Quản Trị Dữ Liệu Với Microsoft SQL Server

Chương: 9

Truy vấn nâng cao và các phép kết nối (Advanced Queries and Joins)

Mục tiêu

- Giải thích về nhóm và thống kê dữ liệu
- Mô tả truy vấn con
- Mô tả về biểu thức bảng (table expressions)
- Giải thích về ghép nối dữ liệu(joins)
- Mô tả về các kiểu ghép nối khác nhau
- Giải thích cách dùng bộ toán tử để kết hợp dữ liệu
- Mô tả bộ toán tử pivoting và grouping

Giới thiệu

- SQL Server 2012 đưa vào nhiều tính năng truy vấn mạnh mẽ giúp bạn có thể lấy dữ liệu nhanh chóng và hiệu quả.
- Dữ liệu có thể được nhóm và/hoặc tổng hợp(aggregated) cùng nhau để thể hiện tông tin tổng kết.
- Sử dụng khái niệm truy vấn con, tập kết quả của một câu lệnh `SELECT` có được sử dụng làm điều kiện cho câu lệnh `SELECT` hoặc truy vấn khác.
- Ghép nối `(Joins)` giúp bạn kết hợp các cột dữ liệu từ hai hay nhiều bảng dựa trên mối quan hệ giữa các bảng.
- Bộ toán tử như `UNION` và `INTERSECT` giúp bạn kết hợp các dòng dữ liệu từ hai hay nhiều bảng.
- Toán tử `PIVOT` và `UNPIVOT` được sử dụng để chuyển(transform) dữ liệu hương cột sang hướng dòng và ngược lại (vice versa).
- Mệnh đề con `GROUPING SET` của mệnh đề `GROUP BY` giúp xác định nhiều nhóm trong một truy vấn đơn.

SQL Server 2012

Nhóm dữ liệu 1-3

Mệnh đề `GROUP BY` phân vùng(partitions) tập kết quả thành một hoặc nhiều tập con. Mỗi tập kết quả có các giá trị và biểu thức chung.

Từ khóa `GROUP BY`, theo sau nó là danh sách các cột được dùng để nhóm. Mỗi cột được nhóm giới hạn số lượng dòng của tập kết quả.

Mỗi cột nhóm, chỉ có một dòng. Mệnh đề `GROUP BY` có thể nhiều hơn một cột nhóm.

- Cú pháp mệnh đề `GROUP BY`:

```
SELECT <select list>
FROM <table source>
WHERE <search condition>
GROUP BY <group by list>
```

Trong đó,

`<group by list>`: danh sách các cột được chọn để nhóm các dòng.

Nhóm dữ liệu 2-3

- Hãy xem bảng `WorkOrderRouting` trog csdl `AdventureWorks2012`.
- Cần tính tổng số giờ tài nguyên(resource hours) cho mỗi đơn hàng công việc(work order).
- Để thực hiện điều này, các bản ghi cần được nhóm theo mã số đơn làm việc là `WorkOrderID` .
- Đoạn code sau minh họa tính và hiển thị tổng số giờ tài nguyên của mỗi đơn hàng công việc theo mã số đơn `WorkOrderID`:

```
SELECT WorkOrderID, SUM(ActualResourceHrs) AS
TotalHoursPerWorkOrder

FROM Production.WorkOrderRouting

GROUP BY WorkOrderID
```

- Trong truy vấn trên, sử dụng hàm dựng sẵn `SUM()` để thực hiện tính tổng.
- `SUM()` là một hàm thống kê(aggregate).

Nhóm dữ liệu 3-3

- Khi truy vấn được thực thi sẽ trả về tất cả mã số đơn công việc (`WorkOrderID`) cùng với tổng số giờ cho mỗi đơn hàng công việc.
- Hình dưới đây cho thấy một phần kết quả

	WorkOrderID	TotalHoursPerWorkOrder
1	13	17.6000
2	14	17.6000
3	15	4.0000
4	16	4.0000
5	17	4.0000
6	18	4.0000
7	19	4.0000
8	20	4.0000
9	21	4.0000
10	22	4.0000

SQL Server 2012

GROUP BY với WHERE 1-2

Có thể sử dụng mệnh đề WHERE cùng với mệnh đề GROUP BY để giới hạn số dòng trước khi thực hiện nhóm.

Các dòng thỏa mãn điều kiện sẽ được xem xét cho việc nhóm.

Các dòng dữ liệu không phù hợp với điều kiện trong mệnh đề WHERE sẽ bị loại bỏ trước khi công việc nhóm được thực hiện.

- Đoạn code dưới đây minh họa một truy vấn giới hạn các dòng được hiển thị. Chỉ hiển thị các bản ghi có `WorkOrderID` nhỏ hơn 50:

```
SELECT WorkOrderID, SUM(ActualResourceHrs) AS
TotalHoursPerWorkOrder
FROM Production.WorkOrderRouting
WHERE WorkOrderID <50
GROUP BY WorkOrderID
```

GROUP BY với WHERE 2-2

- Số bản ghi được trả về là 25, một phần kết quả được thể hiện trong hình sau:

	WorkOrderID	TotalHoursPerWorkOrder
1	13	17.6000
2	14	17.6000
3	15	4.0000
4	16	4.0000
5	17	4.0000
6	18	4.0000
7	19	4.0000
8	20	4.0000

GROUP BY với NULL

Nếu các cột nhóm có chứa giá trị NULL, dòng đó sẽ trở thành một nhóm riêng biệt trong tập kết quả.

Nếu các cột nhóm có chứa nhiều hơn một giá trị NULL, các giá trị NULL được đặt thành một dòng duy nhất.

Hãy xem xét bảng Production.Product. Có một số dòng có chứa giá trị NULL trong cột Class.

- Đoạn dưới đây minh họa lấy và hiển thị giá trung bình của mỗi Class:

```
SELECT Class, AVG (ListPrice) AS 'AverageListPrice'
FROM Production.Product
GROUP BY Class
```

- Kết quả được thể hiện trong hình sau, các giá trị NULL được nhóm thành một dòng duy nhất trong kết quả:

	Class	AverageListPrice
1	NULL	16.314
2	H	1679.4964
3	L	370.6887
4	M	635.5816

GROUP BY với ALL 1-2

Cũng có thể sử dụng từ khóa ALL với mệnh đề GROUP BY và nó chỉ có ý nghĩa khi câu lệnh SELECT có kèm theo mệnh đề WHERE.

Khi sử dụng mệnh đề WHERE trước GROUP BY, các dòng không thỏa mãn điều kiện sẽ bị loại bỏ trước khi thực hiện gộp nhóm, nhưng nếu sử dụng thêm từ khóa ALL, các dòng không thỏa mãn điều kiện vẫn được gộp nhóm và đưa ra trong tập kết quả.

- Cú pháp sử dụng GROUP BY với từ khóa ALL như sau:

Cú pháp

```
SELECT <column_name>
FROM <table_name>
WHERE <condition>
GROUP BY ALL <column _name>
```

GROUP BY với ALL 2-2

- Hãy xem xét bảng `Sales.SalesTerritory`.
- Trong bảng này có cột tên `Group` chỉ ra khu vực địa lý mà các hạt/vùng(territory) bán hàng thuộc về.
- Đoạn code dưới đây tính và hiển thị tổng bán(sales) của mỗi nhóm:

```
SELECT [Group],SUM(SalesYTD) AS 'TotalSales' FROM Sales.SalesTerritory
WHERE [Group] LIKE 'N%' OR [Group] LIKE 'E%'
GROUP BY ALL [Group]
```

- Kết quả cần hiển thị tất cả các nhóm cho dù họ có bán được bất kỳ số lượng nào hay không.
- Để thực hiện được điều đó, đoạn code sử dụng GROUP BY với ALL.
- Ngoài các hàng được hiển thị, nó cũng sẽ hiển thị nhóm 'Pacific' với giá trị null như thể hiện trong hình sau đây:

	Group	TotalSales
1	Europe	13590506.0212
2	North America	33182889.0168
3	Pacific	NULL

- Kết quả như vậy là do vùng Pacific không có bất kỳ số lượng bán nào.

SQL Server 2012

GROUP BY với HAVING 1-2

Mệnh đề HAVING chỉ được sử dụng cùng với mệnh đề GROUP BY trong câu lệnh SELECT để chỉ ra điều kiện lọc dữ liệu sau khi đã tính toán theo nhóm.

Mệnh đề HAVING có vai trò như mệnh đề WHERE, nhưng nó được dùng ở những nơi mà mệnh đề WHERE không thể sử dụng được với các hàm tổng hợp như SUM ().

Sau khi bạn đã tạo ra các nhóm bằng mệnh đề GROUP BY, và bạn muốn tiếp tục lọc các nhóm kết quả nữa.

Mệnh đề HAVING đóng vai trò như bộ lọc trên các nhóm, tương tự cách mệnh đề WHERE lọc trên các dòng được trả về bởi mệnh đề FROM.

GROUP BY với HAVING 2-2

- Cú pháp sử dụng mệnh đề `GROUP BY` với `HAVING` như sau:

```
SELECT <column_name>
FROM <table_name>
GROUP BY <column_name>
HAVING <search_condition>
```

- Đoạn code dưới đây hiển thị các dòng được nhóm theo 'Pacific' và có tổng lượng bán(sales) nhỏ hơn 6000000 :

```
SELECT [Group],SUM(SalesYTD) AS 'TotalSales'
FROM Sales.SalesTerritory
WHERE [Group] LIKE 'N%' OR [Group] LIKE 'E%'
GROUP BY ALL [Group]
HAVING SUM(SalesYTD)< 6000000
```

- Kết quả của đoạn code trên chỉ có một dòng với tên Group là Pacific có tổng bán 5977814.9154.

SQL Server 2012

CUBE 1-2

CUBE là một toán tử thống kê sinh ra(produce) một dòng siêu thống kê (super-aggregate).

Ngoài các dòng dữ liệu được tạo ra bởi GROUP BY, nó còn cung cấp thêm các tổng hợp(summary) của các dòng được phát sinh bởi mệnh đề GROUP BY.

Hiển thị dòng tổng hợp (summary) cho mỗi kết hợp của từng nhóm trong tập kết quả.

Hiển thị dòng tổng hợp với giá trị NULL trong tập kết quả, cũng nhưng đồng thời trả về tất cả các giá trị cho những giá trị.

- Cú pháp sử dụng toán tử CUBE với mệnh đề GROUP BY:

```
SELECT <column_name>
FROM <table_name>
GROUP BY <column_name> WITH CUBE
```

CUBE 2-2

- Đoạn code dưới đây minh họa sử dụng toán tử CUBE:

```
SELECT Name,CountryRegionCode,SUM(SalesYTD) AS TotalSales
FROM Sales.SalesTerritory
WHERE Name <> 'Australia' AND Name<> 'Canada'
GROUP BY Name,CountryRegionCode WITH CUBE
```

- Truy vấn lấy và hiện thị tổng bán theo từng quốc gia, và tổng bán theo từng vùng.
- Kết quả được thể hiện trong hình dưới đây:

	Name	CountryRegionCode	TotalSales
1	Germany	DE	3805202.3478
2	NULL	DE	3805202.3478
3	France	FR	4772398.3078
4	NULL	FR	4772398.3078
5	United Kingdom	GB	5012905.3656
6	NULL	GB	5012905.3656
7	Central	US	3072175.118
8	Northeast	US	2402176.8476
9	Northwest	US	7887186.7882
10	Southeast	US	2538667.2515

SQL Server 2012

ROLLUP 1-2

Ngoài các dòng dữ liệu được tạo ra bởi `GROUP BY` nó còn đưa thêm các dòng tổng hợp(summary) vào tập kết quả.

Tương tự như toán tử `CUBE` nhưng sinh ra một tập kết quả trình bày các nhóm được sắp đặt (arrange) theo thứ tự phân cấp.

Nó sắp đặt các nhóm từ thấp nhất đến cao nhất.

Việc phân cấp các nhóm trong tập kết phụ thuộc vào thứ tự các cột được chỉ ra khi nhóm.

