

Part I

Transact-SQL Basics

CHAPTER 1

Transact-SQL Overview

THE MAIN GOAL OF THIS BOOK is to show how stored procedures and triggers are used to implement real-world solutions with SQL Server 2000. In order to accomplish this goal the reader must have a fundamental understanding of Transact-SQL (T-SQL). With this in mind, the book has been organized so that the material presented in Part One (Chapter 1 to 8) can serve as a reference for both Parts Two and Three. Part One not only covers the basics of T-SQL, but also contains a description of every programming concept (e.g., T-SQL built-in functions) used in the remainder of the book. If you see a code segment or reference that you do not understand in Parts Two and/or Three, be assured that it is explained in Part One.

This chapter begins a series of five whose purpose is to introduce you to the basic concepts of T-SQL. A general overview of the language is covered here, data types in Chapter 2, data definition language (DDL) in Chapter 3, and data manipulation language (DML) in Chapters 4 and 5. If you are a novice user of SQL Server, I encourage you to read these chapters thoroughly before taking on the rest of the book.

If you already have a good understanding of T-SQL you will probably want to jump ahead to Chapter 7 and read about user-defined functions, which are new to SQL Server 2000. If you are a frequent user of views, you will want to make sure you read Chapter 8 so you can understand the new indexing features that are now available. Otherwise, feel free to jump to Part Two and start reading about procedures.

NOTE Sample Code

The sample code for the book can be downloaded at either <http://www.apress.com> or <http://www.SQLBook.com>. Download CodeCentric.zip and extract and access a chapter's sample file(s) as needed. For most chapters there is only a single .sql file, but for Chapters 10, 13, and 14, multiple files are used.

Before You Get Started

This book uses a lot of examples to demonstrate the various functionality available in SQL Server 2000 and each one is included in the example file(s) that accompany each chapter. The example files were created so that you could more easily implement the examples as you read along. Query Analyzer is used extensively to execute the examples, so the various components and functionality available in the tool are discussed here.

Query Analyzer

Query Analyzer is an application that allows you to interact with the SQL Server database engine. It is not really a *part* of SQL Server, but simply a tool that comes with the product that makes it easier to use. Query Analyzer has been greatly enhanced in SQL Server 2000, so you can now use it to accomplish the following:

- Create and execute queries.
- Use templates that contain the basic syntax of commonly used database objects.
- Analyze and script database objects using Object Browser.
- Execute stored procedures using Object Browser.
- Perform query optimization analysis using the Show Execution Plan option.
- Locate database objects using Object Search.
- Debug stored procedures using the T-SQL Debugger.

Query Analyzer Components

When you launch Query Analyzer you are prompted to provide login information. Once you have provided a valid login and connected to the target server, the windows shown in Figure 1-1 are displayed.

Query Window is shown on the right and Object Browser is shown on the left.

