



INSTITUTO FEDERAL  
PIAUÍ  
Campus Parnaíba

# SQL

Prof. Msc Denival A. dos Santos

# Histórico

- A versão original foi desenvolvida pela IBM no laboratório de pesquisa de San José;
- Originalmente chamada de **Sequel**, foi implementada como parte do projeto do **Sistema R** no início dos anos 70;
- Inúmeros produtos dão suporte atualmente para a linguagem SQL;
- SQL - Linguagem de consulta estruturada;
- A linguagem SQL pode ser considerada uma das maiores razões para o sucesso dos bancos de dados relacionais no mundo comercial;
- Um esforço conjunto da ANSI (American National Standards Institute - Instituto nacional americano de padrões) e a ISO (International Standards Organization - Organização Internacional de padrões) chegou a versão-padrão da SQL (ANSI, 1986), chamada SQL-86 ou SQL1;
- Como a especificação do padrão SQL está em expansão, com mais funcionalidades a cada versão, o último padrão é a **SQL-99** e **SQL-2003**.

# Partes da Linguagem SQL

- **Linguagem e definição de dados (DDL)** - proporciona comandos para a definição de esquemas de relação, exclusão de relações, criação de índices e modificação nos esquemas de relações;
- **Linguagem interativa de manipulação de dados (DML)** - Abrange uma linguagem de consulta baseada tanto na álgebra relacional quanto no cálculo relacional de tuplas. Engloba também comandos para inserção, exclusão e modificação de tuplas no banco de dados;
- **Incorporação DML** - foi projetada para aplicações e linguagens de programação de uso geral, como PL/I, Delphi, Java, C#, C++, etc.
- **Definição de visões** - comandos para definição de visões;
- **Autorização** - comandos para especificação de direitos de acesso a relações e visões;
- **Integridade** - comandos para especificação de regras de integridade que os dados que serão armazenados no banco de dados devem satisfazer;
- **Controle de transações** - comandos para a especificação de inicialização e finalização de transações.

# DDL - Linguagem de definição de dados

- O conjunto das relações em um banco de dados deve ser especificado para o sistema por meio de uma linguagem de definição de dados (DDL);
- A **SQL DDL** permite não só a especificação de um conjunto de relações, como também informações acerca de uma das relações, incluindo:
  - ☐ O esquema de cada relação;
  - ☐ O domínio dos valores associados a cada atributo;
  - ☐ As regras de integridade;
  - ☐ O conjunto de índices para manutenção de cada relação;
  - ☐ Informações sobre segurança e autoridade sobre cada relação, etc.
- Principais comandos
  - ☐ **Create** - criar base de dados ou tabelas;
  - ☐ **Drop** - remove base de dados ou tabelas;
  - ☐ **Alter** - altera a estrutura de uma tabela existente.

# Base de Dados

- Para criarmos uma base de dados no MySQL Server, utilizamos:

```
mysql> CREATE DATABASE livraria;  
Query OK, 1 row affected (0.02 sec)
```

- Para seleccionar uma bases de dados existente, utilizamos USE:

```
mysql> USE livraria;  
Reading table information for completion of table and column names  
You can turn off this feature to get a quicker startup with -A  
  
Database changed
```

# Base de Dados

- Para listar as bases de dados existentes, utilizamos:

```
mysql> show databases;
+-----+
| Database                |
+-----+
| information_schema      |
| livraria                |
| mysql                   |
| test                    |
+-----+
4 rows in set (0.03 sec)
```

- Para remover uma base de dados existente, utilizamos:

```
mysql> DROP DATABASE livraria;
Query OK, 0 rows affected (0.08 sec)
```

# Definição de esquema em SQL

- Definimos uma relação SQL usando o comando **CREATE TABLE**.
- Sua sintaxe é:

```
CREATE TABLE nome_tabela (  
    nome_atributo_1 tipo_1 [[NOT]NULL][UNIQUE]  
    [{, nome_atributo_n      tipo_n}]  
    [, PRIMARY KEY (nome(s)_atributo(s))]  
    [{, FOREIGN KEY (nome_atributo)  
        REFERENCES nome_tabela}]  
)
```

