

Banco de Dados Relacionais e Não Relacionais

Prof. Henrique Batista da Silva

Introdução linguagem SQL

Introdução

Junto com o modelo Relacional de Codd, foi posposto uma linguagem chamada DSL/Alpha para manipulação dos dados no modelo.

A IBM criou uma simplificação desta linguagem, chamada Square.

Aprimoramentos na Square levaram a uma linguagem chamada SEQUEL, posteriormente renomeada para SQL.

Introdução

Linguagem padrão para lidar com Banco de Dados (BD).

Praticamente todos SGBDs (Oracle, SQLServer, MySQL, etc.) que estão no mercado suportam a linguagem SQL. Muitos bancos de dados NoSQL suportam o padrão SQL Like

SQL tem uma ligação com o modelo relacional pois o resultado de uma consulta SQL é uma tabela (também chamada de conjunto resultado).

Classes de Comandos SQL

- A linguagem SQL se divide em três subgrupos:
 - Linguagem de Definição de Dados (**DDL - Data Denition Language**):
 - Usada para definição dos esquemas
 - Linguagem de Manipulação de Dados (**DML - Data Manipulation Language**):
 - Utilizada pelos usuários para manipulação dos dados (inclusão, alteração e exclusão).

Classes de Comandos SQL

- Linguagem de Controle de Dados (**DCL - *Data Control Language***):
 - Utilizada para conceder e retirar privilégios de usuários de BD em objetos de BD
- Linguagem de consulta de Dados (**DQL - *Data Query Language***):
 - Usada para recuperação de dados
- Linguagem de controle de transação (**TCL - *transaction Control Language***):
 - Utilizada para lidar com transações no banco de dados.

Classes de Comandos SQL

Por exemplo, para criar uma nova tabela é necessário utilizar o comando CREATE TABLE (definição de dados).

```
CREATE TABLE Aluno (  
    matricula INT,  
    nome VARCHAR(30),  
    CONSTRAINT pk_matricula PRIMARY KEY  
    (matricula)  
);
```

Essa instrução cria uma tabela com duas colunas, *matricula* (chave primária da tabela) e *nome*

Classes de Comandos SQL

Para popular o banco de dados é necessário o comando INSERT.

```
INSERT INTO Aluno (matricula, nome) VALUES (10, 'Maria');
```

Essa instrução adiciona um linha à tabela Aluno com o valor 10 na coluna *matricula* e o valor 'Maria' na coluna *nome*

Classes de Comandos SQL

Com o banco populado é possível recuperar valores com o comando SELECT (manipulação de dados):

```
SELECT nome  
FROM Aluno  
WHERE matricula = 10;
```



Nome
Maria

DDL (Linguagem de Definição de Dados)

Criando Tabelas no Banco

DDL (Linguagem de Definição de Dados)

- É a linguagem que define a ESTRUTURA do BD.
- Seus comandos permitem a criação, alteração e exclusão de objetos em um banco de dados.
- Seus principais comandos são:

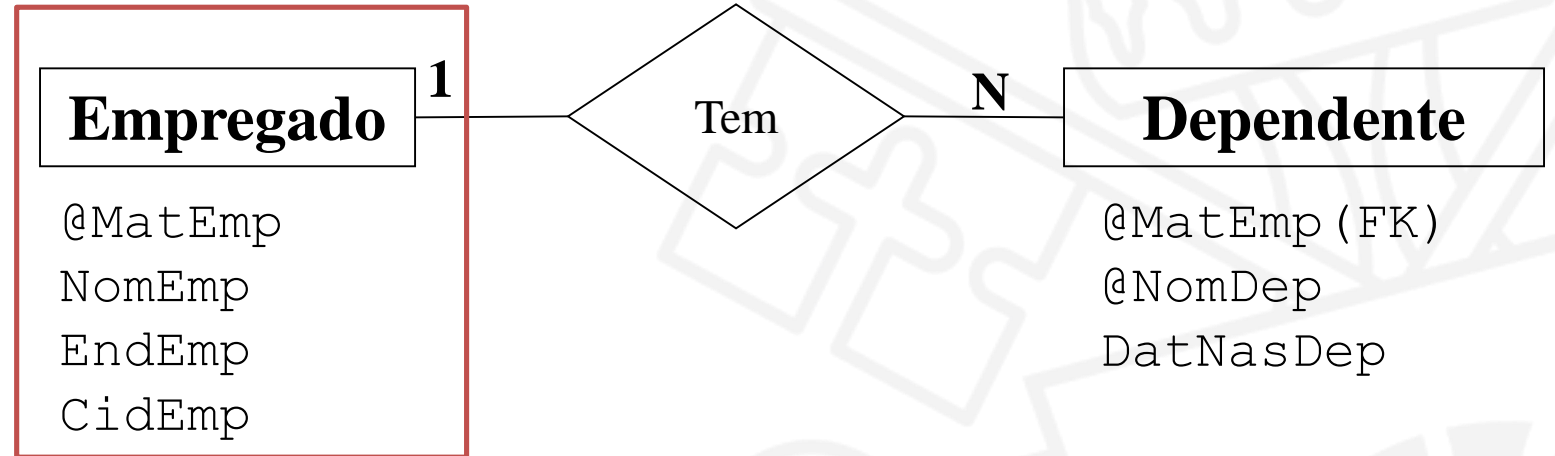
Comando	Definição	Exemplos
CREATE	Cria objetos do BD.	Criação da tabela CLIENTE
ALTER	Altera objetos do BD.	Alteração da tabela CLIENTE incluindo a coluna TELEFONE
DROP	Remove objetos do BD.	Exclusão da tabela CLIENTE

Referência: Navathe. **Sistemas de Banco de Dados** – 2011, 6ª Ed.

Comando Create table

- Comando utilizado para criar uma nova relação (tabela), dando a ela:
 - Nome
 - Especificando seus atributos (suas colunas)
 - Nome e tipo de dado (domínio).
 - Restrições do atributo (not null)
 - Especificando suas restrições (Tipo, chave).
 - Especificado depois que os atributos forem declarados.
 - Ou com o comando ALTER TABLE.

Comando Create table



Create table Empregado

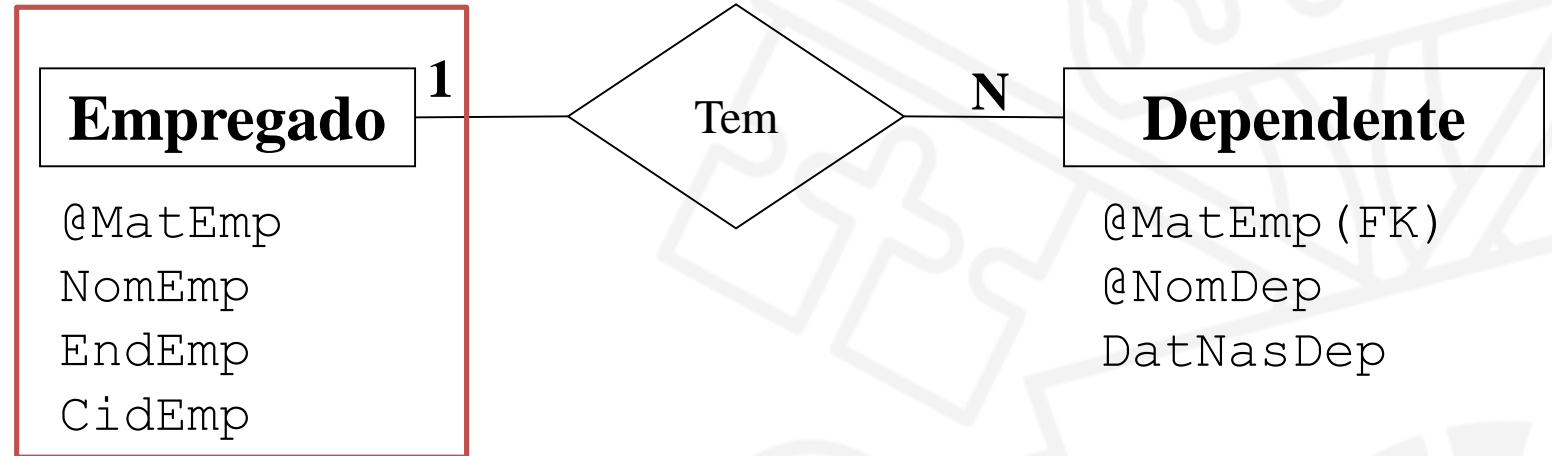
```
(  
  MatEmp smallint not null,  
  NomEmp char(30) not null,  
  EndEmp char(80) null,  
  CidEmp char(20) null,  
  Constraint PK_EMP primary Key(MatEmp)  
);
```

