Documentação de estudos

Certificação Microsoft 70-761 – Queryng Data with Transact-SQL

O estudo não tem como intuito tratar de boas práticas de modelagem, por este motivo, as tabelas encontradas nos arquivos .sql cobre apenas as necessidades de executar exemplos simples para o entendimento dos recursos de consulta do SQL.

O mapa mental para os estudos da certificação pode ser encontrado no seguinte link:

<https://coggle.it/diagram/W64XqwWWyfIZHOaA/t/mind-map-70-761>

**Select**

A cláusula SELECT é utilizada para realizar consultas em SQL Server e muitos outros bancos de dados. Em SQL Server, a ordem de execução de elementos utilizados nas consultas é:

1. FROM
2. WHERE
3. GROUP BY
4. HAVING
5. SELECT
6. ORDER BY

**Datatype**

Ao criar uma tabela, fazer uma consulta ou conversão de valores no SQL Server, precisamos saber o *datatype* dos dados que serão manipulados para não se deparar com erros de conversão implícita ou explícita. Alguns dos tipos de dados possuem compatibilidade e podem ser convertidos implicitamente pelo próprio SQL Server.

As categorias de tipos de dados disponíveis são: Exact numerics (Unicode character strings), Approximate numerics (Binary strings), Date and time (Other data types) e Character strings.

**Funções de conversão**

Para a conversão de valores no SQL Server, podem ser utilizadas as funções CAST, TRY\_CAST, CONVERT, TRY\_CONVERT, PARSE e TRY\_PARSE.

CAST e CONVERT possuem o mesmo objetivo, mas são utilizados com sintaxes diferente. Além disso, CAST foi criado posteriormente e faz parte do padrão ANSI.

As funções de conversão que iniciam com “TRY\_” fazem um tratamento ao converter os dados. Caso a conversão não seja válida, a o retorno será NULL em vez de exibir um possível erro.

**NULLs**

Ao trabalhar com valores nulos, podemos utilizar as funções ISNULL, NULLIF e COALESCE. NULL pode ser descrito como “valor inexistente” ou “nada”, pois apenas indica que não há valor naquela célula. Cada uma dessas funções possui um tratamento específico para NULL.

ISNULL é utilizado para comparar uma coluna ou variável específica e caso este seja NULL, a função retornará um valor especificado pelo usuário no segundo parâmetro da função.

NULLIF verifica se a coluna ou variável específica contém determinado valor e, caso a condição seja verdadeira, a função retorna NULL. O usuário define a coluna/variável como primeiro parâmetro da função e o valor esperado nesta coluna/variável como segundo parâmetro.

A função COALESCE recebe diferentes colunas/variáveis como parâmetro e verifica parâmetro por parâmetro até que encontre um diferente de NULL. Caso não seja encontrado um parâmetro não nulo, a função retorna NULL.

**Functions**

No SQL Server podemos classificar as funções como determinísticas e não determinísticas. As funções determinísticas são aquelas que sempre retornam um mesmo valor com o uso de um conjunto específico de parâmetros, por exemplo, AVG(). Já as funções não determinísticas são as que retornam valores diferentes mesmo com a utilização de um conjunto de parâmetros imutável, por exemplo, GETDATE().

**Scalar functions**

São funções que tratam de elementos únicos e retornam apenas um valor para cada elemento tratado. Pode ser utilizada em situações onde há necessidade de comparar ou tratar valores específicos.

Alguns exemplos de funções escalares são SUBSTRING, YEAR, MONTH. DAY, CHARINDEX, etc.

**Logical functions (determinística)**

São funções que retornam valore booleanos ou fazem comparações entre valores e retorna resultado único de acordo com a condição estabelecida, funções de exemplo são IIF e ISNUMERIC.

**Window functions (não determinística)**

Window function (ou funções de janela) referem-se a operações de classificação realizadas em um determinado conjunto (podem ser chamados de janelas) de linhas e retorna um resultado agregado. Funções que fazem este tipo de tratamento são RANK, ROW\_NUMBER, DENSE\_RANK, etc.

RANK() é utilizada para classificação de partições de um conjunto de resultados. Esta função possui a cláusula PARTITION BY (opcional) que pode particionar o conjunto de resultados de acordo com as categorias existentes na cláusula FROM e a cláusula ORDER BY (obrigatória) para ordenar os resultados pela coluna desejada. Nem sempre RANK() retorna a classificação em inteiros sequenciais, por exemplo, caso as linhas 2 e 3 possuam valores iguais, a função classifica-os como 2 e a próxima linha é classificada como 4.

A função ROW\_NUMBER() é semelhante à RANK(), a diferença entre elas é que ROW\_NUMBER exibe classificação sequencial, independente de que um valor seja igual ou não ao anterior ou próximo.

