

# BANCO DE DADOS

Linguagem SQL – DDL

# LINGUAGEM SQL - DDL

A linguagem SQL é um padrão de linguagem de consulta de Banco de Dados que usa uma combinação de construtores em Álgebra e Cálculo Relacional e possui as seguintes características principais:

- Linguagem de definição de dados (DDL)
- Linguagem interativa de manipulação de dados (DML)
- Definição de Visões • Autorização de usuários
- Integridade de dados
- Controle de Transações

02

# LINGUAGEM SQL - DDL

Segundo Machado (2008), a primeira versão da linguagem SQL recebeu o nome de SEQUEL, entre 1976 e 1977, por razões jurídicas e depois de revisada, teve seu nome alterado para SQL. A IBM tinha um projeto grandioso chamado System R e foi logo colocado em execução e surgiram novas alterações na linguagem.

03

# LINGUAGEM SQL - DDL

Houve um sucesso grande com a nova maneira para realização de consulta em banco, e a utilização da linguagem SQL aumentou. Dessa forma os SGBD's começaram a adquirir a linguagem como padrão DB2 da IBM , Oracle da Oracle Corporation, Sybase da Sybase Inc., Microsoft SQL Server da Microsoft, etc. Dessa forma a SQL tornou-se um padrão em bancos relacionais, faltava apenas tornar-se de direito. Em 1982, o American National Standard Institute (ANSI) tornou a SQL o padrão oficial de linguagem em ambiente relacional sendo hoje utilizada na grande maioria dos sistemas de Bancos de Dados relacionais, tais como MySQL, DB2, SQLServer.

04

# SISTEMA GERENCIADOR DE BANCO DE DADOS

Estudaremos os comandos existentes na linguagem SQL:

- Comandos DDL (Data Definition Language): Conjunto de comandos responsáveis pela criação, alteração e deleção da estrutura das tabelas e índices de um sistema.
- Comandos DML (Data Manipulation Language): Conjunto de comandos responsáveis pela consulta e atualização dos dados armazenados em um banco de dados.

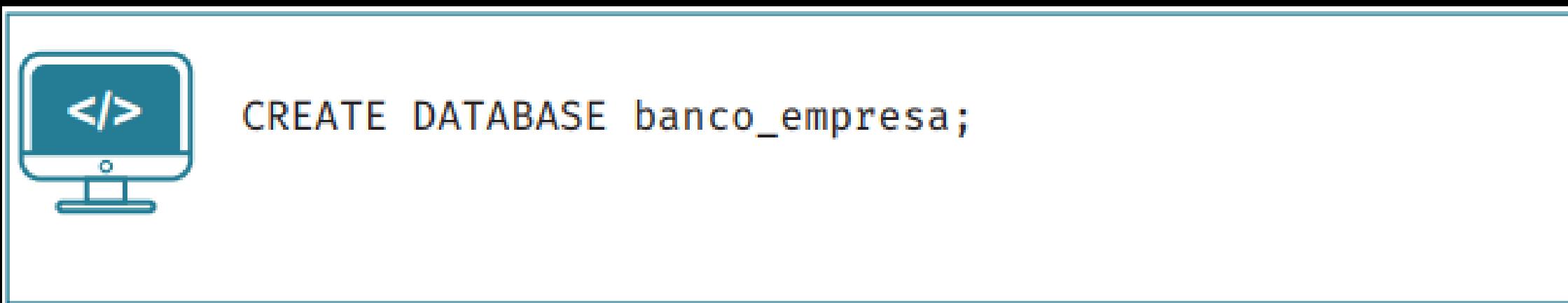
# COMANDO DDL

Os comandos da DDL SÃO:

- CREATE (criação de estrutura),
- ALTER (alterar estrutura)
- DROP (permite remover ou excluir uma estrutura).

# COMANDO CREATE

O Primeiro passo, para criar seu Banco de Dados é cria-lo! Vejamos o exemplo de como criar um banco de dados:



Ao digitar o código acima no MySQL, ele irá criar uma base de dados chamada **banco\_empresa**.

Uma observação **IMPORTANTE** é que um código só será executado e finalizado quando encontrar o ponto-e-vírgula (;).

# COMANDO DDL

Antes de criar as tabelas propriamente ditas devemos dar o comando:



```
USE banco_empresa;
```

Com o comando USE, estamos selecionando o Banco de Dados que iremos criar nossas tabelas.

