Session - 13

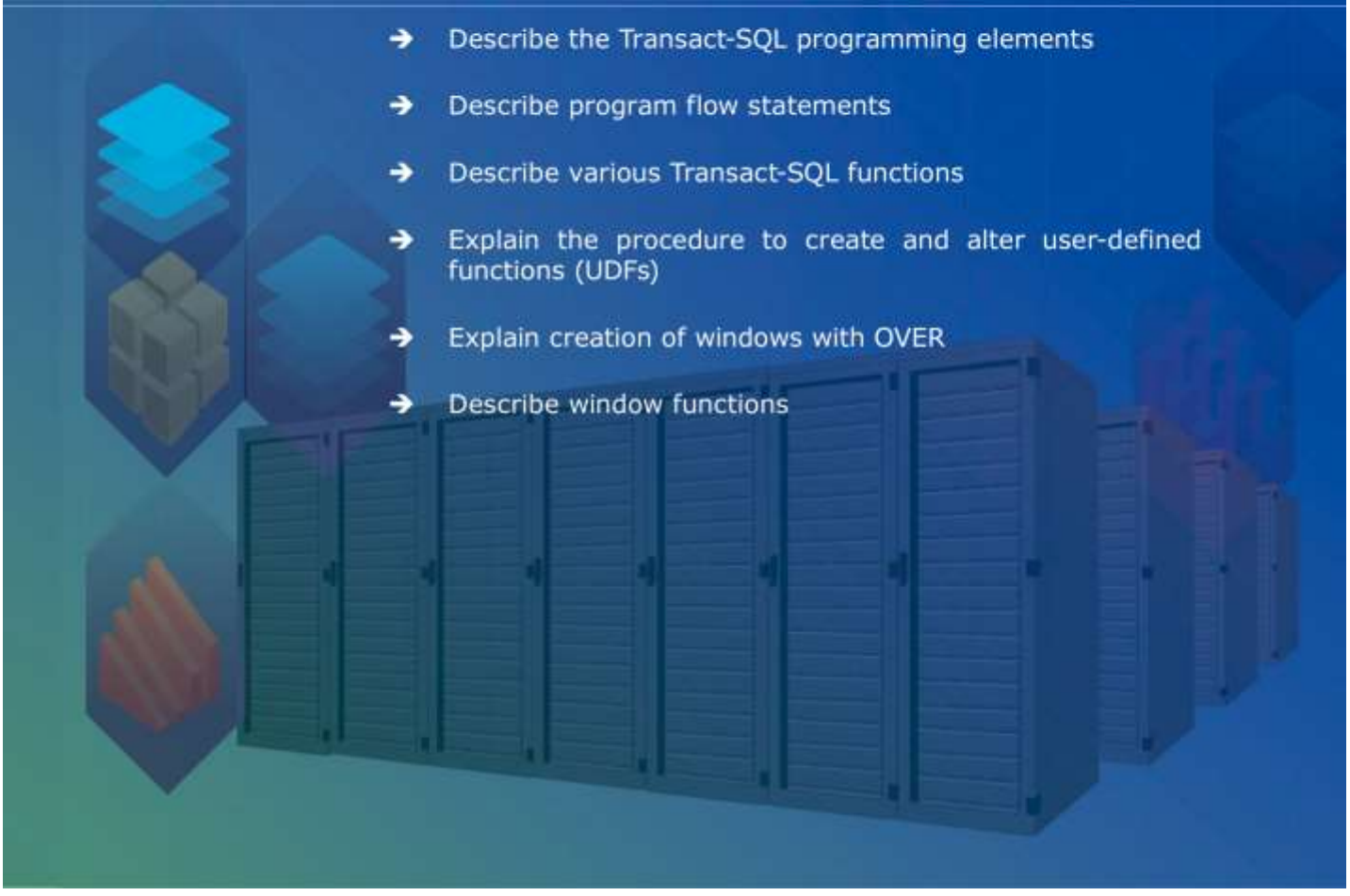
Programming Transact-SQL

Welcome to the Session, **Programming Transact-SQL**.

This session introduces programming with Transact-SQL and describes various Transact-SQL programming elements. The session also describes program flow statements, Transact-SQL functions, and so on. The session further explains the procedure to create and alter user-defined functions, and create windows using the OVER and window functions.

In this Session, you will learn to:

- Describe an overview of Transact-SQL programming
- Describe the Transact-SQL programming elements
- Describe program flow statements
- Describe various Transact-SQL functions
- Explain the procedure to create and alter user-defined functions (UDFs)
- Explain creation of windows with OVER
- Describe window functions



13.1 Giới thiệu

Lập trình Transact-SQL là một phần mở rộng ngôn ngữ thủ tục cho SQL. Lập trình Transact-SQL được mở rộng bằng cách thêm các thường trình con và các cấu trúc lập trình tương tự như các ngôn ngữ cấp cao. Như các ngôn ngữ cấp cao, lập trình Transact-SQL cũng có những quy tắc và cú pháp kiểm soát và cho phép các câu lệnh lập trình để làm việc cùng nhau. Người dùng có thể kiểm soát dòng chảy của các chương trình bằng cách sử dụng các câu lệnh điều kiện như IF và các vòng lặp như là WHILE.

13.2 Các phần tử lập trình Transact-SQL

Các phần tử lập trình Transact-SQL cho phép thực hiện các phép tính khác nhau mà không thể được thực hiện trong một câu lệnh duy nhất. Người dùng có thể nhóm một số câu lệnh Transact-SQL cùng nhau bằng cách sử dụng một trong những cách sau:

- **Khối lệnh**

  Khối lệnh là tập hợp một hoặc nhiều câu lệnh Transact-SQL được gửi như một đơn vị từ một ứng dụng đến máy chủ.

- **Thủ tục lưu trữ**

  Thủ tục lưu trữ là một tập hợp các câu lệnh Transact-SQL được biên dịch sẵn và định nghĩa trước trên máy chủ.

- **Triggers**

  Trigger là một loại thủ tục lưu trữ đặc biệt được thực hiện khi người dùng thực hiện một sự kiện như là phép tính INSERT, DELETE, or UPDATE trên bảng.

- **Scripts**

  Tập lệnh (Script) là một chuỗi các câu lệnh Transact-SQL được lưu trữ trong một tập tin được sử dụng làm đầu vào cho trình biên tập mã SSMS hoặc tiện ích sqlcmd.

Những tính năng sau đây cho phép người dùng làm việc với các câu lệnh Transact-SQL:

- **Các biến**

  Biến cho phép người dùng lưu trữ dữ liệu có thể được sử dụng làm đầu vào trong một câu lệnh Transact-SQL.

- **Điều khiển luồng**

  Điều khiển luồng được sử dụng để đưa vào các cấu trúc điều kiện trong Transact-SQL.

- **Xử lý lỗi**

  Xử lý lỗi là một cơ chế được sử dụng để xử lý các lỗi và cung cấp thông tin cho người dùng về lỗi đã xảy ra.

Concepts

13.2.1 Các khối lệnh Transact-SQL

Khối lệnh Transact-SQL là nhóm một hoặc nhiều câu lệnh Transact-SQL được gửi đến máy chủ như một đơn vị từ một ứng dụng. SQL Server biên dịch khối lệnh SQL vào một đơn vị thực thi duy nhất, còn được gọi là một kế hoạch thực hiện. Trong kế hoạch thực hiện, những câu lệnh SQL được thực hiện từng cái một. Khối lệnh Transact-SQL nên được kết thúc bằng dấu chấm phẩy. Điều kiện này là không bắt buộc, nhưng tiện ích này kết thúc câu lệnh mà không có một dấu chấm phẩy bị phản đối và có thể bị loại bỏ trong các phiên bản mới của SQL Server trong tương lai. Do đó, nên sử dụng các dấu chấm phẩy để kết thúc các khối lệnh.

Lỗi biên dịch như là lỗi cú pháp hạn chế việc biên dịch kế hoạch thực hiện. Vì vậy, nếu lỗi thời gian biên dịch xảy ra, không có câu lệnh trong khối lệnh được thực hiện.

Lỗi thời gian chạy như là vi phạm ràng buộc hoặc tràn số học có một trong những tác động sau :

- Hầu hết các lỗi thời gian chạy dừng câu lệnh hiện tại và các câu lệnh tiếp theo trong khối lệnh.
- Lỗi thời gian chạy cụ thể như là vi phạm ràng buộc dừng chỉ câu lệnh hiện có và các câu lệnh còn lại trong khối lệnh được thực hiện.

Các câu lệnh SQL thực hiện trước khi gặp phải lỗi thời gian chạy sẽ không bị ảnh hưởng. Ngoại lệ duy nhất là khi khối lệnh ở trong một giao tác và lỗi dẫn đến giao tác được phục hồi.

Ví dụ, giả sử có 10 câu lệnh trong một khối lệnh và câu lệnh thứ sáu có lỗi cú pháp, khi đó những câu lệnh còn lại trong khối lệnh sẽ không thực hiện. Nếu khối lệnh được biên dịch và câu lệnh thứ ba không chạy, khi đó, kết quả của hai câu lệnh đầu tiên vẫn không bị ảnh hưởng vì nó đã được thực hiện.

