**Capítulo 2**

Introdução ao

instrução SELECT

Objetivos do exame neste capítulo:

■■ Trabalhar com dados

■■ dados de consulta usando instruções SELECT.

■■ Implementar tipos de dados.

■■ modificar dados

■■ trabalhar com funções.

T ele capítulo anterior lhe forneceu as bases para T-SQL. Este capítulo começa

cobrindo dois dos principais cláusulas-FROM consulta e selecione. Em seguida, continua

cobrindo os tipos de dados suportados pelo Microsoft SQL Server e as considerações em

escolher os tipos de dados apropriados para as suas colunas. Este capítulo também cobre o uso de

built-in funções escalares, a expressão CASE, e variações como ISNULL e COALESCE.

Lições neste capítulo:

■■ Lição 1: Usando o FROM e cláusulas SELECT

■■ Lição 2: Trabalhando com tipos de dados e funções embutidas

**Antes de você começar**

Para concluir as lições deste capítulo, você deve ter:

Experiência ■■ trabalhar com 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.

**Lesson 1: Using the FR OM and SE LECT Clauses**

O FROM e cláusulas SELECT são duas cláusulas principais que aparecem em quase todas as consultas

que recupera dados. Esta lição explica a finalidade destas cláusulas, como usá-los, e

melhores práticas que lhes estão associados.

Após esta lição, você será capaz de:

■■ Escrever consultas que usam o de e selecione cláusulas.

■■ definir a tabela e coluna aliases.

■■ Descrever as melhores práticas associadas com a DA e SELECT cláusulas.

Tempo de aula estimado: 30 minutos

A cláusula FROM

De acordo com o processamento de consultas lógica (ver detalhes no Capítulo 1, "Fundamentos da Consultando,"

explicando o conceito), a cláusula FROM é a primeira cláusula para ser avaliado logicamente em um

Consulta SELECT. A cláusula FROM tem duas funções principais:

■■ É a cláusula onde indicam as tabelas que você deseja consultar.

■■ É a cláusula onde você pode aplicar os operadores de mesa como junta-se às tabelas de entrada.

Este capítulo centra-se no primeiro papel. Capítulo 4, "Combinação de Jogos," e Capítulo 5, "Agrupamento

e em janelas, "cobre o uso de operadores de mesa.

Como um exemplo básico, supondo que você está conectado à TSQL2012 base de dados exemplo, a seguinte

consulta usa a cláusula FROM para especificar que HR.EMPLOYEES é a tabela que está sendo consultado.

SELECT empid, firstname, lastname

FROM HR.Employees;

Observar o uso do nome de duas partes para se referir à mesa. A primeira parte (HR) é o

nome de esquema ea segunda parte (funcionários) é o nome da tabela. Em alguns casos, o t-SQL suporta

omitir o nome do esquema, como na dos seus empregados, caso em que se utiliza um implícito

nome do esquema processo de resolução. É considerada uma boa prática sempre indicam explicitamente

o nome do esquema. Esta prática pode impedi-lo de acabar com um nome de esquema

você não tinha a intenção de ser usado, e também pode remover o custo envolvido na resolução implícita

processo, embora este custo é menor.

Na cláusula FROM, pode apelidar as tabelas consultadas com seus nomes escolhidos. Você pode usar

a forma *<table>* *<alias>,* como em HR.EMPLOYEES E, ou *<table>* AS *<alias>,* como em HR.EMPLOYEES

AS E. Esta última forma é mais legível. Ao usar aliases, a convenção é usar short

nomes, normalmente uma letra que é de algum modo indicativo da tabela consultada, como E para os funcionários.

As razões pelas quais você pode querer tabelas de alias se tornam aparentes no Capítulo 4. Por agora,

é suficiente para que você saiba que a língua suporta tais aliases de tabela e a sintaxe para

atribuí-los.

Note que se você atribuir um alias para uma tabela, é, basicamente, renomear a tabela para a duração

da consulta. O nome da tabela original não é mais visível; apenas o alias é. Normalmente, você pode

prefixo um nome de coluna que se referem a em uma consulta com o nome da tabela, como no Employees.empid.

No entanto, se você alias a tabela Funcionários como E, a referência Employees.empid é inválido;

você tem que usar E.empid, como demonstra o seguinte exemplo.

SELECT E.empid, firstname, lastname

FROM HR.Employees AS E;

Se você tentar executar este código usando o nome completo de mesa como o prefixo da coluna, o código

vai falhar.

Como mencionado, capítulo 4 começa a ter os detalhes do porquê aliasing tabela é necessário.

A cláusula SELECT

A cláusula SELECT de uma consulta tem duas funções principais:

■■ Ele avalia as expressões que definem os atributos em resultado da consulta, atribuindo-lhes

com aliases, se necessário.

■■ Usando uma cláusula DISTINCT, você pode eliminar as linhas duplicadas no resultado, se necessário.

Vou começar com o primeiro papel. Tome a consulta a seguir como um exemplo.

SELECT empid, firstname, lastname

FROM HR.Employees;

A cláusula FROM indica que a tabela HR.EMPLOYEES é a tabela de entrada da consulta.

A cláusula SELECT então projeta apenas três dos atributos da entrada como o retornado

atributos no resultado da consulta.

T-SQL suporta o uso de um asterisco (\*) como uma alternativa ao listando todos os atributos da entrada

mesas, mas isso é considerado uma prática má para um número de razões. Muitas vezes, você precisa voltar

apenas um subconjunto da entrada atributos e usando um \* é apenas uma questão de preguiça. Ao retornar

mais atributos do que você realmente precisa, você pode impedir que o SQL Server de usar o que seria

normalmente considerada cobrindo índices em relação ao conjunto interessante de atributos. Vocês

também enviar dados mais do que é necessário ao longo da rede, e isto pode ter um impacto negativo sobre

o desempenho do sistema. Além disso, a definição da tabela subjacente pode mudar ao longo do tempo;

mesmo que, quando a consulta foi inicialmente autoria, \* realmente representados todos os atributos que você precisava;

ele pode não ser mais o caso em um ponto posterior no tempo. Por estas e outras razões, é

considerada uma prática recomendada para sempre listar explicitamente os atributos que você precisa.

Na cláusula SELECT, você pode atribuir seus próprios atalhos para as expressões que definem a

atributos resultado. Há uma série de formas de suporte do aliasing: *<expressão>* AS *<alias>*

como em empid AS employeeid, *<expression>* *<alias>* como em employeeid empid e *<alias>* =

*<Expressão>* como em employeeid = empid.

*MUNDO REAL* **um método preferido**

Nós preferimos usar o primeiro formulário com a cláusula AS, porque é padrão e vamos encontrá-lo

para ser o mais legível. A segunda forma é tanto ilegíveis e faz com que seja difícil de detectar uma

certa bug no código.

Considere a seguinte consulta.

SELECT empid, firstname lastname

FROM HR.Employees;

O desenvolvedor que foi o autor da consulta destina-se a devolver os atributos EmpID, nome,

e sobrenome, mas perdeu indicando a vírgula entre nome e sobrenome. o

consulta não falha; em vez disso, ele retorna o seguinte resultado.

Empid lastname

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

1 Sara

2 Don

3 Judy

...

Apesar de não ser a intenção do autor, o SQL Server interpreta o pedido como atribuir o

alias de sobrenome ao atributo firstName em vez de retornar ambos. Se você está acostumado a aliasing

expressões com a forma do espaço como uma prática comum, que será mais difícil para você mancha

tais erros.

Voltar para aliasing atributo intencional, há dois usos principais para aqueles. Um deles é renaming-

quando você precisa o atributo resultado a ser nomeados de forma diferente do que o atributo de fonte

exemplo, se você precisa nomear o atributo resultado employeeID vez de empid, como se segue.

SELECT empid AS employeeid, firstname, lastname

FROM HR.Employees;

Outro uso é para atribuir um nome a um atributo que resulta de uma expressão que seria

de outra forma, sem nome. Por exemplo, suponha que você precisa para gerar um atributo resultado de uma

expressão que concatena o atributo firstname, um espaço, e o atributo sobrenome. Vocês

usar a seguinte consulta.

SELECT empid, firstname + N' ' + lastname

FROM HR.Employees;

Você obter um resultado não-relacional porque o atributo resultado não tem nome.

empid

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

1 Sara Davis

2 Don Funk

3 Judy Lew

...

...

Por aliasing a expressão, você atribuir um nome ao atributo resultado, fazendo com que o resultado de

o relacional consulta, como se segue.

SELECT empid, firstname + N' ' + lastname AS fullname

FROM HR.Employees;

Aqui está uma forma abreviada do resultado desta consulta.

empid fullname

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

1 Sara Davis

2 Don Funk

3 Judy Lew

...

Lembre-se das discussões no Capítulo 1 que, se duplicatas são possíveis no resultado,

T-SQL não vai tentar eliminar aqueles menos que seja instruído. Um resultado com duplicatas é considerado

nonrelational porque as relações-estar sets-não são suposto ter duplicatas. Portanto,

se duplicatas são possíveis no resultado, e que pretende eliminá-los, a fim de voltar a

resultado relacional, você pode fazê-lo adicionando uma cláusula DISTINCT, como no seguinte.

SELECT DISTINCT country, region, city

FROM HR.Employees;

A tabela HR.EMPLOYEES tem nove linhas, mas cinco locais distintos; Assim, a saída desse

consulta tem cinco linhas.

country region city

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

UK NULL London

USA WA Kirkland

USA WA Redmond

USA WA Seattle

USA WA Tacoma

Há uma diferença interessante entre o padrão SQL e T-SQL em termos de mínimo

Requisitos de consulta SELECT. De acordo com o padrão SQL, uma consulta SELECT deve ter pelo

mínimo FROM e cláusulas SELECT. Por outro lado, T-SQL suporta uma consulta SELECT com apenas um

Cláusula SELECT e sem uma cláusula FROM. Tal consulta é como se emitida contra um imaginário

tabela que tem apenas uma linha. Por exemplo, a seguinte consulta é inválido acordo com a norma

SQL, mas é válido de acordo com a T-SQL.

SELECT 10 AS col1, 'ABC' AS col2;

A saída dessa consulta é uma única linha com atributos resultantes das expressões com

nomes atribuídos usando os aliases.

col1 col2

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

10 ABC

delimitar identificadores

Ao se referir a identificadores de atributos, esquemas, tabelas e outros objetos, há casos

em que você é obrigado a usar delimitadores vs. casos em que o uso de delimitadores é opcional.

T-SQL suporta tanto um formulário padrão para delimitar identificadores usando aspas duplas, como

em "vendas". "Pedidos", bem como uma forma de propriedade usando colchetes, como em [de vendas]. [Orders].

