

MODELO FÍSICO DE BANCO DE DADOS

ANTONIO DAVID VINISKI
antonio.david@pucpr.br
PUCPR

AGENDA

MODELAGEM LÓGICA DE BANCO DE DADOS

- MYSQL
 - SQL – DDL, DML, DCL
 - Características
- MODELO FÍSICO
 - Conceitos
 - Criação / Exclusão
 - Tipos de Dados
 - Tabelas / Atributos / Restrições
 - Modificações
 - Manipulação do Banco de Dados
 - INSERT, UPDATE, DELETE, SELECT



INTRODUÇÃO

NORMALIZAÇÃO DE DADOS



- O banco de dados precisa garantir a integridade e consistência dos dados que irá armazenar.
- Normalização:
 - Desenvolvimento de um modelo de armazenamento consistente.
 - Acesso eficiente e íntegro aos dados.
 - Conjunto de regras aplicadas sobre as tabelas (entidades) e seus relacionamentos.



MYSQL

SQL – STRUCTURED QUERY LANGUAGE

PROJETO FÍSICO - BANCO DE DADOS

- Linguagem de consulta estruturada.
- Desenvolvida pela IBM, nos anos 70;
 - Inicialmente chamada SEQUEL.
 - Parte do SystemR (protótipo de BD relacional)

SQL – CATEGORIAS

PROJETO FÍSICO - BANCO DE DADOS

- DDL – *Data Definition Language*
 - Composta pelos comandos **Create**, **Alter** e **Drop**
 - Responsável por dar forma ao banco de dados
- DML – *Data Manipulation Language*
 - Comandos **Select**, **Insert**, **Delete** e **Update**
 - Responsável por manipular os dados acrescentando, modificando, apagando e fazendo consultas
- DCL – *Data Control Language*
 - Subgrupo da DML, composta pelos comandos **Grant** e **Revoke**
 - Responsável por controlar acesso dos usuários aos dados.



MYSQL

PROJETO FÍSICO - BANCO DE DADOS

- Desenvolvido pela necessidade de um banco de dados relacional que pudesse tratar grandes quantidades de dados em máquinas de custo relativamente barato.
- O MySQL foi criado na Suécia por dois suecos e um finlandês:
 - David Axmark, Allan Larsson e Michael "Monty" Widenius.
- Conhecido por sua facilidade de uso, sua interface simples, e também sua capacidade de rodar em vários sistemas operacionais.



MYSQL - CARACTERÍSTICAS

PROJETO FÍSICO - BANCO DE DADOS

- Portabilidade (suporta praticamente qualquer plataforma atual)
- Compatibilidade (existem drivers ODBC, JDBC, .NET e módulos de interface para diversas linguagens de programação, como Delphi, Java, C/C++, Python, Perl, PHP, ASP e Ruby).
- Excelente desempenho e estabilidade, facilidade de uso, pouco exigente quanto a recursos de hardware.
- É um Software Livre com base na GPL.
- Contempla a utilização de vários *Storage Engines* como *MyISAM*, *InnoDB*, *Falcon*, *BDB*, *Archive*, *Federated*, *CSV*, *Solid*.
- Suporta controle transacional, triggers, Stored Procedures, Functions.
- Suporta Cursors (Non-Scrollable e Non-Updatable)
- Replicação facilmente configurável
- Interfaces gráficas (MySQL Toolkit) de fácil utilização cedidos pela MySQL Inc

