



# Habilitação Profissional Técnica de Nível Médio de **TÉCNICO EM INFORMÁTICA PARA INTERNET INTEGRADO** AO ENSINO MÉDIO

---

MODELAGEM E DESENVOLVIMENTO DE BANCO DE DADOS

2017

## Inserindo registros na tabela de Categorias

A tabela de categorias é bastante simples, como você deve se lembrar. Apenas um campo código, chave primária, e um campo de descrição. As categorias servirão para classificar os produtos, separando-os por sua natureza comercial. Em nossa aplicação fictícia, vamos vender diversos itens: livros, celulares, tablets, notebooks e material de escritório. Cada um destes itens, se tornará uma categoria.

Este tipo de classificação é bastante comum em aplicações comerciais pois permite realizar relatórios agrupados, filtrar informações e dar mais clareza as visualizações analíticas de negócio.

Vamos ao insert desta tabela, observe o script abaixo:

```
INSERT INTO tbcategorias  
    (descricao)  
VALUES ('Livros'),  
    ('Celulares'),  
    ('Tablets'),  
    ('Notebooks'),
```

('Material de Escritório');

Observe duas questões importantes:

- Não estou atribuindo nenhum valor para o campo Código, pois este campo é do tipo AUTO\_INCREMENT, ou seja, este campo vai se preencher sozinho.
- Estou inserido diversos valores simultaneamente no mesmo comando.

Alternativamente, é possível inserir cada registro linha a linha. Observe:

```
INSERT INTO tbcategorias  
  (descricao)  
VALUES  ('Livros');
```

```
INSERT INTO tbcategorias  
  (descricao)  
VALUES  ('Celulares');
```

```
INSERT INTO tbcategorias  
  (descricao)  
VALUES  ('Tablets');
```

```
INSERT INTO tbcategorias  
  (descricao)  
VALUES  ('Notebooks');
```

```
INSERT INTO tbcategorias  
    (descricao)  
VALUES ('Material de Escritório');
```

## Inserindo registros na tabela de Produtos

A tabela categoria foi carregada anteriormente de maneira propositada. A tabela produtos se relaciona com categorias através de uma chave estrangeira, portando os inserts em produtos só serão válidos se referenciarem categorias presentes na tabela tbCategorias, ou se o valor atribuído a elas for nulo.

Para a tabela de produtos, o *script* abaixo insere exatamente 10 registros:

```
INSERT INTO tbprodutos  
    (descricao,  
    valorvenda,  
    ativo,  
    categoria)  
VALUES ('Integração de Dados na Prática - Técnicas de ETL para Business Intelligence com Microsoft Integration Services 2012',  
55.00,  
1,  
1),  
('T-SQL com Microsoft SQL Server 2012 Express na Prática',  
55.00,  
1,  
1),
```

('Samsung Galaxy S III',  
1999.00,  
1,  
2),  
('Apple IPHONE 5',  
2199.00,  
1,  
2),  
('Samsung Galaxy Tab II',  
1999.00,  
1,  
3),  
('Motorola Xoom',  
1099.00,  
1,  
3),  
('Dell Ultrabook 14 ''',  
2499.00,  
1,  
4),  
('ASUS Ultrabook 14 ''',  
2599.00,  
1,  
4),  
('Fragmentadora Clone',  
1099.00,  
0,  
5),

```
('Suporte para Notebook',  
1099.00,  
1,  
5)
```

O negócio representado pela aplicação fictícia proposta por este livro, estão a venda livros, celulares, tablets e alguns acessórios de informática. Observe que novamente, não está sendo informado valor para o campo código que é preenchido automaticamente pelo banco de dados.

Como você deve estar lembrado, criamos uma tabela de pedidos e itens de pedidos. Desta forma, a tabela de pedidos armazena a referência do pedido e sua relação com o cliente e a data da venda. Porém os produtos adquiridos são armazenados na tabela de itens de pedidos.

Para que os próximos exemplos e exercícios fiquem alinhados com os resultados do livro, execute todos os scripts exatamente como estão listados aqui.

