BANCOS DE DADOS RELACIONAIS

Profº Me. Jones Artur Gonçalves

SELECT FROM DUAS OU MAIS TABELAS

A primeira característica da instrução SELECT que vamos estudar é o caso mais simples de join, ou de como se obter informações de mais de uma tabela simultaneamente no mesmo comando. Ou, como unir ou juntar as informações de mais de uma tabela na mesma instrução.

A sintaxe da instrução SELECT com uso de joins é:

SELECT <lista de colunas>
FROM <nome de uma ou mais tabelas>
WHERE <lista de condições>

SELECT FROM COM DUAS OU MAIS TABELAS

O principal item a ser destacado agora é que, na cláusula FROM, em vez de apresentarmos o **nome de apenas uma tabela, podemos incluir uma lista de tabelas, separadas por vírgulas**.

Como poderíamos selecionar os seguintes dados: nome do funcionário, slário, nome do setor, das tabelas funcionário e setor?

```
Select Funcionario.primeiro_nome, Funcionario.salario,
Setor.nome_setor
from Funcionario, Setor
```

SELECT FROM COM DUAS OU MAIS TABELAS

Quais os problemas dessa solução? Como poderíamos corrigí-lo?

E sem dizermos como as tabelas Funcionários e Setor se relacionam, o SGBD vai interpretar que ele deve trazer para você todos os materiais para cada linha de fornecedor que ele encontrar.

Isto é, ele vai entender que o relacionamento existente é: todas as linhas da tabela de funcionários se relacionam com todas as linhas da tabela de setor e isso não é verdade. Na realidade, um funcionário pertence a um só setor e nós precisamos dizer isso ao SGBD.

SELECT FROM COM DUAS OU MAIS TABELAS

E a forma de dizer isso é incluindo uma restrição na cláusula WHERE onde vamos dizer para o banco de dados que ele deve nos trazer apenas os setores relacionados aos seus funcionários correspondentes. Assim, fica nosso comando complementado com a cláusula **WHERE**

```
Select Funcionario.primeiro_nome, Funcionario.salario,
Setor.nome_setor
from Funcionario, Setor
Where Funcionario.cod_setor = Setor.cod_setor
```

SELECT FROM COM DUAS OU MAIS TABELAS

Vamos introduzir neste ponto um novo conceito que ajudará muito na clareza e facilidade de leitura da instrução SELECT: o *alias* do nome das tabelas. O *alias* é um sinônimo que você atribui a uma tabela. É uma outra forma de se referenciar ao nome da mesma. Quando escrevemos a instrução, prefixamos as colunas com os nomes das respectivas tabelas.

Simples e claro até aqui porque nosso exemplo possui apenas duas colunas. Se a instrução possuísse vinte colunas e você tivesse que prefixar o nome das respectivas tabelas antes de cada uma delas, seria um desperdício de espaço e tempo e, além disso, a legibilidade do comando estaria prejudicada. Por isso existe o alias.

SELECT FROM COM DUAS OU MAIS TABELAS

Veja a seguir o mesmo exemplo reescrito utilizando um alias simples para cada tabela:

```
Select f.primeiro_nome, f.salario,  s.nome_setor
from Funcionario f, Setor s
Where f.cod_setor = s.cod_setor
```

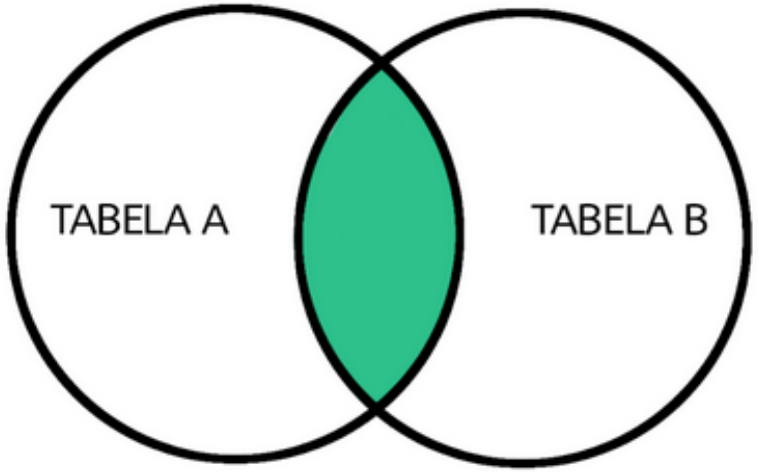
INNER JOINS

Entre os métodos de junção, o INNER JOIN é um dos mais conhecidos e tem a função de retornar os valores em comum de ambas as tabelas.

- Especifica todos os pares de linhas correspondentes retornados.
- Descarta as linhas não correspondentes de ambas as tabelas.
- Quando nenhum tipo de junção é especificado, este é o padrão.

