# Tipos de dados, inserindo registros

Prof. Dr. Nazareno de Oliveira Pacheco
nazareno.pacheco@prof.sc.senac.br

# Tipos de dados

- Como foi visto na última aula, uma tabela é composta por diversas linhas (registros) e colunas (campos)
- Para criar um campo, é necessário definir seu nome e seu tipo

# Tipos de dados

- Existem três categorias principais de tipos dados
  - Numéricos
  - Texto
  - Data
- Cada categoria engloba alguns tipos diferentes

# Tipos de dados numéricos

- Cada tipo de dado usa uma quantidade de bytes para ser armazenado
- Quanto mais bytes:
  - Maior será o intervalo de números que podem ser armazenados neste campo
  - Mas espaço em disco será necessário para armazenar os dados

# Tipos de dados numéricos

| Dados Numéricos Inteiros | | |
|---|---|---|
| Tipo | Escopo com sinal | Escopo sem sinal |
| Tinyint | -128 a 127 | 0 a 255 |
| Smallint | -32.768 a 32.767 | 0 a 65.535 |
| Mediumint | -8.388.608 a 8.388.607 | 0 a 16.777.215 |
| Int | -2.147.483.648 a 2.147.483.647 | 0 a 4.294.967.295 |
| Bigint | -9.223.372.036.854.775.808 a 9.223.372.036.854.775.807 | 0 a 18.446.744.073.709.551.615 |

| Dados Numéricos (Bit e Boolean) | |
|---|---|
| Tipo | Numero máximo de bytes |
| bit | 1 |
| bool ou boolean | 1 |

| Dados Numéricos de Ponto Flutuante e Ponto Fixo | |
|---|---|
| Tipo | Escopo numérico |
| Float(p,e) | -3,402823466E+38 a -1,175494351E-38 e de 1.175494351E-38 a 3,402823466E+38 |
| Double(p,e) | -1,7976931348623157E+308 a -2,2250738585072014E-308 e de 2,2250738585072014E-308 a 1.7976931348623157E+308 |
| Decimal(p,e) | -1,7976931348623157E+308 a -2,2250738585072014E-308 e de 2,2250738585072014E-308 a 1.7976931348623157E+308 |

# smallint x integer x bigint

- Em geral, o ***integer*** oferece melhor desempenho
  - Utilize o ***smallint*** apenas se espaço em disco for extremamente crítico
  - Utilize o ***bigint*** apenas quando integer não for suficiente

# numeric e decimal

- Ambos os tipos são a mesma coisa (sinônimos)
- Podem ser declarados das seguintes forma:
  - numeric(6): um campo numérico com 6 dígitos, ou seja, no máximo 999999
  - numeric(6,2): um campo numérico com 6 dígitos, sendo que 2 deles são após a vírgula. Isto significa um valor máximo de 9999,99

# numeric e decimal

- Guardam sempre o valor exato, ao contrário dos tipos de dados de ponto flutuante (real, double precision)
- São utilizados normalmente para expressar valores monetários e outros valores que requerem exatidão
- A quantidade de bytes utilizada para armazenamento é variável (depende do tamanho do número armazenado)
- Seu desempenho é um pouco menor do que tipos inteiros

# float, double precision

- Armazenam valores expressados em ponto flutuante
- São números com precisão variável
- Tem uma desvantagem importante: sua representação nem sempre é exata
  - Isto significa que o valor armazenado pode ter uma pequena diferença do valor original.
- São mais eficientes do que campos numeric
- São pouco utilizados na prática

# serial, bigserial

- Similares ao integer e bigint, mas com funcionalidades adicionais
- Falaremos deles mais tarde

# Tipos de dados de texto

| Dados de Texto Não-Binário | |
|---|---|
| Tipo de texto | Numero máximo de bytes |
| Tinytext | 255 |
| Text | 65.535 |
| MediumText | 16.777.215 |
| LongText | 4.294.967.295 |
| Varchar | 65.535 |
| Char | 255 |

| Dados de Texto Binário | |
|---|---|
| Tipo de texto | Numero máximo de bytes |
| Tinyblob | 255 |
| Blob | 65.535 |
| Mediumblob | 16.777.215 |
| Longblob | 4.294.967.295 |
| Varbinary | 65.535 |
| Binary | 255 |

| Dados de Texto em Lista | | |
|---|---|---|
| Tipo | Numero máximo de bytes | Descrição do Tipo |
| ENUM('valor1','valor2',...) | 65535 | armazenam um dos valores listados ou NULL |
| SET('valor1','valor2',...) | 64 | armazenam um ou mais dos valores listados ou NULL |

