Programação em Banco de Dados

Autoria: Douglas Fugita de Oliveira Cezar

Tema 02

Cconsultas e União de Consultas

Anhanguera

Tema 02

Consultas e União de Consultas

Autoria: Douglas Fugita de Oliveira Cezar

Como citar esse documento:

CEZAR, Douglas Fugita de Oliveira. *Programação em Banco de Dados:* Consultas e União de Consultas. Caderno de Atividades. Anhanguera Publicações: Valinhos, 2015.

Índice

CONVITE À LEITURA
Pág. 3

POR DENTRO DO TEMA
Pág. 3

ACOMPANHE NA WEB
Pág. 16

AGORA É A SUA VEZ
Pág. 17

FINALIZANDO
Pág. 19

REFERÊNCIAS
Pág. 19

GLOSSÁRIO
Pág. 19

GABARITO
Pág. 20

CONVITEÀLEITURA

Neste caderno de atividades você terá o contato com as estruturas que possibilitarão a recuperação das informações que estão inseridas na base de dados.

A forma como os dados armazenados serão visualizados é muito importante e, por isso, você será introduzido a formas de agrupamento de dados, filtragem de registros por campos específicos e também operações como soma, média, valor máximo e mínimo, contagem, etc.

Dados relacionados que estão em diferentes tabelas também serão relacionados através de comandos específicos, e isto também terá um papel bastante importante na forma de visualização pelo usuário final.

PORDENTRODOTEMA

Consultas e União de Consultas

Uma linguagem de programação de banco de dados pode ser dividida em três grandes grupos: a linguagem de definição de dados (DDL – *Data Definition Language*), a linguagem de controle de dados (DCL – *Data Control Language*) e a linguagem de manipulação de dados (DML – *Data Manipulation Language*).

A DML contém uma série de comandos responsáveis por inserir, excluir, alterar e consultar os dados que estão/serão armazenados no banco de dados.

Para este caderno, será considerado que as tabelas já possuem dados e o quadro 2.1 mostrará os dados que populam a tabela Município.

Quadro 2. 1 – Dados populados na tabela Município

ID_Municipio	Nome Municipio	UF_Municipio
1	Sao Paulo	SP
2	Rio de Janeiro	RJ
3	Belo Horizonte	MG
4	Curitiba	PR
5	Campinas	SP
6	Angra dos Reis	RJ
7	Niteroi	RJ
8	Salvador	BA
9	Feira de Santana	BA
10	Porto Seguro	BA
11	Guarulhos	SP
12	Cuiaba	MT
13	Barra do Garça	MT
14	Porto Alegre	RS
15	Uruguaiana	RS
16	Caxias do Sul	RS

Na linguagem de programação SQL, a forma de você exibir os dados de uma ou mais tabelas, assim como mostrados na figura 2.1, é através do comando SELECT. Para o resultado acima, deverá ser executado o comando como mostrado no quadro 2.2.

Quadro 2. 2 – Comando SELECT *

```
SELECT  *
FROM [dbo].[Municipio]
```

O “*” indica que todos os campos da tabela indicada na cláusula FROM devem ser considerados na consulta. Outra forma de obter o mesmo resultado é indicar um a um, todos os campos que devem fazer parte da consulta, como mostrado no quadro 2.3.

Quadro 2.3 – SELECT com campos explícitos

```
SELECT [ID_Municipio], [Nome Municipio], [UF_Municipio]
  FROM [dbo].[Municipio]
```

Sendo assim, as consultas podem retornar os dados somente para os campos desejados, sendo possível omitir campos que não são necessários para a obtenção do resultado. Um exemplo seria identificar na tabela Município quais são os estados cujas cidades já foram armazenadas, e o comando para isso é mostrado no quadro 2.4.

```
SELECT [UF_Município]
  FROM [dbo].[Município]
```

Quadro 2.4 – SELECT com campo específico e resultado

UF_Municipio
SP
RJ
MG
PR
SP
RJ
RJ
BA
BA
BA
SP
MT
MT
RS
RS
RS

No entanto, o resultado desta execução poderia ser melhor para o objetivo inicial, que era de encontrar os estados que estavam armazenados na tabela município e, para isso, pode ser utilizada a instrução DISTINCT. Esta instrução tem o papel de omitir todos os resultados duplicados, e o seu uso e resultado podem ser vistos no quadro 2.5.

```
SELECT DISTINCT [UF_Municipio]
  FROM [dbo].[Municipio]
```

