Associando tabelas

- INNER JOIN;
- OUTER JOIN;
- CROSS JOIN.

Esta Leitura Complementar refere-se ao conteúdo das Aulas 20 a 22.

Editora IMPACTA

1.1. Introdução

A associação de tabelas, ou simplesmente **JOIN** entre tabelas, tem como principal objetivo trazer, em uma única consulta (um único **SELECT**), dados contidos em mais de uma tabela.

Normalmente, essa associação é feita por meio da chave estrangeira de uma tabela com a chave primária da outra. Mas isso não é um pré-requisito para o **JOIN**, de forma que qualquer informação comum entre duas tabelas servirá para associá-las.

Diferentes tipos de associação podem ser escritos com a ajuda das cláusulas **JOIN** e **WHERE**. Por exemplo, podemos obter apenas os dados relacionados entre duas tabelas associadas. Também podemos combinar duas tabelas de forma que seus dados relacionados e não relacionados sejam obtidos.

Basicamente, existem três tipos de **JOIN** que serão vistos nesta leitura: **INNER JOIN**, **OUTER JOIN** e **CROSS JOIN**.

1.2. INNER JOIN

A cláusula **INNER JOIN** compara os valores de colunas provenientes de tabelas associadas, utilizando, para isso, operadores de comparação. Por meio desta cláusula, os registros de duas tabelas são utilizados para que sejam gerados os dados relacionados de ambas.

A sintaxe de **INNER JOIN** é a seguinte:

```
SELECT <lista_de_campos>
FROM <nome_primeira_tabela> [INNER] JOIN <nome_segunda_tabela> [ON
(condicao)]
```

Em que:

- **condicao**: Define um critério que relaciona as duas tabelas;
- **INNER**: É opcional, se colocarmos apenas **JOIN**, o **INNER** já é subentendido.

	CODFUN	NOME	NUM_DEPE...	DATA_NASCIMENTO	COD_DEP...
1	1	OLAVO TRINDADE	1	1950-06-06 00:00:00.000	4
2	2	JOSE REIS	6	1952-10-09 00:00:00.000	2
3	3	MARCELO SOARES	1	1950-06-06 00:00:00.000	5
4	4	PAULO CESAR JUNIOR	2	1952-03-19 00:00:00.000	8
5	5	JOAO LIMA MACHADO DA SILVA	2	1955-10-30 00:00:00.000	4
6	7	CARLOS ALBERTO SILVA	0	1961-07-06 00:00:00.000	11
7	8	ELIANE PEREIRA	0	1955-01-14 00:00:00.000	6
8	9	RUDGE RAMOS SANTANA DA PENHA	3	1961-07-22 00:00:00.000	2
9	10	MARIA CARMEM	0	1954-03-14 00:00:00.000	5

TB_EMPREGADO

	COD_DEP...	DEPTO
1	1	PESSOAL
2	2	C.P.D.
3	3	CONTROLE DE ESTOQUE
4	4	COMPRAS
5	5	PRODUCAO
6	6	DIRETORIA
7	7	TELEMARKETING
8	8	FINANCEIRO
9	9	RECURSOS HUMANOS
10	10	TREINAMENTO
11	11	PRESIDENCIA
12	12	PORTARIA
13	13	CONTROLADORIA
14	14	P.C.P.

TB_DEPARTAMENTO

Observando as duas tabelas, é possível concluir que o funcionário de **CODFUN** = **5** trabalha no departamento de **COMPRAS**, cujo código é **4**.

A especificação desse tipo de associação pode ocorrer por meio das cláusulas **WHERE** ou **FROM**. Veja os exemplos a seguir:

```
SELECT   TB_EMPREGADO.CODFUN, TB_EMPREGADO.NOME,
         TB_DEPARTAMENTO.DEPTO
FROM TB_EMPREGADO JOIN TB_DEPARTAMENTO
     ON TB_EMPREGADO.COD_DEPTO = TB_DEPARTAMENTO.COD_DEPTO;
```

```
SELECT TB_EMPREGADO.CODFUN, TB_EMPREGADO.NOME, TB_DEPARTAMENTO.DEPTO
FROM TB_EMPREGADO, TB_DEPARTAMENTO
WHERE TB_EMPREGADO.COD_DEPTO = TB_DEPARTAMENTO.COD_DEPTO;
```

