Artigo

Trabalhando com a linguagem T-SQL

Este artigo apresenta o uso da linguagem T-SQL. Serão discutidos alguns recursos que podem ser usados no desenvolvimento e manipulação na consulta a dados usando a ferramenta SQL Server.

A linguagem T-SQL

Este artigo tem como objetivo apresentar o uso da **linguagem T-SQL**. Serão discutidos alguns recursos que podem ser utilizados no desenvolvimento de consultas e manipulação de dados usando

alguns **complementos do T-SQL**. A discussão desse tema é útil para qualquer desenvolvedor **Transact-SQL** que queira aprimorar seus conhecimentos ou que tenha interesse em trabalhar com **consultas avançadas em T-SQL**. Entender como o T-SQL funciona pode ajudar a criar consultas melhores e facilitar a sua compreensão de como corrigir uma consulta que não retorna os resultados desejados.

Guia do artigo:

- Sub consultas
- PIVOT e UNPIVOT
- CROSS APPLY e OUTER APPLY
- Rollup, Cube e Groupping Sets
- Índice
- Níveis de isolamento
- Trigger
- View
- Procedure
- USER FUNCTION
- CTE
- Merge

A **linguagem Transact-SQL é uma extensão ao padrão SQL-92**, sendo a linguagem utilizada por desenvolvedores na construção de aplicações que manipulam dados mantidos no SQL Server. Seus comandos podem ser classificados em quatro grupos, de acordo com sua função: DML (Linguagem de Manipulação de Dados), DDL (Linguagem de Definição de Dados), DCL (Linguagem de Controle



conjunto com os demais comandos de manipulação: INSERT, UPDATE e DELETE.

DDL

Esse subconjunto apoia a criação de objetos no banco de dados, alteração na estrutura da base de dados ou a remoção do banco de dados. Seus principais comandos são:

- CREATE;
- ALTER;
- DROP.

DML

Esse subconjunto é utilizado para realizar consultas, inclusões, alterações e exclusões de dados. Seus principais comandos são:

- SELECT ... FROM ... WHERE ...
- INSERT INTO ... VALUES ...
- UPDATE ... SET ... WHERE ...



Esse subconjunto da linguagem SQL é responsável por controlar os aspectos de autorização de dados e a utilização de licenças por usuários. São conhecidos como comandos DCL (Data Control Language):

Relacionado: Introdução ao T-SQL

- GRANT: comando usado para fornecer acesso ou privilégios sobre os objetos de banco de dados para os usuários;
- DENY: nega a permissão a um usuário ou grupo para realizar uma determinada operação em um objeto ou recurso;
- REVOKE: remove a permissão GRANT ou DENY.

DTL

Esse subconjunto da linguagem SQL é responsável por gerenciar transações executadas no banco. Seus principais comandos são:

- COMMIT: efetiva e envia os dados de forma permanente para o banco de dados;
- ROLLBACK: retorna o banco ao estado anterior, desfazendo as alterações feitas na transação.

Integridade de dados

Outro conceito essencial e presente nos SGBDs é a integridade de dados. Ele diz respeito a uma série de restrições impostas pelos SGBDs para garantir que os dados nunca saiam de um estado consistente para um inconsistente. Existem diferentes tipos de mecanismos de integridade implementados pelos SGBDs. A integridade de entidade garante que cada linha em uma tabela é um registro exclusivamente identificável. Você pode aplicar a integridade de entidade para uma tabela especificando uma restrição PRIMARY KEY. Por exemplo, a coluna ProductID da tabela produtos é uma chave primária para a tabela.

A integridade referencial, definida através da restrição FOREIGN KEY, assegura que as relações entre tabelas permanecem preservadas ao longo do tempo. Já a integridade de domínio garante que os valores de dados dentro de um banco seguem regras definidas para valores: alcance e formato. Um banco de dados pode impor essas regras usando uma variedade de técnicas, incluindo restrições CHECK, UNIQUE e restrições padrão. Essas serão as restrições apresentadas nesse artigo, mas



A lista a seguir fornece alguns exemplos de restrições de integridade de domínio:

- Um nome do produto não pode ser NULL;
- Um nome do produto deve ser exclusivo;
- A data de uma ordem não deve ser no futuro;
- A quantidade de produto em uma ordem deve ser maior do que zero.

Tipos de dados no SQL Server

Há diferentes tipos de dados no SQL Server: de sistema e de usuários (User-Defined Types - UDTs) ou SQL Common Language Runtime (CLR). A seguir temos uma amostra dos tipos disponíveis no SQL Server:

String:

- CHAR;
- VARCHAR;
- NCHAR;
- NVARCHAR;



- **BIT**: pode assumir apenas os valores 0 ou 1. É utilizado para armazenar valores lógicos;
- **TINYINT**: valores inteiros entre 0 e 256;
- **SMALLINT**: valores numéricos inteiros variando de –32.768 até 32.767;
- **INT**: valores numéricos inteiros variando de -2.147.483.648 até 2.147.483.647;
- **BIGINT**: valores numéricos inteiros variando de –92.23.372.036.854.775.808 até 9.223.372.036.854.775.807.

Tipos numéricos aproximados

- FLOAT/DOUBLE;
- REAL.

Data e Hora

- DATETIME;
- SMALLDATETIME.

IDENTITY

tabelas possuem uma coluna ou um conjunto de colunas que identificam a linha: a primary key.