- Cú pháp `ROLLUP` như sau:

Cú pháp:

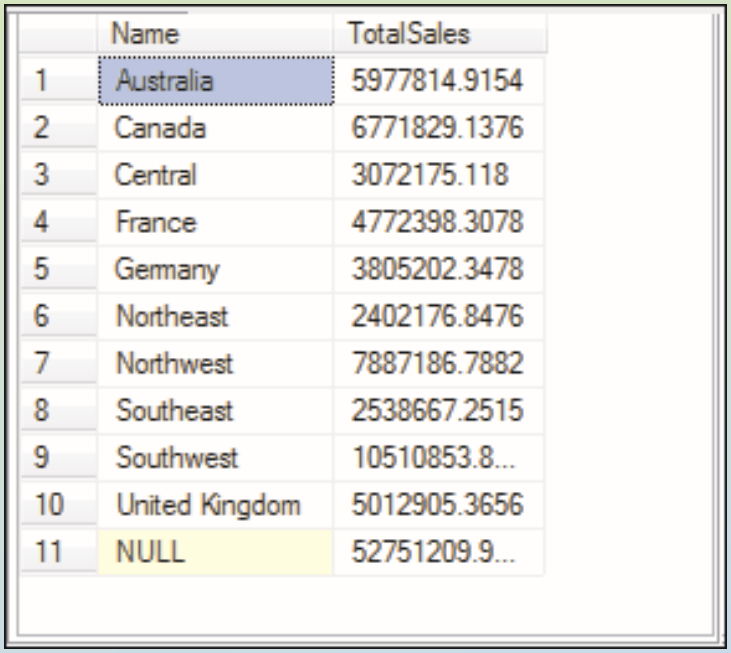
```
SELECT <column_name> FROM <table_name>
GROUP BY <column_name>
WITH ROLLUP
```

ROLLUP 2-2

- Đoạn code dưới đây minh họa sử dụng ROLLUP:

```
SELECT Name, CountryRegionCode,SUM(SalesYTD) AS TotalSales
FROM Sales.SalesTerritory
WHERE Name <> 'Australia' AND Name<> 'Canada'
GROUP BY Name,CountryRegionCode WITH ROLLUP
```

- Nó lấy và hiển thị tổng bán của từng quốc gia, tổng bán của tất cả quốc gia trong từng vùng(region) và sắp đặt chúng theo thứ tự.
- Kết quả được thể hiện trong hình sau:



	Name	TotalSales
1	Australia	5977814.9154
2	Canada	6771829.1376
3	Central	3072175.118
4	France	4772398.3078
5	Germany	3805202.3478
6	Northeast	2402176.8476
7	Northwest	7887186.7882
8	Southeast	2538667.2515
9	Southwest	10510853.8...
10	United Kingdom	5012905.3656
11	NULL	52751209.9...

SQL Server 2012

Các hàm thống kê (Aggregate functions) 1-5

Các hàm thống kê thực hiện tính toán trên tập giá trị và trả về giá trị duy nhất

Các hàm thống kê đều bỏ qua giá trị NULL, nhưng trừ hàm COUNT(*).

Các hàm thống kê thường được sử dụng với mệnh đề GROUP BY của câu lệnh SELECT.

Khi sử dụng hàm thống kê trong phần danh sách cột của câu lệnh SELECT thường tạo ra một cột không có bí danh(alias), do vậy bạn có thể dùng mệnh đề AS cung cấp cho nó một bí danh.

Các hàm thống kê trong mệnh đề SELECT hoạt động trên tất cả các hàng thông qua các giai đoạn SELECT. Nếu không có mệnh đề GROUP BY, tất cả các hàng sẽ được tổng hợp.

Các hàm thống kê 2-5

- SQL Server cung cấp nhiều hàm thống kê dựng sẵn.
- Bảng sau đây liệt kê các hàm thống kê thường hay sử dụng:

Function Name	Syntax	Description
AVG	AVG(<expression>)	Calculates the average of all the non-NULL numeric values in a column.
COUNT or COUNT_BIG	COUNT(*) or COUNT(<expression>)	When (*) is used, this function counts all rows, including those with NULL. The function returns count of non-NULL rows for the column when a column is specified as <expression>. The return value of COUNT function is an int. The return value of COUNT_BIG is a big_int.
MAX	MAX(<expression>)	Returns the largest number, latest date/time, or last occurring string.
MIN	MIN(<expression>)	Returns the smallest number, earliest date/time, or first occurring string.
SUM	SUM(<expression>)	Calculates the sum of all the non-NULL numeric values in a column.

Các hàm thống kê 3-5

- Đoạn code sau đây minh họa sử dụng các hàm thống kê có sẵn trong mệnh đề SELECT:

```
SELECT AVG([UnitPrice]) AS AvgUnitPrice,
MIN([OrderQty])AS MinQty,
MAX([UnitPriceDiscount]) AS MaxDiscount
FROM Sales.SalesOrderDetail;
```

- Do truy vấn không sử dụng mệnh đề GROUP BY, tất cả các dòng trong bảng sẽ được tổng hợp bằng các hàm tổng hợp trong mệnh đề SELECT.
- Kết quả được thể hiện trong hình sau:

	AvgUnitPrice	MinQty	MaxDiscount
1	465.0934	1	0.40

- Khi sử dụng các hàm thống kê trong mệnh đề SELECT, tất cả các cột được tham chiếu trong danh sách cột của SELECT phải được sử dụng trong tham số đầu vào cho các hàm thống kê hoặc phải được tham chiếu trong mệnh đề GROUP BY.

Các hàm thống kê 4-5

- Đoạn code dưới đây minh họa trả về lỗi:

```
SELECT SalesOrderID, AVG(UnitPrice) AS AvgPrice
FROM Sales.SalesOrderDetail;
```

Trả về lỗi do cột `Sales.SalesOrderDetail.SalesOrderID` được liệt kê trong danh sách cột của `SELECT` nhưng không là tham số của hàm thống kê và cũng không được chỉ ra ở mệnh đề `GROUP BY`.

Khi truy vấn không sử dụng mệnh đề `GROUP BY` tất cả các dòng sẽ coi như là một nhóm. Do đó tất cả các cột phải sử dụng như là tham số đầu vào cho các hàm thống kê.

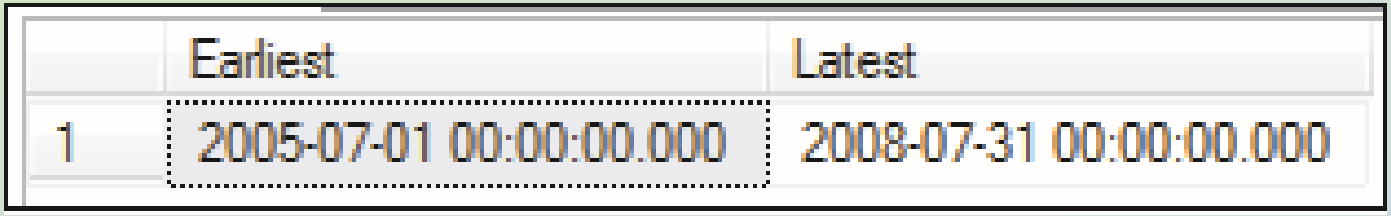
SQL Server 2012

Các hàm thống kê 5-5

- Để tránh lỗi, cần xóa bỏ cột `SalesOrderID` khỏi truy vấn.
- Bên cạnh việc tổng hợp trên các dữ liệu số, các biểu thức thống kê cũng có thể thống kê trên các dữ liệu date, time, và dữ liệu kí tự.
- Đoạn code dưới đây trả về ngày của hóa đơn được lập cũ nhất(earliest) và cũ nhất(latest) bằng hàm `MIN`, `MAX`:

```
SELECT MIN(OrderDate)AS Earliest,
MAX(OrderDate) AS Latest
FROM Sales.SalesOrderHeader;
```

- Kết quả được thể hiện trong hình sau:



	Earliest	Latest
1	2005-07-01 00:00:00.000	2008-07-31 00:00:00.000

Spatial Aggregates 1-4

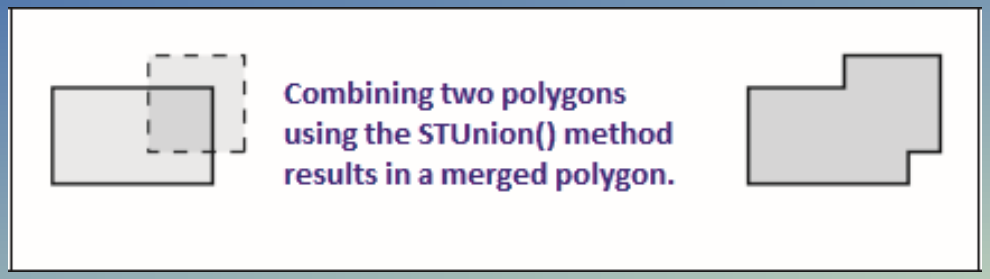
- SQL Server provides several methods that help to aggregate two individual items of geometry or geography data.
- The following table lists the methods:

Method	Description
STUnion	Returns an object that represents the union of a geometry/geography instance with another geometry/geography instance.
STIntersection	Returns an object that represents the points where a geometry/geography instance intersects another geometry/geography instance.
STConvexHull	Returns an object representing the convex hull of a geometry/geography instance.

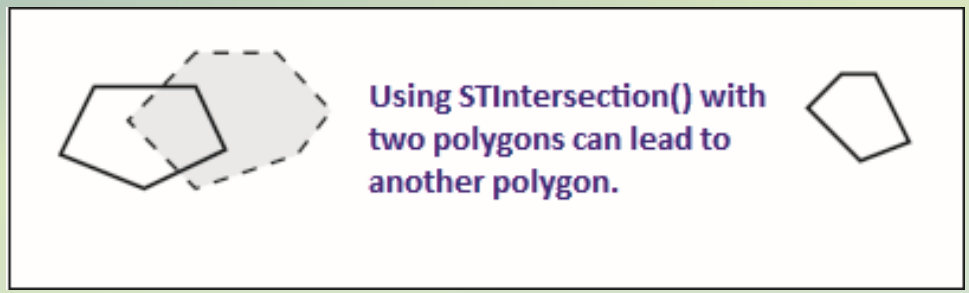
- A set of points is called convex if for any two points, the entire segment is contained in the set.
- The convex hull of a set of points is the smallest convex set containing the set.
- For any given set of points, there is only one convex hull.

Spatial Aggregates 2-4

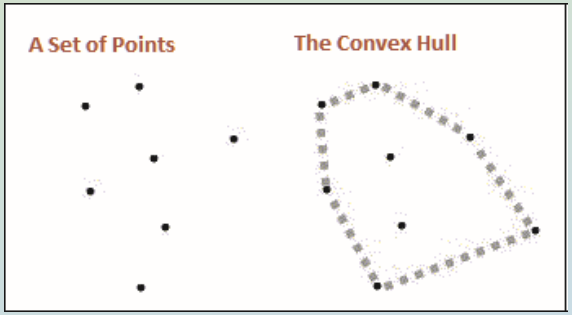
- Following figure depicts an example of `STUnion()`:



- Following figure depicts an example of `STIntersection()`:



- Following figure depicts an example of `STConvexHull()`:

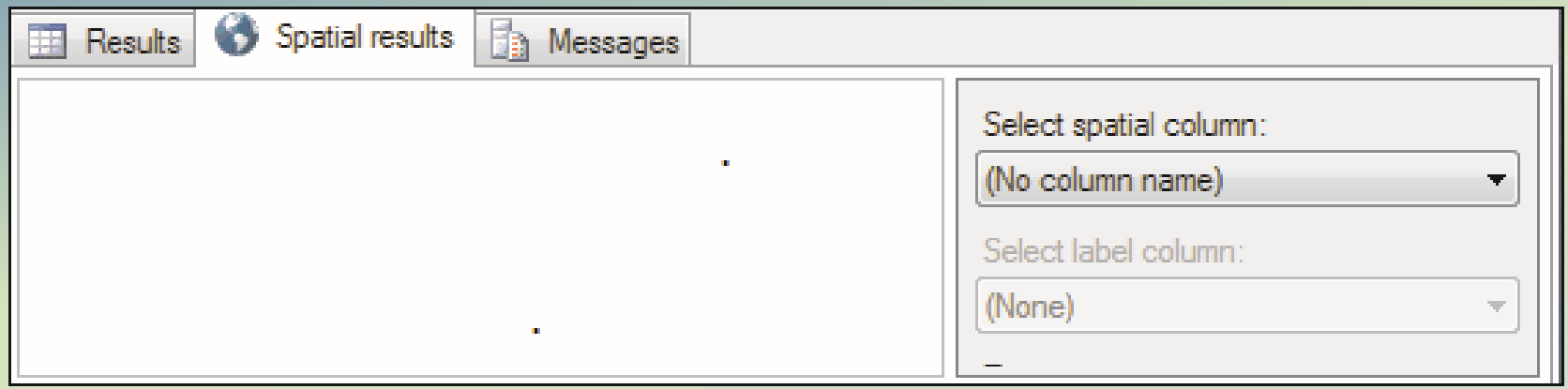


Spatial Aggregates 3-4

- The following code snippet demonstrates the use of `STUnion()`:

```
SELECT geometry::Point(251, 1,
4326).STUnion(geometry::Point(252,2,4326));
```