Quadro 2.5 – Uso do DISTINCT

UF_Município
BA
MG
MT
PR
RJ
RS
SP

Pode ser que em uma consulta seja necessário que alguma condição específica seja atendida como, por exemplo, que sejam listadas todas as cidades que estejam no estado do Rio de Janeiro. Neste caso, a cláusula WHERE deve ser utilizada juntamente com argumentos que representem condições a serem satisfeitas.

As condições devem utilizar **operadores de comparação**, tais como igual a (=), maior que (>), menor que (<) e diferente de (<>). Há também operadores especiais como o BETWEEN, no qual pode ser especificado o intervalo de valores, o LIKE, no qual podem ser especificadas partes de uma palavra e o IN, que utiliza uma lista de valores para a comparação. Várias condições podem ser consideradas em uma mesma cláusula WHERE utilizando os operadores lógicos E (AND), OU (OR) ou NÃO (NOT).

Sendo assim, para filtrar as cidades armazenadas na tabela Municipios que pertencem ao estado do Rio de Janeiro, o comando seria mostrado no quadro 2.6.

```
SELECT [Nome_Municipio], [UF_Municipio]
  FROM [dbo].[Municipio]
  WHERE UF_Municipio = 'RJ'
```

Quadro 2. 6 – Uso da cláusula WHERE

Nome_Município	UF_Município
Rio de Janeiro	RJ
Angra dos Reis	RJ
Niterói	RJ

Embora todas as consultas mostradas até o momento tivessem como base uma única tabela, as consultas com o comando SELECT podem trazer resultados cruzando informações entre duas ou mais tabelas diferentes. Um caso para demonstrar esta necessidade é o apresentado no quadro 2.7, no qual uma consulta foi realizada para exibir o nome dos clientes armazenados e o município ao qual eles pertencem.

```
SELECT   ID_Cliente,
         Nome_Cliente,
         ID_Municipio
  FROM   [dbo].[Clientes]
```

Quadro 2.7 – SELECT separados mostrando o conteúdo de duas tabelas relacionadas

ID_Cliente	Nome_Cliente	ID_Municipio
1	CLIENTE A	3
2	CLIENTE B	2
3	CLIENTE C	2
4	CLIENTE D	1
5	CLIENTE E	4
6	CLIENTE F	5

SELECT * **FROM** Municipio

ID_Municipio	Nome_Municipio	UF_Municipio
1	Sao Paulo	SP
2	Rio de Janeiro	RJ
3	Belo Horizonte	MG
4	Curitiba	PR
5	Campinas	SP
6	Angra dos Reis	RJ
7	Niteroi	RJ
8	Salvador	BA
9	Feira de Santana	BA
10	Porto Seguro	BA
11	Guarulhos	SP
12	Cuiaba	MT
13	Barra do Garça	MT
14	Porto Alegre	RS
15	Uruguaiana	RS
16	Caxias do Sul	RS

Com estas tabelas podemos dizer que o “CLIENTE A” está no município de “Belo Horizonte” e que o “CLIENTE B”, por sua vez, no “Rio de Janeiro”. No entanto, seria muito mais **amigável** se estas informações estivessem disponíveis em uma mesma consulta e, para isso, podemos utilizar a chave “ID_Municipio” que está presente nas duas tabelas. Para isso, precisamos listar na cláusula FOR as duas tabelas envolvidas nesta consulta e o relacionamento será forçado através da cláusula WHERE, como mostrado no quadro 2.8.

```
SELECT   ID_Cliente,
         Nome_Cliente,
         Nome_Municipio,
         UF_Municipio
 FROM    Clientes c, Municipio m
WHERE    c.ID_Municipio = m.ID_Municipio
```

