**Capítulo 3**

Filtrando e classificando dados

Objetivos do exame neste capítulo:

■■ Trabalhar com dados

■■ dados de consulta usando instruções SELECT.

■■ Implementar tipos de dados.

■■ modificar dados

■■ trabalhar com funções.

F iltering e dados de ordenação são os aspectos mais fundamentais, bem como os mais comuns, de

consultar dados. Quase todas as consultas que você escreve necessidades para filtrar dados, e muitas consultas

envolvem triagem. A maneira tradicional de filtrar dados em T-SQL é baseada em predicados. Contudo,

T-SQL também suporta filtragem de dados com base em um outro conceito-um número especificado de linhas

e ordenação. As opções T-SQL suporta baseadas neste conceito são SUPERIORES e OFFSETFETCH.

Quanto à classificação, mesmo que possa parecer um aspecto trivial de consulta, é realmente

uma fonte para um monte de confusão e mal-entendidos, que este capítulo tenta

esclarecer.

Lições neste capítulo:

■■ Lição 1: Filtrando dados com predicados

■■ Lição 2: classificando dados

■■ Lição 3: Filtrando dados com TOP e OFFSET-busca

**Antes de você começar**

Para concluir as lições neste Neste capítulo, você deve ter:

Experiência ■■ trabalhar com Microsoft SQL Server Management Studio (SSMS).

■■ Algum código T-SQL experiência de escrita.

■■ O acesso a um Server 2012 instância do SQL com o banco de dados exemplo TSQL2012 instalado.

**Lição 1: Filtrando dados com predicados**

T-SQL suporta três cláusulas de consulta que permitem filtrar dados com base em predicados. Essa

são a ON, WHERE e HAVING. O ON e HAVING são cobertos mais tarde no

o livro. ON é coberto como parte das discussões sobre a junta no Capítulo 4, "Combinação de Jogos,"

e TENDO é coberto como parte das discussões sobre agrupamento de dados no capítulo 5, "Agrupando

e de janelas. "Lição 1 neste capítulo centra-se na filtragem de dados com a cláusula WHERE.

Após esta lição, você será capaz de:

■■ Use a cláusula WHERE para filtrar dados com base em predicados.

■■ Filtrar dados envolvendo nulos corretamente.

■■ usar argumentos de busca para filtrar os dados de forma eficiente.

■■ Combine predicados com operadores lógicos.

■■ compreender as implicações da lógica de três valores em dados de filtragem.

■■ dados de caracteres Filter.

■■ data Filtro e dados de tempo.

Tempo de aula estimado: 60 minutos

Logic predicados, de três valores, e Search Argumentos

Já nas primeiras consultas SQL que você já escreveu, você muito provavelmente já começou a usar o

Cláusula WHERE para filtrar dados com base em predicados. Inicialmente, parece uma forma muito simples e

conceito simples. Mas com o tempo, à medida que ganhar uma compreensão mais profunda da T-SQL, você provavelmente

perceber que existem filtragem aspectos que não são tão óbvios. Por exemplo, você precisa

para entender como predicados interagir com nulos, e como filtros com base em tais predicados

comporte-se. Você também precisa entender como formar seus predicados para maximizar a eficiência

de suas consultas, e para isso você precisa estar familiarizado com o conceito de um *argumento de pesquisa.*

Alguns dos exemplos neste capítulo usam a tabela HR.EMPLOYEES do TSQL2012

banco de dados de amostra. Aqui está o conteúdo da tabela (apenas colunas relevantes apresentados).

empid cidade região sobrenome país firstname

------ ---------- ------------- -------- ------- ------ ---

1 Sara Davis EUA WA Seattle

2 Don Funk EUA WA Tacoma

3 Judy Lew EUA WA Kirkland

4 Yael Peled EUA WA Redmond

5 Sven Buck UK NULL Londres

6 Paul Suurs UK NULL Londres

7 Russell Rei UK NULL Londres

8 Maria Cameron EUA WA Seattle

9 Zoya Dolgopyatova UK NULL Londres

Para começar com um exemplo simples, considere a seguinte consulta, que filtra apenas os funcionários

dos Estados Unidos.

Empid SELECT, nome, sobrenome, país, região, cidade

DE HR.EMPLOYEES

ONDE country = N'USA ';

Lembre-se do Capítulo 1, "Fundamentos da Consulta", que um predicado é uma expressão lógica.

Quando nulos não são possíveis nos dados (neste caso, a coluna país é definido como não

permitindo nulos), o predicado pode ser avaliada como verdadeira ou falsa. O tipo de lógica utilizada em tal

caso é conhecido como *lógica de dois valores.* Do filtro, onde retorna somente as linhas para as quais o predicado

avalia a verdade. Aqui está o resultado desta consulta.

Empid cidade região sobrenome país firstname

------ ---------- --------- -------- ------- ---------

1 Sara Davis EUA WA Seattle

2 Don Funk EUA WA Tacoma

3 Judy Lew EUA WA Kirkland

4 Yael Peled EUA WA Redmond

8 Maria Cameron EUA WA Seattle

No entanto, quando nulos são possíveis nos dados, as coisas ficam mais complicadas. Considere o cliente

localização colunas país, região e cidade na tabela de Sales.Customers. Suponha-se que estes

colunas refletem a hierarquia local com base na organização de vendas. Para alguns lugares no

mundo, tal como nos Estados Unidos, todas as três colunas de localização são aplicáveis; por exemplo:

País: EUA

Região: WA

Cidade: Seattle

Mas outros lugares, como o Reino Unido, têm apenas duas peças-the aplicáveis ​​país

e da cidade. Em tais casos, a coluna região é definida como NULL; por exemplo:

País: Reino Unido

Região: NULL

Cidade: Londres

Consideremos, então, uma consulta filtrando apenas os funcionários do estado de Washington.

Empid SELECT, nome, sobrenome, país, região, cidade

DE HR.EMPLOYEES

ONDE região = N'WA ';

Lembre-se do Capítulo 1 que, quando nulos são possíveis nos dados, um predicado pode avaliar

como verdadeiro, falso, e desconhecido. Este tipo de lógica é conhecido como *lógica de três valores.* Quando se utiliza um

operador de igualdade no predicado como na consulta anterior, você tem verdadeira quando ambos os operadores

não são NULL e iguais; por exemplo, WA e WA. Você começa falsa quando ambos não são

Nulo e diferente; por exemplo, OU e WA. Até agora, é simples. A parte complicada é

quando as marcas NULL estão envolvidos. Você começa um desconhecido quando pelo menos um operando é NULL; para

exemplo, NULL e WA, ou mesmo NULL e NULL.

Como mencionado, o filtro ONDE retorna linhas para as quais o predicado é avaliada como verdadeira,

o que significa que ele descarta ambos os casos falsos e desconhecidas. Portanto, a consulta retorna apenas

empregados onde a região não é nulo e igual a WA, como mostrado a seguir.

Empid cidade região sobrenome país firstname

------ ---------- --------- -------- ------- ---------

1 Sara Davis EUA WA Seattle

2 Don Funk EUA WA Tacoma

3 Judy Lew EUA WA Kirkland

4 Yael Peled EUA WA Redmond

8 Maria Cameron EUA WA Seattle

Você pode considerar este comportamento tão intuitivo, mas considere um pedido para retornar apenas funcionários

que não são do estado de Washington. Você emitir a seguinte consulta.

Empid SELECT, nome, sobrenome, país, região, cidade

DE HR.EMPLOYEES

ONDE região <> N'WA ';

Executar a consulta e obtém um conjunto vazio de volta:

empid cidade região sobrenome país firstname

------ ---------- --------- -------- ------- ---------

Você pode fazer sentido do resultado?

Como se vê, todos os funcionários que não são do estado de Washington é do Reino Unido;

Recordamos que a região de lugares no Reino Unido é definido como NULL para indicar que é inaplicável. Até

embora possa estar claro para você que alguém do Reino Unido não é do estado de Washington, é

não está claro para T-SQL. Para T-SQL, um NULL representa um valor em falta, que poderia ser o caso, e

poderia ser WA apenas como poderia ser qualquer outra coisa. Por isso, não se pode concluir com certeza que o

região é diferente de WA. Em outras palavras, quando a região é NULL, a região do predicado <>

'WA' avaliada como desconhecido, e a fila é descartada. Assim, tal predicado só retornam

casos que não são nulas e são conhecidos como sendo diferente de WA. Por exemplo, se você tivesse um

empregado na tabela com uma região NY, como um empregado teria sido devolvido.

Sabendo que na tabela Funcionários uma região NULL representa uma falta e inaplicáveis

região, como você faz retorno T-SQL tais empregados quando se olha para os lugares onde o

região é diferente de WA?

Se você está considerando um predicado como a região <> N'WA 'ou região = NULL, você precisa

lembrar que dois nulos não são considerados iguais uns aos outros. O resultado da expressão

NULL = NULL é, de fato, desconhecido, não é verdade. T-SQL fornece o predicado IS NULL para

devolver um verdadeiro quando o operando testado é NULL. Da mesma forma, o predicado é retornos não NULL

verdadeiro quando o operando testado não é NULL. Então, a solução para este problema é usar o seguinte

Formato.

Empid SELECT, nome, sobrenome, país, região, cidade

DE HR.EMPLOYEES

ONDE região <> N'WA '

Ou região for NULL;

Aqui está o resultado desta consulta.

empid cidade região sobrenome país firstname

------ ---------- ------------- -------- ------- ------ -

5 Sven Buck UK NULL Londres

6 Paul Suurs UK NULL Londres

7 Russell Rei UK NULL Londres

9 Zoya Dolgopyatova UK NULL Londres

Filtros de consulta tem um lado importante desempenho para eles. Por um lado, através da filtragem

linhas na consulta (em oposição a no cliente), a reduzir o tráfego de rede. Além disso, com base em

os filtros de consulta que aparecem na consulta, SQL Server pode avaliar a opção de usar índices

para chegar aos dados de forma eficiente sem a necessidade de uma varredura completa da tabela. É importante notar,

no entanto, que o predicado deve ser de uma forma conhecida como um *argumento de busca* (SARG) para permitir

utilização eficiente do índice. Capítulo 15, "Implementando índices e estatísticas", entra em detalhes

sobre a indexação e o uso de argumentos de pesquisa; aqui, vou descrever brevemente o conceito

e fornecer exemplos simples.

Um predicado na *coluna operador* *valor do operador coluna* forma ou *valor* pode ser uma busca

argumento. Por exemplo, predicados como col1 = 10, e col1> 10 são argumentos de pesquisa. Aplicando

manipulação para a coluna filtrada na maioria dos casos, impede o predicado de ser um

argumento de pesquisa. Um exemplo para a manipulação da coluna filtrada é a aplicação de uma função

para isso, como em F (COL1) = 10, em que F é uma função. Existem algumas excepções a esta regra, mas

eles são muito raros.

Por exemplo, suponha que você tem um procedimento armazenado que aceita um parâmetro de entrada @dt

representando uma data de entrada enviado. O procedimento deve retornar ordens que eram

enviado na data de entrada. Se a coluna DataDeEnvio não permitir nulos, você pode usar o

consulta a seguir para resolver esta tarefa.

Orderid SELECT, orderdate, empid

DE Sales.Orders

ONDE DataDeEnvio = @dt;

No entanto, a coluna DataDeEnvio não permite nulos; aqueles que representam as ordens que não eram

ainda enviado. Quando os usuários precisarão de todas as ordens que ainda não foram enviados, os usuários fornecerá

um NULL como a data de entrada enviado, e sua consulta teria de ser capaz de lidar com uma tal

caso. Lembre-se que quando se comparam dois nulos, você começa desconhecido e a linha é filtrada

Fora. Assim, a forma atual do predicado não aborda entradas NULL corretamente. alguns endereço

esta necessidade, utilizando COALESCE ou ISNULL para substituir nulos com um valor que não existe na

os dados normalmente, como a seguir.

