



Creating Reports with SQL Queries

I'm convinced you can combine this with reporting integrity and accuracy.

—Jack Brickhouse

With the data warehouse completed and filled with data, it is time to realize some of the fruits of your efforts by making your first preliminary reports.

In this chapter, we take a look at the process of creating report code using the SQL programming language and some of the decisions that you have to make regarding this code. We walk you through the process of creating report queries from start to finish.

A data warehouse can have thousands of reports against it, and all reports have aspects in common:

- Reports are viewed with user applications such as Excel, PowerPivot, or Reporting Server Reports.
- The underlying code is most often written in SQL or MDX (query languages used for relational and OLAP databases).
- All report code should be consistent and well formed.
- Abstraction layers are used to keep code maintenance costs low.

Reporting queries can become complex very quickly, and complex queries can be overwhelming for developers who do not do a lot SQL programming; therefore, we do what we can to keep them simple. And, to make things more understandable, we break the SQL code into bite-size chunks and break down each of the features that typically make up a standard SQL report query. Using the methods in this chapter, you will soon be writing polished, professional, and accurate queries.

■ **Note** Examples in this chapter are SQL-based. MDX is the other common reporting language in use. We cover MDX next, in Chapter 14.

Identifying the Data

The first step is to determine the type of report your client needs. This falls into the category of “identifying the data.” After having been through the interview process with your client, you should have at least some idea of what they are looking for.

At this point, it is common for the client to be somewhat unsure of what they need. But after they have seen an initial report, it can help them be more specific. This process becomes more exact after you have presented your preliminary versions and examples to them. After your first reports, you will likely be refining them in versions 2, 3, or even 4. This is part of the process of learning what questions to ask before diving into creating the first version of the report.

We recommend getting started by creating a commented header section at the beginning of each SQL script. The comment might look something like the one shown in Listing 13-1.

Listing 13-1. A Script Header

```

/*****
Title:SalesByTitlesByDates
Description: Which titles were sold on which dates
Developer:RRoot
Date: 6/1/2012

Change Log: Who, When, What
CMason,6/2/2013,fixed numerous grammatical errors
*****/

```

The script header is similar to what we have used in the past, but most companies have their own standards of what information they require to be inserted into this header. If the company does not already have a standard for this, now would be a good time to establish one. Once you have a header outlining what you want to accomplish, you need to locate the data within the data warehouse.

Listing 13-2 is a very simple SELECT statement against the fact table from which most reports will originate. Notice that we have formatted the SELECT statement to be more legible by stacking the column listings in a vertical fashion. Over the years Microsoft seems to have settled on this being a best practice, and we agree that it does make things easier to read when you have to go through a large amount of code.

Listing 13-2. A Basic Starter Query

```

SELECT
    OrderNumber
    , OrderDateKey
    , TitleKey
    , StoreKey
    , SalesQuantity
FROM DWPubsSales.dbo.FactSales

```

One simple addition that makes reporting easier long-term is using fully qualified names for objects. This example includes not only the name of the table but its database name of DWPubsSales and schema name of DBO as well. Maintenance on reports includes tracking which reports are connected to which databases and database objects. Using fully qualified names in your SQL queries can help with this process and is a simple addition that takes little time to implement. Besides, you also get a small gain in performance, because the database engine does not have to resolve the object name implicitly. As shown in Figure 13-1, the results you get back are not particularly pleasing to the eye.

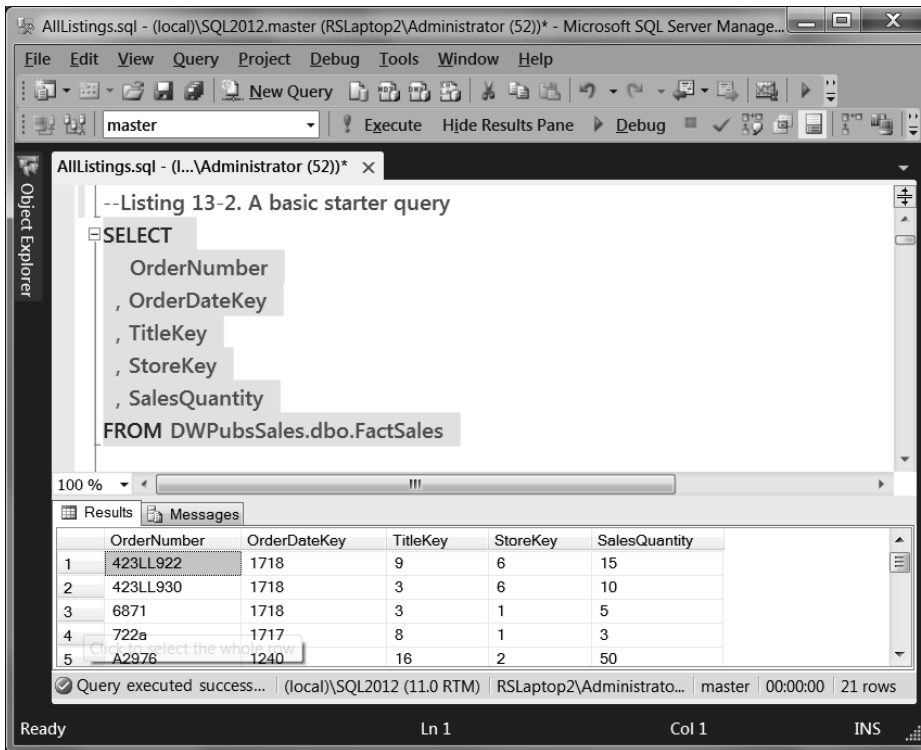


Figure 13-1. The results of the basic query

Additionally, it is not easy to understand. Notice that the order dates, titles, and stores show only numeric key values. Clearly, these reports are not quite user friendly yet, but they will be as soon as you enhance your query with more information. Let's look at some ways of doing so in the following sections.

Joining Table Data

In Listing 13-3 you can see an example of the basic query having been modified to include the title ID and title name from the DimTitles table. Note that you need to fully identify any columns that appear in both tables, like the TitleKey, by prefixing them with the table name. For clarity and code maintenance, we also prefix the TitleID and TitleName columns.

Listing 13-3. Adding a Table to the Query

```
SELECT
    DWPubsSales.dbo.DimTitles.TitleId
, DWPubsSales.dbo.DimTitles.TitleName
, OrderNumber
, OrderDateKey
, DWPubsSales.dbo.FactSales.TitleKey
```

```

, StoreKey
, SalesQuantity

FROM DWPubsSales.dbo.FactSales
JOIN DWPubsSales.dbo.DimTitles
ON DWPubsSales.dbo.FactSales.TitleKey = DWPubsSales.dbo.DimTitles.TitleKey

```

When this query is run, you receive data not only from the FactSales table, but also the DimTitles table, as shown in Figure 13-2. The database engine knows that you want data from both tables because of the JOIN operator in the FROM clause, as well as how the tables are linked because of the ON operator that compares the title key in both tables.

Listing 13-3. Adding an additional table to the query

```

SELECT
    DWPubsSales.dbo.DimTitles.TitleId
    , DWPubsSales.dbo.DimTitles.TitleName
    , OrderNumber
    , OrderDateKey

```

	TitleId	TitleName	OrderNum...	OrderDate...	TitleK...	StoreK...	SalesQuan...
1	MC3021	The Gourmet Microwave	423LL922	1718	9	6	15
2	BU1032	The Busy Executive's Database Guide	423LL930	1718	3	6	10
3	BU1032	The Busy Executive's Database Guide	6871	1718	3	1	5
4	PS2091	Is Anger the Enemy?	722a	1717	8	1	3
5	PC8888	Secrets of Silicon Valley	A2976	1240	16	2	50
6	PS2091	Is Anger the Enemy?	D4482	1718	8	3	10
7	PS2091	Is Anger the Enemy?	N914008	1718	8	4	20
8	MC3021	The Gourmet Microwave	N914014	1718	9	4	25
9	TC4203	Fifty Years in Buckingham Palace Kitchens	P2121	897	6	3	20
10	TC7777	Sushi. Anyone?	P2121	897	7	3	20

Query executed successfully. (local)\SQL2012 (11.0 RTM) | RSLaptop2\Administrato... | master | 00:00:00 | 21 rows

Figure 13-2. Adding the TitleId and TitleName columns to the results

Tip Notice in Figure 13-2 that we “stack” queries in the Management Studio window so to retain the previous working version of our query as well as the new version. It is a good idea to keep multiple versions in the window, highlighting one execute at a time. Doing so allows you to easily revert to an earlier working iteration in the event of trouble.

It is recommended to use fully qualified names to identify your tables, but a very long query can get quite tedious to both read and write. One solution is to use table aliases to represent fully qualified names. In Listing 13-4, you can see the use of the AS keyword to create a table alias. Using the AS keyword makes code much clearer than creating table aliases without it. (That is in contrast to simply putting a space after the table name and then typing in more letters to create the alias.)

Listing 13-4. Creating Table Aliases with the AS Keyword

```

SELECT
    FS.TitleKey
  , DT.TitleName
  , OrderNumber
  , OrderDateKey
  , StoreKey
  , SalesQuantity
FROM DWPubsSales.dbo.FactSales AS FS
INNER JOIN DWPubsSales.dbo.DimTitles AS DT
    ON FS.TitleKey = DT.TitleKey

```

Ordering Results

Another improvement for your report may be reordering the column listings so that they make more sense to the end user. Most modern reporting software can also modify the display of the columns in the user interface, but here we are dealing with just the raw SQL code. Therefore, modifying this display is a simple matter of moving the columns in the select list to their more appropriate positions.

Since this report is about titles, we have moved the title information to the top of the select. We have also placed the TitleName in the first column in the select list, because this provides the most user-friendly column results.

At the same time, we order the results by both title and dates (see Listing 13-5).

Listing 13-5. Reordering the Columns and Rows for Better Results

```

SELECT
    DT.TitleName
  , DT.TitleId
  , FS.TitleKey
  , OrderNumber
  , OrderDateKey
  , StoreKey
  , SalesQuantity
FROM DWPubsSales.dbo.FactSales AS FS
INNER JOIN DWPubsSales.dbo.DimTitles AS DT
    ON FS.TitleKey = DT.TitleKey
ORDER BY [TitleName], [OrderDateKey]

```

To improve the report, some developers like to include column aliases to further simplify or elaborate on the column definitions. Listing 13-6 further improves the outcome of our query by adding two column aliases. Notice the ORDER BY clause works with either the column alias such as [Title] or the column name such as [OrderDateKey].

Listing 13-6. Adding Column Aliases for Better Results

```

SELECT
    [Title] = DT.TitleName
  , DT.TitleId
  , [Internal Data Warehouse Id] = FS.TitleKey

```

```

, OrderNumber
, OrderDateKey
, StoreKey
, SalesQuantity
FROM DWPubsSales.dbo.FactSales AS FS
INNER JOIN DWPubsSales.dbo.DimTitles AS DT
    ON FS.TitleKey = DT.TitleKey
ORDER BY [Title], [OrderDateKey]

```

Figure 13-3 shows the results of this query. It includes the reordered columns and rows, as well as the aliased column names.

