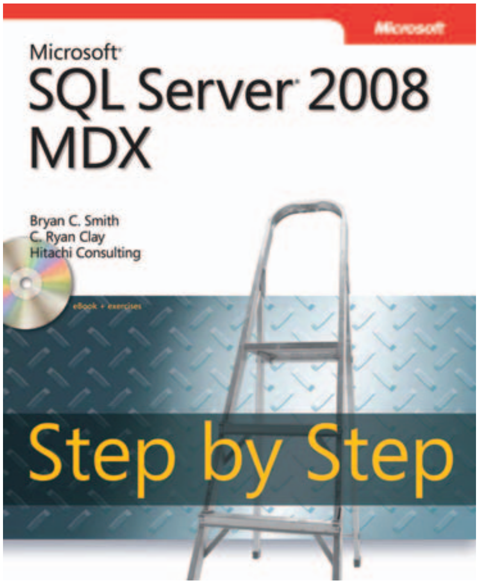


Microsoft® SQL® Server 2008 MDX Step by Step

Bryan C. Smith
C. Ryan Clay
Hitachi Consulting

To learn more about this book, visit Microsoft Learning at
http://www.microsoft.com/learning/en/us/Books/12914.aspx

9780735626188

Table of Contents

Part I MDX Fundamentals

4 Working with Sets 61

5 Working with Expressions 91

Part II MDX Functions

6 Building Complex Sets 123

Part III MDX Applications

Chapter 9

Working with Time

After completing this chapter, you will be able to:

- Explain the requirements for effective time-based analysis in Analysis Services
- Employ MDX functions to calculate common time-based metrics
- Combine time-based expressions to assemble complex metrics

Time is a critical component of business analysis. Analysts interpret the state of the business now, often in relation to what it was in the past, with the goal of understanding what it might be in the future.

To support this, Analysis Services provides a number of time-based MDX functions. Using these functions, powerful metrics can be assembled. In this chapter, you learn how to employ the time-based MDX functions to calculate some of the more frequently requested of these metrics.

Understanding the Time Dimension

Analysis Services has no inherent awareness of the concept of time. Although at first glance this may seem like a shortcoming of the tool, it actually affords you the flexibility to define your time dimension in a way that reflects how time is managed in your specific organization.

At the heart of the time dimension is one or more user-hierarchies referred to as *calendars*. Calendars allow you to drill down in time from higher levels of granularity, such as years, into lower levels of granularity, such as quarters, months, and days. Figure 9-1 illustrates one such calendar hierarchy based on the standard calendar we employ in everyday life.

When employed against calendar hierarchies, the time-based MDX functions give the appearance of time awareness. However, most time-based functions are simply exploiting the basic structure of the hierarchy to return the set or member required. In fact, SQL Server Books Online goes so far as to provide the navigational equivalents of each of the time-based functions. If you require slightly different functionality, you can use the navigational functions to implement it yourself.

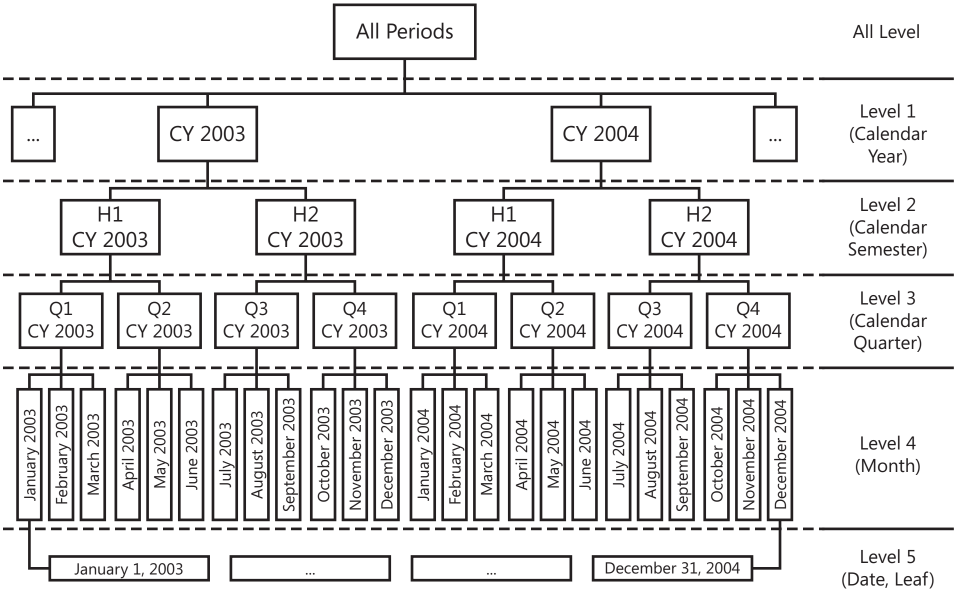


FIGURE 9-1 A user-hierarchy based on the standard calendar

The reliance on the calendar hierarchies for time-based functionality imposes two critical constraints on the attributes of the time dimension. First, the members of the attributes comprising the calendar hierarchies must be ordered in time-based sequence from the past to the present because many time-based functions assume this order. Second, complete sets of members for each attribute should be provided because missing members throw off position-based navigation.

Each of these issues is addressed through cube and ETL-layer design. As an MDX developer, you may not have the responsibility or the access required to ensure that these are addressed in a manner appropriate to your needs. However, if you intend to successfully make use of the time-based functions, you must make sure those responsible for assembling the time dimension are aware of these issues.

Determining the Current Value

A very common request is to return the current value of a metric. Although determining the current value is a seemingly simple request, it can be quite challenging.

First, you need to determine the granularity of the request. We often think of time as continuous, but in Analysis Services time is recorded as discrete members representing ranges of time. Between attributes, these members overlap so that the current date member of one attribute is associated with the current month member of another and the current quarter and year members of still others. Each of these represents quite different ranges of time, but each represents the current time.

Once you know the grain, the next challenge is to determine which member represents the current time. A key characteristic of any data warehouse is latency. The time it takes for changes to data in source systems to be reflected in the data warehouse varies from implementation to implementation, but some degree of latency is always present. Because of this, the data warehouse is only current as of some point in the past. Knowing this simply shifts the challenge from identifying the member associated with the current time to identifying the member associated with the time at which the data is current.

One technique for identifying the time at which the data is current is to employ the VBA time functions *Date*, *Time*, or *Now* to retrieve the current time, and then use the VBA date math functions *DateAdd* or *DateDiff* to adjust the time for latency. You can then use the adjusted value or parts of it extracted by using the VBA *DatePart* function to locate the current time member.

Although effective, this technique requires certainty in the amount of latency in the data. Try as you might, you may not be able to always accurately reflect this in the calculation. Considering the potential complexity of the expression logic as well, other alternatives should be explored.

A preferred alternative is to incorporate a property or attribute within the time dimension identifying a member at an appropriately low level of granularity as current. Relationships between attributes can then be employed to identify current time members at higher levels of granularity. The particulars of this design-time solution to the problem of identifying the current time member vary with the circumstances of your data warehouse, but the approach allows the data warehouse to tell you how up to date it is rather than you telling it how up to date it should be.

Calculating an Accumulating Total

In business, metrics are quite frequently reported as accumulating totals. For example, consider reseller sales in the month of October. Although sales in this month alone are interesting and important, the accumulation of sales over the months of the year up to and including October may be more interesting, especially if you are tracking sales against an annual target.

To calculate accumulating totals, you must determine the set of time members over which a value is to be aggregated. This is done using the *PeriodsToDate* function:

```
PeriodsToDate( [Level , [Member]] )
```

The *PeriodsToDate* function returns the set of members from the start of a given period up to and including a specified member. The *Level* argument identifies the level of the hierarchy representing the period over which the returned set should span, whereas the *Member* argument identifies the set's ending member. You can think of Analysis Services as starting with the specified member, navigating up to its ancestor in the specified level and then back down to the first sibling of the specified member under this shared ancestor. The set returned represents the range of members between and including these two members.

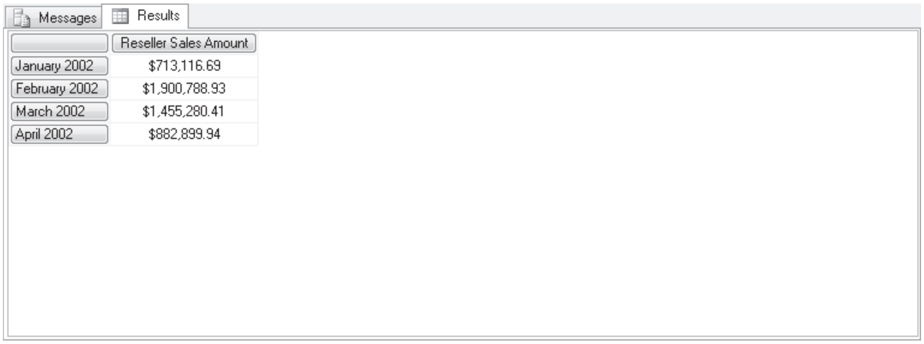
If the *Member* argument is not specified but the *Level* argument is, Analysis Services infers the current member of the hierarchy for the *Member* argument. If neither the *Member* nor the *Level* argument is specified, Analysis Services infers the current member of a hierarchy in a time dimension for the *Member* argument and the parent level of this member for the *Level* argument. For most applications of the *PeriodsToDate* function, you are encouraged to supply both arguments to ensure clarity.