Orderid SELECT, orderdate, empid

DE Sales.Orders

ONDE COALESCE (DataDeEnvio, '19000101') = COALESCE (@dt, '19000101');

O problema é que mesmo que a solução agora retorna o correto resultado, mesmo quando

a entrada é NULL-o predicado não é um argumento de pesquisa. Isto significa que o SQL Server não pode

eficiente usar um índice na coluna DataDeEnvio. Para fazer com que o predicado de um argumento de pesquisa,

é preciso evitar a manipulação da coluna filtrada e reescrever o predicado como o

Segue.

Orderid SELECT, orderdate, empid

DE Sales.Orders

ONDE DataDeEnvio = @dt

OR (DataDeEnvio é nulo e @dt IS NULL);

*Dica exam*

Compreender o impacto da utilização de COALESCE e ISNULL no desempenho é um importante

habilidade para o exame.

Curiosamente, padrão SQL tem um predicado chamado não é distinta de que tem o

mesmo significado que o predicado usado na última consulta (return true quando ambos os lados são iguais

ou quando ambas são nulos, caso contrário false). Infelizmente, T-SQL não suporta este predicado.

Outro exemplo para a manipulação envolve a coluna filtrada numa expressão; para

exemplo, col1 - 1 <= @ n. Às vezes, você pode reescrever o predicado a um formulário que é uma busca

argumento, e então permitir o uso eficiente de indexação. O último predicado, por exemplo, pode ser

reescrito usando a matemática simples como col1 <= @ n + 1.

Em resumo, quando um predicado envolve a manipulação da coluna filtrada, e existem

formas alternativas para a frase que sem a manipulação, você pode aumentar a probabilidade de

utilização eficiente de indexação. Há um par de exemplos adicionais nas seções "Filtragem

Dados de caráter "e" Data de filtragem e de dados Time "mais adiante neste capítulo. E, como mencionado,

cobertura mais ampla do tema é no capítulo 15.

combinando Predicados

Você pode combinar predicados na cláusula WHERE usando os operadores lógicos AND e

OU. Você também pode negar predicados usando o operador NOT lógico. Esta secção começa por

descrever aspectos importantes da negação e, em seguida, discute combinar predicados.

Negação do verdadeiro e falso é simples-NOT verdade é falsa, e não falso é verdadeiro.

O que pode ser surpreendente para alguns é o que acontece quando você nega desconhecida-NOT desconhecido

ainda é desconhecida.

Lembre-se de mais cedo neste capítulo a consulta que retornou todos os empregados a partir de Washington

Estado; a consulta usada região do predicado = N'WA 'na cláusula WHERE. Suponha que

que deseja retornar os funcionários que não são da WA, e para isso você usa o predicado

NÃO region = N'WA '. É evidente que os casos que retornam falso do predicado positivo (dizem que o

região é NY) retorno verdadeiro do predicado negada. É também evidente que os casos que retornam true

do predicado positivo (por exemplo a região é WA) return false do predicado negada.

No entanto, quando a região é NULL, tanto o predicado positivo eo negado um retorno

desconhecido e a linha é descartado. Então, o caminho certo para você incluir casos NULL no

resultado-

se é isso que você sabe que você precisa fazer-se usar o operador nulo é, como em

NÃO region = N'WA 'ou região for NULL.

Como para a combinação de predicados, há várias coisas interessantes a serem observados. alguns precedência

regras determinam a ordem de avaliação lógica dos diferentes predicados. O operador NOT

precede AND e OR, e AND precede OR. Por exemplo, suponha que o filtro WHERE na

sua consulta teve a seguinte combinação de predicados.

ONDE = col1 'w' e col2 = 'x' OR = col3 'y' E = Col4 'z'

Porque e precede OR, você obtém o equivalente do seguinte.

WHERE (col1 = 'w' e col2 = 'X') OR (= col3 'y' E col4 = 'z')

Tentando expressar os operadores como funções pseudo, essa combinação de operadores é

equivalente ou (E (col1 = 'w', = col2 'x'), E (= col3 'y', = Col4 'z')).

Porque parênteses têm a prioridade mais alta entre todos os operadores, você pode sempre

use-os para controlar totalmente a ordem de avaliação lógico que você precisa, como o exemplo a seguir

mostra.

ONDE col1 = 'w' E (= col2 'x' OR col3 = 'y') E col4 = 'z'

Mais uma vez, usando funções pseudo, esta combinação de operadores e uso de parênteses é

equivalente a E (= col1 'w', OR (= col2 'x', = col3 'y'), = Col4 'z').

Lembre-se do capítulo 1 que todas as expressões que aparecem na mesma lógica de processamento de consulta

fase, por exemplo, o ONDE fase são conceitualmente avaliada no mesmo ponto em

Tempo. Por exemplo, considere o seguinte predicado de filtro.

ONDE PropertyType = 'INT' E CAST (propertyval AS INT)> 10

Suponha que a tabela que está sendo consultado detém diferentes valores de propriedade. o PropertyType

coluna representa o tipo da propriedade (um INT, uma data e assim por diante), eo propertyval

coluna contém o valor em uma cadeia de caracteres. Quando PropertyType é 'INT', o valor no

propertyval é conversível para INT; caso contrário, não necessariamente.

Alguns assumem que a menos que as regras de precedência ditar o contrário, predicados serão avaliados

da esquerda para a direita, e que curto-circuito ocorrerá quando possível. Em outras palavras, se o

primeiro predicado PropertyType = 'int' avaliada como falsa, SQL Server não irá avaliar a segunda

ELENCO predicado (propertyval AS INT)> 10 porque o resultado já é conhecido. Com base nisso

pressuposto, a expectativa é que a consulta nunca deve falhar tentando converter algo

que não é conversível.

A realidade, porém, é diferente. SQL Server suporta internamente um conceito de curto-circuito;

no entanto, devido à toda de uma vez conceito na língua, não é necessariamente vai

avaliar as expressões na ordem da esquerda para a direita. Pode decidir, com base em razões relacionadas com os custos,

começar com a segunda expressão, e então, se a segunda expressão é avaliada como verdadeira,

para avaliar a primeira expressão bem. Isto significa que se houver linhas na tabela onde

PropertyType é diferente do 'INT', e nessas linhas propertyval não é conversível para int, o

consulta pode falhar devido a um erro de conversão.

Pode lidar com este problema em um número de maneiras. Uma opção é simples de usar o TRY\_CAST

funcionar em vez de CAST. Quando a expressão de entrada não é conversível para o tipo de destino,

TRY\_CAST retorna um NULL em vez de falha. E comparando um NULL a nada produz desconhecido.

Eventualmente, você vai obter o resultado correto, sem permitir a consulta a falhar. Então seu

Cláusula WHERE devem ser revistas como a seguinte.

ONDE PropertyType = 'INT' E TRY\_CAST (propertyval AS INT)> 10

Filtragem de dados de caracteres

Em muitos aspectos, a filtragem de dados de caracteres, é a mesma que a filtragem de outros tipos de dados. este

seção abrange um par de itens que são específicos para dados de caracteres: a forma adequada de literais e

o predicado LIKE.

Como discutido no Capítulo 2, "Introdução à instrução SELECT", um literal tem um tipo.

Se você escrever uma expressão que envolve operandos de diferentes tipos, SQL Server terá que

aplicar conversão implícita para alinhar os tipos. Em função das circunstâncias, as conversões implícitas

por vezes, pode prejudicar o desempenho. É importante saber a forma adequada de literais de

diferentes tipos e certifique-se de usar os corretos. Um exemplo clássico para a utilização incorrecta

tipos literais é com Unicode cadeias de caracteres (tipos NVARCHAR e NCHAR). A forma correta

para uma cadeia de caracteres Unicode literal é anteceder o literal com um N de capital e delimitar o

literal com aspas simples; por exemplo, N'literal '. Para uma cadeia de caracteres regulares literal,

você apenas delimitar o literal com aspas simples; por exemplo, "literal". É um típico

mau hábito para especificar uma cadeia de caracteres regulares literal quando a coluna filtrada é de um Unicode

tipo, como no exemplo a seguir.

Empid SELECT, nome, sobrenome

DE HR.EMPLOYEES

= ONDE Apelido 'Davis';

Como a coluna eo literal têm tipos diferentes, SQL Server converte um implicitamente

tipo de operando para o outro. Neste exemplo, felizmente, SQL Server converte o tipo do literal

o tipo da coluna, por isso ainda pode contar com eficiência na indexação. No entanto, pode haver casos

onde a conversão implícita prejudica o desempenho. É uma prática recomendada para usar o formulário apropriado, como

na sequência.

Empid SELECT, nome, sobrenome

DE HR.EMPLOYEES

ONDE sobrenome = N'Davis ';

T-SQL fornece o predicado LIKE, que você pode usar para filtrar dados de cadeia de caracteres (regular

e Unicode) com base na correspondência de padrões. A forma de um predicado usando COMO é como se segue.

<column> COMO <pattern>

O predicado LIKE suporta curingas que você pode usar em seus padrões. Tabela 3-1 descreve

os wildcards disponíveis, o seu significado e um exemplo demonstrando sua utilização.

**Tabela 3-1** Wildcards usados ​​em padrões COMO

**Exemplo Significado Wildcard**

% (Sinal de porcentagem) Qualquer cadeia incluindo

um vazio

'D%': string começando com D

\_ (Sublinhado) Um único caractere '\_D%': string, onde segundo personagem é D

*[<Lista de caracteres>]* Um único caractere

partir de uma lista

'[AC]%': string onde o primeiro caractere é A ou C

*[<Intervalo de caracteres>]* Um único caractere

a partir de um intervalo

'[0-9]%': string onde o primeiro caractere é um dígito

[^ *<Lista de caracteres ou intervalo>]* Um único caractere que

não está na lista ou

alcance

'[^ 0-9]%': string onde o primeiro caractere não é um dígito

Como exemplo, suponha que você deseja retornar todos os funcionários cujo sobrenome começa com a

letra D. Você usaria a seguinte consulta.

Empid SELECT, nome, sobrenome

DE HR.EMPLOYEES

ONDE sobrenome GOSTO N'D% ';

Essa consulta retorna o seguinte resultado.

empid firstname lastname

------ ---------- -------------

1 Sara Davis

9 Zoya Dolgopyatova

Se você quer olhar para um personagem que é considerado um curinga, você pode indicá-lo depois

um personagem que você designar como um caractere de escape, utilizando a palavra-chave ESCAPE. Por exemplo,

a expressão col1 LIKE '! \_%' ESCAPE '!' procura por strings que começam com um sublinhado

(\_) Usando um ponto de exclamação (!) Como o caractere de escape.

**Desempenho** *importante* **do predicado LIKE**

Quando o padrão LIKE inicia com um exemplo de prefixo-for conhecido, col LIKE 'ABC%' -

SQL Server pode potencialmente eficiente usar um índice na coluna filtrada; em outras palavras,

SQL Server pode confiar no índice de ordenação. Quando o padrão começa com um exemplo wildcard-for,

col LIKE '% ABC%' - SQL Server não pode contar com índice de encomendar mais. Além disso, quando

à procura de uma cadeia que começa com um prefixo conhecido (por exemplo, ABC) certifique-se de usar o COMO

predicado, como em col LIKE 'ABC%', porque esta forma é considerado um argumento de pesquisa. recordação

que a aplicação de manipulação para a coluna filtrada impede o predicado de ser um

argumento de pesquisa. Por exemplo, o LEFT forma (col, 3) = 'ABC' não é um argumento de pesquisa e

impedirá SQL Server de ser capaz de usar um índice de forma eficiente.

