**Banco de dados SQL SERVER**

**O que é Banco de dados?**

O SQL Server é um gerenciador de bancos de dados relacional, o que significa dizer que as informações que manipula estão armazenadas em campos de tabelas. Para facilitar o entendimento, imagine, por exemplo, uma tabela de clientes, em que cada linha contém os registros de nome, endereço e estado civil.

**Principais tipos de dados**

**Baseados em caracteres**

|  |  |
| --- | --- |
| Tipo de dados | Descrição |
| Char | Armazena N caracteres fixos (até 8.000) no formato não Unicode. Independentemente da quantidade de caracteres utilizados, irá sempre armazenar o tamanho de caracteres do campo, sendo preenchido o restante com espaços em branco |
| Varchar | Armazena N caracteres (até 8.000) no formato não Unicode |
| Text | Armazena caracteres no formato não Unicode. Esse tipo de dado suporte até 2.147.483.647 caracteres e existem funções específicas para trabalhar com esse tipo de dado |

**Baseados em numéricos inteiros**

|  |  |
| --- | --- |
| Tipo de dados | Descrição |
| Smallint | Armazena valores numéricos inteiros, variando de -32.768 a 32.767 |
| Int | Armazena valores numéricos inteiros, variando de -2.147.483.648 a 2.147.483.647 |
| Bigint | Armazena valores numéricos inteiros, variando de -9.223.372.036.854.775.808 a -9.223.372.036.854.775.807 |
| Int | Armazena valores numéricos inteiros, variando de -2.147.483.648 a 2.147.483.647 |
| Money | Valores numéricos decimais variando de -922,337,203,685,477.5808 a +922,337,203,685,477.5807 |
| Numeric (N, N) | Armazena valores numéricos com casas decimais, utilizando precisão. O primeiro número entre os parênteses, representa a quantidade de inteiros a serem armazenados, o segundo número, indica a quantidade de casas decimais do número. |
| Real | Armazena valores numéricos aproximados com precisão de ponto flutuante, variando de -3.40E + 38 a 3.40E + 38 |

**Baseados em data**

|  |  |
| --- | --- |
| Tipo de dados | Descrição |
| Smalldatetime | Armazena data e hora, com precisão de minutos. |
| Datetime | Armazena data e hora, com precisão de centésimos de segundos. |
| Time | Armazena somente hora. Pode armazenas segundos até a fração de 99999999. |
| Date | Armazena somente data |

**Cláusulas SQL**

Como uma frase, uma instrução SQL tem cláusulas. Cada cláusula executa uma função para a instrução SQL. Algumas cláusulas são necessárias em uma instrução SELECT. A tabela a seguir lista as cláusulas SQL mais comuns.

|  |  |
| --- | --- |
| Cláusula | O que ela faz |
| SELECT | Lista os campos de uma tabela que contêm dados de interesse |
| FROM | Lista as tabelas que contêm os campos listados na cláusula SELECT |
| WHERE | Especifica critérios de campo que devem ser atendidos por cada registro a ser incluído nos resultados. |
| Order by | Especifica como classificar os resultados. |
| Group by | Em uma SQL que contém funções agregadas, lista campos que não são resumidos na cláusula SELECT. |
| HAVING | Em uma SQL que contém funções agregadas, especifica condições que se aplicam aos campos resumidos na instrução SELECT. |

**Organização da SQL**

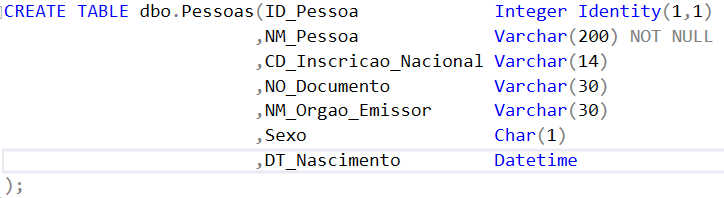
**DDL - Linguagem de Definição de Dados - Define os comandos utilizados para criação (CREATE) de tabelas, views, índices, atualização dessas estruturas (ALTER), assim como a remoção (DROP);**

**CREATE**

O comando CREATE é usado sempre que precisamos criar novos objetos na base de dados. Primeiramente, essa instrução pode ser utilizada para criar um banco completamente vazio, por exemplo:



Já utilizando o comando CREATE TABLE, podemos criar novas tabelas dentro de um banco de dados. Observe:



Repare que, ao criar a tabela, também especificamos quais colunas ela terá e os tipos de dados que cada coluna pode receber. Perceba ainda que, nas colunas de tipo VARCHAR, também foi especificado a quantidade máxima de caracteres que cada campo vai comportar. Na coluna ID\_Pessoa além do tipo Integer foi adicionado a propriedade Identity, ou seja, quando formos inserir uma pessoa, não será necessário colocar manualmente o seu ID\_Pessoa de 1 em 1, pois o Identity é do tipo Autoincrement (serve para que um valor de um campo seja automaticamente inserido toda a vez que se cria um novo registro).

**ALTER**

Por sua vez, o comando ALTER é utilizado quando precisamos alterar a estrutura de um objeto que já existe na base de dados. Um exemplo da aplicação dessa instrução é:



Também é possível alterar o tipo da coluna, exemplo foi criado a coluna sexo do tipo char(1), porém precisamos alterar para que seja possível inserir o tipo de sexo por extenso, para isso é necessário utilizar o ALTER COLUMN.



**DROP**

O comando DROP é o responsável por deletar objetos da nossa base de dados. Utilizamos a sintaxe abaixo para realizar essa ação:



É interessante ressaltar que ele também pode ser usado para excluir o banco de dados completo. Para isso, inserimos a instrução dessa forma:



**CONSTRAINTS**

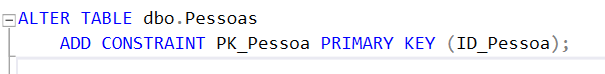
Constraints são objetos usados com a finalidade de estabelecer regras referentes à integridade e à consistência nas colunas das tabelas pertencentes a um sistema de banco de dados.

**Tipos de constraints mais utilizadas**

* **Primary key**

Chave primaria (primary key) ao definirmos o campo ID. Isso indica que essa coluna é uma chave-primária, ou seja, o seu conteúdo é único e representa a identificação do registro.

Observe que na imagem abaixo foi criado uma constraint com o nome “PK\_Pessoa” em que define a coluna ID\_Pessoa como uma chave primaria da tabela.



* **Foreign key**

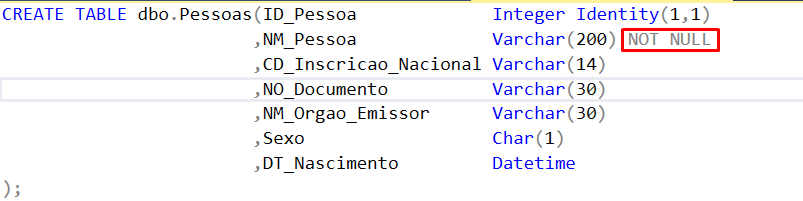
Chave estrangeira (foreign key) é o campo que estabelece o relacionamento entre duas ou mais tabelas. Assim, uma coluna corresponde à mesma coluna que é a chave primária de outra tabela. Dessa forma, deve-se especificar na tabela que contém a chave estrangeira quais são essas colunas e à qual tabela está relacionada.

Observe que na imagem abaixo foi criado uma constraint com o nome “FK\_Pessoa\_Cliente” em que define que a coluna ID\_Pessoa da tabela Clientes é uma chave estrangeria da tabela Pessoas.