	Title	TitleId	Internal Data Warehouse Id	OrderNum...	OrderDate...	StoreK...	SalesQuan...
1	But Is It User Friendly?	PC1035	12	QA879.1	1238	6	30
2	Computer Phobic AND Non-P...	PS1372	14	P3087a	1245	4	20
3	Cooking with Computers: Surr...	BU1111	1	P723	1166	6	25
4	Emotional Security: A New Alg...	PS7777	5	P3087a	1245	4	25
5	Fifty Years in Buckingham Pal...	TC4203	6	P2121	897	3	20
6	Is Anger the Enemy?	PS2091	8	722a	1717	1	3
7	Is Anger the Enemy?	PS2091	8	QA7442.3	1717	2	75
8	Is Anger the Enemy?	PS2091	8	D4482	1718	3	10
9	Is Anger the Enemy?	PS2091	8	N914008	1718	4	20

Figure 13-3. Using column aliases and ORDER BY for our results

Currently our results include the OrderDateKey, but we do not have an understandable date format. This is because the date data is in the DimDates table, and we have not yet included that in our query. We do so in Listing 13-7.

Listing 13-7. Adding Data from the DimDates Table

```

SELECT
    [Title] = DT.TitleName
, DT.TitleId
, [Internal Data Warehouse Id] = FS.TitleKey
, OrderNumber
, OrderDateKey
, [OrderDate] = DD.[Date]
, StoreKey
, SalesQuantity

```

```

FROM DWPubsSales.dbo.FactSales AS FS
INNER JOIN DWPubsSales.dbo.DimTitles AS DT
    ON FS.TitleKey = DT.TitleKey

INNER JOIN DWPubsSales.dbo.DimDates AS DD
    ON FS.OrderDateKey = DD.DateKey

ORDER BY [Title], [OrderDate]

```

Figure 13-4 is an example of the results of adding data from DimDates.

Listing 13-7. Adding data from the DimDates table

```

-- Listing 13-7. Adding data from the DimDates table
SELECT
    [Title] = DT.TitleName
    , DT.TitleId
    , [Internal Data Warehouse Id] = FS.TitleKey
    , OrderNumber

```

	TitleId	Internal Data Warehouse Id	OrderNum...	OrderDate...	OrderDate	StoreK...	Sal
1	PC1035	12	QA879.1	1238	1993-05-22 00:00:00.000	6	30
2	ils: Beha...	PS1372	P3087a	1245	1993-05-29 00:00:00.000	4	20
3	ice Sheets	BU1111	P723	1166	1993-03-11 00:00:00.000	6	25
4	PS7777	5	P3087a	1245	1993-05-29 00:00:00.000	4	25
5	TC4203	6	P2121	897	1992-06-15 00:00:00.000	3	20
6	PS2091	8	722a	1717	1994-09-13 00:00:00.000	1	3
7	PS2091	8	QA7442.3	1717	1994-09-13 00:00:00.000	2	75
8	PS2091	8	D4482	1718	1994-09-14 00:00:00.000	3	10
9	PS2091	8	N914008	1718	1994-09-14 00:00:00.000	4	20

Query executed successfully. (local)\SQL2012 (11.0 RTM) RSLaptop2\Administrato... master 00:00:00 21 rows

Figure 13-4. Adding data from DimDates

The original request did not specifically ask for reports that group titles by their individual publishers, but it is likely to benefit the client. Adding publisher names to the queries would link a fourth table to the results, as shown in Listing 13-8.

Listing 13-8. Adding Publisher Names to the Results

```

SELECT
    DP.PublisherName
    , [Title] = DT.TitleName
    , DT.TitleId
    , [Internal Data Warehouse Id] = FS.TitleKey
    , OrderNumber
    , OrderDateKey
    , [OrderDate] = DD.[Date]
    , StoreKey
    , SalesQuantity
FROM DWPubsSales.dbo.FactSales AS FS
INNER JOIN DWPubsSales.dbo.DimTitles AS DT

```

```

ON FS.TitleKey = DT.TitleKey
INNER JOIN DWPubSales.dbo.DimDates AS DD
ON FS.OrderDateKey = DD.DateKey INNER JOIN DWPubSales.dbo.DimPublishers AS DP
ON DT.PublisherKey = DP.PublisherKey
ORDER BY DP.PublisherName, [Title], [OrderDate]

```

The results now include data from all four tables, as shown in Figure 13-5.

Listing 13-8. Adding publisher names to the results

```

SELECT
    DP.PublisherName
    , [Title] = DT.TitleName
    , DT.TitleId
    , [Internal Data Warehouse Id] = FS.TitleKey

```

	PublisherName	Title	TitleId	Internal...	OrderNum...	OrderDate...	OrderDate	Store
1	Algodata Infosystems	But Is It Use...	PC1035	12	QA879.1	1238	1993-05-22 00:00:00.000	6
2	Algodata Infosystems	Cooking wit...	BU1111	1	P723	1166	1993-03-11 00:00:00.000	6
3	Algodata Infosystems	Secrets of S...	PC8888	16	A2976	1240	1993-05-24 00:00:00.000	2
4	Algodata Infosystems	Straight Tal...	BU7832	10	QQ2299	1397	1993-10-28 00:00:00.000	5
5	Algodata Infosystems	The Busy E...	BU1032	3	423LL930	1718	1994-09-14 00:00:00.000	6
6	Algodata Infosystems	The Busy E...	BU1032	3	6871	1718	1994-09-14 00:00:00.000	1
7	Binnet & Hardley	Computer P...	PS1372	14	P3087a	1245	1993-05-29 00:00:00.000	4
8	Binnet & Hardley	Fifty Years i...	TC4203	6	P2121	897	1992-06-15 00:00:00.000	3
9	Binnet & Hardley	Onions, Lee...	TC3218	15	P2121	897	1992-06-15 00:00:00.000	3

Query executed successfully. (local)\SQL2012 (11.0 RTM) RSLaptop2\Administrato... master 00:00:00 21 rows

Figure 13-5. Adding data from *DimPublishers*

If our current query contains columns that are unimportant to our report, they can easily be excluded by either deleting them or commenting them out. For this report we will not need the *TitleKey*, *OrderNumber*, *OrderDateKey*, or *StoreKey*. We removed these four columns from the select list in Listing 13-9 by commenting them out.

Listing 13-9. Removing Columns Not Needed for Our Query

```

SELECT
    DP.PublisherName
    , [Title] = DT.TitleName
    , DT.TitleId

    --, [Internal Data Warehouse Id] = FS.TitleKey
    --, OrderNumber
    --, OrderDateKey

    , [OrderDate] = DD.[Date]

    --, StoreKey

    , SalesQuantity
FROM DWPubSales.dbo.FactSales AS FS

```



```

INNER JOIN DWPubsSales.dbo.DimTitles AS DT
  ON FS.TitleKey = DT.TitleKey
INNER JOIN DWPubsSales.dbo.DimDates AS DD
  ON FS.OrderDateKey = DD.DateKey
INNER JOIN DWPubsSales.dbo.DimPublishers AS DP
  ON DT.PublisherKey = DP.PublisherKey
ORDER BY DP.PublisherName, [Title], [OrderDate]

```

■ **Tip** Leaving columns in your original code and simply commenting them out, rather than completely removing them, is often helpful for troubleshooting and validating your code. After the code is stabilized, they can then be removed permanently. For instructional purposes and clarity, we remove them from the future listings in this chapter.

Formatting Results Using SQL Functions

To make the data more user-friendly, we use the SQL function `CONVERT` to change the data into a string of characters with the typical United States presentation. The `CONVERT` function was designed by Microsoft to do just this. Microsoft has also incorporated a set of numbers to determine which U.S. format to use. For example, format number 110 gives a date with the *day - month - year* format, while format number 101 gives you a date with the *day/month/year* format (Listing 13-10).

Listing 13-10. Using the `CONVERT` Function

```

SELECT
  DP.PublisherName
, [Title] = DT.TitleName
, [TitleId] = DT.TitleId
, [OrderDate] = CONVERT(varchar(50), [Date], 101)
, SalesQuantity
FROM DWPubsSales.dbo.FactSales AS FS
INNER JOIN DWPubsSales.dbo.DimTitles AS DT
  ON FS.TitleKey = DT.TitleKey
INNER JOIN DWPubsSales.dbo.DimDates AS DD
  ON FS.OrderDateKey = DD.DateKey
INNER JOIN DWPubsSales.dbo.DimPublishers AS DP
  ON DT.PublisherKey = DP.PublisherKey
ORDER BY DP.PublisherName, [Title], [OrderDate]

```

The results now include formatted dates, as shown in Figure 13-6.

Listing 13-10. Using the Convert function

```

SELECT
    DP.PublisherName
    , [Title] = DT.TitleName
    , [TitleId] = DT.TitleId
    , [OrderDate] = Convert(varchar(50), [Date], 101)
    , SalesQuantity
FROM DWPubsSales.dbo.FactSales AS FS
INNER JOIN DWPubsSales.dbo.DimTitles AS DT

```

	PublisherName	Title	TitleId	OrderDate	SalesQuantity
1	Algodata Infosystems	But Is It User Friendly?	PC1035	05/22/1993	30
2	Algodata Infosystems	Cooking with Computers: Surreptitious Balance She...	BU1111	03/11/1993	25
3	Algodata Infosystems	Secrets of Silicon Valley	PC8888	05/24/1993	50
4	Algodata Infosystems	Straight Talk About Computers	BU7832	10/28/1993	15
5	Algodata Infosystems	The Busy Executive's Database Guide	BU1032	09/14/1994	10
6	Algodata Infosystems	The Busy Executive's Database Guide	BU1032	09/14/1994	5

Query executed successfully. (local)\SQL2012 (11.0 RTM) | RSLaptop2\Administrato... | master | 00:00:00 | 21 rows

Figure 13-6. The current results with formatted dates

Filtering Results

Many reports filter data so that instead of showing all of the data in a table, you can select specific data to be made visible. This is done using the SQL `WHERE` clause, as shown in Listing 13-11.

Listing 13-11. Using a `WHERE` Clause to Filter the Results

```

SELECT
    [Title] = DT.TitleName
    , [TitleId] = DT.TitleId
    , [OrderDate] = Convert(varchar(50), [Date], 101)
    , SalesQuantity
FROM DWPubsSales.dbo.FactSales AS FS
JOIN DWPubsSales.dbo.DimTitles AS DT
    ON FS.TitleKey = DT.TitleKey
JOIN DWPubsSales.dbo.DimDates AS DD
    ON FS.OrderDateKey = DD.DateKey

WHERE [TitleId] = 'PS2091'

ORDER BY [Title], [Date]

```

The results now include only filtered data, as shown in Figure 13-7.

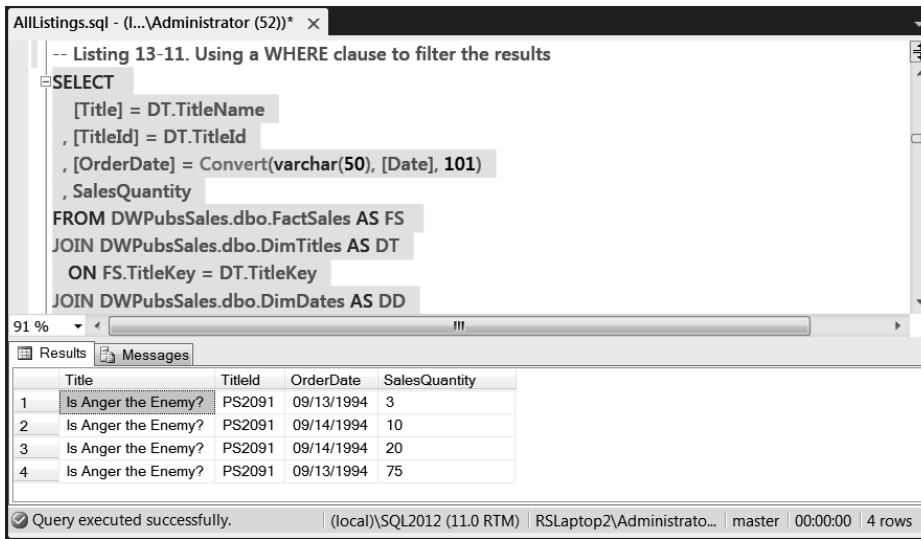


Figure 13-7. Results filtered by a given title ID

You may want to filter not only the title but also the date. To do this, you would change the WHERE clause to search for a particular date, as shown in Listing 13-12.

