Performance and Tuning: Monitoring and Analyzing

Adaptive Server® Enterprise

12.5.1

DOCUMENT ID: DC20022-01-1251-01

LAST REVISED: August 2003

Sybase, Inc., One Sybase Drive, Dublin, CA 94568.

Contents

About This Book

Audience

This manual is intended for database administrators, database designers, developers and system administrators.

Note You may want to use your own database for testing changes and queries. Take a snapshot of the database in question and set it up on a test machine.

How to use this book

Chapter 1, “Introduction to Performance and Tuning” gives a general description of this manual and the other manuals within the Performance and Tuning Series for Adaptive Server.

Chapter 2, “Monitoring Tables” Adaptive Server includes a set of system tables that contains monitoring and diagnostic information. This chapter describes how to query Adaptive Server’s monitoring tables for statistical and diagnostic information.

Chapter 3, “Using Statistics to Improve Performance” describes how to use the update statistics command to create and update statistics.

Chapter 4, “Using the set statistics Commands” explains the commands that provide information about execution.

Chapter 5, “Using set showplan” provides examples of showplan messages.

Chapter 6, “Statistics Tables and Displaying Statistics with optdiag” describes the tables that store statistics and the output of the optdiag command that displays the statistics used by the query optimizer.

Chapter 7, “Tuning with dbcc traceon” explains how to use the dbcc traceon commands to analyze query optimization problems.

Chapter 8, “Monitoring Performance with sp_sysmon” describes how to use a system procedure that monitors Adaptive Server performance

Related documents

- The remaining manuals for the Performance and Tuning Series are:
  - *Performance and Tuning: Basics*
  - *Performance and Tuning: Locking*

    - *Performance and Tuning: Optimizer and Abstract Plans*

- The release bulletin for your platform – contains last-minute information that was too late to be included in the books.

  A more recent version of the release bulletin may be available on the World Wide Web. To check for critical product or document information that was added after the release of the product CD, use the Sybase Technical Library.

- The *Installation Guide* for your platform – describes installation, upgrade, and configuration procedures for all Adaptive Server and related Sybase products.

- *Configuring Adaptive Server Enterprise* for your platform – provides instructions for performing specific configuration tasks for Adaptive Server.

- *What's New in Adaptive Server Enterprise?* – describes the new features in Adaptive Server version 12.5, the system changes added to support those features, and the changes that may affect your existing applications.

- *Transact-SQL User's Guide* – documents Transact-SQL, Sybase's enhanced version of the relational database language. This manual serves as a textbook for beginning users of the database management system. This manual also contains descriptions of the pubs2 and pubs3 sample databases.

- *System Administration Guide* – provides in-depth information about administering servers and databases. This manual includes instructions and guidelines for managing physical resources, security, user and system databases, and specifying character conversion, international language, and sort order settings.

- *Reference Manual* – contains detailed information about all Transact-SQL commands, functions, procedures, and data types. This manual also contains a list of the Transact-SQL reserved words and definitions of system tables.

- The *Utility Guide* – documents the Adaptive Server utility programs, such as isql and bcp, which are executed at the operating system level.

- The *Quick Reference Guide* – provides a comprehensive listing of the names and syntax for commands, functions, system procedures, extended system procedures, data types, and utilities in a pocket-sized book. Available only in print version.

- The *System Tables Diagram* – illustrates system tables and their entity relationships in a poster format. Available only in print version.
- *Error Messages and Troubleshooting Guide* – explains how to resolve frequently occurring error messages and describes solutions to system problems frequently encountered by users.
- *Component Integration Services User's Guide* – explains how to use the Adaptive Server Component Integration Services feature to connect remote Sybase and non-Sybase databases.
- *Java in Adaptive Server Enterprise* – describes how to install and use Java classes as data types, functions, and stored procedures in the Adaptive Server database.
- *XML Services in Adaptive Server Enterprise* – describes the Sybase native XML processor and the Sybase Java-based XML support, introduces XML in the database, and documents the query and mapping functions that comprise XML Services.
- *Using Sybase Failover in a High Availability System* – provides instructions for using Sybase's Failover to configure an Adaptive Server as a companion server in a high availability system.
- *Job Scheduler User's Guide* – provides instructions on how to create and schedule jobs on a local or remote Adaptive Server using the command line or a graphical user interface (GUI).
- *Using Adaptive Server Distributed Transaction Management Features* – explains how to configure, use, and troubleshoot Adaptive Server DTM features in distributed transaction processing environments.
- *EJB Server User's Guide* – explains how to use EJB Server to deploy and execute Enterprise JavaBeans in Adaptive Server.
- *XA Interface Integration Guide for CICS, Encina, and TUXEDO* – provides instructions for using Sybase's DTM XA interface with X/Open XA transaction managers.
- *Glossary* – defines technical terms used in the Adaptive Server documentation.
- *Sybase jConnect for JDBC Programmer's Reference* – describes the jConnect for JDBC product and explains how to use it to access data stored in relational database management systems.

- *Full-Text Search Specialty Data Store User's Guide* – describes how to use the Full-Text Search feature with Verity to search Adaptive Server Enterprise data.
- *Historical Server User's Guide* –describes how to use Historical Server to obtain performance information for SQL Server and Adaptive Server.
- *Monitor Server User's Guide* – describes how to use Monitor Server to obtain performance statistics from SQL Server and Adaptive Server.
- *Monitor Client Library Programmer's Guide* – describes how to write Monitor Client Library applications that access Adaptive Server performance data.

Other sources of information

Use the Sybase Technical Library CD and the Technical Library Product Manuals Web site to learn more about your product:

- The Getting Started CD contains release bulletins and installation guides in PDF format, and may also contain other documents or updated information not included on the Technical Library CD. It is included with your software. To read or print documents on the Getting Started CD you need Adobe Acrobat Reader (downloadable at no charge from the Adobe Web site, using a link provided on the CD).
- The Technical Library CD contains product manuals and is included with your software. The DynaText reader (included on the Technical Library CD) allows you to access technical information about your product in an easy-to-use format.

  Refer to the *Technical Library Installation Guide* in your documentation package for instructions on installing and starting the Technical Library.
- The Technical Library Product Manuals Web site is an HTML version of the Technical Library CD that you can access using a standard Web browser. In addition to product manuals, you will find links to EBFs/Updates, Technical Documents, Case Management, Solved Cases, newsgroups, and the Sybase Developer Network.

  To access the Technical Library Product Manuals Web site, go to Product Manuals at http://www.sybase.com/support/manuals/.

Sybase certifications on the Web

Technical documentation at the Sybase Web site is updated frequently.

❖ **Finding the latest information on product certifications**

1 Point your Web browser to Technical Documents at http://www.sybase.com/support/techdocs/.

2 Select Products from the navigation bar on the left.

3 Select a product name from the product list and click Go.

4 Select the Certification Report filter, specify a time frame, and click Go.

5 Click a Certification Report title to display the report.

❖ **Creating a personalized view of the Sybase Web site (including support pages)**

Set up a MySybase profile. MySybase is a free service that allows you to create a personalized view of Sybase Web pages.

1 Point your Web browser to Technical Documents at http://www.sybase.com/support/techdocs/.

2 Click MySybase and create a MySybase profile.

Sybase EBFs and software updates

❖ **Finding the latest information on EBFs and software updates**

1 Point your Web browser to the Sybase Support Page at http://www.sybase.com/support.

2 Select EBFs/Updates. Enter user name and password information, if prompted (for existing Web accounts) or create a new account (a free service).

3 Select a product.

4 Specify a time frame and click Go.

5 Click the Info icon to display the EBF/Update report, or click the product description to download the software.

Conventions

This section describes conventions used in this manual.

Formatting SQL statements

SQL is a free-form language. There are no rules about the number of words you can put on a line or where you must break a line. However, for readability, all examples and syntax statements in this manual are formatted so that each clause of a statement begins on a new line. Clauses that have more than one part extend to additional lines, which are indented.

Font and syntax conventions

The font and syntax conventions used in this manual are shown in Table 1.0:

Table 1: Font and syntax conventions in this manual

Element	Example
Command names, command option names, utility names, utility flags, and other keywords are bold.	select sp_configure

Element	Example
Database names, datatypes, file names and path names are in *italics*.	master database

Variables, or words that stand for values that you fill in, are in *italics*.	`select`
	column_name
	`from`
	table_name
	`where`
	search_conditions

Parentheses are to be typed as part of the command.	`compute row_aggregate ( column_name )`
Curly braces indicate that you must choose at least one of the enclosed options. Do not type the braces.	`{cash, check, credit}`
Brackets mean choosing one or more of the enclosed options is optional. Do not type the brackets.	`[anchovies]`
The vertical bar means you may select only one of the options shown.	`{die_on_your_feet | live_on_your_knees | live_on_your_feet}`
The comma means you may choose as many of the options shown as you like, separating your choices with commas to be typed as part of the command.	`[extra_cheese, avocados, sour_cream]`
An ellipsis (...) means that you can *repeat* the last unit as many times as you like.	`buy thing = price [cash | check | credit] [, thing = price [cash | check | credit]]...` You must buy at least one thing and give its price. You may choose a method of payment: one of the items enclosed in square brackets. You may also choose to buy additional things: as many of them as you like. For each thing you buy, give its name, its price, and (optionally) a method of payment.

- Syntax statements (displaying the syntax and all options for a command) appear as follows:

```
sp_dropdevice [ device_name]
```

or, for a command with more options:

```
select column_name
    from table_name
    where search_conditions
```

In syntax statements, keywords (commands) are in normal font and identifiers are in lowercase: normal font for keywords, italics for user-supplied words.

- Examples of output from the computer appear as follows:

```
0736 New Age Books Boston MA
0877 Binnet & Hardley Washington DC
1389 Algodata Infosystems Berkeley CA
```

Case

In this manual, most of the examples are in lowercase. However, you can disregard case when typing Transact-SQL keywords. For example, SELECT, Select, and select are the same. Note that Adaptive Server's sensitivity to the case of database objects, such as table names, depends on the sort order installed on Adaptive Server. You can change case sensitivity for single-byte character sets by reconfiguring the Adaptive Server sort order.

See in the *System Administration Guide* for more information.

Expressions

Adaptive Server syntax statements use the following types of expressions:

Table 2: Types of expressions used in syntax statements

Usage	Definition
expression	Can include constants, literals, functions, column identifiers, variables, or parameters
logical expression	An expression that returns TRUE, FALSE, or UNKNOWN
constant expression	An expression that always returns the same value, such as "5+3" or "ABCDE"
float_expr	Any floating-point expression or expression that implicitly converts to a floating value
integer_expr	Any integer expression, or an expression that implicitly converts to an integer value
numeric_expr	Any numeric expression that returns a single value
char_expr	Any expression that returns a single character-type value
binary_expression	An expression that returns a single *binary* or *varbinary* value

Examples

Many of the examples in this manual are based on a database called pubtune. The database schema is the same as the pubs2 database, but the tables used in the examples have more rows: titles has 5000, authors has 5000, and titleauthor has 6250. Different indexes are generated to show different features for many examples, and these indexes are described in the text.

The pubtune database is not provided with Adaptive Server. Since most of the examples show the results of commands such as set showplan and set statistics io, running the queries in this manual on pubs2 tables will not produce the same I/O results, and in many cases, will not produce the same query plans as those shown here.

If you need help

Each Sybase installation that has purchased a support contract has one or more designated people who are authorized to contact Sybase Technical Support. If you cannot resolve a problem using the manuals or online help, please have the designated person contact Sybase Technical Support or the Sybase subsidiary in your area.

CHAPTER 1

Introduction to Performance and Tuning

Tuning Adaptive Server Enterprise for performance can involve several processes in analyzing the “Why?” of slow performance, contention, optimizing and usage.

Adaptive Server employs reports for monitoring the server. This manual explains how statistics are obtained and used for monitoring and optimizing. The stored procedure sp_sysmon produces a large report that shows the performance in Adaptive Server.

You can also use the Sybase Monitor in Sybase Central for realtime information on the status of the server.

The remaining manuals for the Performance and Tuning Series are:

- *Performance and Tuning: Basics*

  This manual covers the basics for understanding and investigating performance questions in Adaptive Server. It guides you in how to look for the places that may be impeding performance.

- *Performance and Tuning: Optimizer*

  The Optimizer in the Adaptive Server takes a query and finds the best way to execute it. The optimization is done based on the statistics for a database or table. The optimized plan stays in effect until the statistics are updated or the query changes. You can update the statistics on the entire table or by sampling on a percentage of the data.

- *Performance and Tuning: Abstract Plans*

  Adaptive Server can generate an abstract plan for a query, and save the text and its associated abstract plan in the sysqueryplans system table. Abstract plans provide an alternative to options that must be specified in the batch or query in order to influence optimizer decisions. Using abstract plans, you can influence the optimization of a SQL statement without having to modify the statement syntax.

- *Performance and Tuning: Locking*

Adaptive Server locks the tables, data pages, or data rows currently used by active transactions by locking them. Locking is a concurrency control mechanism: it ensures the consistency of data within and across transactions. Locking is needed in a multiuser environment, since several users may be working with the same data at the same time.

Carefully considered indexes, built on top of a good database design, are the foundation of a high-performance Adaptive Server installation. However, adding indexes without proper analysis can reduce the overall performance of your system. Insert, update, and delete operations can take longer when a large number of indexes need to be updated.

Each of the manuals has been set up to cover specific information that may be used by the system administrator and the database administrator.

CHAPTER 2

Monitoring Tables

This chapter describes how to query Adaptive Server's monitoring tables for statistical and diagnostic information.

Monitoring system tables in Adaptive Server

Adaptive Server includes a set of system tables that contains monitoring and diagnostic information. The information in these tables provides you with a statistical snapshot of the state of Adaptive Server, which allows you to analyze the server for performance improvements. You can query these system tables in much the same way you currently query any other tables in Adaptive Server. For example, to display statistical information about I/O on Sybase devices:

```
select * from monDeviceIO
Reads    APFReads     Writes   DevSemaphoreRequests  DevSemaphoreWaits IOTime
LogicalName PhysicalName
-------- -----------  -------- --------------------  ----------------- ------
---------------------------   -------------------------------
1563    7           7891    3                     0             134900
master                        /dev/vx/rdsk/sybase/master_vol01
59       0             15       2                         0                800
engcomdb_data_vol01           /dev/vx/rdsk/sybase/engcomdb_data_vol01
5       0           13      2                     0             100
```

```
engcomdb_log_vol01              /dev/vx/rdsk/sybase/engcomdb_log_vol01
126255  59657       8604    2                   0             1408700
qts_db_data_vol01               /dev/vx/rdsk/sybase/qts_db_data_vol01
31      0           9879    2                   0             128400
qts_db_log_vol01                /dev/vx/rdsk/sybase/qts_db_log_vol01
51      0           19      2                   0             400
sadb_data_vol01                 /dev/vx/rdsk/sybase/sadb_data_vol01
5       0           12      2                   0             200
sadb_log_vol01                  /dev/vx/rdsk/sybase/sadb_log_vol01
56       0             25       2                      0                 900
scratchdb_vol01                 /dev/vx/rdsk/sybase/scratchdb_vol01
0       0           0       2                   0              0
rmdb_data_vol01                  /dev/vx/rdsk/sybase/rmdb_data_vol01
0        0           0       2                    0               0
rmdb_log_vol01                  /dev/vx/rdsk/sybase/rmdb_log_vol01
52658    424         99512   2                    0               2231300
sysprocsdev                     /dev/vx/rdsk/sybase/sybsystemprocs_vol01
146      0           3569    2                    0               13700
tempdb_data                     /tmp/tempdb_data
4        0           814     2                    0               400
tempdb_log                           /tmp/tempdb_log
```

Where monDeviceIO is the system table that contains statistical information about disk I/O (physical name, reads, writes, and so on). To perform this query, the user does not need to know if Monitor Server or any other monitoring agent is executing on the same host server; the monDeviceIO table contains all the information they need.

The monitoring tables are not created by default, but must be created using the *installmontables* script. See “Installing the monitoring tables” on page 12 for more information.

Note You must have the mon_role role to query these tables. For more information, see “The mon_role role” on page 5.

Using Transact-SQL to monitor performance

Providing monitoring information as tables enables you to use Transact-SQL to monitor Adaptive Server. For example, to identify the Transact-SQL statements that are currently consuming CPU, enter:

```
select s.SPID, s.CpuTime, t.LineNumber, t.SQLText
from monProcessStatement s, monProcessSQLText t
```

```
where s.SPID=t.SPID
order by s.CpuTime, s.SPID, t.LineNumber desc
```

You can this same query to find the SQL statements that are using the most physical I/O by substituting CpuTime for PhysicalReads.

The information in each monitoring table can be sorted, selected, joined, inserted into another table, and treated much the same as the information in a regular Adaptive Server table.

The tables are read-only because they are in-memory tables that are generated as they are queried.

Access to these tables is restricted to users with the mon_role role.

The definitions for the monitoring tables have a similar syntax to CIS definitions, which allow remote procedures to be defined as local tables. For example, the following syntax is used to create the monNetworkIO table on a server named "loopback":

```
create existing table monNetworkIO (
    PacketsSent int,
    PacketsReceived int,
    BytesSent int,
    BytesReceived int,
)
external procedure
at "loopback...$monNetworkIO"
```

The *mon_role* role

Only users with the mon_role role can access Adaptive Server's monitoring tables. You can provide extra role-based security by modifying the CIS proxy table definitions provided with the monitoring tables. For information about acquiring roles, see Chapter 11, "Managing User Permissions," in the *System Administration Guide*.

Some of the tables may contain sensitive information. For example, monSysSQLText contains all the SQL text that was sent to the Adaptive Server. This may contain information such as updates to employee salary records.

Examples of querying the monitoring tables

This section provides examples of querying the monitoring tables.

- This query determines what monitoring tables are available:

```
select *
from master..monTables
```

- This query determines which parameters will improve performance by including them in a where clause:

```
select * from master..monTableParameters
where TableName="monOpenObjectActivity"
```

See “Processing information through search arguments” on page 8 for more information.

- This query determines what columns exist in a specific monitoring table:

```
select ColumnName, TypeName, Length, Description
from master..monTableColumns
where TableName=”monProcessSQLText
```

You can determine the columns for any of the monitoring tables by substituting its name in the where clause and running the query.

- This example determines which queries are consuming the most CPU:

```
select s.SPID, s.CpuTime, t.LineNumber, t.SQLText
from master..monProcessStatement s, master..monProcessSQLText t
where s.SPID = t.SPID
order by s.CpuTime DESC
```

This query also provides the hit ratio over the life of Adaptive Server, and must be rewritten to apply to a specific time period.

- This query determines the hit ratios for the data cache for the life of Adaptive Server:

```
select "Procedure Cache Hit Ratio" = (Requests-Loads)*100/Requests
from master..monProcedureCache
```

This query also provides the hit ratio over the life of Adaptive Server, and must be rewritten to apply to a specific time period.

Because the values for LogicalReads and CacheSearches are accumulated over time, you must rewrite this query for a specific sampling period (for example, use the changes of values over a 10-minute period). For example, the following queries the monitoring tables for the sampling period:

```
select * into #moncache_prev
```

```
from master..monDataCache
waitfor delay "00:10:00"
select * into #moncache_cur
from master..monDataCache
select p.CacheName,
"Hit Ratio"=(c.LogicalReads-p.LogicalReads)*100 / (c.CacheSearches -
p.CacheSearches)
from #moncache_prev p, #moncache_cur c
where p.CacheName = c.CacheName
```

- This query creates a stored procedure that prints the executed SQL and the backtrace of any stored procedures for diagnostic purposes:

  ```
  create procedure sp_backtrace @spid int as
  begin
  select SQLText
  from master..monProcessSQLText
  where SPID=@spid
  print "Stacktrace:"
  select ContextID, DBName, OwnerName, ObjectName
  from master..monProcessProcedures
  where SPID=@spid
  end
  ```

- Identifies any indices that are not currently in use and can be dropped:

```
select DBID, ObjectID, LastUsedDate, UsedCount
from monOpenObjectActivity
where dbid=5 and ObjectID=1424005073 and IndexID > 1
```

To determine if an index can be dropped:

- All queries that access the table in question have been run. Typically, you can determine this by ensuring that Adaptive Server has been running long enough so that all applications have performed all of their selects on the table.
- Ensure that the object has remained open. That is, the table and its indexes have not been scavenged. You can determine this by looking at the Reused column from the output of sp_monitorconfig for `number of open indexes` and `number of open objects`. For example:

```
exec sp_monitorconfig 'number of open indexes'
exec sp_monitorconfig 'number of open objects'
Usage information at date and time: Oct 22 2002 1:49PM.
Name                    Num_free  Num_active Pct_act
Max_Used Reused
------------------------- ----------- ----------- -------
----     ------
```

```
number of open indexes    496         4           0.80
4        No
Usage information at date and time: Oct 22 2002 1:49PM.
Name                    Num_free  Num_active Pct_act
Max_Used Reused
----------------------- ----------- ----------- -------
----     ------
number of open objects    494         6           1.20
6          No
```

Processing information through search arguments

You must correctly use search arguments when specifying parameters to monitoring tables, or the efficiency of your query deteriorates. If the search condition for your query is not precisely defined on one or more parameters of the monitoring tables, information is collected for the entire result set, which must then be filtered by the Adaptive Server language layer. The following example correctly specifies the search arguments, `where DBID = 1`:

```
select * from monOpenObjectActivity
where DBID = 1
```

Adaptive Server uses certain search arguments (identified in monTableParameters) to filter the result set and reduce the amount of work it performs to produce the result set. These parameters are used only when you specify an exactly matching search argument; for example, when a = 2.

But if the search conditions of the query are more loosely specified, for example if DBID < 2 the DBID parameter cannot be used internally, and a result set containing a row for every object in every database is produced. Also, Adaptive Server's language layer must filter the result set to simply return a row for each object in the master database. This adversely affects your performance.

Query the monTableParameters to determine which arguments should be specified to improve query performance. For example, the following query shows which search arguments should be specified for the monOpenObjectActivity table:

```
select ParameterName, TypeName
from monTableParameters
where TableName = 'monOpenObjectActivity'
ParameterName      TypeName
```

```
-----------------  ------------
DBID               int
ObjectID           int
IndexID              int
```

Wrapping counter datatypes

Some columns in the monitoring tables contain integer counter values which are incremented throughout the life of Adaptive Server. Once a counter reaches the highest value possible (2,147,483,647), it is reset to 0, which is called "wrapping."

Because of wrapping, you should sample these counters over time and use the result of sampling instead of the returned value. For example, use the difference between the current value and the previous value instead of the return value.

The Indicators column of the monTableColumns table specifies which columns are prone to wrapping.

To display a list of columns that are counters, execute:

```
select TableName, ColumnName   from monTableColumns
where (Indicators & 1) = 1
```

Stateful historical monitoring tables

A number of monitoring tables provide the most recent historical monitoring information rather than information about the current state. Adaptive Server maintains context for each client that accesses these tables and only returns information that the client has not previously returned. These "stateful" historical monitoring tables were designed to maximize performance and to avoid duplicate rows when used to populate a repository for historical data.

The following stateful monitoring tables provide data that provides the most recent historical data rather than information about the current state of Adaptive Server:

- monErrorLog
- monDeadLock

- monSysStatement
- monSysSQLText
- monSysPlanText

You can identify stateful historical tables from their monTables.Indicators column:

```
select TableName from monTables where Indicators & 1=1
```

The information returned from stateful historical tables is stored in buffers, one for each historical monitoring table. Use the sp_configure options to configure the size of the buffer and the information to be captured. Which sp_configure options you use depends on which monitoring table you are interested in configuring. For example, for the monSysPlanText table, you configure:

- plan text pipe max messages – configures the number of messages to be stored for the particular buffer.
- plan text pipe active – indicates whether Adaptive Server writes information to the buffer.

Each stored message stored contributes one row to the monitoring table. New messages overwrite old messages in the buffers, so only the most recent messages are returned.

See Chapter 5 of the *System Administration Guide*, "Setting Configuration Parameters" and "Adaptive Server configuration options" on page 13 of this chapter for more information about using sp_configure.

Because Adaptive Server retains a context for every client connection and returns only the data that was added since the previous read, you may get seemingly inconsistent result sets from queries that attempt to filter results using a where clause, because:

- A select from the monitoring table always returns all previously unread messages.
- The filtering is performed by the Adaptive Server language layer.

In the following example, the buffer associated with the monErrorLog table contains two messages:

```
select SPID, ErrorMessage from monErrorLog
SPID      ErrorMessage
------    -----------------------------------
20        An error from SPID 20
21        An error from SPID 21
```

```
(2 rows affected)
```

If you reconnect, the two messages are returned, but you receive the following messages when you filter the result set with a where clause:

```
select SPID, ErrorMessage from monErrorLog
where SPID=20
SPID         ErrorMessage
------      -------------------------------------
20           An error from SPID 20
(1 row affected)
```

And:

```
select SPID, ErrorMessage from monErrorLog
where SPID=21
SPID         ErrorMessage
------      -------------------------------------
(0 rows affected)
```

Even though you never see the row for SPID 21, a result set containing the SPID was passed to Adaptive Server's language layer, which filtered the row before returning the result set to the client, and the message is marked as "read" for this connection.

Note Because of the stateful nature of these tables, you should not use these tables for ad-hoc queries. Instead, you should use a `select * into` or `insert into` to populate a repository.

Transient monitoring data

Because monitoring tables contain stateful data, take care when joining or using aggregates in your queries because data may not be available if the plan requires that the table be queried multiple times. For example:

```
select s.SPID, s.CpuTime, s.LineNumber, t.SQLText
from monProcessStatement s, monProcessSQLText t
where s.SPID=t.SPID
and s.CpuTime = (select max(CpuTime) from monProcessStatement)
```

Here, the monProcessStatement table is queried twice; first to find the maximum CpuTime, and then to match the maximum. When Adaptive Server performs the second query, there are three potential outcomes returned from monProcessStatement:

- The statement performes more work, consuming more CPU, and having a CpuTime value greater than the previous maximum. This returns no results.
- The statement finishes executing. This yields no results unless another statement used exactly the amount of CPU as the previously obtained maximum.
- The statement does not use any additional CPU, and its value of CpuTime still matches the maximum. This is the only scenario that will produce the expected results.

When you are designing queries, keep in mind that, because the data contained in them is transient, joins and aggregates may not return the expected results if the plan requires that the table is queried multiple times.

Installing the monitoring tables

Proxy tables for the monitoring tables are not created by default when you configure Adaptive Server. You must create them using the *installmontables* script located in the *$SYBASE/ASE-12_5/scripts* directory (*%SYBASE%\ASE-12_5\scripts* for NT). This script requires that a server named "loopback" be included in sysservers. To include this server, enter:

```
declare @servernetname varchar(30)
select @servernetname=srvnetname
from sysservers
where srvname=@@servername
exec sp_addserver loopback, NULL, @servernetname
```

Install the *installmontables* script the same way you install the *installmaster* script. For example:

```
isql -Usa -Ppassword -Sserver_name -i $SYBASE/ASE-12_5/scripts/installmontables
```

Adaptive Server configuration options

By default, Adaptive Server does not collect the monitoring information required by the monitoring tables. You must use sp_configure to configure Adaptive Server to start collecting the monitoring information. The monitoring configuration parameters required are displayed when you enter:

```
sp_configure Monitoring
```

The following sections describe the configuration parameters that you must configure before using monitoring tables.

deadlock pipe active

Summary information	
Default value	0
Range of values	0–1
Status	Dynamic
Display level	Comprehensive
Required role	System Administrator

deadlock pipe active controls whether Adaptive Server collects deadlock messages. If both deadlock pipe active and deadlock pipe max messages are enabled, Adaptive Server collects the text for each deadlock. You can retrieve these deadlock messages using monDeadLock.

deadlock pipe max messages

Summary information	
Default value	0
Range of values	0–2147483647
Status	Dynamic
Display level	Comprehensive
Required role	System Administrator

deadlock pipe max messages determines the number of deadlock messages Adaptive Server stores. Adaptive Server allocates memory for storing as many deadlock messages as indicated by the value of deadlock pipe max messages.

enable monitoring

Summary information	
Default value	0
Range of values	0–1
Status	Dynamic
Display level	Comprehensive
Required role	System Administrator

enable monitoring controls whether Adaptive Server collects the monitoring table data. Data is not collected if enable monitoring is set to 0. enable monitoring acts as a master switch that determines whether any of the following configuration parameters are enabled.

errorlog pipe active

Summary information	
Default value	0
Range of values	0–1
Status	Dynamic
Display level	Comprehensive
Required role	System Administrator

errorlog pipe active controls whether Adaptive Server collects error log messages. If both errorlog pipe active and errorlog pipe max messages are enabled, Adaptive Server collects all the messages sent to the error log. You can retrieve these error log messages using monErrorLog.

errorlog pipe max messages

Summary information	
Default value	0
Range of values	0–2147483647
Status	Dynamic
Display level	Comprehensive
Required role	System Administrator

errorlog pipe max messages determines the number of error log messages Adaptive Server stores. Adaptive Server allocates memory for storing as many error messages as indicated by the value of errorlog pipe max messages.

max SQL text monitored

Summary information	
Default value	0
Range of values	0–2147483647
Status	Dynamic
Display level	Comprehensive
Required role	System Administrator

max SQL text monitored specifies the total number of bytes Adaptive Server allocates for each user task to store SQL text.

object lockwait timing

Summary information	
Default value	0
Range of values	0–1
Status	Dynamic
Display level	Comprehensive
Required role	System Administrator

object lockwait timing controls whether Adaptive Server collects timing statistics for requests of locks on objects.

per object statistics active

Summary information	
Default value	0
Range of values	0–1
Status	Dynamic
Display level	Comprehensive

Summary information	
Required role	System Administrator

per object statistic active controls whether Adaptive Server collects statistics for each object.

plan text pipe active

Summary information	
Default value	0
Range of values	0–1
Status	Dynamic
Display level	Comprehensive
Required role	System Administrator

plan text pipe active determines whether Adaptive Server collects query plan text. If both plan text pipe active and plan text pipe max messages are enabled, Adaptive Server collects the plan text for each query. You can use monSysPlanText to retrieve the query plan text for all user tasks.

plan text pipe max messages

Summary information	
Default value	0
Range of values	0–2147483647
Status	Dynamic
Display level	Comprehensive
Required role	System Administrator

plan text pipe max messages determines the number of query plan text messages Adaptive Server stores. Adaptive Server allocates memory for storing as many messages as indicated by the value of this configuration option.

process wait events

Summary information	
Default value	0
Range of values	0–1
Status	Dynamic
Display level	Comprehensive
Required role	System Administrator

process wait events controls whether Adaptive Server collect statistics for each wait event for every task. You can get wait information for a specific task using monProcessWaits.

sql text pipe active

Summary information	
Default value	0
Range of values	0–1
Status	Dynamic
Display level	Comprehensive
Required role	System Administrator

sql text pipe active controls whether Adaptive Server collects SQL text. If this option is enabled and sql text pipe max messages is set, Adaptive Server collects the SQL text for each query. You can use monSysSQLText to retrieve the SQL text for all user tasks.

sql text pipe max messages

Summary information	
Default value	0
Range of values	0–2147483647
Status	Dynamic
Display level	Comprehensive
Required role	System Administrator

sql text pipe max messages specifies the number of SQL text messages Adaptive Server stores. Adaptive Server allocates memory for storing as many messages as indicated by the value of sql text pipe max messages.

statement pipe active

Summary information	
Default value	0
Range of values	0–1
Status	Dynamic
Display level	Comprehensive
Required role	System Administrator

statement pipe active controls whether Adaptive Server collects statement-level statistics. If both statement pipe active and statement pipe max messages are enabled, Adaptive Server collects the statement statistics for each query. You can retrieve the statistics for all executed statements using monSysStatement.

statement pipe max messages

Summary information	
Default value	0
Range of values	0–2147483647
Status	Dynamic
Display level	Comprehensive
Required role	System Administrator

statement pipe max messages determines the number of statement statistics messages Adaptive Server stores. Adaptive Server allocates memory for storing as many messages as indicated by the value statement pipe max messages.

statement statistics active

Summary information	
Default value	0
Range of values	0–1
Status	Dynamic
Display level	Comprehensive
Required role	System Administrator

statement statistic active controls whether Adaptive Server collects the monitoring tables statement-level statistics. You can use monProcessStatement to get statement statistics for a specific task.

SQL batch capture

Summary information	
Default value	0
Range of values	0–1
Status	Dynamic
Display level	Comprehensive
Required role	System Administrator

SQL batch capture controls whether Adaptive Server collects SQL text. If both SQL batch capture and max SQL text monitored are enabled, Adaptive Server collects the SQL text for each batch for each user task.

wait event timing

Summary information	
Default value	0
Range of values	0–1
Status	Dynamic
Display level	Comprehensive
Required role	System Administrator

wait event timing controls whether Adaptive Server collects statistics for individual wait events. A task may have to wait for a variety of reasons (for example, waiting for a buffer read to complete). The monSysWaits table contains the statistics for each wait event. The monWaitEventInfo table contains a complete list of wait events.

System tables for monitor tables

This section lists the tables that are included with the monitoring tables feature.

monTables

Description

Provides a description of all monitoring tables. No configuration options are required for monTables.

Columns

Name	Datatype	Attributes	Description
TableID	int		Unique identifier for the table
Columns	tinyint		Total number of columns in the table
Parameters	tinyint		Total number of optional parameters that can be specified
Indicators	int		Indicators for specific table properties. For example, if the table is a stateful monitoring table then (Indicators & 1) =1
Size	int		Maximum row size (in bytes)
TableName	varchar(30)	null	Table name
Description	varchar(368)	null	Description of the view

monTableParameters

Description

Provides a description of all the optional parameters for each monitoring table. No configuration parameters are required for monTableParameters.

Columns

Name	Datatype	Attributes	Description
TableID	int		Unique identifier for the view
ParameterID	int		Position of the parameter
TypeID	int		Identifier of the datatype of the parameter
Precision	tiny_int		Precision of the parameter, if numeric
Scale	tiny_int		Scale of the parameter, if numeric
Length	small_int		Maximum length of the parameter (in bytes)
TableName	varchar(30)	null	Name of the table
ParameterName	varchar(30)	null	Name of the parameter
TypeName	varchar(20)	null	Name of the datatype of the parameter
Description	varchar(255)	null	Description of the parameter

monTableColumns

Description

Describes all the columns for each monitoring table. No configuration options are required for monTableColumns.