- The following figure displays the output as two points:



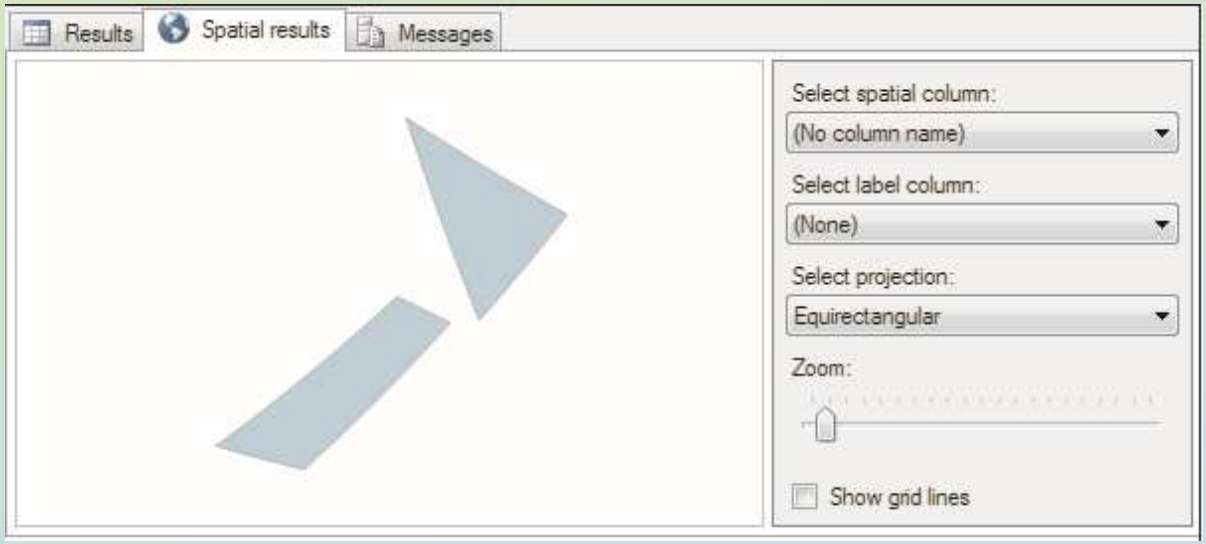
- The following code snippet displays another example:

```
DECLARE @City1 geography
SET @City1 = geography::STPolyFromText(
'POLYGON((175.3 -41.5, 178.3 -37.9, 172.8 -34.6, 175.3 -41.5))',
4326)
```

Spatial Aggregates 4-4

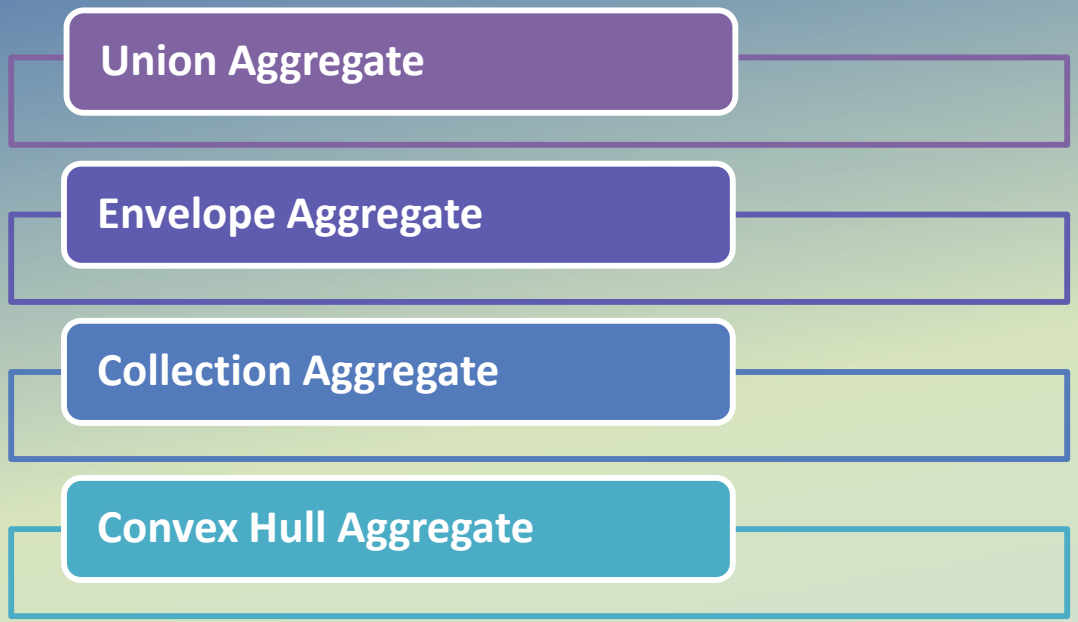
```
DECLARE @City2 geography
SET @City2 = geography::STPolyFromText(
'POLYGON((169.3 -46.6, 174.3 -41.6, 172.5 -40.7, 166.3 -45.8, 169.3 -46.6))',
4326)
DECLARE @CombinedCity geography = @City1.STUnion(@City2)
SELECT @CombinedCity
```

- Here, two variables are declared of the geography type and appropriate values are assigned to them.
- Then, they are combined into a third variable of geography type by using the `STUnion()` method. The following figure shows the output of the code:



New Spatial Aggregates

- SQL Server 2012 has introduced four new aggregates to the suite of spatial operators in SQL Server.



- These aggregates are implemented as static methods, which work for either the `geography` or the `geometry` data types.

- Although aggregates are applicable to all classes of spatial data, they can be best described with polygons.

Union Aggregate 1-2

It performs a union operation on a set of geometry objects.

It combines multiple spatial objects into a single spatial object, removing interior boundaries, where applicable.

- The syntax of `UnionAggregate` is as follows:

Syntax:

```
UnionAggregate ( geometry_operand or geography_operand)
```

where,

`geometry_operand`: is a geometry type table column comprising the set of geometry objects on which a union operation will be performed.

`geography_operand`: is a geography type table column comprising the set of geography objects on which a union operation will be performed.

Union Aggregate 2-2

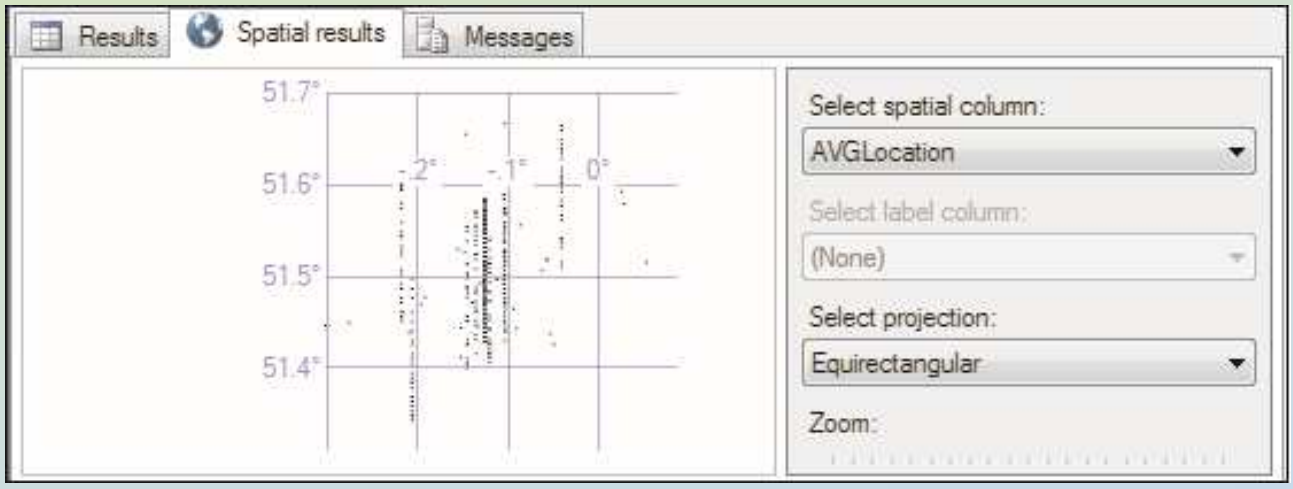
- Following code snippet demonstrates a simple example of using the `Union` aggregate.
- It uses the `Person.Address` table in the `AdventureWorks2012` database.

```
SELECT Geography::UnionAggregate(SpatialLocation)
AS AVGLocation
FROM Person.Address
WHERE City = 'London';
```

- The following figure displays the output:

	AVGLocation
1	0xE61000000104A00100003DA82EC605C249407D37109CAD...

- The following figure displays a visual representation of the spatial data and is viewed by clicking the **Spatial results** tab in the output window:



Envelope Aggregate 1-2

It returns a bounding area for a given set of `geometry` or `geography` objects. It exhibits different behaviors for geography and geometry types.

For the `geometry` type, the result is a 'traditional' rectangular polygon, which closely bounds the selected input objects.

For the `geography` type, the result is a circular object, which loosely bounds the selected input objects.

Furthermore, the circular object is defined using the new `CurvePolygon` feature.

- The following is the syntax of `EnvelopeAggregate`:

Syntax:

```
EnvelopeAggregate (geometry_operand or geography_operand)
```

where,

`geometry_operand`: is a geometry type table column comprising the set of geometry objects.

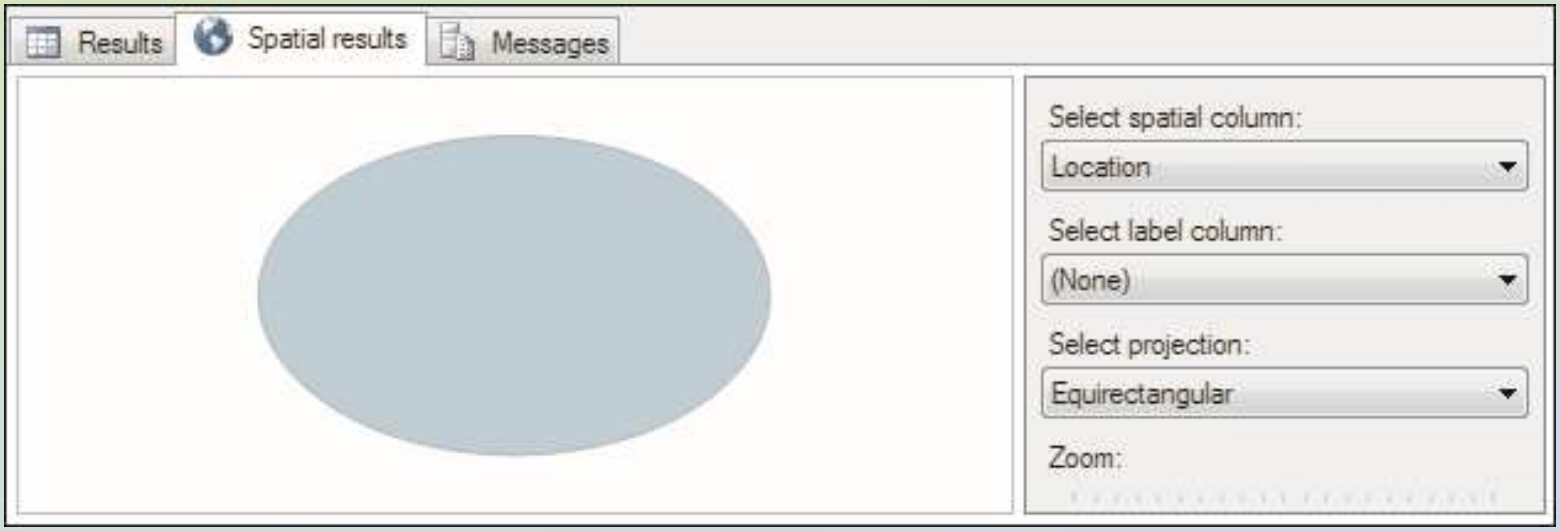
Envelope Aggregate 2-2

`geography_operand`: is a geography type table column comprising the set of geography objects.

- Following code snippet returns a bounding box for a set of objects in a table variable column:

```
SELECT Geography::EnvelopeAggregate(SpatialLocation)
AS Location
FROM Person.Address
WHERE City = 'London'
```

- The following figure shows the visual representation of the output:



Collection Aggregate 1-2

It returns a `GeometryCollection`/`GeographyCollection` instance with one `geometry`/`geography` part for each spatial object(s) in the selection set.

- The syntax of `CollectionAggregate` is as follows:

Syntax:

```
CollectionAggregate (geometry_operand or geography_operand)
```

where,

`geometry_operand`: is a geometry type table column comprising the set of geometry objects.

`geography_operand`: is a geography type table column comprising the set of geography objects.

