

# Banco de Dados

Prof. Ms. Ricardo Alexandre Bontempo

## Mini Curriculum



**Prof. Ms. Ricardo  
Alexandre Bontempo**

Possui graduação Superior em Tecnologia em Informática - Uniclár - União das Faculdades Claretianas (2001)

Tem Mestrado Interdisciplinar em Educação, Adm. e Comunicação pela Universidade São Marcos (2006).

Foi professor na Universidade Anhembí Morumbi, além também de ser Coordenador na Graduação de Design Digital e na Pós-Graduação no curso de Ilustração, Infografia e Motion Graphics,

Já lecionou na Universidade São Judas no curso de Administração, com as matérias de Marketing, Sistemas de Informação, Informática Básica e Estatística,

Já lecionou também na FAM com as matérias de Design Gráfico, Direção de Arte, Linguagem Visual, Construção de Marcas, Fundamentos de Marketing, Pesquisa de Mercado, Comunicação Visual e Mídias.

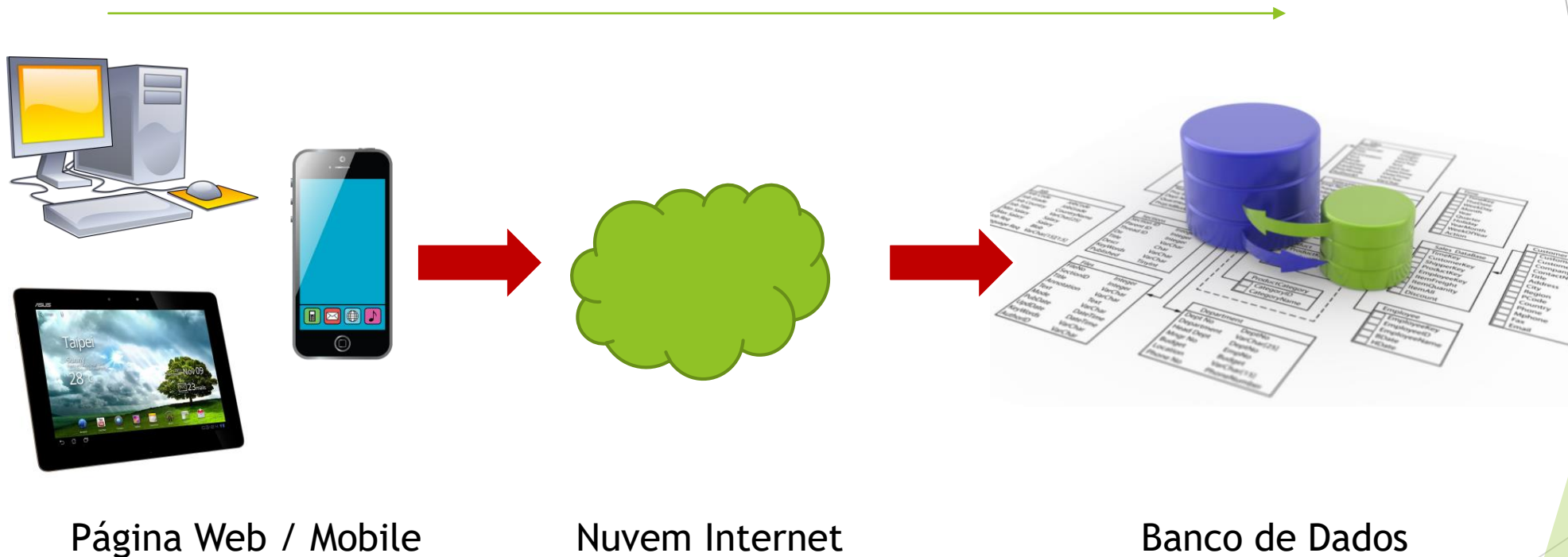
Também Lecionou e foi Coordenador na Universidade Anhanguera nos Cursos de TI e Sistemas de Informação.

Atualmente é professor Monitor no Senac de SBC nas área de Tecnologia.

Além da área acadêmica é consultor na área de análise de sistemas e programação como freelancer tendo ainda a oportunidade de prestar serviços em AS400 para IBM e tratamento de imagens de revistas para a Editora Abril.

# Banco de Dados no Ambiente Web

**Front End** - Em ciência da computação, front-end, interface frontal ou parte frontal



**Back End** - , parte secundária, parte de suporte ou parte de retaguarda são termos generalizados que se referem às etapas inicial e final de um processo.

# Banco de Dados

# Introdução ao Banco de Dados

Bancos de dados ou bases de dados são conjuntos de arquivos relacionados entre si com registros sobre pessoas, lugares ou coisas. São coleções organizadas de dados que se relacionam de forma a criar algum sentido (Informação) e dar mais eficiência durante uma pesquisa ou estudo científico.