# COMANDO CREATE

Após criar o banco de dados, ele não terá nenhuma tabela com atributos, portanto, o próximo passo é criar as tabelas e suas relações. Veremos agora a sintaxe do comando Create Table:

```
CREATE TABLE <nome-tabela>(<nome-coluna> <tipo-dado> [NOT  
NULL],  
  
PRIMARY KEY (<nome-coluna-chave>),  
FOREIGN KEY (<nome-coluna-chave-estrangeira>) REFERENCES  
<nome-tabela-origem> ON DELETE [RESTRICT]  
[CASCADE]  
[SET NULL]);
```

# COMANDO CREATE

- < nome-tabela> - Representa o nome da tabela que será criada.
- < nome-coluna> - Será representado o nome dos campos que serão criados e deve ser colocado um campo após o outro, separado por vírgula.
- < tipo-do-dado> - Define o tipo e tamanho dos campos definidos para a tabela.
- NOT NULL – para chave primária, essa clausula é obrigatória, pois indica que esse atributo deve ter um conteúdo
- NOT NULL WITH DEFAULT - Preenche o campo com valores pré-definidos, de acordo com o tipo do campo, caso não seja especificado o seu conteúdo no momento da inclusão do registro.

Os valores pré-definidos são:

- Campos numéricos - Valor zero.
- Campos alfanuméricos - Caractere branco.
- Campo formato Date - Data corrente.
- Campo formato Time - Horário no momento da operação.

# COMANDO CREATE

- PRIMARY KEY (nome-coluna-chave-primária): Define qual atributo será a chave primária da tabela. Caso ela tenha mais de uma coluna como chave, elas deverão ser relacionadas entre os parênteses.
- FOREIGN KEY (nome-coluna-chave-estrangeira) REFERENCES (nome-tabela origem): serve para definição de chaves estrangeiras, ou seja, os campos que são chaves primárias de outras tabelas. Na opção REFERENCES devemos escrever o nome da tabela de onde veio a chave estrangeira, ou seja, a tabela de origem onde o campo era a chave primária (HEUSER,2004).

# COMANDO CREATE

- ON DELETE – caso haja exclusão de um registro na tabela de origem e existe um registro correspondente nas tabelas filhas essa opção diz os procedimentos que devem ser feitos pelo SGBD.

As opções disponíveis são:

- RESTRICT - Opção default (padrão). Não permite a exclusão na tabela de pai (tabela de origem) de um registro, cuja chave primária exista em alguma tabela filha.
- CASCADE – Realiza a exclusão em todas as tabelas filhas que possuam o valor da chave que será excluída na tabela pai.
- SET NULL - Atribui o valor NULO nas colunas das tabelas filha que contenham o valor da chave que será excluída na tabela pai.

# COMANDO CREATE

- ON UPDATE – caso haja atualização na chave primária de um registro na tabela de origem e existe um registro correspondente nas tabelas filhas essa opção diz os procedimentos que devem ser feitos pelo SGBD.

As opções disponíveis são:

- RESTRICT
- CASCADE
- SET NULL

# COMANDO CREATE

Para facilitar a visualização de cada uma das nomenclaturas vejamos o exemplo para criar uma tabela de um funcionário:

```
CREATE TABLE funcionario (
    Cod_func INT [NOT NULL],
    nome VARCHAR (20)

    PRIMARY KEY (Cod_func)

    FOREIGN KEY (cod_dept) REFERENCES departamento
    ON DELETE [RESTRICT]);
```

# COMANDO CREATE

Para o modelo relacional de uma empresa em que temos um ou vários funcionários, que podem ser alocados em vários projetos (Relação (N:M)) temos o seguinte diagrama:

Funcionário	
<u>CODIGO_func</u>	<u>NOME_func</u>
01	João
02	Maria

É ALOCADO	
<u>CODIGO_FUNC</u>	<u>CODIGO_PROJ</u>
01	100
02	200

Projeto	
<u>CODIGO_proj</u>	<u>NOME_proj</u>
100	VIVA
200	NATUREZA

# COMANDO CREATE

Para criar a tabela Funcionário usamos o seguinte código:



```
CREATE TABLE Funcionario (
    CODIGO_func INTEGER NOT NULL AUTO_INCREMENT,
    NOME_FUNC VARCHAR(20) , PRIMARY KEY(CODIGO_func));
```

# COMANDO CREATE

Para criar a tabela Funcionário usamos o seguinte código:



```
CREATE TABLE Funcionario (
    CODIGO_func INTEGER NOT NULL AUTO_INCREMENT,
    NOME_FUNC VARCHAR(20) , PRIMARY KEY(CODIGO_func));
```

