram selecionadas para serem recuperadas.

Pode-se dividir esta instrução em quatro partes básicas:

- **SELECT** – seguido de atributos que se esperam ver (**obrigatório**)
- **FROM** – seguido da origem dos dados no BD (**obrigatório**)
- **WHERE** – seguido das restrições de recuperação (**opcional**)
- **ORDER BY** – seguido da forma como os dados serão ordenados (**opcional**)

Esta será a instrução mais comumente usada na linguagem SQL.



# Structured Query Language - SQL

O símbolo asterisco ( \* ) significa que todos atributos da relação informada deverão ser recuperados.

Exemplo:            SELECT \* FROM ESTADOS;

```
-----  
| sigla | nome      |  
-----  
| BA    | Bahia     |  
| AL    | ALAGOAS   |  
-----
```

← resultado

2 linhas apresentadas.

SELECT \* FROM CIDADES;

ERRO 1146: Tabela não existe.

← resultado

DELETE FROM ESTADOS;

2 linhas deletadas.

← resultado

# Structured Query Language - SQL

Consulta sem dados na relação e a inserção de dados:

```
SELECT * FROM ESTADOS;
```

não há linhas selecionadas.

← resultado

```
INSERT INTO ESTADOS(sigla, nome) VALUES('PA', 'PARA');
```

1 linha criada.

← resultado

```
INSERT INTO ESTADOS(nome, sigla) VALUES('Parana','PR');
```

1 linha criada.

← resultado

```
INSERT INTO ESTADOS VALUES('AM','AMAZONAS');
```

1 linha criada.

← resultado

```
SELECT NOME FROM ESTADOS;
```

resultado →

nome
PARA
Parana
AMAZONAS

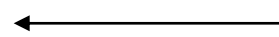
3 linhas apresentadas.

# Structured Query Language - SQL

Realizando algumas atualizações no banco de dados:

```
UPDATE ESTADOS SET NOME = 'BRASIL'
      WHERE NOME = 'PARANA';
```

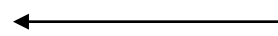
0 linhas atualizadas.



resultado

```
UPDATE ESTADOS SET NOME = 'BRASIL'
      WHERE NOME = 'Parana';
```

1 linha atualizada.

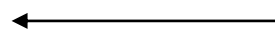


resultado

**ATENÇÃO** que alguns SGBDs respeitam o *case sensite*, de acordo com seu S.O. e sua própria configuração de instalação.

```
UPDATE ESTADOS SET NOME = 'ALAGOAS',
      SIGLA = 'AL' WHERE SIGLA = 'PA';
```

1 linha atualizada.



resultado

# Structured Query Language - SQL

Pesquisando as informações atuais no banco tem-se:

SELECT SIGLA , NOME FROM ESTADOS;	
SIGLA	NOME
-----	-----
AL	ALAGOAS
PR	BRASIL
AM	AMAZONAS

Especificando as colunas desejadas na ordem em que elas serão procuradas e apresentadas

A consulta de somente algumas tuplas pode ser conseguida por meio da cláusula **WHERE**. O uso desta cláusula é **opcional** e faz com que uma ou mais condições tenham que ser atendidas para que uma tupla seja recuperada e apresentada (**projetada ao usuário**).



# Structured Query Language - SQL

A forma básica de uma instrução **SELECT** pode ser generalizada em:

**SELECT** <atributos> **FROM** <tabela> **WHERE** <condições>;

Após o **WHERE** são especificadas as cláusulas condicionais que restringirão as tuplas a serem recuperadas pela consulta elaborada.

```
SELECT NOME FROM ESTADOS
```

```
WHERE SIGLA = 'PR';
```

```
NOME
```



resultado

```
-----
```

```
BRASIL
```

```
UPDATE ESTADOS SET NOME = 'BRASIL'
```

```
WHERE NOME = 'AMAPA';
```

0 linhas atualizadas.



resultado



# Structured Query Language - SQL

```
UPDATE ESTADOS SET NOME = 'BRASIL'
```

```
WHERE SIGLA = 'AM';
```

1 linha atualizada.