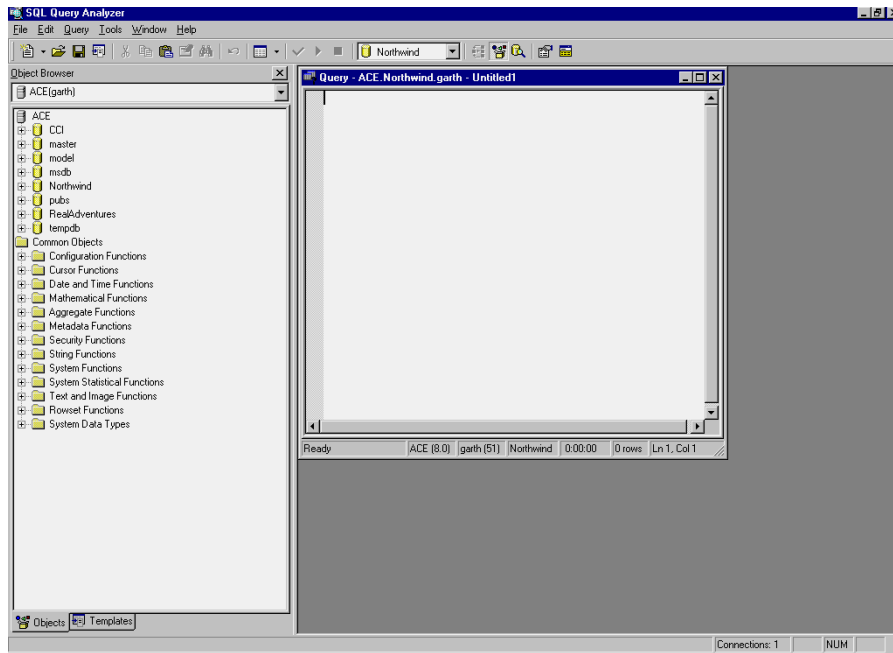


Figure 1-1. Query Analyzer

Query Window

The Query Window is where you will do most of your SQL programming so it is important that you understand the multiple panes of which it is composed.

Table 1-1 describes the panes used in Query Window.

Table 1-1. Query Window Panes

PANE	DESCRIPTION
Editor	Where you type and execute SQL statements. Statements can be executed by clicking the Run button on the toolbar or by pressing CTRL+E. When the pane contains multiple statements, one or more can be executed in isolation by highlighting each and pressing CTRL+E.
Results	Where the results of the statements are displayed. By default, this pane is not displayed until a statement is executed. You can toggle the display of this pane by clicking the right-most icon on the toolbar.
Message	Where error messages are displayed. This pane simply overlays the Results Pane when an executed SQL Statement causes an error message to be displayed.

Table 1-1. Query Window Panes (Continued)

PANE	DESCRIPTION
Execution Plan	Where the graphical representation of the execution plan is displayed. This pane is not displayed by default, but can be activated by clicking Query on the main menu, selecting Show Execution Plan and then executing a SQL Statement. Once the statement has been executed a tab labeled Execution Plan will be displayed next to the Results tab. Simply click it to see a graphical representation of the execution plan.
Trace	Where server trace information is displayed. It is not displayed by default, but can be activated by clicking Query on the main menu, selecting Show Server Trace and then executing a SQL Statement. Once the statement has been processed a tab labeled Trace will be displayed to the right of the Results tab.
Statistics	Where statistics about the statements processed and the connection session are displayed. It is not displayed by default, but can be activated by clicking Query on the main menu, selecting Show Client Statistics and then executing a SQL Statement. Once the statement has been processed a tab labeled Statistics will be displayed to the right of the Results tab.

The last three panes described in Table 1-1 are optional for good reason. They display more advanced information that is used to optimize queries and/or troubleshoot performance problems. The Execution Plan Pane is used extensively in Appendix C, which discusses query optimization techniques. You should, however, postpone reading it until you complete Part Two of the book. The Trace and Statistics Panes are not used in this book, so if you would like to learn more about each one, see Books Online topics: Viewing and Analyzing Traces and Query Window Statistics Pane.

TIP Looking Up References in Books Online

This book contains numerous references to topics in Books Online. To find a particular topic, simply open Books Online via the SQL Server program group, click the Search tab, type in the topic verbatim in the input box and click List Topics.

Object Browser

Object Browser is new to SQL Server 2000, and it is an extremely useful tool. In pre-2000 versions of SQL Server I often needed both Query Analyzer and Enterprise Manager open at the same time in order to program. Object Browser includes the functionality (for example, easy access to a table's column names and data types) that formerly caused me to use Enterprise Manager, so I no longer need to have both applications open while developing.

Object Browser allows you to do the following:

- View all the objects in an instance of SQL Server.
- Create a script that shows the statements used to create an object.
- Create a script that allows you to modify or alter an object.
- View the data in a table or view.
- Execute a stored procedure.
- Add an extended property to an object. An extended property is used to add metadata to a database object.
- Delete a database object.
- View many of the built-in functions available in SQL Server 2000.
- View the available data types and their associated characteristics.

