



Consultando dados

- SELECT;
- Ordenação de dados;
- Operadores relacionais;
- Operadores lógicos;
- Consulta de intervalos com BETWEEN;
- Consulta com base em caracteres;
- Consulta de valores pertencentes ou não a uma lista de elementos;
- Lidando com valores nulos;
- Substituição de valores nulos;
- UNION;
- EXCEPT e INTERSECT.



Esta Leitura Complementar refere-se ao conteúdo das Aulas 14 a 19.

1.1.Introdução

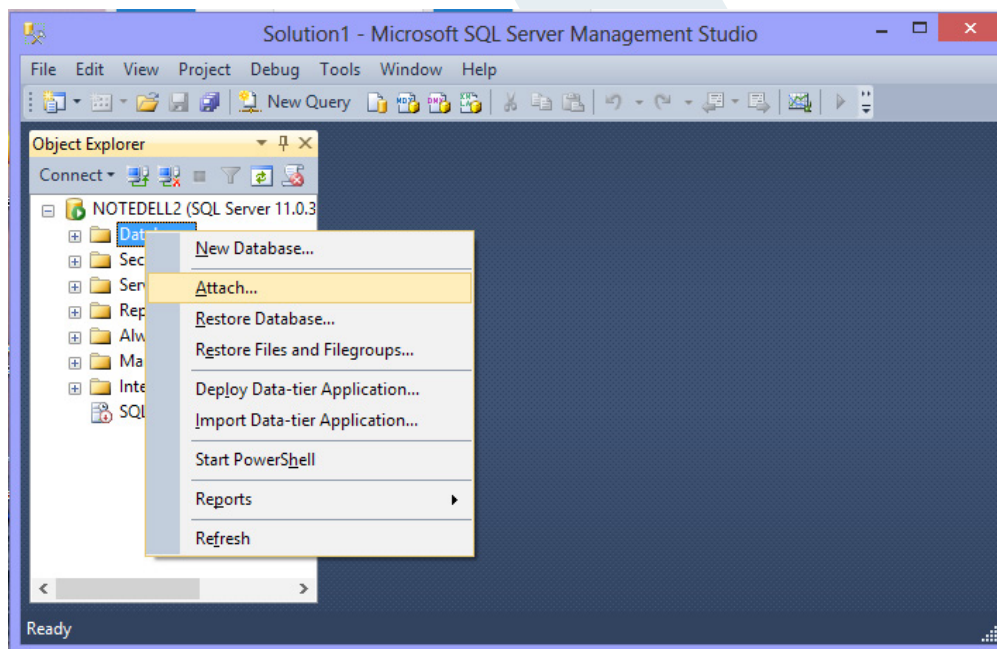
Na linguagem SQL, o principal comando utilizado para a realização de consultas é o **SELECT**. Por meio dele, torna-se possível consultar dados pertencentes a uma ou mais tabelas de um banco de dados.

No decorrer desta leitura, serão apresentadas as técnicas de utilização do comando **SELECT**, bem como algumas diretrizes para a realização de diferentes tipos de consultas SQL.

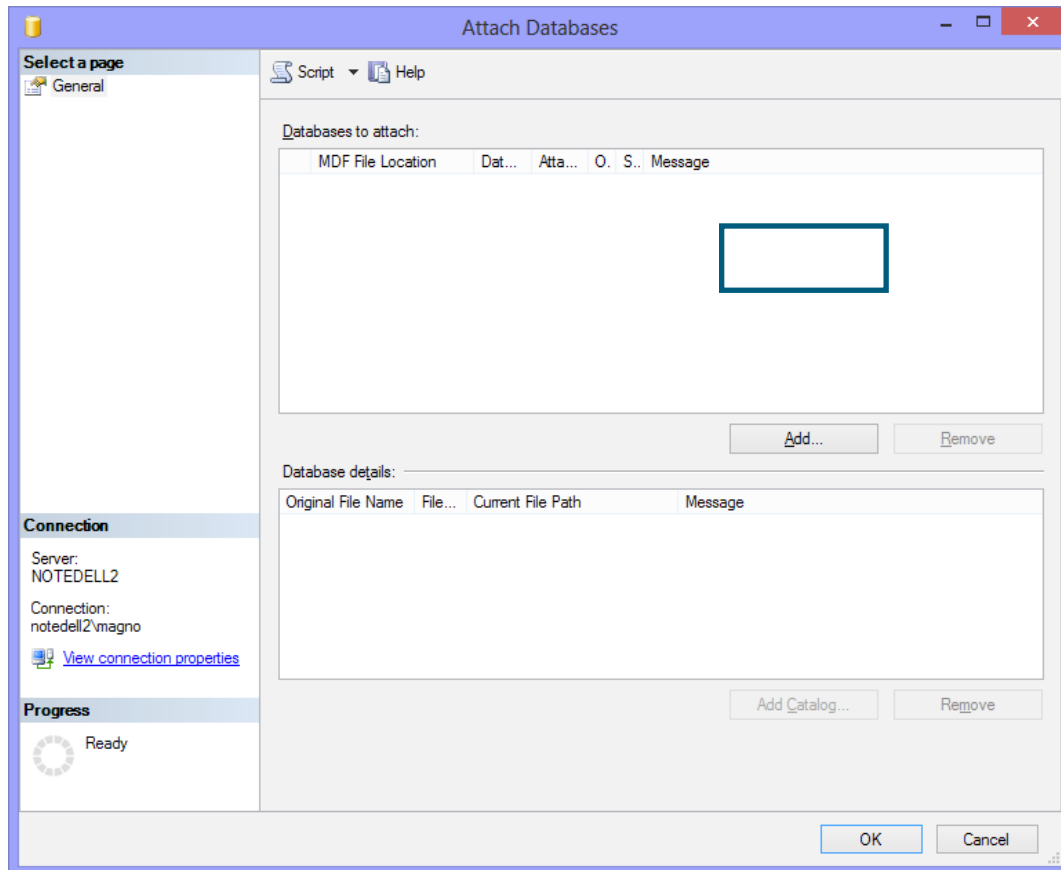
Para que possamos fazer exemplos que demonstrem as técnicas mais apuradas de consulta, precisamos ter um banco de dados com um volume razoável de informações já cadastradas.

Siga os passos adiante:

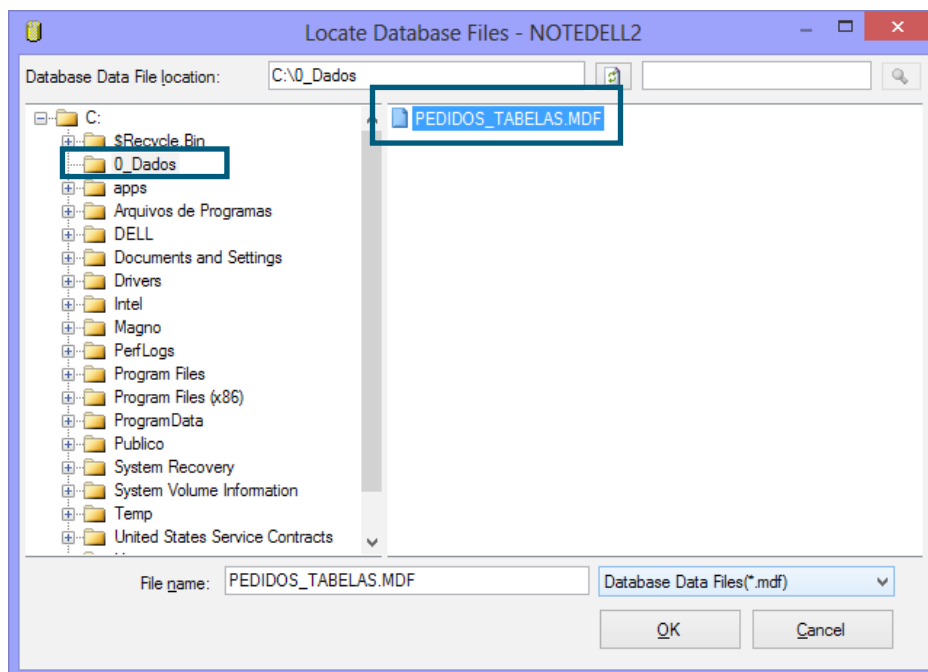
1. No Object Explorer, clique com o botão direito do mouse sobre o item **Databases** e selecione a opção **Attach**:



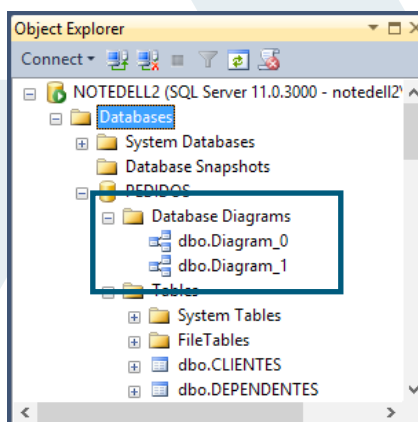
2. Na próxima tela, clique no botão **Add**:



3. Em seguida, procure a pasta **Dados** e selecione o arquivo **PEDIDOS_TABELAS.MDF**:

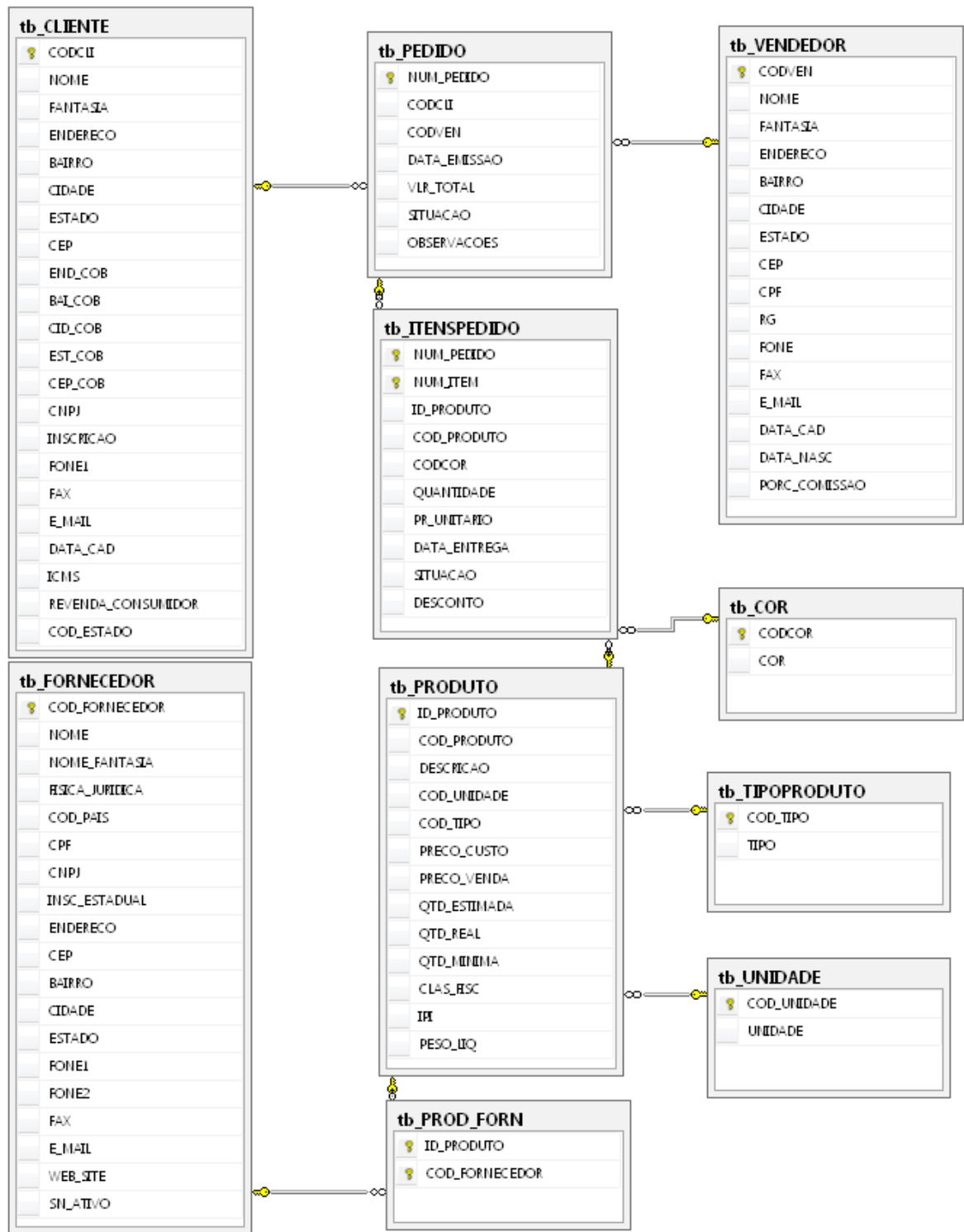


4. Confirme a operação clicando no botão **OK**.

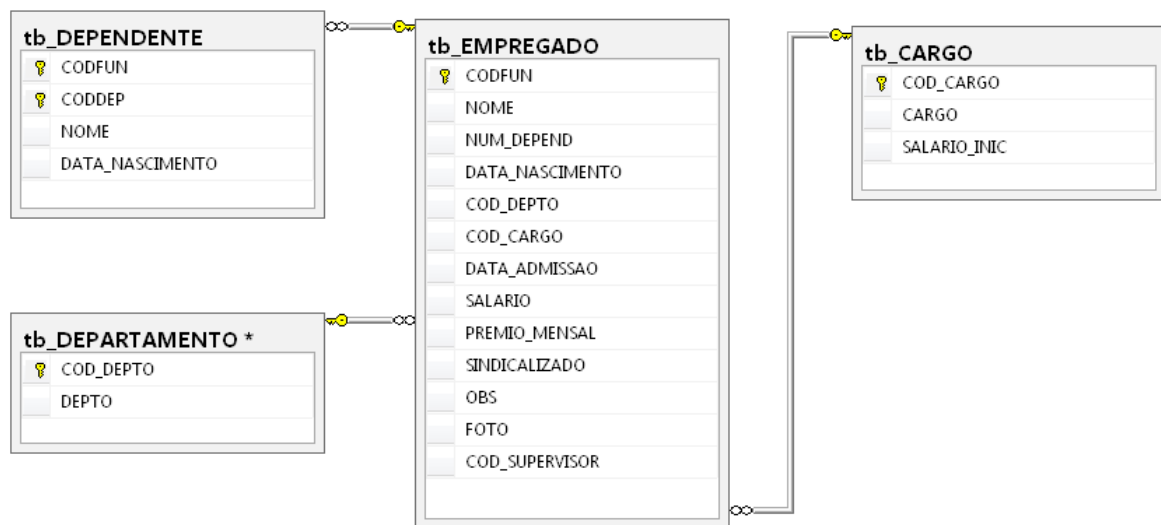


Observe que o banco de dados aparecerá no Object Explorer. Neste banco, foram criados dois diagramas que mostram as tabelas existentes nele. Você conseguirá visualizar o diagrama executando um duplo-clique sobre o nome:

• DIAGRAMA DE PEDIDOS



• DIAGRAMA DE EMPREGADOS



1.2.SELECT

O comando **SELECT** pertence ao grupo de comandos denominado **DML** (Data Manipulation Language, ou Linguagem de Manipulação de Dados), que é composto de comandos para consulta (**SELECT**), inclusão (**INSERT**), alteração (**UPDATE**) e exclusão de dados de tabela (**DELETE**).

A sintaxe de **SELECT**, com seus principais argumentos e cláusulas, é exibida a seguir:

```
SELECT [DISTINCT] [TOP (N) [PERCENT] [WITH TIES]] <lista_de_colunas>
[INTO <nome_tabela>]
FROM tabela1 [JOIN tabela2 ON expressaoJoin [, JOIN tabela3 ON ex-
prJoin [,...]]]
[WHERE <condicaoFiltroLinhas>]
[GROUP BY <listaExprGrupo> [HAVING <condicaoFiltroGrupo>]]
[ORDER BY <campo1> {[DESC] | [ASC]} [, <campo2> {[DESC] | [ASC]}
[,...]]]
```

Em que:

- **[DISTINCT]**: Palavra que especifica que apenas uma única instância de cada linha faça parte do conjunto de resultados. **DISTINCT** é utilizada com o objetivo de evitar a existência de linhas duplicadas no resultado da seleção;
- **[TOP (N) [PERCENT] [WITH TIES]]**: Especifica que apenas um primeiro conjunto de linhas ou uma porcentagem de linhas seja retornado. **N** pode ser um número ou porcentagem de linhas;

- **<lista_de_colunas>**: Colunas que serão selecionadas para o conjunto de resultados. Os nomes das colunas devem ser separados por vírgulas. Caso tais nomes não sejam especificados, todas as colunas serão consideradas na seleção;
- **[INTO nome_tabela]**: **nome_tabela** é o nome de uma nova tabela a ser criada com base nas colunas especificadas em **<lista_de_colunas>** e nas linhas especificadas por meio da cláusula **WHERE**;
- **FROM tabela1 [JOIN tabela2 ON exprJoin [, JOIN tabela3 ON exprJoin [...]]]**:
 - A cláusula **FROM** define tabelas utilizadas no **SELECT**;
 - **expressaoJoin** é a expressão necessária para relacionar as tabelas da cláusula **FROM**;
 - **tabela1, tabela2...** são as tabelas que possuem os valores utilizados na condição de filtragem **<condicaoFiltroLinhas>**.
- **[WHERE <condicaoFiltroLinhas>]**: A cláusula **WHERE** aplica uma condição de filtro que determinará quais linhas farão parte do resultado. Essa condição é especificada em **<condicaoFiltroLinhas>**;
- **[GROUP BY <listaExprGrupo>]**:
 - A cláusula **GROUP BY** agrupa uma quantidade de linhas em um conjunto de linhas. Nele, as linhas são resumidas por valores de uma ou várias colunas ou expressões;
 - **<listaExprGrupo>** representa a expressão na qual será realizada a operação por **GROUP BY**.
- **[HAVING <condicaoFiltroGrupo>]**: A cláusula **HAVING** define uma condição de busca para o grupo de linhas a ser retornado por **GROUP BY**;
- **[ORDER BY <campo1> {[DESC] | [ASC]} [, <campo2> {[DESC] | [ASC]} [...]]]**:
 - A cláusula **ORDER BY** é utilizada para determinar a ordem em que os resultados são retornados;
 - Já **campo1, campo2** são as colunas utilizadas na ordenação dos resultados.
- **{[DESC]/[ASC]}**: **ASC** determina que os valores das colunas especificadas em **campo1, campo2** sejam retornados em ordem ascendente, enquanto **DESC** retorna esses valores em ordem descendente. Ambas são opcionais e a barra indica que são excludentes entre si, ou seja, não podem ser utilizadas simultaneamente. As chaves indicam um grupo excludente de opções. Se nenhuma delas for utilizada, **ASC** será assumido.



Para consultar uma lista de colunas de uma determinada tabela em um banco de dados, basta utilizar a seguinte sintaxe:

```
SELECT <lista_de_colunas> FROM <tabela>
```

Em que:

- **<lista_de_colunas>**: Representa o nome da coluna ou colunas a serem selecionadas. Quando a consulta envolve mais de uma coluna, elas deverão ser separadas por vírgula;
- **<tabela>**: É o nome da tabela a partir de onde será feita a consulta.

Para especificar o banco de dados de origem das tabelas, a partir do qual as informações serão consultadas, utilize a instrução **USE** seguida pelo nome do banco de dados, da seguinte maneira:

```
USE <nome_banco_de_dados>
```

Essa instrução deve ser especificada na parte inicial da estrutura de código, anteriormente às instruções destinadas à consulta. Os exemplos adiante demonstrarão como utilizá-la junto ao **SELECT**.