Calculate year-to-date reseller sales

1. Open the MDX Query Editor to the MDX Step-by-Step database.
2. In the code pane, enter the following query to retrieve reseller sales for the periods to date for the month of April 2002:

```
SELECT
    {([Measures].[Reseller Sales Amount])} ON COLUMNS,
    {
        PeriodsToDate(
            [Date].[Calendar].[Calendar Year],
            [Date].[Calendar].[Month].[April 2002]
            )
        } ON ROWS
FROM [Step-by-Step]
```

3. Execute the query and review the results.



	Reseller Sales Amount
January 2002	$713,116.69
February 2002	$1,900,788.93
March 2002	$1,455,280.41
April 2002	$882,899.94

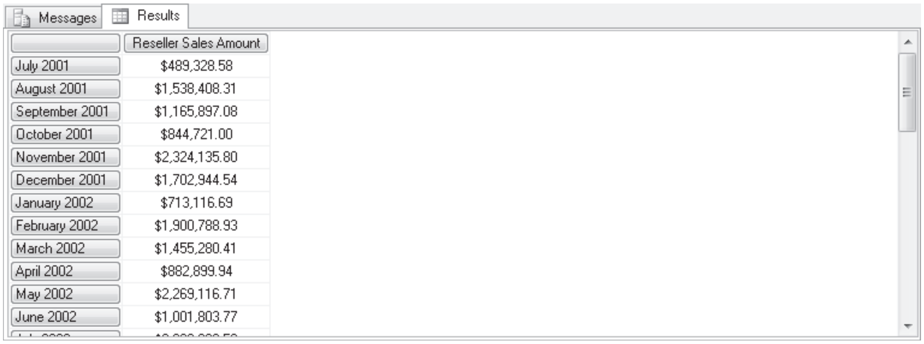
In the preceding query, you use the *PeriodsToDate* function to retrieve all months in the year 2002 prior to and including the month of April. By specifying the Calendar Year level of the Calendar hierarchy, Analysis Services moves from the member April 2002 to its ancestor along this level, CY 2002. It then selects the CY 2002 member's first descendant within the Month level—the level occupied by the specified member April 2002. This first descendant, January 2002, and the specified member, April 2002, then are used to form a range, [Date].[Calendar].[Month].[January 2002]:[Date].[Calendar].[Month].[April 2002], which resolves to the set presented along the *ROWS* axis.

This query demonstrates the basic functionality of the *PeriodsToDate* function, but your goal is to calculate a year-to-date total for reseller sales. Instead of using *PeriodsToDate* to define a set along an axis, you can use the function to define the set over which you aggregate values in a calculated member. As a starting point towards this goal, re-factor the query to return all months along the *ROWS* axis.

4. Modify the query to retrieve reseller sales for each month:

```
SELECT
    {([Measures].[Reseller Sales Amount])} ON COLUMNS,
    {[Date].[Calendar].[Month].Members} ON ROWS
FROM [Step-by-Step]
```

5. Execute the query and review the results.

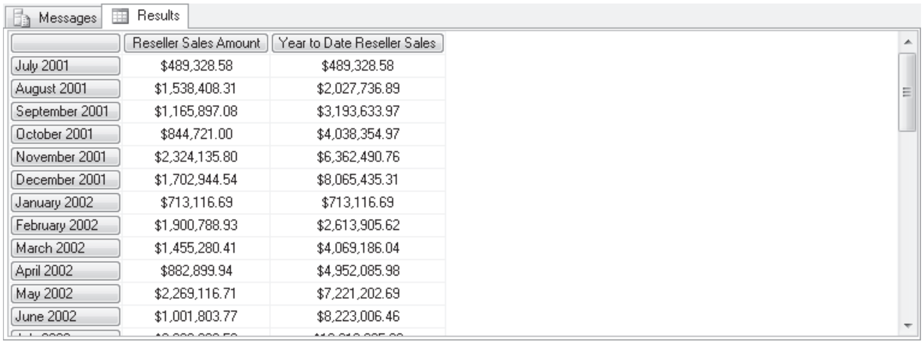


	Reseller Sales Amount
July 2001	$489,328.58
August 2001	$1,538,408.31
September 2001	$1,165,897.08
October 2001	$844,721.00
November 2001	$2,324,135.80
December 2001	$1,702,944.54
January 2002	$713,116.69
February 2002	$1,900,788.93
March 2002	$1,455,280.41
April 2002	$882,899.94
May 2002	$2,269,116.71
June 2002	$1,001,803.77

6. Modify the query to calculate the year-to-date cumulative reseller sales for each member along the *ROWS* axis:

```
WITH
MEMBER [Measures].[Year to Date Reseller Sales] AS
    Aggregate(
        PeriodsToDate(
            [Date].[Calendar].[Calendar Year],
            [Date].[Calendar].CurrentMember
            ),
        ([Measures].[Reseller Sales Amount])
        )
SELECT
    {
        ([Measures].[Reseller Sales Amount]),
        ([Measures].[Year to Date Reseller Sales])
        } ON COLUMNS,
    {[Date].[Calendar].[Month].Members} ON ROWS
FROM [Step-by-Step]
```

7. Execute the query and review the results.



	Reseller Sales Amount	Year to Date Reseller Sales
July 2001	$489,328.58	$489,328.58
August 2001	$1,538,408.31	$2,027,736.89
September 2001	$1,165,897.08	$3,193,633.97
October 2001	$844,721.00	$4,038,354.97
November 2001	$2,324,135.80	$6,362,490.76
December 2001	$1,702,944.54	$8,065,435.31
January 2002	$713,116.69	$713,116.69
February 2002	$1,900,788.93	$2,613,905.62
March 2002	$1,455,280.41	$4,069,186.04
April 2002	$882,899.94	$4,952,085.98
May 2002	$2,269,116.71	$7,221,202.69
June 2002	$1,001,803.77	$8,223,006.46

For each member along the *ROWS* axis, the *PeriodsToDate* function returns the set of members from the start of its calendar year up to and including this member. Over this set, the current measure, Reseller Sales Amount, is aggregated to calculate year-to-date sales. Comparing the year-to-date totals to the monthly sales values for previous months, you can verify this logic.

> **Note** The preceding calculation employs the *Aggregate* function to calculate a running total. For more information on this and the other MDX aggregation functions, see Chapter 7, "Performing Aggregation."

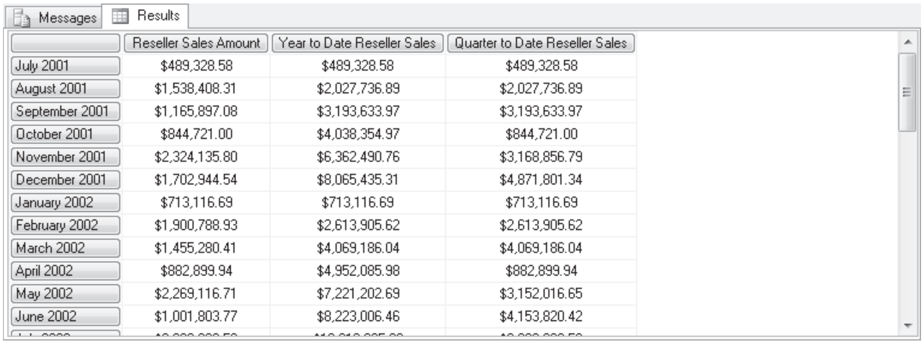
As you review these results, notice between December 2001 and January 2002 the value of the accumulating total "resets." This is because these two members have differing ancestor members within the Calendar Year level. This pattern of accumulation and reset is observed whenever transitions between ancestors occur, as demonstrated in the following calculations of quarter-to-date totals.