In order to use Object Browser to view all the objects in a database, simply expand the target database and expand the folder of the type of object you wish to view. You can place an object's name in the Editor Pane by clicking it and dragging it from the browser window into the pane. The various scripting options that are available for an object are accessed by right-clicking the target object and selecting the desired scripting option from the pop-up menu. To view the data in a table or view, simply right-click the object and select Open. In like manner, executing a stored procedure is as simple as right-clicking the target procedure and selecting Open. An extended property can be added to an object by right-clicking the object and selecting Extended Properties from the pop-up menu. To delete an object simply right-click it and select Delete. To get the syntax for a function shown in the Common Objects area into the Editor Pane, right-click the target and select the desired scripting option. To see the data types available in SQL Server 2000, simply expand the System Data Types folder. A description of the data type is displayed when you place your mouse over each item.

Object Search

Object Search, new to SQL Server 2000, allows you to search for an object in one or more databases. You activate Object Search by clicking the database/magnifying glass icon on the toolbar or by clicking Tools on the main menu and selecting Object Search. The window displayed as a result of either action is shown in Figure 1-2.

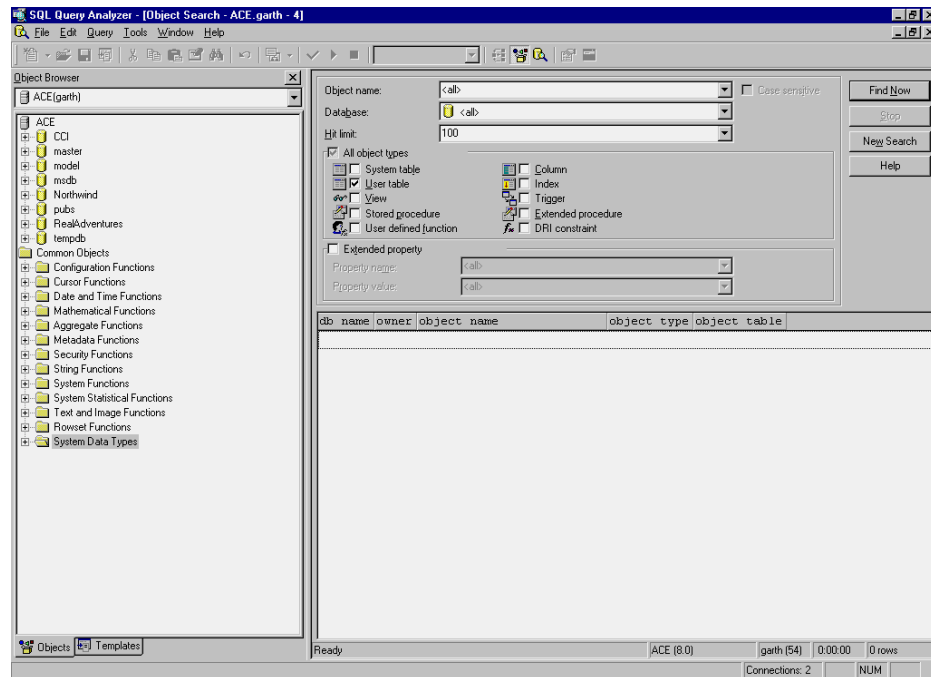


Figure 1-2. Object Search

Object Search gives you the ability to restrict a search by object name, database location, object type, or extended property name or value.

T-SQL Debugger

The T-SQL Debugger is new to SQL Server 2000, but there have been other stored procedure debuggers available as add-ins to Visual C++, Visual Basic, and Visual Interdev. This tool is described in detail in the “Debugging Stored Procedures” section of Chapter 11.

General Comments on Query Analyzer

The following sections should help you better understand some of the additional features available in Query Analyzer.

