Agrupando dados

- Funções de agregação;
- GROUP BY.

Esta Leitura Complementar refere-se ao conteúdo das Aulas 26 e 27.

Editora IMPACTA

1.1. Introdução

Nesta leitura, você aprenderá como a cláusula **GROUP BY** pode ser utilizada para agrupar vários dados, tornando mais prática sua sumarização. Também verá como utilizar funções de agregação para sumarizar dados e como a cláusula **GROUP BY** pode ser usada com a cláusula **HAVING** e com os operadores **ALL**, **WITH ROLLUP** e **CUBE**.

1.2. Funções de agregação

As funções de agregação fornecidas pelo SQL Server permitem sumarizar dados. Por meio delas, podemos somar valores, calcular médias e contar a quantidade de linhas sumarizadas. Os cálculos realizados pelas funções de agregação são feitos com base em um conjunto ou grupo de valores, mas retornam um único valor.

Para obter os valores sobre os quais poderão realizar os cálculos, as funções de agregação geralmente são utilizadas com a cláusula **GROUP BY**. Quando não há uma cláusula **GROUP BY**, os grupos de valores podem ser obtidos de uma tabela inteira filtrada pela cláusula **WHERE**.

Ao utilizar funções de agregação, é preciso prestar atenção a valores **NULL**. A maioria das funções ignora esses valores, o que pode gerar resultados inesperados.

Além de utilizar as funções de agregação fornecidas pelo SQL Server, há a possibilidade de criar funções personalizadas.

1.2.1. Tipos de função de agregação

A seguir, são descritas as principais funções de agregação fornecidas pelo SQL Server:

- **AVG ([ALL | DISTINCT] expressão)**

Esta função calcula o valor médio do parâmetro **expressão** em determinado grupo, ignorando valores **NULL**. Os parâmetros opcionais **ALL** e **DISTINCT** são utilizados para especificar se a agregação será executada em todos os valores do campo (**ALL**) ou aplicada apenas sobre valores distintos (**DISTINCT**).

Veja um exemplo:

```
USE PEDIDOS;
-- neste caso, o grupo corresponde a toda a tabela TB_EMPREGADO
SELECT AVG(SALARIO) AS SALARIO_MEDIO
FROM TB_EMPREGADO;
-- neste caso, o grupo corresponde aos empregados com
-- COD_DEPTO = 2
SELECT AVG(SALARIO) AS SALARIO_MEDIO FROM TB_EMPREGADO
WHERE COD_DEPTO = 2;
```

- **COUNT ({ [ALL | DISTINCT] expressão | * })**

Esta função é utilizada para retornar a quantidade de registros existentes não nulos em um grupo. Ao especificar o parâmetro **ALL**, a função não retornará valores nulos. Os parâmetros de **COUNT** têm a mesma função dos parâmetros de **AVG**.

Veja um exemplo:

```
-- neste caso, o grupo corresponde a toda a tabela TB_EMPREGADO
SELECT COUNT(*) AS QTD_EMPREGADOS
FROM TB_EMPREGADO;
-- neste caso, o grupo corresponde aos empregados com
-- COD_DEPTO = 2
SELECT COUNT(COD_DEPTO) AS QTD_EMPREGADOS FROM TB_EMPREGADO
WHERE COD_DEPTO = 2;
```

Se colocarmos o nome de um campo como argumento da função **COUNT**, não serão contados os registros em que o conteúdo desse campo seja **NULL**.

- **MIN ([ALL | DISTINCT] expressão)**

Esta função retorna o menor valor não nulo de **expressão** existente em um grupo. Os parâmetros de **MIN** têm a mesma função dos parâmetros de **AVG**.

Veja um exemplo:

```
-- neste caso, o grupo corresponde a toda a tabela TB_EMPREGADO
SELECT MIN(SALARIO) AS MENOR_SALARIO FROM TB_EMPREGADO;
-- neste caso, o grupo corresponde aos empregados com
-- COD_DEPTO = 2
SELECT MIN(SALARIO) AS MENOR_SALARIO FROM TB_EMPREGADO
WHERE COD_DEPTO = 2
```

- **MAX ([ALL | DISTINCT] expressão)**

Esta função retorna o maior valor não nulo de **expressão** existente em um grupo. Os parâmetros de **MAX** têm a mesma função dos parâmetros de **AVG**.