Columns

Name	Datatype	Attributes	Description
TableID	int		Unique identifier for the view
ColumnID	int		Position of the column
TypeID	int		Identifier for the datatype of the column
Precision	tinyint		Precision of the column, if numeric
Scale	tinyint		Scale of the column, if numeric
Length	smallint		Maximum length of the column (in bytes)
Indicators	int		Indicators for specific column properties (for example, if the column is prone to wrapping and should be sampled)
TableName	varchar(30)	null	Name of the table
ColumnName	varchar(30)	null	Name of the column

Name	Datatype	Attributes	Description
TypeName	varchar(21)	null	Name of the datatype of the column
Description	varchar(255)	null	Description of the column

monState

Description

Provides information regarding the overall state of Adaptive Server. No configuration options are necessary for monState.

Columns

Name	Datatype	Attributes	Description
LockWaitThreshod	int		Time (in seconds) that processes must have waited for locks in order to be reported
LockWaits	int		Number of processes that have waited longer than LockWaitThreshold seconds
StartDate	datetime		Date and time that Adaptive Server was started
DaysRunning	int		Number of days Adaptive Server has been running
CountersCleared	datetime		Date and time the monitor counters were last cleared
CheckPoints	int		Reports whether any checkpoint is currently running
NumDeadlocks	int	counter	Total number of deadlocks that have occurred
Diagnostic Dumps	int		Reports whether sybmon is performing a shared memory dump
Connections	int		Number of active inbound connections
Max Recovery	int		The maximum time (in minutes), per database, that Adaptive Server uses to complete its recovery procedures in case of a system failure; also, the current Run Value for recovery interval in minutes

monEngine

Description

Provides statistics regarding Adaptive Server engines. Requires the enable monitoring configuration parameter to be enabled.

Columns

Name	Datatype	Attributes	Description
Engine Number	smallint		Number of the Adaptive Server engine
Starttime	datetime		Date that the engine came online
StopTime	datetime		Date that the engine went offline
CurrentKPID	int		Kernel process identifier for the currently executing process.
PreviousKPID	int		Kernel process identifier for the previously executing process
CPUTime	int	counter, reset	Total time (in seconds) the engine has been running
SystemCPUTime	int	counter, reset	Time (in seconds) the engine has been executing system database services
UserCPUTime	int	counter, reset	Time (in seconds) the engine has been executing user commands
IdleCPUTime	int	counter, reset	Time (in seconds) the engine has been in idle spin mode
ContextSwitches	int	counter, reset	Number of context switches
Connections	int	counter	Number of connections handled
ProcessesAffinitied	int		Number of processes that have been affinitied to this engine
Status	varchar(30)	null	Status of the engine (online, offline, and so on)
AffinitiedToCPU	int	null	The number of the CPU that the engine is affinitied to

monDataCache

Description

Returns statistics relating to Adaptive Server data caches. Requires the enable monitoring configuration parameter to be enabled.

Columns

Name	Datatype	Attributes	Description
CacheID	int		Unique identifier for the cache
RelaxedReplacement	int		Whether the cache is using relaxed cache replacement strategy
BufferPools	int		The number of buffer pools within the cache
CacheSearches	int	counter	Cache searches directed to the cache
PhysicalReads	int	counter	Number of buffers read into the cache from disk
PhysicalWrites	int	counter	Number of buffers written from the cache to disk
LogicalReads	int	counter	Number of buffers retrieved from the cache
Stalls	int	counter	Number of “dirty” buffer retrievals
CachePartitions	smallint		Number of partitions currently configured for the cache
CacheName	varchar(30)	null	Name of cache

monProcedureCache

Description

Returns statistics relating to Adaptive Server procedure cache. Requires the enable monitoring configuration parameter to be enabled.

Columns

Name	Datatype	Attributes	Description
Requests	int	counter, reset	Number of stored procedures requested
Loads	int	counter, reset	Number of stored procedures loaded into cache
Writes	int	counter, reset	Number of times a procedure was normalized and the tree written back to sysprocedures

Name	Datatype	Attributes	Description
Stalls	int	counter, reset	Number of times a process had to wait for a free procedure cache buffer when installing a stored procedure into cache

monOpenDatabases

Description

Provides state and statistical information pertaining to databases that are currently in use. Requires the enable monitoring configuration parameter to be enabled.

Columns

Name	Datatype	Attributes	Description
DBID	int		Unique identifier for the database
BackupStartTime	datetime		Date that the last backup started for the database
BackupInProgress	int		Whether a backup is currently in progress for the database
LastBackupFailed	int		Whether the last backup of the database failed
TransactionLogFull	int		Whether the database transaction log is full
AppendLogRequests	int	counter	Number of semaphore requests when attempting to append to the database transaction log
AppendLogWaits	int	counter	Number of times a task had to wait for the append log semaphore to be granted
DatabaseName	varchar(30)	null	Name of the database

monSysWorkerThread

Description

Returns server-wide statistics related to worker threads. Requires the enable monitoring configuration parameter to be enabled.

Columns

Name	Datatype	Attributes	Description
ThreadsActive	int		Number of worker processes active
TotalWorkerThreads	int		Configured maximum number of worker processes
HighWater	int	reset	The maximum number of worker processes that have ever been in use
ParallelQueries	int	counter, reset	Number of parallel queries that were attempted
PlansAltered	int	counter, reset	Number of plans altered due to worker processes not being available
WorkerMemory	int		The amount of memory currently in use by worker processes
TotalWorkerMemory	int		The amount of memory configured for use by worker processes
WorkerMemoryHWM	int	reset	The maximum amount of memory ever used by worker processes
MaxParallelDegree	int		The maximum degree of parallelism that can be used (the current Run Value for max parallel degree) configuration parameter
MaxScanParallelDegree	int		The maximum degree of parallelism that can be for a scan (the current Run Value for max scan parallel degree configuration parameter

monNetworkIO

Description

Returns network I/O statistics. Requires the enable monitoring configuration parameter to be enabled.

Columns

Name	Datatype	Attributes	Description
PacketsSent	int	counter, reset	Number of packets sent
PacketsReceived	int	counter, reset	Number of packets received
BytesSent	int	counter, reset	Number of bytes sent
BytesReceived	int	counter, reset	Number of bytes received

monErrorLog

Description

Returns the most recent error messages from the Adaptive Server error og. The maximum number of messages returned can be tuned with the errorlog pipe max messages. Requires the enable monitoring, errorlog pipe max messages, and errorlog pipe active configuration parameters to be enabled.

Columns

Name	Datatype	Attributes	Description
SPID	smallint		Session process identifier
KPID	int		Kernel process identifier
FamilyID	smallint		SPID of the parent process
EngineNumber	int		Engine on which process was running
ErrorNumber	int		Error message number
Severity	int		Severity of error
Time	datetime		Timestamp when error occurred
ErrorMessage	varchar(512)	null	Text of the error message

monLocks

Description

Returns a list of all locks that are being held, and those that have been requested, by any process, for every object. Requires the enable monitoring and wait event timing configuration parameters to be enabled.

Columns

Name	Datatype	Attributes	Description
SPID	smallint		Session process identifier
KPID	int		Kernel process identifier
DBID	int		Unique identifier for the database
ParentSPID	smallint		Parent process ID
LockID	int		Lock object ID
Context	int		Lock context (bit field). These values are the same as for those of the of the context column in syslocks. See the *Reference Manual* for information about syslocks
ObjectID	int	null	Unique identifier for the object
LockState	varchar(20)	null	Whether the lock has been granted [Granted, Requested]
LockType	varchar(20)	null	Type of lock [‘exclusive table’, ‘shared page’, and so on]
LockLevel	varchar(30)	null	The type of object for which the lock was requested (‘PAGE’, ‘ROW’, and so on)
WaitTIme	int	null	The time (in seconds) that the lock request has not been granted.
PageNumber	int	null	Page that is locked when LockLevel = ‘PAGE’
RowNumber	int	null	Row that is locked when LockLevel = ‘ROW’

monDeadLock

Description

Provides information pertaining to the most recent deadlocks that have occurred in Adaptive Server. You can tune the maximum number of messages returned with deadlock pipe max messages. Requires the enable monitoring, deadlock pipe max messages, and deadlock pipe active configuration parameters to be enabled.

Columns

Name	Datatype	Attributes	Description
DeadLockID	int		Unique identifier for the deadlock

Name	Datatype	Attributes	Description
VictimKPID	int		KPID of the victim process for the deadlock
ResolveTime	datetime		Time at which the deadlock was resolved
ObjectDBID	int		Unique database identifier for database where the object resides
PageNumber	int		Page number for which the lock was requested, if applicable
RowNumber	int		Row number for which the lock was requested, if applicable
HeldFamilyId	smallint		SPID of the parent process of the process holding the lock
HeldSPID	smallint		SPID of process holding the lock
HeldKPID	int		KPID of process holding the lock
HeldProcDBID	int		Unique identifier for the database where the stored procedure that caused the lock to be held resides, if applicable
HeldProcedureID	int		Unique object identifier for the stored procedure that caused the lock to be held, if applicable
HeldBatchID	int		Unique batch identifier for the SQL code being executed by the process holding the lock when it was blocked by another process (not when it acquired the lock)
HeldContextID	int		Unique context identifier for the process holding the lock when it was blocked by another process (not when it acquired the lock)
HeldLineNumber	int		Line number within the batch of the statement being executed by the process holding the lock when it was blocked by another process (not when it acquired the lock)
WaitFamilyId	smallint		SPID of the parent process of the process waiting for the lock
WaitSPID	smallint		SPID of the process waiting for the lock
WaitKPID	int		KPID of the process waiting for the lock

Name	Datatype	Attributes	Description
WaitTime	int		Amount of time in milliseconds that the waiting process was blocked before the deadlock was resolved
ObjectName	varchar(30)	null	Name of the object
HeldUserName	varchar(30)	null	Name of the user for whom the lock is being held
HeldApplName	varchar(30)	null	Name of the application holding the lock
HeldTranName	varchar(255)	null	The name of the transaction in which the lock was acquired
HeldLockType	varchar(20)	null	The type of lock being held
HeldCommand	varchar(30)		The command being executed that caused the lock to be held
WaitUserName	varchar(30)	null	Name of the user for whom the lock is being requested
WaitLockType	varchar(20)	null	The type of lock requested

monWaitClassInfo

Description

Provides a textual description for all of the wait classes (for example, waiting for a disk read to complete). All wait events (see the description for monWaitEventInfo) have been grouped into wait classes that classify the type of event that a process is waiting for.

Columns

Name	Datatype	Attributes	Description
WaitClassID	smallint		Unique identifier for the wait event class
Description	varchar(50)	null	Description of the wait event class

monWaitEventInfo

Description

Provides a textual description for every possible situation where a process is forced to wait within Adaptive Server. For example, wait for buffer read to complete.

Columns

Name	Datatype	Attributes	Description
WaitEventID	smallint		Unique identifier for the wait event type
WaitClassID	smallint		Unique identifier for the wait event class
Description	varchar(50)	null	Description of the wait event type

monCachedObject

Description

Returns statistics for all objects and indexes with pages currently in a data cache.

Columns

Name	Datatype	Attributes	Description
CacheID	int		Unique identifier for the cache
ObjectID	int		Unique identifier for the object
IndexID	int		Unique identifier for the index
DBID	int		Unique identifier for the database
OwnerUserID	int		Unique identifier for the database owner
CachedKB	int		Number of kilobytes of the cache the object is occupying
ProcessesAccessing	int		Number of processes currently accessing the object
CacheName	varchar(30)	null	Name of the cache
DBName	varchar(30)	null	Name of the database
OwnerName	varchar(30)	null	Name of the object owner
ObjectName	varchar(30)	null	Name of the object
ObjectType	varchar(30)	null	Object type

monCachePool

Description

Provides statistics for all pools allocated for all caches. Requires the enable monitoring configuration parameter to be enabled.

Columns

Name	Datatype	Attributes	Description
CacheID	int		Unique identifier for the cache
IOBufferSize	int		Size (in bytes) of the I/O buffer for the pool
AllocatedKB	int		Number of bytes that have been allocated for the pool
PhysicalReads	int	counter	Number of buffers that have been read from disk into the pool
Stalls	int	counter	Number of dirty buffer retrievals
PagesTouched	int	counter	Number of bytes that are currently being used within the pool
PagesRead	int	counter	Number of pages read into the pool
BuffersToMRU	int	counter	The number of buffers that were fetched and replaced in the most recently used portion of the pool
BuffersToLRU	int	counter	The number of buffers that were fetched and replaced in the least recently used portion of the pool
CacheName	varchar(30)	null	Name of the cache

monOpenObjectActivity

Description

Provides statistics for all open objects. Requires the enable monitoring and per object statistics active configuration parameter are enabled.

Columns

Name	Datatype	Attributes	Description
DBID	int		Unique identifier for the database
ObjectID	int		Unique identifier for the object

Name	Datatype	Attributes	Description
IndexID	int		Unique identifier for the index
LogicalReads	int	counter	Total number of buffers read
PhysicalReads	int	counter	Number of buffers read from disk
APFReads	int	counter	Number of APF buffers read
PagesRead	int	counter	Total number of pages read
PhysicalWrites	int	counter	Total number of buffers written to disk
PagesWritten	int	counter	Total number of pages written to disk
RowsInserted	int	null, counter	Number of rows inserted
RowsDeleted	int	null, counter	Number of rows deleted
RowsUpdated	int	null, counter	Number of updates
Operations	int	null, counter	Number of times that the object was accessed
LockRequests	int	null, counter	Number of requests for a lock on the object
LockWaits	int	null, counter	Number of times a task waited for a lock for the object

monIOQueue

Description

Provides device I/O statistics broken down into data and log I/O for normal and temporary databases on each device. monIOQueue requires the enable monitoring configuration parameter to be enabled.

Columns

Name	Datatype	Attributes	Description
IOs	int	counter	Total number of I/O operations
IOTime	int	counter	Amount of time (in milliseconds) spent waiting for I/O requests to be satisfied
LogicalName	varchar(30)	null	Logical name of the device
IOType	varchar(12)	null	Category for grouping I/O (‘user data’, ‘User log’, ‘Tempdb Data’, or ‘Tempdb log’)

monDeviceIO

Description

Returns statistical information relating to devices. monDeviceIO requires the enable monitoring configuration parameter to be enabled.

Columns

Name	Datatype	Attributes	Description
Reads	int	counter, reset	Number of reads from the device (excluding APF)
APFReads	int	counter, reset	Number of APF reads from the device
Writes	int	counter, reset	Number of writes to the device
DevSemaphoreRequests	int	counter, reset	Number of I/O requests
DevSemaphore Waits	int	counter, reset	Number of tasks forced to wait for synchronization of an I/O request
IOTime	int)	counter	Total amount of time (in milliseconds) spent waiting for I/O requests to be satisfied
LogicalName	varchar(30)	null	Logical name of the device
PhysicalName	varchar(128)	null	Full hierarchic file name of the device

monSysWaits

Description

Provides a server-wide view of where processes are waiting for an event. monSysWaits requires the enable monitoring and wait event timing configuration parameters are enabled.

Columns

Name	Datatype	Attributes	Description
WaitEventID	smallint		Unique identifier for the wait event
WaitTime	int	counter	Amount of time (in milliseconds) that tasks have spent waiting for the event
Waits	int		Number of times tasks have waited for the event

monProcess

Description

Provides detailed statistics about processes that are currently executing or waiting. monProcess requires the enable monitoring and wait event timing configuration parameters to be enabled.

Columns

Name	Datatype	Attributes	Description
SPID	smallint		Session process identifier
KPID	int		Kernel process identifier
FamilyID	smallint		The SPID of the parent process, if it exists
BatchID	int		Unique identifier for the SQL batch containing the statement being executed
ContextID	int		The stack frame of the procedure, if it exists
LineNumber	int		Line number of the current statement within the SQL batch
SecondsConnected	int		Number of seconds since this connection was established
WaitEventID	smallint		Unique identifier for the event that the process is waiting for, if the process is currently in a wait state
BlockingSPID	smallint		Session process identifier of the process holding the lock that this process has requested, if waiting for a lock
DBID	int		Unique identifier for the database being used by the current process
EngineNumber	int		Unique identifier of the engine on which the process is executing
Priority	int		Priority at which the process is executing
Login	varchar(30)	null	Login user name
Application	varchar(30)	null	Application name

Name	Datatype	Attributes	Description
Command	varchar(30)	null	Category of process or command that the process is currently executing
NumChildren	int	null	Number of child processes, if executing a parallel query
SecondsWaiting	int	null	Amount of time in seconds that the process has been waiting, if the process is currently in a wait state
BlockingXLOID	int	null	Unique lock identifier for the lock that this process has requested, if waiting for a lock
DBName	varchar(30)	null	Name of process for the database being used by the current process
EngineGroupName	varchar(30)	null	Engine group for the process
ExecutionClass	varchar(30)	null	Execution class for the process
MasterTransactionID	varchar(255)	null	Unique transaction identifier for the current transaction, if in a transaction

monProcessLookup

Description

Provides information enabling processes to be tracked to an application, user, client machine, and so on.

Columns

Name	Datatype	Attributes	Description
SPID	smallint		Session process identifier
KPID	int		Kernel process identifier
Login	varchar(30)	null	Login user name
Application	varchar(30)	null	Application name
ClientHost	varchar(30)	null	Host name of client
ClientIP	varchar(24)	null	IP address of client

Name	Datatype	Attributes	Description
ClientOSPID	varchar(30)	null	OS process identifier of the client application

monProcessActivity

Description

Provides detailed statistics about process activity. monProcessActivity requires the enable monitoring, wait event timing, and per object statistics active configuration parameters to be enabled.

Columns

Name	Datatype	Attributes	Description
SPID	smallint		Session process identifier
KPID	int		Kernel process identifier
CPUTime	int	counter	CPU time (in milliseconds) used by the process
WaitTime	int	counter	Time (in milliseconds) the process has spent waiting
PhysicalReads	int	counter	Number of buffers read from disk
LogicalReads	int	counter	Number of buffers read from cache
PagesRead	int	counter	Number of pages read
PhysicalWrites	int	counter	Number of buffers written to disk
PagesWritten	int	counter	Number of pages written
MemUsageKB	int		Amount of memory (in bytes) allocated to the process
LocksHeld	int		Number of locks process currently holds
TableAccesses	int	counter	Number of pages where data was retrieved without an index
IndexAccesses	int)	counter	Number of pages where data was retrieved using an index
TempDbAccess	int	counter	Number of temporary tables accessed
ULCBytesWritten	int	counter	Number of bytes written to the user log cache for the process
ULCFlushes	int	counter	Total number of times the user log cache was flushed

Name	Datatype	Attributes	Description
ULCFlushFull	int	counter	Number of times the user log cache was flushed because it was full
ULCMaxUsage	int		The maximum ever usage (in bytes) of the user log cache by the process
ULCCurrentUsage	int		The current usage (in bytes) of the Uuer log cache by the process.
Transactions	int	counter	Number of transactions started by the process
Commits	int	counter	Number of transactions committed by the process
Rollbacks	int	counter	Number of transactions rolled back by the process

monProcessNetIO

Description

Provides the network I/O activity information for each process. monProcessNetIO requires the enable monitoring configuration parameters to be enabled.

Columns

Name	Datatype	Attributes	Description
SPID	smallint		Session process identifier
KPID	int		Kernel process identifier
NetworkPacketSize	int		Network packet size the session is currently using.
PacketSent	int	counter	Number of packets sent
PacketsReceived	int	counter	Number of packets received
BytesSent	int	counter	Number of bytes sent
BytesRecieved	int	counter	Number of bytes received

monProcessObject

Description

Provides statistical information regarding objects that have been accessed by processes. monProcessObject requires the enable monitoring and per object statistics active configuration parameters to be enabled.

Columns

Name	Datatype	Attributes	Description
SPID	smallint		Session process identifier
KPID	int		Kernel process identifier
DBID	int		Unique identifier for the database where the object resides
ObjectID	int		Unique identifier for the object
IndexID	int		Unique identifier for the index
OwnerUserID	int		User identifier for the object owner
LogicalReads	int	counter	Number of buffers read from cache
PhysicalReads	int	counter	Number of buffers read from disk
PhysicalAPFReads	int	counter	Number of APF buffers read from disk
DBName	varchar(30)	null	Name of database
ObjectName	varchar(30)	null	Name of the object
ObjectType	varchar(30)	null	Type of object

monProcessWaits

Description

Provides a server-wide view of where processes are waiting for an event. monProcessWaits requires the enable monitoring and process wait events configuration parameters to be enabled.

Columns

Name	Datatype	Attribute	Description
SPID	smallint		Session process identifier
KPID	int		Kernel process identifier
WaitEventID	smallint		Unique identifier for the wait event

Name	Datatype	Attribute	Description
Waits	int	counter	Number of times the process has waited for the event
WaitTime	int	counter	Amount of time (in milliseconds) that the process has waited for the event

monProcessStatement

Description

Provides information for currently executing statements. monProcessStatement requires the enable monitoring, statement statistics active, and per object statistics active configuration parameters to be enabled.

Columns

Name	Datatype	Attributes	Description
SPID	smallint		Session process identifier
KPID	int		Kernel process identifier
DBID	int		Unique identifier for the database
ProcedureID	int		Unique identifier for the procedure
PlanID	int		Unique identifier for the stored plan for the procedure
BatchID	int		Unique identifier for the SQL batch containing the statement
ContextID	int		Stack frame of the procedure, if a procedure
LineNumber	int		Line number of the statement within the SQL batch
StartTime	datetime		Date when the statement began execution
CPUTime	int	counter	Number of milliseconds of CPU used by the statement
WaitTime	int	counter	Number of milliseconds the task has waited during execution of the statement
MemUsageKB	int		Number of kilobytes of memory used for execution of the statement
PhysicalReads	int	counter	Number of buffers read from disk

Name	Datatype	Attributes	Description
LogicalReads	int	counter	Number of buffers read from cache
PagesModified	int	counter	Number of pages modified by the statement
PacketSent	int	counter	Number of network packets sent by Adaptive Server
PacketsReceived	int	counter	Number of network packets received by Adaptive Server
NetworkPacketSize	int		Size (in bytes) of the network packet currently configured for the session
PlansAltered	int	counter	The number of plans altered at execution time

monProcessSQLText

Description

Provides the SQL text that is currently being executed. The maximum size of the SQL text is tuned by max SQL text monitored. monProcessSQLText requires the enable monitoring, max SQL text monitored, and SQL batch capture configuration parameters to be enabled.

Columns

Name	Datatype	Attributes	Description
SPID	smallint		Session process identifier.
KPID	int		Kernel process identifier.
BatchID	int		Unique identifier for the SQL batch containing the SQL text.
LineNumber	int		Line number in SQL batch.
SequenceInLine	int		If the entered line of SQL text exceeds the size of the SQL text column, the text is split over the multiple rows. Each row has a unique, and increasing, SequenceInLine value.
SQLText	varchar(255)	null	SQL text.

monSysPlanText

Description

Provides the most recent generated text plan. Specify the maximum number of messages returned with plan text pipe max messages. monSysPlanText requires the enable monitoring, plan text pipe max messages, and plan text pipe active configuration parameters to be enabled.

Columns

Name	Datatype	Attributes	Description
PlanID	int		Unique identifier for the plan
SPID	smallint		Session process identifier
KPID	int		Kernel process identifier
BatchID	int		Unique identifier for the SQL batch for which the plan was created
ContextID	int		The stack frame of the procedure, if a procedure
SequenceNumber	int		A monotonically increasing number indicating the position of the PlanText column within the entire plan text
DBID	int		Unique identifier for the database where the procedure is stored, if the plan is for a stored procedure
ProcedureID	int		Unique identifier for the procedure, if the plan is for a stored procedure
PlanText	varchar(160)	null	Plan text output

monSysStatement

Description

Provides statistics pertaining to the most recently executed statements. The maximum number of statement statistics returned can be tuned with statement pipe max messages. monSysStatement requires the enable monitoring, statement statistics active, per object statistics active, statement pipe max messages, and statement pipe active configuration parameters to be enabled.

Columns

Name	Datatype	Attributes	Description
SPID	smallint		Session process identifier
KPID	int		Kernel process identifier
DBID	int		Unique identifier for the database
ProcedureID	int		Unique identifier for the procedure
PlanID	int		Unique identifier for the stored plan for the procedure
BatchID	int		Unique identifier for the SQL batch containing the statement
ContextID	int		Current procedure nesting level when executing the statement
LineNumber	int		Line number of the statement within the SQL batch
StartTime	datetime		Date when the statement began execution
EndTime	datetime		Date when the statement finished execution
CPUTime	int	counter	Number of milliseconds of CPU used by the statement.
WaitTime	int	counter	Number of milliseconds the task has waited during execution of the statement
MemUsageKB	int		Number of kilobytes of memory used for execution of the statement
PhysicalReads	int	counter	Number of buffers read from disk
LogicalReads	int	counter	Number of buffers read from cache
PagesModified	int	counter	Number of pages modified by the statement
PacketSent	int	counter	Number of network packets sent by Adaptive Server
PacketsReceived	int	counter	Number of network packets received by Adaptive Server
NetworkPacketSize	int		Size (in bytes) of the network packet currently configured for the session
PlansAltered	int	counter	The number of plans altered at execution time

monCachedProcedures

Description

Provides statistics for all procedures currently stored in the procedure cache. monCachedProcedures does not require any configuration parameters.

Columns

Name	Datatype	Attributes	Description
ObjectID	int		Unique identifier for the procedure
OwnerUID	int		Unique identifier for the database owner
DBID	int		Unique identifier for the database
PlanID	int		Unique identifier for the query plan
MemUsageKB	int		Number of kilobytes of memory used by the procedure
CompileDate	datetime		Date that the procedure was compiled
ObjectName	varchar(30)	null	Name of the procedure
ObjectType	varchar(32)	null	The type of procedure (stored procedure, trigger, and so on)
OwnerName	varchar(30)	null	Number of the object owner
DBName	varchar(30)	null	Name of the database

monSysSQLText

Description

Provides the most recent SQL text that has been executed, or is currently being executed. The maximum number of rows returned can be tuned with sql text pipe max messages. monProcessSQLText requires the enable monitoring, max SQL text monitored, SQL batch capture, sql text pipe max messages, and statement pipe active configuration parameters to be enabled.

Columns

Name	Datatype	Attributes	Description
SPID	smallint		Session process identifier
KPID	int		Kernel process identifier
BatchID	int		Unique identifier for the SQL batch containing the SQL text

Name	Datatype	Attributes	Description
SequenceInBatch	int		Indicates the position of this portion of the SQL text within a batch
SQLText	varchar(255)		SQL text

monProcessProcedures

Description

Returns a list of all procedures that are being executed by processes. monProcessProcedures does not require any configuration options.

Columns

Name	Datatype	Attributes	Description
SPID	smallint		Session process identifier
KPID	int		Kernel process identifier
DBID	int		Unique identifier for the database
OwnerUID	int		Unique identifier for the object owner
ObjectID	int		Unique identifier for the procedure
PlanID	int		Unique identifier for the query plan
MemUsageKB	int		Number of kilobytes of memory used by the procedure
CompileDate	datetime		Compile date of the procedure
ContextID	int		Stack frame of the procedure
DBName	varchar(30)	null	Name of the database that contains the procedure
OwnerName	varchar(30)	null	Name of the object owner
ObjectName	varchar(30)	null	Name of the procedure
ObjectType	varchar(32)	null	Type of procedure (stored procedure, trigger procedure, and so on)

CHAPTER 3

Using Statistics to Improve Performance

Accurate statistics are essential to the query optimization. In some cases, adding statistics for columns that are not leading index keys also improves query performance. This chapter explains how and when to use the commands that manage statistics.

Importance of statistics

Adaptive Server's cost-based optimizer uses statistics about the tables, indexes, and columns named in a query to estimate query costs. It chooses the access method that the optimizer determines has the least cost. But this cost estimate cannot be accurate if statistics are not accurate.

Some statistics, such as the number of pages or rows in a table, are updated during query processing. Other statistics, such as the histograms on columns, are only updated when you run the update statistics command or when indexes are created.

If you are having problems with a query performing slowly, and seek help from Technical Support or a Sybase news group on the Internet, one of the first questions you are likely be asked is "Did you run update statistics?" You can use the optdiag command to see the time update statistics was last run for each column on which statistics exist:

```
Last update of column statistics: Aug 31 2001
4:14:17:180PM
```

Another command you may need for statistics maintenance is delete statistics. Dropping an index does not drop the statistics for that index. If the distribution of keys in the columns changes after the index is dropped, but the statistics are still used for some queries, the outdated statistics can affect query plans.

Updating

The update statistics command updates the column-related statistics such as histograms and densities. So statistics need to be updated on those columns where the distribution of keys in the index changes in ways that affect the use of indexes for your queries.

Running the update statistics command requires system resources. Like other maintenance tasks, it should be scheduled at times when load on the server is light. In particular, update statistics requires table scans or leaf-level scans of indexes, may increase I/O contention, may use the CPU to perform sorts, and uses the data and procedure caches. Use of these resources can adversely affect queries running on the server if you run update statistics at times when usage is high. In addition, some update statistics commands require shared locks, which can block updates. See “Scan types, sort requirements, and locking” on page 56 for more information.

Adding statistics for unindexed columns

When you create an index, a histogram is generated for the leading column in the index. Examples in earlier chapters have shown how statistics for other columns can increase the accuracy of optimizer statistics. For example, see “Using statistics on multiple search arguments” in the *Performance and Tuning: Optimizer*.

You should consider adding statistics for virtually all columns that are frequently used as search arguments, as long as your maintenance schedule allows time to keep these statistics up to date.

In particular, adding statistics for minor columns of composite indexes can greatly improve cost estimates when those columns are used in search arguments or joins along with the leading index key.

update statistics commands

The update statistics commands create statistics, if there are no statistics for a particular column, or replaces existing statistics if they already exist. The statistics are stored in the system tables systabstats and sysstatistics. The syntax is:

```
update statistics table_name
    [ [index_name] | [( column_list ) ] ]
    [using step values ]
    [with consumers = consumers ]

update index statistics table_name [index_name]
    [using step values ]
    [with consumers = consumers ]

update all statistics table_name
```

The effects of the commands and their parameters are:

- For update statistics:
  - *table_name* – Generates statistics for the leading column in each index on the table.
  - *table_name index_name* – Generates statistics for all columns of the index.
  - *table_name* (*column_name*) – Generates statistics for only this column.
  - *table_name* (*column_name*, *column_name*...) – Generates a histogram for the leading column in the set, and multi column density values for the prefix subsets.
  - *using step values* – Identifies the number of steps used. The default is 20 steps. If you need to change the default number of steps, use sp_configure.
- For update index statistics:
  - *table_name* – Generates statistics for all columns in all indexes on the table.
  - *table_name index_name* – Generates statistics for all columns in this index.
- For update all statistics:

- *table_name* – Generates statistics for all columns of a table.
- *using step values* – Identifies the number of steps used. The default is 20 steps. If you need to change the default number of steps, use sp_configure.

Using sampling for *update statistics*

The optimizer for Adaptive Server uses the statistics on a database to set up and optimize queries. The statistics must be as current as possible to generate optimal results.

Run the update statistics commands against data sets, such as tables, to update information about the distribution of key values in specified indexes or columns, for all columns in an index, or for all columns in a table. The commands revise histograms and density values for column-level statistics. The results are then used by the optimizer to calculate the best way to set up a query plan.

update statistics requires table scans or leaf-level scans of indexes, may increase I/O contention, may use the CPU to perform sorts, and uses data and procedure caches. Use of these resources can adversely affect queries running on the server if you run update statistics when usage is high. In addition, some update statistics commands require shared locks, which can block updates.

To reduce I/O contention and resources, run update statistics using a sampling method, which can reduce the I/O and time when your maintenance window is small and the data set is large. If you are updating a large data set or table that is in constant use, being truncated and repopulated, you may want to do a statistical sampling to reduce the time and the size of the I/O.

You must use caution with sampling since the results are not fully accurate. Balance changes to histogram values against the savings in I/O.

Although a sampling of the data set may not be completely accurate, usually the histograms and density values are reasonable within an acceptable range.

When you are deciding whether or not to use sampling, consider the size of the data set, the time constraints you are working with, and if the histogram produced is as accurate as needed.

The percentage to use when sampling depends on your needs. Test various percentages until you receive a result that reflects the most accurate information on a particular data set.

Statistics are stored in the system tables systabstats and sysstatistics.

To update statistics, use:

update statistics *table_name*
[[*index_name*] | [(*column_list*)]]
[using *step* values]
[with *consumers* = *consumers*] [, sampling = *percent*]

update index statistics *table_name* [*index_name*]
[using *step* values]
[with *consumers* = *consumers*] [, sampling = *percent*

update all statistics *table_name* [*index_name*] [using step values]
[with *consumers* = *consumers*] [, sampling = *percent*]

Where:

- *table_name* – generates statistics for the leading column in each index on the table.
- *table_name [index_name]* – generates statistics for all leading columns on the specified index.
- *table_name* (*column_list*) – generates statistics for only this column.
- *table_name* (*column_name*, *column_name*...) – generates a histogram for the first column in the list, and multi-column density values for the prefix subsets.
- *sampling* = percent – the numeric value of the sampling percentage, such as 05 for 5%, 10 for 10%, and so on. The sampling integer is between zero (0) and one hundred (100).

Example:

```
update statistics authors(auth_id) with sampling = 5 percent
```

The serverwide sampling percent can be set using the configuration paramater:

```
sp_configure 'sampling percent', 5
```

This command sets a serverwide sampling of 5% for update statistics that allows you to do the update statistics without the *sampling* syntax. The percentage can be anywhere from zero (0) to one hundred (100) percent.

Column statistics and statistics maintenance

Histograms are kept on a per-column basis, rather than on a per-index basis. This has certain implications for managing statistics:

- If a column appears in more than one index, update statistics, update index statistics or create index updates the histogram for the column and the density statistics for all prefix subsets.

  update all statistics updates histograms for all columns in a table.

- Dropping an index does not drop the statistics for the index, since the optimizer can use column-level statistics to estimate costs, even when no index exists.

  If you want to remove the statistics after dropping an index, you must explicitly delete them with delete statistics.

  If the statistics are useful to the optimizer and you want to keep the statistics without having an index, you need to use update statistics, specifying the column name, for indexes where the distribution of key values changes over time.

- Truncating a table does not delete the column-level statistics in sysstatistics. In many cases, tables are truncated and the same data is reloaded.

  Since truncate table does not delete the column-level statistics, there is no need to run update statistics after the table is reloaded, if the data is the same.