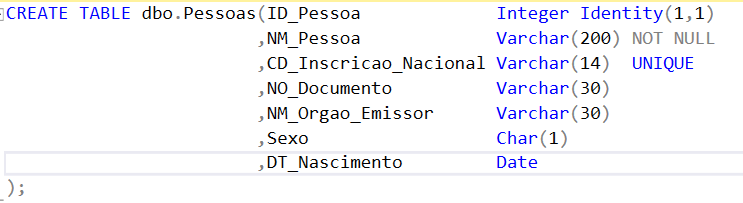


* **Not null**

É utilizado quando não queremos que determinada coluna aceite a inserção de valores nulo, para isso é utilizado o “NOT NULL”



* **Unique**

A restrição UNIQUE garante que todos os valores em uma coluna são diferentes. As restrições UNIQUE e PRIMARY KEY fornecem uma garantia de exclusividade para uma coluna ou conjunto de colunas. Uma restrição PRIMARY KEY automaticamente tem uma restrição UNIQUE.

**DML - Linguagem de Manipulação de Dados - Define os comandos utilizados para manipulação de dados no banco (INSERT, UPDATE e DELETE);**

**INSERT**

O comando INSERT é usado para inserir dados em uma tabela do banco. Para realizar essa ação, podemos usar a seguinte sintaxe:



**UPDATE**

O comando UPDATE é usado para alterar dados em uma tabela do banco. Para realizar essa ação, podemos usar a seguinte sintaxe:



Na qual está alterando apenas um registro, ou seja, onde o ID\_Pessoa for igual a 1.

Caso esse comando for utilizando sem cláusula WHERE, irá alterar todos os nomes de todas as pessoas cadastradas na tabela, exemplo de sintaxe a não ser seguida:



**DELETE**

Por sua vez, o comando DELETE é utilizado para excluir informações da nossa base de dados. A sintaxe para realizar essa ação é bastante simples:



Na qual está excluindo apenas um registro, ou seja, onde o ID\_Cliente for igual a 5.

Caso esse comando for utilizando sem cláusula WHERE, irá excluir todos os nomes de todos os dados dos clientes cadastrados na tabela, exemplo de sintaxe a não ser seguida:



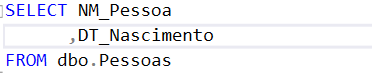
**DQL - Linguagem de Consulta de Dados - Define o comando utilizado para que possamos consultar (SELECT) os dados armazenados no banco;**

**SELECT**

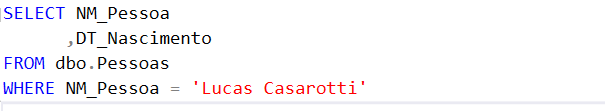
É um dos comandos SQL mais importantes, pois com ele podemos elaborar diversas consultas aos registros da nossa base de dados. É possível, por exemplo, fazer uma pesquisa que retornará todos os campos de uma tabela dessa forma:



Perceba que, nessa consulta, o caractere \* é usado para indicar que os dados de todas as colunas da tabela “Pessoas” devem ser retornados na pesquisa. Por isso, para fazer uma consulta mais específica, os campos desejados devem ser informados da seguinte maneira:



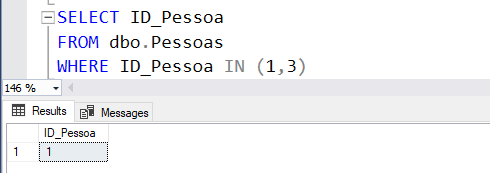
Além disso, utilizando a cláusula WHERE, podemos fazer uma consulta ainda mais refinada. Veja:



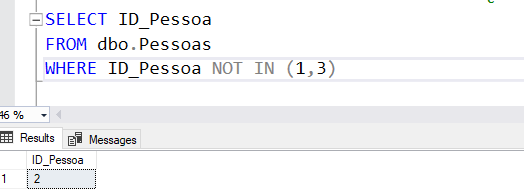
**SELECT IN E NOT IN**

IN é um operador para especificar vários valores em uma cláusula WHERE. Com ele podemos verificar se determinada coluna está sendo mencionada em um determinado grupo de valores.

Ou seja, na imagem abaixo estamos buscando os registros que estão dentro dos parênteses (1,3).

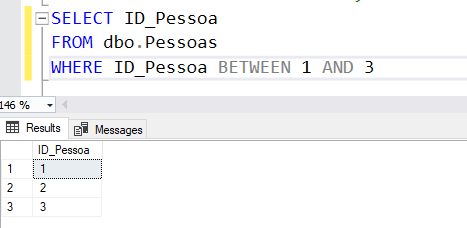


Já o NOT IN, irá buscar todos os registros que estão fora dos parênteses.



**SELECT BETWEEN**

Também é utilizado para selecionar dados em determinado range, funciona da seguinte maneira, BETWEEN tem o ponto inicial e o ponto final, resumidamente ele vai trazer ambos os pontos e tudo que tem entre eles.



**SELECT LIKE E NOT LIKE**

O operador like é utilizado para buscar por uma determinada string dentro de aspas simples.

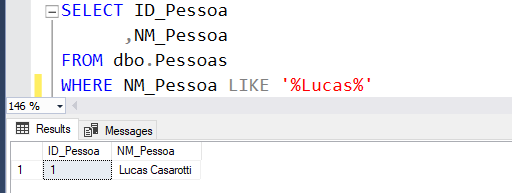
Na imagem abaixo está sendo realizado uma busca na(s) coluna(s) NM\_Pessoa em que tenha o nome “Lucas”, no começo, meio ou fim.

Também é possível realizar uma busca apenas no início ou fim, da seguinte maneira;

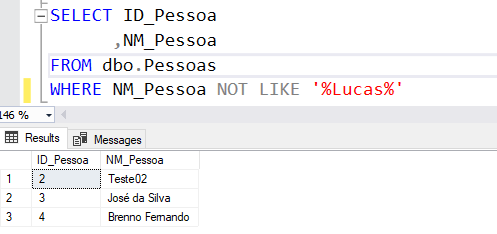
LIKE 'A%' = VAI BUSCAR TODOS OS NOMES COM A LETRA A NO COMEÇO

LIKE '%A%' = VAI BUSCAR TODOS OS NOMES COM A LETRA A

LIKE '%A' = VAI BUSCAR TODOS OS NOMES COM A LETRA A NO FINAL



Já o operador NOT LIKE é utilizado para não buscar o que tenha dentro da determinada string.



Da mesma maneira do LIKE também é possível filtrar no inicio, meio e fim, da seguinte maneira;

NOT LIKE 'A%' = NÃO VAI BUSCAR OS NOMES COM A LETRA A NO COMEÇO

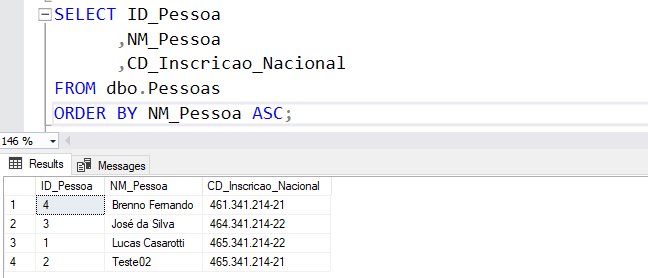
NOT LIKE '%A%' = NÃO VAI BUSCAR OS NOMES COM A LETRA A

NOT LIKE '%A' = NÃO VAI BUSCAR OS NOMES COM A LETRA A NO FINAL

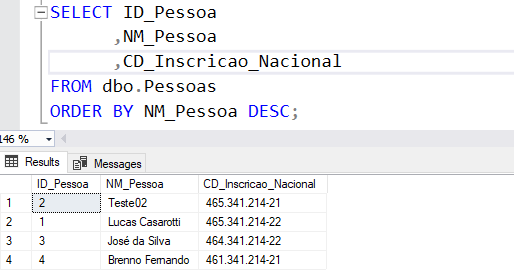
**SELECT ORDER BY**

É utilizado a cláusula ORDER BY para que os dados sejam exibidos em ordem crescente ou decrescente.