Comando Create table

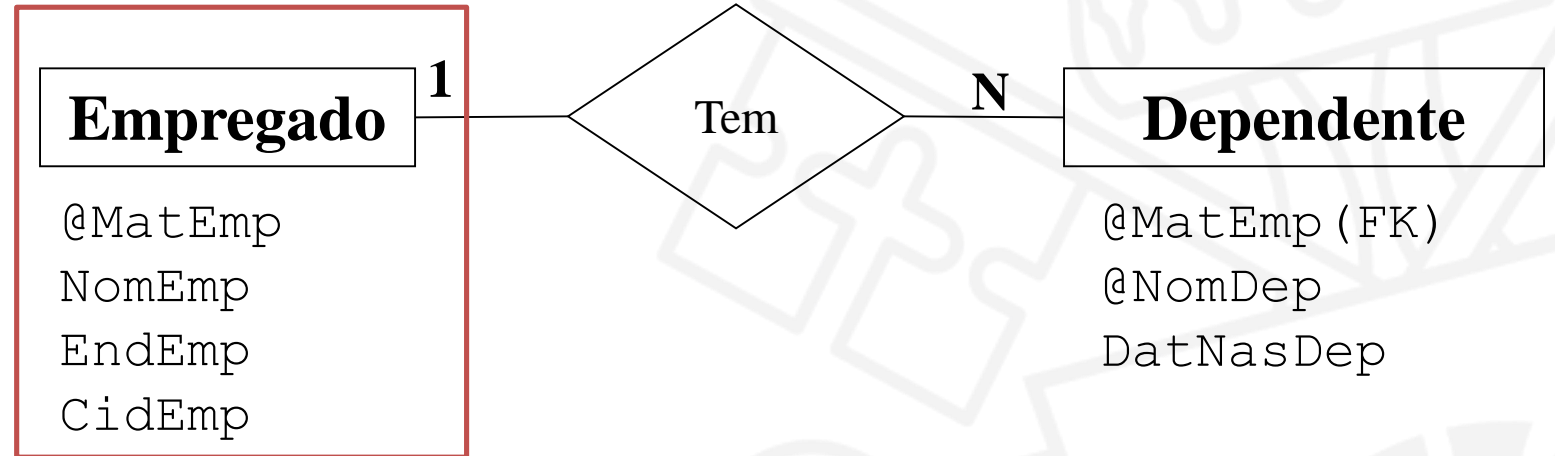
Constraint é uma restrição de chave primária, criada sobre a coluna MatEmp e recebe o nome de PK_EMP (são definidas com a cláusula PRIMARY KEY).

Create table Empregado

```
(  
  MatEmp smallint not null,  
  NomEmp char(30) not null,  
  EndEmp char(80) null,  
  CidEmp char(20) null,  
  Constraint PK_EMP primary Key(MatEmp)  
);
```



Comando Create table



Create table Empregado

```
(  
  MatEmp smallint not null,  
  NomEmp char(30) not null,  
  EndEmp char(80) null, ←  
  CidEmp char(20) null,  
  Constraint PK_EMP primary Key(MatEmp)  
);
```

Não é necessário
especificar

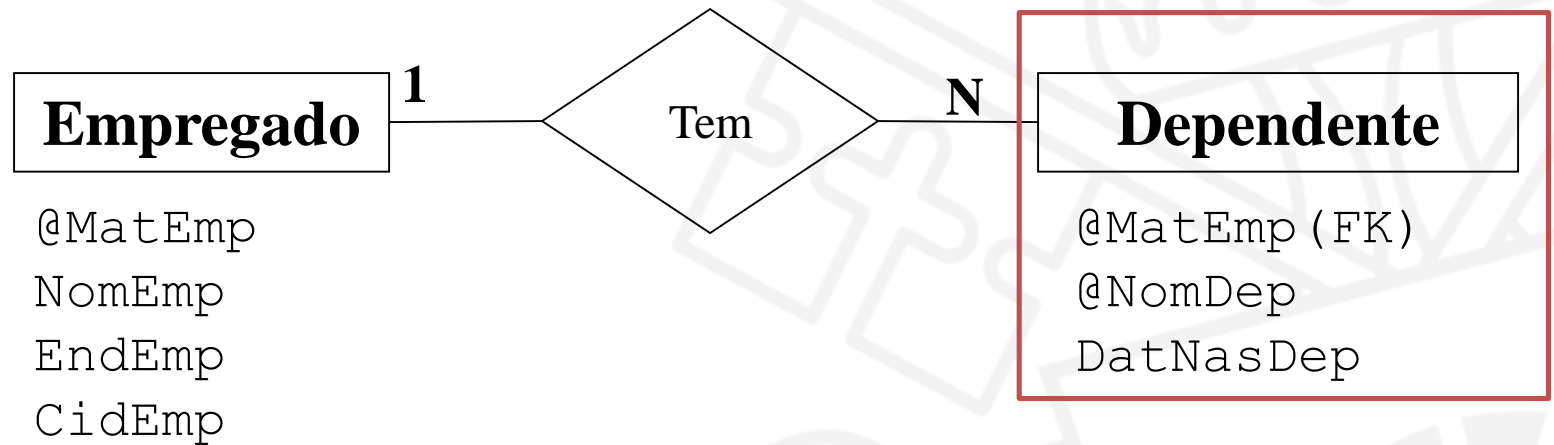
Atributo Chave Estrangeira

- Similar a instrução para criar a tabela Empregado, mas com algumas exceções:
 - Como empregados diferentes podem ter dependentes com o mesmo nome, é preciso mais do que a coluna MatEmp para garantir a unicidade, portanto a tabela tem chave primária composta de duas colunas.

Atributo Chave Estrangeira

- A tabela dependente contém outro tipo de restrição, denominada **restrição de chave estrangeira** (definida com a cláusula FOREIGN KEY).
 - Restringe os valores da coluna MatEmp na tabela Dependente para que sejam incluídos apenas valores existentes na tabela Empregado.

Atributo Chave Estrangeira



```
Create table Dependente
(
    NomDep char(30),
    DatNasDep date,
    MatEmp smallint,
    Constraint PK_DEP primary Key (MatEmp, NomDep),
    Constraint FK_EMP_DEP foreign Key (MatEmp)
        References Empregado (MatEmp)
);
```

Modificando e excluindo tabelas

Comandos para Alterações no Esquema SQL

- Alter table
 - Altera a definição de uma tabela no banco de dados
- Ações:
 - Adicionar uma coluna
 - Retirar uma coluna
 - Adicionar restrições de tabela
 - Retirar restrições de tabela

Referência: Navathe. **Sistemas de Banco de Dados** – 2011, 6ª Ed.

Comandos para Alterações no Esquema SQL

Adicionar uma Coluna

ALTER TABLE <nome da tabela> **ADD** <nome da coluna> <tipo da coluna> [<restrição do atributo>] ;

Exemplo

```
ALTER TABLE EMPREGADO ADD UF_EMP CHAR(2) NULL
```

Referência: Navathe. **Sistemas de Banco de Dados** – 2011, 6ª Ed.