8. Add a quarter-to-date total for reseller sales to the query:

```
WITH
MEMBER [Measures].[Year to Date Reseller Sales] AS
    Aggregate(
        PeriodsToDate(
            [Date].[Calendar].[Calendar Year],
            [Date].[Calendar].CurrentMember
            ),
        ([Measures].[Reseller Sales Amount])
        )
MEMBER [Measures].[Quarter to Date Reseller Sales] AS
    Aggregate(
        PeriodsToDate(
            [Date].[Calendar].[Calendar Quarter],
            [Date].[Calendar].CurrentMember
            ),
        ([Measures].[Reseller Sales Amount])
        )
```

```
SELECT
    {
        ([Measures].[Reseller Sales Amount]),
        ([Measures].[Year to Date Reseller Sales]),
        ([Measures].[Quarter to Date Reseller Sales])
        } ON COLUMNS,
    {[Date].[Calendar].[Month].Members} ON ROWS
FROM [Step-by-Step]
```

9. Execute the query and review the new Quarter To Date Reseller Sales values.



	Reseller Sales Amount	Year to Date Reseller Sales	Quarter to Date Reseller Sales
July 2001	$489,328.58	$489,328.58	$489,328.58
August 2001	$1,538,408.31	$2,027,736.89	$2,027,736.89
September 2001	$1,165,897.08	$3,193,633.97	$3,193,633.97
October 2001	$844,721.00	$4,038,354.97	$844,721.00
November 2001	$2,324,135.80	$6,362,490.76	$3,168,856.79
December 2001	$1,702,944.54	$8,065,435.31	$4,871,801.34
January 2002	$713,116.69	$713,116.69	$713,116.69
February 2002	$1,900,788.93	$2,613,905.62	$2,613,905.62
March 2002	$1,455,280.41	$4,069,186.04	$4,069,186.04
April 2002	$882,899.94	$4,952,085.98	$882,899.94
May 2002	$2,269,116.71	$7,221,202.69	$3,152,016.65
June 2002	$1,001,803.77	$8,223,006.46	$4,153,820.42

Reviewing the results, you can see the same pattern of accumulation and reset with the Quarter To Date Reseller Sales calculated measure as you do with the Year To Date Reseller Sales calculated measure. The only difference is that the pattern is based on a quarterly cycle as opposed to an annual one.

Simplifying Periods-to-Date Calculations

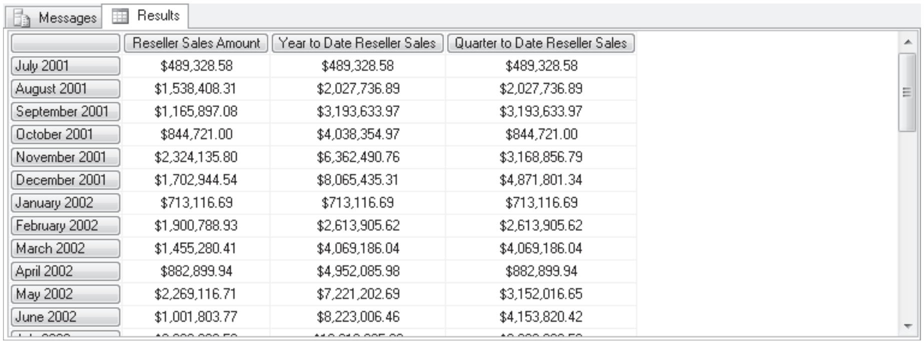
Many of the attributes in a time dimension are assigned *Type* property values at design time, identifying the attributes as representing years, quarters, months, or weeks. Analysis Services can return period-to-date sets based on these type assignments without the identification of a level by name. This functionality is provided through the specialized *Ytd*, *Qtd*, *Mtd*, and *Wtd* functions returning year-to-date, quarter-to-date, month-to-date, and week-to-date sets, respectively:

```
Ytd( [Member] )
Qtd( [Member] )
Mtd( [Member] )
Wtd( [Member] )
```

These functions, collectively referred to as the *xTD* functions, are logically equivalent to the *PeriodsToDate* function with hard-coded level arguments. Their reliance on the proper assignment of *Type* property values at design time makes them more succinct but also makes them dependent on settings into which you may have little insight. If you use the *xTD* functions, it is important for you to verify the set returned.

To demonstrate the use of the *xTD* functions, the last query of the previous exercise is rewritten using *Ytd* and *Qtd* to derive the year-to-date and quarter-to-date sets, respectively:

```
WITH
MEMBER [Measures].[Year to Date Reseller Sales] AS
    Aggregate(
        Ytd([Date].[Calendar].CurrentMember),
        ([Measures].[Reseller Sales Amount])
        )
MEMBER [Measures].[Quarter to Date Reseller Sales] AS
    Aggregate(
        Qtd([Date].[Calendar].CurrentMember),
        ([Measures].[Reseller Sales Amount])
        )
SELECT
    {
        ([Measures].[Reseller Sales Amount]),
        ([Measures].[Year to Date Reseller Sales]),
        ([Measures].[Quarter to Date Reseller Sales])
        } ON COLUMNS,
    {[Date].[Calendar].[Month].Members} ON ROWS
FROM [Step-by-Step]
```



	Reseller Sales Amount	Year to Date Reseller Sales	Quarter to Date Reseller Sales
July 2001	$489,328.58	$489,328.58	$489,328.58
August 2001	$1,538,408.31	$2,027,736.89	$2,027,736.89
September 2001	$1,165,897.08	$3,193,633.97	$3,193,633.97
October 2001	$844,721.00	$4,038,354.97	$844,721.00
November 2001	$2,324,135.80	$6,362,490.76	$3,168,856.79
December 2001	$1,702,944.54	$8,065,435.31	$4,871,801.34
January 2002	$713,116.69	$713,116.69	$713,116.69
February 2002	$1,900,788.93	$2,613,905.62	$2,613,905.62
March 2002	$1,455,280.41	$4,069,186.04	$4,069,186.04
April 2002	$882,899.94	$4,952,085.98	$882,899.94
May 2002	$2,269,116.71	$7,221,202.69	$3,152,016.65
June 2002	$1,001,803.77	$8,223,006.46	$4,153,820.42

Calculating Inception-to-Date

The period-to-date calculations return a value based on a range that is restricted to a particular period, such as a quarter or year. Occasionally, you may wish to calculate an accumulating value across all periods for which data is recorded. This is referred to as an *inception-to-date* value.

You can retrieve the inception-to-date range using the *PeriodsToDate* function with the calendar's (All) member's level as the period identifier, as demonstrated in the following expression:

```
PeriodsToDate(
    [Date].[Calendar].[(All)],
    [Date].[Calendar].CurrentMember
    )
```

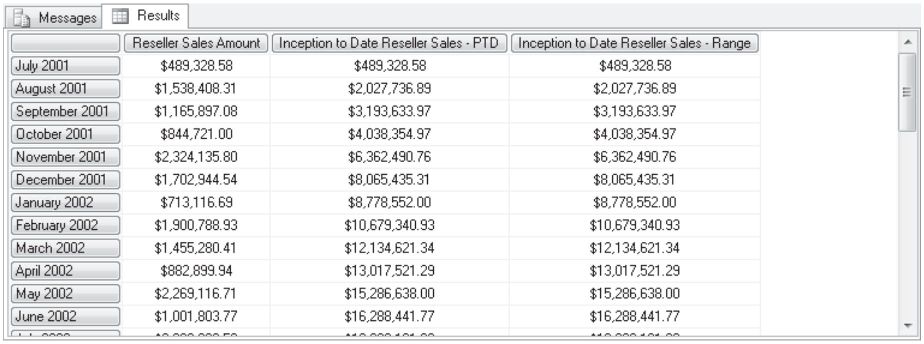
Although this expression is perfectly valid, many MDX developers typically calculate inception-to-date sets employing a range-based shortcut:

```
Null: [Date].[Calendar].CurrentMember
```

The Null member reference forces Analysis Services to evaluate the range from a position just prior to the first member of the level on which the current time member resides. The result is the same set returned by the previous expression that employed the *PeriodsToDate* function.

Whichever technique you employ, measures are aggregated over the set just as with other period-to-date calculations, as demonstrated in the following example:

```
WITH
MEMBER [Measures].[Inception to Date Reseller Sales - PTD] AS
    Aggregate(
        PeriodsToDate(
            [Date].[Calendar].[(All)],
            [Date].[Calendar].CurrentMember
            ),
        ([Measures].[Reseller Sales Amount])
        )
MEMBER [Measures].[Inception to Date Reseller Sales - Range] AS
    Aggregate(
        NULL:[Date].[Calendar].CurrentMember,
        ([Measures].[Reseller Sales Amount])
        )
SELECT
    {
        ([Measures].[Reseller Sales Amount]),
        ([Measures].[Inception to Date Reseller Sales - PTD]),
        ([Measures].[Inception to Date Reseller Sales - Range])
        } ON COLUMNS,
    {[Date].[Calendar].[Month].Members} ON ROWS
FROM [Step-by-Step]
```



	Reseller Sales Amount	Inception to Date Reseller Sales - PTD	Inception to Date Reseller Sales - Range
July 2001	$489,328.58	$489,328.58	$489,328.58
August 2001	$1,538,408.31	$2,027,736.89	$2,027,736.89
September 2001	$1,165,897.08	$3,193,633.97	$3,193,633.97
October 2001	$844,721.00	$4,038,354.97	$4,038,354.97
November 2001	$2,324,135.80	$6,362,490.76	$6,362,490.76
December 2001	$1,702,944.54	$8,065,435.31	$8,065,435.31
January 2002	$713,116.69	$8,778,552.00	$8,778,552.00
February 2002	$1,900,788.93	$10,679,340.93	$10,679,340.93
March 2002	$1,455,280.41	$12,134,621.34	$12,134,621.34
April 2002	$882,899.94	$13,017,521.29	$13,017,521.29
May 2002	$2,269,116.71	$15,286,638.00	$15,286,638.00
June 2002	$1,001,803.77	$16,288,441.77	$16,288,441.77

Calculating Rolling Averages

Analysts often look for changes in values over time. Natural variability in most data can make it difficult to identify meaningful changes. Rolling averages are frequently employed to *smooth out* some of this variation, allowing more significant or longer-term changes to be more readily identified.