1.2.1. Consultando todas as colunas

O código a seguir consulta todas as colunas da tabela **TB_EMPREGADO** do banco de dados **PEDIDOS**:

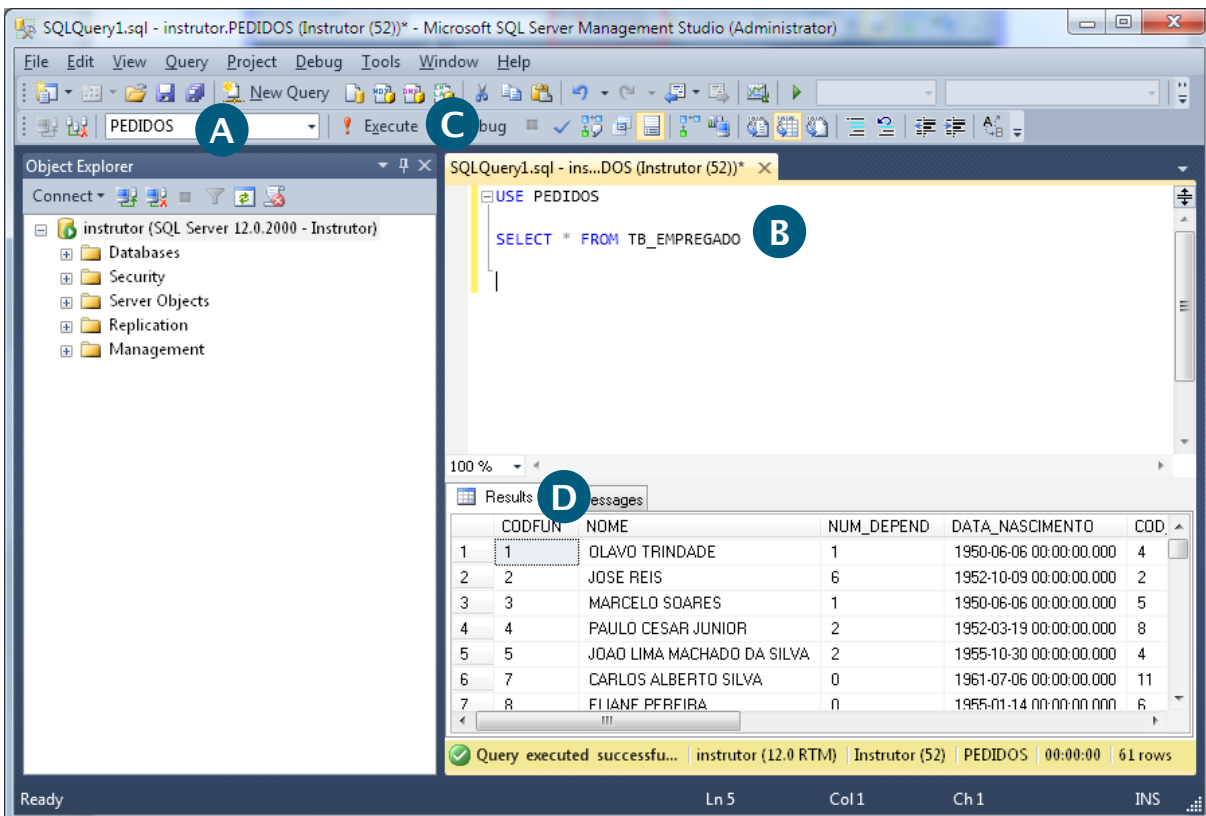
```
USE PEDIDOS;  
SELECT * FROM TB_EMPREGADO;
```


1.2.2.Consultando colunas específicas

Para consultar colunas específicas de uma tabela, deve-se especificar o(s) nome(s) da(s) coluna(s), como mostrado adiante:

```
SELECT <Coluna1, Coluna2, ...> FROM <tabela>
```

O código a seguir consulta todas as colunas da tabela **TB_EMPREGADO**:

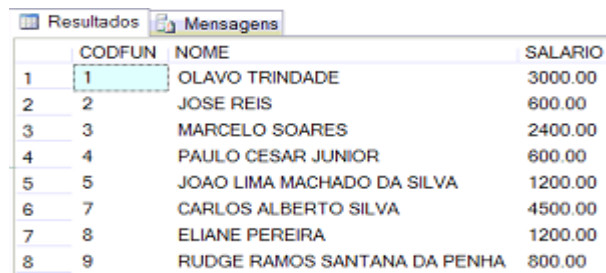


- **A** - Efeito do comando **USE PEDIDOS**. Também é possível selecionar o banco de dados por aqui;
- **B** - Instrução que queremos executar. É necessário selecionar o comando antes de executar;
- **C** - Botão que executa o comando selecionado. É importante saber que se nada estiver selecionado, o SQL tentará executar todos os comandos do script;
- **D** - Resultado da execução do comando **SELECT**.

Veja o seguinte exemplo, em que é feita a consulta nas colunas **CODFUN**, **NOME** e **SALARIO** da tabela **TB_EMPREGADO**:

```
SELECT CODFUN, NOME, SALARIO FROM TB_EMPREGADO;
```

Confira o resultado:



	CODFUN	NOME	SALARIO
1	1	OLAVO TRINDADE	3000.00
2	2	JOSE REIS	600.00
3	3	MARCELO SOARES	2400.00
4	4	PAULO CESAR JUNIOR	600.00
5	5	JOAO LIMA MACHADO DA SILVA	1200.00
6	7	CARLOS ALBERTO SILVA	4500.00
7	8	ELIANE PEREIRA	1200.00
8	9	RUDGE RAMOS SANTANA DA PENHA	800.00

O próximo exemplo efetua cálculos gerando colunas virtuais (não existentes fisicamente nas tabelas):

```
SELECT CODFUN, NOME, SALARIO, SALARIO * 1.10  
FROM TB_EMPREGADO;
```

Veja o resultado:



	CODFUN	NOME	SALARIO	(Nenhum nome de colu...)
1	1	OLAVO TRINDADE	3000.00	3300.0000
2	2	JOSE REIS	600.00	660.0000
3	3	MARCELO SOARES	2400.00	2640.0000
4	4	PAULO CESAR JUNIOR	600.00	660.0000
5	5	JOAO LIMA MACHADO DA SILVA	1200.00	1320.0000
6	7	CARLOS ALBERTO SILVA	4500.00	4950.0000
7	8	ELIANE PEREIRA	1200.00	1320.0000

Observe que não existe identificação para a coluna calculada.

1.2.3. Redefinindo os identificadores de coluna com uso de alias

O nome de uma coluna ou tabela pode ser substituído por uma espécie de apelido, que é criado para facilitar a visualização. Esse apelido é chamado de **alias**.

Costuma-se utilizar a cláusula **AS** a fim de facilitar a identificação do alias, no entanto, não é uma obrigatoriedade. A sintaxe para a utilização de alias é descrita a seguir:

```
SELECT <Coluna1> [[AS] <nome_alias>],  
       <Coluna2> [[AS] <nome_alias>] [,...]  
FROM <tabela>
```

Vejam os seguintes exemplos de consulta com uso de alias:

- Definindo um título para a coluna calculada

```
SELECT CODFUN, NOME, SALARIO,  
       SALARIO * 1.10 AS SALARIO MAIS 10 POR CENTO  
FROM TB_EMPREGADO;
```

Confira o resultado:

	CODFUN	NOME	SALARIO	SALARIO MAIS 10 POR CEN...
1	1	OLAVO TRINDADE	3000.00	3300.0000
2	2	JOSE REIS	600.00	660.0000
3	3	MARCELO SOARES	2400.00	2640.0000
4	4	PAULO CESAR JUNIOR	600.00	660.0000
5	5	JOAO LIMA MACHADO DA SILVA	1200.00	1320.0000
6	7	CARLOS ALBERTO SILVA	4500.00	4950.0000
7	8	ELIANE PEREIRA	1200.00	1320.0000
8	9	RUDGE RAMOS SANTANA DA PENHA	800.00	880.0000
9	10	MARIA CARMEM	1200.00	1320.0000
10	11	FERNANDO OLIVEIRA	1200.00	1320.0000
11	12	JOAO ROBERTO OLIVEIRA	1200.00	1320.0000
12	13	OSMAR PRADO	2400.00	2640.0000
13	14	CASSIANO OLIVEIRA	1200.00	1320.0000
14	15	MARCO ANTONIO	2400.00	2640.0000
15	16	ALTAMIR CARCIO	3300.00	3630.0000
16	17	ANA LUISA MARIA	1200.00	1320.0000

Na verdade, qualquer coluna da tabela pode receber um alias:

```
SELECT CODFUN AS Codigo,  
       NOME AS Nome, SALARIO AS Salario  
FROM TB_EMPREGADO;
```

Se o alias contiver caracteres como espaço, ou outros caracteres especiais, o SQL gera erro, a não ser que este nome seja delimitado por colchetes, apóstrofo ou aspas:

```
SELECT CODFUN AS Codigo, NOME AS Nome, SALARIO AS Salario,  
       DATA_ADMISSAO AS [Data de Admissão]  
FROM TB_EMPREGADO;
```

-- ou

```
SELECT CODFUN AS Codigo, NOME AS Nome, SALARIO AS Salario,  
       DATA_ADMISSAO AS 'Data de Admissão'  
FROM TB_EMPREGADO;
```

-- ou

```
SELECT CODFUN AS Codigo, NOME AS Nome, SALARIO AS Salario,  
       DATA_ADMISSAO AS "Data de Admissão"  
FROM TB_EMPREGADO;
```

-- Campo calculado

```
SELECT CODFUN AS Codigo,  
       NOME AS Nome,  
       SALARIO AS Salario,  
       SALARIO * 1.10 [Salário com 10% de Aumento]  
FROM TB_EMPREGADO
```

1.3. Ordenando dados

Utilizamos a cláusula **ORDER BY** em conjunto com o comando **SELECT** para retornar os dados em uma determinada ordem.