Filtragem de dados de data e hora

Há várias considerações importantes quando a data e os dados de tempo de filtragem que estão relacionados

tanto para a correção de seu código e ao seu desempenho. Você quer pensar em coisas como

como expressar literais, intervalos de filtros e argumentos utilização de busca.

Vou começar com literais. Suponha que você precisa consultar a tabela de Sales.Orders e regresso

apenas os pedidos feitos em 12 de fevereiro de 2007. Você pode usar a seguinte consulta.

Orderid SELECT, orderdate, empid, custid

DE Sales.Orders

ONDE orderdate = '02 / 12/07 ';

Se você é um americano, essa forma provavelmente significa 12 de fevereiro de 2007, a você. No entanto, se

você é britânico, esta forma provavelmente significa 2 de Dezembro de 2007. Se você é japonesa, provavelmente

significa 7 de dezembro de 2002. A questão é, quando o SQL Server converte esta cadeia de caracteres a uma

data e tipo de tempo para alinhá-lo com o tipo da coluna filtrada, como é interpretar o valor?

Como se vê, isso depende da linguagem do logon que executa o código. Cada logon tem um

idioma padrão associado a ele, e o idioma padrão define várias opções de sessão em

nome do logon, incluindo um chamado DATEFORMAT. Um logon com us\_english terá a

DATEFORMAT configuração definida para mdy, British para DMY, e japonês a AMD. O problema é, como

você como um desenvolvedor de expressar uma data, se você quer que ele seja interpretado da maneira desejada,

independentemente de quem executa seu código?

Existem duas abordagens principais. Uma é usar um formulário que é considerado linguagem neutra.

Por exemplo, a forma "20070212" é sempre interpretado como ymd, independentemente de seu idioma.

Observe que o formulário '2007-02-12' é considerado linguagem neutra apenas à data em tipos de dados,

DATETIME2, e DATETIMEOFFSET. Infelizmente, devido a razões históricas, esta forma é considerada

para o DATETIME tipos e SMALLDATETIME dependentes do idioma. A vantagem

o formulário sem os separadores é que é idioma neutro para todos os tipos de data e hora. assim

a recomendação é para escrever a consulta como a seguinte.

Orderid SELECT, orderdate, empid, custid

DE Sales.Orders

ONDE orderdate = "20070212";

*Nota* **Armazenando datas em uma coluna DATETI ME**

O orderdate coluna filtrada é de um tipo de dados DATETIME representando tanto data e hora.

No entanto, o literal especificado no filtro contém apenas uma parte da data. Quando o SQL Server converte a

literal para o tipo da coluna filtrada, ele assume a meia-noite quando uma parte do tempo não é indicado.

Se você quer um tal filtro para retornar todas as linhas a partir da data especificada, você precisa garantir que

você armazenar todos os valores com a meia-noite como o tempo.

Outra abordagem é usar as funções de converter ou analisar, que você pode usar para indicar

como você deseja SQL Server para interpretar a literal que você especificar. A função CONVERT

suporta um número de estilo representando o estilo de conversão, e os suportes função de análise

indicando um nome de cultura. Você pode encontrar detalhes sobre as duas funções no Capítulo 2.

Outro aspecto importante da data de filtragem e de dados em tempo está a tentar sempre que possível

usar argumentos de pesquisa. Por exemplo, suponha que você precisa para filtrar apenas os pedidos feitos em

Fevereiro de 2007. Você pode usar as funções de ano e mês, como no seguinte.

Orderid SELECT, orderdate, empid, custid

DE Sales.Orders

Onde no ano (orderdate) = 2007 E MÊS (orderdate) = 2;

No entanto, porque aqui você aplicar manipulação para a coluna filtrada, o predicado é

não é considerado um argumento de pesquisa e, portanto, o SQL Server não será capaz de confiar em índice

encomenda. Você pode rever o seu predicado como uma gama, como o seguinte.

Orderid SELECT, orderdate, empid, custid

DE Sales.Orders

ONDE orderdate> = '20070201' E orderdate < '20070301';

Agora que você não aplicar a manipulação para a coluna filtrada, o predicado é considerado

um argumento de pesquisa, e não há o potencial para SQL Server para confiar no índice de ordenação.

Se você está se perguntando por que este código expressa o intervalo de datas usando maior ou igual

a (> =) e menos de operadores (<) em vez de usar ENTRE, há uma razão para isso.

Quando você estiver usando ENTRE ea coluna mantem ambos os elementos de data e hora, o que fazer

você usar como o valor final? Como você pode perceber, para diferentes tipos, existem diferentes precisões.

Além do mais, suponha que o tipo é DATETIME, e você usar o seguinte predicado.

ONDE orderdate ENTRE '20070201' E '20070228 23: 59: 59.999'

Precisão deste tipo é de três e um terceiro milissegundos. A milissegundos parte do fim

ponto 999 não é uma multiplicação da unidade de precisão, portanto, SQL Server acaba arredondando o valor

à meia-noite de 1 de Março de 2007. Como resultado, você pode acabar ficando algumas ordens que você está

não deveria ver. Em suma, em vez de ENTRE, use> = e <, e esta forma irá funcionar corretamente

em todos os casos, com todos os tipos de data e hora, se a parte do tempo é aplicável ou não.

**Checagem rápida**

1. Quais são os benefícios de desempenho em usar o filtro ONDE?

2. Qual é a forma de um predicado de filtro que podem contar com índice de ordenação chamado?

**Quick Check Resposta**

1. Você reduzir o tráfego de rede por filtração, o servidor de banco de dados em vez de no

o cliente, e você pode potencialmente usar índices para evitar análises completas das tabelas

envolvidos.

2. Um argumento de pesquisa, ou SARG, para breve.

Prática **Filtrando dados com predicados**

Nesta prática, você exercer o seu conhecimento de filtragem de dados com predicados.

Se você encontrar um problema de completar um exercício, você pode instalar os projectos concluídos

a partir da pasta Solution que é fornecido com o conteúdo complementar para este capítulo e

lição.

Exercício 1 Use a cláusula WHERE para filtrar linhas com NULL

Neste exercício, você pratica o uso da cláusula WHERE para filtrar as ordens não enviados a partir do

Mesa Sales.Orders.

1. SSMS aberto e conectar-se à TSQL2012 base de dados exemplo.

2. Você está convidado a escrever uma consulta que retorna os pedidos que ainda não foram enviados. Tal

encomendas têm um NULL na coluna DataDeEnvio. Para sua primeira tentativa, use o seguinte

consulta.

Orderid SELECT, orderdate, custid, empid

DE Sales.Orders

ONDE DataDeEnvio = NULL;

No entanto, quando você executar esse código, você recebe um conjunto de resultados vazio.

orderid orderdate custid empid

----------- ----------------------- ----------- ----- ------

A razão para isto é que, quando a expressão compara dois nulos, o resultado é desconhecido,

e a linha é filtrado.

3. Rever o predicado de filtro para usar o operador IS NULL em vez de e qualidade (=), como em t ele

Segue.

Orderid SELECT, orderdate, custid, empid

DE Sales.Orders

ONDE DataDeEnvio IS NULL;

Desta vez, você recebe o resultado correto, mostrado aqui na forma abreviada.

orderid orderdate custid empid

----------- ----------------------- ----------- ----- ------

11008 2008-04-08 00: 00: 00.000 20 7

11019 2008-04-13 00: 00: 00.000 64 6

11039 2008-04-21 00: 00: 00.000 47 1

Exercício 2 Use a cláusula WHERE para filtrar um intervalo de datas

Neste exercício, você pratica o uso da cláusula WHERE para filtrar as ordens dentro de um determinado

intervalo de datas da tabela Sales.Orders.

1. Você está convidado a retornar todos os pedidos que foram colocados entre 11 de fevereiro de 2008

e 12 de Fevereiro de 2008. A coluna orderdate você deveria filtrar por é de um

Tipo de data e hora. Com os dados atuais da tabela, todos os valores DataDaEncomenda tem o tempo

definido para a meia-noite, mas suponho que isso não era o caso, ou seja, que a parte do tempo possível

ser um valor diferente de meia-noite. Para sua primeira tentativa, use o predicado BETWEEN, como

segue.

Orderid SELECT, orderdate, custid, empid

DE Sales.Orders

ONDE orderdate ENTRE '20080211' E '20080212 23: 59: 59.999';

Porque 999 não é uma multiplicação de unidade de precisão do tipo DATETIME (três e

milissegundos terceiros), o valor final na gama fica arredondado para a próxima meia-noite, e

o resultado inclui linhas de 13 de fevereiro que você não pedir.

orderid orderdate custid empid

----------- ----------------------- ----------- ----- ------

10881 2008-02-11 00: 00: 00.000 12 4

10887 2008-02-13 00: 00: 00.000 29 8

10886 2008-02-13 00: 00: 00.000 34 1

10884 2008-02-12 00: 00: 00.000 45 4

10883 2008-02-12 00: 00: 00.000 48 8

10882 2008-02-11 00: 00: 00.000 71 4

10885 2008-02-12 00: 00: 00.000 76 6

2. Para corrigir o problema, rever o filtro de intervalo para usar o> = <e operadores, como se segue.

Orderid SELECT, orderdate, custid, empid

DE Sales.Orders

ONDE orderdate> = '20080211' E orderdate < '20080213';

Desta vez, você obter o resultado correto.

Resumo da lição

■■ Com a cláusula WHERE, você pode filtrar dados usando predicados. Predicados em uso T-SQL

lógica de três valores. A cláusula WHERE retorna casos em que o predicado avaliar para

verdadeira e descarta o resto.

■■ de dados de filtragem usando a cláusula WHERE ajuda a reduzir o tráfego de rede e pode potencialmente

permitir o uso de índices para minimizar I / O. É importante tentar frase o seu

predicados como argumentos de busca para permitir a utilização eficiente dos índices.

■■ Ao filtrar diferentes tipos de dados, como caráter e dados de data e tempo, é

importante estar familiarizado com as melhores práticas que irão garantir que você escrever ambos corretos

e código eficiente.

Examine a lição

Responda as seguintes perguntas para testar seu conhecimento da informação nesta lição. Vocês

pode encontrar as respostas para estas perguntas e explicações de por que cada opção de resposta está correta

ou incorrecta na secção "Respostas" no final deste capítulo.

1. O que a lógica do prazo de três valores referem-se em T-SQL?

A. Os três valores possíveis resultado lógico de um predicado: verdadeiro, falso, e NULL

B. Os três valores possíveis resultado lógico de um predicado: verdadeiro, falso, e desconhecida

C. Os três valores possíveis resultado lógico de um predicado: 1, 0 e NULL

D. Os três valores possíveis resultado lógico de um predicado: -1, 0 e 1

2. Qual das seguintes literais são para o tipo de dados DATETIME dependente de idioma?

(Escolha tudo o que se aplicam).

A. '2012-02-12'

B. '02 / 12/2012 '

C. '12 / 02/2012 '

D. "20120212"

3. Qual dos seguintes predicados são argumentos de busca? (Escolha tudo o que se aplicam).

A. DIA (orderdate) = 1

B. companyname LIKE 'A%'

'% A%' C. companyname COMO

D. companyname LIKE '% A'

E. orderdate> = '20120212' E orderdate < '20120213'

**Lição 2: Classificação de dados**

Classificação de dados é suposto ser uma coisa trivial, mas como se vê, é uma fonte de muita confusão

em T-SQL. Esta lição descreve a diferença crítica em T-SQL entre não triados e

dados classificados. Em seguida, ele descreve as ferramentas T-SQL fornece-lhe ordenar os dados.

Após esta lição, você será capaz de:

■■ Use a cláusula ORDER BY para determinar a ordem de linhas em resultado de uma

consulta.

