TABLE OF CONTENTS

Chapter 1: Database Design and Performance
Excerpted from: Microsoft® SQL Server™ 2000 Unleashed, by Ray Rankins, Paul Bertucci, and Paul Jensen

  - Basic Tenets of Designing for Performance
  - Logical Database Design Issues
    - Normalization Conditions
    - Read Committed
    - Normalization Forms
    - Serializable
  - Denormalizing the Database
    - Denormalization Guidelines
    - Querying the syslockinfo Table
    - Viewing Locking Activity with SQL Enterprise Manager
    - Essential Denormalization Techniques
  - Indexes and Performance
    - Evaluating Index Usefulness
    - Distribution Data
  - Index Design Guidelines
    - Clustered Index Indications
    - Nonclustered Index Indications
    - Index Covering
    - Composite Indexes Versus Multiple Indexes
  - SQL Server Index Maintenance
    - Running DBCC SHOWCONTIG
    - Extent Fragmentation
    - Minimizing Locking Contention
    - Fixing Fragmented Tables
    - Setting the Fill Factor
    - Reapplying the Fill Factor
  - Updates and Performance
    - Deferred Update
    - Lock Granularity Hints
    - Update in Place
  - Database File Groups and Performance
  - RAID Technology
    - RAID Level 0
    - RAID Level 1
    - RAID Level 10
    - RAID Level 15
  - Summary

Chapter 2: Monitoring, Optimizing, and Troubleshooting Server Performance
Excerpted from: MCSA MCSE 70-290: Managing and Maintaining a Microsoft® Windows® Server 2003 Environment Exam Cram™2, by Dan Balter

  - Monitoring and Analyzing System Events
    - Working with the Event Viewer
    - Working with the System Monitor
  - Monitoring System Performance
    - Using System Monitoring in Real Time with Task Manager
    - Using Real Time Monitoring with System Monitor
Chapter 3: Delivering Business Intelligence and Analytics on Demand

Chapter 4: Configuring, Tuning, and Optimizing SQL Server Options
Chapter 5: Using Stored Procedures and Cursors

What Is a Stored Procedure?
Stored Procedure Pros and Cons
How to Create a Stored Procedure
How to Modify a Stored Procedure
Control-of-Flow Language
  The DECLARE Statement
  The GOTO Statement
  The BEGIN...END Statement
  The IF...ELSE Statement
  The WAITFOR Statement
  The RETURN Statement
  The WHILE, BREAK, and CONTINUE Statements
  The PRINT Statement
  The RAISERROR Statement
Comments
Parameters Used with Stored Procedures
  Input Parameters
  Output Parameters
Commonly Used Global Variables
How to Debug a Stored Procedure
  Transact-SQL Debugger
  Transact-SQL Debug Statements
What is a Cursor?
Creating a Cursor
  Step 1: DECLARE the Cursor
  Step 3: FETCH the Cursor
  Step 4: CLOSE or DEALLOCATE the Cursor
  Position UPDATE and DELETE
  Global Variables
Putting It All Together
  Example 1: Loop Through a Table
  Example 2: Display Object Names and Object Types
Summary
Chapter 6: Planning an Installation or Upgrade

Development an Installation Strategy and Plan
Step 1: Determine System Strategy and Plan
Step 2: Select the Right Platform
Step 3: Answer Required Questions and Understand Why They Are Important
Step 4: Install SQL Server
Developing an Upgrade Strategy and Plan
Upgrade/Installation Planning FAQ
Summary
The Upgrade Checklist

Chapter 7: Data Access

My Proposal for a Data Access Pattern
How Stored Procedures Are Normally Called with ADO.NET
My Data Access Proposal in Detail
The Generated SQL Script
External Design of My Data Access Proposal
Internal Design of My Data Access Proposal
Debugging
Connection Strategy
OUTPUT Parameters
Tricky Cases of SQL
Disadvantages with the Proposal
The Data Access Proposal in the Architecture
What to Use for Carrying Data
Saying More in Fewer Words When Calling Stored Procedures
XLM
Other Formats
Bitmaps
Data Transportation Between Stored Procedures
Server-Side Caching
New Serve-Side Caching Potential with .NET
Dealing with Schema Changes
Using User-Defined Data Types
Using Too-Large CHAR and VARCHAR Parameters
Dragging Out Parameters from Stored Procedures and Using Them as Constants
Using Views and/or UDFs
Evaluation of My Proposal
What’s Next
References

Chapter 8: ADO.NET- The Data Access Model for the .NET Compact Framework
Excerpted from: *Microsoft® .NET Compact Framework Kick Start*, by Erik Rubin and Ronnie Yates

Introducing ADO.NET on the .NET Compact Framework
Caching Data with the DataSet
Looking Inside the DataSet: DataTables, DataRows, and DataColumns
Inserting Data into a DataSet
Building a DataSet to Hold a Phone Book
Extracting Data from a DataSet
Altering Data in a DataSet
Designing a PhoneBook Application with a DataSet
Troubleshooting Common DataSet-Related Errors
Understanding Constraints
Adding Constraints to a Dataset
Adding a UniqueConstraint
Working with a UniqueConstraint by Example
Preventing NULL Values in a DataColumn
Setting Up Autoincremented Fields
Creating Autoincremented Field Code by Example
Updating the PhoneBook Application with Constraints and Autoincremented Fields
Modeling Relational Data with the DataSet
Deriving DataColumn Values with Expressions and Computed Fields
Expressing Parent-Child Relationships in a DataSet
Creating Bindable Views of Data with a DataView
Sorting with the DataView
Sorting and Filtering Data in a Sample Application
Tracking Changes in a DataRow
Filtering with the DataView
Adding Data into a DataView
Using a DataView in a Sample Application: DataView_SortByRowState_AddTables
Binding Data to a Control
Binding to a DataGrid
Using Data Binding in a Sample Application
Comparing the Compact DataSet with the Desktop DataSet
In Brief

Chapter 9: XQuery and SQL Server
Excerpted from: A First Look at ADO.NET and System.Xml v. 2.0, by Alex Homer, Dave Sussman, and Mark Fussell

A Very Brief Overview of the XQuery Language
Where’s the Data Located?
FLWOR Expressions
The XQueryProcessor Class
The Properties and Methods of the XQueryProcessor Class
The XmlCommand Class
Querying an XML Document with the XQueryProcessor Class
Querying Multiple XML Documents with the XQueryProcessor Class
Using an XQueryProcessor with an XLM View
Using an XQueryProcessor with Multiple XML Views
Using an XQueryProcessor with Multiple XML Views and Multiple Database Connections
Using an XQueryProcessor with Multiple XML Views and XML Documents
The XsltProcessor Class
Transforming XML Documents with the XsltProcessor Class
The Common Query Architecture
SQL Server as an XML Database
The XML Data Type in SQL Server “Yukon”
Typed and Untyped XML Data Columns
The XML Schema Repository in SQL Server “Yukon”
Inserting and Selecting Against an XML Data Column
Querying and updating the Contents of an XML Data Column
Binding Relational Data Inside XML
Indexing an XML Data Column

Summary
Index
Chapter 10: Monitoring and Tuning SQL Server Databases

Introduction
Tools Available for Monitoring and Tuning
   Stored Procedures Used to Diagnose and Optimize
   Database Console Command (DBCC)
   Alternative Mechanisms
Optimizing the OS Configuration
   Using Performance/System Monitor
   Counter Values
   The Event Viewer
   The Window Application Log
   Query Governor Cost Limit
Optimizing SQL Server Configuration
   Current Server Activity
   Stored Procedures
Monitor Activity with the Profiler
   Defining a Profiler Trace
   Profiler Traces to Diagnose Locking
   Using Profiler Results
Trace Playback and Diagnosis
   Playback Requirements
   Performing the Replay
   Templates and Wizards for Specific Monitoring
SQL Server Optimizer
   Query Analyzer
   Proactive/Automated Optimization
Security Objects
   C2 Security
   Statement and Object Permissions
   Security Audits
   Exercises
   Review Questions
   Exam Questions
   Answers to Review Questions
   Answers to Exam Questions