## Linguagem SQL

SQL sigla significa Structured Query Language, que em português é traduzida como Linguagem Estruturada de Dados. Como o próprio nome diz, é uma linguagem de programação voltada para a manipulação de dados em SGBDs (Sistemas de Gerenciamento de Bancos de Dados)

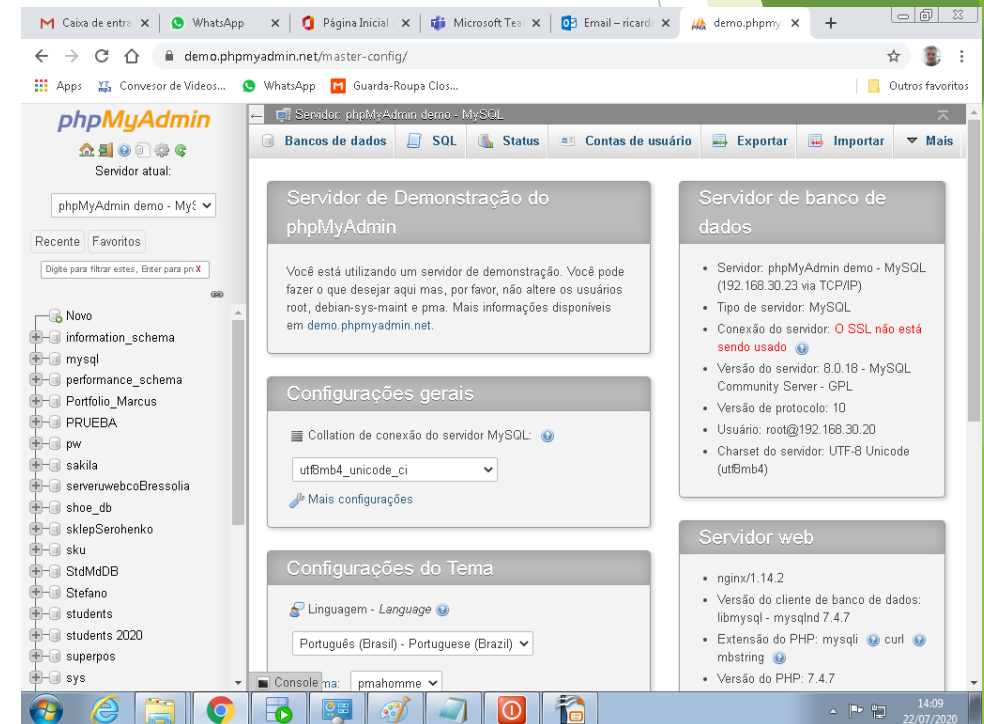


Figura 1: SGBDs (Sistemas de Gerenciamento de Bancos de Dados)

# Bancos de Dados Encontrados no Mercado

Atualmente existem vários tipos de Banco de Dados no mercado, dentre estes seguem alguns modelos abaixo:

1. Oracle
2. SQL Server
3. MySQL
4. PostgreSQL
5. DB2
6. NoSQL
7. MongoDB
8. Redis
9. Influx DB
10. Dynamo DB



ORACLE®



# Banco de Dados Relacional

Um banco de dados relacional é um banco de dados que modela os dados de uma forma que eles sejam percebidos pelo usuário como tabelas, ou mais formalmente relações.

Os bancos de dados relacionais são usados para rastrear inventários, processar transações de comércio eletrônico, gerenciar grandes quantidades de informações essenciais sobre o cliente e muito mais.

# Banco de Dados Relacional

Tabelas (prática) ou Relações (teórica)

- Tabela
- Conjunto de tuplas (linhas).
- Cada linha é composta por campos (atributos)
- Cada campo é identificado por um nome de campo (nome dos atributos)
- Conjunto de campos homônimos em todas as linhas é chamado de coluna

COLUNA				TUPLA
Código Empregado	Nome	Código Departamento	Categoria	
E1	José Silva	D1	Chefe	
E2	Paulo Santos	D2	Produção	
E3	Pedro Silveira	D2	Encarregado	
E4	Adamastor Torres	D1	Auxiliar	

VALOR DO CAMPO



# Estrutura do Banco de Dados

De forma geral estaremos utilizando banco que utiliza a linguagem SQL (Structured Query Language – Linguagem de Consulta Estruturada) como interface. Funciona sob as licenças de software livre e comercial.