■■ Descreva a diferença entre uma consulta com e sem uma ORDER BY

cláusula.

■■ ascendente Controle e descendente direção de ordenação.

■■ Siga encomendar as melhores práticas.

■■ Identificar encomendar restrições quando DISTINCT é usado.

■■ Ordenar por aliases que foram atribuídos na cláusula SELECT.

Tempo de aula estimado: 30 minutos

Compreender Quando a ordem é garantido

Provavelmente um dos aspectos mais confusos de trabalhar com T-SQL é entender quando

um resultado de consulta é garantido para ser devolvido em ordem particular versus quando não é. Corrigir

compreensão deste aspecto das relações de linguagem diretamente para as fundações de T-SQL-

teoria dos conjuntos particularmente matemática. Se você entender isso desde os primeiros estágios de

escrever o código T-SQL, você terá um tempo muito mais fácil do que muitos que simplesmente tem incorreta

suposições e expectativas da língua.

Considere a seguinte consulta como um exemplo.

Empid SELECT, nome, sobrenome, cidade, MÊS (data de nascimento) AS birthmonth

DE HR.EMPLOYEES

ONDE country = N'USA 'E região = N'WA';

Existe uma garantia de que as linhas serão retornados em ordem particular, e em caso afirmativo, o que é

essa ordem?

Alguns fazer uma suposição intuitiva que as linhas serão retornados em ordem de inserção; alguns

assumir a fim de chave primária; alguns supõem fim índice de cluster; outros sabem que não há nenhuma

garantir para qualquer tipo de ordem.

Se você se lembrar do capítulo 1, uma tabela em T-SQL é suposto representar uma relação; uma relação

é um conjunto e um conjunto não tem fim de seus elementos. Com isto em mente, a menos que você instruir explicitamente

a consulta de outra forma, o resultado de uma consulta não tem qualquer ordem garantida. Por exemplo, esta consulta

deu o seguinte resultado quando executado em um sistema.

empid firstname birthmonth cidade sobrenome

------ ---------- --------- --------- -----------

1 Sara Davis Seattle 12

2 Don Funk Tacoma 2

3 Judy Lew Kirkland 8

4 Yael Peled Redmond 9

8 Maria Cameron Seattle 1

Pode parecer que a saída é classificada por empid, mas isso não é garantido. O que poderia

ser mais confuso é que, se você executar a consulta várias vezes, parece que o resultado continua sendo

devolvido na mesma ordem; mas, novamente, isso não é garantido. Quando o motor de banco de dados (SQL

Servidor, neste caso) processa essa consulta, ele sabe que pode retornar os dados em qualquer ordem, porque

não há nenhuma instrução explícita para retornar os dados em uma ordem específica. Pode ser que, devido

a otimização e outras razões, o motor de banco de dados SQL Server escolheu para processar os dados

de modo particular *neste momento*. Há ainda alguma probabilidade de que essas escolhas vão ser repetido se

as circunstâncias físicas permanecem o mesmo. Mas há uma grande diferença entre o que é provável

a acontecer devido à otimização e outras razões e que está realmente garantido.

O motor de banco de dados pode e às vezes faz-mudar escolhas que podem afetar a ordem

em que as linhas são devolvidas, sabendo que ela é livre para fazê-lo. Exemplos de tais mudanças na

escolhas incluem mudanças na distribuição de dados, a disponibilidade de estruturas físicas, tais como índices,

e disponibilidade de recursos como CPUs e memória. Além disso, com as mudanças no motor depois

um upgrade para uma versão mais recente do produto, ou mesmo após a aplicação de um service pack,

aspectos de otimização podem mudar. Por sua vez, estas alterações podem afectar, entre outras coisas, o

ordem das linhas no resultado.

Em suma, este não pode ser suficiente sublinhado: Uma consulta que não tem uma instrução explícita

para retornar as linhas em uma ordem específica não garante a ordem das linhas no resultado.

Quando você precisa de uma tal garantia, a única maneira de fornecê-la é através da adição de um ORDER BY

cláusula para a consulta, e que é o foco da próxima seção.

Usando a cláusula ORDER BY para classificar dados

A única maneira de garantir verdadeiramente que as linhas são retornadas de uma consulta em uma determinada ordem é

adicionando uma cláusula ORDER BY.

Por exemplo, se você quiser retornar informações sobre os empregados do estado de Washington

nos Estados Unidos, classificado por cidade, você especificar a cidade coluna na cláusula ORDER BY como

segue.

empid SELECT, nome, sobrenome, cidade, MÊS (data de nascimento) AS birthmonth

DE HR.EMPLOYEES

ONDE country = N'USA 'E região = N'WA'

ORDER BY cidade;

Aqui está a saída dessa consulta.

empid firstname birthmonth cidade sobrenome

------ ---------- --------- --------- -----------

3 Judy Lew Kirkland 8

4 Yael Peled Redmond 9

8 Maria Cameron Seattle 1

1 Sara Davis Seattle 12

2 Don Funk Tacoma 2

Se você não indicar uma direção para a classificação, ordem crescente é assumido por padrão. Vocês

pode ser explícita e especificar ASC cidade, mas isso significa a mesma coisa que não indicando a direção.

Para descendente de ordenação, você precisa especificar explicitamente DESC, como se segue.

empid SELECT, nome, sobrenome, cidade, MÊS (data de nascimento) AS birthmonth

DE HR.EMPLOYEES

ONDE country = N'USA 'E região = N'WA'

ORDER BY DESC cidade;

Desta vez, a saída mostra os registros na ordem cidade, descendente direção.

empid firstname birthmonth cidade sobrenome

------ ---------- --------- --------- -----------

2 Don Funk Tacoma 2

1 Sara Davis Seattle 12

8 Maria Cameron Seattle 1

4 Yael Peled Redmond 9

3 Judy Lew Kirkland 8

A cidade coluna não é exclusivo dentro do país e região filtrada, e, portanto, o

ordenação das linhas com a mesma cidade (ver Seattle, por exemplo) não é garantida. Nesse caso,

diz-se que a ordem não é determinista. Assim como uma consulta sem uma cláusula ORDER BY

não garante a ordem entre linhas do resultado, em geral, uma consulta com ORDER BY cidade, quando da cidade

não é única, não garante a ordem entre linhas com o mesmo cidade. Felizmente, você pode

especificar várias expressões na lista ORDER BY, separados por vírgulas. Um caso de uso deste

capacidade é para aplicar um desempate para encomendar. Por exemplo, você poderia definir como o empid

coluna de classificação secundária, como se segue.

empid SELECT, nome, sobrenome, cidade, MÊS (data de nascimento) AS birthmonth

DE HR.EMPLOYEES

ONDE country = N'USA 'E região = N'WA'

ORDER BY cidade, EmpID;

Aqui está a saída dessa consulta.

empid firstname birthmonth cidade sobrenome

------ ---------- --------- --------- -----------

3 Judy Lew Kirkland 8

4 Yael Peled Redmond 9

1 Sara Davis Seattle 12

8 Maria Cameron Seattle 1

2 Don Funk Tacoma 2

A lista ORDER BY é agora única; Assim, a ordenação é determinista. Enquanto o subjacente

Os dados não mudar, os resultados são garantidos para poder ser repetido, para além da sua

ordenação apresentação. Você pode indicar a direção de ordenação em um-por-expressão expressão

base, como no ORDER BY DESC col1, col2, DESC col3 (descendente col1, ascendente, em seguida, col2, em seguida,

col3 descendente).

Com T-SQL, você pode classificar por posições ordinais das colunas na lista SELECT, mas é considerado

uma prática ruim. Considere a seguinte consulta como um exemplo.

empid SELECT, nome, sobrenome, cidade, MÊS (data de nascimento) AS birthmonth

DE HR.EMPLOYEES

ONDE country = N'USA 'E região = N'WA'

ORDER BY 4, 1;

Nesta consulta, você está pedindo para ordenar as linhas por a quarta expressão na lista SELECT

(Da cidade), e, em seguida, pela primeira (empid). Nesta consulta particular, é equivalente a usar ORDER

Por cidade, EmpID. No entanto, esta prática é considerado um mau para um número de razões. Para

um, T-SQL faz manter o controle de posições ordinais das colunas em uma tabela, além de uma consulta

resultado, mas este é não relacional. Lembre-se que o cabeçalho de uma relação é um conjunto de atributos, e

um conjunto não tem fim. Além disso, quando você estiver usando posições ordinais, é muito fácil depois de fazer

alterações na lista SELECT para perder mudando os ordinais em conformidade. Por exemplo, suponha

que você decidir aplicar mudanças à sua consulta anterior, voltando da cidade logo após empid em

a lista SELECT. Você aplicar a alteração à lista SELECT, mas se esqueça de alterar o ORDER BY

lista em conformidade, e acabar com a seguinte consulta.

SELECIONAR empid, cidade, nome, sobrenome, MÊS (data de nascimento) AS birthmonth

DE HR.EMPLOYEES

ONDE country = N'USA 'E região = N'WA'

ORDER BY 4, 1;

Agora, a consulta é ordenar os dados por sobrenome e EmpID em vez de cidade e empid.

Em suma, é uma boa prática para se referir a nomes de colunas ou expressões com base nestes, e não

a posições ordinais.

Note que você pode encomendar as linhas do resultado por elementos que você não está voltando. Por exemplo,

os seguintes retornos de consulta, para cada funcionário de qualificação, a identificação do funcionário e da cidade,

ordenar as linhas de resultados pela data de nascimento do empregado.

empid SELECT, cidade

DE HR.EMPLOYEES

ONDE country = N'USA 'E região = N'WA'

ORDER BY data de nascimento;

Aqui está a saída dessa consulta.

cidade empid

----------- ---------------

4 Redmond

1 Seattle

2 Tacoma

8 Seattle

3 Kirkland

Claro, o resultado parece muito mais significativo se você incluiu a data de nascimento

atributo, mas se faz sentido para você não, é perfeitamente válido. A regra é, você pode encomendar

as linhas do resultado por elementos que não fazem parte da lista SELECT, desde que as linhas do resultado

teria sido normalmente permitido lá. Esta regra muda quando a cláusula DISTINCT também é

especificada e por uma boa razão. Quando DISTINCT é usado, as duplicatas são removidos; então o

linhas de resultados não necessariamente mapear para linhas de origem em um one-to-one forma, ao invés de oneto-

muitos. Por exemplo, tentar razão pela qual a seguinte consulta não é válido.

cidade SELECT DISTINCT

DE HR.EMPLOYEES

ONDE country = N'USA 'E região = N'WA'

ORDER BY data de nascimento;

Você pode ter vários empregados de cada um com um nascimento diferente data-da mesma cidade.

Mas você está retornando apenas uma linha para cada cidade distinta no resultado. Assim, dada uma cidade (por exemplo,

Seattle) com vários funcionários, que do empregado datas de nascimento deve aplicar-se como o

ordenando valor? A consulta não basta escolher um; em vez disso, simplesmente falha.

