Session - 10
Using Views, Stored Procedures, and Querying Metadata

Welcome to the Session, **Using Views, Stored Procedures, and Querying Metadata**.

This session explains about views and describes creating, altering, and dropping views. The session also describes stored procedures in detail. The session concludes with an explanation of the techniques to query metadata.

In this Session, you will learn to:

- Define views
- Describe the technique to create, alter, and drop views
- Define stored procedures
- Explain the types of stored procedures
- Describe the procedure to create, alter, and execute stored procedures
- Describe nested stored procedures
- Describe querying SQL Server metadata
  - System Catalog views and functions
  - Querying Dynamic Management Objects

10.1 Giới thiệu

Cơ sở dữ liệu SQL Server có hai thể loại đối tượng chính: những đối tượng lưu trữ dữ liệu và những đối tượng truy cập, thao tác, hoặc cung cấp quyền truy cập vào dữ liệu. Các khung nhìn và thủ tục lưu trữ thuộc về thể loại sau này.

10.2 Views

Khung nhìn là một bảng ảo được tạo thành từ các cột đã chọn từ một hoặc nhiều bảng. Những bảng từ đó khung nhìn được tạo ra được gọi là bảng cơ sở. Những bảng cơ sở này có thể là từ các cơ sở dữ liệu khác nhau. Khung nhìn cũng có thể bao gồm các cột từ các khung nhìn khác được tạo ra trong cùng một cơ sở dữ liệu hoặc một cơ sở dữ liệu khác. Khung nhìn có thể có tối đa là 1024 cột.

Các dữ liệu bên trong khung nhìn từ các bảng cơ sở được tham chiếu trong định nghĩa khung nhìn. Những hàng và cột của khung nhìn được tạo ra dạng động khi khung nhìn được tham chiếu.

10.2.1 Tạo các Views

Một người dùng có thể tạo ra một khung nhìn sử dụng các cột từ các bảng hoặc khung nhìn khác chỉ khi người dùng có quyền truy cập những bảng và khung nhìn này. Khung nhìn được tạo ra sử dụng câu lệnh CREATE VIEW và nó có thể được tạo ra chỉ trong cơ sở dữ liệu hiện tại.

SQL Server xác minh sự tồn tại của các đối tượng được tham chiếu trong định nghĩa khung nhìn

Ghi chú - Trong khi tạo ra một khung nhìn, thử nghiệm câu lệnh SELECT định nghĩa khung nhìn nhằm đảm bảo rằng SQL Server trả về kết quả mong đợi. Bạn tạo ra khung nhìn chỉ sau khi câu lệnh SELECT được thử nghiệm và tập kết quả đã được xác minh

Cú pháp sau đây được sử dụng để tạo ra một khung nhìn.

Cú pháp:

```
CREATE VIEW <view_name>
AS <select_statement>
```

trong đó,

index_name: chỉ ra tên của khung nhìn.

select_statement: chỉ ra câu lệnh SELECT định nghĩa khung nhìn.

Concepts

Code Snippet 1 tạo ra một khung nhìn từ bảng Product để chỉ hiển thị id sản phẩm, số sản phẩm, tên, và mức độ lưu kho an toàn của các sản phẩm.

Code Snippet 1:

```
CREATE VIEW vwProductInfo AS
SELECT ProductID, ProductNumber, Name, SafetyStockLevel
FROM Production.Product;
GO
```

Ghi chú - Những từ vw được đặt đằng trước tên khung nhìn theo các công ước mã hóa đề xuất.

Mã sau đây trong Code Snippet 2 được sử dụng để hiển thị các chi tiết của khung nhìn **vwProductInfo**.

Code Snippet 2:

```
SELECT * FROM vwProductInfo
```

Kết quả sẽ trình bày những cột cụ thể của tất cả các sản phẩm từ bảng Product. Một phần của kết quả được trình bày trong hình 10.1.

	ProductID	ProductNumber	Name	SafetyStockLevel
1	1	AR-5381	Adjustable Race	1000
2	2	BA-8327	Bearing Ball	1000
3	3	BE-2349	BB Ball Bearing	800
4	4	BE-2908	Headset Ball Bearings	800
5	316	BL-2036	Blade	800
6	317	CA-5965	LL Crankarm	500
7	318	CA-6738	ML Crankarm	500
8	319	CA-7457	HL Crankarm	500
9	320	CB-2903	Chainring Bolts	1000
10	321	CN-6137	Chainring Nut	1000

Hình 10.1: Selecting Records from a View

10.2.2 Tạo Views sử dụng từ khóa JOIN

Từ khóa JOIN cũng có thể được sử dụng để tạo ra các khung nhìn. Câu lệnh CREATE VIEW được sử dụng cùng với từ khóa JOIN để tạo ra một khung nhìn sử dụng các cột từ nhiều bảng.

Cú pháp sau đây được sử dụng để tạo ra một khung nhìn với từ khóa JOIN.

Cú pháp:

```
CREATE VIEW <view_name>
AS
SELECT * FROM table_name1
JOIN table_name2
ON table_name1.column_name = table_name2.column_name
```

trong đó,

index_name: chỉ ra tên của khung nhìn.

table_name1: chỉ ra tên của bảng đầu tiên.

JOIN: chỉ ra rằng hai bảng được nối sử dụng từ khóa JOIN.

table_name2: chỉ ra tên của bảng thứ hai.

Code Snippet 3 3 tạo ra một khung nhìn có tên là **vwPersonDetails** với các cột cụ thể từ bảng **Person** và **Employee**. Từ khóa JOIN và ON nối hai bảng dựa trên cột **BusinessEntityID**.

Code Snippet 3:

```
CREATE VIEW vwPersonDetails
AS
SELECT
   p.Title
  ,p.[FirstName]
  ,p.[MiddleName]
  ,p.[LastName]
  ,e.[JobTitle]
FROM [HumanResources].[Employee] e
     INNER JOIN [Person].[Person] p
     ON p.[BusinessEntityID] = e.[BusinessEntityID]
GO
```

Khung nhìn này sẽ chứa các cột **Title, FirstName, MiddleName**, và **LastName** từ bảng **Person** và **JobTitle** từ bảng **Employee**. Sau khi khung nhìn được tạo ra, bạn có thể lấy các bản ghi từ đó, và còn thao tác và chỉnh sửa các bản ghi.

Concepts

Code Snippet 4 trình bày thực hiện một truy vấn trên khung nhìn này.

Code Snippet 4:

```
SELECT * FROM vwPersonDetails
```

The output will be as shown in Hình 10.2.

	Title	FirstName	MiddleName	LastName	JobTitle
1	NULL	Ken	J	Sánchez	Chief Executive Officer
2	NULL	Terri	Lee	Duffy	Vice President of Engineering
3	NULL	Roberto	NULL	Tamburello	Engineering Manager
4	NULL	Rob	NULL	Walters	Senior Tool Designer
5	Ms.	Gail	A	Erickson	Design Engineer
6	Mr.	Jossef	H	Goldberg	Design Engineer
7	NULL	Dylan	A	Miller	Research and Development Manager
8	NULL	Diane	L	Margheim	Research and Development Engineer
9	NULL	Gigi	N	Matthew	Research and Development Engineer
10	NULL	Michael	NULL	Raheem	Research and Development Manager

Hình 10.2: Creating Views with a Join

Như được trình bày trong hình 10.2, tất cả các hàng có thể không có các giá trị cho các cột Title hoặc MiddleName - một số có thể có NULL trong đó. Một người nhìn thấy kết quả này có thể không có khả năng hiểu được ý nghĩa của các giá trị NULL. Do đó, để thay thế tất cả các giá trị NULL ở đầu ra với một chuỗi null, hàm COALESCE() có thể được sử dụng như được trình bày trong Code Snippet 5.

Code Snippet 5:

```
CREATE VIEW vwPersonDetails
AS
SELECT
  COALESCE(p.Title, '') AS Title
 ,p.[FirstName]
 ,COALESCE(p.MiddleName, '') AS MiddleName
 ,p.[LastName]
 ,e.[JobTitle]
FROM [HumanResources].[Employee] e
     INNER JOIN [Person].[Person] p
     ON p.[BusinessEntityID] = e.[BusinessEntityID]
GO
```

Khi khung nhìn này được truy vấn với câu lệnh SELECT, kết quả sẽ được trình bày trong hình 10.3.

	Title	FirstName	MiddleName	LastName	JobTitle
1		Ken	J	Sánchez	Chief Executive Officer
2		Terri	Lee	Duffy	Vice President of Engineering
3		Roberto		Tamburello	Engineering Manager
4		Rob		Walters	Senior Tool Designer
5	Ms.	Gail	A	Erickson	Design Engineer
6	Mr.	Jossef	H	Goldberg	Design Engineer
7		Dylan	A	Miller	Research and Development Manager
8		Diane	L	Margheim	Research and Development Engineer
9		Gigi	N	Matthew	Research and Development Engineer
10		Michael		Raheem	Research and Development Manager

Hình 10.3: Using COALESCE Function in a View

10.2.3 Hướng dẫn và hạn chế về khung nhìn

Khung nhìn có thể được tạo ra sử dụng lệnh CREATE VIEW. Trước khi tạo ra một khung nhìn, hướng dẫn và hạn chế sau đây cần được xem xét:

- Khung nhìn được tạo ra chỉ trong cơ sở dữ liệu hiện tại. Các bảng cơ sở và khung nhìn từ đó khung nhìn được tạo ra có thể là từ các cơ sở dữ liệu hoặc máy chủ khác.
- Các tên khung nhìn phải là duy nhất và không thể được giống như các tên bảng trong lược đồ.
- Khung nhìn không thể được tạo ra trên các bảng tạm thời.
- Khung nhìn không thể có một chỉ mục toàn văn.
- Khung nhìn không thể chứa định nghĩa DEFAULT.
- Câu lệnh CREATE VIEW có thể bao gồm mệnh đề ORDER BY chỉ khi từ khóa TOP được sử dụng.
- Khung nhìn không thể tham chiếu hơn 1024 cột.
- Câu lệnh CREATE VIEW không thể bao gồm từ khóa INTO.
- Câu lệnh CREATE VIEW không thể được kết hợp với các câu lệnh Transact-SQL khác trong một khối lệnh đơn lẻ.

Nếu khung nhìn chứa các cột có giá trị bắt nguồn từ một biểu thức, các cột như vậy phải được đặt tên bí danh. Ngoài ra, nếu khung nhìn chứa các cột có tên tương tự từ các bảng khác nhau, để phân biệt những cột này, tên bí danh phải được chỉ ra.

Concepts

Code Snippet 6 tái sử dụng mã được cho trong Đoạn mã 5 với mệnh đề ORDER BY. Từ khóa TOP hiển thị tên của mười nhân viên đầu tiên với các tên họ của họ theo thứ tự tăng dần.

Code Snippet 6:

```
CREATE VIEW vwSortedPersonDetails
AS
SELECT TOP 10
  COALESCE(p.Title, '') AS Title
 ,p.[FirstName]
 ,COALESCE(p.MiddleName, '') AS MiddleName
 ,p.[LastName]
 ,e.[JobTitle]

FROM [HumanResources].[Employee] e
     INNER JOIN [Person].[Person] p
     ON p.[BusinessEntityID] = e.[BusinessEntityID]
     ORDER BY p.FirstName
GO

--Retrieve records from the view
SELECT * FROM vwSortedPersonDetails
```

10.3 Sửa đổi dữ liệu thông qua các khung nhìn

Khung nhìn có thể được sử dụng để sửa đổi dữ liệu trong các bảng cơ sở dữ liệu. Dữ liệu có thể được chèn, sửa đổi hoặc xóa thông qua một khung nhìn sử dụng các câu lệnh sau :

- **INSERT**
- **UPDATE**
- **DELETE**

10.3.1 INSERT với các khung nhìn

Câu lệnh INSERT được sử dụng để thêm một hàng mới vào một bảng hoặc khung nhìn. Trong khi thực hiện câu lệnh, nếu giá trị cho một cột không được cung cấp, SQL Server Database Engine phải cung cấp một giá trị dựa trên định nghĩa của cột. Nếu Database Engine không thể cung cấp giá trị này, khi đó hàng mới sẽ không được thêm vào.