A rolling average is calculated as the average of values for some number of periods before or after (and including) the period of interest. For example, the three-month rolling average of sales for the month of February might be determined as the average of sales for February, January, and December. A three-month rolling average calculated in this manner is common in business analysis.

The heart of the rolling average calculation is the determination of the set of periods over which values will be averaged. To support the retrieval of this set, the MDX function *LastPeriods* is provided:

```
LastPeriods( n [, Member] )
```

The *LastPeriods* function returns a set of *n* members before or after (and including) a specified member of a time hierarchy. If a positive *n* value is provided, the set returned includes the members preceding the member of interest. If a negative *n* value is provided, the set returned includes the members following the member of interest.

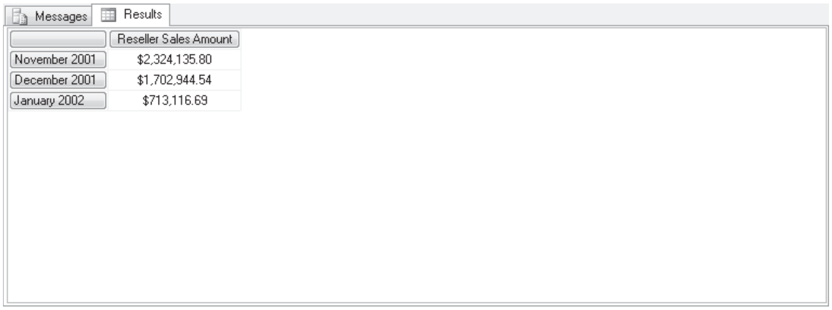
The function's second argument is optional. If the second argument is not supplied, Analysis Services assumes the current member of a hierarchy in a time dimension. For most applications of the *LastPeriods* function, you are encouraged to employ the *Member* argument to ensure clarity.

Calculate the three-month rolling average for reseller sales

1. Open the MDX Query Editor to the MDX Step-by-Step database.
2. In the code pane, enter the following query to retrieve reseller sales for the three periods preceding and including January 2002:

```
SELECT
    {([Measures].[Reseller Sales Amount])} ON COLUMNS,
    {
        LastPeriods(
            3,
            [Date].[Calendar].[Month].[January 2002]
            )
        } ON ROWS
FROM [Step-by-Step]
```

3. Execute the query and review the results.



	Reseller Sales Amount
November 2001	$2,324,135.80
December 2001	$1,702,944.54
January 2002	$713,116.69

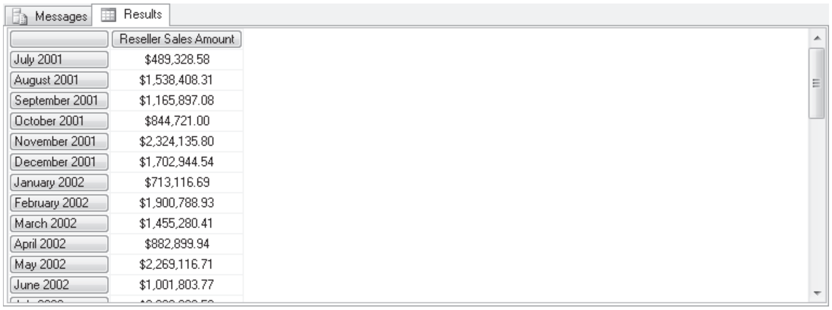
In this query, you use the *LastPeriods* function to retrieve the three-month period preceding and including January 2002. Analysis Services starts with the specified member, January 2002, and treats this as period 1. This leaves *n*-1 or 2 members to return in the set. Because *n* is a positive number, Analysis Services retrieves the January 2002 member's two preceding siblings to complete the set. (Notice that the November and December 2001 siblings were selected without regard for the change in the Calendar Year ancestor between them and the January 2002 member.)

This query demonstrates the basic functionality of the *LastPeriods* function, but your goal is to calculate a rolling average for reseller sales. Instead of using *LastPeriods* to define a set along an axis, you can use the function to define the set over which you will average values in a calculated member. As a starting point towards this goal, re-factor the query to return all months along the *ROWS* axis.

4. Alter the query to retrieve reseller sales for various months:

```
SELECT
    {([Measures].[Reseller Sales Amount])} ON COLUMNS,
    {[Date].[Calendar].[Month].Members} ON ROWS
FROM [Step-by-Step]
```

5. Execute the query and review the results.



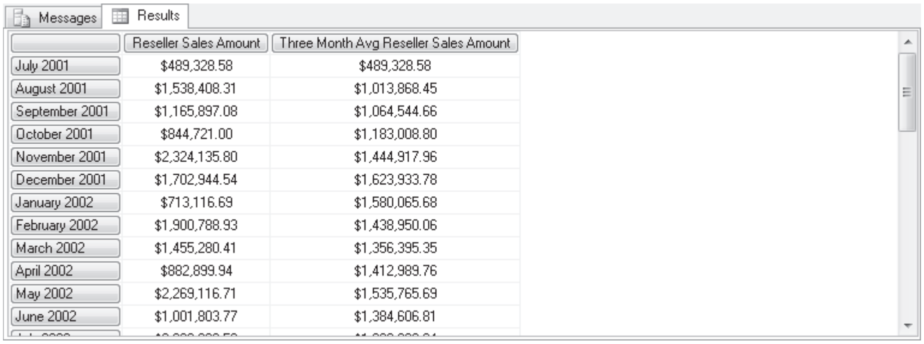
	Reseller Sales Amount
July 2001	$489,328.58
August 2001	$1,538,408.31
September 2001	$1,165,897.08
October 2001	$844,721.00
November 2001	$2,324,135.80
December 2001	$1,702,944.54
January 2002	$713,116.69
February 2002	$1,900,788.93
March 2002	$1,455,280.41
April 2002	$882,899.94
May 2002	$2,269,116.71
June 2002	$1,001,803.77

Reseller sales vary considerably between various months. For example, take a look at the six-month period between October 2001 and March 2002. The wild swings between monthly sales make it difficult to determine any general upward or downward trends during this period. The same is true of the months between June 2002 and December 2002.

6. Alter the query to calculate a three-month rolling average for reseller sales:

```
WITH
MEMBER [Measures].[Three Month Avg Reseller Sales Amount] AS
    Avg(
        LastPeriods(
            3,
            [Date].[Calendar].CurrentMember
            ),
        ([Measures].[Reseller Sales Amount])
        )
SELECT
    {
        ([Measures].[Reseller Sales Amount]),
        ([Measures].[Three Month Avg Reseller Sales Amount])
        } ON COLUMNS,
    {[Date].[Calendar].[Month].Members} ON ROWS
FROM [Step-by-Step]
```

7. Execute the query and compare the monthly reseller sales values to the three-month rolling average values.



	Reseller Sales Amount	Three Month Avg Reseller Sales Amount
July 2001	$489,328.58	$489,328.58
August 2001	$1,538,408.31	$1,013,868.45
September 2001	$1,165,897.08	$1,064,544.66
October 2001	$844,721.00	$1,183,008.80
November 2001	$2,324,135.80	$1,444,917.96
December 2001	$1,702,944.54	$1,623,933.78
January 2002	$713,116.69	$1,580,065.68
February 2002	$1,900,788.93	$1,438,950.06
March 2002	$1,455,280.41	$1,356,395.35
April 2002	$882,899.94	$1,412,989.76
May 2002	$2,269,116.71	$1,535,765.69
June 2002	$1,001,803.77	$1,384,606.81

The three-month rolling average smoothes out some of the variability in the data, making general trends more easily observed. The period from October 2001 to March 2002 that reflected so much variability based on monthly sales totals now appears to be trending only slightly upward. The period from June 2002 and December 2002 that also displayed considerable variability appears to be trending more significantly upward. Without the smoothing effect of the rolling average, these trends would be harder to observe and differentiate.

Performing Period-over-Period Analysis

Historical values are frequently used in data analysis to provide perspective on current values. When comparing historical to current values, it is important you select values from time periods relatively similar to one another. Although no two time periods are exactly alike, analysts often compare values from what are referred to as *parallel periods* to minimize differences resulting from cyclical, time-dependent variations in the data.