1.3.1. Retornando linhas na ordem ascendente

A cláusula **ORDER BY** pode ser utilizada com a opção **ASC**, que faz com que as linhas sejam retornadas em ordem ascendente.

Vejamos um exemplo:

```
SELECT * FROM TB_EMPREGADO ORDER BY NOME;  
SELECT * FROM TB_EMPREGADO ORDER BY NOME ASC;  
SELECT * FROM TB_EMPREGADO ORDER BY SALARIO;  
SELECT * FROM TB_EMPREGADO ORDER BY SALARIO ASC;  
  
SELECT * FROM TB_EMPREGADO ORDER BY DATA_ADMISSAO;
```

1.3.2. Retornando linhas na ordem descendente

A cláusula **ORDER BY** pode ser utilizada com a opção **DESC**, a qual faz com que as linhas sejam retornadas em ordem descendente.

Vejamos um exemplo:

```
SELECT * FROM TB_EMPREGADO ORDER BY NOME DESC;  
SELECT * FROM TB_EMPREGADO ORDER BY SALARIO DESC;  
SELECT * FROM TB_EMPREGADO ORDER BY DATA_ADMISSAO DESC;
```

Caso não especifiquemos **ASC** ou **DESC**, os dados da tabela serão retornados em ordem ascendente.

1.3.3. Ordenando por nome, alias ou posição

É possível utilizar a cláusula **ORDER BY** para ordenar dados retornados. Para isso, utilizamos como identificação da coluna a ser ordenada o seu próprio nome físico (caso exista), o seu alias ou a posição em que aparece na lista do **SELECT**.

- Usando o alias ou a posição da coluna como identificação do campo ordenado

```
-- Pela coluna SALARIO
SELECT CODFUN AS Código,
       NOME AS Nome,
       SALARIO AS Salário,
       SALARIO * 1.10 [Salário com 10% de aumento]
FROM TB_EMPREGADO
ORDER BY Salário;
```

```
-- Idem ao anterior
SELECT CODFUN AS Código,
       NOME AS Nome,
       SALARIO AS Salário,
       SALARIO * 1.10 [Salário com 10% de aumento]
FROM TB_EMPREGADO
ORDER BY 3;
```

```
-- Pela coluna SALARIO * 1.10
SELECT CODFUN AS Código,
       NOME AS Nome,
       SALARIO AS Salário,
       SALARIO * 1.10 [Salário com 10% de aumento]
FROM TB_EMPREGADO
ORDER BY [Salário com 10% de Aumento];
```

```
-- Idem ao anterior
SELECT CODFUN AS Código,
       NOME AS Nome,
       SALARIO AS Salário,
       SALARIO * 1.10 [Salário com 10% de aumento]
FROM TB_EMPREGADO
ORDER BY 4;
```

Vejamos outro exemplo de retorno de dados de acordo com o nome da coluna:

```
SELECT CODFUN, NOME, DATA_ADMISSAO, SALARIO
FROM TB_EMPREGADO
ORDER BY SALARIO;

--

SELECT CODFUN, NOME, DATA_ADMISSAO, SALARIO
FROM TB_EMPREGADO
ORDER BY DATA_ADMISSAO;
```

- Ordenando por várias colunas

Quando a coluna ordenada contém informação repetida, essa informação formará grupos. Observe o exemplo:

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

	COD_DEP...	NOME	DATA_ADMISSAO	SALARIO
9	1	ROGÉRIO FREITAS	1980-01-01 00:00:00.000	4500.00
10	1	RONALDO MATIAS	1990-01-01 00:00:00.000	3300.00
11	1	JOSÉ CARLOS MOREIRA	2000-01-01 00:00:00.000	8300.00
12	1	JOÃO CARLOS DE OLIVEIRA	2000-01-01 00:00:00.000	5000.00
13	1	JOSÉ CARLOS SILVA	1998-01-01 00:00:00.000	3300.00
14	1	CASSIANO OLIVEIRA	1993-02-03 00:00:00.000	1200.00
15	1	ROBERTO PINHEIRO	1981-12-12 00:00:00.000	8300.00
16	2	SEBASTIÃO SILVA	1988-04-06 00:00:00.000	8300.00
17	2	EURICO BRANDÃO	1988-01-09 00:00:00.000	800.00
18	2	JOSE REIS	1987-05-02 00:00:00.000	600.00
19	2	RUDGE RAMOS SANTANA DA PENHA	1985-12-23 00:00:00.000	800.00
20	2	MARIA DA PENHA	1983-07-15 00:00:00.000	4500.00
21	2	MARIANO DE OLIVEIRA	1993-04-03 00:00:00.000	3330.00
22	2	LUIS FERNANDO LEMOS	2005-01-01 00:00:00.000	600.00
23	2	JOAQUIM ALBERTO	2003-04-05 00:00:00.000	500.00
24	3	MARIANA DA SILVA	2006-01-01 00:00:00.000	500.00

Nesse caso, pode ser útil ordenar outra coluna dentro do grupo formado pela primeira:

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

	COD_DEPTO	NOME	DATA_ADMISSAO	SALARIO
1	NULL	JORGE DOS SANTOS ROCHA JUNIOR	2000-07-01 00:00:00.000	NULL
2	NULL	SEVERINO CARLOS MACIEIRA	2000-07-01 00:00:00.000	NULL
3	1	ARNALDO MOURA	1990-01-01 00:00:00.000	890.00
4	1	CASSIANO OLIVEIRA	1993-02-03 00:00:00.000	1200.00
5	1	JOÃO CARLOS DE OLIVEIRA	2000-01-01 00:00:00.000	5000.00
6	1	JORGE ROBERTO SOUZA	2001-10-10 00:00:00.000	4500.00
7	1	JOSÉ CARLOS MOREIRA	2000-01-01 00:00:00.000	8300.00
8	1	JOSÉ CARLOS SILVA	1998-01-01 00:00:00.000	3300.00
9	1	PEDRO PAULO SOUZA	2006-06-24 00:00:00.000	890.00
10	1	ROBERTO CARLOS DA SILVA	2006-06-24 00:00:00.000	4500.00
11	1	ROBERTO MARILDO	2001-09-11 00:00:00.000	800.00
12	1	ROBERTO PINHEIRO	1981-12-12 00:00:00.000	8300.00
13	1	ROGÉRIO FREITAS	1980-01-01 00:00:00.000	4500.00
14	1	RONALDO MATIAS	1990-01-01 00:00:00.000	3300.00
15	2	EURICO BRANDÃO	1988-01-09 00:00:00.000	800.00
16	2	JOAQUIM ALBERTO	2003-04-05 00:00:00.000	500.00

Note que, dentro de cada departamento, os dados estão ordenados pela coluna **NOME**:

```
--
SELECT COD_DEPTO, NOME, DATA_ADMISSAO, SALARIO
FROM TB_EMPREGADO
ORDER BY COD_DEPTO, SALARIO;
--
SELECT COD_DEPTO, NOME, DATA_ADMISSAO, SALARIO
FROM TB_EMPREGADO
ORDER BY COD_DEPTO, DATA_ADMISSAO;
-- Continua valendo o uso do "alias" ou da posição da
-- coluna
SELECT COD_DEPTO, NOME, DATA_ADMISSAO, SALARIO
FROM TB_EMPREGADO
ORDER BY 1, 3;
```

O uso da opção **DESC** (ordenação descendente) é independente para cada coluna no **ORDER BY**:

```
SELECT COD_DEPTO, NOME, DATA_ADMISSAO, SALARIO
FROM TB_EMPREGADO
ORDER BY COD_DEPTO DESC, SALARIO;
--
SELECT COD_DEPTO, NOME, DATA_ADMISSAO, SALARIO
FROM TB_EMPREGADO
ORDER BY COD_DEPTO, SALARIO DESC;
--
SELECT COD_DEPTO, NOME, DATA_ADMISSAO, SALARIO
FROM TB_EMPREGADO
ORDER BY COD_DEPTO DESC, SALARIO DESC;
```

1.3.4.ORDER BY com TOP

A cláusula **TOP** mostra as **N** primeiras linhas de um **SELECT**, no entanto, se a usarmos sem a cláusula **ORDER BY**, o resultado ficará sem sentido. Vejamos o exemplo:

```
-- Lista os 5 primeiros empregados de acordo com a chave
-- primária
SELECT TOP 5 * FROM TB_EMPREGADO;
```

	CODFUN	NOME	NUM_DEPE...	DATA_NASCIMENTO	COD_DEP...	COD_CAR...
1	1	OLAVO TRINDADE	1	1950-06-06 00:00:00.000	4	17
2	2	JOSE REIS	6	1952-10-09 00:00:00.000	2	14
3	3	MARCELO SOARES	1	1950-06-06 00:00:00.000	5	2
4	4	PAULO CESAR JUNIOR	2	1952-03-19 00:00:00.000	8	14
5	5	JOAO LIMA MACHADO DA SILVA	2	1955-10-30 00:00:00.000	4	3

Não sabemos quem são esses cinco funcionários listados. Provavelmente, são os primeiros a serem inseridos na tabela **TB_EMPREGADO**, mas nem isso podemos afirmar com certeza.