Veja um exemplo:

```
-- neste caso, o grupo corresponde a toda a tabela TB_EMPREGADO
SELECT MAX(SALARIO) AS MAIOR_SALARIO
FROM TB_EMPREGADO;
-- neste caso, o grupo corresponde aos empregados com
-- COD_DEPTO = 2
SELECT MAX(SALARIO) AS MAIOR_SALARIO FROM TB_EMPREGADO
WHERE COD_DEPTO = 2
```

- **SUM ([ALL | DISTINCT] expressão)**

Esta função realiza a soma de todos os valores não nulos na **expressão** em um determinado grupo. Os parâmetros de **SUM** têm a mesma função dos parâmetros de **AVG**.

Veja um exemplo:

```
-- neste caso, o grupo corresponde a toda a tabela TB_EMPREGADO
SELECT SUM(SALARIO) AS SOMA_SALARIOS
FROM TB_EMPREGADO;
-- neste caso, o grupo corresponde aos empregados com
-- COD_DEPTO = 2
SELECT SUM(SALARIO) AS SOMA_SALARIOS FROM TB_EMPREGADO
WHERE COD_DEPTO = 2
```

1.3. GROUP BY

Utilizando a cláusula **GROUP BY**, é possível agrupar diversos registros com base em uma ou mais colunas da tabela.

Esta cláusula é responsável por determinar em quais grupos devem ser colocadas as linhas de saída. Caso a cláusula **SELECT** contenha funções de agregação, a cláusula **GROUP BY** realiza um cálculo a fim de chegar ao valor sumário para cada um dos grupos.

Quando especificar a cláusula **GROUP BY**, deve ocorrer uma das seguintes situações: a expressão **GROUP BY** deve ser correspondente à expressão da lista de seleção; ou cada uma das colunas presentes em uma expressão não agregada na lista de seleção deve ser adicionada à lista de **GROUP BY**.

Ao utilizar uma cláusula **GROUP BY**, todas as colunas na lista **SELECT** que não são parte de uma expressão agregada serão usadas para agrupar os resultados obtidos. Para não agrupar os resultados em uma coluna, não se deve colocá-los na lista **SELECT**. Valores **NULL** são agrupados todos em uma mesma coluna, já que são considerados iguais.

Quando utilizamos a cláusula **GROUP BY**, mas não empregamos a cláusula **ORDER BY**, o resultado obtido são os grupos em ordem aleatória, visto que é essencial o uso de **ORDER BY** para determinar a ordem de apresentação dos dados.

Observe, a seguir, a sintaxe da cláusula **GROUP BY**:

```
[ GROUP BY [ ALL ] expressao_group_by [ ,...n ]
[HAVING <condicaoFiltroGrupo>]]
```

Em que:

- **ALL**: É a palavra que determina a inclusão de todos os grupos e conjuntos de resultados. Vale destacar que valores nulos são retornados às colunas resultantes dos grupos que não correspondem aos critérios de busca quando **ALL** é especificada;

- **expressao_group_by**: Também conhecida como coluna agrupada, é uma expressão na qual o agrupamento é realizado. Pode ser especificada como uma coluna ou como uma expressão não agregada que faz referência à coluna que a cláusula **FROM** retornou, mas não é possível especificá-la como um alias de coluna determinado na lista de seleção. Além disso, não podemos utilizar em uma **expressao_group_by** as colunas de um dos seguintes tipos: **image**, **text** e **ntext**;

- **[HAVING <condicaoFiltroGrupo>]**: Determina uma condição de busca para um grupo ou um conjunto de registros. Essa condição é especificada em **<condicaoFiltroGrupo>**.