Quando o identificador é "normal", que delimita é opcional. Em um identificador comum, o identificador

está em conformidade com as regras para identificadores de formatação. As regras dizem que o primeiro caractere deve ser

uma carta no intervalo de à Z (minúsculas ou maiúsculas), sublinhado (\_), sinal de arroba (@), ou o número

placa (#). Caracteres subsequentes podem incluir letras, números decimais, no sinal, sinal de dólar ($),

sinal de número ou sublinhado. O identificador não pode ser uma palavra-chave reservada na T-SQL, não pode

têm incorporado espaços, e não deve incluir caracteres suplementares.

Um identificador que não esteja em conformidade com estas regras devem ser delimitados. Por exemplo, uma

atributo chamado 2006, é considerado um identificador irregular porque começa com um dígito, e

portanto, deve ser delimitada como "2006" ou [2006]. Um identificador regular, como pode ser y2006

referenciado sem delimitadores simplesmente como y2006, ou pode ser opcional com delimitadores. Vocês

pode preferir não para delimitar identificadores regulares porque os delimitadores tendem a atravancar o código.

**Checagem rápida**

1. Quais são as formas de aliasing um atributo em T-SQL?

2. O que é um identificador irregular?

**Quick Check Resposta**

1. Os formulários são *<expressão>* AS *<alias>,* *<expressão>* *<alias>* e

*<Alias>* = *<expressão>.*

2. Um identificador que não segue as regras para a formatação de identificadores; para

exemplo, ele começa com um dígito, tem um espaço incorporado ou é uma reservada T-SQL

palavra-chave.

**Quick Check**

1. What are the forms of aliasing an attribute in T-SQL?

2. What is an irregular identifier?

**Quick Check Answer**

1. The forms are <*expression*> AS <*alias*>, <*expression*> <*alias*>, and

<*alias*> = <*expression*>.

2. An identifier that does not follow the rules for formatting identifiers; for

example, it starts with a digit, has an embedded space, or is a reserved T-SQL

keyword.

Prática **Using the FR OM and SE LECT Clauses**

Nesta prática, você exercer o seu conhecimento de usar o FROM e cláusulas SELECT.

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 Escreva uma consulta simples e usar a tabela Aliases

Neste exercício, você pratica o uso da DE e SELECT cláusulas, incluindo o uso de

aliases de tabela.

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

2. Para a prática de escrever uma consulta simples que usa as de e selecione cláusulas, digite o

seguinte consulta e executá-lo.

USE TSQL2012;

SELECT shipperid, companyname, phone

FROM Sales.Shippers;

A instrução USE garante que você está conectado ao banco de dados TSQL2012 alvo.

A cláusula FROM indica que a tabela Sales.Shippers é a tabela consultada, ea

Cláusula SELECT projeta a CódigoDoTransitário atributos, companyname e telefone a partir desta

mesa. Aqui está o resultado da consulta.

shipperid companyname phone

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

1 Shipper GVSUA (503) 555-0137

2 Shipper ETYNR (425) 555-0136

3 Shipper ZHISN (415) 555-0138

3. Se havia mais de uma tabela envolvida na consulta e outra mesa tinha um atributo

chamado CódigoDoTransitário, você precisa prefixar o atributo CódigoDoTransitário com a tabela

nome, como em Shippers.shipperid. Para abreviar, você pode apelidar a tabela com um nome mais curto,

como S, em seguida, consulte o atributo como S.shipperid. Aqui está um exemplo de aliasing o

mesa e prefixar o atributo com o nome da nova tabela.

SELECT S.shipperid, companyname, phone

FROM Sales.Shippers AS S;

Exercício 2 Use alias e delimitadas Identificadores

Neste exercício, você pratica o uso de apelidos de coluna, incluindo a utilização de identificadores delimitados.

Tal como o seu ponto de partida, você usar a consulta a partir do passo 3 no exercício anterior.

1. Suponha que você queira renomear o telefone atributo resultado para **o número de telefone.** Aqui está um

tentar alias o atributo com o número de telefone identificador sem delimitadores.

SELECT S.shipperid, companyname, phone AS phone number

FROM Sales.Shippers AS S;

2. Este código falha porque o número de telefone não é um identificador regular, e, portanto, tem de

ser delimitado, como se segue.

SELECT S.shipperid, companyname, phone AS [phone number]

FROM Sales.Shippers AS S;

3. Lembre-se que T-SQL suporta tanto uma forma de propriedade para delimitar identificadores usando

colchetes, e a forma padrão usando aspas duplas, como no "telefone

número".

Resumo da lição

■■ A cláusula FROM é a primeira cláusula para ser logicamente processadas em uma consulta SELECT. Nisso

cláusula, você indica as tabelas que você deseja consultar e operadores de mesa. Você pode apelidar

tabelas na cláusula FROM com seus nomes escolhidos e, em seguida, usar o alias tabela como um

prefixo para nomes de atributos.

■■ Com a cláusula SELECT, você pode indicar expressões que definem os atributos de resultados.

Você pode atribuir seus próprios atalhos para os atributos de resultado, e, desta forma, criar um relacional

resultado. Se duplicatas são possíveis no resultado, você pode eliminar aqueles especificando

a cláusula DISTINCT.

■■ Se você usar identificadores regulares como objeto e nomes de atributos, usando delimitadores é opcional.

Se você usar identificadores irregulares, delimitadores são obrigatórios.

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. Qual é a importância da capacidade de atribuir aliases de atributos em T-SQL? (Escolha todas

que se aplicam.)

A. A capacidade de atribuir aliases de atributos é apenas uma característica estética.

B. Uma expressão que é baseado em um cálculo resulta em nenhum nome de atributo, a menos

você atribui um com um alias, e isso não é relacional.

C. T-SQL exige que todos os atributos resultado de uma consulta para ter nomes.

D. Usando aliases de atributos, você pode atribuir o seu próprio nome para um atributo resultado se você

preciso que ele seja diferente do que a fonte nome de atributo.

2. Quais são as cláusulas obrigatórias em uma consulta SELECT, de acordo com a T-SQL?

A. O FROM e cláusulas SELECT

Cláusulas B. O SELECT e WHERE

Cláusula C. A instrução SELECT

D. O FROM e WHERE

3. Qual das seguintes práticas são consideradas más práticas? (Escolha tudo o que se aplicam).

Colunas A. Aliasing usando a cláusula AS

Mesas B. Aliasing usando a cláusula AS

C. Não atribuição de pseudónimos da coluna quando a coluna é um resultado de um cálculo

D. Usando \* na lista SELECT

**Lição 2: Trabalhando com tipos de dados e**

**Funções internas**

Ao definir colunas em tabelas, parâmetros em procedimentos e funções e variáveis ​​em

Lotes T-SQL, você precisa escolher um tipo de dados para aqueles. O tipo de dados constrange os dados

que é suportada, além de encapsular comportamento que opera sobre os dados, expondo-

através de operadores e outros meios. Porque os tipos de dados são um componente tão fundamental

dos seus dados, tudo é construído em cima-suas escolhas de tipos de dados terá implicações dramáticas

para a sua aplicação em muitas camadas diferentes. Portanto, esta é uma área que deve

não ser tomada de ânimo leve, mas em vez tratado com muito cuidado e atenção. Essa é também a razão

porque este tópico é coberto tão cedo neste kit de treinamento, embora os primeiros capítulos do

foco kit de consulta, e só mais tarde capítulos tratam de definição de dados, como a criação e alteração

tabelas. Seu conhecimento de tipos é fundamental tanto para definição de dados e manipulação de dados.

T-SQL suporta muitas funções embutidas que podem ser utilizadas para manipular dados. Porque

funções operam em valores de entrada e retorno valores de saída, uma compreensão do embutido

funções anda de mãos dadas com uma compreensão dos tipos de dados.

Note-se que este capítulo não pretende ser uma cobertura exaustiva de todos os tipos e todas as funções

que a T-SQL suporta a isso exigiria um livro inteiro em seu próprio direito. Em vez disso, este capítulo

explica os fatores que você precisa considerar ao escolher um tipo de dados e aspectos-chave da

de trabalho com funções, usualmente no contexto de certos tipos de dados, como a data e hora de dados ou

dados de caracteres. Para obter detalhes e aspectos técnicos sobre tipos de dados, consulte Books Online para o SQL Server

2012, sob o tema "Tipos de dados (Transact-SQL)" no *http://msdn.microsoft.com/en-us/library*

*/ms187752(v=SQL.110).aspx.* Para obter detalhes sobre as funções internas, consulte o tópico "Funções internas

(Transact-SQL) "no *http://msdn.microsoft.com/en-us/library/ms174318(v=SQL.110).aspx.*

Após esta lição, você será capaz de:

■■ Escolha o tipo de dados apropriado.

■■ Escolha um tipo para suas chaves.

■■ Trabalhar com data e hora, além de dados de caracteres.

■■ Trabalhar com a expressão CASE e funções relacionadas.

Tempo de aula estimado: 50 minutos

Escolhendo o apropriado tipo de dados

Escolher os tipos de dados apropriados para os seus atributos é provavelmente um dos mais importantes

decisões que você vai fazer a respeito de seus dados. SQL Server suporta muitos tipos de dados

de diferentes categorias: exata numérico (INT, numérico), cadeias de caracteres (CHAR, VARCHAR),

Cadeias de caracteres Unicode (NCHAR, NVARCHAR), aproximado numérico (FLOAT, REAL), binário

strings (binário, VARBINARY), data e hora (data, hora, DATETIME2, smalldatetime,

DATA HORA,

DATETIMEOFFSET), e outros. Há muitas opções, por isso pode parecer

uma tarefa difícil, mas contanto que você seguir determinados princípios, você pode ser esperto sobre seu

escolhas, o que resulta numa base de dados robusto, consistente e eficiente.

Um dos grandes pontos fortes do modelo relacional é a importância que atribui à aplicação

de integridade de dados como parte do próprio modelo, em vários níveis. Um aspecto importante

escolher o tipo adequado para a sua dados é lembrar que um tipo é uma restrição. este

significa que ele tem um certo domínio de valores compatíveis e não irá permitir que os valores fora

domínio. Por exemplo, o tipo de data permite que apenas datas válidas. Uma tentativa de introduzir alguma coisa

que não é uma data, como "abc" ou "20120230", é rejeitada. Se você tem um atributo que é suposto

para representar uma data, tais como data de nascimento, e você usar um tipo, como INT ou CHAR, você não faz

beneficiar de built-in de validação de datas. Um tipo INT não vai impedir que um valor como 99999999

e um tipo CHAR não irá impedir que um valor, tais como "20120230".

Muito parecido com um tipo é uma restrição, não nulo é uma restrição bem. Se um atributo não é suposto

para permitir nulos, é importante para impor uma restrição NOT NULL como parte de sua definição.

Caso contrário, NULL irá encontrar o seu caminho em seu atributo.