Já nos exemplos a seguir, conseguimos compreender o resultado:

-- Lista os 5 empregados mais antigos

```
SELECT TOP 5 * FROM TB_EMPREGADO  
ORDER BY DATA_ADMISSAO;
```

-- Lista os 5 empregados mais novos

```
SELECT TOP 5 * FROM TB_EMPREGADO  
ORDER BY DATA_ADMISSAO DESC;
```

-- Lista os 5 empregados que ganham menos

```
SELECT TOP 5 * FROM TB_EMPREGADO  
ORDER BY SALARIO;
```

-- Lista os 5 empregados que ganham mais

```
SELECT TOP 5 * FROM TB_EMPREGADO  
ORDER BY SALARIO DESC;
```

1.3.5.ORDER BY com TOP WITH TIES

TOP WITH TIES é permitida apenas em instruções **SELECT** e quando uma cláusula **ORDER BY** é especificada. Indica que se o conteúdo do campo ordenado na última linha da cláusula **TOP** se repetir em outras linhas, estas deverão ser exibidas também.

Observe a sequência:

```
SELECT CODFUN, NOME, SALARIO  
FROM TB_EMPREGADO  
ORDER BY SALARIO DESC;
```

	CODFUN	NOME	SALARIO
1	18	ROBERTO PINHEIRO	8300.00
2	19	SEBASTIÃO SILVA	8300.00
3	66	JOSÉ CARLOS MOREIRA	8300.00
4	43	JOÃO CARLOS DE OLIVEIRA	5000.00
5	26	ANA MARIA OLIVEIRA	5000.00
6	27	RICARDO SOUZA	5000.00
7	25	MARIA DA PENHA	4500.00
8	7	CARLOS ALBERTO SILVA	4500.00
9	53	ROGÉRIO FREITAS	4500.00
10	51	JORGE ROBERTO SOUZA	4500.00
11	59	MANOEL RIBEIRO	4500.00
12	72	ROBERTO CARLOS DA SILVA	4500.00
13	58	ALBERTO HELENA SILVA	3330.00
14	28	MARIANO DE OLIVEIRA	3330.00
15	16	ALTAMIR CARCIO	3300.00
16	31	MANOEL SANTOS	3300.00

Esse exemplo lista os empregados em ordem descendente de salário. Note que no sétimo registro o salário é de 4500.00 e este valor se repete nos cinco registros seguintes. Se aplicarmos a cláusula **TOP 7**, qual dos seis funcionários com salário de 4500.00 será mostrado, já que o valor é o mesmo?

-- Listar os 7 funcionários que ganham mais

```
SELECT TOP 7 CODFUN, NOME, SALARIO  
FROM TB_EMPREGADO  
ORDER BY SALARIO DESC;
```

	CODFUN	NOME	SALARIO
1	18	ROBERTO PINHEIRO	8300.00
2	19	SEBASTIÃO SILVA	8300.00
3	66	JOSÉ CARLOS MOREIRA	8300.00
4	26	ANA MARIA OLIVEIRA	5000.00
5	27	RICARDO SOUZA	5000.00
6	43	JOÃO CARLOS DE OLIVEIRA	5000.00
7	7	CARLOS ALBERTO SILVA	4500.00

Por qual razão o SQL selecionou o funcionário de **CODFUN 7** como último da lista, se existem outros cinco funcionários com o mesmo salário? Porque ele tem a menor chave primária.

Na maioria das consultas, quando um fato como esse ocorre (empate na última linha), o critério para desempate, se houver, dificilmente será pela menor chave primária. Então, seria interessante que a consulta mostrasse todas as linhas em que o salário fosse o mesmo da última:

-- Listar os 7 empregados que ganham mais, inclusive

-- aqueles empatados com o último

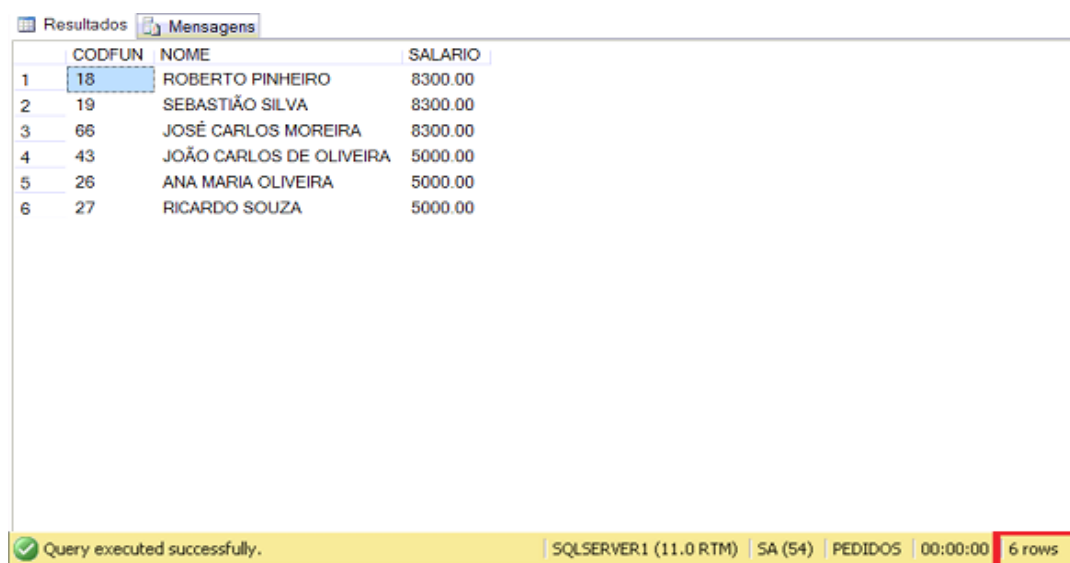
```
SELECT TOP 7 WITH TIES CODFUN, NOME, SALARIO FROM TB_EMPREGADO  
ORDER BY SALARIO DESC;
```

	CODFUN	NOME	SALARIO
1	18	ROBERTO PINHEIRO	8300.00
2	19	SEBASTIÃO SILVA	8300.00
3	66	JOSÉ CARLOS MOREIRA	8300.00
4	43	JOÃO CARLOS DE OLIVEIRA	5000.00
5	26	ANA MARIA OLIVEIRA	5000.00
6	27	RICARDO SOUZA	5000.00
7	25	MARIA DA PENHA	4500.00
8	7	CARLOS ALBERTO SILVA	4500.00
9	53	ROGÉRIO FREITAS	4500.00
10	51	JORGE ROBERTO SOUZA	4500.00
11	59	MANOEL RIBEIRO	4500.00
12	72	ROBERTO CARLOS DA SILVA	4500.00

Também podemos usar a cláusula **TOP** com percentual. A tabela **TB_EMPREGADO** possui sessenta linhas, então, se pedirmos pra ver 10% das linhas, deverão aparecer seis. Confira:

```
-- Mostrar 10% das linhas da tabela TB_EMPREGADO
SELECT TOP 10 PERCENT CODFUN, NOME, SALARIO FROM TB_EMPREGADO
ORDER BY SALARIO DESC;
```

São exibidas as seguintes linhas:



	CODFUN	NOME	SALARIO
1	18	ROBERTO PINHEIRO	8300.00
2	19	SEBASTIÃO SILVA	8300.00
3	66	JOSÉ CARLOS MOREIRA	8300.00
4	43	JOÃO CARLOS DE OLIVEIRA	5000.00
5	26	ANA MARIA OLIVEIRA	5000.00
6	27	RICARDO SOUZA	5000.00

1.3.6. Filtrando consultas

O exemplo a seguir demonstra o que vimos até aqui sobre a instrução **SELECT**:

```
SELECT [TOP (n) [PERCENT] [WITH TIES]]
       <lista_de_colunas>|*
FROM <nome_da_tabela>
[WHERE <critério_de_filtro>]
[ORDER BY <coluna1> [ASC|DESC] [, <coluna2> [ASC|DESC] [, ...]]]
```

A cláusula **WHERE** determina um critério de filtro e que somente as linhas que respeitem esse critério sejam exibidas. A expressão contida no critério de filtro deve retornar **TRUE** (verdadeiro) ou **FALSE** (falso).

1.4. Operadores relacionais

A tabela a seguir exibe os operadores relacionais:

Operador	Descrição
=	Compara, se igual.
<> ou !=	Compara, se diferentes.
>	Compara, se maior que.
<	Compara, se menor que.
>=	Compara, se maior que ou igual.
<=	Compara, se menor que ou igual.

Operadores relacionais sempre terão dois operandos, um à esquerda e outro à sua direita.

Considere os seguintes exemplos:

- **Mostrando os funcionários com SALÁRIO abaixo de 1000**

```
SELECT CODFUN, NOME, COD_CARGO, SALARIO FROM TB_EMPREGADO  
WHERE SALARIO < 1000  
ORDER BY SALARIO;
```

- **Mostrando os funcionários com SALÁRIO acima de 5000**

```
SELECT CODFUN, NOME, COD_CARGO, SALARIO FROM TB_EMPREGADO  
WHERE SALARIO > 5000  
ORDER BY SALARIO;
```

- **Mostrando os funcionários com campo COD_DEPTO menor ou igual a 3**

```
SELECT * FROM TB_EMPREGADO  
WHERE COD_DEPTO <= 3  
ORDER BY COD_DEPTO;
```

- **Mostrando os funcionários com campo COD_DEPTO igual a 2**

```
SELECT * FROM TB_EMPREGADO  
WHERE COD_DEPTO = 2  
ORDER BY COD_DEPTO;
```

- Mostrando os funcionários com campo COD_DEPTO diferente de 2

```
SELECT * FROM TB_EMPREGADO  
WHERE COD_DEPTO <> 2  
ORDER BY COD_DEPTO;
```

Embora pareça estranho, os sinais relacionais também podem ser usados para campos alfanuméricos. Vejamos os seguintes exemplos:

- Mostrando os funcionários que tenham NOME alfabeticamente maior que RAQUEL

```
SELECT CODFUN, NOME, SALARIO  
FROM TB_EMPREGADO  
WHERE NOME > 'RAQUEL'  
ORDER BY NOME;
```