MYSQL - CLIENTES

PROJETO FÍSICO - BANCO DE DADOS

Some of our customers

YouTube

PayPal

LinkedIn

facebook



ebay

CISCO

verizon

Uber

shopify

NETFLIX

GitHub

Walmart

Booking.com

Zappos

BANK OF AMERICA

Flipkart

paytm

Alibaba.com

WeChat

Tencent
腾讯

淘宝网
Taobao.com

airbnb

TOYOTA

MYSQL - CONCEITOS

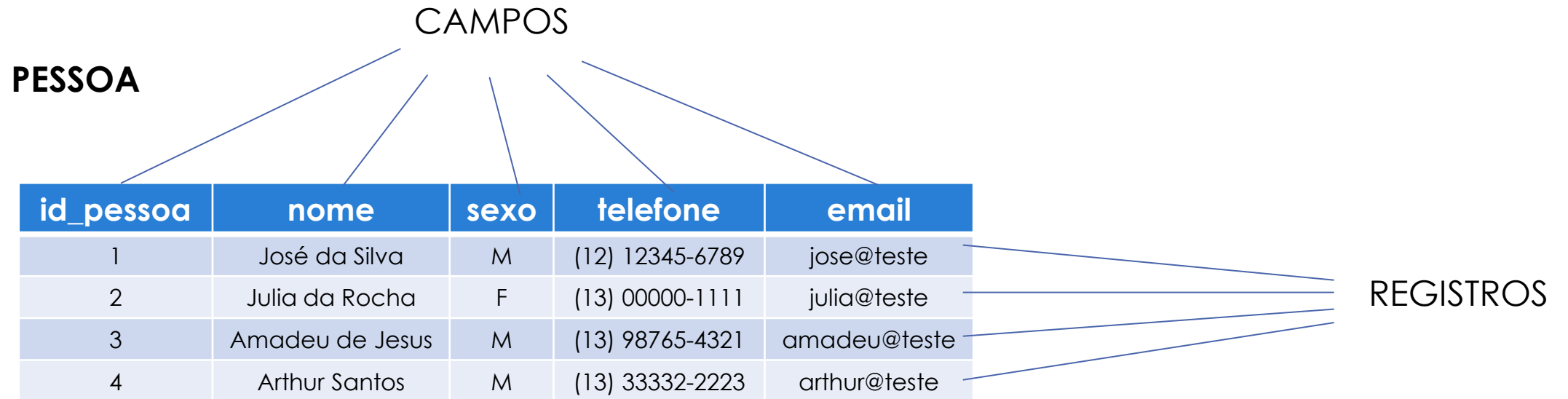
PROJETO FÍSICO - BANCO DE DADOS

- Banco de dados
 - Conjunto de tabelas
- Tabela
 - Conjunto de registros (Ex. Conjunto de clientes)
- Registro
 - Conjunto de campos que representa uma entidade (Ex. Dados de um cliente)
- Campo
 - Menor unidade de informação a ser armazenada.



EXEMPLO 1

PROJETO FÍSICO - BANCO DE DADOS



MYSQL - CRIANDO BANCO DE DADOS

PROJETO FÍSICO - BANCO DE DADOS

- Quando se cria um banco de dados com o MySQL em ambiente Windows, é criada apenas uma pasta vazia, dentro da qual serão armazenados os arquivos gerados utilizando os comandos para criar tabelas.
- Como o MySQL trata o banco de dados como um esquema, temos duas opções de sintaxe para criar um banco de dados:

CREATE DATABASE restaurante;

CREATE SCHEMA restaurante;



CLÁUSULA IF NOT EXIST

PROJETO FÍSICO - BANCO DE DADOS

- Não podemos criar bancos de dados com nomes iguais.
- Se tentarmos, será exibida a mensagem:
 - Can't create database 'restaurante'. Database exists
- Para evitar esse erro usamos a cláusula IF NOT EXISTS

CREATE DATABASE IF NOT EXISTS restaurante;

CREATE SCHEMA IF NOT EXISTS restaurante;



EXCLUINDO BANCO DE DADOS

PROJETO FÍSICO - BANCO DE DADOS

- Ao apagar um banco de dados, todas as tabelas e dados também serão excluídos.

DROP DATABASE restaurante;

DROP SCHEMA restaurante;

- Com o DROP também podemos usar a cláusula IF EXISTS para que não seja gerada uma exceção.

DROP DATABASE IF EXISTS restaurante;

DROP SCHEMA IF EXISTS restaurante;



SCHEMA/DATABASE EXISTENTES

PROJETO FÍSICO - BANCO DE DADOS

- Caso deseje mostrar os schemas/databases existentes:

SHOW DATABASES;

SHOW SCHEMAS;

- Todas as instruções executadas que não referenciem um respectivo banco de dados são direcionadas a um database padrão.
- Para definir o database padrão basta usar o comando **USE**.

USE restaurante;



APÓS A CRIAÇÃO DO DATABASE

PROJETO FÍSICO - BANCO DE DADOS

- Bancos de dados: pasta vazia
 - Precisamos criar tabelas dentro dele.
- Para criar tabelas precisamos definir sua estrutura = seus campos
 - Exemplo:
 - Tabela: cliente
 - Campos:
 - id_cliente, nome, email
- Também precisamos definir tipos dos Campos



TIPOS DE DADOS

TIPOS DE DADOS

PROJETO FÍSICO - BANCO DE DADOS

- No MySQL os tipos de dados são divididos em três grupos:
 - Tipos Numéricos.
 - Tipos de Data.
 - Tipo de *String*.
- Convenções:
 - **M** = tamanho do dado
 - **Inteiro**: comprimento máximo de exibição
 - **Ponto Flutuante ou fixo**: quantidade de dígitos que poderão ser armazenados
 - **Strings**: tamanho máximo da *string*.
 - **D**: Quantidade de dígitos depois da vírgula (ponto flutuante)



TIPOS DE DADOS NUMÉRICOS

PROJETO FÍSICO - BANCO DE DADOS

- **ZEROFILL**: esse atributo é responsável por preencher com zeros à esquerda do valor informado.
 - Ex.: em um **TINYINT**(3) **ZEROFILL** ao invés de armazenar o valor como "1", seria armazenado como "001";
- **Signed x Unsigned**: um número **signed** aceita dados negativos, enquanto o **unsigned** não.
 - Ex: **TINYINT** pode ter um range (faixa) de valores de 255:
 - **signed** seu valor pode variar de -128 a 127,
 - **unsigned** seu valor irá de 0 a 255;

ZEROFILL quando utilizado automaticamente adiciona o atributo **UNSIGNED** ao campo, ou seja, passa a não aceitar valores negativos.

TIPOS DE DADOS NUMÉRICOS

PROJETO FÍSICO - BANCO DE DADOS

- **BIT**[(**M**)]: como seu nome sugere, trata-se um dado do tipo bit, **M** indica o número de bits que o dado ocupará, podendo ser entre 1 e 64.
 - Caso não seja informado, o valor padrão é 1;
- **TINYINT**[(**M**)] [**UNSIGNED**] [**ZEROFILL**]: um inteiro muito pequeno. Seu valor varia entre -128 e 127 caso seja **signed**, ou de 0 a 255 quando **unsigned**;
- **BOOL** ou **BOOLEAN**: na prática é o mesmo que usar um **TINYINT**(1). Quando 0, é considerado falso, quando 1 (ou outro valor diferente de 0), é considerado verdadeiro.