Alter Table (Adicionar Coluna)

É necessário entrar com um valor para o novo atributo UF_EMP de cada tupla de EMPREGADO. Pode ser feito pela especificação da cláusula *default* ou pelo comando UPDATE.

Neste caso, se não for especificado nenhuma cláusula *default*, o novo atributo será inserido como NULL automaticamente, a restrição NOT NULL não é permitida neste caso.

Alter Table (Remover Coluna)

Remover uma coluna:

Para remover uma coluna, deve-se optar por CASCADE ou RESTRICT em termos de comportamento para eliminação.

```
ALTER TABLE <nome da tabela> DROP COLUMN <nome da coluna>
```

Exemplo

```
ALTER TABLE EMPREGADO DROP COLUMN UF_EMP
```

Alguns SGBDs exigem a **COLUMN**, após **DROP**

Referência: Navathe. **Sistemas de Banco de Dados** – 2011, 6ª Ed.

Alter Table (Modificar Restrições)

Ainda é possível modificar as restrições especificadas em uma tabela, adicionando ou eliminando uma restrição.

Eliminando uma restrição:

Para ser eliminada, a restrição precisa ter recebido um nome quando foi especificada.

```
ALTER TABLE <nome da tabela> DROP CONSTRAINT <nome> ;
```

Alter Table (Modificar Restrições)

Excluindo uma Primary Key:

```
ALTER TABLE DEPENDENTE DROP CONSTRAINT PK_DEP
```

Excluindo uma Foreign Key:

```
ALTER TABLE DEPENDENTE DROP CONSTRAINT FK_EMP_DEP
```

Alter Table (Modificar Restrições)

Adicionando uma Restrição:

É realizada pela palavra chave **ADD**, seguida da nova restrição:
Se for uma Primary Key:

```
ALTER TABLE <nome da tabela> ADD CONSTRAINT <nome>  
PRIMARY KEY (<nomes da colunas>);
```

Exemplo

```
ALTER TABLE EMPREGADO ADD CONSTRAINT PK_EMP  
PRIMARY KEY (MATEMP)
```

Referência: Navathe. **Sistemas de Banco de Dados** – 2011, 6ª Ed.

Alter Table (Modificar Restrições)

Adicionando uma Restrição:
Se for uma Foreign Key:

```
ALTER TABLE <nome da tabela> ADD CONSTRAINT <nome>  
FOREIGN KEY (<nome coluna>) REFERENCES <nome tabela>;
```

Exemplo

```
ALTER TABLE DEPENDENTE ADD CONSTRAINT FK_EMP_DEP  
FOREIGN KEY (MATEMP) REFERENCES EMPREGADO (MatEmp)
```

Obs.: para criar uma restrição, a coluna que será PK, já tem que ter sido criada anteriormente como uma coluna normal

Referência: Navathe. **Sistemas de Banco de Dados** – 2011, 6ª Ed.

DROP Table

Pode ser usado para eliminar uma tabela de um banco de dados
Sintaxe:

```
DROP TABLE <nome da tabela> ;
```

Exemplo

```
DROP TABLE DEPENDENTE
```

Referência: Navathe. **Sistemas de Banco de Dados** – 2011, 6ª Ed.

DML (Linguagem de Manipulação de Dados)

DML (Linguagem de Manipulação de Dados)

Linguagem que manipula os dados no banco.
Seus comandos permitem a recuperação, inserção, alteração e exclusão de dados.
Seus principais comandos são:

Comando	Definição	Exemplos
INSERT	Adiciona dados a uma tabela.	Inserir o João como novo cliente na tabela CLIENTE
UPDATE	Atualiza os dados de uma tabela.	Alterar o número do telefone do cliente João na tabela CLIENTE
DELETE	Exclui dados de uma tabela.	Exclui o cliente Geraldo da tabela CLIENTE

Referência: Navathe. **Sistemas de Banco de Dados** – 2011, 6ª Ed.

Comando INSERT

- É usado para adicionar uma única tupla (linha) em uma relação (tabela).
- Existem três componentes principais no comando INSERT:
 - O nome da tabela na qual serão adicionado os dados.
 - Os nomes das colunas que serão populadas dentro da tabela.
 - Os valores que serão usados para popular as colunas.

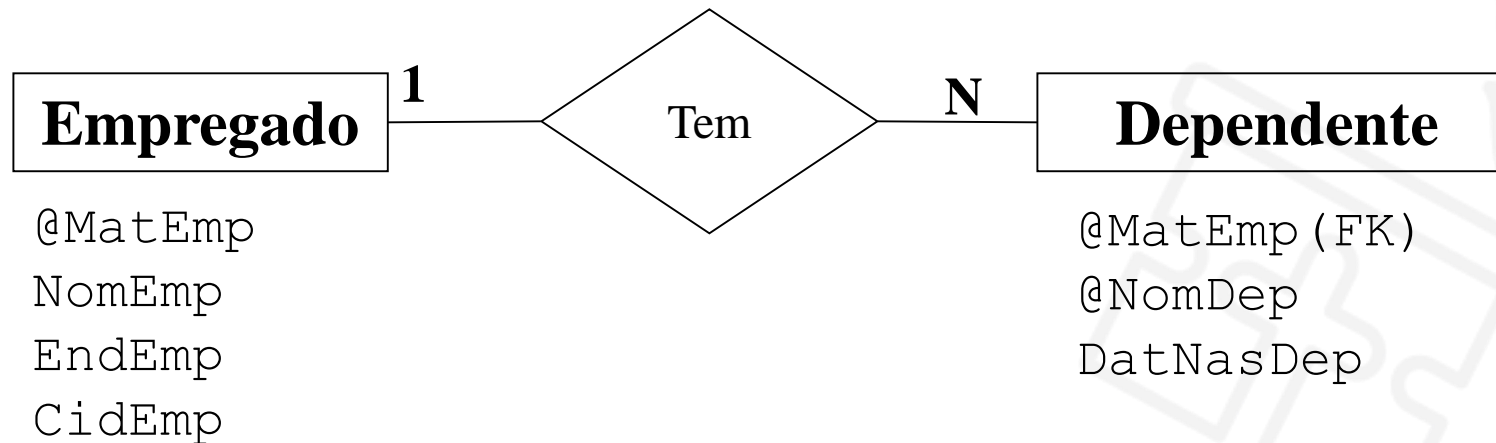
Referência: Navathe. **Sistemas de Banco de Dados** – 2011, 6ª Ed.

Comando INSERT

- Os valores devem ser relacionados na mesma ordem em que as colunas foram especificados no comando CREATE TABLE, caso o nome das colunas não seja explicitados.
- Sintaxe:

INSERT INTO <nome da tabela>[(<nome da coluna 1>, <nome da coluna 2>, <nome da coluna n>)] **VALUES** (<valor constante 1>, <valor constante 2>, <valor constante n>)

Comando INSERT



```
INSERT INTO Empregado (MatEmp, NomEmp, EndEmp , CidEmp )  
VALUES (10, 'Marcos Ramos', 'Rua X', 'Itabira');  
INSERT INTO Empregado VALUES (11, 'Marcos Ramos', null, 'SP');  
INSERT INTO Empregado (NomEmp, EndEmp , MatEmp, CidEmp )  
VALUES ('Mario Andrade', 'Rua 4', 12, 'Belo Horizonte');
```

```
INSERT INTO Dependente (NomDep, DatNasDep, MatEmp)  
VALUES ('Carlos Ramos', '2003-01-10', 10);  
INSERT INTO Dependente (NomDep, DatNasDep, MatEmp)  
VALUES ('Joao Andrade', '2003-10-01', 11);
```