UNIQUEIDENTIFIER

Também conhecido como GUID (Identificador global exclusivo), o tipo uniqueidentifier é armazenado como um valor binário de 16 bytes e para chamá-lo você precisa usar a função `NEWID()`. A principal vantagem do uso de GUIDs é que eles são exclusivos em todo o espaço e tempo. A grande desvantagem de usá-los é que eles são grandes, sendo um dos maiores tipos de dados do SQL Server, então usá-los tornará o índice mais lento. Esse é um exemplo de um GUID formatado: B85E62C3-DC56-40C0-852A-49F759AC68FB. Já esse é um exemplo de um GUID binário: 0xff19966f868b11d0b42d00c04fc964ff

CONSTRAINTS

O principal objetivo de uma restrição é fazer cumprir uma regra no banco de dados, por exemplo, para garantir que os dados inseridos sejam válidos e que obedeçam às regras de negócio. **Na linguagem T-SQL temos cinco constraints**. A **Listagem 1** apresenta um script que exemplifica o uso

ainda não exista outra restrição de chave primária na tabela;

- Foreign key: o objetivo principal da chave estrangeira é assegurar a integridade referencial dos dados. Ela apoia a definição do relacionamento entre duas tabelas;
- Unique: garante que valores duplicados não serão inseridos na coluna;
- Check: impõe integridade de domínio limitando os valores aceitos por uma coluna. A verificação de restrição é usada para limitar a gama de valores que pode ser colocada em uma coluna;
- Default: define um valor padrão caso não seja inserido nenhum valor no campo. É utilizado para inserir um valor predefinido para uma coluna, sendo que o valor padrão será adicionado a todos os novos registros se nenhum outro valor for especificado durante a inserção.

```
1  CREATE TABLE Tb_Pessoa
2  (
3   Pessoa_Id INT PRIMARY KEY,
4   Nome_Fk INT FOREIGN KEY REFERENCES Tb_Pessoa (Pessoa_Id) NOT NULL,
5   Cidade VARCHAR (50) UNIQUE,
6   Idade INT CHECK (Idade >18),
7   Data DATE DEFAULT GETDATE ()
```





Figura 1. Tabela com as cinco constraints criada no SQL Server

Funções de Data

O SQL Server possui dois tipos básicos para armazenar e trabalhar com datas: `datetime` e `smalldatetime`. A diferença é que o tipo `datetime` armazena até centésimos de segundos e o `smalldatetime` até segundos. O `datetime` aceita datas até no mínimo 01/01/1753, abaixo disso ele gerará erro. Já o `smalldatetime` armazena datas de até no mínimo 01/01/1900.

As principais funções de data no SQL Server são: `DATEPART`, `DATEADD` e `DATEDIFF`. Elas trabalham referenciando unidades de data, que são: Year (ano), Month (mês) e Day (dia). O DATEADD retorna a data através da soma do número especificado, como mostra a **Figura 2**. Já o DATEPART retorna a parte especificada de uma data, como mostra a **Figura 3**. Por fim, o DATEDIFF retorna o cálculo da

Figura 2. Retornando a soma do dia 12 com o número 6 como especificado

Figura 3. Retornando o mês somente

Figura 4. Retornando a subtração de duas datas

Funções de String

de uma expressão de caracteres, binários, texto ou imagem. O código a seguir apresenta um exemplo de uso e a **Figura 5** mostra o resultado de sua execução:

```
1 SELECT SUBSTRING(FirstName, 1, 1), LastName, FirstName
2   FROM Person.Person;
```

Figura 5. Retorna a letra inicial do sobrenome

A função `REPLACE()` é capaz de localizar uma determinada expressão em uma string e, ao encontrá-la, realiza a substituição por outra expressão conforme definido nos parâmetros, como mostra a **Figura 6**.

Figura 6. Substituindo a string “cde” pelo “xxx” da cadeia de caracteres “abcdefg”

A função `REPLICATE()` realiza a repetição de caracteres referentes ao número de vezes definido em um de seus parâmetros. O exemplo apresentado na **Listagem 2** reproduz um caractere 0 quatro vezes na frente de uma linha de produção de código. A **Figura 7** apresenta o resultado de sua execução.

```
1 SELECT Name,
2   Replicate ( '0', 4) + [ProductLine] AS 'Code Line'
3   FROM [Production].[Product]
4   WHERE [ProductLine] = 'T'
5   ORDER BY [Name];
```

Listagem 2. Exemplo utilizando a função `replicate`

Já a função `LEFT()` retorna a parte da esquerda de uma cadeia de caracteres, dependendo do número de caracteres especificado levando em consideração os espaços, como mostra o exemplo da **Figura 8**. Observe que no exemplo, o número 5 especificado corresponde à quantidade de caracteres que deverá ser retornada na consulta na parte esquerda.

Figura 8. Uso da função `LEFT()`

A função `RIGHT()` trabalha de forma semelhante ao LEFT, porém retorna a parte da direita de uma cadeia de caracteres levando em consideração os espaços, como mostra o exemplo da **Figura 9**. Nesse exemplo, o número 5 especificado corresponde à quantidade de caracteres que deverá ser retornada na consulta na parte direita.