Displaying Results

In pre-2000 versions of SQL Server the data displayed in the Results Pane was shown in plain text. In SQL Server 2000, the default behavior is to display the results in a grid. You also have the ability to pipe the results generated by a statement to a file. You can toggle these options by clicking Query on the main menu and selecting the desired option. Should you want to revert back to pre-2000 behavior and always display results in text, you change the default setting by completing the following:

1. Click Tools on the main menu and select Options.
2. Click the Results tab and change the option for the Default Results Target select box.
3. Click OK. Now the default is permanently changed.

Commenting Multiple Statements

As you build the statements used to create the various database objects required by your project, you often fill the Editor Pane with quite a bit of text. Should you have statements that you do not want to execute, but are not quite ready to delete from the pane, you can easily comment them out by completing the following:

1. Highlight the statement(s) you wish to comment out.
2. Click Edit on the main menu, highlight Advanced and select Comment Out.

This adds the inline comment marks double-dashes (--) to the start of each line. You can use the same Advanced option to remove the comment marks as well.

Add-In Tools

You can add shortcuts to commonly used programs to the Tools option from the main menu. For example, I often use the GWD Text Editor when writing ASP pages, so I added a shortcut to GWD using this approach. In order to add a program shortcut complete the following:

1. Click Tools on the main menu and select Customize.

2. Click the Tools tab and then click the Add icon (top-most icon with red arrow pointing to the right).
3. Type in a descriptor for the shortcut and press Enter.
4. Type in or select (using the Ellipsis button) the location and name of the file used to launch the application and click OK.

Once this process is complete, a shortcut to the application is available under the Tools option. Simply highlight Tools, select the shortcut and the program is launched.

Shortcuts to Stored Procedures

You can add shortcuts to commonly used stored procedures by associating the procedure with a defined key-combination. In order to create a hot-key combo shortcut to a stored procedure complete the following:

1. Click Tools on the main menu and select Customize.
2. Select an open key-combo, type in the stored procedure name, and click OK.

Once this is complete the stored procedure will be executed when both the Editor Pane is active and the key-combo is pressed. We will cover how this done in detail in the middle of Chapter 11 in the sidebar titled “Configure Hot-Key Combo in Query Analyzer.”

Explore Query Analyzer

Query Analyzer contains quite a bit of functionality and has a number of configuration options. Since you will be spending so much time using this tool I encourage you to explore all the options available in the Tools, Options dialog. Understanding how Query Analyzer can work for you will make you a more efficient programmer.

Transact SQL

T-SQL is Microsoft's implementation of SQL (Structured Query Language) and includes both standards-dictated and extended functionality. In order to fully understand the last sentence you need to know that there is a *standard* SQL that is defined and published by ANSI (American National Standards Institute) and

ISO (International Organization for Standardization). The ANSI/ISO Standard is commonly referred to as ANSI SQL and includes all the functionality, syntax and data types that a software vendor's implementation of SQL must support in order to be ANSI-compliant. ANSI SQL (current version SQL-99) has three levels of compliance: entry, intermediate and full. To meet one of the levels of compliance a vendor's implementation of SQL must support all the functionality, syntax and data types of the particular level. SQL Server 2000 is entry-level compliant with SQL-92—the previous ANSI Standard—and, just so you know, there is no implementation of SQL (e.g., Oracle's PL/SQL) that is fully compliant with the ANSI Standard.

At this point you may be wondering two things:

1. Why is T-SQL only compliant with entry-level SQL-92?
2. Why would SQL 2000 provide extended functionality that is not required to be ANSI-compliant?

The answer to the first question is simple: it would have taken too much time to implement the new standard in the 2000 version of the product. According to Microsoft, it does plan to adhere to entry-level compliance of SQL-99 with the next major release of SQL Server. (I have no idea when this will be, and Microsoft isn't giving too many hints.) Don't worry about the lack of compliance with the new standard, though. Rest assured that you will be able to perform all the data definition and manipulation operations necessary to build world-class database applications with the current version of the product.