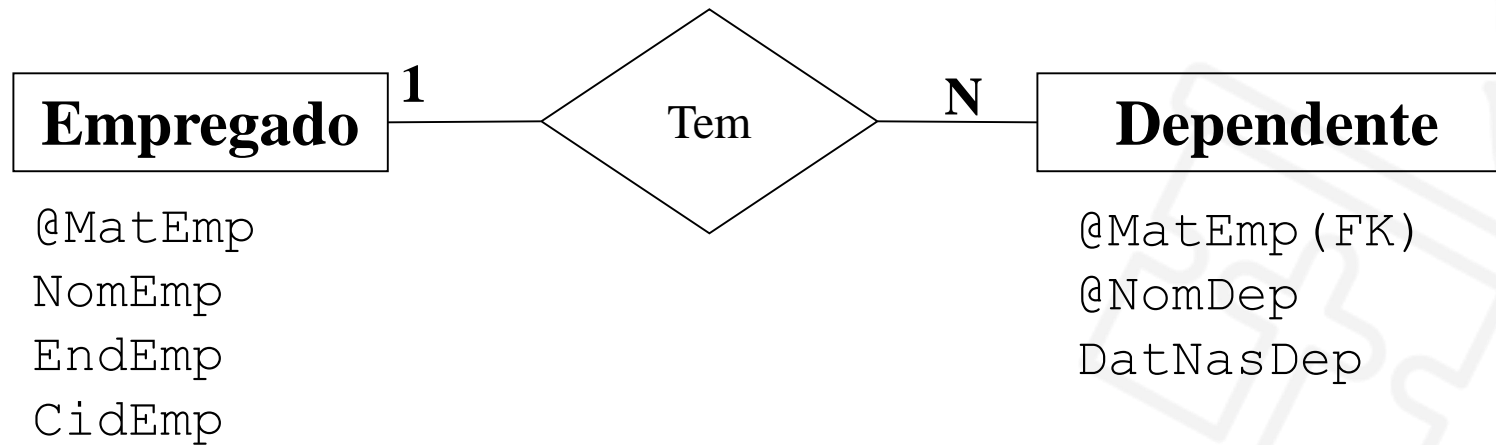
Comando DELETE

- Remove tuplas (linhas) de uma relação (tabela).
- Possui a cláusula **WHERE** para filtrar as linhas a serem excluídas.
 - Poderão se excluídas nenhuma, uma ou várias linhas em um único comando.
- As remoções poderão propagar-se nas tuplas de outras relações.
- Cláusula WHERE omitida indica que todas as linhas na tabela devem ser excluídas (resulta em tabela vazia).
- Sintaxe:

DELETE FROM <nome da tabela>
[**WHERE** <condição da seleção>];

Referência: Navathe. **Sistemas de Banco de Dados** – 2011, 6ª Ed.

Comando DELETE



```
DELETE FROM EMPREGADO  
WHERE MatEmp = 12;
```

```
DELETE FROM EMPREGADO;  
--Apaga Todas as linhas da relação empregado  
(tabela empregado se torna vazia).
```

Comando UPDATE

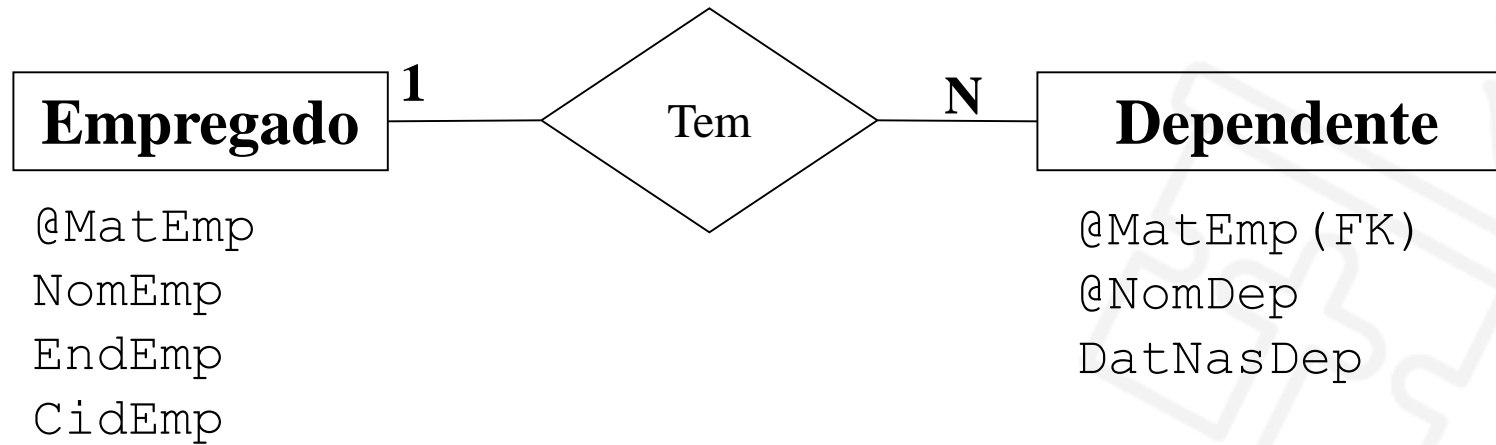
- Colunas de linhas existentes em uma tabela podem ser modificadas.
- Possui a cláusula **WHERE** para filtrar as linhas a serem alteradas.
- Uma cláusula adicional **SET** dentro do comando UPDATE, **especifica os atributos que serão modificados e seus novos valores.**
- Sintaxe:

UPDATE <nome da tabela>

SET <nome da coluna 1> = <expressão do valor 1>, <nome da coluna n> =
<expressão do valor n>

[**WHERE** <condição da seleção>];

Comando DELETE



```
UPDATE Empregado
SET EndEmp = 'Rua Agua Santa'
WHERE MatEmp = 10;
```

Prática

Abra o banco de dados e apague e atualize alguns registros nas tabelas Empregado e Dependente. Novamente, fique atento com as restrições (Domínio, Chave, Entidade, Integridade referencial)

Observe o que acontece ao tentar apagar registros que violem a integridade referencial.

Prática

Agora altere a tabela dependente para propagar a exclusão de um registro em empregado a todos os seus dependentes. Use o comando:

```
Alter TABLE Dependente DROP CONSTRAINT FK_EMP_DEP
```

```
Alter table Dependente ADD constraint FK_EMP_DEP foreign key  
(MatEmp) references EMPREGADO (MatEmp) On delete Cascade
```

Lembre-se que antes você deve excluir os dependentes.

Agora repita o procedimento anterior para apagar um empregado que contém dependentes e veja o resultado.

DQL (Linguagem de consulta de Dados)

Estrutura do comando SELECT

Comando SELECT

Comando SELECT é a instrução Básica para recuperar informações.

O formato básico da declaração SELECT é composto por três cláusulas (SELECT, FROM e WHERE):

Sintaxe:

```
SELECT <lista atributos>  
FROM <lista de tabelas>  
WHERE <condição>
```

Comando SELECT

<lista atributos> é uma lista de nomes de atributos cujo valores são recuperados.

<lista de tabelas> é uma lista dos nomes das relações necessárias para o processamento de consulta.

<condição> é uma expressão condicional que identifica as tuplas que serão recuperadas.

Comando SELECT

Exemplo:

```
SELECT *  
FROM Empregado;
```

Esta consulta seleciona todas as colunas da tabela EMPREGADO.

```
SELECT NomEmp, CidEmp  
FROM Empregado  
WHERE MatEmp = 11;
```

Esta consulta seleciona as colunas **NomEmp** e **cidEmp** da tabela EMPREGADO, cujo valor do atributo MatEmp seja igual a 11.

Comando SELECT

Esta consulta seleciona as colunas **NomEmp** e **cidEmp** da tabela EMPREGADO, cujo valor do atributo MatEmp seja igual a 11.

Obs.: Como a coluna MatEmp é atributo chave a consulta irá retornar apenas uma linha, a linha do empregado com MatEmp igual a 11 (pois não há outra linha com valor 11 para MatEmp).

Comando SELECT

É uma das ultimas cláusulas a ser avaliada pelo SGBD.
Antes de gerar o resultado, é necessário conhecer todas as colunas que podem fazer parte do resultado.