TIPOS DE DADOS NUMÉRICOS

PROJETO FÍSICO - BANCO DE DADOS

- **SMALLINT**[(M)] [**UNSIGNED**] [**ZEROFILL**]: de -32768 a 32767 ou 0 a 65535, quando **SIGNED** ou **UNSIGNED**.
- **MEDIUMINT**[(M)] [**UNSIGNED**] [**ZEROFILL**]: de -8388608 a 8388607 ou 0 a 16777215, quando **SIGNED** ou **UNSIGNED**.
- **INT**[(M)] [**UNSIGNED**] [**ZEROFILL**]: um inteiro de tamanho padrão, seu valor pode ser de -2147483648 a 2147483647 ou de 0 a 4294967295.
 - **INTEGER** [(M)] [**UNSIGNED**] [**ZEROFILL**].
- **BIGINT**[(M)] [**UNSIGNED**] [**ZEROFILL**]: um inteiro grande, seu range é -9223372036854775808 a 9223372036854775807 ou 0 a 18446744073709551615

TIPOS DE DADOS NUMÉRICOS

PROJETO FÍSICO - BANCO DE DADOS

- **DECIMAL**[(**M** [, **D**))] [**UNSIGNED**] [**ZEROFILL**]: um número de ponto fixo. **M** indica a quantidade total de dígitos (precisão) e **D** indica quantos deles estarão “depois da vírgula”.
 - O separador de decimais (.) e o indicador de números negativos (-) não são contados em **M**.
 - No caso de o valor de **D** ser 0, o número não possuirá separador de decimais.
 - O valor máximo de **M** é 65 e de **D** é 30, caso omitidos, assumem os valores 10 e 0, respectivamente.
 - Todas as operações com decimais acontecem com uma precisão (**M**) de 65 dígitos.
- Um **DECIMAL** também pode ser representado pelas seguintes formas:
 - - **DEC**[(**M** [, **D**))] [**UNSIGNED**] [**ZEROFILL**];
 - - **NUMERIC**[(**M** [, **D**))] [**UNSIGNED**] [**ZEROFILL**];
 - - **FIXED**[(**M** [, **D**))] [**UNSIGNED**] [**ZEROFILL**]

TIPOS DE DADOS NUMÉRICOS

PROJETO FÍSICO - BANCO DE DADOS

- **FLOAT***[p]* [**UNSIGNED**] [**ZEROFILL**]: Um número de ponto flutuante.
 - *p* representa a precisão em bits, mas o MySQL usa este valor apenas para determinar se deve usar FLOAT ou DOUBLE para o tipo de dados resultante.
 - Se *p* for de 0 a 24, o tipo de dados se torna FLOAT sem valores M ou D.
 - Se *p* for de 25 a 53, o tipo de dados se torna DOUBLE sem valores M ou D.
 - O intervalo da coluna resultante é o mesmo para os tipos de dados FLOAT de precisão simples ou DOUBLE de precisão dupla descritos anteriormente nesta seção.

TIPOS DE DADOS NUMÉRICOS

PROJETO FÍSICO - BANCO DE DADOS

- **DOUBLE [UNSIGNED] [ZEROFILL]**: Um número de ponto flutuante de tamanho normal (precisão dupla).
 - Os valores permitidos são -1,7976931348623157E+308 a -2,2250738585072014E-308, 0 e 2,2250738585072014E-308 a 1,7976931348623157E+308.
 - Esses são os limites teóricos, baseados no padrão IEEE.
 - O alcance real pode ser um pouco menor dependendo do seu hardware ou sistema operacional.
- Outras formas de representação:
 - **DOUBLE PRECISION [UNSIGNED] [ZEROFILL], REAL[(M,D)] [UNSIGNED] [ZEROFILL]**
 - **Exception:** Se **REAL_AS_FLOAT** SQL mode está habilitado, REAL representa **FLOAT** ao invés de **DOUBLE**.

TIPOS NUMÉRICOS

PROJETO FÍSICO - BANCO DE DADOS

Tipo de dados	ARMAZENAMENTO	RANGE	
		SIGNED	UNSIGNED
TINYINT	1 byte	-128 a 128	0 a 255
SMALLINT	2 byte	-32768 a 32767	0 a 65535
MEDIUMINT	3 byte	-8388608 a 8388607	0 a 16777215
INT - INTEGER	4 byte	-2147483648 a 2147483647	0 a 4294967295
BIGINT	8 byte	-9223372036854775808 a 9223372036854775807	0 a 18446744073709551615
FLOAT(p)	4 bytes se $0 \leq p \leq 24$, 8 bytes se $25 \leq p \leq 53$		
FLOAT	4 byte	-3.402823466E+38 a 3.402823466E+38	0 a 3.402823466E+38
DOUBLE	8 byte	-1.7976931348623157E+308 a 1.7976931348623157E+308	0 a 1.7976931348623157E+308
DECIMAL(M,D)	Varia de acordo com M	Varia de acordo com o valor de M	
BIT(M)	Aproximadamente $(M+7)/8$ bytes	1 a 64	

TIPOS DE DADOS - DATA

PROJETO FÍSICO - BANCO DE DADOS

- Os tipos de dados de data e hora para representar valores temporais são:

DATE



TIME



DATETIME



TIMESTAMP



YEAR



TIPOS DE DADOS - DATA

PROJETO FÍSICO - BANCO DE DADOS

- **DATE:** usado para valores de data, mas sem a parte de hora.
 - O MySQL recupera e exibe valores **DATE** no formato 'AAAA-MM-DD'.
 - O intervalo suportado é '1000-01-01' a '9999-12-31'.
- **DATETIME:** usado para valores que contêm partes de data e hora.
 - O MySQL recupera e exibe os valores **DATETIME** no formato 'AAAA-MM-DD hh:mm:ss'.
 - O intervalo suportado é '1000-01-01 00:00:00' a '9999-12-31 23:59:59'.
- **TIMESTAMP** também usado para valores que contêm partes de data e hora.
 - Tem um intervalo de '1970-01-01 00:00:01' UTC a '2038-01-19 03:14:07' UTC.