Quadro 2. 8 – SELECT relacionando duas tabelas

ID_Cliente	Nome_Cliente	Nome Municipio	UF_Municipio
4	CLIENTE D	Sao Paulo	SP
2	CLIENTE B	Rio de Janeiro	RJ
3	CLIENTE C	Rio de Janeiro	RJ
1	CLIENTE A	Belo Horizonte	MG
5	CLIENTE E	Curitiba	PR
6	CLIENTE F	Campinas	SP

Qualquer um dos campos envolvendo estas duas tabelas são passíveis de serem utilizados como filtros através da cláusula WHERE, por exemplo, caso seja necessário descobrir quais clientes estão no estado de São Paulo, como pode ser visto no quadro 2.9.

```
SELECT   ID_Cliente,Nome_Cliente,Nome_Municipio,UF_Municipio
 FROM    Clientes c, Municipio m
WHERE    c.ID_Municipio = m.ID_Municipio
AND      UF_Municipio = 'SP'
```

Quadro 2. 9 – SELECT utilizando campo da tabela relacionada como filtro

ID_Cliente	Nome_Cliente	Nome_Municipio	UF_Municipio
4	CLIENTE D	Sao Paulo	SP
6	CLIENTE F	Campinas	SP

Uma outra opção para o relacionamento entre tabelas é utilizar a cláusula JOIN, que explicita uma junção entre tabelas. Os JOINs podem ser divididos em dois tipos: OUTER JOIN e INNER JOIN.

A opção pelo OUTER JOIN deve ser feita sempre que todos os resultados de uma das tabelas devam ser mostrados, mesmo que alguns dos dados não tenham nenhum relacionamento com a outra tabela.

Para relacionar as tabelas com o OUTER JOIN, é necessário saber qual das tabelas terão todos os resultados preservados e isto deverá ser indicado através das três formas de OUTER JOIN: LEFT OUTER JOIN, RIGHT OUTER JOIN e FULL OUTER JOIN.

PORDENTRODOTEMA

O LEFT OUTER JOIN preserva todos os resultados da tabela à esquerda do comando e o RIGHT OUTER JOIN, por sua vez, preserva os resultados da tabela à direita. Já o FULL OUTER JOIN preserva o resultado das duas tabelas. Um exemplo da execução do OUTER JOIN pode ser visto no quadro 2.10.

```
SELECT   ID_Cliente,
         Nome_Cliente,
         Nome_Municipio,
         UF_Municipio
 FROM    Clientes c LEFT OUTER JOIN Municipio m
ON       ( c.ID_Municipio = m.ID_Municipio)
```

Quadro 2.10 – LEFT e RIGHT OUTER JOIN

ID_Cliente	Nome_Cliente	Nome_Municipio	UF_Municipio
1	CLIENTE A	Belo Horizonte	MG
2	CLIENTE B	Rio de Janeiro	RJ
3	CLIENTE C	Rio de Janeiro	RJ
4	CLIENTE D	Sao Paulo	SP
5	CLIENTE E	Curitiba	PR
6	CLIENTE F	Campinas	SP

```
SELECT   ID_Cliente,
         Nome_Cliente,
         Nome_Municipio,
         UF_Municipio
 FROM    Clientes c RIGHT OUTER JOIN Municipio m
ON       ( c.ID_Municipio = m.ID_Municipio)
```

Quadro 2.10 – LEFT e RIGHT OUTER JOIN

ID_Cliente	Nome_Cliente	Nome_Municipio	UF_Municipio
4	CLIENTE D	Sao Paulo	SP
2	CLIENTE B	Rio de Janeiro	RJ
3	CLIENTE C	Rio de Janeiro	RJ
1	CLIENTE A	Belo Horizonte	MG
5	CLIENTE E	Curitiba	PR
6	CLIENTE F	Campinas	SP
(null)	(null)	Angra dos Reis	RJ
(null)	(null)	Niteroi	RJ
(null)	(null)	Salvador	BA
(null)	(null)	Feira de Santana	BA
(null)	(null)	Porto Seguro	BA
(null)	(null)	Guarulhos	SP
(null)	(null)	Cuiaba	MT
(null)	(null)	Barra do Garça	MT
(null)	(null)	Porto Alegre	RS
(null)	(null)	Uruguaiana	RS
(null)	(null)	Caxias do Sul	RS

No quadro 2.10, pode ser observado que o LEFT OUTER JOIN preservou todos os resultados da tabela da esquerda (Clientes) e os relacionando com as tabelas da direita (Município). Já o RIGHT OUTER JOIN preservou todos os resultados da tabela da direita (Município), e os relacionou com os da tabela da esquerda (Clientes). Observe também que, mesmo que não haja relacionamento com a tabela, todos os registros são exibidos, como é o caso dos registros que estão com valor NULL (**nulo**).