# character, char

**1.CHAR:** O tipo de dados CHAR é usado para armazenar dados de texto fixos com um comprimento específico. Nesse sentido, ele é usado ao criar tabelas para armazenar valores de texto com o mesmo tamanho em todas as colunas. Veja o exemplo que cria uma tabela “meus_valores” com uma coluna “nome” e armazena textos de no máximo 10 caracteres de comprimento.

```
CREATE TABLE meus_valores (
 nome CHAR(10)
);
```

**2.VARCHAR:** O tipo de dados VARCHAR é semelhante ao tipo CHAR, mas permite que os valores de texto sejam variáveis, com comprimentos diferentes em cada coluna. Dessa forma, ele é usado ao criar tabelas para armazenar valores de texto com comprimentos variáveis. Veja o exemplo abaixo que cria uma tabela “meus_valores” com uma coluna “email” que pode armazenar textos de até 200 caracteres de comprimento, sendo flexível quanto ao tamanho dos dados armazenados :

```
CREATE TABLE meus_valores (
email VARCHAR(200)
 );
```

# text

**3.TEXT:** Utiliza-se o tipo de dados TEXT para armazenar dados de texto longos e complexos, com comprimentos que variam de 1 a 4 GB. Assim, armazena valores de texto com muito conteúdo, como artigos de blog e documentos. Veja o exemplo:

**INSERT INTO meus_valores (descricao) VALUES ('Este é um exemplo de valor de texto longo’);**

**4.BLOB:** Utiliza-se o tipo de dados BLOB para armazenar dados binários, como imagens, vídeos e arquivos de documentos. Nesse sentido, armazena valores de grande porte e pode armazenar até 65.535 bytes de dados. Veja o exemplo:

**INSERT INTO meus_valores (imagem) VALUES (FILE('/caminho/para/imagem.jpg'));**

# Tipos de dados de texto completo

Utiliza-se o tipo de dados **FULLTEXT** para armazenar dados de texto e realizar consultas de texto completo. Assim, suporta consultas de texto completo em vários campos e permite o uso de operações de colisão e proximidade para encontrar resultados mais precisos.

```
CREATE TABLE livros
(id INT PRIMARY KEY,
titulo TEXT,
autor TEXT,
conteudo TEXT,
KEY(conteudo)(FULLTEXT) );
```

```
INSERT INTO livros (id, titulo, autor, conteudo) VALUES (1, 'Livro 1', 'Autor 1', 'Conteúdo 1, Conteúdo 2, Conteúdo 3');
```

Neste exemplo, a coluna “conteudo” é do tipo TEXT e é indexada com FULLTEXT. Assim, isso permite que o MySQL realize consultas de texto completo em todas as colunas indexadas com FULLTEXT.

# Tipos de dados de data

| Dados Temporais | | |
|---|---|---|
| Tipo | Formato padrão | Valores permitidos |
| Date | AAAA-MM-DD | 1000-01-01 a 9999-12-31 |
| Datetime | AAAA-MM-DD HH:MI:SS | 1000-01-01 00:00:00 a 9999-12-31 23:59:00 |
| Timestamp | AAAA-MM-DD HH:MI:SS | 1970-01-01 00:00:00 a 2037-12-31 23:59:00 |
| Year | AAAA | 1901 a 2155 |
| Time | HHH:MI:SS | -838:59:59 a 838:59:59 |

# date

- Armazena uma data, sem consideração pelo horário
- Exemplos:
  - 17/07/2012
  - 24/12/1979
  - 17/12/2010