Những quy tắc sau đây được áp dụng để sử dụng các khối lệnh:

1. Câu lệnh **CREATE FUNCTION, CREATE DEFAULT, CREATE RULE, CREATE TRIGGER, CREATE PROCEDURE, CREATE VIEW,** và **CREATE SCHEMA** không thể được sử dụng cùng với các câu lệnh khác trong một khối lệnh. Câu lệnh CREATE SQL bắt đầu khối lệnh và tất cả các câu lệnh khác có bên trong khối lệnh sẽ được coi như là một phần của định nghĩa câu lệnh CREATE.
2. Không có thay đổi được thực hiện trong bảng và các cột mới tham khảo cùng một khối lệnh.
3. Nếu câu lệnh đầu tiên trong một khối lệnh có câu lệnh EXECUTE, khi đó, từ khóa EXECUTE là không cần thiết. Nó chỉ cần thiết khi câu lệnh EXECUTE không tồn tại trong câu lệnh đầu tiên trong khối lệnh.

Code Snippet 1 tạo ra một khung nhìn trong khối lệnh.

Code Snippet 1:

```
USE AdventureWorks2012;
GO
CREATE VIEW dbo.vProduct
```

Concepts

```
AS
SELECT ProductNumber, Name
FROM Product;
GO
SELECT *
FROM dbo.vProduct;
GO
```

Trong đoạn mã này, một khung nhìn được tạo ra trong khối lệnh. **CREATE VIEW** là câu lệnh duy nhất trong khối lệnh, những lệnh **GO** là rất cần thiết để tách câu lệnh **CREATE VIEW** ra khỏi các câu lệnh **SELECT** và **USE**. Đây là một ví dụ đơn giản để chứng minh việc sử dụng một khối lệnh. Trong thực tế, một số lượng lớn các câu lệnh có thể được sử dụng trong một khối lệnh đơn lẻ. Nó cũng có thể kết hợp hai hay nhiều khối lệnh trong một giao tác.

Code Snippet 2 trình bày ví dụ về điều này.

Code Snippet 2:

```
BEGIN TRANSACTION
GO
USE AdventureWorks2012;
GO
CREATE TABLE Company
(
Id_Num int IDENTITY(100, 5),
Company_Name nvarchar(100)
)
GO
INSERT Company (Company_Name)
 VALUES (N'A Bike Store')
INSERT Company (Company_Name)
 VALUES (N'Progressive Sports')
INSERT Company (Company_Name)
 VALUES (N'Modular Cycle Systems')
INSERT Company (Company_Name)
 VALUES (N'Advanced Bike Components')
```

Concepts

```
INSERT Company (Company_Name)
 VALUES (N'Metropolitan Sports Supply')
INSERT Company (Company_Name)
 VALUES (N'Aerobic Exercise Company')
INSERT Company (Company_Name)
 VALUES (N'Associated Bikes')
INSERT Company (Company_Name)
 VALUES (N'Exemplary Cycles')
GO
SELECT Id_Num, Company_Name
FROM dbo. Company
ORDER BY Company_Name ASC;
GO
COMMIT;
GO
```

Trong đoạn mã này, một số khối lệnh được kết hợp thành một giao tác. Các câu lệnh **BEGIN TRANSACTION** và **COMMIT** kèm theo các câu lệnh giao tác. Các câu lệnh **CREATE TABLE, BEGIN TRANSACTION, SELECT, COMMIT**, và **USE** là trong các khối lệnh một câu lệnh. Các câu lệnh **INSERT** tất cả được đưa vào trong một khối lệnh.

13.2.2 Các biến Transact-SQL

Biến cho phép người dùng lưu trữ dữ liệu để sử dụng như là đầu vào trong một câu lệnh Transact-SQL. Ví dụ, người dùng có thể tạo ra một truy vấn yêu cầu các loại giá trị dữ liệu khác nhau đã chỉ định trong mệnh đề WHERE mỗi lần truy vấn được thực thi. Ở đây, người dùng có thể sử dụng các biến trong mệnh đề WHERE, và viết logic để lưu trữ các biến với dữ liệu thích hợp.

SQL Server cung cấp các câu lệnh sau đây để đặt và khai báo các biến cục bộ.

- **DECLARE**

  Biến được khai báo với câu lệnh DECLARE trong phần thân của khối lệnh. Những biến này được gán các giá trị bằng cách sử dụng câu lệnh SELECT hoặc SET. Những biến này được khởi tạo với các giá trị NULL nếu người dùng đã không cung cấp một giá trị tại thời điểm khai báo..

  Sau đây là cú pháp cơ bản để khai báo một biến cục bộ.

  Cú pháp:

```
DECLARE { { @local_variable [AS] data_type } | [ = value ] }
```

Concepts

trong đó,

@local_variable : chỉ ra tên của các biến và bắt đầu với ký hiệu @.

data_type : chỉ ra kiểu dữ liệu. Biến không thể thuộc kiểu dữ liệu image, text, hoặc ntext.

=value : Gán một giá trị nội tuyến cho một biến. Giá trị có thể là một biểu thức hoặc một giá trị hằng số. Giá trị phải phù hợp với loại khai báo biến hay nó nên được mặc nhiên chuyển đổi sang loại đó.

Code Snippet 3 sử dụng một biến cục bộ để lấy thông tin liên hệ cho các tên gọi bắt đầu bằng Man.

Code Snippet 3:

```
USE AdventureWorks2012;
GO
DECLARE @find varchar(30) = 'Man%';
SELECT p.LastName, p.FirstName, ph.PhoneNumber
FROM Person.Person AS p
JOIN Person.PersonPhone AS ph ON p.BusinessEntityID = ph.BusinessEntityID
WHERE LastName LIKE @find;
```

Trong đoạn mã này, một biến cục bộ có tên là @find được sử dụng để lưu trữ các tiêu chí tìm kiếm, mà sau đó sẽ được sử dụng để lấy thông tin liên hệ. Ở đây, những tiêu chí bao gồm tất cả các tên gọi bắt đầu bằng Man. Hình 13.1 hiển thị đầu ra.

Results | Messages

	LastName	FirstName	PhoneNumber
1	Manchepalli	Ajay	1 (11) 500 555-0174
2	Manek	Parul	1 (11) 500 555-0146
3	Manzanares	Tomas	1 (11) 500 555-0178

Hình 13.1: Contact Information

- **SET**

  Câu lệnh SET thiết lập biến cục bộ được tạo ra bằng câu lệnh DECLARE với giá trị chỉ định.

Concepts

Sau đây là cú pháp cơ bản để đặt một biến cục bộ.

Cú pháp:

```
SET
{ @local_variable = { expression}
}
|
{ @local_variable
  {+= | -= | *= | /= | %= | &= | ^= | |= } expression
}
```

trong đó,

@local_variable : chỉ ra tên của biến và bắt đầu với ký hiệu @.

= : Gán giá trị ở bên phải cho biến ở phía bên trái.

{= | += | -= | *= | /= | %= | &= | ^= | |= } : chỉ ra các toán tử gán hỗn hợp. Đó là:

+= Cộng và sau đó, gán

-= Trừ và sau đó, gán

*= Nhân và sau đó, gán

/= Chia và sau đó, gán

%= Modulo và sau đó, gán

&= Bitwise AND và sau đó, gán

^= Bitwise XOR và sau đó, gán

|= Bitwise OR và sau đó, gán

biểu thức: chỉ ra bất kỳ biểu thức hợp lệ nào thậm chí có thể bao gồm một truy vấn con vô hướng.

Code Snippet 4 trình bày việc sử dụng SET để gán giá trị chuỗi cho một biến.

Code Snippet 4:

```
DECLARE @myvar char(20);
SET @myvar = 'This is a test';
```

Trong đoạn mã này, biến @myvar được gán một giá trị chuỗi.

- **SELECT**

Câu lệnh **SELECT** chỉ ra rằng biến cục bộ chỉ định đã được tạo ra sử dụng **DECLARE** nên được đặt thành biểu thức đã cho.

Concepts

Sau đây là cú pháp của câu lệnh SELECT.

Cú pháp:

```
SELECT { @local_variable { = | += | -= | *= | /= | %= | &= | ^= | |= } expression } [
,...n ] [ ; ]
```

trong đó,

@local_variable : chỉ ra các biến cục bộ để một giá trị sẽ được gán cho nó.

= : Gán giá trị ở bên phải cho biến ở phía bên trái.

{= | += | -= | *= | /= | %= | &= | ^= | |= }: chỉ ra các toán tử gán hỗn hợp.

biểu thức: chỉ ra bất kỳ biểu thức hợp lệ nào thậm chí có thể bao gồm một truy vấn con vô hướng.

Code Snippet 5 trình bày cách sử dụng SELECT để trả về một giá duy nhất.

Code Snippet 5:

```
USE AdventureWorks2012;
GO
DECLARE @var1 nvarchar(30);
SELECT @var1 = 'Unnamed Company';

SELECT @var1 = Name
FROM Sales.Store
WHERE BusinessEntityID = 10;

SELECT @var1 AS 'Company Name';
```

Trong đoạn mã này, biến @var1 được gán Công ty vô danh làm giá trị của nó.

Truy vấn đối với bảng Store sẽ trả lại không hàng là giá trị được chỉ ra cho BusinessEntityID không tồn tại trong bảng này. Sau đó biến sẽ giữ lại giá trị Công ty vô danh và sẽ được hiển thị với đầu đề Tên công ty. Hình 13.2 hiển thị đầu ra.