← resultado

```
SELECT * FROM ESTADOS;
```

```
SI  NOME
```

```
-----
```

```
AL  ALAGOAS
```

```
PR  BRASIL
```

```
AM  BRASIL
```

← resultado

```
SELECT NOME, SIGLA FROM ESTADOS
```

```
WHERE NOME = 'BRASIL';
```

```
NOME
```

```
SIGLA
```

```
-----
```

```
BRASIL
```

```
PR
```

```
BRASIL
```

```
AM
```

↑  
resultado

```
SELECT * FROM ESTADOS
```

```
WHERE NOME='BRASIL' AND SIGLA='AM';
```

```
SIGLA  NOME
```

```
-----  
AM
```

```
-----  
BRASIL
```

# Structured Query Language - SQL

Na cláusula **WHERE** serão utilizados alguns operadores de comparação e lógicos para que a condição seja elaborada e possa ser constatada com mais realidade.

Relacional (comparação)	
Oper.	Significado
=	igual a
>	maior que
<	menor que
>=	maior ou igual que
<=	menor ou igual que

Lógico	
Oper.	Significado
<b>AND</b>	E lógico
<b>OR</b>	OU lógico
<b>NOT</b>	Negação

SELECT NOME FROM ESTADOS

WHERE SIGLA = 'SP' **OR** NOME = 'ALAGOAS';

NOME

-----

ALAGOAS



resultado



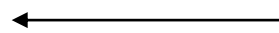
# Structured Query Language - SQL

A cláusula das outras instruções de manipulação da informação também atendem as características dos operadores lógicos e de comparação. Então, corrija as outras duas tuplas que aparecem com o nome de 'BRASIL' para o correto nome, de acordo com a sua sigla já cadastrada.

```
UPDATE ESTADOS SET NOME = 'PARANA'
```

```
WHERE SIGLA = 'PR';
```

1 linha atualizada.



resultado

```
UPDATE ESTADOS SET NOME = 'AMAZONAS'
```

```
WHERE SIGLA = 'AM';
```

1 linha atualizada.



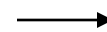
resultado

Agora, recupere somente as tuplas que sofreram alteração e verifique se elas estão corretas.

```
SELECT * FROM ESTADOS
```

```
WHERE SIGLA = 'PR' OR SIGLA = 'AM';
```

resultado



SIGLA NOME

-----

PR PARANA

AM AMAZONAS



# Structured Query Language - SQL

A apresentação dos dados recuperados pelo **SELECT** pode ser classificada (ordenada) de forma ascendente (crescente) ou descendente (decrescente).

Para isso se pode incluir a cláusula **opcional ORDER BY** no final da instrução **SELECT**.

```
SELECT NOME FROM ESTADOS
```

```
ORDER BY NOME ASC;
```

```
NOME
```

```
-----
```

```
ALAGOAS
```

```
AMAZONAS
```

```
PARANA
```

```
SELECT NOME FROM ESTADOS
```

```
ORDER BY NOME DESC;
```

resultado

```
NOME
```

```
-----
```

```
PARANA
```

```
AMAZONAS
```

```
ALAGOAS
```

# Structured Query Language - SQL

Uma classificação com mais que um atributo pode ser feita com a especificação ordenada de quais os atributos a serem usados na classificação e a ordem de prioridade para cada um.

Os atributos devem ser especificados separados por vírgula e devem estar em ordem de prioridade, conforme sua especificação na cláusula **ORDER BY**, em que o primeiro atributo especificado será classificado primeiramente e o segundo, respeitando a classificação do primeiro, será classificado em seguida, dentro da primeira classificação.

Exemplo:

```
SELECT NOME FROM EMPREGADO WHERE SALARIO < 0  
ORDER BY NOME, SALARIO; →
```


↓

Primeira classificação será feita por nome e dentro da classificação por nome será feita por salário

Sem a especificação de ASC ou DESC a consulta é classificada de forma ASC, por definição (padrão)

# Structured Query Language - SQL

Diante do avanço no estudo das instruções SQL, e vislumbrando a eficiência e agilidade dos profissionais envolvidos em trabalhar sobre as instruções SQL são empregadas boas práticas na elaboração dessas instruções, por exemplo:



```
SELECT idEmpregado, nome  
  FROM EMPREGADO  
  WHERE salario < 0  
  ORDER BY nome, salario;
```

Note que a instrução acima possui cada uma de suas cláusulas em um linhada, estando a primeira palavra-chave da cláusula de instrução alinha ao seu final (à direita).