### **Inserindo registros na tabela de Pedidos e Itens de Pedidos**

Observe o script abaixo, que carrega os pedidos. Como você deve estar lembrado, temos uma tabela de Pedidos que armazena apenas o cabeçalho do pedido, registrando que é o cliente e a data da venda, e se relaciona com a tabela de itens de pedido com os respectivos produtos e quantidades adquiridas.

Digite e execute o script abaixo:

```
INSERT INTO tbpedidos  
(cliente,datavenda)
```

```
VALUES (6,'20121221'),
(2,'20121221'),
(6,'20121121'),
(4,'20121121'),
(4,'20121021'),
(3,'20121021'),
(2,'20120921'),
(3,'20120921'),
(2,'20120821'),
(4,'20120821'),
(3,'20120721'),
(6,'20120721'),
(4,'20120621'),
(2,'20120621'),
(4,'20120521'),
(4,'20120521');
```

O insert em itens de pedido deve referenciar os pedidos que acabamos de inserir. Observe o script abaixo e execute-o no MySQL Workbench.

```
INSERT INTO tbitenspedidos
(pedido,produto,valorvenda,quantidade)
VALUES (1,1,50,1),
(2,2,50,2),
(3,3,1500,1),
(4,4,2000,2),
(5,5,1800,1),
(6,6,1000,2),
```

(7,7,2100,3),  
(8,8,2000,1),  
(9,9,1000,2),  
(10,10,1000,4),  
(11,1,55,10),  
(12,2,55,10),  
(13,3,1500,2),  
(14,4,2000,3),  
(15,5,1500,2),  
(16,6,1000,1),  
(16,1,50,1),  
(15,2,50,2),  
(14,3,1500,1),  
(13,4,2000,2),  
(12,5,1800,1),  
(11,6,1000,2),  
(10,7,2100,3),  
(9,8,2000,1),  
(8,9,1000,2),  
(7,10,1000,4),  
(6,1,55,10),  
(5,2,55,10),  
(4,3,1500,2),  
(3,4,2000,3),  
(2,5,1500,2),  
(1,6,1000,1)



## JOINS em Bancos de dados

O JOIN é uma cláusula da linguagem SQL que permite criar consultas combinando resultados de uma ou duas tabelas através de valores comuns entre uma ou várias colunas de cada tabela. Quando falo de uma tabela, me refiro a casos específicos de chamados “SELF JOINS”, ou JOINS de uma tabela com ela mesmo.

Para entender melhor os JOINS e sua aplicação, tente visualizar as tabelas como conjuntos e os registros como elementos de cada um destes conjuntos. Através dos JOINS podemos consultar os elementos de cada um destes conjuntos e as respectivas intersecções, uniões e diferenças.

### O comando INNER JOIN

O padrão ANSI SQL, define quatro tipos de *JOIN*: *INNER*, *OUTTER*, *LEFT* e *RIGHT*. O *INNER JOIN* consulta os registros de duas tabelas, verificando todos os registros de cada uma e selecionando os que tem valores em comum baseado no critério estabelecido no JOIN.

Para listar o nome de cada cliente e a data de cada venda, podemos usar o comando INNER JOIN:

```
SELECT nome,datavenda
FROM tbclientes C
     INNER JOIN tbpedidos V
           ON ( C.codigo = V.cliente )
```

SQL File 2\*

SQL File 4\*

1

2

3

4

5

6

7

SELECT

NOME

,DataVenda

FROM

tbClientes C

INNER JOIN tbPedidos V ON (C.Codigo=V.Cliente)

Filter:

Export:

Autosize:

	NOME	DataVenda
▶	Mario Alberto	2012-12-21 00:00:00
	Mario Alberto	2012-09-21 00:00:00
	Mario Alberto	2012-08-21 00:00:00
	Mario Alberto	2012-06-21 00:00:00
	Paulo Cunha	2012-10-21 00:00:00
	Paulo Cunha	2012-09-21 00:00:00
	Paulo Cunha	2012-07-21 00:00:00
	Maria das Dores	2012-11-21 00:00:00
	Maria das Dores	2012-10-21 00:00:00
	Maria das Dores	2012-08-21 00:00:00
	Maria das Dores	2012-06-21 00:00:00
	Maria das Dores	2012-05-21 00:00:00
	Maria das Dores	2012-05-21 00:00:00
	Paula Dutra	2012-12-21 00:00:00
	Paula Dutra	2012-11-21 00:00:00
	Paula Dutra	2012-07-21 00:00:00

Esta query deve retornar

Esta query deve retornar exatamente, 16 registros.