Assim, caso a cláusula DISTINCT é usado, você está limitado na lista ORDER BY para únicos elementos

que aparecem na lista SELECT, como no seguinte consulta.

cidade SELECT DISTINCT

DE HR.EMPLOYEES

ONDE country = N'USA 'E região = N'WA'

ORDER BY cidade;

Agora, a consulta é perfeitamente sensato, retornando o seguinte resultado.

cidade

---------

Kirkland

Redmond

Seattle

Tacoma

O que também é interessante notar sobre a cláusula ORDER BY é que ele é avaliado conceitualmente

após as maioria das outras cláusulas de consulta cláusula, ao contrário SELECT. Isto significa que a coluna

aliases atribuídos na cláusula SELECT são realmente visível a cláusula ORDER BY. como

exemplo, a consulta a seguir usa a função mês para devolver o mês de nascimento, atribuindo

a expressão com o birthmonth coluna alias. A consulta, em seguida, refere-se ao apelido de coluna

birthmonth diretamente na cláusula ORDER BY.

empid SELECT, nome, sobrenome, cidade, MÊS (data de nascimento) AS birthmonth

DE HR.EMPLOYEES

ONDE country = N'USA 'E região = N'WA'

ORDER BY birthmonth;

Essa consulta retorna o seguinte resultado.

empid firstname birthmonth cidade sobrenome

------ ---------- --------- --------- -----------

8 Maria Cameron Seattle 1

2 Don Funk Tacoma 2

3 Judy Lew Kirkland 8

4 Yael Peled Redmond 9

1 Sara Davis Seattle 12

Outro aspecto complicado de ordenação é tratamento de nulos. Lembre-se que um NULL representa uma

valor em falta, então quando se compara um NULL a qualquer coisa, você recebe o resultado lógico desconhecido.

Esse é o caso mesmo quando se comparam dois nulos. Portanto, não é tão trivial para perguntar como nulos

deve se comportar em termos de classificação. Eles devem todo o tipo juntos? Se assim for, que eles deveriam espécie antes

ou depois de valores não-nulo? Padrão SQL diz que nulos deve classificar juntos, mas deixa aos

a implementação para decidir se a classificá-los antes ou depois de valores não nulos. em SQL

Servidor a decisão foi para classificá-los antes não-nulos (quando se utiliza um sentido ascendente).

Como exemplo, os seguintes retornos de consulta para cada pedido o código do pedido e data de envio,

ordenados por este último.

orderid SELECT, DataDeEnvio

DE Sales.Orders

ONDE custid = 20

ORDER BY DataDeEnvio;

Lembre-se que as ordens não enviados têm um NULL na coluna DataDeEnvio; Assim, eles

tipo antes que os pedidos enviados, como mostra o resultado da consulta.

orderid DataDeEnvio

----------- -----------------------

11008 NULL

11072 NULL

10258 2006/07/23 Premium 00: 00: 00.000

10263 2006-07-31 00: 00: 00.000

10351 2006-11-20 00: 00: 00.000

...

Padrão SQL suporta as opções nulos FIRST e LAST NULLS para controlar como nulos

tipo, mas T-SQL não suporta essa opção. Como um desafio interessante, veja se você pode descobrir

para fora como para classificar os pedidos por data enviados ascendente, mas tem nulos tipo passado. (Dica: você

pode especificar expressões na cláusula ORDER BY; pensar em como usar a expressão CASE para

realizar esta tarefa.)

Então lembre-se, uma consulta sem uma cláusula ORDER BY retorna um resultado relacional (pelo menos do

uma perspectiva de encomenda), e, portanto, não garante qualquer ordem. A única maneira de garantir

ordem é com uma cláusula ORDER BY. De acordo com o padrão SQL, uma consulta com um ORDER BY

cláusula conceitualmente retorna um *cursor*e não uma relação.

A indexação é discutido mais tarde no kit de treinamento, mas por agora, basta dizer que a criação de

os índices certos pode ajudar a SQL Server evitar a necessidade de realmente classificar os dados para resolver um

ORDER BY pedido. Sem bons índices, SQL Server precisa para classificar os dados e classificação

pode ser caro, especialmente quando um grande conjunto é envolvido. Se você não precisa retornar os dados

classificadas, certifique-se de não especificar uma cláusula ORDER BY, para evitar custos desnecessários.

**Checagem rápida**

1. Como você garante a ordem das linhas no resultado de uma consulta?

2. O que é a diferença entre o resultado de uma consulta com e uma sem uma

ORDER BY cláusula?

**Quick Check Resposta**

1. A única maneira de fazer isso é através da adição de uma cláusula ORDER BY.

2. Sem uma cláusula ORDER BY, o resultado é relacional (a partir de uma perspectiva de pedidos);

com uma cláusula ORDER BY, o resultado é conceitualmente o que o padrão

chama um cursor.

Prática **classificando dados**

Nesta prática, você exercer o seu conhecimento da classificação de dados com a cláusula ORDER BY.

Se você encontrar um problema de completar um exercício, você pode instalar os projectos concluídos

a partir da pasta Solution que é fornecido com o conteúdo complementar para este capítulo e

lição.

Exercício 1 Use a cláusula ORDER BY com pedidos não-determinístico

Neste exercício, você pratica usando a cláusula ORDER BY para classificar os dados, praticando nondeterministic

encomenda.

1. SSMS aberto e conectar-se à TSQL2012 base de dados exemplo.

2. Você está convidado a escrever uma consulta que retorna as ordens para o cliente 77. Use o seguinte

consulta.

orderid SELECT, empid, CódigoDoTransitário, DataDeEnvio

DE Sales.Orders

ONDE custid = 77;

Você obter o seguinte conjunto de resultados.

orderid empid CódigoDoTransitário DataDeEnvio

-------- ------ ---------- -----------------------

10992 1 3 2008-04-03 00: 00: 00.000

10805 2 3 2008-01-09 00: 00: 00.000

10708 6 2 2007-11-05 00: 00: 00.000

10310 8 2 2006-09-27 00: 00: 00.000

Note-se que porque você não especificou uma cláusula ORDER BY, não há nenhuma garantia de que o

linhas serão retornados na ordem mostrado no código anterior. A única garantia de que

você é que você vai obter este conjunto específico de linhas.

3. Você será solicitado a rever a sua consulta de forma que as linhas serão ordenadas por CódigoDoTransitário. Adicionar

uma cláusula ORDER BY, como se segue.

orderid SELECT, empid, CódigoDoTransitário, DataDeEnvio

DE Sales.Orders

ONDE custid = 77

ORDER BY CódigoDoTransitário;

A consulta agora retorna o seguinte resultado.

orderid empid CódigoDoTransitário DataDeEnvio

-------- ------ ---------- -----------------------

10708 6 2 2007-11-05 00: 00: 00.000

10310 8 2 2006-09-27 00: 00: 00.000

10992 1 3 2008-04-03 00: 00: 00.000

10805 2 3 2008-01-09 00: 00: 00.000

Agora você garante que as linhas serão devolvidos pela ordenação CódigoDoTransitário, mas é o

ordenando determinista? Por exemplo, você pode dizer com certeza o que vai ser o fim

entre linhas com o mesmo ID shipper? A resposta é não.

Exercício 2 Use a cláusula ORDER BY com pedidos determinística

Neste exercício, você pratica usando a cláusula ORDER BY para classificar os dados, praticando determinista

encomenda.

1. Você começa este passo com a consulta que você escreveu na etapa 3 do Exercício 1. Você é dado um

obrigação de acrescentar ordenação secundária por data enviado, descendente. Adicionar CódigoDoTransitário

DESC para a cláusula ORDER BY, como se segue.

orderid SELECT, empid, CódigoDoTransitário, DataDeEnvio

DE Sales.Orders

ONDE custid = 77

ORDER BY CódigoDoTransitário, DESC DataDeEnvio;

A consulta agora retorna o seguinte resultado.

orderid empid CódigoDoTransitário DataDeEnvio

-------- ------ ---------- -----------------------

10708 6 2 2007-11-05 00: 00: 00.000

10310 8 2 2006-09-27 00: 00: 00.000

10992 1 3 2008-04-03 00: 00: 00.000

10805 2 3 2008-01-09 00: 00: 00.000

Ao contrário no passo 3, agora é garantido que as linhas com o mesmo ID remetente será

ordenadas por data enviado, descendente. Está mandando agora determinista? você pode dizer com

certeza que será a ordem entre linhas com o mesmo ID remetente e enviado

encontro? A resposta ainda é não, porque a combinação de colunas CódigoDoTransitário e

DataDeEnvio não é único, não importa o que os valores atuais que você vê na tabela

pode levar você a pensar. Tecnicamente, poderia haver várias linhas no resultado deste

consultar com os mesmos valores CódigoDoTransitário e DataDeEnvio.

2. Você está convidado a rever a consulta a partir do passo 1, garantindo ordenação determinista.

É necessário definir um critério de desempate. Por exemplo, definir orderid DESC como um desempate, como

segue.

orderid SELECT, empid, CódigoDoTransitário, DataDeEnvio

DE Sales.Orders

ONDE custid = 77

ORDER BY CódigoDoTransitário, DataDeEnvio DESC, orderid DESC;

Agora, em caso de empate nos valores CódigoDoTransitário e DataDeEnvio, a linha com o maior

valor orderid serão classificados em primeiro lugar.

Resumo da lição

■■ consultas normalmente retornar um resultado relacional, onde ordenação não é garantida. Se você precisar

para garantir a ordenação apresentação, você precisa adicionar uma cláusula ORDER BY à sua consulta.

■■ Com a cláusula ORDER BY, você pode especificar uma lista de expressões para ordenação primário,

ordenação secundário, e assim por diante. Com cada expressão, você pode indicar ASC ou DESC

para subir ou descer a encomenda, com ascendente sendo o padrão.

■■ Mesmo quando uma cláusula ORDER BY é especificada, o resultado ainda poderia ter nondeterministic

encomenda. Em caso de pedido determinista, a lista ORDER BY deve ser exclusivo.

■■ Você pode usar posições ordinais das expressões da lista SELECT na ORDER BY

cláusula, mas isso é considerado uma má prática.

■■ Você pode classificar por elementos que não aparecem na lista SELECT a menos que o DISTINCT

cláusula também é especificado.

■■ Porque a cláusula ORDER BY é conceitualmente avaliada após a cláusula SELECT,

pode se referir a aliases atribuídos na cláusula SELECT na cláusula ORDER BY.

■■ Para fins de classificação, SQL Server considera nulos como sendo menor do que os não-NULL

marcas e iguais entre si. Isto significa que quando a ordenação ascendente é utilizado,

eles meio juntos antes de marcas não-NULL.

Examine a lição

Responda as seguintes perguntas para testar seu conhecimento da informação nesta lição. Vocês

pode encontrar as respostas para estas perguntas e explicações de por que cada opção de resposta está correta

ou incorrecta na secção "Respostas" no final deste capítulo.

1. Quando uma consulta não tem uma cláusula ORDER BY, o que é a ordem em que as linhas

são devolvidos?

A. ordem arbitrária

B. primária ordem da chave

C. ordem do índice em cluster

D. fim de Inclusão

2. Você quer resultar linhas a ser classificados por orderdate descendente, e depois por orderid,

descendente. Qual das seguintes cláusulas dá o que você quer?

A. ORDER BY orderdate, orderid DESC

B. ORDER BY DESC orderdate, orderid DESC

C. ORDER BY OrderDate DESC, orderid DESC

D. DESC ORDER BY orderdate, orderid

3. Você quer que resultar linhas a ser classificados por orderdate ascendente, e depois por orderid,

ascendente. Qual das seguintes cláusulas dá o que você quer? (Escolha todas

que se aplicam.)

A. ORDER BY ASC (orderdate, orderid)

B. ORDER BY orderdate, orderid ASC

C. ORDER BY ASC orderdate, orderid ASC

D. ORDER BY orderdate, orderid

**Lição 3: filtrar dados com TOP e -FETCH OFFSET**

A primeira lição coberta filtrar dados usando predicados, ea segunda triagem coberta

dados. Esta terceira lição em sentido mistura filtragem e conceitos de classificação. Muitas vezes, você precisa filtrar

baseado em dados de pedido e dado um número especificado de linhas. Pense sobre os pedidos, tais como

"Retorno as três ordens mais recentes" e "devolver os cinco produtos mais caros." O filtro

envolve alguma especificação de encomenda e um número de linhas solicitado. T-SQL fornece dois

opções para lidar com essas necessidades de filtragem: um é a opção TOP proprietário e o outro é o

padrão opção OFFSET-busca que foi introduzido no SQL Server 2012.