# Tipos de domínios em SQL

- O padrão SQL-92 aceita uma variedade de tipos de domínios embutidos, incluindo os seguintes:
  - **Char(n)** - é uma cadeia de caracteres de tamanho fixo, com tamanho n definido pelo usuário;
  - **Varchar(n)** - é uma cadeia de caracteres de tamanho variável, com o tamanho n máximo definido pelo usuário;
  - **Int** - é um número inteiro;
  - **Smallint** - é um número inteiro pequeno;
  - **Numeric(p,d)** - é um número de ponto fixo cuja precisão é definida pelo usuário;
  - **Date** - é um calendário contendo um ano (dia, mês e ano);
  - **Time** - representa horário (hora, minuto e segundo)

Obs.: **Timestamp** - engloba os campos DATE e TIME

**Varchar2(n)** - utilizado em ORACLE



# Definição de esquema em SQL

- Para criarmos uma tabela no MySQL Server, utilizamos:

```
MariaDB [livraria]> create table livro(  
-> idLivro int not null,  
-> titulo varchar(100),  
-> preco numeric(9,2),  
-> primary key(idLivro)  
-> );  
Query OK, 0 rows affected (0.44 sec)
```

- Para listar as tabelas existentes, utilizamos:

```
mysql> SHOW TABLES;  
+-----+  
| Tables_in_livraria |  
+-----+  
| Livro               |  
+-----+  
1 row in set (0.00 sec)
```

- Para remover uma tabela existente, utilizamos:

```
mysql> DROP TABLE Livro;  
Query OK, 0 rows affected (0.00 sec)
```

# Exemplo

```
MariaDB [livraria] > create database hospital;
Query OK, 1 row affected (0.00 sec)

MariaDB [livraria] > use hospital;
Database changed
MariaDB [hospital] > create table ambulatorio(
    -> idAmbulatorio int not null,
    -> andar int not null,
    -> capacidade int,
    -> primary key(idAmbulatorio)
    -> );
Query OK, 0 rows affected (0.26 sec)

MariaDB [hospital] > create table medico(
    -> idMedico int not null,
    -> nome varchar(70) not null,
    -> idade int,
    -> especialidade varchar(40),
    -> idAmbulatorio int not null,
    -> primary key(idMedico),
    -> foreign key(idAmbulatorio) references ambulatorio(idAmbulatorio)
    -> );
Query OK, 0 rows affected (0.21 sec)

MariaDB [hospital] > show tables;
+-----+
| Tables_in_hospital |
+-----+
| ambulatorio         |
| medico              |
+-----+
2 rows in set (0.00 sec)
```

# Modificação na estrutura de Tabelas

- As definições de uma tabela básica ou de outros elementos do esquema que possuírem denominação poderão se alteradas pelo comando **ALTER**.
- Para as tabelas básicas, as ações de alteração possíveis compreendem: adicionar ou remover uma coluna(atributo), alterar a definição de uma coluna e adicionar ou eliminar restrições na tabela.
- Sintaxe:

**ALTER TABLE** nome\_tabela **RENAME** [novo\_nome]

**ADD [COLUMN]** nome\_atributo\_1 tipo\_1 [{RIs}]

[{, nome\_atributo\_n tipo\_n [{RIs}]}] |

**MODIFY [COLUMN]** nome\_atributo\_1 tipo\_1 [{RIs}]

[{, nome\_atributo\_n tipo\_n [{RIs}]}] |

**DROP COLUMN** nome\_atributo\_1 [{, nome\_atributo\_n }] |

**CHANGE** [nomeAntigo novoNome tipo\_1] |

**[ADD | DROP] [PRIMARY KEY ... | FOREIGN KEY ...]**

# Modificação na estrutura de Tabelas

- Para modificar a estrutura de uma tabela no MySQL Server, utilizamos:

```
mysql> ALTER TABLE Livro RENAME livros;  
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> ALTER TABLE Livro ADD paginas INTEGER;  
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> ALTER TABLE Livro DROP COLUMN paginas;  
Query OK, 0 rows affected (0.00 sec)
```