	Company Name
1	Unnamed Company

Hình 13.2: Generic Name

Mặc dù cả hai câu lệnh SET và SELECT trông tương tự nhau, nhưng không phải vậy. Dưới đây là một vài sự khác biệt giữa hai câu lệnh này:

- Có thể gán mỗi lần chỉ một biến sử dụng SET. Tuy nhiên, sử dụng SELECT bạn có thể làm nhiều phép gán cùng một lúc.
- SET có thể chỉ gán một giá trị vô hướng khi gán từ một truy vấn. Nó gây nên một lỗi và không làm việc nếu truy vấn trả về nhiều giá trị/hàng. Tuy nhiên, SELECT gán một trong những giá trị trả lại cho biến và người dùng thậm chí sẽ không biết rằng nhiều giá trị đã được trả lại.

Ghi chú - Để gán các biến, đề nghị sử dụng SET @local_variable thay vì SELECT @local_ variable.

13.3 Các từ đồng nghĩa

Từ đồng nghĩa là các đối tượng cơ sở dữ liệu phục vụ cho các mục đích sau:

- Chúng cung cấp một tên khác cho một đối tượng cơ sở dữ liệu khác, còn được gọi là đối tượng cơ bản, có thể tồn tại trên một máy chủ từ xa hoặc cục bộ.
- Chúng thể hiện một lớp trừu tượng bảo vệ ứng dụng máy khách từ các sửa đổi được thực hiện cho vị trí và tên của đối tượng cơ sở.

Ví dụ, hãy xem xét bảng Department của AdventureWorks2012 nằm trên máy chủ đầu tiên có tên là **Server1**. Để tham khảo bảng này từ máy chủ thứ hai **Server2**, ứng dụng máy chủ phải sử dụng tên bốn phần Server1.AdventureWorks2012.Person.Department. Nếu vị trí của bảng đã được sửa đổi, ví dụ, đến một máy chủ khác, ứng dụng máy khách sẽ phải được sửa chữa để phản ánh sự thay đổi đó. Để giải quyết cả hai vấn đề này, người dùng có thể tạo ra một từ đồng nghĩa **DeptEmpTable**, trên **Server2** cho bảng Department trên **Server1**. Bây giờ, ứng dụng máy khách chỉ phải sử dụng tên duy nhất **DeptEmpTable**, để tham khảo tới bảng Employee.

Tương tự như vậy, nếu vị trí của bảng Department thay đổi, người dùng phải sửa đổi từ đồng nghĩa **DeptEmpTable**, để trỏ đến vị trí mới của bảng Department. Do không có câu lệnh ALTER SYNONYM, đầu tiên bạn phải thả bỏ từ đồng nghĩa **DeptEmpTable**, và sau đó, tạo ra lại từ đồng nghĩa có cùng tên, nhưng trỏ từ đồng nghĩa tới vị trí mới của Department.

Ghi chú - Từ đồng nghĩa là một phần của lược đồ, và tương tự như các đối tượng lược đồ khác, tên đồng nghĩa phải là duy nhất.

Table 13.11 liệt kê các đối tượng cơ sở dữ liệu mà người dùng có thể tạo ra các từ đồng nghĩa.

Các đối tượng của cơ sở dữ liệu
Thủ tục lưu trữ mở rộng
Hàm có giá trị bảng SQL
Thủ tục lưu trữ SQL

Concepts

Các đối tượng của Cơ sở dữ liệu
Bảng (do người dùng định nghĩa)
Nhân bản-lọc-thủ tục
Hàm vô hướng SQL
Hàm có giá trị bảng trong dòng SQL
Dạng xem

Table 13.1:Database Objects

- **Từ đồng nghĩa và lược đồ**

  Giả sử người dùng muốn tạo ra một từ đồng nghĩa và có một lược đồ mặc định không thuộc sở hữu của họ. Trong trường hợp này, họ có thể hội đủ điều kiện tên đồng nghĩa với tên lược đồ mà họ thực sự sở hữu. Xem xét ví dụ mà người dùng sở hữu một lược đồ **Resources**, nhưng **Materials** là lược đồ mặc định của người dùng. Nếu người dùng này muốn tạo ra một từ đồng nghĩa, họ phải thêm tiền tố cho tên của từ đồng nghĩa với lược đồ **Resources**.

- **Cấp quyền trên các từ đồng nghĩa**

  Chỉ các thành viên của vai trò db_owner hoặc db_ddladmin hoặc chủ sở hữu đồng nghĩa được phép cấp quyền trên một từ đồng nghĩa. Người dùng có thể từ chối, cấp, hoặc thu hồi tất cả hoặc bất kỳ quyền nào trên một từ đồng nghĩa. Bảng 13.2 hiển thị danh sách các quyền được áp dụng trên một từ đồng nghĩa.

Permissions
DELETE
INSERT
TAKE OWNERSHIP
VIEW DEFINITION
CONTROL
EXECUTE
SELECT
UPDATE

Table 13.2: Permissions

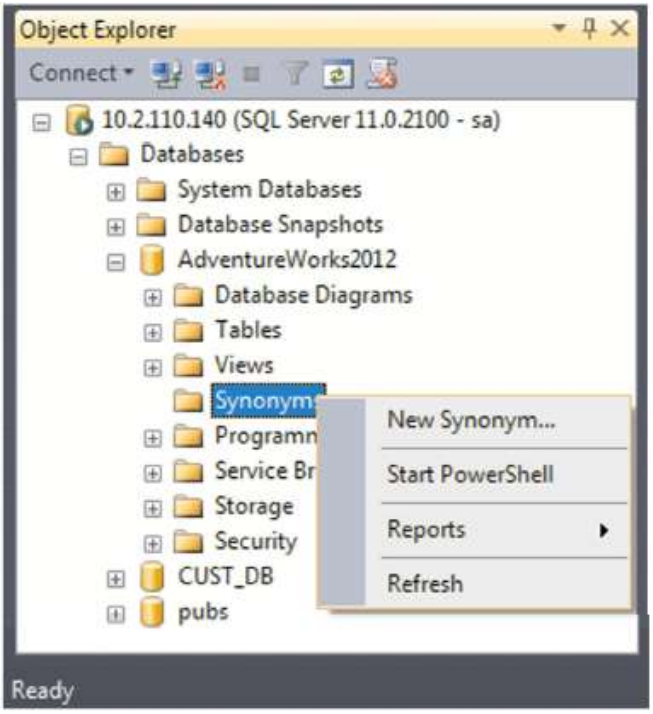
Concepts

- Làm việc với các từ đồng nghĩa

  Người dùng có thể làm việc với các từ đồng nghĩa trong SQL Server 2012 sử dụng Transact-SQL hoặc SSMS. Để tạo ra một từ đồng nghĩa sử dụng SSMS, thực hiện các bước sau:

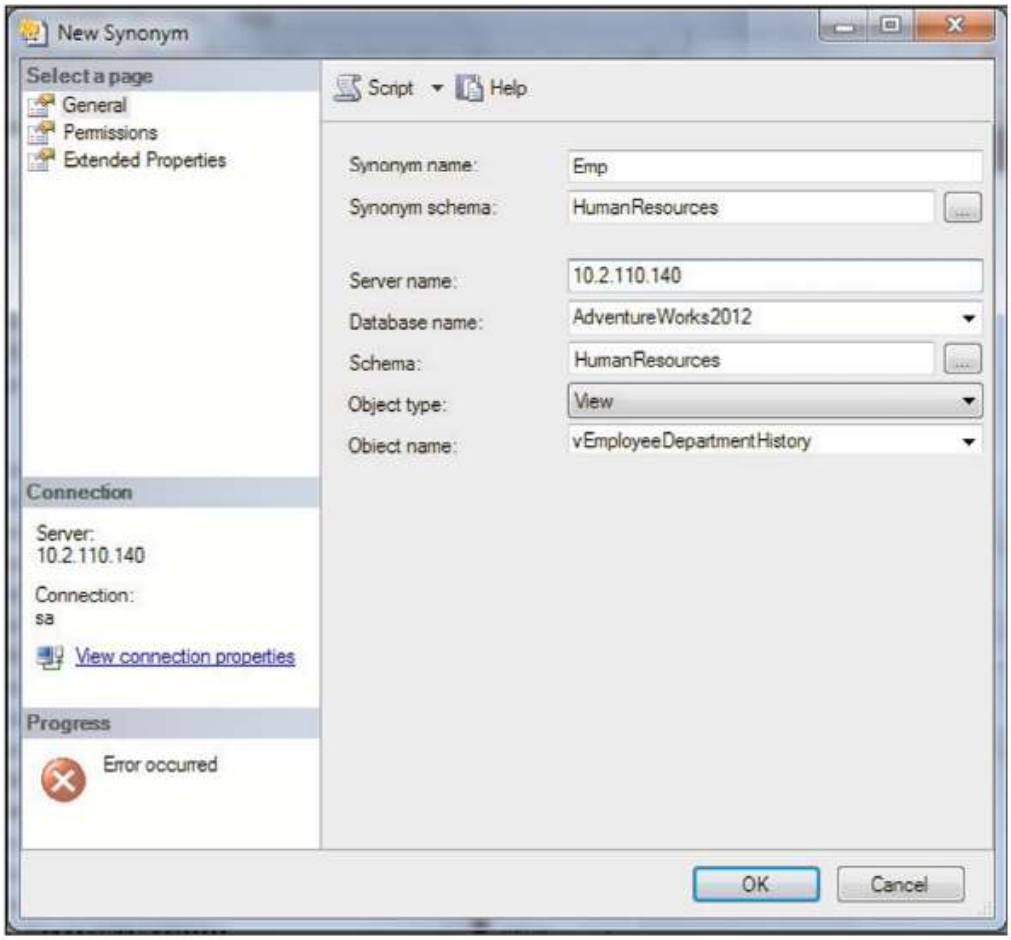
  1. Trong **Object Explorer**, mở rộng cơ sở dữ liệu nơi bạn muốn tạo ra một từ đồng nghĩa mới.