A cláusula SELECT **determina quais entre todas as colunas possíveis deverão ser incluídas no conjunto resultado da consulta.**

Comando SELECT

A cláusula FROM define as tabelas usadas em uma consulta.

```
SELECT *  
FROM Empregado
```


Prática

Altere a tabela Empregado e adicione o campo salário.
Atualize os registros existentes de Empregado para adicionar o valor dos salários.

```
ALTER TABLE Empregado ADD Salario FLOAT NULL
```

```
UPDATE Empregado  
SET Salario = 2000  
WHERE MatEmp = 10;
```

```
UPDATE Empregado  
SET Salario = 1000  
WHERE MatEmp = 11;
```

```
UPDATE Empregado  
SET Salario = 3000  
WHERE MatEmp = 12;
```

Expressões aritméticas, alias (atributo e relação) e uso do asterisco

Expressões aritméticas

Expressões com dados numéricos e de data usando operadores aritméticos:

Operador	Descrição
+	Adição
-	Subtração
*	Multiplicação
/	Divisão

```
SELECT NomEmp, Salario, Salario+300  
FROM EMPREGADO
```

Expressões aritméticas

- A multiplicação e a divisão têm prioridade sobre a adição e a subtração.
- Os operadores com a mesma prioridade são avaliados da esquerda para a direita.
- Parênteses são usados para forçar a avaliação priorizada e para esclarecer as instruções.
- Exemplo:

```
SELECT NomEmp, Salario, 12*Salario+100  
FROM EMPREGADO
```

```
SELECT NomEmp, Salario, 12*(Salario+100)  
FROM EMPREGADO
```

Aliases de Colunas

Cada coluna retornada no comando SELECT, o nome do atributo é apresentado como resultado.

Porém, **é possível atribuir rótulos próprios adicionando um alias (apelido) de coluna após cada elemento da cláusula SELECT.**

É útil para cálculos, especialmente se o nome da coluna for longo.

```
SELECT NomEmp AS NOME_EMPREGADO, Salario AS  
SAL_EMPREGADO  
FROM EMPREGADO EMP
```

```
SELECT NomEmp, Salario, 12*(Salario) AS  
SALARIO_ANUAL  
FROM EMPREGADO
```

Aliases de Colunas

Quando múltiplas tabelas são utilizadas em uma única consulta, é necessário uma maneira para identificar a qual tabela a coluna pertence.

Principalmente quando há colunas de tabelas diferentes com o mesmo nome.

Desta forma, define-se um alias a cada tabela para utilizá-lo ao longo da consulta.

```
SELECT NomEmp AS NOME_EMPREGADO, Salario AS SAL_EMPREGADO  
FROM EMPREGADO AS EMP
```

Definindo Alias de Relações

Em SQL é possível dar nomes alternativos pela adição do qualificador **AS** (alias).

O alias pode ser usado tanto nos atributos quanto nas relações e em ambas as cláusulas SELECT e FROM.

O alias de relações é utilizado principalmente quando é necessário realizar junção de duas ou mais relações.

Neste caso, **o alias é importante para a explicitar a distinção dos atributos de cada uma das relações** durante toda a consulta.

Definindo Alias de Relações

Exemplo:

Recupere o nome e o endereço de todos os empregados.

```
SELECT E.NomEmp, E.EndEmp  
FROM Empregado E
```

O Uso do **AS** não é obrigatório para pseudônimo de relações.

Uso do asterisco

Para recuperar **todos os valores dos atributos** da tuplas selecionadas, não é necessário especificar todos os seus nomes.

Utiliza-se apenas um asterisco (*), que significa selecionar todos os atributos.

```
SELECT *  
FROM Empregado
```

```
SELECT *  
FROM Empregado  
WHERE MatEmp = 11;
```

Distinct e Cláusula Where

Prática

Execute o script “modelo_relacional.sql” fornecido junto ao material de aula. Verifique as alterações realizadas pelo script.

Removendo Linhas Duplicadas

Uma consulta pode retornar linhas de dados duplicadas. Por exemplo, ao selecionar todos os códigos do clientes que possuem conta:

```
SELECT CODCLIENTE  
FROM CONTA;
```

Alguns clientes podem ter mais de uma conta, o mesmo código do cliente será exibido para cada conta de cada cliente.

Removendo Linhas Duplicadas

O objetivo é obter o conjunto específico de cliente que possuem conta (independente de quantas contas ele possui).

É necessário utilizar a cláusula DISTINCT, diretamente após a cláusula SELECT:

```
SELECT DISTINCT CODCLIENTE  
FROM CONTA;
```

Cláusula WHERE

A cláusula WHERE é utilizada para restringir as linhas selecionadas.

A cláusula WHERE segue a cláusula FROM.

```
SELECT NOMEMP, CARGO  
FROM EMPREGADO  
WHERE BAIRRO = 'SAVASSI'
```

Cláusula WHERE

Exemplo:

```
SELECT NomEmp, CidEmp  
FROM Empregado  
WHERE MatEmp = 11;
```

- Esta consulta envolve somente a relação EMPREGADO relacionada na cláusula FROM
- A consulta seleciona as tuplas de EMPREGADO que satisfazem a condição da cláusula WHERE
- Então, projeta o resultado dos atributos NomEmp e CidEmp relacionados nos cláusula SELECT.

Cláusula WHERE

- A cláusula SELECT, em SQL, especifica atributos para projeção.
- A cláusula WHERE especifica a condição de seleção.
 - Somente aquelas tuplas que satisfazem a condição serão selecionadas.

Cláusula WHERE

A ausência da cláusula Where significa que não há nenhuma condição para seleção de tuplas.

Assim, todas as tuplas da relação da cláusula FROM serão retornadas no resultado da consulta.

```
SELECT NomEmp  
FROM Empregado;
```

Cláusula WHERE

Se for especificada mais de uma relação na cláusula FROM, então o produto cartesiano dessas relações será obtido.

```
SELECT NomEmp, EndEmp  
FROM Empregado, Departamento
```

É extremamente importante especificar todas as condições de seleção e junção (abordaremos este tópico mais tarde) na cláusula WHERE para evitar resultados incorretos ou muito grandes.

Operadores de Comparação

Operadores de Comparação

Em SQL, os operadores lógicos básicos de comparação usado são:

=, <, <=, >, >=, e <>

Operadores de: igualdade, menor, menor ou igual, maior, maior ou igual e diferente.

Operadores de Comparação

- Operadores lógicos de comparação:

AND, OR e NOT

- Operado AND retorna verdadeiro (*true*), se todas as condições forem verdadeiras.
- Operador OR retorna verdadeiro (*true*), se pelo menos umas das condições forem verdadeiras.
- Operador NOT retorna verdadeiro (*true*), se condição for falsa.

LIKE

- Em SQL é possível criar condição para comparação de partes de uma cadeia de caracteres por meio do operador de comparação LIKE.
- Esse operador pode ser usado para comparação de padrões de cadeia.
- As partes das cadeias podem ser especificadas usando dois caracteres:
 - % : substitui um número arbitrário de caracteres.
 - Underscore (_): substitui um único caracter.