Figura 9. Uso da função `RIGHT()`

Já a função `RTRIM()` retorna uma expressão de caracteres depois de remover espaços em branco à direita. A **Listagem 3** apresenta um exemplo de seu uso e a **Figura 10** mostra o resultado de sua execução. Em complemento, a função `LTRIM()` retorna uma expressão de caracteres depois de remover espaços em branco à esquerda.

```
1 DECLARE @STRING_TRIM VARCHAR(60);
2 SET @STRING_TRIM = '     7 espaços após essa frase.';
3 SELECT @STRING_TRIM + 'Proxima string.' AS Com_Espaço;
4
5 SELECT LTRIM (@STRING_TRIM) + ' Proxima string.' AS Sem_Espaço;
```

Listagem 3. Exemplo utilizando a função `rtrim`

Figura 10. Removendo os espaços em branco a direita após a primeira frase

A função `STUFF()` insere ou substitui uma cadeia de caracteres dentro de um campo texto. Para isso, ela especifica a posição do primeiro e último caracteres da cadeia que serão substituídos e, em seguida, efetua a exclusão da cadeia e insere a nova, como mostra o exemplo da **Figura 11**. Nesse exemplo, definimos que na 5ª posição da frase "SQL Server" será inserida a palavra Microsoft.

Figura 11. Uso da função `STUFF()`

A função `LEN()` retorna a quantidade de caracteres da cadeia especificada, eliminando os espaços em branco à direita, diferente da função `datalenght()` que conta o número de bytes. O código a

```
3 | FROM [Person].[Person]
```

Figura 12. Retorna a quantidade de caracteres para cada nome da coluna FirstName

As funções CHARINDEX e PATINDEX retornam à posição inicial de um padrão que você especifica. A diferença entre elas está no fato de que a PATINDEX permite usar caracteres curingas. Ambas as funções usam dois argumentos. Com PATINDEX incluímos o sinal “%” antes e depois do teste padrão. Caso estejamos à procura de um padrão como o primeiro ou os últimos caracteres em uma coluna, devemos omitir o primeiro “%” ou o último “%”. Para CHARINDEX, o padrão não pode incluir caractere curinga. O segundo argumento é uma expressão de caracteres, geralmente um nome de coluna. A **Figura 13** mostra um exemplo de uso que retorna o início da posição da string “SQL” na frase “Microsoft SQL Server”.



Figura 13. Uso da função `CHARINDEX`

Essa chamada retornará a localização da cadeia “SQL”, começando na sequência de “Microsoft SQL Server”. Nesse caso, a função `CHARINDEX` retornará o número 11, que como você pode ver, é a posição inicial de “S” na cadeia “Microsoft SQL Server”.

Agora temos o seguinte comando `CHARINDEX`:

```
1 | SELECT CHARINDEX ('7.0', 'Microsoft SQL Server 2000')
```

Para esse exemplo, o resultado será zero porque a string procurada não pode ser identificada no texto.

A função `PATINDEX` retorna a posição inicial do padrão dentro da sequência pesquisada, como

- %: é usado para representar qualquer caractere ou conjunto de caracteres antes, depois ou toda string;
- [ ]: é usado para procurar um caractere único dentro de um intervalo;
- [^]: é usado para procurar por uma sequência sem o caractere definido no colchete após o símbolo ^ e na posição especificada.
- _ (Sublinhado): usado para encontrar uma string sem considerar o valor do caractere inicial.

Figura 14. Retorna a posição inicial de “BC” na cadeia de caracteres

Por fim, a função `UPPER()` retorna os dados da string em maiúsculo. O código a seguir apresenta um exemplo de seu uso e a **Figura 15** traz o resultado de sua execução:

```
3 | FROM [Person].[Person]
```

Figura 15. Retornando todos os nomes da coluna em letra maiúscula

Tipos de joins

O uso de junções permite consultar dados de duas ou mais tabelas com base em suas relações. Existem diferentes tipos de junção que podem ser utilizados em consultas SQL. A **Figura 16** ilustra a diferença entre eles.

Um INNER JOIN seleciona todas as linhas de ambas as tabelas, desde que haja uma correspondência entre as colunas delas. Ele retorna dados apenas quando as duas tabelas possuem chaves correspondentes na cláusula ON do JOIN. Esse tipo de JOIN é usado quando se quer recuperar dados em mais de uma tabela através da igualdade de suas foreign keys. Na **Listagem 4** é realizada uma consulta aos funcionários que são de Seattle. A **Figura 17** apresenta o resultado da consulta.

```
1  SELECT p.LastName, p.FirstName, e.JobTitle, a.City, sp.StateProvinceCode
2   FROM HumanResources.Employee e
3   INNER JOIN Person.Person p
4   ON p.BusinessEntityID = e.BusinessEntityID
5   INNER JOIN Person.BusinessEntityAddress bea
6   ON bea.BusinessEntityID = e.BusinessEntityID
7   INNER JOIN Person.Address a
8   ON a.AddressID = bea.AddressID
9   INNER JOIN Person.StateProvince sp
10  ON sp.StateProvinceID = a.StateProvinceID
11  WHERE a.City = 'Seattle'
```

Listagem 4. Tabela `HumanResources.Employee` realizando join com mais quatro tabelas diferentes

Figura 17. Resultado do select

Com LEFT JOIN todos os dados da tabela à esquerda são retornados. Na tabela à direita, os dados correspondentes são retornados, além de valores NULL. Já o RIGHT JOIN retorna todos os dados da tabela à direita e, da tabela à esquerda, apenas aqueles que possuem correspondência são retornados, além de valores NULL (onde existe um registro na tabela à direita, mas não na tabela à esquerda).

O FULL OUTER JOIN, conhecido como OUTER JOIN ou simplesmente FULL JOIN, retorna todos os registros. Por fim, temos o CROSS JOIN (método implícito), que efetua um produto cartesiano das tabelas e, por isso mesmo, não tem a cláusula ON. Esse comando não permite especificar uma condição para a junção, dessa forma, o resultado será uma combinação de todas as linhas de ambas as tabelas, como mostra a **Figura 18**.