Após esta lição, você será capaz de:

■■ Filtrar dados usando a opção TOP.

■■ Filtrar dados usando a opção OFFSET-busca.

tempo de aula estimado: 45 minutos

Filtrando dados com TOP

Com a opção de topo, você pode filtrar um número solicitado ou a porcentagem de linhas a partir da consulta

resultar baseada em ordem indicada. Você especificar a opção TOP na cláusula SELECT seguido

pelo número de linhas solicitado entre parênteses (BIGINT tipo de dados). A especificação ordenação

do filtro superior baseia-se na mesma ordem BY que é normalmente utilizado para apresentação

encomenda.

Como exemplo, a seguinte consulta retorna as três ordens mais recentes.

SELECT TOP (3) orderid, orderdate, custid, empid

DE Sales.Orders

ORDER BY DESC orderdate;

Você especificar 3 como o número de linhas que você deseja filtrar e OrderDate DESC como a ordenação

especificação. De modo a obter as três fileiras com as datas mais recentes ordem. Aqui está a saída do

esta consulta.

orderid orderdate custid empid

----------- ----------------------- ----------- ----- ------

11077 2008-05-06 00: 00: 00.000 65 1

11076 2008-05-06 00: 00: 00.000 9 4

11075 2008-05-06 00: 00: 00.000 68 8

*Nota***TOP e Parênteses**

T-SQL suporta especificando o número de linhas para filtrar usando a opção TOP em SELECIONAR

consultas sem parênteses, mas isso é apenas por razões de compatibilidade com versões anteriores. O correto

sintaxe é com parênteses.

Você também pode especificar uma porcentagem de linhas para filtrar, em vez de um número. Para fazer isso, especifique um

valor flutuar no intervalo de 0 a 100 em parênteses, ea palavra-chave por cento depois

os parênteses, como se segue.

SELECT TOP (1) por cento orderid, orderdate, custid, empid

DE Sales.Orders

ORDER BY DESC orderdate;

A opção POR CENTO coloca um limite máximo para o número resultante de linhas, se não é todo. Nisso

exemplo, sem a opção TOP, o número de linhas no resultado é 830. Filtragem de 1 por cento

dá-lhe 8,3, e depois o teto desse valor dá-lhe 9; portanto, a consulta retorna 9 linhas.

orderid orderdate custid empid

----------- ----------------------- ----------- ----- ------

11076 2008-05-06 00: 00: 00.000 9 4

11077 2008-05-06 00: 00: 00.000 65 1

11075 2008-05-06 00: 00: 00.000 68 8

11074 2008-05-06 00: 00: 00.000 73 7

11070 2008-05-05 00: 00: 00.000 44 2

11071 2008-05-05 00: 00: 00.000 46 1

11073 2008-05-05 00: 00: 00.000 58 2

11072 2008-05-05 00: 00: 00.000 20 4

11067 2008-05-04 00: 00: 00.000 17 1

A primeira opção não é limitada a uma entrada constante; em vez disso, ele permite que você especifique um selfcontained

expressão. De uma perspectiva prática, essa capacidade é especialmente importante quando

você precisa passar um parâmetro ou uma variável como entrada, como o código a seguir demonstra.

DECLARE @ n AS BIGINT = 5;

SELECT TOP (@ n) orderid, orderdate, custid, empid

DE Sales.Orders

ORDER BY DESC orderdate;

Esta consulta gera a seguinte saída.

orderid orderdate custid empid

----------- ----------------------- ----------- ----- ------

11076 2008-05-06 00: 00: 00.000 9 4

11077 2008-05-06 00: 00: 00.000 65 1

11075 2008-05-06 00: 00: 00.000 68 8

11074 2008-05-06 00: 00: 00.000 73 7

11070 2008-05-05 00: 00: 00.000 44 2

Na maioria dos casos, você precisa de sua opção TOP que confiar em alguma especificação de encomenda, mas como ele

Acontece que, uma cláusula ORDER BY não é obrigatória. Por exemplo, a seguinte consulta é tecnicamente

válido.

SELECT TOP (3) orderid, orderdate, custid, empid

DE Sales.Orders;

No entanto, a consulta não é determinista. A consulta filtra três linhas, mas você não tem nenhuma garantia

Quais as três linhas serão devolvidos. Você acaba ficando quaisquer que sejam três fileiras SQL Server

aconteceu para acessar em primeiro lugar, e isso é dependente de otimização. Por exemplo, esta consulta deu

a seguinte saída em um sistema.

orderid orderdate custid empid

----------- ----------------------- ----------- ----- ------

11011 2008-04-09 00: 00: 00.000 1 3

10952 2008-03-16 00: 00: 00.000 1 1

10835 2008-01-15 00: 00: 00.000 1 1

Mas não há nenhuma garantia de que as mesmas linhas será devolvido se você executar a consulta novamente. E se

você está realmente depois de três linhas arbitrárias, pode ser uma boa idéia para adicionar uma cláusula ORDER BY

com a expressão (SELECT NULL) para que as pessoas saibam que a sua escolha é intencional e não

um descuido. Veja como sua consulta ficaria.

SELECT TOP (3) orderid, orderdate, custid, empid

DE Sales.Orders

ORDER BY (SELECT NULL);

Note-se que mesmo quando você tem uma cláusula ORDER BY, para que a consulta seja

completamente determinista, a ordem deve ser exclusivo. Por exemplo, considerar novamente o primeiro

consultar a partir desta secção.

SELECT TOP (3) orderid, orderdate, custid, empid

DE Sales.Orders

ORDER BY DESC orderdate;

A coluna orderdate não é original, assim que a ordem em caso de empates é arbitrária. quando esta

consulta foi executada, o sistema retornou a seguinte saída.

orderid orderdate custid empid

----------- ----------------------- ----------- ----- ------

11077 2008-05-06 00: 00: 00.000 65 1

11076 2008-05-06 00: 00: 00.000 9 4

11075 2008-05-06 00: 00: 00.000 68 8

Mas o que se há outras linhas no resultado sem TOP que tenham a mesma data do pedido

como na última fila aqui? Você nem sempre se preocupam com a garantia determinista ou repetível

resultados; mas se o fizer, há duas opções disponíveis para você. Uma opção é pedir para incluir todos os laços

com a última linha, adicionando a opção com laços, como se segue.

SELECT TOP (3) COM LAÇOS orderid, orderdate, custid, empid

DE Sales.Orders

ORDER BY DESC orderdate;

Claro, isso pode resultar em voltar mais linhas do que você pediu, como a saída deste

consulta mostra.

orderid orderdate custid empid

----------- ----------------------- ----------- ----- ------

11077 2008-05-06 00: 00: 00.000 65 1

11076 2008-05-06 00: 00: 00.000 9 4

11075 2008-05-06 00: 00: 00.000 68 8

11074 2008-05-06 00: 00: 00.000 73 7

A outra opção para garantir o determinismo é para quebrar os laços adicionando um tiebreaker

que faz com que a ordem original. Por exemplo, no caso de empate na data do pedido, suponha que você

queria a linha com o maior fim ID de "ganhar". Para fazer isso, adicione orderid DESC para o seu fim

BY, como se segue.

SELECT TOP (3) COM LAÇOS orderid, orderdate, custid, empid

DE Sales.Orders

ORDER BY DESC orderdate, orderid DESC;

Aqui está a saída dessa consulta.

orderid orderdate custid empid

----------- ----------------------- ----------- ----- ------

11077 2008-05-06 00: 00: 00.000 65 1

11076 2008-05-06 00: 00: 00.000 9 4

11075 2008-05-06 00: 00: 00.000 68 8

A consulta agora é determinístico, e os resultados são garantidos para ser repetível, enquanto

que os dados subjacentes não mudar.

Para concluir esta seção, gostaria apenas de notar que a opção TOP também pode ser usado em

instruções de modificação de limitar quantas linhas são modificados, mas as modificações são cobertos

mais adiante neste kit de treinamento.

Filtrando dados com offset-busca

A opção OFFSET-busca é uma opção de filtragem que, como TOP, você pode usar para filtrar dados com base

em um número especificado de linhas e ordenação. Mas, ao contrário do TOP, isto é padrão, e também tem uma

pular capacidade, tornando-o útil para fins de paginação ad-hoc.

As cláusulas OFFSET buscar aparecem logo após a cláusula ORDER BY, e de fato, em

T-SQL, eles exigem uma cláusula ORDER BY para estar presente. Primeiro você especifique a cláusula OFFSET

indicando quantas linhas você quiser pular (0 se você não quiser pular houver); Você, então, opcionalmente

especifique a cláusula FETCH indicando quantas linhas você deseja filtrar. Por exemplo,

a consulta a seguir define ordenação com base na data ordem descendente, seguido por ID do pedido

descendente; Em seguida, ele salta 50 linhas e vai buscar os próximos 25 linhas.

orderid SELECT, orderdate, custid, empid

DE Sales.Orders

ORDER BY DESC orderdate, orderid DESC

DESVIO 50 linhas FETCH PRÓXIMOS 25 apenas linhas;

Aqui está uma forma abreviada da saída.

orderid orderdate custid empid

----------- ----------------------- ----------- ----- ------

11027 2008-04-16 00: 00: 00.000 10 1

11026 2008-04-15 00: 00: 00.000 27 4

...

11004 2008-04-07 00: 00: 00.000 50 3

11003 2008-04-06 00: 00: 00.000 78 3

A cláusula ORDER BY agora desempenha dois papéis: Um papel está dizendo a opção OFFSET-busca

quais as linhas que necessita para filtrar. Outra função é determinar ordenação apresentação na consulta.

Como mencionado, em T-SQL, o DESVIO-fetch opção requer uma cláusula ORDER BY para ser

presente. Além disso, em T-SQL-ao contrário do padrão SQL-a cláusula FETCH requer uma cláusula OFFSET

estar presente. Então, se você deseja filtrar algumas linhas, mas ignorar nenhum, você ainda precisa especificar

a cláusula de compensação com 0 linhas.

A fim de tornar a sintaxe intuitivo, você pode usar as palavras-chave NEXT ou PRIMEIRO alternadamente.

Ao pular algumas linhas, talvez seja mais intuitiva para que você use as palavras-chave

FETCH NEXT para indicar quantas linhas para filtrar; mas quando não pular nenhuma linha, pode ser

mais intuitiva para que você use as palavras-chave FETCH FIRST, como se segue.

orderid SELECT, orderdate, custid, empid

DE Sales.Orders

ORDER BY DESC orderdate, orderid DESC

OFFSET 0 linhas FETCH primeiras 25 apenas linhas;

Por razões semelhantes, você pode usar o formulário ROW singular ou as linhas forma plural de forma intercambiável,

tanto para o número de linhas para pular e para o número de linhas para filtrar. Mas é

não gosto de você receberá um erro se disser BUSCAR próximo 1 linhas ou FETCH próximos 25 ROW. Depende de

para que você use uma forma adequada, assim como com o Inglês.

Enquanto em T-SQL, a cláusula FETCH requer uma cláusula OFFSET, ea cláusula OFFSET não

exigem uma cláusula FETCH. Em outras palavras, indicando uma cláusula OFFSET, você está pedindo para

pular algumas linhas; em seguida, ao não indicar uma cláusula FETCH, você está pedindo para retornar todas as restantes

linhas. Por exemplo, os seguintes pedidos de consulta para saltar 50 linhas, retornando todo o resto.

orderid SELECT, orderdate, custid, empid

DE Sales.Orders

ORDER BY DESC orderdate, orderid DESC

DESVIO 50 linhas;

Aqui está uma forma abreviada da saída.

orderid orderdate custid empid

----------- ----------------------- ----------- ----- ------

11027 2008-04-16 00: 00: 00.000 10 1

11026 2008-04-15 00: 00: 00.000 27 4

...