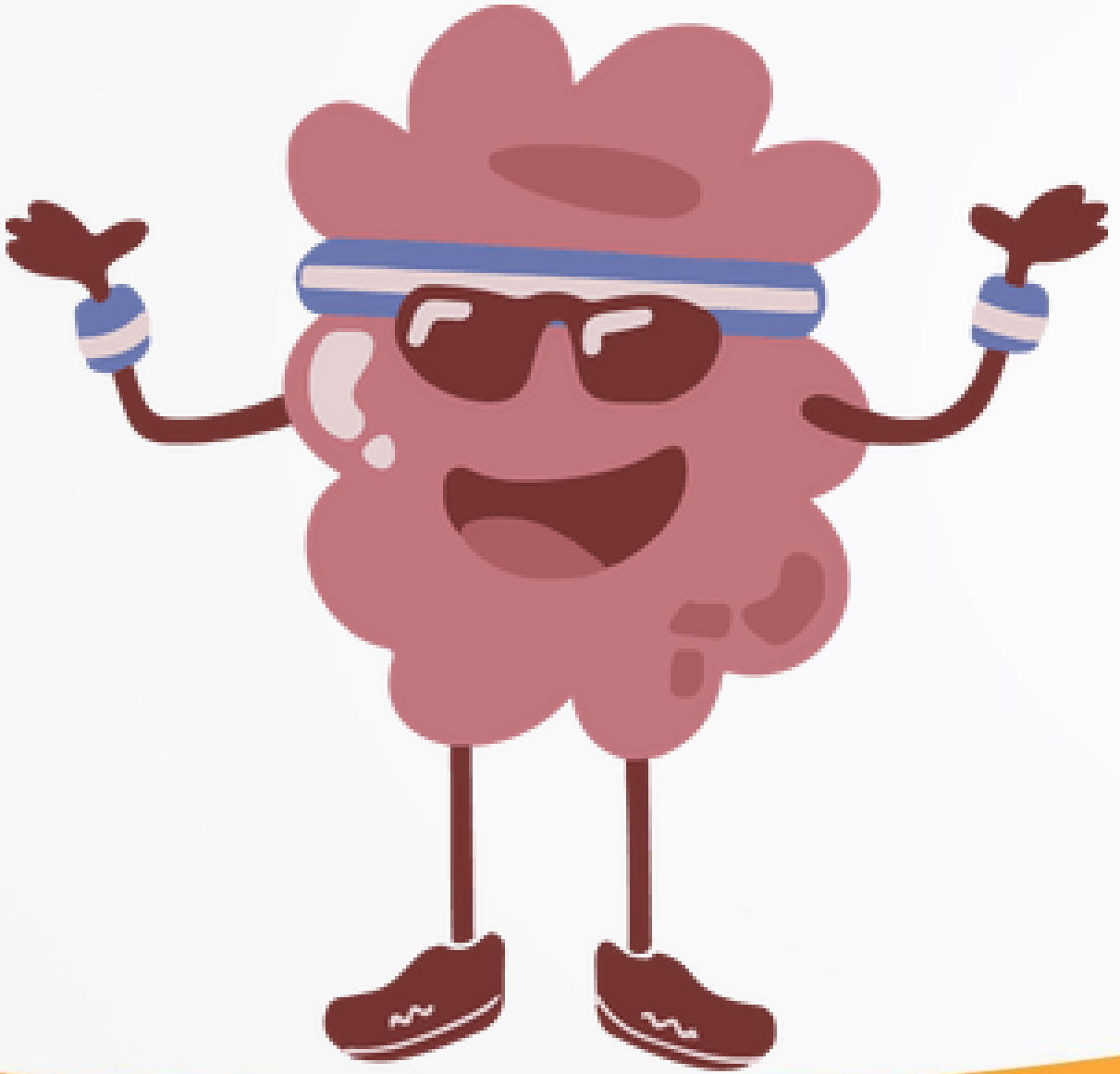
# time

- Armazena um horário, sem consideração pela data
- Exemplos
  - 10:26
  - 13:30
  - 17:30

# timestamp

- Armazena uma combinação de data e horário
- Exemplos
  - 17/07/2012 -10:26
  - 24/12/1979 - 13:30
  - 17/12/2010 - 17:30
- Muitos outros SGBD também utilizam o tipo datetime

# Exercícios



# Exercícios (2)

Crie uma tabela que armazene programas de televisão. Cada programa tem as seguintes informações

- Nome
- Emissora
- Data e horário de início
- Data e horário de fim

# Exercícios (3)

Crie uma tabela que armazene os dados de funcionários. Cada funcionário tem as seguintes informações

- CPF
- Número de matrícula
- Nome
- Horário de início de expediente
- Horário do final do expediente
- Salário

# Exercícios (4)

Crie uma tabela que controle a entrada e saída de carros em um estacionamento. É necessário registrar as seguintes informações

- Placa do carro
- Horário de entrada
- Horário de saída

# Exercícios (5)

Crie uma tabela que armazene dados dos livros de uma biblioteca. Cada livro tem:

- Código ISBN
- Título
- Autor
- Idioma
- Resumo
- Data de lançamento
- Preço

# Exercícios (5)

Crie uma tabela que armazene dados dos livros de uma livraria. Cada livro tem:

- Código ISBN
- Título
- Autor
- Idioma
- Resumo
- Data de lançamento
- Preço

# Exercícios (6)

Crie uma tabela que armazene a solução de cada exercício cadastrado até o momento

Cada exercício deve ter os seguintes dados:

- Número do exercício
- Descrição do exercício
- Data do exercício
- Descrição da solução