  If you reload the table with data that has a different distribution of key values, you need to run update statistics.

- You can drop and re-create indexes without affecting the index statistics, by specifying 0 for the number of steps in the with statistics clause to create index. This create index command does not affect the statistics in sysstatistics:

  ```
  create index title_id_ix on titles(title_id)
      with statistics using 0 values
  ```

  This allows you to re-create an index without overwriting statistics that have been edited with optdiag.

- If two users attempt to create an index on the same table, with the same columns, at the same time, one of the commands may fail due to an attempt to enter a duplicate key value in sysstatistics.

Creating and updating column statistics

Creating statistics on unindexed columns can improve the performance of many queries. The optimizer can use statistics on any column in a where or having clause to help estimate the number of rows from a table that match the complete set of query clauses on that table.

Adding statistics for the minor columns of indexes and for unindexed columns that are frequently used in search arguments can greatly improve the optimizer's estimates.

Maintaining a large number of indexes during data modification can be expensive. Every index for a table must be updated for each insert and delete to the table, and updates can affect one or more indexes.

Generating statistics for a column without creating an index gives the optimizer more information to use for estimating the number of pages to be read by a query, without entailing the processing expense of index updates during data modification.

The optimizer can apply statistics for any columns used in a search argument of a where or having clause and for any column named in a join clause. You need to determine whether the expense of creating and maintaining the statistics on these columns is worth the improvement in query optimization.

The following commands create and maintain statistics:

- update statistics, when used with the name of a column, generates statistics for that column without creating an index on it.

  The optimizer can use these column statistics to more precisely estimate the cost of queries that reference the column.

- update index statistics, when used with an index name, creates or updates statistics for all columns in an index.

  If used with a table name, it updates statistics for all indexed columns.

- update all statistics creates or updates statistics for all columns in a table.

Good candidates for column statistics are:

- Columns frequently used as search arguments in where and having clauses
- Columns included in a composite index, and which are not the leading columns in the index, but which can help estimate the number of data rows that need to be returned by a query.

See “How scan and filter selectivity can differ” on page 185 for information on how additional column statistics can be used in query optimization.

When additional statistics may be useful

To determine when additional statistics are useful, run queries using dbcc traceon(302) and statistics io. If there are significant discrepancies between the “rows to be returned” and I/O estimates displayed by dbcc traceon(302) and the actual I/O displayed by statistics io, examine these queries for places where additional statistics can improve the estimates. Look especially for the use of default density values for search arguments and join columns.

See “Tuning with dbcc traceon(302)” on page 171 for more information.

Adding statistics for a column with *update statistics*

This command adds statistics for the price column in the titles table:

```
update statistics titles (price)
```

This command specifies the number of histogram steps for a column:

```
update statistics titles (price)
    using 50 values
```

This command adds a histogram for the titles.pub_id column and generates density values for the prefix subsets pub_id; pub_id, pubdate; and pub_id, pubdate, title_id:

```
update statistics titles(pub_id, pubdate, title_id)
```

Note Running update statistics with a table name updates histograms and densities for leading columns for indexes only.

It does not update the statistics for unindexed columns.

To maintain these statistics, you must run update statistics and specify the column name, or run update all statistics.

Adding statistics for minor columns with *update index statistics*

To create or update statistics on all columns in an index, use update index statistics. The syntax is:

```
update index statistics table_name [index_name]
        [using step values]
         [with consumers = consumers ]
```

Adding statistics for all columns with *update all statistics*

To create or update statistics on all columns in a table, use update all statistics. The syntax is:

```
update all statistics table_name
```

Choosing step numbers for histograms

By default, each histogram has 20 steps which provides good performance and modeling for columns that have an even distribution of values. A higher number of steps can increase the accuracy of I/O estimates for:

- Columns with a large number of highly duplicated values
- Columns with unequal or skewed distribution of values
- Columns that are queried using leading wild cards in like queries

Note If your database was updated from a pre-11.9 version of the server, the number of steps defaults to the number of steps that were used on the distribution page.

Disadvantages of too many steps

Increasing the number of steps beyond what is needed for good query optimization can hurt Adaptive Server performance, largely due to the amount of space that is required to store and use the statistics. Increasing the number of steps:

- Increases the disk storage space required for sysstatistics

- Increases the cache space needed to read statistics during query optimization
- Requires more I/O, if the number of steps is very large

During query optimization, histograms use space borrowed from the procedure cache. This space is released as soon as the query is optimized.

Choosing a step number

See "Choosing the number of steps for highly duplicated values" on page 156 for more information.

For example, if your table has 5000 rows, and one value in the column that has only one matching row, you may need to request 5000 steps to get a histogram that includes a frequency cell for every distinct value. The actual number of steps is not 5000; it is either the number of distinct values plus one (for dense frequency cells) or twice the number of values plus one (for sparse frequency cells).

Scan types, sort requirements, and locking

Table 3-1 shows the types of scans performed during update statistics, the types of locks acquired, and when sorts are needed.

Table 3-1: Scans, sorts, and locking during update statistics

update statistics specifying	Scans and sorts performed	Locking
Table name		
Allpages-locked table	Table scan, plus a leaf-level scan of each nonclustered index	Level 1; shared intent table lock, shared lock on current page
Data-only-locked table	Table scan, plus a leaf-level scan of each nonclustered index and the clustered index, if one exists	Level 0; dirty reads
Table name and clustered index name		
Allpages-locked table	Table scan	Level 1; shared intent table lock, shared lock on current page
Data-only-locked table	Leaf level index scan	Level 0; dirty reads
Table name and nonclustered index name		

update statistics specifying	Scans and sorts performed	Locking
Allpages-locked table	Leaf level index scan	Level 1; shared intent table lock, shared lock on current page
Data-only-locked table	Leaf level index scan	Level 0; dirty reads
Table name and column name		
Allpages-locked table	Table scan; creates a worktable and sorts the worktable	Level 1; shared intent table lock, shared lock on current page
Data-only-locked table	Table scan; creates a worktable and sorts the worktable	Level 0; dirty reads

Sorts for unindexed or non leading columns

For unindexed columns and columns that are not the leading columns in indexes, Adaptive Server performs a serial table scan, copying the column values into a worktable, and then sorts the worktable in order to build the histogram. The sort is performed in serial, unless the with consumers clause is specified.

See Chapter 9, "Parallel Sorting" in the *Performance and Tuning: Optimizer* for information on parallel sort configuration requirements.

Locking, scans, and sorts during *update index statistics*

The update index statistics command generates a series of update statistics operations that use the same locking, scanning, and sorting as the equivalent index-level and column-level command. For example, if the salesdetail table has a nonclustered index named sales_det_ix on salesdetail(stor_id, ord_num, title_id), this command:

```
update index statistics salesdetail
```

performs these update statistics operations:

```
update statistics salesdetail sales_det_ix
update statistics salesdetail (ord_num)
update statistics salesdetail (title_id)
```

Locking, scans and sorts during *update all statistics*

The update all statistics commands generates a series of update statistics operations for each index on the table, followed by a series of update statistics operations for all unindexed columns, followed by an update partition statistics operation.

Using the *with consumers* clause

The with consumers clause for update statistics is designed for use on partitioned tables on RAID devices, which appear to Adaptive Server as a single I/O device, but which are capable of producing the high throughput required for parallel sorting. See chapter 9, Parallel Sorting in the *Performance and Tuning: Optimizer and Abstract Plans* for more information.

Reducing *update statistics* impact on concurrent processes

Since update statistics uses dirty reads (transaction isolation level 0) for data-only locked tables, it can be run while other tasks are active on the server, and does not block access to tables and indexes. Updating statistics for leading columns in indexes requires only a leaf-level scan of the index, and does not require a sort, so updating statistics for these columns does not affect concurrent performance very much.

However, updating statistics for unindexed and non leading columns, which require a table scan, worktable, and sort can affect concurrent processing.

- Sorts are CPU intensive. Use a serial sort, or a small number of worker processes if you want to minimize CPU utilization. Alternatively, you can use execution classes to set the priority for update statistics.

  See “Using Engines and CPUs” in the *Performance and Tuning: Basics*.

- The cache space required for merging sort runs is taken from the data cache, and some procedure cache space is also required. Setting the number of sort buffers to a low value reduces the space used in the buffer cache.

  If number of sort buffers is set to a large value, it takes more space from the data cache, and may also cause stored procedures to be flushed from the procedure cache, since procedure cache space is used while merging sorted values.

Creating the worktables for sorts also uses space in tempdb.

Using the *delete statistics* command

In pre-11.9 versions of SQL Server and Adaptive Server, dropping an index removes the distribution page for the index. In version 11.9.2, maintaining column-level statistics is under explicit user control, and the optimizer can use column-level statistics even when an index does not exist. The delete statistics command allows you to drop statistics for specific columns.

If you create an index and then decide to drop it because it is not useful for data access, or because of the cost of index maintenance during data modifications, you need to determine:

- Whether the statistics on the index are useful to the optimizer.
- Whether the distribution of key values in the columns for this index are subject to change over time as rows are inserted and deleted.

  If the distribution of key values changes, you need to run update statistics periodically to maintain useful statistics.

This example command deletes the statistics for the price column in the titles table:

```
delete statistics titles(price)
```

Note The delete statistics command, when used with a table name, removes all statistics for a table, even where indexes exist.

You must run update statistics on the table to restore the statistics for the index.

When row counts may be inaccurate

Row count values for the number of rows, number of forwarded rows, and number of deleted rows may be inaccurate, especially if query processing includes many rollback commands. If workloads are extremely heavy, and the housekeeper wash task does not run often, these statistics are more likely to be inaccurate.

Running update statistics corrects these counts in systabstats.

Running dbcc checktable or dbcc checkdb updates these values in memory.

When the housekeeper wash task runs, or when you execute sp_flushstats, these values are saved in systabstats.

Note The configuration parameter housekeeper free write percent must be set to 1 or greater to enable housekeeper statistics flushing.

CHAPTER 4

Using the *set statistics* Commands

Contains a guide to using the set statistics command.

Command syntax

The syntax for the set statistics commands is:

set statistics {io, simulate, subquerycache, time} [on | off]

You can issue a single command:

```
set statistics io on
```

You can combine more than one command on a single line by separating them with commas:

```
set statistics io, time on
```

Using simulated statistics

The optdiag utility command allows you to load simulated statistics and perform query diagnosis using those statistics. Since you can load simulated statistics even for tables that are empty, using simulated statistics allows you to perform tuning diagnostics in a very small database that contains only the tables and indexes. Simulated statistics do not overwrite any existing statistics when they are loaded, so you can also load them into an existing database.

Once simulated statistics have been loaded, instruct the optimizer to use them (rather than the actual statistics):

```
set statistics simulate on
```

For complete information on using simulated statistics, see “Using simulated statistics” on page 162.

Checking subquery cache performance

When subqueries are not flattened or materialized, a subquery cache is created to store results of earlier executions of the subquery to reduce the number of expensive executions of the subquery.

See “Displaying subquery cache information” on page 140 in the *Performance and Tuning: Optimizer and Abstract Plans* for information on using this option.

Checking compile and execute time

set statistics time displays information about the time it takes to parse and execute Adaptive Server commands.

```
Parse and Compile Time 57.
SQL Server cpu time: 5700 ms.

Execution Time 175.
SQL Server cpu time: 17500 ms.  SQL Server elapsed time: 70973 ms.
```

The meaning of this output is:

- Parse and Compile Time – The number of CPU ticks taken to parse, optimize, and compile the query. See below for information on converting ticks to milliseconds.
- SQL Server cpu time – Shows the CPU time in milliseconds.
- Execution Time – The number of CPU ticks taken to execute the query.
- SQL Server cpu time – The number of CPU ticks taken to execute the query, converted to milliseconds.
- SQL Server elapsed time – The difference in milliseconds between the time the command started and the current time, as taken from the operating system clock.

This output shows that the query was parsed and compiled in 57 clock ticks. It took 175 ticks, or 17.5 seconds, of CPU time to execute. Total elapsed time was 70.973 seconds, indicating that Adaptive Server spent some time processing other tasks or waiting for disk or network I/O to complete.

Converting ticks to milliseconds

To convert ticks to milliseconds

$$\text{Milliseconds} = \frac{\text{CPYU\_ticks} * \text{clock\_rate}}{1000}$$

To see the *clock_rate* for your system, execute:

```
sp_configure "sql server clock tick length"
```

See the *System Administration Guide* for more information.

Reporting physical and logical I/O statistics

set statistics io reports information about physical and logical I/O and the number of times a table was accessed. set statistics io output follows the query results and provides actual I/O performed by the query.

For each table in a query, including worktables, statistics io reports one line of information with several values for the pages read by the query and one row that reports the total number of writes. If a System Administrator has enabled resource limits, statistics io also includes a line that reports the total actual I/O cost for the query. The following example shows statistics io output for a query with resource limits enabled:

```
   select avg(total_sales)
   from titles
Table: titles  scan count 1,  logical reads: (regular=656 apf=0
total=656), physical reads: (regular=444 apf=212 total=656),  apf
IOs used=212
Total actual I/O cost for this command: 13120.
Total writes for this command: 0
```

The following sections describe the four major components of statistics io output:

- Actual I/O cost
- Total writes
- Read statistics
- Table name and “scan count”

Total actual I/O cost value

If resource limits are enabled, statistics io prints the “Total actual I/O cost” line. Adaptive Server reports the total actual I/O as a unitless number. The formula for determining the cost of a query is Cost = All physical IOs * 18 + All logical IOs * 2

This formula multiplies the “cost” of a logical I/O by the number of logical I/Os and the “cost” of a physical I/O by the number of physical I/Os.

For the example above that performs 656 physical reads and 656 logical reads, 656 * 2 + 656 * 18 = 13120, which is the total I/O cost reported by statistics io.

Statistics for writes

statistics io reports the total number of buffers written by the command. Read-only queries report writes when they cause dirty pages to move past the wash marker in the cache so that the write on the page starts.

Queries that change data may report only a single write, the log page write, because the changed pages remain in the MRU section of the data cache.

Statistics for reads

statistics io reports the number of logical and physical reads for each table and index included in a query, including worktables. I/O for indexes is included with the I/O for the table.

Table 4-1 shows the values that statistics io reports for logical and physical reads.

Table 4-1: statistics io output for reads

Output	Description
logical reads	
regular	Number of times that a page needed by the query was found in cache; only pages not brought in by asynchronous prefetch (APF) are counted here.
apf	Number of times that a request brought in by an APF request was found in cache.
total	Sum of *regular* and *apf* logical reads.
physical reads	
regular	Number of times a buffer was brought into cache by regular asynchronous I/O
apf	Number of times that a buffer w.as brought into cache by APF.
total	Sum of *regular* and *apf* physical reads.
apf IOs used	Number of buffers brought in by APF in which one or more pages were used during the query.

Sample output with and without an index

Using statistics io to perform a query on a table without an index and the same query on the same table with an index shows how important good indexes can be to query and system performance. Here is a sample query:

```
select title
from titles
where title_id = "T5652"
```

statistics io without an index

With no index on title_id, statistics io reports these values, using 2K I/O:

```
Table: titles scan count 1, logical reads:(regular=624
apf=0 total=624), physical reads:(regular=230 apf=394
total=624), apf IOs used=394
Total actual I/O cost for this command: 12480.
Total writes for this command: 0
```

This output shows that:

- The query performed a total of 624 logical I/Os, all regular logical I/Os.
- The query performed 624 physical reads. Of these, 230 were regular asynchronous reads, and 394 were asynchronous prefetch reads.
- All of the pages read by APF were used by the query.

statistics io with an Index

With a clustered index on title_id, statistics io reports these values for the same query, also using 2K I/O:

```
Table: titles  scan count 1,  logical reads: (regular=3 apf=0
total=3),
physical reads: (regular=3 apf=0 total=3),  apf IOs used=0
Total actual I/O cost for this command: 60.
Total writes for this command: 0
```

The output shows that:

- The query performed 3 logical reads.
- The query performed 3 physical reads: 2 reads for the index pages and 1 read for the data page.

statistics io output for cursors

For queries using cursors, statistics io prints the cumulative I/O since the cursor was opened:

```
1> open c
```

```
Table: titles scan count 0, logical reads: (regular=0 apf=0 total=0),
physical reads: (regular=0 apf=0 total=0), apf IOs used=0
Total actual I/O cost for this command: 0.
```

```
Total writes for this command: 0
             1> fetch c


title_id type         price
 -------- ------------ ----------------------
 T24140   business                     201.95
Table: titles scan count 1, logical reads: (regular=3 apf=0 total=3),
physical reads: (regular=0 apf=0 total=0), apf IOs used=0
Total actual I/O cost for this command: 6.
Total writes for this command: 0
             1> fetch c

   title_id type         price
    -------- ------------ ----------------------
    T24226   business                     201.95
   Table: titles scan count 1, logical reads: (regular=4 apf=0
   total=4), physical reads: (regular=0 apf=0 total=0), apf IOs
   used=0
   Total actual I/O cost for this command: 8.
   Total writes for this command: 0
```

Scan count

statistics io reports the number of times a query accessed a particular table. A "scan" can represent any of these access methods:

- A table scan.
- An access via a clustered index. Each time the query starts at the root page of the index and follows pointers to the data pages, it is counted as a scan.
- An access via a nonclustered index. Each time the query starts at the root page of the index and follows pointers to the leaf level of the index (for a covered query) or to the data pages, it is counted.
- If queries run in parallel, each worker process access to the table is counted as a scan.

Use showplan, as described in Chapter 5, "Using set showplan," to determine which access method is used.

Queries reporting a scan count of 1

Examples of queries that return a scan count of 1 are:

- A point query:

```
select title_id
from titles
    where title_id = "T55522"
```

- A range query:

```
select au_lname, au_fname
    from authors
    where au_lname > "Smith"
    and au_lname < "Smythe"
```

If the columns in the where clauses of these queries are indexed, the queries can use the indexes to scan the tables; otherwise, they perform table scans. In either case, they require only a single scan of the table to return the required rows.

Queries reporting a scan count of more than 1

Examples of queries that return larger scan count values are:

- Parallel queries that report a scan for each worker process.
- Queries that have indexed where clauses connected by or report a scan for each or clause. If the query uses the special OR strategy, it reports one scan for each value. If the query uses the OR strategy, it reports one scan for each index, plus one scan for the RID list access.

  This query uses the special OR strategy, so it reports a scan count of 2 if the titles table has indexes on title_id and another on pub_id:

```
select title_id
from titles
    where title_id = "T55522"
    or pub_id  = "P988"
Table: titles scan count 2,logical reads: (regular=149 apf=0
total=149), physical reads: (regular=63 apf=80 total=143), apf IOs
used=80
Table: Worktable1  scan count 1, logical reads: (regular=172 apf=0
total=172), physical reads: (regular=0 apf=0 total=0), apf IOs
```

  The I/O for the worktable is also reported.

- Nested-loop joins that scan inner tables once for each qualifying row in the outer table. In the following example, the outer table, publishers, has three publishers with the state “NY”, so the inner table, titles, reports a scan count of 3:

```
select title_id
```

```
                from titles t, publishers p
                where t.pub_id = p.pub_id
                    and p.state = "NY"
Table: titles scan count 3,logical reads: (regular=442 apf=0
total=442), physical reads: (regular=53 apf=289 total=342), apf IOs
used=289
Table: publishers scan count 1, logical reads: (regular=2 apf=0
total=2), physical reads: (regular=2 apf=0 total=2), apf IOs used=0
```

This query performs a table scan on publishers, which occupies only 2 data pages, so 2 physical I/Os are reported. There are 3 matching rows in publishers, so the query scans titles 3 times, using an index on pub_id.

- Merge joins with duplicate values in the outer table restart the scan for each duplicate value, and report an additional scan count each time.

Queries reporting scan count of 0

Multistep queries and certain other types of queries may report a scan count of 0. Some examples are:

- Queries that perform deferred updates
- select...into queries
- Queries that create worktables

Relationship between physical and logical reads

If a page needs to be read from disk, it is counted as a physical read and a logical read. Logical I/O is always greater than or equal to physical I/O.

Logical I/O always reports 2K data pages. Physical reads and writes are reported in buffer-sized units. Multiple pages that are read in a single I/O operation are treated as a unit: they are read, written, and moved through the cache as a single buffer.

Logical reads, physical reads, and 2K I/O

With 2K I/O, the number of times that a page is found in cache for a query is logical reads minus physical reads. When the total number of logical reads and physical reads is the same for a table scan, it means that each page was read from disk and accessed only once during the query.

When pages for the query are found in cache, logical reads are higher than physical reads. This happens frequently with pages from higher levels of the index, since they are reused often, and tend to remain in cache.

Physical reads and large I/O

Physical reads are not reported in pages, but in buffers, that is, the actual number of times Adaptive Server accesses the disk.

- If the query uses 16K I/O (showplan reports the I/O size), a single physical read brings 8 data pages into cache.
- If a query reports 100 16K physical reads, it has read 800 data pages into cache.
- If the query needs to scan each of those data pages, it reports 800 logical reads.
- If a query, such as a join query, must read the page multiple times because other I/O has flushed the page from the cache, each physical read is counted.

Reads and writes on worktables

Reads and writes are reported for any worktable that needs to be created for the query. When a query creates more than one worktable, the worktables are numbered in statistics io output to correspond to the worktable numbers used in showplan output.

Effects of caching on reads

If you are testing a query and checking its I/O, and you execute the same query a second time, you may get surprising physical read values, especially if the query uses LRU replacement strategy.

The first execution reports a high number of physical reads; the second execution reports 0 physical reads.

The first time you execute the query, all the data pages are read into cache and remain there until other server processes flush them from the cache. Depending on the cache strategy used for the query, the pages may remain in cache for a longer or shorter period of time.

- If the query uses the fetch-and-discard (MRU) cache strategy, the pages are read into the cache at the wash marker.

In small or very active caches, pages read into the cache at the wash marker are flushed quickly.

- If the query uses LRU cache strategy to read the pages in at the top of the MRU end of the page chain, the pages remain in cache for longer periods of time.

During actual use on a production system, a query can be expected to find some of the required pages already in the cache, from earlier access by other users, while other pages need to be read from disk. Higher levels of indexes, in particular, tend to be frequently used, and tend to remain in the cache.

If you have a table or index bound to a cache that is large enough to hold all the pages, no physical I/O takes place once the object has been read into cache.

However, during query tuning on a development system with few users, you may want to clear the pages used for the query from cache in order to see the full physical I/O needed for a query. You can clear an object's pages from cache by:

- Changing the cache binding for the object:
  - If a table or index is bound to a cache, unbind it, and rebind it.
  - If a table or index is not bound to a cache, bind it to any cache available, then unbind it.

  You must have at least one user-defined cache to use this option.
- If you do not have any user-defined caches, you can execute a sufficient number of queries on other tables, so that the objects of interest are flushed from cache. If the cache is very large, this can be time-consuming.
- The only other alternative is rebooting the server.

For more information on testing and cache performance, see "Testing data cache performance" on page 218 *Performance and Tuning: Basics*.

statistics io and merge joins

statistics io output does not include sort costs for merge joins. If you have allow resource limits enabled, the sort cost is not reported in the "Total estimated I/O cost" and "Total actual I/O cost" statistics. Only dbcc traceon(310) shows these costs.

CHAPTER 5

Using *set showplan*

This chapter describes each message printed by the showplan utility. showplan displays the steps performed for each query in a batch, the keys and indexes used for the query, the order of joins, and special optimizer strategies.

Using

To see the query plan for a query, use:

```
set showplan on
```

To stop displaying query plans, use:

```
set showplan off
```

You can use showplan in conjunction with other set commands.

When you want to display showplans for a stored procedure, but not execute them, use the set fmtonly command.

See Chapter 4, “Query Tuning Tools,” in *Performance and Tuning: Optimizer* for information on how options affect each other’s operation.

Note Do not use set noexec with stored procedures - compilation and execution will not occur and you will not get the necessary output

---

Basic *showplan* messages

This section describes showplan messages that are printed for most select, insert, update, and delete operations.

Query plan delimiter message

```
QUERY PLAN FOR STATEMENT N (at line N)
```

Adaptive Server prints this line once for each query in a batch. Its main function is to provide a visual cue that separates one section of showplan output from the next section. Line numbers are provided to help you match query output with your input.

Step message

```
STEP N
```

showplan output displays "STEP *N*" for every query, where *N* is an integer, beginning with "STEP 1". For some queries, Adaptive Server cannot retrieve the results in a single step and breaks the query plan into several steps. For example, if a query includes a group by clause, Adaptive Server breaks it into at least two steps:

- One step to select the qualifying rows from the table and to group them, placing the results in a worktable
- Another step to return the rows from the worktable

This example demonstrates a single-step query.

```
    select au_lname, au_fname
    from authors
    where city = "Oakland"
QUERY PLAN FOR STATEMENT 1 (at line 1).

    STEP 1
        The type of query is SELECT.

        FROM TABLE
            authors
        Nested iteration.
        Table Scan.
        Forward scan.
```

```
Positioning at start of table.
Using I/O Size 2 Kbytes for data pages.
With LRU Buffer Replacement Strategy for data pages.
```

Multiple-step queries are demonstrated following "GROUP BY message" on page 83.

Query type message

```
The type of query is query type.
```

This message describes the type of query for each step. For most queries that require tuning, the value for *query type* is SELECT, INSERT, UPDATE, or DELETE. However, the *query type* can include any Transact-SQL command that you issue while showplan is enabled. For example, here is output from a create index command:

```
STEP 1
        The type of query is CREATE INDEX.
        TO TABLE
            titleauthor
```

FROM TABLE message

```
FROM TABLE
    tablename [ correlation_name ]
```

This message indicates which table the query is reading from. The "FROM TABLE" message is followed on the next line by the table name. If the from clause includes correlation names for tables, these are printed after the table names. When queries create and use worktables, the "FROM TABLE" prints the name of the worktable.

When your query joins one or more tables, the order of "FROM TABLE" messages in the output shows you the order in which the query plan chosen by the optimizer joins the tables. This query displays the join order in a three-table join:

```
    select a.au_id, au_fname, au_lname
        from titles t, titleauthor ta, authors a
    where a.au_id = ta.au_id
        and ta.title_id = t.title_id
        and au_lname = "Bloom"
QUERY PLAN FOR STATEMENT 1 (at line 1).
```

```
    STEP 1
        The type of query is SELECT.

        FROM TABLE
            authors
            a
        Nested iteration.
        Index : au_lname_ix
        Forward scan.
        Positioning by key.
        Keys are:
            au_lname  ASC
        Using I/O Size 2 Kbytes for index leaf pages.
       With LRU Buffer Replacement Strategy for index leaf pages.
        Using I/O Size 2 Kbytes for data pages.
        With LRU Buffer Replacement Strategy for data pages.

        FROM TABLE
            titleauthor
            ta
        Nested iteration.
        Index : at_ix
        Forward scan.
        Positioning by key.
       Index contains all needed columns. Base table will not be
read.
        Keys are:
            au_id  ASC
        Using I/O Size 2 Kbytes for index leaf pages.
       With LRU Buffer Replacement Strategy for index leaf pages.

        FROM TABLE
            titles
            t
        Nested iteration.
        Using Clustered Index.
        Index : title_id_ix
        Forward scan.
        Positioning by key.
       Index contains all needed columns. Base table will not be
read.
        Keys are:
            title_id  ASC
        Using I/O Size 2 Kbytes for index leaf pages.
       With LRU Buffer Replacement Strategy for index leaf pages.
```

The sequence of tables in this output shows the order chosen by the query optimizer, which is not the order in which they were listed in the from clause or where clause:

- First, the qualifying rows from the authors table are located (using the search clause on au_lname).
- Then, those rows are joined with the titleauthor table (using the join clause on the au_id columns).
- Finally, the titles table is joined with the titleauthor table to retrieve the desired columns (using the join clause on the title_id columns).

FROM TABLE and referential integrity

When you insert or update rows in a table that has a referential integrity constraint, the showplan output includes "FROM TABLE" and other messages indicating the method used to access the referenced table. This salesdetail table definition includes a referential integrity check on the title_id column:

```
create table salesdetail (
        stor_id            char(4),
        ord_num            varchar(20),
        title_id           tid
            references titles(title_id),
        qty                smallint,
        discount            float )
```

An insert to salesdetail, or an update on the title_id column, requires a lookup in the titles table:

```
insert salesdetail values ("S245", "X23A5", "T10", 15,
40.25)
QUERY PLAN FOR STATEMENT 1 (at line 1).

    STEP 1
        The type of query is INSERT.
        The update mode is direct.

        FROM TABLE
            titles
        Using Clustered Index.
        Index : title_id_ix
        Forward scan.
        Positioning by key.
        Keys are:
            title_id
```

```
 Using I/O Size 2 Kbytes for index leaf pages.
With LRU Buffer Replacement Strategy for index leaf pages.
 TO TABLE
     salesdetail
```

The clustered index on title_id_ix is used to verify the referenced value.

TO TABLE message

```
TO TABLE
    tablename
```

When a command such as insert, delete, update, or select into modifies or attempts to modify one or more rows of a table, the "TO TABLE" message displays the name of the target table. For operations that require an intermediate step to insert rows into a worktable, "TO TABLE" indicates that the results are going to the "Worktable" table rather than to a user table. This insert command shows the use of the "TO TABLE" statement:

```
insert sales
values ("8042", "QA973", "12/7/95")
QUERY PLAN FOR STATEMENT 1 (at line 1).

    STEP 1
        The type of query is INSERT.
        The update mode is direct.
        TO TABLE
            sales
```

Here is a command that performs an update:

```
update publishers
set city = "Los Angeles"
where pub_id = "1389"
QUERY PLAN FOR STATEMENT 1 (at line 1).

  STEP 1
       The type of query is UPDATE.
       The update mode is direct.

       FROM TABLE
           publishers
       Nested iteration.
       Using Clustered Index.
       Index : publ_id_ix
       Forward scan.
```

```
Positioning by key.
Keys are:
    pub_id  ASC
Using I/O Size 2 Kbytes for index leaf pages.
With LRU Buffer Replacement Strategy for index leaf pages.
Using I/O Size 2 Kbytes for data pages.
With LRU Buffer Replacement Strategy for data pages.
TO TABLE
    publishers
```

The update query output indicates that the publishers table is used as both the "FROM TABLE" and the "TO TABLE". In the case of update operations, the optimizer needs to read the table that contains the row(s) to be updated, resulting in the "FROM TABLE" statement, and then needs to modify the row(s), resulting in the "TO TABLE" statement.

Update mode messages

Adaptive Server uses different modes to perform update operations such as insert, delete, update, and select into. These methods are called **direct update mode** and **deferred update mode**.

Direct update mode

```
The update mode is direct.
```

Whenever possible, Adaptive Server uses direct update mode, since it is faster and generates fewer log records than deferred update mode.

The direct update mode operates as follows:

1 Pages are read into the data cache.

2 The changes are recorded in the transaction log.

3 The change is made to the data page.

4 The transaction log page is flushed to disk when the transaction commits.

For more information on the different types of direct updates, see "How Update Operations Are Performed" on page 112.

Adaptive Server uses direct update mode for the following delete command:

```
delete
from authors
where au_lname = "Willis"
```

```
          and au_fname = "Max"
QUERY PLAN FOR STATEMENT 1 (at line 1).

    STEP 1
        The type of query is DELETE.
        The update mode is direct.

        FROM TABLE
            authors
        Nested iteration.
        Using Clustered Index.
        Index : au_names_ix
        Forward scan.
        Positioning by key.
        Keys are:
            au_lname  ASC
            au_fname  ASC
        Using I/O Size 2 Kbytes for index leaf pages.
       With LRU Buffer Replacement Strategy for index leaf pages.
        Using I/O Size 2 Kbytes for data pages.
        With LRU Buffer Replacement Strategy for data pages.
        TO TABLE
            authors
```

Deferred mode

```
        The update mode is deferred.
```

In deferred mode, processing takes place in these steps:

1 For each qualifying data row, Adaptive Server writes transaction log records for one deferred delete and one deferred insert.

2 Adaptive Server scans the transaction log to process the deferred inserts, changing the data pages and any affected index pages.

Consider the following insert...select operation, where mytable is a heap without a clustered index or a unique nonclustered index:

```
        insert mytable
            select title, price * 2
                from mytable
QUERY PLAN FOR STATEMENT 1 (at line 1).

 STEP 1
     The type of query is INSERT.
     The update mode is deferred.
```

```
FROM TABLE
    mytable
Nested iteration.
Table Scan.
Forward scan.
Positioning at start of table.
Using I/O Size 2 Kbytes for data pages.
With LRU Buffer Replacement Strategy for data pages.
TO TABLE
    mytable
```

This command copies every row in the table and appends the rows to the end of the table.

It needs to differentiate between the rows that are currently in the table (prior to the insert command) and the rows being inserted so that it does not get into a continuous loop of selecting a row, inserting it at the end of the table, selecting the row that it just inserted, and reinserting it.

The query processor solves this problem by performing the operation in two steps:

1 It scans the existing table and writes insert records into the transaction log for each row that it finds.

2 When all the "old" rows have been read, it scans the log and performs the insert operations.

Deferred index and deferred varcol messages

```
The update mode is deferred_varcol.

The update mode is deferred_index.
```

These showplan messages indicate that Adaptive Server may process an update command as a deferred index update.

Adaptive Server uses deferred_varcol mode when updating one or more variable-length columns. This update may be done in deferred or direct mode, depending on information that is available only at runtime.

Adaptive Server uses deferred_index mode when the index is unique or may change as part of the update. In this mode, Adaptive Server deletes the index entries in direct mode but inserts them in deferred mode.

Optimized using messages

These messages are printed when special optimization options are used for a query.

Simulated statistics message

```
Optimized using simulated statistics.
```

The simulated statistics message is printed when:

- The set statistics simulate option was active when the query was optimized, and
- Simulated statistics have been loaded using optdiag.

Abstract plan messages

```
Optimized using an Abstract Plan (ID : N).
```

The message above is printed when an abstract plan was associated with the query. The variable prints the ID number of the plan.

```
Optimized using the Abstract Plan in the PLAN clause.
```

The message above is printed when the plan clause is used for a select, update, or delete statement. See Creating and Using Abstract Plans in the *Performance and Tuning Guide: Optimizer* for more information.

showplan messages for query clauses

Use of certain Transact-SQL clauses, functions, and keywords is reflected in showplan output. These include group by, aggregates, distinct, order by, and select into clauses.

Use of certain Transact-SQL clauses, functions, and keywords is reflected in showplan output. These include group by, aggregates, distinct, order by, and select into clauses.