A respeito do **INNER JOIN**, é importante considerar as seguintes informações:

- A principal característica do **INNER JOIN** é que somente trará registros que encontrem correspondência nas duas tabelas, ou seja, se existir um empregado com **COD_DEPTO** igual a 99 e na tabela de departamentos não existir um **COD_DEPTO** de mesmo número, esse empregado não aparecerá no resultado final;

- O **SELECT** apontará um erro de sintaxe se existirem campos de mesmo nome nas duas tabelas e não indicarmos de qual tabela vem cada campo.

Neste treinamento, faremos o **JOIN** sempre na cláusula **FROM**, usando a palavra **JOIN** e o critério com **ON**. Usar a cláusula **WHERE** para fazer o **JOIN** pode tornar o comando confuso e mais vulnerável a erros de resultado.

Veja os seguintes exemplos:

- **Exemplo 1**

```
SELECT CODFUN, NOME, DEPTO
FROM TB_EMPREGADO JOIN TB_DEPARTAMENTO
     ON TB_EMPREGADO.COD_DEPTO = TB_DEPARTAMENTO.COD_DEPTO;
```

Este exemplo está correto, porque não há duplicidade nos campos **CODFUN**, **NOME** e **DEPTO**.

- **Exemplo 2**

```
SELECT CODFUN, NOME, DEPTO
FROM TB_EMPREGADO JOIN TB_DEPARTAMENTO
     ON COD_DEPTO = COD_DEPTO;
```

Este exemplo está errado, porque o campo **COD_DEPTO** existe nas duas tabelas. É obrigatório indicar de qual tabela vamos pegar os campos.

> Sempre use o nome da tabela antes do nome do campo, mesmo que ele exista em apenas uma das tabelas.

Quando tivermos nomes de tabelas muito extensos, podemos simplificar a escrita, dando apelidos às tabelas. Veja os exemplos:

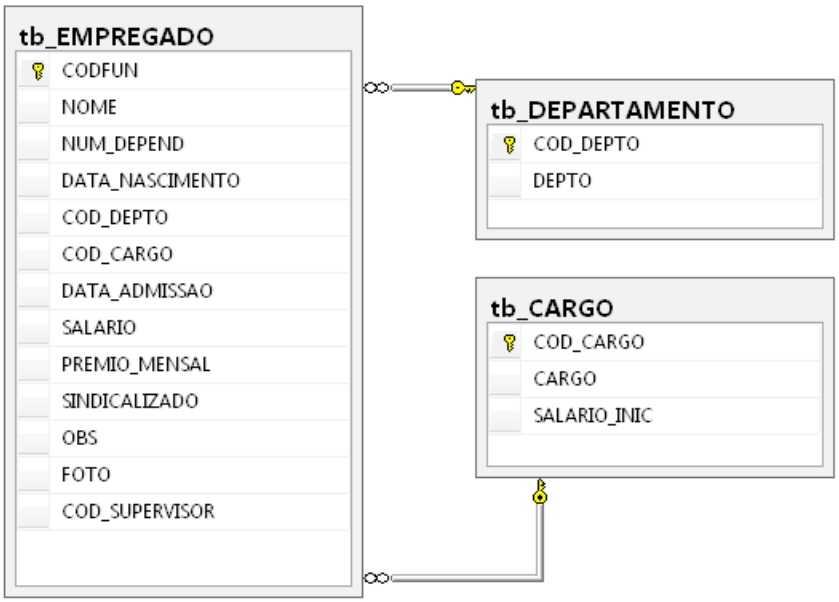
```
SELECT E.CODFUN, E.NOME, D.DEPTO
FROM TB_EMPREGADO AS E JOIN TB_DEPARTAMENTO AS D
     ON E.COD_DEPTO = D.COD_DEPTO;
```

```
-- OU
SELECT E.CODFUN, E.NOME, D.DEPTO
FROM TB_EMPREGADO E JOIN TB_DEPARTAMENTO D
     ON E.COD_DEPTO = D.COD_DEPTO;
```

O que acabamos de demonstrar é um **JOIN**, ou associação. Quando relacionamos duas tabelas, é preciso informar qual campo permite essa ligação. Normalmente, o relacionamento se dá entre a chave estrangeira de uma tabela e a chave primária da outra, mas isso não é uma regra.