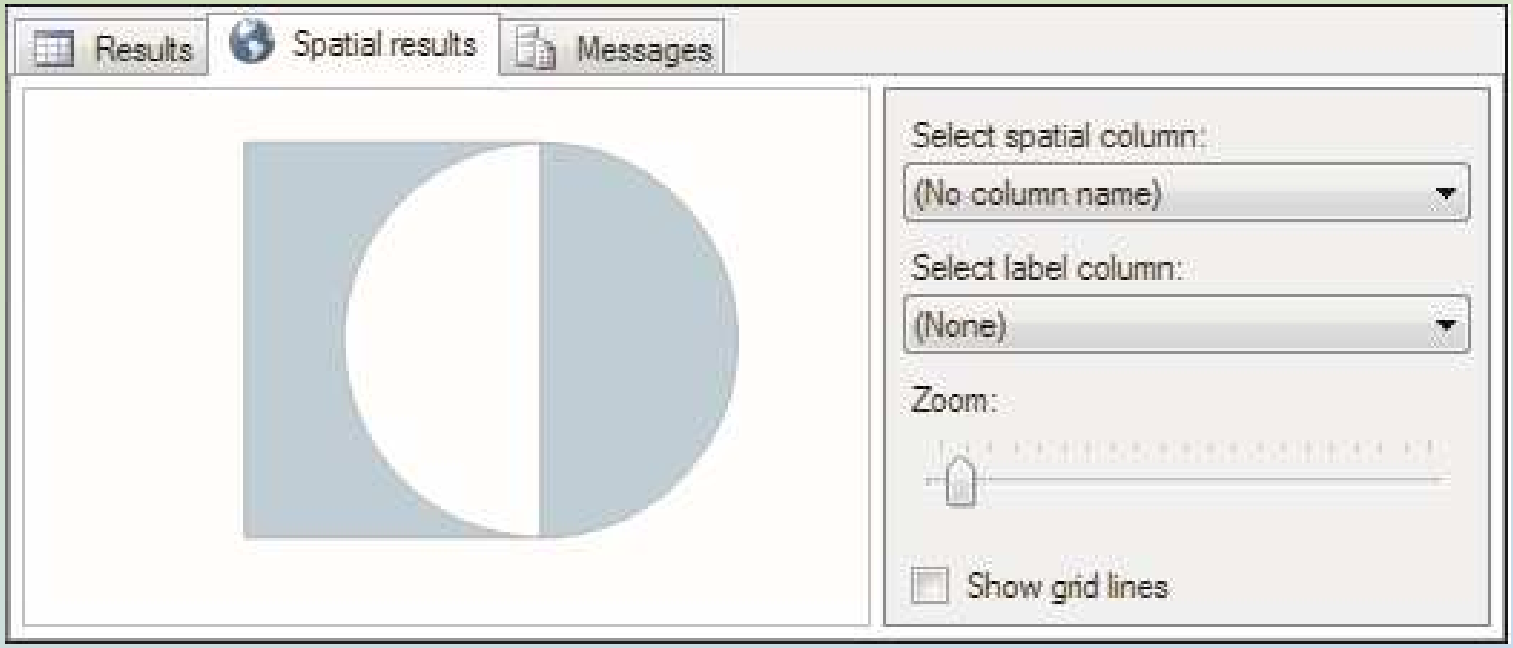
- Following code snippet returns a `GeometryCollection` instance that contains a `CurvePolygon` and a `Polygon`:

```
DECLARE @CollectionDemo TABLE
(
shape geometry,
```

Collection Aggregate 2-2

```
shapeType nvarchar(50)
)
INSERT INTO @CollectionDemo(shape,shapeType)
VALUES('CURVEPOLYGON(CIRCULARSTRING(2 3, 4 1, 6 3, 4 5, 2 3))',
 'Circle'),
('POLYGON((1 1, 4 1, 4 5, 1 5, 1 1))', 'Rectangle');
SELECT geometry::CollectionAggregate(shape)
FROM @CollectionDemo;
```

- The following figure shows the output of the code:



Convex Hull Aggregate 1-2

- It returns a convex hull polygon, which encloses one or more spatial objects for a given set of `geometry`/`geography` objects.
- The following is the syntax of `ConvexHullAggregate`:

Syntax:

```
ConvexHullAggregate (geometry_operand or geography_operand)
```

where,

`geometry_operand`: is a geometry type table column comprising the set of geometry objects.

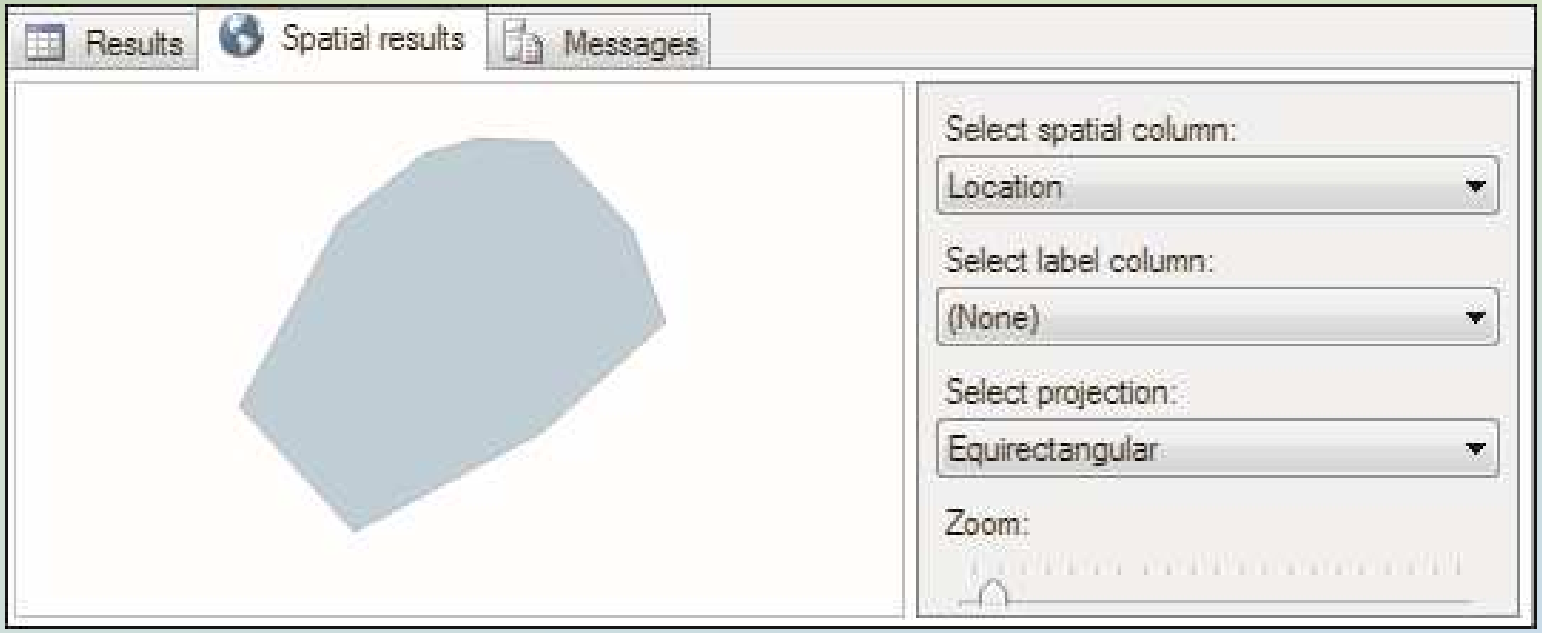
`geography_operand`: is a geography type table column comprising the set of geography objects.

Convex Hull Aggregate 2-2

- Following code snippet demonstrates the use of `ConvexHullAggregate`:

```
SELECT Geography::ConvexHullAggregate(SpatialLocation)
AS Location
FROM Person.Address
WHERE City = 'London'
```

- The output is shown in the following figure:



SQL Server 2012

Các truy vấn con 1-3

Bạn có thể sử dụng một câu lệnh SELECT hoạc một truy vấn trả về các bản ghi được để làm điều kiện cho câu truy lệnh SELECT hoặc truy vấn khác.

Truy vấn lồng(inner query) còn được gọi là truy vấn cha, và truy vấn bên trong được gọi là truy vấn con. Mục đích của truy vấn con là để trả về kết quả cho truy vấn bên ngoài.

Nói cách khác, lệnh truy vấn bên trong nên trả về cột hoặc các cột được sử dụng trong các điều kiện của lệnh truy vấn bên ngoài.

Dạng đơn giản nhất của một truy vấn con là chỉ trả về một cột. Truy vấn cha có thể sử dụng kết quả của truy vấn con này bằng một dấu =

- Cú pháp dạng truy vấn con cơ bản nhất như sau:

```
SELECT <ColumnName> FROM <table>

WHERE <ColumnName> = ( SELECT <ColumnName> FROM
<Table> WHERE <ColumnName> = <Condition>?>)
```

Các truy vấn con 2-3

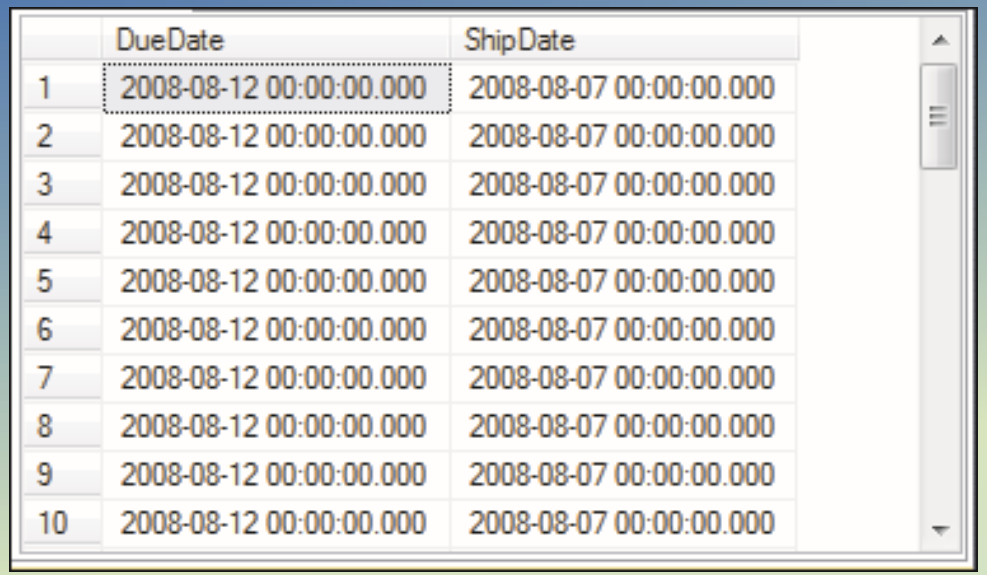
- Trong truy vấn con, câu lệnh SELECT trong cùng nhất được thực thi đầu tiên, và kết quả nó được truyền làm điều kiện cho lệnh SELECT bên ngoài.
- Hãy xem xét kịch bản trong đó cần xác định ngày đáo hạn(due date) và ngày chuyển hàng(ship date) của các đơn đặt hàng gần đây nhất.
- Đoạn code dưới đây trình bày để thực hiện được điều đó:

```
SELECT DueDate, ShipDate
FROM Sales.SalesOrderHeader
WHERE Sales.SalesOrderHeader.OrderDate =
(SELECT MAX(OrderDate) FROM Sales.SalesOrderHeader)
```

- Đoạn code trên có sử dụng truy vấn con để đạt được momg muốn.
- Truy vấn con sẽ lấy các ngày hóa đơn gần đây nhất.
- Sau đó kết quả của nó được truyền cho các truy vấn bên ngoài, hiển thị ngày đáo hạn(due date) và ngày chuyển hàng(ship date) cho tất cả các đơn đặt hàng đã được thực hiện vào ngày cụ thể.

Các truy vấn con 3-3

- Hình sau đây thể hiện một phần kết quả của đoạn code:



	DueDate	ShipDate
1	2008-08-12 00:00:00.000	2008-08-07 00:00:00.000
2	2008-08-12 00:00:00.000	2008-08-07 00:00:00.000
3	2008-08-12 00:00:00.000	2008-08-07 00:00:00.000
4	2008-08-12 00:00:00.000	2008-08-07 00:00:00.000
5	2008-08-12 00:00:00.000	2008-08-07 00:00:00.000
6	2008-08-12 00:00:00.000	2008-08-07 00:00:00.000
7	2008-08-12 00:00:00.000	2008-08-07 00:00:00.000
8	2008-08-12 00:00:00.000	2008-08-07 00:00:00.000
9	2008-08-12 00:00:00.000	2008-08-07 00:00:00.000
10	2008-08-12 00:00:00.000	2008-08-07 00:00:00.000

- Dựa trên kết trả về, truy vấn con được phân loại như sau:

Truy vấn con vô hướng (Scalar subqueries)

- trả về một giá trị. Ở đây, các truy vấn bên ngoài cần phải được viết để xử lý kết quả duy nhất.