# COMANDO CREATE

Para criar a tabela Alocação usamos o seguinte código:



```
CREATE TABLE Alocacao(
    CODIGO_func INTEGER NOT NULL,
    CODIGO_proj INTEGER NOT NULL,
    FOREIGN KEY(CODIGO_func) REFERENCES
    Funcionario(CODIGO_func),
    FOREIGN KEY(CODIGO_proj) REFERENCES Projeto(CODIGO_
    proj));
```

# COMANDO CREATE

Para criar a tabela Projeto usamos o seguinte código:



```
CREATE TABLE Projeto (
    CODIGO_proj INTEGER NOT NULL AUTO_INCREMENT,
    NOME_proj VARCHAR(30) , PRIMARY KEY(CODIGO_proj));
```

# COMANDO CREATE

Para o modelo relacional de uma banda e músico (cuja cardinalidade é 1:N) conforme mostra figura abaixo, temos as tabelas:

Banda
Codigo_banda
nome_banda
02

Musico
Codigo_musico
Codigo_banda
Nome_musico
tempo_experiencia

# COMANDO CREATE

Para criar a tabela Projeto usamos o seguinte código:

```
CREATE TABLE Banda (
   Codigo_banda INTEGER NOT NULL AUTO_INCREMENT,
   Nome_banda VARCHAR(20) NULL,
   PRIMARY KEY(Codigo_banda));
```

```
CREATE TABLE Musico (
   codigo_musico INTEGER NOT NULL AUTO_INCREMENT,
   Codigo_bandaFK INTEGER NOT NULL,
   nome_musico VARCHAR(20),
   tempo_experiencia VARCHAR(20),
   PRIMARY KEY(codigo_musico),
   FOREIGN KEY (Codigo_bandaFK)
    REFERENCES Banda (Codigo_banda));
```

# COMANDO ALTER

Outro comando da linguagem DDL é o comando ALTER. Com esse comando podemos alterar a estrutura de uma tabela, por exemplo, adicionar mais atributos a uma tabela (Entidade). Essa clausula permite acrescentar modificar e alterar nomes e formatos de colunas de tabelas.

```
ALTER TABLE <nome-tabela>
ADD <nome-coluna> <tipo-do-dado>

ALTER TABLE <nome-tabela>
MODIFY <nome-coluna> <tipo-do-dado> [NULL]
```

# COMANDO CREATE

Onde cada campo representa:

- nome-tabela - nome da tabela que será atualizada.
- nome-coluna - nome da coluna que será criada.
- tipo-do-dado - tipo e tamanho dos campos definidos para a tabela.
- ADD - inclusão de uma nova coluna na estrutura da tabela. Na coluna correspondente a esse campo já existente será preenchido o valor NULL (Nulo).
- MODIFY - Permite a alteração na característica da coluna especificada.

# COMANDO ALTER

Vamos supor que a empresa precisa do endereço de todos os funcionários. Na tabela Funcionário trabalhada logo acima temos os atributos (CÓDIGO\_func, NOME\_func) precisamos adicionar o atributo endereço na tabela, digitando o Código abaixo:



```
ALTER TABLE Funcionario  
ADD endereço char(20);
```

# COMANDO ALTER

necessite alterar o tipo de algum atributo de sua tabela, por exemplo, alterar o endereço para varchar(30), então o código seria esse:



```
ALTER TABLE Funcionario MODIFY endereço VARCHAR(30) NOT NULL;
```

# COMANDO ALTER

E para alterar o nome do atributo, por exemplo, trocar endereço por endereço\_completo, utilizariammos o seguinte comando:



```
ALTER TABLE Funcionario CHANGE endereço endereço_completo  
varchar(30) NOT NULL;
```

# COMANDO DROP

Finalizando, para deletar o endereço, ou seja, a empresa não necessita guardar os dados de endereço em seu banco de dados, utilizariamos o seguinte código:



```
ALTER TABLE Funcionario DROP endereço;
```

# COMANDO DROP

E a ultima cláusula da linguagem DDL que iremos abordar aqui é o comando que permite deletar a estrutura da tabela do Banco de Dados, ou seja, com esse comando a tabela inteira será excluída e eliminada! Esse é o comando **DROP TABLE**.

Onde:

nome-tabela - Representa o nome da tabela que será deletada.

Exemplos do comando DROP TABLE, utilizando as tabelas desta Unidade.

- Drop Table alocacao;
- Drop Table banda;
- Drop Table funcionario;
- Drop Table musico;
- Drop Table projeto;

# COMANDO DROP

**Utilize o comando **DROP**, apenas quando revisar sua tabela e tiver certeza sobre a exclusão dos dados!**