Table 5-1: showplan messages for various clauses

Message	Explanation
GROUP BY	The query contains a group by statement.

Message	Explanation
The type of query is SELECT (into WorktableN).	The step creates a worktable to hold intermediate results.
Evaluate Grouped type AGGREGATE Evaluate Ungrouped type AGGREGATE.	The query contains an aggregate function. "Grouped" indicates that there is a grouping column for the aggregate (vector aggregate). "Ungrouped" indicates that there is no grouping column (scalar aggregate). The variable indicates the type of aggregate.
Evaluate Grouped ASSIGNMENT OPERATOR Evaluate Ungrouped ASSIGNMENT OPERATOR	The query includes compute (ungrouped) or compute by (grouped).
WorktableN created for DISTINCT.	The query contains the distinct keyword in the select list and requires a sort to eliminate duplicates.
WorktableN created for ORDER BY.	The query contains an order by clause that requires ordering rows.
This step involves sorting.	The query includes on order by or distinct clause, and results must be sorted.
Using GETSORTED	The query created a worktable and sorted it. GETSORTED is a particular technique used to return the rows.
The sort for WorktableN is done in Serial. The sort for WorktableN is done in Parallel.	Indicates how the sort for a worktable is performed.

GROUP BY message

```
GROUP BY
```

This statement appears in the showplan output for any query that contains a group by clause. Queries that contain a group by clause are always executed in at least two steps:

- One step selects the qualifying rows into a worktable and groups them.
- Another step returns the rows from the worktable.

Selecting into a worktable

```
The type of query is SELECT (into WorktableN).
```

Queries using a group by clause first put qualifying results into a worktable. The data is grouped as the table is generated. A second step returns the grouped rows.

The following example returns a list of all cities and indicates the number of authors that live in each city. The query plan shows the two steps: the first step selects the rows into a worktable, and the second step retrieves the grouped rows from the worktable:

```
   select city, total_authors = count(*)
       from authors
       group by city
QUERY PLAN FOR STATEMENT 1 (at line 1).

    STEP 1
        The type of query is SELECT (into Worktable1).
        GROUP BY
        Evaluate Grouped COUNT AGGREGATE.

        FROM TABLE
            authors
        Nested iteration.
        Table Scan.
        Forward scan.
        Positioning at start of table.
        Using I/O Size 16 Kbytes for data pages.
        With LRU Buffer Replacement Strategy for data pages.
        TO TABLE
            Worktable1.

    STEP 2
        The type of query is SELECT.

        FROM TABLE
            Worktable1.
        Nested iteration.
        Table Scan.
        Forward scan.
        Positioning at start of table.
        Using I/O Size 16 Kbytes for data pages.
        With MRU Buffer Replacement Strategy for data pages.
```

Grouped aggregate message

```
Evaluate Grouped type AGGREGATE
```

This message is printed by queries that contain aggregates and group by or compute by.

The variable indicates the type of aggregate—COUNT, SUM OR AVERAGE, MINIMUM, or MAXIMUM.

avg reports both COUNT and SUM OR AVERAGE; sum reports SUM OR AVERAGE. Two additional types of aggregates (ONCE and ANY) are used internally by Adaptive Server while processing subqueries.

See "Internal Subquery Aggregates" on page 864.

Grouped aggregates and *group by*

When an aggregate function is combined with group by, the result is called a grouped aggregate, or **vector aggregate**. The query results have one row for each value of the grouping column or columns.

The following example illustrates a grouped aggregate:

```
    select type, avg(advance)
    from titles
    group by type
QUERY PLAN FOR STATEMENT 1 (at line 1).

  STEP 1
       The type of query is SELECT (into Worktable1).
       GROUP BY
       Evaluate Grouped COUNT AGGREGATE.
       Evaluate Grouped SUM OR AVERAGE AGGREGATE.

       FROM TABLE
           titles
       Nested iteration.
       Table Scan.
       Forward scan.
       Positioning at start of table.
       Using I/O Size 16 Kbytes for data pages.
       With LRU Buffer Replacement Strategy for data pages.
       TO TABLE
           Worktable1.

   STEP 2
       The type of query is SELECT.

       FROM TABLE
           Worktable1.
```

```
Nested iteration.
Table Scan.
Forward scan.
Positioning at start of table.
Using I/O Size 16 Kbytes for data pages.
With MRU Buffer Replacement Strategy for data pages.
```

In the first step, the worktable is created, and the aggregates are computed. The second step selects the results from the worktable.

compute by message

```
Evaluate Grouped ASSIGNMENT OPERATOR
```

Queries using compute by display the same aggregate messages as group by, with the "Evaluate Grouped ASSIGNMENT OPERATOR" message.

The values are placed in a worktable in one step, and the computation of the aggregates is performed in a second step. This query uses type and advance, like the group by query example above:

```
select type, advance  from titles
having title like "Compu%"
order by type
compute avg(advance) by type
```

In the showplan output, the computation of the aggregates takes place in step 2:

```
QUERY PLAN FOR STATEMENT 1 (at line 1).

    STEP 1
        The type of query is INSERT.
        The update mode is direct.
        Worktable1 created for ORDER BY.

        FROM TABLE
            titles
        Nested iteration.
        Index : title_ix
        Forward scan.
        Positioning by key.
        Keys are:
            title  ASC
        Using I/O Size 2 Kbytes for index leaf pages.
       With LRU Buffer Replacement Strategy for index leaf pages.
        Using I/O Size 2 Kbytes for data pages.
        With LRU Buffer Replacement Strategy for data pages.
```

```
        TO TABLE
            Worktable1.

    STEP 2
        The type of query is SELECT.
        Evaluate Grouped SUM OR AVERAGE AGGREGATE.
        Evaluate Grouped COUNT AGGREGATE.
        Evaluate Grouped ASSIGNMENT OPERATOR.
        This step involves sorting.

        FROM TABLE
            Worktable1.
        Using GETSORTED
        Table Scan.
        Forward scan.
        Positioning at start of table.
        Using I/O Size 16 Kbytes for data pages.
        With MRU Buffer Replacement Strategy for data pages.
```

Ungrouped aggregate message

```
Evaluate Ungrouped type AGGREGATE.
```

This message is reported by:

- Queries that use aggregate functions, but do not use group by
- Queries that use compute

Ungrouped aggregates

When an aggregate function is used in a select statement that does not include a group by clause, it produces a single value. The query can operate on all rows in a table or on a subset of the rows defined by a where clause.

When an aggregate function produces a single value, the function is called a **scalar aggregate**, or an ungrouped aggregate. Here is showplan output for an ungrouped aggregate:

```
    select avg(advance)
    from titles
    where type = "business"
QUERY PLAN FOR STATEMENT 1 (at line 1).

    STEP 1
        The type of query is SELECT.
```

```
        Evaluate Ungrouped COUNT AGGREGATE.
        Evaluate Ungrouped SUM OR AVERAGE AGGREGATE.

        FROM TABLE
            titles
        Nested iteration.
        Index : type_price
        Forward scan.
        Positioning by key.
        Keys are:
            type  ASC
        Using I/O Size 2 Kbytes for index leaf pages.
        With LRU Buffer Replacement Strategy for index leaf pages.
        Using I/O Size 2 Kbytes for data pages.
        With LRU Buffer Replacement Strategy for data pages.

    STEP 2
        The type of query is SELECT.
```

This is a two-step query, similar to the showplan from the group by query shown earlier.

Since the scalar aggregate returns a single value, Adaptive Server uses an internal variable to compute the result of the aggregate function, as the qualifying rows from the table are evaluated. After all rows from the table have been evaluated (step 1), the final value from the variable is selected (step 2) to return the scalar aggregate result.

compute messages

```
Evaluate Ungrouped ASSIGNMENT OPERATOR
```

When a query includes compute to compile a scalar aggregate, showplan prints the "Evaluate Ungrouped ASSIGNMENT OPERATOR" message. This query computes an average for the entire result set:

```
select type, advance from titles
where title like "Compu%"
order by type
compute avg(advance)
```

The showplan output shows that the computation of the aggregate values takes place in the step 2:

```
QUERY PLAN FOR STATEMENT 1 (at line 1).

    STEP 1
```

```
    The type of query is INSERT.
    The update mode is direct.
    Worktable1 created for ORDER BY.

    FROM TABLE
        titles
    Nested iteration.
    Index : title_ix
    Forward scan.
    Positioning by key.
    Keys are:
        title  ASC
    Using I/O Size 2 Kbytes for index leaf pages.
    With LRU Buffer Replacement Strategy for index leaf pages.
    Using I/O Size 2 Kbytes for data pages.
    With LRU Buffer Replacement Strategy for data pages.
    TO TABLE
        Worktable1.

STEP 2
    The type of query is SELECT.
    Evaluate Ungrouped SUM OR AVERAGE AGGREGATE.
    Evaluate Ungrouped COUNT AGGREGATE.
    Evaluate Ungrouped ASSIGNMENT OPERATOR.
    This step involves sorting.

    FROM TABLE
        Worktable1.
    Using GETSORTED
    Table Scan.
    Forward scan.
    Positioning at start of table.
    Using I/O Size 16 Kbytes for data pages.
    With MRU Buffer Replacement Strategy for data pages.
```

messages for *order by* and *distinct*

Some queries that include distinct use a sort step to enforce the uniqueness of values in the result set. distinct queries and order by queries do not require the sorting step when the index used to locate rows supports the order by or distinct clause.

For those cases where the sort is performed, the distinct keyword in a select list and the order by clause share some showplan messages:

- Each generates a worktable message.
- The message “This step involves sorting.”.
- The message “Using GETSORTED”.

Worktable message for *distinct*

```
Worktable N created for DISTINCT.
```

A query that includes the distinct keyword excludes all duplicate rows from the results so that only unique rows are returned. When there is no useful index, Adaptive Server performs these steps to process queries that include distinct:

1 It creates a worktable to store all of the results of the query, including duplicates.

2 It sorts the rows in the worktable, discards the duplicate rows, and then returns the rows.

Subqueries with existence joins sometimes create a worktable and sort it to remove duplicate rows.

See “Flattening in, any, and exists subqueries” on page 145 for more information.

The “Worktable*N* created for DISTINCT” message appears as part of “Step 1” in showplan output. “Step 2” for distinct queries includes the messages “This step involves sorting” and “Using GETSORTED”. See “Sorting messages” on page 812.

```
    select distinct city
    from authors
QUERY PLAN FOR STATEMENT 1 (at line 1).

    STEP 1
        The type of query is INSERT.
        The update mode is direct.
        Worktable1 created for DISTINCT.

        FROM TABLE
            authors
        Nested iteration.
        Table Scan.
        Forward scan.
        Positioning at start of table.
        Using I/O Size 16 Kbytes for data pages.
        With LRU Buffer Replacement Strategy for data pages.
```

```
        TO TABLE
            Worktable1.

    STEP 2
        The type of query is SELECT.
        This step involves sorting.

        FROM TABLE
            Worktable1.
        Using GETSORTED
        Table Scan.
        Forward scan.
        Positioning at start of table.
        Using I/O Size 16 Kbytes for data pages.
        With MRU Buffer Replacement Strategy for data pages.
```

Worktable message for *order by*

```
Worktable N created for ORDER BY.
```

Queries that include an order by clause often require the use of a temporary worktable. When the optimizer cannot use an index to order the result rows, it creates a worktable to sort the result rows before returning them. This example shows an order by clause that creates a worktable because there is no index on the city column:

```
select *
from authors
order by city
QUERY PLAN FOR STATEMENT 1 (at line 1).

    STEP 1
        The type of query is INSERT.
        The update mode is direct.
        Worktable1 created for ORDER BY.

        FROM TABLE
            authors
        Nested iteration.
        Table Scan.
        Forward scan.
        Positioning at start of table.
        Using I/O Size 16 Kbytes for data pages.
        With LRU Buffer Replacement Strategy for data pages.
        TO TABLE
            Worktable1.
```

```
STEP 2
    The type of query is SELECT.
    This step involves sorting.

    FROM TABLE
        Worktable1.
    Using GETSORTED
    Table Scan.
    Forward scan.
    Positioning at start of table.
    Using I/O Size 16 Kbytes for data pages.
    With MRU Buffer Replacement Strategy for data pages.
```

order by queries and indexes

Certain queries using order by do not require a sorting step, depending on the type of index used to access the data.

See Chapter 8, "Indexing for Performance," for more information.

Sorting messages

These messages report on sorts.

Step involves sorting message

```
This step involves sorting.
```

This showplan message indicates that the query must sort the intermediate results before returning them to the user. Queries that use distinct or that have an order by clause not supported by an index require an intermediate sort. The results are put into a worktable, and the worktable is then sorted.

For examples of this message, see "Worktable message for distinct" on page 810 and "Worktable message for order by" on page 811.

GETSORTED message

```
Using GETSORTED
```

This statement indicates one of the ways that Adaptive Server returns result rows from a table.

In the case of "Using GETSORTED," the rows are returned in sorted order. However, not all queries that return rows in sorted order include this step. For example, order by queries whose rows are retrieved using an index with a matching sort sequence do not require "GETSORTED."

The "Using GETSORTED" method is used when Adaptive Server must first create a temporary worktable to sort the result rows and then return them in the proper sorted order. The examples for distinct on and for order by on show the "Using GETSORTED" message.

Serial or parallel sort message

```
The sort for WorktableN is done in Serial.

The sort for WorktableN is done in Parallel.
```

These messages indicate whether a serial or parallel sort was performed for a worktable. They are printed after the sort manager determines whether a given sort should be performed in parallel or in serial.

If set noexec is in effect, the worktable is not created, so the sort is not performed, and no message is displayed.

Messages describing access methods, caching, and I/O cost

showplan output provides information about access methods and caching strategies.

Auxiliary scan descriptors message

```
Auxiliary scan descriptors required: N
```

When a query involving referential integrity requires a large number of user or system tables, including references to other tables to check referential integrity, this showplan message indicates the number of auxiliary scan descriptors needed for the query. If a query does not exceed the number of pre allocated scan descriptors allotted for the session, the "Auxiliary scan descriptors required" message is not printed.

The following example shows partial output for a delete from the employees table, which is referenced by 30 foreign tables:

```
          delete employees
          where empl_id = "222-09-3482"
QUERY PLAN FOR STATEMENT 1 (at line 1).

Auxiliary scan descriptors required: 4


    STEP 1
        The type of query is DELETE.
        The update mode is direct.

        FROM TABLE
            employees
        Nested iteration.
        Using Clustered Index.
        Index : employees_empl_i_10080066222
        Forward scan.
        Positioning by key.
        Keys are:
            empl_id  ASC
        Using I/O Size 2 Kbytes for index leaf pages.
       With LRU Buffer Replacement Strategy for index leaf pages.
        Using I/O Size 2 Kbytes for data pages.
        With LRU Buffer Replacement Strategy for data pages.

        FROM TABLE
            benefits
        Index : empl_id_ix
        Forward scan.
        Positioning by key.
       Index contains all needed columns. Base table will not be
read.
        Keys are:
            empl_id  ASC
        Using I/O Size 2 Kbytes for index leaf pages.
       With LRU Buffer Replacement Strategy for index leaf pages.
        .
        .
        .
        FROM TABLE
            dependents
        Index : empl_id_ix
        Forward scan.
        Positioning by key.
```

```
        Index contains all needed columns. Base table will not be
    read.
        Keys are:
            empl_id  ASC
        Using I/O Size 2 Kbytes for index leaf pages.
        With LRU Buffer Replacement Strategy for index leaf pages.
        TO TABLE
            employees
```

Nested iteration message

```
    Nested Iteration.
```

This message indicates one or more loops through a table to return rows. Even the simplest access to a single table is an iteration, as shown here:

```
    select * from publishers
QUERY PLAN FOR STATEMENT 1 (at line 1).

    STEP 1
        The type of query is SELECT.

        FROM TABLE
            publishers
        Nested iteration.
        Table Scan.
        Forward scan.
        Positioning at start of table.
        Using I/O Size 2 Kbytes for data pages.
        With LRU Buffer Replacement Strategy for data pages.
```

For queries that perform nested-loop joins, access to each table is nested within the scan of the outer table.

See “Nested-Loop Joins” on page 128 for more information.

Merge join messages

```
    Merge join (outer table).

    Merge join (inner table).
```

Merge join messages indicate the use of a merge join and the table’s position (inner or outer) with respect to the other table in the merge join. Merge join messages appear immediately after the table name in the

```
    FROM TABLE
```

output. This query performs a mixture of merge and nested-loop joins:

```
select pub_name, au_lname, price
from titles t, authors a, titleauthor ta,
            publishers p
where t.title_id = ta.title_id
    and a.au_id = ta.au_id
    and p.pub_id = t.pub_id
    and type = 'business'
    and price < $25
```

Messages for merge joins are printed in bold type in the showplan output:

```
QUERY PLAN FOR STATEMENT 1 (at line 1).
Executed in parallel by coordinating process and 3 worker
processes.


    STEP 1
        The type of query is INSERT.
        The update mode is direct.
       Executed in parallel by coordinating process and 3
worker processes.

        FROM TABLE
            titles
            t
        Merge join (outer table).
        Parallel data merge using 3 worker processes.
        Using Clustered Index.
        Index : title_id_ix
        Forward scan.
        Positioning by key.
        Keys are:
            title_id  ASC
        Using I/O Size 16 Kbytes for data pages.
       With LRU Buffer Replacement Strategy for data pages.

        FROM TABLE
            titleauthor
            ta
        Merge join (inner table).
        Index : ta_ix
        Forward scan.
        Positioning by key.
       Index contains all needed columns. Base table will
```

```
not be read.
        Keys are:
            title_id  ASC
        Using I/O Size 16 Kbytes for index leaf pages.
       With LRU Buffer Replacement Strategy for index leaf
pages.

        FROM TABLE
            authors
            a
        Nested iteration.
        Index : au_id_ix
        Forward scan.
        Positioning by key.
        Keys are:
            au_id  ASC
        Using I/O Size 2 Kbytes for index leaf pages.
       With LRU Buffer Replacement Strategy for index leaf
pages.
        Using I/O Size 2 Kbytes for data pages.
       With LRU Buffer Replacement Strategy for data pages.
        TO TABLE
            Worktable1.
        Worktable1 created for sort merge join.

    STEP 2
        The type of query is INSERT.
        The update mode is direct.
        Executed by coordinating process.

        FROM TABLE
            publishers
            p
        Nested iteration.
        Table Scan.
        Forward scan.
        Positioning at start of table.
        Using I/O Size 2 Kbytes for data pages.
       With LRU Buffer Replacement Strategy for data pages.
        TO TABLE
            Worktable2.
        Worktable2 created for sort merge join.

    STEP 3
        The type of query is SELECT.
        Executed by coordinating process.
```

```
    FROM TABLE
        Worktable1.
    Merge join (outer table).
    Serial data merge.
    Table Scan.
    Forward scan.
    Positioning at start of table.
    Using I/O Size 2 Kbytes for data pages.
  With LRU Buffer Replacement Strategy for data pages.

    FROM TABLE
        Worktable2.
    Merge join (inner table).
    Table Scan.
    Forward scan.
    Positioning at start of table.
    Using I/O Size 2 Kbytes for data pages.
  With LRU Buffer Replacement Strategy for data pages.

Total estimated I/O cost for statement 1 (at line 1): 4423.

The sort for Worktable1 is done in Serial

The sort for Worktable2 is done in Serial
```

This query performed the following joins:

- A full-merge join on titles and titleauthor, with titles as the outer table
- A nested-loop join with the authors table
- A sort-merge join with the publishers table

Worktable message

```
Worktable N created for sort merge join.
```

If a merge join requires a sort for a table, a worktable is created and sorted into order by the join key. A later step in the query uses the worktable as either an inner table or outer table.

Table scan message

```
Table Scan.
```

This message indicates that the query performs a table scan. The following query shows a typical table scan:

```
select au_lname, au_fname
from authors
QUERY PLAN FOR STATEMENT 1 (at line 1).

    STEP 1
        The type of query is SELECT.

        FROM TABLE
            authors
        Nested iteration.
        Table Scan.
        Forward scan.
        Positioning at start of table.
        Using I/O Size 16 Kbytes for data pages.
        With LRU Buffer Replacement Strategy for data
pages.
```

Clustered index message

```
Using Clustered Index.
```

This showplan message indicates that the query optimizer chose to use the clustered index on a table to retrieve the rows. The following query shows the clustered index being used to retrieve the rows from the table:

```
select title_id, title
from titles
where title_id like "T9%"
QUERY PLAN FOR STATEMENT 1 (at line 1).

    STEP 1
        The type of query is SELECT.

        FROM TABLE
            titles
        Nested iteration.
        Using Clustered Index.
        Index : title_id_ix
        Forward scan.
        Positioning by key.
        Keys are:
            title_id  ASC
        Using I/O Size 16 Kbytes for index leaf pages.
```

```
            With LRU Buffer Replacement Strategy for index
        leaf pages.
            Using I/O Size 16 Kbytes for data pages.
            With LRU Buffer Replacement Strategy for data
        pages.
```

Index name message

```
Index : indexname
```

This message indicates that the query is using an index to retrieve the rows. The message includes the index name.

If the line above this message in the output is "Using Clustered Index," the index is clustered; otherwise, the index is nonclustered.

The keys used to position the search are reported in the "Keys are..." message.

See "Keys message" on page 105.

This query illustrates the use of a nonclustered index to find and return rows:

```
    select au_id, au_fname, au_lname
    from authors
    where au_fname = "Susan"
QUERY PLAN FOR STATEMENT 1 (at line 1).

    STEP 1
        The type of query is SELECT.

        FROM TABLE
            authors
        Nested iteration.
        Index : au_names_ix
        Forward scan.
        Positioning by key.
        Keys are:
            au_fname  ASC
        Using I/O Size 16 Kbytes for index leaf pages.
        With LRU Buffer Replacement Strategy for index
    leaf pages.
        Using I/O Size 2 Kbytes for data pages.
        With LRU Buffer Replacement Strategy for data
    pages.
```

Scan direction messages

```
Forward scan.

Backward scan.
```

These messages indicate the direction of a table or index scan.

The scan direction depends on the ordering specified when the indexes were created and the order specified for columns in the order by clause.

Backward scans cam be used when the order by clause contains the asc or desc qualifiers on index keys, in the exact opposite of those in the create index clause. The configuration parameter allow backward scans must be set to 1 to allow backward scans.

The scan-direction messages are followed by positioning messages. Any keys used in the query are followed by “ASC” or “DESC”. The forward and backward scan messages and positioning messages describe whether a scan is positioned:

- At the first matching index key, at the start of the table, or at the first page of the leaf-level pages chain, and searching toward end of the index, or
- At the last matching index key, or end of the table, or last page of the leaf-level page chain, and searching toward the beginning.

If allow backward scans is set to 0, all accesses use forward scans.

This example uses a backward scan:

```
   select *
   from sysmessages
   where description like "%Optimized using%"
   order by error desc
QUERY PLAN FOR STATEMENT 1 (at line 1).

        STEP 1
            The type of query is SELECT.

            FROM TABLE
                sysmessages
            Nested iteration.
            Table Scan.
            Backward scan.
            Positioning at end of table.
            Using I/O Size 2 Kbytes for data pages.
            With LRU Buffer Replacement Strategy for data
   pages.
```

This query using the max aggregate also uses a backward scan:

```
   select max(error) from sysmessages
QUERY PLAN FOR STATEMENT 1 (at line 1).

        STEP 1
            The type of query is SELECT.
            Evaluate Ungrouped MAXIMUM AGGREGATE.

            FROM TABLE
                sysmessages
            Nested iteration.
            Index : ncsysmessages
            Backward scan.
            Positioning by key.
            Scanning only up to the first qualifying row.
            Index contains all needed columns. Base table
    will not be read.
            Keys are:
                error  ASC
            Using I/O Size 2 Kbytes for index leaf pages.
           With LRU Buffer Replacement Strategy for index
    leaf pages.

        STEP 2
            The type of query is SELECT.
```

Positioning messages

```
Positioning at start of table.

Positioning at end of table.

Positioning by Row IDentifier (RID).

Positioning by key.

Positioning at index start.

Positioning at index end.
```

These messages describe how access to a table or to the leaf level of an index takes place. The choices are:

```
Positioning at start of table.
```

Indicates a forward table scan, starting at the first row of the table.

```
Positioning at end of table.
```

Indicates a backward table scan, starting at the last row of the table.

```
Positioning by Row IDentifier (RID).
```

It is printed after the OR strategy has created a dynamic index of row IDs.

See "Dynamic index message (OR strategy)" on page 107 for more information about how row IDs are used.

```
Positioning by key.
```

Indicates that the index is used to position the search at the first qualifying row. It is printed for:

- Direct access an individual row in a point query
- Range queries that perform matching scans of the leaf level of an index
- Range queries that scan the data pages when there is a clustered index on an allpages-locked table
- Indexed accesses to inner tables in joins

```
Positioning at index start.
Positioning at index end.
```

These messages indicate a nonmatching index scan, used when the index covers the query. Matching scans are positioned by key.

Forward scans are positioned at the start of the index; backward scans are positioned at the end of the index.

Scanning messages

```
Scanning only the last page of the table.
```

This message indicates that a query containing an ungrouped (scalar) max aggregate can access only the last page of the table to return the value.

```
Scanning only up to the first qualifying row.
```

This message appears only for queries that use an ungrouped (scalar) min aggregate. The aggregated column needs to be the leading column in the index.

Note For indexes with the leading key created in descending order, the use of the messages for min and max aggregates is reversed:

min uses "Positioning at index end"

while max prints "Positioning at index start" and "Scanning only up to the first qualifying row."

See *Performance and Tuning Guide: Optimizing and Abstract Plans* for more information.

Index covering message

```
Index contains all needed columns. Base table will not
be read.
```

This message indicates that an index covers the query. It is printed both for matching and nonmatching scans. Other messages in showplan output help distinguish these access methods:

- A matching scan reports "Positioning by key."

  A nonmatching scan reports "Positioning at index start," or "Positioning at index end" since a nonmatching scan must read the entire leaf level of the index.

- If the optimizer uses a matching scan, the "Keys are..." message reports the keys used to position the search. This message is not included for a nonmatching scan.

The next query shows output for a matching scan, using a composite, nonclustered index on au_lname, au_fname, au_id:

```
    select au_fname, au_lname, au_id
    from authors
    where au_lname = "Williams"
QUERY PLAN FOR STATEMENT 1 (at line 1).

        STEP 1
            The type of query is SELECT.

            FROM TABLE
```

```
            authors
        Nested iteration.
        Index : au_names_id
        Forward scan.
        Positioning by key.
        Index contains all needed columns. Base table
will not be read.
        Keys are:
            au_lname  ASC
        Using I/O Size 2 Kbytes for index leaf pages.
        With LRU Buffer Replacement Strategy for index
leaf pages.
```

With the same composite index on au_lname, au_fname, au_id, this query performs a nonmatching scan, since the leading column of the index is not included in the where clause:

```
   select au_fname, au_lname, au_id
   from authors
   where au_id = "A93278"
QUERY PLAN FOR STATEMENT 1 (at line 1).

       STEP 1
        The type of query is SELECT.

        FROM TABLE
            authors
        Nested iteration.
        Index : au_names_id
        Forward scan.
        Positioning at index start.
        Index contains all needed columns. Base table
will not be read.
        Using I/O Size 16 Kbytes for index leaf pages.
        With LRU Buffer Replacement Strategy for index
leaf pages.
```

Note that the showplan output does not contain a "Keys are..." message, and the positioning message is "Positioning at index start." This query scans the entire leaf level of the nonclustered index, since the rows are not ordered by the search argument.

Keys message

```
Keys are:
```

```
key [ ASC | DESC ] ...
```

This message is followed by the index key(s) used when Adaptive Server uses an index scan to locate rows. The index ordering is printed after each index key, showing the order, ASC for ascending or DESC for descending, used when the index was created. For composite indexes, all leading keys in the where clauses are listed.

Matching index scans message

```
Using N Matching Index Scans.
```

This showplan message indicates that a query using or clauses or an in (*values list*) clause uses multiple index scans (also called the "special OR strategy") instead of using a dynamic index.

Multiple matching scans can be used only when there is no possibility that the or clauses or in list items will match duplicate rows – that is, when there is no need to build the worktable and perform the sort to remove the duplicates.

For more information on how queries containing or are processed, see *Performance and Tuning Guide: Optimizer*.

For queries that use multiple matching scans, different indexes may be used for some of the scans, so the messages that describe the type of index, index positioning, and keys used are printed for each scan.

The following example uses multiple matching index scans to return rows:

```
    select title
        from titles
        where title_id in ("T18168","T55370")
QUERY PLAN FOR STATEMENT 1 (at line 1).

     STEP 1
         The type of query is SELECT.

         FROM TABLE
             titles
         Nested iteration.
         Using 2 Matching Index Scans
         Index : title_id_ix
         Forward scan.
         Positioning by key.
         Keys are:
             title_id
```

```
        Index : title_id_ix
        Forward scan.
        Positioning by key.
        Keys are:
            title_id
        Using I/O Size 2 Kbytes for data pages.
      With LRU Buffer Replacement Strategy for data pages.
```

Dynamic index message (OR strategy)

```
Using Dynamic Index.
```

The term *dynamic index* refers to a worktable of row IDs used to process some queries that use or clauses or an in (values list) clause. When the OR strategy is used, Adaptive Server builds a list of all the row IDs that match the query, sorts the list to remove duplicates, and uses the list to retrieve the rows from the table.

For a full explanation, see *Performance and Tuning Guide: Optimizer.*

For a query with two SARGs that match the two indexes (one on au_fname, one on au_lname), the showplan output below includes three "FROM TABLE" sections:

- The first two "FROM TABLE" blocks in the output show the two index accesses, one for the first name "William" and one for the last name "Williams".

  These blocks include the output "Index contains all needed columns," since the row IDs can be retrieved from the leaf level of a nonclustered index.

- The final "FROM TABLE" block shows the "Using Dynamic Index" output and "Positioning by Row IDentifier (RID)."

  In this step, the dynamic index is used to access the data pages to locate the rows to be returned.

```
    select au_id, au_fname, au_lname
    from authors
    where au_fname = "William"
        or au_lname = "Williams"
QUERY PLAN FOR STATEMENT 1 (at line 1).

    STEP 1
        The type of query is SELECT.
```

```
FROM TABLE
    authors
Nested iteration.
Index : au_fname_ix
Forward scan.
Positioning by key.
Index contains all needed columns. Base table will not be read.
Keys are:
    au_fname  ASC
Using I/O Size 2 Kbytes for index leaf pages.
With LRU Buffer Replacement Strategy for index leaf pages.

FROM TABLE
    authors
Nested iteration.
Index : au_lname_ix
Forward scan.
Positioning by key.
Index contains all needed columns. Base table will not be read.
Keys are:
    au_lname  ASC
Using I/O Size 2 Kbytes for index leaf pages.
With LRU Buffer Replacement Strategy for index leaf pages.

FROM TABLE
    authors
Nested iteration.
Using Dynamic Index.
Forward scan.
Positioning by Row IDentifier (RID).
Using I/O Size 2 Kbytes for data pages.
With LRU Buffer Replacement Strategy for data pages.
```

Reformatting Message

```
WorktableN Created for REFORMATTING.
```

When joining two or more tables, Adaptive Server may choose to use a reformatting strategy to join the tables when the tables are large and the tables in the join do not have a useful index.

The reformatting strategy:

- Inserts the needed columns from qualifying rows of the smaller of the two tables into a worktable.

- Creates a clustered index on the join column(s) of the worktable. The index is built using keys to join the worktable to the other table in the query.
- Uses the clustered index in the join to retrieve the qualifying rows from the table.

See *Performance and Tuning Guide: Optimizer* for more information on reformatting.

The following example illustrates the reformatting strategy. It performs a three-way join on the titles, titleauthor, and titles tables. There are no indexes on the join columns in the tables (au_id and title_id), so Adaptive Server uses the reformatting strategy on two of the tables:

```
    select au_lname, title
    from authors a, titleauthor ta, titles t
    where a.au_id = ta.au_id
    and t.title_id = ta.title_id
QUERY PLAN FOR STATEMENT 1 (at line 1).


STEP 1
        The type of query is INSERT.
        The update mode is direct.
        Worktable1 created for REFORMATTING.

        FROM TABLE
            titleauthor
            ta
        Nested iteration.
        Table Scan.
        Forward scan.
        Positioning at start of table.
        Using I/O Size 2 Kbytes for data pages.
        With LRU Buffer Replacement Strategy for data pages.
        TO TABLE
            Worktable1.

    STEP 2
        The type of query is INSERT.
        The update mode is direct.
        Worktable2 created for REFORMATTING.

        FROM TABLE
            authors
            a
```

```
        Nested iteration.
        Table Scan.
        Forward scan.
        Positioning at start of table.
        Using I/O Size 2 Kbytes for data pages.
        With LRU Buffer Replacement Strategy for data pages.
        TO TABLE
            Worktable2.

    STEP 3
        The type of query is SELECT.

        FROM TABLE
            titles
            t
        Nested iteration.
        Table Scan.
        Forward scan.
        Positioning at start of table.
        Using I/O Size 2 Kbytes for data pages.
        With LRU Buffer Replacement Strategy for data pages.

        FROM TABLE
            Worktable1.
        Nested iteration.
        Using Clustered Index.
        Forward scan.
        Positioning by key.
        Using I/O Size 2 Kbytes for data pages.
        With LRU Buffer Replacement Strategy for data pages.

        FROM TABLE
            Worktable2.
        Nested iteration.
        Using Clustered Index.
        Forward scan.
        Positioning by key.
        Using I/O Size 2 Kbytes for data pages.
        With LRU Buffer Replacement Strategy for data pages.
```

This query was run with set sort_merge off. When sort-merge joins are enabled, this query chooses a sort-merge join instead.

Trigger Log Scan Message

```
                Log Scan.
```

When an insert, update, or delete statement causes a trigger to fire, and the trigger includes access to the inserted or deleted tables, these tables are built by scanning the transaction log.