2. Chọn thư mục **Synonyms**, kích chuột phải vào nó và sau đó, bấm vào **New Synonym**… như được trình bày trong hình 13.3.



Hình 13.3: Creating a New Synonym

3. Trong hộp thoại **New Synonym**, cung cấp thông tin như được trình bày trong hình 13.4.



Hình 13.4: Adding Details in the New Synonym Dialog Box

trong đó,

Synonym name: là tên mới cho đối tượng. Ở đây, Emp là tên.

Synonym schema: là tên mới cho đối tượng lược đồ. Ở đây, cùng một tên lược đồ HumanResources được sử dụng cho từ đồng nghĩa và loại đối tượng.

Server name: là tên của máy chủ sẽ được kết nối. Ở đây, tên máy chủ được chỉ ra là 10.2.110.140.

Database name: là tên cơ sở dữ liệu để kết nối đối tượng. Ở đây, AdventureWorks2012 là tên cơ sở dữ liệu.

Schema: là lược đồ sở hữu đối tượng.

Concepts

Object type và **Object name**: tương ứng là loại đối tượng và tên. Ở đây, loại đối tượng được lựa chọn là view và tên đối tượng tham khảo từ đồng nghĩa là vEmployeeDepartmentHistory.

Để tạo ra một từ đồng nghĩa sử dụng Transact-SQL, thực hiện các bước sau:

1. Kết nối với Database Engine.
2. Bấm vào **New Query** trong thanh tiêu chuẩn.
3. Viết truy vấn để tạo ra từ đồng nghĩa trong cửa sổ truy vấn.

Sau đây là cú pháp để tạo ra từ đồng nghĩa.

Cú pháp:

```
CREATE SYNONYM [ schema_name_1. ] synonym_name FOR <object>


<object> ::=
{
    [ server_name.[ database_name ] . [ schema_name_2 ].| database_name . [
schema_name_2 ].| schema_name_2. ] object_name
}
```

trong đó,

schema_name_1: chỉ ra rằng lược đồ trong đó đồng nghĩa được tạo ra. synonym_name: chỉ ra tên từ đồng nghĩa mới.

server_name: chỉ ra tên máy chủ, nơi đối tượng cơ sở được đặt.

database_name: chỉ ra tên cơ sở dữ liệu nơi đối tượng cơ sở được đặt. schema_name_2: chỉ ra tên giản đồ của đối tượng cơ sở.

object_name: chỉ ra tên đối tượng cơ sở, được tham chiếu bởi từ đồng nghĩa.

Code Snippet 6 tạo ra một từ đồng nghĩa từ bảng hiện có.

Code Snippet 6:

```
USE tempdb;
GO
CREATE SYNONYM MyAddressType
FOR AdventureWorks2012.Person.AddressType;
GO
```

Trong đoạn mã này, một từ đồng nghĩa được tạo ra từ một bảng hiện có hiện diện trong cơ sở dữ liệu AdventureWorks2012.

Concepts

4. Bấm vào **Execute** trên thanh công cụ để hoàn thành việc tạo ra từ đồng nghĩa.

13.4 Các câu lệnh luồng chương trình

Có nhiều loại chương trình khác nhau của các hàm và câu lệnh luồng chương trình được hỗ trợ bởi Transact-SQL. Một số trong số này như sau:

- **Ngôn ngữ Điều khiển luồng Transact-SQL**

  Ngôn ngữ Điều khiển luồng xác định luồng thực thi các câu lệnh Transact-SQL, khối câu lệnh, hàm do người dùng định nghĩa, và các thủ tục lưu trữ.

  Theo mặc định, các câu lệnh Transact-SQL được thực hiện tuần tự, theo thứ tự chúng xảy ra. Các phần tử ngôn ngữ điều khiển luồng cho phép các câu lệnh được thực hiện theo một thứ tự cụ thể, có liên quan đến nhau, và làm cho phụ thuộc lẫn nhau sử dụng các cấu trúc tương tự như các ngôn ngữ lập trình.

  Table 13.3 lists some of the Transact-SQL control-of-flow language keywords.

Control-Of-Flow Language Keywords
RETURN
THROW
TRY....CATCH
WAITFOR
WHILE
BEGIN....END
BREAK
CONTINUE
GOTO label
IF...ELSE

Table 13.3: Keywords

- **BEGIN....END**

  Những câu lệnh BEGIN…END bao quanh một loạt các câu lệnh Transact-SQL để nhóm các câu lệnh Transact-SQL được thực thi.
  Sau đây là cú pháp cơ bản cho câu lệnh BEGIN và END.

  Cú pháp:

```
BEGIN
     {
    sql_statement | statement_block
     }
END
```

Concepts

trong đó,

{ sql_statement| statement_block }: Là bất kỳ câu lệnh Transact-SQL hợp lệ nào được định nghĩa sử dụng một khối câu lệnh.

Code Snippet 7 trình bày việc sử dụng các câu lệnh BEGIN và END.

Code Snippet 7:

```
USE AdventureWorks2012;
GO
BEGIN TRANSACTION;
GO
IF @@TRANCOUNT = 0
BEGIN
 SELECT FirstName, MiddleName
 FROM Person.Person WHERE LastName = 'Andy';
 ROLLBACK TRANSACTION;
 PRINT N'Rolling back the transaction two times would cause an error.';
END;
ROLLBACK TRANSACTION;
PRINT N'Rolled back the transaction.';
GO
```

Trong đoạn mã này, các câu lệnh BEGIN và END mô tả một chuỗi các câu lệnh Transact-SQL được thực thi với nhau. Giả sử BEGIN và END không được đưa vào, khi đó, các câu lệnh ROLLBACK TRANSACTION sẽ thực thi và cả hai thông báo PRINT sẽ được hiển thị.

- **IF...ELSE**

Câu lệnh **IF…ELSE** bắt buộc một điều kiện về việc thực thi một câu lệnh Transact-SQL. Câu lệnh Transact-SQL được tuân thủ với từ khóa IF và điều kiện này thực hiện chỉ khi điều kiện được thỏa mãn và trả về TRUE. Từ khóa ELSE là một câu lệnh Transact-SQL tùy chọn thực hiện chỉ khi điều kiện IF không được thỏa mãn và trả về FALSE.

Sau đây là cú pháp cho câu lệnh **IF…ELSE**.

Cú pháp :

```
IF Boolean_expression
     { sql_statement | statement_block }
[ ELSE
     { sql_statement | statement_block } ]
```

Concepts

trong đó,

Boolean_expression: chỉ ra biểu thức trả về giá trị TRUE hoặc FALSE.

{ sql_statement| statement_block }: Là bất kỳ câu lệnh hoặc gộp nhóm câu lệnh Transact-SQL nào được định nghĩa sử dụng khối câu lệnh. Nếu khối câu lệnh không được sử dụng, điều kiện IF hoặc ELSE có thể ảnh hưởng đến hiệu suất của chỉ có một câu lệnh Transact-SQL. Để định nghĩa khối câu lệnh, từ khóa BEGIN và END được sử dụng.

Code Snippet 8 trình bày việc sử dụng các câu lệnh IF…ELSE.

Code Snippet 8:

```
USE AdventureWorks2012
GO
DECLARE @ListPrice money;
SET @ListPrice = (SELECT MAX(p.ListPrice)
   FROM Production.Product AS p
   JOIN Production.ProductSubcategory AS s
   ON p.ProductSubcategoryID = s.ProductSubcategoryID
   WHERE s.[Name] = 'Mountain Bikes');
         PRINT @ListPrice
IF @ListPrice < 3000
PRINT 'All the products in this category can be purchased for an amount less
than 3000'
ELSE
PRINT 'The prices for some products in this category exceed 3000'
```

Trong đoạn mã này, câu lệnh IF…ELSE được sử dụng để tạo thành một câu lệnh điều kiện. Đầu tiên, biến @ListPrice được định nghĩa và truy vấn được tạo ra để trả về giá niêm yết tối đa của loại sản phẩm Xe đạp leo núi. Sau đó, mức giá này được so sánh với giá trị là 3000 để xác định xem sản phẩm có thể được mua cho một số tiền ít hơn 3000 hay không. Nếu có, một thông báo thích hợp được in sử dụng câu lệnh PRINT đầu tiên. Nếu không, khi đó câu lệnh PRINT thứ hai thực hiện.

- **WHILE**

Câu lệnh WHILE chỉ rõ một điều kiện cho thực thi lặp lại của khối câu lệnh. Các câu lệnh được thực thi lặp lại miễn là điều kiện đã chỉ định là đúng. Việc thực thi các câu lệnh trong vòng lặp WHILE có thể được kiểm soát bằng cách sử dụng các từ khóa **BREAK** và **CONTINUE**.