TIPOS DE DADOS - DATA

PROJETO FÍSICO - BANCO DE DADOS

- **TIME:** usado para armazenar horas do dia ou intervalos de tempo
 - O MySQL recupera e exibe valores no formato 'hh:mm:ss'
 - ou formato 'hhh:mm:ss' para valores de horas grandes.
 - Os valores de **TIME** podem variar de '-838:59:59' a '838:59:59'.
- **YEAR:** Usado para armazenar um ano 'yyyy'.
 - formato com 2 ou 4 algarismos.
 - faixa de 1901 até 2155.

TIPOS DE DADOS - STRING

PROJETO FÍSICO - BANCO DE DADOS



- Os tipos de dados de *string* são: **CHAR**, **VARCHAR**, **BINARY**, **VARBINARY**, **BLOB**, **TEXT**, **ENUM**, **SET**.
- Cada coluna de “caractere” (ou seja, uma coluna do tipo **CHAR**, **VARCHAR**, um tipo **TEXT** ou qualquer sinônimo) possui um conjunto de caracteres de coluna (**CHARACTER SET**) e um agrupamento de colunas (**COLLATION**).
 - A sintaxe de definição de coluna para **CREATE TABLE** e **ALTER TABLE** tem cláusulas opcionais para especificar o conjunto de caracteres da coluna e o agrupamento:

```
col_name {CHAR | VARCHAR | TEXT} (col_length) [CHARACTER  
SET charset_name] [COLLATE collation_name]
```

```
col_name {ENUM | SET} (val_list) [CHARACTER SET  
charset_name] [COLLATE collation_name]
```



TIPOS DE DADOS STRING- CHARSET

PROJETO FÍSICO - BANCO DE DADOS



○ EXEMPLOS:

```
CREATE TABLE t1 (  
    col1 CHAR(10) CHARACTER SET utf8mb4  
) CHARACTER SET latin1 COLLATE latin1_bin;
```

```
CREATE TABLE t1 (  
    col1 CHAR(10) CHARACTER SET utf8mb4  
    COLLATE utf8mb4_unicode_ci  
) CHARACTER SET latin1 COLLATE latin1_bin;
```

○ Verifique as opções disponíveis e suas COLLECTIONs padrões:

SHOW CHARSET;

SHOW CHARACTER SET;

SHOW COLLATION WHERE CHARSET = 'utf8mb3';

TIPOS DE DADOS - STRING

PROJETO FÍSICO - BANCO DE DADOS

- **CHAR**[(**M**)] [**CHARACTER SET** charset_name] [**COLLATE** collation_name]: uma string de tamanho fixo (preenchida), representado pelo **M** que pode assumir um valor entre 0 e 255.
 - Caso todas as posições da *string* não sejam utilizadas, ela será completada com espaços até atingir o tamanho total.
 - Se o **M** for omitido, ele assume um valor padrão de 1.
- **VARCHAR** [(**M**)] [**CHARACTER SET** charset_name] [**COLLATE** collation_name]: uma *string* de tamanho variável. O valor de **M** pode variar de 0 a 65535, mas o valor máximo está intimamente ligado ao *charset* utilizado.
 - Por exemplo, caracteres utf8 utilizam 3 bytes para armazenar cada caractere, portanto o valor máximo de **M** para este *charset* é de 2184;
 - Para o *charset* ASCII, os caracteres utilizam apenas 1 byte, ou seja, **M** pode valer até 65535.

TIPOS DE DADOS - STRING

PROJETO FÍSICO - BANCO DE DADOS

- **BINARY[(M)]**: similar o tipo **CHAR**, porém armazena os valores binários das strings ao invés dos caracteres não binários.
 - M pode ser informado para definir o tamanho da coluna e caso omitido, recebe o valor 1..
- **VARBINARY(M)**: similar ao VARCHAR, porém armazena os valores binários das strings ao invés dos caracteres.
 - **M** indica o tamanho máximo.

TIPOS DE DADOS - STRING

PROJETO FÍSICO - BANCO DE DADOS

- **BLOB** [(**M**)]: seu nome significa “*Binary Large Object*”, ou seja um objeto binário grande.
 - Seu tamanho máximo é 65535 bytes (65KB).
 - Caso um valor seja informado para **M**, o MySQL vai criar o menor subtipo de **BLOB** possível para armazenar aquela quantidade de dado
- Seus subtipos são:
 - **TINYBLOB**: tamanho máximo de 255 bytes;
 - **MEDIUMBLOB**: tamanho máximo de 16.777.326 bytes (16MB);
 - **LONGBLOB**: tamanho máximo de 4.294.967.295 bytes (4GB).

TIPOS DE DADOS - STRING

PROJETO FÍSICO - BANCO DE DADOS

- **TEXT** [(M)][**CHARACTER SET** charset_name] [**COLLATE** collation_name]: suas características são como as do BLOB
 - *charset* influencia no tamanho máximo, assim como nos dados **VARCHAR**.
 - O valor máximo de M é 65535.
- Seus subtipos são:
 - **TINYTEXT**: suporta até 255 caracteres;
 - **MEDIUMTEXT**: suporta até 16.777.326 caracteres;
 - **LONGTEXT**: suporta até 4.294.967.295.