To understand parallel periods, consider the month of April 2003. This month is the fourth month of the calendar year 2003. In a business heavily influenced by annual cycles, you might compare values for this month to those for the month of April in a prior year. In doing so, you might accurately (or inaccurately) assume that differences in current and historical values are due to factors other than the annual cyclical influence.

Should you compare values for April 2003 to those of January 2003 or October 2002? Your first response may be to say no. However, if your business is heavily influenced by quarterly cycles, this might be completely appropriate. April 2003 is the first month of a calendar quarter. January 2003 is the first month of the prior quarter and is therefore a parallel member based on quarter. October 2002 is also a parallel member except that it is from two quarters prior. What constitutes an appropriate parallel period for your analysis is highly dependent upon the time-based cycles influencing your business.

To assist you with the retrieval of parallel period members, Analysis Services provides the *ParallelPeriod* function:

```
ParallelPeriod( [Level [,n [, Member]]] )
```

The function's first argument identifies the level of the time hierarchy across which you wish to identify the parallel period member. If no level is identified, the parent level of the current time member is assumed.

The function's second argument identifies how far back along the identified level you wish to go to retrieve the parallel member. If no value is provided, a value of 1 is assumed, indicating the prior period.

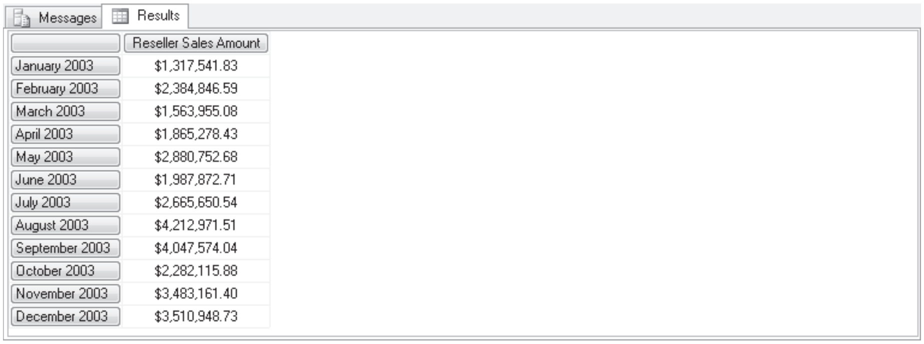
The function's final argument identifies the member for which the parallel period is to be determined. The position of this member relative to its ancestor in the specified level determines the member retrieved from the historical period. If no member is identified, the current time member is assumed.

Calculate growth over prior period

1. Open the MDX Query Editor to the MDX Step-by-Step database.
2. In the code pane, enter the following query to retrieve reseller sales for the months of calendar year 2003:

```
SELECT
    {([Measures].[Reseller Sales Amount])} ON COLUMNS,
    {
        Descendants(
            [Date].[Calendar].[Calendar Year].[CY 2003],
            [Date].[Calendar].[Month],
            SELF
            )
         } ON ROWS
FROM [Step-by-Step]
```

3. Execute the query and review the results.



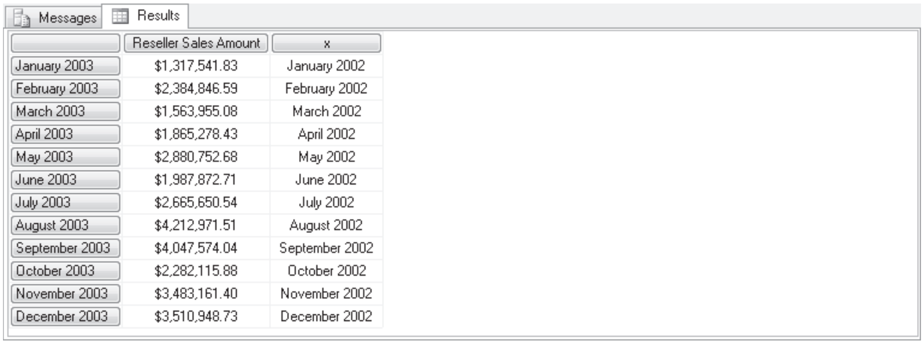
	Reseller Sales Amount
January 2003	$1,317,541.83
February 2003	$2,384,846.59
March 2003	$1,563,955.08
April 2003	$1,865,278.43
May 2003	$2,880,752.68
June 2003	$1,987,872.71
July 2003	$2,665,650.54
August 2003	$4,212,971.51
September 2003	$4,047,574.04
October 2003	$2,282,115.88
November 2003	$3,483,161.40
December 2003	$3,510,948.73

The query returns reseller sales for the months of calendar year 2003. To assess the strength of these numbers in a business influenced by annual sales cycles, you might compare them to sales in the prior year. To do this, start by identifying the prior period for each month.

4. Alter the query to identify the parallel period in the prior year for each month:

```
WITH
MEMBER [Measures].[x] AS
    ParallelPeriod(
        [Date].[Calendar].[Calendar Year],
        1,
        [Date].[Calendar].CurrentMember
        ).Name
SELECT
    {
        ([Measures].[Reseller Sales Amount]),
        ([Measures].[x])
        } ON COLUMNS,
    {
        Descendants(
            [Date].[Calendar].[Calendar Year].[CY 2003],
            [Date].[Calendar].[Month],
            SELF
            )
         } ON ROWS
FROM [Step-by-Step]
```

5. Execute the query and review the results.



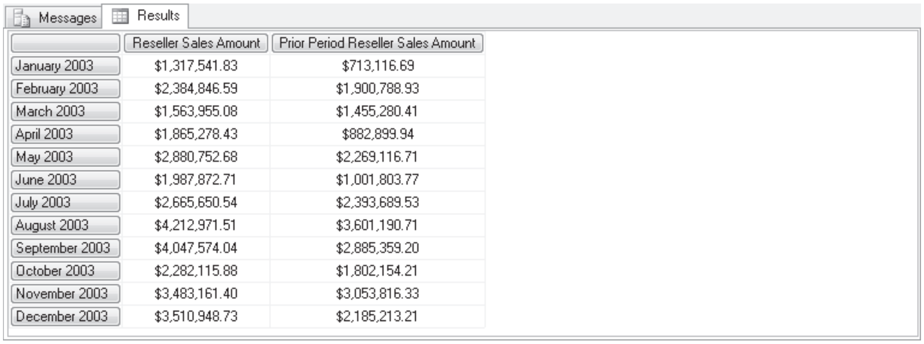
	Reseller Sales Amount	x
January 2003	$1,317,541.83	January 2002
February 2003	$2,384,846.59	February 2002
March 2003	$1,563,955.08	March 2002
April 2003	$1,865,278.43	April 2002
May 2003	$2,880,752.68	May 2002
June 2003	$1,987,872.71	June 2002
July 2003	$2,665,650.54	July 2002
August 2003	$4,212,971.51	August 2002
September 2003	$4,047,574.04	September 2002
October 2003	$2,282,115.88	October 2002
November 2003	$3,483,161.40	November 2002
December 2003	$3,510,948.73	December 2002

In the preceding query, the *ParallelPeriod* function is used to identify the parallel period in the prior year for each month in calendar year 2003 along the *ROWS* axis. The *ParallelPeriod* function returns a member and the name of that member is returned with a new calculated member to verify that the appropriate member is being identified. Now that you are comfortable the correct member is being located, you can use the returned member to determine prior period sales.

6. Alter the query to calculate prior period sales:

```
WITH
MEMBER [Measures].[Prior Period Reseller Sales Amount] AS
    (
        ParallelPeriod(
            [Date].[Calendar].[Calendar Year],
            1,
            [Date].[Calendar].CurrentMember
            ),
        [Measures].[Reseller Sales Amount]
        )
    ,FORMAT="Currency"
SELECT
    {
        ([Measures].[Reseller Sales Amount]),
        ([Measures].[Prior Period Reseller Sales Amount])
        } ON COLUMNS,
    {
        Descendants(
            [Date].[Calendar].[Calendar Year].[CY 2003],
            [Date].[Calendar].[Month],
            SELF
            )
         } ON ROWS
FROM [Step-by-Step]
```