# Exercícios (7)

Crie uma tabela que armazene os dados dos usuários de um sistema qualquer

Cada usuário tem as seguintes informações

- Login
- Nome completo
- Senha
- E-mail
- Data do último acesso

# Até o momento...

- Até o momento apenas criamos bases de dados e tabelas
- Para isso usamos os comandos
  - CREATE DATABASE
  - CREATE TABLE
- Estes comandos só servem para definir como o nosso banco de dados é estruturado, mas não servem para adicionar novas informações

# DDL x DML

- É possível dividir o SQL em duas partes principais
- DDL (*Data Definition Language*):
  - São os comandos que definem como será a estrutura do nosso banco de dados
  - Incluem instruções para criar (*CREATE*), alterar (*ALTER*) ou apagar (*DROP*) estruturas como bases de dados (*DATABASE*), tabelas (*TABLE*) e índices

# DDL x DML

- DML (*Data Manipulation Language*)
  - Incluem comandos para inserir (*INSERT*), alterar (*UPDATE*), apagar (*DELETE*) ou pesquisar (*SELECT*) registros em uma base de dados que já foi definida pela DDL
- Existe também a DCL (*Data Control Language*), que gerencia a questão de permissões em um banco de dados. Falaremos dela mais tarde
- Para manipular registros em nossas tabelas, precisamos utilizar os comandos da DML

# Inserindo registros

- O comando utilizado para inserir novos registros em uma tabela é o INSERT INTO
- Este comando pode ser utilizado das seguintes formas:

```
INSERT INTO nome_tabela
VALUES (valor_coluna1, valor_coluna2,...);

OU

INSERT INTO nome_tabela (nome_col1, nome_col2)
VALUES (valor_coluna1, valor_coluna2,...);
```

# Exemplos de utilização

```
INSERT INTO funcionario
VALUES (2345,’Nazareno Pacheco', 22000);

INSERT INTO funcionario (matricula, nome, salario)
VALUES (2345, 'Nazareno Pacheco', 22000);

INSERT INTO funcionario (salario, nome, matricula)
VALUES (22000, 'Nazareno Pacheco', 2345);
```

# Sugestão

- Procure sempre utilizar esta sintaxe:

```
INSERT INTO nome_tabela (nome_col1, ...)
VALUES (valor_col1...);
```

- Apesar de mais extensa, ela é mais segura, pois evita confusão com a sequência das colunas sendo informadas

# Representando valores

- Os valores representados em um INSERT podem ser de três tipos principais:
  - Números
  - Texto
  - Data

# Representando números

- Números podem ser representados normalmente, utilizando . como separador das casas decimais
- Ao se inserir um número com mais casas decimais do que o campo de destino, este número será arredondado
- Observação: vamos falar do **select** mais a frente

# Experimente!

```
create table teste (
   coluna_integer integer,
   coluna_numeric numeric(4,2)
);

insert into teste (coluna_integer, coluna_numeric)
values (1.1, 1.1);
insert into teste (coluna_integer, coluna_numeric)
values (1.9, 1.9);
insert into teste (coluna_integer, coluna_numeric)
values (1.45, 1.45);
insert into teste (coluna_integer, coluna_numeric)
values (1.55, 1.55);
insert into teste (coluna_integer, coluna_numeric)
values (1.001, 1.001);

select * from teste;
```

# Representando texto

- Todo texto deve ser definido entre '  ' (aspas simples)

```
CREATE TABLE teste_varchar (
   coluna_varchar varchar(100)
);

INSERT INTO teste_varchar (coluna_varchar)
VALUES ('Meu teste');

INSERT INTO teste_varchar (coluna_varchar)
VALUES ('Varchar aceita símbolos $#@ e números 123');

SELECT * FROM teste_varchar;
```

# Representando data, hora

- Datas podem ser representadas utilizando um texto no seguinte formato:

  'aaaa-mm-dd'

  sendo
  - aaaa = ano (sempre com 4 dígitos)
  - mm = mês (sempre com 2 dígitos)
  - dd = dia (sempre com 2 dígitos)

# Representando data

- Datas podem ser representadas utilizando um texto no seguinte formato:

  'aaaa-mm-dd'

  sendo
  - aaaa = ano (sempre com 4 dígitos)
  - mm = mês (sempre com 2 dígitos)
  - dd = dia (sempre com 2 dígitos)

- Existem outras representações possíveis, mas esta é uma das mais recomendadas

# Representando horas

- Horas (time) podem ser representadas utilizando um texto nos seguintes formatos:

'hh:mm:ss.SSS'

'hh:mm:ss'