Part II  Final Review
Fast Facts
SQL Server 2000 Requirements
   SQL Server 2000 Enterprise Edition
Database Design for SQL Server 2000
   ER Modeling
   Primary Keys
   Foreign Keys
   One-To-One Relationship
   One-To-Many Relationship
   Many-To-Many Relationship
   Entity Integrity
   Domain Integrity
   Referential Integrity
   Advanced Referential Integrity Options
SQL Server Data Types
Physical Database Design and Implementation
   Database Creation
   Shrinking Files
Column Properties
Check Constraints
Clustered Indexing
Nonclustered Indexing
Encryption Can Secure Definitions
Schema Binding
Indexed Views
Data Integrity Options

Querying and Modifying Data
SQL SELECT
DATEADD, DATEDIFF, DATENAME, DATEPART
Inserting Data
Inserting Data Using SELECT
Deleting Data
Updating Data

Advanced Data Retrieval and Modification
Joins
Using GROUPS BY
Using GROUP BY and HAVING
Using COMPUTE and COMPUTE BY
Using OPENROWSET and OPEN QUERY
Linked Server
XML
Data Transformations

Programming SQL Server 2000
Scripts, Batches, and Transactions
Variable Types
Global Variables
Using Cursors
Lock Isolation Levels
Designing and Managing Transactions
Statement Permissions
Object Permissions
User Roles
Fixed Roles
Application Roles

Working with Views
Views
Partitioned Views
Broken Ownership Chains

Use of Triggers
Recursive Triggers
INSTEAD OF Triggers

Stored Producers and User-Defined Functions
Stored Procedures
Cursor Status
Error Handling
User-Defined Functions

Boosting Performance with Indexes
Clustered Indexes
Nonclustered Indexes
Index Selectivity
Indexed Views
Indexed Views Requirements and Restrictions

Implementing and Understanding Replication Methodologies
Monitoring and Tuning SQL Server Databases
Tools
Simple Network Management Protocol (SNMP)
Chapter 11: XML and SQL Server 2000
Excerpted from: XML and SQL Server: Developing Web Applications, by Daniel K. Appelquist

Retrieving Data in XML Format
FOR XML
FOR XML AUTO
FOR XML EXPLICIT
Communicating with SQL Server over the Web
Under the Hood
Retrieving Data in XML Format-Continued
SQL Queries in URLs
Template Files
XPath Queries
HTTP Post Queries
XML Views
Defining XMLViews
Let SQL Server Do the Work
Working with XML Documents
OPENXML
Summary

Chapter 12: SQL Server Internals

SQL Server Memory Management
The Buffer Manager and Memory Pools
The Buffer Manager
Accessing Memory Buffers
The Checkpoint Process
The Lazywriter Process
Keeping Pages with Cache Permanently
Large Memory Support
The Log Manager
SQL Server Process Management
   SQL Server Threads
SQL Server Disk I/O
   Asynchronous I/O
   Scatter-Gather I/O
   Read Ahead Reads
   Merry-go-Round Scan
SQL Server Storage Structures
Database Files and Filegroups
   Primary Data File
   Secondary Data Files
   The Log File
   Using Filegroups
   On-Demand Disk Management
Database Pages
   Page Types
   Examining Page Content
   Data Pages
   Index Pages
   Differential Changed Map Pages
   Bulk Changed Map Pages
Tables
   Size Limits for Rows and Columns
   Heap Tables
   Clustered Tables
Indexes
   Clustered Indexes
   Nonclustered Indexes
   SQL Server Index Maintenance
Data Modification and Performance
   Inserting Data
   Deleting Rows
   Updating Rows
Summary
Introducing the VERITAS “SQL Server Performance Series”

WHAT IS THE “SQL SERVER PERFORMANCE SERIES?”

The VERITAS “SQL Server Performance Series” is a yearlong educational program designed by VERITAS and developed in concert with the Pearson Technology Group. The program provides expert content on topical SQL Server development and runtime concerns with a focus on application performance management. The foundation of the program is a custom e-book. The e-book consists of twelve unique chapters from twelve different books. Each month a new chapter will be released and promoted. VERITAS will also conduct a Webcast each month with one or multiple authors. Each Webcast will focus on the specific content of the chapter released that month and will provide a forum to ask industry experts SQL Server questions.

CONTENT OF THE E-BOOK

Unlike more traditional e-books that offer content from a single book, the e-book from the “SQL Server Performance Series” is a compilation of content from twelve different books and over twenty different authors. Each chapter will focus on a different aspect of SQL Server application development and deployment.

Chapter 1 Database Design and Performance
This chapter will help you understand some of the key application design issues that will ensure you have a reliable and high-performance application. These factors include logical design, physical design, choice of hardware, network bandwidth, client and server configuration, data access techniques, and application architecture.

Chapter 2 Monitoring, Optimizing, and Troubleshooting Server Performance
This chapter describes how Server performance can degrade over time as more users, more workstations, and more demands are placed on server resources and ultimately impact the performance of SQL Server. Windows Server 2003 offers administrators several built-in tools for monitoring, optimizing, and troubleshooting a server's performance.

Chapter 3 Delivering Business Intelligence and Analytics on Demand
This chapter provides an overview of data warehousing and analytic applications that support business intelligence. The design of these systems is fundamentally different from the more common transaction-oriented relational databases so we delve into some of the implementation details of data warehouses.
Chapter 4  Configuring, Tuning, and Optimizing SQL Server Options
This chapter will delve into what can be done in the SQL Server configurable options—particularly, what can be improved that SQL Server isn't automatically tuning already. By setting the values of several key SQL Server configuration parameters, you can fine-tune SQL Server to provide excellent performance and throughput.

Chapter 5  Using Stored Procedures and Cursors
This chapter will enhance your knowledge of stored procedures and cursors. As a DBA, you use stored procedures frequently. Microsoft supplies many stored procedures that you use to perform database and system maintenance. You will also find that you are frequently required to write your own stored procedures to perform specific DBA tasks for your organization or to help a group of developers solve a complex business problem.

Chapter 6  Planning an Installation or Upgrade
In this chapter, you will learn to develop plans and strategies to help you correctly install or upgrade SQL Server. Why bother with a planning stage? Why not just skip right to the installation or upgrade? SQL Server installation and upgrading is a simple process, but, by planning, you can make the correct decisions that affect the performance and operation of SQL Server before the installation.

Chapter 7  Data Access
Most distributed applications rely on a data tier: a single layer where all information resides. The data tier is one of the most critical areas of a distributed application because a disproportionate amount of the weight tends to fall on it. All too often, the data tier becomes the first bottleneck of a growing system as demand outstrips the system's ability to create connections or retrieve data.

Chapter 8  ADO.Net – The data access model for the .Net Compact Framework
This chapter will first examine the retrieval and update of data from a SQL Server database. It will also examine in detail the process of reconnecting to the data source and sending updated data back into the database.

Chapter 9  XQuery and SQL Server
In this chapter, we'll look in more detail at the XQuery language, the XQueryProcessor class, and how they can be used to perform queries over XML documents loaded into an XPathDocument2 and over SQL Server using XML views.

Chapter 10  Monitoring and Tuning SQL Server Databases
This chapter will review how SQL Server and the operating system work together to provide a productive database management environment. There are many SQL Server hooks into the operating system and OS resources available to observe the database server as it operates. Other tools allow for quick diagnosis of problems that may be affecting the server.
Chapter 11  XML and SQL Server 2000
You will come away from this chapter with concrete knowledge of how XML and SQL can be married in a powerful way at the level of a relational database server. You can apply this knowledge to your own projects, using either SQL Server 2000 or some other relational database server, such as Oracle, that might have similar features.

Chapter 12  SQL Server Internals
This chapter looks at the internal architecture as well as the storage structures in SQL Server and how the storage structures are maintained and managed. This information will help you better understand various issues associated with migrating from an earlier version or from a different RDBMS.
As was already discussed in Chapter 37, “Monitoring SQL Server Performance,” many components of a system can be monitored and tuned to yield high performance. These addressable components can be server hardware, network configuration, memory allocation, disk systems usage, database design techniques, and so on. This chapter will delve a bit more into what can be done in the SQL Server configurable options—particularly, what can be improved that SQL Server isn’t automatically tuning already. By setting the values of several key SQL Server configuration parameters, you can fine-tune SQL Server to provide excellent performance and throughput.