ASC para crescente

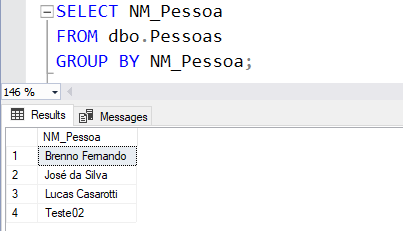


DESC para decrescente

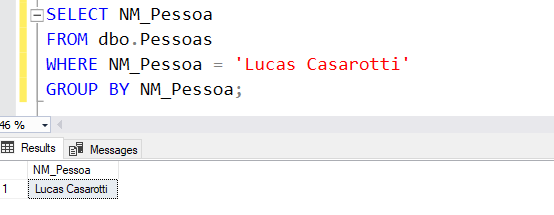


**SELECT GROUP BY**

A cláusula GROUP BY agrupa linhas baseado em semelhanças entre elas. Por exemplo, se em uma tabela você tiver o mesmo nome duas vezes ele irá agrupar em uma única linha.

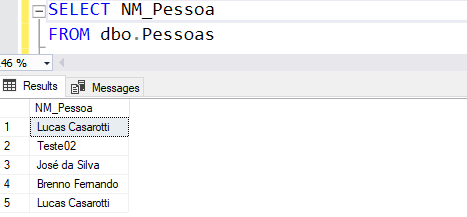


Observe que tabela cliente tem dois registros com o mesmo nome se utilizarmos o group by esses registros serão agrupados em uma única linha.

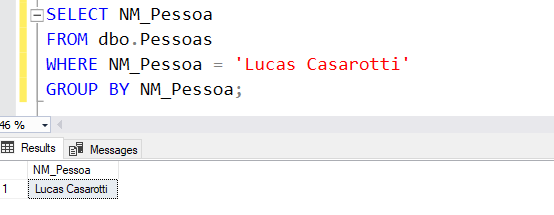


**SELECT HAVING**

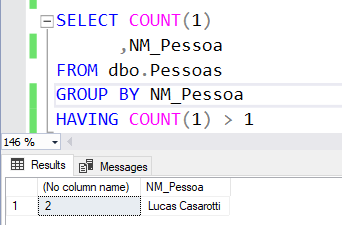
A cláusula HAVING é utilizada para especificar condições de filtragem em grupos de registros. É muito utilizado em conjunto com a cláusula GROUP BY para filtrar as colunas agrupadas. Resumidamente tem a mesma função do WHERE, porém não é possivel utilizar a cláusula WHERE em conjunto com a função COUNT, por esse motivo utilizamos o HAVING.



Por exemplo, na tabela pessoas temos nomes diferentes, e somente o nome “Lucas Casarotti” se repete, se utilizarmos a cláusula GROUP BY será exibidos o nome “Lucas Casarotti” em apenas uma linha.

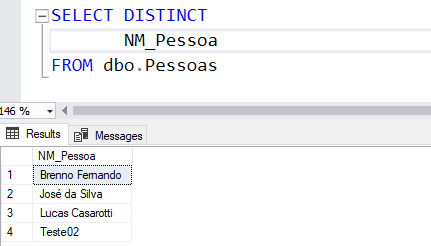


Caso sejá necessário filtrar a linha que tenha mais registros é necessário utilizar o HAVING.

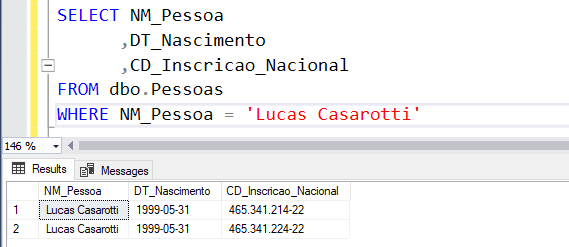


**SELECT DISTINCT**

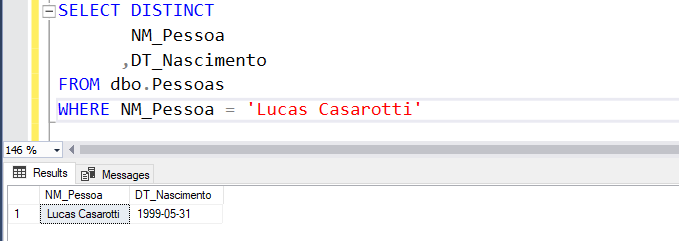
É utilizado para que não mostre resultados repetidos, porém estamos utilizando o distinct em apenas uma coluna, é possível utilizar em mais de uma coluna, mas para que seja feito a distinção ambas colunas devem ser iguais, caso contrarío mostrará os demais registros mesmo que em uma coluna os nomes sejam iguais.



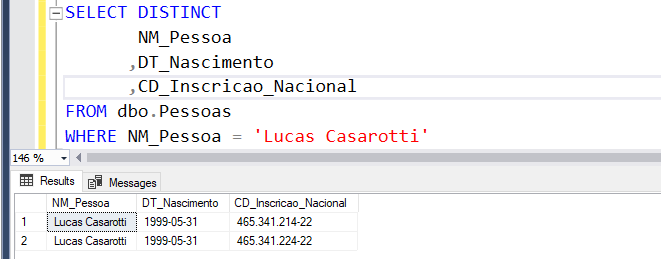
Exemplo, vamos utilizar mais de uma coluna no select distinct, para isso temos os seguintes registros.



Na tabela pessoa existe dois registros com o mesmo nome, porém a coluna CD\_Inscricao\_Nacional é diferente, é possível utilizar o distinct da seguinte maneira.



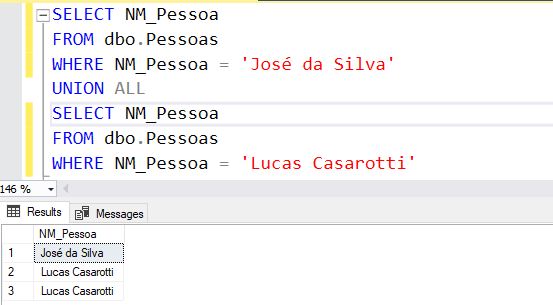
Como o registros das colunas NM\_Pessoa, DT\_Nascimento são iguais é possível utilizar o distinct para mostrar apenas um registro, caso queiramos acrescentar a coluna CD\_Inscricao\_Nacional o resultado será o seguinte.



**SELECT UNION E UNION ALL**

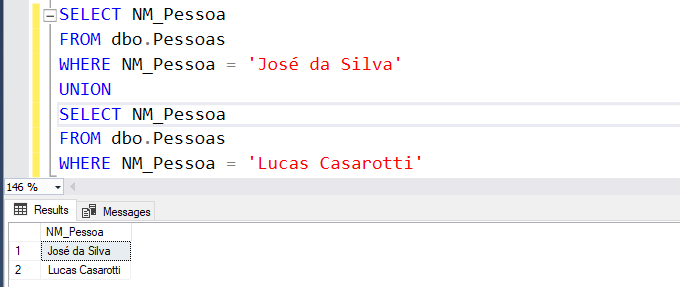
São utilizado para combinar consultas diferentes, funciona da seguinte maneira.

UNION ALL



Caso no primeiro select não exista nenhuma pessoa chamada “José da Silva” mostrará apenas o resultado do seguinte select ou vice-versa.