TIPOS DE DADOS - STRING

PROJETO FÍSICO - BANCO DE DADOS

- **ENUM**('valor1','valor2',...) [**CHARACTER SET** charset_name] [**COLLATE** collation_name]: um enumerador que pode receber apenas 1 valor
 - o valor de uma coluna deste tipo só pode ser selecionado se estiver informado na lista de valores (valor1, valor2), ser NULL ou vazio, este último indica erro.
 - Dados do tipo **ENUM** são armazenados como um inteiro pelo banco.
- **ENUM** pode conter uma coleção de no máximo 65535 elementos.

TIPOS DE DADOS - STRING

PROJETO FÍSICO - BANCO DE DADOS

- **SET**('valor1','valor2',...) [**CHARACTER SET** charset_name] [**COLLATE** collation_name]: uma string que pode ser composta de 0 ou mais valores,
 - cada valor deve ser escolhido na lista de valores informados na criação da coluna (valor1, valor2).
- Dados do tipo **SET** são armazenados como um inteiro pelo banco.
- Um **SET** pode conter uma coleção de no máximo 64 elementos.

TIPOS DE DADOS - STRING

PROJETO FÍSICO - BANCO DE DADOS

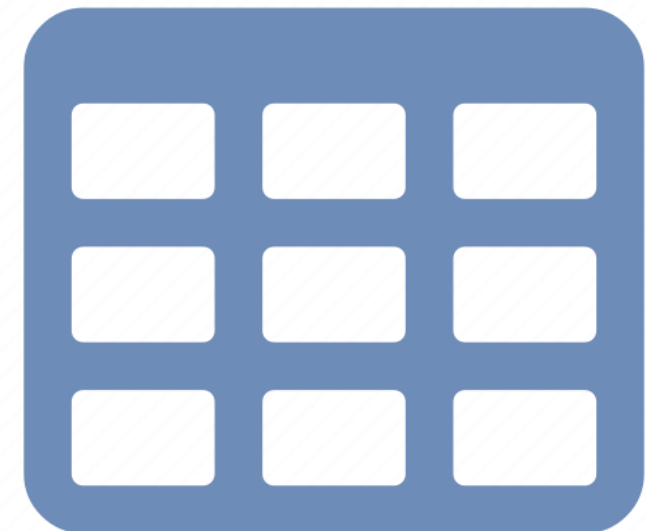
- O tamanho máximo de cada elemento dos tipos **ENUM** e **SET** tem que obedecer a algumas regras: $M \leq 255$ e $(M \times w) \leq 120$.
 - **M** é o comprimento literal e **w** é o valor em bytes do tamanho máximo de cada caractere do charset. Exemplo:
 - Usando o charset **utf8**, o comprimento máximo de cada caractere é 3 bytes, portanto nosso $w=3$, isso limita o valor de M a 340 por $(340 \times 3) = 1020$, ou seja, cada elemento do **ENUM** pode ter no máximo 340 caracteres.

INSTRUÇÕES EM TABELAS

INSTRUÇÕES - TABELAS

PROJETO FÍSICO - BANCO DE DADOS

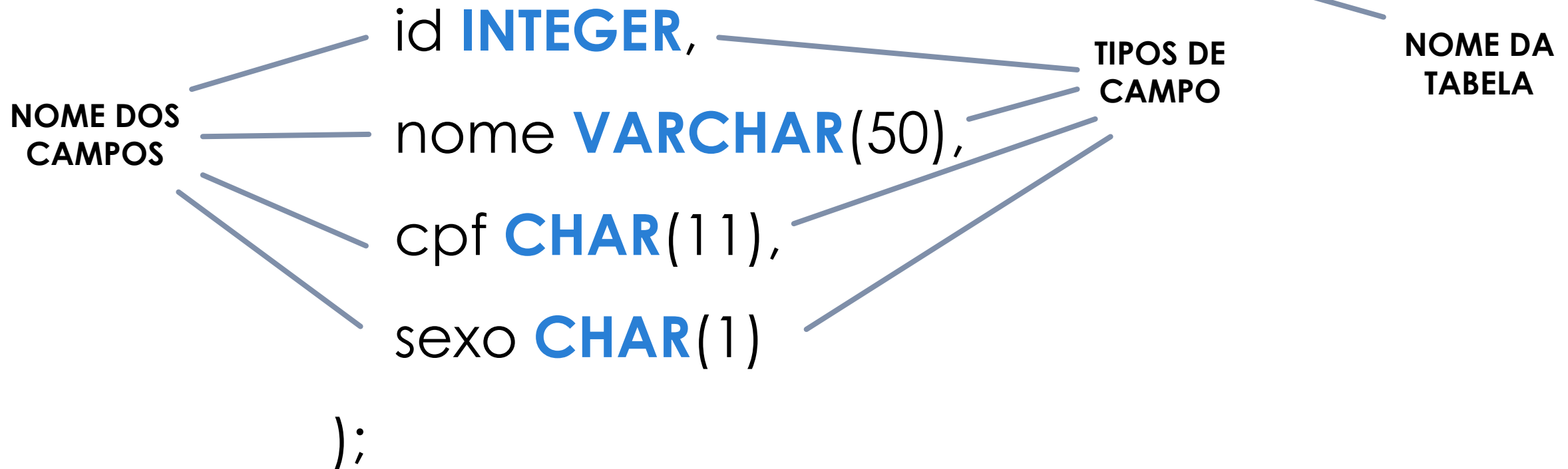
- **SQL** oferece três instruções para definição do esquema da base de dados:
 - **CREATE TABLE**
 - define a estrutura da tabela;
 - cria a tabela vazia;
 - **DROP TABLE**
 - Elimina a tabela da base de dados;
 - **ALTER TABLE**
 - Permite modificar a definição da tabela;