- Escolhemos o Banco de Dados
- Criamos o DataBase ou Schema
- Criamos a Tabela
- Definimos os Campos
- Vinculamos os Tipos de Dados
- Definimos Chave Primária e Chave Estrangeira

# Estrutura do Banco de Dados

Elemento que identifica uma linha ou estabelece relações entre linhas de várias tabelas

Vou enfatizar as duas principais abaixo:

- Chave primária
- Chave estrangeira

# Chave Primária

Coluna ou combinação de colunas que identifica unicamente um registro em uma tabela

Exemplo:

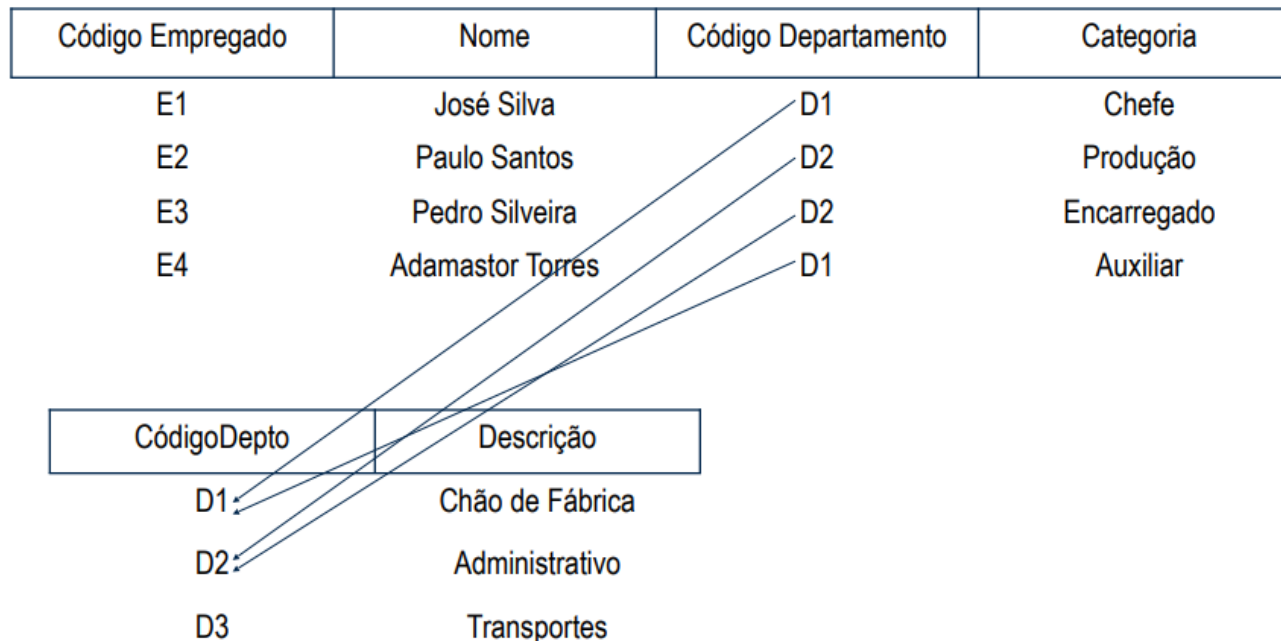
- Número de registro na tabela EMPREGADO
- Número do empregado e número do dependente na tabela DEPENDENTE
- A combinação dos campos deve ser única

Registro	Num Depend	Nome	Tipo	Nascimento
1	1	Miguel	Filho	01/12/2008
1	2	Luana	Filha	12/05/2009

Em um sistema de organização de arquivos, chave pode ser utilizado par caracterizar uma informação que será utilizada para gerar um índice, não necessariamente identifica unicamente um registro.

# Chave Estrangeira (Foreign Key)

- Uma chave estrangeira é uma informação ou conjunto de informações que referenciam informações já existentes (chaves primárias) em outras tabelas
- Em outras palavras, é através das chaves estrangeiras que são implementados os relacionamentos entre as entidades.



# Tipos de Dados

Os tipos de dados são classificados em diferentes categorias e permitem N formatos. Abaixo uma descrição de cada categoria e de cada tipo de dado do SQL Server 2005:

## - Baseados em Caracteres:

- **Char(n)** – Trata-se de um datatype que aceita como valor qualquer dígito, sendo que o espaço ocupado no disco é de um dígito por caractere. É possível utilizar até 8 mil dígitos.
- **Varchar(n)** – Também aceita como valor qualquer dígito e o espaço ocupado em disco é de um dígito por caractere. Permite usar também no máximo 8 mil dígitos. A diferença pro Char, é que o Varchar geralmente é usado quando não sei o tamanho fixo de um campo.
- **Text** – Qualquer dígito pode ser usado neste datatype, sendo ocupado 1 byte por caractere, o equivalente a **2.147.483.647** bytes.