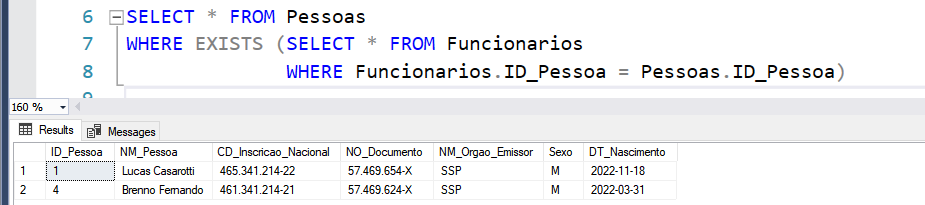
UNION



A diferença do UNION para o UNION ALL é que ele realiza a distinção dos resultado assim como o SELECT DISTINCT.

**SELECT EXISTS**

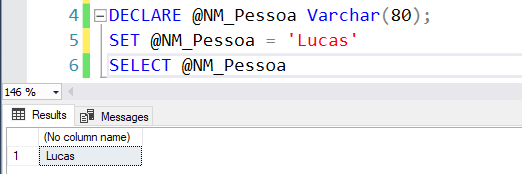
Será executado se o resultado da subquery for true.



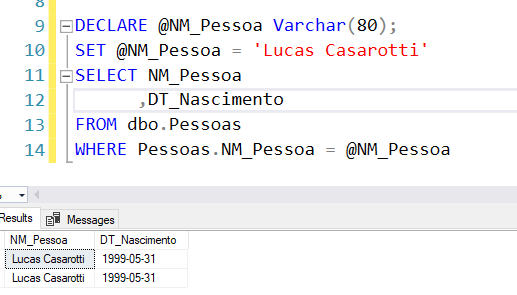
Ou seja, se existir algum funcionário que tenha o ID\_Pessoa

**Variáveis**

Assim como na programação, no SQL também é possível criar e utilizar variáveis, podemos utiliza-las para receber um valor e depois atribuir em uma determinada coluna de uma tabela, entre outras coisas**.**



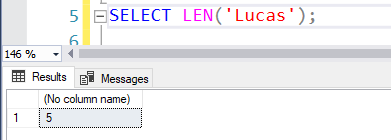
Podemos também as utilizar para filtrar determinado valor em uma tabela.



**Funções de texto**

* **Função LEN**

É utilizada para retornar o tamanho do campo.



* **Função CHARINDEX**

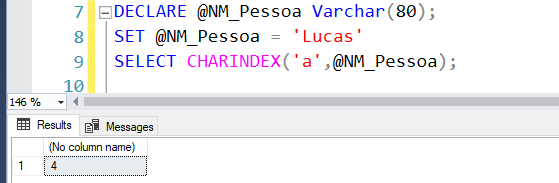
É utilizada para buscar um caractere dentro de uma sequência e mostrar em qual posição está.

Exemplo:

Criamos uma variável que recebe o nome ‘Lucas’, se utilizarmos a função LEN na variável será retornado o número 5, porque o nome Lucas tem 5 posições.

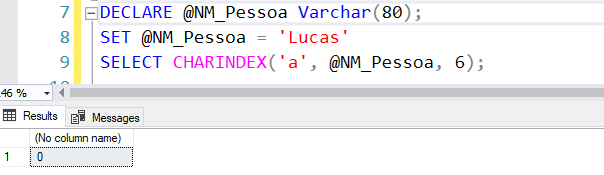
Se utilizarmos a função CHARINDEX para buscar a letra ‘a’ na variável será retornado o número 4, pois a letra ‘a’ está na posição 4 no nome ‘Lucas’.

Buscando a letra ‘a’ em qualquer posição da variável.



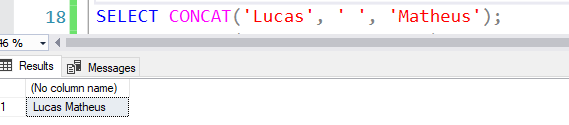
Também podemos escolher a partir da posição que queremos buscar.

Buscando a letra ‘a’ a partir da posição 6.



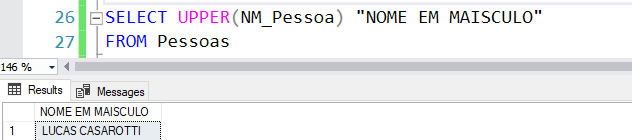
* **Função CONCAT**

É utilizada para juntar um ou mais valores.



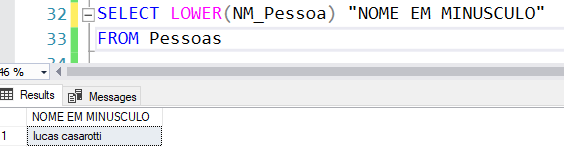
* **Função UPPER**

É utilizada para deixar as letras maiúsculas.



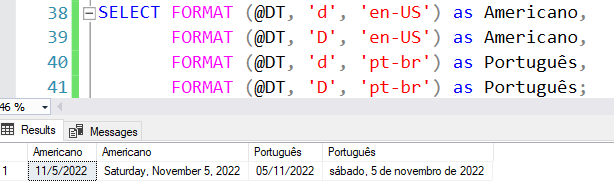
* **Função LOWER**

É utilizada para deixar as letras minúsculas.



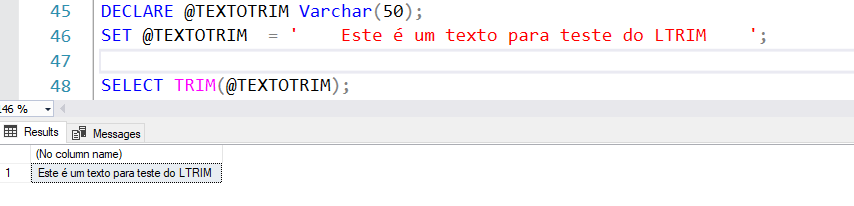
* **Função FORMAT**

É utilizada para formatar o valor especifico no formato fornecido.



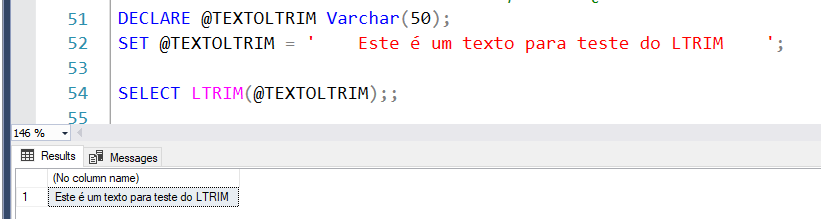
* **Função TRIM**

É utilizada para os espaços a esquerda e direita.



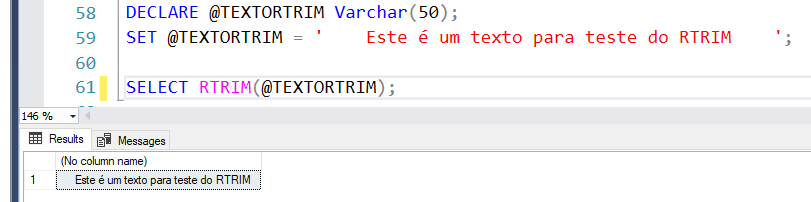
* **Função LTRIM**

É utilizada para retirar os espaços a esquerda.



* **Função RTRIM**

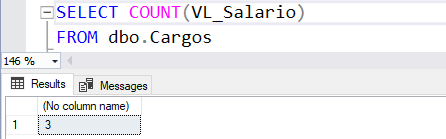
É utilizada para retirar os espaços a direita.



**Funções de agregação**

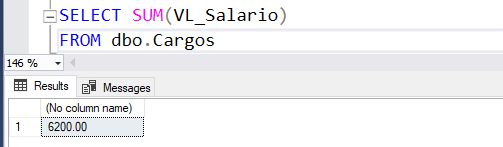
* **Função COUNT**

É utilizada para somar a quantidade de registros de uma coluna.