Truy vấn con đa trị (Multi-valued subqueries)

- trả về kết quả tương tự như một cột của bảng. Ở đây truy vấn ngoài cần phải được viết để xử lý các kết quả đa trị có thể.

SQL Server 2012

Làm việc với truy vấn đa trị 1-4

Nếu toán tử = được sử dụng với truy vấn con, truy vấn con đó chỉ được trả về một giá trị vô hướng duy nhất.

Nếu trả về nhiều hơn một giá trị, truy vấn sẽ bị lỗi và không được thực thi.

Các từ khóa `ANY, ALL, IN`, và `EXISTS` có thể được sử dụng trong mệnh đề `WHERE` của câu lệnh `SELECT`, khi đó truy vấn trả về một cột hay có nhiều dòng.

Những từ khóa này còn được gọi là các vị từ (predicates), được sử dụng với các truy vấn đa trị.

Ví dụ, hãy xem xét tất cả họ(first names) và tên (last names) của nhân viên mà tiêu đề công việc (job title) của họ là 'Research and Development Manager' cần được hiển thị.

Ở đây, truy vấn lồng(inner query) có thể trả về nhiều hơn một dòng khi có thể có nhiều hơn một nhân viên với tiếu đề công việc như vậy.

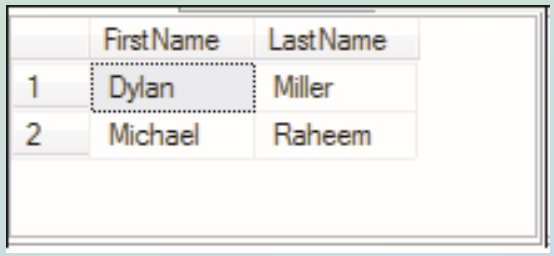
Để đảm bảo rằng các truy vấn bên ngoài có thể sử dụng các kết quả của các truy vấn lồng bên trong, từ khóa `IN` sẽ được sử dụng.

Làm việc với truy vấn đa trị 2-4

- Dưới đây là đoạn code minh họa cho ví dụ:

```
SELECT FirstName, LastName FROM Person.Person
WHERE Person.Person.BusinessEntityID IN (SELECT BusinessEntityID
FROM HumanResources.Employee WHERE JobTitle ='Research and Development
Manager');
```

- Truy vấn con ở đây lấy `BusinessEntityID` từ bảng `HumanResources.Employee` các bản ghi có tiêu đề công việc 'Research and Development Manager'.
- Kết quả này sẽ được truyền cho truy vấn bên ngoài, để lấy ra các bản ghi từ bảng `Person.Person` có `BusinessEntityID` khớp(match) với các `BusinessEntityID` trong kết quả của truy vấn con.
- Cuối cùng, các bản ghi khớp(macth) sẽ lấy họ(first name) và tên (last name) để hiện thị.
- Kết quả sẽ hiện thị như trong hình dưới đây:



	FirstName	LastName
1	Dylan	Miller
2	Michael	Raheem

Làm việc với truy vấn đa trị 3-4

- Một số điểm cần nhớ khi sử dụng truy vấn con:

Các kiểu dữ liệu ntext, text, và image không được sử dụng trong phần danh sách(list) trong câu lệnh SELECT của truy vấn con.

Khi sử dụng phép toán so sánh (mà không có từ khóa ANY, ALL theo sau) với kết quả của truy vấn con, thì phần list trong câu lệnh SELECT của truy vấn con phải là một biểu thức, hoặc tên một cột.

Do truy vấn con được sử dụng cùng với các phép toán so sánh (mà không có các từ khóa ALL hoặc ANY theo sau) chỉ trả về một giá trị, nên không thể đưa(include) và mệnh đề GROUP BY và HAVING

Không thể sử dụng từ khóa DISTINCT với truy con được đặt trong GROUP BY.

Chỉ có thể dùng ORDER BY trong truy vấn con nếu cũng có sử dụng TOP.

Làm việc với truy vấn đa trị 4-4

- Bên cạnh truy vấn con trả về đơn trị và truy vấn con trả về đa trị, bạn cũng có thể chọn giữa truy vấn con độc lập (self-contained subqueries) và truy vấn con có liên quan(correlated subqueries). Chúng được định nghĩa như sau:

Các truy vấn con độc lập (Self-contained subqueries)

- Là những câu truy vấn được viết độc lập (standalone), không phụ thuộc vào truy vấn bên ngoài.
- Một truy vấn con độc lập được xử lý mỗi khi truy vấn bên ngoài chạy và truyền kết quả của nó tới truy vấn bên ngoài (outer query).

Truy vấn Correlated (Correlated subqueries)

- Các truy vấn này tham chiếu đến một hoặc nhiều cột của truy vấn bên ngoài, do vậy nó phụ thuộc vào truy vấn bên ngoài.
- Truy vấn con correlated không thể chạy tách riêng khỏi truy vấn bên ngoài

SQL Server 2012

Từ khóa EXISTS 1-2

- Từ khóa `EXISTS` được sử dụng với truy vấn con để kiểm tra xem truy vấn có trả về bất kỳ dòng dữ liệu nào hay không.
- `EXISTS` cho kết quả là `FALSE` nếu truy vấn con không trả về bất kỳ dòng dữ liệu nào, ngược lại trả về `TRUE`.
- Cú pháp sử dụng từ khóa `EXISTS` như sau:

Cú pháp:

```
SELECT <ColumnName> FROM <table>
WHERE [NOT] EXISTS
(
<Subquery_Statement>
)
```

Trong đó,

`Subquery_Statement`: chỉ ra truy vấn con.

Từ khóa EXISTS 2-2

- Đoạn code dưới đây minh họa sử dụng từ khóa EXISTS:

```
SELECT FirstName, LastName FROM Person.Person AS A
WHERE EXISTS (SELECT *
FROM HumanResources.Employee As B
WHERE JobTitle ='Research and
Development Manager' AND A.BusinessEntityID=B.BusinessEntityID);
```

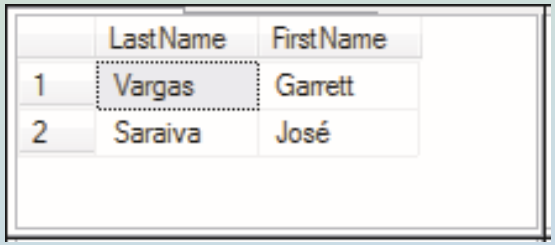
- Truy vấn con lấy tất cả các bản ghi với tiêu đề công việc là `'Research and Development` Manager' và `BusinessEntityId` của nó được dùng để so sánh trong bảng Person.
- Nếu không có bản ghi nào thỏa mãn cả hai điều kiện này, các truy vấn con bên trong sẽ không trả lại bất kỳ dòng nào.
- Như vậy, trong trường hợp này, `EXISTS` sẽ trả về false và truy vấn bên ngoài cũng sẽ không trả lại bất kỳ dòng nào.
- Tuy nhiên, đoạn code trả về hai dòng do các điều kiện đã cho được thỏa mãn.
- Tương tự như vậy, có thể sử dụng từ khóa `NOT EXISTS`

Các truy vấn lồng nhau (nested)

- Một truy vấn con được định nghĩa bên trong một truy vấn con khác được gọi là truy vấn con lồng (nested subquery).
- Khi bạn muốn lấy và hiển thị tên của mọi người đến từ Canada.
- Không có cách náo để lấy trực tiếp thông tin này, là do bảng `Sales.SalesTerritory` không có quan hệ với bảng `Person.Person`.
- Vì vậy một truy vấn con lồng được sử dụng ở đây như trong đoạn code dưới đây:

```
SELECT LastName, FirstName
FROM Person.Person
WHERE BusinessEntityID IN
        (SELECT BusinessEntityID
         FROM Sales.SalesPerson
         WHERE TerritoryID IN
                 (SELECT TerritoryID
                  FROM Sales.SalesTerritory
                  WHERE Name='Canada')
        )
```

- Kết quả được thể hiện trong hình sau:



	LastName	FirstName
1	Vargas	Garrett
2	Saraiva	José

SQL Server 2012

Truy vấn tương quan (Correlated) 1-2

Là một truy vấn con có tham chiếu tới các giá trị của truy vấn cha, kết quả trả về của nó của nó phụ thuộc vào giá trị của truy vấn bên ngoài.

Điều này cho thấy truy vấn con được thực hiện lặp lại nhiều lần, mỗi lần cho một dòng của truy vấn bên ngoài

Giả sử bạn muốn lấy tất cả các BusinessEntityID của mọi người có thông tin liên lạc được chỉnh sửa trước năm 2012.

- Đoạn code dưới đây minh họa sử dụng truy vấn con tương quan:

```
SELECT e.BusinessEntityID
FROM Person.BusinessEntityContact e
WHERE e.ContactTypeID IN (
                          SELECT c.ContactTypeID
                          FROM Person.ContactType c
                          WHERE YEAR(e.ModifiedDate)
                                              >=2012
)
```

Truy vấn tương quan (Correlated) 2-2

- Trong đoạn code trên, truy vấn con bên trong lấy ra `ContactTypeId` từ bảng `ContactType` của tất cả các cá nhân(person) có ngày sửa đổi(ModifiedDate) từ trước năm 2012.
- Các kết quả này sau đó được truyền cho truy vấn bên ngoài, để truy vấn bên ngoài lấy ra các `BusinessEntityID` từ bảng `Person.BusinessEntityContact` có `ContactTypeId` khớp với các `ContactTypeId` được truy vấn con lấy ra.
- Hình dưới đây cho thấy một phần của kết quả:

```
SELECT e.Busine
FROM Person.Bus
WHERE e.Contact



)
```

	BusinessEntityID
1	292
2	294
3	296
4	298
5	300
6	302
7	304

Phép ghép nối - Joins 1-3

- Các phép ghép nối được sử dụng để lấy dữ liệu từ hai hoặc nhiều bảng dựa trên mối quan hệ luận lý(logical relationship) giữa các bảng.
- Nó định nghĩa cách thức hai bảng được quan hệ với nhau bằng cách :

Chỉ ra cột từ mỗi bảng được sử dụng cho việc ghép nối. Một ghép nối đặc thù chỉ ra khóa ngoại từ một bảng kết hợp(associated) với khóa chính từ một bảng khác.

Toán tử luận lý =, <> được sử dụng được sử dụng để so sánh giá trị từ các cột.

- Ghép nối có thể được chỉ ra trong các mệnh đề `FROM` hoặc `WHERE`.
- Cú pháp của câu lệnh `JOIN` như sau:

```
SELECT <ColumnName1>, <ColumnName2>...<ColumnNameN>
FROM Table_A AS Table_Alias_A
JOIN
Table_B AS Table_Alias_B
ON
Table_Alias_A.<CommonColumn> = Table_Alias_B.<CommonColumn>
```

Trong đó,

`<ColumnName1>, <ColumnName2>`: một danh sách cột được hiển thị

Phép ghép nối - Joins 2-3

`Table_A`: là tên của bảng phía bên trái của từ khóa `JOIN`.
`Table_B`: là tên của bảng phía bên phải của từ khóa `JOIN`.
`AS Table_Alias`: là cách đặt tên bí danh cho bảng. Có thể xác định một bí danh cho bảng trong truy vấn để không cần phải dùng đến tên đầy đủ của nó (Tên đầy đủ của bảng thường dài, nó gồm tên lược đồ với tên bảng).
`<CommonColumn>`: là cột chung có trong hai cả bảng tham gia ghép nối. Trong trường hợp đó, ghép nói chỉ thành công nếu các cột có các giá trị khớp nhau.

- Giả sử bạn muốn liệt kê `first names`, `last names` và `JobTitle` của nhân viên từ bảng `HumanResources.Employee` và `Person.Person`.
- Để lấy thông tin từ hai bảng, bạn cần ghép nối chúng dựa trên cột chung `BusinessEntityID` như trong đoạn code dưới đây:

```
SELECT A.FirstName, A.LastName, B.JobTitle
FROM Person.Person A JOIN HumanResources.Employee B
                ON A.BusinessEntityID = B.BusinessEntityID;
```

Phép ghép nối - Joins 3-3

- Trong ví dụ trên, tên các bảng `HumanResources.Employee` và `Person.Person` được đặt bí danh là `A` và `B`. Chúng được ghép với nhau dựa trên các định danh thực thể business trong cột `BusinessEntityID`.
- Sau đó câu lệnh `SELECT` lấy các cột mong muốn qua các bí danh.
- Kết quả được thể hiện trong hình sau:

	FirstName	LastName	JobTitle
1	Ken	Sánchez	Chief Executive Officer
2	Terri	Duffy	Vice President of Engineering
3	Roberto	Tamburello	Engineering Manager
4	Rob	Walters	Senior Tool Designer
5	Gail	Erickson	Design Engineer
6	Jossef	Goldberg	Design Engineer
7	Dylan	Miller	Research and Development Manager

Phép nối Inner Join

Là phép ghép nối không phân biệt thứ tự, (bảng nào viết trước hay viết sau không ảnh hưởng đến kết quả truy vấn. Phép ghép nối chỉ kết hợp các bản ghi từ hai bảng được so khớp qua cột chung.