Na ilustração abaixo, é possível ver a representação desse retorno de forma gráfica. Observe:

```
SELECT <select_list>
FROM Tabela A
INNER JOIN Tabela B
ON A.Key = B.Key
```



INNER JOINS

Veja a seguir o mesmo exemplo reescrito utilizando Inner Join:

```
Select f.primeiro_nome, f.salario,  s.nome_setor
from Funcionario f
Inner join Setor s on f.cod_setor = s.cod_setor
```

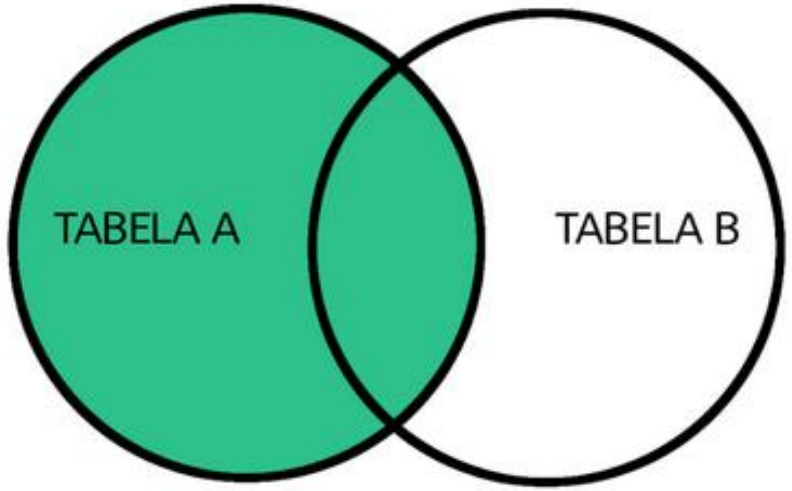
```
Select c.nome_cliente, p.num_pedido
from cliente c
Inner join pedido p on c.cod_cliente = p.cod_cliente
```

LEFT OUTER JOINS

O LEFT OUTER JOIN é usado para retornar todos os registros da tabela esquerda, além dos registros da tabela à direita que têm valores em comum com a tabela esquerda.

Na ilustração abaixo, é possível ver a representação desse retorno de forma gráfica. Observe:

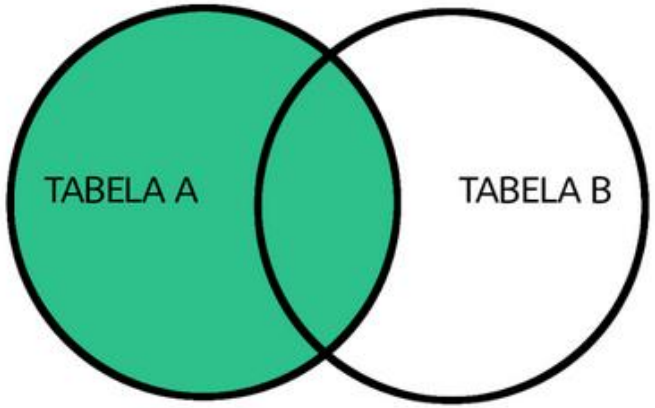
```
SELECT <select_list>
FROM Tabela A
LEFT OUTER JOIN Tabela B
ON A.Key = B.Key
```



LEFT OUTER JOINS

Para cada linha da tabela A, a consulta a compara com todas as linhas da tabela B. Se um par de linhas fizer com que a condição de junção seja avaliado como Verdadeiro, os valores da coluna dessas linhas serão combinados para formar uma nova linha que será incluída no conjunto de resultados.

Se uma linha da tabela "esquerda" A não tiver nenhuma linha correspondente da tabela "direita" B, a consulta irá combinar os valores da coluna da linha da tabela "esquerda" A com NULL para cada valor da coluna da tabela da "direita" B que não satisfaça a condição de junto (Falso).



LEFT OUTER JOINS

Veja a seguir o mesmo exemplo reescrito utilizando Left outer Join:

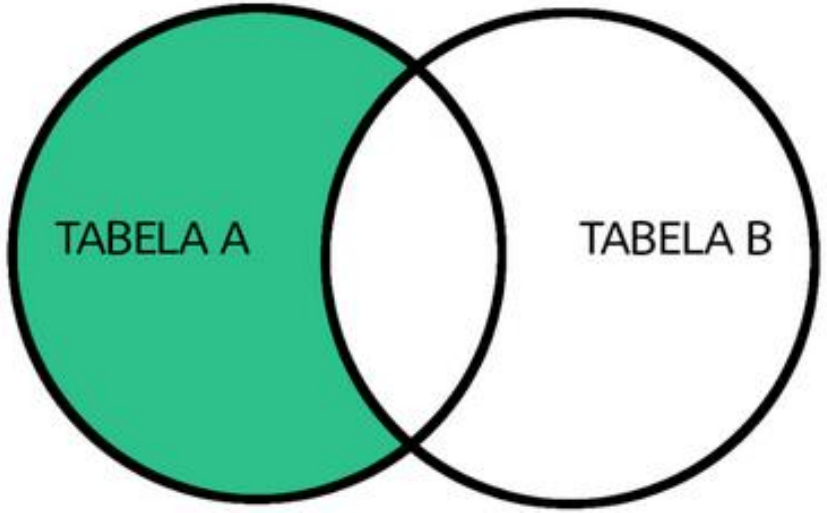
```
Select c.nome_cliente, p.num_pedido
from cliente c
Left outer join pedido p on c.cod_cliente = p.cod_cliente
```

LEFT EXCLUDING JOINS

Já com o LEFT EXCLUDING JOIN é possível retornar todos os dados da tabela esquerda que não têm valores correspondentes na tabela da direita. Na imagem mostrada abaixo, vemos uma representação desse tipo de resultado.