Concepts

Sau đây là cú pháp cho câu lệnh WHILE.

Cú pháp:

```
WHILE Boolean_expression
      { sql_statement | statement_block | BREAK | CONTINUE }
```

trong đó,

Boolean_expression: chỉ ra biểu thức trả về giá trị TRUE hoặc FALSE.

{sql_statement | statement_block}: Là bất kỳ câu lệnh Transact-SQL nào định nghĩa khối câu lệnh.

BREAK: Kết quả trong một đầu ra từ vòng lặp WHILE phía trong nhất. Mỗi câu lệnh xuất hiện sau từ khóa END, đánh dấu sự kết thúc vòng lặp, được thực thi.

CONTINUE: Dẫn đến vòng lặp WHILE được khởi động lại. Các câu lệnh sau từ khóa CONTINUE trong phần thân của vòng lặp không được thực thi.

Code Snippet 9 trình bày việc sử dụng câu lệnh WHILE.

Code Snippet 9:

```
DECLARE @flag int
SET @flag = 10
WHILE (@flag <=95)
 BEGIN
  IF @flag%2 =0
   PRINT @flag
   SET @flag = @flag + 1
   CONTINUE;
 END
GO
```

Sử dụng đoạn mã này, tất cả các số chẵn bắt đầu từ 10 đến 95 được hiển thị. Điều này đạt được bằng cách sử dụng vòng lặp WHILE cùng với câu lệnh IF.

Tương tự như vậy, vòng lặp IF cũng có thể được sử dụng với các truy vấn và câu lệnh Transact-SQL khác.

Concepts

13.5 Các hàm Transact-SQL

Các hàm Transact-SQL thường được sử dụng như sau:

- **Các hàm xác định và không xác định**

  Các hàm do người dùng định nghĩa có các thuộc tính định nghĩa khả năng của Công cụ cơ sở dữ liệu SQL Server. Công cụ sở dữ liệu được sử dụng để tạo chỉ mục kết quả của hàm thông qua các cột tính để hàm này gọi hoặc các dạng xem có chỉ mục tham khảo những hàm này. Một thuộc tính như vậy là tính xác định của một hàm.

  Hàm xác định trả về cùng một kết quả mỗi khi chúng được gọi với một tập hợp xác định các giá trị đầu vào và chỉ ra cùng một trạng thái của cơ sở dữ liệu. Các hàm không xác định trả về các kết quả khác nhau mỗi lần chúng được gọi với tập hợp đã chỉ định các giá trị đầu vào mặc dù cơ sở dữ liệu được truy cập vẫn giữ nguyên.

  Ví dụ, nếu một người dùng gọi hàm **DAY()** trên một cột cụ thể, nó luôn luôn trả về ngày dạng số cho tham số ngày được truyền vào. Tuy nhiên, nếu người dùng gọi hàm **DATENAME(),** kết quả không thể dự đoán được vì nó có thể khác nhau mỗi lần, tùy thuộc vào phần nào của ngày được truyền làm đầu vào. Như vậy, ở đây, **DAY()** là một hàm xác định, trong khi **DATENAME()** là một hàm không xác định. Tương tự như vậy, **RAND** (không có bất kỳ giá trị mầm nào), **@@TIMETICKS**, **@@CONNECTIONS**, và **GETDATE()** là không xác định.

  Người dùng không thể ảnh hưởng đến tính xác định của các hàm dựng sẵn. Mỗi hàm dựng sẵn là xác định hoặc không xác định tùy thuộc vào cách hàm đó được thực hiện bằng SQL Server.

  Table 13.4 liệt kê một số các hàm dựng sẵn xác định và không xác định.

Các hàm dựng sẵn xác định	Các hàm dựng sẵn không xác định
POWER	@@TOTAL_WRITE
ROUND	CURRENT_TIMESTAMP
RADIANS	GETDATE
EXP	GETUTCDATE
FLOOR	GET_TRANSMISSION_STATUS
SQUARE	NEWID
SQRT	NEWSEQUENTIALID
LOG	@@CONNECTIONS
YEAR	@@CPU_BUSY
ABS	@@DBTS
ASIN	@@IDLE
ACOS	@@IOBUSY
SIGN	@@PACK_RECEIVED

Concepts

Deterministic Built-in Functions	Non-Deterministic Built-in Functions
SIN	@@PACK_SENT

Table 13.4: Các hàm dựng sẵn xác định và không xác định

Ngoài ra còn có một số hàm không phải là luôn luôn xác định nhưng bạn có thể sử dụng chúng trong các dạng xem có chỉ mục nếu chúng được đưa ra một cách xác định. Bảng 13.5 liệt kê một số những hàm này.

Hàm	Mô tả
CONVERT	Là xác định chỉ khi một trong những điều kiện này tồn tại: ➢ Có một loại nguồn sql_variant . ➢ Có một loại đích sql_variant và loại nguồn là không xác định. ➢ Có loại nguồn hoặc đích của nó là smal datetime hoặc datetime, có loại nguồn hoặc đích là một chuỗi ký tự, và có một kiểu không xác định đã chỉ ra. Tham số kiểu phải là một hằng số sẽ là xác định.
CAST	Là xác định chỉ khi nó được sử dụng với smalldatetime, sql_variant, hoặc datetime.
ISDATE	Là xác định trừ khi được sử dụng với hàm CONVERT, tham số kiểu CONVERT được chỉ ra, và kiểu không bằng 0, 100, 9, hoặc 109.
CHECKSUM	Là xác định, với ngoại lệ của CHECKSUM(*).

Table 13.5: Deterministic Functions

- **Gọi thủ tục lưu trữ mở rộng từ các hàm**

Hàm gọi các thủ tục lưu trữ mở rộng là không xác định vì các thủ tục lưu trữ mở rộng có thể dẫn đến các tác dụng phụ lên cơ sở dữ liệu. Các thay đổi đối với trạng thái toàn cầu của một cơ sở dữ liệu như là một sự thay đổi đến một nguồn lực bên ngoài, hoặc các cập nhật cho bảng, tập tin, hoặc mạng được gọi là tác dụng phụ. Ví dụ, gửi e-mail, hoặc xóa một tập tin có thể gây ra tác dụng phụ. Trong khi thực thi một thủ tục lưu trữ mở rộng từ một hàm do người dùng định nghĩa, người dùng không thể đảm bảo rằng nó sẽ trả về một tập kết quả phù hợp.

Concepts

Do đó, các hàm do người dùng định nghĩa tạo ra các tác dụng phụ trên cơ sở dữ liệu không được khuyến khích.

- **Các hàm có giá trị vô hướng**

  Hàm có giá trị vô hướng (SVF) luôn luôn trả về một giá trị int, bit, hoặc string. Kiểu dữ liệu được trả về từ và các tham số đầu vào của SVF có thể là bất kì kiểu dữ liệu nào ngoại trừ text, ntext, image, cursor, và timestamp.

  Hàm vô hướng trong dòng có một câu lệnh duy nhất và không có phần thân hàm. Hàm vô hướng đa câu lệnh bao quanh phần thân hàm trong khối BEGIN...END.

- **Các hàm có giá trị bảng**

  Hàm có giá trị bảng là hàm do người dùng định nghĩa trả về một bảng. Tương tự như một hàm vô hướng trong dòng, hàm có giá trị bảng trong dòng có một câu lệnh duy nhất và không có phần thân hàm.

  Code Snippet 10 trình bày việc sử dụng hàm có giá trị bảng.

  Code Snippet 10:

```
USE AdventureWorks2012;
GO
IF OBJECT_ID (N'Sales.ufn_CustDates', N'IF') IS NOT NULL
 DROP FUNCTION Sales.ufn_ufn_CustDates;
GO
CREATE FUNCTION Sales.ufn_CustDates ()
RETURNS TABLE
AS
RETURN
(
 SELECT A.CustomerID, B.DueDate, B.ShipDate
FROM Sales.Customer A
LEFT OUTER JOIN
Sales.SalesOrderHeader B
ON
A.CustomerID = B.CustomerID AND YEAR(B.DueDate)<2012
);
```

Ở đây, hàm có giá trị bảng trong dòng định nghĩa phép nối bên ngoài bên trái giữa các bảng Sales.Customer và Sales.SalesOrderHeader.

Concepts

Những bảng này được kết nối dựa trên các id khách hàng. Trong trường hợp này, tất cả các bản ghi từ bảng bên trái và chỉ các bản ghi so khớp từ bảng bên phải được trả về. Sau đó bảng kết quả được trả về từ hàm có giá trị bảng.

Hàm này được gọi ra như được trình bày trong Code Snippet 11.

Code Snippet 11:

```
SELECT * FROM Sales.ufn_CustDates();
```

Kết quả sẽ là đầu ra của phép nối được trình bày theo một định dạng bảng.

13.6 Thay đổi các hàm do người dùng định nghĩa

Người dùng có thể sửa đổi các hàm do người dùng định nghĩa trong SQL Server 2012 bằng cách sử dụng Transact-SQL hoặc SSMS. Việc thay đổi các hàm do người dùng định nghĩa không sửa đổi các cấp phép của các hàm, nó cũng sẽ không ảnh hưởng đến bất kỳ thủ tục lưu trữ, trigger, hoặc hàm.