TABELAS – CRIANDO UMA TABELA

PROJETO FÍSICO - BANCO DE DADOS

CREATE TABLE restaurante.pessoa(



RESTRIÇÕES NAS TABELAS

PROJETO FÍSICO - BANCO DE DADOS

- Uma maneira de limitar os dados que podem ser inseridos em uma tabela é a definição dos tipos.
- Podemos também definir outros tipos de restrições, como:
 - Restrições de integridade de domínio.
 - Restrições de integridade de vazio.
 - Restrições de integridade de chave.
 - Restrições de integridade referencial.

RESTRIÇÕES - VAZIO


PROJETO FÍSICO - BANCO DE DADOS

- Restrições de Vazio:
 - NÃO NULO
 - A clausula **NOT NULL** especifica que uma coluna não permite valor vazio.

```
CREATE TABLE restaurante.pessoa(  
  id INTEGER NOT NULL,  
  nome VARCHAR(50) NOT NULL,  
  cpf CHAR(11) NOT NULL,  
  sexo CHAR(1)  
);
```

NÃO PERMITEM NULOS

PERMITE NULO



RESTRIÇÕES - UNICIDADE

PROJETO FÍSICO - BANCO DE DADOS

- Define que os dados contidos na coluna (ou grupo de colunas) sejam únicos (não se repitam) em relação a todos os outros registros (linhas) da tabela.
 - A cláusula **UNIQUE** especifica a unicidade de um campo ou um conjunto de campos.

```
CREATE TABLE restaurante.pessoa(  
  id INTEGER NOT NULL,  
  nome VARCHAR(50) NOT NULL,  
  cpf CHAR(11) UNIQUE NOT NULL,  
  sexo CHAR(1)  
);
```

cpf único,
não se
repete

OU

```
CREATE TABLE restaurante.pessoa(  
  id INTEGER NOT NULL,  
  nome VARCHAR(50) NOT NULL,  
  cpf CHAR(11) NOT NULL,  
  sexo CHAR(1),  
  UNIQUE(cpf) ——— cpf único  
);
```

RESTRIÇÕES - UNICIDADE

PROJETO FÍSICO - BANCO DE DADOS

```
CREATE TABLE restaurante.pessoa(  
    id INTEGER NOT NULL,  
    nome VARCHAR(50) NOT NULL,  
    sexo CHAR(1),  
    cpf CHAR(11),  
    UNIQUE(nome, cpf)  
);
```

Combinação nome,cpf
única, não se repete

Se uma restrição de unicidade faz referência a um grupo de colunas, elas são separadas por vírgula

RESTRIÇÕES – CHAVE PRIMÁRIA

PROJETO FÍSICO - BANCO DE DADOS

- A **chave primária** indica que a coluna, ou grupo de colunas, pode ser usado como identificador único para as linhas da tabela.
- É a junção da restrição de **unicidade** com a restrição de **vazio**.
 - Apenas a restrição de unicidade não garante identificador único pois não exclui os valores nulos.
- Uma tabela pode ter **no máximo uma chave primária**, (mesmo contendo um conjunto de colunas, é uma única chave) mas pode ter várias restrições de unicidade e não nulo.

RESTRIÇÕES – CHAVE PRIMÁRIA

PROJETO FÍSICO - BANCO DE DADOS

CHAVE
PRIMÁRIA

```
CREATE TABLE restaurante.pessoa(  
  id INTEGER NOT NULL PRIMARY KEY,  
  nome VARCHAR(50) NOT NULL,  
  cpf CHAR(11) UNIQUE NOT NULL,  
  sexo CHAR(1)  
);
```

```
CREATE TABLE restaurante.pessoa(  
  id INTEGER NOT NULL,  
  nome VARCHAR(50) NOT NULL,  
  cpf CHAR(11) UNIQUE NOT NULL,  
  sexo CHAR(1),  
  PRIMARY KEY(id)  
);
```

CHAVE
PRIMÁRIA

RESTRIÇÕES – AUTO INCREMENTO

PROJETO FÍSICO - BANCO DE DADOS

- O auto incremento faz com que conforme novos registros são criados, automaticamente estes obtém valores que correspondem ao valor deste mesmo campo no registro anterior, somado a 1.

```
CREATE TABLE restaurante.pessoa(  
    id INTEGER AUTO_INCREMENT NOT NULL,  
    nome VARCHAR(50) NOT NULL,  
    cpf CHAR(11) UNIQUE NOT NULL,  
    sexo CHAR(1),  
    PRIMARY KEY(id)  
);
```

RESTRIÇÕES – CHAVE ESTRANGEIRA

PROJETO FÍSICO - BANCO DE DADOS

- A **chave estrangeira** é responsável por relacionar duas tabelas.
- **FOREIGN KEY**: Comando SQL para definir a chave estrangeira.
 - Para definir uma chave estrangeira, a tabela que ela referencia já deve estar criada no banco de dados.
- Exemplo:
 - Criação da tabela cliente.
 - Cliente é um tipo especial de pessoa.
 - Criação da tabela comanda.
 - Comanda pertence a um cliente.