Na ilustração abaixo, é possível ver a representação desse retorno de forma gráfica. Observe:

```
SELECT <select_list>
FROM Tabela A
LEFT OUTER JOIN Tabela B
ON A.Key = B.Key
WHERE B.CAMPO IS NULL
```



LEFT EXCLUDING JOINS

Prosseguindo com nossa demonstração, repare que agora acrescentamos a cláusula WHERE no fim do comando. Dessa forma, definimos um filtro para excluir do retorno os dados em comum entre as tabelas.

Veja a seguir o mesmo exemplo reescrito:

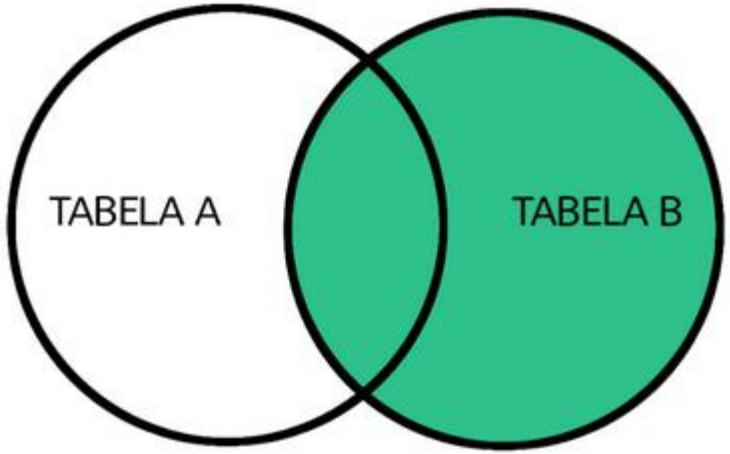
```
Select c.nome_cliente, p.num_pedido
from cliente c
Left outer join pedido p on c.cod_cliente = p.cod_cliente
Where P.cod_cliente is null
```

RIGHT OUTER JOINS

O RIGHT OUTER JOIN é usado para retornar todos os registros da tabela direita, além dos registros da tabela à esquerda que têm valores em comum com a tabela direita.

Na ilustração abaixo, é possível ver a representação desse retorno de forma gráfica. Observe:

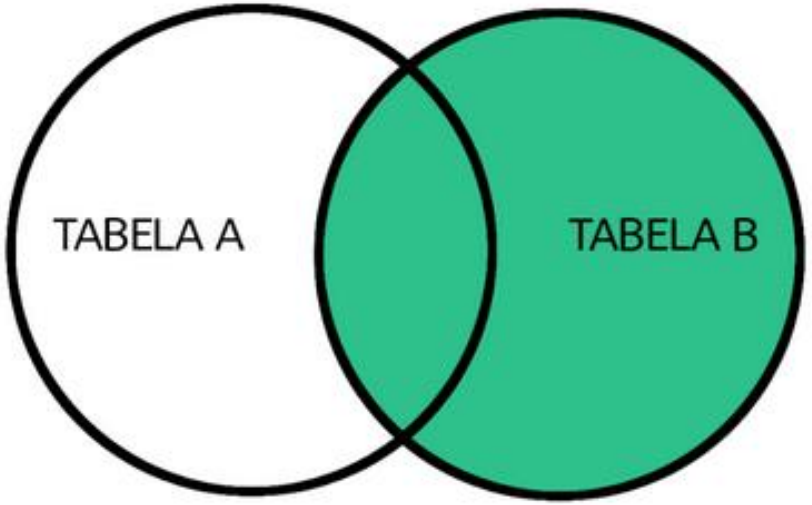
```
SELECT <select_list>
FROM Tabela A
RIGHT OUTER JOIN Tabela B
ON A.Key = B.Key
```



RIGHT OUTER JOINS

A RIGHT OUTER JOIN retorna um conjunto de resultados que inclui todas as linhas da tabela "direita" B, com ou sem linhas correspondentes na tabela "esquerda" A.

Se uma linha na tabela direita B não tiver nenhuma linha correspondente da tabela "esquerda" A, a coluna da tabela "esquerda" A no conjunto de resultados será nula igualmente ao que acontece no LEFT OUTER JOIN.



RIGHT OUTER JOINS

Veja a seguir o mesmo exemplo reescrito utilizando Right outer Join:

```
Select f.primeiro_nome, f.salario,  s.nome_setor
from Funcionario f
Right outer join Setor s on f.cod_setor = s.cod_setor
```

RIGHT OUTER JOINS

Neste caso vamos inserir um novo setor na tabela de setores (Marketing por exemplo), para termos um setor onde não há funcionários cadastrados.