Figura 18. Representação do cross join

EXCEPT, INTERSECT, UNION e UNION ALL

O comando INTERSECT funciona como o operador AND, selecionando apenas o valor presente nas duas tabelas. Ele retorna todas as linhas presentes tanto no resultado da consulta1 como na consulta2. As linhas duplicadas são eliminadas por padrão.

Já o comando `EXCEPT` retorna somente as linhas, a partir da primeira instrução, que não existem na segunda instrução. Ele retorna todas as linhas que estão no resultado da consulta1, mas não no resultado da consulta2. Observe as duas tabelas Departamento_A (**Figura 19**) e Departamento_B (**Figura 20**) e logo em seguida o resultado dos dois comandos na **Figura 21**.

Figura 20. Departamento_B

Figura 21. Diferença entre `EXCEPT` e intersect

Existem algumas regras que precisam ser observadas ao combinarmos os resultados de consultas que usam `EXCEPT` ou INTERSECT. Por exemplo, a quantidade e a ordem das colunas devem ser iguais nas consultas consideradas. Além disso, tem-se que os tipos de dados devem ser compatíveis nas colunas consideradas. Por exemplo, não poderíamos fazer o seguinte: consultar no Departamento_A o NOME e CARGO e não os considerar também na tabela Departamento_B. Ao fazer isso, teríamos

Muitas vezes você se depara com a tarefa de comparar duas ou mais tabelas ou resultados da consulta para determinar qual informação é a mesma e qual não é. Uma das formas mais comuns para fazer essa comparação é usar o UNION ou UNION ALL. O UNION faz a união de duas ou mais tabelas. Essa instrução efetivamente faz um SELECT DISTINCT, no qual não serão retornados valores repetidos. Já o UNION ALL retorna todos os valores das duas ou mais tabelas, sendo que a diferença em relação ao UNION é que se tivermos dois ou mais valores repetidos, eles serão retornados.

Funções de conversão

CAST e CONVERT

Cast é uma função do padrão ANSI e Convert é uma função do engine do SQL Server (T-SQL). Ambas convertem explicitamente uma expressão de um tipo de dados em outro. CAST é uma variante sintática de CONVERT. No CONVERT, se o comprimento não for especificado, o padrão será de 30 caracteres. Para ambos, temos dois tipos de conversão:

- **Conversão implícita:** o engine do SQL Server realiza a conversão de forma automática sem especificar a função `Cast` ou `Convert`;



AdventureWorks2012. Observe na **Figura 22** que o tipo de dado na coluna ModifiedDate é datetime e retorna as informações no formato AAAA/MM/DD, que é o comportamento padrão do SQL Server. Em seguida, modificamos o tipo de dado retornado utilizando as funções CAST e CONVERT, conforme pode ser observado nas **Figuras 23** e **24**.

Figura 22. Tabela com formato com formato padrão, ano, mês e dia

Figura 23. Convertendo a data usando o estilo 103 para retornar em novo formato

Figura 24. Cast convertendo uma expressão de um tipo de dados em outro

TRY_CONVERT, TRY_CAST e TRY_PARSE

A funcionalidade básica de ambas as funções é a mesma: verificar se um valor fornecido em um tipo de dados pode ser convertido e efetuar a conversão para outro tipo de dados. Caso a conversão falhe, será retornado o valor NULL. A **Figura 25** apresenta um exemplo utilizando as funções no qual não foi possível converter a data 02-2013-20 por um erro no formato, retornando então null. Entretanto, o uso do try_parse permitiu realizar a operação com sucesso.

Figura 25. Uso das funções TRY_CONVERT, TRY_CAST e TRY_PARSE

Windows Functions são formas de obter diferentes perspectivas sobre um conjunto de dados sem ter que fazer chamadas repetidas ao servidor. Elas foram implementadas para solucionar problemas já mapeados e difíceis de serem solucionados, começando a partir do SQL Server 2005. Atualmente temos diversas Windows Functions implementadas.

Elas são funções para serem trabalhadas com um conjunto de linhas que são definidas por uma cláusula OVER. Essa cláusula possibilita trabalhar com totais, agrupamentos, ordenações, cálculos complexos dentre outros. Os tipos de Windows Functions são:

- Funções de agregação: SUM, AVG, MIN, MAX, COUNT.
- Funções de classificação: ROW_NUMBER, RANK, DENSE_RANK, NTILE;
- OFFSET: LAG, LEAD, FIRST_VALUE, LAST_VALUE.

As funções de agregação executam um cálculo em um conjunto de valores, sendo que elas computam um único resultado para várias linhas de entrada. As funções agregadas normalmente são usadas com a cláusula GROUP BY (utilizado para agrupar seu resultado por uma ou mais colunas) e, com exceção do COUNT, as funções ignoram valores nulos.