Além disso, você quer ter certeza de que você não confunda a formatação de um valor com a sua

digitar. Às vezes, as pessoas usam cadeias de caracteres de datas de loja, porque eles pensam de armazenar uma

data em um certo formato. A formatação de um valor é suposto ser da responsabilidade do

aplicação quando os dados são apresentados. O tipo é uma propriedade do valor armazenado no banco de dados,

eo formato de armazenamento interno não deve ser sua preocupação. Este aspecto tem a ver com uma

princípio importante no modelo relacional chamado de *independência de dados física.*

Um tipo de dados encapsula comportamento. Ao usar um tipo inadequado, você perde todo o comportamento

que é encapsulado no tipo na forma de operadores e funções que o suportam. Como

um exemplo simples, para tipos que representam números, o sinal de mais (+) operador representa disso,

mas para cadeias de caracteres, o mesmo operador representa concatenação. Se você escolheu um inadequado

escreva para o seu valor, você às vezes tem que converter o tipo (explícita ou implicitamente),

e às vezes manipular o valor um pouco, a fim de tratá-lo como o que é suposto ser.

Outro princípio importante na escolha do tipo adequado para os seus dados é o tamanho. Frequentemente

um dos principais aspectos que afetam o desempenho da consulta é a quantidade de I / O os envolvidos. Uma consulta

que lê menos simplesmente tende a correr mais rápido. Quanto maior for o tipo que você usa, mais espaço de armazenamento

ele usa. Tabelas com muitos milhões de linhas, se não milhares, são comuns hoje em dia.

Quando você começa multiplicando o tamanho de um tipo pelo número de linhas na tabela, os números

pode rapidamente tornar-se significativo. Como exemplo, suponha que tem um atributo que representa

os resultados dos testes, que são inteiros no intervalo de 0 a 100. Usando um tipo de dados INT para este

propósito é um exagero. Seria usar 4 bytes por valor, enquanto um TINYINT usaria apenas 1 byte,

e é, portanto, o tipo mais apropriado neste caso. Do mesmo modo, para os dados que é suposto

para representar datas, as pessoas têm uma tendência a usar DATETIME, que utiliza 8 bytes de armazenamento.

Se o valor é suposto representar uma data sem uma hora, você deve usar DATE, que

usa apenas 3 bytes de armazenamento. Se o valor é suposto representar tanto a data ea hora,

deve considerar DATETIME2 ou SMALLDATETIME. O primeiro exige armazenamento entre 6 a 8

bytes (dependendo de precisão), e como um valor acrescentado, proporciona uma ampla gama de datas e

melhorado, precisão controlável. Este último utiliza apenas 4 bytes de armazenamento, por isso, enquanto a sua apoiada

intervalo de datas e precisão cobrir suas necessidades, você deve usá-lo. Em suma, você deve

usar o menor tipo que serve suas necessidades. Embora, naturalmente, isso não se aplica no curto

correr, mas no longo prazo. Por exemplo, utilizando um tipo int para uma chave de uma tabela que em um ponto

ou de outra vai crescer a um grau de bilhões de linhas é uma má idéia. Você deve estar usando BIGINT.

Mas usar INT para um atributo que representa os resultados dos testes ou DATETIME para data e hora valores

que exigem uma precisão minutos são escolhas ruins, mesmo quando se pensa a longo prazo.

Tenha muito cuidado com a imprecisa tipos float e real. As duas primeiras frases na

documentação que descreve estes tipos deve dar-lhe um bom senso de sua natureza: "aproxi-

tipos de dados número para uso com flutuação de dados numéricos de ponto. Dados de ponto flutuante

é aproximada; portanto, nem todos os valores da gama tipo de dados pode ser representado exactamente. "

(Você pode encontrar esta documentação no Books Online para o SQL Server 2012 artigo "flutuar e

verdadeira [Transact-SQL] "no *http://msdn.microsoft.com/en-us/library/ms173773.aspx.)* O benefício

nestes tipos é que eles podem representar números muito grandes e muito pequenas para além do que qualquer

outro tipo numérico que suporta SQL Server pode representar. Assim, por exemplo, se você precisa

representar números muito grandes ou muito pequenas para fins científicos e não precisam completa

precisão, você pode encontrar estes tipos útil. Eles também são bastante econômicos (4 bytes para o Real e

8 bytes para float). Mas não usá-los para as coisas que são supostamente para ser mais preciso.

**Problema Float** *Mundo Real*

Lembramo-nos de um caso em que um cliente usado FLOAT para representar números de código de barras de

produtos, e foi então surpreendido por não obter o produto certo ao digitalizar o

códigos de barras dos produtos. Além disso, recentemente, temos uma consulta sobre a conversão de um valor flutuante para

NUMERIC, resultando em um valor diferente do que o que foi digitado. Aqui é o caso.

DECLARE @f como float = '29545428.022495';

SELECT CAST (@f como numérico (28, 14)) como o valor;

Você consegue adivinhar o que a saída desse código é? Aqui está.

Valor

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

29545428,02249500200000

Como mencionado, alguns valores não podem ser representadas com precisão.

*Real World* **Float Trouble**

We remember a case where a customer used FLOAT to represent barcode numbers of

products, and was then surprised by not getting the right product when scanning the

products’ barcodes. Also, recently, we got a query about conversion of a FLOAT value to

NUMERIC, resulting in a different value than what was entered. Here’s the case.

DECLARE @f AS FLOAT = '29545428.022495';

SELECT CAST(@f AS NUMERIC(28, 14)) AS value;

Can you guess what the output of this code is? Here it is.

Value

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

29545428.02249500200000

As mentioned, some values cannot be represented precisely.

Em suma, certifique-se de usar tipos exata numéricos quando você precisa para representar valores

precisamente, e reservar a utilização dos tipos numéricos aproximados apenas para casos em que você está

certeza de que é aceitável para a aplicação.

Outro aspecto importante na escolha de um tipo tem a ver com a escolha de tipos fixos (CHAR,

NCHAR, BINÁRIO) vs. dinâmicos queridos (VARCHAR, nvarchar VARBINARY). Tipos fixos usar o

armazenamento para o tamanho indicado; por exemplo, CHAR (30) utiliza o armazenamento por 30 caracteres, se

você realmente especificar 30 caracteres ou menos. Isto significa que as atualizações não vai exigir a linha a

fisicamente expandir, e, portanto, não é necessária nenhuma mudança de dados. Assim, por atributos que são atualizados

com freqüência, onde o desempenho de atualização é uma prioridade, você deve considerar os tipos fixos. Nota

que quando a compressão é usado especificamente remar tipos lojas de compressão-SQL Server fixos

como as variáveis, mas com menos sobrecarga.

Tipos de variáveis ​​usar o armazenamento para o que você entra, mais um par de bytes de informação compensado

(ou 4 bits com a compactação de linha). Assim, por muito diversos tamanhos de cordas, se você usar variável

tipos que você pode salvar uma grande quantidade de armazenamento. Como já foi mencionado, a menos de armazenamento utilizado, a menos

existe para uma consulta para ler, e o mais rápido a consulta pode executar. Tipos de comprimento tão variável

são geralmente preferíveis em tais casos, quando o desempenho de leitura é uma prioridade.

Com cadeias de caracteres, há também a questão da utilização de tipos de caracteres normais (CHAR,

VARCHAR) vs. tipos de Unicode (NCHAR, NVARCHAR). O uso anterior 1 byte de armazenamento por

caráter e suporte apenas um idioma (com base nas propriedades de agrupamento) para além do Inglês. o

o uso dos últimos 2 bytes de armazenamento por caractere (a não ser comprimido) e suporte a vários idiomas.

Se um par de substituição é necessária, um personagem vai exigir 4 bytes de armazenamento. Assim, se os dados estiverem

em vários idiomas e você precisa de representar apenas um idioma além do Inglês em sua

dados, você pode se beneficiar do uso tipos de caracteres regulares, com requisitos de armazenamento mais baixos.

Quando os dados é internacional, ou seu aplicativo nativamente trabalha com dados Unicode, você deve

usar tipos de dados Unicode para que você não perder informações. Os requisitos de armazenamento de uma maior

Dados Unicode são mitigados começando com SQL Server 2008 R2 com a compressão Unicode.

Ao usar tipos que podem ter um comprimento que lhes estão associados, como CHAR e VARCHAR,

T-SQL suporta omitindo o comprimento e, em seguida, usa um tamanho padrão. No entanto, em diferentes contextos,

os padrões podem ser diferentes. É considerada uma das melhores práticas para estar sempre explícito sobre

o comprimento, como no CHAR (1) ou VARCHAR (30).

Ao definir atributos que representam a mesma coisa em diferentes tabelas, especialmente

aqueles que serão posteriormente utilizadas como colunas de junção (como a chave primária em uma tabela eo estrangeira

chave em outra) -é muito importante ser consistente com os tipos. Caso contrário, quando se comparam

um atributo com outro, SQL Server tem de aplicar a conversão implícita de um atributo de

escreva para o outro, e isso pode ter implicações de desempenho negativas, como a prevenção eficaz

uso de índices.

Você também quer ter certeza de que quando da indicação de um literal de um tipo, você pode usar o formulário correto.

Por exemplo, os literais de cadeias de caracteres regulares são delimitados por aspas simples, como em

'abc', enquanto literais de cadeias de caracteres Unicode são delimitados com um capital N e, em seguida, única

entre aspas, como em N'abc '. Quando uma expressão envolve elementos com diferentes tipos, SQL

Servidor precisa aplicar conversão implícita quando possível, e isso pode resultar em um desempenho

penalidades. Note-se que, em alguns casos, a interpretação de um literal pode não ser o que você pensa de forma intuitiva.

A fim de forçar um literal de ser de um certo tipo, você pode precisar aplicar conversão explícita

com funções como CAST, CONVERT, analisar ou TRY\_CAST, TRY\_CONVERT e TRY\_PARSE.

Como um exemplo, o literal 1 é considerado um INT pelo SQL Server em qualquer contexto. Se você precisar o

literal 1 para ser considerado, por exemplo, um pouco, você precisa converter o tipo do literal explicitamente, como

em CAST (1 AS BIT). Da mesma forma, o literal 4000000000 é considerado NUMERIC e não BIGINT. E se

você precisa do literal para ser o último, ELENCO uso (4000000000 AS BIGINT). A diferença entre

as funções sem a TRY e os seus homólogos com a TRY é que aqueles sem a TRY

falhar se o valor não é conversível, enquanto aqueles com o TRY retornar um NULL nesse caso. Para

exemplo, o seguinte código de falha.

SELECT CAST ( 'abc' AS INT);

Por outro lado, o código a seguir retorna um NULL.

