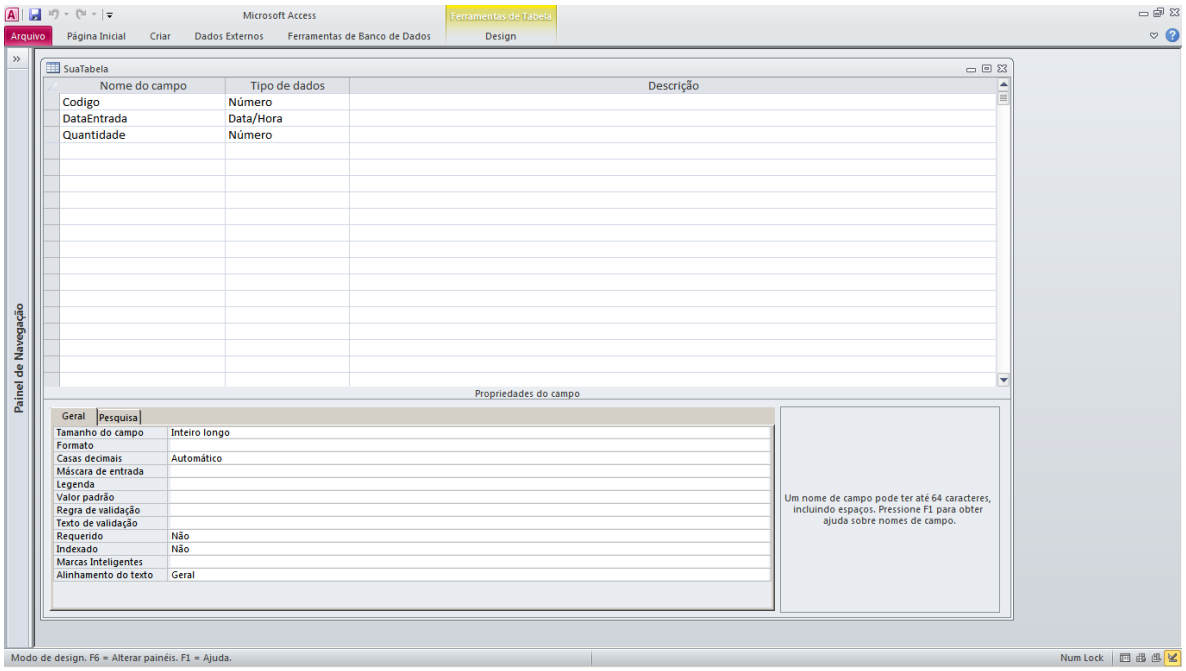
SQL – UMA ABORDAGEM INTERESSANTE

Por Eduardo V. Machado (Good Guy)

SQL é uma linguagem de consulta estruturada, do inglês *Structured Query Language* . É uma linguagem de pesquisa declarativa para banco de dados relacional (base de dados relacional). Muitas das características originais do SQL foram inspiradas na álgebra relacional. Sem entrar em detalhes quanto a sua origem, os comandos SQL tornaram mais fácil as rotinas de consulta no banco de dados Access.