Note that with each release of SQL Server, less needs to be tuned from the SQL Server configuration point of view. With the advent of self tuning or self configuring options, it is only a matter of time before most of your server tuning time will be spent elsewhere, such as with the operating system, disk systems, and network interfaces—and only because SQL Server can’t reach there, yet.

**SQL Server Instance Architecture**

Figure 40.1 illustrates the address space architecture of an instance of SQL Server 2000. When you fire up a SQL Server instance, two main areas are allocated: the code area and the
CHAPTER 40  Configuring, Tuning, and Optimizing SQL Server Options

memory pool area. The code area will be mostly static executable code of the SQL Server Kernel, Server Net-library DLLs, Open Data Services code, the Stack Space, and a variable code area that contains distributed query OLE DB providers, OLE Automation objects, and extended-stored procedures as they are needed by user requests.

FIGURE 40.1  SQL Server 2000 instance architecture.

The memory pool area of SQL Server is the most dynamically changing part of an instance. Even now, the once-static System Data Structures and User Connection Structures (Connection context) are controlled by user requests and dynamically allocate structures as they are needed.

By default, SQL Server will try to keep the amount of virtual memory allocations on the computer at 4–10 MB less than the physical memory available.

The rest of the memory pool area is divided into Procedure Cache, Data Cache (buffer cache), and Log Cache. SQL Server is actively adjusting these for optimal performance. It wasn’t that long ago that the system administrator had to do all of this manually. Many of the configurable options will directly relate to optimizing this address space.
Configuration Options

For SQL Server 2000, this chapter will discuss the configurable options in two distinct categories: basic options and advanced options. The advanced options are a super-set of the basic options. As each option is discussed, it will be noted whether it is self-configuring. Several self-configuring options are available that adjust themselves dynamically according to the needs of the system. In most cases, this eliminates the need for setting the values manually. Sometimes you won’t want to rely on certain self-configuring values, depending on how SQL Server is being used.

As you can see in Figure 40.2, SQL Server provides configuration parameters that the system administrator can set to maximize the performance of a system. You can set these parameters either by using the sp_configure system-stored procedure or by using SQL Server Enterprise Manager and the server configuration properties.

Now, the only questions that need to get answered are “What configuration options do I need to set that aren’t already fine?” And, “How do I set them?”

To answer the “what” question, you will first determine for what purpose the applications are using SQL Server. This must include understanding variables such as the number of potential connections to support, the amount of resources available on the box, the size of the database, the type of data accesses occurring, and the workload being put on SQL Server. After you know this, it will be easy to determine the configuration option setting to adjust.
Figure 40.3 shows a generalization of the types of applications that you might find in the real world that would be implemented using SQL Server 2000 and the general behavior that they elicit. It is not a complete list, just a generalized list. The four basic categories of application processing are online transaction processing (OLTP), applications, data warehouse/data mart applications, online analytic processing (OLAP) applications, and mixed applications (some OLTP along with some data marts on one SQL Server). Because these configuration options are set at the SQL Server level, it is important to know the combined behavior of all application processing.

<table>
<thead>
<tr>
<th>Application type</th>
<th>General processing behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>OLTP</td>
<td>Mix of reads, writes &amp; deletes, Large number of users</td>
</tr>
<tr>
<td>OLAP</td>
<td>Big loads, then read-only, Medium to small number of users</td>
</tr>
<tr>
<td>Data Warehouse</td>
<td>Incremental loads, then read-only, big extracts and reporting, Medium number of users</td>
</tr>
<tr>
<td>Mixed Server</td>
<td>Mix of reads, writes, deletes, big loads, big extracts, Large number of users</td>
</tr>
</tbody>
</table>

FIGURE 40.3 General application processing types for SQL Server.

For each SQL Server configuration option that is discussed, you will address the correct setting based on the generalized application processing behaviors that need to be supported.

The “How do I set them?” question will be discussed now. The next few sections describe all of the types of configuration options available on SQL Server 2000. Those sections will show you how to set these configuration options using both Enterprise Manager and sp_configure system-stored procedure. The rule will be that you can certainly set a configuration option using Enterprise Manager, but that you should keep an sp_configure version of that setting change as a backup in case you need to rebuild the entire server configuration from scratch. In addition, keeping an sp_configure version around in a file will provide a great audit trail of what you did and for what reason.

You have already seen, in Figure 40.2, the Enterprise Manager Configuration Properties dialog box. Now, you will look at the sp_configure equivalent. By executing sp_configure without parameters, you will be given the list of options that can be addressed. When you have just installed a SQL Server instance, you will be able to see only the basic configuration options.
Ten basic configuration options are available:

```
sp_configure
go
```

<table>
<thead>
<tr>
<th>name</th>
<th>minimum</th>
<th>maximum</th>
<th>config_value</th>
<th>run_value</th>
</tr>
</thead>
<tbody>
<tr>
<td>allow updates</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>default language</td>
<td>0</td>
<td>9999</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>max text repl size (B)</td>
<td>0</td>
<td>2147483647</td>
<td>65536</td>
<td>65536</td>
</tr>
<tr>
<td>nested triggers</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>remote access</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>remote login timeout (s)</td>
<td>0</td>
<td>2147483647</td>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td>remote proc trans</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>remote query timeout (s)</td>
<td>0</td>
<td>2147483647</td>
<td>600</td>
<td>600</td>
</tr>
<tr>
<td>show advanced options</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>user options</td>
<td>0</td>
<td>32767</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

By default, all SQL Server users have permission to run this system-stored procedure, but only users who have sysadmin and serveradmin fixed server roles (such as sa) can actually set the value of a parameter.

The proper syntax of the `sp_configure` command is as follows:

```
sp_configure [parameter_name [, parameter_value ]]  
```

where `parameter_name` is the name of the configuration parameter you want to set, and `parameter_value` is the value for the parameter. Both of these parameters are optional.

Parameters set by `sp_configure` take effect at the server level.

Following is a brief explanation of the output of the `sp_configure` command. As you can see, the output consists of five columns:

- **Name**—Name of the configurable option.
- **Minimum**—This is the minimum legal value allowed for this parameter. Passing an illegal value causes SQL Server to return an error.
- **Maximum**—This is the maximum legal value allowed for this parameter. Passing an illegal value causes SQL Server to return an error.
- **Config_value**—This column reflects the values that are going to take effect the next time SQL Server is started. If you change static parameters, the new values are listed under this column.
- **Run_value**—This column reflects the values that SQL Server is currently using. If you change any dynamic parameters, the new values are listed in this column. At the time of SQL Server startup, the `config_value` for all the parameters is copied into `run_value`. Immediately after restart, both columns (`run_value` and `config_value`) should display the same values corresponding to each parameter.
If you specify only the parameter name, SQL Server returns the current configuration value for that particular parameter.

```sql
sp_configure 'allow updates'
go
```

<table>
<thead>
<tr>
<th>name</th>
<th>minimum</th>
<th>maximum</th>
<th>config_value</th>
<th>run_value</th>
</tr>
</thead>
<tbody>
<tr>
<td>allow updates</td>
<td></td>
<td></td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Many more configuration options are available from the total of 36 options. These consist of the original 10 basic options plus 26 advanced options. To see a complete list of all options, you must turn on the Show Advanced Options configuration option with the value 1.

In addition, when using `sp_configure` to change a setting, use the `RECONFIGURE WITH OVERRIDE` statement for the change to take effect immediately. You can also choose to use just the `RECONFIGURE` statement. Depending on the configuration option, it will take effect immediately or not until the server has been restarted.

The following commands will set the Show Advanced Options configuration option and then retrieve the complete list of these options:

```sql
exec sp_configure 'Show Advanced Options', 1/* Advanced config options */
go
RECONFIGURE WITH OVERRIDE /* to have it take effect immediately */
go
sp_configure
```

```
name               minimum maximum config_value run_value
------------------- --------------- --------------- ---------------
affinity mask      -2147483648 2147483647 0 0
allow updates      0 1 0 0
awe enabled        0 1 0 0
c2 audit mode      0 1 0 0
cost threshold for parallelism 0 32767 5 5
cursor threshold  -1 2147483647 -1 -1
default full-text language 0 2147483647 1033 1033
default language   0 9999 0 0
fill factor (%)    0 100 0 0
index create memory (KB) 704 2147483647 0 0
lightweight pooling 0 1 0 0
locks              5000 2147483647 0 0
max degree of parallelism 0 32 0 0
```
Microsoft suggests that only highly experienced SQL Server administrators change these advanced configuration options. You have been warned!