SELECT TRY\_CAST ( 'abc' AS INT);

Quanto à diferença entre CAST, converter e analisar, com CAST, você indica o

expressão e o tipo de destino; com converter-se, há um terceiro argumento que representa o

estilo para a conversão, que é suportada por algumas conversões, como entre o personagem

cordas e data e hora valores. Por exemplo, converta (data, '1/2/2012', 101) converte o

cadeia de caracteres literal até à data usando estilo 101 que representa o padrão dos Estados Unidos. Com

PARSE, você pode indicar a cultura usando qualquer idioma suportado pelo Microsoft .NET

Estrutura. Por exemplo, analisar ( '1/2/2012' AS data usando "en-US") analisa a entrada literal como

a data usando uma cultura Estados Unidos Inglês.

Ao usar expressões que envolvem operandos de diferentes tipos, SQL Server normalmente converte

o que tem o menor tipo de dados precedência para aquele com o maior. Considerar

a expressão 1 + '1' como um exemplo. Um operando é INT eo outro é VARCHAR. Se vocês

olhar no Books Online para o SQL Server 2012, em "Tipo de dados Precedência (Transact-SQL)," no

*http://msdn.microsoft.com/en-us/library/ms190309.aspx,* você vai achar que INT precede VARCHAR;

Assim, SQL Server converte o valor implicitamente VARCHAR '1' para o valor inteiro 1, e o

resultado da expressão é, portanto, 2 e não a string '11'. Claro, você sempre pode tomar

controle usando conversão explícita.

Se todos os operandos da expressão são do mesmo tipo, que também vai ser o tipo de

o resultado, e você não pode querer que ele seja o caso. Por exemplo, o resultado da expressão

02/05 em T-SQL é o valor INT 2 e não o valor numérico de 2,5, porque ambos os operadores são

números inteiros, e, portanto, o resultado é um número inteiro. Se você estivesse lidando com duas colunas de números inteiros,

como col1 / col2, e queria a divisão de ser numérico, você precisaria para converter as colunas

explicitamente, como em CAST (col1 como numérico (12, 2)) / CAST (col2 como numérico (12, 2)).

A escolha de um tipo de dados para Chaves

Ao definir as chaves inteligentes em suas chaves tabelas, ou seja com base em atributos já existentes

derivado da aplicação-não há nenhuma pergunta sobre os tipos, porque você já

escolheu aqueles para os seus atributos. Mas quando você precisa criar keys-os substitutos que são

adicionados exclusivamente com o fim de ser utilizado como teclas de que você precisa para determinar um apropriado

escreva para o atributo em adição a um mecanismo para gerar os valores de chave. A realidade

é que você vai ouvir muitas opiniões diferentes sobre o que é a melhor solução de alguns baseados em

teoria, e alguns apoiada pela evidência empírica. Mas diferentes sistemas e diferentes cargas de trabalho

pode acabar com diferentes soluções óptimas. Além do mais, em alguns sistemas, escrever

desempenho pode ser a prioridade, enquanto que em outros, o desempenho de leitura é. uma solução de

pode fazer as inserções mais rápido, mas as leituras mais lento, e outra solução poderia funcionar a outra

caminho de volta. No final do dia, para fazer escolhas inteligentes, é importante saber a teoria,

aprender sobre as experiências dos outros, mas, eventualmente, ter certeza de que você executar benchmarks na

sistema de destino. A este respeito, uma frase em um livro chamado *Bubishi* por Patrick McCarthy (Tuttle

Publishing, 2008) é muito apropriado. Ele diz: "A sabedoria é o conhecimento em ação."

Note-se que esta seção se refere a elementos como objetos de seqüência, a propriedade coluna de identidade,

e índices, que são cobertos com mais detalhes mais adiante no livro. Capítulo 11, "Outros Dados

Aspectos modificação ", abrange objetos de seqüência e a propriedade de identidade, e Capítulo 15,

"A implementação de índices e estatísticas", abrange índices. Você pode querer voltar a esta secção

depois de terminar os capítulos.

As opções típicas que as pessoas usam para gerar chaves substitutas são:

■■ **A propriedade coluna de identidade**Uma propriedade que gera automaticamente chaves no

um atributo de um tipo numérico com uma escala de 0; ou seja, qualquer tipo integer (TINYINT,

SMALLINT, INT, BIGINT) ou NUMERIC / DECIMAL com uma escala de 0.

■■ **A sequência objeto**um objeto independente no banco de dados a partir da qual você pode

obter novos valores de seqüência. Como identidade, ele suporta qualquer tipo numérico com uma escala

0. Ao contrário de identidade, não está amarrado a uma coluna particular; Em vez disso, como mencionado, é uma

objeto independente no banco de dados. Você também pode solicitar um novo valor a partir de uma sequência

objeto antes de usá-lo. Há um número de outras vantagens sobre identidade que vai

ser abordados no Capítulo 11.

■■ **nonsequential GUI Ds**Você pode gerar identificadores exclusivos globais não seqüenciais para

ser armazenados em um atributo de um tipo UNIQUEIDENTIFIER. Você pode usar a função T-SQL

NEWID para gerar um novo GUID, possivelmente, invocando-a com uma expressão padrão anexado

para a coluna. Você também pode gerar um de qualquer lugar, por exemplo, o cliente-

usando uma interface de programação de aplicações (API), que gera um novo GUID. o

GUIDs são garantidos para ser único em espaço e tempo.

■■ **Sequential GUI Ds**Você pode gerar GUIDs sequenciais dentro da máquina, usando

a função NEWSEQUENTIALID T-SQL.

■■ **soluções personalizadas**Se você não quiser usar as ferramentas embutidas que o SQL Server fornece

para gerar chaves, você precisa desenvolver sua própria solução personalizada. O tipo de dados

para a chave, em seguida, depende de sua solução.

*Dica exam*

Compreender as ferramentas embutidas T-SQL fornece para a geração de chaves substitutas como o

sequência objecto, propriedade coluna de identidade, eo NEWID e NEWSEQUENTIALID

funções, e seu impacto no desempenho, é uma habilidade importante para o exame.

*Exam Tip*

Understanding the built-in tools T-SQL provides for generating surrogate keys like the

sequence object, identity column property, and the NEWID and NEWSEQUENTIALID

functions, and their impact on performance, is an important skill for the exam.

Uma coisa a considerar em relação a sua escolha de gerador de chave substituta e os dados

tipo envolvido é o tamanho do tipo de dados. Quanto maior for o tipo, mais armazenamento é necessário,

e, consequentemente, a mais lenta são lê. A solução usando um tipo de dados INT requer 4 bytes por

valor, BIGINT requer 8 bytes, UNIQUEIDENTIFIER requer 16 bytes, e assim por diante. o armazenamento

requisitos para a sua chave substituta pode ter um efeito em cascata se o seu índice de cluster

é definido nas mesmas colunas de chave (o padrão para uma restrição de chave primária). o cluster

colunas de chave do índice são usadas por todos os índices agrupados internamente como os meios para localizar linhas

na mesa. Então, se você definir um índice agrupado numa coluna x, e sem clusters índices-one

na coluna de um, um de b, e um no c-seus índices não clusterizados são criados internamente

coluna (A, X), (b, x), e (C, x), respectivamente. Em outras palavras, o efeito é multiplicado.

Quanto ao uso de chaves sequenciais (como com a identidade, sequência, e NEWSEQUENTIALID)

vs. aqueles não seqüenciais (como acontece com NEWID ou um gerador de chaves personalizado randomizados), existem

vários aspectos a considerar.

Começando com chaves sequenciais, todas as linhas de ir para a extremidade direita do índice. Quando uma página é

completo, SQL Server aloca uma nova página e preenche-lo. Isso resulta em menos fragmentação no índice,

o que é benéfico para o desempenho de leitura. Além disso, as inserções pode ser mais rápido quando uma única sessão

é carregar os dados, e os dados residem em uma única unidade ou um pequeno número de unidades.

No entanto, com os subsistemas de armazenamento high-end que têm muitos fusos, a situação pode ser

diferente. Ao carregar dados de várias sessões, você vai acabar com a página de contenção trinco

( *Travas*são objetos usado para sincronizar o acesso a páginas de banco de dados) contra o mais à direita

páginas de lista vinculada do nível folha de índice. Este gargalo impede o uso do rendimento integral

do subsistema de armazenamento.

Observe que, se você decidir usar chaves sequenciais e você estiver usando os numéricos, você pode

sempre começam com o valor mais baixo no tipo de usar toda a gama. Por exemplo, em vez de

começando com 1 em um tipo INT, você poderia começar com 2,147,483,648.

Considere chaves não seqüenciais, tais como aqueles aleatórios gerados com NEWID ou com um

solução personalizada. Ao tentar forçar uma linha em uma página já está cheio, o SQL Server executa uma

A página clássico split-aloca uma nova página e move metade das linhas da página original para

o novo. A divisão da página tem um custo, além de que resulta em fragmentação do índice. fragmentação do índice

pode ter um impacto negativo sobre o desempenho da lê. No entanto, em termos de inserção

desempenho, se o subsistema de armazenamento contém muitos fusos e você está carregando dados de

várias sessões, a ordem aleatória pode realmente ser melhor do que sequencial apesar das divisões.

Isso porque não há nenhum ponto quente na extremidade direita do índice, e você usar o subsistema de armazenamento do

rendimento disponível melhor. Um bom exemplo para um ponto de referência que demonstra este

estratégia pode ser encontrado em um blog por Thomas Kejser em *http://blog.kejser.org/2011/10/05*

*/ impulsionando-insert-speed-by-chaves de geração escalável /*.

Note-se que divide e índice de fragmentação pode ser mitigado pelo índice periódica reconstrói como

parte da manutenção habituais actividades de supondo que você tem uma janela disponível para isso.

Se, por razões acima mencionadas você decidir confiar em chaves geradas de forma aleatória, você

ainda terá de decidir entre GUIDs e uma solução gerador de chaves personalizado aleatória. como já

mencionado, GUID são armazenadas em um tipo UNIQUEIDENTIFIER que é de 16 bytes de tamanho; isso é

grande. Mas um dos principais benefícios de GUID é o facto de que eles podem ser gerados em qualquer lugar

e não entrar em conflito através do tempo e espaço. Você pode gerar GUIDs não apenas em SQL Server usando

a função NEWID, mas em qualquer lugar, usando APIs. Caso contrário, você poderia vir acima com um costume

solução que gera chaves aleatórias menores. A solução pode até ser uma mistura de uma ferramenta built-in

e alguns ajustes no topo. Por exemplo, você pode encontrar uma solução criativa por Wolfgang 'Rick'

Kutschera no *http://dangerousdba.blogspot.com/*

*2011/10 / world.html dia-sequências-salva-*.

Rick utiliza o objeto de seqüência SQL Server, mas vira

os bits dos valores para que a inserção

é distribuída através da folha de índice.

Para concluir esta seção sobre teclas e tipos de chaves, lembre-se que existem múltiplos

opções. Menor é geralmente melhor, mas depois há a questão do hardware que você

usar, e onde suas prioridades de desempenho são. Lembre-se também que, embora seja muito importante

para fazer palpites, é também importante para soluções de referência no alvo

meio Ambiente.