Diferentemente do OUTER JOIN, o INNER JOIN somente exibe os registros no qual ambas as tabelas estão relacionadas, como pode ser visto no quadro 2.11.

```
SELECT    ID_Cliente,Nome_Cliente,Nome_Municipio,UF_Municipio
 FROM     Municipio m INNER JOIN Clientes c
ON        ( c.ID_Municipio = m.ID_Municipio)
```

Quadro 2.11 – INNER JOIN

ID_Cliente	Nome_Cliente	Nome_Municipio	UF_Municipio
4	CLIENTE D	Sao Paulo	SP
2	CLIENTE B	Rio de Janeiro	RJ
3	CLIENTE C	Rio de Janeiro	RJ
1	CLIENTE A	Belo Horizonte	MG
5	CLIENTE E	Curitiba	PR
6	CLIENTE F	Campinas	SP

O resultado desta execução é idêntico ao LEFT JOIN, mas não passa de coincidência. O LEFT JOIN foi auxiliado pela CONSTRAINT que obriga o preenchimento do campo ID_Municipio e, por isso, não é possível a criação de um registro na tabela Clientes que não tenha relacionamento com a tabela Municipio.

A utilização de JOINs, ou então do relacionamento forçado através das cláusulas FROM e WHERE, como você pôde acompanhar anteriormente, facilitou a visualização das informações, reunindo dados de várias tabelas em um único conjunto de dados.

Os resultados de uma consulta podem sofrer agregações, como por exemplo, a soma dos valores das vendas para cada cliente ou o valor médio de venda de cada produto. A estrutura do comando SELECT permite que sejam utilizadas funções de agregação como soma (SUM), contagem (COUNT) e Média (AVG).

Para exemplificar as funções de agregação, serão introduzidas as tabelas do quadro 2.12, juntamente com o conteúdo das mesmas.

```
SELECT * FROM [dbo].[NF]
GO
```

Quadro 2.12 – Tabelas Produto, Nota Fiscal de Venda e Itens da Nota Fiscal.

Numero_NF	Serie_NF	ID_Cliente	ID_Vendedor	DataEmissao_NF	DataSaida_NF
5	1	1	1	01/03/2014	02/03/2014
7	1	3	2	01/03/2014	06/03/2014
8	1	1	1	02/03/2014	02/03/2014
9	1	4	1	02/03/2014	04/03/2014
10	1	6	4	04/03/2014	06/03/2014
12	1	2	2	08/03/2014	08/03/2014

```
SELECT * FROM [dbo].[NFItens]
GO
```

Número_NF	Série_NF	Item_NFItens	ID_Produto	Qtdade_NFItens	PrecoVenda_NFItens
5	1	1	1	20	2
5	1	2	2	20	4
7	1	1	3	10	3,5
7	1	2	1	20	1,5
7	1	3	5	2	5
8	1	1	4	1	12
9	1	1	6	2	20,5
10	1	1	6	6	16,5
12	1	1	1	10	1,5

```
SELECT * FROM [dbo].[Produto]
GO
```

ID_Produto	Nome_Produto	ValorVenda_Produto	UnidadeMedida_Produto
1	Lapis	1,49	UN
2	Caneta azul	3,5	UN
3	Caderno 100fl	15	UN
4	Grampo	12,9	CX
5	Borracha	4,2	UN
6	Folha Sulfite	18,9	PCT