## - Baseados em Caracteres Unicode:

- **Nchar(n)** – Neste datatype, pode usar qualquer dígito, sendo ocupados 2 bytes a cada caractere. É possível usar até 8 mil bytes.
- **Nvarchar(n)** – Igual ao tipo anterior, com a única diferença que uso esse tipo quando não sei o tamanho fixo de um campo. 2 bytes são ocupados a cada caractere. É possível usar até **8 mil** bytes.
- **Ntext** – Também aceita qualquer dígito, 2 bytes são ocupados a cada caractere. Podem ser usados até **1.073.741.823** bytes.

# Tipos de Dados

- **Baseados em Numéricos Inteiros:**
  - **Bigint** – Aceita valores entre  $-2^{63}$  e  $2^{63}-1$ , sendo que esse datatype ocupa 8 bytes.
  - **Int** – Os valores aceitos aqui variam entre  $-2^{31}$  a  $2^{31}-1$ . Ocupa 4 bytes.
  - **Smallint** – Aceita valores entre -32768 até 32767 e ocupa 2 bytes.
  - **Tinyint** – Os valores aceitos aqui variam entre 0 e 255, ocupa apenas 1 byte.
  - **Bit** – É um tipo de dado inteiro (conhecido também como booleano), cujo valor pode corresponder a **NULL**, 0 ou 1. Podemos converter valores de string **TRUE** e **FALSE** em valores de bit, sendo que TRUE corresponde a **1** e FALSE a **0**.
- **Baseados em Numéricos Exatos:**
  - **Decimal(P,S)** – Os valores aceitos variam entre  $-10^{38}-1$  e  $10^{38}-1$ , sendo que o espaço ocupado varia de acordo com a precisão. Se a precisão for de 1 a 9, o espaço ocupado é de 5 bytes. Se a precisão é de 10 a 19, o espaço ocupado é de 9 bytes, já se a precisão for de 20 a 28, o espaço ocupado é de 13 bytes, e se a precisão for de 29 a 38, o espaço ocupado é de 17 bytes.
  - **Numérico(P,S)** – Considerado um sinônimo do datatype decimal, o numérico também permite valores entre  $-10^{38}-1$  e  $10^{38}-1$  e o espaço ocupado é o mesmo do anterior.

# Tipos de Dados

## - Baseados em Numéricos Aproximados:

- **Float[(n)]** – O mesmo que **double precision** quando o valor de **n** é 53, este datatype aceita valores entre  $-1.79E + 308$  e  $1.79E + 308$ . O espaço ocupado varia de acordo com o valor de **n**. Se esse valor estiver entre 1 e 24, a precisão será de 7 dígitos, sendo que o espaço ocupado será de 4 bytes. Se o valor de **n** estiver entre 25 e 53, sua precisão será de 15 dígitos, assim sendo o espaço ocupado será de 8 bytes.
- **Real** – Este datatype é similar ao **float(n)** quando o valor de **n** é 24. Os valores aceitos variam entre  $-3.40E + 38$  e  $3.40E + 38$ . Esse datatype ocupa 4 bytes.

## - Baseados em Tipos de Dados Especiais:

- **Uniqueidentifier** – O formato hexadecimal é usado para o armazenamento de dados binários, sendo que este datatype ocupa 16 bytes.
- **Timestamp** – Um valor binário é gerado pelo SQL Server, sendo que esse datatype ocupa 8 bytes.
- **Bit** – Este datatype pode apresentar **0**, **1** ou **NULL**, como valor, sendo ocupado 1 byte. Também utilizado como um tipo de dado **int**.

# Tipos de Dados

## - Baseados em Valores Numéricos Monetários:

- **Money** – Este datatype aceita valores entre  $-2^{63}$  e  $2^{63}-1$ , sendo que 8 bytes são ocupados.
- **Smallmoney** – É possível usar valores entre  $-2^{31}$  e  $2^{31}-1$ , sendo que 4 bytes são ocupados.

## - Baseados em Data e Hora:

- **Datetime** – Permite o uso de valores entre **1/1/1753** e **31/12/9999**. Este datatype ocupa 8 bytes e sua precisão atinge **3.33** milisegundos.
- **Smalldatetime** – Aceita o uso de valores entre 1/1/1900 e 06/06/2079, sendo que sua precisão é de 1 minuto e ocupa 4 bytes em disco.



# Tipos de Dados

## - Baseados em Binários:

- **Binary[(n)]** – Este datatype representa os dados que serão usados no formato binário. O espaço ocupado é de  $n+4$  bytes, sendo que **n** pode variar entre 1 e 8000 bytes.
- **Varbinary[(n)]** – Aqui também é usado o formato binário, o espaço ocupado e a variação de **n** é igual ao anterior.
- **Image** – O formato binário também é usado aqui, sendo que o espaço ocupado é de  $2^{31}-1$  bytes ou **2.147.483.647**.