Insert into Setor (nome_setor) values ('Marketing')

Agora vamos fazer uma nova consulta com Right Outer Join

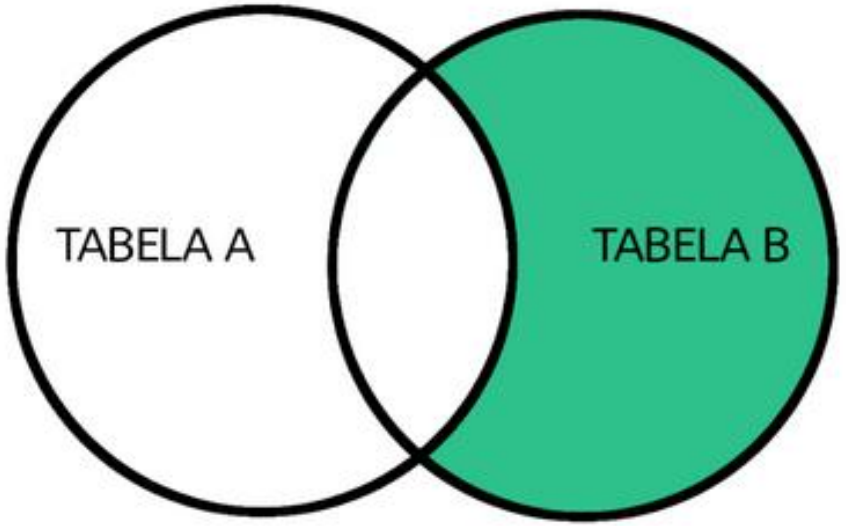
```
Select f.primeiro_nome, f.salario,  s.nome_setor
from Funcionario f
Right outer join Setor s on f.cod_setor = s.cod_setor
```

RIGHT EXCLUDING JOINS

Como é mostrado na figura abaixo, o RIGHT EXCLUDING JOIN é responsável por retornar os dados da tabela da direita que não têm valores iguais na tabela esquerda.

Na ilustração abaixo, é possível ver a representação desse retorno de forma gráfica. Observe:

```
SELECT <select_list>
FROM Tabela A
RIGHT OUTER JOIN Tabela B
ON A.Key = B.Key
WHERE A.CAMPO IS NULL
```



RIGHT EXCLUDING JOINS

Novamente, para excluir os dados com valores em comum do retorno, usamos um filtro na cláusula WHERE.

Veja a seguir o mesmo exemplo reescrito:

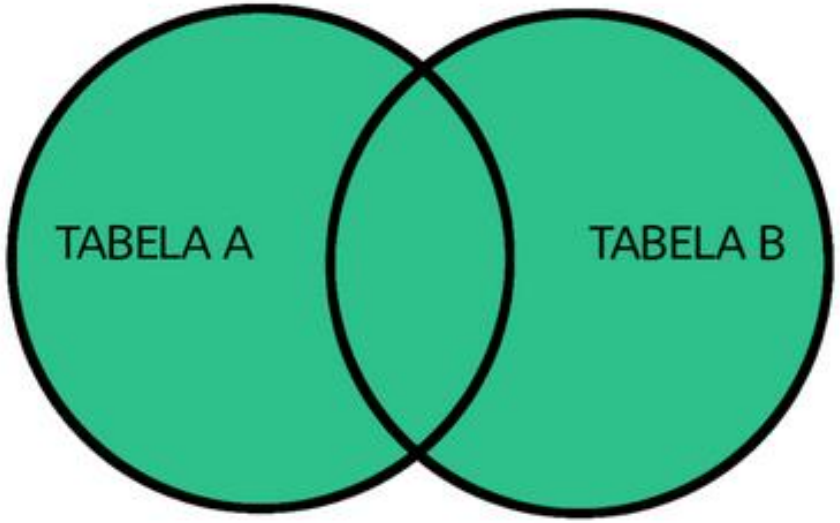
```
Select f.primeiro_nome, f.salario,  s.nome_setor
from Funcionario f
Right outer join Setor s on f.cod_setor = s.cod_setor
Where f.Cod_setor is null
```

FULL OUTER JOIN

O FULL JOIN, também conhecido como FULL OUTER JOIN, retorna todos os dados de ambas as tabelas quando há uma relação entre elas.

Na ilustração abaixo, é possível ver a representação desse retorno de forma gráfica. Observe:

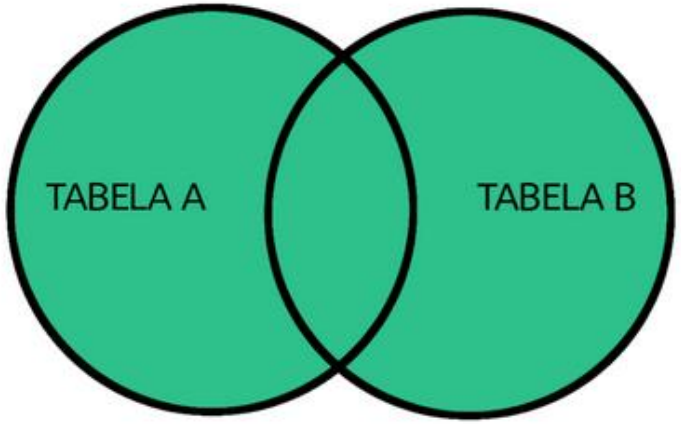
```
SELECT <select_list>
FROM Tabela A
FULL OUTER JOIN Tabela B
ON A.Key = B.Key
```



FULL OUTER JOIN

A cláusula FULL JOIN retorna todas as linhas das tabelas unidas, correspondidas ou não, ou seja, você pode dizer que a FULL JOIN combina as funções da LEFT JOIN e da RIGHT JOIN. FULL JOIN é um tipo de junção externa, por isso também é chamada junção externa completa.