Listing 13-12. Using a WHERE Clause to Filter Based on a Given Date

```

SELECT
    DP.PublisherName
    , [Title] = DT.TitleName
    , [TitleId] = DT.TitleId
    , [OrderDate] = Convert(varchar(50), [Date], 101)
    , SalesQuantity
FROM DWPubsSales.dbo.FactSales AS FS
INNER JOIN DWPubsSales.dbo.DimTitles AS DT
    ON FS.TitleKey = DT.TitleKey
INNER JOIN DWPubsSales.dbo.DimDates AS DD
    ON FS.OrderDateKey = DD.DateKey
INNER JOIN DWPubsSales.dbo.DimPublishers AS DP
    ON DT.PublisherKey = DP.PublisherKey

WHERE [Date] = '09/13/1994'

ORDER BY DP.PublisherName, [Title], [OrderDate]

```

The results will now only show data where the exact date is found (Figure 13-8).

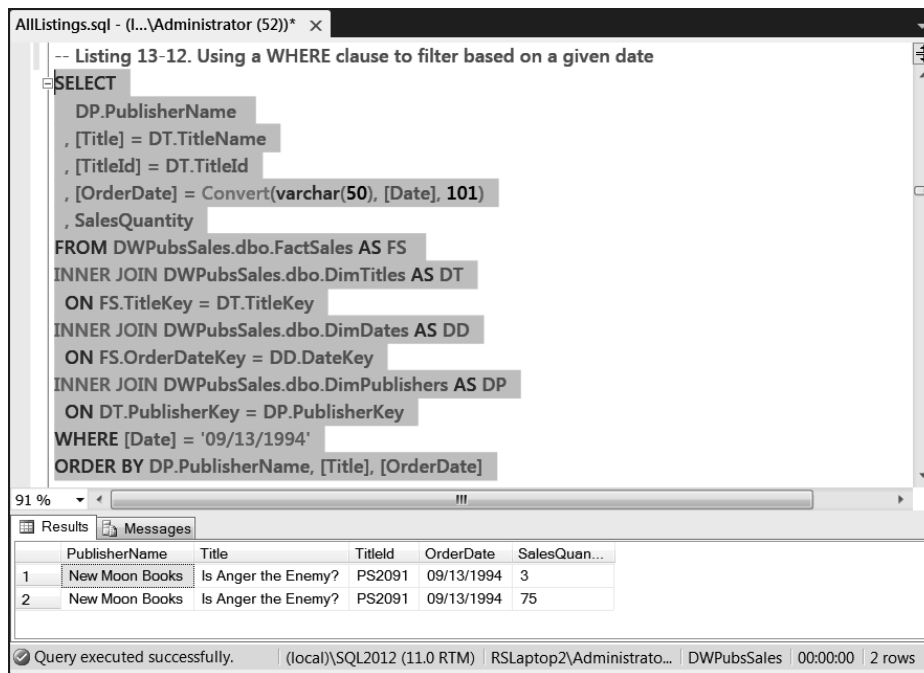


Figure 13-8. Filtering results based on a given date

When you are working with filters, it is often necessary to add more flexibility in your queries by using tools like the SQL wildcard searches. One classic example of a wildcard search is where you use the percentage sign wildcard symbol (%) in conjunction with the SQL LIKE operator to indicate that zero or more missing characters should be ignored for the purpose of a pattern match. In Listing 13-13 you can see an example of this where the query asks SQL Server to find all the data where the letters *PS* are followed by zero or more number of characters.

Listing 13-13. Filtering Results Based on Title ID with a Wildcard Symbol

```
SELECT
    DP.PublisherName
    , [Title] = DT.TitleName
    , [TitleId] = DT.TitleId
    , [OrderDate] = CONVERT(varchar(50), [Date], 101)
    , SalesQuantity
FROM DWPubsSales.dbo.FactSales AS FS
INNER JOIN DWPubsSales.dbo.DimTitles AS DT
    ON FS.TitleKey = DT.TitleKey
INNER JOIN DWPubsSales.dbo.DimDates AS DD
    ON FS.OrderDateKey = DD.DateKey
INNER JOIN DWPubsSales.dbo.DimPublishers AS DP
    ON DT.PublisherKey = DP.PublisherKey
WHERE [TitleId] LIKE 'PS%' -- % means zero or more characters
ORDER BY DP.PublisherName, [Title], [OrderDate]
```

Now the results include only data with a prefix of *PS*, as shown in Figure 13-9.

Listing 13-13. Filtering results based on title ID with a wildcard symbol

```

SELECT
    DP.PublisherName
    , [Title] = DT.TitleName
    , [TitleId] = DT.TitleId
    , [OrderDate] = Convert(varchar(50), [Date], 101)
    , SalesQuantity

```

	PublisherName	Title	TitleId	OrderDate	SalesQuan...
1	Binnet & Hardley	Computer Phobic AND Non-Phobic Individuals: Beh...	PS1372	05/29/1993	20
2	New Moon Books	Emotional Security: A New Algorithm	PS7777	05/29/1993	25
3	New Moon Books	Is Anger the Enemy?	PS2091	09/13/1994	75
4	New Moon Books	Is Anger the Enemy?	PS2091	09/13/1994	3
5	New Moon Books	Is Anger the Enemy?	PS2091	09/14/1994	10
6	New Moon Books	Is Anger the Enemy?	PS2091	09/14/1994	20
7	New Moon Books	Life Without Fear	PS2106	05/29/1993	25
8	New Moon Books	Prolonged Data Deprivation: Four Case Studies	PS3333	05/29/1993	15

Query executed successfully. | (local)\SQL2012 (11.0 RTM) | RSLaptop2\Administrato... | DWPubsSales | 00:00:00 | 8 rows

Figure 13-9. Results filtered with a wildcard search

The LIKE operator is certainly useful, and you will find plenty of opportunities to use it. Here are two other operators we have found useful over the years. The first is the IN operator. It filters results based on a list of possible choices. When a row has data that matches one of the listed items, the row is returned as part of the result set. Listing 13-14 shows an example of the IN operator being used.

Listing 13-14. Using the IN Operator

```

SELECT
    DP.PublisherName
    , [Title] = DT.TitleName
    , [TitleId] = DT.TitleId
    , [OrderDate] = CONVERT(varchar(50), [Date], 101)
    , SalesQuantity
FROM DWPubsSales.dbo.FactSales AS FS
INNER JOIN DWPubsSales.dbo.DimTitles AS DT
    ON FS.TitleKey = DT.TitleKey
INNER JOIN DWPubsSales.dbo.DimDates AS DD
    ON FS.OrderDateKey = DD.DateKey
INNER JOIN DWPubsSales.dbo.DimPublishers AS DP
    ON DT.PublisherKey = DP.PublisherKey
WHERE [Date] IN ( '09/13/1994' , '05/29/1993' )
ORDER BY DP.PublisherName, [Title], [OrderDate]

```

Figure 13-10 shows the results of the query.



Figure 13-10 is a screenshot of the SQL Server Enterprise Manager interface. The top pane shows a query window titled 'AllListings.sql - (local)\Administrator (52))' containing the following SQL query:

```
-- Listing 13-14. Using the IN operator
SELECT
    DP.PublisherName
    , [Title] = DT.TitleName
    , [TitleId] = DT.TitleId
    , [OrderDate] = Convert(varchar(50), [Date], 101)
    , SalesQuantity
FROM DWPubsSales.dbo.FactSales AS FS
INNER JOIN DWPubsSales.dbo.DimTitles AS DT
```

The bottom pane shows the 'Results' tab with a table containing 6 rows of data. The table has the following columns: PublisherName, Title, TitleId, OrderDate, and SalesQuantity.

	PublisherName	Title	TitleId	OrderDate	SalesQuantity
1	Binnet & Hardley	Computer Phobic AND Non-Phobic Individuals: Beh...	PS1372	05/29/1993	20
2	New Moon Books	Emotional Security: A New Algorithm	PS7777	05/29/1993	25
3	New Moon Books	Is Anger the Enemy?	PS2091	09/13/1994	3
4	New Moon Books	Is Anger the Enemy?	PS2091	09/13/1994	75
5	New Moon Books	Life Without Fear	PS2106	05/29/1993	25
6	New Moon Books	Prolonged Data Deprivation: Four Case Studies	PS3333	05/29/1993	15

The status bar at the bottom indicates: 'Query executed successfully. (local)\SQL2012 (11.0 RTM) | RSLaptop2\Administrato... | DWPubsSales | 00:00:00 | 6 rows'.

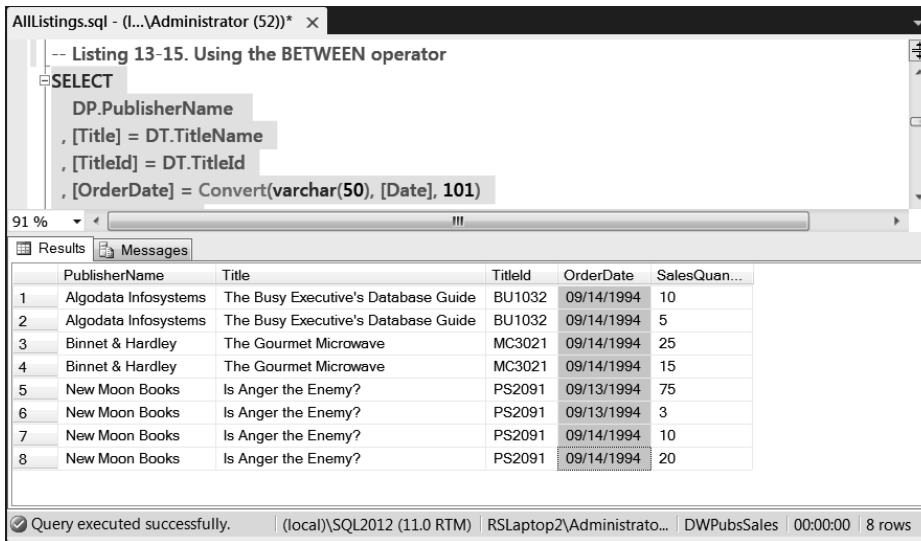
Figure 13-10. Results filtered with the *IN* operator

The second operator that is worthy of mentioning is the *BETWEEN* operator. It finds data that is within a range, such as numeric or alphabetical. Listing 13-15 demonstrates this operator using dates.

Listing 13-15. Using the *BETWEEN* Operator

```
SELECT
    DP.PublisherName
    , [Title] = DT.TitleName
    , [TitleId] = DT.TitleId
    , [OrderDate] = CONVERT(varchar(50), [Date], 101)
    , SalesQuantity
FROM DWPubsSales.dbo.FactSales AS FS
INNER JOIN DWPubsSales.dbo.DimTitles AS DT
    ON FS.TitleKey = DT.TitleKey
INNER JOIN DWPubsSales.dbo.DimDates AS DD
    ON FS.OrderDateKey = DD.DateKey
INNER JOIN DWPubsSales.dbo.DimPublishers AS DP
    ON DT.PublisherKey = DP.PublisherKey
WHERE [Date] BETWEEN '09/13/1994' AND '09/14/1994'
ORDER BY DP.PublisherName, [Title], [OrderDate]
```

As you can see in Figure 13-11, all rows of data are returned that are within the range of dates specified by the *BETWEEN* operator.



