Managing SQL in Visual Studio .NET
By Paul Kimmel October 16, 2003

SQL is an important part of modern programming. Whether you are building a Web application, a client-server Windows application, or an enterprise solution encompassing Windows, the Web, and distributed components such as XML Web Services or .NET Remoting, you are probably using a database.

Microsoft has done a great job incorporating tools into Visual Studio .NET, but due to the finite amount of screen real estate, some really great features are tucked away and easy to overlook. This article points out one of these great features, the SQL designing tools, built right into Visual Studio .NET. After you read this article, you will know how to visually design relationships, filter SQL statements, edit SQL code, and run your queries right in the Visual Studio .NET IDE.

Connections in the Server Explorer

The Server Explorer—accessible from View|Server Explorer—is a focus point for several tools tucked away into Visual Studio .NET. The Data Connections node contains OLEDB database connections, and the Servers node (see Figure 1) contains a SQL Servers child node. If you have a network connection or a local copy of SQL Server, such as MSDE, those database servers will show up here.

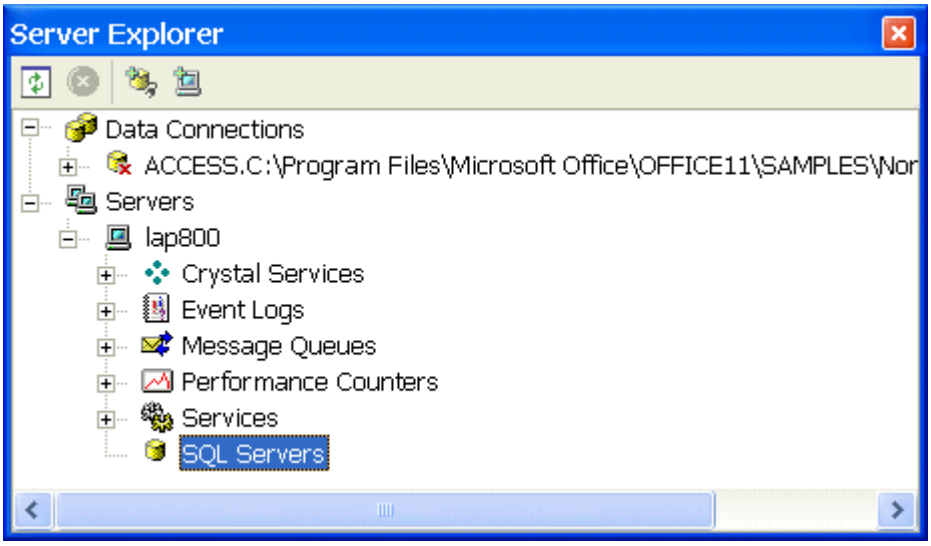


Figure 1: The Data Connections and SQL Servers nodes in the Server Explorer.

If you right-click on the Data Connections node, you can select Add Connection or Create SQL Server Database to connect to an existing database or create a new database. If you right-click on the SQL Servers node, you can register a SQL Server instance in Visual Studio .NET. Select any of these databases, press F4, and the Properties window will provide you with detailed information about the particular database. Of particular use is the ConnectString property that will provide you with convenient access to what can oft be a cryptic string of connection information. For example, if you are working with ADO.NET and need a connection string to initialize a connection object, you can copy a correct connection string from the Properties window.