# Sintaxe dos Comandos Create DataBase e Table

**Create DataBase** = O DataBase é o repositório que iremos criar do Banco de Dados o nome do Banco da Empresa.

## Sua Sintaxe

```
Create DataBase nome_banco de dados
```

**Create Table** = O comando Create Table permite criar a tabela dentro do Banco de Dados

## Sua Sintaxe

```
CREATE TABLE Nome_da_Tabela
```

```
(
```

```
Atributo1    Tipo,
```

```
Atributo2    Tipo,
```

```
Atributo3    Tipo
```

```
)
```

**Atributos:** são os item da sua tabela. Exemplo: código, nome, e-mail, telefone, endereço.

**Tipos:** são as definições que os seus atributos recebem. Exemplo: inteiro, booleano, caracteres. Mais a frente vamos ver os tipos de dados do SQL.

# Sintaxe do Comandos Insert

**Insert** = O comando para inclusão no banco de dados, que possui a seguinte estrutura

## Sua Sintaxe

INSERT INTO nome\_tabela (lista-de-campos) VALUES (lista\_dados)

--OU

INSERT INTO nome\_tabela VALUES (lista\_dados)

- Nome\_tabela: nome da tabela no qual será inserido os dados.
- Lista-de-campos: nome das colunas que receberão os valores.
- Lista-dados: valores que serão inseridos na tabela. Estes campos devem estar na mesma ordem descrita em lista-de-campos, todos separados por vírgula. Se for utilizado um **comando SELECT** o mesmo deve retornar a mesma quantidade de colunas com os mesmos tipos de dados especificados em lista-de-campos.

# Sintaxe do Comandos Insert

Exemplo da sintaxe do comando Insert.

```
INSERT INTO EMPREGADOS(CODIGO, NOME, SALARIO, SECAO)  
VALUES(1, "HELBERT CARVALHO", 1.500, 1)  
INSERT INTO EMPREGADOS VALUES(1,"HELBERT CARVALHO",1500,1)
```

Na segunda opção foi omitida a declaração dos campos. Essa sintaxe funciona somente se for repassado valores para todas as colunas. Podemos também passar valores através de um comando SELECT, conforme abaixo:

```
INSERT INTO EMPREGADOS(CODIGO,NOME, SALARIO, SECAO)  
SELECT CODIGO,NOME,SALARIO, SECAO  
FROM EMPREGADOS_FILIAL  
WHERE DEPARTAMENTO = 2
```

# Sintaxe do Comando Select

## SELECT simples

O **comando SELECT** permite recuperar os dados de um objeto do banco de dados, como uma tabela, view e, em alguns casos, uma stored procedure (alguns bancos de dados permitem a criação de procedimentos que retornam valor). A sintaxe mais básica do comando é:

```
SELECT <lista_de_campos>  
FROM <nome_da_tabela></nome_da_tabela></lista_de_campos>
```

Exemplo:

```
SELECT CODIGO, NOME FROM CLIENTES  
SELECT * FROM CLIENTES
```

O caractere \* representa todos os campos. Apesar de prático, este caractere não é muito utilizado, pois, para o **SGBD** é mais rápido receber o comando com todos os campos explicitados.

# Sintaxe do Comando Where

## WHERE

A cláusula Where permite ao comando SQL passar condições de filtragem.

A cláusula Where permite ao comando SQL passar condições de filtragem. Veja o exemplo da **Listagem 1**.

```
SELECT CODIGO, NOME FROM CLIENTES  
WHERE CODIGO = 10;
```

```
SELECT CODIGO, NOME FROM CLIENTES  
WHERE UF = 'RJ';
```

```
SELECT CODIGO, NOME FROM CLIENTES  
WHERE CODIGO >= 100 AND CODIGO <= 500;
```

```
SELECT CODIGO, NOME FROM CLIENTES  
WHERE UF = 'MG' OR UF = 'SP';
```

# Sintaxe do Comando SELECT e o operador LIKE

## LIKE

Para busca realizar buscas parcial ou total de uma string, utilizaremos o operador like e o %

```
SELECT CODIGO, NOME FROM CLIENTES  
WHERE NOME LIKE 'MARCOS%'
```

Neste comando, todos os clientes cujos nomes iniciam com Marcos serão retornados.

```
SELECT CODIGO, NOME FROM CLIENTES  
WHERE NOME LIKE '%MARCOS%'
```

O uso de máscara no início e no fim da string fornece maior poder de busca, mas causa considerável perda de performance. Este recurso deve ser utilizado com critério.