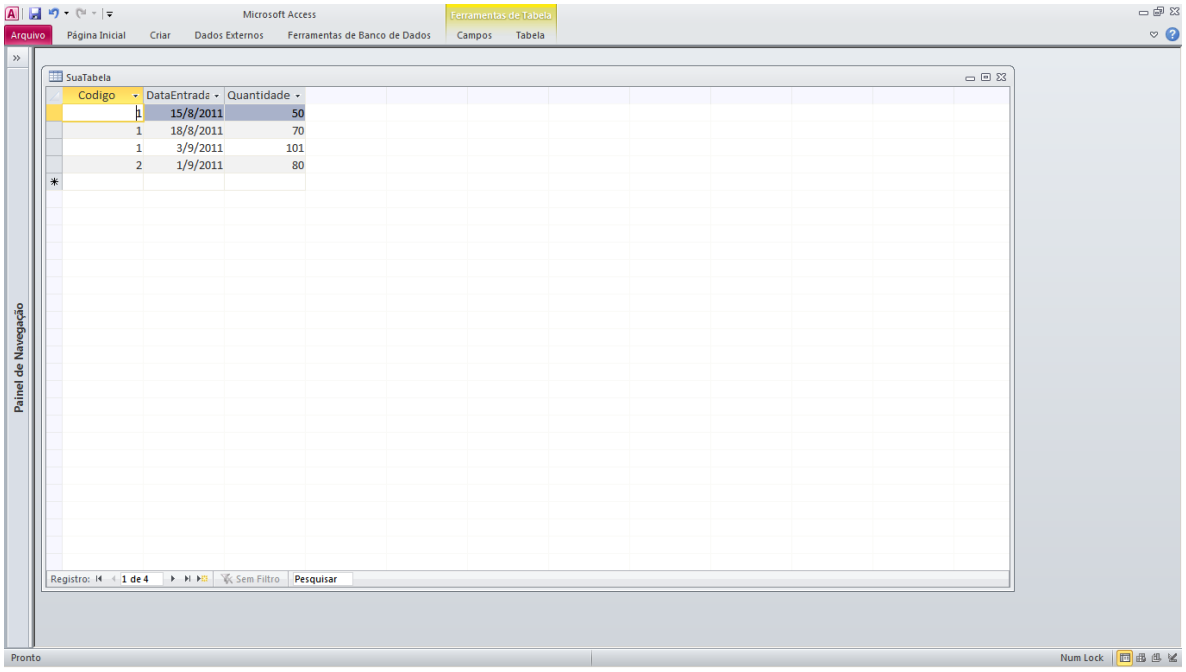
É possível visualizar uma sintaxe SQL elaborada automaticamente pelo próprio Access quando criamos uma consulta à tabela no ambiente Access. Consideremos, por exemplo, a figura abaixo:

Temos uma tabela chamada “SuaTabela” com os seguintes campos:

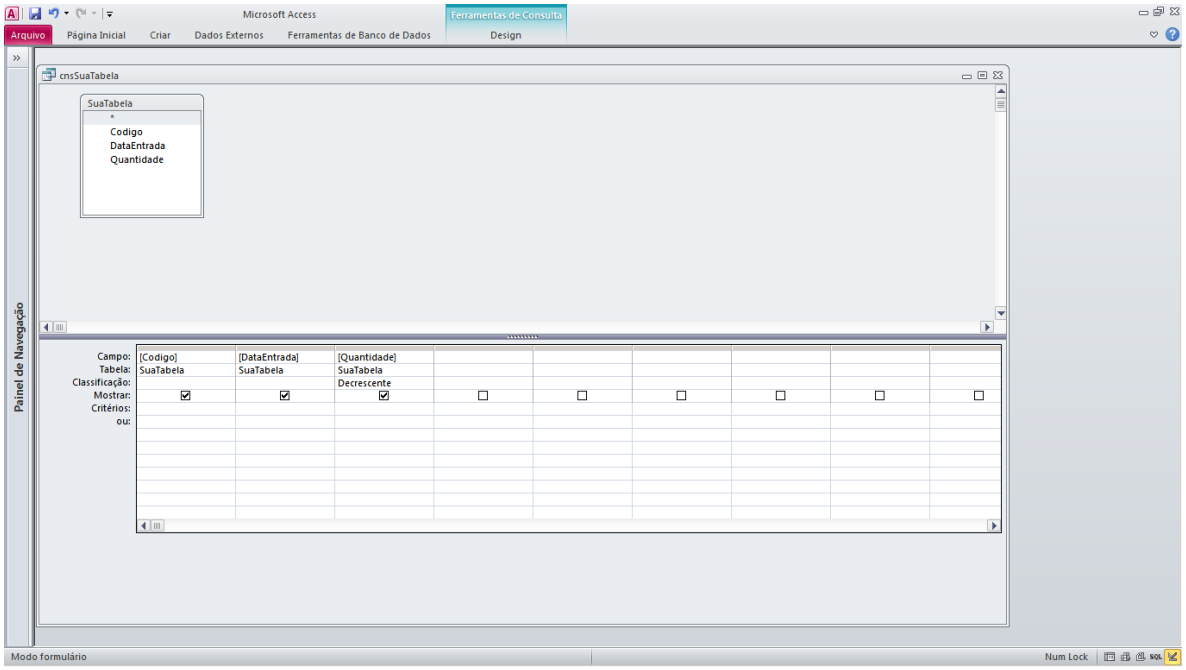


SQL – UMA ABORDAGEM INTERESSANTE

Por Eduardo V. Machado (Good Guy)



Criamos então uma consulta que tenha o registro de maior quantidade em ordem decrescente:



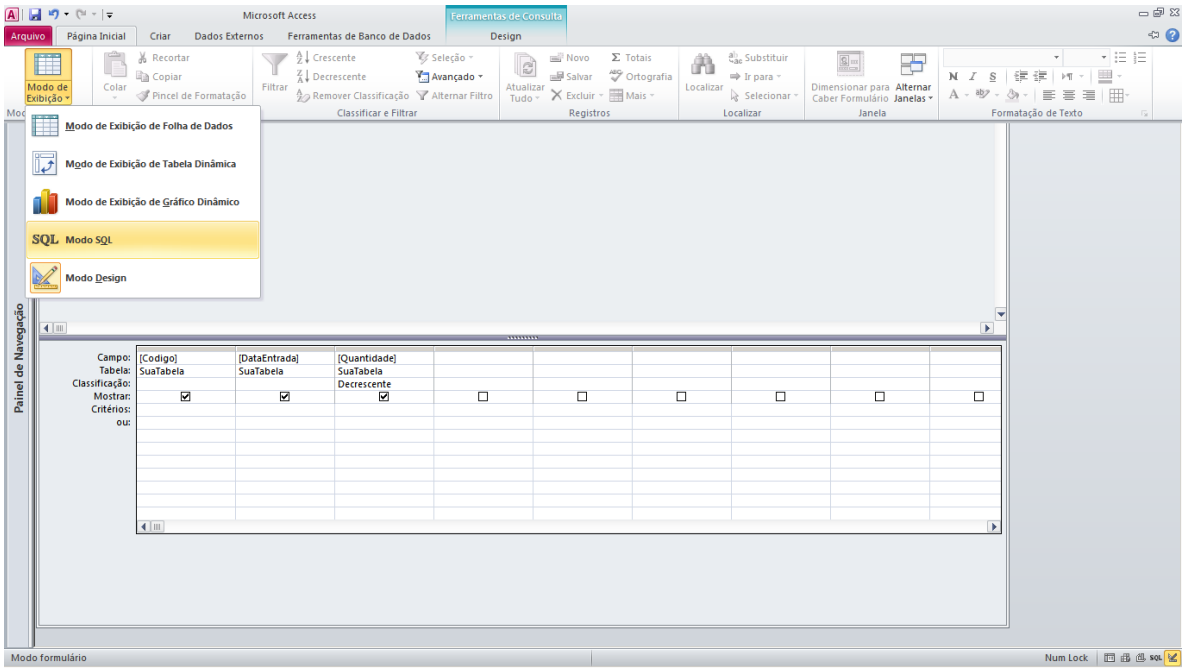
Este é o modo Design da sua consulta onde você pode trabalhar quaisquer tabelas que queira inserir no ambiente superior com o simples clicar do botão direito do mouse. Abaixo, temos as colunas referentes aos campos da tabela com seus campos modificadores: Classificação(Crescente ou Decrescente) e Critérios.