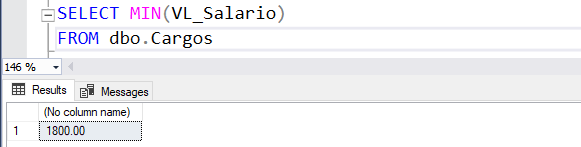
* **Função SUM**

É utilizada para somar o valor total da coluna.



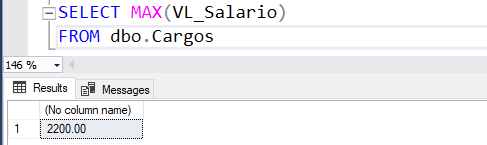
* **Função MIN**

É utilizada para buscar o valor mínimo da coluna.



* **Função MAX**

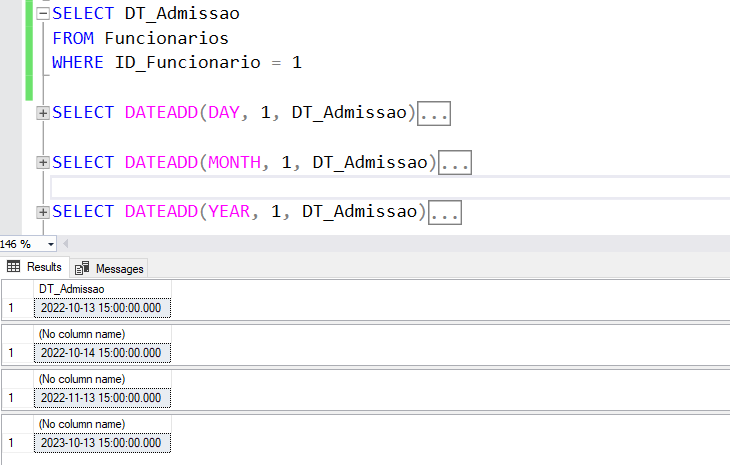
É utilizada para buscar o valor máximo da coluna



**Funções de data**

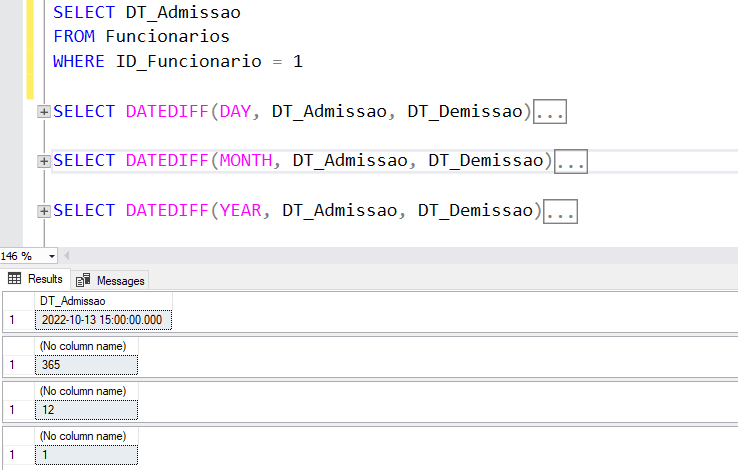
* **Função DATEADD**

É utilizada para adicionar ou subtrair dias, meses ou anos de uma data especifica.

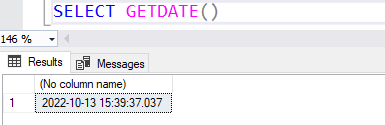


* **Função DATEDIFF**

É utilizada para pegar o intervalo entre dias, meses ou anos entre duas datas.



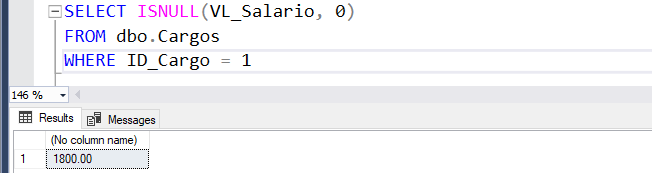
* **Função GETDATE**

É utilizada para pegar a data atual. 

**Outras funções**

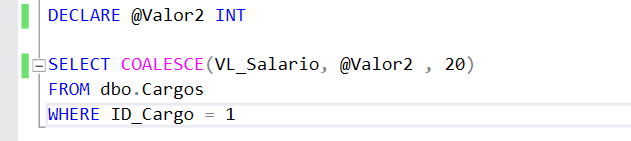
* **Função ISNULL**

É utilizado para substituir o valor nulo por outro, caso o VL\_Salario seja nulo ele vai ser substituído por 0.



* **Função COALESCE**

É parecido com a função ISNULL, porém aceita mais de dois valores.

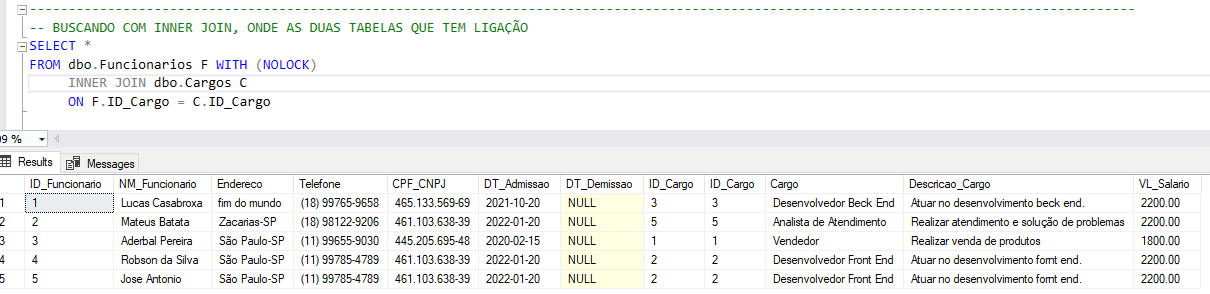


Exemplo,caso o primeiro valor for nulo irá pegar o segundo, caso o segundo valor também for nulo irá pegar o terceiro e assim sucessivamente.

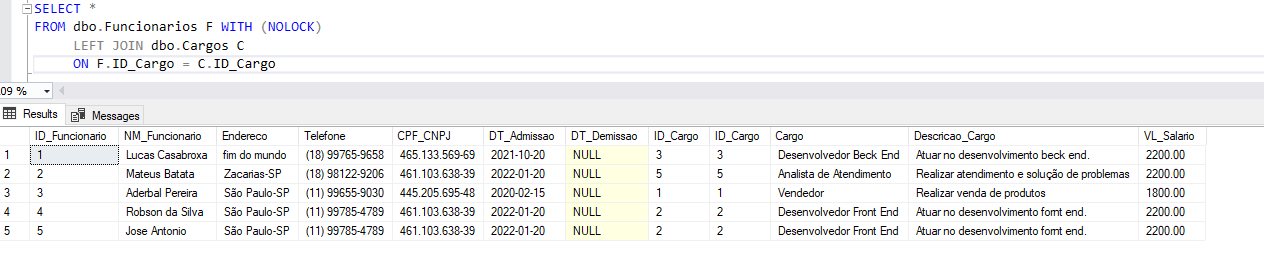
Nesteexemplo utilizamos variáveis que será explicaremos logo.