RESTRIÇÕES – CHAVE ESTRANGEIRA

PROJETO FÍSICO - BANCO DE DADOS

Tabela **cliente** possui o atributo **id**, que referencia o atributo **id** da tabela **pessoa**.

```
CREATE TABLE restaurante.cliente(  
    id INTEGER NOT NULL,  
    data_criacao DATE NOT NULL,  
    PRIMARY KEY (id),  
    FOREIGN KEY (id) REFERENCES pessoa(id)  
);
```

Tabela pessoa deve existir.

```
CREATE TABLE restaurante.comanda(  
    id INTEGER NOT NULL AUTO_INCREMENT,  
    data_hora DATETIME NOT NULL,  
    cliente_id INTEGER NOT NULL,  
    PRIMARY KEY (id),  
    FOREIGN KEY (cliente_id) REFERENCES cliente(id)  
);
```

Tabela cliente deve existir.

Tabela **comanda** possui o atributo **cliente_id**, que referencia o atributo **id** da tabela **cliente**.

ALTERAÇÕES DE TABELAS

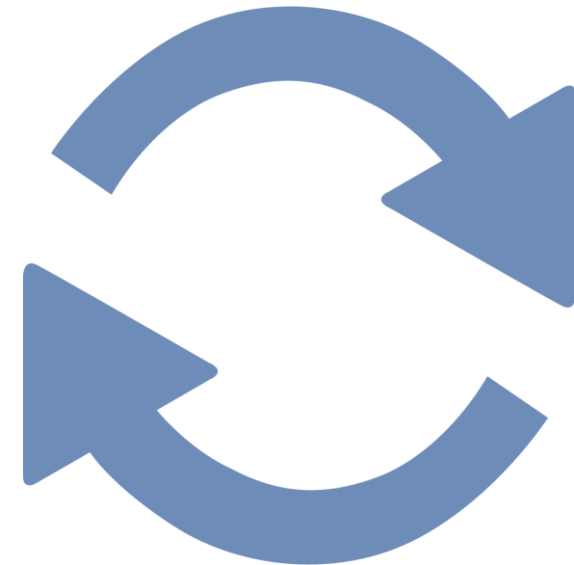
PROJETO FÍSICO - BANCO DE DADOS

- Quando notamos que as necessidades da aplicação mudaram ou que foi cometido um erro, podemos modificar a estrutura das tabelas já criadas.
 - Podemos incluir ou excluir colunas.
 - Adicionar restrições.
 - Modificar o tipo dos atributos e seus domínios.
 - Modificar nome de coluna ou da própria tabela.
- Tudo isso pode ser feito através do comando **ALTER TABLE**.

ALTERAÇÕES DE TABELAS

PROJETO FÍSICO - BANCO DE DADOS

- **ALTER TABLE ADD** <campo> <tipo>
 - Insere novo campo.
- **ALTER TABLE DROP** <campo>
 - Remove determinado campo.
- **ALTER TABLE MODIFY** <campo> <tipo>
 - Modifica o tipo de determinado campo.
- **ALTER TABLE CHANGE** <campo> <tipo>
 - Troca o nome de determinado campo.



ALTERAÇÕES DE TABELAS -ADD

PROJETO FÍSICO - BANCO DE DADOS

- Inserir na tabela pessoa o campo data de nascimento:

ALTER TABLE pessoa **ADD** data_nascimento **DATE NOT NULL**;

- Para que o campo seja inserido em uma posição específica da tabela.

ALTER TABLE pessoa **ADD** email **VARCHAR**(40) **NOT NULL AFTER** nome;

ALTER TABLE pessoa **ADD** data_nascimento **DATE NOT NULL FIRST**;

Sempre que adicionamos um campo em uma tabela que já contém registros, não podemos utilizar a cláusula **NOT NULL**.

ALTERAÇÕES DE TABELAS - MODIFY

PROJETO FÍSICO - BANCO DE DADOS

- O campo nome foi criado com limite de 50 caracteres.
 - trocar para 60 o limite de caracteres permitido.

ALTER TABLE pessoa **MODIFY** nome **VARCHAR**(60) **NOT NULL**;

- Para verificar as alterações, utilize o comando **DESCRIBE** pessoa.

ALTERAÇÕES DE TABELAS - CHANGE

PROJETO FÍSICO - BANCO DE DADOS

- Trocar o nome do campo data_nascimento para data_nasc

ALTER TABLE pessoa **CHANGE** data_nascimento data_nasc **DATE NOT NULL;**

- Execute o **DESCRIBE** cliente para observar as alterações.

ALTERAÇÕES DE TABELAS - DROP

PROJETO FÍSICO - BANCO DE DADOS

- Excluir o campo email da tabela pessoa

ALTER TABLE pessoa **DROP** email;

- Execute o **DESCRIBE** cliente para observar as alterações.

RENOMEANDO A TABELA

PROJETO FÍSICO - BANCO DE DADOS

- Para alterar o nome da tabela comanda para consumo:

ALTER TABLE comanda **RENAME TO** consumo;

ATIVIDADE 1

PROJETO FÍSICO - BANCO DE DADOS

- Criação do **SCRIPT** para definição da estrutura física de armazenamento do banco de dados do restaurante.
 - A. Criar todas as tabelas necessárias.
 - B. Criar os campos possíveis para cada tabela.
 - I. Definir os tipos dos campos.
 - II. Criar as chaves primárias.
 - III. Adicionar as chaves estrangeiras (se necessário), responsáveis por realização a associação (relacionamento) entre as tabelas.