Concepts

Giá trị cho cột này được cung cấp tự động nếu:

- Cột có thuộc tính IDENTITY.
- Cột có giá trị mặc định đã chỉ ra.
- Cột có kiểu dữ liệu timestamp.
- Cột có các giá trị null.
- Cột là một cột tính.

Trong khi sử dụng câu lệnh INSERT trên khung nhìn, nếu có bất kỳ quy tắc nào bị vi phạm, bản ghi sẽ không được chèn vào.

Trong ví dụ sau đây, khi dữ liệu được đưa vào thông qua khung nhìn, việc chèn không diễn ra bởi khung nhìn được tạo ra từ hai bảng cơ sở.

Đầu tiên, tạo một bảng **Employee_Personal_Details** như được trình bày trong Code Snippet 7.
Code Snippet 7:

```
CREATE TABLE Employee_Personal_Details
(
EmpID  int NOT NULL,
FirstName varchar(30) NOT NULL,
LastName varchar(30) NOT NULL,
Address varchar(30)
)
```

Sau đó, tạo một bảng **Employee_Personal_Details** như được trình bày trong Code Snippet 8.

Code Snippet 8:

```
CREATE TABLE Employee_Salary_Details
(
EmpID
int NOT NULL,
Designation varchar(30),
Salary int NOT NULL
)
```

Code Snippet 9 tạo ra một khung nhìn **vwEmployee_Details** sử dụng các cột từ bảng **Employee_Personal_Details** và **Employee_Salary_Details** bằng cách nối hai bảng trên cột **EmpID**.

Code Snippet 9:

```
CREATE VIEW vwEmployee_Details
AS
SELECT e1.EmpID, FirstName, LastName, Designation, Salary
FROM Employee_Personal_Details e1
JOIN Employee_Salary_Details e2
ON e1.EmpID=e2.EmpID
```

Code Snippet 10 sử dụng câu lệnh INSERT để chèn dữ liệu thông qua khung nhìn **vwEmployee_Details**. Tuy nhiên, dữ liệu không được chèn vào bởi khung nhìn được tạo ra từ hai bảng cơ sở.

Code Snippet 10:

```
INSERT INTO vwEmployee_Details VALUES (2,'Jack','Wilson','Software
Developer',16000)
```

Thông báo lỗi sau được hiển thị khi câu lệnh INSERT được thực thi.

'Msg 4405, Level 16, State 1, Line 1
Khung nhìn hoặc hàm 'vEmployee_Details' là không thể cập nhật vì sửa đổi ảnh hưởng đến nhiều bảng cơ sở.'

Giá trị có thể được chèn vào các cột thuộc kiểu dữ liệu do người dùng định nghĩa bằng cách:

- Chỉ ra giá trị của loại do người dùng định nghĩa.
- Việc gọi một hàm do người dùng định nghĩa trả về giá trị thuộc loại do người dùng định nghĩa.

Các quy tắc và hướng dẫn sau đây phải được tuân thủ khi sử dụng câu lệnh INSERT:

- Câu lệnh INSERT phải chỉ ra các giá trị cho tất cả các cột trong một khung nhìn trong bảng bên dưới không cho phép các giá trị null và không có định nghĩa DEFAULT.
- Khi có một phép tự nối với cùng một khung nhìn hoặc bảng cơ sở, câu lệnh INSERT không làm việc.

Concepts

Code Snippet 11 tạo ra một khung nhìn **vwEmpDetails** sử dụng bảng **Employee_Personal_Details**. Bảng **Employee_Personal_Details** chứa một cột có tên là **LastName** không cho phép các giá trị null được chèn vào.

Code Snippet 11:

```
CREATE VIEW vwEmpDetails
AS
SELECT FirstName, Address
FROM Employee_Personal_Details
GO
```

Code Snippet 12 cố gắng chèn các giá trị vào khung nhìn **vwEmpDetails**.

Code Snippet 12:

```
INSERT INTO vwEmpDetails VALUES ('Jack','NYC')
```

Việc chèn này không được phép bởi khung nhìn không chứa cột **LastName** từ bảng cơ sở và cột đó không cho phép các giá trị null.

10.3.2 UPDATE với Views

Có thể sử dụng câu lệnh UPDATE để thay đổi dữ liệu trong một khung nhìn. Việc cập nhật một khung nhìn còn cập nhật bảng bên dưới.

Xem xét một ví dụ Code Snippet 13 tạo ra một bảng có tên là **Product_Details**.

Code Snippet 13:

```
CREATE TABLE Product_Details
(
ProductID int,
ProductName varchar(30),
Rate money
)
```

Concepts

Giả sử một số bản ghi được thêm vào trong bảng như được trình bày trong hình 10.4.

	ProductID	ProductName	Rate
1	5	DVD Writer	2250.00
2	4	DVD Writer	1250.00
3	6	DVD Writer	1250.00
4	2	External Hard Drive	4250.00
5	3	External Hard Drive	4250.00

Hình 10.4: Records in the Product_Details

Code Snippet 14 tạo ra một khung nhìn dựa trên bảng **Product_Details**.

Code Snippet 14:

```
CREATE VIEW vwProduct_Details
AS
SELECT
ProductName, Rate FROM Product_Details
```

Code Snippet 15 cập nhật khung nhìn nhằm thay đổi tất cả giá của các máy ghi DVD thành 3000.

Code Snippet 15:

```
UPDATE vwProduct_Details
SET Rate=3000
WHERE ProductName='DVD Writer'
```

Kết quả của mã này ảnh hưởng đến không chỉ khung nhìn **vwProduct_Details**, mà còn cả bảng bên dưới nơi khung nhìn đã được tạo ra.

Hình 10.5 trình bày bảng đã cập nhật được tự động cập nhật vì khung nhìn này.

	ProductID	ProductName	Rate
1	5	DVD Writer	3000.00
2	4	DVD Writer	3000.00
3	6	DVD Writer	3000.00
4	2	External Hard Drive	4250.00
5	3	External Hard Drive	4250.00

Hình 10.5: Updated Table

Các kiểu dữ liệu giá trị lớn bao gồm varchar(max), nvarchar(max), và varbinary(max). Để cập nhật dữ liệu có các kiểu dữ liệu giá trị lớn, mệnh đề .WRITE được sử dụng. Mệnh đề .WRITE chỉ ra rằng một phần của giá trị trong cột sẽ được sửa đổi. Mệnh đề .WRITE không thể được sử dụng để cập nhật giá trị NULL trong một cột. Ngoài ra, không thể sử dụng mệnh đề này để đặt giá trị cột thành NULL.

Cú pháp:

```
column_name .WRITE (expression, @Offset, @Length)
```

trong đó,

column_name: chỉ ra tên của cột thuộc kiểu dữ liệu có giá trị lớn.

Expression: chỉ ra tên được sao chép vào cột.

@Offset: chỉ ra điểm khởi đầu trong giá trị của cột, tại đó biểu thức được ghi.

@Length: chỉ ra độ dài của phần trong cột.

@Offset và **@Length** được chỉ ra theo byte cho kiểu dữ liệu varbinary và varchar và theo các ký tự cho kiểu dữ liệu nvarchar.

Giả sử bảng **Product_Details** được sửa đổi để đưa vào cột **Description** có kiểu dữ liệu nvarchar(max).

Khung nhìn được tạo ra dựa trên bảng này, có các cột **ProductName**, **Description**, và **Rate** như được trình bày trong Code Snippet 16.

Code Snippet 16:

```
CREATE VIEW vwProduct_Details
AS
SELECT
ProductName,
Description,
Rate FROM Product_Details
```

Code Snippet 17 sử dụng câu lệnh UPDATE trên khung nhìn **vwProduct_Details**. Mệnh đề **.WRITE** được sử dụng để thay đổi giá trị của Internal trong cột Description thành External.

Code Snippet 17:

```
UPDATE vwProduct_Details
SET Description .WRITE (N'Ex',0,2)
WHERE ProductName='Portable Hard Drive'
```

Là kết quả của đoạn mã này, tất cả các hàng trong khung nhìn có tên sản phẩm là 'Ổ cứng di động' sẽ được cập nhật với External thay vì Internal trong cột **Description**.

Concepts

Hình 10.6 trình bày một mẫu đầu ra của khung nhìn sau khi cập nhật.

	ProductName	Description	Rate
1	Hard Disk Drive	Internal 120 GB	3570.00
2	Portable Hard Drive	External Drive 500 GB	5580.00
3	Portable Hard Drive	External Drive 500 GB	5580.00
4	Hard Disk Drive	Internal 120 GB	3570.00
5	Portable Hard Drive	External Drive 500 GB	5580.00

Hình 10.6: Updating a View

Những quy tắc và hướng dẫn sau đây phải được tuân thủ khi sử dụng câu lệnh **UPDATE**:

- Giá trị của cột có thuộc tính IDENTITY không thể được cập nhật.
- Không thể cập nhật các bản ghi nếu bảng cơ sở có chứa cột TIMESTAMP.
- Trong khi cập nhật một hàng, nếu ràng buộc hoặc quy tắc bị vi phạm, câu lệnh bị chấm dứt, lỗi được trả về, và không có bản ghi nào được cập nhật.
- Khi có một phép tự nối với cùng một khung nhìn hoặc bảng cơ sở, câu lệnh UPDATE không làm việc.

10.3.3 DELETE với Views

SQL Server cho phép bạn xóa các hàng từ khung nhìn. Các hàng có thể được xóa từ khung nhìn sử dụng câu lệnh **DELETE**. Khi các hàng được xóa khỏi khung nhìn, các hàng tương ứng sẽ bị xóa khỏi bảng cơ sở.

Ví dụ, hãy xem xét khung nhìn **vwCustDetails** liệt kê thông tin tài khoản của các khách hàng khác nhau. Khi khách hàng đóng tài khoản, những chi tiết của khách hàng này cần phải được xóa. Điều này được thực hiện bằng cách sử dụng câu lệnh **DELETE**.

Cú pháp sau đây được sử dụng để xóa dữ liệu khỏi khung nhìn.

Cú pháp :

```
DELETE FROM <view_name>
WHERE <search_condition>
```

Giả sử rằng bảng có tên là **Customer_Details** và khung nhìn **vwCustDetails** dựa trên bảng này được tạo ra.