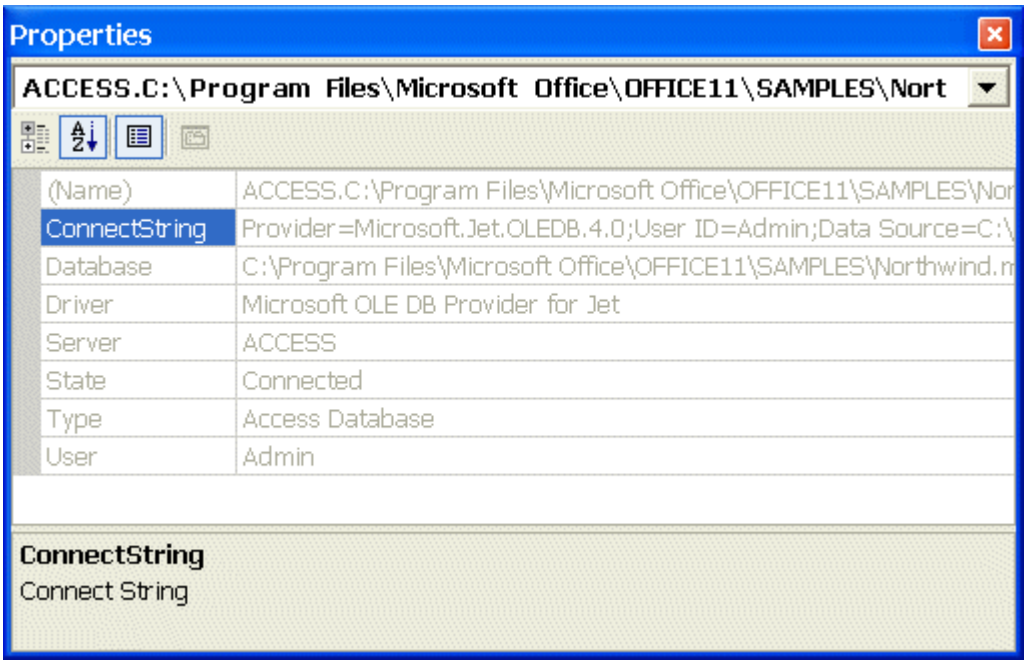


Figure 2: The Properties for a database instance in the Server Explorer.

Browsing Tables, Views, and Stored Procedures

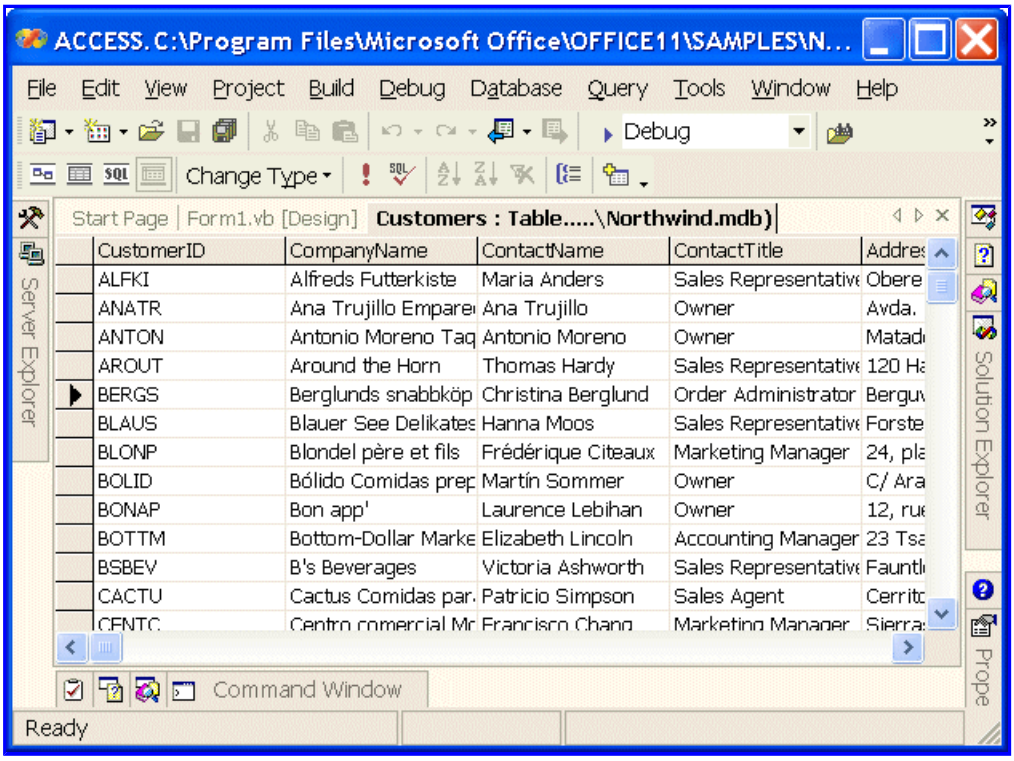
If individual Data Connections or SQL Server instances in the Server Explorer are expanded, the context menus for those items and those sub-nodes permit you to create and explore tables, views, and stored procedures. Thus, you can still use the SQL Enterprise Manager or MS-Access (or whatever database tool you generally use) to manage data, but you don't have to leave Visual Studio .NET for most of tasks.