Quando não existem linhas correspondentes para a linha da tabela esquerda, as colunas da tabela direita serão nulas. Da mesma forma, quando não existem linhas correspondentes para a linha da tabela direita, a coluna da tabela esquerda será nula.



FULL OUTER JOIN

Veja a seguir o mesmo exemplo:

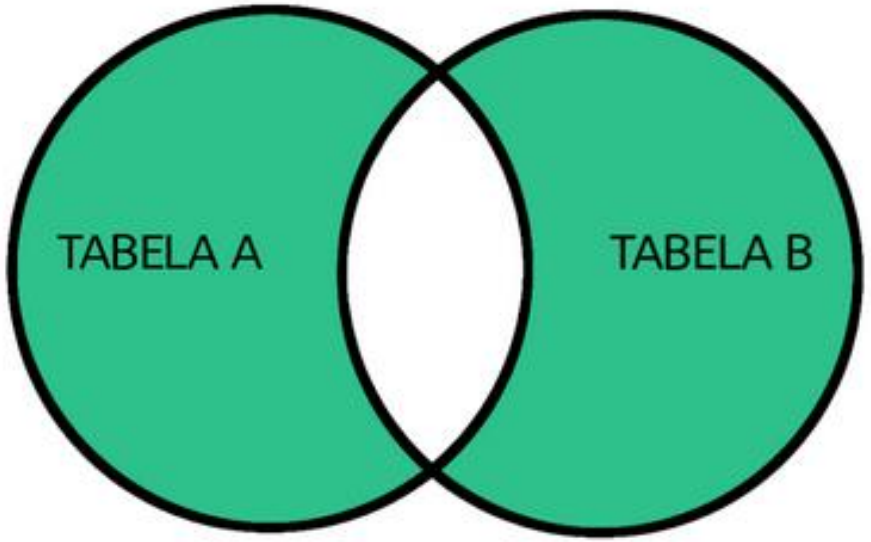
```
Select c.nome_cliente, p.num_pedido
from cliente c
Full outer join pedido p on c.cod_cliente = p.cod_cliente

Select f.primeiro_nome, f.salario,  s.nome_setor
from Funcionario f
Full outer join Setor s on f.cod_setor = s.cod_setor
```

FULL EXCLUDING JOIN

Com o FULL EXCLUDING JOIN podemos retornar todos os registros que não têm valores repetidos entre as tabelas. Ou seja, ele exclui do retorno todos os dados duplicados, como é possível ver na figura abaixo:

```
SELECT <select_list>
FROM Tabela A
FULL OUTER JOIN Tabela B
ON A.Key = B.Key
WHERE A.CAMPO IS NULL or
B.CAMPO ISNULL
```



FULL EXCLUDING JOIN

Agora, perceba que acrescentamos no código um filtro com múltiplas condições usando o operador OR na cláusula WHERE. Dessa forma, conseguimos excluir os nomes em comum entre as tabelas, retornando apenas os dados que não se repetem. Veja a seguir o mesmo exemplo:

```
Select f.primeiro_nome, f.salario,  s.nome_setor
from Funcionario f
full outer join Setor s on f.cod_setor = s.cod_setor
Where f.Cod_setor is null or s.Cod_setor is null
```

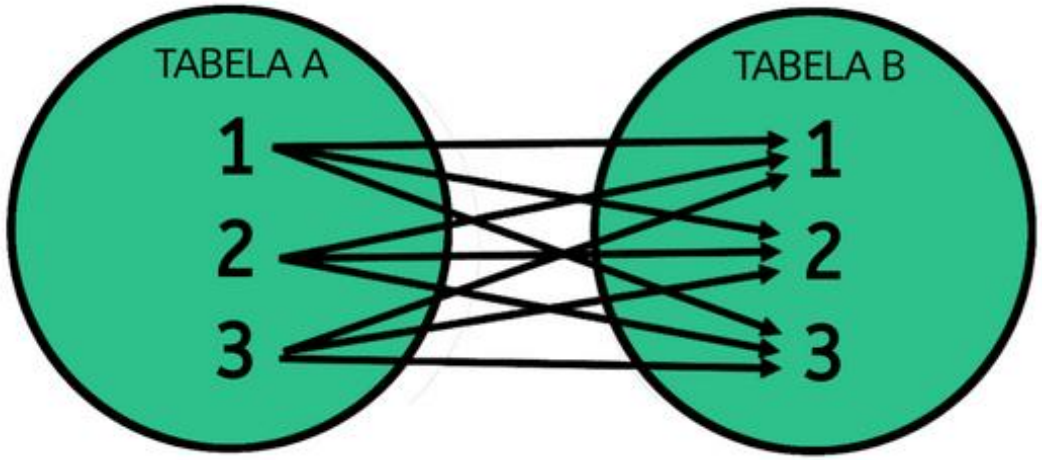
```
Select c.nome_cliente, p.num_pedido
from cliente c
Full outer join pedido p on c.cod_cliente = p.cod_cliente
Where c.Cod_cliente is null or p.cod_cliente is null
```

CROSS JOIN

A cláusula CROSS JOIN retorna todas as linhas das tabelas por cruzamento, ou seja, para cada linha da tabela esquerda queremos todos os linhas da tabelas direita ou vice-versa. Ele também é chamado de produto cartesiano entre duas tabelas. Porém, para isso é preciso que ambas tenham o campo em comum, para que a ligação exista entre as duas tabelas.