Funções de Data e Hora

T-SQL suporta um número de data e hora funções que permitem manipular a sua data

e dados de tempo. Suporte para data e hora funções continua melhorando, com as duas últimas versões

do SQL Server adicionando uma série de novas funções.

Esta seção cobre algumas das funções importantes apoiados pela T-SQL e fornece

alguns exemplos. Para a lista completa, bem como os detalhes técnicos e elementos de sintaxe, consulte

Manuais Online do SQL Server 2012, sob o Data tópico "e dados em tempo tipos e funções

(Transact-SQL) "no *http://msdn.microsoft.com/en-us/library/ms186724(v=SQL.110).aspx*.

Atual Data e Hora

Uma importante categoria de funções é a categoria que retorna a data e hora atual.

As funções nesta categoria são GETDATE, CURRENT\_TIMESTAMP, GETUTCDATE, SYSDATETIME,

SYSUTCDATETIME, e SYSDATETIMEOFFSET.

GETDATE é T-específicas do SQL, retornando a data ea hora atual na instância do SQL Server

estiver conectado a como um tipo de dados DATETIME. CURRENT\_TIMESTAMP é o mesmo, só que é

padrão, e, portanto, a uma recomendada para uso. SYSDATETIME e SYSDATETIMEOFFSET

são semelhantes, retornando somente os valores como o DATETIME2 mais preciso e DATETIMEOFFSET

tipos (incluindo offset), respectivamente. Note-se que não há funções internas para retornar

a data atual ou a hora atual; para obter tais informações, basta lançar o SYSDATETIME

funcionar a data ou a hora, respectivamente. Por exemplo, para obter a data atual, use

Elenco (SYSDATETIME () como a data). A função GETUTCDATE retorna a data e hora atuais

em termos UTC como um tipo DATETIME e SYSUTCDATE faz o mesmo, só retornando o resultado

como o tipo de DATETIME2 mais precisa.

Data e Hora Parts

Esta seção abrange de data e hora funções que quer extrair uma parte de uma data e hora

valor (como DATEPART) ou construir uma data e valor de tempo a partir de peças (como DATEFROMPARTS).

Usando a função DATEPART, você pode extrair de uma data de entrada e valor de tempo a desejar

parte, como um ano, minuto ou nanossegundo, e retornar a parte extraída como um inteiro. Para

exemplo, o DATEPART expressão (mês, '20120212') retorna 2. T-SQL fornece as funções

Ano, mês, e dia como abreviaturas para DATEPART, e você não precisa especificar o

parte. A função DATENAME é semelhante ao DATEPART, só que retorna o nome da peça

como uma sequência de caracteres, em oposição ao valor inteiro. Note que a função é dependentes do idioma.

Ou seja, se o idioma eficaz na sua sessão é us\_english, a expressão

DATENAME (mês, '20120212') retorna 'fevereiro', mas para o italiano, ele retorna "febbraio".

T-SQL fornece um conjunto de funções que construir um data desejada e valor de tempo de seu numérica

peças. Você tem essa função para cada um dos seis tipos de data e hora disponíveis: DATEFROMPARTS,

DATETIME2FROMPARTS, DATETIMEFROMPARTS, DATETIMEOFFSETFROMPARTS,

SMALLDATETIMEFROMPARTS, e TIMEFROMPARTS. Por exemplo, para construir um valor de data de

suas partes, você usaria uma expressão como DATEFROMPARTS (2012, 02, 12).

Finalmente, a função FIMMÊS calcula a respectiva data de fim de meses para a entrada

data e valor de tempo. Por exemplo, suponha que hoje era 12 de fevereiro de 2012. A expressão

EOMONTH (SYSDATETIME ()), então, retornar a data '2012-02-29'. Esta função suporta uma

segunda entrada opcional que indica quantos meses para adicionar ao resultado.

Adicionar e Diff

T-SQL suporta funções de adição e data diferença e tempo chamado DATEADD e DATEDIFF.

DATEADD é uma função muito utilizada. Com ele, você pode adicionar um número solicitado

de unidades de uma parte especificada para um valor de data e hora especificadas. Por exemplo, a expressão

DATEADD (ano, 1, '20120212') acrescenta um ano para a data de entrada 12 de fevereiro de 2012.

DATEDIFF é outra função comumente usado; ele retorna a diferença em termos de uma solicitado

parte entre dois valores de data e hora. Por exemplo, a expressão DATEDIFF (dia,

"20110212", "20120212") calcula a diferença em dias entre 12 de fevereiro de 2011 e

12 de fevereiro de 2012, retornando o valor Nota 365. que esta função olha apenas para as partes

do que foi solicitado e acima na hierarquia não-data e hora abaixo. Por exemplo,

o DATEDIFF expressão (ano, '20111231', '20120101') olha apenas para a parte do ano e, portanto,

retorna 1. Ela não olha para o mês e dia partes dos valores.

compensar

T-SQL suporta duas funções relacionadas com a data e hora valores com um deslocamento: SWITCHOFFSET

e TODATETIMEOFFSET.

Com a função SWITCHOFFSET, você pode retornar um valor DATETIMEOFFSET entrada em um

solicitado termo offset. Por exemplo, considere o SWITCHOFFSET expressão (SYSDATETIMEOF

FDefina (), "-08: 00 '). Independentemente de o deslocamento da instância que você está conectado, você pedir para

apresentar a data atual e valor de tempo em termos de compensação "-08: 00 '. Se o sistema é compensado é, digamos,

"-05: 00 ', a função vai compensar isso subtraindo três horas a partir do valor de entrada.

A função TODATETIMEOFFSET é usada para uma finalidade diferente. Você usá-lo para construir uma

valor DATETIMEOFFSET de duas entradas: a primeira é um valor de data e hora em que não é offsetaware,

eo segundo é o deslocamento. Você pode usar esta função quando a migração de dados que

não é compensada-aware, onde você manter a data local e valor de tempo em um atributo, eo

compensada em outro, para compensar-aware dados. Digamos que você tenha a data local e hora em um atribute

chamada dt, e o deslocamento em um atributo chamado theoffset. Você adiciona um atributo chamado dto de

um tipo DATETIMEOFFSET para a mesa. Você, então, atualizar o novo atributo para a expressão

TODATETIMEOFFSET (dt, theoffset), e depois soltar os atributos originais dt e theoffset

a partir da tabela.

O código a seguir demonstra o uso de ambas as funções.

SELECT

SWITCHOFFSET('20130212 14:00:00.0000000 -08:00', '-05:00') AS [SWITCHOFFSET],

TODATETIMEOFFSET('20130212 14:00:00.0000000', '-08:00') AS [TODATETIMEOFFSET];

Aqui é a saída desse código.

SWITCHOFFSET TODATETIMEOFFSET

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

2013-02-12 17:00:00.0000000 -05:00 2013-02-12 14:00:00.0000000 -08:00

Funções de caracteres

T-SQL não foi realmente concebido para suportar muito sofisticada manipulação de cadeia de caracteres

funções, de modo que você não vai encontrar um conjunto muito grande de tais funções. Esta seção descreve o personagem

funções de cadeia que T-SQL suporta, organizados em categorias.

Concatenação

cadeia de caracteres de concatenação é uma necessidade muito comum. T-SQL suporta duas maneiras para concatenar

corda-um com o operador de adição (+), e outro com a função CONCAT.

Aqui está um exemplo para concatenação de cadeias de caracteres em uma consulta usando o operador +.

SELECT empid, country, region, city,

country + N',' + region + N',' + city AS location

FROM HR.Employees;

Here’s the result of this query.

empid country region city location

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

1 USA WA Seattle USA,WA,Seattle

2 USA WA Tacoma USA,WA,Tacoma

3 USA WA Kirkland USA,WA,Kirkland

4 USA WA Redmond USA,WA,Redmond

5 UK NULL London NULL

6 UK NULL London NULL

7 UK NULL London NULL

8 USA WA Seattle USA,WA,Seattle

9 UK NULL London NULL

Observe que quando qualquer uma das entradas é NULL, o operador + retorna um NULL. Isso é

comportamento padrão que pode ser alterado, desligando a opção de sessão chamado CONCAT\_NULL\_

YIELDS\_NULL\_INPUT, embora não seja recomendado para dependem do comportamento fora do padrão. Se vocês

quer substituir uma NULL com uma cadeia vazia, há uma série de maneiras para você fazer isso

programaticamente. Uma opção é usar COALESCE (< *expressão*>, ''). Por exemplo, neste dados,

única região pode ser NULL, assim você pode usar a seguinte consulta para substituir uma vírgula além de região

com uma cadeia vazia quando a região é NULL.

SELECT empid, country, region, city,

country + COALESCE( N',' + region, N'') + N',' + city AS location

FROM HR.Employees;

Another option is to use the CONCAT function which, unlike the + operator, substitutes a

NULL input with an empty string. Here’s how the query looks.

SELECT empid, country, region, city,

CONCAT(country, N',' + region, N',' + city) AS location

FROM HR.Employees;

Here’s the output of this query.

empid country region city location

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

1 USA WA Seattle USA,WA,Seattle

2 USA WA Tacoma USA,WA,Tacoma

3 USA WA Kirkland USA,WA,Kirkland

4 USA WA Redmond USA,WA,Redmond

5 UK NULL London UK,London

6 UK NULL London UK,London

7 UK NULL London UK,London

8 USA WA Seattle USA,WA,Seattle

9 UK NULL London UK,London

Outra opção consiste em utilizar a função CONCAT que, ao contrário do operador +, um substitui

entrada NULL com uma cadeia vazia. Veja como a consulta parece.

Observe que, desta vez, quando a região era NULL, ele foi substituído por uma cadeia vazia.

Extração Substring e posição

Esta seção abrange as funções que você pode usar para extrair uma substring de uma string, e identificar

a posição de uma substring dentro de uma cadeia.

Com a função SUBSTRING, você pode extrair uma substring de uma string dada como o primeiro

argumento, começando com a posição determinada como o segundo argumento e um comprimento dado como o

terceiro argumento. Por exemplo, o SUBSTRING expressão ( "abcde ', 1, 3) retorna' abc '. Se o terceiro

argumento é maior do que o que seria chegar ao fim da cadeia, a função não faz

falhou; em vez disso, ele apenas extrai o substring até o fim da cadeia.

As funções ESQUERDA e DIREITA extrair um número solicitado de caracteres da esquerda,

extremidades direita da cadeia de entrada, respectivamente. Por exemplo, LEFT ( 'ABCDE', 3) retorna 'abc' e

RIGHT ( 'ABCDE', 3) retorna «CDE».

A função CHARINDEX retorna a posição da primeira ocorrência da cadeia fornecida

como o primeiro argumento dentro da cadeia fornecida como o segundo argumento. Por exemplo, a expressão

CHARINDEX ( '', 'Itzik Ben-Gan') olha para a primeira ocorrência de um espaço no segundo

entrada, retornando 6 neste exemplo. Note que você pode fornecer um terceiro argumento indicando

a função de onde começar a procurar.