*Query com INNER JOIN*

O *INNER JOIN* é considerado o tipo de *JOIN* “padrão”. Podemos executar a *query* acima, com outra sintaxe eliminando o *INNER* e obteremos o mesmo resultado.

```
SELECT nome,datavenda
FROM tbclientes C

JOIN tbpedidos V
ON ( C.codigo = V.cliente )
```

Através do *JOIN*, estamos trazendo a intersecção dos registros da tabela *Clientes* (ou conjunto A) e da tabela *Pedidos* (conjunto B).

### **comando LEFT JOIN**

O comando *LEFT JOIN* entre duas tabelas hipotéticas A e B, vai trazer todos os registros da tabela A independente do critério estabelecido no predicado do *JOIN*. Ou seja, se a tabela A contém 100 registros e nenhum deles tem um correspondente na outra baseado no critério de comparação, a *query* ainda sim vai trazer 100 registros, porém onde a correspondência existir, os dados correspondentes serão resgatados.

```
SELECT
    nome,
    datavenda
FROM tbclientes C
LEFT JOIN tbpedidos V
ON ( C.codigo = V.cliente )
```

A *query* acima vai produzir exatamente 17 registros, um a mais que o resultado da *query* anterior. Observe que para o cliente com o código 1, de nome “*João da Silva*”, a coluna *DataVenda* está “nula”.

The screenshot shows a SQL IDE with two tabs: 'SQL File 2\*' and 'SQL File 4\*'. The active tab contains the following SQL query:

```
1 SELECT
2     NOME
3     ,DataVenda
4 FROM
5     tbClientes C
6 LEFT JOIN tbPedidos V ON (C.Codigo=V.Cliente)
7
```

Below the query editor, there is a 'Filter:' field, an 'Export:' button, and an 'Autosize:' button. The results pane displays a table with two columns: 'NOME' and 'DataVenda'. The first row for 'João da Silva' has a 'NULL' value in the 'DataVenda' column, indicated by a small red icon. The subsequent rows show various names and dates.

NOME	DataVenda
João da Silva	NULL
Maria das Dores	2012-05-21 00:00:00
Maria das Dores	2012-05-21 00:00:00
Maria das Dores	2012-06-21 00:00:00
Mario Alberto	2012-06-21 00:00:00
Paula Dutra	2012-07-21 00:00:00
Paulo Cunha	2012-07-21 00:00:00
Mario Alberto	2012-08-21 00:00:00
Maria das Dores	2012-08-21 00:00:00