**Cláusulas INNER JOIN, LEFT JOIN e RIGHT JOIN**

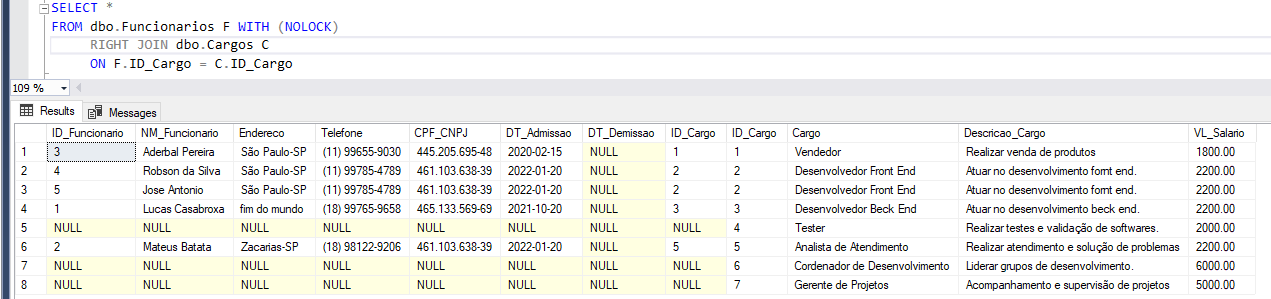
A cláusula INNER JOIN permite usar um operador de comparação para comparar os valores de colunas provenientes de tabelas associadas. Por meio desta cláusula, os registros de duas tabelas são usados para que sejam gerados os dados relacionados de ambas. Usamos as cláusulas WHERE e FROM para especificar esse tipo de associação.



A cláusula LEFT JOIN ou LEFT OUTER JOIN permite obter não apenas os dados relacionados de duas tabelas, mas também os dados não relacionados encontrados na tabela à esquerda da cláusula JOIN. Caso não existam dados relacionados entre as tabelas à esquerda e a direita do JOIN, os valores resultantes de todas as colunas da lista de seleção da tabela à direita serão nulos.



Ao contrário do LEFT JOIN, a cláusula RIGHT JOIN ou RIGHT OUTER JOIN retorna todos os dados encontrados na tabela à direita de JOIN. Caso não existam dados associados entre as tabelas à esquerda e à direita de JOIN, serão retornados valores nulos.



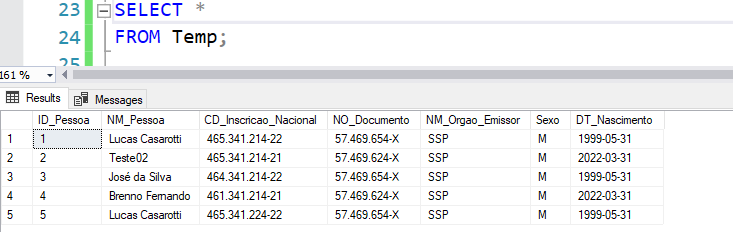
**BEGIN TRANSACTION**

É uma zona segura, nela podemos fazer alterações, atualizações e deletes, tudo o que quiser dentro de uma tabela sem que essa tabela sofra alterações indesejadas, o que vai determinar se essas alterações serão realizadas vai ser o comando “COMMIT” e para desfazer as alterações será o “ROLLBACK”.

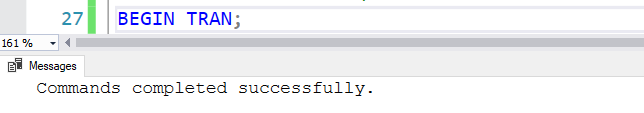
Exemplo:

Primeiro iremos criar uma tabela temporária a partir dos registros da tabela pessoas.





Para iniciar uma transação é necessário executar o comando “BEGIN TRAN” ou “BEGIN TRANSACTION”.

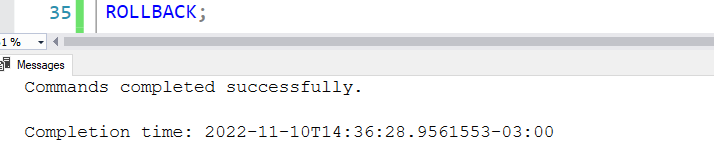


Agora iremos executar um update para alterar todos os registros da tabela temporária.

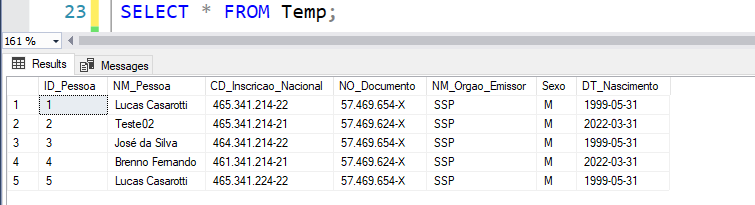




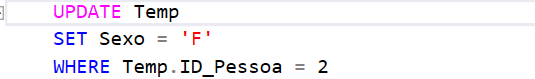
Caso tenhamos feito as alterações de forma incorreta, podemos utilizar o comando “ROLLBACK” para desfazer todas as alterações.

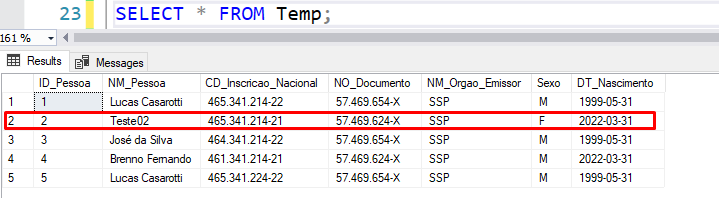


Se buscarmos os registros da tabela temporária, podemos ver que as alterações foram desfeitas.

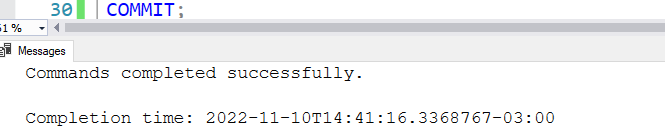


Agora iremos realizar o update de forma “correta” alterando apenas uma pessoa.





Caso as alterações estiverem corretas, para salvar na tabela basta executar o comando “COMMIT”.



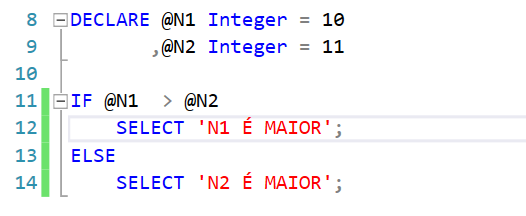
COMMIT: Alteração será realizada na tabela.

ROLLBACK: Alteração não será realizada na tabela.

Resumidamente o “BEGIN TRAN” serve para proteger os dados da tabela sem que sofram alterações indesejadas, podendo reverter essas alterações.

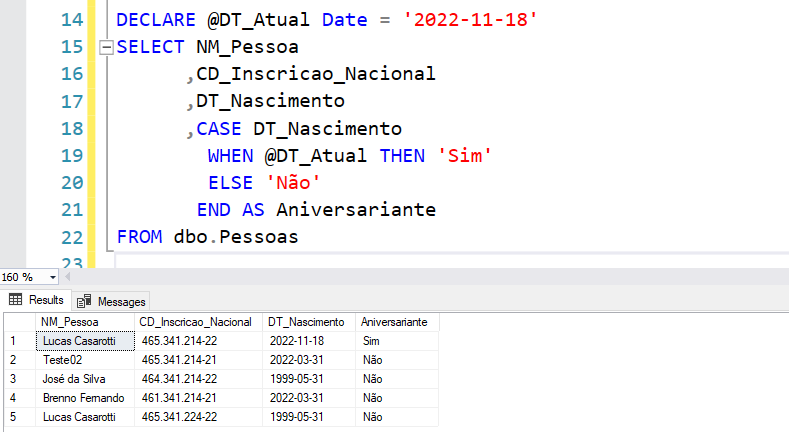
**IF e ELSE**

São utilizados para testar condições quando um comando é executado. O IF e Else funcionam similarmente aos comandos das linguagens de programação, resumidamente é uma estrutura que só vai executar determinado bloco se (IF) existir determinada condição, se não (ELSE) tiver essa condição executara outro bloco.