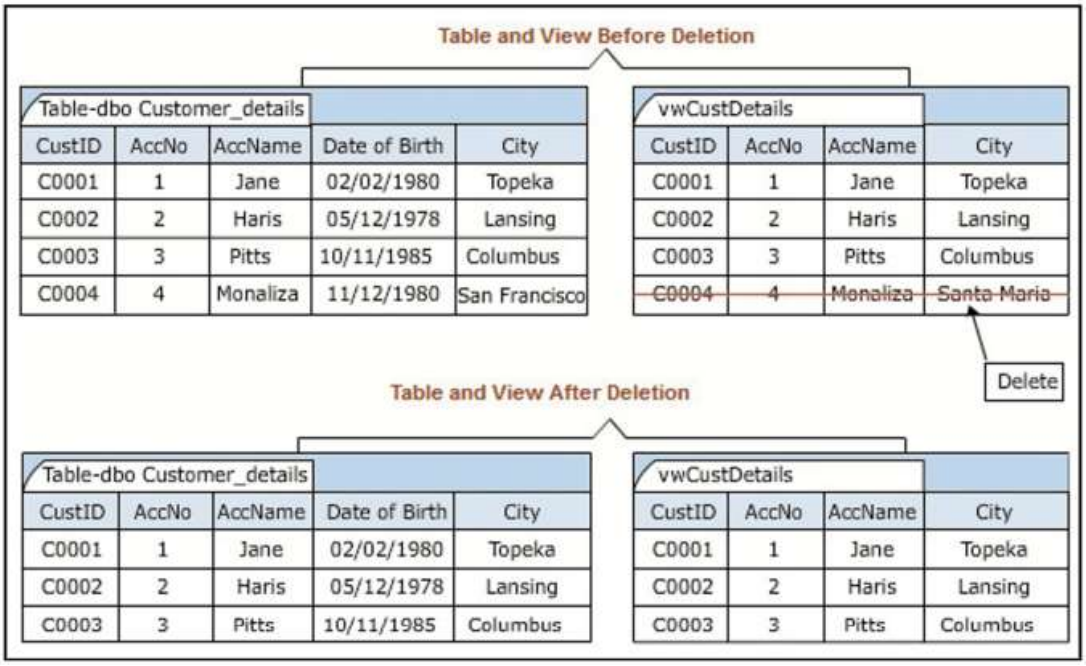
Code Snippet 18 được sử dụng để xóa bản ghi từ khung nhìn **vwCustDetails** có **CustID** C0004.

Code Snippet 18:

```
DELETE FROM vwCustDetails WHERE CustID='C0004'
```

Concepts

Hình 10.7 mô tả logic việc xóa khỏi các khung nhìn.



Hình 10.7: Deleting from Views

10.4 Thay đổi các khung nhìn

Bên cạnh việc sửa đổi dữ liệu trong khung nhìn, người dùng cũng có thể thay đổi định nghĩa của khung nhìn. Có thể sửa đổi hoặc thay đổi khung nhìn bằng cách thả bỏ và tái tạo nó hoặc thực thi câu lệnh ALTER VIEW. Câu lệnh ALTER VIEW sửa đổi khung nhìn hiện có mà không cần phải tổ chức lại các cấp quyền của nó và các thuộc tính khác. ALTER VIEW có thể được áp dụng cho các khung nhìn có chỉ mục, tuy nhiên, nó thả bỏ vô điều kiện tất cả các chỉ mục trên khung nhìn đó.

Các khung nhìn thường được thay đổi khi người dùng yêu cầu thêm thông tin hoặc thay đổi trong định nghĩa bảng bên dưới.

> **Ghi chú** - Sau khi khung nhìn được tạo ra, nếu cấu trúc của các bảng bên dưới của nó được thay đổi bằng cách thêm các cột, những cột mới này không xuất hiện trong khung nhìn. Điều này là do danh sách cột được giải thích chỉ khi bạn lần đầu tiên tạo ra khung nhìn. Để xem các cột mới trong khung nhìn, bạn phải thay đổi khung nhìn.

Cú pháp sau đây được sử dụng để thay đổi khung nhìn.

Cú pháp:

```
ALTER VIEW <view_name>
AS <select_statement>
```

Concepts

Code Snippet 19 thay đổi khung nhìn **vwProductInfo** để đưa vào cột **ReOrderPoint.**

Code Snippet 19:

```
ALTER VIEW vwProductInfo AS
SELECT ProductID, ProductNumber, Name, SafetyStockLevel, ReOrderPoint
FROM Production.Product;
GO
```

10.5 Dropping Views

Có thể loại bỏ khung nhìn khỏi cơ sở dữ liệu nếu nó không còn cần thiết. Điều này được thực hiện bằng cách sử dụng câu lệnh DROP VIEW. Khi khung nhìn được thả bỏ, dữ liệu trong các bảng cơ sở vẫn không bị ảnh hưởng. Định nghĩa của khung nhìn và thông tin khác liên quan đến khung nhìn sẽ bị xóa khỏi danh mục hệ thống. Tất cả các sự cho phép cho khung nhìn đó cũng sẽ bị xóa. Nếu người dùng truy vấn bất kỳ khung nhìn nào tham chiếu tới khung nhìn đã thả bỏ, người dùng sẽ nhận được thông báo lỗi

Ghi chú - Chủ sở hữu khung nhìn có sự cho phép để thả bỏ khung nhìn và sự cho phép này là không thể chuyển nhượng. Tuy nhiên, quản trị viên hệ thống hoặc chủ sở hữu cơ sở dữ liệu có thể thả bỏ bất kỳ đối tượng nào bằng cách xác định tên chủ sở hữu trong câu lệnh DROP VIEW.

Cú pháp sau đây được sử dụng để thả bỏ khung nhìn.

Cú pháp:

```
DROP VIEW <view_name>
```

Code Snippet 20 xóa khung nhìn **vwProductInfo.**

Code Snippet 20:

```
DROP VIEW vwProductInfo
```

10.6 Định nghĩa của khung nhìn

Định nghĩa của khung nhìn giúp hiểu dữ liệu của nó được lấy từ các bảng nguồn như thế nào. Có một số thủ tục lưu trữ hệ thống có thể giúp lấy các định nghĩa khung nhìn.

Thủ tục lưu trữ sp_helptext hiển thị thông tin liên quan đến khung nhìn khi tên của khung nhìn được cho là tham số của nó. Có thể thu được thông tin về định nghĩa của khung nhìn nếu thông tin đó không bị mã hóa.

Cú pháp sau đây được sử dụng để xem thông tin định nghĩa của khung nhìn.

Cú pháp :

```
sp_helptext <view_name>
```

Concepts

Code Snippet 21 hiển thị thông tin về khung nhìn **vwProductPrice**.

Code Snippet 21:

```
EXEC sp_helptext vwProductPrice
```

Việc thực thi đoạn mã này sẽ hiển thị định nghĩa về khung nhìn như được trình bày trong hình 10.8.

	Text
1	CREATE VIEW vwProductPrice AS
2	SELECT ProductID, ProductNumber, Name, SafetySto...
3	FROM Production.Product;

Hình 10.8: Using sp_helptext to Display View Definitions

10.7 Tạo một khung nhìn sử dụng các hàm dựng sẵn

Có thể tạo ra các khung nhìn bằng cách sử dụng các hàm dựng sẵn của SQL Server. Khi các chức năng được sử dụng, cột xuất phát phải bao gồm tên cột trong câu lệnh **CREATE VIEW**..

Hãy xem xét khung nhìn đã được tạo ra trong Code Snippet 16. Nó đã được tái tạo trong Code Snippet 22 để sử dụng hàm AVG().

Code Snippet 22:

```
CREATE VIEW vwProduct_Details
AS
SELECT
ProductName,
AVG(Rate) AS AverageRate
FROM Product_Details
GROUP BY ProductName
```

Ở đây, hàm AVG() tính toán tỷ lệ trung bình của các sản phẩm tương tự bằng cách sử dụng mệnh đề GROUP BY. Hình 10.9 trình bày kết quả khi khung nhìn được truy vấn.

	ProductName	AverageRate
1	Hard Disk Drive	3570.00
2	Portable Hard Drive	5580.00

Hình 10.9: Using Built-in Functions with Views

Concepts

10.8 CHECK OPTION

CHECK OPTION là một tùy chọn liên quan đến câu lệnh **CREATE VIEW**. Nó được sử dụng để đảm bảo rằng tất cả các bản cập nhật trong khung nhìn đáp ứng những điều kiện đã nêu trong định nghĩa khung nhìn. Nếu những điều kiện không được thỏa mãn, công cụ cơ sở dữ liệu trả về lỗi. Như vậy, **CHECK OPTION** được sử dụng để thực thi tính toàn vẹn miền, nó sẽ kiểm tra định nghĩa của khung nhìn để thấy rằng các điều kiện **WHERE** trong câu lệnh **SELECT** không bị vi phạm.

Mệnh đề **WITH CHECK OPTION** bắt buộc tất cả các câu lệnh sửa đổi đã thực thi đối với khung nhìn này để thực hiện theo điều kiện được đặt trong câu lệnh **SELECT**. Khi hàng được sửa đổi, **WITH CHECK OPTION** đảm bảo rằng dữ liệu vẫn có thể nhìn thấy thông qua khung nhìn.

Cú pháp sau đây tạo ra một khung nhìn sử dụng **CHECK OPTION**.

Cú pháp:

```
CREATE VIEW <view_name>
AS select_statement [WITH CHECK OPTION]
```

trong đó,

WITH CHECK OPTION: chỉ ra rằng dữ liệu đã thay đổi trong khung nhìn tiếp tục đáp ứng định nghĩa khung nhìn.

Code Snippet 23 tạo ra lại khung nhìn **vwProductInfo** có SafetyStockLevel nhỏ hơn hoặc bằng 1000

Code Snippet 23:

```
CREATE VIEW vwProductInfo AS
SELECT ProductID, ProductNumber,Name,SafetyStockLevel,
ReOrderPoint
FROM Production.Product
WHERE SafetyStockLevel <=1000
WITH CHECK OPTION;
GO
```

Trong Code Snippet 23, câu lệnh UPDATE được sử dụng để sửa đổi khung nhìn **vwProductInfo** bằng cách thay đổi giá trị của cột SafetyStockLevel cho sản phẩm có id 321 đến 2500.

Code Snippet 24:

```
UPDATE vwProductInfo SET SafetyStockLevel= 2500
WHERE ProductID=321
```

Concepts

Câu lệnh UPDATE không thực thi được vì nó vi phạm định nghĩa khung nhìn, chỉ ra rằng SafetyStockLevel phải nhỏ hơn hoặc bằng 1000. Do đó, không có hàng bị ảnh hưởng trong khung nhìn **vwProductInfo**.

Ghi chú - Bất kỳ bản cập nhật nào được thực hiện trên các bảng cơ sở không được xác nhận đối với khung nhìn này, ngay cả khi CHECK OPTION được chỉ ra.

10.9 Tùy chọn SCHEMABINDING

Có thể liên kết khung nhìn với cấu trúc của bảng cơ sở sử dụng tùy chọn SCHEMABINDING. Có thể sử dụng tùy chọn này với câu lệnh CREATE VIEW hoặc ALTER VIEW. Khi tùy chọn SCHEMABINDING được chỉ ra, khung nhìn sửa đổi bảng cơ sở sẽ ảnh hưởng đến các định nghĩa khung nhìn. Định nghĩa khung nhìn trước tiên phải được sửa đổi hoặc xóa để loại bỏ sự phụ thuộc vào bảng sẽ được sửa đổi.

Trong khi sử dụng tùy chọn SCHEMABINDING trong khung nhìn, bạn phải chỉ ra tên lược đồ cùng với tên đối tượng trong câu lệnh SELECT.

Cú pháp sau đây được sử dụng để tạo ra một khung nhìn với tùy chọn SCHEMABINDING.

Cú pháp:

```
CREATE VIEW <view_name> WITH SCHEMABINDING
AS <select_statement>
```

trong đó,

index_name: chỉ ra tên của khung nhìn.

WITH SCHEMABINDING: chỉ ra rằng khung nhìn phải được ràng buộc với một lược đồ.

select_statement: Chỉ ra câu lệnh SELECT định nghĩa khung nhìn đó.

Code Snippet 25 tạo ra khung nhìn **vwNewProductInfo** với tùy chọn SCHEMABINDING để ràng buộc khung nhìn với lược đồ Production, là lược đồ của bảng Product.