The answer to the second question is that Microsoft wants to make T-SQL easier to use than pure ANSI SQL. Let's face it, certain aspects of SQL are a little difficult to comprehend and not everyone has the time learn how to implement all their solutions using ANSI SQL. If Microsoft has the ability to simplify certain data definition and manipulation operations by extending the functionality of ANSI SQL, the company will produce a more marketable product and certainly make the lives (at least the programming aspect) of its customers a little easier.

There is, however, a group of professionals (let's call them SQL purists) who are against extending the functionality of ANSI SQL because using SQL that is not ANSI-compliant produces non-portable code. Non-portable means you cannot take the exact code that works in SQL Server and use it in another RDBMS (relational database management system) like Oracle or Informix. The purists make a valid point; but quite frankly, I don't think there are a lot of developers whose code will be used with more than one RDBMS. As far as I am concerned, the only time the extra effort required to use pure ANSI SQL is justified is when you are a software vendor (e.g., SAP) who supports multiple RDBMSs.

ANSI versus T-SQL Example

Let's take a look at an example that shows how a T-SQL extension of ANSI SQL can make programming a little easier. Most of you are familiar with columns whose value auto-increments when a new row is added to a table. Those of you with Access experience will know that the application uses an AutoNumber data type to implement this functionality. SQL Server does not have a data type that causes a column to auto-increment. Instead, it uses a property associated with a column's definition to indicate that the value will be incremented when a new row is added. The following CREATE TABLE statement shows how to create a column whose value increments by 1 when a new row is added to the table. Please note that the result of executing the statement in Query Analyzer is shown below the "--Results--" indicator.

```
USE tempdb
go
CREATE TABLE Customers
(
  Cus_UniqueID int IDENTITY(1,1) PRIMARY KEY,
  Cus_Name varchar(30)
)
--Results--
The command(s) completed successfully.
```

The **Cus_UniqueID** column is defined with an int (integer) data type and the IDENTITY property is what makes the column increment by 1 when a new row is added. The (1,1) after IDENTITY indicates the seed and increment value can be set when the table is created. If you wanted to have an initial value of 10 and increment it by 5 each time a record is added, you would use (10,5). You can also omit the (seed, increment value) part of the IDENTITY property, in which case the values default to one and one. If you use Enterprise Manager to create a table it is no more work than clicking a check box to add the property to a column. It is a little more work than Access, but still pretty easy.

The following code shows how to insert and review a row inserted into the **Customers** table.

```
USE tempdb
Go
INSERT Customers (Cus_Name) VALUES ( 'Big Red Machine Co.')
```



```
SELECT Cus_UniqueID, Cus_Name
FROM Customers
--Results--
(1 row(s) affected)
```

```

Cus_UniqueID Cus_Name
-----
1           Big Red Machine Co.

```

When a column is defined with the IDENTITY property, it is not referenced in the INSERT statements executed on the table. SQL Server determines the next value and populates the column accordingly.

The ANSI SQL approach to implementing an auto-incrementing column does not involve setting any column-level properties. Instead, you modify the INSERT statement executed on the table so that a new value is calculated when a record is added. The following table structure is needed to demonstrate this approach.

```

USE tempdb
go
CREATE TABLE Customers_ANSI
(
    Cus_UniqueID int PRIMARY KEY,
    Cus_Name varchar(30)
)
--Results--
The command(s) completed successfully.

```

The following INSERT statement will calculate the next value dynamically when a new row is added to the **Customers_ANSI** table.

```

USE tempdb
go
INSERT Customers_ANSI
SELECT COALESCE(MAX(Cus_UniqueID)+1,1),
       'Big Red Machine Co.'
FROM Customers_ANSI

SELECT Cus_UniqueID, Cus_Name
FROM Customers_ANSI
--Results--
(1 row(s) affected)

```

```

Cus_UniqueID Cus_Name
-----
1           Big Red Machine Co.

```

It is not important that you understand exactly *how* the INSERT statement works. The point here is to show that using a T-SQL extension is quite a bit easier than using pure ANSI SQL.