SQL Server internally maintains two tables: syscurconfigs and sysconfigures. The syscurconfigs table contains the current configuration values of SQL Server parameters. These values are shown under the run_value column of sp_configure. The following is what you might expect to see if you query this table directly:

```
SELECT value, config, substring (comment,1,50),status FROM master..syscurconfigs
```

<table>
<thead>
<tr>
<th>value</th>
<th>config</th>
<th>comment</th>
<th>status</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>1</td>
<td>Major revision number of config data.</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>2</td>
<td>Minor revision number of config data.</td>
<td>0</td>
</tr>
<tr>
<td>13</td>
<td>3</td>
<td>Reconfigure revision number of config data</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>Configuration boot source.</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>101</td>
<td>Maximum recovery interval in minutes</td>
<td>3</td>
</tr>
<tr>
<td>0</td>
<td>102</td>
<td>Allow updates to system tables</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>103</td>
<td>Number of user connections allowed</td>
<td>2</td>
</tr>
<tr>
<td>ID</td>
<td>Config Value</td>
<td>Description</td>
<td>Change</td>
</tr>
<tr>
<td>----</td>
<td>----------------------</td>
<td>--------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>0</td>
<td>106</td>
<td>Number of locks for all users</td>
<td>2</td>
</tr>
<tr>
<td>0</td>
<td>107</td>
<td>Number of open database objects</td>
<td>2</td>
</tr>
<tr>
<td>0</td>
<td>109</td>
<td>Default fill factor percentage</td>
<td>2</td>
</tr>
<tr>
<td>0</td>
<td>1537</td>
<td>Media retention period in days</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>115</td>
<td>Allow triggers to be invoked within triggers</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>117</td>
<td>Allow remote access</td>
<td>0</td>
</tr>
<tr>
<td>2049</td>
<td>1127</td>
<td>Two digit year cutoff</td>
<td>3</td>
</tr>
<tr>
<td>1033</td>
<td>1126</td>
<td>Default full-text language</td>
<td>3</td>
</tr>
<tr>
<td>0</td>
<td>124</td>
<td>Default language</td>
<td>1</td>
</tr>
<tr>
<td>255</td>
<td>503</td>
<td>Maximum worker threads.</td>
<td>2</td>
</tr>
<tr>
<td>0</td>
<td>542</td>
<td>Create DTC Transaction for RPC</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>543</td>
<td>Remote connection inactivity timeout</td>
<td>0</td>
</tr>
<tr>
<td>4096</td>
<td>505</td>
<td>Network packet size</td>
<td>3</td>
</tr>
<tr>
<td>0</td>
<td>1505</td>
<td>Memory for index create sorts (kBytes)</td>
<td>3</td>
</tr>
<tr>
<td>0</td>
<td>1517</td>
<td>Priority boost</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>518</td>
<td>Show advanced options</td>
<td>1</td>
</tr>
<tr>
<td>20</td>
<td>1519</td>
<td>Remote login timeout</td>
<td>1</td>
</tr>
<tr>
<td>600</td>
<td>1520</td>
<td>Remote query timeout</td>
<td>1</td>
</tr>
<tr>
<td>-1</td>
<td>1531</td>
<td>Cursor threshold</td>
<td>3</td>
</tr>
<tr>
<td>1024</td>
<td>1540</td>
<td>Minimum memory per query (kBytes)</td>
<td>3</td>
</tr>
<tr>
<td>-1</td>
<td>1541</td>
<td>Query wait (s)</td>
<td>3</td>
</tr>
<tr>
<td>0</td>
<td>1532</td>
<td>Set working set size</td>
<td>2</td>
</tr>
<tr>
<td>0</td>
<td>1534</td>
<td>User options</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1535</td>
<td>Affinity mask</td>
<td>2</td>
</tr>
<tr>
<td>65536</td>
<td>1536</td>
<td>Max text repl size</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>1538</td>
<td>Cost threshold for parallelism</td>
<td>3</td>
</tr>
<tr>
<td>0</td>
<td>1539</td>
<td>Maximum degree of parallelism</td>
<td>3</td>
</tr>
<tr>
<td>0</td>
<td>1543</td>
<td>Minimum server memory size (MB)</td>
<td>3</td>
</tr>
<tr>
<td>2147483647</td>
<td>1544</td>
<td>Maximum server memory size (MB)</td>
<td>3</td>
</tr>
<tr>
<td>0</td>
<td>1545</td>
<td>Maximum estimated cost of query allowed to run</td>
<td>3</td>
</tr>
<tr>
<td>0</td>
<td>1546</td>
<td>User mode scheduler uses lightweight pooling</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>1547</td>
<td>Scan for startup stored procedures</td>
<td>2</td>
</tr>
<tr>
<td>0</td>
<td>544</td>
<td>C2 audit mode</td>
<td>2</td>
</tr>
<tr>
<td>0</td>
<td>1548</td>
<td>AWE enabled in server</td>
<td>2</td>
</tr>
</tbody>
</table>

The sysconfigures table stores the new values of the parameters that were changed since the last SQL Server startup. These values are shown in the config_value column of sp_configure.

SELECT value, config, substring (comment,1,50),status from master..sysconfigures
Go
<table>
<thead>
<tr>
<th>value</th>
<th>config</th>
<th>status</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>101 Maximum recovery interval in minutes</td>
<td>3</td>
</tr>
<tr>
<td>0</td>
<td>102 Allow updates to system tables</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>103 Number of user connections allowed</td>
<td>2</td>
</tr>
<tr>
<td>0</td>
<td>106 Number of locks for all users</td>
<td>2</td>
</tr>
<tr>
<td>0</td>
<td>107 Number of open database objects</td>
<td>2</td>
</tr>
<tr>
<td>0</td>
<td>109 Default fill factor percentage</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>115 Allow triggers to be invoked within triggers</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>117 Allow remote access</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>124 default language</td>
<td>1</td>
</tr>
<tr>
<td>255</td>
<td>503 Maximum worker threads</td>
<td>2</td>
</tr>
<tr>
<td>4096</td>
<td>505 Network packet size</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>518 show advanced options</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>542 Create DTC transaction for remote procedures</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>544 c2 audit mode</td>
<td>2</td>
</tr>
<tr>
<td>1033</td>
<td>1126 default full-text language</td>
<td>3</td>
</tr>
<tr>
<td>2049</td>
<td>1127 two digit year cutoff</td>
<td>3</td>
</tr>
<tr>
<td>0</td>
<td>1505 Memory for index create sorts (kBytes)</td>
<td>3</td>
</tr>
<tr>
<td>0</td>
<td>1517 Priority boost</td>
<td>2</td>
</tr>
<tr>
<td>20</td>
<td>1519 remote login timeout</td>
<td>1</td>
</tr>
<tr>
<td>600</td>
<td>1520 remote query timeout</td>
<td>1</td>
</tr>
<tr>
<td>-1</td>
<td>1531 cursor threshold</td>
<td>3</td>
</tr>
<tr>
<td>0</td>
<td>1532 set working set size</td>
<td>2</td>
</tr>
<tr>
<td>0</td>
<td>1534 user options</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1535 affinity mask</td>
<td>2</td>
</tr>
<tr>
<td>65536</td>
<td>1536 Maximum size of a text field in replication.</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1537 Tape retention period in days</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>1538 cost threshold for parallelism</td>
<td>3</td>
</tr>
<tr>
<td>0</td>
<td>1539 maximum degree of parallelism</td>
<td>3</td>
</tr>
<tr>
<td>1024</td>
<td>1540 minimum memory per query (kBytes)</td>
<td>3</td>
</tr>
<tr>
<td>-1</td>
<td>1541 maximum time to wait for query memory (s)</td>
<td>3</td>
</tr>
<tr>
<td>0</td>
<td>1543 Minimum size of server memory (MB)</td>
<td>3</td>
</tr>
<tr>
<td>2147483647</td>
<td>1544 Maximum size of server memory (MB)</td>
<td>3</td>
</tr>
<tr>
<td>0</td>
<td>1545 Maximum estimated cost allowed by query governor</td>
<td>3</td>
</tr>
<tr>
<td>0</td>
<td>1546 User mode scheduler uses lightweight pooling</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>1547 scan for startup stored procedures</td>
<td>2</td>
</tr>
<tr>
<td>0</td>
<td>1548 AWE enabled in the server</td>
<td>2</td>
</tr>
<tr>
<td>0</td>
<td>1549 affinity64 mask</td>
<td>2</td>
</tr>
</tbody>
</table>

Dynamic parameters are written to both of these tables. Static parameters are written only to the sysconfigures table. At SQL Server restart, all of the values are copied from the sysconfigures table to the syscurconfig table.
Fixing an Incorrect Option Setting

Setting a value too high for a parameter might cause SQL Server to crash during startup. For example, if you set the value of the memory option to a value that is higher than the physical memory on the machine, SQL Server will not start. In this case, you start SQL Server with the -f option. This causes SQL Server to start with the default parameter values (the same values used by the setup program when you installed SQL Server). After SQL Server is running, change the incorrect value to the correct one and restart SQL Server without the -f option.