É mais fácil entender o CROSS JOIN como um "Join sem cláusula ON", ou seja, todas as combinações de linhas de A e B são devolvidas.

```
SELECT <select_list>
FROM Tabela A
CROSS JOIN Tabela B
```



CROSS JOIN

Veja a seguir o mesmo exemplo:

```
Select f.primeiro_nome, f.salario,  s.nome_setor
from Funcionario f
Cross join Setor s
Order by f.primeiro_nome
```

```
Select c.nome_cliente, p.num_pedido
from cliente c
Cross join pedido p
Order by c.Nome_cliente
```

GERAÇÃO DE DADOS SUMARIZADOS

As funções agregadas retornam valores resumo, para a tabela inteira ou para grupo de colunas dentro da tabela. O resultado aparece como uma nova coluna no resultado da consulta.

A SQL oferece a habilidade para computar funções em grupos de tuplas (linhas) usando a cláusula GROUP BY .

GERAÇÃO DE DADOS SUMARIZADOS

O atributo ou atributos dados na cláusula GROUP BY são usados para formar grupos. Tuplas com o mesmo valor em todos os atributos na cláusula cláusula GROUP BY são colocados em um grupo. A SQL inclui funções para computar :

- média : **avg**
- mínimo : **min**
- máximo : **max**
- total : **sum**
- contar : **count**

RESUMO

funções agregadas - produzem um único valor baseado em uma determinada coluna

cláusula GROUP BY - permite organizar em grupos estes dados sumarizados

cláusula HAVING - aplica restrições aos grupos gerados, ou seja, é usada com a cláusula GROUP BY para filtrar grupos no conjunto de resultados.

FUNÇÕES AGREGADAS

Sintaxe

```
SELECT nome_atributo1,...,
função_agregada1  ([ALL/ DISTINCT] expressão)....
FROM  nome_tabela
WHERE condição
```

Expressões incluem nomes de colunas, constantes ou funções conectadas por operadores aritméticos.

FUNÇÕES AGREGADAS

➢ **Tabela**

Função	**Parâmetros**	**descrição**
AVG	([ALL/ DISTINCT] expressão)	Média de valores na coluna especificada, todos ou distintos
COUNT	([ALL/ DISTINCT] expressão)	Número de valores na coluna, todos ou distintos
COUNT	(*)	Número de linhas selecionadas
MAX	(expressão)	Maior valor na coluna
MIN	(expressão)	Menor valor na coluna
SUM	([ALL/ DISTINCT] expressão)	Somatório de valores na coluna, todos ou distintos

As funções agregadas são utilizadas junto à lista de atributos do comando SELECT e/ou na cláusula HAVING

FUNÇÕES AGREGADAS

➢ **Exemplo**

MAX, MIN - Listar o menor e o maior salário de vendedor

```
SELECT min(salario_fixo) AS 'MENOR SALARIO', max(salario_fixo) AS
'MAIOR SALARIO'
FROM vendedor
```

SUM - Mostrar a quantidade total pedida para o produto de código ‘78’

```
SELECT SUM (quantidade)
FROM item_pedido
WHERE cod_produto = 3
```

FUNÇÕES AGREGADAS

➢ **Exemplo**

AVG - Qual a média dos salários fixos dos vendedores?

```
SELECT avg(salario_fixo) AS MEDIA_SALARIO
FROM vendedor
```

COUNT - Quantos vendedores ganham acima de R$ 2.500,00 de salário fixo

```
SELECT count (*) from vendedor
WHERE salario_fixo  > 2500.00
```

GROUP BY E HAVING

Sintaxe

```
SELECT   nome_atributo1, nome_atributo2,...
FROM  nome_tabela1, nome_tabela2...
            WHERE  condição
        [GROUP BY  expressão,....]
        [HAVING condição]
```

EXEMPLO

Listar QUANTOS produtos cada pedido contém

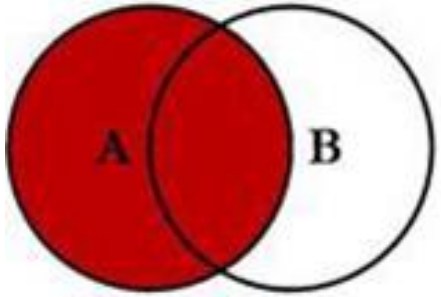
```
SELECT num_pedido, total_produtos = count (*)
FROM item_pedido
GROUP by num_pedido
```

EXEMPLO

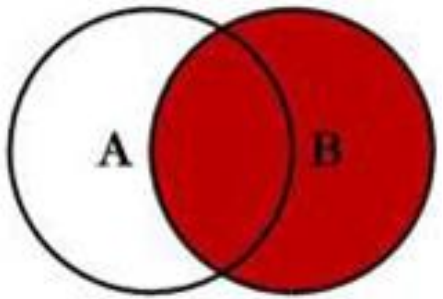
Listar os pedidos que têm mais do que um produto

```
SELECT num_pedido, total_produtos = COUNT (*)
FROM item_pedido
WHERE quantidade > 5
GROUP by num_pedido
HAVING count(*) > 1
```