	CODFUN	NOME	SALARIO
1	69	RENAN CARLOS DE OLIVEIRA	3300.00
2	27	RICARDO SOUZA	5000.00
3	72	ROBERTO CARLOS DA SILVA	4500.00
4	49	ROBERTO MARILDO	800.00
5	18	ROBERTO PINHEIRO	8300.00
6	53	ROGÉRIO FREITAS	4500.00
7	61	RONALDO MATIAS	3300.00
8	9	RUDGE RAMOS SANTANA DA PENHA	800.00
9	19	SEBASTIÃO SILVA	8300.00
10	68	SEVERINO CARLOS MACIEIRA	NULL
11	21	SILVIO OLIVEIRA	500.00

- Mostrando os funcionários que tenham NOME alfabeticamente menor que ELIANA

```
SELECT CODFUN, NOME, SALARIO  
FROM TB_EMPREGADO  
WHERE NOME < 'ELIANA'  
ORDER BY NOME;
```

	CODFUN	NOME	SALARIO
1	58	ALBERTO HELENA SILVA	3330.00
2	16	ALTAMIR CARCIO	3300.00
3	17	ANA LUISA MARIA	1200.00
4	47	ANA LUIZA SOUSA	800.00
5	26	ANA MARIA OLIVEIRA	5000.00
6	57	ANTONIO CARLOS	500.00
7	55	ARLINDO SOARES	500.00
8	48	ARMANDO LEMOS BRITO	1200.00
9	22	ARNALDO FARIA	800.00
10	52	ARNALDO MOURA	890.00
11	50	AUGUSTO SILVEIRA DA SILVA	890.00
12	7	CARLOS ALBERTO SILVA	4500.00
13	30	CARLOS MAGNO P SOUZA	1200.00
14	29	CARLOS ROBERTO DA SILVA	2400.00
15	46	CARLOS ROBERTO JUNIOR	3300.00
16	14	CASSIANO OLIVEIRA	1200.00

1.5. Operadores lógicos

A filtragem de dados em uma consulta também pode ocorrer com a utilização dos operadores lógicos **AND**, **OR** ou **NOT**, cada qual permitindo uma combinação específica de expressões, conforme apresentado adiante:

- O operador **AND** combina duas expressões e exige que sejam verdadeiras, ou seja, **TRUE**;
- O operador **OR** verifica se pelo menos uma das expressões retornam **TRUE**;
- O operador **NOT** inverte o resultado lógico da expressão à sua direita, ou seja, se a expressão é verdadeira ele retorna falso e vice-versa.

Acompanhe os seguintes exemplos:

- **Mostrando funcionários do departamento 2 que ganhem mais de 5000**

```
SELECT * FROM TB_EMPREGADO
WHERE COD_DEPTO = 2 AND SALARIO > 5000;
```

	CODFUN	NOME	NUM_DEPEND	DATA_NASCIMENTO	COD_DEPTO	COD_CARGO	DATA_ADMISS
1	19	SEBASTIÃO SILVA	0	1951-12-19 00:00:00.000	2	1	1988-04-06 00:

- **Mostrando funcionários do departamento 2 ou aqueles que ganhem mais de 5000**

```
SELECT * FROM TB_EMPREGADO
WHERE COD_DEPTO = 2 OR SALARIO > 5000;
```

	CODFUN	NOME	NUM_DEPE...	DATA_NASI
1	2	JOSE REIS	6	1952-10-09
2	9	RUDGE RAMOS SANTANA DA PENHA	3	1961-07-22
3	18	ROBERTO PINHEIRO	4	1950-04-12
4	19	SEBASTIÃO SILVA	0	1951-12-19
5	20	EURICO BRANDÃO	0	1950-04-19
6	25	MARIA DA PENHA	0	1958-02-12
7	28	MARIANO DE OLIVEIRA	3	1954-01-18
8	38	LUIS FERNANDO LEMOS	0	1978-07-23
9	40	JOAQUIM ALBERTO	0	1991-03-04
10	66	JOSÉ CARLOS MOREIRA	0	1900-01-01

É importante saber onde utilizar o **AND** e o **OR**. Vamos supor que foi pedido para listar todos os funcionários do **COD_DEPTO** igual a 2 e também igual a 5. Se fossemos escrever o comando exatamente como foi pedido, digitaríamos o seguinte:

```
SELECT * FROM TB_EMPREGADO
WHERE COD_DEPTO = 2 AND COD_DEPTO = 5;
```

No entanto, essa consulta não vai produzir nenhuma linha de resultado. Isso porque um mesmo empregado não está cadastrado nos departamentos 2 e 5 simultaneamente. Um empregado está cadastrado ou (**OR**) no departamento 2 ou (**OR**) no departamento 5.

Precisamos entender que a pessoa que solicita a consulta está visualizando o resultado pronto e acaba utilizando "e" (**AND**) no lugar de "ou" (**OR**). Sendo assim, é importante saber que, na execução do **SELECT**, ele avalia os dados linha por linha. Então, o correto é o seguinte:

```
SELECT * FROM TB_EMPREGADO
WHERE COD_DEPTO = 2 OR COD_DEPTO = 5;
```

Vejamos outros exemplos da utilização de **AND** e **OR**:

- **Mostrando funcionários com SALARIO entre 3000 e 5000**

```
SELECT * FROM TB_EMPREGADO
WHERE SALARIO >= 3000 AND SALARIO <= 5000
ORDER BY SALARIO;
```

- **Mostrando funcionários com SALARIO abaixo de 3000 ou acima de 5000**

```
SELECT * FROM TB_EMPREGADO
WHERE SALARIO < 3000 OR SALARIO > 5000
ORDER BY SALARIO;
-- Também pode ser feito usando o operador NOT. Aqueles
-- que não estão entre 3000 e 5000, estão fora dessa
-- faixa.
SELECT * FROM TB_EMPREGADO
WHERE NOT (SALARIO >= 3000 AND SALARIO <= 5000)
ORDER BY SALARIO;
```

1.6.Consultando intervalos com BETWEEN

A cláusula **BETWEEN** permite filtrar dados em uma consulta tendo como base uma faixa de valores, ou seja, um intervalo entre um valor menor e outro maior. Podemos utilizá-la no lugar de uma cláusula **WHERE** com várias expressões contendo os operadores **>=** e **<=** ou interligadas pelo operador **OR**.

A funcionalidade da cláusula **BETWEEN** assemelha-se à dos operadores **AND**, **>=** e **<=**, no entanto, vale considerar que, por meio dela, a consulta torna-se ainda mais simples de ser realizada.

Os dois comandos a seguir são equivalentes, ou seja, exibem o mesmo resultado:

- **Funcionários com SALARIO entre 3000 e 5000**

```
SELECT * FROM TB_EMPREGADO
WHERE SALARIO >= 3000 AND SALARIO <= 5000
ORDER BY SALARIO;
```

```
SELECT * FROM TB_EMPREGADO
WHERE SALARIO BETWEEN 3000 AND 5000
ORDER BY SALARIO;
```

O operador **BETWEEN** também pode ser usado para dados do tipo data ou alfanuméricos:

```
SELECT * FROM TB_EMPREGADO
WHERE DATA_ADMISSAO BETWEEN '2000.1.1' AND '2000.12.31'
ORDER BY DATA_ADMISSAO;
```

Além de **BETWEEN**, podemos utilizar **NOT BETWEEN**, que permite consultar os valores que não se encontram em uma determinada faixa de valores. O exemplo a seguir pesquisa valores não compreendidos no intervalo especificado:

```
SELECT * FROM TB_EMPREGADO
WHERE SALARIO < 3000 OR SALARIO > 5000
ORDER BY SALARIO;
```

Em vez de <, > e **OR**, podemos utilizar **NOT BETWEEN** mais o operador **AND** para pesquisar os mesmo valores da consulta anterior:

```
SELECT * FROM TB_EMPREGADO
WHERE SALARIO NOT BETWEEN 3000 AND 5000
ORDER BY SALARIO;
-- OU ENTÃO
SELECT * FROM TB_EMPREGADO
WHERE NOT SALARIO BETWEEN 3000 AND 5000
ORDER BY SALARIO;
```

1.7.Consulta com base em caracteres

O operador **LIKE** é usado para fazer pesquisas em dados do tipo string (**CHAR**, **VARCHAR**, **NCHAR** e **NVARCHAR**). É útil quando não sabemos de forma exata o dado que queremos pesquisar. Por exemplo, sabemos que o nome da pessoa começa com MARIA, mas não sabemos o restante do nome, ou sabemos que o nome contém a palavra RAMOS, mas não sabemos o nome completo.

Vejamos os seguintes exemplos:

- **Nomes que começam com MARIA**

```
SELECT * FROM TB_EMPREGADO  
WHERE NOME LIKE 'MARIA%';
```

O sinal % é um curinga que equivale a uma quantidade qualquer de caracteres, inclusive nenhum.