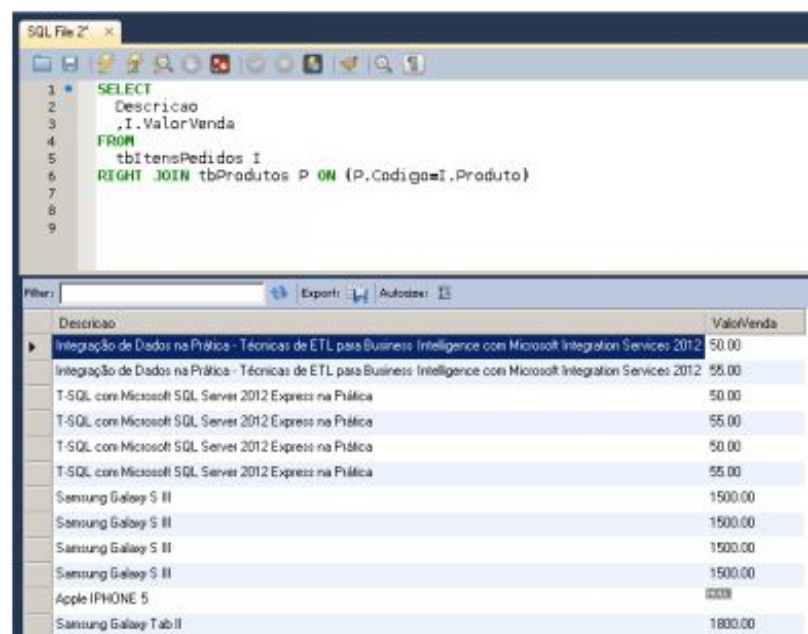
### *Query com LEFT JOIN*

Basicamente, através do LEFT JOIN estamos trazendo todos os registros da tabela Clientes, mais a intersecção com a tabela Vendas.

## O comando RIGHT JOIN

O comando *RIGHT JOIN*, produz um resultado semelhante ao *LEFT JOIN* porém com a inversão da comparação. Para que você possa entender melhor este conceito, em nosso banco de dados criamos uma tabela chamada *Produtos* que se relaciona com a tabela vendas. Podemos listar a descrição de todos os produtos e a data da venda de cada um deles junto com os produtos que não realizaram vendas através de uma query com o comando *RIGHT JOIN*:

```
SELECT descricao,  
       I.valorvenda  
FROM   tbItensPedidos I  
       RIGHT JOIN tbProdutos P  
       ON ( P.codigo = I.produto )
```



The screenshot shows a SQL query window with the following text:

```
1 SELECT  
2     descricao  
3     ,I.valorvenda  
4 FROM  
5     tbItensPedidos I  
6     RIGHT JOIN tbProdutos P ON (P.codigo=I.produto)  
7  
8  
9
```

Below the query, the results are displayed in a table with two columns: 'Descricao' and 'ValorVenda'.

Descricao	ValorVenda
Integração de Dados na Prática - Técnicas de ETL para Business Intelligence com Microsoft Integration Services 2012	50.00
Integração de Dados na Prática - Técnicas de ETL para Business Intelligence com Microsoft Integration Services 2012	55.00
T-SQL com Microsoft SQL Server 2012 Express na Prática	50.00
T-SQL com Microsoft SQL Server 2012 Express na Prática	55.00
T-SQL com Microsoft SQL Server 2012 Express na Prática	50.00
T-SQL com Microsoft SQL Server 2012 Express na Prática	55.00
Samsung Galaxy S III	1500.00
Samsung Galaxy S III	1500.00
Samsung Galaxy S III	1500.00
Samsung Galaxy S III	1500.00
Apple IPHONE 5	2000.00
Samsung Galaxy Tab II	1800.00

*Query com Right JOIN*

```
SELECT descricao,  
       I.valorvenda  
FROM   tbProdutos P  
       LEFT JOIN tbItensPedidos I  
       ON ( P.codigo = I.produto )
```

Observe que podemos chegar exatamente ao mesmo resultado, usando *LEFT JOIN* e invertendo a posição das tabelas:



## O comando CASE

Diferente das linguagens de programação mais comuns, no SQL o CASE não tem o comportamento de controle de fluxo. O comando CASE, avalia uma lista de condições verificadas em um ou mais campos e retorna apenas um de vários resultados possíveis.

A sintaxe do CASE não é complexa. Depois do comando CASE, as condições são avaliadas uma a uma com os argumentos WHEN e THEN.

```
SELECT
  CASE
    WHEN <Expressão lógica 1>
      THEN <Resultado da expressão 1>
    WHEN <Expressão lógica 2>
      THEN <Resultado da expressão 2>
    ELSE
      <Resultado fora das condições listadas>
    END
  FROM
    <Tabela>
```

```
SELECT
  Codigo
  ,Descricao
  ,CASE
    WHEN ValorVenda between 0 AND 100      THEN 'Bronze'
    WHEN ValorVenda between 100 AND 200 THEN 'Prata'
    WHEN ValorVenda between 200 AND 300 THEN 'Ouro'
    WHEN ValorVenda > 300 THEN 'Platina'
    WHEN ValorVenda IS NULL THEN 'Platina'
    ELSE
      'Não classificado'
    END AS TipoProduto
  FROM
    tbProdutos
```

Através do exemplo acima, criamos uma classificação baseada no valor de venda de cada produto. Observe que na avaliação lógica, entre o comando *WHEN* e *THEN*, podemos ter qualquer condição e neste exemplo foi utilizado o comparador *BETWEEN*. Execute o *script* acima e observe os resultados. Experimente também mudar as condições e os resultados de cada comparação.