Você pode combinar ou ninho, funções na mesma expressão. Por exemplo, suponha que você

consultar uma tabela com um atributo chamado fullname formatado como '< *primeiro*> < *última*>', e você precisa

escrever uma expressão que extrai a primeira parte do nome. Você pode usar a seguinte expressão.

LEFT (fullname, CHARINDEX ( '', nome completo) - 1)

T-SQL também suporta uma função chamada PATINDEX que, como CHARINDEX, você pode usar para

localizar a primeira posição de uma cadeia dentro de outra cadeia. Mas enquanto que com CHARINDEX você está

procurando uma string constante, com PATINDEX você está procurando um padrão. O padrão é

formado muito semelhantes aos padrões de gosto que você provavelmente está familiarizado com, onde você usa

wildcards como% para qualquer cadeia, \_ para um único caractere, e colchetes ([]), representando uma

único caractere de uma determinada lista ou intervalo. Se você não estiver familiarizado com tal construção padrão,

Consulte os tópicos "PATINDEX (Transact-SQL)" e "Like (Transact-SQL)" nos manuais online

para o SQL Server 2012 em *http://msdn.microsoft.com/en-us/library/ms188395(v=SQL.110).aspx*

e *http://msdn.microsoft.com/en-us/library/ms179859(v=SQL.110).aspx*. Como um exemplo, a

PATINDEX expressão ( '% [0-9]%', 'abcd123efgh') procura a primeira ocorrência de um dígito (a

caracteres na gama 0-9) na segunda entrada, retornando à posição seis neste caso.

Cadeia Comprimento

T-SQL fornece duas funções que você pode usar para medir o comprimento de uma entrada de valor

LEN e DATALENGTH.

A função LEN devolve o comprimento de uma cadeia de entrada em termos de número de caracteres.

Note que ele retorna o número de caracteres, não bytes, se a entrada é um regular

caractere ou cadeia de caracteres Unicode. Por exemplo, a expressão LEN (N'xyz ') retorna 3. Se

existem espaços à direita, LEN remove-los.

A função DATALENGTH retorna o comprimento da entrada, em termos de número de bytes.

Isto significa, por exemplo, que, se a entrada é uma cadeia de caracteres Unicode, contará 2 bytes

por personagem. Por exemplo, a expressão DATALENGTH (N'xyz ') retorna 6. Note-se também que,

ao contrário LEN, a função DATALENGTH não remove espaços à direita.

Alteração de Cordas

T-SQL suporta um número de funções que você pode usar para aplicar alterações a uma cadeia de entrada.

Essas são REPLACE, replicar e material.

Com a função REPLACE, você pode substituir, em uma cadeia de entrada fornecido como o primeiro argumento

todas as ocorrências da cadeia fornecida como o segundo argumento, com a string fornecidas

como o terceiro argumento. Por exemplo, a expressão REPLACE ( '. 1.2.3.', '.', '/') Substitui todas as ocorrências

de um ponto (.) com uma barra (/), retornando a string '/ 1/2/3 /'.

A função REPLICATE lhe permite replicar uma cadeia de entrada de um número solicitado de

vezes. Por exemplo, a expressão REPLICATE ( '0', 10) replica a cadeia '0' dez vezes, voltando

'0000000000'.

A função STUFF opera em uma cadeia de entrada fornecido como o primeiro argumento; então, a partir

a posição do carácter indicado como o segundo argumento, exclui o número de caracteres

indicado pelo terceiro argumento. Em seguida, ele insere nessa posição o texto indicado como o

quarto argumento. Por exemplo, o material de expressão ( ', X, Y, Z', 1, 1, '') remove o primeiro caractere

da cadeia de entrada, retornando 'x, y, z'.

formatação de Cordas

Esta seção abrange as funções que você pode usar para aplicar opções de formatação de uma cadeia de entrada.

Aqueles são o superior, inferior, LTRIM, RTRIM, e funções de formato.

Os primeiros quatro funções são auto-explicativos (formulário caixa de entrada, a forma minúscula

da entrada, a entrada após a remoção dos espaços à esquerda, ea entrada após a remoção dos espaços à direita).

Note-se que não há nenhuma função TRIM que remove os espaços à esquerda e à direita; alcançar

isso, você precisa para aninhar uma chamada de função dentro de outra, como em RTRIM (LTRIM (<input>)).

Com a função de formatação, você pode formatar um valor de entrada com base em uma seqüência de formato, e

opcionalmente especificar a cultura como uma terceira entrada, quando relevante. Você pode usar qualquer cadeia de formato

suportado pelo .NET Framework. (Para mais detalhes, consulte os tópicos "FORMATO (Transact-SQL)" e

"Formatando tipos" na *http://msdn.microsoft.com/en-us/library/hh213505(v=sql.110).aspx*e

*http://msdn.microsoft.com/en-us/library/26etazsy.aspx*.) Como um exemplo, o formato de expressão (

1759, '000000000') formata o número de entrada como uma cadeia de caracteres com um tamanho fixo de 10

caracteres com zeros à esquerda, retornando '0000001759'.

Expressão CASE e funções relacionadas

T-SQL suporta uma expressão chamada CASE e uma série de funções relacionadas que você pode usar

para aplicar a lógica condicional para determinar o valor retornado. Muitas pessoas referem-se incorretamente

CASE como uma declaração. Uma declaração realiza algum tipo de ação ou controla o fluxo do

código, e que não o faz CASE é; CASE retorna um valor e, portanto, é uma expressão.

A expressão CASE tem duas formas de o *simples*formulário eo *procurou*formulário. Aqui está um

exemplo da forma simples caso lavrado contra a TSQL2012 base de dados exemplo.

SELECT productid, productname, unitprice, discontinued,

CASE discontinued

WHEN 0 THEN 'No'

WHEN 1 THEN 'Yes'

ELSE 'Unknown'

END AS discontinued\_desc

FROM Production.Products;

A forma simples compara uma *expressão de entrada*(neste caso, o atributo descontinuado)

para múltiplo escalar possível *quando as expressões*(neste caso, 0 e 1), e retorna o *resultado*

*expressão*(neste caso, "Não" e "Sim", respectivamente) associado com o primeiro jogo. Se há

páreo e uma cláusula ELSE é especificado, a *expressão outra coisa*(neste caso, 'Desconhecido') é retornado.

Se não há nenhuma cláusula ELSE, o padrão é NULL MAIS. Aqui está uma forma abreviada do

saída desta consulta.

productid productname unitprice discontinued discontinued\_desc

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

1 Product HHYDP 18.00 0 No

2 Product RECZE 19.00 0 No

3 Product IMEHJ 10.00 0 No

4 Product KSBRM 22.00 0 No

5 Product EPEIM 21.35 1 Yes

...

...

A forma procurou da expressão CASE é mais flexível. Em vez de comparar uma entrada

expressão de múltiplas expressões possíveis, ele usa predicados nas cláusulas WHEN, eo

primeiro predicado que avalia a verdade determina que quando a expressão é retornado. Se não houver nenhum

verdade, a expressão CASE retorna a expressão mais. Aqui está um exemplo.

SELECT productid, productname, unitprice,

CASE

WHEN unitprice < 20.00 THEN 'Low'

WHEN unitprice < 40.00 THEN 'Medium'

WHEN unitprice >= 40.00 THEN 'High'

ELSE 'Unknown'

END AS pricerange

FROM Production.Products;

Neste exemplo, a expressão CASE retorna uma descrição de preço unitário do produto

alcance. Quando o preço unitário é inferior a US $ 20,00, ele retorna 'Low', quando é US $ 20,00 ou mais e

abaixo de US $ 40,00, ele retorna 'Medium', e quando é US $ 40,00 ou mais, ele retorna 'High'. Há uma

cláusula ELSE para a segurança; se a entrada é NULL, a expressão outra devolvido é 'desconhecido'. Aviso prévio

que o segundo quando predicado não precisa verificar se o valor é de R $ 20,00 ou mais

explicitamente. Isso porque o quando os predicados são avaliados em ordem eo primeiro quando predicado

não foi avaliado como verdadeiro. Aqui está uma forma abreviada da saída dessa consulta.

productid productname unitprice pricerange

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

1 Product HHYDP 18.00 Low

2 Product RECZE 19.00 Low

3 Product IMEHJ 10.00 Low

4 Product KSBRM 22.00 Medium

5 Product EPEIM 21.35 Medium

...

T-SQL suporta um número de funções que podem ser considerados como abrevia da CASE

expressão. Essas são as funções COALESCE e NULLIF padrão e não padrão do

ISNULL, IIF e escolha

A função COALESCE aceita uma lista de expressões como entrada e retorna a primeira vez que

não é NULL, ou NULL se todos são nulos. Por exemplo, o COALESCE expressão (NULL, 'x', 'y')

retorna 'x'. De modo mais geral, a expressão:

COALESCE (<exp1>, <exp2>, ..., <expn>)

é semelhante à que se segue.

CASE

WHEN <exp1> IS NOT NULL THEN <exp1>

WHEN <exp2> IS NOT NULL THEN <exp2>

…

WHEN <expn> IS NOT NULL THEN <expn>

ELSE NULL

END

Um uso típico de COALESCE é substituir um NULL com outra coisa. Por exemplo, a

expressão COALESCE (região, '') retorna região se não é NULL e retorna uma string vazia se

é NULL.

T-SQL suporta uma função fora do padrão chamado ISNULL que é semelhante ao padrão

COALESCE, mas é um pouco mais limitado no sentido de que ele suporta apenas duas entradas. Gostar

COALESCE, ele retorna a primeira entrada que não é NULL. Assim, em vez de COALESCE (região, ''), você

poderia usar ISNULL (região, ''). Geralmente, recomenda-se manter características padrão, a menos

há alguma flexibilidade ou desempenho vantagem no recurso fora do padrão que é maior

prioridade. ISNULL é realmente mais limitado do que coalesce, de modo geral, recomenda-se a

furar a se aglutinar.

Existem algumas diferenças sutis entre COALESCE e ISNULL que você pode ser

interessado. Uma diferença é que na entrada determina o tipo de saída. Considere o

seguinte código.

DECLARAR

@x AS VARCHAR (3) = NULL,

@y como VARCHAR (10) = '1234567890';

SELECT COALESCE (@x, @y) AS [COALESCE], ISNULL (@x, @y) AS [ISNULL];

Aqui é a saída desse código.

COALESCE ISNULL

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

1234567890 123

Observe-se que o tipo da expressão COALESCE é determinada pelo elemento devolvido,

Considerando o tipo de expressão ISNULL é determinado pela primeira entrada.