# Sintaxe do Comando **SELECT** e o operador **LIKE**

Por padrão, a SQL diferencia caixa baixa de caixa alta. Para eliminar essa diferença, utiliza a função **UPPER**. Veja abaixo:

```
SELECT CODIGO, NOME FROM CLIENTES  
WHERE UPPER(NOME) LIKE 'MARCOS% BONTEMPO%'
```



# Sintaxe do Comando Ordenação

## ORDENAÇÃO

A ordenação pode ser definida com o comando ORDER BY. Assim como no comando WHERE, o campo de ordenação não precisa estar listado como campo de visualização.

```
SELECT CODIGO, NOME FROM CLIENTES  
ORDER BY NOME ASC
```

```
SELECT CODIGO, NOME FROM CLIENTES  
ORDER BY NOME DESC
```

# Sintaxe do Comando Select Max()

## Função Max()

A função MAX analisa um conjunto de valores e retorna o maior entre eles. No exemplo abaixo utilizamos essa função para encontrar o preço de venda mais alto:

## Sintaxe

Select Max(Campo Desejado) from Nome da Tabela

## Exemplo

```
SELECT  
  max(preco venda)  
FROM  
  produtos
```

# Sintaxe do Comando Select Min()

## Função Min()

MIN analisa um grupo de valores e retorna o menor entre eles. A seguir utilizamos essa função para conhecer o preço de venda mais baixo:

## Sintaxe

Select Min(Campo Desejado) from Nome da Tabela

## Exemplo

```
SELECT  
  min(preco venda)  
FROM  
  produtos
```

# Sintaxe do Comando Select SUM()

## Função SUM()

A função SUM realiza a soma dos valores em uma única coluna e retorna esse resultado.

### Sintaxe

Select Sum(Campo\_Desejado) from tabela desejada where condição

Para somar todos os preços de venda dos produtos de uma categoria, podemos utilizar essa função informando a coluna PreçoVenda, como exemplificado a seguir:

```
SELECT
    sum(preco venda)
FROM
    produtos
WHERE
    categoria = 1
```

# Sintaxe do Comando Select Count()

## Função Count()

A função COUNT() retorna o número de linhas que corresponde a um determinado critério.

SQL COUNT(nome\_coluna)

A função COUNT(nome\_coluna) retorna o número de valores (valores nulos não serão contados) da coluna especificada:

```
SELECT COUNT(nome_coluna) FROM nome_tabela
```

# Sintaxe do Comando Select AVG()

## Função AVG()

Com a função AVG podemos calcular a média aritmética dos valores em uma única coluna.

### Sintaxe

Select avg(Campo\_Desejado) from tabela desejada

Usamos essa função no exemplo a seguir, tomando como parâmetro a coluna PreçoVenda da tabela produtos.

```
SELECT  
    avg(precoventa)  
FROM  
    produtos
```

# Sintaxe do Comando para Junção de Tabelas

## JUNÇÃO DE TABELAS

O SELECT permite juntar duas ou mais tabelas no mesmo resultado. Isso pode ser feito de várias formas.

```
SELECT CLIENTES.CODIGO, CLIENTES.NOME, PEDIDOS.DATA  
FROM CLIENTES, PEDIDOS  
WHERE CLIENTES.CODIGO = PEDIDOS.CODCLIENTE
```

Nesta linha as tabelas relacionadas CLIENTES e PEDIDOS são unificadas através do campo chave, em uma operação de igualdade. Repare que os nomes dos campos passam a ser prefixados pelo nome das tabelas, resolvendo duplicidades.

# Sintaxe do Comando para Junção de Tabelas

Uma versão resumida desse comando pode ser como abaixo:

```
SELECT A.CODIGO, A.NOME, B.DATA, B.VALOR  
FROM CLIENTES A, PEDIDOS B  
WHERE A.CODIGO = B.CODCLIENTE
```

O uso de aliases no código SQL torna a manutenção mais simples.



# Sintaxe do Comando JOIN

## COMANDO JOIN

A junção de tabelas no comando SELECT também pode ser feita com o comando JOIN. Este comando deve ser utilizado com a palavra reservada INNER ou com a palavra OUTER:

- INNER: Semelhante ao uso do operador “=” na junção de tabelas. Aqui os registros sem correspondências não são incluídos. Esta cláusula é opcional e pode ser omitida no comando JOIN.
- OUTER: Os registros que não se relacionam também são exibidos. Neste caso, é possível definir qual tabela será incluída na seleção, mesmo não tendo correspondência. Para exemplificar, temos as tabelas abaixo:

## Sintaxe do Comando JOIN

```
SELECT A.CODIGO, A.DESCRICAO, B.DESCRICAO, B.QTD  
FROM PRODUTOS A  
INNER JOIN COMPONENTES B  
ON (A.CODIGO = B.CODPRODUTO)
```

Este comando pode ser escrito na versão resumida abaixo:

```
SELECT A.CODIGO, A.DESCRICAO, B.DESCRICAO  
FROM PRODUTOS A  
JOIN COMPONENTES B  
ON (A.CODIGO = B.CODPRODUTO)
```