Na figura a seguir, você pode notar um ícone de chave na posição horizontal localizado na linha que liga as tabelas **TB_EMPREGADO** e **TB_DEPARTAMENTO**. Essa chave está ao lado da tabela **TB_DEPARTAMENTO**, o que indica que é a chave primária dessa tabela (**COD_DEPTO**), e se relaciona com a tabela **TB_EMPREGADO,** que também possui um campo **COD_DEPTO**, que é a chave estrangeira.

Há, também, um **JOIN** entre as tabelas **TB_EMPREGADO** e **TB_CARGO**. Podemos perceber a existência de uma chave horizontalmente posicionada na linha que liga essas tabelas, e ela está ao lado de **TB_CARGO**. Então, é a chave primária de **TB_CARGO** (**COD_CARGO**) que se relaciona com a tabela **TB_EMPREGADO**. Em **TB_EMPREGADO**, também temos um campo **COD_CARGO**.



Normalmente, os campos que relacionam duas tabelas possuem o mesmo nome nas duas tabelas. Porém, isso não é uma condição necessária para que o **JOIN** funcione.

Quando executamos um **SELECT**, a primeira cláusula lida é **FROM**, antes de qualquer coisa. Depois é que as outras cláusulas são processadas. No código a seguir, temos dois erros: **E.CODIGO_DEPTO** (que não existe) e **FROM EMPREGADS** (erro de digitação):

```
SELECT
  E.CODFUN, E.NOME, E.CODIGO_DEPTO, E.COD_CARGO, D.DEPTO
FROM TB_EMPREGADS E
  JOIN TB_DEPARTAMENTO D ON E.COD_DEPTO = D.COD_DEPTO;
```

Executado o código anterior, a seguinte mensagem de erro será exibida:

```
Mensagem 208, Nível 16, Estado 1, Linha 1
Invalid object name 'EMPREGADS'
```

O primeiro erro acusado na execução do código foi na cláusula **FROM**, o que prova que ela é processada antes.

A seguir, temos outro exemplo de **JOIN**:

```
-- TB_EMPREGADO e TB_CARGO (cargos)
SELECT E.CODFUN, E.NOME, C.CARGO
FROM TB_EMPREGADO E JOIN TB_CARGO C ON E.COD_CARGO = C.COD_CARGO;
-- OU
SELECT E.CODFUN, E.NOME, C.CARGO
FROM TB_CARGO C JOIN TB_EMPREGADO E ON E.COD_CARGO = C.COD_CARGO;
```

No próximo exemplo, temos o uso de **JOIN** para consultar três tabelas:

```
-- Consultar 3 tabelas
SELECT
  E.CODFUN, E.NOME, E.COD_DEPTO, E.COD_CARGO, D.DEPTO, C.CARGO
FROM TB_EMPREGADO E
  JOIN TB_DEPARTAMENTO D ON E.COD_DEPTO = D.COD_DEPTO
  JOIN TB_CARGO C ON E.COD_CARGO = C.COD_CARGO;
-- OU
SELECT
  E.CODFUN, E.NOME, E.COD_DEPTO, E.COD_CARGO, D.DEPTO, C.CARGO
FROM TB_DEPARTAMENTO D
  JOIN TB_EMPREGADO E ON E.COD_DEPTO = D.COD_DEPTO
  JOIN TB_CARGO C  ON E.COD_CARGO = C.COD_CARGO;
-- OU
SELECT
  E.CODFUN, E.NOME, E.COD_DEPTO, E.COD_CARGO, D.DEPTO, C.CARGO
FROM TB_CARGO C
  JOIN TB_EMPREGADO E ON E.COD_CARGO = C.COD_CARGO
  JOIN TB_DEPARTAMENTO D  ON E.COD_DEPTO = D.COD_DEPTO;
```