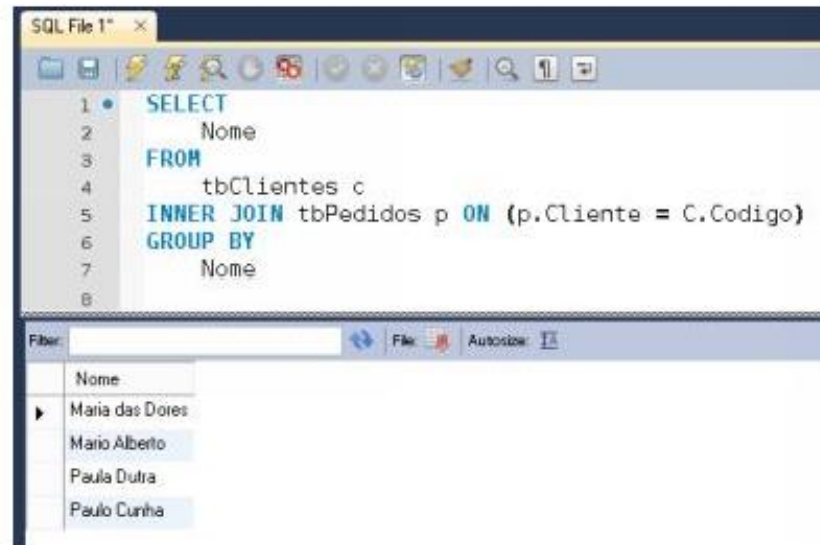
## Agrupando dados com GROUP BY

A cláusula *GROUP BY* permite agrupar registros baseados em um critério estabelecido no argumento da instrução posicionado logo após o comando.

Com o comando GROUP BY, podemos agrupar os registros e todos os clientes que realizaram mais de uma compra no mesmo dia, aparecerão agrupados. Execute o Script abaixo:

```
SELECT nome  
FROM tbclientes c  
    INNER JOIN tbpedidos p  
        ON ( p.cliente = C.codigo )  
GROUP BY nome
```

O seu resultado deve ser igual a este:



*Query com GROUP BY*

## Funções de agregação

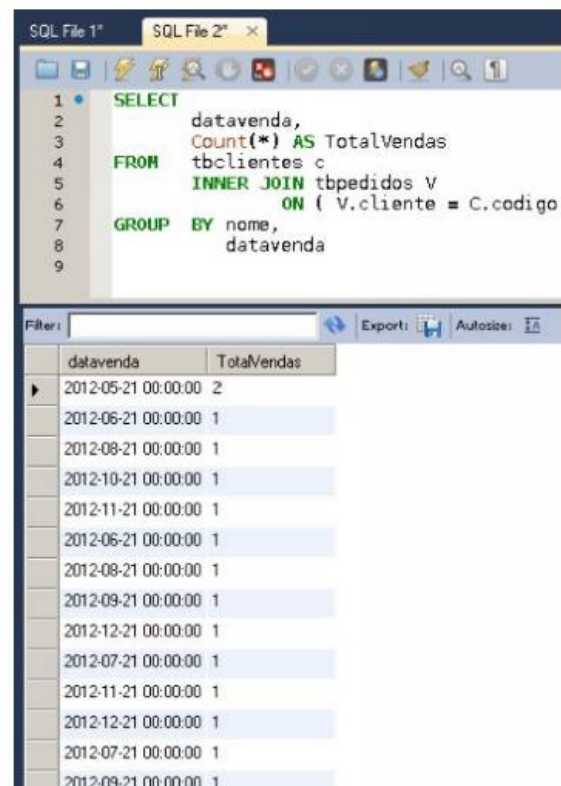
Funções de agregação são funções que realizam cálculos com um conjunto de valores determinados pela condição estabelecida em uma cláusula *GROUP BY*, retornando um valor único para aquele conjunto. Para que você possa entender melhor a utilização das funções de agregação, acompanhe os próximos exemplos.