## Criar as tabelas abaixo:

-- Criando a tabela Cago

```
CREATE TABLE CARGO (  
    CodCargo char(2) NOT NULL,  
    NomeCargo varchar(50) NOT NULL,  
    ValorCargo DECIMAL(4,2) NOT NULL,  
    PRIMARY KEY (`CodCargo`)  
);
```

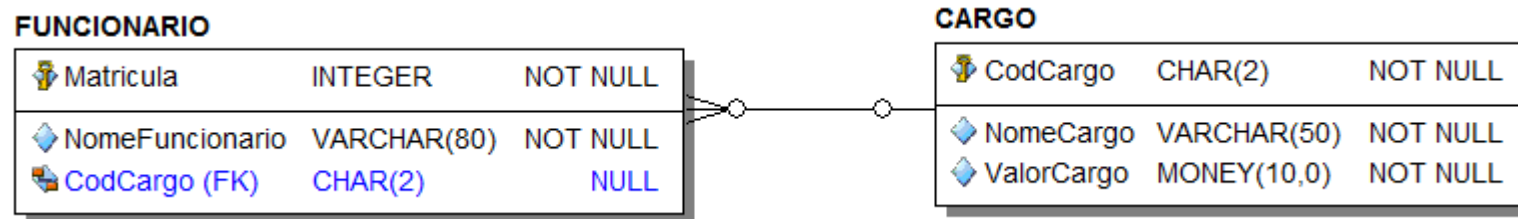
-- Criando a tabela Funcionário

```
CREATE TABLE FUNCIONARIO (  
    Matricula int(3) NOT NULL,  
    NomeFuncionario varchar(50) NOT NULL,  
    CodCargo char(2) NOT NULL,  
    PRIMARY KEY (`Matricula`),  
    FOREIGN KEY (CodCargo) REFERENCES CARGO(CodCargo)  
);
```

As questões são duas:

- 1.O que são cada uma dessas junções (joins)?
- 2.Como usar?

Para definir e exemplificar as junções acima citadas considere o modelo da figura abaixo:



Podemos notar pelo modelo que pode existir no banco de dados funcionários sem cargos e cargos sem funcionários.

Para exemplificar melhor, observe o conteúdo das tabelas na figura abaixo:

TODAS AS LINHAS....bTestes (sa (75))

```
SELECT * FROM CARGO AS C --> Apelidamos a tabelas Cargo de C neste artigo
SELECT * FROM FUNCIONARIO AS F --> Apelidamos funcionário de F neste artigo
```

Results Messages

	CodCargo	NomeCargo	ValorCargo
1	C1	CAIXA	800,00
2	C2	VENDEDOR	1200,00
3	C3	GERENTE	2400,00

CARGO AS C

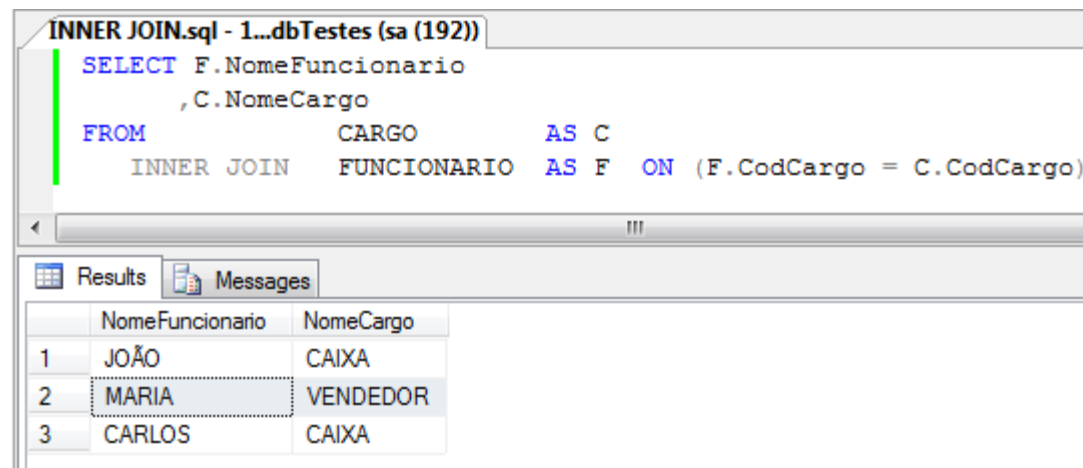
	Matricula	NomeFuncionario	CodCargo
1	100	JOÃO	C1
2	110	MARIA	C2
3	120	CARLOS	C1
4	130	TADEU	NULL

FUNCIONARIO AS F

# INNER JOIN

Quando queremos juntar duas ou mais tabelas por coincidência. Para cada linha da tabela FUNCINARIO queremos o CARGO correspondente que internamente (INNER), em seus valores de atributos, coincidam. No caso de FUNIONÁRIO e CARGO os atributos internos coincidentes são codigoCargo na tabela CARGO e codigoCargo na tabela FUNCIONARIO.