A função DENSE\_RANK() também é utilizada com objetivo de classificação, mas diferente da RANK(), caso exista valores iguais, a classificação não deixa espaços entre a sequência, por exemplo, se o registro 2 e 3 possuírem o mesmo valor, eles recebem a classificação 2 e o próximo registro será o 3.

A função NTILE(*integer\_expression*) é utilizada para particionar um conjunto de dados igualmente e recebe como parâmetro um número inteiro que significa o número de grupos em que o *dataset* será dividido. Caso não seja utilizado PARTITION BY, a função dividirá o *dataset* inteiro no número de grupos especificado. Com a utilização de PARTITON BY, a função dividirá em grupo os resultados dentro de cada partição de forma distinta.

**Aggregate functions (determinísticas e não determinísticas, depende do uso)**

Funções agregadas tem como objetivo realizar cálculos em conjuntos de valores especificados na query e retornar um valor único e agregado baseado nas colunas especificadas. Para estas funções, geralmente é utilizada a cláusula GROUP BY, que agrega as colunas não calculadas e disponibiliza estes valores como “categorias”. Exemplos destas funções são COUNT, AVG, MIN, MAX, etc.

**Analytics functions**

Funções analíticas são utilizadas para auxiliar na análise e comparação de dados em um conjunto de resultados.

FIRST\_VALUE(*scalar\_expression*) retorna o primeiro valor em um conjunto ordenado de dados. A função deve receber um parâmetro, geralmente uma coluna, e retornará o primeiro valor do conjunto de resultados completo ou da partição especificada na cláusula PARTITION BY.

LAST\_VALUE(*scalar\_expression*) retorna o último valor em um conjunto ordenado de dados. A função deve receber um parâmetro, geralmente uma coluna, e retornará o último valor do conjunto de resultados completo ou da partição especificada na cláusula PARTITION BY.

LAG(*scalar\_expression [,offset] [,default]*) acessa dados de linhas anteriores de um mesmo conjunto de resultados e retorna um valor especificado pelo usuário. A função deve receber um parâmetro obrigatoriamente e 2 parâmetros opcionais. *Offset* indica há quantas linhas anteriores a expressão deve ser retornada e *default* indica o valor padrão caso o retorno seja NULL.

LEAD(*scalar\_expression [,offset] [,default]*) acessa dados de linhas à frente de um mesmo conjunto de resultados e retorna um valor especificado pelo usuário. A função deve receber um parâmetro obrigatoriamente e 2 parâmetros opcionais. *Offset* indica o número de linhas à frente que a expressão deve ser retornada e *default* indica o valor padrão caso o retorno seja NULL.

**SET XACT\_ABORT {ON | OFF}**

Especifica se o SQL deve reverter a transação atual caso gere um erro.

Quando SET XACT\_ABORT for ON, se uma instrução Transact-SQL gerar um erro em tempo de execução, a transação inteira será encerrada e revertida.

Quando SET XACT\_ABORT é OFF, em alguns casos, somente a instrução Transact-SQL que gerou o erro é revertida e a transação continua a ser processada. Dependendo da gravidade do erro, a transação inteira pode ser revertida mesmo quando SET XACT\_ABORT é OFF. OFF é a configuração padrão.

Os erros de compilação, como erros de sintaxe, não são afetados por SET XACT\_ABORT.

**Views**

Uma *view*é uma tabela virtual criada a partir de uma *query* no banco de dados e pode referenciar mais de uma tabela. *Views* podem ser utilizadas para criar visualizações simples para usuários, disponibilizar a consulta dos dados sem dar acesso às tabelas originais, utiliza como fonte de dados sem a utilização de subqueries ou tabelas derivadas, etc.

Uma *view* pode armazenar apenas um índice *cluster* e outros índices *noncluster*. A adição de índices em *views* pode auxiliar na performance de consulta e retorno dos dados através da utilização do *query optimizer*.

**SET IDENTITY\_INSERT {ON | OFF}**

Permite que o usuário possa inserir um registro com valor explícito na coluna de identidade. Para que seja utilizado, o usuário necessita ter permissão de alteração na tabela.

**Cross apply e Outer apply**

A cláusula APPLY permite a combinação entre duas ou mais tabelas, de forma semelhante à cláusula JOIN. Diferente do JOIN, APPLY é utilizado através de queries. A query chamada pelo operador APPLY é executada para cada linha da tabela referenciada (tabela encontrada no FROM) e sua vantagem quando comparado à cláusula JOIN é a performance em queries longas e a possibilidade do uso de funções.

A cláusula APPLY possui duas variações, CROSS APPLY e OUTER APPLY.

CROSS APPLY retorna os dados das duas ou mais tabelas que correspondem à query do operador APPLY.

OUTER APPLY retorna todos os dados das colunas especificadas das duas tabelas, inclusive resultados que não correspondem à cláusula APPLY.