Code Snippet 25:

```
CREATE VIEW vwNewProductInfo
WITH SCHEMABINDING AS
SELECT ProductID, ProductNumber, Name, SafetyStockLevel
FROM Production.Product;
GO
```

Concepts

10.10 Sử dụng sp_refreshview

Trong khi tạo một khung nhìn, tùy chọn SCHEMABINDING được sử dụng để ràng buộc khung nhìn với lược đồ của những bảng được đưa vào trong khung nhìn. Tuy nhiên, cũng có thể tạo ra khung nhìn mà không chọn tùy chọn SCHEMABINDING. Trong trường hợp này, nếu thay đổi được thực hiện cho các đối tượng nằm dưới (bảng hoặc khung nhìn) để khung nhìn phụ thuộc vào đó, thủ tục lưu trữ sp_refreshview phải được thực thi. Thủ tục lưu trữ sp_refreshview cập nhật siêu dữ liệu cho khung nhìn. Nếu thủ tục sp_refreshview không được thực hiện, siêu dữ liệu của khung nhìn không được cập nhật để phản ánh những thay đổi trong những bảng cơ sở này. Điều này dẫn đến việc tạo ra các kết quả bất ngờ khi khung nhìn được truy vấn.

Thủ tục lưu trữ sp_refreshview trả về giá trị mã số 0 nếu việc thực thi thành công hoặc trả về một số khác số 0 trong trường hợp việc thực thi đã thất bại.

Cú pháp sau đây được sử dụng để chạy thủ tục lưu trữ sp_refreshview.

Cú pháp:

```
sp_refreshview '<view_name>'
```

Code Snippet 26 tạo ra bảng **Customers** với các cột **CustID**, **CustName**, và **Address**.

Code Snippet 26:

```
CREATE TABLE Customers
(
CustID int,
CustName varchar(50),
Address varchar(60)
)
```

Code Snippet 27 tạo ra khung nhìn **vwCustomers** dựa trên bảng **Customers**.

Code Snippet 27:

```
CREATE VIEW vwCustomers
AS
SELECT * FROM Customers
```

Code Snippet 28 thực thi truy vấn **SELECT** trên khung nhìn.

Code Snippet 28:

```
SELECT * FROM vwCustomers
```

Đầu ra của Code Snippet 28 trình bày ba cột **CustID**, **CustName**, và **Address**.

Concepts

Code Snippet 29 sử dụng câu lệnh ALTER TABLE để thêm một cột **Age** vào bảng **Customers**.

Code Snippet 29:

```
ALTER TABLE Customers ADD Age int
```

Code Snippet 30 thực thi truy vấn SELECT trên khung nhìn.

Code Snippet 30:

```
SELECT * FROM vwCustomers
```

Cột đã cập nhật **Age** không được nhìn thấy trong khung nhìn.

Để giải quyết điều này, thủ tục lưu trữ sp_refreshview phải được thực thi trên khung nhìn **vwCustomers** như được trình bày trong Code Snippet 31.

Code Snippet 31:

```
EXEC sp_refreshview 'vwCustomers'
```

Khi truy vấn SELECT được chạy một lần nữa treen khung nhìn, cột **Age** được nhìn thấy trong kết quả. Điều này là do thủ tục sp_refreshview làm mới siêu dữ liệu cho khung nhìn **vwCustomers**.

Không thể thả bỏ các bảng được gắn với lược đồ vào khung nhìn trừ khi khung nhìn được thả bỏ hoặc thay đổi làm sao nó không có ràng buộc lược đồ nào. Nếu khung nhìn không được thả bỏ hoặc thay đổi và bạn cố gắng thả bỏ bảng, Công cụ cơ sở dữ liệu trả về một thông báo lỗi.

Ngoài ra, khi câu lệnh ALTER TABLE ảnh hưởng đến định nghĩa khung nhìn của một khung nhìn có gắn lược đồ, câu lệnh ALTER TABLE thất bại.

Hãy xem xét khung nhìn có gắn lược đồ đã được tạo ra trong Đoạn mã 25. Nó phụ thuộc vào bảng Production.Product. Code Snippet cố gắng sửa đổi kiểu dữ liệu của cột ProductID trong bảng Production.Product từ int thành varchar(7).

Code Snippet 32:

```
ALTER TABLE Production.Product ALTER COLUMN ProductID varchar(7)
```

Công cụ cơ sở dữ liệu trả về thông báo lỗi như bảng được ràng buộc lược đồ với khung nhìn **vwNewProductInfo** và do đó, không thể thay đổi được bởi nó vi phạm định nghĩa khung nhìn của khung nhìn.

10.11 Các thủ tục lưu trữ

Thủ tục lưu trữ là một nhóm các câu lệnh Transact-SQL hành động như một khối mã duy nhất thực hiện một nhiệm vụ cụ thể. Khối mã này được xác định bằng một tên được gán và được lưu trữ trong cơ sở dữ liệu ở dạng đã biên soạn. Thủ tục lưu trữ cũng có thể là tham chiếu đến một phương pháp .NET Framework Common Language Runtime (CLR).

Thủ tục lưu trữ rất hữu ích các nhiệm vụ lặp lại phải được thực hiện. Điều này giúp loại bỏ sự cần thiết phải gõ lặp lại nhiều câu lệnh Transact-SQL và sau đó biên dịch lặp lại chúng.

Concepts

Các thủ tục lưu trữ có thể nhận các giá trị ở dạng các tham số đầu vào và trả về các giá trị đầu ra như được định nghĩa bằng các tham số đầu ra.

Việc sử dụng các thủ tục lưu trữ đem lại nhiều lợi thế hơn so với việc sử dụng câu lệnh Transact-SQL. Chúng bao gồm:

- **Cải thiện bảo mật**

  Người quản trị cơ sở dữ liệu có thể cải thiện bảo mật bằng cách kết hợp các đặc quyền cơ sở dữ liệu với các thủ tục được lưu trữ. Người dùng có thể được cho phép để thực hiện một thủ tục lưu trữ ngay cả khi người dùng không có sự cho phép để truy cập những bảng hoặc khung nhìn này.

- **Thực thi biên dịch sẵn**

  Các thủ tục lưu trữ được biên dịch trong quá trình thực thi đầu tiên. Đối với mỗi lần thực thi tiếp theo, SQL Server tái sử dụng phiên bản biên dịch sẵn này. Điều này làm giảm thời gian và nguồn lực cần thiết để biên dịch.

- **Giảm lưu lượng máy Client/Server**

  Các thủ tục lưu trữ giúp làm giảm lưu lượng mạng. Khi các câu lệnh Transact-SQL được thực thi một cách riêng biệt, có mức sử dụng mạng riêng để thực thi mỗi câu lệnh. Khi một thủ tục lưu trữ được thực thi, các câu lệnh Transact-SQL được thực thi cùng nhau như một đơn vị đơn lẻ. Đường dẫn mạng không được sử dụng một cách riêng biệt cho việc thực thi từng câu lệnh riêng lẻ. Điều này làm giảm lưu lượng mạng.

- **Tái sử dụng mã**

  Các thủ tục lưu trữ có thể được sử dụng nhiều lần. Điều này giúp loại bỏ sự cần thiết phải gõ lặp lại hàng trăm câu lệnh Transact-SQL mỗi khi một nhiệm vụ tương tự sẽ được thực hiện.

10.11.1 Các loại thủ tục lưu trữ

SQL Server hỗ trợ nhiều loại thủ tục lưu trữ khác nhau. Những công cụ này được mô tả như sau:

- **Các thủ tục lưu trữ do người dùng định nghĩa**

  Các thủ tục lưu trữ do người dùng định nghĩa còn được gọi là các thủ tục lưu trữ tùy chỉnh. Những thủ tục này được sử dụng để tái sử dụng các câu lệnh Transact-SQL để thực hiện các nhiệm vụ lặp lại. Có hai loại thủ tục lưu trữ do người dùng định nghĩa, những thủ tục lưu trữ Transact-SQL và thủ tục lưu trữ Common Language Runtime (CLR).

  Thủ tục lưu trữ Transact-SQL bao gồm các câu lệnh Transact-SQL trong khi các thủ tục lưu trữ CLR được dựa trên những phương pháp CLR của khuôn khổ .NET. Cả hai thủ tục lưu trữ có thể lấy và trả về các tham số do người dùng định nghĩa.

- **Các thủ tục lưu trữ mở rộng**

  Thủ tục lưu trữ mở rộng giúp SQL Server trong việc tương tác với hệ điều hành. Thủ tục lưu trữ mở rộng không phải là đối tượng cư trú của SQL Server. Chúng là các thủ tục được thực hiện như các thư viện liên kết động (DLL) được thực thi bên ngoài môi trường SQL Server. Ứng dụng tương tác với SQL Server gọi DLL tại thời gian chạy. DLL được tải tự động và chạy bằng SQL Server. SQL Server phân bổ không gian để chạy các thủ tục lưu trữ mở rộng. Các thủ tục lưu trữ mở rộng sử dụng tiền tố 'xp'. Các nhiệm vụ phức tạp hoặc không thể được thực thi sử dụng các câu lệnh Transact-SQL được thực hiện bằng cách sử dụng thủ tục lưu trữ mở rộng.

- **Các thủ tục lưu trữ hệ thống**

  Thủ tục lưu trữ hệ thống thường được sử dụng để tương tác với các bảng hệ thống và thực hiện các tác vụ hành chính như cập nhật các bảng hệ thống. Thủ tục lưu trữ hệ thống được bắt đầu với 'sp_'. Những thủ tục này nằm trong cơ sở dữ liệu Resource. Những thủ tục này có thể được nhìn thấy trong lược đồ sys của mỗi hệ thống và cơ sở dữ liệu do người dùng định nghĩa. Thủ tục lưu trữ hệ thống cho phép các quyền GRANT, DENY, và REVOKE.

  Thủ tục lưu trữ hệ thống là một tập hợp các câu lệnh Transact-SQL biên dịch sẵn được thực thi như một đơn vị đơn lẻ. Thủ tục hệ thống được sử dụng trong các hoạt động hành chính và thông tin cơ sở dữ liệu. Những thủ tục này đem lại khả năng dễ dàng truy cập vào thông tin siêu dữ liệu về các đối tượng cơ sở dữ liệu như các bảng hệ thống, bảng do người dùng định nghĩa, khung nhìn, và chỉ mục.

  Thủ tục lưu trữ hệ thống xuất hiện một cách logic trong lược đồ sys của hệ thống và các cơ sở dữ liệu do người dùng định nghĩa. Khi tham chiếu một lưu trữ thủ tục hệ thống, mã định danh lược đồ sys được sử dụng. Thủ tục lưu trữ hệ thống được lưu trữ vật lý trong cơ sở dữ liệu Resource ẩn và có tiền tố sp_. Thủ tục lưu trữ hệ thống được sở hữu bởi người quản trị cơ sở dữ liệu.

Ghi chú - Bảng hệ thống được tạo ra theo mặc định tại thời điểm tạo ra một cơ sở dữ liệu mới. Những bảng này lưu trữ thông tin siêu dữ liệu về các đối tượng do người dùng định nghĩa như các bảng và khung nhìn. Người dùng không thể truy cập hoặc cập nhật các bảng hệ thống sử dụng các thủ tục lưu trữ hệ thống ngoại trừ thông qua các quyền được cấp bởi người quản trị cơ sở dữ liệu.

10.11.2 Phân loại các thủ tục lưu trữ hệ thống

Các thủ tục lưu trữ hệ thống có thể được phân loại thành các thể loại khác nhau tùy thuộc vào các tác vụ mà chúng thực hiện. Một số thể loại quan trọng như sau:

- **Các thủ tục lưu trữ danh mục**

  Tất cả thông tin về các bảng trong cơ sở dữ liệu người dùng được lưu trữ trong một tập hợp các bảng đã gọi danh mục hệ thống đó. Thông tin từ danh mục hệ thống có thể được truy cập bằng cách sử dụng các thủ tục danh mục. Ví dụ, thủ tục lưu trữ danh mục sp_tables hiển thị danh sách của tất cả các bảng trong cơ sở dữ liệu hiện tại.

- **Các thủ tục lưu trữ bảo mật**

  Thủ tục lưu trữ bảo mật được sử dụng để quản lý tính bảo mật của cơ sở dữ liệu. Ví dụ, thủ tục lưu trữ bảo mật sp_changedbowner được sử dụng để thay đổi chủ sở hữu của cơ sở dữ liệu hiện tại..