Observe o resultado da instrução:

```
SELECT COD_DEPTO, SALARIO FROM TB_EMPREGADO ORDER BY COD_DEPTO;
```

	COD_DEPTO	SALARIO
4	1	890.00
5	1	4500.00
6	1	800.00
7	1	4500.00
8	1	890.00
9	1	4500.00
10	1	3300.00
11	1	8300.00
12	1	5000.00
13	1	3300.00
14	1	1200.00
15	1	8300.00
16	2	8300.00
17	2	800.00
18	2	600.00
19	2	800.00
20	2	4500.00
21	2	3330.00
22	2	600.00
23	2	500.00

Veja que o campo **COD_DEPTO** se repete e, portanto, forma grupos. Em uma situação dessas, podemos gerar totalizações para cada um dos grupos utilizando a cláusula **GROUP BY**.

A seguir, veja exemplos da utilização de **GROUP BY**:

- **Exemplo 1**

```
-- Total de salário de cada departamento
SELECT COD_DEPTO, SUM( SALARIO ) AS TOT_SAL
FROM TB_EMPREGADO
GROUP BY COD_DEPTO
ORDER BY TOT_SAL;
```

- **Exemplo 2**

```
-- GROUP BY + JOIN
SELECT E.COD_DEPTO, D.DEPTO, SUM( E.SALARIO ) AS TOT_SAL
FROM TB_EMPREGADO E
     JOIN TB_DEPARTAMENTO D ON E.COD_DEPTO = D.COD_DEPTO
GROUP BY E.COD_DEPTO, D.DEPTO
ORDER BY TOT_SAL;
```

- **Exemplo 3**

```
-- Consulta do tipo RANKING utilizando TOP n + ORDER BY
-- Os 5 departamentos que mais gastam com salários
SELECT TOP 5 E.COD_DEPTO, D.DEPTO, SUM( E.SALARIO ) AS TOT_SAL
FROM TB_EMPREGADO E
     JOIN TB_DEPARTAMENTO D ON E.COD_DEPTO = D.COD_DEPTO
GROUP BY E.COD_DEPTO, D.DEPTO
ORDER BY TOT_SAL DESC;
```

- **Exemplo 4**

```
-- Os 10 clientes que mais compraram em Janeiro de 2014
SELECT TOP 10 C.CODCLI, C.NOME, SUM(P.VLR_TOTAL) AS TOT_COMPRADO
FROM TB_PEDIDO P JOIN TB_CLIENTE C ON P.CODCLI = C.CODCLI
WHERE P.DATA_EMISSAO BETWEEN '2014.1.1' AND '2014.1.31'
GROUP BY C.CODCLI, C.NOME
ORDER BY TOT_COMPRADO DESC;
```

1.3.1. Utilizando ALL

ALL inclui todos os grupos e conjuntos de resultados. A seguir, veja um exemplo da utilização de **ALL**:

```
-- Clientes que compraram em janeiro de 2014. Veremos que
-- todas as linhas do resultado terão um total não nulo.
SELECT C.CODCLI, C.NOME, SUM(P.VLR_TOTAL) AS TOT_COMPRADO
FROM TB_PEDIDO P JOIN TB_CLIENTE C ON P.CODCLI = C.CODCLI
WHERE P.DATA_EMISSAO BETWEEN '2014.1.1' AND '2014.1.31'
GROUP BY C.CODCLI, C.NOME;
```

```
-- Neste caso, aparecerão também os clientes que não
-- compraram. Totais estarão nulos.
SELECT C.CODCLI, C.NOME, SUM(P.VLR_TOTAL) AS TOT_COMPRADO
FROM TB_PEDIDO P JOIN TB_CLIENTE C ON P.CODCLI = C.CODCLI
WHERE P.DATA_EMISSAO BETWEEN '2014.1.1' AND '2014.1.31'
GROUP BY ALL C.CODCLI, C.NOME;
```

1.3.2. Utilizando HAVING

A cláusula **HAVING** determina uma condição de busca para um grupo ou um conjunto de registros, definindo critérios para limitar os resultados obtidos a partir do agrupamento de registros. Ela é utilizada para estreitar um conjunto de resultados por meio de critérios e valores agregados e para filtrar linhas após o agrupamento ter sido feito e antes dos resultados serem retornados ao cliente.