PORDENTRODOTEMA

Você pode observar que as tabelas do quadro 2.12 fazem referência a várias outras tabelas. Por exemplo, a tabela NF possui campos de chave estrangeira os quais referenciam as tabelas que armazenam dados de clientes e de vendedores. Para que faça sentido os dados apresentados pela função de agregação, as tabelas deverão ser relacionadas por JOINs, como pode ser visto no quadro 2.13.

```
SELECT    n.Numero_NF as [Numero NF],
          n.Serie_NF as [Serie NF],
          c.Nome_Cliente as Cliente,
          v.Nome_Vendedor as Vendedor,
          n.DataEmissao_NF as [Data de Emissao],
          p.Nome_Produto as [Produto],
          ni.Qtdade_NFItens as [Quantidade],
          ni.PrecoVenda_NFItens as [Preco de Venda]
 FROM NF n
 INNER JOIN NFItens ni
   ON      n.Numero_NF = ni.Numero_NF AND n.Serie_NF = ni.Serie_NF
 INNER JOIN Produto p
   ON      ni.ID_Produto = p.ID_Produto
 INNER JOIN Clientes c
   ON n.ID_Cliente = c.ID_Cliente
 INNER JOIN Vendedor v
   ON n.ID_Vendedor = v.ID_Vendedor
ORDER BY n.Serie_NF, n.Numero_NF
```

Quadro 2.13 – Resultado do JOIN das tabelas da figura 2.12 e seus relacionamentos

Numero NF	Serie NF	Cliente	Vendedor	Data de Emissao	Produto	Quantidade	Preco de Venda
5	1	CLIENTE A	HUGO	01/03/2014	Lapis	20	2
5	1	CLIENTE A	HUGO	01/03/2014	Caneta azul	20	4
7	1	CLIENTE C	LUIZ	01/03/2014	Caderno 100fl	10	3,5
7	1	CLIENTE C	LUIZ	01/03/2014	Lapis	20	1,5
7	1	CLIENTE C	LUIZ	01/03/2014	Borracha	2	5
8	1	CLIENTE A	HUGO	02/03/2014	Grampo	1	12
9	1	CLIENTE D	HUGO	02/03/2014	Folha Sulfite	2	20,5
10	1	CLIENTE F	DONALD	04/03/2014	Folha Sulfite	6	16,5
12	1	CLIENTE B	LUIZ	08/03/2014	Lapis	10	1,5

Com este JOIN apresentado no quadro 2.13, será feito uma soma do valor das notas fiscais emitidas através da função de agregação SUM e os valores agregados serão agrupados por cliente, como exibido no quadro 2.14.

```
SELECT    c.Nome_Cliente as Cliente,
      SUM (ni.PrecoVenda_NFItens * ni.Qtdade_NFItens) as [Preco de Venda]
 FROM     NF n
 INNER JOIN NFItens ni
   ON     n.Numero_NF = ni.Numero_NF AND n.Serie_NF = ni.Serie_NF
 INNER JOIN Produto p
   ON     ni.ID_Produto = p.ID_Produto
 INNER JOIN Clientes c
   ON     n.ID_Cliente = c.ID_Cliente
GROUP BY c.Nome_Cliente
ORDER BY c.Nome_Cliente
```

POR DENTRO DO TEMA

Quadro 2.14 – Resultado de operação SUM

Cliente	Preco de Venda
CLIENTE A	132
CLIENTE B	15
CLIENTE C	75
CLIENTE D	41
CLIENTE F	99

Ao utilizar uma função de agregação, note que somente os campos principais foram deixados no SELECT. Além disto, você também deve se atentar ao fato de que todos os campos pelo qual o resultado será agrupado deverá ser relacionado na cláusula GROUP BY, como exemplo do quadro 2.14.

Em um único SELECT pode haver quantas funções de agregação quanto necessárias, bem como agrupar por vários níveis diferentes.

ACOMPANHE NA WEB

Relação de funções de agregação do MS SQL Server

- Relação de funções de agregação do MS SQL Server. Neste site você pode verificar todas as funções de agregação, bem como exemplos de uso e sua sintaxe.

Link para acesso: <http://technet.microsoft.com/pt-br/library/ms173454.aspx>. Acesso em: 01 abr. 2014.