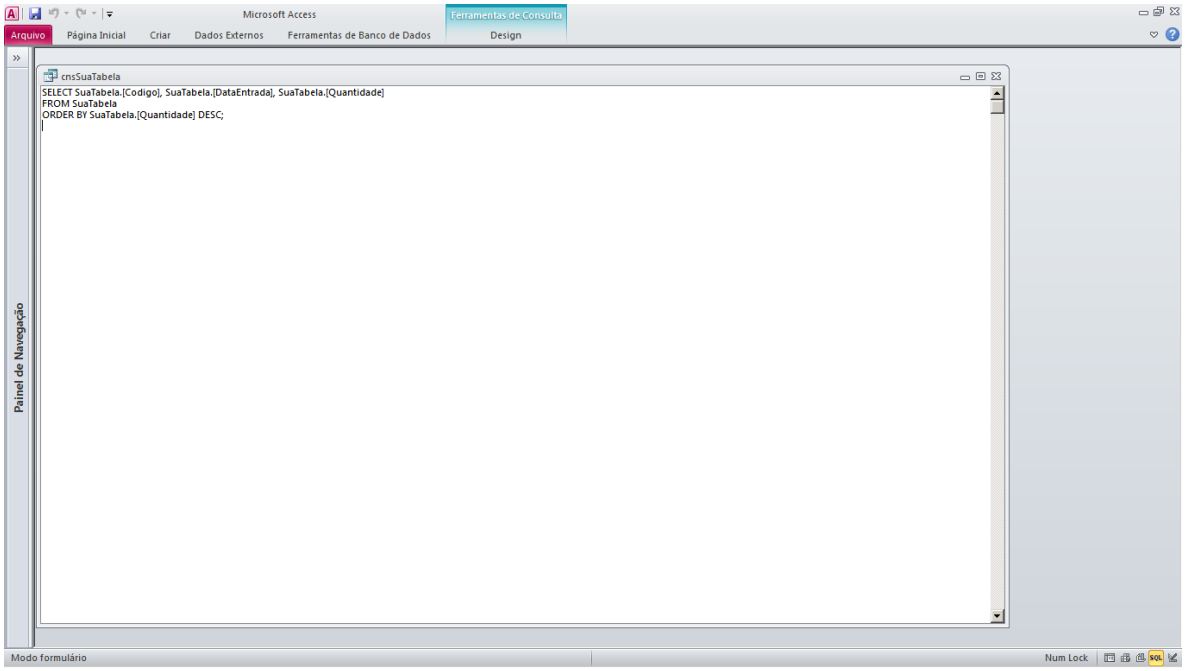
SQL – UMA ABORDAGEM INTERESSANTE

Por Eduardo V. Machado (Good Guy)

Em Critérios, podemos criar a nossa lógica de pesquisa com operadores lógicos que determinarão que tipo de consulta desejamos. Por exemplo, valores maiores que 500, nomes que comecem com a letra “A”, etc.

Vamos, agora, visualizar esta mesma consulta no modo SQL:





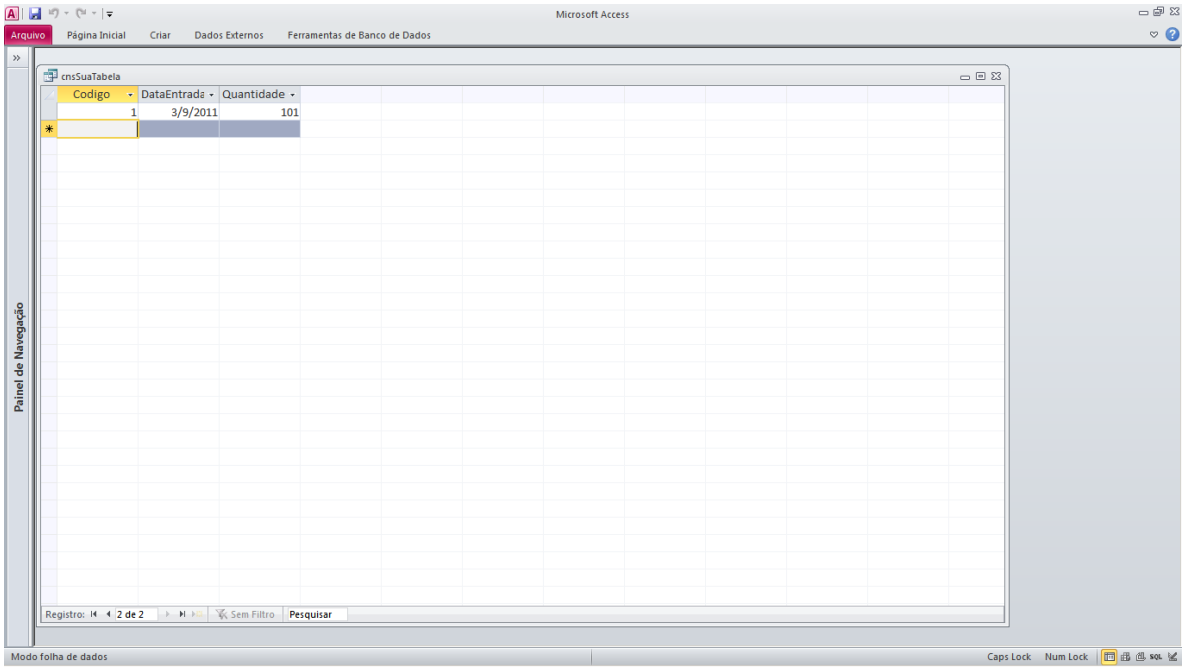
A sintaxe criada foi:

```
SELECT SuaTabela.[Codigo], SuaTabela.[DataEntrada],
SuaTabela.[Quantidade]
FROM SuaTabela
ORDER BY SuaTabela.[Quantidade] DESC;
```

Altere esta sintaxe para:

```
SELECT TOP 1 SuaTabela.[Codigo], SuaTabela.[DataEntrada],
SuaTabela.[Quantidade]
FROM SuaTabela
ORDER BY SuaTabela.[Quantidade] DESC;
```

Cujo resultado gerado será:



Observe que só retornou um só registro. Este resultado foi determinado pela cláusula TOP 1. Se quiséssemos mais de um, era só aumentar o valor depois de TOP. Portanto, nossa consulta nos trouxe como resultado o registro que possui o maior número para quantidade em ordem decrescente.

COMANDO SELECT (SELECIONAR)

A interrogação de qualquer base de dados relacional faz-se utilizando o comando SELECT. A sintaxe do comado é a seguinte:

SELECT $Campo_1$, $Campo_2$, ... , $Campo_N$, *

FROM Tabela$_1$, ... , Tabela$_K$

[WHERE condição]

[GROUP BY …]

[HAVING …]

[ORDER BY …]

As cláusulas entre [ ] são opcionais; no entanto, sempre que aparecerem terá de ser pela ordem indicada;

Cláusula FROM (DE, TEM ORIGEM EM)

Especifica a origem da consulta, de onde ela vem. Significa 'DE' (tem origem em). Geralmente tem origem em tabelas.

Se a cláusula WHERE (ONDE) for omitida, a **condição** será considerada **verdadeira**;

Se indicarmos * no SELECT em vez de campos, serão seleccionados **todos** os campos das tabelas envolvidas;

O resultado de uma consulta de SQL com o comando SELECT é **sempre** uma tabela;

A ordem dos campos de saída é a que consta no SELECT, e pode ser diferente daquela usada na criação da tabela;

Podemos fazer selecção de expressões ou constantes:

Ex.: nomes e idades daqui a cinco anos

```
SELECT nome, idade +5
FROM pessoas
```

Se pretendermos eliminar os duplicados usamos:

```
SELECT DISTINCT Campo1, .....;
```

SELECT ALL Campo1, ... é equivalente a SELECT Campo1, ...;

Cláusula WHERE (ONDE)

A cláusula WHERE admite os seguintes operadores:

Relacionais	
=	igual a
>	maior que
>=	maior ou igual a
<	menor que
<=	menor ou igual a
<> (mais vulgar) ou !=	diferente

Lógicos	
$cond_1$ AND $cond_2$	conjunção
$cond_1$ OR $cond_2$	disjunção
NOT condição	negação