Setting Configuration Options with SQL Enterprise Manager

As mentioned previously, you can set SQL Server configuration options by using SQL Server Enterprise Manager. Simply invoke Enterprise Manager from the Microsoft SQL Server program group and right-click on the Server folder. Figure 40.4 shows the SQL Server Properties (Configure) dialog box and the different tabs available. Don’t be misled: Some of these tab options are not configuration options that correspond to sp_configure, such as the Security tab and some functions under the Database tab. However, as you will see, Microsoft has done a nice job of organizing and presenting the option information. You will also see that several of the configuration options cannot be managed from this interface. The ones not addressed are rarely used options, such as the AWE-enabled option.

![FIGURE 40.4 Enterprise Manager and configuration properties.](image)

Remember: You must have sysadmin and serveradmin fixed server roles to make changes.
Obsolete Configuration Options

Some of the configuration options available in SQL Server 7.0 are obsolete in SQL Server 2000. Some options, such as “time slice,” were just introduced in SQL Server 7.0 and then immediately became obsolete.

The following parameters are obsolete in SQL Server 2000:

<table>
<thead>
<tr>
<th>Obsolete Configuration Options</th>
</tr>
</thead>
<tbody>
<tr>
<td>default sort order id</td>
</tr>
<tr>
<td>resource timeout</td>
</tr>
<tr>
<td>extended memory size</td>
</tr>
<tr>
<td>spin counter</td>
</tr>
<tr>
<td>language in cache</td>
</tr>
<tr>
<td>time slice</td>
</tr>
<tr>
<td>language neutral full-text</td>
</tr>
<tr>
<td>unicode comparison style</td>
</tr>
<tr>
<td>max async IO</td>
</tr>
<tr>
<td>unicode locale id</td>
</tr>
</tbody>
</table>

One of the all-time favorite options in SQL Server 7.0 was max async IO. The reason that option became obsolete is because it has been completely automated with SQL Server 2000. Previously, max async IO was used to specify the number of simultaneous disk I/O requests that SQL Server 7.0 (and earlier versions) could submit to Windows NT and Windows 2000 during a checkpoint operation. It invariably helped overall throughput on systems that used RAID devices that had extensive disk cache mechanisms. Now, SQL Server 2000 adjusts this automatically.

Configuration Options and Performance

This section explains essential information about most of the SQL Server configuration options and their impact on SQL Server performance. Many of the options don’t have performance implications, but they will be discussed anyway. As part of each option’s explanation, an indication of whether the option is Advanced or Basic will be given along with the option’s default value and whether the option is self-configuring or not. Possible values will also be listed depending on the generalized application processing types that were identified earlier.

Affinity Mask

Type—Advanced option

Default value—0

SQL Server supports symmetric multiprocessing (SMP). SMP support means that a thread is not tied to a particular processor on the machine. This allows SQL Server to run multiple threads simultaneously, resulting in a high level of load balancing across processors. A value of 0 (the default) allows Windows Scheduling algorithm to set the threads affinity.
This qualifies the affinity mask as self-configuring by the operating system. However, when a server is experiencing a heavy load because of other applications running on the same server, it might be desirable to bind thread affinity to a processor.

The affinity mask is a bitmapped field. Starting from the least significant digit, each bit that is set to 1 represents the processor on which SQL Server will spawn its threads. Processors are numbered from 0 to 7. An example of the bit mask values for the first seven processors of an eight-processor system follows; decimal values are shown in parentheses:

<table>
<thead>
<tr>
<th>Bit Mask</th>
<th>Processors Used</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000001 (1)</td>
<td>0</td>
</tr>
<tr>
<td>00000011 (3)</td>
<td>0,1</td>
</tr>
<tr>
<td>00000111 (7)</td>
<td>0,1,2</td>
</tr>
<tr>
<td>00001111 (15)</td>
<td>0,1,2,3</td>
</tr>
<tr>
<td>00011111 (31)</td>
<td>0,1,2,3,4</td>
</tr>
<tr>
<td>00111111 (63)</td>
<td>0,1,2,3,4,5</td>
</tr>
<tr>
<td>01111111 (127)</td>
<td>0,1,2,3,4,5,6</td>
</tr>
</tbody>
</table>

You usually leave the eighth processor alone because many system processes—such as domain controllers—default to that processor.

As an example, if you want to create the affinity for one SQL Server instance thread to use four processors of an eight-processor system, you could set this bit mask to be 15 (00001111). The result would be that SQL Server spawns its threads only on those processors, thus reducing overall reloading of the processor cache. This can be especially evident during heavy system loads.

```
sp_configure 'affinity mask', 15
GO
RECONFIGURE
GO
```

In general, the default affinity value is able to provide ample load balancing across processors. Based on your particular processing load and application types, you will want to allocate CPUs accordingly. Below is a general recommendation of what to specify based on the different application types you are running:

- **OLTP**: Use default value of 0.
- **Data warehouse**: Potentially use 75 percent of available processors to maximize on the huge data loads, large reporting, and number of users.
• OLAP: Use default value of 0.
• Mixed: Use default value of 0.

From Enterprise Manager, SQL Server Properties, Processor tab, just select the targeted processors in the Processor Control section.

**Allow Update**

- Type—Basic
- Default value—0

By default, SQL Server does not allow updates to internal system tables. When you set the allow update value to 1, any user with proper permissions can update system tables.

Sometimes you might want to allow access to system tables.

Following are a couple scenarios:

- In many database applications, business logic is written in stored procedures and triggers. To protect your intellectual property, you might not want anyone to see and modify this logic. One way to accomplish this goal is to delete entries from the syscomments table for the objects you want to protect.

- My database is currently running off a Jaz drive connected to my machine, and sometimes, my Jaz drive is disconnected. In such cases during startup, SQL Server fails to initialize the database file and marks the database corrupt by setting the status bit in the sysdatabases table to 256. To bring the database back to life, I connect my Jaz drive, change the status bit to the original value, and restart the machine. Then SQL Server recognizes the drive and starts the database normally.

**CAUTION**

Only highly experienced users of SQL Server should use the Allow Update option. For example, if you create a stored procedure that modifies system tables when the Allow Update option is turned on, the procedure will continue to be able to modify the system table even if you turn off allow update again. Therefore, be very careful when you set this parameter value to 1.

Even more important is to turn off Allow Update as soon as you are finished with the task of modifying system tables.

From Enterprise Manager, SQL Server Properties, Server Settings tab, select the Allow Modifications box, as shown in Figure 40.5.
FIGURE 40.5 SQL Server Properties: Server Settings.

**AWE Enabled**

Type—Advanced

Default value—0

SQL Server can use the Advanced Windowing Extensions (AWE) API to support large amounts of physical memory—in fact, as much as 8GB of memory on Windows 2000 Advanced Server and up to 64GB on Windows 2000 Data Center. The default of 0 tells SQL Server to use dynamic memory in standard 32-bit virtual address spaces. By enabling AWE, the SQL Server instance does not dynamically manage the size of the address space. The instance holds all memory acquired at startup until it is shut down, and memory pages come from the Windows non pageable pool. This means that none of the memory of the instance can be swapped out. You end up with a great deal of all activity occurring in memory only. This is potentially a fast database engine.