É importante lembrar que essa cláusula só pode ser utilizada em parceria com **GROUP BY**. Se uma consulta é feita sem **GROUP BY**, a cláusula **HAVING** pode ser usada como cláusula **WHERE**.

A cláusula **HAVING** é diferente da cláusula **WHERE**. Esta última restringe os resultados obtidos após a aplicação da cláusula **FROM**, ao passo que a cláusula **HAVING** filtra o retorno do agrupamento.

O código do exemplo a seguir utiliza a cláusula **HAVING** para consultar os departamentos que totalizam mais de R$100.000,00 em salários:

```
SELECT E.COD_DEPTO, D.DEPTO, SUM( E.SALARIO ) AS TOT_SAL
FROM TB_EMPREGADO E
     JOIN TB_DEPARTAMENTO D ON E.COD_DEPTO = D.COD_DEPTO
GROUP BY E.COD_DEPTO, D.DEPTO HAVING SUM(E.SALARIO) > 100000
ORDER BY TOT_SAL;
```

O próximo código consulta os clientes que compraram mais de R$5.000,00 em janeiro de 2014:

```
SELECT C.CODCLI, C.NOME, SUM(P.VLR_TOTAL) AS TOT_COMPRADO
FROM TB_PEDIDO P JOIN TB_CLIENTE C ON P.CODCLI = C.CODCLI
WHERE P.DATA_EMISSAO BETWEEN '2014.1.1' AND '2014.1.31'
GROUP BY C.CODCLI, C.NOME HAVING SUM(P.VLR_TOTAL) > 5.000
ORDER BY TOT_COMPRADO;
```

Já o próximo código consulta os clientes que não realizaram compras em janeiro de 2014:

```
SELECT C.CODCLI, C.NOME, SUM(P.VLR_TOTAL) AS TOT_COMPRADO
FROM TB_PEDIDO P JOIN TB_CLIENTE C ON P.CODCLI = C.CODCLI
WHERE P.DATA_EMISSAO BETWEEN '2014.1.1' AND '2014.1.31'
GROUP BY ALL C.CODCLI, C.NOME HAVING SUM(P.VLR_TOTAL) IS NULL;
```

1.3.3. Utilizando WITH ROLLUP

Esta cláusula determina que, além das linhas normalmente retornadas por **GROUP BY**, também sejam obtidas como resultado as linhas de sumário. O sumário dos grupos é feito em uma ordem hierárquica, a partir do nível mais baixo até o mais alto. A ordem que define a hierarquia do grupo é determinada pela ordem na qual são definidas as colunas agrupadas. Caso essa ordem seja alterada, a quantidade de linhas produzidas pode ser afetada.

Utilizada em parceria com a cláusula **GROUP BY**, a cláusula **WITH ROLLUP** acrescenta uma linha na qual são exibidos os subtotais e totais dos registros já distribuídos em colunas agrupadas.

Suponha que esteja trabalhando em um banco de dados com as seguintes características:

- O cadastro de produtos está organizado em categorias (tipos);
- Há vários produtos que pertencem a uma mesma categoria;
- As categorias de produtos são armazenadas na tabela **TIPOPRODUTO**.

O **SELECT** a seguir mostra as vendas de cada categoria de produto (**TIPOPRODUTO**) que os vendedores (tabela **TB_VENDEDOR**) realizaram para cada cliente (tabela **TB_CLIENTE**) no primeiro semestre de 2013:

```
SELECT
  V.NOME AS VENDEDOR, C.NOME AS CLIENTE,
  T.TIPO AS TIPO_PRODUTO, SUM( I.QUANTIDADE ) AS QTD_TOT
FROM
  TB_PEDIDO Pe
  JOIN TB_CLIENTE C ON Pe.CODCLI = C.CODCLI
  JOIN TB_VENDEDOR V ON Pe.CODVEN = V.CODVEN
  JOIN TB_ITENSPEDIDO I ON Pe.NUM_PEDIDO = I.NUM_PEDIDO
  JOIN TB_PRODUTO Pr ON I.ID_PRODUTO = Pr.ID_PRODUTO
  JOIN TB_TIPOPRODUTO T ON Pr.COD_TIPO = T.COD_TIPO
WHERE Pe.DATA_EMISSAO BETWEEN '2013.1.1' AND '2013.6.30'
GROUP BY V.NOME , C.NOME, T.TIPO;
```