7. Execute the query and review the results.



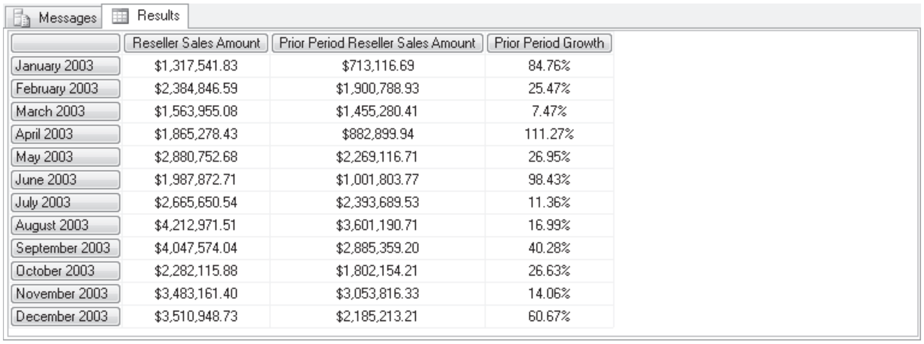
	Reseller Sales Amount	Prior Period Reseller Sales Amount
January 2003	$1,317,541.83	$713,116.69
February 2003	$2,384,846.59	$1,900,788.93
March 2003	$1,563,955.08	$1,455,280.41
April 2003	$1,865,278.43	$882,899.94
May 2003	$2,880,752.68	$2,269,116.71
June 2003	$1,987,872.71	$1,001,803.77
July 2003	$2,665,650.54	$2,393,689.53
August 2003	$4,212,971.51	$3,601,190.71
September 2003	$4,047,574.04	$2,885,359.20
October 2003	$2,282,115.88	$1,802,154.21
November 2003	$3,483,161.40	$3,053,816.33
December 2003	$3,510,948.73	$2,185,213.21

Using the member returned by the *ParallelPeriod* function to assemble a tuple allows you to retrieve reseller sales for the prior period. This newly calculated measure is returned along the *COLUMNS* axis for comparison against sales in the months displayed across the rows. To facilitate comparison, you might wish to present the percent change in sales from the prior period.

8. Alter the query to calculate the percent change in sales (growth) between the current and prior periods:

```
WITH
MEMBER [Measures].[Prior Period Reseller Sales Amount] AS
    (
        ParallelPeriod(
            [Date].[Calendar].[Calendar Year],
            1,
            [Date].[Calendar].CurrentMember
            ),
        [Measures].[Reseller Sales Amount]
        )
    ,FORMAT="Currency"
MEMBER [Measures].[Prior Period Growth] AS
    (
        ([Measures].[Reseller Sales Amount])-
            ([Measures].[Prior Period Reseller Sales Amount])
        ) /
        ([Measures].[Prior Period Reseller Sales Amount])
    ,FORMAT="Percent"
SELECT
    {
        ([Measures].[Reseller Sales Amount]),
        ([Measures].[Prior Period Reseller Sales Amount]),
        ([Measures].[Prior Period Growth])
        } ON COLUMNS,
    {
        Descendants(
            [Date].[Calendar].[Calendar Year].[CY 2003],
            [Date].[Calendar].[Month],
            SELF
            )
         } ON ROWS
FROM [Step-by-Step]
```

9. Execute the query and review the results.



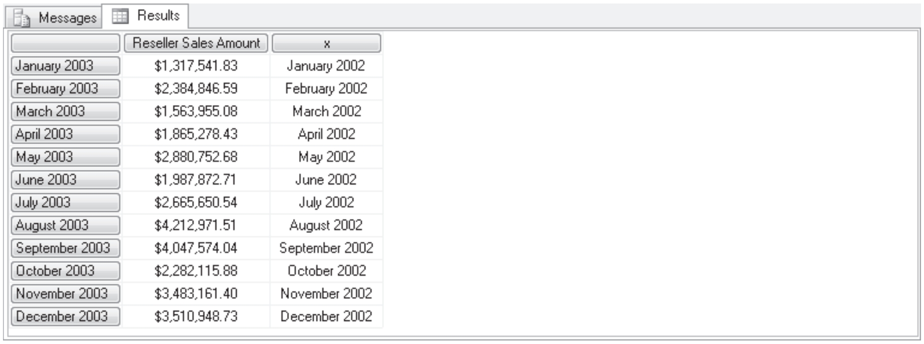
	Reseller Sales Amount	Prior Period Reseller Sales Amount	Prior Period Growth
January 2003	$1,317,541.83	$713,116.69	84.76%
February 2003	$2,384,846.59	$1,900,788.93	25.47%
March 2003	$1,563,955.08	$1,455,280.41	7.47%
April 2003	$1,865,278.43	$882,899.94	111.27%
May 2003	$2,880,752.68	$2,269,116.71	26.95%
June 2003	$1,987,872.71	$1,001,803.77	98.43%
July 2003	$2,665,650.54	$2,393,689.53	11.36%
August 2003	$4,212,971.51	$3,601,190.71	16.99%
September 2003	$4,047,574.04	$2,885,359.20	40.28%
October 2003	$2,282,115.88	$1,802,154.21	26.63%
November 2003	$3,483,161.40	$3,053,816.33	14.06%
December 2003	$3,510,948.73	$2,185,213.21	60.67%

The results show each month of calendar year 2003 experienced considerable growth in reseller sales from those of the month in the prior year.

A Word of Caution

As explained at the start of this chapter, the time-based MDX functions are not time-aware and simply employ basic navigation for their functionality. This is illustrated by rewriting the query in Step 4 of the previous exercise with the navigation functions *Cousin*, *Ancestor*, and *Lag*:

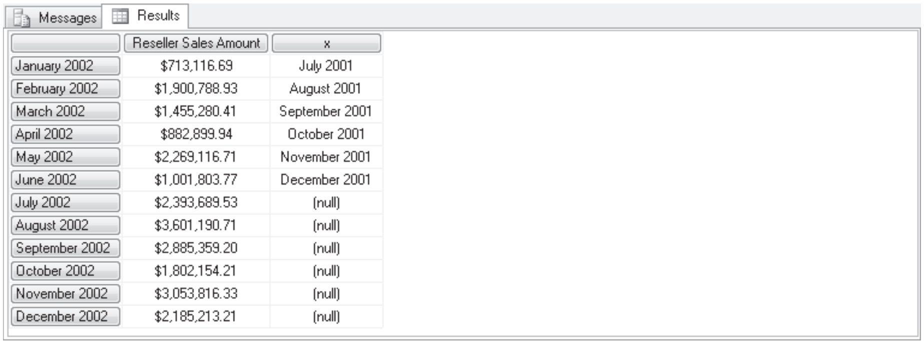
```
WITH
MEMBER [Measures].[x] AS
    Cousin(
        [Date].[Calendar].CurrentMember,
        Ancestor(
            [Date].[Calendar].CurrentMember,
            [Date].[Calendar].[Calendar Year]
            ).Lag(1)
        ).Name
SELECT
    {
        ([Measures].[Reseller Sales Amount]),
        ([Measures].[x])
        } ON COLUMNS,
    {
        Descendants(
            [Date].[Calendar].[Calendar Year].[CY 2003],
            [Date].[Calendar].[Month],
            SELF
            )
         } ON ROWS
FROM [Step-by-Step]
```



	Reseller Sales Amount	x
January 2003	$1,317,541.83	January 2002
February 2003	$2,384,846.59	February 2002
March 2003	$1,563,955.08	March 2002
April 2003	$1,865,278.43	April 2002
May 2003	$2,880,752.68	May 2002
June 2003	$1,987,872.71	June 2002
July 2003	$2,665,650.54	July 2002
August 2003	$4,212,971.51	August 2002
September 2003	$4,047,574.04	September 2002
October 2003	$2,282,115.88	October 2002
November 2003	$3,483,161.40	November 2002
December 2003	$3,510,948.73	December 2002

As previously mentioned, the use of basic navigation to provide time-based functionality imposes some constraints on your time dimension. One of these is that all members of a time period should be provided in the cube. Again, the query in Step 4 from the previous exercise provides a very clear demonstration of why this is important. Here is that query adjusted to present the months of calendar year 2002 along the *ROWS* axis:

```
WITH
MEMBER [Measures].[x] AS
    ParallelPeriod(
        [Date].[Calendar].[Calendar Year],
        1,
        [Date].[Calendar].CurrentMember
        ).Name
SELECT
    {
        ([Measures].[Reseller Sales Amount]),
        ([Measures].[x])
        } ON COLUMNS,
    {
        Descendants(
            [Date].[Calendar].[Calendar Year].[CY 2002],
            [Date].[Calendar].[Month],
            SELF
            )
         } ON ROWS
FROM [Step-by-Step]
```



	Reseller Sales Amount	x
January 2002	$713,116.69	July 2001
February 2002	$1,900,788.93	August 2001
March 2002	$1,455,280.41	September 2001
April 2002	$882,899.94	October 2001
May 2002	$2,269,116.71	November 2001
June 2002	$1,001,803.77	December 2001
July 2002	$2,393,689.53	(null)
August 2002	$3,601,190.71	(null)
September 2002	$2,885,359.20	(null)
October 2002	$1,802,154.21	(null)
November 2002	$3,053,816.33	(null)
December 2002	$2,185,213.21	(null)

Notice in the results of this query that the month of January 2002 has a parallel period of July 2001. January 2002 is the first month-level descendant of calendar year 2002. Its parallel period in the prior year is the first month-level descendant of calendar year 2001. Because the first month recorded in 2001 is July, July 2001 becomes the parallel period of January 2002 based on simple navigation. Apply this logic to July 2002, the seventh month-level descendant of calendar year 2002, and you see why it has no parallel period in 2001, a year in which only six months were recorded.