This example shows the output for the update to the titles table when this insert fires the totalsales_trig trigger on the salesdetail table:

```
    insert salesdetail values ('7896', '234518', 'TC3218',
    75, 40)
QUERY PLAN FOR STATEMENT 1 (at line 1).

    STEP 1
        The type of query is UPDATE.
        The update mode is direct.

        FROM TABLE
            titles
        Nested iteration.
        Table Scan.
        Forward scan.
        Positioning at start of table.
        Using I/O Size 2 Kbytes for data pages.
        With LRU Buffer Replacement Strategy for data pages.

        FROM TABLE
            salesdetail
        EXISTS TABLE : nested iteration.
        Log Scan.
        Forward scan.
        Positioning at start of table.

        Run subquery 1 (at nesting level 1).
        Using I/O Size 2 Kbytes for data pages.
        With LRU Buffer Replacement Strategy for data pages.
        TO TABLE
            titles

NESTING LEVEL 1 SUBQUERIES FOR STATEMENT 4.

  QUERY PLAN FOR SUBQUERY 1 (at nesting level 1 and at line 23).

    Correlated Subquery.
    Subquery under an EXPRESSION predicate.
```

```
STEP 1
    The type of query is SELECT.
    Evaluate Ungrouped SUM OR AVERAGE AGGREGATE.

    FROM TABLE
        salesdetail
    Nested iteration.
    Log Scan.
    Forward scan.
    Positioning at start of table.
    Using I/O Size 2 Kbytes for data pages.
    With MRU Buffer Replacement Strategy for data pages.
```

I/O Size Messages

```
Using I/O size N Kbtyes for data pages.

Using I/O size N Kbtyes for index leaf pages.
```

The messages report the I/O sizes used in the query. The possible sizes are 2K, 4K, 8K, and 16K.

If the table, index, LOB object, or database used in the query uses a data cache with large I/O pools, the optimizer can choose large I/O. It can choose to use one I/O size for reading index leaf pages, and a different size for data pages. The choice depends on the pool size available in the cache, the number of pages to be read, the cache bindings for the objects, and the cluster ratio for the table or index pages.

See Chapter 14, “Memory Use and Performance,” for more information on large I/O and the data cache.

Cache strategy messages

```
With <LRU/MRU> Buffer Replacement Strategy for data
pages.

With <LRU/MRU> Buffer Replacement Strategy for index
leaf pages.
```

These messages indicate the cache strategy used for data pages and for index leaf pages.

See "Overview of cache strategies" on page 180 for more information on cache strategies.

Total estimated I/O cost message

```
Total estimated I/O cost for statement N (at line N): X.
```

Adaptive Server prints this message only if a System Administrator has configured Adaptive Server to enable resource limits. Adaptive Server prints this line once for each query in a batch. The message displays the optimizer's estimate of the total cost of logical and physical I/O. If the query runs in parallel, the cost per thread is printed. System Administrators can use this value when setting compile-time resource limits.

See "Total actual I/O cost value" on page 780 for information on how cost is computed

If you are using dbcc traceon(310), this value is the sum of the values in the FINAL PLAN output for the query.

The following example demonstrates showplan output for an Adaptive Server configured to allow resource limits:

```
    select au_lname, au_fname
    from authors
    where city = "Oakland"
QUERY PLAN FOR STATEMENT 1 (at line 1).

    STEP 1
        The type of query is SELECT.

        FROM TABLE
            authors
        Nested iteration.
        Table Scan.
        Forward scan.
        Positioning at start of table.
        Using I/O Size 16 Kbytes for data pages.
        With LRU Buffer Replacement Strategy for data pages.

Total estimated I/O cost for statement 1 (at line 1): 1160.
```

For more information on creating resource limits, see in the *System Administration Guide*.

showplan messages for parallel queries

showplan reports information about parallel execution, showing which query steps are executed in parallel.

showplan reports information about parallel execution, explicitly stating which query steps are executed in parallel.

Table 5-2: showplan messages for parallel queries

Message	Explanation
Executed in parallel by coordinating process and N worker processes.	Indicates that a query is run in parallel, and shows the number of worker processes used.
Executed in parallel by N worker processes.	Indicates the number of worker processes used for a query step.
Executed in parallel with a N-way hash scan. Executed in parallel with a N-way partition scan.	Indicates the number of worker processes and the type of scan, hash-based of partition-based, for a query step.
Parallel work table merge. Parallel network buffer merge. Parallel result buffer merge.	Indicates the way in which the results of parallel scans were merged.
Parallel data merge using N worker processes.	Indicates that a merge join used a parallel data merge, and the number of worker processes used.
Serial data merge.	Indicates that the merge join used a serial data merge.
AN ADJUSTED QUERY PLAN WILL BE USED FOR STATEMENT N BECAUSE NOT ENOUGH WORKER PROCESSES ARE AVAILABLE AT THIS TIME. ADJUSTED QUERY PLAN:	Indicates that a run-time adjustment to the number of worker processes was required.

Executed in parallel messages

The Adaptive Server optimizer uses parallel query optimization strategies only when a given query is eligible for parallel execution. If the query is processed in parallel, showplan uses three separate messages to report:

- The fact that some or all of the query was executed by the coordinating process and worker processes. The number of worker processes is included in this message.

- The number of worker processes for each step of the query that is executed in parallel.
- The degree of parallelism for each scan.

Note that the degree of parallelism used for a query step is not the same as the total number of worker processes used for the query.

For more examples of parallel query plans, see Chapter 7, “Parallel Query Optimization.”

Coordinating process message

```
Executed in parallel by coordinating process and N worker processes.
```

For each query that runs in parallel mode, showplan reports prints this message, indicating the number of worker processes used.

Worker processes message

```
Executed in parallel by N worker processes.
```

For each step in a query that is executed in parallel, showplan reports the number of worker processes for the step following the “Type of query” message.

Scan type message

```
Executed in parallel with a N-way  hash scan.
Executed in parallel with a N-way partition scan.
```

For each step in the query that accesses data in parallel, showplan prints the number of worker processes used for the scan, and the type of scan, either “hash” or “partition.”

Merge messages

Results from the worker processes that process a query are merged using one of the following types of merge:

- Parallel worktable merge
- Parallel network buffer merge
- Parallel result buffer merge

Merge message for worktables

```
Parallel work table merge.
```

Grouped aggregate results from the worktables created by each worker process are merged into one result set.

In the following example, titles has two partitions. The showplan information specific to parallel query processing appears in bold.

```
   select type, sum(total_sales)
       from titles
       group by type
QUERY PLAN FOR STATEMENT 1 (at line 1).

    STEP 1
        The type of query is SELECT (into Worktable1).
        GROUP BY
        Evaluate Grouped SUM OR AVERAGE AGGREGATE.
       Executed in parallel by coordinating process and 2 worker
processes.

        FROM TABLE
            titles
        Nested iteration.
        Table Scan.
        Forward scan.
        Positioning at start of table.
        Executed in parallel with a 2-way partition scan.
        Using I/O Size 16 Kbytes for data pages.
        With LRU Buffer Replacement Strategy for data pages.
        TO TABLE
            Worktable1.

        Parallel work table merge.

    STEP 2
        The type of query is SELECT.
        Executed by coordinating process.

        FROM TABLE
            Worktable1.
        Nested iteration.
        Table Scan.
        Forward scan.
        Positioning at start of table.
        Using I/O Size 16 Kbytes for data pages.
        With MRU Buffer Replacement Strategy for data pages.
```

See “Merge join messages” on page 824 for an example that uses parallel processing to perform sort-merge joins.

Merge message for buffer merges

```
Parallel network buffer merge.
```

Unsorted, non aggregate results returned by the worker processes are merged into a network buffer that is sent to the client. In the following example, titles has two partitions.

```
    select title_id from titles
QUERY PLAN FOR STATEMENT 1 (at line 1).
Executed in parallel by coordinating process and 2 worker processes.

    STEP 1
        The type of query is SELECT.
        Executed in parallel by coordinating process and 2 worker
processes.

        FROM TABLE
            titles
        Nested iteration.
        Table Scan.
        Forward scan.
        Positioning at start of table.
        Executed in parallel with a 2-way partition scan.
        Using I/O Size 16 Kbytes for data pages.
        With LRU Buffer Replacement Strategy for data pages.

        Parallel network buffer merge.
```

Merge message for result buffers

```
Parallel result buffer merge.
```

Ungrouped aggregate results or unsorted, non aggregate variable assignment results from worker processes are merged.

Each worker process stores the aggregate in a result buffer. The result buffer merge produces a single value, ranging from zero-length (when the value is NULL) to the maximum length of a character string.

In the following example, titles has two partitions:

```
    select sum(total_sales)
    from titles
QUERY PLAN FOR STATEMENT 1 (at line 1).
```

```
Executed in parallel by coordinating process and 2 worker
processes.

    STEP 1
        The type of query is SELECT.
        Evaluate Ungrouped SUM OR AVERAGE AGGREGATE.
       Executed in parallel by coordinating process and 2 worker
processes.

        FROM TABLE
            titles
        Nested iteration.
        Table Scan.
        Forward scan.
        Positioning at start of table.
        Executed in parallel with a 2-way partition scan.
        Using I/O Size 16 Kbytes for data pages.
        With LRU Buffer Replacement Strategy for data pages.

        Parallel result buffer merge.

    STEP 2
        The type of query is SELECT.
        Executed by coordinating process.
```

Data merge messages

```
Parallel data merge using N worker processes.

Serial data merge.
```

The data merge messages indicate whether a serial or parallel data merge was performed. If the merge is performed in parallel mode, the number of worker processes is also printed.

For sample output, see “Merge join messages” on page 95“.

Runtime adjustment message

```
AN ADJUSTED QUERY PLAN WILL BE USED FOR STATEMENT N BECAUSE NOT
ENOUGH WORKER PROCESSES ARE AVAILABLE AT THIS TIME.
ADJUSTED QUERY PLAN:
```

showplan output displays this message and an adjusted query plan when fewer worker processes are available at runtime than the number specified by the optimized query plan.

showplan messages for subqueries

Since subqueries can contain the same clauses that regular queries contain, their showplan output can include many of the messages listed in earlier sections.

The showplan messages for subqueries, shown in "Subquery optimization" on page 131 in the *Performance and Tuning: Optimizer*, include delimiters so that you can spot the beginning and the end of a subquery processing block, the messages that identify the type of subquery, the place in the outer query where the subquery is executed, and messages for special types of processing that is performed only in subqueries.

The showplan messages for subqueries include special delimiters that allow you to easily spot the beginning and end of a subquery processing block, messages to identify the type of subquery, the place in the outer query where the subquery is executed, or special types of processing performed only in subqueries

Table 5-3: showplan messages for subqueries

Message	Explanation
Run subquery N (at nesting level N).	This message appears at the point in the query where the subquery actually runs. Subqueries are numbered in order for each side of a union.
NESTING LEVEL N SUBQUERIES FOR STATEMENT N.	Shows the nesting level of the subquery.
QUERY PLAN FOR SUBQUERY N (at nesting level N and at line N). END OF QUERY PLAN FOR SUBQUERY N.	These lines bracket showplan output for each subquery in a statement. Variables show the subquery number, the nesting level, and the input line.
Correlated Subquery.	The subquery is correlated.
Non-correlated Subquery.	The subquery is not correlated.
Subquery under an IN predicate.	The subquery is introduced by in.
Subquery under an ANY predicate.	The subquery is introduced by any.
Subquery under an ALL predicate.	The subquery is introduced by all.
Subquery under an EXISTS predicate.	The subquery is introduced by exists.
Subquery under an EXPRESSION predicate.	The subquery is introduced by an expression, or the subquery is in the select list.

Message	Explanation
Evaluate Grouped ANY AGGREGATE. Evaluate Grouped ONCE AGGREGATE. Evaluate Grouped ONCE-UNIQUE AGGREGATE. or Evaluate Ungrouped ANY AGGREGATE. Evaluate Ungrouped ONCE AGGREGATE. Evaluate Ungrouped ONCE-UNIQUE AGGREGATE.	The subquery uses an internal aggregate.
EXISTS TABLE: nested iteration	The query includes an exists, in, or any clause, and the subquery is flattened into a join.

For information about how Adaptive Server optimizes certain types of subqueries by materializing results or by flattening the queries to joins, see "Subquery optimization" on page 131 in the *Performance and Tuning: Optimizer*.

For basic information on subqueries, subquery types, and the meaning of the subquery predicates, see the *Transact-SQL User's Guide*.

Output for flattened or materialized subqueries

Certain forms of subqueries can be processed more efficiently when:

- The query is flattened into a join query, or
- The subquery result set is materialized as a first step, and the results are used in a second step with the rest of the outer query.

When the optimizer chooses one of these strategies, the query is not processed as a subquery, so you will not see the subquery message delimiters. The following sections describe showplan output for flattened and materialized queries.

Flattened queries

Adaptive Server can use one of several methods to flatten subqueries into joins.

These methods are described in "Flattening in, any, and exists subqueries" on page 145.

Subqueries executed as existence joins

When subqueries are flattened into existence joins, the output looks like normal showplan output for a join, with the possible exception of the message "EXISTS TABLE: nested iteration."

This message indicates that instead of the normal join processing, which looks for every row in the table that matches the join column, Adaptive Server uses an existence join and returns TRUE as soon as the first qualifying row is located.

For more information on subquery flattening, see "Flattened subqueries executed as existence joins" on page 148.

Adaptive Server flattens the following subquery into an existence join:

```
    select title
    from titles
    where  title_id in
        (select title_id
            from titleauthor)
    and title like "A Tutorial%"
QUERY PLAN FOR STATEMENT 1 (at line 1).

    STEP 1
        The type of query is SELECT.

        FROM TABLE
            titles
        Nested iteration.
        Index : title_ix
        Forward scan.
        Positioning by key.
        Keys are:
            title  ASC
        Using I/O Size 16 Kbytes for index leaf pages.
        With LRU Buffer Replacement Strategy for index leaf pages.
        Using I/O Size 2 Kbytes for data pages.
        With LRU Buffer Replacement Strategy for data pages.

        FROM TABLE
            titleauthor
        EXISTS TABLE : nested iteration.
        Index : ta_ix
        Forward scan.
        Positioning by key.
      Index contains all needed columns. Base table will not be read.
        Keys are:
```

```
        title_id  ASC
    Using I/O Size 2 Kbytes for index leaf pages.
    With LRU Buffer Replacement Strategy for index leaf pages.
```

Subqueries using unique reformatting

If there is not a unique index on publishers.pub_id, this query is flattened by selecting the rows from publishers into a worktable and then creating a unique clustered index. This process is called unique reformatting:

```
    select title_id
    from titles
    where pub_id in
    (select pub_id from publishers where state = "TX")
QUERY PLAN FOR STATEMENT 1 (at line 1).


    STEP 1
        The type of query is INSERT.
        The update mode is direct.
        Worktable1 created for REFORMATTING.

        FROM TABLE
            publishers
        Nested iteration.
        Table Scan.
        Forward scan.
        Positioning at start of table.
        Using I/O Size 2 Kbytes for data pages.
        With LRU Buffer Replacement Strategy for data pages.
        TO TABLE
            Worktable1.

    STEP 2
        The type of query is SELECT.

        FROM TABLE
            Worktable1.
        Nested iteration.
        Using Clustered Index.
        Forward scan.
        Positioning at start of table.
        Using I/O Size 2 Kbytes for data pages.
        With LRU Buffer Replacement Strategy for data pages.

        FROM TABLE
            titles
```

```
Nested iteration.
Table Scan.
Forward scan.
Positioning at start of table.
Using I/O Size 2 Kbytes for data pages.
With LRU Buffer Replacement Strategy for data pages.
```

For more information, see "Flattened subqueries executed using unique reformatting" on page 135 in the *Performance and Tuning: Optimizer*.

Subqueries using duplicate elimination

This query performs a regular join, selecting all of the rows into a worktable. In the second step, the worktable is sorted to remove duplicates. This process is called duplicate elimination:

```
    select  title_id, au_id, au_ord
    from titleauthor ta
    where title_id in (select ta.title_id
        from titles t, salesdetail sd
        where t.title_id = sd.title_id
        and ta.title_id = t.title_id
        and type = 'travel' and qty > 10)
QUERY PLAN FOR STATEMENT 1 (at line 1).

    STEP 1
        The type of query is INSERT.
        The update mode is direct.
        Worktable1 created for DISTINCT.

        FROM TABLE
            salesdetail
            sd
        Nested iteration.
        Table Scan.
        Forward scan.
        Positioning at start of table.
        Using I/O Size 16 Kbytes for data pages.
        With LRU Buffer Replacement Strategy for data pages.

        FROM TABLE
            titles
            t
        Nested iteration.
        Using Clustered Index.
        Index : title_id_ix
        Forward scan.
```

```
    Positioning by key.
    Keys are:
        title_id  ASC
    Using I/O Size 2 Kbytes for data pages.
    With LRU Buffer Replacement Strategy for data pages.

    FROM TABLE
        titleauthor
        ta
    Nested iteration.
    Index : ta_ix
    Forward scan.
    Positioning by key.
    Keys are:
        title_id  ASC
    Using I/O Size 2 Kbytes for index leaf pages.
    With LRU Buffer Replacement Strategy for index leaf pages.
    Using I/O Size 2 Kbytes for data pages.
    With LRU Buffer Replacement Strategy for data pages.
    TO TABLE
        Worktable1.

STEP 2
    The type of query is SELECT.
    This step involves sorting.

    FROM TABLE
        Worktable1.
    Using GETSORTED
    Table Scan.
    Forward scan.
    Positioning at start of table.
    Using I/O Size 16 Kbytes for data pages.
    With MRU Buffer Replacement Strategy for data pages.
```

Materialized queries

When Adaptive Server materializes subqueries, the query is executed in two steps:

1. The first step stores the results of the subquery in an internal variable or worktable.
2. The second step uses the internal variable or worktable results in the outer query.

This query materializes the subquery into a worktable:

```
                select type, title_id
                from titles
                where total_sales in (select max(total_sales)
                            from sales_summary
                            group by type)
QUERY PLAN FOR STATEMENT 1 (at line 1).

    STEP 1
        The type of query is SELECT (into Worktable1).
        GROUP BY
        Evaluate Grouped MAXIMUM AGGREGATE.

        FROM TABLE
            sales_summary
        Nested iteration.
        Table Scan.
        Forward scan.
        Positioning at start of table.
        Using I/O Size 2 Kbytes for data pages.
        With LRU Buffer Replacement Strategy for data pages.
        TO TABLE
            Worktable1.

    STEP 2
        The type of query is SELECT.

        FROM TABLE
            titles
        Nested iteration.
        Table Scan.
        Forward scan.
        Positioning at start of table.
        Using I/O Size 16 Kbytes for data pages.
        With LRU Buffer Replacement Strategy for data pages.

        FROM TABLE
            Worktable1.
        EXISTS TABLE : nested iteration.
        Table Scan.
        Forward scan.
        Positioning at start of table.
        Using I/O Size 16 Kbytes for data pages.
        With MRU Buffer Replacement Strategy for data pages.
```

The showplan message "EXISTS TABLE: nested iteration," near the end of the output, shows that Adaptive Server performs an existence join.

Structure of subquery *showplan* output

When a query contains subqueries that are not flattened or materialized:

- The showplan output for the outer query appears first. It includes the message "Run subquery *N* (at nesting level *N*)", indicating the point in the query processing where the subquery executes.
- For each nesting level, the query plans at that nesting level are introduced by the message "NESTING LEVEL *N* SUBQUERIES FOR STATEMENT *N*."
- The plan for each subquery is introduced by the message "QUERY PLAN FOR SUBQUERY *N* (at nesting level *N* and at line *N*)", and the end of its plan is marked by the message "END OF QUERY PLAN FOR SUBQUERY *N*." This section of the output includes information showing:
  - The type of query (correlated or uncorrelated)
  - The predicate type (IN, ANY, ALL, EXISTS, or EXPRESSION)

Subquery execution message

```
Run subquery N (at nesting level N).
```

This message shows the place where the subquery execution takes place in the execution of the outer query. Adaptive Server executes the subquery at the point in the outer query where it need to be run least often.

The plan for this subquery appears later in the output for the subquery's nesting level. The first variable in this message is the subquery number; the second variable is the subquery nesting level.

Nesting level delimiter message

```
NESTING LEVEL N SUBQUERIES FOR STATEMENT N.
```

This message introduces the showplan output for all the subqueries at a given nesting level. The maximum nesting level is 16.

Subquery plan start delimiter

```
QUERY PLAN FOR SUBQUERY N (at nesting level N and at line N).
```

This statement introduces the showplan output for a particular subquery at the nesting level indicated by the previous NESTING LEVEL message.

Line numbers to help you match showplan output to your input.

Subquery plan end delimiter

```
END OF QUERY PLAN FOR SUBQUERY N.
```

This statement marks the end of the query plan for a particular subquery.

Type of subquery

```
Correlated Subquery.

Non-correlated Subquery.
```

A subquery is either correlated or non correlated.

- A correlated subquery references a column in a table that is listed in the from list of the outer query. If the subquery is correlated, showplan includes the message "Correlated Subquery."
- A non correlated subquery can be evaluated independently of the outer query. Non correlated subqueries are sometimes materialized, so their showplan output does not include the normal subquery showplan messages.

Subquery predicates

```
Subquery under an IN predicate.

Subquery under an ANY predicate.

Subquery under an ALL predicate.

Subquery under an EXISTS predicate.

Subquery under an EXPRESSION predicate.
```

Subqueries introduced by in, any, all, or exists are quantified predicate subqueries. Subqueries introduced by >, >=, <, <=, =, != are expression subqueries.

Internal subquery aggregates

Certain types of subqueries require special internal aggregates, as listed in Table 5-4. Adaptive Server generates these aggregates internally – they are not part of Transact-SQL syntax and cannot be included in user queries.

Table 5-4: Internal subquery aggregates

Subquery type	Aggregate	Effect
Quantified predicate	ANY	Returns TRUE or FALSE to the outer query.
Expression	ONCE	Returns the result of the subquery. Raises error 512 if the subquery returns more than one value.
Subquery containing distinct	ONCE-UNIQUE	Stores the first subquery result internally and compares each subsequent result to the first. Raises error 512 if a subsequent result differs from the first.

Messages for internal aggregates include “Grouped” when the subquery includes a group by clause and computes the aggregate for a group of rows, otherwise the messages include “Ungrouped”; the subquery the aggregate for all rows in the table that satisfy the correlation clause.

Quantified predicate subqueries and the ANY aggregate

```
Evaluate Grouped ANY AGGREGATE.

Evaluate Ungrouped ANY AGGREGATE.
```

All quantified predicate subqueries that are not flattened use the internal ANY aggregate. Do not confuse this with the any predicate that is part of SQL syntax.

The subquery returns TRUE when a row from the subquery satisfies the conditions of the subquery predicate. It returns FALSE to indicate that no row from the subquery matches the conditions.

For example:

```
select type, title_id
from titles
where price > all
    (select price
        from titles
        where advance < 15000)
```

```
QUERY PLAN FOR STATEMENT 1 (at line 1).

    STEP 1
        The type of query is SELECT.

        FROM TABLE
            titles
        Nested iteration.
        Table Scan.
        Forward scan.
        Positioning at start of table.

        Run subquery 1 (at nesting level 1).
        Using I/O Size 16 Kbytes for data pages.
        With LRU Buffer Replacement Strategy for data pages.

NESTING LEVEL 1 SUBQUERIES FOR STATEMENT 1.

  QUERY PLAN FOR SUBQUERY 1 (at nesting level 1 and at line 4).

    Correlated Subquery.
    Subquery under an ALL predicate.

    STEP 1
        The type of query is SELECT.
        Evaluate Ungrouped ANY AGGREGATE.

        FROM TABLE
            titles
        EXISTS TABLE : nested iteration.
        Table Scan.
        Forward scan.
        Positioning at start of table.
        Using I/O Size 16 Kbytes for data pages.
        With LRU Buffer Replacement Strategy for data pages.

  END OF QUERY PLAN FOR SUBQUERY 1.
```

Expression subqueries and the ONCE aggregate

```
Evaluate Ungrouped ONCE AGGREGATE.

Evaluate Grouped ONCE AGGREGATE.
```

Expression subqueries return only a single value. The internal ONCE aggregate checks for the single result required by an expression subquery.

This query returns one row for each title that matches the like condition:

```
   select title_id, (select city + " " + state
                        from publishers
                        where pub_id = t.pub_id)
   from titles t
   where title like "Computer%"
QUERY PLAN FOR STATEMENT 1 (at line 1).

    STEP 1
        The type of query is SELECT.

        FROM TABLE
            titles
            t
        Nested iteration.
        Index : title_ix
        Forward scan.
        Positioning by key.
        Keys are:
            title  ASC

        Run subquery 1 (at nesting level 1).
        Using I/O Size 16 Kbytes for index leaf pages.
       With LRU Buffer Replacement Strategy for index leaf pages.
        Using I/O Size 2 Kbytes for data pages.
        With LRU Buffer Replacement Strategy for data pages.

NESTING LEVEL 1 SUBQUERIES FOR STATEMENT 1.

  QUERY PLAN FOR SUBQUERY 1 (at nesting level 1 and at line 1).

    Correlated Subquery.
    Subquery under an EXPRESSION predicate.


    STEP 1
        The type of query is SELECT.
        Evaluate Ungrouped ONCE AGGREGATE.

        FROM TABLE
            publishers
        Nested iteration.
        Table Scan.
        Forward scan.
        Positioning at start of table.
        Using I/O Size 2 Kbytes for data pages.
```

```
            With LRU Buffer Replacement Strategy for data pages.

        END OF QUERY PLAN FOR SUBQUERY 1.
```

Subqueries with *distinct* and the ONCE-UNIQUE aggregate

```
                Evaluate Grouped ONCE-UNIQUE AGGREGATE.

                Evaluate Ungrouped ONCE-UNIQUE AGGREGATE.
```

When the subquery includes distinct, the ONCE-UNIQUE aggregate indicates that duplicates are being eliminated:

```
            select pub_name from publishers
            where pub_id =
            (select distinct titles.pub_id from titles
                where publishers.pub_id = titles.pub_id
                and price > $1000)
QUERY PLAN FOR STATEMENT 1 (at line 1).

    STEP 1
        The type of query is SELECT.

        FROM TABLE
            publishers
        Nested iteration.
        Table Scan.
        Forward scan.
        Positioning at start of table.

        Run subquery 1 (at nesting level 1).
        Using I/O Size 2 Kbytes for data pages.
        With LRU Buffer Replacement Strategy for data pages.

NESTING LEVEL 1 SUBQUERIES FOR STATEMENT 1.

  QUERY PLAN FOR SUBQUERY 1 (at nesting level 1 and at line 3).

    Correlated Subquery.
    Subquery under an EXPRESSION predicate.


    STEP 1
        The type of query is SELECT.
        Evaluate Ungrouped ONCE-UNIQUE AGGREGATE.

        FROM TABLE
```

```
            titles
        Nested iteration.
        Index : pub_id_ix
        Forward scan.
        Positioning by key.
        Keys are:
            pub_id  ASC
        Using I/O Size 16 Kbytes for index leaf pages.
        With LRU Buffer Replacement Strategy for index leaf pages.
        Using I/O Size 2 Kbytes for data pages.
        With LRU Buffer Replacement Strategy for data pages.

END OF QUERY PLAN FOR SUBQUERY 1.
```

Existence join message

```
    EXISTS TABLE: nested iteration
```

This message indicates a special form of nested iteration. In a regular nested iteration, the entire table or its index is searched for qualifying values.

In an existence test, the query can stop the search as soon as it finds the first matching value.

The types of subqueries that can produce this message are:

- Subqueries that are flattened to existence joins
- Subqueries that perform existence tests

Subqueries that perform existence tests

There are several ways you can write queries that perform an existence test, for example, using exists, in, or =any. These queries are treated as if they were written with an exists clause. The following example shows an existence test. This query cannot be flattened because the outer query contains or:

```
    select au_lname, au_fname
    from authors
    where exists
        (select *
        from publishers
        where authors.city = publishers.city)
    or city = "New York"
QUERY PLAN FOR STATEMENT 1 (at line 1).
```

```
    STEP 1
        The type of query is SELECT.

        FROM TABLE
            authors
        Nested iteration.
        Table Scan.
        Forward scan.
        Positioning at start of table.

        Run subquery 1 (at nesting level 1).
        Using I/O Size 16 Kbytes for data pages.
        With LRU Buffer Replacement Strategy for data pages.

NESTING LEVEL 1 SUBQUERIES FOR STATEMENT 1.

  QUERY PLAN FOR SUBQUERY 1 (at nesting level 1 and at line 4).

    Correlated Subquery.
    Subquery under an EXISTS predicate.


    STEP 1
        The type of query is SELECT.
        Evaluate Ungrouped ANY AGGREGATE.

        FROM TABLE
            publishers
        EXISTS TABLE : nested iteration.
        Table Scan.
        Forward scan.
        Positioning at start of table.
        Using I/O Size 2 Kbytes for data pages.
        With LRU Buffer Replacement Strategy for data pages.

END OF QUERY PLAN FOR SUBQUERY 1.
```

CHAPTER 6

Statistics Tables and Displaying Statistics with *optdiag*

This chapter explains how statistics are stored and displayed.

For more information on managing statistics, see Chapter 3, "Using Statistics to Improve Performance."

System tables that store statistics

The systabstats and sysstatistics tables store statistics for all tables, indexes, and any unindexed columns for which you have explicitly created statistics. In general terms:

- systabstats stores information about the table or index as an object, that is, the size, number of rows, and so forth.

  It is updated by query processing, data definition language, and update statistics commands.

- sysstatistics stores information about the values in a specific column.

  It is updated by data definition language and update statistics commands.

For more information, see "Effects of SQL commands on statistics" on page 168.

systabstats table

The systabstats table contains basic statistics for tables and indexes, for example:

- Number of data pages for a table, or the number of leaf level pages for an index
- Number of rows in the table
- Height of the index
- Average length of data rows and leaf rows
- Number of forwarded and deleted rows
- Number of empty pages
- Statistics to increase the accuracy of I/O cost estimates, including cluster ratios, the number of pages that share an extent with an allocation page, and the number of OAM and allocation pages used for the object
- Stopping points for the reorg command so that it can resume processing

systabstats stores one row for each table and nonclustered index in the database. The storage for clustered index information depends on the locking scheme for the table:

- If the table is a data-only-locked table, systabstats stores an additional row for a clustered index.
- If the table is an allpages-locked table, the data pages are treated as the leaf level of the index, so the systabstats entry for a clustered index is stored in the same row as the table data.

  The indid column for clustered indexes on allpages-locked tables is always 1.

See the *Adaptive Server Reference Manual* for more information.

sysstatistics table

The sysstatistics table stores one or more rows for each indexed column on a user table. In addition, it can store statistics for unindexed columns.

The first row for each column stores basic statistics about the column, such as the density for joins and search arguments, the selectivity for some operators, and the number of steps stored in the histogram for the column. If the index has multiple columns, or if you specify multiple columns when you generate statistics for unindexed columns, there is a row for each prefix subset of columns.

For more information on prefix subsets, see "Column statistics" on page 146.

Additional rows store histogram data for the leading column. Histograms do not exist if indexes were created before any data was inserted into a table (run update statistics after inserting data to generate the histogram).

See "Histogram displays" on page 151 for more information.

See the *Adaptive Server Reference Manual* for more information.

Viewing statistics with the *optdiag* utility

The optdiag utility displays statistics from the systabstats and sysstatistics tables. optdiag can also be used to update sysstatistics information. Only a System Administrator can run optdiag.

optdiag syntax

The syntax for optdiag is:

```
optdiag [binary] [simulate] statistics
   {-i input_file |
   database[.owner[.[table[.column]]]]
      [-o output_file]}
   [-U username] [-P password]
   [-I interfaces_file]
   [-S server]
   [-v] [-h] [-s] [-Tflag_value]
   [-z language] [-J client_charset]
   [-a display_charset]
```

You can use optdiag to display statistics for an entire database, for a single table and its indexes and columns, or for a particular column.

To display statistics for all user tables in the pubtune database, placing the output in the *pubtune.opt* file, use the following command:

```
optdiag statistics pubtune -Usa -Ppasswd
-o pubtune.opt
```

This command displays statistics for the titles table and for any indexes on the table:

```
optdiag statistics pubtune..titles -Usa -Ppasswd
        -o titles.opt
```

See *Utility Programs Manual* for your platform for more information on the optdiag command. The following sections provide information about the output from optdiag.

optdiag header information

After printing the version information for optdiag and Adaptive Server, optdiag prints the server name and summarizes the arguments used to display the statistics.

The header of the optdiag report lists the objects described in the report:

```
Server name:                            "test_server"

Specified database:                     "pubtune"
Specified table owner:                  not specified
Specified table:                        "titles"
Specified column:                       not specified
```

Table 6-1 describes the output.

Table 6-1: Table and column information

Row Label	Information Provided
Server name	The name of the server, as stored in the *@@servername* variable. You must use sp_addserver, and restart the server for the server name to be available in the variable.
Specified database	Database name given on the optdiag command line.
Specified table owner	Table owner given on the optdiag command line.
Specified table	Table name given on the optdiag command line.
Specified column	Column name given on the optdiag command line.

Table statistics

This optdiag section reports basic statistics for the table.

Sample output for table statistics

```
Table owner:                          "dbo"

Statistics for table:                 "titles"

     Data page count:                 662
     Empty data page count:           10
     Data row count:                  4986.0000000000000000
     Forwarded row count:             18.0000000000000000
     Deleted row count:               87.0000000000000000
     Data page CR count:              86.0000000000000000
     OAM + allocation page count:     5
     First extent data pages:         3
     Data row size:                   238.8634175691937287

  Derived statistics:
     Data page cluster ratio:         0.9896907216494846
```

Table 6-2: Table statistics

Row label	Information provided
Table owner	Name of the table owner. You can omit owner names on the command line by specifying *dbname..tablename*. If multiple tables have the same name, and different owners, optdiag prints information for each table with that name.
Statistics for table	Name of the table.
Data page count	Number of data pages in the table.
Empty data page count	Count of pages that have deleted rows only.
Data row count	Number of data rows in the table.
Forwarded row count	Number of forwarded rows in the table. This value is always 0 for an allpages-locked table.
Deleted row count	Number of rows that have been deleted from the table. These are committed deletes where the space has not been reclaimed by one of the functions that clears deleted rows. This value is always 0 for an allpages-locked table.
Data page CR count	A counter used to derive the data page cluster ratio. See “Data page CR count” on page 140.
OAM + allocation page count	Number of OAM pages for the table, plus the number of allocation units in which the table occupies space. These statistics are used to estimate the cost of OAM scans on data-only-locked tables. The value is maintained only on data-only-locked tables.