**MERGE**

A cláusula merge é utilizada para operações em grande escala com o objetivo de comparar dados semelhantes entre duas tabelas e unificar os dados de uma tabela alvo com os dados de uma tabela fonte.

Para ser utilizada, a cláusula necessita de 2 tabelas, uma fonte (de onde os dados serão retirados) e uma tabela alvo (tabela que receberá novos registros ou terá seus registros alterados de acordo com os dados da tabela fonte).

Exemplo de código para utilização:

MERGE INTO StudentTarget as T

USING StudentSource as S

ON T.Id = S.Id

WHEN MATCHED THEN

UPDATE SET T.Name = S.Name

WHEN NOT MATCHED BY TARGET THEN

INSERT (Id, Name) VALUES (S.Id, S.Name);

WHEN NOT MATCHED BY SOURCE THEN

DELETE;

(O código acima pode sem encontrado no arquivo ‘Merge statement.sql’ na pasta de POC’s)

Na cláusula MERGE INTO deve ser especificada a tabela alvo, na cláusula USING especificamos a tabela fonte. Utilizando a cláusula ON, semelhante ao JOIN, é digitada as colunas que serão comparadas para que para que as alterações sejam feitas.

Ao especificar WHEN MATCHED, estamos dizendo que onde a comparação da coluna for correspondida, deve ser aplicado o UPDATE.

Em WHEN NOT MATCHED BY TARGET THEN, especificamos que linhas que existem na tabela fonte e não existem na tabela alvo, devem ser inseridas na tabela alvo.

E, por último, WHEN NOT MATCHED BY SOURCE THEN, especificamos que linhas que existem na tabela alvo e não existem na tabela fonte, devem ser deletadas.

É importante lembrar que este foi apenas um exemplo simples, mas que após cada condição, podemos aplicar tratamentos mais complexos para cada uma das necessidades.

* 1. **– Operadores de conjuntos**

Há 3 operadores unários, são eles + (positivo), - (negativo) e ~ (bitwise):

Positivo: O operador unário positivo é utilizado geralmente em declaração de variáveis. Ele não afeta valores negativos, para realizar a conversão de negativo para positivo é utilizada a função ABS().

Negativo: O operador unário negativo é utilizado para conversão de valores e possui as características matemáticas básicas de conversão de operadores (ex. negativo com negativo = positivo).

Bitwise: Este operador faz a conversão bit a bit.

Ex. Na expressão 0000 0000 1010 1010, ao aplicar o bitwise a expressão será convertida para 1111 1111 0101 0101.

O operador EXCEPT e INTERSECT retornam o resultado da comparação entre consultas ou expressões. Como regra básica, as consultas precisam ter o mesmo número de colunas com o mesmo tipo de dados para que não haja erro em conversões implícitas. O EXCEPT retorna os valores existentes na primeira consulta e que não existe na segunda consulta. O INTERSECT retorna os valores existentes em ambas as consultas ou expressões.

Os operadores UNION e UNION ALL unificam os dados de consultas distintas. Como regra básica, as consultas precisam ter o mesmo número de colunas com o mesmo tipo de dados para que não haja erro em conversões implícitas. O operador UNION faz com que diversos conjuntos de resultados sejam unificados e se tornem apenas um conjunto de resultado eliminando duplicatas. Ao utilizar a cláusula ALL o conjunto de resultados retorna todas as linhas de todos os resultados sem remoção de duplicatas, e caso não seja especificado, o conjunto de resultados não terá duplicatas.

* 1. **– Consultar várias tabelas usando junções**

Através da cláusula JOIN possui diferentes categorias e possibilita fazer consultas em múltiplas tabelas de um banco de dados. A regra para utilização desta cláusula é fazer comparações e junções entre tabelas que possuam relacionamentos (chaves primárias, estrangeiras ou valores correlacionados).

JOIN ou INNER JOIN é utilizado para retornar dados existentes em ambas as tabelas utilizadas na query. Esta categoria é indicada quando se quer excluir possíveis NULLs que podem existir no relacionamento.

LEFT JOIN ou LEFT OUTER JOIN une registros existentes em ambas as tabelas e retorna ainda os dados da tabela à esquerda que não possuem um relacionamento com a tabela da direita.

RIGHT JOIN ou RIGHT OUTER JOIN une registros existentes em ambas as tabelas e retorna ainda os dados da tabela à direita que não possuem um relacionamento com a tabela da esquerda.

FULL JOIN ou FULL OUTER JOIN retorna a listagem unindo as regras do inner, left e right join.

CROSS JOIN é utilizado para juntar tabelas por cruzamento e o relacionamento de chaves entre tabelas não é requerido. O cross join faz com que cada linha da tabela A tenha todos os registros da tabela B, ou vice-versa.

Ao usar JOINs, podemos utilizar também operadores AND e OR para aumentar o filtro de registros nas junções.