A **Listagem 5** utiliza as funções MIN, MAX, AVG e COUNT com a cláusula OVER para fornecer valores



```
SELECT DISTINCT Name,
 MIN(Rate) OVER (PARTITION BY edh.DepartmentID) AS Salario_Min,
 MAX(Rate) OVER (PARTITION BY edh.DepartmentID) AS Salario_Max,
 AVG(Rate) OVER (PARTITION BY edh.DepartmentID) AS Salario_Medio,,
 COUNT(edh.BusinessEntityID) OVER (PARTITION BY edh.DepartmentID)
  AS FuncionarioDepart
 FROM HumanResources.EmployeePayHistory AS eph
 JOIN HumanResources.EmployeeDepartmentHistory AS edh
 ON eph.BusinessEntityID = edh.BusinessEntityID
 JOIN HumanResources.Department AS d
 ON d.DepartmentID = edh.DepartmentID
 WHERE edh.EndDate IS NULL
 ORDER BY Name;
```

Listagem 5. Exemplo de uso do MIN, MAX, AVG e COUNT com a cláusula OVER

Figura 26. Resultados dos valores mínimo, máximo, médio e quantidade para cada departamento

classificação:

- **ROW_NUMBER:** atribui um número sequencial para cada linha no conjunto de resultados;
- **RANK:** define o número de posição de cada linha no conjunto de resultados. Se os valores na coluna de classificação são os mesmos, eles recebem o mesmo valor. No entanto, o próximo número na sequência de classificação é ignorado;
- **DENSE_RANK:** define o número de posição de cada linha no conjunto de resultados. Se os valores na coluna de classificação são os mesmos, eles recebem o mesmo valor. O próximo número na sequência ranking é então usado para classificar a linha ou linhas que se seguem;
- **NTILE:** divide o conjunto de resultados para o número de grupos especificados com um argumento para a função. Um número de grupo é então atribuído a cada fila identificando qual o grupo que a linha pertence.

Todas as funções de classificação começam com o número 1 ao atribuir valores e classificam os dados com base na coluna especificada na cláusula ORDER BY. Assim, com a cláusula ORDER BY, identificamos a coluna sobre a qual deseja basear o ranking. Também podemos especificar se as linhas devem ser classificadas em ordem crescente ou decrescente. A **Listagem 6** apresenta um exemplo e, em seguida, a **Figura 27** apresenta o seu resultado.



```
3  SalesLastYear,
4   ROW_NUMBER () OVER (ORDER BY SalesLastYear ASC) AS RowNumber,
5   RANK () OVER (ORDER BY SalesLastYear ASC) AS Vendas_Rank,
6   DENSE_RANK () OVER (ORDER BY SalesLastYear ASC) AS DenseRank,
7   NTILE (4) OVER (ORDER BY SalesLastYear ASC) AS Ntile_rank
8  FROM Sales.SalesPerson;
```

Listagem 6. Tabela Sales.SalesPerson do banco de dados AdventureWorks2012 trabalhando com as funções classificação

Figura 27. Retorno das quatro formas de classificação

A cláusula OFFSET-FETCH fornece uma opção para buscar apenas uma janela ou página de resultados do conjunto de resultados. Ela pode ser usada somente com a cláusula ORDER BY e considera as seguintes opções:

os valores em uma linha seguinte;

- **LAG:** usamos essa função em uma instrução SELECT para comparar valores na linha atual com valores em uma linha anterior;
- **FIRST_VALUE:** retorna o primeiro valor em um conjunto ordenado de valores no SQL Server;
- **LAST_VALUE:** retorna o último valor em um conjunto ordenado de valores no SQL Server.

Na **Listagem 7**, LEAD e LAG acessam dados de uma linha subsequente e linha anterior filtrando pela coluna SalesOrderId com os ids 43670, 43669, 43667, 43663. A **Figura 28** apresenta o resultado. Para o conjunto de resultados retornados, é muito claro que a função `LEAD` retorna o valor da linha seguinte e a função `LAG` retorna o valor que foi encontrado na linha anterior.

```
SELECT s.SalesOrderID , s.SalesOrderDetailID , s.OrderQty ,
 LEAD ( SalesOrderDetailID ) OVER ( ORDER BY SalesOrderDetailID
  ) LeadValue ,
 LAG ( SalesOrderDetailID ) OVER ( ORDER BY SalesOrderDetailID
  ) LagValue
 FROM Sales.SalesOrderDetail s
 WHERE SalesOrderID IN ( 43670 , 43669 , 43667 , 43663 )
 ORDER BY s.SalesOrderID , s.SalesOrderDetailID , s.OrderQty
```

Figura 28. Exemplo com LEAD e LAG

Expressão CASE e IIF

Na cláusula `CASE`, se o resultado for verdade, então o valor definido na expressão é o resultado obtido. Caso seja falso, o processo se repetirá em todas as cláusulas `WHEN` seguintes. Se o resultado de nenhuma condição `WHEN` for verdadeiro, então o valor da expressão `CASE` será o valor do resultado na cláusula `ELSE`. Se a cláusula `ELSE` for omitida e nenhuma condição for satisfeita, o resultado será nulo.

A cláusula IIF é uma forma abreviada de se usar a expressão `CASE`. De forma semelhante, se a expressão for verdadeira retorna, é retornado um valor, se for falsa, retorna outro especificado.

INSERT, UPDATE ou DELETE.