Suponha que o resultado retornado seja o seguinte:

	VENDEDOR	CLIENTE	TIPO_PRODUTO	QTD_TOT
1	CELSON MARTINS	3R (ARISTEU,ADALTON)	ACES.CHAVEIRO	111
2	CELSON MARTINS	3R (ARISTEU,ADALTON)	ACESSORIOS P/CANETA	170
3	CELSON MARTINS	3R (ARISTEU,ADALTON)	CANETA	600
4	CELSON MARTINS	3R (ARISTEU,ADALTON)	CHAVEIRO	513
5	CELSON MARTINS	3R (ARISTEU,ADALTON)	MATL DIVERSOS	982
6	CELSON MARTINS	3R (ARISTEU,ADALTON)	PORTA LAPIS	1352
7	CELSON MARTINS	3R (ARISTEU,ADALTON)	REGUA	19
8	CELSON MARTINS	ALLAN HEBERT RELOGIOS E PRESENTES	CANETA	153
9	CELSON MARTINS	ALLAN HEBERT RELOGIOS E PRESENTES	CHAVEIRO	295
10	CELSON MARTINS	ALLAN HEBERT RELOGIOS E PRESENTES	MATL DIVERSOS	211
11	CELSON MARTINS	ALLAN HEBERT RELOGIOS E PRESENTES	PORTA LAPIS	162

Note que um dos vendedores é **CELSON MARTINS** e que alguns de seus clientes são **3R (ARISTEU.ADALTON)** e **ALLAN HEBERT RELOGIOS E PRESENTES**. Também é possível perceber, por exemplo, que o cliente **3R (ARISTEU.ADALTON)** comprou **111** unidades de produtos do tipo **ACES.CHAVEIRO** do vendedor **CELSON MARTINS**, enquanto **ALLAN HEBERT RELOGIOS E PRESENTES** comprou **153** produtos do tipo **CANETA** do mesmo vendedor.

Terminados os registros de vendas do **CELSON MARTINS**, iniciam-se os registros de venda do próximo vendedor, e assim por diante, como você pode ver a seguir:

	VENDEDOR	CLIENTE	TIPO_PRODUTO	QTD_TOT
168	CELSON MARTINS	VILSON CORDEIRO DO ROSARIO	REGUA	143
169	CELSON MARTINS	VILSON CORDEIRO DO ROSARIO	YO-YO	516
170	DORINHA	3R (ARISTEU,ADALTON)	CANETA	750
171	DORINHA	3R (ARISTEU,ADALTON)	PORTA LAPIS	30

Agora, acrescente a cláusula **WITH ROLLUP** após a linha de **GROUP BY**. O código anterior ficará assim:

```
SELECT
  V.NOME AS VENDEDOR, C.NOME AS CLIENTE,
  T.TIPO AS TIPO_PRODUTO, SUM( I.QUANTIDADE ) AS QTD_TOT
FROM   TB_PEDIDO Pe
  JOIN TB_CLIENTE C ON Pe.CODCLI = C.CODCLI
  JOIN TB_VENDEDOR V ON Pe.CODVEN = V.CODVEN
  JOIN TB_ITENSPEDIDO I ON Pe.NUM_PEDIDO = I.NUM_PEDIDO
  JOIN TB_PRODUTO Pr ON I.ID_PRODUTO = Pr.ID_PRODUTO
  JOIN TB_TIPOPRODUTO T ON Pr.COD_TIPO = T.COD_TIPO
WHERE Pe.DATA_EMISSAO BETWEEN '2013.1.1' AND '2013.6.30'
GROUP BY V.NOME , C.NOME, T.TIPO
WITH ROLLUP;
```