### Usando o COUNT

A função *COUNT* pode ser usada para contar o número de registros estabelecidos em um condição *GROUP BY*. Por exemplo, para contar o número de compras realizadas por dia, podemos usar o seguinte *Script*:

```
SELECT
    datavenda,
    Count(*) AS TotalVendas
FROM tbclientes c
    INNER JOIN tbpedidos V
        ON ( V.cliente = C.codigo )
GROUP BY nome,
    datavenda
```

Se você executou o exemplo acima você obteve o seguinte resultado:



The screenshot shows a SQL IDE with two tabs: 'SQL File 1\*' and 'SQL File 2\*'. The active tab displays a SQL query. Below the query editor, there is a 'Filter:' field and a table of results. The table has two columns: 'datavenda' and 'TotalVendas'. The results show a list of dates and their corresponding total sales counts.

datavenda	TotalVendas
2012-05-21 00:00:00	2
2012-06-21 00:00:00	1
2012-08-21 00:00:00	1
2012-10-21 00:00:00	1
2012-11-21 00:00:00	1
2012-06-21 00:00:00	1
2012-08-21 00:00:00	1
2012-09-21 00:00:00	1
2012-12-21 00:00:00	1
2012-07-21 00:00:00	1
2012-11-21 00:00:00	1
2012-12-21 00:00:00	1
2012-07-21 00:00:00	1
2012-09-21 00:00:00	1

Com esta *query*, podemos obter informações interessantes para acompanhamento de resultados de metas organizacionais, resumizando indicadores de desempenho.