AWE enabled is usually used in conjunction with the max server memory option to control how much memory each SQL Server instance will use.

```
sp_configure 'awe enabled', 1
GO
RECONFIGURE
GO
```
Setting of this option will vary according to the following application types:

- **OLTP**: If memory is available, set to 1.
- **Data warehouse**: Not appropriate for this type.
- **OLAP**: If memory is available and you are not using OLAP file options, set to 1.
- **Mixed**: If memory is available, set to 1.

### Cost Threshold for Parallelism

**Type**—Advanced  
**Default value**—5

SQL Server now supports parallel query execution. Before a query is executed, SQL Server’s cost-based optimizer estimates the cost of execution for a serial plan, a plan that uses a single thread. The option to set the cost threshold for parallelism allows you to specify a threshold in seconds; if the cost of the serial execution plan (in seconds) is greater than the value specified by this parameter, SQL Server will consider a parallel query execution plan. A query will not become a candidate for parallel query execution simply based on this fact. Because parallel query execution is supported only on an SMP server, this value is ignored for non-SMP hardware. For an application that uses many complex queries, set this value to a lower number so that you can take advantage of the parallel query execution capabilities of SQL Server.

```
sp_configure 'cost threshold for parallelism', 2  
go  
RECONFIGURE  
Go
```

Setting of this option will vary according to the following application types:

- **OLTP**: Use default value of 5.
- **Data warehouse**: Many complex queries are candidates for parallelism. Set to low value, perhaps 2 (seconds).
- **OLAP**: Use default value.
- **Mixed**: Use default value.
Cursor Threshold
Type—Advanced
Default value—\(1\)

This option allows you to specify when SQL Server should generate a cursor result set asynchronously. If the optimizer estimates that the number of rows returned by the cursor is greater than the value specified by this parameter, it will generate the result set asynchronously. The optimizer makes this decision based on the distribution statistics for each table that is participating in the join in the cursor.

To determine the optimal value for this parameter, make sure that statistics are up to date (by running `update statistics`) for the tables used in the cursors. By default, SQL Server generates a cursor result set synchronously. If you are using a fair amount of cursors that return a large number of result sets, setting this value to a higher value will result in better performance. Setting this value to \(0\) will force SQL Server to always generate a cursor result set asynchronously.

```sql
sp_configure 'cursor threshold', 100000
GO
RECONFIGURE
GO
```

Setting of this option will vary according to the following application types:

- OLTP: Use default value.
- Data warehouse: A data warehousing environment is the largest potential user of this option due to the high volume of result rows returned by applications using data warehouses. Setting this value to 100,000 is a good starting point.
- OLAP: Use default value.
- Mixed: Use default value.

Default Language
Type—Basic
Default value—\(0\)

This option specifies the language ID currently in use by SQL Server. The default value is \(0\), which specifies the U.S. English system. As you add languages on the server, SQL Server assigns a new ID for each language. You can then use these IDs to specify the default language of your choice. You can add languages using the SQL Server setup program. Adding a language allows SQL Server to display error messages and date/time values in the format that is appropriate for that language. Set this option in the Server Settings tab of the SQL Server Properties dialog box.
Fill Factor

Type—Basic, static

Default value—0

The Fill Factor option allows you to define the percentage of free space on a data page or an index page when you create an index or a table. The value can range from 1–100. Setting the value to 80 would mean each page would be 80 percent full at the time of the create index. SQL Server also allows you to specify the value of fill factor at the server level by providing a fill factor parameter.

```sql
sp_configure 'fill factor', 90
go
RECONFIGURE
```

Setting of this option will vary according to the following application types:

- OLTP: This is a good candidate for leaving space free in pages due to the update, delete, and insert characteristics. Try 90 percent full value first and watch the page split activity.
- Data warehouse: Use default value.
- OLAP: Use default value.
- Mixed: Use default value.

As you can see in Figure 40.6, you set fill factor from the Database Settings tab in the SQL Server Properties dialog box. Just click on the Fixed option and slide the bar until you have the desired fullness of a page.

![SQL Server Properties: Database settings.](image)
Index Create Memory (KB)

Type—Advanced, Self-configuring
Default value—0

The index create memory option is used to control the amount of memory used by index creation sorts. It is a self-configuring option and usually doesn’t need to be adjusted. However, if you are having problems with the creation of large indexes, you might want to try specifying a KB value here that will contain the sort portion of the index create.

```
sp_configure 'index create memory', 1000
GO
RECONFIGURE
GO
```

Lightweight Pooling

Type—Advanced
Default value—0

Lightweight pooling is relevant to SMP environments that are having excessive context switching. By flipping this switch, you might get better throughput by performing the context switching inline, thus helping to reduce user/kernel ring transitions. Lightweight pooling causes SQL Server to switch to fiber mode scheduling.

```
sp_configure 'lightweight pooling', 1
GO
RECONFIGURE
GO
```

Setting of this option will vary according to the following application types:

- OLTP: This is a good candidate for usage if on an SMP environment.
- Data warehouse: This has a good potential for usage if on an SMP environment.
- OLAP: Use default value.
- Mixed: Use default value.
Locks
Type—Advanced, Self-configuring
Default value—0

In earlier versions of SQL Server, the DBA had to specify the number of locks available to SQL Server. If this parameter was set to a low value, a query requiring a large number of locks would fail at runtime. Setting it too high would result in wasting memory that otherwise could be used to cache data. SQL Server 2000 can handle locks dynamically if this parameter is set to the default value (0). SQL Server initially allocates 2 percent of memory available to SQL Server. As lock resource structures are consumed, the lock manager allocates more lock resources to the pool to a maximum of 40 percent of the memory available on SQL Server. Unless you are certain of the overall lock consumption of your application, you probably don’t need to change this value.

```sql
sp_configure 'locks', 10000
GO
RECONFIGURE
GO
```

Max Degree of Parallelism
Type—Advanced
Default value—0

This option specifies the number of threads to be used for parallel query execution. On a non-SMP server, this value is always ignored. For an SMP server, a default value of 0 signifies that all the CPUs will be used for parallel query execution. If you set this value to 1, all query plans will be serialized. If the affinity mask option is on, parallel query execution will take place only on the CPUs for which the affinity mask bit is turned on. In that way, the two options can be used in conjunction. The application types assessment will be the same as described in the affinity mask option.

This option can be set up using the Processor tab of the SQL Server Configuration dialog box; then, in the Parallelism box, choose the number of processors to use for parallelism from the drop-down list box.

```sql
sp_configure 'max degree of parallelism', 4
GO
RECONFIGURE
GO
```
Max Server Memory and Min Server Memory

Type—Advanced, Self-configuring
Default value—2147483647 and 0

Max server memory specifies the maximum amount of memory (in terms of MB) that is available to SQL Server. It is used in conjunction with min server memory and essentially establishes an upper and lower bound for memory allocation. SQL Server uses this memory for user connections, locks, internal data structures, and caching the data. This is the memory pool described earlier. The default value of 2147483647 for the Max Server Memory option means that SQL Server will perform dynamic allocation of memory from the operating system based on available physical memory on the machine. The default value of 0 for the Min Server Memory option means that SQL Server will start allocation memory as it is needed, and then never go below the minimum value after it is reached.

The SQL Server lazywriter process is responsible for making sure that enough memory is available to SQL Server for the optimal number of buffers and Windows so that no excess paging occurs at the operating-system level. The lazywriter process frequently checks physical memory available on the machine. If the memory available is greater than 5MB, lazywriter assigns excess memory to the SQL Server buffer cache.

In addition, watch the Working Set performance counter that shows the amount of memory used by a process (SQL Server in this case). If this number is consistently below the amount of memory for which SQL Server is configured, then SQL Server is configured for more memory than it needs. You can also adjust the Set Working Set Size configuration option.

If SQL Server is the only application running on the machine, you might want to perform static memory allocation. Be careful when you allocate fixed memory to SQL Server. If you allocate more memory to SQL Server than the machine has, SQL Server will fail to start. Use the -f option during startup to bring up SQL Server with the default configuration. Change the value to the correct value, and restart SQL Server.

```sql
sp_configure 'max server memory', 200
GO
RECONFIGURE
GO
sp_configure 'min server memory', 10
GO
RECONFIGURE
GO
```

For a strict fixed allocation of memory for SQL Server, make the min and max values the desired allocation size the same (like 200MB). A fixed amount of memory will then be allocated for SQL Server.
Figure 40.7 shows two possible settings for these configuration options. One shows the Dynamic Configure option set and a minimum and maximum value are established (other than the defaults). The other one shows the fixed memory specification. This fixed memory setting will result in the minimum and maximum values being set to the same desired value.