Na consulta a seguir, temos dois erros. O primeiro é que não há **JOIN** entre **TB_DEPARTAMENTO** e **TB_CARGO**, como é mostrado no diagrama de tabelas do SSMS. O segundo é que não podemos fazer referência a uma tabela (**E.COD_CARGO**), antes de abri-la:

```
SELECT
  E.CODFUN, E.NOME, E.COD_DEPTO, E.COD_CARGO, D.DEPTO, C.CARGO
FROM TB_CARGO C
  JOIN TB_DEPARTAMENTO D  ON E.COD_DEPTO = D.COD_DEPTO   --<< (1)
  JOIN TB_EMPREGADO E ON E.COD_CARGO = C.COD_CARGO;
```

A seguir, temos mais um exemplo da utilização de **JOIN**, desta vez com seis tabelas:

```
/*
         Join com 6 tabelas. Vai exibir:
      TB_ITENSPEDIDO.NUM_PEDIDO
      TB_ITENSPEDIDO.NUM_ITEM
      TB_ITENSPEDIDO.COD_PRODUTO
      TB_PRODUTO.DESCRICAO
      TB_ITENSPEDIDO.QUANTIDADE
      TB_ITENSPEDIDO.PR_UNITARIO
      TB_TIPOPRODUTO.TIPO
      TB_UNIDADE.UNIDADE
      TB_COR.COR
      TB_PEDIDO.DATA_EMISSAO

      Filtrando TB_PEDIDO emitidos em Janeiro de 2014
*/
SELECT
   I.NUM_PEDIDO, I.NUM_ITEM, I.COD_PRODUTO, PR.DESCRICAO,
   I.QUANTIDADE, I.PR_UNITARIO, T.TIPO, U.UNIDADE, CR.COR,
   PE.DATA_EMISSAO
FROM TB_ITENSPEDIDO I
   JOIN TB_PRODUTO PR   ON I.ID_PRODUTO   = PR.ID_PRODUTO
   JOIN TB_COR CR     ON I.CODCOR       = CR.CODCOR
   JOIN TB_TIPOPRODUTO T ON PR.COD_TIPO    = T.COD_TIPO
   JOIN TB_UNIDADE U    ON PR.COD_UNIDADE = U.COD_UNIDADE
   JOIN TB_PEDIDO PE    ON I.NUM_PEDIDO   = PE.NUM_PEDIDO
WHERE PE.DATA_EMISSAO BETWEEN '2014.1.1' AND '2014.1.31';
```

É possível, também, associar valores em duas colunas não idênticas. Nessa operação, utilizamos os mesmos operadores e predicados utilizados em qualquer **INNER JOIN**. A associação de colunas só é funcional quando associamos uma tabela a ela mesma, o que é conhecido como autoassociação ou self-join.

Em uma autoassociação, utilizamos a mesma tabela duas vezes na consulta, porém, especificamos cada instância da tabela por meio de aliases, que são utilizados para especificar os nomes das colunas durante a consulta.

Observe que, na tabela **TB_EMPREGADO**, temos o campo **CODFUN** (código do funcionário) e temos também o campo **COD_SUPERVISOR**, que corresponde ao código do funcionário que é supervisor de cada empregado. Portanto, se quisermos consultar o nome do funcionário e do seu supervisor, precisaremos fazer um **JOIN**, da seguinte forma:

```
SELECT E.CODFUN, E.NOME AS FUNCIONARIO, S.NOME AS SUPERVISOR
FROM TB_EMPREGADO E JOIN TB_EMPREGADO S
     ON E.COD_SUPERVISOR = S.CODFUN;
```