As subquerys devem sempre ser informadas entre parênteses, pois assim o SQL Server consegue saber que estamos nos referindo a uma subquery. Quando possível, devemos utilizar um JOIN ao invés de uma subquery, pois o otimizador de consultas do SGBD pode executar algumas operações a mais quando se utiliza subqueries, prejudicando o seu desempenho. Na **Listagem 8** temos três selects operando em uma sub consulta que ao fina retornará as colunas `SalesPersonID`, `OrderDate`, `DailyTotal`, especificadas no primeiro select. A **Figura 29** apresenta o resultado obtido.

```
SELECT
  SH3.SalesPersonID,
  SH3.OrderDate,
  SH3.DailyTotal,
  SUM(SH4.DailyTotal) RunningTotal
 FROM
 (SELECT SH1.SalesPersonID, SH1.OrderDate,
  SUM(SH1.TotalDue) DailyTotal
  FROM Sales.SalesOrderHeader SH1
  WHERE SH1.SalesPersonID  IS NOT NULL
  GROUP BY SH1.SalesPersonID
```

```
16      FROM Sales.SalesOrderHeader SH2
17       WHERE SH2.SalesPersonID  IS NOT NULL
18        GROUP BY SH2.SalesPersonID,
19    SH2.OrderDate) SH4
20    ON SH3.SalesPersonID = SH4.SalesPersonID
21    AND SH3.OrderDate > SH4.OrderDate
22    GROUP BY SH3.SalesPersonID,
23     SH3.OrderDate,
24     SH3.DailyTotal
25     ORDER BY SH3.SalesPersonID,
26     SH3.OrderDate
```

Listagem 8. Exemplo de sub consulta

Figura 29. Resultado da sub consulta

normalizada para um formato diferente, e assim exibir as informações que você precisa de uma maneira mais legível aos olhos do usuário que requisitou as informações.

O Unpivot é um operador relacional, atuando ao contrário da operação pivot. É um mecanismo que transforma colunas em linhas. O código apresentado na **Listagem 9** retorna uma tabela de duas colunas que possui apenas quatro linhas.

```
1 SELECT DaysToManufacture,
2   AVG(StandardCost) AS Custo_medio
3   FROM Production.Product
4   GROUP BY DaysToManufacture;
```

Listagem 9. Exemplo de consulta

Já o código da **Listagem 10** exibe o mesmo resultado, mas dessa vez os valores `DaysToManufacture` serão títulos de coluna. Uma coluna é criada para três dias[3], embora os resultados sejam NULL. A **Figura 30** apresenta o resultado.



```
3  FROM
4   (SELECT DaysToManufacture, StandardCost
5    FROM Production.Product) AS Tabela_fonte
6     PIVOT
7    (
8     AVG(StandardCost)
9     FOR DaysToManufacture IN ([0], [1], [2], [3], [4])
10   ) AS TabelaPivot;
```

Listagem 10. Exemplo dinamizado

Figura 30. Conjunto de resultados, o primeiro resultado sem o PIVOT e o segundo com o PIVOT

Agora suponha que exista uma tabela no banco de dados armazenada como pivot, como mostra a **Figura 31** e que desejamos girar os campos da coluna Emp1, Emp2, Emp3, Emp4, Emp5 para linhas. A coluna que conterá os valores de coluna que você está girando será chamada de `Employee` e a

Figura 31. Tabela temporária criada no formato pivot para exemplificar

```
SELECT VendorID, Employee, Orders
  FROM
  (SELECT VendorID, Emp1, Emp2, Emp3, Emp4, Emp5
  FROM #pvt) p

  UNPIVOT
  (Orders FOR Employee IN
  (Emp1, Emp2, Emp3, Emp4, Emp5)
  )AS "unpivot";
```

Listagem 11. Exemplo utilizando unpivot

Figura 32. Resultado do Unpivot, transformado linhas em colunas em comparação com a **Figura 31**

CROSS APPLY e OUTER APPLY

O operador APPLY permite unir duas tabelas ou expressões SQL. Ele possui duas variantes, CROSS APPLY e OUTER APPLY. O CROSS APPLY é equivalente à expressão INNER JOIN, porém sem a cláusula ON. Já o OUTER APPLY é equivalente à expressão LEFT JOIN. Com INNER JOIN, não teríamos como inserir o HAVING, nem realizar GROUP BY, fazendo com que o banco de dados realize várias consultas ou sub consultas exigindo mais tempo de construção.

A maioria das consultas que empregam CROSS APPLY pode ser reescrita usando um INNER JOIN. Contudo, o CROSS APPLY pode render melhor plano de execução e melhor desempenho.

Agrupamento de dados (Rollup, Cube e Groupping Sets)

Agora expandiremos um pouco o tema Group By, discutindo o Rollup, Cube e Grouping Sets. O

especificar somente os agrupamentos que quisermos usar.

Os operadores ROLLUP, CUBE ou GROUPING SETS podem gerar o mesmo conjunto de resultados como ao usar UNION ALL para combinar agrupamentos de consultas individuais; entretanto, o uso de um operador GROUP BY normalmente é mais eficiente. O GROUPING SETS permite especificar mais de uma opção GROUP BY no mesmo conjunto de registros. O operador ROLLUP é utilizada para calcular o valor agregado para cada nível de uma hierarquia.

Índice

Para acessar dados dentro das tabelas, há dois modos que o SQL Server trabalha: o Table Scan e os índices. No table Scan é realizada uma varredura física, linha a linha, até encontrar a informação solicitada. Já o índice cria atalhos para acesso aos dados de forma que o desempenho da operação seja otimizado.

No SQL Server temos dois tipos de índices: clusterizados e não clusterizados. O tipo clusterizado é um índice gerado na própria estrutura de armazenamento dos dados. Esse índice fará com que os dados da sua tabela fiquem organizados fisicamente na sequência. Por isso, em uma tabela pode



Já o índice não clusterizado é um índice criado em uma estrutura separada dos dados físicos. São criadas páginas de índices que irão conter os apontadores para os registros físicos. Eles são eficientes quando precisamos ter várias maneiras de pesquisar os dados dentro de uma tabela. Por exemplo, em uma tabela que contém os livros de uma livraria, armazenamos o nome do livro, o ISBN, o autor e a editora. Quando pesquisamos um livro, poderemos pesquisar por qualquer uma dessas colunas, nesse caso, precisaremos ter índices para cada uma das colunas, então criaremos índices non-clustered associados a elas.