LIKE

Exemplo:

Recupere todos os empregados cujos endereços sejam de Belo Horizonte:

```
SELECT NomEmp  
FROM Empregado  
WHERE CidEmp LIKE '%Belo Horizonte%';
```

Recupere todos os empregados cujo a segunda letra do nome seja 'a':

```
SELECT NomEmp  
FROM Empregado  
WHERE NomEmp LIKE '_a%';
```

BETWEEN

- Em SQL também é possível comparar valores dentro de um determinado intervalo:
- Exemplo:
 - Recupere todos os empregados do departamento 1 que ganham entre 30 mil e 40 mil.

```
SELECT *  
FROM Empregado  
WHERE (Salario BETWEEN 30000 AND 40000) AND CodDepto = 1;
```


NOT IN

- Também é possível realizar comparações com uma lista de valores ao invés de apenas um único valor.
- O operador IN (lista), pode ser utilizados para comparação com uma lista de valores.
- Exemplo:
 - Recupere os empregados cujo o cargo não seja nem de vendedor e nem de presidente.

```
SELECT NomEmp  
FROM EMPREGADO  
WHERE CARGO NOT IN ('Vendedor', 'Presidente')
```

IS NULL

- Utilizando o operador IS NULL é possível comparar e verificar se o valor do atributo é nulo.
- Exemplo:
 - Recupere os empregados que trabalham no departamento 10 e que não receberam nenhuma comissão.

```
SELECT NomEmp  
FROM EMPREGADO  
WHERE COMISSAO IS NULL AND CodDepto = 1
```

```
SELECT NomEmp  
FROM EMPREGADO  
WHERE COMISSAO IS NOT NULL AND CodDepto = 1
```

Funções Agregadas

Funções Agregadas em SQL

O SQL tem funcionalidades que incorporam conceitos de funções agregadas.

Há diversas funções para este fim:

Max()

Retorna o valor máximo dentro de um conjunto.

Min()

Retorna o valor mínimo dentro de um conjunto.

Avg()

Retorna o valor médio de um conjunto.

Sum()

Retorna a soma dos valores de um conjunto.

Count()

Retorna a quantidade de valores de um conjunto.

Funções Agregadas em SQL

Exemplo 1:

Encontre a soma dos salários, o maior salário, o menor salário, e a média salarial de todos os empregados.

```
SELECT SUM (salario), MAX (salario), MIN (salario), AVG (salario)
FROM Empregado;
```

Funções Agregadas em SQL

Exemplo 3

Recupera o número total de empregados da empresa:

```
SELECT COUNT (*)  
FROM Empregado;
```

Neste exemplo, * refere-se às linhas (tuplas), logo COUNT devolverá o número de linhas do resultado da consulta.

Order By, Group By e Having

Ordenando o resultado das consultas

SQL permite que o usuário ordene as tuplas do resultado de uma consulta pelos valores de seus atributos, utilizando a cláusula **ORDER BY.**

```
SELECT NomEmp  
FROM EMPREGADO  
ORDER BY NomEmp
```

A ordenação padrão é de forma ascendente. Pode-se também especificar a palavra-chave DESC (sempre depois do nome da coluna) para ordenar os valores de forma descendente.

```
SELECT NomEmp  
FROM EMPREGADO  
ORDER BY NomEmp DESC
```


Agrupamento

Em muitos casos, é necessário **aplicar as funções agregadas para os subgrupos de tuplas de uma relação**. No qual os subgrupos são escolhidos com base em alguns atributos.

SQL possui a cláusula **Group By** para esse fim. Esta especifica os atributos de agrupamento, que também poderiam aparecer na cláusula Select.

Agrupamento

Exemplo:

Para cada departamento, recupere seu número, número de empregados que nele trabalham e a média de seus salários.

```
SELECT CodDepto, COUNT (*) as 'qtd', AVG (Salario) as 'sal'  
FROM Empregado  
GROUP BY CodDepto;
```

As tuplas de Empregado serão particionadas em grupos, **cada grupo tendo o mesmo valor para o atributo do agrupamento.**

A função COUNT() e AVG (), serão aplicada **em cada grupo de tuplas**

Agrupamento

Exemplo:

<u>CPF</u>	Nome	Sal	CodDepto
123	José	1000	1
223	Maria	2000	1
323	João	2000	2
123	Alberto	4000	2
121	Carla	3000	3

Agrupamento das tuplas
de Empregado por meio do
valor do CodDepto

Resultado da consulta

CodDepto	COUNT(*)	AVG(Sal)
1	2	1500
2	2	3000
3	1	3000

Agrupamento

Algumas vezes é necessário recuperar os valores das funções agregadas apenas para os grupos que satisfazem certas condições.

Como a cláusula GROUP BY é executada após a avaliação da cláusula WHERE, você não pode adicionar condição de filtro à cláusula WHERE para este propósito.

Por exemplo: Suponha que seja necessário recuperar, para cada projeto, o seu **numero**, seu **nome** e **numero de empregados** que nele trabalham. **Mas somente os projetos com mais de dois empregados.**

Agrupamento

Exemplo:

Suponha que seja necessário recuperar, para cada **projeto**, o seu **numero**, seu **nome** e **numero de empregados** que nele trabalham.

```
SELECT P.CodProjeto, P.ProjNome, COUNT (*)  
FROM Projeto P, Trabalha_Em T  
WHERE P.CodProjeto = T.CodProjeto  
GROUP BY P.CodProjeto, P.ProjNome
```

Agrupamento

Exemplo:

Suponha que seja necessário recuperar, para cada projeto, o seu numero, seu nome e numero de empregados que nele trabalham. **Mas somente os projetos com mais de dois empregados.**

```
SELECT P.CodProjeto, P.ProjNome, COUNT (*)  
FROM Projeto P, Trabalha_Em T  
WHERE P.CodProjeto = T.CodProjeto AND COUNT (*) > 2  
GROUP BY P.CodProjeto, P.ProjNome
```

Agrupamento

Exemplo:

Suponha que seja necessário recuperar, para cada projeto, o seu numero, seu nome e numero de empregados que nele trabalham. **Mas somente os projetos com mais de dois empregados.**

```
SELECT P.CodProjeto, P.ProjNome, COUNT (*)  
FROM Projeto P, Trabalha_Em T  
WHERE P.CodProjeto = T.CodProjeto AND COUNT (*) > 2  
GROUP BY P.CodProjeto, P.ProjNome
```

Incorreto, pois os grupos ainda não foram gerados no momento em que a cláusula WHERE é avaliada.

Agrupamento

Exemplo:

Suponha que seja necessário recuperar, para cada projeto, o seu numero, seu nome e numero de empregados que nele trabalham. **Mas somente os projetos com mais de dois empregados.**

```
SELECT P.CodProjeto, P.ProjNome, COUNT (*)  
FROM Projeto P, Trabalha_Em T  
WHERE P.CodProjeto = T.CodProjeto  
GROUP BY P.CodProjeto, P.ProjNome  
HAVING COUNT (*) > 2
```


Agrupamento

Enquanto as condições de seleção da cláusula WHERE limitam as tuplas nas quais as funções serão aplicadas, a cláusula HAVING serve para escolher grupos inteiros.

Consultas Aninhadas

Consultas Aninhadas

Algumas consultas dependem da busca de valores presentes no banco de dados para, então, usá-lo na condição de comparação.

Essas consultas podem ser formuladas por meio das consultas aninhadas.

Um bloco completo de *select-from-where* dentro da cláusula WHERE da consulta externa.

Consultas Aninhadas

Exemplo:

Recupere o Nome do Funcionário que tem o maior salário Bruto da empresa:

```
SELECT NomEmp  
FROM EMPREGADO  
WHERE SALARIOBRUTO = (SELECT MAX(SALARIOBRUTO)  
FROM EMPREGADO)
```

Recupera o maior salário da tabela funcionário

Pesquisa todos os salários da tabela funcionário e compara com o valor MAX. Quando encontrar, projeta o nome do funcionário (campo NOME da tupla com SALARIO = MAX)

Consultas Aninhadas

Exemplo:

Recupere o CPF de todos os empregados que trabalham a mesma quantidade de horas em algum dos projetos em que o empregado 'John' (CPF = '123') trabalhe.

```
SELECT  DISTINCT MatEmp
FROM    Trabalha_Em
WHERE    Horas IN (SELECT Horas
                     FROM Trabalha_Em
                     WHERE MatEmp = '123');
```

Recupera todas as horas trabalhadas nos projetos do CPF 123. O resultado é um conjunto

Consultas Aninhadas


Trabalha_Em

<u>CPF</u>	<u>CodProj</u>	Horas
123	1	20
223	1	10
323	2	10
123	2	15
121	3	20
112	3	15
332	3	20
333	4	20

```
SELECT DISTINCT MatEmp
FROM Trabalha_Em
WHERE Horas IN (SELECT Horas
FROM Trabalha_Em
WHERE MatEmp = '123');
```