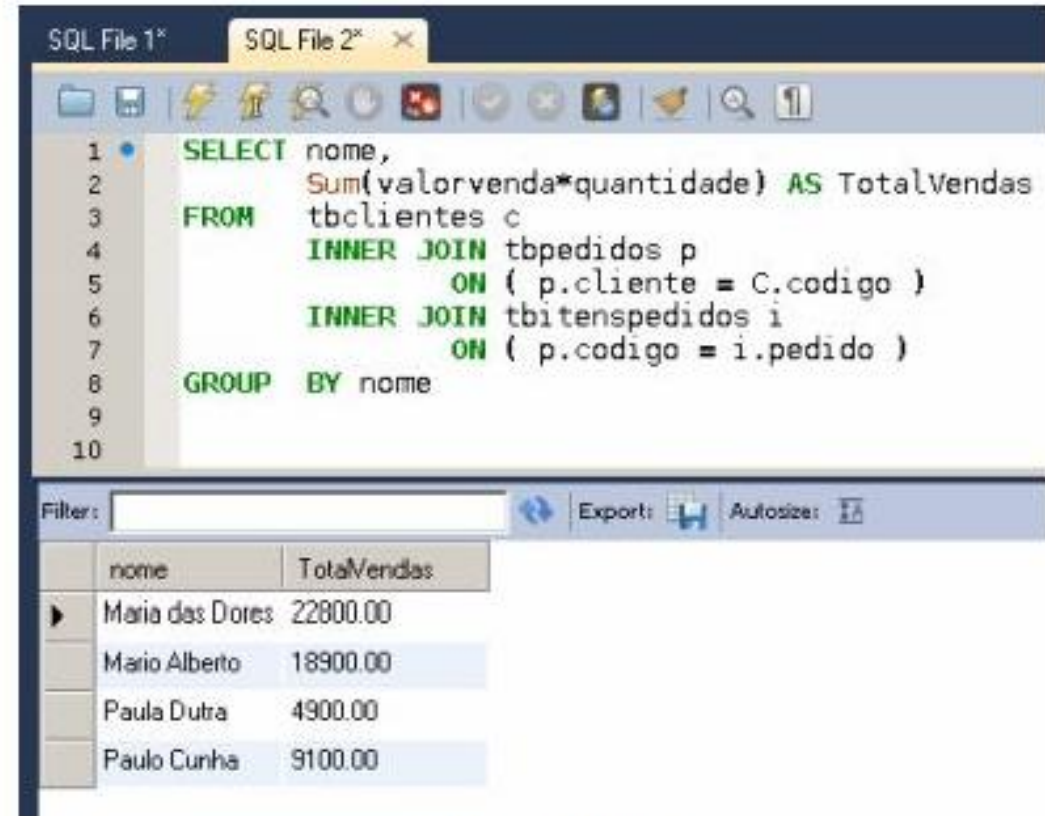


## Somando valores com SUM

A função *SUM*, soma valores numéricos dentro de um conjunto de valores estabelecidos pelo GROUP BY. A tabela Vendas, tem um campo determinado “*ValorVenda*”, que armazena o valor pelo qual o produto foi vendido. Para totalizar o total vendido para cada cliente, podemos executar a seguinte *query*:

```
SELECT nome,  
       Sum(valorvenda*quantidade) AS TotalVendas  
FROM   tbclientes c  
       INNER JOIN tbpedidos p  
           ON (p.cliente = C.codigo)  
       INNER JOIN tbitenspedidos i  
           ON (p.codigo = i.pedido)  
GROUP BY nome
```

Se você executou o *Script* acima, chegou ao seguinte resultado:



The screenshot shows a SQL IDE with two tabs: 'SQL File 1\*' and 'SQL File 2\*'. The active tab 'SQL File 2\*' contains the following SQL query:

```
1 SELECT nome,  
2       Sum(valorvenda*quantidade) AS TotalVendas  
3 FROM   tbclientes c  
4       INNER JOIN tbpedidos p  
5           ON ( p.cliente = C.codigo )  
6       INNER JOIN tbitenspedidos i  
7           ON ( p.codigo = i.pedido )  
8 GROUP BY nome  
9  
10
```

Below the query editor, there is a 'Filter:' field and buttons for 'Export:' and 'Autosize:'. The results of the query are displayed in a table with two columns: 'nome' and 'TotalVendas'.

nome	TotalVendas
Maria das Dores	22800.00
Mario Alberto	18900.00
Paula Dutra	4900.00
Paulo Cunha	9100.00

*Query com SUM*

## Usando o AVG

O comando AVG permite calcular a média de valores dentro de um conjunto estabelecido pelo comando *GROUP BY*. Por exemplo, podemos calcular a média de valor de venda por produto através da seguinte *query*:

```
SELECT descricao,  
       Sum(i.valorvenda * quantidade) AS TotalVendas,  
       Avg(i.valorvenda * quantidade) AS ValorMedio  
FROM   tbprodutos p  
       INNER JOIN tbitenspedidos i  
             ON ( p.codigo = i.produto )  
GROUP BY descricao
```

Observe que estou adicionando o total da venda, para que você possa visualizar melhor a utilização das funções de agregação. Quando a instrução GROUP BY é utilizada, é possível combinar diferentes funções de agregação, sem um limite de utilização.

## MIN e MAX

As funções de agregação MIN e MAX, retornam respectivamente os valores mínimos e máximos de um determinado campo ou combinação aritmética de campos, de acordo com o critério especificado na cláusula GROUP BY.

Para obter o valor máximo e mínimo para cada produto vendido no banco de dados projetomysql, o seguinte script retorna a informação desejada:

```
SELECT descricao,  
       Min(i.valorvenda * quantidade) AS ValorMinimo,  
       Max(i.valorvenda * quantidade) AS ValorMaximo  
FROM   tbprodutos p  
       INNER JOIN tbitenspedidos i  
             ON ( p.codigo = i.produto )  
GROUP BY descricao
```

## A cláusula HAVING

A cláusula *having* permite filtrar resultados dos conjuntos agregados pela condição *GROUP BY*. O *script* abaixo, retorna todos os clientes com vendas de valor total maior que 300:

```
SELECT
    descricao,
    Sum(quantidade * i.valorvenda) AS TotalVendas
FROM  tbitenspedidos i
      INNER JOIN tbprodutos P
            ON (i.produto = P.codigo )
GROUP BY descricao
HAVING Sum(quantidade * i.valorvenda) > 5000
```