- Cú pháp sử dụng inner join như sau:

```
SELECT <ColumnName1>, <ColumnName2>...<ColumnNameN> FROM
Table_A AS Table_Alias_A INNER JOIN Table_B AS Table_Alias_B
ON Table_Alias_A.<CommonColumn> = Table_Alias_B.<CommonColumn>
```

- Đoạn code dưới đây minh họa sử dụng inner join:

```
SELECT A.FirstName, A.LastName, B.JobTitle
FROM Person.Person A INNER JOIN HumanResources.Employee B
                ON A.BusinessEntityID = B.BusinessEntityID;
```

Outer Join

Phép ghép nối Outer join là câu lệnh ghép nối trả về tất cả các dòng của một trong các bảng được chỉ ra trong mệnh đề FROM, cùng với các dòng thỏa mãn các điều kiện trong mệnh đề WHERE hoặc HAVING của câu lệnh SELECT.

- Có hai loại outer join sau được sử dụng phổ biến:

Left Outer Join

Right Outer Join

SQL Server 2012

Left Outer Join 1-3

Phép ghép nối Left outer join trả về toàn bộ các bản ghi có trong bảng bên trái và các bản ghi từ bảng bên phải khớp với điều kiện ghép nối được chỉ ra sau từ khóa ON.

- Cú pháp sử dụng `left outer join` như sau:

```
SELECT <ColumnList>
FROM Table_A AS Table_Alias_A LEFT OUTER JOIN Table_B AS
Table_Alias_B
ON Table_Alias_A.<CommonColumn> = Table_Alias_B.<CommonColumn>
```

- Giả sử bạn muốn lấy tất cả các id của khách hàng từ bảng `Sales. Bảng Customers` và thông tin hóa đơn như ngày giao hàng (ship dates) và ngày đáo hạn (due dates), lấy cả các khách hàng không có bất kỳ hóa đơn nào.
- Vì số lượng bản ghi sẽ rất lớn, nên giới hạn chỉ những đơn đặt hàng được đặt trước năm 2012.

Left Outer Join 2-3

- Đoạn code dưới đây minh họa thực hiện điều đó bằng left outer join:

```
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;
```

- Trong truy vấn trên, In the query, left outer join được đặt giữa các bảng `Sales.Customer` và `Sales.SalesOrderHeader`.
- Các bảng được ghép dựa trên các id của khách hàng.
- Trong trường hợp này, tất cả cá bản ghi của bảng bên trái là `Sales.Customer` và các bản ghi của bảng bên phải khớp với điều kiện ghép được chỉ ra sau từ khóa `ON` được trả về.

Left Outer Join 3-3

- Hình dưới đây thể hiện kết quả trả về

	CustomerID	DueDate	ShipDate
3...	18178	2008-08-12 00:00:00.000	2008-08-07 00:00:00.000
3...	13671	2008-08-12 00:00:00.000	2008-08-07 00:00:00.000
3...	11981	2008-08-12 00:00:00.000	2008-08-07 00:00:00.000
3...	18749	2008-08-12 00:00:00.000	2008-08-07 00:00:00.000
3...	15251	2008-08-12 00:00:00.000	2008-08-07 00:00:00.000
3...	15868	2008-08-12 00:00:00.000	2008-08-07 00:00:00.000
3...	18759	2008-08-12 00:00:00.000	2008-08-07 00:00:00.000
3...	215	NULL	NULL
3...	46	NULL	NULL
3...	169	NULL	NULL
3...	507	NULL	NULL
3...	630	NULL	NULL

- Như trong hình trình bày kết quả, một số các bản ghi được hiện thị với cột `DueDate` và `ShipDate` là `NULL`.
- Điều này là do một số khách hàng không có hóa đơn đặt hàng nào, do vậy các bản ghi của họ được hiển thị với các ngày là `NULL`.

Right Outer Join 1-2

- Right outer join lấy tất cả các bản ghi có trong bảng bên phải (bảng thứ 2) , bất kể là nó có khớp với dữ liệu nào trong bảng bên trái hay không.
- Cú pháp sử dụng right outer join như sau:

```
SELECT <ColumnList>
FROM Left_Table_Name AS Table_A AS Table_Alias_A
RIGHT OUTER JOIN
Table_B AS Table_Alias_B
ON
Table_Alias_A.<CommonColumn> = Table_Alias_B.<CommonColumn>
```

Right Outer Join 2-2

- Giả sử bạn muốn lấy toàn bộ tên của các sản phẩm từ bảng `Product` và tất cả các id của hóa đơn bán tương ứng từ bảng `SalesOrderDetail`, kể cả các bản ghi sản phẩm không khớp với dữ liệu có trong bảng `SalesOrderDetail`.
- Đoạn code sau đây minh họa sử dụng right outer join:

```
SELECT P.Name, S.SalesOrderID
FROM Sales.SalesOrderDetail S RIGHT OUTER JOIN
Production.Product P
ON P.ProductID = S.ProductID;
```

- Trong đoạn code trên, tất cả các bản ghi trong bảng `Product` sẽ được hiển thị cho dù là chúng có được bán hay không.

Ghép nối đệ qui (Self-Join) 1-2

- Ghép nối đệ qui (self-join) được sử dụng để lấy các bản ghi có quan hệ với các bản ghi khác trong cùng một bảng. Trong ghép nối đệ qui, một bảng ghép nối với chính nó.
- Giả sử bảng **`Employee`** trong cơ sở dữ liệu Sterling có cột **`mgr_id`** biểu thị cho các nhà quản lý mà các nhân viên của họ sẽ báo cáo tới.
- Giả sử bảng đã có các bản ghi phù hợp được chèn.
- Một người quản lý cũng là một nhân viên. Có nghĩa là **`mgr_id`** trong bảng chính là **`emp_id`** của bảng nhân viên.
- Ví dụ, **`Anabela`** là một nhân viên có **`emp_id`** là **`ARD36773F`** nhưng **`Anabela`** cũng là giám đốc của Victoria, Palle, Karla, và các nhân viên khác như trong bảng dưới đây:

	emp_id	fname	minit	lname	job_id	job_lvl	pub_id	hire_date	mgr_id
1	PMA42628M	Paolo	M	Accorti	13	35	0877	1992-08-27 00:00:00.000	POK93028M
2	PSA89086M	Pedro	S	Afonso	14	89	1389	1990-12-24 00:00:00.000	POK93028M
3	VPA30890F	Victoria	P	Ashworth	6	140	0877	1990-09-13 00:00:00.000	ARD36773F
4	H-B39728F	Helen		Bennett	12	35	0877	1989-09-21 00:00:00.000	POK93028M
5	L-B31947F	Lesley		Brown	7	120	0877	1991-02-13 00:00:00.000	ARD36773F
6	F-C16315M	Francisco		Chang	4	227	9952	1990-11-03 00:00:00.000	MAS70474F
7	PTC11962M	Philip	T	Cramer	2	215	9952	1989-11-11 00:00:00.000	MAS70474F
8	A-C71970F	Aria		Cruz	10	87	1389	1991-10-26 00:00:00.000	POK93028M
9	AMD15433F	Ann	M	Devon	3	200	9952	1991-07-16 00:00:00.000	MAS70474F
10	ARD36773F	Anabela	R	Doming...	8	100	0877	1993-01-27 00:00:00.000	NULL
11	PHF38899M	Peter	H	Franken	10	75	0877	1992-05-17 00:00:00.000	POK93028M
12	PXH22250M	Paul	X	Henriot	5	159	0877	1993-08-19 00:00:00.000	MAS70474F

Ghép nối đệ qui 2-2

- Để lấy danh sách tên các nhà quản lý cùng với các chi tiết khác, bạn có thể sử dụng ghép nối đệ qui để ghép nối bảng employee với chính nó, sau đó lấy ra các bản ghi mong muốn.
- Đoạn code dưới đây minh họa phép ghép nối đệ qui:

```
SELECT TOP 7 A.fname + ' ' + A.lname AS 'Employee Name', B.fname
+ ' '+B.lname AS 'Manager'

FROM Employee AS A INNER JOIN Employee AS B ON A.mgr_id =
B.emp_id
```

- Trong đoạn code trên bảng, the `Employee` được ghép nối với chính nó dựa trên các cột `mgr_id` và `emp_id`.
- Hình dưới đây thể hiện kết quả của đoạn mã trên:

	Employee Name	Manager
1	Paolo Accorti	Pirkko Koskitalo
2	Pedro Afonso	Pirkko Koskitalo
3	Victoria Ashworth	Anabela Domingues
4	Helen Bennett	Pirkko Koskitalo
5	Lesley Brown	Anabela Domingues
6	Francisco Chang	Margaret Smith
7	Philip Cramer	Margaret Smith

SQL Server 2012

Câu lệnh MERGE 1-6

- Câu lệnh `MERGE` cho phép bạn duy trì (maintain) bảng đích (target table) dựa trên các điều kiện nào đó trên bảng nguồn (source table) bằng việc dùng một câu lệnh đơn.
- Bạn có thể thực hiện các hành động sau trong một câu lệnh `MERGE`:

Chèn một dòng mới từ bảng nguồn nếu dòng không có(missing) trong bảng đích

Cập nhật(Update) dòng ở bảng đích nếu bản ghi có tồn tại trong bảng nguồn.

Xóa(Delete) dòng bảng đích nếu dòng không có (missing) trong bảng nguồn.

- Ví dụ, giả sử bảng `Products` duy trì các bản ghi của tất cả các sản phẩm.
- Một bảng **`NewProducts`** duy trì các bản ghi của sản phẩm mới.
- Bạn muốn cập nhật bảng Products với các bản ghi từ bảng **`NewProducts`**.

Câu lệnh MERGE 2-6

- Ở đây bảng **`NewProducts`** là bảng nguồn và bảng Products là bảng đích.
- BảngProducts có chứa các bản ghi của các sản phẩm hiện có với dữ liệu được cập nhật và các sản phẩm mới.
- Hình dưới đây trình bày dữ liệu trong hai bảng:

Products

	ProductID	Name	Type	PurchaseDate
1	101	Rivets	Hardware	2012-12-01
2	102	Nuts	Hardware	2012-12-01
3	103	Washers	Hardware	2011-01-01
4	104	Rings	Hardware	2013-01-15
5	105	Paper Clips	Stationery	2013-01-01

NewProducts

	ProductID	Name	Type	PurchaseDate
1	102	Nuts	Hardware	2012-12-01
2	103	Washers	Hardware	2011-01-01
3	107	Rings	Hardware	2013-01-15
4	108	Paper Clips	Stationery	2013-01-01

Câu lệnh MERGE 3-6

- Giả sử bạn muốn:
  - So sánh last và first name của các khách hàng từ hai bảng nguồn và đích.
  - Cập nhật thông tin khách hàng trong bảng đích nếu last và first name trong bảng đích trùng bảng nguồn
  - Chèn các bản ghi mới vào bảng đích nếu last và first name trong bảng nguồn không có trong bảng đích
  - Xóa các bản ghi đang có trong bảng đích nếu last và first name không trùng với bảng nguồn
- MERGE cũng cho phép bạn tùy chọn việc hiển thị các bản ghi được chèn(insert), được cập nhật (update), được xóa bằng mệnh đề `OUTPUT`.
- Cú pháp câu lệnh `MERGE` như sau:

```
MERGE target_table
USING source_table
ON match_condition
WHEN MATCHED THEN UPDATE SET Col1 = val1 [, Col2 = val2...]
WHEN [TARGET] NOT MATCHED THEN INSERT (Col1 [,Col2...] VALUES (Val1 [,
Val2...])
WHEN NOT MATCHED BY SOURCE THEN DELETE
[OUTPUT $action, Inserted.Col1, Deleted.Col1,...] ;
```

Câu lệnh MERGE 4-6

Trong đó,

`target_table`: là bảng mà ở đó dữ liệu được thực hiện thay đổi.