If all twelve months for calendar year 2001 had been recorded, this problem could have been avoided. However, this problem would now be deferred to the fiscal calendar whose years start prior to 2001. In other words, there is no way in this dimension to provide complete sets of members under each period.

So what's the solution to this problem? The short answer is there really isn't one. You as the query developer must be aware of boundary issues such as this when developing queries employing time-based functions. You might have data at the head and tail of the time dimension extended to cover periods for which no data is recorded to avoid misalignment as illustrated previously, but you still need to be aware that no data is recorded for those periods so that some forms of analysis, such as period-over-period growth, might not be appropriate.

Combining Time-Based Metrics

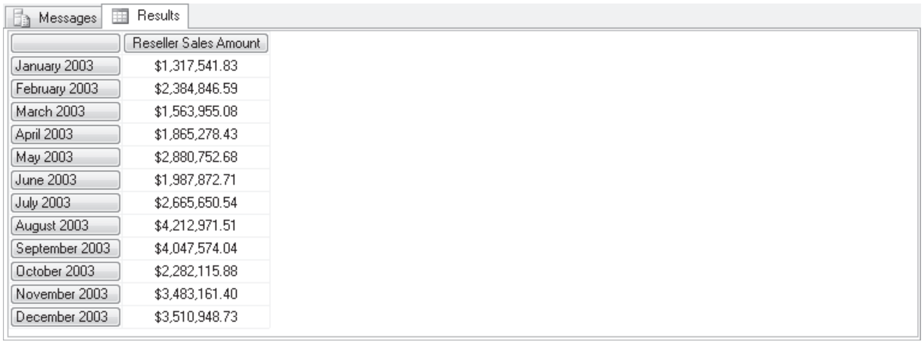
Throughout this chapter, you have explored the various time-based functions and how they can be used to enhance business analysis and solve business problems. Although each of these functions is valuable on its own, they are often used in combination to provide even greater insight and clarity into the analysis of business data. These may seem like very challenging metrics to assemble, but in reality they are no more complex than most other metrics calculated throughout this book. The trick is to remember tuple and expression basics.

Calculate year-to-date and prior period year-to-date sales

1. Open the MDX Query Editor to the MDX Step-by-Step database.
2. Enter the following query to retrieve reseller sales for the months of calendar year 2003:

```
SELECT
    {
        ([Measures].[Reseller Sales Amount])
        } ON COLUMNS,
    {
        Descendants(
            [Date].[Calendar].[CY 2003],
            [Date].[Calendar].[Month],
            SELF
            )
        } ON ROWS
FROM [Step-by-Step]
```

3. Execute the query and review the results.



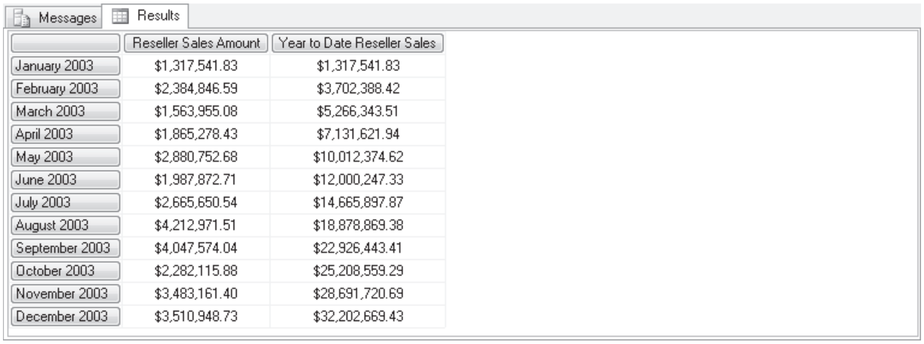
	Reseller Sales Amount
January 2003	$1,317,541.83
February 2003	$2,384,846.59
March 2003	$1,563,955.08
April 2003	$1,865,278.43
May 2003	$2,880,752.68
June 2003	$1,987,872.71
July 2003	$2,665,650.54
August 2003	$4,212,971.51
September 2003	$4,047,574.04
October 2003	$2,282,115.88
November 2003	$3,483,161.40
December 2003	$3,510,948.73

The query returns reseller sales by month for calendar year 2003. Using the *PeriodsToDate* function, you can calculate year-to-date sales just like before.

4. Alter the query to calculate a year-to-date sales:

```
WITH
MEMBER [Measures].[Year to Date Reseller Sales] AS
    Aggregate(
        PeriodsToDate(
            [Date].[Calendar].[Calendar Year],
            [Date].[Calendar].CurrentMember
            ),
        ([Measures].[Reseller Sales Amount])
        )
    ,FORMAT="Currency"
SELECT
    {
        ([Measures].[Reseller Sales Amount]),
        ([Measures].[Year to Date Reseller Sales])
        } ON COLUMNS,
    {
        Descendants(
            [Date].[Calendar].[CY 2003],
            [Date].[Calendar].[Month],
            SELF
            )
        } ON ROWS
FROM [Step-by-Step]
```

5. Execute the query and review the results.



	Reseller Sales Amount	Year to Date Reseller Sales
January 2003	$1,317,541.83	$1,317,541.83
February 2003	$2,384,846.59	$3,702,388.42
March 2003	$1,563,955.08	$5,266,343.51
April 2003	$1,865,278.43	$7,131,621.94
May 2003	$2,880,752.68	$10,012,374.62
June 2003	$1,987,872.71	$12,000,247.33
July 2003	$2,665,650.54	$14,665,897.87
August 2003	$4,212,971.51	$18,878,869.38
September 2003	$4,047,574.04	$22,926,443.41
October 2003	$2,282,115.88	$25,208,559.29
November 2003	$3,483,161.40	$28,691,720.69
December 2003	$3,510,948.73	$32,202,669.43

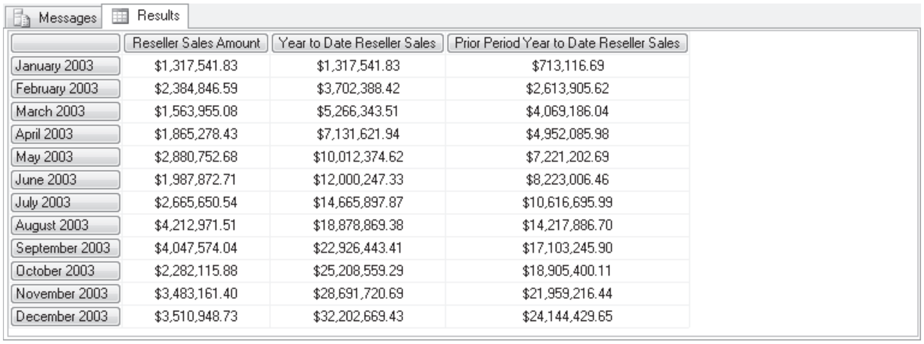
Using the Year To Date Reseller Sales calculated member in a tuple, you can easily calculate year-to-date sales for the prior period.

6. Alter the query to calculate the prior period year-to-date sales:

```
WITH
MEMBER [Measures].[Prior Period Year to Date Reseller Sales] AS
    (
        ParallelPeriod(
            [Date].[Calendar].[Calendar Year],
            1,
            [Date].[Calendar].CurrentMember
            ),
        [Measures].[Year to Date Reseller Sales]
        )
    ,FORMAT="Currency"
```

```
MEMBER [Measures].[Year to Date Reseller Sales] AS
    Aggregate(
        PeriodsToDate(
            [Date].[Calendar].[Calendar Year],
            [Date].[Calendar].CurrentMember
            ),
        ([Measures].[Reseller Sales Amount])
        )
    ,FORMAT="Currency"
SELECT
    {
        ([Measures].[Reseller Sales Amount]),
        ([Measures].[Year to Date Reseller Sales]),
        ([Measures].[Prior Period Year to Date Reseller Sales])
        } ON COLUMNS,
    {
        Descendants(
            [Date].[Calendar].[CY 2003],
            [Date].[Calendar].[Month],
            SELF
            )
        } ON ROWS
FROM [Step-by-Step]
```

7. Execute the query and review the results.



	Reseller Sales Amount	Year to Date Reseller Sales	Prior Period Year to Date Reseller Sales
January 2003	$1,317,541.83	$1,317,541.83	$713,116.69
February 2003	$2,384,846.59	$3,702,388.42	$2,613,905.62
March 2003	$1,563,955.08	$5,266,343.51	$4,069,186.04
April 2003	$1,865,278.43	$7,131,621.94	$4,952,085.98
May 2003	$2,880,752.68	$10,012,374.62	$7,221,202.69
June 2003	$1,987,872.71	$12,000,247.33	$8,223,006.46
July 2003	$2,665,650.54	$14,665,897.87	$10,616,695.99
August 2003	$4,212,971.51	$18,878,869.38	$14,217,886.70
September 2003	$4,047,574.04	$22,926,443.41	$17,103,245.90
October 2003	$2,282,115.88	$25,208,559.29	$18,905,400.11
November 2003	$3,483,161.40	$28,691,720.69	$21,959,216.44
December 2003	$3,510,948.73	$32,202,669.43	$24,144,429.65

This exercise demonstrates a very simple approach to combining calculated members that use time-based functions. When formulating complex metrics, you can easily lose sight of the basic techniques allowing logic in one calculated member to be leveraged for another. As easily as you combined a period-to-date calculation with a prior period calculation, you could extend this query to include the difference, variance, or percent growth of the current year year-to-date values compared to the prior year year-to-date values or any flavors thereof.