- **Nomes que começam com MA**

```
SELECT * FROM TB_EMPREGADO  
WHERE NOME LIKE 'MA%';
```

- **Nomes que começam com M**

```
SELECT * FROM TB_EMPREGADO  
WHERE NOME LIKE 'M%';
```

Na verdade, esse recurso de consulta pode buscar palavras que estejam contidas no texto, seja no início, no meio ou no final dele. Acompanhe outros exemplos:

- **Nomes que terminam com MARIA**

```
SELECT * FROM TB_EMPREGADO  
WHERE NOME LIKE '%MARIA';
```

- **Nomes que terminam com SOUZA**

```
SELECT * FROM TB_EMPREGADO  
WHERE NOME LIKE '%SOUZA';
```

- **Nomes que terminam com ZA**

```
SELECT * FROM TB_EMPREGADO  
WHERE NOME LIKE '%ZA';
```

- **Nomes contendo a palavra MARIA**

```
SELECT * FROM TB_EMPREGADO  
WHERE NOME LIKE '%MARIA%';
```

- **Nomes contendo a palavra SOUZA**

```
SELECT * FROM TB_EMPREGADO  
WHERE NOME LIKE '%SOUZA%';
```


Outro caractere curinga que pode ser utilizado em consultas com **LIKE** é o subscrito (**_**), que equivale a um único caractere qualquer. Veja alguns exemplos:

- **Nomes iniciados por qualquer caractere, mas que o segundo caractere seja a letra A**

```
SELECT * FROM TB_EMPREGADO  
WHERE NOME LIKE '_A%';
```

- **Nomes cujo penúltimo caractere seja a letra Z**

```
SELECT * FROM TB_EMPREGADO  
WHERE NOME LIKE '%Z_';
```

- **Nomes terminados em LU e seguidos de 3 outras letras**

```
SELECT * FROM TB_EMPREGADO  
WHERE NOME LIKE '%LU___';
```

Podemos, também, fornecer várias opções para um determinado caractere da chave de busca, como no exemplo a seguir, que busca nomes que contenham **SOUZA** ou **SOUSA**:

```
SELECT * FROM TB_EMPREGADO  
WHERE NOME LIKE '%SOU[SZ]A%';
```

Veja outro exemplo, que busca nomes contendo **JOSÉ** ou **JOSE**:

```
SELECT * FROM TB_EMPREGADO  
WHERE NOME LIKE '%JOS[EÉ]%' ;
```

Além disso, podemos utilizar o operador **NOT LIKE**, que atua de forma oposta ao operador **LIKE**. Com **NOT LIKE**, obtemos como resultado de uma consulta os valores que não possuem os caracteres ou sílabas determinadas.

O exemplo a seguir busca nomes que não contenham a palavra **MARIA**:

```
SELECT * FROM TB_EMPREGADO  
WHERE NOME NOT LIKE '%MARIA%';
```

O exemplo a seguir busca nomes que não contenham a sílaba **MA**:

```
SELECT * FROM TB_EMPREGADO  
WHERE NOME NOT LIKE '%MA%';
```

1.8.Consultando valores pertencentes ou não a uma lista de elementos

O operador **IN**, que pode ser utilizado no lugar do operador **OR** em determinadas situações, permite verificar se o valor de uma coluna está presente em uma lista de elementos.

O operador **NOT IN**, por sua vez, ao contrário de **IN**, permite obter como resultado o valor de uma coluna que não pertence a uma determinada lista de elementos.

O exemplo a seguir busca todos os empregados cujo código do departamento (**COD_DEPTO**) seja **1, 3, 4** ou **7**:

```
SELECT * FROM TB_EMPREGADO
WHERE COD_DEPTO IN (1,3,4,7)
ORDER BY COD_DEPTO;
```

O exemplo adiante busca, nas colunas **NOME** e **ESTADO** da tabela **TB_CLIENTE**, os clientes dos estados do Amazonas (**AM**), Paraná (**PR**), Rio de Janeiro (**RJ**) e São Paulo (**SP**):

```
SELECT NOME, ESTADO FROM TB_CLIENTE
WHERE ESTADO IN ('AM', 'PR', 'RJ', 'SP');
```

Já o próximo exemplo busca, nas colunas **NOME** e **ESTADO** da tabela **TB_CLIENTE**, os clientes que não são dos estados do Amazonas (**AM**), Paraná (**PR**), Rio de Janeiro (**RJ**) e São Paulo (**SP**):

```
SELECT NOME, ESTADO FROM TB_CLIENTE
WHERE ESTADO NOT IN ('AM', 'PR', 'RJ', 'SP');
```

1.9.Lidando com valores nulos

Quando um **INSERT** não faz referência a uma coluna existente em uma tabela, o conteúdo dessa coluna ficará nulo (**NULL**):

```
-- Este INSERT menciona apenas a coluna NOME,
-- as outras colunas da tabela ficarão
-- com seu conteúdo NULO
INSERT INTO TB_EMPREGADO (NOME)
VALUES ('JOSE EMANUEL');

-- Ver o resultado
SELECT * FROM TB_EMPREGADO;
```

Confira o resultado:

59	70	PEDRO PAULO SOUZA	0	1980-07-01 00:00:00.000	1	6
60	72	ROBERTO CARLOS DA SILVA	0	2000-01-01 00:00:00.000	1	11
61	73	JOSE EMANUEL	NULL	NULL	NULL	NULL

Com relação a valores nulos, vale considerar as seguintes informações:

- Não é um valor zero, nem uma string vazia. É **NULL**;
- Valores nulos não aparecem quando o campo faz parte da cláusula **WHERE**, a não ser que a cláusula deixe explícito que deseja visualizar também os nulos;
- Se envolvido em cálculos, retorna sempre um valor **NULL**.

Observe a consulta a seguir e note que o valor nulo que inserimos anteriormente não aparece nem entre os menores salários e nem entre os maiores:

```
SELECT CODFUN, NOME, SALARIO FROM TB_EMPREGADO
WHERE SALARIO < 800 OR SALARIO > 8000
ORDER BY SALARIO;
```

	CODFUN	NOME	SALARIO
1	21	SILVIO OLIVEIRA	500.00
2	23	JOAQUIM JUNIOR FILHO	500.00
3	35	MARIANA DA SILVA	500.00
4	40	JOAQUIM ALBERTO	500.00
5	45	MARIA LUIZA	500.00
6	55	ARLINDO SOARES	500.00
7	57	ANTONIO CARLOS	500.00
8	38	LUIS FERNANDO LEMOS	600.00
9	2	JOSE REIS	600.00
10	4	PAULO CESAR JUNIOR	600.00
11	18	ROBERTO PINHEIRO	8300.00
12	19	SEBASTIÃO SILVA	8300.00
13	66	JOSÉ CARLOS MOREIRA	8300.00

Como dito anteriormente, caso faça parte de cálculo, o resultado é sempre **NULL**:

```
SELECT CODFUN, NOME, SALARIO, PREMIO_MENSAL,
       SALARIO + PREMIO_MENSAL AS RENDA_TOTAL
FROM TB_EMPREGADO
```

```
-- filtrar somente os nulos
WHERE SALARIO IS NULL;
```

	CODFUN	NOME	SALARIO	PREMIO_MENSAL	RENDA_TOTAL
1	44	JORGE DOS SANTOS ROCHA JUNIOR	NULL	528.71	NULL
2	68	SEVERINO CARLOS MACIEIRA	NULL	528.71	NULL
3	73	JOSE EMANUEL	NULL	NULL	NULL

Para lidar com valores nulos, evitando problemas no resultado final da consulta, pode-se empregar funções como **IS NULL**, **IS NOT NULL**, **NULLIF** e **COALESCE**, as quais serão apresentadas nos tópicos subsequentes.

O exemplo a seguir busca, na tabela **TB_EMPREGADO**, os registros cujo **COD_CARGO** seja nulo:

```
SELECT * FROM TB_EMPREGADO
WHERE COD_CARGO IS NULL;
```

Este exemplo busca, na tabela **TB_EMPREGADO**, os registros cuja **DATA_NASCIMENTO** seja nula:

```
SELECT * FROM TB_EMPREGADO
WHERE DATA_NASCIMENTO IS NULL;
```

O exemplo adiante busca, na tabela **TB_EMPREGADO**, os registros cuja **DATA_NASCIMENTO** não seja nula:

```
SELECT * FROM TB_EMPREGADO
WHERE DATA_NASCIMENTO IS NOT NULL;
```

1.10.Substituindo valores nulos

Em síntese, as funções **ISNULL** e **COALESCE** permitem retornar outros valores quando valores nulos são encontrados durante a filtragem de dados. Adiante, apresentaremos a descrição dessas funções.

1.10.1.ISNULL

Esta função permite definir um valor alternativo que será retornado, caso o valor de argumento seja nulo. Vejamos os exemplos adiante:

- **Exemplo 1**

```
SELECT
    CODFUN, NOME, SALARIO, PREMIO_MENSAL,
    ISNULL(SALARIO,0) + ISNULL(PREMIO_MENSAL,0) AS RENDA_TOTAL
FROM TB_EMPREGADO
WHERE SALARIO IS NULL;
```

	CODFUN	NOME	SALARIO	PREMIO_MENSAL	RENDA_TOTAL
1	44	JORGE DOS SANTOS ROCHA JUNIOR	NULL	528.71	528.71
2	68	SEVERINO CARLOS MACIEIRA	NULL	528.71	528.71
3	73	JOSE EMANUEL	NULL	NULL	0.00

- **Exemplo 2**

```
SELECT
    CODFUN, NOME,
    ISNULL(DATA_NASCIMENTO, '1900.1.1') AS DATA_NASC
FROM TB_EMPREGADO;
```

1.10.2.COALESCE

Esta função é responsável por retornar o primeiro argumento não nulo em uma lista de argumentos testados.

O código a seguir tenta exibir o campo **EST_COB** dos clientes da tabela **TB_CLIENTE**. Caso esse campo seja **NULL**, o código tenta exibir o campo **ESTADO**. Caso este último também seja **NULL**, será retornado **NC**:

```
SELECT
    CODCLI, NOME, COALESCE(EST_COB, ESTADO, 'NC') AS EST_COBRANCA
FROM TB_CLIENTE
ORDER BY 3;
```

1.11.UNION

A cláusula **UNION** combina resultados de duas ou mais queries em um conjunto de resultados simples, incluindo todas as linhas de todas as queries combinadas. Ela é utilizada quando é preciso recuperar todos os dados de duas tabelas, sem fazer associação entre elas.

Para utilizar **UNION**, é necessário que o número e a ordem das colunas nas queries sejam iguais, bem como os tipos de dados sejam compatíveis. Se os tipos de dados forem diferentes em precisão, escala ou extensão, as regras para determinar o resultado serão as mesmas das expressões de combinação.

O operador **UNION**, por padrão, elimina linhas duplicadas do conjunto de resultados.

Veja o seguinte exemplo:

```
SELECT NOME, FONE1 FROM TB_CLIENTE
UNION
SELECT NOME, FONE1 FROM TB_CLIENTE ORDER BY NOME;
```

1.11.1.Utilizando UNION ALL

A **UNION ALL** é a cláusula responsável por unir informações obtidas a partir de diversos comandos **SELECT**. Para obter esses dados, não há necessidade de que as tabelas que os possuem estejam relacionadas.