**FIGURE 40.7** SQL Server Properties: Memory settings.

Setting of this option will vary according to the following application types:

- **OLTP**: For those with heavy loads, this is a good candidate for high fixed memory settings.
- **Data warehouse**: Use default values.
- **OLAP**: Use default value.
- **Mixed**: For those with heavy loads, this is a good candidate for high fixed memory settings.

**Max Text Repl Size**

- **Type—Basic, dynamic**
- **Default value—65536**

This parameter specifies the maximum size of the text and image datatypes for columns that are participating in replication during single insert, update, writetext, and updatetext statements. You might need to raise this value if the image sizes with which your application deals are consistently large and the data is part of a replication configuration.

```
sp_configure 'max text repl size', 131072
GO
RECONFIGURE
GO
```
Max Worker Threads

Type—Basic

Default value—255

SQL Server uses native operating system threads. This parameter specifies the maximum number of threads available for SQL Server processes. One or more threads are used for supporting each network protocol (such as TCP/IP and named pipes). SQL Server is configured to listen. The checkpoint and lazywriter processes also consume threads. A pool of threads is used to handle user connections. When the number of connections is lower than the max worker thread parameter value, a thread is created for each connection. When more connections are on the server than the value defined by the max worker thread parameter, SQL Server provides thread pooling for efficient resource utilization.

More threads can create overhead on the system processors. Therefore, lowering this value might sometimes improve the performance of the system. For a system with a few hundred user connections, a reasonable value for this parameter is 125. You might want to experiment with various values to determine the appropriate setting for this parameter. An SMP environment can easily handle more threads, and you can increase the number of threads accordingly.

sp_configure 'max worker threads', 125
go
RECONFIGURE
go

Figure 40.8 shows the setting of the Max Worker Threads option from the Processor tab of the SQL Server Properties dialog box. Choose a value between 10–1,024 for this option.

Setting of this option will vary according to the following application types:

- OLTP: For SMP environments, set the value upward because those environments can handle servicing more threads. This will yield performance gains.
- Data warehouse: Use default value.
- OLAP: Use default value.
- Mixed: For SMP environments, set the value upward because those environments can handle servicing more threads. This will yield performance gains.
**FIGURE 40.8** SQL Server Properties: Processor settings.

**Min Memory Per Query**
Type—Advanced
Default value—1024KB

Min memory per query specifies the minimum amount of memory that will be allocated for the execution of a query. Normally, the SQL Server query processor will attempt to determine the optimal amount of memory for a query. This option allows the sysadmin to specify this value instead. Increasing this value usually improves queries that handle hash and sort operations on a large volume of data. This option replaces the Sort Page option in SQL Server 7.0 and earlier.

```
sp_configure 'min memory per query', 2048
GO
RECONFIGURE
GO
```

Looking back at Figure 40.7, the Memory tab of the SQL Server Configuration dialog box, you can see the Minimum query memory value option set at 1024 (the default).

Setting of this option will vary according to the following application types:

- OLTP: Use default value.
- Data warehouse: This is a good opportunity to better service numerous canned queries in this environment. Set the value higher than the default.
Nested Triggers

As the name suggests, nested triggers specifies whether a trigger event on a table will fire another trigger. The nesting level of triggers is 32, and it used to be 16. If you reach this limit of 32, SQL Server will give an error and roll back the transaction. The default value of 1 means that a trigger on a table can cause another trigger to fire.

Take a look again at Figure 40.5; you can see the Nested Trigger option checked under the Server Behavior box.

Network Packet Size

This parameter specifies the default network packet size for SQL Server. Setting this value to a higher number (which should be divisible by 512) can improve the performance of applications that involve a large amount of data transfer from the server. Check your network configuration and set an appropriate value for this parameter. In this same regard, you can improve performance by lowering the size value for applications that are small in data transfer size. However, the usual scenario is to increase this size to accommodate large amounts of data transfer, as with Bulk Loads.

```
sp_configure 'network packet size', 8192
GO
RECONFIGURE
GO
```

TIP

You can also specify the network packet size from the client when you connect to SQL Server (using the -a option for isql, osql, and bcp). Setting the network packet size from a client can be useful when the default network packet size is adequate for general application needs. However, a larger packet size might be needed for some specific operations, such as bulk copy. You can also call OLE DB, ODBC, and DB-Library functions to change the packet size.
Setting of this option will vary according to the following application types:

- **OLTP:** Possibly decrease this size to 512 if all queries deal with small amounts of data transfer, which is often the case in OLTP or ATM applications.
- **Data warehouse:** Perhaps increase this to 8192 to handle the consistently large data transfers in this environment.
- **OLAP:** Use default value.
- **Mixed:** Use default value.

### Open Objects

**Type—Advanced, Self-configuring**

**Default value—0**

This option is self-configuring by default (when 0 is the setting). SQL Server will increase or decrease the number of open object descriptors in memory based on the needs of the server. You will rarely have to change from the default.

```
sp_configure 'open objects', 10000
go
RECONFIGURE
```

### Priority Boost

**Type—Advanced**

**Default value—0**

This option is used to specify the process priority of SQL Server processes on the Windows NT or Windows 2000 operating system. The default value of 0 means that SQL Server should run on the same priority level—a priority base of 7—as other applications on the machine. Priority boost can be turned on if you have plenty of horsepower to deal with all other services on the box, as in an SMP environment. When turning on priority boost, the priority base of SQL Server is elevated to 13.

```
sp_configure 'priority boost', 1
go
RECONFIGURE
```
NOTE

Don’t set the value of this parameter to 1, except in the case of a dedicated SQL Server with SMP hardware.

From Figure 40.8, you can see the Boost SQL Server Priority on Windows option in the Processor control function. Use care when applying this option.

**Query Governor Cost Limit**

Type—Advanced
Default value—0

Queries are often the cause of major performance problems. SQL Server can handle the queries, but many are poorly written and don’t restrict the search criteria enough. This can result in runaway queries that are returning large result sets and can adversely affect the entire server’s performance. A method to control this is to cut the query off at the pass by specifying a maximum cost limit to queries. If any query’s cost is greater than this maximum value, the query is not allowed to execute. This value is server-wide and cannot be applied to just one query.

```
sp_configure 'query governor cost limit', 300
GO
RECONFIGURE
GO
```

Query governor cost limit can be set by going to Server Behavior options, SQL Server Configuration dialog box, Server Settings tab.

Setting of this option will vary according to the following application types:

- **OLTP**: Use default value.
- **Data warehouse**: This is a must-have option for this environment. Try setting this value to 300 seconds, and then get ready for the users to scream at you. On the positive side, the server won’t get bogged down or freeze again.
- **OLAP**: For OLAP that use SQL Server storage, set this value to 600 seconds to get started, and then reduce it over time.
- **Mixed**: Same protection opportunity here. This won’t affect the OLTP queries, so it is safe to apply.
Query Wait
  Type—Advanced
  Default value—1

Queries that are memory intensive and involve huge sorts might take a long time to execute based on the available memory during execution. SQL Server internally calculates the timeout interval for such queries. Usually, this is quite a large number. You can override this value by specifying a value (in seconds) using the query wait parameter of SQL Server. If you set this value too low, you risk more frequent query timeouts when your system is under a heavy load and a highly concurrent environment.

```
sp_configure 'query wait', 20
GO
RECONFIGURE
GO
```

Recovery Interval
  Type—Advanced, Self-configuring
  Default value—0

Recovery interval is used to specify the maximum time (in minutes) that SQL Server would require to recover a database during startup. During startup, SQL Server rolls forward all the changes that were committed during a SQL Server crash and rolls back the changes that were not committed. Based on the value specified by this parameter, SQL Server determines when to issue a checkpoint in every database of SQL Server so that in the event of a crash, SQL Server can recover the databases in a time specified by recovery interval. If the value of the recovery interval parameter is low, SQL Server will issue checkpoints more frequently to allow a recovery to be faster; however, frequent checkpoints can slow down the performance. Setting recovery interval too high will create a longer recovery time for databases in the event of a crash. The default value of 0 leaves this option open to SQL Server to determine the best value.

```
sp_configure 'recovery interval', 10
GO
RECONFIGURE
GO
```

Figure 40.6 shows the Recovery Interval option setting of 0 within the Recovery section. Values must be between 1 and 32,767.
Setting of this option will vary according to the following application types:

- **OLTP**: Use default value.
- **Data warehouse**: This is an opportunity to save on checkpoints and not degrade performance in this mostly read-only environment. Set this value high.
- **OLAP**: Same performance opportunity here in this read-only environment.
- **Mixed**: Use default value.