Listing 13-15. Using the BETWEEN operator

```

SELECT
    DP.PublisherName
    , [Title] = DT.TitleName
    , [TitleId] = DT.TitleId
    , [OrderDate] = Convert(varchar(50), [Date], 101)

```

	PublisherName	Title	TitleId	OrderDate	SalesQuan...
1	Algodata Infosystems	The Busy Executive's Database Guide	BU1032	09/14/1994	10
2	Algodata Infosystems	The Busy Executive's Database Guide	BU1032	09/14/1994	5
3	Binnet & Hardley	The Gourmet Microwave	MC3021	09/14/1994	25
4	Binnet & Hardley	The Gourmet Microwave	MC3021	09/14/1994	15
5	New Moon Books	Is Anger the Enemy?	PS2091	09/13/1994	75
6	New Moon Books	Is Anger the Enemy?	PS2091	09/13/1994	3
7	New Moon Books	Is Anger the Enemy?	PS2091	09/14/1994	10
8	New Moon Books	Is Anger the Enemy?	PS2091	09/14/1994	20

Query executed successfully. (local)\SQL2012 (11.0 RTM) RSLaptop2\Administrato... DWPubsSales 00:00:00 8 rows

Figure 13-11. Results filtered by with the BETWEEN operator

Tip Note that with the BETWEEN operator the results are inclusive. This is not always true of all SQL operators, so whenever you use an operator that defines results based on two points, you should always test to check whether the results are inclusive or exclusive.

You can also combine operators in the WHERE clause to fine-tune your result set. Listing 13-16 shows an example of using both the BETWEEN and LIKE operators in the same WHERE statement.

Listing 13-16. Combining Multiple Operators in a WHERE Statement

```

SELECT
    DP.PublisherName
    , [Title] = DT.TitleName
    , [TitleId] = DT.TitleId
    , [OrderDate] = Convert(varchar(50), [Date], 101)
    , SalesQuantity
FROM DWPubsSales.dbo.FactSales AS FS
INNER JOIN DWPubsSales.dbo.DimTitles AS DT
    ON FS.TitleKey = DT.TitleKey
INNER JOIN DWPubsSales.dbo.DimDates AS DD
    ON FS.OrderDateKey = DD.DateKey
INNER JOIN DWPubsSales.dbo.DimPublishers AS DP
    ON DT.PublisherKey = DP.PublisherKey
WHERE
    [Date] BETWEEN '09/13/1994' AND '09/14/1994'
    AND
    [TitleId] LIKE 'PS%'
ORDER BY DP.PublisherName, [Title], [OrderDate]

```

Figure 13-12 shows the results of this query.

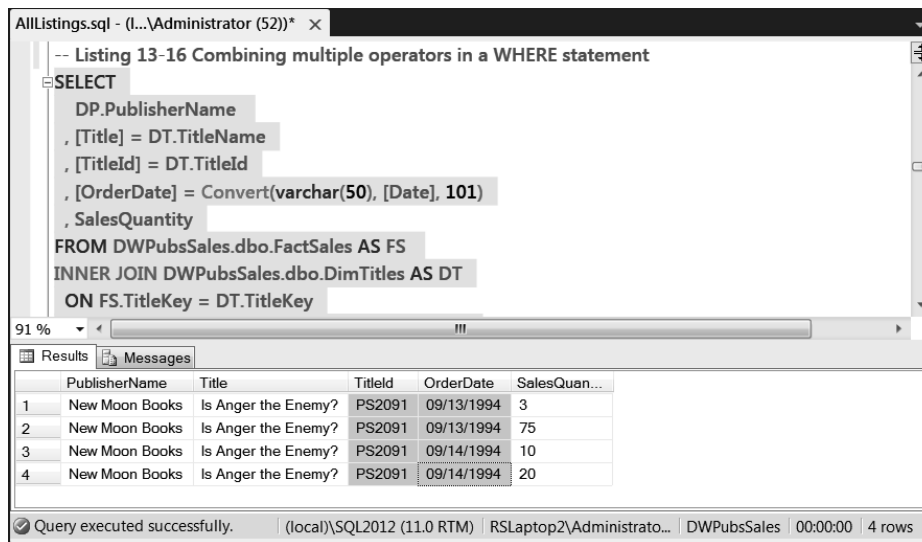


Figure 13-12. Results filtered by multiple operators

Adding Dynamic Filters with Parameters

In many cases, report builders prefer to set the wildcard or date values themselves. You can allow this by defining a set of parameters to further customize the report results. This can be programmed by creating SQL variables to use for report parameters. Listing 13-17 shows an example.

Listing 13-17. Adding Variables to Your Query

```
DECLARE
    @StartDate datetime = '09/13/1994'
    , @EndDate datetime = '09/14/1994'
    , @Prefix nVarchar(3) = 'PS%'

SELECT
    DP.PublisherName
    , [Title] = DT.TitleName
    , [TitleId] = DT.TitleId
    , [OrderDate] = CONVERT(varchar(50), [Date], 101)
    , SalesQuantity
FROM DWPubsSales.dbo.FactSales AS FS
INNER JOIN DWPubsSales.dbo.DimTitles AS DT
    ON FS.TitleKey = DT.TitleKey
INNER JOIN DWPubsSales.dbo.DimDates AS DD
    ON FS.OrderDateKey = DD.DateKey
INNER JOIN DWPubsSales.dbo.DimPublishers AS DP
    ON DT.PublisherKey = DP.PublisherKey
WHERE
    [Date] BETWEEN @StartDate AND @EndDate
```



```

AND
[TitleId] LIKE @Prefix
ORDER BY DP.PublisherName, [Title], [OrderDate]

```

The report parameters can now be used in the WHERE clause by replacing the hard-coded values with our variables. Now, if we run our report query, we will see rows of data based on whatever dates and prefix we supply to the query, as shown in Figure 13-13.

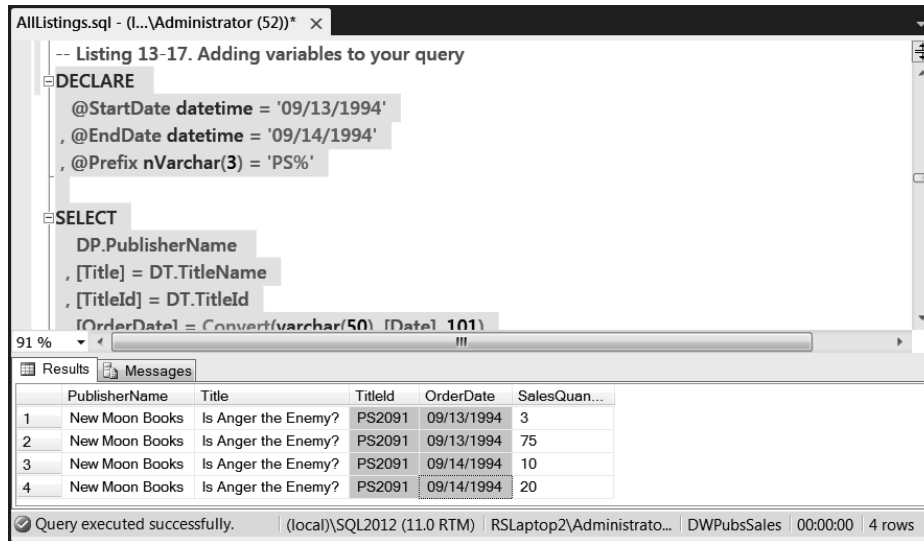


Figure 13-13. Results filtered with parameters

You can expect that some clients want the option of showing every row on some occasions while showing only some of the rows at other times. There are a number ways to do this, but one way is to add a parameter that serves as a flag to indicate whether you want to show all the results or just filtered results.

In Listing 13-18, you can see that we have added a parameter called @ShowAll. This parameter's job is to toggle between showing all the results or only some of the results.

Listing 13-18. Adding a Parameter Flag to Show All Data as Needed

```

DECLARE

    @ShowAll nvarchar(4) = 'True'

    , @StartDate datetime = '09/13/1994'
    , @EndDate datetime = '09/14/1994'
    , @Prefix nvarchar(3) = 'PS%'
SELECT
    DP.PublisherName
    , [Title] = DT.TitleName
    , [TitleId] = DT.TitleId
    , [OrderDate] = CONVERT(varchar(50), [Date], 101)
    , SalesQuantity

```

```

FROM DWPubsSales.dbo.FactSales AS FS
INNER JOIN DWPubsSales.dbo.DimTitles AS DT
    ON FS.TitleKey = DT.TitleKey
INNER JOIN DWPubsSales.dbo.DimDates AS DD
    ON FS.OrderDateKey = DD.DateKey
INNER JOIN DWPubsSales.dbo.DimPublishers AS DP
    ON DT.PublisherKey = DP.PublisherKey
WHERE

```

```
@ShowAll = 'True'
```

```
OR
```

```
[Date] BETWEEN @StartDate AND @EndDate
```

```
AND
```

```
[TitleId] LIKE @Prefix
```

```
ORDER BY DP.PublisherName, [Title], [OrderDate]
```

When this query is executed, the results are similar to those shown in Figure 13-14.

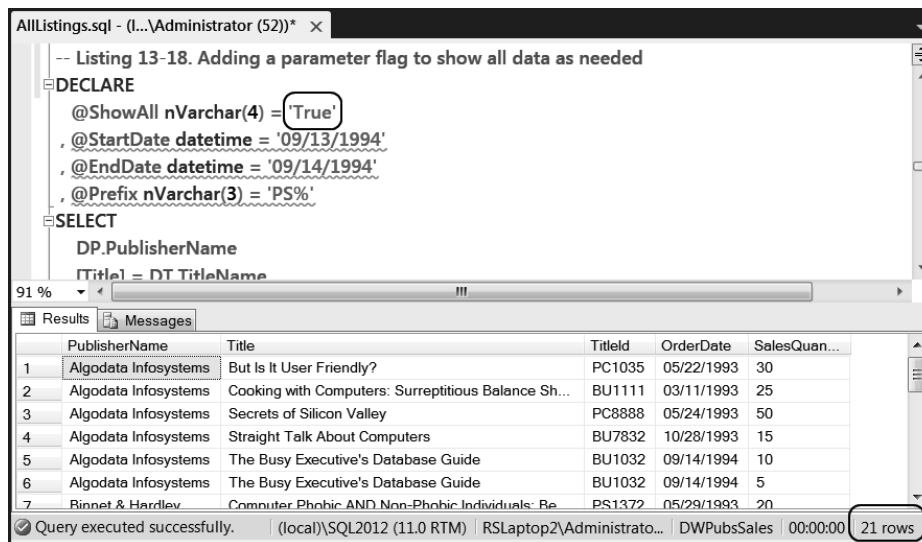


Figure 13-14. Using a parameter flag to show all results

Adding Aggregations

One common request is for reports to display totals and subtotals. In SQL queries, this can be accomplished by using aggregate functions. To demonstrate this, let's go ahead and summarize the sales quantities using the SUM aggregate function, as shown in Listing 13-19.