Row label	Information provided
First extent data pages	Number of pages that share the first extent in an allocation unit with the allocation page. These pages need to be read using 2K I/O, rather than large I/O. This information is maintained only for data-only-locked tables.
Data row size	Average length of a data row, in bytes. The size includes row overhead. This value is updated only by update statistics, create index, and alter table...lock.
Index height	Height of the index, not counting the leaf level. This row is included in the table-level output only for clustered indexes on allpages-locked tables. For all other indexes, the index height appears in the index-level output. This value does not apply to heap tables.

Data page CR count

The "Data Page CR count" is used to compute the data page cluster ratio, which can help determine the effectiveness of large I/O for table scans and range scans. This value is updated only when you run update statistics.

Table-level derived statistics

The "Derived statistics" in the table-level section reports the statistics derived from the "Data Page CR count" and data page count. Table 6-3 describes the output.

Table 6-3: Cluster ratio for a table

Row label	Information provided
Data page cluster ratio	The data page cluster ratio is used to estimate the effectiveness of large I/O. It is used to estimate the number of I/Os required to read an allpages-locked table by following the page chain, and to estimate the number of large I/Os required to scan a data-only-locked table using an OAM scan.
Space utilization	The ratio of the minimum space usage for this table, and the current space usage.
Large I/O efficiency	Estimates the number of useful pages brought in by each large I/O.

Data page cluster ratio

For allpages-locked tables, the data page cluster ratio measures how well the pages are sequenced on extents, when the table is read in page-chain order. A cluster ratio of 1.0 indicates perfect sequencing. A lower cluster ratio indicates that the page chain is fragmented.

For data-only-locked tables, the data page cluster ratio measures how well the pages are packed on the extents. A cluster ratio of 1.0 indicates complete packing of extents. A low data page cluster ratio indicates that extents allocated to the table contain empty pages.

For an example of how the data page cluster ratio is used, see “How cluster ratios affect large I/O estimates” on page 69 in the *Performance and Tuning: Optimizer*.

Space utilization

Space utilization uses the average row size and number of rows to compute the expected minimum number of data pages, and compares it to the current number of pages. If space utilization is low, running reorg rebuild on the table or dropping and re-creating the clustered index can reduce the amount of empty space on data pages, and the number of empty pages in extents allocated to the table.

If you are using space management properties such as fillfactor or reservepagegap, the empty space that is left for additional rows on data pages of a table with a clustered index and the number of empty pages left in extents for the table affects the space utilization value.

If statistics have not been updated recently and the average row size has changed or the number of rows and pages are inaccurate, space utilization may report values greater than 1.0.

Large I/O efficiency

Large I/O efficiency estimates the number of useful pages brought in by each large I/O. For examples, if the value is.5, a 16K I/O returns, on average, 4 2K pages needed for the query, and 4 other pages, either empty pages or pages that share the extent due to lack of clustering. Low values are an indication that re-creating the clustered index or running reorg rebuild on the table could improve I/O performance.

Index statistics

This optdiag section is printed for each nonclustered index and for a clustered index on a data-only-locked table. Information for clustered indexes on allpages-locked tables is reported as part of the table statistics. Table 6-4 describes the output.

Sample output for index statistics

```
Statistics for index:                "title_id_ix" (nonclustered)
Index column list:                      "title_id"
     Leaf count:                        45
     Empty leaf page count:             0
     Data page CR count:                4952.0000000000000000
     Index page CR count:               6.0000000000000000
     Data row CR count:                 4989.0000000000000000
     First extent leaf pages:           0
     Leaf row size:                     17.8905999999999992
     Index height:                      1

  Derived statistics:
     Data page cluster ratio:           0.0075819672131148
     Index page cluster ratio:          1.0000000000000000
     Data row cluster ratio:            0.0026634382566586
```

Table 6-4: Index statistics

Row label	Information provided
Statistics for index	Index name and type.
Index column list	List of columns in the index.
Leaf count	Number of leaf-level pages in the index.
Empty leaf page count	Number of empty leaf pages in the index.
Data page CR count	A counter used to compute the data page cluster r.atio for accessing a table using the index. See “Index-level derived statistics” on page 143.
Index page CR count	A counter used to compute the index page cluster ratio. See “Index-level derived statistics” on page 143.
Data row CR count	A counter used to compute the data row cluster ratio See “Index-level derived statistics” on page 143.
First extent leaf pages	The number of leaf pages in the index stored in the first extent in an allocation unit. These pages need to be read using 2K I/O, rather than large I/O. This information is maintained only for indexes on data-only-locked tables.
Leaf row size	Average size of a leaf-level row in the index. This value is only updated by update statistics, create index, and alter table...lock.
Index height	Index height, not including the leaf level.

Index-level derived statistics

The derived statistics in the index-level section are based on the “CR count” values shown in “Index statistics” on page 142.

Table 6-5: Cluster ratios for a nonclustered index

Row label	Information provided
Data page cluster ratio	The fraction of row accesses that do not require an additional extent I/O because of storage fragmentation, while accessing rows in order by this index using large I/O. It is a measure of the sequencing of data pages on extents.
Index page cluster ratio	The fraction of index leaf page accesses via the page chain that do not require extra extent I/O. It is a measure of the sequencing of index pages on extents.
Data row cluster ratio	The fraction of data page accesses that do not require an extra I/O when accessing data rows in order by this index. It is a measure of the sequencing of rows on data pages.
Space utilization	The ratio of the minimum space usage for the leaf level of this index, and the current space usage.
Large I/O efficiency	Estimates the number of useful pages brought in by each large I/O.

Data page cluster ratio

The data page cluster ratio is used to compute the effectiveness of large I/O when this index is used to access the data pages. If the table is perfectly clustered with respect to the index, the cluster ratio is 1.0. Data page cluster ratios can vary widely. They are often high for some indexes, and very low for others.

See “How cluster ratios affect large I/O estimates” on page 69 in the *Performance and Tuning: Optimizer* for more information.

Index page cluster ratio

The index page cluster ratio is used to estimate the cost of large I/O for queries that need to read a large number of leaf-level pages from nonclustered indexes or clustered indexes on data-only-locked tables. Some examples of such queries are covered index scans and range queries that read a large number of rows.

On newly created indexes, the "Index page cluster ratio" is 1.0, or very close to 1.0, indicating optimal clustering of index leaf pages on extents. As index pages are split and new pages are allocated from additional extents, the ratio drops. A very low percentage could indicate that dropping and re-creating the index or running reorg rebuild on the index would improve performance, especially if many queries perform covered scans.

See "How cluster ratios affect large I/O estimates" on page 69 in the *Performance and Tuning: Optimizer* for more information.

Data row cluster ratio

The data row cluster ratio is used to estimate the number of pages that need to be read while using this index to access the data pages. This ratio may be very high for some indexes, and quite low for others.

Space utilization for an index

Space utilization uses the average row size and number of rows to compute the expected minimum size of leaf-level index pages and compares it to the current number of leaf pages.

If space utilization is low, running reorg rebuild on index or dropping and re-creating the index can reduce the amount of empty space on index pages, and the number of empty pages in extents allocated to the index.

If you are using space management properties such as fillfactor or reservepagegap, the empty space that is left for additional rows on leaf pages, and the number of empty pages left in extents for the index affects space utilization.

If statistics have not been updated recently and the average row size has changed or the number of rows and pages are inaccurate, space utilization may report values greater than 1.0.

Large I/O efficiency for an index

Large I/O efficiency estimates the number of useful pages brought in by each large I/O. For examples, if the value is.5, a 16K I/O returns, on average, 4 2K pages needed for the query, and 4 other pages, either empty pages or pages that share the extent due to lack of clustering.

Low values are an indication that re-creating indexes or running reorg rebuild could improve I/O performance.

Column statistics

optdiag column-level statistics include:

- Statistics giving the density and selectivity of columns. If an index includes more than one column, optdiag prints the information described in Table 6-6 for each prefix subset of the index keys. If statistics are created using update statistics with a column name list, density statistics are stored for each prefix subset in the column list.
- A histogram, if the table contains one or more rows of data at the time the index is created or update statistics is run. There is a histogram for the leading column for:
  - Each index that currently exists (if there was at least one non-null value in the column when the index was created)
  - Any indexes that have been created and dropped (as long as delete statistics has not been run)
  - Any column list on which update statistics has been run

  There is also a histogram for:
  - Every column in an index, if the update index statistics command was used
  - Every column in the table, if the update all statistics command was used

optdiag also prints a list of the columns in the table for which there are no statistics. For example, here is a list of the columns in the authors table that do not have statistics:

```
No statistics for column(s):        "address"
(default values used)               "au_fname"
                                    "phone"
                                    "state"
                                    "zipcode"
```

Sample output for column statistics

The following sample shows the statistics for the city column in the authors table:

```
Statistics for column:                "city"
Last update of column statistics:     Jul 20 1998  6:05:26:656PM

     Range cell density:              0.0007283200000000
     Total density:                   0.0007283200000000
     Range selectivity:               default used (0.33)
     In between selectivity:          default used (0.25)
```

Table 6-6: Column statistics

Row label	Information provided
Statistics for column	Name of the column; if this block of information provides information about a prefix subset in a compound index or column list, the row label is "Statistics for column group."
Last update of column statistics	Date the index was created, date that update statistics was last run, or date that optdiag was last used to change statistics.
Statistics originated from upgrade of distribution page	Statistics resulted from an upgrade of a pre-11.9 distribution page. This message is not printed if update statistics has been run on the table or index or if the index has been dropped and re-created after an upgrade. If this message appears in optdiag output, running update statistics is recommended.
Statistics loaded from Optdiag	optdiag was used to change sysstatistics information. create index commands print warning messages indicating that edited statistics are being overwritten. This row is not displayed if the statistics were generated by update statistics or create index.
Range cell density	Density for equality search arguments on the column. See "Range cell and total density values" on page 148.
Total density	Join density for the column. This value is used to estimate the number of rows that will be returned for a join on this column. See "Range cell and total density values" on page 148.
Range selectivity	Prints the default value of .33, unless the value has been updated using optdiag input mode. This is the value used for range queries if the search argument is not known at optimize time.
In between selectivity	Prints the default value of .25, unless the value has been updated using optdiag input mode. This is the value used for range queries if the search argument is not known at optimize time.

Range cell and total density values

Adaptive Server stores two values for the density of column values:

- The "Range cell density" measures the duplicate values only for range cells.

  If there are any frequency cells for the column, they are eliminated from the computation for the range-cell density.

  If there are only frequency cells for the column, and no range cells, the range-cell density is 0.

  See "Understanding histogram output" on page 152 for information on range and frequency cells.

- The "Total density" measures the duplicate values for all columns, those represented by both range cells and frequency cells.

Using two separate values improves the optimizer's estimates of the number of rows to be returned:

- If a search argument matches the value of a frequency cell, the fraction of rows represented by the weight of the frequency cell will be returned.
- If a search argument falls within a range cell, the range-cell density and the weight of the range cell are used to estimate the number of rows to be returned.

For joins, the optimizer bases its estimates on the average number of rows to be returned for each scan of the table, so the total density, which measures the average number of duplicates for all values in the column, provides the best estimate. The total density is also used for equality arguments when the value of the search argument is not known when the query is optimized.

See "Range and in-between selectivity values" on page 150 for more information.

For indexes on multiple columns, the range-cell density and total density are stored for each prefix subset. In the sample output below for an index on titles (pub_id, type, pubdate), the density values decrease with each additional column considered.

```
Statistics for column:              "pub_id"
Last update of column statistics:   Feb  4 1998 12:58PM

     Range cell density:            0.0335391029690461
     Total density:                 0.0335470400000000

Statistics for column group:        "pub_id", "type"
Last update of column statistics:   Feb  4 1998 12:58PM
```

```
     Range cell density:            0.0039044009265108
     Total density:                 0.0039048000000000

Statistics for column group:        "pub_id", "type", "pubdate"
Last update of column statistics:   Feb  4 1998 12:58PM

     Range cell density:            0.0002011791956201
     Total density:                 0.0002011200000000
```

With 5000 rows in the table, the increasing precision of the optimizer's estimates of rows to be returned depends on the number of search arguments used in the query:

- An equality search argument on only pub_id results in the estimate that 0.0335391029690461 * 5000 rows, or 168 rows, will be returned.
- Equality search arguments for all three columns result in the estimate that 0.0002011791956201 * 5000 rows, or only 1 row will be returned.

This increasing level of accuracy as more search arguments are evaluated can greatly improve the optimization of many queries.

Range and in-between selectivity values

optdiag prints the default values for range and in-between selectivity, or the values that have been set for these selectivities in an earlier optdiag session. These values are used for range queries when search arguments are not known when the query is optimized.

For equality search arguments whose value is not known, the total density is used as the default.

Search arguments cannot be known at optimization time for:

- Stored procedures that set variables within a procedure
- Queries in batches that set variables for search arguments within a batch

Table 2-2 on page 21 in the *Performance and Tuning: Optimizer* shows the default values that are used. These approximations can result in suboptimal query plans because they either overestimate or underestimate the number of rows to be returned by a query.

See "Updating selectivities with optdiag input mode" on page 159 for information on using optdiag to supply selectivity values.

Histogram displays

Histograms store information about the distribution of values in a column. Table 6-7 shows the commands that create and update histograms and which columns are affected.

Table 6-7: Commands that create histograms

Command	Histogram for
create index	Leading column only
update statistics	
table_name or *index_name*	Leading column only
column_list	Leading column only
update index statistics	All indexed columns
update all statistics	All columns

Sample output for histograms

```
Histogram for column:              "city"
Column datatype:                   varchar(20)
Requested step count:              20
Actual step count:                 20
```

optdiag first prints summary data about the histogram, as shown in Table 6-8.

Table 6-8: Histogram summary statistics

Row label	Information provided
Histogram for column	Name of the column.
Column datatype	Datatype of the column, including the length, precision and scale, if appropriate for the datatype.
Requested step count	Number of steps requested for the column.
Actual step count	Number of steps generated for the column. This number can be less than the requested number of steps if the number of distinct values in the column is smaller than the requested number of steps.

Histogram output is printed in columns, as described in Table 6-9.

Table 6-9: Columns in optdiag histogram output

Column	Information provided
Step	Number of the step.
Weight	Weight of the step.
(Operator)	<, <=, or =, indicating the limit of the value. Operators differ, depending on whether the cell represents a range cell or a frequency call.
Value	Upper boundary of the values represented by a range cell or the value represented by a frequency count.

No heading is printed for the Operator column.

Understanding histogram output

A histogram is a set of cells in which each cell has a weight. Each cell has an upper bound and a lower bound, which are distinct values from the column. The weight of the cell is a floating-point value between 0 and 1, representing either:

- The fraction of rows in the table within the range of values, if the operator is <=, or
- The number of values that match the step, if the operator is =.

The optimizer uses the combination of ranges, weights, and density values to estimate the number of rows in the table that are to be returned for a query clause on the column.

Adaptive Server uses equi-height histograms, where the number of rows represented by each cell is approximately equal. For example, the following histogram on the city column on pubtune..authors has 20 steps; each step in the histogram represents about 5 percent of the table:

```
Step     Weight                Value

   1     0.00000000     <=     "APO
Miamh\377\377\377\377\377\377\377"
   2     0.05460000     <=     "Atlanta"
   3     0.05280000     <=     "Boston"
   4     0.05400000     <=     "Charlotte"
   5     0.05260000     <=     "Crown"
   6     0.05260000     <=     "Eddy"
   7     0.05260000     <=     "Fort Dodge"
   8     0.05260000     <=     "Groveton"
```

```
 9     0.05340000     <=     "Hyattsville"
10     0.05260000     <=     "Kunkle"
11     0.05260000     <=     "Luthersburg"
12     0.05340000     <=     "Milwaukee"
13     0.05260000     <=     "Newbern"
14     0.05260000     <=     "Park Hill"
15     0.05260000     <=     "Quicksburg"
16     0.05260000     <=     "Saint David"
17     0.05260000     <=     "Solana Beach"
18     0.05260000     <=     "Thornwood"
19     0.05260000     <=     "Washington"
20     0.04800000     <=     "Zumbrota"
```

The first step in a histogram represents the proportion of null values in the table. Since there are no null values for city, the weight is 0. The value for the step that represents null values is represented by the highest value that is less than the minimum column value.

For character strings, the value for the first cell is the highest possible string value less than the minimum column value (“APO Miami” in this example), padded to the defined column length with the highest character in the character set used by the server. What you actually see in your output depends on the character set, type of terminal, and software that you are using to view optdiag output files.

In the preceding histogram, the value represented by each cell includes the upper bound, but excludes the lower bound. The cells in this histogram are called **range cells**, because each cell represents a range of values.

The range of values included in a range cell can be represented as follows:

lower_bound < (values for cell) <= upper bound

In optdiag output, the lower bound is the value of the previous step, and the upper bound is the value of the current step.

For example, in the histogram above, step 4 includes Charlotte (the upper bound), but excludes Boston (the lower bound). The weight for this step is.0540, indicating that 5.4 percent of the table matches the query clause:

```
where city > Boston and city <= "Charlotte"
```

The operator column in the optdiag histogram output shows the <= operator. Different operators are used for histograms with highly duplicated values.

Histograms for columns with highly duplicated values

Histograms for columns with highly duplicated values look very different from histograms for columns with a large number of discrete values. In histograms for columns with highly duplicated values, a single cell, called a **frequency cell**, represents the duplicated value.

The weight of the frequency cell shows the percentage of columns that have matching values.

Histogram output for frequency cells varies, depending on whether the column values represent one of the following:

- A dense frequency count, where values in the column are contiguous in the domain. For example, 1, 2, 3 are contiguous integer values
- A sparse frequency count, where the domain of possible values contains values not represented by the discrete set of values in the table
- A mix of dense and sparse frequency counts.

Histogram output for some columns includes a mix of frequency cells and range cells.

Histograms for dense frequency counts

The following output shows the histogram for a column that has 6 distinct integer values, 1–6, and some null values:

```
Step     Weight                Value

   1     0.13043478      <     1
   2     0.04347826      =     1
   3     0.17391305     <=     2
   4     0.30434781     <=     3
   5     0.13043478     <=     4
   6     0.17391305     <=     5
   7     0.04347826     <=     6
```

The histogram above shows a **dense frequency count**, because all the values for the column are contiguous integers.

The first cell represents null values. Since there are null values, the weight for this cell represents the percentage of null values in the column.

The “Value” column for the first step displays the minimum column value in the table and the < operator.

Histograms for sparse frequency counts

In a histogram representing a column with a *sparse frequency count*, the highly duplicated values are represented by a step showing the discrete values with the = operator and the weight for the cell.

Preceding each step, there is a step with a weight of 0.0, the same value, and the < operator, indicating that there are no rows in the table with intervening values. For columns with null values, the first step will have a nonzero weight if there are null values in the table.

The following histogram represents the type column of the titles table. Since there are only 9 distinct types, they are represented by 18 steps.

```
Step     Weight                    Value

   1     0.00000000        <       "UNDECIDED   "
   2     0.11500000        =       "UNDECIDED   "
   3     0.00000000        <       "adventure   "
   4     0.11000000        =       "adventure   "
   5     0.00000000        <       "business    "
   6     0.11040000        =       "business    "
   7     0.00000000        <       "computer    "
   8     0.11640000        =       "computer    "
   9     0.00000000        <       "cooking     "
  10     0.11080000        =       "cooking     "
  11     0.00000000        <       "news        "
  12     0.10660000        =       "news        "
  13     0.00000000        <       "psychology  "
  14     0.11180000        =       "psychology  "
  15     0.00000000        <       "romance     "
  16     0.10800000        =       "romance     "
  17     0.00000000        <       "travel      "
  18     0.11100000        =       "travel      "
```

For example, 10.66% of the values in the type column are “news,” so for a table with 5000 rows, the optimizer estimates that 533 rows will be returned.

Histograms for columns with sparse and dense values

For tables with some values that are highly duplicated, and others that have distributed values, the histogram output shows a combination of operators and a mix of frequency cells and range cells.

The column represented in the histogram below has a value of 30.0 for a large percentage of rows, a value of 50.0 for a large percentage of rows, and a value 100.0 for another large percentage of rows.

There are two steps in the histogram for each of these values: one step representing the highly duplicated value has the = operator and a weight showing the percentage of columns that match the value. The other step for each highly duplicated value has the < operator and a weight of 0.0. The datatype for this column is numeric(5,1).

```
Step     Weight                Value

   1     0.00000000     <=     0.9
   2     0.04456094     <=     20.0
   3     0.00000000      <     30.0
   4     0.29488859      =     30.0
   5     0.05996068     <=     37.0
   6     0.04292267     <=     49.0
   7     0.00000000      <     50.0
   8     0.19659241      =     50.0
   9     0.06028834     <=     75.0
  10     0.05570118     <=     95.0
  11     0.01572739     <=     99.0
  12     0.00000000      <     100.0
  13     0.22935779      =     100.0
```

Since the lowest value in the column is 1.0, the step for the null values is represented by 0.9.

Choosing the number of steps for highly duplicated values

The histogram examples for frequency cells in this section use a relatively small number of highly duplicated values, so the resulting histograms require less than 20 steps, which is the default number of steps for create index or update statistics.

If your table contains a large number of highly duplicated values for a column, and the distribution of keys in the column is not uniform, increasing the number of steps in the histogram can allow the optimizer to produce more accurate cost estimates for queries with search arguments on the column.

For columns with dense frequency counts, the number of steps should be at least one greater than the number of values, to allow a step for the cell representing null values.

For columns with sparse frequency counts, use at least twice as many steps as there are distinct values. This allows for the intervening cells with zero weights, plus the cell to represent the null value. For example, if the titles table in the pubtune database has 30 distinct prices, this update statistics command creates a histogram with 60 steps:

```
update statistics titles
using 60 values
```

This create index command specifies 60 steps:

```
create index price_ix on titles(price)
with statistics using 60 values
```

If a column contains some values that match very few rows, these may still be represented as range cells, and the resulting number of histogram steps will be smaller than the requested number. For example, requesting 100 steps for a state column may generate some range cells for those states represented by a small percentage of the number of rows.

Changing statistics with *optdiag*

A System Administrator can use optdiag to change column-level statistics.

Warning! Using optdiag to alter statistics can improve the performance of some queries. Remember, however, that optdiag overwrites existing information in the system tables, which can affect all queries for a given table.

Use extreme caution and test all changes thoroughly on all queries that use the table. If possible, test the changes using optdiag simulate on a development server before loading the statistics into a production server.

If you load statistics without simulate mode, be prepared to restore the statistics, if necessary, either by using an untouched copy of optdiag output or by rerunning update statistics.

Do not attempt to change any statistics by running an update, delete, or insert command.

optdiag output from a 32-bit Adaptive Server can be used to change statistics in another 32-bit Adaptive Server, but not a 64-bit Adaptive Server. Similarly, optdiag output from a 64-bit Adaptive Server should not be used as input to a 32-bit Adaptive Server.

After you change statistics using optdiag, running create index or update statistics overwrites the changes. The commands succeed, but print a warning message. This message indicates that altered statistics for the titles.type column have been overwritten:

```
WARNING: Edited statistics are overwritten. Table: 'titles'
(objectid 208003772), column: 'type'.
```

Using the *optdiag* binary mode

Because precision can be lost with floating point numbers, optdiag provides a binary mode. The following command displays both human-readable and binary statistics:

```
optdiag binary statistics pubtune..titles.price
        -Usa -Ppasswd -o price.opt
```

In binary mode, any statistics that can be edited with optdiag are printed twice, once with binary values, and once with floating-point values. The lines displaying the float values start with the optdiag comment character, the pound sign (#).

This sample shows the first few rows of the histogram for the city column in the authors table:

```
Step  Weight              Value

  1   0x3d2810ce    <=    0x41504f204d69616d68ffffffffffffffffffff
# 1   0.04103165    <=    "APO Miamh\377\377\377\377\377\377\377\377"
  2   0x3d5748ba    <=    0x41746c616e7461
# 2   0.05255959    <=    "Atlanta"
  3   0x3d5748ba    <=    0x426f79657273
# 3   0.05255959    <=    "Boyers"
  4   0x3d58e27d    <=    0x4368617474616e6f6f6761
# 4   0.05295037    <=    "Chattanooga"
```

When optdiag loads this file, all uncommented lines are read, while all characters following the pound sign are ignored. To edit the float values instead of the binary values, remove the pound sign from the lines displaying the float values, and insert the pound sign at the beginning of the corresponding line displaying the binary value.

When you must use binary mode

Two histogram steps in optdiag output can show the same value due to loss of precision, even though the binary values differ. For example, both 1.999999999 and 2.000000000 may be displayed as 2.000000000 in decimal, even though the binary values are 0x3fffffffffbb47d0 and 0x4000000000000000. In these cases, you should use binary mode for input.

If you do not use binary mode, optdiag issues an error message indicating that the step values are not increasing and telling you to use binary mode. optdiag skips loading the histogram in which the error occurred, to avoid losing precision in sysstatistics.

Updating selectivities with *optdiag* input mode

You can use optdiag to customize the server-wide default values for selectivities to match the data for specific columns in your application. The optimizer uses range and in-between selectivity values when the value of a search argument is not known when a query is optimized.

The server-wide defaults are:

- Range selectivity – 0.33
- In-between selectivity – 0.25

You can use optdiag to provide values to be used to optimize queries on a specific column. The following example shows how optdiag displays default values:

```
Statistics for column:              "city"
Last update of column statistics:   Feb  4 1998  8:42PM

     Range cell density:            0x3f634d23b702f715
#    Range cell density:            0.0023561189228464
     Total density:                 0x3f46fae98583763d
#    Total density:                 0.0007012977830773
     Range selectivity:             default used (0.33)
#    Range selectivity:             default used (0.33)
     In between selectivity:        default used (0.25)
#    In between selectivity:        default used (0.25)
```

To edit these values, replace the entire “default used (0.33)” or “default used (0.25)” string with a float value. The following example changes the range selectivity to .25 and the in-between selectivity to .05, using decimal values:

```
Range selectivity:              0.250000000
In between selectivity:         0.050000000
```

Editing histograms

You can edit histograms to:

- Remove a step, by transferring its weight to an adjacent line and deleting the step
- Add a step or steps, by spreading the weight of a cell to additional lines, with the upper bound for column values the step is to represent

Adding frequency count cells to a histogram

One common reason for editing histograms is to add frequency count cells without greatly increasing the number of steps. The changes you will need to make to histograms vary, depending on whether the values represent a dense or sparse frequency count.

Editing a histogram with a dense frequency count

To add a frequency cell for a given column value, check the column value just less than the value for the new cell. If the next-lesser value is as close as possible to the value to be added, then the frequency count determined simply.

If the next lesser column value to the step to be changed is as close as possible to the frequency count value, then the frequency count cell can be extracted simply.

For example, if a column contains at least one 19 and many 20s, and the histogram uses a single cell to represent all the values greater than 17 and less than or equal to 22, optdiag output shows the following information for the cell:

```
Step     Weight             Value
...
4     0.100000000    <=     17
5     0.400000000    <=     22
...
```

Altering this histogram to place the value 20 on its own step requires adding two steps, as shown here:

```
...
4     0.100000000    <=     17
```

```
5     0.050000000    <=     19
6     0.300000000    <=     20
7     0.050000000    <=     22
...
```

In the altered histogram above, step 5 represents all values greater than 17 and less than or equal to 19. The sum of the weights of steps 5, 6, and 7 in the modified histogram equals the original weight value for step 5.

Editing a histogram with a sparse frequency count

If the column has no values greater than 17 and less than 20, the representation for a sparse frequency count must be used instead. Here are the original histogram steps:

```
Step     Weight            Value
...
4     0.100000000    <=     17
5     0.400000000    <=     22
...
```

The following example shows the zero-weight step, step 5, required for a sparse frequency count:

```
...
4     0.100000000    <=     17
5     0.000000000     <     20
6     0.350000000     =     20
7     0.050000000    <=     22
...
```

The operator for step 5 must be <. Step 6 must specify the weight for the value 20, and its operator must be =.

Skipping the load-time verification for step numbering

By default, optdiag input mode checks that the numbering of steps in a histogram increases by 1. To skip this check after editing histogram steps, use the command line flag -T4:

```
optdiag statistics pubtune..titles -Usa -Ppassword
    -T4 -i titles.opt
```

Rules checked during histogram loading

During histogram input, the following rules are checked, and error messages are printed if the rules are violated:

- The step numbers must increase monotonically, unless the `-T4` command line flag is used.
- The column values for the steps must increase monotonically.
- The weight for each cell must be between 0.0 and 1.0.
- The total of weights for a column must be close to 1.0.
- The first cell represents null values and it must be present, even for columns that do not allow null values. There must be only one cell representing the null value.
- Two adjacent cells cannot both use the < operator.

Re-creating indexes without losing statistics updates

If you need to drop and re-create an index after you have updated a histogram, and you want to keep the edited values, specify 0 for the number of steps in the create index command. This command re-creates the index without changing the histogram:

```
create index title_id_ix on titles(title_id)
with statistics using 0 values
```

Using simulated statistics

optdiag can generate statistics that can be used to simulate a user environment without requiring a copy of the table data. This permits analysis of query optimization using a very small database. For example, simulated statistics can be used:

- For Technical Support replication of optimizer problems
- To perform “what if” analysis to plan configuration changes

In most cases, you will use simulated statistics to provide information to Technical Support or to perform diagnostics on a development server.

See “Requirements for loading and using simulated statistics” on page 165 for information on setting up a separate database for using simulated statistics.

You can also load simulated statistics into the database from which they were copied. Simulated statistics are loaded into the system tables with IDs that distinguish them from the actual table data. The set statistics simulate on command instructs the server to optimize queries using the simulated statistics, rather than the actual statistics.

optdiag syntax for simulated statistics

This command displays simulate-mode statistics for the pubtune database:

```
optdiag simulate statistics pubtune -o pubtune.sim
```

If you want binary simulated output, use:

```
optdiag binary simulate statistics pubtune -
o pubtune.sim
```

To load these statistics, use:

```
optdiag simulate statistics -i pubtune.sim
```

Simulated statistics output

Output for the simulate option to optdiag prints a row labeled “simulated” for each row of statistics, except histograms. You can modify and load the simulated values, while retaining the file as a record of the actual values.

- If binary mode is specified, there are three rows of output:
  - A binary “simulated” row
  - A decimal “simulated” row, commented out
  - A decimal “actual” row, commented out
- If binary mode is not specified, there are two rows:
  - A “simulated” row
  - An “actual” row, commented out

Here is a sample of the table-level statistics for the titles table in the pubtune database:

```
Table owner:                          "dbo"
Table name:                           "titles"
```

```
Statistics for table:                  "titles"

     Data page count:                  731.0000000000000000 (simulated)
#    Data page count:                  731.0000000000000000 (actual)
     Empty data page count:            1.0000000000000000 (simulated)
#    Empty data page count:            1.0000000000000000 (actual)
     Data row count:                   5000.0000000000000000 (simulated)
#    Data row count:                   5000.0000000000000000 (actual)
     Forwarded row count:              0.0000000000000000 (simulated)
#    Forwarded row count:              0.0000000000000000 (actual)
     Deleted row count:                0.0000000000000000 (simulated)
#    Deleted row count:                0.0000000000000000 (actual)
     Data page CR count:               0.0000000000000000 (simulated)
#    Data page CR count:               0.0000000000000000 (actual)
     OAM + allocation page count:      6.0000000000000000 (simulated)
#    OAM + allocation page count:      6.0000000000000000 (actual)
     First extent data pages:          0.0000000000000000 (simulated)
#    First extent data pages:          0.0000000000000000 (actual)
     Data row size:                    190.0000000000000000 (simulated)
#    Data row size:                    190.0000000000000000 (actual)
```

In addition to table and index statistics, the simulate option to optdiag copies out:

- Partitioning information for partitioned tables. If a table is partitioned, these two lines appear at the end of the table statistics:

```
     Pages in largest partition:       390.0000000000000000 (simulated)
#    Pages in largest partition:       390.0000000000000000 (actual)
```

- Settings for the parallel processing configuration parameters:

```
Configuration Parameters:
     Number of worker processes:        20 (simulated)
#    Number of worker processes:        20 (actual)
     Max parallel degree:              10 (simulated)
#    Max parallel degree:              10 (actual)
     Max scan parallel degree:         3 (simulated)
#    Max scan parallel degree:         3 (actual)
```

- Cache configuration information for the default data cache and the caches used by the specified database or the specified table and its indexes. If tempdb is bound to a cache, that cache's configuration is also included. Here is sample output for the cache used by the pubtune database:

```
Configuration for cache:               "pubtune_cache"
```

```
        Size of 2K pool in Kb:          15360 (simulated)
#       Size of 2K pool in Kb:          15360 (actual)
        Size of 4K pool in Kb:          0 (simulated)
#       Size of 4K pool in Kb:          0 (actual)
        Size of 8K pool in Kb:          0 (simulated)
#       Size of 8K pool in Kb:          0 (actual)
        Size of 16K pool in Kb:         0 (simulated)
#       Size of 16K pool in Kb:         0 (actual)
```

If you want to test how queries use a 16K pool, you could alter the simulated statistics values above to read:

```
Configuration for cache:                "pubtune_cache"

     Size of 2K pool in Kb:             10240 (simulated)
#    Size of 2K pool in Kb:             15360 (actual)
     Size of 4K pool in Kb:             0 (simulated)
#    Size of 4K pool in Kb:             0 (actual)
     Size of 8K pool in Kb:             0 (simulated)
#    Size of 8K pool in Kb:             0 (actual)
     Size of 16K pool in Kb:            5120 (simulated)
#    Size of 16K pool in Kb:            0 (actual)
```

Requirements for loading and using simulated statistics

To use simulated statistics, you must issue the set statistics simulate on command before running the query.

For more information, see “Running queries with simulated statistics” on page 167.

To accurately simulate queries:

- Use the same locking scheme and partitioning for tables
- Re-create any triggers that exist on the tables and use the same referential integrity constraints
- Set any non default cache strategies and any non default concurrency optimization values
- Bind databases and objects to the caches used in the environment you are simulating

- Include any set options that affect query optimization (such as set parallel_degree) in the batch you are testing
- Create any view used in the query
- Use cursors, if they are used for the query
- Use a stored procedure, if you are simulating a procedure execution

Simulated statistics can be loaded into the original database, or into a database created solely for performing "what-if" analysis on queries.

Using simulated statistics in the original database

When the statistics are loaded into the original database, they are placed in separate rows in the system tables, and do not overwrite existing non-simulated statistics. The simulated statistics are only used for sessions where the set statistics simulate command is in effect.

While simulated statistics are not used to optimize queries for other sessions, executing any queries by using simulated statistics may result in query plans that are not optimal for the actual tables and indexes, and executing these queries may adversely affect other queries on the system.