O resultado será o seguinte:

	VENDEDOR	CLIENTE	TIPO_PRODUTO	QTD_TOT
1	CELSON MARTINS	3R (ARISTEU,ADALTON)	ACES.CHAVEIRO	111
2	CELSON MARTINS	3R (ARISTEU,ADALTON)	ACESSORIOS P/CANETA	170
3	CELSON MARTINS	3R (ARISTEU,ADALTON)	CANETA	600
4	CELSON MARTINS	3R (ARISTEU,ADALTON)	CHAVEIRO	513
5	CELSON MARTINS	3R (ARISTEU,ADALTON)	MATL DIVERSOS	982
6	CELSON MARTINS	3R (ARISTEU,ADALTON)	PORTA LAPIS	1352
7	CELSON MARTINS	3R (ARISTEU,ADALTON)	REGUA	19
8	CELSON MARTINS	3R (ARISTEU,ADALTON)	NULL	3747
9	CELSON MARTINS	ALLAN HEBERT RELOGIOS E PRESENTES	CANETA	153
10	CELSON MARTINS	ALLAN HEBERT RELOGIOS E PRESENTES	CHAVEIRO	295
11	CELSON MARTINS	ALLAN HEBERT RELOGIOS E PRESENTES	MATL DIVERSOS	211
12	CELSON MARTINS	ALLAN HEBERT RELOGIOS E PRESENTES	PORTA LAPIS	162
13	CELSON MARTINS	ALLAN HEBERT RELOGIOS E PRESENTES	NULL	821

Observe na figura anterior que, após a última linha do vendedor **CELSON MARTINS**, existe um **NULL** na coluna **TIPO_PRODUTO**, o que significa que o valor apresentado na coluna **QTD_TOT** corresponde ao total vendido por esse vendedor para o cliente **3R (ARISTEU.ADALTON)**, ou seja, a coluna **TIPO_PRODUTO** (**NULL**) não foi considerada para a totalização. Isso se repetirá até o último cliente que comprou de **CELSON MARTINS**.

Antes de iniciar as totalizações do vendedor seguinte, existe uma linha na qual apenas o nome do vendedor não é **NULL**, o que significa que o total apresentado na coluna **QTD_TOT** representa o total vendido pelo vendedor **CELSON MARTINS**, ou seja, **55912** produtos, independentemente do cliente e do tipo de produto:

	VENDEDOR	CLIENTE	TIPO_PRODUTO	QTD_TOT
213	CELSON MARTINS	VILSON CORDEIRO DO ROSARIO	YO-YO	516
214	CELSON MARTINS	VILSON CORDEIRO DO ROSARIO	NULL	3266
215	CELSON MARTINS	NULL	NULL	55912
216	DORINHA	3R (ARISTEU,ADALTON)	CANETA	750
217	DORINHA	3R (ARISTEU,ADALTON)	PORTA LAPIS	30

Na última linha do resultado, temos **NULL** nas três primeiras colunas. O total corresponde ao total vendido (**1022534**) no período mencionado, independentemente do vendedor, do cliente ou do tipo de produto:

1931	MARCELO	WALEU IND E COM DE PLASTICOS LTDA	PORTA LAPIS	110
1932	MARCELO	WALEU IND E COM DE PLASTICOS LTDA	NULL	805
1933	MARCELO	NULL	NULL	216732
1934	NULL	NULL	NULL	1022534

1.3.4. Utilizando WITH CUBE

A cláusula **WITH CUBE** tem a finalidade de determinar que as linhas de sumário sejam inseridas no conjunto de resultados. A linha de sumário é retornada para cada combinação possível de grupos e de subgrupos no conjunto de resultados.

Visto que a cláusula **WITH CUBE** é responsável por retornar todas as combinações possíveis de grupos e de subgrupos, a quantidade de linhas não está relacionada à ordem em que são determinadas as colunas de agrupamento, sendo, portanto, mantida a quantidade de linhas já apresentada.