Concepts

- **Các thủ tục lưu trữ con trỏ**

  Thủ tục con trỏ được sử dụng để thực hiện chức năng của một con trỏ. Ví dụ, thủ tục lưu trữ con trỏ sp_cursor_list liệt kê tất cả các con trỏ được mở bằng kết nối này và mô tả các thuộc tính của chúng.

- **Các thủ tục lưu trữ truy vấn phân phối**

  Thủ tục lưu trữ phân phối được sử dụng trong việc quản lý các truy vấn phân phối. Ví dụ, thủ tục lưu trữ truy vấn phân phối sp_indexes trả về thông tin chỉ mục cho bảng từ xa đã chỉ định.

- **Các thủ tục lưu trữ Database Mail và SQL Mail**

  Các thủ tục lưu trữ Database Mail và SQL Mail được sử dụng để thực hiện các hoạt động email từ bên trong SQL Server. Ví dụ, thủ tục lưu trữ thư cơ sở dữ liệu sp_send_dbmail sẽ gửi các thông báo e-mail đến người nhận đã chỉ định. Thông báo này có thể bao gồm một tập kết quả truy vấn hoặc tập tin đính kèm hoặc cả hai.

10.11.3 Các thủ tục lưu trữ tạm thời

Thủ tục lưu trữ được tạo ra để sử dụng tạm thời trong một phiên được gọi là thủ tục lưu trữ tạm thời. Những thủ tục này được lưu trữ trong cơ sở dữ liệu tempdb. Cơ sở dữ liệu hệ thống tempdb là một nguồn lực toàn cầu có sẵn cho tất cả người dùng được kết nối với một thể hiện của SQL Server. Nó chứa tất cả các bảng tạm thời và các thủ tục lưu trữ tạm thời.

SQL Server hỗ trợ hai loại thủ tục lưu trữ tạm thời cụ thể là cục bộ và toàn cầu. Sự khác biệt giữa hai loại này được đưa ra trong bảng 10.1.

Thủ tục tạm thời cục bộ	Thủ tục tạm thời toàn cầu
Chỉ thấy được cho người dùng đã tạo ra nó	Thấy được tất cả người dùng
Thả bỏ vào cuối phiên hiện tại	Thả bỏ vào cuối phiên cuối cùng
Thủ tục tạm thời cục bộ	Thủ tục tạm thời toàn cầu
Chỉ có thể được sử dụng bởi chủ nhân của nó	Có thể được sử dụng bởi bất kỳ người dùng nào
Sử dụng tiền tố # trước tên thủ tục	Sử dụng tiền tố ## trước tên thủ tục

Table 10.1: Differences Between local and Global Temporary Procedures

Ghi chú - Phiên được thiết lập khi người dùng kết nối với cơ sở dữ liệu và kết thúc khi dùng ngắt kết nối.
Tên đầy đủ của một thủ tục lưu trữ tạm thời toàn cầu bao gồm cả tiền tố # # không thể vượt quá 128 ký tự. Tên đầy đủ của một thủ tục lưu trữ tạm thời cục bộ bao gồm cả tiền tố # không thể vượt quá 116 ký tự.

10.11.4 Các thủ tục lưu trữ từ xa

Thủ tục lưu trữ chạy trên SQL Server từ xa được gọi là thủ tục lưu trữ từ xa. Thủ tục lưu trữ từ xa có thể được sử dụng chỉ khi máy chủ từ xa cho phép truy cập từ xa.

Khi một thủ tục lưu trữ từ xa được thực thi từ một thể hiện cục bộ của SQL Server đến máy tính khách hàng, có thể gặp phải lỗi hủy câu lệnh. Khi một lỗi như vậy xảy ra, câu lệnh đã gây ra lỗi bị chấm dứt nhưng thủ tục từ xa tiếp tục được thực hiện.

10.11.5 Các thủ tục lưu trữ mở rộng

Thủ tục lưu trữ mở rộng được sử dụng để thực hiện các tác vụ mà không thể được thực hiện sử dụng các câu lệnh Transact-SQL tiêu chuẩn. Các thủ tục lưu trữ mở rộng sử dụng tiền tố 'xp_'. Những thủ tục lưu trữ này được chứa trong lược đồ dbo của cơ sở dữ liệu tổng thể. Cú pháp sau đây được sử dụng để thực thi thủ tục lưu trữ mở rộng

Cú pháp:

```
EXECUTE <procedure_name>
```

Code Snippet 33 thực thi thủ tục lưu trữ mở rộng xp_fileexist để kiểm tra xem tập tin MyTest.txt có tồn tại hay không.

CodeSnippet33:

```
EXECUTE xp_fileexist 'c:\MyTest.txt'
```

Ghi chú - Khi bạn thực thi một thủ tục lưu trữ mở rộng, hoặc trong một khối lệnh hoặc trong một mô-đun, xác định tên thủ tục lưu trữ với master.dbo.

10.12 Các thủ tục lưu trữ tùy chỉnh hoặc do người dùng định nghĩa

Trong SQL Server, người dùng được phép tạo ra các thủ tục lưu trữ tùy chỉnh để thực hiện các tác vụ khác nhau. Các thủ tục lưu trữ như vậy được gọi là thủ tục lưu trữ do người dùng định nghĩa hoặc tùy chỉnh.

Ví dụ, hãy xem xét bảng **Customer_Details** lưu trữ các chi tiết về tất cả các khách hàng.
Bạn sẽ cần phải gõ các câu lệnh Transact-SQL mỗi khi bạn muốn xem các chi tiết về khách hàng. Thay vào đó, bạn có thể tạo ra một thủ tục lưu trữ tùy chỉnh sẽ hiển thị những chi tiết này bất cứ khi nào thủ tục đó được thực thi.

Tạo ra một thủ tục lưu trữ tùy chỉnh yêu cầu quyền CREATE PROCEDURE trong cơ sở dữ liệu và quyền ALTER trên lược đồ trong đó thủ tục đang được tạo ra.

Concepts

Cú pháp sau đây được sử dụng để tạo ra thủ tục lưu trữ tùy chỉnh.

Cú pháp:

```
CREATE { PROC | PROCEDURE } procedure_name
  [ { @parameter data_type } ]
AS <sql_statement>
```

trong đó,

procedure_name: chỉ ra tên của thủ tục.

@parameter: chỉ ra các tham số đầu vào/đầu ra trong thủ tục.

data_type: chỉ ra kiểu dữ liệu của các tham số.

sql_statement: chỉ ra một hoặc nhiều câu lệnh Transact-SQL được đưa vào trong thủ tục.

Code Snippet 34 tạo ra và sau đó thực thi một thủ tục lưu trữ tùy chỉnh, **uspGetCustTerritory**, sẽ hiển thị các chi tiết của khách hàng như mã khách hàng, mã lãnh thổ, và tên lãnh thổ.

Code Snippet 34:

```
CREATE PROCEDURE uspGetCustTerritory
AS
SELECT TOP 10 CustomerID, Customer.TerritoryID, Sales.SalesTerritory.Name
FROM Sales.Customer JOIN Sales.SalesTerritory ON Sales.Customer.TerritoryID=
Sales.SalesTerritory.TerritoryID
```

Để thực thi thủ tục lưu trữ này, lệnh EXEC được sử dụng như được trình bày trong Code Snippet 35.

Code Snippet 35:

```
EXEC uspGetCustTerritory
```

Kết quả được trình bày trong hình 10.10

	CustomerID	TerritoryID	Name
1	15	9	Australia
2	33	9	Australia
3	51	9	Australia
4	69	9	Australia
5	87	9	Australia
6	105	9	Australia
7	123	9	Australia
8	141	9	Australia
9	159	9	Australia
10	177	9	Australia

Hình 10.10: Output of a Simple Stored Procedure

10.12.1 Sử dụng các tham số

Lợi thế thực sự của một thủ tục lưu trữ đi vào hình ảnh chỉ khi một hoặc nhiều tham số được sử dụng với nó. Dữ liệu được truyền giữa thủ tục lưu trữ và chương trình gọi khi việc gọi được thực hiện cho một thủ tục lưu trữ. Truyền dữ liệu này được thực hiện bằng cách sử dụng các tham số. Tham số thuộc hai loại như sau:

- **Tham số đầu vào**

  Tham số đầu vào cho phép chương trình gọi truyền các giá trị cho một thủ tục lưu trữ. Những giá trị này được nhận vào các biến đã định nghĩa trong thủ tục lưu trữ.

- **Tham số đầu ra**

  Các tham số đầu ra cho phép thủ tục lưu trữ truyền các giá trị trở lại chương trình gọi. Những giá trị này được nhận vào các biến bằng chương trình gọi.

Những cái này bây giờ được mô tả chi tiết.

- **Tham số đầu vào**

  Các giá trị được truyền từ chương trình gọi đến thủ tục lưu trữ và những giá trị này được chấp nhận vào các tham số đầu vào của thủ tục lưu trữ đó. Các tham số đầu vào được định nghĩa vào thời điểm tạo ra thủ tục lưu trữ. Những giá trị này đã truyền vào các tham số đầu vào có thể là hằng số hoặc biến. Những giá trị này được truyền cho thủ tục tại thời điểm gọi thủ tục. Thủ tục lưu trữ thực hiện các tác vụ đã chỉ định sử dụng những giá trị này.

Concepts

Cú pháp sau đây được sử dụng để tạo ra thủ tục lưu trữ.

```
CREATE PROCEDURE <procedure_name>
@parameter <data_type>
AS <sql_statement>
```

trong đó,

data_type: chỉ ra kiểu dữ liệu do hệ thống định nghĩa.

Cú pháp sau đây được sử dụng để thực thi một thủ tục lưu trữ và truyền giá làm các tham số đầu vào.

Cú pháp :

```
EXECUTE <procedure_name> <parameters>
```

Code Snippet 36 tạo ra thủ tục lưu trữ **uspGetSales** với tham số **territory** chấp nhận tên của lãnh thổ và hiển thị các chi tiết bán hàng và mã nhân viên bán hàng cho lãnh thổ đó. Sau đó, đoạn mã thực thi thủ tục lưu trữ với Northwest được truyền làm tham số đầu vào.

Code Snippet 36:

```
CREATE PROCEDURE uspGetSales
@territory varchar(40)
AS
SELECT BusinessEntityID, B.SalesYTD, B.SalesLastYear
FROM Sales.SalesPerson A
JOIN Sales.SalesTerritory B
ON A.TerritoryID = B.TerritoryID
WHERE B.Name = @territory;

--Execute the stored procedure
EXEC uspGetSales 'Northwest'
```

Kết quả được trình bày trong Hình 10.11.

	BusinessEntityID	SalesYTD	SalesLastYear
1	280	7887186.7882	3298694.4938
2	283	7887186.7882	3298694.4938
3	284	7887186.7882	3298694.4938

Hình 10.11: Using Stored Procedure with Parameters

Concepts

- **Các tham số đầu ra**

Thủ tục lưu trữ đôi khi cần phải trả kết quả ngược lại cho chương trình gọi. Việc truyền dữ liệu này từ thủ tục lưu trữ cho chương trình gọi được thực hiện bằng cách sử dụng các tham số đầu ra.

Các tham số đầu ra được định nghĩa vào thời điểm tạo ra thủ tục. Để chỉ ra một tham số đầu ra, từ khóa OUTPUT được sử dụng trong khi khai báo tham số. Ngoài ra, câu lệnh gọi phải có một biến đã chỉ định với từ khóa OUTPUT để chấp nhận đầu ra từ thủ tục được gọi.

Cú pháp sau đây được sử dụng để truyền các tham số đầu ra trong một thủ tục lưu trữ và sau đó, thực thi thủ tục lưu trữ với tham số OUTPUT được chỉ định.