- **Giới hạn và hạn chế**

  ALTER FUNCTION không cho phép người dùng thực hiện những hành động sau:

  - Sửa đổi hàm có giá trị vô hướng thành một hàm có giá trị bảng.
  - Sửa đổi hàm trong dòng thành một hàm đa câu lệnh.
  - Sửa đổi hàm Transact-SQL thành hàm CLR.

- **Các quyền**

  Cho phép ALTER là bắt buộc trên lược đồ hoặc hàm. Nếu hàm chỉ ra loại do người dùng định nghĩa, khi đó cần phải có cho phép EXECUTE trên loại đó.

- **Sửa đổi một hàm do người dùng định nghĩa sử dụng SSMS**

  Người dùng cũng có thể thay đổi các hàm do người dùng định nghĩa sử dụng SSMS.

  Để sửa đổi hàm do người dùng định nghĩa sử dụng SSMS, thực hiện các bước sau:

  1. Bấm vào dấu cộng (+) bên cạnh cơ sở dữ liệu có chứa hàm sẽ được sửa đổi.
  2. Bấm vào dấu cộng (+) bên cạnh thư mục Programmability.
  3. Bấm vào dấu cộng (+) bên cạnh thư mục có chứa hàm sẽ được sửa đổi. Có ba loại thư mục như sau:
     - Các hàm có giá trị bảng
     - Các hàm có giá trị vô hướng
     - Các hàm tổng hợp
     - Các hàm tổng hợp

4. Bấm chuột phải vào hàm sẽ được sửa đổi và sau đó, chọn **Modify**. Đoạn mã cho hàm này xuất hiện trong cửa sổ trình soạn thảo truy vấn.
5. Trong cửa sổ trình soạn thảo truy vấn, thực hiện các thay đổi cần thiết cho phần thân câu lệnh ALTER FUNCTION.
6. Bấm vào **Execute** trên thanh công cụ để thực hiện câu lệnh ALTER FUNCTION.

➢ **Sửa đổi hàm do người dùng định nghĩa sử dụng Transact-SQL**

Để sửa đổi hàm do người dùng định nghĩa sử dụng Transact-SQL, thực hiện các bước sau:

- Trong **Object Explorer**, kết nối với thể hiện Công cụ cơ sở dữ liệu.
- Trên **thanh Standard**, bấm vào **New Query.**
- Gõ mã ALTER FUNCTION vào **Query Editor.**
- Bấm vào **Execute** trên thanh công cụ để thực hiện câu lệnh ALTER FUNCTION.

Code Snippet 12 trình bày việc sửa đổi một hàm có giá trị bảng.

Code Snippet 12:

```
USE [AdventureWorks2012]
GO
ALTER FUNCTION [dbo].[ufnGetAccountingEndDate]()
RETURNS [datetime]
AS
BEGIN
 RETURN DATEADD(millisecond, -2, CONVERT(datetime, '20040701', 112));
END;
```

13.7 Sự tạo ra các cửa sổ với OVER

Hàm cửa sổ là một hàm áp dụng cho một bộ sưu tập các hàng. Từ 'cửa sổ' được sử dụng để tham khảo tới bộ sưu tập các hàng nơi hàm hoạt động trên đó.

Trong Transact-SQL, mệnh đề OVER được sử dụng để định nghĩa một cửa sổ trong một tập kết quả truy vấn. Việc sử dụng các cửa sổ và mệnh đề OVER với các hàm cung cấp một số lợi thế. Ví dụ, chúng giúp tính toán các giá trị tổng hợp. Chúng còn cho phép số dòng trong một tập kết quả được tạo ra dễ dàng.

Concepts

13.7.1 Tạo cửa sổ các thành phần

Ba thành phần cốt lõi của việc tạo ra các cửa sổ với mệnh đề OVER như sau:

- **Phân vùng**

  Phân vùng là một tính năng giới hạn cửa sổ của tính toán gần đây để chỉ những hàng từ tập kết quả chứa cùng một giá trị trong các cột phân vùng như ở hàng hiện có. Nó sử dụng mệnh đề **PARTITION BY**.

  Code Snippet 13 trình bày việc sử dụng mệnh đề **PARTITION BY** và **OVER** với các hàm tổng hợp. Ở đây, việc sử dụng mệnh đề **OVER** chứng minh là có hiệu quả tốt hơn là sử dụng các truy vấn phụ để tính toán các giá trị tổng hợp.

  Code Snippet 13:

```
USE AdventureWorks2012;
GO
SELECT SalesOrderID, ProductID, OrderQty
  ,SUM(OrderQty) OVER (PARTITION BY SalesOrderID) AS Total
  ,MAX(OrderQty) OVER (PARTITION BY SalesOrderID) AS MaxOrderQty
FROM Sales.SalesOrderDetail
WHERE ProductId IN (776, 773);
GO
```

Kết quả của mã này được trình bày trong hình 13.5.

	SalesOrderID	ProductID	OrderQty	Total	MaxOrderQty
1	43659	776	1	3	2
2	43659	773	2	3	2
3	43661	776	4	6	4
4	43661	773	2	6	4
5	43664	773	1	1	1
6	43665	773	1	2	1
7	43665	776	1	2	1
8	43667	773	1	1	1
9	43670	773	2	3	2
10	43670	776	1	3	2
11	43672	776	2	2	2

Hình 13.5: Partitioning

- **Sắp thứ tự**

Phần tử sắp thứ tự định nghĩa sắp thứ tự cho tính toán trong phân vùng đó. Trong một phần tử sắp thứ tự SQL tiêu chuẩn tất cả các hàm được hỗ trợ. Trước đây, SQL Server không hỗ trợ cho những phần tử sắp thứ tự với các hàm tổng hợp vì nó chỉ hỗ trợ phân vùng. Trong SQL Server 2012, có sự hỗ trợ cho các phần tử sắp thứ tự với các hàm tổng hợp. Phần tử sắp thứ tự có ý nghĩa khác nhau ở một số mức độ cho các thể loại hàm khác nhau. Với các hàm xếp hạng, sắp thứ tự là tự phát.

Code Snippet 14 trình bày ví dụ về phần tử sắp thứ tự.

Code Snippet 14:

```
SELECT CustomerID, StoreID,
RANK() OVER(ORDER BY StoreID DESC) AS Rnk_All,
RANK() OVER(PARTITION BY PersonID
ORDER BY CustomerID DESC) AS Rnk_Cust
FROM Sales.Customer;
```

Đoạn mã này sử dụng hàm RANK() trả về thứ hạng của mỗi hàng trong phân vùng của một tập kết quả. Thứ hạng của một hàng được xác định bằng cách thêm 1 vào số các thứ hạng đến trước hàng được chỉ định. Ví dụ, trong khi sử dụng sắp thứ tự giảm dần, hàm RANK() trả về một cái nhiều hơn số lượng hàng trong phân vùng tương ứng có một giá trị sắp thứ tự lớn hơn cái đã chỉ định.

Hình 13.6 hiển thị kết quả của Code Snippet 14.

	CustomerID	StoreID	Rnk_All	Rnk_Cust
1	701	844	813	1
2	700	1030	633	2
3	699	842	815	3
4	698	640	1009	4
5	697	1032	631	5
6	696	840	817	6
7	695	638	1011	7
8	694	1034	629	8
9	693	838	819	9
10	692	802	855	10
11	691	1036	627	11

Hình 13.6: Ordering

Code Snippet 15 hiển thị một truy vấn với hai tính toán **RANK** và mệnh đề **ORDER BY**.

Code Snippet 15:

```
SELECT TerritoryID, Name, SalesYTD,
```

Concepts

```
RANK() OVER(ORDER BY SalesYTD DESC) AS Rnk_One,
RANK() OVER(PARTITION BY TerritoryID
ORDER BY SalesYTD DESC) AS Rnk_Two
FROM Sales.SalesTerritory;
```

Đoạn mã này sử dụng hàm RANK() trả về thứ hạng của mỗi hàng trong phân vùng của một tập kết quả. Nói chung, thứ hạng của một hàng được xác định bằng cách thêm 1 vào số lượng thứ hạng đến trước hàng đã chỉ định. Ở đây trong đoạn mã này, hàm RANK() đầu tiên tạo ra thuộc tính Rnk_One phụ thuộc vào phân vùng mặc định, và hàm RANK thứ hai tạo ra Rnk_Two sử dụng phân vùng rõ ràng bằng TerritoryID.

Hình 13.7 hiển thị các phân vùng được định nghĩa cho một mẫu ba kết quả tính toán trong truy vấn: một giá trị Rnk_One và hai giá trị Rnk_Two.

	TerritoryID	Name	SalesYTD	Rnk_One	Rnk_Two
1	1	Northwest	7887186.7882	2	1
2	2	Northeast	2402176.8476	10	1
3	3	Central	3072175.118	8	1
4	4	Southwest	10510853.8739	1	1
5	5	Southeast	2538667.2515	9	1
6	6	Canada	6771829.1376	3	1
7	7	France	4772398.3078	6	1
8	8	Germany	3805202.3478	7	1
9	9	Australia	5977814.9154	4	1
10	10	United Kingdom	5012905.3656	5	1

Hình 13.7: Partitioning and Ranking

- **Tạo khung**

Tạo khung là một tính năng cho phép bạn chỉ ra việc chia nhỏ thêm các hàng trong một phân vùng cửa sổ. Điều này được thực hiện bằng cách chỉ ra ranh giới trên và dưới cho khung cửa sổ thể hiện các hàng cho hàm cửa sổ. Nói một cách đơn giản, khung tương tự như di chuyển cửa sổ trên dữ liệu bắt đầu và kết thúc tại các vị trí quy định. Khung cửa sổ có thể được định nghĩa bằng cách sử dụng các mệnh đề phụ ROW hoặc RANGE và cung cấp ranh giới bắt đầu và kết thúc.