Níveis de isolamento

A instrução SET TRANSACTION ISOLATION LEVEL está presente no SQL Server já a algum tempo. Os níveis de isolamento são importantes no contexto em que existem muitas transações acontecendo ao mesmo tempo. Os níveis trabalham os bloqueios nas linhas das tabelas ou nas tabelas para que não haja falta de consistência após o termino de uma transação.

O nível de isolamento read commited é parecido com o read uncommited, a principal diferença é que seu código lerá apenas os dados confirmados ao executar o modo READ COMMITED. Ele



Pelo fato dele ler somente as informações realmente escritas no banco de dados, se uma transação estiver trabalhando com a tabela que deseja ler, o SQL esperará liberar a transação para então fazer a leitura.

Já o read uncommited (leitura não confirmada) especifica que as instruções podem ler linhas que foram modificadas por outras transações e que ainda não foram confirmadas. Ele não oferece nenhuma garantia de isolamento, mas tem o melhor desempenho.

O nível de isolamento repeatable read (leitura repetível) evita leituras sujas e leituras não repetitivas. Uma leitura suja é uma operação de leitura que ocorre nos dados que foram modificados por uma transação que ainda não foi consolidada. Uma leitura não repetitiva pode ocorrer quando os bloqueios de leitura não são adquiridos na execução de uma operação de leitura.

Por fim, o nível de isolamento serializable é o mais restritivo de todos. Ele coloca um bloqueio no conjunto de dados impedindo que outros usuários atualizem ou insiram linhas até que a transação seja concluída. Esse nível especifica que todas as transações ocorrem de uma forma completamente isolada. Use esse tipo somente quando necessário, pois ele pode prejudicar o desempenho de seu banco.



Um trigger é um objeto de base de dados que está *ligado* a uma tabela. Em muitos aspectos, é semelhante a uma procedure (procedimento armazenado).A principal diferença entre um trigger e uma procedure é que o primeiro está ligado a uma tabela e só é *acionado* quando um INSERT, UPDATE ou DELETE ocorre. Você especifica a ação(s) e a modificação que dispara o gatilho quando ele é criado. Para criar um trigger é utilizado o comando CREATE TRIGGER, para alterar ALTER TRIGGER e para deletar DROP TRIGGER.

Um trigger é classificado em dois tipos: INSTEAD OF, INSTEAD AFTER.

- AFTER: o trigger somente é executado após a gravação dos dados na tabela. O gatilho associado a um evento será acionado somente após a linha passar por todas as verificações, tais como chave primária, regras e restrições. Se a inserção falhar, o SQL Server não disparará o gatilho AFTER;
- INSTEAD OF: o trigger será disparado antes do registro ser modificado na tabela.

A criação de um trigger envolve duas etapas:

1. Um comando SQL que vai disparar o trigger (INSERT, DELETE, UPDATE);
2. A ação que o trigger vai executar (bloco de códigos SQL).

- Quando o gatilho será disparado;
- Os comandos que irão determinar o que será executado na ação.

A **Listagem 12** mostra a sintaxe de criação de um trigger.

```
1 CREATE TRIGGER [NOME DO TRIGGER]
2 ON [NOME DA TABELA]
3 [FOR/ AFTER/ INSTEAD OF] [INSERT/UPDATE/DELETE]
4 AS
5 -- CORPO DO TRIGGER
```

Listagem 12. Sintaxe simples da criação de um trigger

View

Views são tabelas virtuais acessadas frequentemente e que facilitam as consultas no banco de dados.

O uso de view é particularmente útil quando se deseja dar o foco a um determinado tipo de informação mantida pelo banco de dados. Imagine um banco de dados corporativo que é acessado

Isso permite que diferentes usuários vejam as mesmas informações sob uma perspectiva diferente.

Na **Listagem 13**, vemos uma view criada para facilitar a consulta aos funcionários que são de Seattle. Dessa forma você pode ter informações desses funcionários em uma “tabela virtual”.

```
1  CREATE VIEW dbo.SeattleOnly
2  AS
3  SELECT p.LastName, p.FirstName, e.JobTitle, a.City, sp.StateProvinceCode
4   FROM HumanResources.Employee e
5   INNER JOIN Person.Person p
6   ON p.BusinessEntityID = e.BusinessEntityID
7   INNER JOIN Person.BusinessEntityAddress bea
8   ON bea.BusinessEntityID = e.BusinessEntityID
9   INNER JOIN Person.Address a
10  ON a.AddressID = bea.AddressID
11  INNER JOIN Person.StateProvince sp
12  ON sp.StateProvinceID = a.StateProvinceID
13  WHERE a.City = 'Seattle'
```

Listagem 13. View criada para facilitar uma consulta frequente entre tabelas diferentes, filtrando pela cidade de Seattle

Procedures são **conjuntos de instruções T-SQL** executadas dentro de um único plano de execução. Elas podem melhorar a performance (como ela é armazenada dentro do banco, ela é executada rapidamente) e criam mecanismos de segurança nos dados do banco.