A outra diferença entre COALESCE e ISNULL é quando você estiver usando SELECT INTO,

que é discutido em mais detalhes no Capítulo 11. Suponha que a lista SELECT de uma SELECT INTO

declaração contém as expressões COALESCE (col1, 0) AS newcol1 vs. ISNULL (col1, 0) AS

newcol1. Se o col1 atributo de origem é definido como NOT NULL, ambas as expressões produzirá

um atributo na tabela de resultados definidas como NOT NULL. No entanto, se o atributo de origem é COL1

definido como permitindo nulos, COALESCE criará um atributo resultado permitindo nulos, enquanto

ISNULL criará um que não permite nulos.

*Dica exam*

COALESCE e ISNULL pode afetar o desempenho quando você está combinando conjuntos; por exemplo,

com junta-se ou quando você está filtrando dados. Considere um exemplo onde você tem duas tabelas

T1 e T2 e você precisa se juntar a eles com base em uma partida entre T1.col1 e T2.col1. o

atributos não permitir nulos. Normalmente, uma comparação entre dois nulos produz desconhecida,

e isso faz com que a linha a ser descartado. Você quer tratar dois nulos como iguais. o que

alguns fazem em tal caso é o uso COALESCE ou ISNULL para substituir um NULL com um valor que

eles sabem não pode aparecer nos dados. Por exemplo, se os atributos são números inteiros, e você

saber que você tem inteiros única positivas em seus dados (você pode até ter restrições que

garantir isso), você pode tentar usar o COALESCE predicado (T1.col1, -1) = COALESCE (T2.

COL1, -1), ou ISNULL (T1.col1, -1) = ISNULL (T2.col1, -1). O problema com esta forma é que,

porque você aplicar manipulação para os atributos que você está comparando, o SQL Server não vai

contar com índice de ordenação. Isto pode resultar em não utilizar índices disponíveis de forma eficiente. Em vez disso, ele

É recomendável usar a forma mais: T1.col1 = T2.col1 OR (T1.col1 é nulo e T2.col1

IS NULL), que SQL Server entende como apenas uma comparação que considera nulos como

igual. Com este formulário, o SQL Server pode usar eficientemente a indexação.

*Exam Tip*

COALESCE and ISNULL can impact performance when you are combining sets; for example,

with joins or when you are filtering data. Consider an example where you have two tables

T1 and T2 and you need to join them based on a match between T1.col1 and T2.col1. The

attributes do allow NULLs. Normally, a comparison between two NULLs yields unknown,

and this causes the row to be discarded. You want to treat two NULLs as equal. What

some do in such a case is use COALESCE or ISNULL to substitute a NULL with a value that

they know cannot appear in the data. For example, if the attributes are integers, and you

know that you have only positive integers in your data (you can even have constraints that

ensure this), you might try to use the predicate COALESCE(T1.col1, -1) = COALESCE(T2.

col1, -1), or ISNULL(T1.col1, -1) = ISNULL(T2.col1, -1). The problem with this form is that,

because you apply manipulation to the attributes you’re comparing, SQL Server will not

rely on index ordering. This can result in not using available indexes efficiently. Instead, it

is recommended to use the longer form: T1.col1 = T2.col1 OR (T1.col1 IS NULL AND T2.col1

IS NULL), which SQL Server understands as just a comparison that considers NULLs as

equal. With this form, SQL Server can efficiently use indexing.

T-SQL também suporta a função NULLIF padrão. Esta função aceita duas expressões de entrada,

retorna NULL se forem iguais, e retorna a primeira entrada, se eles não são. Por exemplo,

considerar o NULLIF expressão (col1, col2). Se col1 é igual a col2, a função retorna um

NULO; caso contrário, retorna o valor col1.

Quanto IIF e escolha, estas são funções de T-SQL fora do padrão que foram adicionadas para simplificar

migrações de plataformas Microsoft Access. Porque estas funções não são padrão e

existem alternativas padrão simples com expressões CASE, geralmente não é recomendado

que você usá-los. No entanto, quando você estiver migrando de Acesso ao SQL Server, essas funções

pode ajudar com a migração mais suave, e depois, gradualmente, você pode refatorar seu código para usar

as funções padrão disponíveis. Com a função IIF, você pode retornar um valor se uma entrada

predicado é verdadeira e outro valor contrário. A função tem o seguinte formulário.

IIF(<predicate>, <true\_result>, <false\_or\_unknown\_result>)

Esta expressão é equivalente à que se segue.

CASE WHEN <predicate> THEN <true\_result> ELSE <false\_or\_unknown\_result> END

Por exemplo, o IIF expressão (orderyear = 2012, Quant, 0) retorna o valor no atributo qty

quando o atributo orderyear é igual a 2012, e zero caso contrário.

A função ESCOLHA permite que você forneça uma posição e uma lista de expressões e retorna

a expressão na posição indicada. A função usa o seguinte formulário.

choose (<pos>, <exp1>, <exp2>, ..., <expn>)

Por exemplo, a expressão ESCOLHA (2, 'x', 'y', 'z') 'y' retorna. Mais uma vez, é simples

para substituir um ESCOLHA expressão com uma expressão CASE logicamente equivalente; mas o ponto

no apoio ESCOLHA, bem como IIF, é simplificar as migrações do Access para o SQL Server como um

solução temporária.

**Checagem rápida**

1. Você usaria o tipo FLOAT para representar um preço unitário do produto?

2. Qual é a diferença entre NEWID e NEWSEQUENTIALID?

3. Que função retorna a data atual e valor de tempo como um tipo DATETIME2?

4. Quando concatenando cadeias de caracteres, o que é a diferença entre o sinal de adição

(+) Operador ea função CONCAT?

**Quick Check Resposta**

1. Não, porque FLOAT é um tipo de dados aproximados e não podem representar todos os valores

precisamente.

2. A função NEWID gera valores GUID em ordem aleatória, enquanto que o

função ID NEWSEQUENTIAL gera GUIDs que aumentam em um sequencial

ordem.

3. A função SYSDATETIME.

4. O operador + por padrão produz um resultado NULL na entrada NULL, enquanto o

função CONCAT trata nulos como cadeias vazias.

**Quick Check**

1. Would you use the type FLOAT to represent a product unit price?

2. What is the difference between NEWID and NEWSEQUENTIALID?

3. Which function returns the current date and time value as a DATETIME2 type?

4. When concatenating character strings, what is the difference between the plus

(+) operator and the CONCAT function?

**Quick Check Answer**

1. No, because FLOAT is an approximate data type and cannot represent all values

precisely.

2. The NEWID function generates GUID values in random order, whereas the

NEWSEQUENTIAL ID function generates GUIDs that increase in a sequential

order.

3. The SYSDATETIME function.

4. The + operator by default yields a NULL result on NULL input, whereas the

CONCAT function treats NULLs as empty strings.

Prática **Trabalhando com tipos de dados e funções embutidas**

Nesta prática, você exercer o seu conhecimento de tipos de dados e funções. Você consultar dados de

tabelas existentes e manipular atributos existentes usando funções. Está equipado com exercícios

que contêm os pedidos para escrever consultas que abordam determinadas tarefas. Recomenda-se que

você primeiro tentar escrever a consulta a si mesmo e, em seguida, comparar sua resposta com a solução dada.

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 A plicar concatenação e usar uma data e hora Função

Neste exercício, você pratica concatenação eo uso de uma função de data e hora.

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

2. Faça uma consulta em relação a tabela de HR.EMPLOYEES que retorna a identificação do funcionário, a plena

nome do empregado (concatenar o primeiro nome atributos, espaço e sobrenome), e

o ano de nascimento (aplicar uma função para o atributo data de nascimento). Aqui está uma consulta possível

que realiza esta tarefa.

SELECT empid,

firstname + N' ' + lastname AS fullname,

YEAR(birthdate) AS birthyear

FROM HR.Employees;

Exercício 2 Funções Uso de Data e Hora adicionais

Neste exercício, você pratica o uso de data e hora funções adicionais.

Escreva uma expressão que calcula a data do último dia do mês atual. também escrever

uma expressão que calcula o último dia do ano em curso. É claro, há um número de

maneiras para alcançar tais tarefas. Aqui está uma maneira de calcular o final do mês atual.

SELECT EOMONTH(SYSDATETIME()) AS end\_of\_current\_month;

E aqui está uma forma de calcular o fim do ano em curso.

SELECT DATEFROMPARTS(YEAR(SYSDATETIME()), 12, 31) AS end\_of\_current\_year;

Usando a função ANO, você extrair o ano em curso. Em seguida, forneça o ano em curso, juntamente

com o mes 12 e no dia 31 para os DATEFROMPARTS funcionar para a construção do último dia

do corrente ano.

Exercício 3 Uso de Cordas e de conversão de Funções

Neste exercício, você praticar o uso das funções de string e conversão.

1. Crie uma consulta contra a tabela de Production.Products que retorna o numérica existente

ID do produto, além da identificação do produto formatado como uma cadeia de caracteres de tamanho fixo com 10

dígitos com zeros à esquerda. Por exemplo, para identificação do produto 42, você precisa retornar a string

'0000000042'. Uma forma de atender a essa necessidade é usando o código a seguir.

SELECT productid,

RIGHT(REPLICATE('0', 10) + CAST(productid AS VARCHAR(10)), 10) AS str\_productid

FROM Production.Products;

2. Usando a função REPLICATE, você gera uma corda feita de 10 zeros. Em seguida, você

concatenar o formulário do caráter da identificação do produto. Então você extrair o mais à direita 10

caracteres da seqüência de resultado.

Você pode pensar em uma maneira mais simples para alcançar a mesma tarefa utilizando as novas funções que eram

introduzido no SQL Server 2012? Uma maneira muito mais simples para conseguir a mesma coisa é através da utilização

a função de formatação, como na consulta seguinte.

productid SELECT,

FORMATO (productid, 'd10') AS str\_productid

DE Production.Products;

Resumo da lição

■■ Suas escolhas de tipos de dados para os seus atributos terá um efeito dramático sobre a funcionalidade

e desempenho do código T-SQL que interage com os dados, ainda mais

assim para atributos utilizados como chaves. Portanto, muito cuidado e consideração deve ser tomada

ao escolher tipos.

■■ T-SQL suporta um número de funções que você pode usar para aplicar a manipulação de data

e dados de tempo, dados de cadeia de caracteres, e outros tipos de dados. Lembre-se que T-SQL

foi projetado principalmente para lidar com a manipulação de dados, e não a formatação e similares

precisa. Portanto, nessas áreas, você encontrará tipicamente apenas um apoio bastante básico. Tal

tarefas geralmente são melhor tratadas no cliente.

■■ T-SQL fornece a expressão CASE que lhe permite devolver um valor com base em condicional

lógica, para além de um número de funções que podem ser consideradas as abreviaturas

de CASE.

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. Por que é importante usar o tipo apropriado para atributos?