Code Snippet 16 hiển thị một truy vấn đối với ProductInventory, tính toán tổng lượng chạy cho mỗi sản phẩm và vị trí.

Code Snippet 16:

```
SELECT ProductID, Shelf, Quantity,
SUM(Quantity) OVER(PARTITION BY ProductID
```

Concepts

```
ORDER BY LocationID
ROWS BETWEEN UNBOUNDED PRECEDING
AND CURRENT ROW) AS RunQty
FROM Production.ProductInventory;
```

Trong đoạn mã này, hàm cửa sổ áp dụng tổng hợp SUM cho thuộc tính Quantity, phân vùng cửa sổ theo ProductID, sắp thứ tự các hàng phân vùng theo LocationID, và tạo khung các hàng phân vùng phụ thuộc vào thứ tự đã cho giữa cách quãng không bị chặn (không có điểm ranh giới dưới) và hàng hiện tại. Nói cách khác, kết quả sẽ là tổng của tất cả các hàng trước trong khung, bao gồm cả dòng hiện tại. Hình 13.8 hiển thị kết quả của Code Snippet 16.

	ProductID	Shelf	Quantity	RunQty
1	1	A	408	408
2	1	B	324	732
3	1	A	353	1085
4	2	A	427	427
5	2	B	318	745
6	2	A	364	1109
7	3	A	585	585
8	3	B	443	1028
9	3	A	324	1352
10	4	A	512	512
11	4	B	422	934

Hình 13.8: **Framing**

13.8 Các hàm Windows

Một số loại hàm cửa sổ khác nhau như sau:

- **Các hàm xếp hạng**

  Những hàm này trả về giá trị thứ hạng cho mỗi hàng trong một phân vùng. Dựa trên hàm được sử dụng, nhiều hàng sẽ trả về cùng một giá trị như những hàng khác. Các hàm xếp hạng là không xác định.

  Table 13.6 liệt kê các hàm xếp hạng khác nhau.

Các hàm xếp hạng	Mô tả
NTILE	Trải các hàng trong một phân vùng có thứ tự vào một số nhóm đã cho, bắt đầu từ 1. Đối với mỗi hàng, hàm trả về số nhóm có hàng đó.

Concepts

Các hàm xếp hạng	Mô tả
ROW NUMBER	Lấy số thứ tự của một hàng trong một phân vùng của một tập kết quả, bắt đầu từ 1 cho hàng đầu tiên trong mỗi phân vùng.
DENSE RANK	Trả về thứ hạng của các hàng trong phân vùng của một tập kết quả, mà không có bất kỳ khoảng trống trong xếp hạng. Thứ hạng của một hàng là một điểm cộng số lượng thứ hạng riêng biệt đến trước hàng được nói đến.

Table 13.6: Ranking Functions

Code Snippet 17 trình bày việc sử dụng các hàm xếp hạng.

Code Snippet 17:

```
USE AdventureWorks2012;
GO
SELECT p.FirstName, p.LastName
  ,ROW_NUMBER() OVER (ORDER BY a.PostalCode) AS 'Row Number'
  ,NTILE(4) OVER (ORDER BY a.PostalCode) AS 'NTILE'
  ,s.SalesYTD, a.PostalCode
FROM Sales.SalesPerson AS s
  INNER JOIN Person.Person AS p
    ON s.BusinessEntityID = p.BusinessEntityID
  INNER JOIN Person.Address AS a
    ON a.AddressID = p.BusinessEntityID
WHERE TerritoryID IS NOT NULL
  AND SalesYTD <> 0;
```

Hàm NTILE() phá vỡ một bộ sưu tập đầu vào đã cho vào N nhóm logic có kích thước bằng nhau. Để xác định bao nhiêu hàng thuộc vào mỗi nhóm, SQL Server phải xác định tổng số hàng trong bộ sưu tập đầu vào. Mệnh đề OVER quyết định thứ tự của các hàng khi họ đã được chia thành các nhóm. Có thể thực hiện gộp nhóm theo một thứ tự và trả về tập kết quả theo thứ tự khác.

Hình 13.9 hiển thị kết quả của Code Snippet 17.

	FirstName	LastName	Row Number	NTILE	SalesYTD	PostalCode
1	Michael	Blythe	1	1	3763178.1787	98027
2	Linda	Mitchell	2	1	4251368.5497	98027
3	Jillian	Carson	3	1	3189418.3662	98027
4	Garrett	Vargas	4	1	1453719.4653	98027
5	Tsvi	Reiter	5	2	2315185.611	98027
6	Pamela	Ansman-Wolfe	6	2	1352577.1325	98027
7	Shu	Ito	7	2	2458535.6169	98055
8	José	Saraiva	8	2	2604540.7172	98055
9	David	Campbell	9	3	1573012.9383	98055
10	Tete	Mensa-Annan	10	3	1576562.1966	98055
11	Lynn	Tsoflias	11	3	1421810.9242	98055

Hình 13.9: Ranking Functions

➢ Các hàm OFFSET

Những loại hàm bù trừ khác nhau như sau:

➢ SWITCHOFFSET

Hàm này trả về giá trị DATETIMEOFFSET được sửa đổi từ bù trừ múi giờ đã lưu trữ cho bù trừ múi giờ mới cụ thể.

Sau đây là cú pháp cho hàm SWITCHOFFSET.

Cú pháp:

```
SWITCHOFFSET ( DATETIMEOFFSET, time_zone )
```

trong đó,

DATETIMEOFFSET: là một biểu thức được giải thành giá trị datetimeoffset(n).

time_zone: chỉ ra chuỗi ký tự ở định dạng [+|-]TZH:TZM hoặc một số nguyên có dấu (phút) trình bày độ lệch múi giờ, và được giả định là nhận thức và điều chỉnh giờ làm việc theo mùa hạ.

Code Snippet 18 trình bày việc sử dụng hàm **SWITCHOFFSET.**

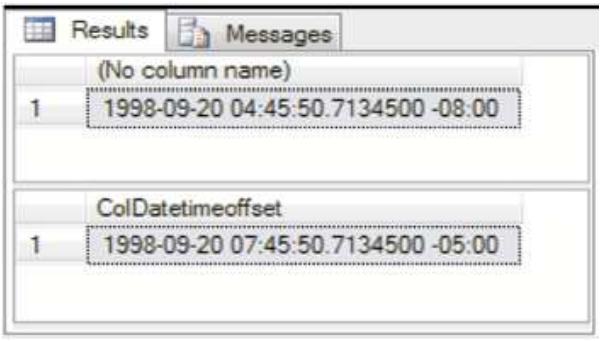
Code Snippet 18:

```
CREATE TABLE Test
  (
```

Concepts

```
ColDatetimeoffset datetimeoffset
 );
GO
INSERT INTO Test
VALUES ('1998-09-20 7:45:50.71345 -5:00');
GO
SELECT SWITCHOFFSET (ColDatetimeoffset, '-08:00')
FROM Test;
GO
--Returns: 1998-09-20 04:45:50.7134500 -08:00
SELECT ColDatetimeoffset
FROM Test;
```

Hình 13.10 hiển thị đầu ra của Code Snippet 18.



Hình 13.10: Use of SWITCHOFFSET Function

- **DATETIMEOFFSETFROMPARTS**

Hàm này trả về giá trị datetimeoffset cho ngày và thời gian cụ thể với độ chính xác và độ lệch đã chỉ định.

Sau đây là cú pháp cho **DATETIMEOFFSETFROMPARTS**.

Cú pháp:

```
DATETIMEOFFSETFROMPARTS ( year, month, day, hour,
minute, seconds, fractions, hour_offset,
minute_offset, precision )
```

trong đó,

year: chỉ ra biểu thức số nguyên cho năm.

month: chỉ ra biểu thức số nguyên cho tháng.

day: chỉ ra biểu thức số nguyên cho ngày.

hour: chỉ ra biểu thức số nguyên cho giờ.

minute: chỉ ra biểu thức số nguyên cho phút.

seconds: chỉ ra biểu thức số nguyên cho giây.

fractions: chỉ ra biểu thức số nguyên cho các phân số.

hour_offset: chỉ ra biểu thức số nguyên cho phần giờ của độ lệch múi giờ.

minute_offset: chỉ ra biểu thức số nguyên cho phần phút của độ lệch múi giờ.

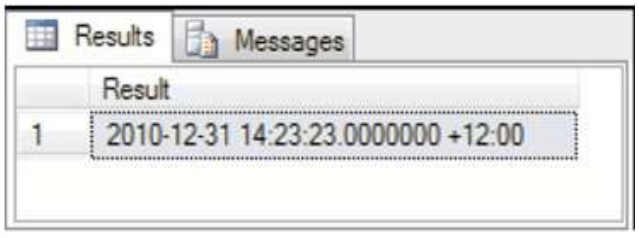
precision: chỉ ra độ chính xác theo số nguyên của giá trị datetimeoffset để được trả về.