Considerando a forma como o SQL Server é utilizado no dia a dia em muitas organizações, é possível afirmar que grande parte do **desenvolvimento em T-SQL** gira em torno da construção de stored procedures. Muitas dessas rotinas são implementadas com o intuito de produzir resultados dinâmicos, empregando para isso uma consulta SQL simples ou até agrupamentos mais complexos de instruções (podendo envolver uma série de cálculos ou mesmo junções de dados provenientes de diferentes fontes).

O exemplo apresentado na **Listagem 14** cria um procedimento armazenado que retorna informações de um funcionário específico passando os valores para o primeiro nome do empregado e último nome. O resultado de sua execução é apresentado na **Figura 33**.

```
1 CREATE PROCEDURE HumanResources.uspGetEmployees
2  @LastName nvarchar(50),
3  @FirstName nvarchar(50)
```

```
8 |   WHERE FirstName = @FirstName AND LastName = @LastName;
```

Listagem 14. Exemplo de procedure

Figura 33. Retorno do procedimento apresentado na **Listagem 14**

Funções de usuários – USER FUNCTION

Esse é um bom recurso a se utilizar, pois são objetos que criam planos de execução e assim dão melhor performance em suas chamadas. Elas sempre retornam valores e são usadas como parte de uma expressão.

Na **Listagem 15** é criada uma função com valor de tabela embutida no banco de dados AdventureWorks2012. A função considera um parâmetro de entrada, um ID cliente (loja), e retorna

```
CREATE FUNCTION Sales.ufn_SalesByStore (@storeid int)
 RETURNS TABLE
 AS
 RETURN
 (
  SELECT P.ProductID, P.Name, SUM(SD.LineTotal) AS 'Total'
  FROM Production.Product AS P
  JOIN Sales.SalesOrderDetail AS SD ON SD.ProductID = P.ProductID
  JOIN Sales.SalesOrderHeader AS SH ON SH.SalesOrderID = SD.SalesOrderID
  JOIN Sales.Customer AS C ON SH.CustomerID = C.CustomerID
  WHERE C.StoreID = @storeid
  GROUP BY P.ProductID, P.Name
 );
```

Listagem 15. Exemplo de user function

A CTE (Common Table Expression) é um recurso utilizado desde o SQL Server 2005. Ela pode incluir referências em si mesma, e assim se tornar uma expressão de tabela comum recursiva, sendo possível ter várias definições de consulta de CTE em uma CTE não recursiva.

Essa cláusula também pode ser usada em uma instrução CREATE VIEW como parte da instrução SELECT que a define. As CTE podem ser definidas em rotinas definidas pelo usuário, tais como: funções, procedures, triggers ou views.

Ela é composta de um nome de expressão representando a CTE, uma lista de colunas opcionais e uma consulta definindo a CTE. Ela é bem parecida com uma tabela derivada, que não é armazenada como um objeto e que existe apenas durante a execução da consulta. A **Listagem 16** apresenta um exemplo de uso de CTE que recupera a quantidade de cargos para cada id. O resultado de sua execução é apresentado na **Figura 35**.

```
1 WITH ExemploCTE AS
2 (SELECT JobTitle, COUNT(*) NumTitles
3 FROM HumanResources.Employee
4 GROUP BY JobTitle)
```

Listagem 16. Recuperando a quantidade de cargos para cada id

Figura 35. A coluna NumTitles retorna a quantidade de cargos correspondente à coluna BusinessEntity

A expressão de tabela comum contém três partes principais:

- O nome da CTE (que segue o WITH palavra-chave);
- A lista de colunas (opcional);
- A consulta (aparece entre parênteses após o AS).

Merge

uma origem definida com uma tabela de destino com base em uma condição que você especificar.

O MERGE funciona basicamente como uma inserção, atualizando e excluindo as informações necessárias dentro da mesma solicitação. Para utilizá-lo, especificamos um "Source" conjunto de registros, uma tabela de "Target", e definimos como deve ser a junção entre os dois. Então especificamos o tipo de modificação de dados que ocorrerá quando os registros entre os dois dados são combinados ou não são correspondidos.

OMERGE é muito útil, especialmente quando se trata de tabelas presentes em data warehouses, que podem ser muito grandes e requerem ações específicas a serem tomadas quando as linhas estão ou não estão presentes:

- WHEN MATCHED THEN: quando existirem linhas no destino correspondentes às presentes na origem;
- WHEN NOT MATCHED [BY TARGET] THEN: quando não há linhas da origem que correspondem com o destino;
- WHEN NOT MATCHED BY SOURCE THEN: quando há linhas no destino, mas não há linhas correspondentes na origem.



mais de uma vez na mesma cláusula MATCHED.

Inicialmente, deve ser indicado qual tabela será atualizada. Em seguida, definimos a origem dos dados que serão considerados no processo de merge. Essa etapa é seguida da cláusula ON, através da qual definimos como será realizada a ligação entre as duas tabelas. A **Listagem 17** apresenta a sintaxe do Merge.

```
1 MERGE tabelaDestino d
2   USING tabelaOrigem o
3   ON o.coluna = d.coluna
4   WHEN MATCHED THEN
5   DELETE;
```

Listagem 17. Sintaxe básica do Merge

O Transact SQL e o SQL Server oferecem as tecnologias e os recursos que as empresas e organizações necessitam para armazenar grandes quantidades de dados, não deixando de ser útil também até mesmo em pequenas aplicações. O uso de junções e funções de comparação, conversão e



também com a manipulação avançada dos dados armazenados no SGBD. O conhecimento dessa ferramenta e linguagem é essencial para qualquer profissional da área de desenvolvimento de software.

Confira também