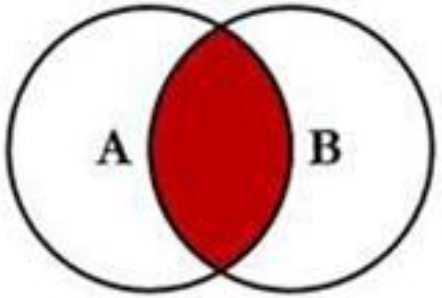
Junção



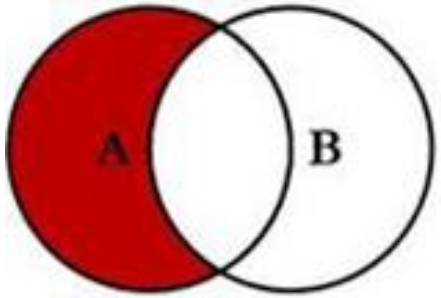

```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
```



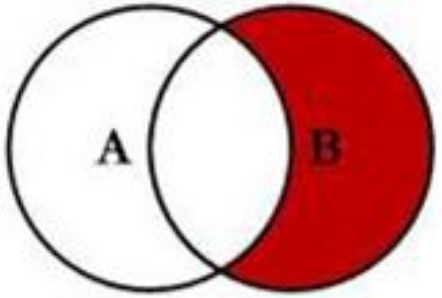

```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
```



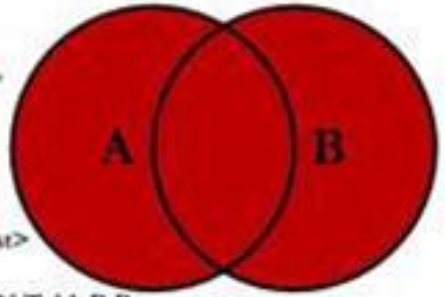

```
SELECT <select_list>
FROM TableA A
INNER JOIN TableB B
ON A.Key = B.Key
```



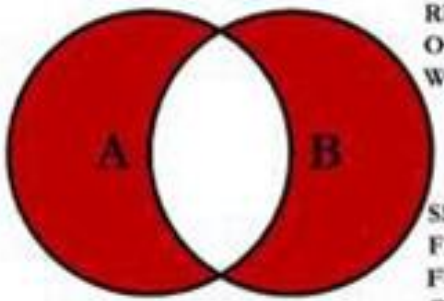

```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
WHERE B.Key IS NULL
```




```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
```




```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
```




```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
OR B.Key IS NULL
```


https://medium.com/@gugoiscardoso/subquery-vs-inner-join-c1e62c4d7bf4

Atualizando Dados

Em certas situações, podemos desejar mudar um valor em uma tupla sem mudar *todos* os valores na tupla.

Para isto, a instrução UPDATE pode ser usada.

UPDATE

- **Sintaxe**

```
UPDATE nome_tabela
SET  { nome_atributo1 = expressão [, nome_atributo2 = expressão]...}
WHERE  condição
```

Opera somente em uma tabela .

SET - especifica as colunas a serem alteradas e os valores novos.

WHERE - especifica quais as linhas que serão atualizadas.

Se os novos valores violarem as restrições de integridade o comando UPDATE não é efetivado.

UPDATE

- **Exemplo**

Alterar o valor unitário do produto ‘parafuso’ de R$ 1.25 para R$ 1.62

```
Update produto
Set valor_unitario = 1.62
Where descricao = 'parafuso'
```

UPDATE

- **Baseado em outras tabelas**

```
UPDATE nome_tabela
        SET  { nome_atributo=  ( cláusula SELECT)
        WHERE  condição
```

A cláusula SELECT aninhada deverá retornar um único valor. É executada um vez para cada linha. O valor calculado pelo SELECT atualizará a linha especificada. Somente linhas que satisfaçam a cláusula WHERE mais externa serão atualizadas.

UPDATE

➢ **Exemplo**

Acrescentar 2,5% ao preço unitário dos produtos que estejam abaixo da média dos preços, para aqueles comprados a Quilo

```
UPDATE produto
SET valor_unitario = valor_unitario * 1.025
WHERE valor_unitario <
    (SELECT avg(valor_unitario)
    FROM produto
    WHERE unidade = 'KG')
```

Excluindo dados

Remove linhas de uma tabela.

Sintaxe:

DELETE FROM <nome_tabela>
    WHERE <condição_de_seleção>;

Podemos **remover** apenas tuplas inteiras ;
Não podemos **remover** valores apenas em atributos particulares.

DELETE

➢ Exemplo

Apagar todos os vendedores com faixa de comissão nula

```
DELETE FROM vendedor
    WHERE faixa_comissao IS NULL
```

A cláusula DELETE remove uma ou mais linhas de uma única tabela , de acordo com a condição especificada na cláusula WHERE. Se a cláusula WHERE não for especificada todas as linhas serão removidas.