Listing 13-19. Adding Aggregate Values to Our Results

```

DECLARE
    @ShowAll nVarChar(4) = 'False'
    , @StartDate datetime = '09/13/1994'
    , @EndDate datetime = '09/14/1994'
    , @Prefix nVarChar(3) = 'PS%'

```

```

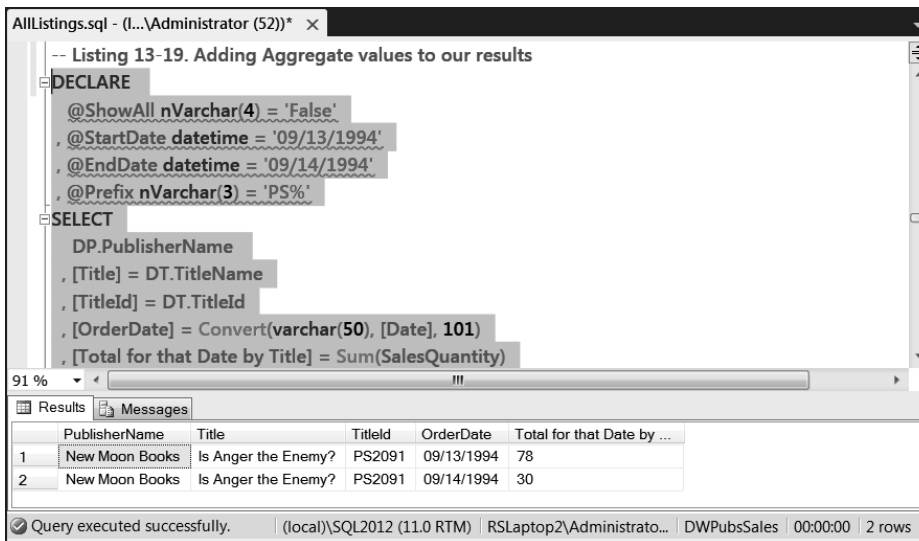
SELECT
    DP.PublisherName
    , [Title] = DT.TitleName
    , [TitleId] = DT.TitleId
    , [OrderDate] = CONVERT(varchar(50), [Date], 101)
    , [Total for that Date by Title] = SUM(SalesQuantity)
FROM DWPubsSales.dbo.FactSales AS FS
INNER JOIN DWPubsSales.dbo.DimTitles AS DT
    ON FS.TitleKey = DT.TitleKey
INNER JOIN DWPubsSales.dbo.DimDates AS DD
    ON FS.OrderDateKey = DD.DateKey
INNER JOIN DWPubsSales.dbo.DimPublishers AS DP
    ON DT.PublisherKey = DP.PublisherKey
WHERE
    @ShowAll = 'True'
    OR
    [Date] BETWEEN @StartDate AND @EndDate
    AND
    [TitleId] LIKE @Prefix

GROUP BY
    DP.PublisherName
    , DT.TitleName
    , DT.TitleId
    , [Date]

ORDER BY DP.PublisherName, [Title], [OrderDate]

```

Figure 13-15 shows the results of the aggregations.



Listing 13-19. Adding Aggregate values to our results

```

-- Listing 13-19. Adding Aggregate values to our results
-- DECLARE
    @ShowAll nVarchar(4) = 'False'
    , @StartDate datetime = '09/13/1994'
    , @EndDate datetime = '09/14/1994'
    , @Prefix nVarchar(3) = 'PS%'
-- SELECT
    DP.PublisherName
    , [Title] = DT.TitleName
    , [TitleId] = DT.TitleId
    , [OrderDate] = Convert(varchar(50), [Date], 101)
    , [Total for that Date by Title] = Sum(SalesQuantity)

```

	PublisherName	Title	TitleId	OrderDate	Total for that Date by ...
1	New Moon Books	Is Anger the Enemy?	PS2091	09/13/1994	78
2	New Moon Books	Is Anger the Enemy?	PS2091	09/14/1994	30

Query executed successfully. | (local)\SQL2012 (11.0 RTM) | RSLaptop2\Administrato... | DWPubsSales | 00:00:00 | 2 rows

Figure 13-15. Results with aggregations

■ **Note** Often, reporting software will add the aggregations for you. When this happens, do not include the aggregate values in your SQL code. This is also true for parameters; some applications filter the results using parameters only after all the results have been returned to the application. A common example for both of these can be found within Microsoft Excel. Although it is possible to submit parameter values and consume aggregate data, you are usually better off letting Excel do the aggregation and filtering for you after all the data has been returned.

EXERCISE 13-1. CREATING REPORT QUERIES

In this exercise, you create your own custom report query similar to the one we just walked through. To make this process simpler, we do not include parameters at this point. You combine data from three tables, aggregate the values, and then format your results. If you build the query a little at a time, as described in this chapter, you will find your query will be completed in no time at all!

1. Review the data in the FactSales table by executing the following SQL code in a query window:

```
SELECT * from FactSales
```
2. Review the data in the DimStores table by executing the following SQL code in a query window:

```
SELECT * from DimStores
```
3. Review the data in the DimDates table by executing the following SQL code in a query window:

```
SELECT * FROM DimDates
```
4. Combine the data from these tables into a report query that returns store names, store IDs, order dates, and the sum quantity by store and date. Your results should look like Figure 13-16.

If you get stuck, the answers for this exercise are in the downloadable example files for this book. Look in the folder for Chapter 13.

	StoreName	StoreId	OrderDate	Total Qty by Store
1	Barnum's	7066	05/24/1993	50
2	Barnum's	7066	09/13/1994	75
3	Bookbeat	8042	03/11/1993	25
4	Bookbeat	8042	05/22/1993	30
5	Bookbeat	8042	09/14/1994	25
6	Doc-U-Mat: Quality Laundry and Books	7131	05/29/1993	85
7	Doc-U-Mat: Quality Laundry and Books	7131	09/14/1994	45
8	Eric the Read Books	6380	09/13/1994	3
9	Eric the Read Books	6380	09/14/1994	5
10	Fricative Bookshop	7896	02/21/1993	35
11	Fricative Bookshop	7896	10/28/1993	15
12	Fricative Bookshop	7896	12/12/1993	10
13	News & Brews	7067	06/15/1992	80
14	News & Brews	7067	09/14/1994	10

Figure 13-16. The results needed for the sales by stores report

In this exercise, you wrote a report query that captured sales data based on sales by stores. In the next exercise, you use this code to create a reporting stores procedure. But, we have a few other items to talk about first.

Using Subqueries

Multiple queries can be used at the same time to achieve a single result. There are a couple ways to do this, and using subqueries is a popular choice. Subqueries are often used in a WHERE clause to retrieve a set of values that can be examined conditionally. Another use for subqueries is to fill up the variables with data based on the results of the subquery. Listing 13-20 shows an example of this.

Listing 13-20. Fill from Your Querying Parameters with Subqueries

```
DECLARE
    @ShowAll nVarchar(4) = 'False'
    , @StartDate datetime = '09/13/1994'
    , @EndDate datetime = '09/14/1994'
    , @Prefix nVarchar(3) = 'PS%'
    , @AverageQty int

    SELECT @AverageQty = Avg(SalesQuantity) FROM DWPubsSales.dbo.FactSales

SELECT
    DP.PublisherName
    , [Title] = DT.TitleName
    , [TitleId] = DT.TitleId
    , [OrderDate] = CONVERT(varchar(50), [Date], 101)
    , [Total for that Date by Title] = Sum(SalesQuantity)

    , [Average Qty in the FactSales Table] = @AverageQty

FROM DWPubsSales.dbo.FactSales AS FS
INNER JOIN DWPubsSales.dbo.DimTitles AS DT
    ON FS.TitleKey = DT.TitleKey
INNER JOIN DWPubsSales.dbo.DimDates AS DD
    ON FS.OrderDateKey = DD.DateKey
INNER JOIN DWPubsSales.dbo.DimPublishers AS DP
    ON DT.PublisherKey = DP.PublisherKey
WHERE
    @ShowAll = 'True'
    OR
    [Date] BETWEEN @StartDate AND @EndDate
    AND
    [TitleId] LIKE @Prefix
GROUP BY
    DP.PublisherName
    , DT.TitleName
    , DT.TitleId
    , Convert(varchar(50), [Date], 101)
ORDER BY DP.PublisherName, [Title], [OrderDate]
```

Now we have report code that uses two SELECT statements. The primary query gets the report data, but the secondary SELECT statement acts as a subquery to the primary. The purpose of this subquery is to gather the average of all the sales quantity values in the fact table and then add that result to the next SELECT statement's results. If we were to use the AVERAGE function within the second SELECT statement, we would get only the average of the filtered values, not all of the quantity values in the fact sales table. Figure 13-17 shows the results.

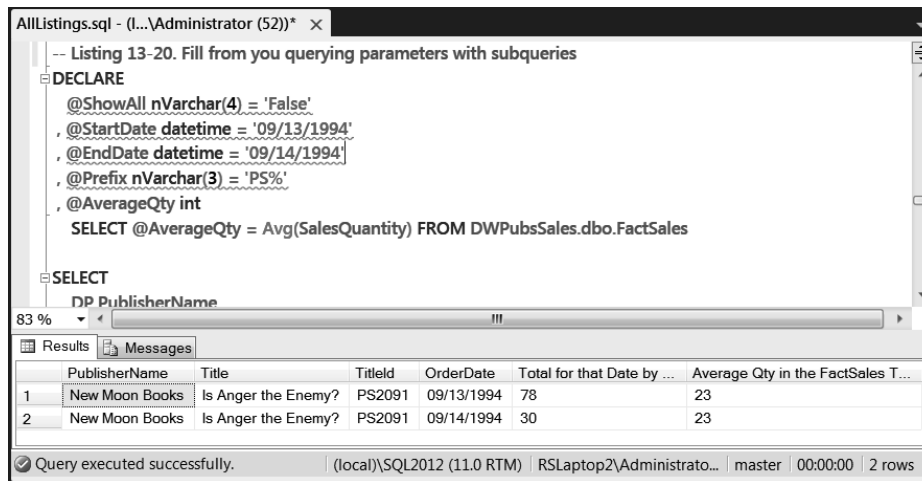


Figure 13-17. Results with parameters filled by a subquery

Creating KPI Queries

As reporting software has become more available, it is common for users to have to review many reports during their day-to-day work. This often results in information overload. One way to help alleviate this issue is to create some kind of indicator in your report that can be used to provide a quick determination of the value's meaning.

One way of accomplishing this is to provide key performance indicator (KPI) values within your result set. The CASE operator is a common method of including KPI's in a SQL statement. Listing 13-21 shows the use of this operator.