Using simulated statistics in another database

When statistics are loaded into a database created solely for performing "what-if" analysis on queries, the following steps must be performed first:

- The database named in the input file must exist; it can be as small as 2MB. Since the database name occurs only once in the input file, you can change the database name, for example, from production to test_db.
- All tables and indexes included in the input file must exist, but the tables do not need to contain data.
- All caches named in the input file must exist. They can be the smallest possible cache size, 512K, with only a 2K pool. The simulated statistics provide the information for pool configuration.

Dropping simulated statistics

Loading simulated statistics adds rows describing cache configuration to the sysstatistics table in the master database. To remove these statistics, use delete shared statistics. The command has no effect on the statistics in the database where the simulated statistics were loaded.

If you have loaded simulated statistics into a database that contains real table and index statistics, you can drop simulated statistics in one of these ways:

- Use delete statistics on the table which deletes all statistics, and run update statistics to re-create only the non simulated statistics.
- Use optdiag (without simulate mode) to copy statistics out; then run delete statistics on the table, and use optdiag (without simulate mode) to copy statistics in.

Running queries with simulated statistics

set statistics simulate on tells the optimizer to optimize queries using simulated statistics:

```
set statistics simulate on
```

In most cases, you also want to use set showplan on or dbcc traceon(302).

If you have loaded simulated statistics into a production database, use set noexec on when you run queries using simulated statistics so that the query does not execute based on statistics that do not match the actual tables and indexes. This lets you examine the output of showplan and dbcc traceon(302) without affecting the performance of the production system.

showplan messages for simulated statistics

When set statistics simulate is enabled and there are simulated statistics available, showplan prints the following message:

```
Optimized using simulated statistics.
```

If the server on which the simulation tests are performed has the parallel query options set to smaller values than the simulated values, showplan output first displays the plan using the simulated statistics, and then an adjusted query plan. If set noexec is turned on, the adjusted plan is not displayed.

Character data containing quotation marks

In histograms for character and datetime columns, all column data is contained in double quotes. If the column itself contains the double-quote character, optdiag displays two quotation marks. If the column value is:

```
a quote "mark"
```

optdiag displays:

```
"a quote" "mark"
```

The only other special character in optdiag output is the pound sign (#). In input mode, all characters on the line following a pound sign are ignored, except when the pound sign occurs within quotation marks as part of a column name or column value.

Effects of SQL commands on statistics

The information stored in systabstats and sysstatistics is affected by data definition language (DDL). Some data modification language also affects systabstats. Table 6-10 summarizes how DDL affects the systabstats and sysstatistics tables.

Table 6-10: Effects of DDL on systabstats and sysstatistics

Command	Effect on systabstats	Effect on sysstatistics
alter table...lock	Changes values to reflect the changes to table and index structure and size. When changing from allpages locking to data-only locking, the indid for clustered indexes is set to 0 for the table, and a new row is inserted for the index.	Same as create index, if changing from allpages to data-only locking or vice versa; no effect on changing between data-only locking schemes.
alter table to add, drop or modify a column definition	If the change affects the length of the row so that copying the table is required,	
create table	Adds a row for the table. If a constraint creates an index, see the create index commands below.	No effect, unless a constraint creates an index. See the create index commands below.
create clustered index	For allpages-locked tables, changes indid to 1 and updates columns that are pertinent to the index; for data-only-locked tables, adds a new row.	Adds rows for columns not already included; updates rows for columns already included.

Command	Effect on systabstats	Effect on sysstatistics
create nonclustered index	Adds a row for the nonclustered index.	Adds rows for columns not already included; updates rows for columns already included.
delete statistics	No effect.	Deletes all rows for a table or just the rows for a specified column.
drop index	Removes rows for nonclustered indexes and for clustered indexes on data-only-locked tables. For clustered indexes on allpages-locked tables, sets the indid to 0 and updates column values.	Does not delete actual statistics for the indexed columns. This allows the optimizer to continue to use this information. Deletes simulated statistics for nonclustered indexes. For clustered indexes on allpages-locked tables, changes the value for the index ID in the row that contains simulated table data.
drop table	Removes all rows for the table.	Removes all rows for the table.
reorg	Updates restart points, if used with a time limit; updates number of pages and cluster ratios if page counts change; affects other values such as empty pages, forwarded or deleted row counts, depending on the option used.	The rebuild option recreates indexes.
truncate table	Resets values to reflect an empty table. Some values, like row length, are retained.	No effect; this allows reloading a truncated table without rerunning update statistics.
update statistics		
table_name	Updates values for the table and for all indexes on the specified table.	Updates histograms for the leading column of each index on the table; updates the densities for all indexes and prefix subsets of indexes.
index_name	Updates values for the specified index.	Updates the histogram for the leading column of the specified index; updates the densities for the prefix subsets of the index.
column_name(s)	No effect.	Updates or creates a histogram for a column and updates or creates densities for the prefix subsets of the specified columns.
update index statistics		

Command	Effect on systabstats	Effect on sysstatistics
table_name	Updates values for the table and for all columns in all indexes on the specified table.	Updates histograms for all columns of each index on the table; updates the densities for all indexes and prefix subsets of indexes.
index_name	Updates values for the specified index	Updates the histogram for all column of the specified index; updates the densities for the prefix subsets of the index.
update all statistics		
table_name	Updates values for the table and for all columns in the specified table.	Updates histograms for all columns on the table; updates the densities for all indexes and prefix subsets of indexes.

How query processing affects *systabstats*

Data modification can affect many of the values in the systabstats table. To improve performance, these values are changed in memory and flushed to systabstats periodically by the housekeeper chores task.

If you need to query systabstats directly, you can flush the in-memory statistics to systabstats with sp_flushstats. This command flushes the statistics for the titles table and any indexes on the table:

```
sp_flushstats titles
```

If you do not provide a table name, sp_flushstats flushes statistics for all tables in the current database.

Note Some statistics, particularly cluster ratios, may be slightly inaccurate because not all page allocations and deallocations are recorded during changes made by data modification queries. Run update statistics or create index to correct any inconsistencies.

CHAPTER 7

Tuning with *dbcc traceon*

This chapter describes the output of the dbcc traceon(302, 310) diagnostic tools. These tools can be used for debugging problems with query optimization.

Tuning with *dbcc traceon(302)*

showplan tells you the final decisions that the optimizer makes about your queries. dbcc traceon(302) can often help you understand why the optimizer makes choices that seem incorrect. It can help you debug queries and decide whether to use certain options, like specifying an index or a join order for a particular query. It can also help you choose better indexes for your tables.

When you turn on dbcc traceon(302), you eavesdrop on the optimizer as it examines query clauses and applies statistics for tables, search arguments, and join columns.

The output from this trace facility is more detailed than showplan and statistics io output, but it provides information about why the optimizer made certain query plan decisions.

The query cost statistics printed by dbcc traceon(302) can help to explain, for example, why a table scan is chosen rather than an indexed access, why index1 is chosen rather than index2, and so on.

dbcc traceon(310)

dbcc traceon(310) output can be extremely lengthy and is hard to understand without a thorough understanding of the optimizer. You often need to have your showplan output available as well to understand the join order, join type, and the join columns and indexes used.

The most relevant parts of dbcc traceon(310) output, however, are the per-table total I/O estimates.

Invoking the *dbcc* trace facility

To start the dbcc traceon(302) trace facility, execute the following command from an isql batch, followed by the query or stored procedure that you want to examine:

```
dbcc traceon(3604, 302)
```

This is what the trace flags mean:

Trace flag	Explanation
3604	Directs trace output to the client, rather than to the error log.
302	Prints trace information on index selection.

To turn off the output, use:

```
dbcc traceoff(3604, 302)
```

dbcc traceon(302) is often used in conjunction with dbcc traceon(310), which provides more detail on the optimizer's join order decisions and final cost estimates. dbcc traceon(310) also prints a "Final plan" block at the end of query optimization. To enable this trace option also, use:

```
dbcc traceon(3604, 302, 310)
```

To turn off the output, use:

```
dbcc traceoff(3604, 302, 310)
```

See "dbcc traceon(310) and final query plan costs" on page 189 for information on dbcc traceon(310).

General tips for tuning with *dbcc traceon(302)*

To get helpful output from dbcc traceon(302), be sure that your tests cause the optimizer to make the same decisions that it would make while optimizing queries in your application.

- You must supply the same parameters and values to your stored procedures or where clauses.
- If the application uses cursors, use cursors in your tuning work
- If you are using stored procedures, make sure that they are actually being optimized during the trial by executing them with recompile.

Checking for join columns and search arguments

In most cases, Adaptive Server uses only one index per table in a query. This means that the optimizer must often choose between indexes when there are multiple where clauses supporting both search arguments and join clauses. The optimizer first matches the search arguments to available indexes and statistics and estimates the number of rows and pages that qualify for each available index.

The most important item that you can verify using dbcc traceon(302) is that the optimizer is evaluating all possible where clauses included in the query.

If a SARG clause is not included in the output, then the optimizer has determined it is not a valid search argument. If you believe your query should benefit from the optimizer evaluating this clause, find out why the clause was excluded, and correct it if possible.

Once all of the search arguments have been examined, each join combination is analyzed. If the optimizer is not choosing a join order that you expect, one of the first checks you should perform is to look for the sections of dbcc traceon(302) output that show join order costing: there should be two blocks of output for each join.

If there is only one output for a given join, it means that the optimizer cannot consider using an index for the missing join order.

The most common reasons for clauses that cannot be optimized include:

- Use of functions, arithmetic, or concatenation on the column in a SARG, or on one of the join columns

- Datatype mismatches between SARGs and columns or between two columns in a join
- Numerics compared against constants that are larger than the definition of the column in a SARG, or joins between columns of different precision and scale

See “Search arguments and useful indexes” on page 15 in *Performance and Tuning: Optimizer* for more information on requirements for search arguments.

Determining how the optimizer estimates I/O costs

Identifying how the optimizer estimates I/O often leads to the root of the problems and to solutions. You can to see when the optimizer uses actual statistics and when it uses default values for your search arguments.

Structure of *dbcc traceon(302)* output

dbcc traceon(302) prints its output as the optimizer examines the clauses for each table involved in a query. The optimizer first examines all search clauses and determines the cost for each possible access method for the search clauses for each table in the query. It then examines each join clause and the cost of available indexes for the joins.

dbcc traceon(302) output prints each search and join analysis as a block of output, delimited with a line of asterisks.

The search and join blocks each contain smaller blocks of information:

- A table information block, giving basic information on the table
- A block that shows the cost of a table scan
- A block that displays the clauses being analyzed
- A block for each index analyzed
- A block that shows the best index for the clauses in this section

For joins, each join order is represented by a separate block. For example, for these joins on titles, titleauthor, and authors:

```
where titles.title_id = titleauthor.title_id
   and authors.au_id = titleauthor.au_id
```

there is a block for each join, as follows:

- titles, titleauthor
- titleauthor, titles
- titleauthor, authors
- authors, titleauthor

Additional blocks and messages

Some queries generate additional blocks or messages in dbcc traceon(302) output, as follows:

- Queries that contain an order by clause contain additional blocks for displaying the analysis of indexes that can be used to avoid performing a sort.

  See "Sort avert messages" on page 179 for more information.

- Queries using transaction isolation level 0 (dirty reads) or updatable cursors on allpages-locked tables, where unique indexes are required, return a message like the following:

  ```
  Considering unique index 'au_id_ix', indid 2.
  ```

- Queries that specify an invalid prefetch size return a message like the following:

  ```
  Forced data prefetch size of 8K is not available.
  The largest available prefetch size will be used.
  ```

Table information block

This sample output shows the table information block for a query on the titles table:

```
Beginning selection of qualifying indexes for table 'titles',
correlation name 't', varno = 0, objectid 208003772.
   The table (Datapages) has 5000 rows, 736 pages,
   Data Page Cluster Ratio 0.999990
   The table has 5 partitions.
   The largest partition has 211 pages.
   The partition skew is 1.406667.
```

Identifying the table

The first two lines identify the table, giving the table name, the correlation name (if one was used in the query), a varno value that identifies the order of the table in the from clause, and the object ID for the table.

In the query, titles is specified using "t" as a correlation name, as in:

```
from titles t
```

The correlation name is included in the output only if a correlation name was used in the query. The correlation name is especially useful when you are trying to analyze the output from subqueries or queries doing self-joins on a table, such as:

```
from sysobjects o1, sysobjects o2
```

Basic table data

The next lines of output provide basic data about the table: the locking scheme, the number of rows, and the number of pages in the table. The locking scheme is one of: Allpages, Datapages, or Datarows.

Cluster ratio

The next line prints the data page cluster ratio for the table.

Partition information

The following lines are included only for partitioned tables. They give the number of partitions, plus the number of pages in the largest partition, and the skew:

```
The table has 5 partitions.
The largest partition has 211 pages.
The partition skew is 1.406667.
```

This information is useful if you are tuning parallel queries, because:

- Costing for parallel queries is based on the cost of accessing the table's largest partition.
- The optimizer does not choose a parallel plan if the partition skew is 2.0 or greater.

See Chapter 7, "Parallel Query Processing," in *Performance and Tuning: Optimizer* for more information on parallel query optimization.

Base cost block

The optimizer determines the cost of a table scan as a first step. It also displays the caches used by the table, the availability of large I/O, and the cache replacement strategy.

The following output shows the base cost for the titles table:

```
Table scan cost is 5000 rows, 748 pages,
   using data prefetch (size 16K I/O),
  in data cache 'default data cache' (cacheid 0) with LRU replacement
```

If the cache used by the query has only a 2K pool, the prefetch message is replace by:

```
using no data prefetch (size 2K I/O)
```

Concurrency optimization message

For very small data-only-locked tables, the following message may be included in this block:

```
If this table has useful indexes, a table scan will
not be considered because concurrency optimization
is turned ON for this table.
```

For more information, see "Concurrency optimization for small tables" on page 55 in *Performance and Tuning: Optimizer*.

Clause block

The clause block prints the search clauses and join clauses that the optimizer considers while it estimates the cost of each index on the table. Search clauses for all tables are analyzed first, and then join clauses.

Search clause identification

For search clauses, the clause block prints each of the search clauses that the optimizer can use. The list should be compared carefully to the clauses that are included in your query. If query clauses are not listed, it means that the optimizer did not evaluate them because it cannot use them.

For example, this set of clauses on the titles table:

```
where type = "business"
    and title like "B%"
    and total_sales > 12 * 1000
```

produces this list of optimizable search clauses, with the table names preceding the column names:

```
Selecting best index for the SEARCH CLAUSE:
        titles.title < 'C'
        titles.title >= 'B'
        titles.type = 'business'
        titles.total_sales > 12000
```

Notice that the like has been expanded into a range query, searching for >= 'B' and <'C'. All of the clauses in the SQL statement are included in the dbcc traceon(302) output, and can be used to help optimize the query.

If search argument transitive closure and predicate factoring have added optimizable search arguments, these are included in this costing block too.

See "Search arguments and useful indexes" on page 15 in *Performance and Tuning: Optimizer* for more information.

When search clauses are not optimizable

The following set of clauses on the authors table includes the substring function on the au_fname column:

```
where substring(au_fname,1,4) = "Fred"
    and city = "Miami"
```

Due to the use of the substring function on a column name, the set of optimizable clauses does not include the where clause on the au_fname column:

```
Selecting best index for the SEARCH CLAUSE:
    authors.city = 'Miami'
```

Values unknown at optimize time

For values that are not known at optimize time, dbcc traceon(302) prints "unknown-value." For example, this clause uses the getdate function:

```
where pubdate > getdate()
```

It produces this message in the search clause list:

```
titles.pubdate > unknown-value
```

Join clause identification

Once all of the search clauses for each table have been analyzed, the join clauses are analyzed and optimized.

Each table is analyzed in the order listed in the from clause. dbcc traceon(302) prints the operator and table and column names, as shown in this sample output of a join between titleauthor and titles, during the costing of the titleauthor table:

```
Selecting best index for the JOIN CLAUSE:
        titleauthor.title_id = titles.title_id
```

The table currently undergoing analysis is always printed on the left in the join clause output. When the titles table is being analyzed, titles is printed first:

```
Selecting best index for the JOIN CLAUSE:
        titles.title_id = titleauthor.title_id
```

If you expect an index for a join column to be used, and it is not, check for the JOIN CLAUSE output with the table as the leading table. If it is not included in the output, check for datatype mismatches on the join columns.

Sort avert messages

If the query includes an order by clause, additional messages are displayed. The optimizer checks to see if an index matches the ordering required by the order by clause, to avoid incurring sort costs for the query.

This message is printed for search clauses:

```
Selecting best index for the SEARCH SORTAVERT CLAUSE:
    titles.type = 'business'
```

The message for joins shows the column under consideration first. This message is printed while the optimizer analyzes the titles table:

```
Selecting best index for the JOIN SORTAVERT CLAUSE:
    titles.title_id = titleauthor.title_id
```

At the end of the block for the search or join clause, one of two messages is printed, depending on whether an index exists that can be used to avoid performing a sort step. If no index is available, this message is printed:

```
No sort avert index has been found for table 'titles'
(objectid 208003772, varno = 0).
```

If an index can be used to avoid the sort step, the sort-avert message includes the index ID, the number of pages that need to be accessed, and the number of rows to be returned for each scan. This is a typical message:

```
The best sort-avert index is index 3, costing 9 pages
and generating 8 rows per scan.
```

This message does not mean that the optimizer has decided to use this index. It means simply that, if this index is used, it does not require a sort.

If you expect an index to be used to avoid a sort, and you see the "No sort avert index" message, check the order by clauses in the query for the use of asc and desc to request ascending and descending ordering, and check the ordering specifications for the index.

For more information, see "Costing for queries using order by" on page 79 in *Performance and Tuning: Optimizer*.

Column block

This section prints the selectivity of each optimizable search argument or join clause. Selectivity is used to estimate the number of matching rows for a search clause or join clause.

The optimizer uses column statistics, if they exist and if the value of the search argument is known at optimize time. If not, the optimizer uses default values.

Selectivities when statistics exist and values are known

This shows the selectivities for a search clause on the title column, when an index exists for the column:

```
Estimated selectivity for title,
   selectivity = 0.001077, upper limit = 0.060200.
```

For equality search arguments where the value falls in a range cell:

- The selectivity is the "Range cell density" displayed by optdiag.
- The upper limit is the weight of the histogram cell.

If the value matches a frequency cell, the selectivity and upper limit are the weight of that cell.

For range queries, the upper limit is the sum of the weights of all histogram cells that contain values in the range. The upper limit is used only in cases where interpolation yields a selectivity that is greater than the upper limit.

The upper limit is not printed when the selectivity for a search argument is 1.

For join clauses, the selectivity is the "Total density" displayed by optdiag.

When the optimizer uses default values

The optimizer uses default values for selectivity:

- When the value of a search argument is not known at the time the query is optimized
- For search arguments where no statistics are available

In both of these cases, the optimizer uses different default values, depending on the operators used in the query clause.

Unknown values

Unknown values include variables that are set in the same batch as the query and values set within a stored procedure. This message indicates an unknown value for a column where statistics are available and the equality (=) operator is used:

```
SARG is a local variable or the result of a function or an expression,
using the total density to estimate selectivity.
```

Similar messages are printed for open-ended range queries and queries using between.

If no statistics are available

If no statistics are available for a column, a message indicates the default selectivity that will be used. This message is printed for an open-ended range query on the total_sales table:

```
No statistics available for total_sales,
using the default range selectivity to estimate selectivity.

Estimated selectivity for total_sales,
    selectivity = 0.330000.
```

See “Default values for search arguments” on page 21 for the default values used for search arguments and “When statistics are not available for joins” on page 23 in *Performance and Tuning: Optimizer* for the default values used for joins.

You may be able to improve optimization for queries where default values are used frequently, by creating statistics on the columns.

See “Creating and updating column statistics” on page 53.

Out-of-range messages

Out-of-range messages are printed when a search argument is out of range of the values included in the histogram for an indexed column.

The following clause searches for a value greater than the last title_id:

```
where title_id > "Z"
```

dbcc traceon(302) prints:

```
Estimated selectivity for title_id,
   selectivity = 0.000000, upper limit = 0.000000.
Lower bound search value ''Z'' is greater than the largest value
in sysstatistics for this column.
```

For a clause that searches for a value that is less than the first key value in an index, dbcc traceon(302) prints:

```
Estimated selectivity for title_id,
    selectivity = 0.000000, upper limit = 0.000000.
Upper bound search value ''B'' is less than the smallest value
```

```
in sysstatistics for this column.
```

If the equality operator is used instead of a range operator, the messages read:

```
Estimated selectivity for title_id,
   selectivity = 0.000000, upper limit = 0.000000.
Equi-SARG search value ''Zebracode'' is greater than the largest
value in sysstatistics for this column.
```

or:

```
Estimated selectivity for title_id,
   selectivity = 0.000000, upper limit = 0.000000.
Equi-SARG search value ''Applepie'' is less than the smallest value
in sysstatistics for this column.
```

These messages may simply indicate that the search argument used in the query is out of range for the values in the table. In that case, no rows are returned by the query. However, if there are matching values for the out-of-range keys, it may indicate that it is time to run update statistics on the table or column, since rows containing these values must have been added since the last time the histogram was generated.

There is a special case for search clauses using the >= operator and a value that is less than or equal to the lowest column value in the histogram. For example, if the lowest value in an integer column is 20, this clause:

```
where col1 >= 16
```

produces this message:

```
Lower bound search condition '>= 16' includes all values in this
column.
```

For these cases, the optimizer assumes that all non-null values in the table qualify for this search condition.

"Disjoint qualifications" message

The "disjoint qualifications" message often indicates a user error in specifying the search clauses. For example, this query searches for a range where there could be no values that match both of the clauses:

```
where advance > 10000
and advance < 1000
```

The selectivity for such a set of clauses is always 0.0, meaning that no rows match these qualifications, as shown in this output:

```
Estimated selectivity for advance,
   disjoint qualifications, selectivity is 0.0.
```

Forcing messages

dbcc traceon(302) prints messages if any of the index, I/O size, buffer strategy, or parallel force options are included for a table or if an abstract plan specifying these scan properties was used to optimize the query. Here are sample messages for a query using an abstract plan:

```
For table 'titles':
User forces index 2 (index name = type_price_ix)
User forces index and data prefetch of 16K
User forces MRU buffer replacement strategy on index and data
pages
User forces parallel strategy.  Parallel Degree = 3
```

Unique index messages

When a unique index is being considered for a join or a search argument, the optimizer knows that the query will return one row per scan. The message includes the index type, the string "returns 1 row," and a page estimate, which includes the number of index levels, plus one data page:

```
Unique clustered index found, returns 1 row, 2 pages
Unique nonclustered index found, returns 1 row, 3 pages
```

Other messages in the column block

If the statistics for the column have been modified using optdiag, dbcc traceon(302) prints:

```
Statistics for this column have been edited.
```

If the statistics result from an upgrade of an earlier version of the server or from loading a database from an pre-11.9 version of the server, dbcc traceon(302) prints:

```
Statistics for this column were obtained from upgrade.
```

If this message appears, run update statistics for the table or index.

Index selection block

While costing index access, dbcc traceon(302) prints a set of statistics for each useful index. This index block shows statistics for an index on au_lname in the authors table:

```
Estimating selectivity of index 'au_names_ix', indid 2
   scan selectivity 0.000936, filter selectivity 0.000936
   5 rows, 3 pages, index height 2,
   Data Row Cluster Ratio 0.990535,
   Index Page Cluster Ratio 0.538462,
   Data Page Cluster Ratio 0.933579
```

Scan and filter selectivity values

The index selection block includes, a scan selectivity value and a filter selectivity value. In the example above, these values are the same (0.000936).

For queries that specify search arguments on multiple columns, these values are different when the search arguments include the leading key, and some other index key that is not part of a prefix subset.

That is, if the index is on columns A, B, C, D, a query specifying search arguments on A, B, and D will have different scan and filter selectivities. The two selectivities are used for estimating costs at different levels:

	Scan Selectivity	Filter Selectivity
Used to estimate:	Number of index rows and leaf-level pages to be read	Number of data pages to be accessed
Determined by:	Search arguments on leading columns in the index	All search arguments on the index under consideration. even if they are not part of the prefix subset for the index

How scan and filter selectivity can differ

This statement creates a composite index on titles:

```
create index composite_ix
on titles (pub_id, type, price)
```

Both of the following clauses can be used to position the start of the search and to limit the end point, since the leading columns are specified:

```
where pub_id = "P099"
where pub_id = "P099" and type = "news"
```

The first example requires reading all the index pages where pub_id equals "P099", while the second reads only the index pages where both conditions are true. In both cases, these queries need to read the data rows for each of the index rows that are examined, so the scan and filter selectivity are the same.

In the following example, the query needs to read all of the index leaf-level pages where pub_id equals "P099", as in the queries above. But in this case, Adaptive Server examines the value for price, and needs to read only those data pages where the price is less than $50:

```
where pub_id = "P099" and price < $50
```

In this case, the scan and filter selectivity differ. If column-level statistics exist for price, the optimizer combines the column statistics on pub_id and price to determine the filter selectivity, otherwise the filter selectivity is estimated using the default range selectivity.

In the dbcc traceon(302) output below, the selectivity for the price column uses the default value, 0.33, for an open range. When combined with the selectivity of 0.031400 for pub_id, it yields the filter selectivity of 0.010362 for composite_ix:

```
Selecting best index for the SEARCH CLAUSE:
    titles.price < 50.00
    titles.pub_id = 'P099'

Estimated selectivity for pub_id,
   selectivity = 0.031400, upper limit = 0.031400.

No statistics available for price,
using the default range selectivity to estimate selectivity.

Estimated selectivity for price,
   selectivity = 0.330000.

Estimating selectivity of index 'composite_ix', indid 6
   scan selectivity 0.031400, filter selectivity 0.010362
   52 rows, 57 pages, index height 2,
   Data Row Cluster Ratio 0.013245,
   Index Page Cluster Ratio 1.000000,
   Data Page Cluster Ratio 0.100123
```

Other information in the index selection block

The index selection block prints out an estimate of the number of rows that would be returned if this index were used and the number of pages that would need to be read. It includes the index height.

For a single-table query, this information is basically all that is needed for the optimizer to choose between a table scan and the available indexes. For joins, this information is used later in optimization to help determine the cost of various join orders.

The three cluster ratio values for the index are printed, since estimates for the number of pages depend on cluster ratios.

If the index covers the query, this block includes the line:

```
Index covers query.
```

This message indicates that the data pages of the table do not have to be accessed if this index is chosen.

Best access block

The final section for each SARG or join block for a table shows the best qualifying index for the clauses examined in the block.

When search arguments are being analyzed, the best access block looks like:

```
The best qualifying  index is 'pub_id_ix' (indid 5)
   costing 153 pages,
   with an estimate of 168 rows to be returned per scan of the table,
   using index prefetch (size 16K I/O) on leaf pages,
   in index cache 'default data cache' (cacheid 0) with LRU
replacement
   using no data prefetch (size 2K I/O),
   in data cache 'default data cache' (cacheid 0) with LRU replacement
Search argument selectivity is 0.033539.
```

If no useful index is found, the final block looks like:

```
The best qualifying access is a table scan,
   costing 621 pages,
   with an estimate of 1650 rows to be returned per scan of the table,
   using data prefetch (size 16K I/O),
   in data cache 'default data cache' (cacheid 0) with LRU replacement
```

```
Search argument selectivity is 0.330000.
```

For joins, there are two best access blocks when a merge join is considered during the join-costing phase, one for nested-loop join cost, and one for merge-join cost:

```
The best qualifying Nested Loop join index is 'au_city_ix' (indid
4)
   costing 6 pages,
  with an estimate of 4 rows to be returned per scan of the table,
   using index prefetch (size 16K I/O) on leaf pages,
   in index cache 'default data cache' (cacheid 0) with LRU
replacement
   using no data prefetch (size 2K I/O),
   in data cache 'default data cache' (cacheid 0) with LRU
replacement
Join selectivity is 0.000728.


The best qualifying Merge join index is 'au_city_ix' (indid 4)
   costing 6 pages,
  with an estimate of 4 rows to be returned per scan of the table,
   using no index prefetch (size 2K I/O) on leaf pages,
   in index cache 'default data cache' (cacheid 0) with LRU
replacement
   using no data prefetch (size 2K I/O),
   in data cache 'default data cache' (cacheid 0) with LRU
replacement
Join selectivity is 0.000728.
```

Note that the output in this block estimates the number of "rows to be returned per scan of the table." At this point in query optimization, the join order has not yet been chosen.

If this table is the outer table, the total cost of accessing the table is 6 pages, and it is estimated to return 4 rows.

If this query is an inner table of a nested-loop join, with a cost of 6 pages each time, each access is estimated to return 4 rows. The number of times the table will be scanned depends on the number of estimated qualifying rows for the other table in the join.

If no index qualifies as a possible merge-join index, dbcc traceon(302) prints:

```
If this access path is selected for merge join, it
will be sorted
```

dbcc traceon(310) and final query plan costs

The end of each search clause and join clause block prints the best index for the search or join clauses in that particular block. If you are concerned only about the optimization of the search arguments, dbcc traceon(302) output has probably provided the information you need.

The choice of the best query plan also depends on the join order for the tables, which is the next step in query optimization after the index costing step completes. dbcc traceon(310) provides information about the join order selection step.

It starts by showing the number of tables considered at a time during a join. This message shows three-at-a-time optimization, with the default for set table count, and a 32-table join:

```
QUERY IS CONNECTED
Number of tables in join: 32
Number of tables considered at a time: 3
Table count setting: 0 (default value used)
```

dbcc traceon(310) prints the first plan that the optimizer considers, and then each cheaper plan, with the heading "NEW PLAN."

To see all of the plans, use dbcc traceon(317). It prints each plan considered, with the heading "WORK PLAN." This may produce an extremely large amount of output, especially for queries with many tables, many indexes, and numerous query clauses.

If you use dbcc traceon(317), also use dbcc traceon(3604) and direct the output to a file, rather than to the server's error log to avoid filling up the error log device.

dbcc traceon(310) or (317) prints the join orders being considered as the optimizer analyzes each of the permutations. It uses the *varno*, representing the order of the tables in the from clause. For example, for the first permutation, it prints:

```
0 - 1 - 2 -
```

This is followed by the cost of joining the tables in this order. The permutation order for subsequent join orders follows, with "NEW PLAN" and the analysis of each table for the plan appearing whenever a cheaper plan is found. Once all plans have been examined, the final plan is repeated, with the heading "FINAL PLAN". This is the plan that Adaptive Server uses for the query.

Flattened subquery join order message

For some flattened subqueries, certain join orders are possible only if a sort is later used to remove duplicate results. When one of these join orders is considered, the following message is printed right after the join permutation order is printed:

```
2 - 0 - 1 -

This join order created while converting an exists join to a
regular join, which can happen for subqueries, referential
integrity, and select distinct.
```

For more information on subqueries and join orders, see “Flattened subqueries using duplicate elimination” on page 136 in *Performance and Tuning: Optimizer*.

Worker process information

Just before printing final plan information, dbcc traceon(310) prints the parallel configuration parameters and session level settings in effect when the command was run.

```
PARALLEL:
    number of worker processes = 20
    max parallel degree = 10
    min(configured,set) parallel degree = 10
    min(configured,set) hash scan parallel degree = 3
```

If session-level limits or simulated statistics in effect when the query is optimized, those values are shown in the output.

Final plan information

The plan chosen by the optimizer is displayed in the final plan block. Information about the cost of each table is printed; the output starts from the outermost table in the join order.

```
select pub_name, au_lname, price
from titles t, authors a, titleauthor ta,
            publishers p
where t.title_id = ta.title_id
    and a.au_id = ta.au_id
    and p.pub_id = t.pub_id
```

```
                    and type = 'business'
                    and price < $25
FINAL PLAN (total cost = 3909)

varno=0 (titles) indexid=1 (title_id_ix)
path=0xd6b25148 pathtype=pll-mrgscan-outer
method=NESTED ITERATION
scanthreads=3
outerrows=1 outer_wktable_pgs=0 rows=164 joinsel=1.000000
jnpgs_per_scan=3 scanpgs=623
data_prefetch=YES data_iosize=16 data_bufreplace=LRU
scanlio_perthrd=211 tot_scanlio=633 scanpio_perthrd=116
tot_scanpio=346
outer_srtmrglio=0 inner_srtmrglio=0
corder=1

varno=2 (titleauthor) indexid=3 (ta_ix)
path=0xd6b20000 pathtype=pll-mrgscan-inner
method=FULL MERGE JOIN
scanthreads=3 mergethreads=3
outerrows=164 outer_wktable_pgs=0 rows=243 joinsel=0.000237
jnpgs_per_scan=2 scanpgs=87
index_prefetch=YES index_iosize=16 index_bufreplace=LRU
scanlio_perthrd=29 total_scanlio=87 scanpio_perthrd=29
tot_scanpio=87
outer_srtmrglio_perthrd=0 tot_outer_srtmrglio=0
inner_srtmrglio_perthrd=0 tot_inner_srtmrglio=0
corder=2

varno=1 (authors) indexid=3 (au_id_ix)
path=0xd6b20318 pathtype=join
method=NESTED ITERATION
scanthreads=1
outerrows=243 rows=243 joinsel=0.000200 jnpgs_per_scan=3
index_prefetch=NO index_iosize=2 index_bufreplace=LRU
data_prefetch=NO data_iosize=2 data_bufreplace=LRU
scanlio=82 scanpio=9
corder=1

jnvar=2 refcost=0 refpages=0 reftotpages=0 ordercol[0]=1
ordercol[1]=1

varno=3 (publishers) indexid=0 ()
path=0xd6b1f150 pathtype=sclause
method=SORT MERGE JOIN
scanthreads=1
```

```
outerrows=243 outer_wktable_pgs=7 rows=243 joinsel=0.033333
jnpgs_per_scan=1 scanpgs=3
data_prefetch=NO data_iosize=2 data_bufreplace=LRU
scanlio=3 scanpio=3
outer_srtmrglio_perthrd=88 tot_outer_srtmrglio=250
inner_srtmrglio_perthrd=31 tot_inner_srtmrglio=30
corder=0

Sort-Merge Cost of Inner = 98
Sort-Merge Cost of Outer = 344
```

For the showplan output for the same query, see “Merge join messages” on page 95.

Table 7-1 shows the meaning of the values in the output.