Veja também a Figura 1 e a Figura 2, lá você notará que codigoCargo é chave primária da tabela CARGO e chave estrangeira na tabela FUNCIONARIO. Para efetivarmos a junção das duas tabelas se fará necessário ligar (ON) as duas tabelas por seus atributos internos (INNER) coincidentes.



The screenshot shows a SQL query editor window titled "INNER JOIN.sql - 1...dbTestes (sa (192))". The query is as follows:

```
SELECT F.NomeFuncionario
      ,C.NomeCargo
FROM      CARGO      AS C
INNER JOIN FUNCIONARIO AS F ON (F.CodCargo = C.CodCargo)
```

Below the query, there are tabs for "Results" and "Messages". The "Results" tab is active, displaying a table with the following data:

	NomeFuncionario	NomeCargo
1	JOÃO	CAIXA
2	MARIA	VENDEDOR
3	CARLOS	CAIXA

# Sintaxe dos Comandos de Funções de Agrupamento

## FUNÇÕES DE AGRUPAMENTO

São cinco as funções básicas de agrupamento:

AVG: Retorna a média do campo especificado

```
SELECT AVG(VALOR) FROM PEDIDOS
```

MIN/MAX/SUM: Respectivamente retorna o menor valor, o maior e o somatório de um grupo de registros:

```
SELECT MIN(VALOR) FROM PEDIDOS
```

```
SELECT MAX(VALOR) FROM PEDIDOS
```

```
SELECT AVG(VALOR) FROM PEDIDOS
```

COUNT: Retorna a quantidade de itens da seleção

```
SELECT COUNT(CODIGO) FROM CLIENTES
```

# Sintaxe dos Comandos de Funções de Agrupamento

## AGRUPAMENTO

Um poderoso recurso do comando **SELECT** é o parâmetro **GROUP BY**. Através dele podemos retornar informações agrupadas de um conjunto de registros, estabelecendo uma condição de agrupamento. É um recurso muito utilizado na criação de relatórios.

```
SELECT CODCLIENTE, MAX(VLOR)  
FROM PEDIDOS  
GROUP BY CODCLIENTE
```

O comando acima retorna o maior valor de pedido de cada cliente. Observe o resultado:

```
SELECT CODCLIENTE, COUNT(*)  
FROM PEDIDOS  
GROUP BY CODCLIENTE
```

Abaixo vemos quantos pedidos foram feitos por cada cliente.



# Sintaxe do Comando UPDATE

## UPDATE

O comando para atualizar registros é UPDATE, que tem a seguinte sintaxe:

```
UPDATE nome_tabela  
SET CAMPO = "novo_valor"  
WHERE CONDIÇÃO
```

Onde:

- Nome\_tabela: nome da tabela que será modificada
- Campo: campo que terá seu valor alterado
- Novo\_valor: valor que substituirá o antigo dado cadastrado em campo
- Where: Se não for informado, a tabela inteira será atualizada
- Condição: regra que impõe condição para execução do comando

# Sintaxe do Comando DELETE

## DELETE

O comando utilizado para apagar dados é o DELETE.

DELETE FROM nome\_tabela

WHERE condição

Onde:

- Nome\_tabela: nome da tabela que será modificada
- Where: cláusula que impõe uma condição sobre a execução do comando

# Sintaxe do Comando ALTER TABLE

## ALTER TABLE

O comando Alter Table, permite fazer a alteração da estrutura de uma tabela podendo inserir ou excluir campos na estrutura da tabela.

### Excluindo colunas ALTER TABLE - DROP

```
ALTER TABLE nome-tabela  
DROP COLUMN nome-coluna;
```

Exemplo - Excluindo a coluna ID\_autor da tabela tbl\_livro:

```
ALTER TABLE tbl_livro  
DROP COLUMN ID_autor;
```

Pode-se excluir uma chave primária também:

```
ALTER TABLE tabela  
DROP PRIMARY KEY;
```

# Sintaxe do Comando ALTER TABLE

**Adicionar colunas: ALTER TABLE - ADD**

Sintaxe:

```
ALTER TABLE tabela  
ADD coluna tipo_dados constraints;
```

Exemplo no banco de dados db\_biblioteca: Vamos adicionar a coluna id\_autor na tabela tbl\_livro, com a constraint de chave estrangeira (a chave primária está na tabela tbl\_autores):

```
ALTER TABLE tbl_livro  
ADD ID_Autor SMALLINT NOT NULL;
```

# Muito Obrigado



**Prof. Ms. Ricardo Alexandre Bontempo**