MANIPULAÇÃO DO DATABASE/SCHEMA

MANIPULAÇÃO DE DADOS

PROJETO FÍSICO - BANCO DE DADOS

- Existem **quatro operações básicas** que permitem a manipulação dos dados em um DATABASE.
 - A. **INCLUIR.**
 - B. **APAGAR.**
 - C. **ALTERAR.**
 - D. **PESQUISAR**
- Muito importante lembrar que a **sintaxe do MYSQL** deve ser utilizada de forma correta para que as operações desejadas sobre os dados sejam efetivamente realizadas.

INSERINDO REGISTROS

PROJETO FÍSICO - BANCO DE DADOS

- Para adicionar dados a uma tabela, usamos o comando **INSERT**.

```
INSERT INTO restaurante.pessoa (nome, cpf, sexo, data_nasc)  
VALUES ('João da Silva', '11111111111', 'M', '2000-06-09');
```

```
INSERT INTO restaurante.cliente(id, data_criacao)  
VALUES (1 , '2022-08-30');
```

```
INSERT INTO restaurante.comanda(data_hora, cliente_id)  
VALUES ('2022-08-30 22:45:00', 1);
```

INSERINDO REGISTROS

PROJETO FÍSICO - BANCO DE DADOS

- Para adicionar mais de um registro em uma única instrução **INSERT**, separamos as *tuplas* com os valores por vírgula:

```
INSERT INTO restaurante.pessoa(nome, cpf, sexo, data_nasc)
VALUES ('Jorge dos Santos', '22222222222', 'M', '2002-03-18'),
      ('Maria de Jesus', '33333333333', 'F', '1998-01-16');
```

```
INSERT INTO restaurante.cliente (id, data_criacao)
VALUES (2, '2022-08-30'), (3, '2022-08-30');
```

```
INSERT INTO restaurante.comanda (data_hora, cliente_id)
VALUES ('2022-07-14', 1), ('2022-08-30', 2), ('2022-08-31', 3);
```

INSERINDO REGISTROS

PROJETO FÍSICO - BANCO DE DADOS


- Todos os campos que contém texto, ou seja, **CHAR**, **VARCHAR**, **BLOB**, **TEXT**, etc., têm de ficar entre apóstrofos.
 - Para campos do tipo número, não se usam apóstrofos.
- A entrada **NULL** em um campo do tipo autoincremento, permite que o MySQL providencie o conteúdo deste campo de forma automática.
 - No caso do primeiro campo, o valor será 1, no segundo 2, no terceiro 3 e assim consecutivamente.
- Se possuíssemos um campo **DATE**, a entrada **NULL** faria com que o valor gravado no registro se torne a data atual.

PESQUISANDO REGISTROS

PROJETO FÍSICO - BANCO DE DADOS


- As pesquisas no MySQL são feitas através do comando **SELECT**.
 - Podemos informar quais os campos desejamos retornar.
 - Também podemos utilizar (*) para retornar todos os campos da tabela.
- EXEMPLO:

SELECT nome, data_nasc **FROM** restaurante.pessoa;



Retorna os valores dos campos nome e data_nasc de todos os registros da tabela cliente

SELECT * **FROM** restaurante.pessoa;



Retorna todos os campos da tabela cliente (id, nome, email e data_nasc) de todos os registros

FILTRANDO REGISTROS

PROJETO FÍSICO - BANCO DE DADOS

- Para filtrar os registros que serão mostrados em uma instrução **SELECT**, utilizamos a cláusula **WHERE**.
 - Algumas opções de condicionais são: igualdade (=), maior que(>), maior ou igual a (>=), menor que (<), menor ou igual a (<=).
- EXEMPLO:

```
SELECT id, nome, data_nasc FROM restaurante.pessoa  
WHERE data_nasc >= '2000-01-01';
```



Retorna os valores dos campos id, nome e data_nasc dos registros da tabela cliente cuja campo data_nasc for maior ou igual que '2000-01-01'

ALTERANDO REGISTROS

PROJETO FÍSICO - BANCO DE DADOS

- Para realizar a alteração de um ou mais campos em determinado registro (ou conjunto de registros) usamos a cláusula **UPDATE**.
- EXEMPLO:

UPDATE restaurante.pessoa **SET** nome = 'Jorge Augusto dos Santos',
cpf = '444444444444' **WHERE** pessoa.id = 2;



Modifica o nome e o email do cliente que possui o id igual a 2

APAGANDO REGISTROS

PROJETO FÍSICO - BANCO DE DADOS

- Para apagar um ou mais registros de uma tabela usamos a cláusula **DELETE**.
- EXEMPLO:

```
DELETE FROM restaurante.consumo WHERE cliente_id = 1 AND  
data_criacao = '2022-07-14';
```



Deleta o(s) registro(s) da tabela comanda cujos campos **cliente_id** e **data_criacao** sejam 1 e '2022-07-14', respectivamente. Caso exista mais de um registro com essas especificações, ambos serão apagados.

ATIVIDADE 2

PROJETO FÍSICO - BANCO DE DADOS

- Criação do **SCRIPT** de manipulação de dados no banco de dados do restaurante.
 - A. Inserir no mínimo um registro em cada tabela presente no banco de dados.
 - B. Realizar consultas simples nas tabelas.
 - I. Consultas que acessem somente uma tabela.
 - II. Consultas que exijam a filtragem (**WHERE**).
 - C. Cria quatro instruções de modificação dos dados nas tabelas (**UPDATE**).