`source_table`: là bảng mà từ đó các dòng sẽ được thực hiện thêm, sửa, xóa, tới bảng đích.

`match_conditions`: là điều kiện ghép nối (JOIN) và các toán tử so sánh bất kỳ .`match_condition`.

`MATCHED`: là true nếu một dòng trong bảng đích (`target_table`) và bảng `source_table` trùng nhau.

`NOT MATCHED`: là true nếu một dòng trong bảng nguồn(`source_table`) không tồn tại trong bảng đích.

`SOURCE NOT MATCHED`: là true nếu một dòng có trong bảng đích (`target_table`) nhưng không có trong bảng nguồn (`source_table`).

`OUTPUT`: là mệnh đề tùy chọn cho phép hiển thị các bản ghi vừa được thêm/sửa/xóa trong bảng đích (`target_table`).

- Câu lệnh `MERGE` kết thúc bởi dấu chấm phẩy (;).

Câu lệnh MERGE 5-6

- Đoạn code dưới đây minh họa cách dùng câu lệnh `MERGE`. Nó sử dụng csdl **`Sterling`** :

```
MERGE INTO Products AS P1
USING NewProducts AS P2
ON P1.ProductId = P2.ProductId
WHEN MATCHED THEN
UPDATE SET
P1.Name = P2.Name,
P1.Type = P2.Type,
P1.PurchaseDate = P2.PurchaseDate
WHEN NOT MATCHED THEN
INSERT (ProductId, Name, Type, PurchaseDate)
VALUES (P2.ProductId, P2.Name, P2.Type, P2.PurchaseDate)
WHEN NOT MATCHED BY SOURCE THEN
DELETE
OUTPUT $action, Inserted.ProductId, Inserted.Name, Inserted.Type,
Inserted.PurchaseDate, Deleted.ProductId,Deleted.Name, Deleted.Type,
Deleted.PurchaseDate;
```

Câu lệnh MERGE 6-6

- Hình dưới đây thể hiện kết quả của đoạn code:

	$action	ProductId	Name	Type	PurchaseDate	ProductId	Name	Type	PurchaseDate
1	INSERT	107	Rings	Hardware	2013-01-15	NULL	NULL	NULL	NULL
2	INSERT	108	Paper Clips	Stationery	2013-01-01	NULL	NULL	NULL	NULL
3	DELETE	NULL	NULL	NULL	NULL	101	Rivets	Hardware	2012-12-01
4	UPDATE	102	Nuts	Hardware	2012-12-01	102	Nuts	Hardware	2012-12-01
5	UPDATE	103	Washers	Hardware	2011-01-01	103	Washers	Hardware	2011-01-01
6	DELETE	NULL	NULL	NULL	NULL	104	Rings	Hardware	2013-01-15
7	DELETE	NULL	NULL	NULL	NULL	105	Paper Clips	Stationery	2013-01-01

- Bảng **`NewProducts`** là bảng nguồn (source table) và bảng **`Products`** là bảng đích (target table).
- Điều kiện so khớp là cột **`ProductId`** của cả hai bảng
- Nếu điều kiện so khớp là sai [false (`NOT MATCHED`)], thì các bản ghi mới được chèn tới bảng đích (target table).
- Nếu điều kiện so khớp là đúng [true (`MATCHED`)] , thì các bản ghi được cập nhật(updated) vào bảng đích (target table) từ bảng nguồn (source table).
- Nếu các bản ghi hiện có trong bảng đích (target table) không khớp với những bảng nguồn (`NOT MATCHED BY SOURCE`), thì sẽ bị xóa khỏi bảng đích.
- Câu lệnh cuối cùng hiển thị một báo cáo các dòng được inserted/updated/deleted như trong hình minh họa kết quả.

SQL Server 2012

Common Table Expressions (CTEs) 1-5

CTE là đối tượng lưu một tập kết quả tạm thời được định nghĩa và sử dụng trong phạm vi của câu lệnh `SELECT`, `INSERT`, `UPDATE`, `DELETE`, hoặc `CREATE VIEW` đang được thực thi .

CTE là một biểu thức có tên được định nghĩa trong một truy vấn.

CTE được định nghĩa ở phần bắt đắt đầu của một truy vấn và có thể được tham chiếu nhiều lần trong câu truy vấn chứa nó(outer query).

CTE tự tham chiếu đến chính nó được gọi là CTE đệ quy.

Lợi thế chính của CTEs được cải thiện khả năng đọc và dễ dàng trong việc bảo trì các truy vấn phức tạp.

- Cú pháp để tạo CTE như sau

```
WITH <CTE_name>
AS ( <CTE_definition> )
```

Common Table Expressions (CTEs) 2-5

- Đoạn code dưới đây lấy và hiển thị số lượng khách đặt hàng theo năm trong bảng `Sales.SalesOrderHeader`:

```
WITH CTE_OrderYear
AS
(
SELECT YEAR(OrderDate) AS OrderYear, CustomerID
FROM Sales.SalesOrderHeader
)
SELECT OrderYear, COUNT(DISTINCT CustomerID) AS CustomerCount
FROM CTE_OrderYear
GROUP BY OrderYear;
```

- Trong đó, **`CTE_OrderYear`** là tên được chỉ ra cho CTE.
- Định nghĩa CTE được bắt đầu bằng từ khóa `WITH...AS`.
- Sau đó, CTE được sử dụng trong câu lệnh `SELECT` để lấy và hiển thị kết quả mong muốn. Hình dưới đây thể hiện kết quả của đoạn code trên:

	OrderYear	CustomerCount
1	2007	9864
2	2008	11844
3	2005	1216
4	2006	3094

SQL Server 2012

Common Table Expressions (CTEs) 3-5

- Các nguyên tắc cần nhớ khi định nghĩa CTE:

CTE bị giới hạn trong phạm vi thực thi của câu truy vấn bên ngoài (outer query). Do vậy, khi truy vấn bên ngoài kết thúc, vòng đời của CTE sẽ kết thúc.

Bạn cũng cần xác định tên cho một CTE, và xác định các tên duy nhất cho mỗi cột được tham chiếu trong mệnh đề SELECT của CTE.

Cũng có thể sử dụng các bí danh nội tuyến(inline) và bên ngoài(external) cho các cột trong CTE..

Một định nghĩa CTE có thể được tham chiếu nhiều lần trong cùng truy vấn.

Nhiều CTE cũng có thể được định nghĩa trong cùng mệnh đề WITH.

Common Table Expressions (CTEs) 4-5

- Đoạn code sau minh họa định nghĩa 2 CTE bằng mệnh đề `WITH`:

```
WITH CTE_Students
AS
(
Select StudentCode, S.Name,C.CityName, St.Status
FROM Student S
INNER JOIN City C
ON S.CityCode = C.CityCode
INNER JOIN Status St
ON S.StatusId = St.StatusId)
,
StatusRecord -- This is the second CTE being defined
AS
(
SELECT Status, COUNT(Name) AS CountofStudents
FROM CTE_Students
GROUP BY Status
)
SELECT * FROM StatusRecord
```

Common Table Expressions (CTEs) 5-5

- Đoạn code minh họa định nghĩa 2 CTE bằng mệnh đề WITH.
- Đoạn code giả sử có ba bảng với tên Student, City, và Status được tạo.
- Giả sử đã có một số bản ghi được chèn vào tất cả ba bảng, kết quả cho như hình sau:

	Status	CountofStudents
1	Failed	2
2	Passed	2

SQL Server 2012

Toán tử UNION 1-2

Kết quả từ hai câu lệnh truy vấn khác nhau có thể được kết hợp thành một tập kết quả duy nhất bằng toán tử UNION.

Các câu lệnh truy vấn phải tương thích về kiểu và số lượng các cột.

Tên các cột trong mỗi câu lệnh có thể khác nhau, nhưng kiểu dữ liệu phải tương thích(giống nhau).

- Cú pháp toán tử `UNION` như sau:

```
Query_Statement1
UNION [ALL]
Query_Statement2
```

Trong đó,

`Query_Statement1` và `Query_Statement2` là các câu lệnh `SELECT`.

Toán tử UNION 2-2

- Đoạn code sau minh họa sử dụng toán tử UNION:

```
SELECT Product.ProductId FROM Production.Product
UNION
SELECT ProductId FROM Sales.SalesOrderDetail
```

- Nó sẽ liệt kê tất cả id của các sản phẩm của hai bảng khớp với nhau.
- Nếu có thêm mệnh đề `ALL`, sẽ lấy toàn bộ các bản ghi của hai truy vấn bao gồm cả các bản ghi trùng nhau.

```
SELECT Product.ProductId FROM Production.Product
UNION ALL
SELECT ProductId FROM Sales.SalesOrderDetail
```

- Mặc định, toán tử `UNION` bỏ đi các bản ghi trùng nhau trong tập kết quả..
- Tuy nhiên, nếu bạn sử dụng mệnh đề ALL với toán tử `UNION`, thì tấ cả các dòng được trả về.
- Ngoài UNION, còn có các toán tử khác kết hợp dữ liệu từ nhiều bảng như `INTERSECT` và `EXCEPT`.

Toán tử INTERSECT 1-2

Toán tử INTERSECT được sử dụng với hai câu lệnh truy vấn để trả về tập các dòng chung duy nhất của cả hai câu lệnh truy.

- Cú pháp toán tử INTERSECT như sau:

```
Query_statement1
INTERSECT
Query_statement2
```

Trong đó,

Query_Statement1 và Query_Statement2 là các câu lệnh truy vấn.

- Đoạn code dưới đây minh họa sử dụng toán tử INTERSECT:

```
SELECT Product.ProductId FROM Production.Product
INTERSECT
SELECT ProductId FROM Sales.SalesOrderDetail
```

Toán tử INTERSECT 2-2

- Các nguyên tắc cơ bản sử dụng `INTERSECT` như sau:

Số lượng cột và thứ tự của các cột được chỉ ra trong hai lệnh truy vấn phải giống nhau.

Kiểu dữ liệu của các cột phải phù hợp.

- Kết quả của việc lấy giao (intersection) của hai bảng `Production.Product` và `Sales.SalesOrderDetail` chỉ có các id khớp với các bản ghi trong bảng `Production.Product`.

SQL Server 2012

Toán tử EXCEPT 1-2

Toán tử EXCEPT trả về các tất cả các dòng không trùng lặp chỉ có trong kết quả của truy vấn bên trái và không có trong kết quả truy vấn bên phải của toán tử EXCEPT.

- Cú pháp toán tử EXCEPT(trừ) như sau:

```
Query_statement1
EXCEPT
Query_statement2
```

- Hai quy tắc áp dụng cho toán tử INTERSECT cũng được áp dụng cho toán tử EXCEPT.

Toán tử EXCEPT 2-2

- Đoạn code sau đây minh họa sử dụng toán tử `EXCEPT`:

```
SELECT Product.ProductId FROM Production.Product
EXCEPT
SELECT ProductId FROM Sales.SalesOrderDetail
```

- Nếu thứ tự hai bảng trong ví dụ này được thay đổi, thì chỉ có các dòng từ bảng `Production` được trả về.

- Bảng `Product` không có dòng nào khớp với các dòng hiện có trong bảng `Sales.SalesOrderDetail.`

- Do đó, toán tử `EXCEPT` chọn tất cả các dòng từ bảng đầu tiên ngoại trừ những dòng có xuất hiện trong bảng hai.

- Vì vậy khi bạn sử dụng toán tử `EXCEPT`, thứ tự của hai bảng trong các truy vấn là quan trọng.

SQL Server 2012

Toán tử PIVOT 1-6

- Cú pháp ngắn gọn của toán `PIVOT` như sau:

```
SELECT <non-pivoted column>,
    [first pivoted column] AS <column name>,
    [second pivoted column] AS <column name>,
    ...
    [last pivoted column] AS <column name>
FROM
    (<SELECT query that produces the data>)
    AS <alias for the source query>
PIVOT
(
    <aggregation function>(<column being aggregated>)
FOR
[<column that contains the values that will become column headers>]
    IN ( [first pivoted column], [second pivoted column],
    ... [last pivoted column])
) AS <alias for the pivot table>
<optional ORDER BY clause>;
```

Toán tử PIVOT 2-6

Trong đó,

`table_source`: là bảng hoặc biểu thức bảng.

`aggregate_function`: là hàm thống kê do người dùng định nghĩa hoặc hàm dựng sẵn nhận một hoặc nhiều tham số đầu vào.

`value_column`: là cột giá trị của toán tử PIVOT.

`pivot_column`: là cột xoay (pivot) của toán tử PIVOT. Cột này phải là một kiểu có thể được chuyển kiểu mặc định(implicitly) hoặc ép(explicitly) sang kiểu `nvarchar()`.