Com isso a visão para compreensão da lógica envolvida é mais fácil e ágil aos profissionais da área.

# Structured Query Language - SQL

Baseado no exemplo anterior, observe o comando DDL que criaria uma relação compatível com a consulta solicitada pelo **SELECT** anteriormente proposto.

```
CREATE TABLE EMPREGADO (  
  idEmpregado      int(8)  NOT NULL,    -- código funcional  
  nome             varchar(40) NOT NULL,  -- nome do empregado  
  dtNascer         date     NOT NULL,    -- data de nascimento  
  salario          decimal(7.2) NOT NULL, -- valor do seu salário  
  cpf              int(11)  -- número do CPF, se tiver (não é obrigatório)  
);
```

## Padrões no Identificador dos atributos (há outros)

Identificador único => **id**Empregado (para identificadores)

Data de nascimento => **dt**Nascer (para datas)

# Structured Query Language - SQL

## FUNÇÕES DE GRUPO EM SQL

As funções de grupo trabalham com conjunto(s) de tuplas, ao invés de uma única tupla por vez, fornecendo o resultado do **SELECT** também avaliado por grupo.

As funções de grupo padrões disponíveis no **SQL** são:

**AVG** - valor médio do grupo, **ignorando os atributos nulos**

**COUNT** - conta quantidade de tuplas (\* - inclui nulos e duplicat.)

**MAX** - valor máximo **ignorando o nulo**

**MIN** - valor mínimo **ignorando o nulo**

**STDDEV** - desvio padrão dos valores selecionados, **sem nulos**

**SUM** - valores somados **ignorando os nulos**

**VARIANCE** - variação dos valores selecionados, **sem nulos**



# Structured Query Language - SQL

As funções de grupo **ignoram valores nulos** quando o atributo for usado para agrupar valores.

A exceção é a função COUNT que pode ser usada com o parâmetro pré-definido \* (asterisco) que solicitará a quantidade de tuplas que existe na relação referenciada.

```
SELECT COUNT(*)  
FROM JOSE.CIDADE;
```

Todas as demais funções de grupo **ignoram os valores nulos**, porém estas podem fazer uso também da função **NVL()**, caso seja necessário trabalhar com nulos.

=> A função **NVL** está disponível em **ORACLE** e não no **MySQL**.



# Structured Query Language - SQL

## FUNÇÃO NVL()

Esta função (**NVL**) permite o tratamento de atributos que **possuam o valor NULO armazenado**.

Ela pode ser aplicada sobre qualquer tipo de dado (numérico, caracter, data), em que o atributo que possuir seu valor **nulo** armazenado retornará um outro dado definido para este tipo específico.

valor de origem ou expressão  
que possa conter nulo

**NVL**(<expr1>,<expre2>)

valor de conversão para nulo  
que será retornado pela  
função

Exemplo de conversão:

NUMÉRICO = **NVL**(<coluna numérica> , 0)

DATA = **NVL**(<coluna de data> , '01-JAN-95')

CARACTER = **NVL**(<coluna caracter> , 'indisponível')

# Structured Query Language - SQL

## Atributo de Auto Incremento na Tabela

A implementação de um atributo (coluna da tabela) de **auto incremento** permite que um número inteiro único seja gerado quando um novo registro for inserido no **MySQL**, nunca sendo repetido este número.

A palavra ou expressão chave **AUTO\_INCREMENT** deve estar indicada em um único atributo da tabela. Seu valor inicial padrão é 1 e o incremento será de 1 em 1.

Porém, é possível definir um valor inicial diferente do padrão, assim como alterar o valor do auto incremento (**o passo**) para seguir a partir de um valor arbitrário que será indicado por uma instrução DDL (***alter table***).