A. Porque o tipo do seu atributo permite-lhe controlar a formatação do

valores

B. Como o tipo restringe os valores a um certo domínio de valores suportados

C. Como o tipo impede duplicatas

D. Como o tipo impede nulos

2. Qual das seguintes funções você consideraria usar para gerar chaves substitutas?

(Escolha tudo o que se aplicam).

A. NEWID

B. NEWSEQUENTIALID

C. GETDATE

D. CURRENT\_TIMESTAMP

3. Qual é a diferença entre a expressão CASE simples eo caso procurou

expressão?

A. A expressão CASE simples é usado quando o modelo de recuperação de banco de dados é simples,

ea expressão CASE pesquisada é utilizado quando estiver cheia ou a granel registrado.

B. A simples expressão CASE compara uma expressão de entrada para múltipla possível

expressões nas cláusulas WHEN, e a expressão CASE pesquisada utiliza independente

predicados nas cláusulas WHEN.

C. A expressão CASE simples pode ser usado em qualquer lugar em uma consulta, eo procurou

expressão CASE pode ser usado somente na cláusula WHERE.

D. A expressão CASE simples pode ser usado em qualquer lugar em uma consulta, eo procurou

expressão CASE pode ser usada somente em filtros de consulta (ON, onde, tendo).

**Cenários de caso**

Nos seguintes cenários, você aplicar o que você aprendeu sobre a instrução SELECT.

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

Case Scenario 1: Analisando o uso de tipos

Você está contratado como consultor para ajudar a problemas de desempenho endereço em um sistema existente. o

sistema foi desenvolvido originalmente usando o SQL Server 2005 e foi recentemente atualizado

para o SQL Server 2012. taxas de gravação no sistema são bastante baixo, e seu desempenho é mais

que suficiente. Além disso, o desempenho de gravação não é uma prioridade. No entanto, o desempenho de leitura é uma prioridade,

e, atualmente, não é satisfatório. Um dos principais objetivos do trabalho de consultoria

é fornecer recomendações que ajudarão a melhorar o desempenho de leitura. Você tem uma reunião

com representantes do cliente, e eles pedem suas recomendações em diferentes

áreas potenciais de melhoria. Uma das áreas que perguntar sobre é o uso de tipos de dados.

Sua tarefa é responder às seguintes questões de clientes:

1. Nós temos muitos atributos que representam uma data, como data do pedido, data da fatura, e assim por diante,

e atualmente usamos o tipo de dados DATETIME para aqueles. Você recomendaria degola

para o tipo ou substituí-lo com outro já existente? Quaisquer outras recomendações ao longo

linhas semelhantes?

2. Nós temos nossa própria solução tabela personalizada particionamento porque estamos no Padrão

edição do SQL Server. Nós usamos uma chave substituta de um tipo UNIQUEIDENTIFIER com o

função NEWID invocada por uma expressão de restrição padrão como a chave primária para o

tabelas. Nós escolhemos esta abordagem, porque não queremos que as chaves para entrar em conflito através dos diferentes

tabelas. Esta chave primária também é nosso cluster chave de índice. Você tem alguma recomendação

a respeito de nossa escolha de uma chave?

Cenário Caso 2: Revisando o uso de funções

A mesma empresa que você contratou para rever o uso de tipos de dados gostaria que você também

avaliar o uso de funções. Eles colocam a seguinte pergunta a você:

■■ Nossa aplicação tem trabalhado com SQL Server até agora, mas devido a uma recente fusão com

outra empresa, temos de apoiar outras plataformas de banco de dados também. O que você pode

recomendar, em termos de uso de funções?

**práticas sugeridas**

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

seguinte tarefas.

Analisar os tipos de dados no banco de dados de exemplo

Para praticar o seu conhecimento de tipos de dados, analisar os tipos de dados no banco de dados de exemplo

TSQL2012.

■■ **Prática 1**Usando o Object Explorer no SSMS, navegue para a base de dados exemplo

TSQL2012. Analisar as escolhas dos tipos de dados para as diferentes atributos e tentar

raciocinar sobre as escolhas. Além disso, avaliar se as escolhas efectuadas são ideal e

pensar sobre se há qualquer espaço para melhorias em alguns casos.

■■ **Prática 2**Visita Books Online em "Tipo de dados Precedência (Transact-SQL)", em *http: //*

*msdn.microsoft.com/en-us/library/ms190309.aspx*. Identificar a ordem de precedência

entre os tipos INT, DATETIME e VARCHAR. Tente raciocinar sobre Microsoft de

escolha desta ordem de precedência.

Analisar amostras de código em Books Online para o SQL Server 2012

Para entender melhor o uso de funções embutidas, analisar e executar os exemplos de código

fornecidas em Books Online para o SQL Server 2012.

■■ **Prática 1**Visite o artigo Books Online "dados de data e hora tipos e funções

(Transact-SQL) "no *http://msdn.microsoft.com/en-us/library/ms186724(v=SQL.110).aspx*.

De lá, siga os links que levam a artigos sobre funções individuais que parecem

útil. Nesses artigos, vá para a seção Exemplos. Analisar esses exemplos,

executá-los, e certifique-se que você entendê-los.

■■ **Prática 2**Semelhante à Prática 1, vá para o artigo Books Online "Funções para String

(Transact-SQL) "no *http://msdn.microsoft.com/en-us/library/ms181984(v=SQL.110).aspx*.

Siga os links para as funções que parecem útil. Nesses artigos, ir para os Exemplos

seção. Analisar e executar os exemplos, e certifique-se de compreendê-los.

**respostas**

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

cenários neste capítulo.

Lição 1

1. **Corrigir respostas: B e D**

A. **incorreta:**Atributo aliasing permite que você para atender às exigências relacionais, por isso é

certamente mais do que um recurso estético.

B. **correto:**O modelo relacional exige que todos os atributos têm nomes.

C. **incorreto:**T-SQL permite que um atributo resultado seja sem um nome quando a expressão

é baseado em um cálculo sem um alias.

D. **correto:**Você pode atribuir o seu próprio nome para um atributo de resultado usando um alias.

2. **Resposta correta: C**

A. **incorreto:**Os de e selecione cláusulas são obrigatórias em uma consulta SELECT acordo

para o SQL padrão, mas não T-SQL.

B. **incorreto:**A cláusula WHERE é opcional no T-SQL.

C. **correto:**De acordo com a T-SQL, a única cláusula obrigatória é a cláusula SELECT.

D. **incorreto:**Os e para onde cláusulas são opcionais em T-SQL.

3. **Corrigir respostas: C e D**

A. **incorreta:**Aliasing colunas com a cláusula AS é padrão e considerado uma melhor

prática.

B. **incorreto:**tabelas serrilhado com a cláusula AS é padrão e considerado uma melhor

prática.

C. **correto:**Não aliasing uma coluna que é um resultado de um cálculo é nonrelational

e é considerado uma má prática.

D. **correto:**Usando \* na lista SELECT é considerado uma má prática.

lição 2

1. **Resposta correta: B**

A. **incorreto:**A formatação não é uma responsabilidade do tipo ou a camada de dados em geral;

em vez disso, é da responsabilidade da camada de apresentação.

B. **correto:**O tipo deve ser considerada uma restrição porque limita os valores

permitido.

C. **incorreto:**O tipo em si não impede duplicatas. Se você precisa para evitar duplicatas,

você usar uma restrição de chave ou único primário.

D. **incorreto:**Um tipo não impede nulos. Para isso, use uma restrição NOT NULL.

2. **Corrigir respostas: A e B**

A. **correta:**A função NEWID cria GUIDs em ordem aleatória. Você consideraria

quando a sobrecarga de tamanho não é uma questão importante e a capacidade de gerar uma única

valor através do tempo e espaço, de qualquer lugar, em ordem aleatória é uma prioridade mais elevada.

B. **correto:**A função NEWSEQUENTIALID gera GUIDs em ordem crescente

dentro da máquina. Ela ajuda a reduzir a fragmentação e funciona bem quando um único

sessão carrega os dados, eo número de unidades é pequeno. No entanto, você deve

considerar cuidadosamente uma alternativa usando outro gerador de chaves, como um objeto de seqüência,

com um tipo menor, quando possível.

C. **incorreto:**Não há nenhuma garantia de que GETDATE irá gerar valores exclusivos; por conseguinte,

não é uma boa escolha para gerar chaves.

D. **incorreto:**A função CURRENT\_TIMESTAMP é simplesmente a versão padrão do

GETDATE, por isso também não garante exclusividade.

3. **Resposta correta: B**

A. **incorreta:**expressões CASE não tem nada a ver com a recuperação de banco de dados

modelo.

B. **correto:**A diferença entre os dois é que a forma simples compara

expressões e a forma procurou usa predicados.

C. **incorreto:**Ambas as expressões CASE são permitidos sempre que uma expressão escalar é

permitido em qualquer lugar na consulta.

D. **incorreto:**Ambas as expressões CASE são permitidos sempre que uma expressão escalar é

permitido em qualquer lugar na consulta.

Case Scenario 1

1. O tipo de dados DATETIME usa 8 bytes de armazenamento. SQL Server 2012 suporta a DATA

tipo de dados, que utiliza 3 bytes de armazenamento. Em todos esses atributos que representam uma data

somente, recomenda-se passar a usar DATE. Quanto mais baixo for o requisito de armazenamento,

melhor a lê pode executar.

Quanto a outras recomendações, a regra geral "menor é melhor, desde que você

cobrir as necessidades do atributo no longo prazo "é adequado para o desempenho de leitura. Para

exemplo, se você tem descrições de diferentes comprimentos armazenados em um tipo CHAR ou NCHAR,

considerar a mudança para VARCHAR ou NVARCHAR, respectivamente. Além disso, se você está atualmente

usando tipos de Unicode mas precisam armazenar cadeias de apenas um idioma, digamos, US inglês

considerar o uso de personagens regulares em vez disso.

2. Por um lado, o tipo UNIQUEIDENTIFIER é grande-16 bytes. E porque é também o

agrupado chave de índice, ele é copiado para todos os índices não clusterizados. Além disso, devido ao aleatório

ordem em que a função NEWID gera valores, há provavelmente um nível elevado

de fragmentação no índice. Uma abordagem diferente a considerar (e testar!) Está a mudar

para um tipo inteiro e usando o objeto de seqüência para gerar as chaves que não fazer

conflito entre tabelas. Devido ao tamanho reduzido do tipo, com o efeito multiplicado

em índices agrupados, o desempenho de leituras provavelmente vai melhorar. Os valores serão

crescente, e, como resultado, haverá uma menor fragmentação, o que também tem uma probabilidade

efeito positivo sobre lê.

Cenário Caso 2

■■ Para melhorar a portabilidade do código, é importante o uso de código padrão quando possível,

e isto naturalmente aplica-se mais especificamente à utilização de funções internas. Para

exemplo, o uso COALESCE e não ISNULL, use CURRENT\_TIMESTAMP e não getDate,

e usar CASE e não IIF.