Occasional exceptions to the general utility of Visual Studio .NET for managing databases can be managed by falling back on the tools shipped by the database provider. For instance, you can write and test stored procedures for SQL Server in Visual Studio .NET, but you will need to use the stored procedure builder that ships with UDB to implement stored procedures for a UDB database. (Although you can run stored procedures in Visual Studio .NET, current implementations don't permit editing stored procedures supported by non-SQL Server providers.)

Viewing the Data

We will use the Northwind sample database for our examples because of its ubiquity. If you don't have a sample database listed in the Data Connections node, use the Add Connection feature to add a database connection.

The easiest way to view data is to expand the database and its Tables node. Double-click on the table whose data you'd like to view and a datagrid view corresponding to a SELECT * FROM *tablename* is displayed (see Figure 3). In addition to showing the data, the Query toolbar is displayed—represented by the twelve buttons on the bottom of the toolbar (the third one from the left contains the acronym SQL).



[Click here for a larger image.]

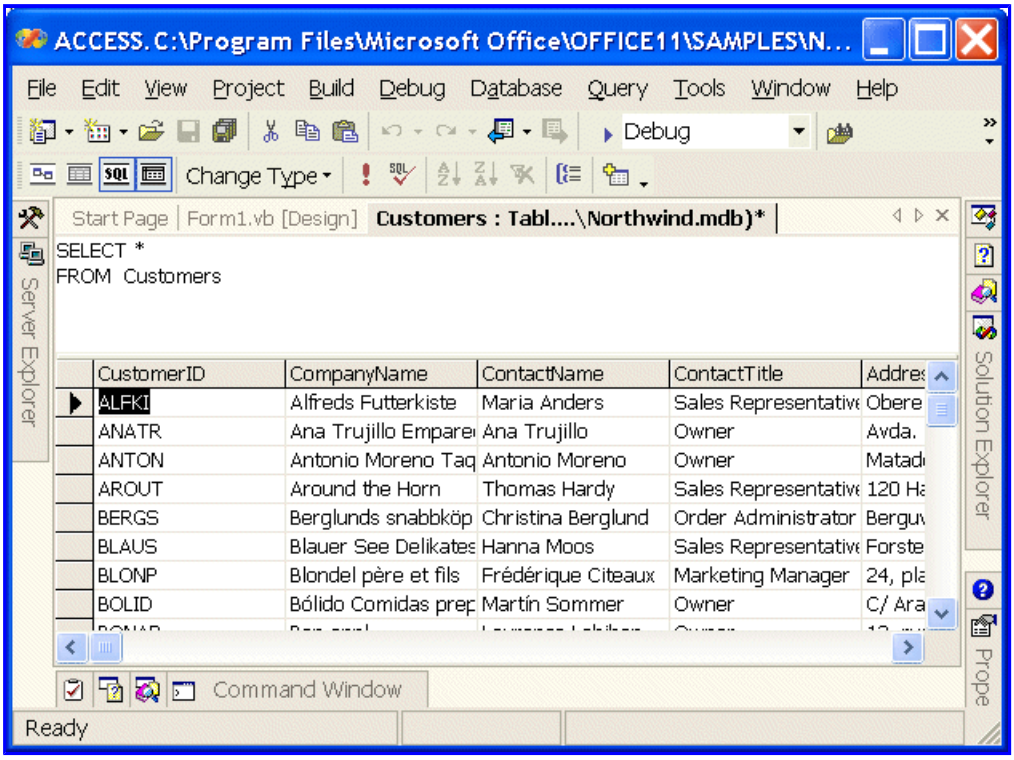
Figure 3: The center shows the Results Pane controlled by the fourth Query button from the left.

The context menu for the Results Pane contains options for navigating through the results shown. In addition to the context menu, the Query menu contains additional features for managing the data in the Results Pane and performing other database tasks. For example, to create a new table from the existing table in the Results Pane, select Query|Change Type|Make Table and the Results Pane's query will be modified to include an INTO clause that will cause results from the current view to be copied into a new table named by you. In essence, you can create a copy of the current table.

To view the default query or the modified make table query, click the SQL button from the Query toolbar to display the SQL Pane.