1.3. OUTER JOIN

A cláusula **INNER JOIN**, vista anteriormente, tem como característica retornar apenas as linhas em que o campo de relacionamento exista em ambas as tabelas. Se o conteúdo do campo chave de relacionamento existe em uma tabela, mas não na outra, essa linha não será retornada pelo **SELECT**. Vejamos um exemplo de **INNER JOIN**:

```
-- INNER JOIN
SELECT * FROM TB_EMPREGADO; -- retorna 61 linhas
--
SELECT  -- retorna 58 linhas
  E.CODFUN, E.NOME, E.COD_DEPTO, E.COD_CARGO, C.CARGO
FROM TB_EMPREGADO E
  INNER JOIN TB_CARGO C ON E.COD_CARGO = C.COD_CARGO;
-- OU (a palavra INNER é opcional)
SELECT  -- retorna 58 linhas
  E.CODFUN, E.NOME, E.COD_DEPTO, E.COD_CARGO, C.CARGO
FROM TB_EMPREGADO E
  JOIN TB_CARGO C ON E.COD_CARGO = C.COD_CARGO;
```

```
/*
   Existem 61 linhas na tabela TB_EMPREGADO, mas quando fazemos
   INNER JOIN com TB_CARGO, retorna apenas com 58 linhas.
   A explicação para isso é que existem 3 linhas em TB_EMPREGADO
   com COD_CARGO inválido, inexistente em TB_DEPARTAMENTO.
*/
```

Uma cláusula **OUTER JOIN** retorna todas as linhas de uma das tabelas presentes em uma cláusula **FROM**. Dependendo da tabela (ou tabelas) cujos dados são retornados, podemos definir alguns tipos de **OUTER JOIN**, como veremos a seguir.

- **LEFT JOIN**

A cláusula **LEFT JOIN** ou **LEFT OUTER JOIN** permite obter não apenas os dados relacionados de duas tabelas, mas também os dados não relacionados encontrados na tabela à esquerda da cláusula **JOIN**. Ou seja, a tabela à esquerda sempre terá todos os seus dados retornados em uma cláusula **LEFT JOIN**. Caso não existam dados relacionados entre as tabelas à esquerda e à direita de **JOIN**, os valores resultantes de todas as colunas de lista de seleção da tabela à direita serão nulos.

Veja exemplos da utilização de **LEFT JOIN**:

```
/*
  OUTER JOIN: Exibe também as linhas que não tenham correspondência.
  No exemplo a seguir, mostramos TODAS as linhas da tabela
  que está à esquerda da palavra JOIN (TB_EMPREGADO)
*/
 --
SELECT  -- retorna 61 linhas
  E.CODFUN, E.NOME, E.COD_DEPTO, E.COD_CARGO, C.CARGO
FROM TB_EMPREGADO E
  LEFT OUTER JOIN TB_CARGO C ON E.COD_CARGO = C.COD_CARGO;
-- OU (a palavra OUTER é opcional)
SELECT  -- retorna 61 linhas
  E.CODFUN, E.NOME, E.COD_DEPTO, E.COD_CARGO, C.CARGO
FROM TB_EMPREGADO E
  LEFT JOIN TB_CARGO C ON E.COD_CARGO = C.COD_CARGO;
-- Observe o resultado e veja que existem 3 empregados
-- com CARGO = NULL porque o campo COD_CARGO não foi preenchido
```

O **SELECT** a seguir verifica, na tabela **TB_EMPREGADO**, os empregados que não possuem um código de departamento (**COD_DEPTO**) válido:

```
-- Filtrar somente os registros não correspondentes
SELECT  -- retorna 3 linhas
  E.CODFUN, E.NOME, E.COD_DEPTO, E.COD_CARGO, C.CARGO
FROM TB_EMPREGADO E
  LEFT JOIN TB_CARGO C ON E.COD_CARGO = C.COD_CARGO
WHERE C.COD_CARGO IS NULL;
```

- **RIGHT JOIN**

Ao contrário da **LEFT OUTER JOIN**, a cláusula **RIGHT JOIN** ou **RIGHT OUTER JOIN** retorna todos os dados encontrados na tabela à direita de **JOIN**. Caso não existam dados associados entre as tabelas à esquerda e à direita de **JOIN**, serão retornados valores nulos.