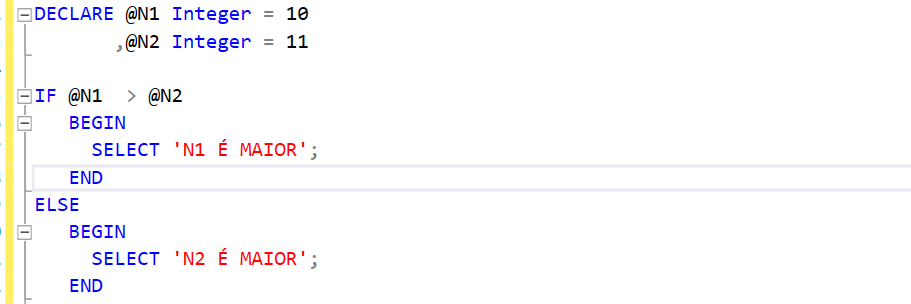
**CASE**

É uma estrutura de condição utilizada em conjunto com o SELECT, nela podemos retornar uma coluna com determinado nome e preenchida de acordo com a condição. Desta forma reduzimos a necessidade de gastar várias linhas de código fora do comando para fazer esse tipo de verificação.



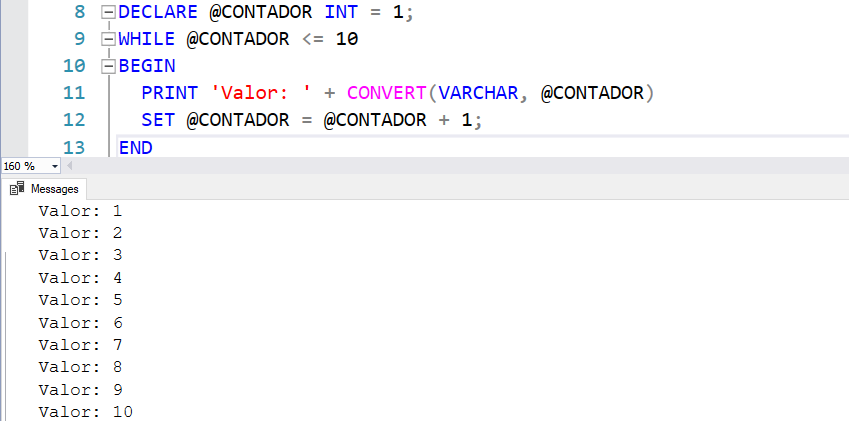
**BEGIN END**

Os elementos BEGIN e END tem o objetivo de iniciar e finalizar um bloco de comandos, de maneira que este possa ser posteriormente executado. Podemos aninhar blocos de comando utilizando estes elementos. Caso seja executado um bloco de comandos logo após a realização de um teste de condição, os elementos BEGIN e END, são usados logo após um comando IF ou WHILE.

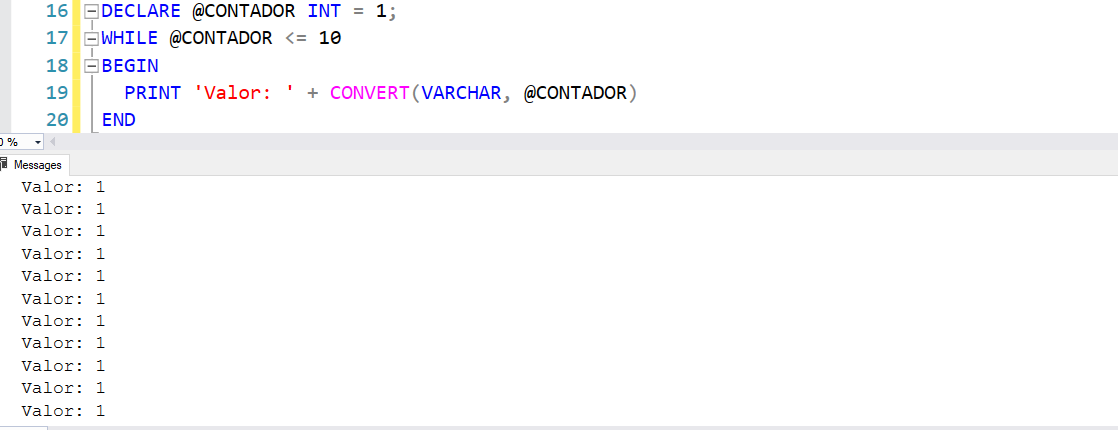


**WHILE**

É uma estrutura de repetição, como o próprio nome já diz ela é utilizada para repetir enquanto determinada condição for verdadeira. Neste exemplo abaixo podemos ver que enquanto a variável contador for menor ou igual a dez, será exibido uma mensagem com o valor da variável contador.



Lembrando que devemos atribuir o valor da variável mais um sempre que entrar no while, desta forma evitamos entrar em um loop infinito.

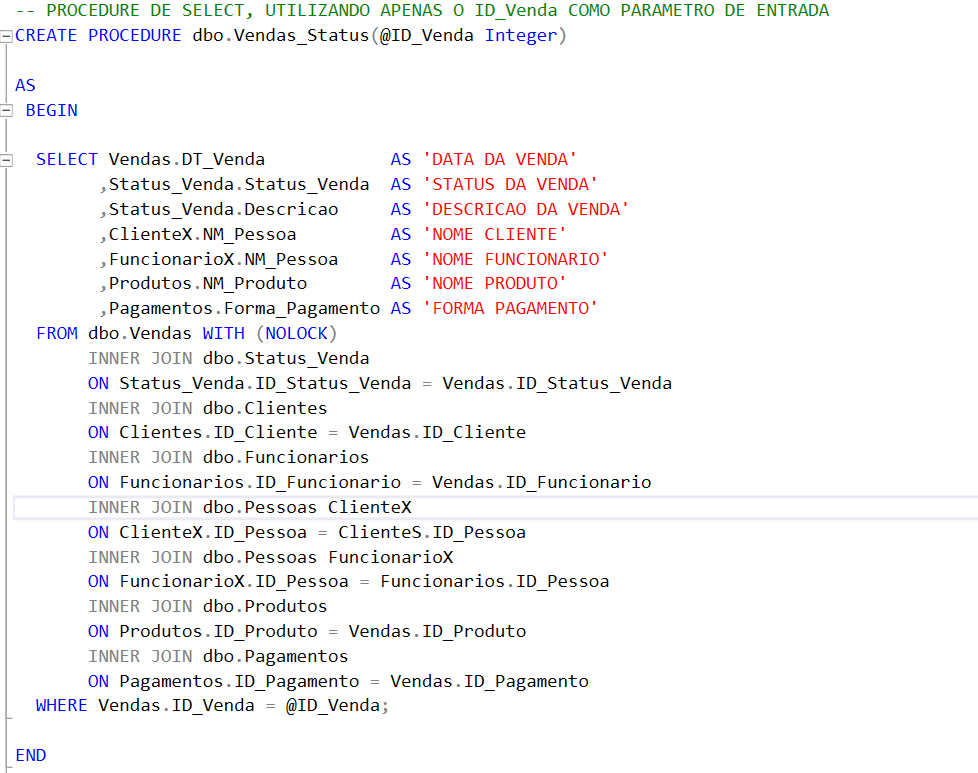


**PROCEDURE**

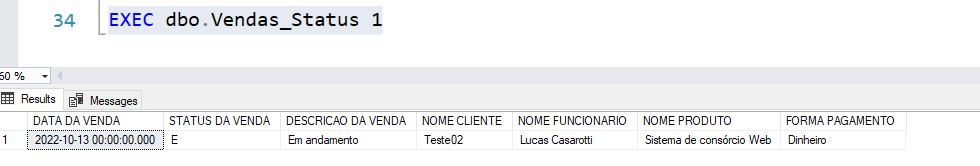
Uma procedure é um bloco de comandos ou instruções SQL organizados para executar uma ou mais tarefas. Ela pode ser utilizada para ser acionada através de uma chamada simples que executa uma série de outros comandos.