Writing SQL

The combined SQL Pane and Results Pane containing the default SELECT statement and the results of that SELECT statement are shown in Figure 4. The SQL Pane is just an editor. You can write any valid SQL in this editor, and save and load this SQL at any time to and from an external file. In addition, Visual Studio .NET is a great first-line tool for managing data during development and unit testing.



[Click here for a larger image.]

Figure 4: The SQL Pane shown top and the Results Pane shown bottom, demonstrating the coordination between SQL code and related results.

When you have written code, you can change the type of the query from the Change Type toolbar dropdown list. You can run the SQL from the Run Query toolbar button (red exclamation mark), verify the syntax of the current query (with the SQL button with a checkmark), and a GROUP BY clause (second button from the far right) and add additional tables to the query. (You'll appreciate the Add Table because it will write syntactically correct join statements.)

For example, we can write

```
SELECT * FROM CUSTOMERS C, ORDERS O
         WHERE C.CustomerID = O.CustomerID
```

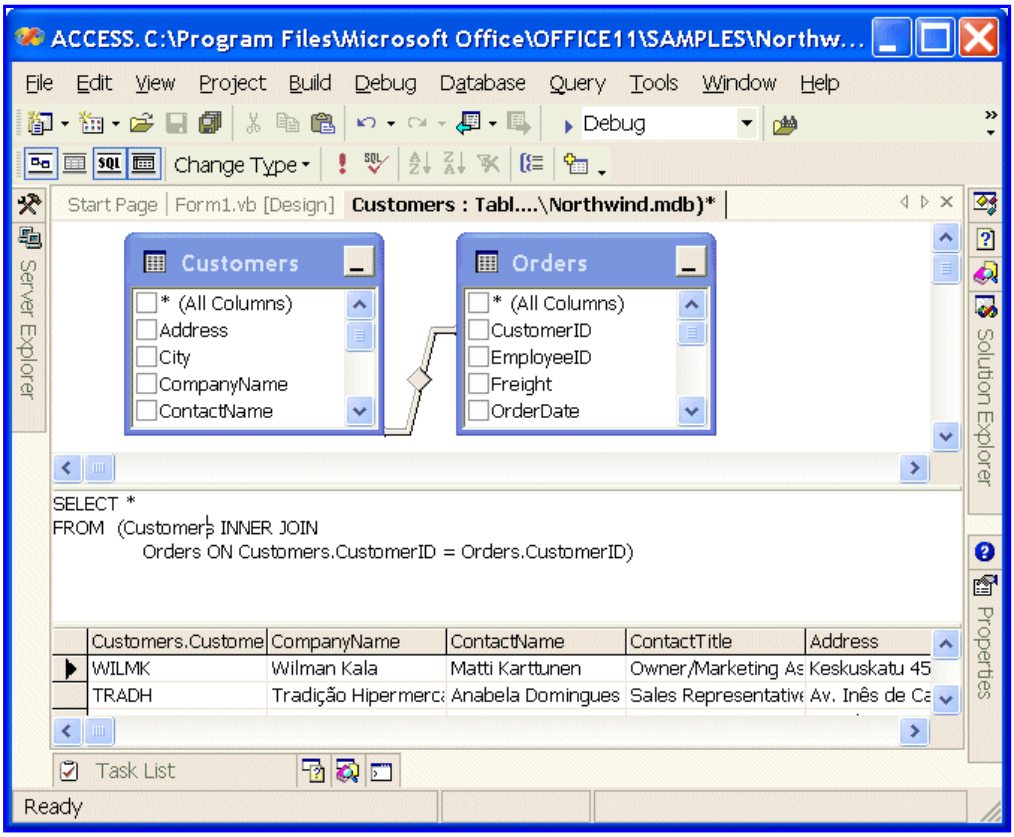
But this WHERE predicate will leave out all customers that currently have no orders and orders that have been orphaned because customer records have been deleted. A JOIN predicate is capable of returning customers with or without orders, only customers with orders, or orders with customers and orphaned orders (orders with no customers). Unfortunately, the syntax for JOIN predicates is more convoluted and consequently harder to remember than WHERE predicates, but the SQL Pane can be a useful reminder for VB.NET programmers who don't write SQL every day. Here is the equivalent JOIN, called an INNER JOIN, that is equivalent to the preceding SQL statement, but was generated by the IDE.

```
SELECT *
FROM   (Customers INNER JOIN
        Orders ON Customers.CustomerID = Orders.CustomerID)
```

If one is an infrequent author of SQL or a new programmer learning SQL, Visual Studio .NET can be an aid in the learning process.