# Structured Query Language - SQL

## Características do Atributo de **Auto Incremento**

- Ao inserir novos valores na tabela, não é necessário especificar o valor do atributo de *auto-incremento*;
- Só é permitido usar um atributo de auto incremento por tabela, geralmente do tipo inteiro;
- Necessita também da restrição (*constraint*) **NOT NULL**, que será configurado automaticamente, mas deve ser incluída na documentação e no *script* que cria a tabela corretamente;
- Exige criação da chave primária (**PK**) no atributo.



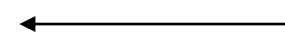
# Structured Query Language - SQL

Exemplo:

```
CREATE TABLE EQUIPE (  
    idEquipe int(5) NOT NULL AUTO_INCREMENT,  
    nome varchar(100) NOT NULL,  
    constraint equipe_PK PRIMARY KEY(idEquipe)  
) ENGINE = InnoDB AUTO_INCREMENT = 1 ;
```

```
INSERT INTO EQUIPE (nome) VALUES ('Flamengo');
```

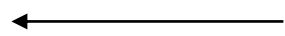
1 linha criada.



resultado

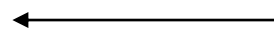
```
INSERT INTO EQUIPE VALUES (null, 'Vasco');
```

1 linha criada.



resultado

```
SELECT * FROM EQUIPE;
```



conferir seu resultado

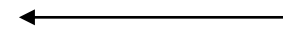


# Structured Query Language - SQL

## Exemplo:

```
INSERT INTO EQUIPE VALUES('Fluminense');
```

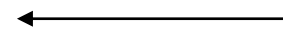
1 linha criada.



resultado

```
INSERT INTO EQUIPE VALUES(7, 'Botafogo');
```

1 linha criada.



resultado

```
SELECT *
```

```
FROM EQUIPE;
```



conferir seu resultado

```
INSERT INTO EQUIPE (nome) VALUES('Palmeiras');
```

1 linha criada.

Qual resultado (valor) será apresentado no atributo **idEquipe** dessa última inserção (*Palmeiras*)?



# Structured Query Language - SQL

Outros bancos de dados (SGBD) empregam outras formas de controle sobre valores únicos e sequenciais relevantes as implementações relacionais em SGBD's.

Por exemplo:

**ORACLE** utiliza os objetos de banco de dados conhecidos como SEQUENCE;

**PostgreSQL** usa também a SEQUENCE, mas com alguns outros recursos e sintaxes diferentes do **ORACLE**.



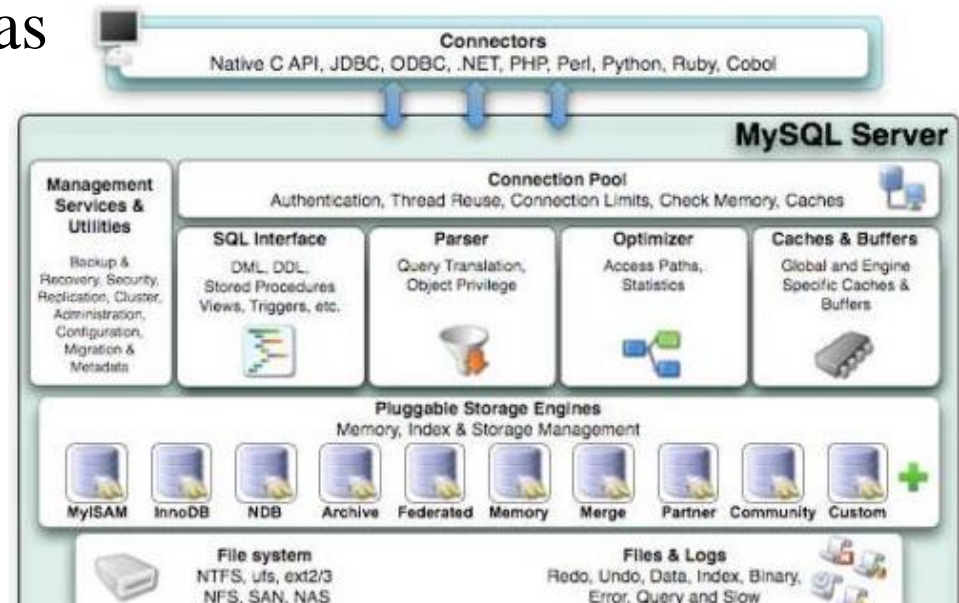
# Structured Query Language - SQL

## Opção **ENGINE** no **MySQL**

A **Engenharia de Armazenamento** (*Storage Engine*) confere características modulares que podem ser atribuídas as relações (tabelas) de uma base de dados no **MySQL**.