### Remote Proc Trans

**Type**—Basic  
**Default value**—$0$

Remote proc trans allows remote procedures that are taking part in multiserver transactions to use MS-DTC so that transaction integrity is maintained across servers. The default value of $0$ means the remote procedure calls will not use MS-DTC. Data modification at the remote server will not be a part of transactions at the local server. If you set this parameter to $1$, SQL Server uses MS-DTC to preserve transaction integrity across servers.

```
sp_configure 'remote proc trans', 1  
go
RECONFIGURE  
go
```

Figure 40.9 illustrates the Enforce Distributed Transactions (MTS) option being set in the Remote Server Connections section of the Connections tab. This will primarily be important in OLTP environments.

Setting of this option will vary according to the following application types:

- **OLTP**: If you are having to support distributed transactions, this option should be set to On.
- **Data warehouse**: Use default value.
- **OLAP**: Use default value.
- **Mixed**: If you are having to support distributed transactions, this option should be set to On.
Scan for Startup Procs

Type—Advanced

Default value—0

When this option is set to 1, SQL Server will scan for and execute all automatically executed stored procedures on the server on startup. To set a stored procedure to become automatically executed, you use the `sp_procoption` system-stored procedure. Executing a stored procedure at startup time is typically done when you want to have certain processing occur that creates the proper working environment for all subsequent database processing on the server. Executing at startup can also be done when you want to make sure that certain stored procedures’ execution plans (with proper optimizer decisions) are already in procedure cache before anyone else has requested their execution.

```sql
sp_configure 'scan for startup procs', 1
GO
RECONFIGURE
GO
```
Set Working Set Size

Type—Advanced
Default value—0

When set working set size is set to 1, SQL Server allocates and locks the requested server fixed memory amount at startup. Min server memory and max server memory options must contain the same value, reflecting a fixed memory size request. This effectively guarantees you the memory for as long as SQL Server is up and running and doesn’t ever get pages swapped out when SQL Server is idle. This option has no effect when the memory option is set to 0; SQL Server is configured for dynamic memory allocation.

```
sp_configure 'set working set size', 1
GO
RECONFIGURE
GO
```

Looking at Figure 40.7, you can request that SQL server use a fixed memory size and also reserve physical memory for SQL Server. Set working set size is activated when you check this Reserve Physical Memory for SQL Server box. You cannot use this option when you are allowing SQL Server to dynamically configure memory.

Setting of this option will vary according to the following application types:

- **OLTP**: For those with heavy loads, this is a good candidate for high fixed memory settings.
- **Data Warehouse**: Use default values.
- **OLAP**: Use default value.
- **Mixed**: For those with heavy loads, this is a good candidate for high fixed memory settings.

Show Advanced Options

Type—Advanced, dynamic
Default value—0

By default, you will not see the advanced configuration parameters of SQL Server. By setting show advanced options to 1, you will be able to see all the SQL Server parameters that can be set by the `sp_configure` command.
User Connections
Type—Advanced, Self-configuring
Default value—0

User connections specifies the number of concurrent users that are allowed on SQL Server. When the value is 0, SQL Server can configure the needed user connections dynamically as they are needed. If you specify a value, you will be limited to this maximum number of user connections until you specify a larger value. If you specify other than a 0 value, the memory allocation for user connections will be allocated at SQL Server startup time and burn up portions of the memory pool. Each connection takes up 40KB of memory space. If you configure SQL Server for 100 connections, SQL Server will pre-allocate 4MB (40KB×100) for user connections. You can see that setting this value too high might eventually impact performance because the extra memory could have been used to cache data. In general, user connections are now best left to be self-configuring.

sp_configure 'user connections', 200
GO
RECONFIGURE
GO

Looking back at the SQL Server Properties dialog box and the Connections tab in Figure 40.9, you can set a value for the User Connections option by entering a value in the Maximum Concurrent User Connections box. This value must be between 5 and 32,767.

User Options
Type—Basic, static
Default value—0

User options allows you to specify certain defaults for all the options allowed with the SET T-SQL command. Individual users can override these values by using the SET command. You are essentially able to establish these options for all users unless the users override them for their own needs. User options is a bit-mask field, and each bit represents a user option. Table 40.1 outlines the values that you can set with this parameter.
### TABLE 40.1 Specifying User Options

<table>
<thead>
<tr>
<th>Bit Mask Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>DISABLE_DEF_CNST_CHK controls interim/deferred constraint checking.</td>
</tr>
<tr>
<td>2</td>
<td>IMPLICIT_TRANSACTIONS controls whether a transaction is started implicitly when a statement is executed.</td>
</tr>
<tr>
<td>4</td>
<td>CURSOR_CLOSE_ON_COMMIT controls the behavior of cursors after a commit has been performed.</td>
</tr>
<tr>
<td>8</td>
<td>ANSI_WARNINGS controls truncation and null in aggregate warnings.</td>
</tr>
<tr>
<td>16</td>
<td>ANSI_PADDING controls padding of fixed-length variables.</td>
</tr>
<tr>
<td>32</td>
<td>ANSI_NULLS controls null handling when using equality operators.</td>
</tr>
<tr>
<td>64</td>
<td>ARITHABORT terminates a query when an overflow or divide-by-zero error occurs during query execution.</td>
</tr>
<tr>
<td>128</td>
<td>ARITHIGNORE returns NULL when an overflow or divide-by-zero error occurs during a query.</td>
</tr>
<tr>
<td>256</td>
<td>QUOTED_IDENTIFIER differentiates between single and double quotation marks when evaluating an expression.</td>
</tr>
<tr>
<td>512</td>
<td>NOCOUNT turns off the message returned at the end of each statement that states how many rows were affected by the statement.</td>
</tr>
<tr>
<td>1024</td>
<td>ANSI_NULL_DFLT_ON alters the session’s behavior to use ANSI compatibility for nullability. New columns that are defined without explicit nullability are defined to allow NULLs.</td>
</tr>
<tr>
<td>2048</td>
<td>ANSI_NULL_DFLT_OFF alters the session’s behavior to not use ANSI compatibility for nullability. New columns defined without explicit nullability are defined not to allow NULLs.</td>
</tr>
<tr>
<td>4096</td>
<td>CONCAT_NULL_YIELDS_NULL will have SQL Server return a NULL when concatenating a NULL value with a string.</td>
</tr>
<tr>
<td>8192</td>
<td>NUMERIC_ROUNDABORT will have SQL Server generate an error if loss of precision ever occurs in an expression.</td>
</tr>
<tr>
<td>16384</td>
<td>XACT_ABORT will have SQL Server roll back a transaction if a Transact-SQL statement raises a runtime error.</td>
</tr>
</tbody>
</table>

For a given user connection, you can use the @@options global variable to see the values that have been set.

```sql
sp_configure 'user options', 256
GO
RECONFIGURE
GO
```

As you can see from the SQL Server Configuration dialog box and the Connections tab in Figure 40.9, the Default connection options can be checked or unchecked according to what you want to be in effect server-wide for all user connections. Again, a user can override these with his own SET command during a session.
Summary

Dealing with the large number of configurable options in SQL Server is a big undertaking. Not only do you need to know about the internal address space of SQL Server, but you also need to understand what type of applications will be running on the server so that the configuration decision that you make is not counter productive.

Many of the configurable options have a direct effect on the most dynamic part of SQL Server: the memory pool. This is truly where all the action is. Whether you have chosen to let SQL Server help you manage this space dynamically via self-configuring options, or you have decided to manage this yourself, you must constantly monitor the current settings and be prepared to modify them at any time. In general, you will be able to start with the default values given to the server at installation time and then slowly enhance these options over time.

In the next chapter, a detailed explanation of using XML in SQL Server 2000 will be provided that should allow you to start using this feature almost immediately.