You must decide which approach to use in your applications. As for me, I am going to risk the wrath of the SQL purists and use the IDENTITY approach in both this book and my real-world applications.

TIP ANSI versus T-SQL Extension

There is no single resource that details what functionality available in T-SQL is not ANSI-compliant. There is, however, a configurable option that can be set to warn you when you are using code that does not adhere to the specified level of compliance. The option is called FIPS_Flagger and accepts the three levels of compliancy (entry, intermediate, and full) as arguments.

FIPS_Flagger is a very unusual name when compared to other configurable settings available in SQL Server 2000. The name is derived from the Federal Information Processing Standard (FIPS), which was written by the National Institute of Standards and Technology under the direction of Congress. FIPS defines the minimum functionality to which a product must adhere before the US Government can purchase it.

For more information on this option, see the Books Online topic: Set Fips_Flagger.

T-SQL Syntax Elements

Like any other programming language, T-SQL is composed of different syntax elements. The syntax elements of T-SQL are listed in Table 1-2.

Table 1-2. Syntax Elements of T-SQL

ELEMENT	DESCRIPTION
Identifiers	Names of the database objects within an instance of SQL Server. All objects (e.g., databases, tables, stored procedures, views) have an associated identifier. There are two types of identifiers available in SQL Server 2000: Regular and Delimited. Regular identifiers conform to the rules for specifying an identifier (e.g., no embedded spaces) while Delimited identifiers do not. When a Delimited identifier is referenced, it must be surrounded by either brackets or double-quotes (e.g., [Table Name]). For more information on the rules for identifiers, please see the Books Online topic: Using Identifiers.
Data Types	Types of data a column, variable or parameter can contain. The data types available in SQL Server 2000 are discussed in Chapter 2.
Functions	Code elements that accept zero, one or more parameters and return a value to the calling statement. SQL Server 2000 has both built-in and user-defined functions. The GETDATE() function (which returns the server's current system time) is an example of a built-in function. One might create a user-defined function to more easily calculate the number of business days in a given date range.
Expressions	SQL Statements or elements that can be resolved to a single value. A column, variable or IF statement are examples of statements or elements that resolve to a single value.
Comments	Text descriptions that allow code to be more easily understood. There are two methods of commenting code in SQL Server 2000. You can either use a double-dash (--) to comment a single line of code or the foreshlash asterisk...asterisk foreshlash (/...*/) to comment one or more lines of code.
Keywords	Reserved words used by SQL Server 2000. There are more than 150 reserved keywords in this version of SQL Server and they are detailed in the Books Online topic: Reserved Keywords.

T-SQL DDL and DML

T-SQL is comprised of two types of statements: data definition language (DDL) and data manipulation language (DML). The statements that compose each are listed in the following sections.

DDL Statements

DDL statements are used to create, alter or delete a database object. These statements affect the *structure* of the object and no data is added, updated or deleted. The DDL statements in T-SQL follow. The text of each statement should make it obvious the function it performs.

CREATE DATABASE	ALTER DATABASE
DROP DATABASE	CREATE TABLE
ALTER TABLE	DROP TABLE
CREATE VIEW	ALTER VIEW
DROP VIEW	CREATE PROCEDURE
ALTER PROCEDURE	DROP PROCEDURE
CREATE TRIGGER	ALTER TRIGGER
DROP TRIGGER	CREATE FUNCTION
ALTER FUNCTION	DROP FUNCTION

DML Statements

DML statements are used to insert, update and delete data held in the objects defined with DDL statements. The DML statements available in T-SQL are:

SELECT	INSERT
UPDATE	DELETE
TRUNCATE TABLE	

In addition to the statements listed here there is another *class* of DML statements that are categorized as control-of-flow. The control-of-flow statements are discussed in detail in Chapter 4.