'hh:mm'

sendo

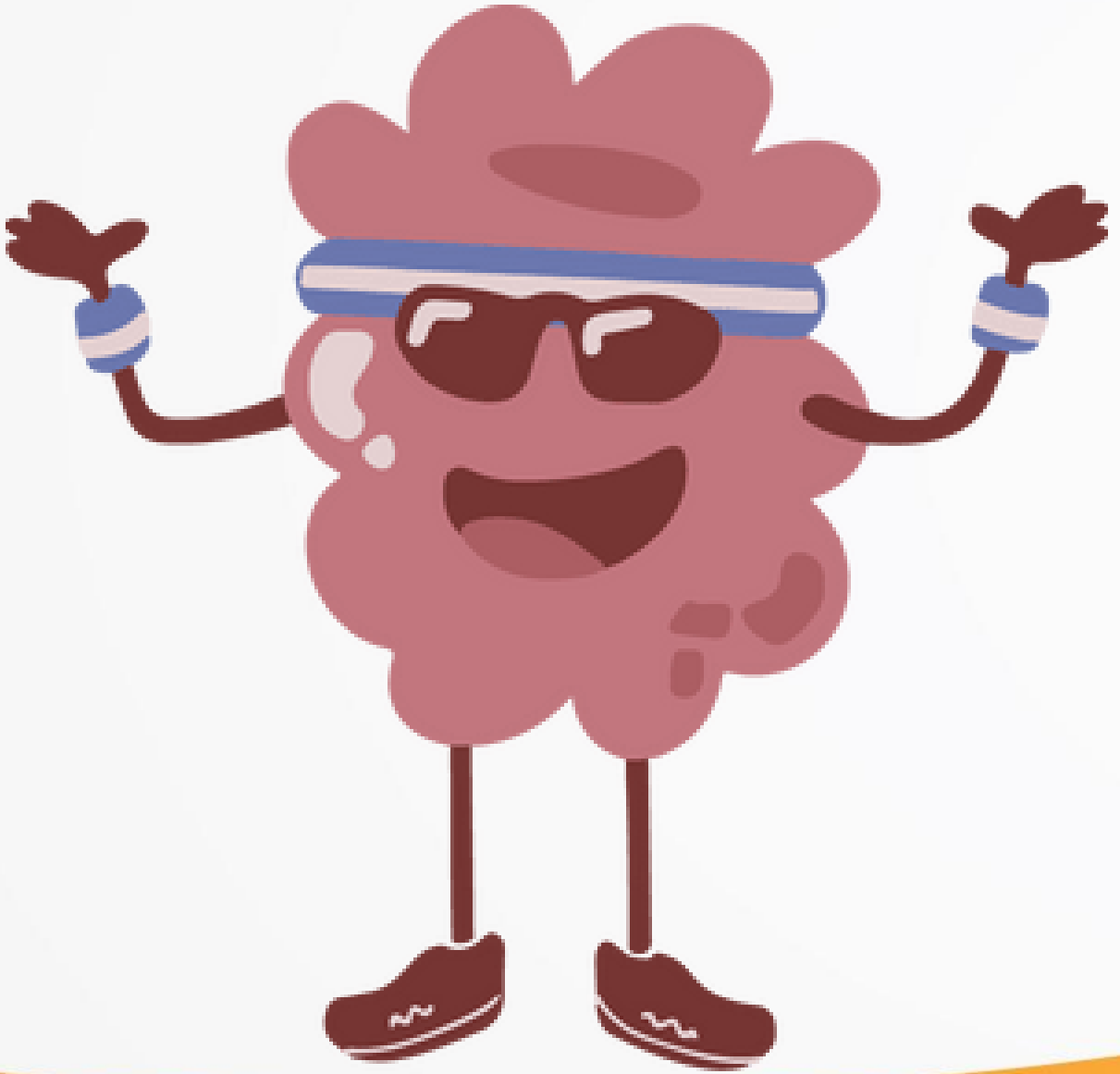
  - hh = hora (sempre com 4 dígitos)
  - mm = minutos (sempre com 2 dígitos)
  - ss = segundos (sempre com 2 dígitos)
  - SSS = milisegundos
- Se os segundos e/ou milissegundos não forem informados, serão considerados = 0

# Representando data e hora

- Datas e horas (timestamp) podem ser representadas combinando a representação de data e hora em um mesmo texto, separado por espaço

'aaaa-mm-dd hh:mm:ss'

# Exercícios



# Exercícios (8)

- Insira 3 registros em cada uma das tabelas criadas nos exercícios anteriores
- Para ver o resultado, utilize o comando

SELECT * FROM nome_tabela