Describing Relations in the Diagram Pane

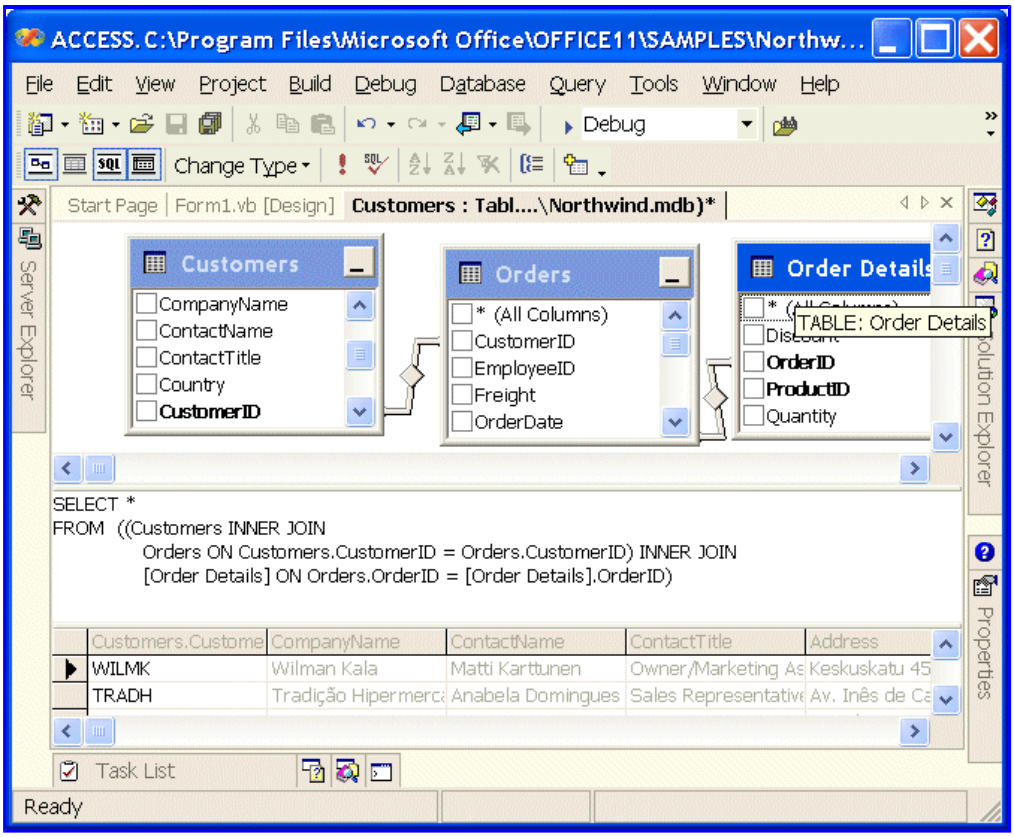
If you have used Microsoft Access or a database modeling tool such as DataArchitect or ERwin, you are familiar with the notion of visual modeling tools. The Diagram Pane is such a tool. If we click the Show Diagram Pane toolbar button on the Query toolbar, we can see the combined Diagram, SQL, and Results Panes together in Visual Studio .NET. Each of these windows is a facet of the same thing, a query showing all customers and their orders, each represented in a different way.



Click here for a larger image.

Figure 5: The Diagram, SQL, and Results Panes representing the Northwind Customers and their Orders.

We can specify new relations by dragging columns to join on from one table to another. To delete existing relationships, click on the graphic line between the two tables and press the Delete key. Alternatively, deleting the JOIN predicate from the SQL Pane and the Diagram Pane will be updated to reflect the new SQL statement. If we want to add more detail to the query, we can drag additional tables from the Server Explorer and specify additional JOIN statements. For instance, to include the Order Details to our query, drag the Order Details table from the Server Explorer and drop the table in the Diagram Pane. If the new table contains a field matching a field in tables already in the Diagram Pane, the new relationship is already mapped in the Diagram Pane and the SQL Pane is updated accordingly (see Figure 6).