Listing 13-21. Adding KPIs

```
DECLARE
    @ShowAll nVarchar(4) = 'True'
    , @StartDate datetime = '09/13/1994'
    , @EndDate datetime = '09/14/1994'
    , @Prefix nVarchar(3) = 'PS%'
    , @AverageQty int
    SELECT @AverageQty = Avg(SalesQuantity) FROM DWPubsSales.dbo.FactSales

SELECT
    DP.PublisherName
    , [Title] = DT.TitleName
    , [TitleId] = DT.TitleId
    , [OrderDate] = CONVERT(varchar(50), [Date], 101)
```

```

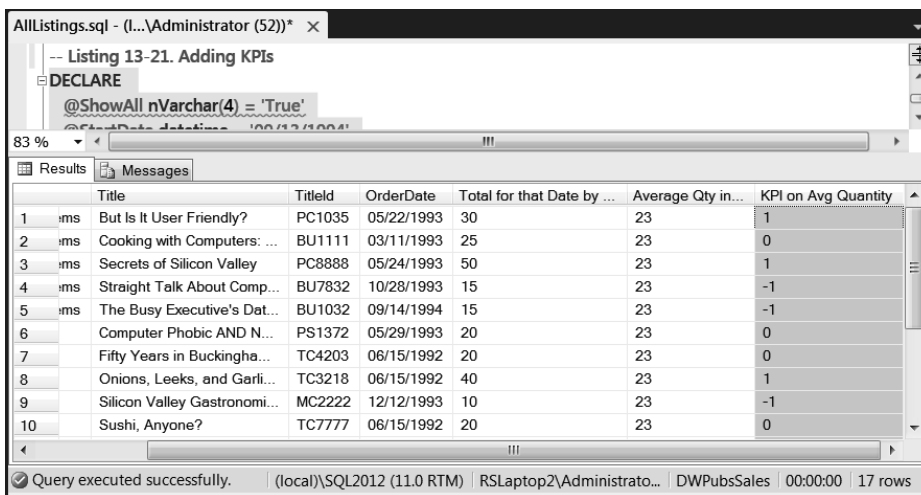
, [Total for that Date by Title] = Sum(SalesQuantity)
, [Average Qty in the FactSales Table] = @AverageQty

, [KPI on Avg Quantity] = CASE
WHEN Sum(SalesQuantity)
    between (@AverageQty- 5) and (@AverageQty+5) THEN 0
WHEN Sum(SalesQuantity) < (@AverageQty- 5) THEN -1
WHEN Sum(SalesQuantity) > (@AverageQty + 5) THEN 1
END

FROM DWPubsSales.dbo.FactSales AS FS
INNER JOIN DWPubsSales.dbo.DimTitles AS DT
    ON FS.TitleKey = DT.TitleKey
INNER JOIN DWPubsSales.dbo.DimDates AS DD
    ON FS.OrderDateKey = DD.DateKey
INNER JOIN DWPubsSales.dbo.DimPublishers AS DP
    ON DT.PublisherKey = DP.PublisherKey
WHERE
    @ShowAll = 'True'
OR
    [Date] BETWEEN @StartDate AND @EndDate
AND
    [TitleId] LIKE @Prefix
GROUP BY
    DP.PublisherName
    , DT.TitleName
    , DT.TitleId
    , CONVERT(varchar(50), [Date], 101)
ORDER BY DP.PublisherName, [Title], [OrderDate]

```

Figure 13-18 shows the results with KPIs.



Query executed successfully. (local)\SQL2012 (11.0 RTM) RSLaptop2\Administrato... DWPubsSales 00:00:00 17 rows

	Title	TitleId	OrderDate	Total for that Date by ...	Average Qty in...	KPI on Avg Quantity
1	ms But Is It User Friendly?	PC1035	05/22/1993	30	23	1
2	ms Cooking with Computers: ...	BU1111	03/11/1993	25	23	0
3	ms Secrets of Silicon Valley	PC8888	05/24/1993	50	23	1
4	ms Straight Talk About Comp...	BU7832	10/28/1993	15	23	-1
5	ms The Busy Executive's Dat...	BU1032	09/14/1994	15	23	-1
6	Computer Phobic AND N...	PS1372	05/29/1993	20	23	0
7	Fifty Years in Buckingha...	TC4203	06/15/1992	20	23	0
8	Onions, Leeks, and Garli...	TC3218	06/15/1992	40	23	1
9	Silicon Valley Gastronomi...	MC2222	12/12/1993	10	23	-1
10	Sushi, Anyone?	TC7777	06/15/1992	20	23	0

Figure 13-18. Results with KPIs

Adding Abstraction Layers

Microsoft and other vendors have often cautioned people against the direct use of table objects from their database systems. Instead, the preferred model is to use abstraction layers whenever possible. Abstraction layers for Microsoft SQL Server include the use of views, functions, and stored procedures. Let's take a look the two most common ones, views and stored procedures.

Using Views

The most basic abstraction tool is a *view*. It is also the most limited. Views are saved SELECT statements that are stored in Microsoft's SQL Server system tables. These hidden system tables store the text as well as binary valued conversions, though the mechanism of how the SQL code is stored is immaterial to its use. When you create a view, SQL Server stores it within the database, and you can use that SELECT statement again by just referring to it in other SQL code.

Listing 13-22 shows an example of a view being wrapped around our current SELECT statement. Some important things to note are that SQL views cannot contain a number of features we have included in our SQL code so far. For example, you cannot include variables inside a view's definition, so the variable declarations must be excluded. In addition, any use of those variables within the view will not work, and all those lines must be excluded as well. Finally, you cannot use an ORDER BY clause within a view. As such, the ORDER BY clause must also be removed.

Listing 13-22. Creating a View

```
CREATE VIEW vQuantitiesByTitleAndDate
AS

--DECLARE
-- @ShowAll nVarchar(4) = 'True'
--, @StartDate nVarchar(10) = '09/13/1994'
--, @EndDate nVarchar(10) = '09/14/1994'
--, @Prefix nVarchar(3) = 'PS%'
--, @AverageQty int
--SELECT @AverageQty = Avg(SalesQuantity) FROM DWPubSales.dbo.FactSales

SELECT
    DP.PublisherName
    , [Title] = DT.TitleName
    , [TitleId] = DT.TitleId
    , [OrderDate] = CONVERT(varchar(50), [Date], 101)
    , [Total for that Date by Title] = Sum(SalesQuantity)

--, [Average Qty in the FactSales Table] = @AverageQty
--, [KPI on Avg Quantity] = CASE
--    --WHEN Sum(SalesQuantity)
--    -- between (@AverageQty- 5) and (@AverageQty + 5) THEN 0
--    --WHEN Sum(SalesQuantity) < (@AverageQty- 5) THEN -1
--    --WHEN Sum(SalesQuantity) > (@AverageQty + 5) THEN 1
--    --END

FROM DWPubSales.dbo.FactSales AS FS
INNER JOIN DWPubSales.dbo.DimTitles AS DT
    ON FS.TitleKey = DT.TitleKey
```



```

INNER JOIN DWPubSales.dbo.DimDates AS DD
    ON FS.OrderDateKey = DD.DateKey
INNER JOIN DWPubSales.dbo.DimPublishers AS DP
    ON DT.PublisherKey = DP.PublisherKey

--WHERE
-- @ShowAll = 'True'
-- OR
-- [Date] BETWEEN @StartDate AND @EndDate
-- AND
-- [TitleId] LIKE @Prefix

GROUP BY
    DP.PublisherName
    , DT.TitleName
    , DT.TitleId
    , CONVERT(varchar(50), [Date], 101)

--ORDER BY DP.PublisherName, [Title], [OrderDate]

```

■ **Note** With all these restrictions, you might wonder why one would ever use a VIEW statement, but keep in mind that many reporting applications can provide features you cannot include in the view itself once the view has returned all of the unfiltered results to the application.

Once a view has been created, you can use the view by selecting it as if it were a table. The code to do so will look similar to Listing 13-23.

Listing 13-23. Using Your View Without Filters

```

SELECT
    PublisherName
    , [Title]
    , [TitleId]
    , [OrderDate]
    , [Total for that Date by Title]
FROM vQuantitiesByTitleAndDate

```

When the SELECT Statement is run, you receive all the results from the view including the aggregated daily totals. Figure 13-19 shows these results.

Listing 13-23. Using your view without filters

```

SELECT
    PublisherName
    , [Title]
    , [TitleId]
    , [OrderDate]
    , [Total for that Date by Title]
FROM vQuantitiesByTitleAndDate

```

	PublisherName	Title	TitleId	OrderDate	Total for that Date by
1	Algodata Infosystems	But Is It User Friendly?	PC1035	05/22/1993	30
2	Algodata Infosystems	Cooking with Computers: Surreptitious Balance She...	BU1111	03/11/1993	25
3	Algodata Infosystems	Secrets of Silicon Valley	PC8888	05/24/1993	50
4	Algodata Infosystems	Straight Talk About Computers	BU7832	10/28/1993	15
5	Algodata Infosystems	The Busy Executive's Database Guide	BU1032	09/14/1994	15

Query executed successfully. (local)\SQL2012 (11.0 RTM) RSLaptop2\Administrato... DWPubsSales 00:00:00 17 rows

Figure 13-19. Unfiltered Results from the view

The ORDER BY clause can be used when you retrieve results from the view, which can be quite helpful since a view cannot store variables. Listing 13-24 shows an example of filtering the results of your view using the same options we used before the view.

Listing 13-24. Using Your View with the Previous Filters

```

DECLARE
    @ShowAll nVarChar(4) = 'False'
    , @StartDate datetime = '09/13/1994'
    , @EndDate datetime = '09/14/1994'
    , @Prefix nVarChar(3) = 'PS%'
    , @AverageQty int
    SELECT @AverageQty = AVG(SalesQuantity) FROM DWPubsSales.dbo.FactSales

SELECT
    PublisherName
    , [Title]
    , [TitleId]
    , [OrderDate]
    , [Total for that Date by Title]
    , [Average Qty in the FactSales Table] = @AverageQty
    , [KPI on Avg Quantity] = CASE
        WHEN [Total for that Date by Title]
            between (@AverageQty - 5) and (@AverageQty + 5) THEN 0
        WHEN [Total for that Date by Title] < (@AverageQty - 5) THEN -1
        WHEN [Total for that Date by Title] > (@AverageQty + 5) THEN 1
    END

FROM vQuantitiesByTitleAndDate

```

```

WHERE
  @ShowAll = 'True'
OR
  [OrderDate] BETWEEN @StartDate AND @EndDate
AND
  [TitleId] LIKE @Prefix
ORDER BY PublisherName, [Title], [OrderDate]

```

Now we get the same results as before, but client reporting applications that expect to only use views will have them available as needed.

One change that should be noted is that since the view contains an alias column, our SQL code must be modified to include the new column names. You can see an example of this in the modified version of the WHERE clause shown in Listing 13-24. Whereas before you were using the [Date] column, you must now use the [OrderDate] column as defined by the view. The same is true in places where we reuse the Sum(SalesQuantity) in our calculations inside of the CASE operator; we must now use the [Total for that Date by Title] column of the view.

When this SELECT statement is run, all the results from the view will still be gathered in memory, but the final results will include only filtered data. Figure 13-20 shows what this looks like.

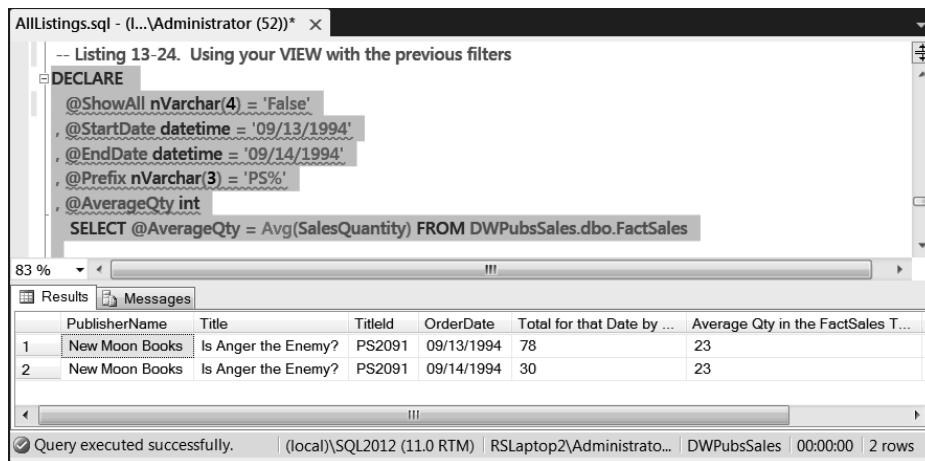


Figure 13-20. Filtered results from the view with the KPIs added and results ordered

Note that the result set looks remarkably similar to the one before you started using the view (see Figure 13-18). This is because we are getting the same results. The only difference is that now we are using a view.