Especiais	
$campo_N$ [NOT]BETWEEN $valor_1$ AND $valor_2$	verifica intervalos de valores
$campo_N$ [NOT]IN (valor1, ..., $valor_N$)	verifica conjuntos de valores
$campo_N$ IS [NOT] NULL	trata valores NULL
$campo_N$ LIKE '.....'	compara strings
$tabela_1$ [NOT] CONTAINS $tabela_2$	compara tabelas
[NOT] EXISTS tabela	verifica tabelas vazias

Nota: um campo tem valor NULL quando não está preenchido;

Ex.1: **nomes dos clientes moradores em cidades onde existem vendedores**

```
SELECT DISTINCT nome
FROM clientes
 WHERE cidade IN (SELECT DISTINCT cidade  FROM vendedores)
```

Ex.2: **nomes dos clientes com o mesmo número e a mesma cidade de algum**
Vendedor

```
SELECT DISTINCT nome
FROM clientes
 WHERE <cidade, n_cliente> IN (SELECT DISTINCT cidade,
n_vend FROM vendedores)
```

Ex.3: **procura nomes de clientes cuja cidade vem especificada.**

```
SELECT nome
FROM clientes
WHERE cidade = “NITERÓI”
```

Precedência dos operadores:

Ordem decrescente ↓		
	( )	Parêntesis
	* , /	multiplicação, divisão
	+ , -	soma, subtracção
	NOT	negação lógica
	AND	conjunção lógica
	OR	disjunção lógica

Cláusula AS (COMO)

Esta cláusula renomeia um campo de saída;

```
Ex.: SELECT idade AS idade_agora, idade +5 AS idade_depois
     FROM pessoas
```

Cláusula GROUP BY(AGRUPAR POR)

Os dados selecionados na sentença "SELECT" que leva o "GROUP BY" devem ser:

- Uma constante.
- Uma função de grupo (SUM, COUNT, AVG…)
- Uma coluna expressa no GROUP BY.

A cláusula GROUP BY serve para calcular propriedades de um ou mais conjuntos de filas. Se se seleciona mais de um conjunto de filas, GROUP BY controla que as filas da tabela original sejam agrupadas em um temporário.

A cláusula HAVING se emprega para controlar qual dos conjuntos de filas se visualiza. Avalia-se sobre a tabela que devolve o GROUP BY. Não pode existir sem GROUP BY.

HAVING é parecido ao WHERE, porém trabalha com grupos de filas; pergunta por uma característica de grupo, ou seja, pergunta pelos resultados das funções de grupo, o qual WHERE não pode fazer.

Ex.: Temos a seguinte tabela “Pedidos”:

Codigo	DatadoPedido	ValordoPedido	Cliente
1	12/11/2008	1000,00	Eduardo
2	23/10/2008	1600,00	Lucas
3	02/09/2008	700,00	Eduardo
4	03/09/2008	300,00	Eduardo
5	30/08/2008	2000,00	Mateus
6	04/10/2008	100,00	Lucas

Agora, queremos encontrar a soma total(pedido total) de cada cliente.

Temos que usar a cláusula GROUP BY para agrupar os Clientes.

Usamos a seguinte cláusula SQL:

```
SELECT Cliente,SUM(ValordoPedido) FROM Pedidos
GROUP BY Cliente
```

O conjunto de resultados se parecerá com isto:

Cliente	SUM(ValordoPedido)
Eduardo	2000,00
Lucas	1700,00
Mateus	2000,00

Vamos ver o que acontece se omitirmos a cláusula GROUP BY.

```
SELECT Cliente,SUM(ValordoPedido) FROM Pedidos
```

O conjunto de resultados se parecerá com isto:

Cliente	SUM(ValordoPedido)
Eduardo	5700,00
Lucas	5700,00
Eduardo	5700,00
Eduardo	5700,00
Mateus	5700,00
Lucas	5700,00

Como vê, o conjunto de resultados acima não é o que queremos.