Essas características, também chamadas de Motores de Armazenamento, são nativas do **MySQL** (InnoDB, CSV, Federated, MyISAM, Memory, Archive), podendo outras personalizadas serem integradas a este SGBD.

**Consulte para saber mais ↓**



<https://medium.com/@claudioldf/diferen%C3%A7as-entre-as-storage-engines-do-mysql-myisam-innodb-e-outras-1dda5e3bac05>

# Structured Query Language - SQL

## Exemplo:

Suponha que as tabelas criadas em outros SGBD sejam sempre *Ferraris*, ou seja, tenham potente motor, grande velocidade e outros apetrechos de luxo.

Imagine também que exista a necessidade de cruzar uma avenida bem congestionada e você deverá escolher entre os veículos disponíveis qual será o mais rápido:

**uma Ferrari ou uma (excelente) Bicicleta**

No **MySQL** é possível escolher qual motor (*storage engine*) será usado de maneira coerente com a necessidade da atividade que será realizada. Assim, pode-se dirigir uma Ferrari (**tabelas transacionais**) ou pedalar uma bicicleta (**não transacionais**), conforme seja mais conveniente ao BD.



# Structured Query Language - SQL

Quais são as principais características, recursos ou funcionalidades inerentes as **Engenharias de Armazenamento** (*Storage Engines*) do **MySQL**:

- a) Capacidade Transacional: capacidade da tabela aceitar múltiplos acessos (de múltiplos usuários/aplicações), com colisão e travamento mínimos, sem que um usuário interfira com a operação do outro, além das propriedades **A.C.I.D.**;
- b) Meio de armazenamento: no **MySQL** a forma de armazenamento e uso das tabelas depende do motor escolhido;
- c) Índices: de acordo com o motor têm-se diferentes tipos de índices;
- d) Integridade Referencial: há motores que usam e outros que não usam e/ou respeitam a chave estrangeira (*foreign key*);



# Structured Query Language - SQL

- e) Tipo de bloqueio: capacidade de poder bloquear o acesso a um único registro (linha), vários registros ou a tabela inteira;
- f) Cópia de segurança *on line* (BOL – *Backup On Line*): realizar *backup* sem parar o trabalho do BD e de seus usuários;
- g) Reparação Automática (*Auto Recovery*): havendo falha no banco de dados (BD) alguns motores registrarão no LOG do erro que foi encontrado e consertado automaticamente.

Em resumo, *storage engine* (motor) não é simplesmente um tipo de tabela, mas características únicas e exclusivas do **MySQL** que permite escolher um conjunto de recursos pré-definidos para melhor atender aos requisitos e as necessidades **DE CADA TABELA** em sua base de dados, respeitando também o hardware disponível.





# Exercício de Fixação

- 1) Elabore a descrição em SQL das tabelas resultantes dos esquemas representados a seguir e implemente o motor de armazenamento (**ENGINE**) em **MySQL** que trabalha somente com transações seguras e respeitando as restrições de integridade referencial (chaves primárias e estrangeiras estabelecidas nos relacionamentos entre tabelas).

ALUNO

<u>matricula</u>	nome	rg	telefone	codigo
------------------	------	----	----------	--------

(n:1)

CURSO

<u>codigo</u>	nome
---------------	------

(1:n)

DISCIPLINA

<u>idDisciplina</u>	nome	codigo
---------------------	------	--------

# Exercício de Fixação

... continuação do exercício 1

Na solução do exercício 1 será criado um *script* com instruções DDL da Linguagem SQL.

Um *script* corresponde a uma documentação do Banco de Dados (SGBD) e por isso deve ser elaborado com extremo cuidado e completamente coerente com o DLD ou com o Diagrama de Esquemas (DE).