Naming Convention

A discussion on naming convention can often lead to harsh words from even the nicest developers. In my seven years of working with databases I have seen quite a few different naming conventions, and the ones that bothered me the most had little or no consistency. One could argue that a naming convention that is not consistent really is not a naming convention at all. I agree and think consistency should be maintained at all costs.

The naming convention you use is not dictated by SQL Server (other than, of course, the rules for identifiers), but by either personal preference or that mandated by your employer. The naming convention used for table and column names referenced in this book is described here. The naming convention used for other database objects is described when those objects are introduced if the convention is not intuitively obvious.

I use mixed-case plural descriptors for table names. A table that holds information about customers would be named **Customers**. A table that holds information about the orders a customer places would be named **CustomersOrders**.

I use the first three characters of the table name in which a column is located as a prefix for column names when a one-part name is used and the first three characters of each part when a multi-part table name is used. The primary keys for the **Customers** and **CustomersOrders** tables would be **Cus_UniqueID** and **CusOrd_UniqueID**, respectively. The “UniqueID” portion is not dictated by the table name, but by the value held in the column. The column that held the customers’ names would be **Cus_Name** and the one that held the addresses would be **Cus_Address**. When a foreign key is placed in a table the original column name is used. A partial listing of the columns contained in **Customers** and **CustomersOrders** is shown in Figure 1-3.

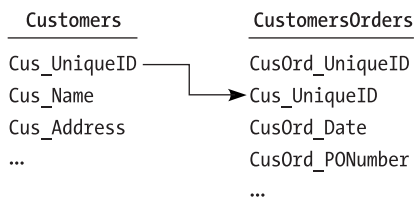


Figure 1-3. **Customers** and **CustomersOrders** Table Layouts

I have often seen naming conventions that attempt to embed the data type in a column name, but this has never appealed to me. From a relational database standpoint, I am more interested in foreign key references than the data type used to define a column. Plus, I do not see much value in knowing a column holds character data unless I also know the maximum allowable length. Column names like **strCustomerName** or **intCustomerID** only leave me wondering if the maximum length of the customer’s name is 30, 40, or 50 characters and if the maximum value used to uniquely identify a customer is 255 or 32,767 or something larger.

Deja.com Power Search

I am a big fan of using the resources on the Internet to help me find better ways to implement solutions for my clients. One of the most helpful sites I have found is Deja.com at <http://www.deja.com>. Deja.com has a newsgroup search feature called Power Search that allows you to focus on a particular newsgroup (their terminology is Forum) and specify search criteria, such as Subject and Author.

You can access Power Search at http://www.deja.com/home_ps.shtml and use the available parameters to limit the scope of the search. When I want to research a SQL Server administrative issue, for example, I populate the Subject field with a keyword used to designate the topic and the Forum field with "microsoft.public.sqlserver.administration." I then execute the query by clicking Search.

If you are not sure which newsgroup is the most applicable for your issue, you can use the wildcard character for searching multiple newsgroups at the same time. For example, if you wanted to search all the Microsoft SQL Server newsgroups, you would populate Forum with "microsoft.public.sqlserver.*"

Using Power Search is a heck of a lot faster than posting to a newsgroup and then waiting for someone to respond. Plus, it is highly likely that the particular problem you are trying to solve has been previously addressed in the newsgroups, so why waste bandwidth and other people's time by requesting help when the answer is only a few clicks away?

Before You Go

This chapter presented an introduction to Query Analyzer, a general overview of T-SQL, and the naming convention used for the tables and columns presented in this book. It also provided a very useful Internet-related tip to help you quickly find information, troubleshoot problems, or see how other developers are implementing similar solutions.

The next chapter provides a detailed description of the data types available in SQL Server 2000. It also covers a couple of topics that should allow you to better understand why there are so many different data types available and some of the advantages of using one over another.



<http://www.springer.com/978-1-893115-83-5>

Code Centric: T-SQL Programming with Stored
Procedures and Triggers

Wells, G.

2001, XXIII, 695 p. 137 illus., Softcover

ISBN: 978-1-893115-83-5

A product of Apress