Using Stored Procedures

Perhaps the most flexible tool you can use for abstracting your report data is SQL stored procedures. Stored procedures can include almost all SELECT clauses, built-in functions, operators, and even multiple SELECT statements. Because of this flexibility, chances are very high that you will want to create stored procedures for your reporting applications.

Listing 13-25 shows an example of a stored procedure that uses the code we have built so far.

Listing 13-25. Creating a Stored Procedure

```
-- Using Stored Procedures
CREATE PROCEDURE pSelQuantitiesByTitleAndDate
(
    -- 1) Define the parameter list:
    -- Parameter Name, Data Type, Default Value --
    @ShowAll nVarchar(4) = 'True' -- 'True|False'
    , @StartDate datetime = '01/01/1990' -- 'Any valid date'
    , @EndDate datetime = '01/01/2100' -- 'Any valid date'
    , @Prefix nVarchar(3) = '%' -- 'Any three wildcard search characters'
    --, @AverageQty int
    --SELECT @AverageQty = Avg(SalesQuantity) FROM DWPubSales.dbo.FactSales
)
AS

BEGIN -- the body of the stored procedure --

    -- 2) Set the @AverageQty variable here since you cannot use subqueries in the
    -- a stored procedures parameter list.
    DECLARE @AverageQty int
    SELECT @AverageQty = Avg(SalesQuantity) FROM DWPubSales.dbo.FactSales

    --3) Get the Report Data

    SELECT
    DP.PublisherName
    , [Title] = DT.TitleName
    , [TitleId] = DT.TitleId
    , [OrderDate] = CONVERT(varchar(50), [Date], 101)
    , [Total for that Date by Title] = SUM(SalesQuantity)
    , [Average Qty in the FactSales Table] = @AverageQty
    , [KPI on Avg Quantity] = CASE
        WHEN Sum(SalesQuantity)
            BETWEEN (@AverageQty- 5) AND (@AverageQty + 5) THEN 0
        WHEN Sum(SalesQuantity) < (@AverageQty- 5) THEN -1
        WHEN Sum(SalesQuantity) > (@AverageQty + 5) THEN 1
    END
    FROM DWPubSales.dbo.FactSales AS FS
    INNER JOIN DWPubSales.dbo.DimTitles AS DT
        ON FS.TitleKey = DT.TitleKey
    INNER JOIN DWPubSales.dbo.DimDates AS DD
        ON FS.OrderDateKey = DD.DateKey
    INNER JOIN DWPubSales.dbo.DimPublishers AS DP
        ON DT.PublisherKey = DP.PublisherKey
    WHERE
        @ShowAll = 'True'
    OR
        [Date] BETWEEN @StartDate AND @EndDate
```

```

AND
  [TitleId] LIKE @Prefix+ '%'
GROUP BY
  DP.PublisherName
  , DT.TitleName
  , DT.TitleId
  , CONVERT(varchar(50), [Date], 101)
ORDER BY DP.PublisherName, [Title], [OrderDate]

```

END -- the body of the stored procedure --

Key points of interest are the definition of the parameter list at the beginning of a stored procedure and before the AS keyword. This serves the same purpose as declaring variables to be used to create dynamic queries.

One interesting aspect of stored procedure parameters is your ability to define default values. When a stored procedure is executed and a value is not supplied for the parameter, the default will be used. If a value is given, however, then the stored procedure will use the given value.

Note It is common for developers to include additional comments in their stored procedures. Listing 13-25 indicates the kind of data that is expected within the parameters. Other additions that are not shown here are: a header directly after the AS keyword that indicates information about who created the stored procedure, when it was first created, which objects it interacts with, and a list of changes that have occurred over time. This header information, while important, is not shown here in order to keep the code as small as possible. But we show an example of what it should look like in just a moment.

After the parameters have been defined for the stored procedure, the AS keyword is used to identify the beginning of the stored procedure body. The body of the stored procedure contains your reporting queries and any additional queries, and might be required to obtain the report data. Although not required, BEGIN and END are used to identify where the body of the stored procedure starts and finishes, and it is a recommended practice to include both.

Once the stored procedure has been created, you can execute it by using the EXECUTE keyword, which can be written out or, as shown in Listing 13-26, can be executed using just the first four letters of the keyword, EXEC.

Listing 13-26. Using Your Stored Procedure with Default Values

```
EXEC pSelQuantitiesByTitleAndDate
```

When this code runs, the stored procedure utilizes all the default values since no additional parameter values were given. In this case, the default value of True for the @ShowAll parameter is sufficient for the query to return all the results unfiltered, as shown in Figure 13-21.

-- Listing 13-26. Using your stored procedure
EXEC pSelQuantitiesByTitleAndDate

	PublisherName	Title	TitleId	OrderDate	Total for that Date by ...
1	Algodata Infosystems	But Is It User Friendly?	PC1035	05/22/1993	30
2	Algodata Infosystems	Cooking with Computers: Surreptitious Balance She...	BU1111	03/11/1993	25
3	Algodata Infosystems	Secrets of Silicon Valley	PC8888	05/24/1993	50
4	Algodata Infosystems	Straight Talk About Computers	BU7832	10/28/1993	15
5	Algodata Infosystems	The Busy Executive's Database Guide	BU1032	09/14/1994	15
6	Binnet & Hardley	Computer Phobic AND Non-Phobic Individuals: Beh...	PS1372	05/29/1993	20
7	Binnet & Hardley	Fifty Years in Buckingham Palace Kitchens	TC4203	06/15/1992	20
8	Binnet & Hardley	Onions, Leeks, and Garlic: Cooking Secrets of the ...	TC3218	06/15/1992	40
9	Binnet & Hardley	Silicon Valley Gastronomic Treats	MC2222	12/12/1993	10
10	Binnet & Hardley	Sushi, Anyone?	TC7777	06/15/1992	20
11	Binnet & Hardley	The Gourmet Microwave	MC3021	09/14/1994	40

Query executed successfully. (local)\SQL2012 (11.0 RTM) | RSLaptop2\Administrato... | DWPubsSales | 00:00:00 | 17 rows

Figure 13-21. Results for the stored procedure using the default values

Of course, you can also supply values for each parameter, as shown in Listing 13-27.

Listing 13-27. Using Your Stored Procedure to Set All Values

EXEC pSelQuantitiesByTitleAndDate

```

@ShowAll = 'False'
, @StartDate = '09/13/1994'
, @EndDate = '09/14/1994'
, @Prefix = 'PS'

```

The results of this query display only two rows because of the filters placed on the query (Figure 13-22).

-- Listing 13-27. Using your stored procedure to set all values
EXEC pSelQuantitiesByTitleAndDate
@ShowAll = 'False'
, @StartDate = '09/13/1994'
, @EndDate = '09/14/1994'
, @Prefix = 'PS'

	PublisherName	Title	TitleId	OrderDate	Total for that Date by ...	Average Qty in the FactSales T...	KP
1	New Moon Books	Is Anger the Enemy?	PS2091	09/13/1994	78	23	1
2	New Moon Books	Is Anger the Enemy?	PS2091	09/14/1994	30	23	1

Query executed successfully. (local)\SQL2012 (11.0 RTM) | RSLaptop2\Administrato... | DWPubsSales | 00:00:00 | 2 rows

Figure 13-22. Results for the stored procedure using provided parameter values

Using default values for your parameters can be convenient because when executing the stored procedure, not all parameter values need to be set to be able to use any one of them. In Listing 13-28 we are setting only two of the four possible parameter values.

Listing 13-28. Accepting Defaults for Some Parameters

```
EXEC pSelQuantitiesByTitleAndDate
```

```
    @ShowAll = 'False'
, @Prefix = 'PS'
```

When this query runs, the results include anything that has the *PS*% prefix regardless of the date the order was placed. Figure 13-23 shows the results.

	PublisherName	Title	TitleId	OrderDate	Total for that Date by ...	Average
1	Binnet & Hardley	Computer Phobic AND Non-Phobic Individuals: Beh...	PS1372	05/29/1993	20	23
2	New Moon Books	Emotional Security: A New Algorithm	PS7777	05/29/1993	25	23
3	New Moon Books	Is Anger the Enemy?	PS2091	09/13/1994	78	23
4	New Moon Books	Is Anger the Enemy?	PS2091	09/14/1994	30	23
5	New Moon Books	Life Without Fear	PS2106	05/29/1993	25	23
6	New Moon Books	Prolonged Data Deprivation: Four Case Studies	PS3333	05/29/1993	15	23

Query executed successfully. (local)\SQL2012 (11.0 RTM) | RSLaptop2\Administrato... | DWPubsSales | 00:00:00 | 6 rows

Figure 13-23. Results for the stored procedure setting some values

After your stored procedures are created and tested, you may want to make some changes. For example; production stored procedures commonly include a header at the beginning of the stored procedure, much like the one that was at the beginning of our script file. Listing 13-29 places a header just before the beginning of the stored procedure body.

Listing 13-29. Adding a Header to Your Stored Procedure

```
ALTER PROCEDURE pSelQuantitiesByTitleAndDate
(
    -- 1) Define the parameter list:
    -- Parameter Name, Data Type, Default Value --
    @ShowAll nVarchar(4) = 'True' -- 'True|False'
, @StartDate datetime = '01/01/1990' -- 'Any valid date'
, @EndDate datetime = '01/01/2100' -- 'Any valid date'
, @Prefix nVarchar(3) = '%' -- 'Any three wildcard search characters'
)
AS

/*****
Title:pSelQuantitiesByTitleAndDate
Description: Which titles were sold on which dates
Developer:RRoot
*****/
```

Date: 9/1/2011**Change Log: Who, When, What****CMason,9/2/2011,fixed even more grammatical errors**

```

*****/

```

```

BEGIN -- the body of the stored procedure --
...

```

You can alter the stored procedure code to include the header using the ALTER keyword. When this code is run, the new version of the stored procedure is saved in the database. Whenever the database is backed up, the stored procedure will be backed up as well, making it easier to maintain your report queries.

EXERCISE 13-2. CREATING STORED PROCEDURES

In this exercise, you create your own custom report stored procedure similar to the one discussed in this chapter. This time, you include parameters that filter the results based on a start date and end date. Try using building the query a little at a time, as described in this chapter. You may find that creating a report procedure is easier than you think.

1. Create a stored procedure that allows report users to filter the results based on a range of dates using the code generated in Exercise 13-1.

Execute the stored procedure to verify that it works. (Your results should look like Figure 13-24.)

If you get stuck, the answers for this exercise can be found in the example files for this book. Look in the folder for Chapter 13.

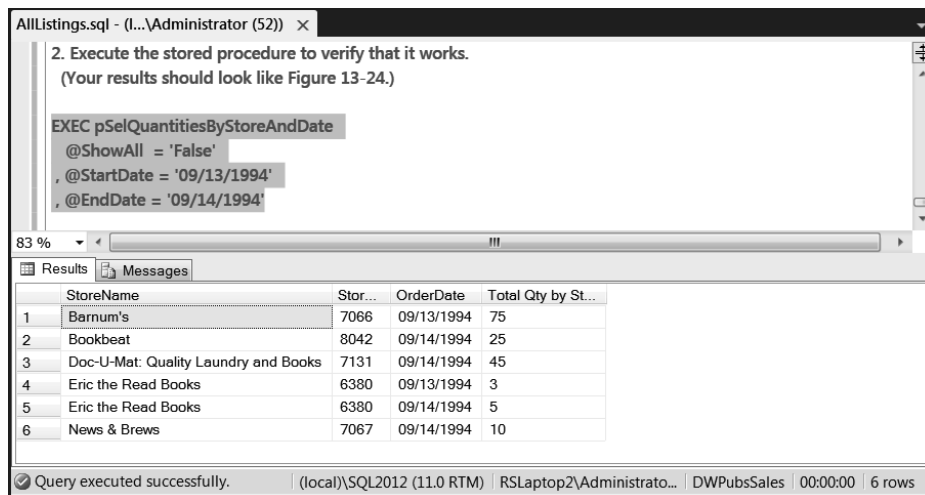


Figure 13-24. Results showing sales by store

In this exercise, you wrote a report query that captured sales data based on sales by stores.