Code Snippet 19 trình bày việc sử dụng hàm **DATETIMEOFFSETFROMPARTS**.

Code Snippet 19:

```
SELECT DATETIMEOFFSETFROMPARTS ( 2010, 12, 31, 14, 23, 23, 0, 12, 0,
7 )
AS Result;
```

Đoạn mã này hiển thị giá trị datetimeoffset cho ngày và thời gian đã cho với độ chính xác và độ lệch đã chỉ định.
Hình 13.11 hiển thị đầu ra của Code Snippet 19.



Hình 13.11: Use of DATETIMEOFFSETFROMPARTS Function

Concepts

➢ **SYSDATETIMEOFFSET**

Những hàm trả về giá trị datetimeoffset(7), trong đó có ngày và thời gian của máy tính trên đó thể hiện của SQL Server đang chạy.

Sau đây là cú pháp cho **SYSDATETIMEOFFSET**.

Cú pháp:

```
SYSDATETIMEOFFSET ()
```

Code Snippet 20 displays the different formats used by the date and time functions.

Code Snippet 20:

```
SELECT SYSDATETIME() AS SYSDATETIME
  ,SYSDATETIMEOFFSET() AS SYSDATETIMEOFFSET
  ,SYSUTCDATETIME() AS SYSUTCDATETIME
```

Hình 13.12 trình bày việc sử dụng hàm **SYSDATETIMEOFFSET**.

	SYSDATETIME	SYSDATETIMEOFFSET	SYSUTCDATETIME
1	2013-02-08 16:08:16.6565247	2013-02-08 16:08:16.6565247 +05:30	2013-02-08 10:38:16.6565247

Hình 13.12: Use of SVSDATETIMEOFFSET Function

➢ **Các hàm phân tích**

SQL Server 2012 hỗ trợ một số hàm phân tích. Những hàm này tính toán tổng giá trị dựa trên một nhóm các hàng. Các hàm phân tích tính toán tổng số chạy, trung bình di chuyển, hoặc N kết quả đầu trong một nhóm.

Table 13.7 liệt kê một số những hàm này.

Hàm	Mô tả
LEAD	Cung cấp quyền truy cập vào dữ liệu từ hàng tiếp theo trong cùng một tập kết quả mà không sử dụng phép tự nối.
LAST VALUE	Lấy giá trị cuối cùng trong một tập giá trị có thứ tự.
LAG	Cung cấp quyền truy cập vào dữ liệu từ hàng trước trong cùng một tập kết quả mà không sử dụng phép tự nối.
FIRST_VALUE	Lấy giá trị đầu tiên trong một tập giá trị có thứ tự.
CUME_DIST	Tính toán phân phối tích lũy của một giá trị trong một nhóm các giá trị.
PERCENTILE- CONT	Tính phần trăm dựa trên phân phối liên tục của giá trị cột trong SQL.

Hàm	Mô tả
PERCENTILE_DISC	Tính toán phần trăm cụ thể cho các giá trị đã sắp xếp trong toàn bộ tập hàng hoặc trong các phân vùng riêng biệt của một tập hàng.

Table 13.7: Analytic Functions

Code Snippet 21trình bày việc sử dụng hàm **LEAD()**.

Code Snippet 21:

```
USE AdventureWorks2012;
GO
SELECT BusinessEntityID, YEAR(QuotaDate) AS QuotaYear, SalesQuota AS
NewQuota,
 LEAD(SalesQuota, 1,0) OVER (ORDER BY YEAR(QuotaDate)) AS FutureQuota
FROM Sales.SalesPersonQuotaHistory
WHERE BusinessEntityID = 275 and YEAR(QuotaDate) IN ('2007','2008');
```

Trong đoạn mã này, hàm LEAD() được sử dụng để trả lại sự khác biệt trong các hạn ngạch bán hàng cho một nhân viên cục thể trong những năm tiếp theo.

Code Snippet 22 trình bày việc sử dụng hàm **FIRST_VALUE()**

Code Snippet 22:

```
USE AdventureWorks2012;
GO
SELECT Name, ListPrice,
  FIRST_VALUE(Name) OVER (ORDER BY ListPrice ASC) AS LessExpensive
FROM Production.Product
WHERE ProductSubcategoryID = 37
```

Trong đoạn mã này, hàm FIRST_VALUE() so sánh các sản phẩm trong thể loại sản phẩm 37 và trả về tên sản phẩm rẻ hơn.

Concepts

13.9 Kiểm tra tiến độ của bạn

1. Câu nào sau đây không phải là tính năng kiểm soát việc sử dụng nhiều câu lệnh Transact- SQL cùng một lúc?

(A)	Tập lệnh	(C)	Điều khiển luồng
(B)	Xử lý lỗi	(D)	Các biến

2. Điều nào sau đây được sử dụng để đặt và khai báo các biến cục bộ được cung cấp bởi SQL Server?

a.	DECLARE
b.	SET
c.	DELETE
d.	INSERT

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

3. Điều nào sau đây không phải là cấp phép được áp dụng trên một từ đồng nghĩa?

(A)	GRANT	(C)	DELETE
(B)	CONTROL	(D)	UPDATE

4. Đoạn mã nào sau đây sử dụng biến cục bộ để lấy thông tin liên hệ cho các tên gọi bắt đầu bằng Per?

(A)	USE AdventureWorks2012; GO DECLARE @find varchar(30); DECLARE @find varchar(30) = 'Per%'; SET @find = 'Per%'; SELECT p.LastName, p.FirstName, ph.PhoneNumber FROM Person.Customer AS p JOIN Person.Phone AS ph ON p.BusinessEntityID = ph.BusinessEntityID WHERE LastName LIKE @find;
(B)	USE AdventureWorks2012; GO DECLARE find varchar(30); DECLARE find varchar(30) = 'Per%'; SET find = 'Per%'; SELECT p.LastName, p.FirstName, ph.PhoneNumber FROM Person.Customer AS p JOIN Person.Phone AS ph ON p.BusinessEntityID = ph.BusinessEntityID WHERE LastName LIKE find;
(C)	USE AdventureWorks2012; GO @find varchar(30); @find varchar(30) = 'Per%'; SET @find = 'Per%'; SELECT p.LastName, p.FirstName, ph.PhoneNumber FROM Person.Customer AS p JOIN Person.Phone AS ph ON p.BusinessEntityID = ph.BusinessEntityID WHERE LastName LIKE @find;

Concepts

(D)	USE AdventureWorks2012; GO SET @find varchar(30); SET @find varchar(30) = 'Per%'; SET @find = 'Per'; SELECT p.LastName, p.FirstName, ph.PhoneNumber FROM Person.Customer AS p JOIN Person.Phone AS ph ON p.BusinessEntityID = ph.BusinessEntityID WHERE LastName LIKE @find;

5. Hàm nào sau đây không phải là hàm không xác định?

(A)	@@PACK_SENT	(C)	@@DBTS
(B)	@@IOBUSY	(D)	IDLE

13.9.1 Đáp án

1.	A
2.	C
3.	A
4.	A
5.	D

Tóm tắt

- Transact-SQL cung cấp các phần tử lập trình cơ bản như là các biến, phần tử điều khiển luồng, cấu trúc điều kiện, và vòng lặp.
- Khối lệnh là tập hợp một hoặc nhiều câu lệnh Transact-SQL được gửi như một đơn vị từ một ứng dụng đến máy chủ.
- Biến cho phép người dùng lưu trữ dữ liệu để sử dụng làm đầu vào trong các câu lệnh Transact-SQL khác.
- Các đồng nghĩa đem lại cách để có bí danh cho đối tượng cơ sở dữ liệu có thể tồn tại trên một máy chủ từ xa hoặc cục bộ.
- Các hàm xác định mỗi lần trả về cùng một kết quả mỗi khi chúng được gọi với một tập hợp xác định các giá trị đầu vào và chỉ ra cùng một trạng thái của cơ sở dữ liệu.
- Các hàm không xác định trả về các kết quả khác nhau mỗi lần chúng được gọi với tập hợp đã chỉ định các giá trị đầu vào mặc dù cơ sở dữ liệu được truy cập vẫn giữ nguyên.
- Hàm cửa sổ là một hàm áp dụng cho một bộ sưu tập các hàng.

1. Zen Technologies là một công ty hàng đầu trong lĩnh vực dệt nằm ở California. Ban quản lý của công ty muốn đưa ra giải thưởng trung thành với tất cả nhân viên hoàn thành nhiệm kỳ 5 năm trong tổ chức. Sử dụng cùng một bảng **Employee**, tạo ra một khối lệnh Transact-SQL để trả về **EmployeeID**, **FirstName**, **Department**, và **HireDate** của tất cả các nhân viên như vậy. Giả sử rằng ban quản lý cho tăng lương hai mươi lăm phần trăm cho tất cả mọi người hoàn thành một năm trong tổ chức. Chủ tịch hội đồng quản trị của tổ chức muốn tham gia vào cuộc khảo sát lương toàn quốc gia.

   Hãy viết khối lệnh để xác định tổng số tiền lương trả cho một bộ phận trong đó sử dụng nhiều hơn một nhân viên. Ban quản lý muốn tìm ra bộ phận đã thuê số lượng người lao động lớn nhất trong năm năm qua.

   Gợi ý:

   - Sử dụng hàm **DATETIMEOFFSETFROMPARTS**.

Concepts