Nenhum *script* fica sem sua documentação mínima (cabeçalho), indicando a qual projeto ele pertence, o que ele realiza ou implementa e quem o criou ou modificou.



# Structured Query Language - SQL

## DOCUMENTANDO SCRIPT (cabeçalho)

A elaboração de qualquer script de banco de dados exige uma documentação mínima contendo:

- Identificação de qual projeto o *script* pertence;
- Banco de dados e base(s) de dados envolvidas;
- Autor ou autores do *script*;
- Data de criação e de alteração do *script*;
- Relato de cada alteração realizada no *script*;
- Síntese de todos os objetos e recursos de banco de dados que fazem parte do projeto que o *script* pertence.



# Structured Query Language - SQL

Exemplo do **cabeçalho obrigatório** que documenta um *script*:

```
-- ----- Acompanhamento Escolar -----
```

→ Identificação do projeto

```
--      SCRIPT DE CRIACAO (DDL)
```

```
-- Data Criacao .....: 21/03/2018
```

```
-- Autor(es) .....: Ana Maria Braga
```

```
-- Banco de Dados .....: MySQL 5.7
```

```
-- Base de Dados (nome) ...: bdEscolar
```

```
--
```

→ Dados fundamentais do *script*

```
-- Ultimas Alteracoes
```

```
-- 28/04/2018 => Criacao de nova tabela
```

```
--           => Incluída propriedade AUTO_INCREMENT na tabela CURSO
```

```
-- 30/04/2018 => Ajuste complementar na documentacao deste script
```

```
--
```

→ Alterações no *script* e no BD

```
-- PROJETO => 01 Base de Dados
```

```
--           => 03 Tabelas
```

```
--           => 02 Perfis de Usuario
```

```
-- -----
```

→ Síntese dos objetos de BD existentes no projeto

```
CREATE DATABASE bdEscolar;
```

```
:
```

→ Todos comandos SQL contidos no *script*

# Exercício de Fixação

- 2) Coerente com o *script* que implementa o exercício anterior, crie um novo *script* , bem documentado, que apagará cada uma das tabelas implementadas pelo script do exercício 1 desta aula, mas excluindo uma tabela por vez neste novo *script*.
- 3) Diante do *script* de implementação da base de dados do exercício 1 desta aula, elabore um novo *script* que simplesmente realizará a inserção coerente de pelos menos 3 tuplas em cada tabela criada, mantendo a coerência com cada dado que será armazenado em cada atributo, a fim de também realizar um teste sobre cada um dos atributos, relacionamentos e tabelas que guardarão os dados coerentes às necessidades do mundo real em que este projeto será implantando.

# Exercício de Fixação

- 4) A **empresa FUI** é organizada em departamentos. Cada departamento tem um nome, um número único e um empregado que gerencia o departamento. Deve-se saber a data em que este empregado iniciou como gerente do departamento. Um departamento pode ter diversas localizações. Um departamento controla um número de projetos, cada qual com um nome, um número único e uma única localização. São guardados o nome do empregado, idade, matrícula, endereço (rua, número, bairro), salário, sexo e data de nascimento. Um empregado está associado a um departamento, mas pode trabalhar em diversos projetos, não necessariamente controlados pelo mesmo departamento. Deve-se saber o número de horas semanais que cada empregado trabalha em cada projeto, bem como um único supervisor direto de cada empregado. Cada empregado pode possuir vários dependentes, devendo-se saber, para cada dependente, o nome, sexo, data de nascimento e a sua ligação (dependência) com o empregado.

# Referência de Criação e Apoio ao Estudo

## Material para Consulta e Apoio ao Conteúdo

- ELMASRI, R. e Navathe, S. B., Fundamentals of Database Systems, Addison-Wesley, 3rd edition, 2000
  - Capítulo 8
- SILBERSCHATZ, A. & Korth, H. F., Sistemas de Banco de Dados - livro
  - Capítulo 4
- Universidade de Brasília (UnB Gama)
  - <https://sae.unb.br/cae/conteudo/unbfga/>  
(escolha a disciplina **Sistemas Banco Dados 1**)