Exemplo:

A procedure abaixo é um select com várias tabelas, buscando através do parâmetro de entrada ID\_Venda.



Executando a procedure, passando ID\_Venda = 1 de parâmetro de entrada.



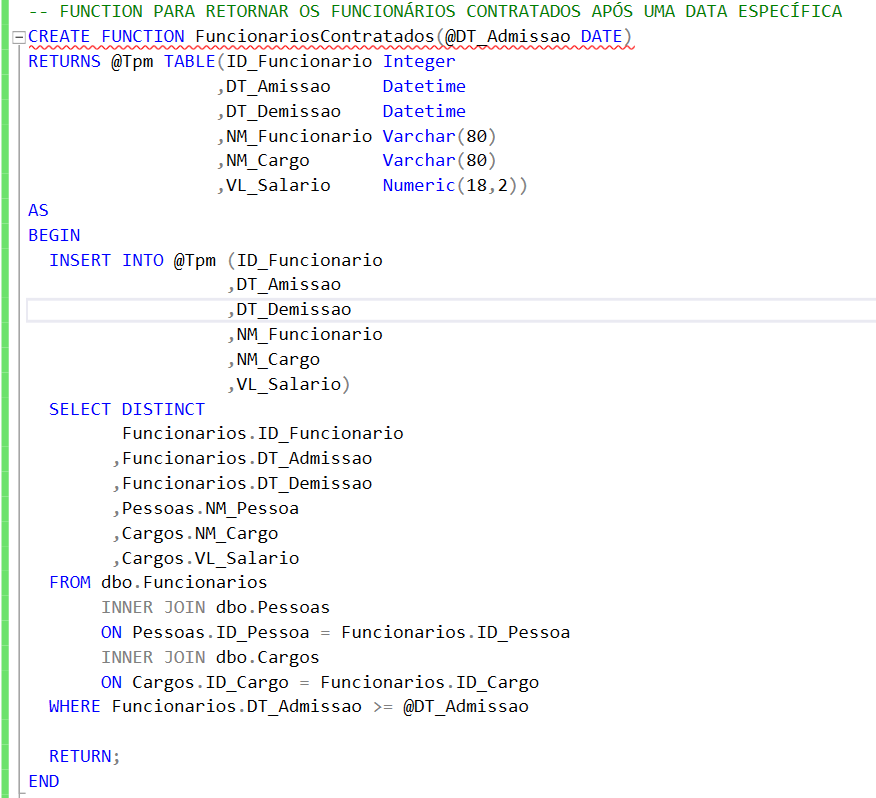
**FUNCTION**

É parecida com a PROCEDURE, a diferença é que utilizamos a procedure para executar comandos mais complexos, já a function é mais utilizada para retorno.

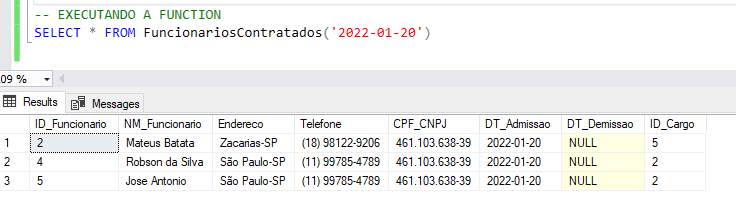
Exemplo:

A function abaixo é utilizada para retornar os funcionários contratados após uma data específica, essa data especifica é um parâmetro de entrada em que o usuário digita.

Exemplo: funcionários contratos na data 2022-01-20 ou superior



Executando a function, passando uma data, como parâmetro de entrada.

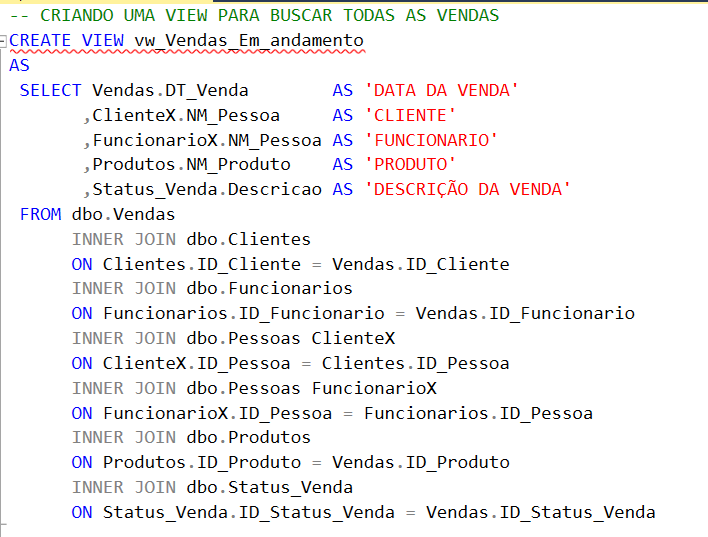
**VIEW**

VIEWS (ou visões em uma tradução livre) pode ser entendida como uma tabela virtual, composta pelo resultado de uma consulta a outras tabelas ou mesmo outras VIEWS.

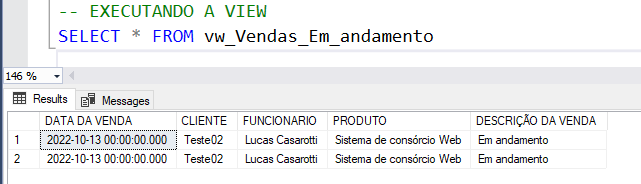
Ao contrário das tabelas normais, uma visão não faz parte fisicamente do banco, existindo apenas dinamicamente a partir das demais tabelas. Alterações dos dados nas tabelas refletem as alterações também na visão.

Podemos utiliza-las para executar um select muito grande, ao invés de ficar digitando toda vez, podemos deixar a view criada e dar um select nela.

Exemplo: A view abaixo é utilizada para retornar as vendas em andamento.



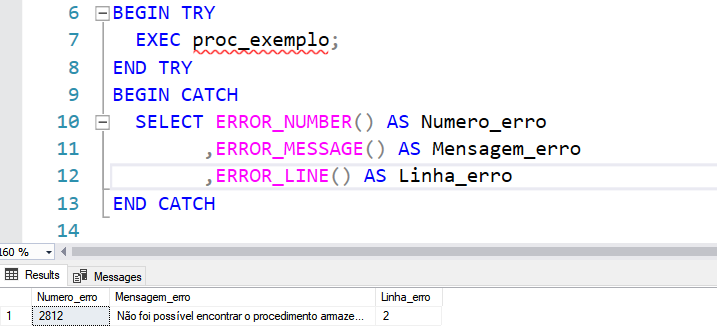
Executando a view



**TRY CATCH**

É utilizado para tratamento de erros, é parecido a estrutura de condição IF e ELSE, porém nesse caso é executado um comando dentro do bloco TRY, caso esse der errado o próximo bloco a ser executado é o CATCH que faz a tratativa do erro.

Neste exemplo abaixo estamos executando a procedure “proc\_exemplo”, caso essa procedure não exista executa o select do catch, que retornará o número do erro, mensagem do erro e a linha do erro.



**Referências**

<https://support.microsoft.com/pt-br/office/acesso-sql-conceitos-b%C3%A1sicos-vocabul%C3%A1rio-e-sintaxe-444d0303-cde1-424e-9a74-e8dc3e460671>

<https://blog.betrybe.com/desenvolvimento-web/comandos-sql/>