The *OpeningPeriod* and *ClosingPeriod* Functions

We would be remiss if we did not mention the *OpeningPeriod* and *ClosingPeriod* functions. The introduction of expanded support for semi-additive measures in the 2005 release of

Analysis Services has diminished the role of these functions, which return the first and last members of a period:

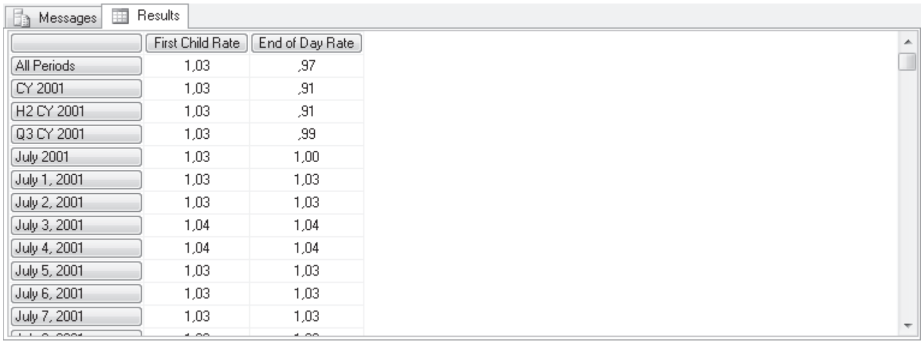
```
OpeningPeriod( [Level [, Member]] )
ClosingPeriod( [Level [, Member]] )
```

The *OpeningPeriod* and *ClosingPeriod* functions return the first or last member, respectively, of the descendants from a given level and a specified member. If no level is specified, Analysis Services assumes the topmost level of the time hierarchy. If no member is specified, Analysis Services assumes the current time member. As with the other time-based functions, you are encouraged to supply both arguments to ensure clarity.

As previously mentioned, both the *OpeningPeriod* and *ClosingPeriod* functions have seen their use diminished with recent releases of Analysis Services. Historically, these functions have been used to calculate values now returned by the FirstChild, FirstNonEmpty, LastChild, and LastNonEmpty aggregate functions. These aggregate functions are frequently employed with finance facts, exchange rates, and other snapshot facts to identify period starting and ending values.

For example, the end-of-day exchange rate employs the LastNonEmpty aggregate function to provide access to the last available value within a given period. But what if you needed to determine the end-of-day exchange rate at the start of a period? The following query illustrates the use of the *OpeningPeriod* function to calculate this value:

```
WITH
MEMBER [Measures].[First Child Rate] AS
    (
        OpeningPeriod(
            [Date].[Calendar].[Date],
            [Date].[Calendar].CurrentMember
            ),
             [Measures].[End of Day Rate]
            )
       ,FORMAT="Standard"
SELECT
    {
        ([Measures].[First Child Rate]),
        ([Measures].[End of Day Rate])
        } ON COLUMNS,
    {[Date].[Calendar].Members} ON ROWS
FROM [Step-by-Step]
WHERE ([Destination Currency].[Destination Currency].[Euro])
```



	First Child Rate	End of Day Rate
All Periods	1,03	,97
CY 2001	1,03	,91
H2 CY 2001	1,03	,91
Q3 CY 2001	1,03	,99
July 2001	1,03	1,00
July 1, 2001	1,03	1,03
July 2, 2001	1,03	1,03
July 3, 2001	1,04	1,04
July 4, 2001	1,04	1,04
July 5, 2001	1,03	1,03
July 6, 2001	1,03	1,03
July 7, 2001	1,03	1,03

This query provides both the first and last available end-of-day exchange rates for the specified period. The former is provided through the MDX *OpeningPeriod* function; the latter is provided through a cube aggregate function. You could further extend the query to identify the difference or variance in exchange rates across the opening and closing of the period.

Chapter 9 Quick Reference

To	Do this
Retrieve the periods-to-date for any specified period	Use the *PeriodsToDate* function to return a set of sibling members from the same level as a given member, starting with the first sibling and ending with the given member, as constrained by a specified level of a calendar hierarchy. For example, the following query retrieves the periods-to-date over the calendar year for each of the Month members along the *ROWS* axis to calculate a year-to-date total for reseller sales:

```
WITH
MEMBER [Measures].[Year to Date Reseller Sales] AS
    Aggregate(
        PeriodsToDate(
            [Date].[Calendar].[Calendar Year],
            [Date].[Calendar].CurrentMember
            ),
        ([Measures].[Reseller Sales Amount])
        )
SELECT
    {
        ([Measures].[Reseller Sales Amount]),
        ([Measures].[Year to Date Reseller Sales])
        } ON COLUMNS,
    {[Date].[Calendar].[Month].Members} ON ROWS
FROM [Step-by-Step]
```

To	Do this
Retrieve the periods-to-date for a year	Use the *Ytd* function to return a set of sibling members from the same level as a given member, starting with the first sibling and ending with the given member, as constrained by the Year level of a calendar hierarchy. For example, the following query retrieves the year-to-date periods for each of the Month members along the *ROWS* axis to calculate a year-to-date total for reseller sales:

```
WITH
MEMBER [Measures].[Year to Date Reseller Sales] AS
    Aggregate(
        Ytd([Date].[Calendar].CurrentMember),
        ([Measures].[Reseller Sales Amount])
        )
SELECT
    {
        ([Measures].[Reseller Sales Amount]),
        ([Measures].[Year to Date Reseller Sales])
        } ON COLUMNS,
    {[Date].[Calendar].[Month].Members} ON ROWS
FROM [Step-by-Step]
```

For quarter-to-date, month-to-date, and week-to-date calculations, use the *Qtd*, *Mtd*, and *Wtd* functions, respectively, in a similar manner.

Retrieve a number of prior periods	Use the *LastPeriods* function to retrieve a set of members up to and including a specified member. For example, the following query retrieves the last three months for each of the Month members along the *ROWS* axis to calculate a rolling three-month average for reseller sales:

```
WITH
MEMBER [Measures].[Three Month Avg Reseller Sales Amount] AS
    Avg(
        LastPeriods(
            3,
            [Date].[Calendar].CurrentMember
            ),
        ([Measures].[Reseller Sales Amount])
        )
SELECT
    {
        ([Measures].[Reseller Sales Amount]),
        ([Measures].[Three Month Avg Reseller Sales Amount])
        } ON COLUMNS,
    {[Date].[Calendar].[Month].Members} ON ROWS
FROM [Step-by-Step]
```

To	Do this
Retrieve a parallel member	Use the *ParallelPeriod* function to identify a member from a prior period in the same relative position as a specified member. For example, the following query retrieves prior period reseller sales for each of the Month members along the *ROWS* axis:

```
WITH
MEMBER [Measures].[Prior Period Reseller Sales Amount] AS
    (
        ParallelPeriod(
            [Date].[Calendar].[Calendar Year],
            1,
            [Date].[Calendar].CurrentMember
            ),
        [Measures].[Reseller Sales Amount]
        )
    ,FORMAT="Currency"
SELECT
    {
        ([Measures].[Reseller Sales Amount]),
        ([Measures].[Prior Period Reseller Sales Amount])
        } ON COLUMNS,
    {
        Descendants(
            [Date].[Calendar].[Calendar Year].[CY 2003],
            [Date].[Calendar].[Month],
            SELF
            )
         } ON ROWS
FROM [Step-by-Step]
```

Retrieve the opening period or closing period	Use the *OpeningPeriod* or *ClosingPeriod* functions, respectively. For example, the following query employs the *OpeningPeriod* function to retrieve the exchange rate for the first day in each period:

```
WITH
MEMBER [Measures].[First Child Rate] AS
    (
        OpeningPeriod(
            [Date].[Calendar].[Date],
            [Date].[Calendar].CurrentMember
            ),
             [Measures].[End of Day Rate]
            )
       ,FORMAT="Standard"
SELECT
    {
        ([Measures].[First Child Rate]),
        ([Measures].[End of Day Rate])
        } ON COLUMNS,
    {[Date].[Calendar].Members} ON ROWS
FROM [Step-by-Step]
WHERE ([Destination Currency].[Destination Currency].[Euro])
```