Using Your Code in Reporting Applications

Once you have created your code, views, or stored procedures, you can use them in a number of reporting applications. For example, Microsoft Excel allows you to easily connect to a database and access both views and tables, as shown in Figure 13-25.

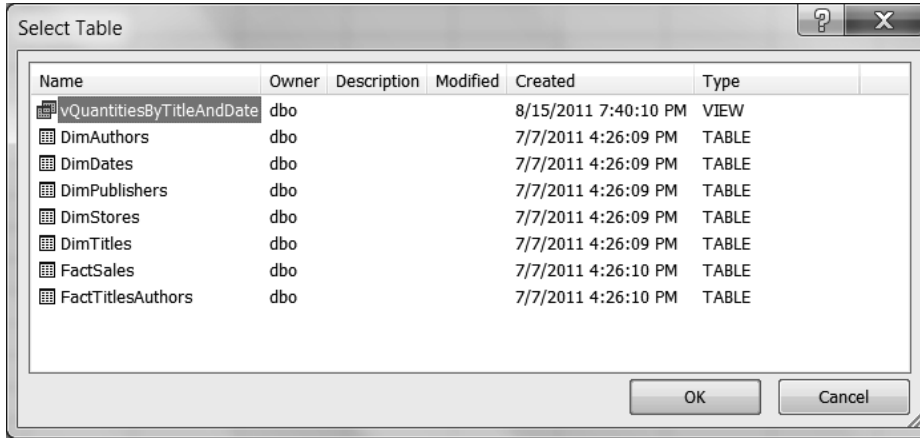


Figure 13-25. Connecting to a view from Microsoft Excel

In Excel you can display the information in either a tabular or pivot table format. The pivot table format provides options such as sorting, filtering, and subtotals, and it easily generates charts, making it a popular choice. This is significant, because as we have seen, many of these features cannot be stored directly as part of a SQL view's code. Figure 13-26 shows an example of a pivot table using the view we created.

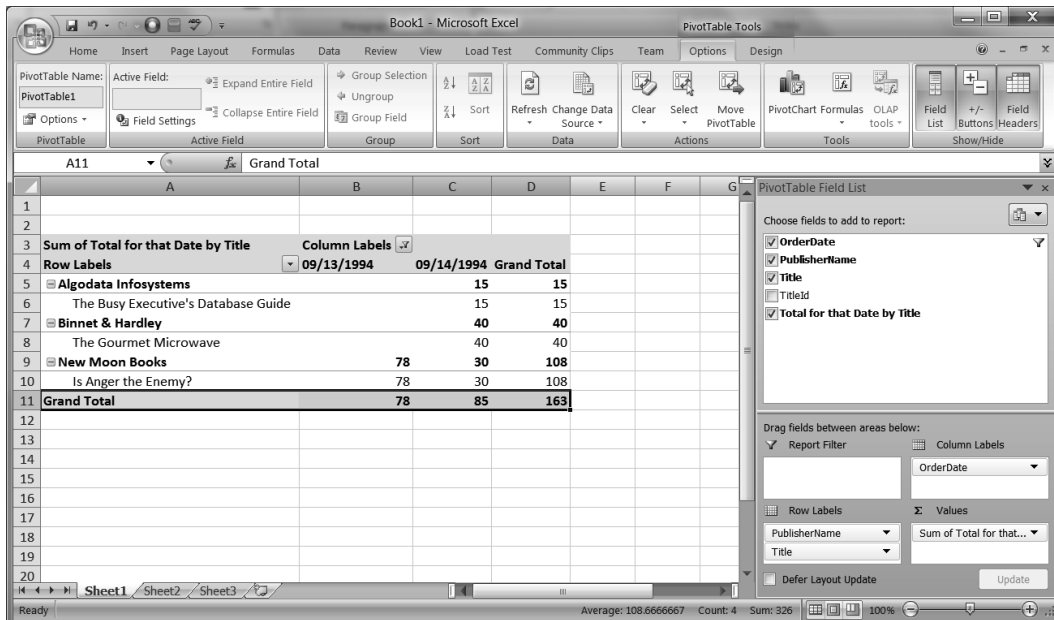


Figure 13-26. Displaying the report with an Excel pivot table

Excel can use SQL stored procedures as well, but it's not as straightforward as it could be. Other applications such as Microsoft Reporting Server (SSRS), however, have no such problems. In SSRS you can effortlessly launch a wizard and include the stored procedure name you want to use to capture your report data. Figure 13-27 shows the SSRS Report Wizard on its final dialog window, which list the details about the wizard configuration. Note that the stored procedure name was used as a query.

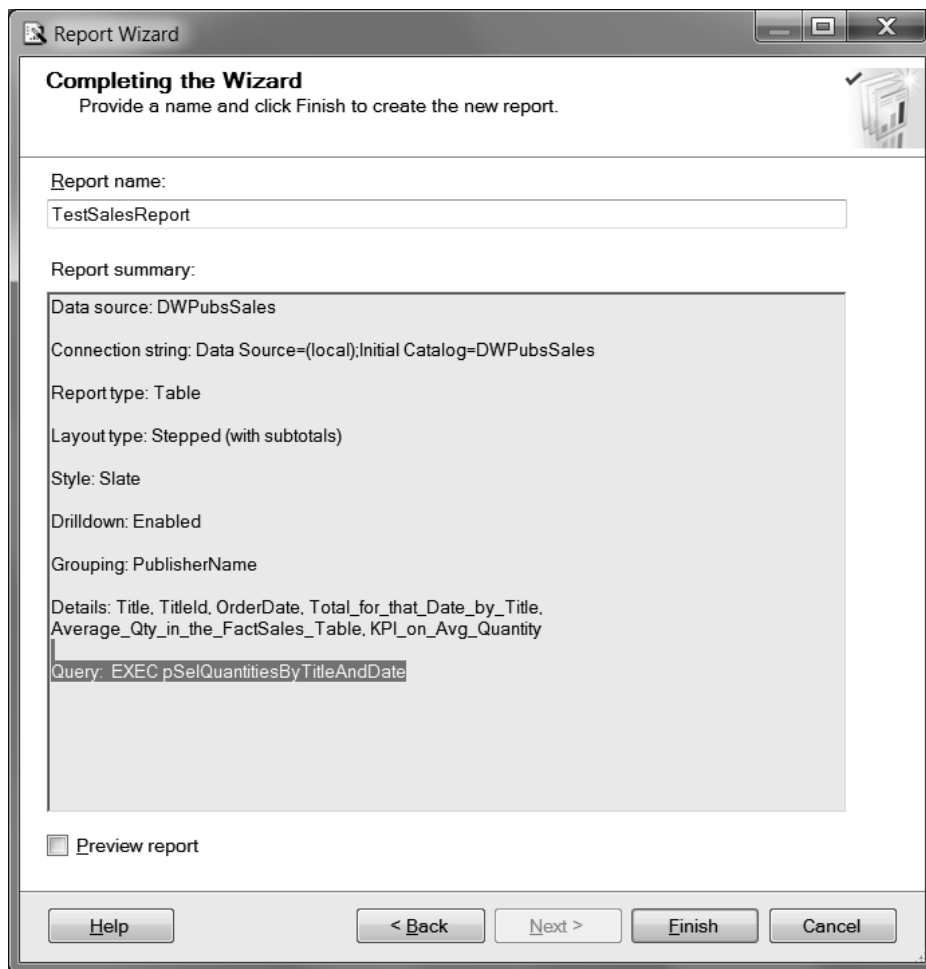
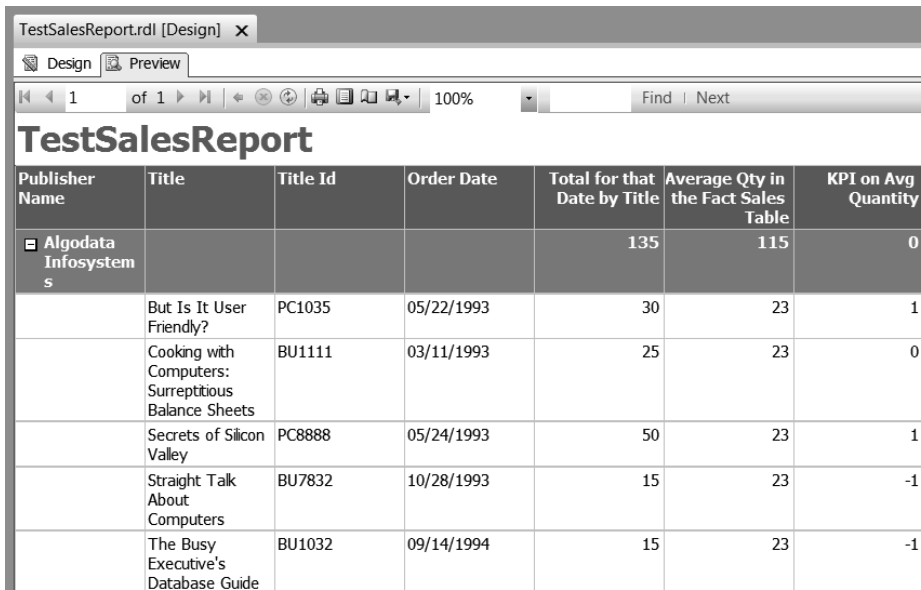


Figure 13-27. Getting report data from the stored procedure with Reporting Services

When the wizard completes, your SSRS project will include a new report that contains data generated from your stored procedure (Figure 13-28).



Publisher Name	Title	Title Id	Order Date	Total for that Date by Title	Average Qty in the Fact Sales Table	KPI on Avg Quantity
Algodata Infosystems				135	115	0
	But Is It User Friendly?	PC1035	05/22/1993	30	23	1
	Cooking with Computers: Surreptitious Balance Sheets	BU1111	03/11/1993	25	23	0
	Secrets of Silicon Valley	PC8888	05/24/1993	50	23	1
	Straight Talk About Computers	BU7832	10/28/1993	15	23	-1
	The Busy Executive's Database Guide	BU1032	09/14/1994	15	23	-1

Figure 13-28. Displaying the results of the stored procedure in Reporting Services

We will delve deeper into both of these scenarios in the chapters on Excel reporting and reporting services.

Moving On

You will need quite a bit of practice to become an expert at writing report queries. Report stored procedures often included hundreds of lines of code. These stored procedures take many days to write and test correctly. When stored procedures become this large, it is very easy to make mistakes that can cost a lot of time and money. The methodology of incrementally building and reviewing your results, using consistent formatting and abstraction layers, make reporting queries easier to maintain and cost effective.

LEARN BY DOING

In this “Learn by Doing” exercise, you create several SQL queries using the Northwind database. We included an outline of the steps to perform and an example of how the authors handled them in two Word documents. These documents are found in the folder

C:_BISolutionsBookFiles_LearnByDoing\Chapter13Files. Please see the ReadMe.doc file for detailed instructions.

What's Next?

There is a lot to understand to become good at writing report queries. This includes proper syntax, performance considerations, and checking validity. While experience is your best teacher, we also recommend this book, that may help you along the way.

SQL Server 2008 Transact-SQL Recipes

A Problem-Solution Approach

By Joseph Sack

ISBN13: 978-1-59059-980-8