Consultas Aninhadas

Trabalha_Em



<u>CPF</u>	<u>CodProj</u>	Horas
123	1	20
223	1	10
323	2	10
123	2	15
121	3	20
112	3	15
332	3	20
333	4	20

```
SELECT DISTINCT MatEmp
FROM Trabalha_Em
WHERE Horas IN (SELECT Horas
FROM Trabalha_Em
WHERE MatEmp = '123');
```

Consultas Aninhadas

Trabalha_Em

<u>CPF</u>	<u>CodProj</u>	Horas
123	1	20
223	1	10
323	2	10
123	2	15
121	3	20
112	3	15
332	3	20
333	4	20

```
SELECT DISTINCT MatEmp
FROM Trabalha_Em
WHERE Horas IN (SELECT Horas
FROM Trabalha_Em
WHERE MatEmp = '123');
```

Resultado

Conj_Resultado

Horas
20
15

Consultas Aninhadas

Trabalha_Em

<u>CPF</u>	<u>CodProj</u>	Horas
123	1	20
223	1	10
323	2	10
123	2	15
121	3	20
112	3	15
332	3	20
333	4	20

```
SELECT DISTINCT MatEmp
FROM Trabalha_Em
WHERE Horas IN (SELECT Horas
FROM Trabalha_Em
WHERE MatEmp = '123');
```

Consultas Aninhadas

Trabalha_Em ↓

<u>CPF</u>	<u>CodProj</u>	Horas
123	1	20
223	1	10
323	2	10
123	2	15
121	3	20
112	3	15
332	3	20
333	4	20

```
SELECT DISTINCT MatEmp
FROM Trabalha_Em
WHERE Horas IN (SELECT Horas
FROM Trabalha_Em
WHERE MatEmp = '123');
```

IN →

Conj_Resultado

Horas
20
15

Consultas Aninhadas

Trabalha_Em ↓

<u>CPF</u>	<u>CodProj</u>	Horas
123	1	20
223	1	10
323	2	10
123	2	15
121	3	20
112	3	15
332	3	20
333	4	20

```
SELECT DISTINCT MatEmp
FROM Trabalha_Em
WHERE Horas IN (SELECT Horas
FROM Trabalha_Em
WHERE MatEmp = '123');
```

IN →

Conj_Resultado

Horas
20
15

Consultas Aninhadas

Trabalha_Em

<u>CPF</u>	<u>CodProj</u>	Horas
123	1	20
223	1	10
323	2	10
123	2	15
121	3	20
112	3	15
332	3	20
333	4	20

```
SELECT DISTINCT MatEmp
FROM Trabalha_Em
WHERE Horas IN (SELECT Horas
FROM Trabalha_Em
WHERE MatEmp = '123');
```

IN

Conj_Resultado

Horas
20
15

Resultado

<u>MatEmp</u>
123
123
121
112
332
333

Consultas Aninhadas

Trabalha_Em

<u>CPF</u>	<u>CodProj</u>	Horas
123	1	20
223	1	10
323	2	10
123	2	15
121	3	20
112	3	15
332	3	20
333	4	20

```
SELECT DISTINCT MatEmp
FROM Trabalha_Em
WHERE Horas IN (SELECT Horas
FROM Trabalha_Em
WHERE MatEmp = '123');
```

IN

Conj_Resultado

Horas
20
15

DISTINCT

<u>MatEmp</u>
123
121
112
332
333

Consultas Aninhadas

A palavras ALL pode ser combinada com algum dos operadores de comparação.

Exemplo:

Recupere o nome dos empregados cujo salários são maiores que os salários de todos os empregados do departamento 2.

```
SELECT NomEmp
FROM Empregado
WHERE Salario > ALL (SELECT Salario
                     FROM Empregado
                     WHERE CodDepto = 2);
```

Consultas Aninhadas

Exemplo (Alias):

Recupere o nome de cada um dos empregados que tenham dependentes cujo o sexo seja o mesmo do empregado em questão:

```
SELECT NomEmp  
FROM Empregado AS E  
WHERE E.MatEmp IN (SELECT MatEmp  
                   FROM Dependente  
                   WHERE E.Sexo = Sexo);
```

Atributo Sexo da tabela Empregado



Função EXISTS

É usada para verificar se o resultado de uma consulta aninhada é vazio (não contém nenhuma tupla) ou não.

Exemplo:

Recupere o nome dos empregados que possuem dependentes:

```
SELECT NomEmp
FROM Empregado E
WHERE Exists ( SELECT *
                FROM Dependente D
                WHERE D.MatEmp = E.MatEmp)
```

Se existir qualquer tupla no conjunto resultado da consulta interna, a tupla Empregado será selecionada.

Função EXISTS

Pode-se também utilizar NOT EXISTS:

Exemplo:

Recupere o nome dos empregados que **não** possuem dependentes:

```
SELECT NomEmp
FROM Empregado E
WHERE NOT EXISTS (SELECT *
                  FROM Dependente D
                  WHERE D.MatEmp = E.MatEmp)
```

Se **não** existir nenhuma tupla no conjunto resultado da consulta interna, a tupla Empregado será selecionada.

JOINS

Vínculos de tabelas

Em SQL é possível realizar vínculos entre duas ou mais relações do banco de dados.

Para tanto, **todas as relações envolvidas devem estar presentes na cláusula FROM.**

Quando mais de um tabela aparece na cláusula FROM, as condições para vincular as tabelas devem ser incluídas (condição de junção).

Vínculos de tabelas

A operação de junção é usada para combinar as tuplas relacionadas em duas relações dentro de uma tupla única.

Importante pois permite processar os relacionamentos entre as relações.

Exemplo:

Recuperar o nome do gerente de cada departamento. É necessário avaliar cada tupla de departamento com a tupla empregado na qual o valor do CPF case com o valor do CPFGerente da tupla departamento.

Vínculos de tabelas

O resultado da junção é uma relação Q , em que há uma tupla (em Q) para cada combinação de tuplas (uma de R e uma de S), quando a combinação satisfazer a condição de junção.

Na junção, **apenas as combinações de tuplas que satisfazem a condição de junção aparecerão no resultado.**

Vínculos de tabelas

O conceito de junção de tabelas foi incorporado à SQL para **especificar uma tabela que fosse resultado da junção das tabelas na cláusula FROM.**

Exemplo:

Recupere o nome e endereço de todos os empregados que trabalham no departamento 'Pesquisa':

Vínculos de tabelas

Resposta 1:

```
SELECT E.NomEmp, E.EndEmp  
FROM Empregado E, Departamento D  
WHERE D.deptoNome = 'Pesquisa' AND D.codDepto = E.CodDepto;
```

O mecanismo de vinculação das duas relações é a afiliação do funcionário e um departamento armazenada na tabela Empregado (pelo atributo FK).

Condição de junção junto com a condição de seleção.

Vínculos de tabelas

O SGBD é instruído a usar o valor da coluna "**CodDepto**" na relação Empregado para procurar o nome do departamento associado na relação departamento.

A condição de junção é realizada na cláusula Where (**D. codDepto = E. CodDepto**).