Para utilizar a cláusula **UNION ALL**, é necessário considerar as seguintes regras:

- O nome (alias) das colunas, quando realmente necessário, deve ser incluído no primeiro **SELECT**;
- A inclusão de **WHERE** pode ser feita em qualquer comando **SELECT**;
- É possível escrever qualquer **SELECT** com **JOIN** ou subquery, caso seja necessário;
- É necessário que todos os comandos **SELECT** utilizados apresentem o mesmo número de colunas;
- É necessário que todas as colunas dos comandos **SELECT** tenham os mesmos tipos de dados em sequência. Por exemplo, uma vez que a segunda coluna do primeiro **SELECT** baseia-se no tipo de dado decimal, é preciso que as segundas colunas dos outros **SELECT** também apresentem um tipo de dado decimal;
- Para que tenhamos dados ordenados, o último **SELECT** deve ter uma cláusula **ORDER BY** adicionada em seu final;
- Devemos utilizar a cláusula **UNION** sem **ALL** para a exibição única de dados repetidos em mais de uma tabela.

Enquanto **UNION**, por padrão, elimina linhas duplicadas do conjunto de resultados, **UNION ALL** inclui todas as linhas nos resultados e não remove as linhas duplicadas. A seguir, veja um exemplo da utilização de **UNION ALL**:

```
SELECT NOME, FONE1 FROM TB_CLIENTE  
UNION ALL  
SELECT NOME, FONE1 FROM TB_CLIENTE ORDER BY NOME;
```

1.12.EXCEPT e INTERSECT

Os resultados de duas instruções **SELECT** podem ser comparados por meio dos operadores **EXCEPT** e **INTERSECT**, resultando, assim, em novos valores.

O operador **INTERSECT** retorna os valores encontrados nas duas consultas, tanto a que está à esquerda quanto a que está à direita do operador na sintaxe a seguir:

```
<instrução_select_1>  
INTERSECT  
<instrução_select_2>
```

O operador **EXCEPT** retorna os valores da consulta à esquerda que não se encontram também na consulta à direita:

```
<instrução_select_1>  
EXCEPT  
<instrução_select_2>
```

Os operadores **EXCEPT** e **INTERSECT** podem ser utilizados juntamente com outros operadores em uma expressão. Neste caso, a avaliação da expressão segue uma ordem específica: em primeiro lugar, as expressões em parênteses; em seguida, o operador **INTERSECT**; e, por fim, o operador **EXCEPT**, avaliado da esquerda para a direita, de acordo com sua posição na expressão.

Também podemos utilizar **EXCEPT** e **INTERSECT** para comparar mais de duas queries. Quando for assim, a conversão dos tipos de dados é feita pela comparação de duas consultas ao mesmo tempo, de acordo com a ordem de avaliação que apresentamos.

Para utilizar **EXCEPT** e **INTERSECT**, é necessário que as colunas estejam em mesmo número e ordem em todas as consultas, e que os tipos de dados sejam compatíveis.

Primeiramente, vamos escolher o banco de dados a ser utilizado:

```
USE PEDIDOS;
```

A seguir, temos exemplos que demonstram a utilização dos operadores **EXCEPT** e **INTERSECT**:

- **Exemplo 1**

A instrução a seguir lista os códigos de departamento que possuem funcionários que ganham mais de R\$ 5.000,00:

```
SELECT COD_DEPTO FROM TB_DEPARTAMENTO  
INTERSECT  
SELECT COD_DEPTO FROM TB_EMPREGADO  
WHERE SALARIO > 5000
```

O resultado do código anterior é exibido a seguir:

	COD_DEPTO
1	1
2	2



- **Exemplo 2**

O exemplo a seguir lista os departamentos sem um funcionário sequer cadastrado:

```
SELECT COD_DEPTO FROM TB_DEPARTAMENTO  
EXCEPT  
SELECT COD_DEPTO FROM TB_EMPREGADO
```

O resultado do código anterior é exibido a seguir:

	COD_DEPTO
1	10
2	13

- **Exemplo 3**

O exemplo a seguir lista os cargos que não possuem um funcionário sequer cadastrado:

```
SELECT COD_CARGO FROM TB_CARGO  
EXCEPT  
SELECT COD_CARGO FROM TB_EMPREGADO
```

O resultado do código anterior é exibido a seguir:

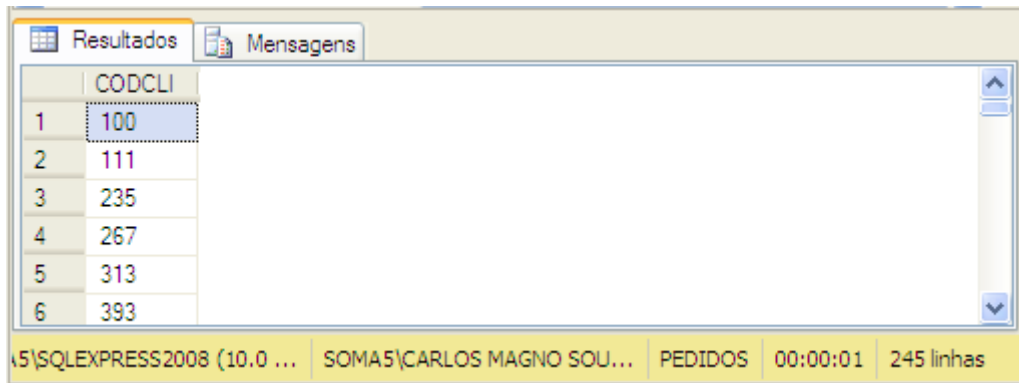
	COD_CARGO
1	7
2	13
3	15

- **Exemplo 4**

O código a seguir consulta os clientes que compraram em janeiro de 2014:

```
SELECT CODCLI FROM TB_CLIENTE  
INTERSECT  
SELECT CODCLI FROM TB_PEDIDO  
WHERE DATA_EMISSAO BETWEEN '2014.1.1' AND '2014.1.31'
```


O resultado do código anterior é exibido a seguir:



A screenshot of the SQL Server Enterprise Manager interface. The 'Resultados' (Results) tab is active, displaying a table with two columns: an implicit row number and 'CODCLI'. The table contains six rows of data. The status bar at the bottom indicates the connection is to '5\SQLEXPRESS2008 (10.0 ...)', the user is 'SOMA5\CARLOS MAGNO SOU...', the database is 'PEDIDOS', the execution time is '00:00:01', and there are '245 linhas' (245 lines).

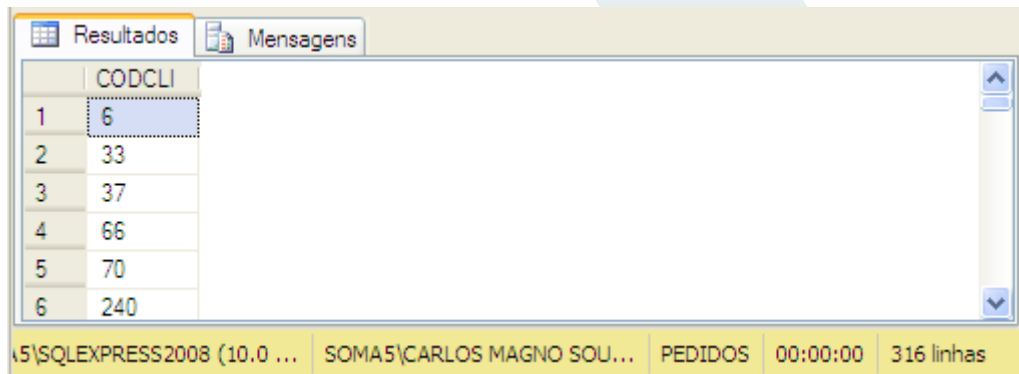
	CODCLI
1	100
2	111
3	235
4	267
5	313
6	393

- **Exemplo 5**

O código a seguir consulta os clientes que não compraram em janeiro de 2014:

```
SELECT CODCLI FROM TB_CLIENTE  
EXCEPT  
SELECT CODCLI FROM TB_PEDIDO  
WHERE DATA_EMISSAO BETWEEN '2014.1.1' AND '2014.1.31'
```

O resultado do código anterior é exibido a seguir:



A screenshot of the SQL Server Enterprise Manager interface. The 'Resultados' (Results) tab is active, displaying a table with two columns: an implicit row number and 'CODCLI'. The table contains six rows of data. The status bar at the bottom indicates the connection is to '5\SQLEXPRESS2008 (10.0 ...)', the user is 'SOMA5\CARLOS MAGNO SOU...', the database is 'PEDIDOS', the execution time is '00:00:00', and there are '316 linhas' (316 lines).

	CODCLI
1	6
2	33
3	37
4	66
5	70
6	240

Pontos principais

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

- Na linguagem SQL, o principal comando utilizado para a realização de consultas é o **SELECT**. Pertencente à categoria **DML** (Data Manipulation Language), esse comando é utilizado para consultar todos os dados de uma fonte de dados ou apenas uma parte específica deles;
- Às vezes, é necessário que o resultado da consulta de dados seja fornecido em uma ordem específica, de acordo com um determinado critério. Para isso, contamos com opções e cláusulas. Uma delas é a cláusula **ORDER BY**, que considera certa ordem para retornar dados de consulta;
- A cláusula **WHERE** é utilizada para definir critérios com o objetivo de filtrar o resultado de uma consulta. As condições definidas nessa cláusula podem ter diferentes propósitos, tais como a comparação de dados na fonte de dados, a verificação de dados de determinadas colunas e o teste de colunas nulas ou valores nulos;
- **UNION** permite a união de duas ou mais consultas em uma única;
- Uma forma de podermos comparar consultas é a utilização dos comandos **INTERSECT** e **EXCEPT**.