A quantidade de linhas de sumário no conjunto de resultados é especificada de acordo com a quantidade de colunas incluídas na cláusula **GROUP BY**. Cada uma dessas colunas é vinculada sob o valor **NULL** do agrupamento, o qual é aplicado a todas as outras colunas.

A cláusula **WITH CUBE**, em conjunto com **GROUP BY**, gera totais e subtotais, apresentando vários agrupamentos de acordo com as colunas definidas com **GROUP BY**.

Para explicar o que faz **WITH CUBE**, considere o exemplo utilizado para **WITH ROLLUP**. No lugar desta última cláusula, utilize **WITH CUBE**. O código ficará assim:

```
SELECT
  V.NOME AS VENDEDOR, C.NOME AS CLIENTE,
  T.TIPO AS TIPO_PRODUTO, SUM( I.QUANTIDADE ) AS QTD_TOT
FROM  TB_PEDIDO Pe
JOIN TB_CLIENTE C ON Pe.CODCLI = C.CODCLI
JOIN TB_VENDEDOR V ON Pe.CODVEN = V.CODVEN
JOIN TB_ITENSPEDIDO I ON Pe.NUM_PEDIDO = I.NUM_PEDIDO
JOIN TB_PRODUTO Pr ON I.ID_PRODUTO = Pr.ID_PRODUTO
JOIN TB_TIPOPRODUTO T ON Pr.COD_TIPO = T.COD_TIPO
WHERE Pe.DATA_EMISSAO BETWEEN '2013.1.1' AND '2013.6.30'
GROUP BY V.NOME , C.NOME, T.TIPO
WITH CUBE;
```

O resultado é o seguinte:

	VENDEDOR	CLIENTE	TIPO_PRODUTO	QTD_TOT
1	DORINHA	ABILIO MARTINS PINTO JR-ME	ABRIDOR	10
2	LEIA	ABILIO MARTINS PINTO JR-ME	ABRIDOR	10
3	NULL	ABILIO MARTINS PINTO JR-ME	ABRIDOR	20
4	LEIA	ALAMBRINDES GRAFICA EDITORA LTDA.	ABRIDOR	2200
5	NULL	ALAMBRINDES GRAFICA EDITORA LTDA.	ABRIDOR	2200

Esse tipo de resultado não existia com a opção **WITH ROLLUP**. Neste caso, a coluna **VENDEDOR** é **NULL** e o total corresponde ao total de produtos do tipo **ABRIDOR** comprado pelo cliente **ABILIO** (20 produtos). Já **ALAMBRINDES GRAFICA EDITORA LTDA** comprou 2200 produtos do tipo **ABRIDOR**. **WITH CUBE** inclui todas as outras subtotalizações possíveis no resultado.

Neste outro trecho do mesmo resultado retornado, as colunas **VENDEDOR** e **CLIENTE** são nulas e o total corresponde ao total vendido de produtos do tipo **CANETA**, ou seja, **526543**:

1211	NULL	ZINHO'S COM DE BRINDES LTDA ME	CANETA	8450
1212	NULL	NULL	CANETA	526543

Seguindo a mesma regra, se apenas o **CLIENTE** não é nulo, o total corresponde ao total comprado por este cliente, independentemente de vendedor ou de produto:

3531	LEIA	VITOR BRIGIO	NULL	1705
3532	NULL	VITOR BRIGIO	NULL	1705

Pontos principais

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

- As funções de agregação fornecidas pelo SQL Server permitem sumarizar dados. Por meio delas, é possível somar valores, calcular média e contar resultados. Os cálculos feitos pelas funções de agregação são feitos com base em um conjunto ou grupo de valores, porém retornam um único valor. Para obter os valores sobre os quais você poderá realizar os cálculos, geralmente são utilizadas as funções de agregação com a cláusula **GROUP BY**;

- Utilizando a cláusula **GROUP BY**, é possível agrupar diversos registros com base em uma ou mais colunas da tabela.