Vínculos de tabelas

```
SELECT E.NomEmp, E.EndEmp
FROM Empregado E, Departamento D
WHERE D.CodDepto = E.CodDepto
      AND D.deptoNome = 'Pesquisa'
```

Empregado

<u>CPF</u>	Nome	End	CodDepto
123	José	Rua 3	1
223	Maria	Rua 10	2
323	João	Rua 10	2
123	Alberto	Rua 15	3
121	Carla	Rua 20	1

Departamento

<u>Codigo</u>	Nome
1	Pesquisa
2	Vendas
3	Administrativo

Vínculos de tabelas

```
SELECT E.NomEmp, E.EndEmp
FROM   Empregado E, Departamento D
WHERE  D.CodDepto = E.CodDepto
      AND D.deptoNome = 'Pesquisa'
```

Empregado

<u>CPF</u>	Nome	End	CodDepto
123	José	Rua 3	1
223	Maria	Rua 10	2
323	João	Rua 10	2
123	Alberto	Rua 15	3
121	Carla	Rua 20	1

Departamento

<u>Codigo</u>	Nome
1	Pesquisa
2	Vendas
3	Administrativo




Vínculos de tabelas

```
SELECT E.NomEmp, E.EndEmp
FROM Empregado E, Departamento D
WHERE D.CodDepto = E.CodDepto
AND D.deptoNome = 'Pesquisa'
```


FK para a tabela Departamento

Empregado



<u>CPF</u>	Nome	End	CodDepto
123	José	Rua 3	1
223	Maria	Rua 10	2
323	João	Rua 10	2
125	Alberto	Rua 15	3
121	Carla	Rua 20	1

Departamento



<u>Codigo</u>	Nome
1	Pesquisa
2	Vendas
3	Administrativo

Vínculos de tabelas

```
SELECT E.NomEmp, E.EndEmp  
FROM   Empregado E, Departamento D  
WHERE  D.CodDepto = E.CodDepto  
       AND D.deptoNome = 'Pesquisa'
```

Empregado

<u>CPF</u>	Nome	End	CodDepto
123	José	Rua 3	1
223	Maria	Rua 10	2
323	João	Rua 10	2
125	Alberto	Rua 15	3
121	Carla	Rua 20	1

Departamento

<u>Codigo</u>	Nome
1	Pesquisa
2	Vendas
3	Administrativo

Vínculos de tabelas

```
SELECT E.NomEmp, E.EndEmp
FROM   Empregado E, Departamento D
WHERE  D.CodDepto = E.CodDepto
      AND D.deptoNome = 'Pesquisa'
```

Empregado

<u>CPF</u>	Nome	End	CodDepto
123	José	Rua 3	1
223	Maria	Rua 10	2
323	João	Rua 10	2
125	Alberto	Rua 15	3
121	Carla	Rua 20	1

Departamento

<u>Codigo</u>	Nome
1	Pesquisa
2	Vendas
3	Administrativo

Conj. Resultado

Nome	End
José	Rua 3
Carla	Rua 20

JOIN

Resposta 2:

```
SELECT E.NomEmp, E.EndEmp  
FROM Empregado E JOIN Departamento D ON E.CodDepto = D.CodDepto  
WHERE D.DeptoNome = 'Pesquisa';
```

Condição de seleção separada da condição de junção.
A cláusula FROM contém uma única *tabela juntada (joined table)*.

JOIN

Execute as duas consultas e veja o resultado:

```
select E.MatEmp as 'Matrícula Empregado', E.NomEmp as 'Nome  
Empregado', D.NomDep as 'Nome Dependente'  
from Empregado E, Dependente D  
WHERE D.MatEmp = E.MatEmp
```

```
select E.MatEmp as 'Matrícula Empregado', E.NomEmp as 'Nome  
Empregado', D.NomDep as 'Nome Dependente'  
from Empregado E Join Dependente D ON D.MatEmp = E.MatEmp
```

Tips de JOINS (conteúdo extra)

Tipos de JOINS

O tipo de junção padrão entre duas tabelas é a interseção entre as linhas das duas tabelas:

```
SELECT *  
FROM Empregado E JOIN Departamento D ON  
E.CodDepto = D.CodDepto
```

Execute a consulta e veja o resultado

Vamos conhecer todos os tipos de Joins entre múltiplas tabelas

Tipos de JOINS

- Inner Join
 - Tipo de junção que retorna somente as linhas que satisfazem a condição de junção entre ambas as tabelas (se omitir o a palavra INNER, por default o resultado será o mesmo)

```
SELECT *  
FROM Empregado E INNER JOIN Departamento D ON E.CodDepto = D.CodDepto
```

Tipos de JOINS

- Outer Join
 - Left Outer Join: retorna todas as linhas da tabela da esquerda e somente as linhas da tabela da direita que satisfazem a condição de junção (preenche como NULL campos das linhas da tabela da esquerda que não satisfazem a condição de junção)

```
SELECT *  
FROM Empregado E LEFT OUTER JOIN Departamento D ON E.CodDepto = D.CodDepto
```

Tipos de JOINS

- Outer Join
 - Right Outer Join: retorna todas as linhas da tabela da direita e somente as linhas da tabela da esquerda que satisfazem a condição de junção (preenche como NULL campos das linhas da tabela da direita que não satisfazem a condição de junção)

```
SELECT *  
FROM Empregado E RIGHT OUTER JOIN Departamento D ON E.CodDepto = D.CodDepto
```

Tipos de JOINS

- Outer Join
 - Full Outer Join: retorna todas as linhas de ambas as tabelas (preenche como NULL campos das linhas que não satisfazem a condição de junção)

```
SELECT *  
FROM Empregado E FULL OUTER JOIN Departamento D ON E.CodDepto = D.CodDepto
```

Tipos de JOINS

- Cross Join
 - Retorna o produto cartesiano de todas as tabelas participantes da junção. Não requer a condição de junção. A quantidade de linhas retornadas é $n \times m$, sendo n e m a quantidade de linhas das tabelas da esquerda e da direita, respectivamente.

```
SELECT *  
FROM Empregado E CROSS JOIN Departamento D
```

Tipos de JOINS

- Self Join
 - Usado para a junção (Inner, Outer ou Cross) de uma tabela com ela mesma, quando a tabela possui um campo de chave estrangeira referenciando sua própria chave primária. Porém, não existe uma cláusula "self" no SQL.

`SELECT *`

`FROM Empregado E INNER JOIN Empregado Emp ON E.CodSupervisor = Emp.MatEmp`

Observe que está sendo feito um join com a própria tabela

Junção (Join) em mais de duas Tabelas

É possível fazer junção entre várias tabelas: retorne o nome dos empregados que trabalham em algum projeto e o nome de todos os projetos em que eles trabalham

```
SELECT E.NomEmp, P.ProjNome  
FROM Empregado E INNER JOIN Trabalha_Em T ON E.MatEmp = T.MatEmp  
INNER JOIN Projeto P ON T.CodProjeto = P.CodProjeto
```


Dicas!

Links para saber mais:

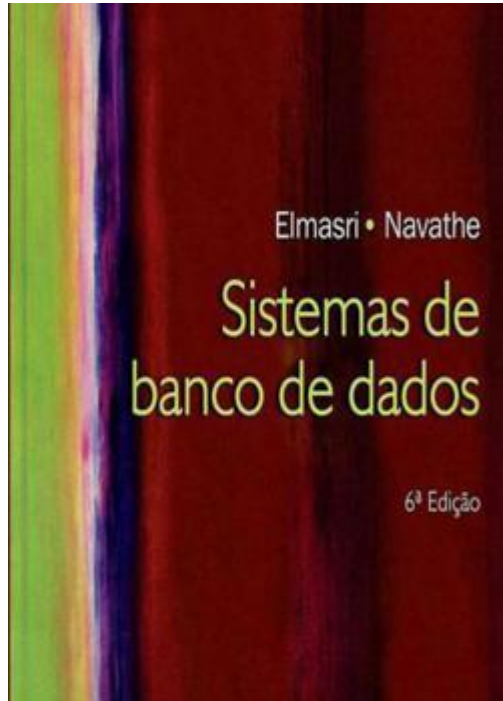
Microsoft T-SQL

<https://docs.microsoft.com/pt-br/sql/t-sql/lesson-1-creating-database-objects?view=sql-server-ver15>

Oracle PL/SQL

<https://www.oracle.com/br/database/technologies/appdev/plsql.html>

Principais Referências



Navathe. **Sistemas de Banco de Dados** – 2011, 6ª Ed.



PUC Minas
Virtual