DELETE

- **Baseado em dados de outras tabelas**

Sintaxe

```
DELETE [FROM] nome_tabela *  verificar isto
          [FROM nome_tabela1, [nome_tabela2]...
       [WHERE condição]
```

DELETE

➢ **Exemplo**

Remover da tabela de pedidos os pedidos que contenham produtos que custem menos de R$0,20

Versão ANSI

```
DELETE FROM pedido
WHERE num_pedido IN
    (SELECT num_pedido
    FROM produto P, item_pedido I
 WHERE P.cod_produto = I.cod_produto and
valor_unitario < 0.20)
```

DELETE

➢ **Exemplo**

Remover da tabela de pedidos os pedidos que contenham produtos que custem menos de R$0,20

Versão Transaction SQL

```
DELETE FROM pedido
      FROM produto P, item_pedido I
WHERE I.num_pedido = pedido.num_pedido AND
P.cod_produto = I.cod_produto AND
valor_unitario < 0.20
```

BIBLIOGRAFIA

BÁSICA:

DATE, C. J. PROJETO DE BANCO DE DADOS E TEORIA RELACIONAL: FORMAS NORMAIS E TUDO O MAIS. SÃO PAULO: NOVATEC, 2015.

ELMASRI, R.; NAVATHE, S. B. SISTEMAS DE BANCO DE DADOS: FUNDAMENTOS E APLICAÇÕES. 7 ED. SÃO PAULO: PEARSON, 2019.

HEUSER, C. A. PROJETO DE BANCO DE DADOS. 6 ED. PORTO ALEGRE: BOOKMAN, 2010.

COMPLEMENTAR:

HARRINGTON, J. L. Projeto de Bancos de Dados Relacionais: Teoria e Prática. São Paulo: Campus, 2002.

MACHADO, F. N. R., Banco de dados: projeto e implementação. 2 ed. São Paulo: Érica, 2008.

NADEAU, Tom et al. Projeto e Modelagem de Banco de Dados. 5 ed. Rio de Janeiro: Elsevier Brasil, 2013.

SILBERSCHATZ, Abraham; SUNDARSHAN, S.; KORTH, Henry F. Sistema de banco de dados. Rio de Janeiro: Elsevier Brasil, 2016.

Referências

ALVES, W. P. FUNDAMENTOS DE BANCOS DE DADOS. ÉRICA, 2004

HEUSER, CARLOS ALBERTO. PROJETO DE BANCO DE DADOS. SAGRA LUZZATTO, 2004.

TEOREY, TOBY J. PROJETO E MODELAGEM DE BANCO DE DADOS. ELSEVIER, 2007.

O.K. TAKAI; I.C.ITALIANO; J.E. FERREIRA, INTRODUÇÃO A BANCO DE DADOS

OSVALDO KOTARO, APOSTILA, DCC-IME-USP – FEVEREIRO - 2005

MATTOSO, MARTA, INTRODUÇÃO À BANCO DE DADOS – AULA

GILLENSON, MARK L. FUNDAMENTOS DE SISTEMAS DE GERÊNCIA DE BANCO DE DADOS. LTC, 2006.

BANCO DE DADOS BÁSICO, UNICAMP, CENTRO DE COMPUTAÇÃO, SLIDES.

BOGORNY VANIA, MODELO ENTIDADE-RELACIONAMENTO, SLIDES.

WWW.JOINVILLE.UDESC.BR/PORTAL/PROFESSORES/MAIA/.../6___MODELO_ER.PPT DATA DE ACESSO: 01/07/2015

ABREU, FELIPE MACHADO; ABREU, MAURÍCIO – PROJETO DE BANCO DE DADOS – UMA VISÃO PRÁTICA - ED. ÉRICA – SÃO PAULO

HEUSER, CARLOS ALBERTO. PROJETO DE BANCO DE DADOS – UMA VISÃO PRÁTICA. PORTO ALEGRE: SAGRA LUZATTO, 2004.

KORTH, H. F.; SUDARSHAN, S; SILBERSCHATZ, A. SISTEMA DE BANCO DE DADOS. 5A ED. EDITORA CAMPUS, 2006. - CAPÍTULO 6

HTTP://WWW.PROFTONINHO.COM/DOCS/MODELAGEM_AULA_6_ENTID_ASSOC.PDF DATA DE ACESSO: 01/07/2015

HTTPS://MATERIALPUBLIC.IMD.UFRN.BR/CURSO/DISCIPLINA/4/56/1/6 DATA DE ACESSO: 01/02/2023

ELMASRI, R.; NAVATHE S. B. SISTEMAS DE BANCO DE DADOS. 4 ED. EDITORA ADDISON-WESLEY. 2005. - CAPÍTULO 3

DAVENPORT, THOMAS H.; PRUSAK, LAURENCE. CONHECIMENTO EMPRESARIAL: COMO AS ORGANIZAÇÕES GERENCIAM O SEU CAPITAL INTELECTUAL. RIO DE JANEIRO: CAMPUS, 1998.

HTTP://WWW.IME.UNICAMP.BR/~HILDETE/DADOS.PDF ACESSO EM: 12 MAIO 2016.

OBRIGADO