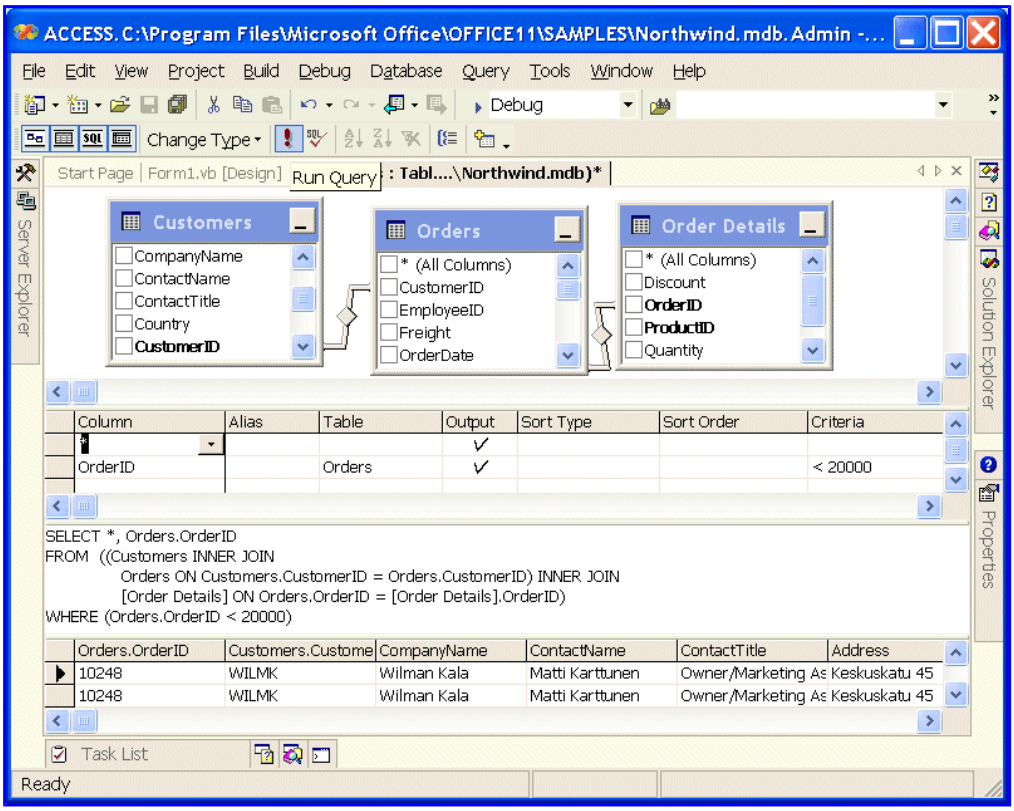


[Click here for a larger image.]

Figure 6: The Order Details table is added to the Diagram Pane, the relationship between Orders and Order Details is automatically mapped, and the SQL Pane is updated to include the additional JOIN predicate.

Using the Grid Pane to Fine-Tune Queries

The final pane we'll talk about is the Grid Pane. The Grid Pane represents a columnar list of the criteria of the current query. If we show the Grid Pane for our query, we'll see (see Figure 7) that the query is comprised of columns, those columns' aliases if applicable, tables, sort type, and sort criteria (filter information). Figure 7 shows the first set of criteria, which is the SELECT * criteria. The second row was added manually by me to filter orders that have an OrderID < 20000.



[Click here for a larger image.]

Figure 7: Use the Grid Pane to fine tune the WHERE predicates for your filters.

Summary

Visual Studio .NET offers a lot of tools in a centralized location for managing most aspects of software development from the IDE. In this article, you learned a bit about managing connections from the Server Explorer and using the four visual metaphors for managing SQL: the Grid Pane, SQL Pane, Diagram Pane, and Results Pane. With these tools, you can generate perfect SQL even if you aren't a SQL programmer, and you never have to leave Visual Studio .NET for most of your day-to-day SQL tasks.

About the Author

Paul Kimmel has been using B.A.S.I.C. and its progeny for 25 years and has written several books on Visual Basic, including the recently released *Visual Basic .NET Power Coding* from Addison Wesley. Pick up your copy at http://www.amazon.com/. Paul is also available for short- and long-term consulting and .NET training. You may contact him at pkimmel@softconcepts.com.

# # #