# Modificação na estrutura de Tabelas

**ALTER TABLE** Ambulatórios  
**ADD** nome VARCHAR(30)

**ALTER TABLE** Médicos  
**DROP PRIMARY KEY**

**ALTER TABLE** Pacientes  
**DROP COLUMN** doenca

**ALTER TABLE** Funcionários  
**ADD FOREIGN KEY**(nroa) **REFERENCES** Ambulatórios(nroa)

**ALTER TABLE** Funcionarios  
**ADD CONSTRAINT** fk\_nroa **FOREIGN KEY**(nroa) **REFERENCES** Ambulatorios(nroa)

**ALTER TABLE** Cidade  
**DROP FOREIGN KEY** fk\_cliente

**ALTER TABLE** Pacientes  
**MODIFY COLUMN** nome\_pac varchar(50)

# Campos auto incremento

- O auto incremento permite que um número único seja gerado quando um novo registro é inserido em uma tabela.
- Em MYSQL trata-se da palavra chave AUTO\_INCREMENT, cujo valor inicial padrão é 1, e se incrementa de 1 em 1.

```
create table cor (  
    id int not null auto_increment,  
    nome varchar(35) not null,  
    primary key(id)  
)
```

# Restrições

## Restrição de valores nulos

- A SQL permite que a declaração de domínio de um atributo inclua a especificação de **not null**, proibindo, assim, a inserção de valores nulos para esse atributo;
- Qualquer modificação que possa resultar na inserção de um valor nulo em um domínio em um domínio **not null** gera um diagnóstico de erro;
- Há muitas situações em que a proibição de valores nulos é desejável. Um caso em particular no qual é imprescindível a proibição de valores nulos é uma **chave primária** de um esquema de relação.

```
Create table Pessoa(  
    cpf varchar(11) not null, // Não aceita valor nulo.  
    nome varchar(50) null,    // Aceita valor nulo.  
    endereco varchar(50),    // Por default aceita valor nulo.  
    primary key(cpf)  
)
```

# Restrições

## Restrição de valores duplicados

- Há situações onde o valor armazenado em um campo deve ter um registro em relação aos outros registros da tabela.
- Para isso utilizamos a cláusula **UNIQUE**.

```
create table pessoa (  
    id int not null,  
    nome varchar(60) not null,  
    cpf varchar(11) unique,  
    endereco varchar(60),  
    primary key(id)  
)
```



# Restrições

## Restrição de chave primária

- A chave primária tem como função identificar univocamente uma linha do registro da tabela.
- Toda tabela deve possuir um campo chave, e quando ele é definido, ficam implícitas as cláusulas **UNIQUE** e **NOT NULL** para este campo, não sendo necessário a especificação delas.
- Da mesma forma, a cláusula NULL fica implícita quando não se digita nada.

```
Create table Pessoa(  
    cpf varchar(11) not null,  
    nome varchar(50) null,  
    endereco varchar(50),  
    primary key(cpf) // Define o cpf como chave primária  
)
```

# Restrições

## Restrição de chave estrangeira

- A chave estrangeira (*Foreign key*) especifica que o valor do atributo deve corresponder a algum valor existente em um atributo de outra entidade. A chave estrangeira mantém a integridade referencial entre duas entidades relacionadas. Ela é a chave de uma relação 1 para muitos onde precisa-se de uma chave de identificação da tabela pai na tabela filho.
- **Observação:** No MySQL uso de FOREIGN KEYS só é suportado pelo *engine InnoDB e MariaDB*.

```
create table pai(  
  idPai int not null,  
  nome varchar(60),  
  primary key(idPai)  
)
```


```
create table filho(  
  idFilho int not null,  
  nome varchar(60),  
  idPai int not null,  
  primary key(idFilho),  
  foreign key(idPai) references Pai(idPai)  
)
```

# Restrições

## Restrição de valores padrão

- Pode-se declarar, ao criar uma tabela, que determinado campo já venha com um valor padrão, isso significa que, se nada for inserido nesse campo, ele terá aquele valor padrão armazenado - **DEFAULT**.

```
-- Criação da tabela
Create table Cidade(
    id_cidade int not null,
    nome varchar(50) null,
    uf varchar(2) Default "PI",
    primary key(id_cidade)
)
-- Povoar a tabela
insert into Cidade values(1,"Timon","MA");
insert into Cidade(id_cidade, nome) values(2,"Parnaíba");
-- Saída
```

 id_cidade	nome	uf
1	Timon	MA
2	Parnaíba	PI