10249 2006-07-05 00: 00: 00.000 79 6

10248 2006-07-04 00: 00: 00.000 85 5

(780 row (s) affected)

Como mencionado anteriormente, a opção OFFSET-busca requer uma cláusula ORDER BY. Mas o que

se você precisa para filtrar um determinado número de linhas com base em ordem arbitrária? Para fazer isso, você pode

especifique a expressão (SELECT NULL) na cláusula ORDER BY, como se segue.

orderid SELECT, orderdate, custid, empid

DE Sales.Orders

ORDER BY (SELECT NULL)

OFFSET 0 linhas buscar 3 primeiros apenas linhas;

Este código simplesmente filtra três linhas arbitrárias. Aqui está a saída de um sistema retornou após

executar o código.

orderid orderdate custid empid

----------- ----------------------- ----------- ----- ------

11011 2008-04-09 00: 00: 00.000 1 3

10952 2008-03-16 00: 00: 00.000 1 1

10835 2008-01-15 00: 00: 00.000 1 1

Com cláusulas tanto o deslocamento e a busca, você pode usar expressões como entradas. Isto é muito

útil quando você precisa para calcular os valores de entrada dinamicamente. Por exemplo, suponha que

você está implementando um conceito de paginação onde você retornar ao usuário uma página de linhas de cada vez.

O usuário passa como parâmetros de entrada para o procedimento ou uma função do número de página são

depois (parâmetro @pagenum) e tamanho de página (@pagesize parâmetro). Isso significa que você precisa

para ignorar tantas linhas quanto @pagenum menos um vezes @pagesize, e buscar a próxima @pagesize

linhas. Isso pode ser implementado usando o código a seguir (usando variáveis ​​locais para simplificar).

DECLARE @pagesize AS BIGINT = 25, @pagenum AS BIGINT = 3;

orderid SELECT, orderdate, custid, empid

DE Sales.Orders

ORDER BY DESC orderdate, orderid DESC

OFFSET (@pagesize - 1) \* ROWS @pagesize buscar linhas @pagesize seguinte somente;

Com essas entradas, o código retorna a seguinte saída.

orderid orderdate custid empid

----------- ----------------------- ----------- ----- ------

10477 2007-03-17 00: 00: 00.000 60 5

10476 2007-03-17 00: 00: 00.000 35 8

...

10454 2007-02-21 00: 00: 00.000 41 4

10453 2007-02-21 00: 00: 00.000 4 1

(25 row (s) affected)

Você pode se sentir livre para alterar os valores de entrada e ver como o resultado muda em conformidade.

Como a opção OFFSET-busca é padrão e TOP não é, nos casos em que eles são logicamente

equivalente, é recomendado para manter o primeiro. Lembre-se que compensam-busca também

tem uma vantagem sobre a parte superior no sentido de que ele suporta uma capacidade de salto. No entanto, para

Agora, COMPENSAR-fetch não suporta opções semelhantes a CENTO do TOP e WITH TIES.

Do ponto de vista de desempenho, você deve avaliar a indexação do ORDER BY colunas para

apoiar o TOP e as opções de offset FETCH. Este índice serve a um propósito muito semelhante ao

indexação colunas filtrada e pode ajudar a evitar os dados desnecessários de digitalização, bem como a classificação.

**Checagem rápida**

1. Como você garante resultados determinísticos com TOP?

2. Quais são os benefícios de usar offset buscar por cima?

**Quick Check Resposta**

1. Por retornando todos os laços usando a opção com laços ou definindo única

ordenando para desempatar.

2. OFFSET-busca é padrão e TOP não é; Também, COMPENSAR-busca suporta uma skipping

capacidade que TOP não.

Prática **Filtrando dados com TOP e -FETCH OFFSET**

Nesta prática, você exercer o seu conhecimento de filtragem de dados com TOP e OFFSET-busca.

Se você encontrar um problema de completar um exercício, você pode instalar os projectos concluídos

a partir da pasta Solution que é fornecido com o conteúdo complementar para este capítulo e

lição.

Exercício 1 Use a opção do topo

Neste exercício, você pratica usando a opção TOP para filtrar dados.

1. SSMS aberto e conectar-se à TSQL2012 base de dados exemplo.

2. Você está encarregado de escrever uma consulta em relação a tabela de Production.Products, retornando

os cinco produtos mais caros da categoria 1. Escreva a seguinte consulta.

SELECT TOP (5) productid, PreçoUnitário

DE Production.Products

ONDE categoryid = 1

ORDER BY DESC PreçoUnitário;

Você obter o seguinte conjunto de resultados.

PreçoUnitário productid

----------- ---------------------

38 263,50

43 46.00

2 19.00

1 18.00

35 18.00

Essa consulta retorna o resultado desejado, exceto que ele não tem qualquer manipulação dos laços. em outra

palavras, a ordem entre os produtos com o mesmo preço unitário é não-determinístico.

3. Você será solicitado a fornecer soluções para transformar a consulta anterior em um determinista

um-um solução que inclui laços e outro que rompe os laços. Primeiro, endereço

a versão que inclui todos os laços usando a opção com laços. Adicionar esta opção para o

consulta, como se segue.

SELECT TOP (5) com laços productid, PreçoUnitário

DE Production.Products

ONDE categoryid = 1

ORDER BY DESC PreçoUnitário;

Você obtém o seguinte resultado, que inclui laços.

PreçoUnitário productid

----------- ---------------------

38 263,50

43 46.00

2 19.00

1 18.00

39 18.00

35 18.00

76 18.00

4. Abordar a segunda versão que quebra os laços usando productid, descendente, como

segue.

SELECT TOP (5) productid, PreçoUnitário

DE Production.Products

ONDE categoryid = 1

ORDER BY DESC PreçoUnitário, ProductID DESC;

Esta consulta gera a seguinte saída.

PreçoUnitário productid

----------- ---------------------

38 263,50

43 46.00

2 19.00

76 18.00

39 18.00

Exercício 2 Use a opção de deslocamento-busca

Neste exercício, você pratica usando a opção OFFSET-busca para filtrar dados.

1. SSMS aberto e conectar-se à TSQL2012 base de dados exemplo.

2. Você está convidado a escrever um conjunto de consultas que página através de produtos, cinco de cada vez,

na ordenação de preço unitário, usando a identificação do produto como desempate. Comece por escrever uma consulta

que retorna os primeiros cinco produtos.

productid SELECT, categoryid, PreçoUnitário

DE Production.Products

ORDER BY PreçoUnitário, productid

OFFSET 0 linhas FETCH primeiras 5 linhas SOMENTE;

Você poderia ter usado a primeira ou a palavra-chave seguinte, mas dizer que você decidiu usar

Primeiro porque foi a opção mais natural quando não pular nenhuma linha. esta consulta

gera a seguinte saída.

productid categoryid PreçoUnitário

----------- ----------- ---------------------

33 4 2,50

24 1 4,50

13 8 6,00

52 5 7,00

54 6 7,45

3. Em seguida, escreva uma consulta que retorna os próximos cinco linhas (linhas 6 a 10), utilizando a seguinte

consulta.

productid SELECT, categoryid, PreçoUnitário

DE Production.Products

ORDER BY PreçoUnitário, productid

OFFSET 5 linhas FETCH próximos 5 apenas linhas;

Desta vez, use a palavra-chave seguinte, porque você está ignorando algumas linhas. esta consulta

gera a seguinte saída.

productid categoryid PreçoUnitário

----------- ----------- ---------------------

75 1 7,75

23 5 9,00

19 3 9,20

45 8 9,50

47 3 9.50

4. Da mesma forma, escrever a seguinte consulta para retornar as linhas 11 a 15:

productid SELECT, categoryid, PreçoUnitário

DE Production.Products

ORDER BY PreçoUnitário, productid

DESVIO 10 linhas FETCH próximos 5 apenas linhas;

Esta consulta gera a seguinte saída.

productid categoryid PreçoUnitário

----------- ----------- ---------------------

41 8 9,65

3 2 10,00

21 3 10.00

74 7 10.00

46 8 12.00

Você iria seguir um processo semelhante para as páginas subsequentes.

Resumo da lição

■■ com a parte superior e as opções de offset FETCH, você pode filtrar os dados com base em um determinado

número de linhas e ordenação.

■■ A cláusula ORDER BY que é normalmente usado na consulta para a apresentação de pedidos é

também utilizado por cima e compensar FETCH para indicar quais linhas para filtrar.

■■ A opção TOP é uma característica T-SQL proprietário que você pode usar para indicar um número ou

um por cento das linhas de filtrar.

■■ Você pode fazer uma consulta TOP determinista de duas maneiras: uma é usando os laços com

opção para retornar todos os laços, ea outra é usando ordenação única para romper os laços.

■■ A opção OFFSET-busca é uma opção padrão semelhante ao TOP, suportado pelo SQL Server

2012. Ao contrário TOP, ele permite que você especifique quantas linhas a serem saltadas antes de indicar como

muitas linhas para filtrar. Como tal, ele pode ser usado para fins de paging ad-hoc.

■■ Ambos TOP e compensar-busca expressões de apoio como entradas e não apenas constantes.

Examine a lição

Responda as seguintes perguntas para testar seu conhecimento da informação nesta lição. Vocês

pode encontrar as respostas para estas perguntas e explicações de por que cada opção de resposta está correta

ou incorrecta na secção "Respostas" no final deste capítulo.

1. Você executar uma consulta com um (3) opção TOP. Qual das seguintes opções mais precisão

descreve quantas linhas serão devolvidos?

A. Menos de três linhas

B. Três linhas ou menos

C. Três fileiras

D. Três linhas ou mais

E. Mais de três linhas

F. Menos de três, três, ou mais de três linhas

2. Você executar uma consulta com o TOP (3) com laços e ordenação não exclusivo. Qual dos

as seguintes opções com mais precisão descreve quantas linhas serão devolvidos?

A. Menos de três linhas

B. Três linhas ou menos

C. Três fileiras

D. Três linhas ou mais

E. Mais de três linhas

F. Menos de três, três, ou mais de três linhas

3. Qual das seguintes opções de offset FETCH são válidos em T-SQL? (Escolha tudo o que se aplicam).

A. instrução SELECT ... ORDER BY orderid OFFSET 25 linhas

B. instrução SELECT ... ORDER BY orderid FETCH PRÓXIMOS 25 apenas linhas

C. instrução SELECT ... ORDER BY orderid OFFSET 25 linhas FETCH PRÓXIMOS 25 apenas linhas

D. instrução SELECT ... <sem ORDER BY> OFFSET 0 linhas FETCH primeiras 25 apenas linhas

**Cenários de caso**

Nos seguintes cenários, você aplicar o que você aprendeu sobre a filtragem e classificação de dados.

Você pode encontrar as respostas para essas perguntas na seção "Respostas" no final deste capítulo.

Cenário Caso 1: Filtragem e Desempenho Classificando

recomendações

Você está contratado como consultor para ajudar a problemas de desempenho de consulta de endereço em uma fábrica de cerveja

a executar o SQL Server 2012. Você rastrear uma carga de trabalho típica submetidos ao sistema e observar

muito consulta lenta vezes executado. Você vê um monte de tráfego de rede. Você vê que muitas consultas retornam

todas as linhas para o cliente e, em seguida, o cliente lida com a filtragem. Consultas que filtram os dados com frequência

manipular as colunas filtrados. Todas as consultas têm cláusulas ORDER BY, e quando você perguntar

sobre isso, você disse que não é realmente necessário, mas os desenvolvedores se acostumou a fazer

assim apenas no caso. Você identifica uma série de operações de classificação caros. O cliente está procurando

recomendações para melhorar o desempenho e pede-lhe as seguintes perguntas:

1. Nada pode ser feito para melhorar a forma de filtragem é tratado?

2. Existe algum mal em especificando ORDER BY mesmo quando os dados não precisam ser

voltou ordenado?