Cú pháp:

```
EXECUTE <procedure_name> <parameters>
```

Code Snippet 37 tạo ra thủ tục lưu trữ **uspGetTotalSales** với tham số đầu vào **@territory** nhận tên của lãnh thổ và tham số đầu ra **@sum** để hiển thị tổng doanh thu năm đến nay trên lãnh thổ đó.

Code Snippet 37:

```
CREATE PROCEDURE uspGetTotalSales
@territory varchar(40), @sum int OUTPUT
AS
SELECT @sum= SUM(B.SalesYTD)
FROM Sales.SalesPerson A
JOIN Sales.SalesTerritory B
ON A.TerritoryID=B.TerritoryID
WHERE B.Name = @territory
```

Code Snippet 38 khai báo biến **sumsales** để nhận kết quả của thủ tục **uspGetTotalSales**.

Code Snippet 38:

```
DECLARE @sumsales money;
EXEC uspGetTotalSales 'Northwest', @sum = @sum OUTPUT;
PRINT 'The year-to-date sales figure for this territory is ' +
convert(varchar(100),@sumsales);
GO
```

Đoạn mã này truyền Northwest làm đầu vào cho thủ tục lưu trữ **uspGetTotalSales** và nhận kết quả vào biến **sumsales**. Kết quả được in bằng cách sử dụng lệnh PRINT.

Concepts

Các tham số OUTPUT có các đặc điểm sau:

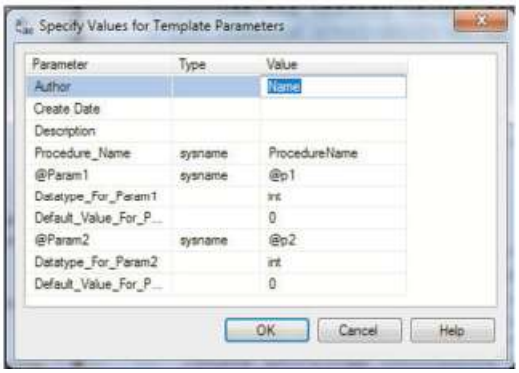
- Tham số không thể thuộc kiểu dữ liệu văn bản và hình ảnh.
- Câu lệnh gọi phải có một biến để nhận giá trị trả về.
- Có thể sử dụng biến trong các câu lệnh Transact-SQL tiếp theo trong khối lệnh hoặc thủ tục gọi.
- Các tham số đầu ra có thể là chỗ dành sẵn con trỏ.

Mệnh đề OUTPUT trả về thông tin từ mỗi hàng trên đó các câu lệnh INSERT, UPDATE, và DELETE đã được thực thi. Mệnh đề này rất hữu ích để lấy giá trị của một nhận dạng hoặc cột tính sau hoạt động INSERT hoặc UPDATE.

10.12.2 Sử dụng SSMS để tạo ra các thủ tục lưu trữ

Bạn cũng có thể tạo ra một thủ tục lưu trữ do người dùng định nghĩa bằng cách sử dụng SSMS. Những bước để thực hiện điều này như sau:

1. Khởi chạy **Object Explorer.**
2. Trong **Object Explorer**, kết nối với một thể hiện của Công cụ cơ sở dữ liệu.
3. Sau khi kết nối thành công với thể hiện, mở rộng thể hiện đó.
4. Mở rộng Databases và sau đó, mở rộng cơ sở dữ liệu AdventureWorks2012.
5. Mở rộng **Programmability**, bấm chuột phải **Stored Procedures**, và sau đó bấm vào **New Stored Procedure.**
6. Trên menu **Query**, bấm vào **Specify Values for Template Parameters**. Hộp thoại **Specify Values for Template Parameters** được hiển thị như được trình bày trong hình 10.12



Hình 10.12: Specify Values for Template Parameters Dialog Box

Concepts

7. Trong hộp thoại **Specify Values for Template Parameters**, nhập các giá trị sau cho các tham số như được trình bày trong bảng 10.2.

Tham số	Giá trị
Tác giả	Tên của bạn
Ngày tạo ra	Ngày hôm nay
Mô tả	Trả về số liệu năm bán hàng cho một
Procedure Name	uspGetTotals
@Param1	@territory
@Datatype_For_Param1	varchar(50)
Default_Value_For_Param1	NULL
@Param2	
@Datatype_For_Param2	
Default_Value_For_Param2	

Table 10.2: Parameter Values

8. Sau khi nhập những chi tiết này, bấm vào **OK**.

9. Trong Query Editor, thay câu lệnh SELECT bằng câu lệnh sau:

```
SELECT BusinessEntityID, B.SalesYTD, B.SalesLastYear
FROM Sales.SalesPerson A
JOIN Sales.SalesTerritory B
ON A.TerritoryID = B.TerritoryID WHERE B.Name = @territory;
```

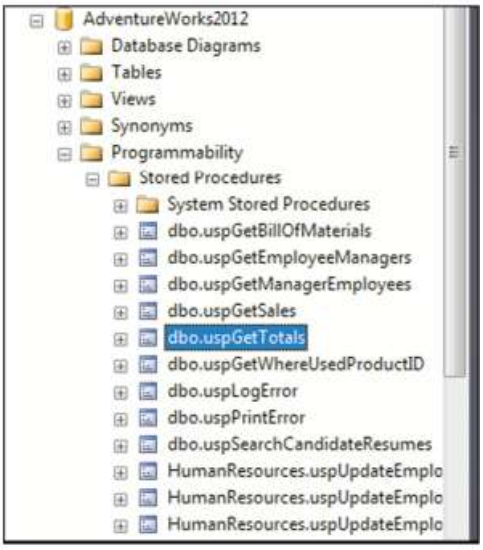
10. Để kiểm tra cú pháp, trên menu **Query**, bấm vào **Parse**. Nếu thông báo lỗi được trả về, so sánh các câu lệnh với thông tin này và chỉnh sửa khi cần thiết.

11. Để tạo ra thủ tục này, từ menu **Query**, bấm vào **Execute**. Thủ tục được tạo ra như một đối tượng trong cơ sở dữ liệu.

12. Để xem thủ tục được liệt kê trong **Object Explorer**, bấm chuột phải vào **Stored Procedures** và chọn **Refresh**.

Concepts

Tên thủ tục sẽ được hiển thị trong cây **Object Explorer** như được trình bày trong hình 10.13.



Hình 10.13: Stored Procedure Seen in Object Explorer

13. Để chạy thủ tục này, trong Object Explorer, bấm chuột phải vào tên thủ tục lưu trữ **uspGetTotals** và chọn **Execute Stored Procedure.**

14. Trong cửa sổ **Execute Procedure**, nhập **Northwest** làm giá trị cho tham số **@territory.**

Ghi chú- Trong SQL Server 2012, thủ tục lưu trữ có thể có dung lượng lên đến 250 MB. Nói cách khác, những byte trong văn bản nguồn của một thủ tục lưu trữ không thể vượt quá 250 MB.

10.13 Xem các định nghĩa thủ tục lưu trữ

Định nghĩa của một thủ tục lưu trữ có thể được xem bằng cách sử dụng thủ tục lưu trữ hệ thống sp_helptext. Để xem định nghĩa, bạn phải chỉ ra tên của thủ tục lưu trữ làm tham số khi thực thi sp_helptext. Định nghĩa này là ở dạng các câu lệnh Transact-SQL.

Các câu lệnh Transact-SQL của định nghĩa thủ tục bao gồm câu lệnh CREATE PROCEDURE cũng như các câu lệnh SQL định nghĩa phần thân của thủ tục.

Cú pháp sau đây được sử dụng để xem định nghĩa của thủ tục lưu trữ.

Cú pháp:

```
sp_helptext '<procedure_name>'
```

Concepts

Code Snippet 39 hiển thị định nghĩa của thủ tục lưu trữ có tên là **uspGetTotals**.

Code Snippet 39:

```
EXEC sp_helptext uspGetTotals
```

10.14 Sửa chữa và thả bỏ các thủ tục lưu trữ

Những cấp quyền liên quan đến thủ tục lưu trữ bị mất khi thủ tục lưu trữ được tái tạo. Tuy nhiên, khi một thủ tục lưu trữ được thay đổi, những cấp quyền đã định nghĩa cho thủ tục lưu trữ vẫn như cũ mặc dù định nghĩa thủ tục được thay đổi.

Có thể thay đổi thủ tục bằng cách sử dụng câu lệnh **ALTER PROCEDURE**. Cú pháp sau đây được sử dụng để sửa đổi thủ tục lưu trữ.

Cú pháp:

```
ALTER PROCEDURE <procedure_name>
@parameter <data_type> [ OUTPUT ]
[ WITH { ENCRYPTION | RECOMPILE } ]
AS <sql_statement>
```

trong đó,

ENCRYPTION : mã hóa định nghĩa thủ tục lưu trữ.

RECOMPILE : chỉ ra rằng thủ tục được biên dịch vào thời gian chạy.

sql_statement : chỉ ra các câu lệnh Transact-SQL được đưa vào trong phần thân của thủ tục.

Code Snippet 40 sửa đổi định nghĩa của thủ tục lưu trữ có tên là **uspGetTotals** để thêm cột mới CostYTD được lấy từ Sales.SalesTerritory.

Code Snippet 40:

```
ALTER PROCEDURE [dbo].[uspGetTotals]
     @territory varchar = 40
AS
     SELECT BusinessEntityID, B.SalesYTD, B.CostYTD, B.SalesLastYear
FROM Sales.SalesPerson A
JOIN Sales.SalesTerritory B
ON A.TerritoryID = B.TerritoryID
WHERE B.Name = @territory;
GO
```

Concepts

Ghi chú- Khi bạn thay đổi định nghĩa của một thủ tục lưu trữ, những đối tượng phụ thuộc có thể thất bại khi được thực thi. Điều này xảy ra nếu các đối tượng phụ thuộc không được cập nhật để phản ánh những thay đổi được thực hiện cho thủ tục lưu trữ.

- **Hướng dẫn sử dụng câu lệnh ALTER PROCEDURE**

Các thủ tục lưu trữ được thay đổi bằng cách sử dụng câu lệnh ALTER PROCEDURE. Các sự kiện sau đây phải được xem xét trong khi sử dụng câu lệnh ALTER PROCEDURE:

- Khi thủ tục lưu trữ được tạo ra bằng cách sử dụng các tùy chọn như là tùy chọn WITH ENCRYPTION, những tùy chọn này cũng nên được đưa vào trong câu lệnh ALTER PROCEDURE.
- Câu lệnh ALTER PROCEDURE thay đổi một thủ tục đơn lẻ. Khi thủ tục lưu trữ gọi các thủ tục lưu trữ khác, những thủ tục lưu trữ lồng nhau không bị ảnh hưởng bằng cách thay đổi thủ tục gọi.
- Những bộ tạo lập thủ tục lưu trữ, các thành viên của vai trò máy chủ sysadmin và các thành viên của vài trò cơ sở dữ liệu cố định db _ owner và db _ ddladmin có quyền thực thi câu lệnh ALTER PROCEDURE.
- Đề nghị là bạn không nên sửa đổi các thủ tục lưu trữ hệ thống. Nếu bạn cần thay đổi chức năng của một thủ tục lưu trữ hệ thống, khi đó sẽ tạo ra một thủ tục lưu trữ hệ thống do người dùng định nghĩa bằng cách sao chép các câu lệnh từ một thủ tục lưu trữ hiện có và sửa đổi thủ tục do người dùng định nghĩa này.

- **Thả bỏ các thủ tục lưu trữ**

Thủ tục lưu trữ có thể được thả bỏ nếu chúng không còn cần thiết. Nếu thủ tục lưu trữ khác gọi một thủ tục đã xóa, thông báo lỗi được hiển thị.