AGORAÉASUAVEZ

Instruções:

Agora, chegou a sua vez de exercitar seu aprendizado. A seguir, você encontrará algumas questões de múltipla escolha e dissertativas. Leia cuidadosamente os enunciados e atente-se para o que está sendo pedido.

Questão 1

Explique a importância de exibir as informações de forma amigável ao extrair os dados de um banco de dados.

Questão 2

Marque abaixo as funções de agregação:

a) ( ) INNER JOIN

b) ( ) SUM

c) ( ) ALTER

d) ( ) LEFT OUTER JOIN

e) ( ) AVG

f) ( ) COUNT

AGORA É A SUA VEZ

Questão 3

Para preservar todos os dados da primeira tabela, e exibir os dados da segunda tabela que estiverem relacionado com a primeira, deve ser utilizada a seguinte cláusula de agregação:

a) ( ) INNER JOIN

b) ( ) LEFT OUTER JOIN

c) ( ) RIGHT OUTER JOIN

d) ( ) FULL OUTER JOIN

e) ( ) Nenhuma das respostas anteriores.

Questão 4

Utilizando como base a estrutura de dados mostrada neste caderno, crie uma consulta na qual seja possível obter o valor médio de venda de cada produto.

Questão 5

Utilizando como base a estrutura de dados mostrada neste caderno, crie uma consulta na qual seja possível identificar quais itens foram comprados por cada cliente e qual a quantidade de compras daquele item.

FINALIZANDO

Neste caderno você teve a oportunidade de aprender sobre o comando SELECT, responsável por recuperar as informações armazenadas nas tabelas do banco de dados.

Foram apresentados conceitos introdutórios que devem ser expandidos através de leituras complementares, pois seu conhecimento é de vital importância para a utilização eficaz de um banco de dados.

REFERÊNCIAS

MICROSOFT SQL Server. *Funções de Agregação* (*Transact- SQL*). Disponível em: <http://technet.microsoft.com/pt-br/library/ms173454.aspx>. Acesso em: 01 jun. 2014

SILBERSCHATZ, Abrahan; KORTH, Henry; SUDARSHAN, S. *Sistema de Banco de Dados*. Elsevier, 2012.

GLOSSÁRIO

Operadores de comparação: elementos que comparam dois valores e retornam como resultado um valor TRUE (verdadeiro) ou FALSE (falso).

Amigável: que seja feito de forma que o uso da informação seja facilitado.

Nulo: campo que não possui nenhum valor.

GABARITO

Questão 1

Resposta: O dado precisa ser facilmente interpretado no momento em que ele é disponibilizado para o destinatário final e este é o motivo de ser necessário ser amigável.

Questão 2

Resposta: b, e, f

Questão 3

Resposta: A – A globalização se consolida a partir a chamada “revolução tecnológica” com o advento das tecnologias digitais, da ampliação dos meios de comunicação e do desenvolvimento científico de ponta. No modelo neoliberal, a soberania dos Estados é reduzida em detrimento do capital privado.

Questão 4

Resposta:

```
SELECT  p.Nome_Produto as Produto,
        avg(ni.PrecoVenda_NFItens) as [Preco de Venda]
 FROM NF n
 INNER JOIN NFItens ni
   ON n.Numero_NF = ni.Numero_NF AND n.Serie_NF = ni.Serie_NF
 INNER JOIN Produto p
   ON ni.ID_Produto = p.ID_Produto
 GROUP BY p.Nome_Produto
ORDER BY p.Nome_Produto
```

GABARITO

Questão 5

Resposta:

```
SELECT  c.Nome_Cliente as Cliente,
        p.Nome_Produto as Produto,
        COUNT(n.Numero_NF) as Contagem
 FROM NF n
 INNER JOIN NFItens ni
   ON n.Numero_NF = ni.Numero_NF AND n.Serie_NF = ni.Serie_NF
 INNER JOIN Produto p
   ON ni.ID_Produto = p.ID_Produto
 INNER JOIN Clientes c
   ON n.ID_Cliente = c.ID_Cliente
 INNER JOIN Vendedor v
   ON n.ID_Vendedor = v.ID_Vendedor
GROUP BY c.Nome_Cliente, p.Nome_Produto
ORDER BY c.Nome_Cliente
```


Anhanguera