3. Todas as recomendações relacionadas com as consultas com TOP e OFFSET-busca?

Cenário Caso 2: Explicações um desenvolvedor júnior

Está a tutoria um desenvolvedor júnior sobre filtragem e classificação de dados com T-SQL. o desenvolvedor

parece estar confuso sobre determinados temas e levanta algumas perguntas para você. Responda

o seguinte com o melhor de seu conhecimento:

1. Quando tento referir-se a um alias de coluna que eu definido na lista SELECT na ONDE

cláusula, eu recebo um erro. Você pode explicar por que isso não é permitido eo que as soluções alternativas

estamos?

2. Referindo-se a um alias de coluna na cláusula ORDER BY parece ser apoiada. Por que é

aquele?

3. Por que é que a Microsoft tornou obrigatório para especificar uma cláusula ORDER BY quando se utiliza

DESVIO-busca, mas não quando usando TOP? Isso significa que apenas consultas TOP podem

têm ordenação não-determinístico?

**práticas sugeridas**

Para ajudar você a dominar com sucesso os objetivos do exame apresentados neste capítulo, preencha o

seguinte tarefas.

Identificar as fases de consulta de processamento lógico e

comparar Filtros

Para praticar o seu conhecimento de processamento de consulta lógica, liste os elementos que você aprendeu

sobre tão longe em sua ordem correta.

■■ **Prática 1**Neste capítulo, você aprendeu sobre como usar a cláusula WHERE para filtrar dados

com base em predicados, a cláusula ORDER BY para classificar os dados, e parte superior e OFFSETFETCH

opções como uma outra maneira de filtrar os dados. Combinado com o seu conhecimento da

Capítulo 1, listar a consulta elementos SELECT, FROM, WHERE, GROUP BY, HAVING, ORDER

BY, TOP, e OFFSET-busca, a fim de processamento de consulta lógica correta. Observe que

porque TOP e OFFSET-busca não podem ser combinados na mesma consulta, você precisa

criar duas dessas listas.

■■ **Prática 2**Lista as capacidades que o filtro OFFSET-busca tem que não estão disponíveis

de topo em SQL Server 2012, e também o contrário.

Entenda Determinismo

Lembre-se que uma consulta determinista é aquele que tem apenas um resultado correto. Para demonstrar o seu

conhecimento do determinismo consulta, fornecem exemplos para determinista e não-determinístico

consultas.

■■ **Prática 1**Fornecer exemplos para consultas com determinista e não-determinístico

encomenda. Descreva com suas próprias palavras o que é necessário para obter ordenação determinista.

■■ **Prática 2**Fornecer exemplos para consultas determinísticos e não-determinísticos usando

TOP e OFFSET-busca. Explicar como você pode aplicar o determinismo em ambos os casos.

**respostas**

Esta seção contém as respostas para as perguntas de revisão lição e soluções para o caso

cenários neste capítulo.

Lição 1

1. **Resposta correta: B**

A. **incorreta:**NULL não faz parte dos três resultados lógicos possíveis de um predicado em

T-SQL.

B. **correto:**Three-valorizado lógica refere-se verdadeiro, falso, e desconhecido.

C. **incorreto:**1, 0 e NULL não fazem parte dos resultados dos três lógicas possíveis de um

predicado.

D. **incorreto:**-1, 0 e 1 não fazem parte dos resultados dos três lógicas possíveis de um

predicado.

2. **Corrigir respostas: A, B, e C**

A. **correta:**A forma '2012-02-12' é linguagem neutra para a data de tipos de dados,

DATETIME2 e DATETIMEOFFSET, mas a linguagem-dependente para DATETIME e

SMALLDATETIME.

B. **correta:**A forma '02 / 12/2012 'é dependente de idioma.

C. **correta:**A forma '12 / 02/2012 'é dependente de idioma.

D. **incorreto:**A forma "20120212" é linguagem neutra.

3. **Corrigir respostas: B e E**

A. **incorreta:**Esse predicado se aplica manipulação para a coluna filtrada, e, portanto,

não é um argumento de pesquisa.

B. **correto:**O predicado LIKE é um argumento de pesquisa quando o padrão começa com um

prefixo conhecido.

C. **incorreto:**O predicado LIKE não é um argumento de pesquisa quando o padrão começa com

um wild card.

D. **incorreta:**O predicado LIKE não é um argumento de pesquisa quando o padrão começa com

um wild card.

E. **correcta:**Porque nenhuma manipulação é aplicada à coluna filtrada, o predicado

é um argumento de pesquisa.

lição 2

1. **Resposta correta: A**

A. **correta:**Sem uma cláusula ORDER BY, a ordenação não garantida, e está a ser dito

arbitrária-é dependente de otimização.

B. **incorreto:**Sem uma cláusula ORDER BY, não há nenhuma garantia para encomendar.

C. **incorreto:**Sem uma cláusula ORDER BY, não há nenhuma garantia para encomendar.

D. **incorreto:**Sem uma cláusula ORDER BY, não há nenhuma garantia para encomendar.

2. **Resposta correta: C**

A. **incorreta:**Este usa ascendente ordenação para orderdate e descendo apenas para

orderid.

B. **incorreto:**Esta é uma sintaxe inválida.

C. **correto:**A sintaxe correta é especificar DESC após cada expressão cujo ordenamento

direcção precisa ser descendente.

D. **incorreto:**Esta é uma sintaxe inválida.

3. **Resposta correta: B, C e D**

A. **incorreta:**Esta é uma sintaxe inválida.

B. **correta:**A direção padrão é ascendente, de modo que esta cláusula usa ordem crescente para

tanto orderdate e orderid.

C. **correta:**Esta cláusula usa explicitamente para que ambos orderdate e orderid ascendente.

D. **correta:**A direção padrão é ascendente, de modo que esta cláusula usa ordem crescente para

tanto orderdate e orderid.

lição 3

1. **Resposta correta: B**

A. **incorreto:**Se há pelo menos três fileiras no resultado da consulta, sem TOP, a

consulta retornará três linhas.

B. **correto:**Se houver menos linhas do que três no resultado da consulta sem TOP, a

consulta retornará apenas as linhas. Se existem três linhas ou mais, sem TOP, o

consulta retornará três linhas.

C. **incorreto:**Se houver menos linhas do que três no resultado da consulta sem TOP, a

consulta retornará apenas as linhas.

D. **incorreto:**A não ser que a opção com laços é usado, a consulta não retornará mais de

o número de linhas solicitado.

E. **incorreto:**A não ser que a opção com laços é usado, a consulta não retornará mais de

o número de linhas solicitado.

F. **incorreto:**A não ser que a opção com laços é usado, a consulta não retornará mais de

o número de linhas solicitado.

2. **Resposta correta: F**

A. **incorreto:**Se há pelo menos três fileiras no resultado da consulta, sem TOP, a

consulta retornará pelo menos três linhas.

B. **incorreto:**Se houver mais de três linhas no resultado, assim como os laços com o

terceira linha, a consulta retornará mais de três linhas.

C. **incorreto:**Se houver menos linhas do que três no resultado da consulta sem TOP, a

consulta retornará apenas as linhas. Se há mais do que três linhas no resultado, conforme

bem como os laços com a terceira fila, a consulta retornará mais de três linhas.

D. **incorreto:**Se houver menos linhas do que três no resultado da consulta sem TOP, a

consulta retornará apenas as linhas.

E. **incorreto:**Se existem três linhas ou menos no resultado da consulta sem TOP, a consulta

não voltará mais de três linhas.

F. **correto:**Se houver menos linhas do que três no resultado da consulta sem TOP, a

consulta retornará apenas as linhas. Se não houver pelo menos três linhas no resultado e

qualquer vínculo com a terceira, a consulta retornará três linhas. Se há mais do que três

linhas no resultado, assim como os laços com a terceira fila, a consulta irá retornar mais de

três linhas.

3. **Resposta correta: A e C**

A. **correta:**T-SQL suporta indicando uma cláusula OFFSET, sem uma cláusula FETCH.

B. **incorreto:**Ao contrário do padrão SQL, T-SQL não suporta a cláusula FETCH sem

uma cláusula de compensação.

C. **correta:**T-SQL suporta indicando tanto OFFSET e FETCH cláusulas.

D. **incorreto:**não T-SQL não suporta offset FETCH sem uma cláusula ORDER BY.

Case Scenario 1

1. Por um lado, tanto quanto possível de filtragem deve ser feito no banco de dados. fazendo mais

da filtragem no cliente significa que você está digitalizando mais dados, o que aumenta a

estresse sobre o subsistema de armazenamento, e também que você causar tráfego de rede desnecessário.

Quando você filtrar nas bases de dados, por exemplo, utilizando a cláusula WHERE, você

deve usar argumentos de pesquisa que aumentam a probabilidade para o uso eficiente dos índices.

Você deve tentar, tanto quanto possível, para evitar manipular as colunas filtrados.

2. Adicionando uma cláusula ORDER BY significa que o SQL Server necessita para garantir a devolução do

linhas na ordem solicitada. Se não houver índices existentes para apoiar a ordenação

requisitos, SQL Server não terá escolha, mas para classificar os dados. Sorting é caro

com grandes conjuntos. Portanto, a recomendação geral é a de evitar a adição de cláusulas ORDER BY

a consultas quando não existem requisitos de pedido. E quando você precisa fazer para voltar

as linhas em uma ordem específica, considere arranjar índices de apoio que podem impedir

SQL Server da necessidade de realizar operações de classificação caros.

3. A principal maneira de ajudar a consultas com TOP e OFFSET-busca um bom desempenho é arranjando

índices para apoiar os elementos de ordenação. Isso pode impedir a digitalização de todos os dados, em

Além de classificar.

Cenário Caso 2

1. Para ser capaz de entender por que você não pode se referir a um alias que foi definido no

lista SELECT na cláusula WHERE, você precisa entender o processamento de consultas lógico.

Mesmo que o digitado-in ordem das cláusulas é SELECT-FROM-WHERE-GROUP

BY-TER-ORDER BY, a ordem de processamento de consulta lógica é FROM-WHERE-GROUP

BY-TER-SELECT-ORDER BY. Como você pode ver, a cláusula WHERE é avaliada antes de

a cláusula SELECT, e, portanto, aliases definidos na cláusula SELECT não são visíveis para

a cláusula WHERE.

2. Lógico ordem de processamento de consulta explica por que a cláusula ORDER BY pode se referir a aliases

definido na cláusula SELECT. Isso porque a cláusula ORDER BY é logicamente avaliado

após a cláusula SELECT.

3. A cláusula ORDER BY é obrigatória quando se utiliza OFFSET-FETCH porque esta cláusula é

padrão, e SQL padrão decidiu torná-lo obrigatório. Microsoft simplesmente seguido

o padrão. Quanto TOP, esta característica é proprietário, e quando a Microsoft projetou,

eles escolheram para permitir o uso TOP em um completamente não-determinístico forma, sem uma

ORDER BY. Note-se que o fato de que OFFSET-busca requer uma cláusula ORDER BY

não significa que você deve usar ordenação determinista. Por exemplo, se a sua encomenda

BY lista não é único, a ordem não é determinista. E se você quiser a ordem para ser

completamente não-determinístico, você pode especificar ORDER BY (SELECT NULL) e então é

equivalente a não especificar uma cláusula ORDER BY em tudo