Nếu thủ tục mới được tạo ra sử dụng cùng tên cũng như cùng các tham số như thủ tục đã thả bỏ, tất cả các lần gọi đến thủ tục đã thả bỏ sẽ được thực thi thành công. Điều này là do bây giờ chúng sẽ tham khảo thủ tục mới, trong đó có cùng tên và các tham số như thủ tục đã xóa.

Trước khi thả bỏ một thủ tục lưu trữ, hãy thực thi thủ tục lưu trữ hệ thống sp_depends để xác định đối tượng nào phụ thuộc vào thủ tục này.

Thủ tục được thả bỏ sử dụng câu lệnh DROP PROCEDURE. Cú pháp sau đây được sử dụng để thả bỏ thủ tục lưu trữ.

Cú pháp :

```
DROP PROCEDURE <procedure_name>
```

Code Snippet 41 thả bỏ thủ tục lưu trữ **uspGetTotals**.

Code Snippet 41:

```
DROP PROCEDURE uspGetTotals
```

Concepts

10.15 Các thủ tục lưu trữ lồng nhau

SQL Server 2012 cho phép các thủ tục lưu trữ được gọi bên trong các thủ tục lưu trữ khác. Những thủ tục được gọi đến lượt mình có thể gọi các thủ tục khác. Kiến trúc gọi một thủ tục từ một thủ tục khác này được gọi là kiến trúc thủ tục lưu trữ lồng nhau.

Khi thủ tục lưu trữ gọi một thủ tục lưu trữ khác, mức độ lồng nhau được cho là tăng lên một. Tương tự như vậy, khi thủ tục gọi hoàn thành việc thực thi của nó và chuyển điều khiển trở lại cho thủ tục gọi, mức độ lồng nhau được cho là giảm đi một. Mức lồng nhau tối đa được SQL Server 2012 hỗ trợ là 32. Nếu mức lồng nhau vượt quá 32, quá trình gọi thất bại. Ngoài ra, lưu ý rằng nếu thủ tục lưu trữ cố gắng truy cập hơn 64 cơ sở dữ liệu, hoặc nhiều hơn hai cơ sở dữ liệu trong kiến trúc lồng nhau, sẽ có lỗi.

Code Snippet 42 được sử dụng để tạo ra thủ tục lưu trữ **NestedProcedure** gọi hai thủ tục lưu trữ khác đã được tạo ra trước đó thông qua Code Snippets 34 và 36.

Code Snippet 42:

```
CREATE PROCEDURE NestedProcedure
AS
BEGIN
EXEC uspGetCustTerritory
EXEC uspGetSales 'France'
END
```

Khi thủ tục **NestedProcedure** được thực thi, thủ tục này đến lượt mình gọi các thủ tục lưu trữ **uspGetCustTerritory** và **uspGetSales** và truyền giá trị France làm tham số đầu vào cho thủ tục lưu trữ **uspGetSales**.

Ghi chú- Mặc dù có thể có tối đa là 32 mức lồng nhau, không có giới hạn về số lượng thủ tục lưu trữ có thể được gọi từ một thủ tục lưu trữ đã cho.

10.15.1 Hàm @@NESTLEVEL

Mức lồng nhau của thủ tục hiện tại có thể được xác định bằng cách sử dụng hàm @@NESTLEVEL. Khi hàm@@NESTLEVEL được thực thi trong chuỗi Transact-SQL, giá trị trả về là mức lồng nhau hiện tại + 1.

Nếu bạn sử dụng sp_executesql để thực thi hàm @@NESTLEVEL, giá trị trả về là mức lồng nhau hiện tại + 2 (như là một thủ tục lưu trữ có tên là sp_executesql, được thêm vào chuỗi lồng nhau).

Cú pháp:

```
@@NESTLEVEL
```

Concepts

trong đó,

@@**NESTLEVEL**: Là hàm trả về một giá trị số nguyên chỉ ra mức lồng nhau.

Code Snippet 43tạo ra và thực hiện thủ tục **Nest_Procedure** sẽ thực thi hàm @@NESTLEVEL để xác định mức lồng nhau trong ba kịch bản khác nhau.

Code Snippet 43:

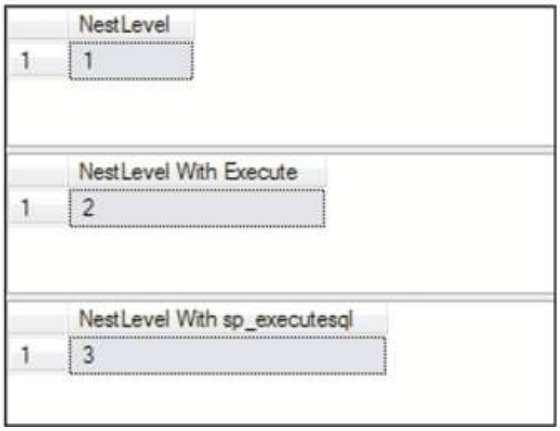
```
CREATE PROCEDURE Nest_Procedure
AS
SELECT @@NESTLEVEL AS NestLevel;
EXECUTE ('SELECT @@NESTLEVEL AS [NestLevel With Execute]');
EXECUTE sp_executesql N'SELECT @@NESTLEVEL AS [NestLevel With sp_executesql]';
```

Code Snippet 44thực thi thủ tục lưu trữ **Nest_Procedure**.

Code Snippet 44:

```
EXECUTE Nest_Procedure
```

Ba kết quả được hiển thị trong hình 10.14 cho ba phương pháp khác nhau được sử dụng để gọi hàm @@**NESTLEVEL**.



Hình 10.14:Using @@NESTLEVEL

Ghi chú - Thủ tục lưu trữ **sp_executesql** còn có thể được sử dụng để thực thi các câu lệnh Transact-SQL.

10.16 Truy vấn siêu dữ liệu hệ thống

Những thuộc tính của một đối tượng như bảng hoặc khung nhìn được lưu trữ trong các bảng hệ thống đặc biệt. Những thuộc tính này được gọi là siêu dữ liệu. Tất cả các đối tượng SQL tạo ra siêu dữ liệu. Siêu dữ liệu này có thể được xem bằng cách sử dụng các khung nhìn hệ thống, là các khung nhìn được xác định trước của SQL Server.

Concepts

Có hơn 230 khung nhìn hệ thống khác nhau và chúng được tự động đưa vào cơ sở dữ liệu do người dùng tạo ra. Những khung nhìn này được nhóm lại thành một số lược đồ khác nhau.

- **Các khung nhìn danh mục hệ thống**

  Những cái này chứa thông tin về danh mục trong hệ thống SQL Server. Danh mục tương tự như một hàng tồn kho các đối tượng. Những khung nhìn này có chứa một loạt các siêu dữ liệu. Trong các phiên bản trước đây của SQL Server, người dùng phải truy vấn một số lượng lớn các bảng hệ thống, khung nhìn hệ thống, và các hàm hệ thống. Trong SQL Server 2012, tất cả các siêu dữ liệu danh mục người dùng có thể truy cập có thể dễ dàng được tìm thấy bằng cách truy vấn chỉ các khung nhìn danh mục.

  Code Snippet 45 lấy một danh sách các bảng và các thuộc tính người dùng từ khung nhìn danh mục hệ thống sys.tables.

  Code Snippet 45:

```
SELECT name, object_id, type, type_desc
FROM sys.tables;
```

- **Các khung nhìn lược đồ thông tin**

  Người dùng có thể truy vấn các khung nhìn lược đồ thông tin để trả về siêu dữ liệu hệ thống. Những khung nhìn này sẽ hữu ích cho các công cụ của bên thứ ba có thể không phải là cụ thể cho SQL Server. Các khung nhìn lược đồ thông tin cung cấp một khung nhìn nội bộ, hệ thống độc lập với bảng của siêu dữ liệu SQL Server. Các khung nhìn lược đồ thông tin cho phép các ứng dụng làm việc một cách chính xác mặc dù các thay đổi đáng kể đã được thực hiện cho những bảng hệ thống nằm dưới.

  Những điểm sau đây trong bảng 10.3 sẽ giúp quyết định xem nên truy vấn các khung nhìn hệ thống cụ thể cho SQL Server hay các khung nhìn lược đồ thông tin:

Các khung nhìn lược đồ thông tin	Các khung nhìn hệ thống SQL
Chúng được lưu trữ trong lược đồ riêng INFORMATION_SCHEMA.	Chúng xuất hiện trong lược đồ sys.
Chúng sử dụng thuật ngữ tiêu chuẩn thay vì thuật ngữ SQL Server. Ví dụ, chúng sử dụng danh mục thay vì cơ sở dữ liệu và tên miền thay vì kiểu dữ liệu do người dùng định nghĩa.	Chúng tuân thủ bảng thuật ngữ SQL SQL Server.
Chúng có thể không tiếp xúc với tất cả các siêu dữ liệu có sẵn cho các khung nhìn danh mục của SQL Server. Ví dụ, sys.columns bao gồm các thuộc tính cho thuộc tính nhận dạng và thuộc tính cột tính, trong khi Information_schema.columns lại không	Chúng có thể tiếp xúc với tất cả các siêu có sẵn cho các khung nhìn danh mục của SQL Server

Table 10.3: Information schema views and SQL Server-specific system views

Code Snippet 46 lấy dữ liệu từ khung nhìn **INFORMATION_SCHEMA.TABLES** trong cơ sở dữ liệu **AdventureWorks2012** :

Code Snippet 46:

```
SELECT TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME, TABLE_TYPE
FROM INFORMATION_SCHEMA.TABLES;
```

- **Các hàm siêu dữ liệu hệ thống**

  Ngoài các khung nhìn, SQL Server cung cấp một số các hàm dựng sẵn trả về siêu dữ liệu cho truy vấn. Chúng bao gồm các hàm vô hướng và hàm có giá trị bảng, có thể trả về thông tin về các thiết lập hệ thống, tùy chọn phiên, và một loạt các đối tượng.

  Các hàm siêu dữ liệu SQL Server đến ở một loạt các định dạng. Một số xuất hiện tương tự như các hàm vô hướng tiêu chuẩn, như là **ERROR_NUMBER()**. Những cái khác sử dụng tiền tố đặc biệt, chẳng hạn như @@**VERSION** hoặc **$PARTITION**. Bảng 10.4 trình bày một số hàm siêu dữ liệu hệ thống thông thường.

Tên hàm	Mô tả	Ví dụ
OBJECT_ID(<object_name>)	Trả về mã đối tượng của một đối tượng cơ sở dữ liệu.	OBJECT_ID('Sales.Customer')
OBJECT_NAME(<object_id>)	Trả về tên tương ứng với một mã đối tượng.	OBJECT_NAME(197575742)
@@ERROR	Trả về 0 nếu câu lệnh cuối cùng đã thành công, nếu không trả về số lỗi.	@@ERROR
SERVERPROPERTY(<property>)	Trả về giá trị thuộc tính máy chủ đã chỉ định.	SERVERPROPERTY('Collation')

Table 10.4: Common System Metadata Functions

Code Snippet 47 sử dụng câu lệnh SELECT để truy vấn hàm siêu dữ liệu hệ thống.

Code Snippet 47:

```
SELECT SERVERPROPERTY('EDITION') AS EditionName;
```

10.17 Truy vấn các đối tượng quản lý động

Đầu tiên được giới thiệu trong SQL Server 2005, Khung nhìn quản lý động (DMV) và Hàm quản lý động (DMF) là các đối tượng quản lý động trả về thông tin trạng thái máy chủ và cơ sở dữ liệu. DMV và DMF được gọi chung là đối tượng quản lý động. Chúng cung cấp cái nhìn sâu sắc hữu ích vào hoạt động của phần mềm và có thể được sử dụng để kiểm tra trạng thái của thể hiện SQL Server, xử lý sự cố, và điều chỉnh hiệu năng.