`IN (column_list)`: là các giá trị trong cột `pivot_column` mà sẽ trở thành các tên cột của bảng kết quả (output table). Danh sách không được có chứa bất kỳ tên cột nào đã tồn tại trong bảng đầu vào `table_source` được pivot.

`table_alias`: là tên bí danh của bảng kết quả (output table).

- Kết quả của toán tử là bảng có chứa tất cả các cột của bảng `table_source` ngoại trừ các `pivot_column` và `value_column`.

- Các cột của bảng `table_source`, ngoại trừ các the `pivot_column` và `value_column`, được gọi là các cột nhóm (grouping columns) của toán tử pivot.

SQL Server 2012

Toán tử PIVOT 3-6

- Trong điều kiện đơn giản, để sử dụng toán tử PIVOT, bạn cần phải cung cấp ba thành phần cho toán tử:

Nhóm (Grouping)

- Trong mệnh đề `FROM`, các cột đầu vào phải được cung cấp.
- Toán tử `PIVOT` sử dụng các cột này để xác định những cột nào dùng cho việc nhóm dữ liệu để thống kê.

Phân tán (Spreading)

- Ở đây, một danh sách các giá trị được phân cách dấu phẩy xuất hiện dữ liệu nguồn được cung cấp, sẽ được sử dụng như các tiêu đề cột cho dữ liệu được pivot.

Tổng hợp (Aggregation)

- Một hàm tổng hợp như là hàm `SUM` được thực hiện trên các dòng đã được nhóm.

Toán tử PIVOT 4-6

- Đoạn code dưới đây không sử dụng toán tự `PIVOT` và minh họa thống kê bằng một `GROUP BY` đơn.
- Khi số lượng bản ghi lớn, kết quả được giới hạn đến 5 bằng các chỉ ra `TOP 5`.

```
SELECT TOP 5 SUM(SalesYTD) AS TotalSalesYTD, Name
FROM Sales.SalesTerritory
GROUP BY Name
```

- Kết quả được trình bày trong hình minh họa sau:

	TotalSalesYTD	Name
1	7887186.7882	Northwest
2	2402176.8476	Northeast
3	3072175.118	Central
4	10510853.8739	Southwest
5	2538667.2515	Southeast

Toán tử PIVOT 5-6

- Hiển thị top 5 năm doanh số bán hàng cho đến nay với tên lãnh thổ và được nhóm theo tên lãnh thổ.
- Vẫn truy vấn như vậy được viết lại trong đoạn mã sau đây bằng việc sử dụng PIVOT, do vậy dữ liệu được chuyển đổi từ hướng dựa theo hàng(row-based orientation) sang hướng dựa theo cột (column-based orientation).

```
-- Pivot table with one row and six columns
SELECT TOP 5 'TotalSalesYTD' AS GrandTotal,
[Northwest], [Northeast], [Central], [Southwest],[Southeast]
FROM
(SELECT TOP 5 Name, SalesYTD
FROM Sales.SalesTerritory
) AS SourceTable
PIVOT
(
SUM(SalesYTD)
FOR Name IN ([Northwest], [Northeast], [Central],
[Southwest], [Southeast])
) AS PivotTable;
```

Toán tử PIVOT 6-6

- Kết quả đoạn code được thể hiện trong hình dưới đây:

	GrandTotal	Northwest	Northeast	Central	Southwest	Southeast
1	TotalSalesYTD	7887186.7882	2402176.8476	3072175.118	10510853.8739	2538667.2515

- Như thể hiện trong hình, dữ liệu được chuyển đổi và tên các lãnh thổ(territory) bây giờ được chuyển thành tên các cột thay vì dòng. Điều này cải thiện khả năng đọc.
- Một thách thức lớn trong việc viết các truy vấn sử dụng PIVOT là cần cung cấp một danh sách cố định các thành phần phân tán(spreading elements) cho toán tử PIVOT, ví dụ như chỉ ra tên các lãnh thổ trong đoạn code.
- Nó sẽ không có tính khả thi hoặc thực tế để thực hiện điều này khi thành phần phân tán có số lượng lớn.
- Để khắc phục điều này, các nhà phát triển có thể sử dụng SQL động (dynamic SQL).
- SQL động(Dynamic SQL) cung cấp một biện pháp để xây dựng một chuỗi ký tự và được truyền cho SQL Server, được thông dịch như là một lệnh, và sau đó thực thi.

SQL Server 2012

Toán tử UNPIVOT 1-3

- `UNPIVOT` thực hiện thao tác ngược lại của PIVOT, bằng cách xoay các cột thành các hàng.
- Unpivoting không khôi phục lại dữ liệu gốc.
- UNPIVOT không có khả năng phân bổ các giá trị để trả về giá trị chi tiết ban đầu.
- Thay vì chuyển dòng thành các cột, kết quả unpivoting trong các cột được chuyển đổi thành các dòng.
- SQL Server cung cấp toán tử bảng UNPIVOT để trả về một hiển thị bảng (tabular) theo hướng dòng từ dữ liệu được pivot.
- Khi dữ liệu không pivoting, một hoặc nhiều cột được định nghĩa như là nguồn được chuyển đổi thành các dòng. .
- Dữ liệu trong các cột này được phân bổ(spread), hoặc phân chia(split), thành một hoặc nhiều dòng mới, tùy thuộc vào bao nhiêu cột đang được unpivoted.
- Để sử dụng toán tử `UNPIVOT` , bạn cần cung cấp ba thành phần như sau:

Các cột nguồn được dùng để unpivoted

Một tên cho cột mới, là cột sẽ hiển thị các giá trị được unpivoted

Một tên cho cột, là cột sẽ hiển thị các tên của các giá trị được unpivoted

Toán tử UNPIVOT 2-3

- Đoạn code dưới đây trình bày việc chuyển bảng pivot tạm thời thành bảng lâu dài(permanent) do đó cùng bảng có thể được sử dụng cho việc minh họa toán tử UNPIVOT:

```
-- Pivot table with one row and six columns
SELECT TOP 5 'TotalSalesYTD' AS GrandTotal,
[Northwest], [Northeast], [Central], [Southwest],[Southeast]
FROM
(SELECT TOP 5 Name, SalesYTD
FROM Sales.SalesTerritory
) AS SourceTable
PIVOT
(
SUM(SalesYTD)
FOR Name IN ([Northwest], [Northeast], Central], [Southwest],
[Southeast])
) AS PivotTable;
```

Toán tử UNPIVOT 3-3

- Đoạn code sau đây minh họa sử dụng toán tử UNPIVOT:

```
SELECT Name, SalesYTD FROM
(SELECT GrandTotal, Northwest, Northeast, Central, Southwest,
Southeast
 FROM TotalTable) P
UNPIVOT
(SalesYTD FOR Name IN
(Northwest, Northeast, Central, Southwest, Southeast)
)AS unpvt;
```

- Kết quả đoạn code trên được minh họa trong hình sau:

	GrandTotal	Northwest	Northeast	Central	Southwest	Southeast
1	TotalSalesYTD	7887186.7882	2402176.8476	3072175.118	10510853.8739	2538667.2515

SQL Server 2012

GROUPING SETS 1-3

Toán tử `GROUPING SETS` cho phép bạn thống kê theo nhiều nhóm cột, và cuối mỗi nhóm có thêm một dòng tổng cộng tùy chọn, được biểu diễn bằng cặp ngoặc `()`.

Sẽ hiệu quả khi sử dụng toán tử `GROUPING SETS` thay cho việc sử dụng nhiều `GROUP BY` với mệnh đề UNION, do sau này sẽ làm tăng thêm chi phí xử lý trên máy chủ cơ sở dữ liệu.

- Cú pháp toán tử `GROUPING SETS` như sau:

```
GROUP BY
GROUPING SETS ( <grouping set list> )
```

Trong đó,

`grouping set list`: bao gồm một hoặc nhiều cột, được phân cách bởi các dấu phẩy.

- Một cặp ngoặc tròn `()`, mà không chỉ ra tên bất kỳ cột nào là biểu thị cho grand total.

GROUPING SETS 2-3

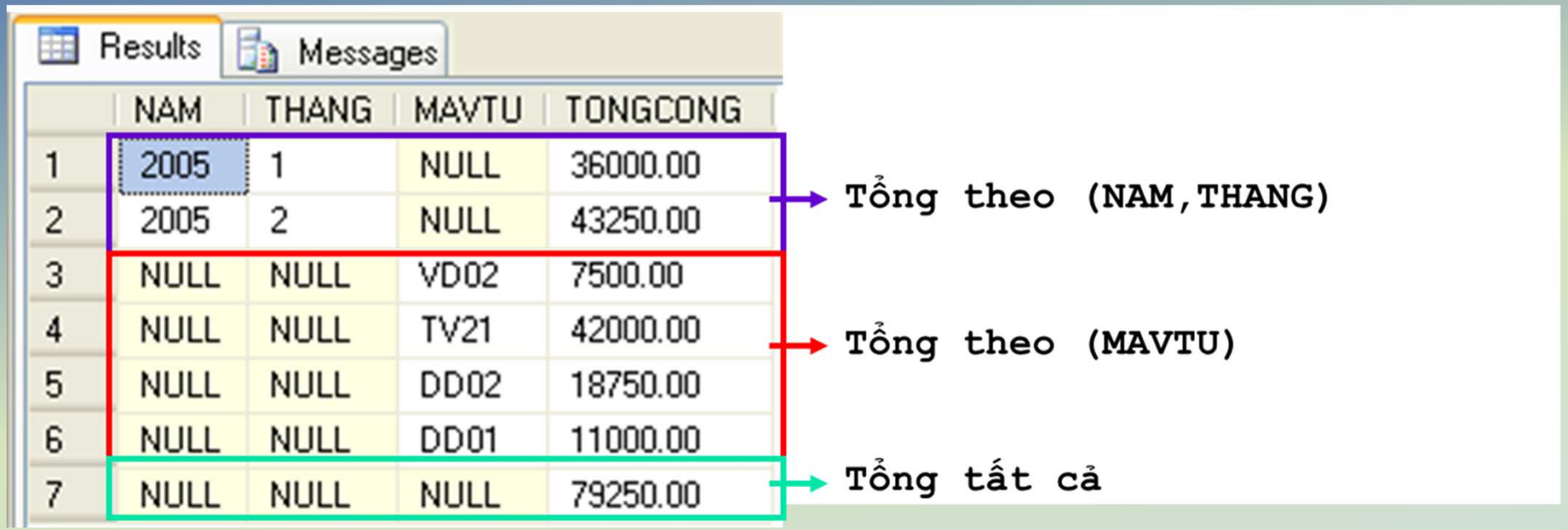
- Đoạn code minh họa sử dụng toán tử GROUPING SETS.

```
SELECT  NAM,THANG,MAVTU,SUM(THANHTIEN) AS TONGCONG
FROM VW_THANHTIEN_NHAP
GROUP BY
  GROUPING SETS
  (
      (MAVTU),
      (NAM,THANG),
      ()--means a grand total row
  )
Go
```

	NAM	THANG	MAVTU	THANHTIEN
▶	2005	1	DD01	2000.0000
	2005	1	DD02	3500.0000
	2005	1	DD01	500.0000
	2005	1	DD02	1750.0000
	2005	1	DD01	3750.0000
	2005	1	VD02	7500.0000
	2005	2	DD01	1250.0000
	2005	2	TV21	42000.0000
	2005	1	DD01	3500.0000
	2005	1	DD02	13500.0000
*	NULL	NULL	NULL	NULL

GROUPING SETS 3-3

- Kết quả đoạn code thể hiện trong hình dưới đây:



- Đoạn code sử dụng `GROUPING SETS` để hiển thị tổng thành tiền của mỗi vật tư và tổng thành tiền theo năm tháng.
- Giá trị `NULL` trong cột Mavtu chỉ ra tổng tiền cho mỗi tháng của năm.
- Giá trị `NULL` trong cả ba cột chỉ rằng đây là tổng cộng (grand total).

Tóm tắt

- Mệnh đề GROUP BY và cá hàm thống kê cho phép tổng hợp dữ liệu để trình bày thông tin tổng hợp.
- Truy vấn con cho phép tập kết quả của một câu lệnh SELECT này sử dụng để làm điều kiện cho một câu lệnh SELECT khác.
- Việc ghép nối (Joins) giúp bạn kết hợp cá cột dữ liệu từ hai hay nhiều bảng dựa trên mối quan hệ logic giữa các bảng.
- Các toán tử tập hợp và giúp bạn kết hợp/gộp các dòng dữ liệu từ từ hai hay nhiều bảng
- Mệnh đề con GROUPING SET của mệnh đề GROUP BY giúp cho việc tạo ra nhiều nhóm con trong duy nhất một truy vấn.