Veja o seguinte uso de **RIGHT JOIN**. Da mesma forma que existem empregados que não possuem um **COD_DEPTO** válido, podemos verificar se existe algum departamento sem nenhum empregado cadastrado. Nesse caso, deveremos exibir todos os registros da tabela que está à direita (**RIGHT**) da palavra **JOIN**, ou seja, da tabela **TB_DEPARTAMENTO**:

```
SELECT
  E.CODFUN, E.NOME, E.COD_DEPTO, E.COD_CARGO, D.DEPTO
FROM TB_EMPREGADO E RIGHT JOIN TB_DEPARTAMENTO D ON E.COD_DEPTO =
D.COD_DEPTO;
-- O resultado terá 2 departamentos que
-- não retornaram dados de empregados.
```

```
-- Filtrar somente os registros não correspondentes
SELECT
  E.CODFUN, E.NOME, E.COD_DEPTO, E.COD_CARGO, D.DEPTO
FROM TB_EMPREGADO E RIGHT JOIN TB_DEPARTAMENTO D ON E.COD_DEPTO =
D.COD_DEPTO
WHERE E.COD_DEPTO IS NULL;
```

- **FULL JOIN**

Todas as linhas da tabela à esquerda de **JOIN** e da tabela à direita serão retornadas pela cláusula **FULL JOIN** ou **FULL OUTER JOIN**. Caso uma linha de dados não esteja associada a qualquer linha da outra tabela, os valores das colunas da lista de seleção serão nulos. Caso contrário, os valores obtidos serão baseados nas tabelas utilizadas como referência.

A seguir, é exemplificada a utilização de **FULL JOIN**:

```
-- FULL JOIN une o LEFT e o RIGHT JOIN
SELECT
  E.CODFUN, E.NOME, E.COD_DEPTO, E.COD_CARGO, C.CARGO
FROM TB_EMPREGADO E FULL JOIN TB_CARGO C ON E.COD_CARGO = C.COD_CAR-
GO;
-- Observe o resultado e veja que existem 2 departamentos que
-- não retornaram dados de empregados.
```

```
-- Filtrar somente os registros não correspondentes
SELECT
  E.CODFUN, E.NOME, E.COD_DEPTO, E.COD_CARGO, C.CARGO
FROM TB_EMPREGADO E FULL JOIN TB_CARGO C ON E.COD_CARGO = C.COD_CAR-
GO
WHERE E.COD_CARGO IS NULL OR C.COD_CARGO IS NULL;
```

1.4. CROSS JOIN

Todos os dados da tabela à esquerda de **JOIN** são cruzados com os dados da tabela à direita de **JOIN**, ao utilizarmos **CROSS JOIN**. As possíveis combinações de linhas em todas as tabelas são conhecidas como produto cartesiano. O tamanho do produto cartesiano será definido pelo número de linhas na primeira tabela multiplicado pelo número de linhas na segunda tabela. É possível cruzar informações de duas ou mais tabelas.

Quando **CROSS JOIN** não possui uma cláusula **WHERE**, gera um produto cartesiano das tabelas envolvidas. Se adicionarmos uma cláusula **WHERE**, **CROSS JOIN** se comportará como uma **INNER JOIN**.

A seguir, temos um exemplo da utilização de **CROSS JOIN**:

```
-- CROSS JOIN
SELECT  -- retorna 854 linhas
  E.CODFUN, E.NOME, E.COD_DEPTO, E.COD_CARGO, D.DEPTO
FROM TB_EMPREGADO E CROSS JOIN TB_DEPARTAMENTO D;
```

A **CROSS JOIN** deve ser utilizada apenas quando for realmente necessário um produto cartesiano, já que o resultado gerado pode ser muito grande.

Pontos principais

Atente para os tópicos a seguir. Eles devem ser estudados com muita atenção, pois representam os pontos mais importantes da leitura.

- A associação de tabelas pode ser realizada, por exemplo, para converter em informação os dados encontrados em duas ou mais tabelas. As tabelas podem ser combinadas por meio de uma condição ou um grupo de condições de junção;

- É importante ressaltar que as tabelas devem ser associadas em pares, embora seja possível utilizar um único comando para combinar várias tabelas. Um procedimento muito comum é a associação da chave primária da primeira tabela com a chave estrangeira da segunda tabela;

- **JOIN** é uma cláusula que permite a associação entre várias tabelas, com base na relação existente entre elas. Por meio dessa cláusula, os dados de uma tabela são utilizados para selecionar dados pertencentes à outra tabela;

- Há diversos tipos de **JOIN**: **INNER JOIN**, **OUTER JOIN** (**LEFT JOIN**, **RIGHT JOIN** e **FULL JOIN**) e **CROSS JOIN**.