Concepts

Cả DMV và DMF trả về dữ liệu dưới dạng bảng nhưng sự khác biệt là trong khi DMF bình thường nhận ít nhất một tham số, DMV không nhận các tham số. SQL Server 2012 cung cấp gần 200 đối tượng quản lý động. Để truy vấn các DMV, cần thiết phải có quyền VIEW SERVER STATE hoặc VIEW DATABASE STATE, tùy thuộc vào phạm vi của DMV.

10.17.1 Phân loại và truy vấn các DMV

Table 10.5 liệt kê quy ước đặt tên giúp tổ chức DMV theo hàm.

Mẫu đặt tên	Mô tả
db	cơ sở dữ liệu có liên quan
io	số liệu thống kê nhập/xuất
Os	Thông tin hệ điều hành SQL Server
"tran"	giao tác có liên quan
"exec"	truy vấn siêu dữ liệu liên quan đến thực thi

Table 10.5: Organizing DMVS by Function

Để truy vấn một đối tượng quản lý động, bạn sử dụng câu lệnh SELECT như khi bạn làm với bất kỳ khung nhìn do người dùng định nghĩa hoặc hàm có giá trị bảng. Ví dụ, Đoạn mã 48 trả về một danh sách các kết nối người dùng hiện tại từ khung nhìn sys.dm_exec_sessions.

sys.dm_exec_sessions là một DMV thuộc phạm vi máy chủ hiển thị thông tin về tất cả các kết nối người dùng hoạt động và các tác vụ nội bộ. Thông tin này bao gồm người dùng đăng nhập, thiết lập phiên hiện tại, phiên bản máy khách, tên chương trình máy khách, thời gian máy khách đăng nhập, và vân vân. sys.dm_exec_sessions có thể được sử dụng để xác định một phiên cụ thể và tìm kiếm thông tin về nó.

Code Snippet 48:

```
SELECT session_id, login_time, program_name
FROM sys.dm_exec_sessions
WHERE login_name = 'sa' and is_user_process = 1;
```

Ở đây, is_user_process là một cột trong khung nhìn xác định xem phiên có phải là một phiên hệ thống hay không. Giá trị 1 chỉ ra rằng nó không phải là một phiên hệ thống mà là một phiên người dùng. Cột program_name xác định tên của chương trình máy khách đã khởi tạo phiên này. Cột login_time thiết lập thời gian khi phiên bắt đầu. Đầu ra của Đoạn mã 48 được trình bày trong hình 10.15.

	session_id	login_time	program_name
1	51	2013-01-29 12:26:08.443	Microsoft SQL Server Management Studio
2	53	2013-01-29 12:26:20.247	Microsoft SQL Server Management Studio - Query

Hình 10.15: Querying the sys.dm_exec_sessions DMV

10.18 Kiểm tra tiến độ của bạn

1. Câu nào sau đây về khung nhìn là đúng?

a.	Khung nhìn cho phép bạn xem và thao tác các phần đã chọn của bảng.
b.	Chỉ có thể chọn các cột từ một bảng cho một khung nhìn, các hàng thì không thể.
c.	Các khung nhìn hệ thống hiển thị thông tin về hệ thống hoặc máy.
d.	Các khung nhìn có tối đa 1024 cột.
e.	Khi dữ liệu trong khung nhìn được thay đổi, nó không được phản ánh trong bảng nằm dưới.

(A)	a, c, d, e	(C)	a, b, d
(B)	a,c	(D)	Tất cả các câu bên trên

2. Bạn đang tạo ra khung nhìn **Supplier _ View** với các cột **FirstName**, **LastName**, và **City** từ bảng **Supplier _ Details**. Đoạn mã nào sau đây vi phạm định nghĩa về khung nhìn? **CREATE VIEW Supplier_View**?

(A)	`CREATE VIEW Supplier_View AS SELECT FirstName, LastName, City FROM Supplier_Details WHERE City IN('New York', 'Boston', 'Orlando')`	(C)	`CREATE VIEW Supplier_View AS SELECT FirstName, LastName, City FROM Supplier_Details ORDER BY FirstName`
(B)	`CREATE VIEW Supplier_View AS SELECT TOP 100 FirstName, LastName, City FROM Supplier_Details WHERE FirstName LIKE 'A%' ORDER BY FirstName`	(D)	`CREATE VIEW Supplier_View AS SELECT TOP 100 FirstName, LastName, City FROM Supplier_Details`

3. Câu nào sau đây về tùy chọn **CHECK OPTION** và **SCHEMABINDING** là đúng?

a.	CHECK OPTION đảm bảo tính toàn vẹn thực thể.
b.	Tùy chọn SCHEMABINDING liên kết khung nhìn với lược đồ của bảng cơ sở.
c.	Khi hàng được sửa đổi, WITH CHECK OPTIONđảm bảo rằng dữ liệu vẫn có thể nhìn thấy thông qua khung nhìn.
d.	Tùy chọn SCHEMABINDING đảm bảo bảng cơ sở không thể được sửa đổi theo cách sẽ ảnh hưởng đến định nghĩa khung nhìn.
e.	Tùy chọn SCHEMABINDING không thể được sử dụng với các câu lệnh ALTER VIEW.

(A)	a,b,c	(C)	b,C, d
(B)	b, c	(D)	c, d, e

4. Bạn muốn tạo ra khung nhìn **Account _ Details** với tùy chọn SCHEMABINDING. Đoạn mã nào sau đây sẽ đạt được mục tiêu này?

(A)	CREATE VIEW Account_Details AS SELECT AccNo, City FROM dbo.Customer_Details WITH SCHEMABINDING	(C)	CREATE VIEW Account_Details WITH SCHEMABINDING AS SELECT AccNo, City FROM dbo.Customer_Details
(B)	CREATE VIEW Account_Details SCHEMABINDING AS SELECT AccNo, City FROM Customer_Details	(D)	CREATE VIEW Account_Details WITH SCHEMABINDING AS SELECT AccNo, City FROM Customer_Details

Concepts

5. Bảng **Item _ Details** được tạo ra với các cột **ItemCode**, **ItemName**, **Price**, và **Quantity**. Cột **ItemCode** được định nghĩa là PRIMARY KEY, **ItemName** được định nghĩa với các ràng buộc UNIQUE và NOT NULL, **Price** được định nghĩa với ràng buộc NOT NULL, và **Quantity** được định nghĩa với ràng buộc NOT NULL và có giá trị mặc định đã chỉ định. Khung nhìn nào sau đây được tạo ra bằng cách sử dụng các cột từ bảng **Item _ Details** có thể được sử dụng để chèn các bản ghi vào bảng này?

(A)	`CREATE VIEW ItemDetails AS SELECT ItemCode, ItemName, Price FROM Item_Details`	(C)	`CREATE VIEW ItemDetails AS SELECT ItemName, Price, Quantity FROM Item_Details`
(B)	`CREATE VIEW ItemDetails AS SELECT ItemCode, Price, Quantity FROM Item_Details`	(D)	`CREATE VIEW ItemDetails AS SELECT ItemCode, ItemName, Quantity FROM Item_Details`

6. Câu nào sau đây về các thủ tục lưu trữ là đúng?

a.	Thủ tục lưu trữ là một nhóm các câu lệnh Transact-SQL hành động như một khối mã được sử dụng để thực hiện một tác vụ đặc thù.
b.	Tất cả các thủ tục lưu trữ hệ thống được xác định bằng tiền tố 'xp_'.
c.	Thủ tục lưu trữ phân phối được sử dụng trong việc quản lý các truy vấn phân phối.
d.	Các thủ tục Database Mail và SQL Mail được sử dụng để thực hiện các hoạt động e-mail trong SQL Server.
e.	Các thủ tục lưu trữ do người dùng định nghĩa còn được gọi là các thủ tục lưu trữ tùy chỉnh.

(A)	a,d	(C)	a, c, d,e
(B)	b, c, e	(D)	d

10.18.1 Đáp án

1.	B
2.	C
3.	C
4.	B
5.	A
6.	C

Tóm tắt

- Khung nhìn là một bảng ảo được tạo thành từ các cột đã chọn từ một hoặc nhiều bảng và được tạo ra bằng cách sử dụng lệnh CREATE VIEW trong SQL Server.
- Người dùng có thể thao tác dữ liệu trong các khung nhìn, như là chèn vào các khung nhìn, sửa đổi dữ liệu trong các khung nhìn, và xóa khỏi các khung nhìn.
- Thủ tục lưu trữ là một nhóm các câu lệnh Transact-SQL hành động như một khối mã duy nhất thực hiện một nhiệm vụ cụ thể.
- SQL Server hỗ trợ nhiều loại thủ tục lưu trữ khác nhau, chẳng hạn như Thủ tục lưu trữ do người dùng định nghĩa, Thủ tục lưu trữ mở rộng, và Thủ tục lưu trữ hệ thống.
- Thủ tục lưu trữ hệ thống có thể được phân loại thành các loại khác nhau như là Thủ tục lưu trữ danh mục, Thủ tục lưu trữ bảo mật, và Thủ tục lưu trữ con trỏ.
- Các tham số đầu vào và đầu ra có thể được sử dụng với các thủ tục lưu trữ để truyền và nhận dữ liệu từ các thủ tục lưu trữ.
- Những thuộc tính của đối tượng như là bảng hoặc khung nhìn được lưu trữ trong các bảng hệ thống đặc biệt và được gọi là siêu dữ liệu.
- DMV và DMF là đối tượng quản lý động trả về thông tin trạng thái máy chủ và cơ sở dữ liệu. DMV và DMF được gọi chung là đối tượng quản lý động.

1. Trong SQL Server 2012 Management Studio, xác định vị trí các thủ tục lưu trữ mở rộng được định nghĩa dưới cơ sở dữ liệu chính và thực thi những thủ tục sau đây trong cửa sổ truy vấn:

   sys.xp_readerrorlog

   sys.xp_getnetname

   sys.xp_fixeddrives

2. ShoezUnlimited là một cửa hàng giày hợp thời trang có trụ sở tại Miami. Họ lưu kho các loại giày dép khác nhau ở cửa hàng của mình và bán chúng để có lợi nhuận. ShoezUnlimited duy trì các chi tiết của tất cả các sản phẩm trong một cơ sở dữ liệu SQL Server 2012. Ban quản lý muốn nhà phát triển của họ sử dụng các thủ tục lưu trữ cho các nhiệm vụ thường được thực hiện. Giả sử bạn là nhà phát triển đó, hãy thực hiện các nhiệm vụ sau đây :

   Tạo bảng Shoes có cấu trúc như được trình bày trong bảng 10.6 trong cơ sở dữ liệu ShoezUnlimited.

Tên trường	Loại dữ liệu	Khóa	Mô tả
ProductCode	varchar(S)	Primary Key	Mã sản phẩm xác định duy nhất mỗi chiếc giày
BrandName	varchar(30)		Tên thương hiệu của giày
Category	varchar(30)		Loại giày, ví dụ như là giày thể thao, mặc giản dị, mặc hội hè, và vân vân
UnitPrice	money		Giá giày tính bằng đô la
QtyOnHand	int		Số lượng có sẵn

Table 10.6: Shoes Table

3. Thêm ít nhất 5 bản ghi cho bảng. Đảm bảo rằng giá trị của cột **QtyOnHand** nhiều hơn 20 cho mỗi đôi giày.
4. Viết các câu lệnh để tạo ra thủ tục lưu trữ có tên là **PriceIncrease** sẽ tăng **unitprice** của tất cả các đôi giày lên đến 10 đô la.
5. Viết các câu lệnh để tạo ra thủ tục lưu trữ **QtyOnHand** sẽ giảm số lượng có sẵn của các thương hiệu đã chỉ định xuống còn 25. Tên thương hiệu cần được cung cấp làm đầu vào.
6. Thực thi các thủ tục lưu trữ **PriceIncrease** và **QtyOnHand**.