Explicação de por que não se deve usar a cláusula SELECT omitindo GROUP BY: A cláusula SELECT acima possui 2 colunas específicas (Cliente e SUM(ValordoPedido). O “SUM(ValordoPedido)” retorna um valor inteiro(que é a soma total da coluna “ValordoPedido”), enquanto “Cliente” retorna 6 valores (um valor para cada linha na tabela “Pedidos”). Isto, portanto, não nos dá o resultado correto. Contudo, você observou que a cláusula GROUP BY resolve este problema?

GROUP BY com Mais de Uma Coluna

Podemos também usar a cláusula GROUP BY em mais de uma coluna, dessa maneira: Separando com vírgula as colunas:

```
SELECT Cliente,DatadoPedido,SUM(ValordoPedido) FROM Pedidos
GROUP BY Cliente,DatadoPedido
```

Cláusula HAVING

A cláusula HAVING vem adicionada a SQL porque a palavra-chave (ou cláusula) WHERE não se pode usar com funções agregadas.

Sintaxe SQL HAVING

```
SELECT nome_da_coluna, função agregada (nome_da_coluna)
FROM nome_da_tabela
WHERE valor do operador da coluna_nome
```

```
GROUP BY coluna_nome
HAVING função agregada(coluna_nome) valor do operador
```

Exemplo de SQL HAVING

Temos a seguinte tabela de “Pedidos”:

Codigo	DatadoPedido	PrecodoPedido	Cliente
1	2008/11/12	1000,00	Eduardo
2	2008/10/23	1600,00	Lucas
3	2008/09/02	700,00	Eduardo
4	2008/09/03	300,00	Eduardo
5	2008/08/30	2000,00	Fernando
6	2008/10/04	100,00	Lucas

Agora queremos achar se algum dos clientes tem um pedido total cujo valor seja menor que 2000.

Emprega-se a seguinte sintaxe SQL:

```
SELECT Cliente,SUM(PrecodoPedido) FROM Pedidos
GROUP BY Cliente
HAVING SUM(PrecodoPedido)<2000
```

O conjunto de resultados se parecerá assim:

Cliente	SUM(PrecodoPedido)
Lucas	1700,00

Agora queremos saber se os clientes “Eduardo” ou “Fernando” tem um pedido total cujo valor seja maior que 1500.

Acrescentamos uma cláusula WHERE à sintaxe SQL:

```
SELECT Cliente,SUM(PrecodoPedido) FROM Pedidos
WHERE Cliente='Eduardo' OR Cliente='Fernando'
GROUP BY Cliente
HAVING SUM(PrecodoPedido)>1500
```

O conjunto de resultado se parecerá assim:

Cliente	SUM(PrecodoPedido)
Eduardo	2000,00
Fernando	2000,00

Cláusula ORDER BY (ORDENAR POR)

A sua sintaxe é:

ORDER BY $campo_1$[ASC|DESC], $campo_2$[ASC|DESC],...

A ordem assumida é ascendente (ASC); no entanto, podemos explicitar essa ordem escrevendo ASC, que não terá nenhum efeito prático;

Podemos indicar o campo de ordenação pela sua posição no comando SELECT:

Seleccionar na tabela Comissão o valor a receber, o montante do imposto (17%) e o valor líquido, para os indivíduos cujo Id é 14 ou 25, apresentando a ordenação por Id e Imposto:

```
SELECT Id, Valor AS líquido, Valor*0,17 AS Imposto,
Valor+Valor*0,17 AS Bruto

FROM comissão
WHERE Id IN (14,25)
ORDER BY Id, Valor*0,17
```

Este comando é pouco eficiente, na medida em que, se quisermos alterar o valor da expressão do imposto, teremos que o fazer em vários locais; a expressão mais eficiente seria:

```
SELECT Id, Valor AS líquido, Valor*0,17 AS Imposto,
Valor+Valor*0,17 AS Bruto

FROM comissão
WHERE Id IN (14,25)
ORDER BY Id, 3          (posição do campo no comando SELECT)
```