Table 7-1: dbcc traceon(310) output

Label	Information provided
varno	Indicates the table order in the from clause, starting with 0 for the first table. The table name is provided in parentheses.
indexid	The index ID, followed by the index name, or 0 for a table scan.
pathtype	The access method for this table. See Table 7-2.
method	The method used for the scan or join: • NESTED ITERATION • NESTED ITERATION with Tuple Filtering • REFORMATTING • REFORMATTING with Unique Reformatting • OR OPTIMIZATION • SORT MERGE JOIN • RIGHT MERGE JOIN • LEFT MERGE JOIN • FULL MERGE JOIN
scanthreads	Number of worker processes to be used for the scan of this table.
merge threads	Number of threads to use for a parallel data merge, for a sort-merge join.
outerrows	Number of rows that qualify from the outer tables in the query or 1, for the first table in the join order.
outer_wktable_pgs	For a merge join, the number of pages in the worktable that is outer to this table, or tables in a full-merge join.
rows	Number of rows estimated to qualify in this table or as a result of this join. For a parallel query, this is the maximum number of rows per worker process.
joinsel	The join selectivity.
jnpgs_per_scan	Number of index and data pages to be read for each scan.
scanpgs	The total number of index and data pages to be read for the table.
index_prefetch	YES if large I/O will be used on index leaf pages (not printed for table scans and allpages-locked table clustered index scans).

Label	Information provided
index_iosize	The I/O size to be used on the index leaf pages (not printed for table scans and allpages-locked table clustered index scans).
index_bufreplace	The buffer replacement strategy to be used on the index leaf pages (not printed for table scans and allpages-locked table clustered index scans).
data_prefetch	YES if large I/O will be used on the data pages; NO if large I/O will not be used (not printed for covered scans).
data_iosize	The I/O size to be used on the data pages (not printed for covered scans).
data_bufreplace	The buffer replacement strategy to be used on the data pages (not printed for covered scans).
scanlio	Estimated total logical I/O for a serial query.
scanpio	Estimated total physical I/O for a serial query.
scanlio_perthrd	Estimated logical I/O per thread, for a parallel query.
tot_scanlio	Estimated total logical I/O, for a parallel query.
scanpio_perthrd	Estimated physical I/O per thread, for a parallel query.
tot_scanpio	Estimated total physical I/O, for a parallel query.
outer_srtmrglio_perthrd	Estimated logical I/O on the outer table to perform the sort-merge, per thread.
tot_outer_srtmrglio	Estimated total logical I/O on the outer table to perform a sort-merge.
inner_srtmrglio_perthrd	Estimated logical I/O on the inner table to perform a sort-merge join, per thread.
tot_inner_srtmrglio	Estimated total logical I/O on the inner table to perform a sort-merge join.
corder	The order of the column used as a search argument or join key.
jnvar	The *varno* of the table to which this table is being joined, for second and subsequent tables in a join.
refcost	The total cost of reformatting, when reformatting is considered as an access method.
refpages	The number of pages read in each scan of the table created for formatting. Included for the second and subsequent tables in the join order.
reftotpages	The number of pages in the table created for formatting. Included for the second and subsequent tables in the join order.

Label	Information provided
ordercol[0]	The order of the join column from the inner table.
ordercol[1]	The order of the join column from the outer table.

Table 7-2 shows the access methods that correspond to the *pathtype* information in the dbcc traceon(310) output.

Table 7-2: pathtypes in dbcc traceon(310) output

pathtype	Access method
sclause	Search clause
join	Join
orstruct	or clause
join-sort	Join, using a sort-avert index
sclause-sort	Search clause, using a sort-avert index
pll-sarg-nc	Parallel index hash scan on a search clause
pll-join-nc	Parallel index hash scan on a join clause
pll-sarg-cl	Parallel clustered index scan on a search clause
pll-join-cl	Parallel clustered index scan on a join
pll-sarg-cp	Parallel partitioned clustered index scan on a search clause
pll-join-cp	Parallel partitioned clustered index scan on a join clause
pll-partition	Parallel partitioned table scan
pll-nonpart	Parallel nonpartitioned table scan
pll-mrg-scan-inner	Parallel sort-merge join, with this table as the inner table
pll-mrg-scan-outer	Parallel sort-merge join, with this table as the outer table

Sort-merge costs

If the query plan includes a sort-merge join, the cost of creating the worktables and sorting them are printed. These messages include the total cost that is added to the query cost:

```
Sort-Merge Cost of Inner = 538
Sort-Merge Cost of Outer = 5324
```

These are the total costs of performing the sort-merge work, representing the logical I/O on the worktables multiplied by 2.

CHAPTER 8

Monitoring Performance with *sp_sysmon*

This chapter describes output from sp_sysmon, a system procedure that produces Adaptive Server performance data. It includes suggestions for interpreting its output and deducing possible implications.

sp_sysmon output is most valuable when you have a good understanding of your Adaptive Server environment and its specific mix of applications. Otherwise, you may find that sp_sysmon output has little relevance.

Using

When you invoke sp_sysmon, it clears all accumulated data from a set of counters that will be used during the sample interval to accumulate the results of user and system activity. At the end of the sample interval, the procedure reads the values in the counters, prints the report, and stops executing.

sp_sysmon contributes 5 to 7% overhead while it runs on a single CPU server, and more on multiprocessor servers. The amount of overhead increases with the number of CPUs.

Warning! sp_sysmon and Adaptive Server Monitor use the same internal counters. sp_sysmon resets these counters to 0, producing erroneous output for Adaptive Server Monitor when it is used simultaneously with sp_sysmon.

Also, starting a second execution of sp_sysmon while an earlier execution is running clears all the counters, so the first iteration of reports will be inaccurate.

Our performance tuning tips are based on the sampling interval. You need to review any recommendations thoroughly, based on your system requirements, before you incorporate them in your production system. You should set up a test area with your data and test any changes before implementing.

Also since sp_sysmon only gets a snapshot view of the system, these recommendations might not be applicable when the workload changes.

When to run

You can run sp_sysmon both before and after tuning Adaptive Server configuration parameters to gather data for comparison. This data gives you a basis for performance tuning and lets you observe the results of configuration changes.

Use sp_sysmon when the system exhibits the behavior you want to investigate. For example, if you want to find out how the system behaves under typically loaded conditions, run sp_sysmon when conditions are normal and typically loaded.

In this case, it would not make sense to run sp_sysmon for 10 minutes starting at 7:00 p.m., before the batch jobs begin and after most of the day's OLTP users have left the site. Instead, it would be best to run sp_sysmon both during the normal OLTP load and during batch jobs.

In many tests, it is best to start the applications, and then start sp_sysmon when the caches have had a chance to reach a steady state. If you are trying to measure capacity, be sure that the amount of work you give the server keeps it busy for the duration of the test.

Many of the statistics, especially those that measure data per second, can look extremely low if the server is idle during part of the sample interval.

In general, sp_sysmon produces valuable information when you use it:

- Before and after cache or pool configuration changes
- Before and after certain sp_configure changes
- Before and after the addition of new queries to your application mix
- Before and after an increase or decrease in the number of Adaptive Server engines
- When adding new disk devices and assigning objects to them
- During peak periods, to look for contention or bottlenecks
- During stress tests to evaluate an Adaptive Server configuration for a maximum expected application load
- When performance seems slow or behaves abnormally

It can also help with micro-level understanding of certain queries or applications during development. Some examples are:

- Working with indexes and updates to see if certain updates reported as deferred_varcol are resulting direct vs. deferred updates
- Checking caching behavior of particular queries or a mix of queries
- Tuning the parameters and cache configuration for parallel index creation

Invoking

There are two ways to use sp_sysmon:

- Using a fixed time interval to provide a sample for a specified number of minutes
- Using the begin_sample and end_sample parameters to start and stop sampling

You can also tailor the output to provide the information you need:

- You can print the entire report.
- You can print just one section of the report, such as "Cache Management" or "Lock Management."

Note The Cache Wizard section is a special section of the report. You need to specify the section for Cache Wizard to get output on it. See "Output" on page 208

- You can include application-level detailed reporting for named applications (such as isql, bcp, or any named application) and for combinations of named applications and user names. (The default is to omit this section.)

Fixed time intervals

To invoke sp_sysmon, execute the following command using isql:

```
sp_sysmon interval [, section  [, applmon]]
```

interval must be in the form "hh:mm:ss". To run sp_sysmon for 10 minutes, use this command:

```
sp_sysmon "00:10:00"
```

The following command prints only the "Data Cache Management" section of the report:

```
sp_sysmon "00:10:00", dcache
```

For information on the *applmon* parameter, see "Specifying the application detail parameter" on page 202.

Using *begin_sample* and *end_sample*

With the begin_sample and end_sample parameters, you can invoke sp_sysmon to start sampling, issue queries, and end the sample and print the results at any point in time. For example:

```
sp_sysmon begin_sample
execute proc1
execute proc2
select sum(total_sales) from titles
```

```
sp_sysmon end_sample
```

Note On systems with many CPUs and high activity, counters can overflow if the sample period is too long.

If you see negative results in your sp_sysmon output, reduce your sample time.

Specifying report sections for output

To print only a single section of the report, use one of the values listed in Table 8-1 for the second parameter.

Table 8-1: sp_sysmon report sections

Report section	Parameter
Application Management	appmgmt
Cache Wizard	cache wizard
Data Cache Management	dcache
Disk I/O Management	diskio
ESP Management	esp
Houskeeper Task Activity	housekeeper
Index Management	indexmgmt
Kernel Utilization	kernel
Lock Management	locks
Memory Management	memory
Metadata Cache Management	mdcache*
Monitor Access to Executing SQL	monaccess
Network I/O Management	netio
Parallel Query Management	parallel
Procedure Cache Management	pcache
Recovery Management	recovery
Task Management	taskmgmt
Transaction Management	xactmgmt
Transaction Profile	xactsum
Worker Process Management	wpm

* Most of the information available through sp_sysmon_mdcache can be obtained throu using sp_monitorconfig.

Specifying the application detail parameter

If you specify the third parameter to sp_sysmon, the report includes detailed information by application or by application and login name. This parameter is valid only when you print the entire report or when you request the Application Management section by specifying appmgmt as the section. It is ignored if you specify it and request any other section of the report.

The third parameter must be one of the following:

Parameter	Information reported
appl_only	CPU, I/O, priority changes, and resource limit violations by application name.
appl_and_login	CPU, I/O, priority changes, and resource limit violations by application name and login name. Can be used with all sections.
no_appl	Skips the application and login section of the report. This is the default.

This example runs sp_sysmon for 5 minutes and prints the "Application Management" section, including the application and login detail report:

```
sp_sysmon "00:05:00", appmgmt, appl_and_login
```

See "Per application or per application and login" on page 239 for sample output.

Cache Wizard syntax

To obtain output from the Cache Wizard:

```
sp_sysmon begin_sample
sp_sysmon { end_sample | interval }
[, 'cache wizard'  [, top_N [, filter] ] ]
```

Parameters

Use these parameters:

- top_N

  A varchar datatype that limits the list of objects reported in the Object Section based on the ranking criteria for the number of logical reads in the specified interval (as displayed in the LR/sec column).

The order of ranking is ascending or descending based on whether the specified value is a positive or negative integer. The entire list of objects occupying the cache at the end of the interval can be obtained by specifying a value of '0'. The default value 10.

- Filter

  A varchar datatype that allows you to specify a pattern for the cache(s) included in the report.

  For example, if it is specified as default data cache, the report will only contain information about the default data cache. If it is specified as emp%, the output includes information on all caches with a name matching this pattern.

  If no value is given the output contains all the caches with the default data cache appearing first, followed by the other caches in alphabetical order

For more information, see "Output" on page 208 for the Cache Wizard.

Redirecting output to a file

A full sp_sysmon report contains hundreds of lines of output. Use isql input and output redirect flags to save the output to a file.

See the *Utility Programs* manual for more information on isql.

How to use the reports

sp_sysmon can give you information about Adaptive Server system behavior both before and after tuning. It is important to study the entire report to understand the full impact of the changes you make. Sometimes removing one performance bottleneck reveals another.

It is also possible that your tuning efforts might improve performance in one area, while actually causing performance degradation in another area.

In addition to pointing out areas for tuning work, sp_sysmon output is valuable for determining when further tuning will not pay off in additional performance gains.

It is just as important to know when to stop tuning Adaptive Server, or when the problem resides elsewhere, as it is to know what to tune.

Other information can contribute to interpreting sp_sysmon output:

- Information on the configuration parameters in use, from sp_configure or the configuration file
- Information on the cache configuration and cache bindings, from sp_cacheconfig and sp_helpcache
- Information on disk devices, segments, and the objects stored on them

Reading output

sp_sysmon displays performance statistics in a consistent tabular format. For example, in an SMP environment running nine Adaptive Server engines, the output typically looks like this:

```
Engine Busy Utilization:
    Engine 0                      98.8 %
    Engine 1                      98.8 %
    Engine 2                      97.4 %
    Engine 3                      99.5 %
    Engine 4                      98.7 %
    Engine 5                      98.7 %
    Engine 6                      99.3 %
    Engine 7                      98.3 %
    Engine 8                      97.7 %
  -----------             ---------------          ----------------
  Summary:                Total:  887.2 %          Average:  98.6 %
```

Rows

Most rows represent a specific type of activity or event, such as acquiring a lock or executing a stored procedure. When the data is related to CPUs, the rows show performance information for each Adaptive Server engine in the SMP environment. Often, when there are groups of related rows, the last row is a summary of totals and an average.

The sp_sysmon report indents some rows to show that one category is a subcategory of another. In the following example, “Found in Wash” is a subcategory of “Cache Hits”, which is a subcategory of “Cache Searches”:

```
Cache Searches
      Cache Hits                202.1        3.0    12123     100.0 %
         Found in Wash            0.0        0.0        0       0.0 %
      Cache Misses                0.0        0.0        0       0.0 %
```

```
-------------------------  ---------  ---------  -------
  Total Cache Searches          202.1        3.0    12123
```

Many rows are not printed when the “count” value is 0.

Columns

Unless otherwise stated, the columns represent the following performance statistics:

- “per sec”– average per second during sampling interval
- “per xact” – average per committed transaction during sampling interval
- “count” – total number during the sample interval
- “% of total” – varies, depending on context, as explained for each occurrence

Interpreting the data

When tuning Adaptive Server, the fundamental measures of success appear as increases in throughput and reductions in application response time. Unfortunately, tuning Adaptive Server cannot be reduced to printing these two values.

In most cases, your tuning efforts must take an iterative approach, involving a comprehensive overview of Adaptive Server activity, careful tuning and analysis of queries and applications, and monitoring locking and access on an object-by-object basis.

Per second and per transaction data

Weigh the importance of the per second and per transaction data on the environment and the category you are measuring. The per transaction data is generally more meaningful in benchmarks or in test environments where the workload is well defined.

It is likely that you will find per transaction data more meaningful for comparing test data than per second data alone because in a benchmark test environment, there is usually a well-defined number of transactions, making comparison straightforward. Per transaction data is also useful for determining the validity of percentage results.

Percent of total and count data

The meaning of the "% of total" data varies, depending on the context of the event and the totals for the category. When interpreting percentages, keep in mind that they are often useful for understanding general trends, but they can be misleading when taken in isolation.

For example, 50% of 200 events is much more meaningful than 50% of 2 events.

The "count" data is the total number of events that occurred during the sample interval. You can use count data to determine the validity of percentage results.

Per engine data

In most cases, per engine data for a category shows a fairly even balance of activity across all engines. Two exceptions are:

- If you have fewer processes than CPUs, some of the engines will show no activity.
- If most processes are doing fairly uniform activity, such as simple inserts and short selects, and one process performs some I/O intensive operation such as a large bulk copy, you will see unbalanced network and disk I/O.

Total or summary data

Summary rows provide an overview of Adaptive Server engine activity by reporting totals and averages.

Be careful when interpreting averages because they can give false impressions of true results when the data is skewed. For example, if one Adaptive Server engine is working 98% of the time and another is working 2% of the time, a 49% average can be misleading.

Sample interval and time reporting

The heading of an sp_sysmon report includes the software version, server name, run date, the date and time the sample interval started, the time it completed, and the duration of the sample interval.

```
===============================================================================
Sybase Adaptive Server Enterprise System Performance Report
```

```
===============================================================================
Server Version:       Adaptive Server Enterprise/12.5.1/XXXXX/P/Sun_svr4/OS 5.
Server Name:           Server is coffee
Run Date:              Jul 17, 2003
Statistics Cleared at: Jul 17, 2003 09:02:35
Statistics Sampled at: Jul 17, 2003 09:04:35
Sample Interval:       00:02:00

===============================================================================
```

Cache Wizard

The Cache Wizard section can aid in the monitoring and configuring of data caches for optimal performance.

Cache Wizard allows you to identify:

- Hot objects (objects that are often accessed). The output is ranked by the number of logical reads in a named cache or default data cache.
- Spinlock contention on the cache.
- The usage of the cache and buffer pools.
- The percentage of hits at a cache, buffer pool and object level. (THIS SENTENCE NEEDS TO BE MADE MORE CLEARER.
- The effectiveness of large I/O.
- The effectiveness of APF.
- The cache occupancy by the various objects.

The Cache Wizard section appears in the sp_sysmon output only when you include the cache wizard parameter. You can include two parameters with Cache Wizard, topN and filter. See syntax details on “Cache Wizard syntax” on page 202.

A Cache Wizard recommendation section is printed at the end of the output on this report.

Before you run sp_sysmon with the Cache Wizard section, you must first install the monitoring tables.

Preparing to run the cache wizard

sp_sysmon retrieves the information for the Cache Wizard from the monitoring tables and monitor counters.

To install the monitoring tables:

- Add the loopback server to sysservers:

  sp_addserver loopback, NULL, <*srvnetname*>

- Grant the mon_role role to sa_role:

  sp_role 'grant', mon_role, sa_role

- Install the monitoring tables with the installmontables script (located in *$SYBASE/ASE-12_5/scripts*):

  ```
  $SYBASE/$SYBASE_OCS/bin/isql -Usa -P -i
  $SYBASE/$SYBASE_ASE/scripts/installmontables
  ```

- The Cache Wizard requires the following configuration parameters to be enabled. If these configuration parameters are not enabled, sp_sysmon automatically enables them at the beginning of the interval and disables them at the end.

  enable monitoring - set to 1. This option is dynamic.
  per object statistics active - set to 1. This option is dynamic.

For more information about configuration parameters, see *Chapter 5, "Setting Configuration Parameters" in the System Administration Guide*.

Output

The Cache Wizard output contains five main sections for each cache. This is followed by a recommendation section and a legend section at the end of the report. The three main sections for each cache are:

- cache section provides summary statistics for a specific cache:

```
------------------
default data cache
------------------
Run Size        :   100.00 Mb   Usage%              :      2.86
LR/sec          :    41.10     PR/sec              :     22.57  Hit%:  45.09
Cache Partitions:        4      Spinlock Contention%:      0.00
```

Usage%

Each time a set of pages is brought into the cache, it is tracked track if that page gets is referenced (used). Once the page is removed from the cache this count gets reduced. This value gives the current usage of the cache as a % of the cache size.

LR/sec

A logical read is any read in a read from the cache (hit) or a disk read (miss). LR/sec is the number of logical reads in the cache during the interval divided by the sample interval.

PR/sec

A physical read is a read from disk (miss). PR/sec is the number of physical reads in the cache during the interval divided by the sample interval.

Hit%

Ratio of the hits to total cache reads, such as the ratio of (LR/sec - PR/sec) to LR/sec

- Buffer pool section breaks down the 'cache section' statistics into the various buffer pools in the cache.

```
Buffer Pool Information
-------------------------------------------------------------------------
IO Size Wash Size Run Size APF%   LR/sec   PR/sec   Hit%   APF-Eff% Usage%
------- ---------- ------- ------ ------- -------- ------ -------- ------
4 Kb     3276 Kb    16.00 Mb  10.00     0.47     0.13  71.43    n/a     0.20
2 Kb    17200 Kb    84.00 Mb  10.00    40.63    22.43  44.79    n/a     3.37
```

APF-Eff% ratio between pages brought for the account of Asynchronous Prefetch (APF) and used, to the number of pages brought in account of APF.

Usage% is similar to the Cache section, tracks if a page brought into the buffer pool is referenced or not. This gives the ratio of the pages referenced in the buffer pool to the run size of the buffer pool.

- Object section reports statistics on the objects occupying the cache at the end of the interval. The output of this section can be limited by using the topN parameter. The objects are always displayed in the ascending order of PR/sec.

```
Object Statistics
-------------------------------------------------------------------------
Object                      LR/sec  PR/sec  Hit%   Obj_Cached% Cache_Occp%
----------------------------- ------- ------- ------ ----------- -----
```

```
empdb.dbo.t1                    0.57    0.30  47.06      56.25       0.02
empdb.dbo.t2                    0.30    0.30   0.00      56.25       0.02
empdb.dbo.t3                    0.30    0.30   0.00      56.25       0.02

Object                           Obj Size    Size in Cache
------------------------------ ----------- -------------
empdb.dbo.t1                         32 Kb       18 Kb
empdb.dbo.t2                         32 Kb       18 Kb
empdb.dbo.t3                         32 Kb       18 Kb
```

- Recommendations section gives a set of recommendations where applicable based on the data collected in the sample interval:

```
The various recommendations are as follows:
Usage% for 'default data cache' is low (< 5%)
Usage% for  4k buffer pool in cache:default data cache is low (< 5%)
Consider using Named Caches or creating more cache partitions for
'default data cache' or both
Consider increasing the 'wash size' of the 2k pool for 'default data
cache'
Consider adding a large I/O pool for 'default data cache'
```

- Legend explains the various terms used in the output. Some of the terms from the output are explained here in greater detail.

Sample output for Cache Wizard

```
sp_sysmon '00:00:30', 'cache wizard'

===============================================================================
Cache Wizard
===============================================================================

------------------
default data cache
------------------
Run Size        :   100.00 Mb   Usage%              :      2.86
LR/sec          :    41.10      PR/sec              :     22.57   Hit%:  45.09
Cache Partitions:        4      Spinlock Contention%:      0.00

Buffer Pool Information
-------------------------------------------------------------------------------
IO Size Wash Size  Run Size    APF%   LR/sec   PR/sec   Hit%   APF-Eff% Usage%
------- ---------- ----------- ------ -------- -------- ------ -------- ------
4 Kb     3276 Kb    16.00 Mb  10.00     0.47     0.13  71.43    n/a     0.20
2 Kb    17200 Kb    84.00 Mb  10.00    40.63    22.43  44.79    n/a     3.37
```

```
(1 row affected)

Object Statistics

-------------------------------------------------------------------------------
Object                          LR/sec  PR/sec  Hit%   Obj_Cached% Cache_Occp%
------------------------------- ------- ------- ------ ----------- -----------
empdb.dbo.t1                      0.57    0.30  47.06       56.25        0.02
empdb.dbo.t2                      0.30    0.30   0.00       56.25        0.02
empdb.dbo.t3                      0.30    0.30   0.00       56.25        0.02
empdb.dbo.t4                      0.30    0.30   0.00       56.25        0.02
empdb.dbo.t5                      0.30    0.30   0.00       56.25        0.02
empdb.dbo.t6                      0.30    0.30   0.00       56.25        0.02
empdb.dbo.t8                      0.30    0.30   0.00       56.25        0.02
empdb.dbo.t7                      0.57    0.20  64.71       62.50        0.02
tempdb.dbo.tempcachedobjstats     3.63    0.00 100.00       50.00        0.01
tempdb.dbo.tempobjstats           0.47    0.00 100.00       25.00        0.00

Object                          Obj Size    Size in Cache
------------------------------- ----------- -------------
empdb.dbo.t1                         32 Kb        18 Kb
empdb.dbo.t2                         32 Kb        18 Kb
empdb.dbo.t3                         32 Kb        18 Kb
empdb.dbo.t4                         32 Kb        18 Kb
empdb.dbo.t5                         32 Kb        18 Kb
empdb.dbo.t6                         32 Kb        18 Kb
empdb.dbo.t8                         32 Kb        18 Kb
empdb.dbo.t7                         32 Kb        20 Kb
tempdb.dbo.tempcachedobjstats        16 Kb         8 Kb
tempdb.dbo.tempobjstats              16 Kb         4 Kb

---------
company_cache
---------
Run Size        :     1.00 Mb   Usage%              :      0.39
LR/sec          :     0.07      PR/sec             :      0.07   Hit%:   0.00
Cache Partitions:        1      Spinlock Contention%:      0.00

Buffer Pool Information
-------------------------------------------------------------------------------
IO Size Wash Size  Run Size    APF%   LR/sec   PR/sec   Hit%   APF-Eff% Usage%
------- ---------- ----------- ------ -------- -------- ------ -------- ------
2 Kb      204 Kb     1.00 Mb  10.00     0.07     0.07   0.00    n/a     0.39

Object Statistics
```

```
------------------------------------------------------------------------------

Object               LR/sec  PR/sec  Hit%   Obj_Cached% Cache_Occp%
-------------------- ------- ------- ------ ----------- -----------
empdb.dbo.history      0.07    0.07   0.00       25.00        0.39

Object               Obj Size    Size in Cache
-------------------- ----------- -------------
empdb.dbo.history         16 Kb        4 Kb

-------------
companydb_cache
-------------
Run Size        :     5.00 Mb   Usage%              :    100.00
LR/sec          :   380.97      PR/sec              :     56.67   Hit%:  85.13
Cache Partitions:        1      Spinlock Contention%:      0.00

Buffer Pool Information
------------------------------------------------------------------------------
IO Size Wash Size  Run Size    APF%   LR/sec   PR/sec   Hit%   APF-Eff% Usage%
------- ---------- ----------- ------ -------- -------- ------ -------- ------
2 Kb     1024 Kb     5.00 Mb  10.00   380.97    56.67  85.13  98.42   100.00


Object Statistics
------------------------------------------------------------------------------

Object                         LR/sec  PR/sec  Hit%   Obj_Cached% Cache_Occp%
------------------------------ ------- ------- ------ ----------- -----------
company_db.dbo.emp_projects     41.07   22.80  44.48       19.64        9.45
company_db.dbo.dept_det         93.03   20.67  77.79       99.08       54.53
company_db.dbo.emp_perf        116.70    2.63  97.74       97.77       34.18
company_db.dbo.dept_locs         0.43    0.17  61.54       50.00        0.16

Object                         Obj Size    Size in Cache
------------------------------ ----------- -------------
company_db.dbo.emp_projects       2464 Kb      484 Kb
company_db.dbo.dept_det           2818 Kb     2792 Kb
company_db.dbo.emp_perf           1790 Kb     1750 Kb
company_db.dbo.dept_locs            16 Kb        8 Kb

TUNING RECOMMENDATIONS
------------------------------------------------------------------------
Usage% for 'default data cache' is low (< 5%)
Usage% for  4k buffer pool in cache:default data cache is low (< 5%)
Usage% for  2k buffer pool in cache:default data cache is low (< 5%)
```

```
Usage% for 'company_cache' is low (< 5%)
 Usage% for  2k buffer pool in cache:company_cache is low (< 5%)

 Consider adding a large I/O pool for 'companydb_cache'


LEGEND
------
LR/sec    - number of logical reads per second, i.e. sum of cache & disk reads
PR/sec    - number of physical reads per second i.e. disk reads
Run Size- size of cache or buffer pool in Kilobytes
Cache Partitions- number of cache partitions
Spinlock Contention%- Percentage spinlock contention for the cache
Hit%      - ratio of hits to total searches
Usage%    - ratio of pages referenced to Run Size

Wash Size- wash size of buffer pool in Kilobytes
APF%      - asynchronous prefetch % for this buffer pool
APF-Eff%- Ratio of buffers found in cache and brought in because
               of APF to the number of APF disk reads performed

Object    - combination of db, owner, object and index name
Obj Size- size of the object in Kilobytes
Size in Cache- size occupied in cache in Kilobytes at the end of sample
Obj_Cached%- Ratio of 'Size in Cache' to 'Obj Size'
Cache_Occp%- Ratio of 'Size in Cache' to 'Run Size' of cache
```

Kernel utilization

"Kernel Utilization" reports Adaptive Server activities. It tells you how busy Adaptive Server engines were during the time that the CPU was available to Adaptive Server, how often the CPU yielded to the operating system, the number of times that the engines checked for network and disk I/O, and the average number of I/Os they found waiting at each check.

Sample output

The following sample shows sp_sysmon output for "Kernel Utilization" in an environment with eight Adaptive Server engines.

```
Kernel Utilization
------------------

Your Runnable Process Search Count is set to 2000
and I/O Polling Process Count is set to 10
```

In this example, the CPU did not yield to the operating system, so there are no detail rows.

Engine busy utilization

"Engine Busy Utilization" reports the percentage of time the Adaptive Server Kernel is busy executing tasks on each Adaptive Server engine (rather than time spent idle). The summary row gives the total and the average active time for all engines combined.

The values reported here may differ from the CPU usage values reported by operating system tools. When Adaptive Server has no tasks to process, it enters a loop that regularly checks for network I/O, completed disk I/Os, and tasks in the run queue.

Operating system commands to check CPU activity may show high usage for a Adaptive Server engine because they are measuring the looping activity, while "Engine Busy Utilization" does not include time spent looping—it is considered idle time.

One measurement that cannot be made from inside Adaptive Server is the percentage of time that Adaptive Server had control of the CPU vs. the time the CPU was in use by the operating system. Check your operating system documentation for the correct commands.

If you want to reduce the time that Adaptive Server spends checking for I/O while idle, you can lower the sp_configure parameter runnable process search count. This parameter specifies the number of times a Adaptive Server engine loops looking for a runnable task before yielding the CPU.

For more information, see the *System Administration Guide.*

"Engine Busy Utilization" measures how busy Adaptive Server engines were during the CPU time they were given. If the engine is available to Adaptive Server for 80% of a 10-minute sample interval, and "Engine Busy Utilization" was 90%, it means that Adaptive Server was busy for 7 minutes and 12 seconds and was idle for 48 seconds.

This category can help you decide whether there are too many or too few Adaptive Server engines. Adaptive Server's high scalability is due to tunable mechanisms that avoid resource contention.

By checking sp_sysmon output for problems and tuning to alleviate contention, response time can remain high even at "Engine Busy" values in the 80 to 90% range. If values are consistently very high (more than 90%), it is likely that response time and throughput could benefit from an additional engine.

The "Engine Busy Utilization" values are averages over the sample interval, so very high averages indicate that engines may be 100% busy during part of the interval.

When engine utilization is extremely high, the housekeeper wash task writes few or no pages out to disk (since it runs only during idle CPU cycles.) This means that a checkpoint finds many pages that need to be written to disk, and the checkpoint process, a large batch job, or a database dump is likely to send CPU usage to 100% for a period of time, causing a perceptible dip in response time.

If the "Engine Busy Utilization" percentages are consistently high, and you want to improve response time and throughput by adding Adaptive Server engines, check for increased resource contention in other areas after adding each engine.

In an environment where Adaptive Server is serving a large number of users, performance is usually fairly evenly distributed across engines. However, when there are more engines than tasks, you may see some engines with a large percentage of utilization, and other engines may be idle. On a server with a single task running a query, for example, you may see output like this:

```
Engine Busy Utilization         CPU Busy   I/O Busy       Idle
------------------------        --------   --------   --------
   Engine 0                         0.0 %      0.2 %     99.8 %
   Engine 1                         0.0 %
-----------           ---------------           ----------------
  Summary                Total    97.2 %           Average   16.2 %
```

In an SMP environment, tasks have soft affinity to engines. Without other activity (such as lock contention) that could cause this task to be placed in the global run cue, the task continues to run on the same engine.

CPU yields by engine

"CPU Yields by Engine" reports the number of times each Adaptive Server engine yielded to the operating system. "% of total" data is the percentage of times an engine yielded as a percentage of the combined yields for all engines.

"Total CPU Yields" reports the combined data over all engines.

If the "Engine Busy Utilization" data indicates low engine utilization, use "CPU Yields by Engine" to determine whether the "Engine Busy Utilization" data reflects a truly inactive engine or one that is frequently starved out of the CPU by the operating system.

When an engine is not busy, it yields to the CPU after a period of time related to the runnable process search count parameter. A high value for "CPU Yields by Engine" indicates that the engine yielded voluntarily.

If you also see that "Engine Busy Utilization" is a low value, then the engine really is inactive, as opposed to being starved out.

Engine Busy CPY Yields :

Low	Low	Engine is CPU starved
Low	High	Engine inactive
High	Low	
High	High	Engine busy/active

See the *System Administration Guide* for more information.

```
CPU Yields by Engine             per sec      per xact       count  % of total
```

```
-------------------------  ------------  ------------  ----------  ----------
Total CPU Yields                    0.0           0.0           0       n/a
```

Network checks

"Network Checks" includes information about blocking and non-blocking network I/O checks, the total number of I/O checks for the interval, and the average number of network I/Os per network check.

Adaptive Server has two ways to check for network I/O: blocking and non-blocking modes.

```
Network Checks
   Non-Blocking                   683.5       20503.5       82014       93.2 %
   Blocking                        49.5        1484.5        5938        6.8 %
-------------------------  ------------  ------------  ----------
  Total Network I/O Checks        732.9       21988.0       87952
  Avg Net I/Os per Check            n/a           n/a     0.00000       n/a
```

Non–blocking

"Non-Blocking" reports the number of times Adaptive Server performed non-blocking network checks. With non-blocking network I/O checks, an engine checks the network for I/O and continues processing, whether or not it found I/O waiting.

Blocking

"Blocking" reports the number of times Adaptive Server performed blocking network checks.

After an engine completes a task, it loops waiting for the network to deliver a runnable task. After a certain number of loops (determined by the sp_configure parameter runnable process search count), the Adaptive Server engine goes to sleep after a blocking network I/O.

When an engine yields to the operating system because there are no tasks to process, it wakes up once per clock tick to check for incoming network I/O. If there is I/O, the operating system blocks the engine from active processing until the I/O completes.

If an engine has yielded to the operating system and is doing blocking checks, it might continue to sleep for a period of time after a network packet arrives. This period of time is referred to as the *latency period.* You can reduce the latency period by increasing the runnable process search count parameter so that the Adaptive Server engine loops for longer periods of time.

See the *System Administration Guide* for more information.

Total network I/O checks

"Total Network I/O Checks" reports the number of times an engine polls for incoming and outgoing packets. This category is helpful when you use it with "CPU Yields by Engine."

When an engine is idle, it loops while checking for network packets. If "Network Checks" is low and "CPU Yields by Engine" is high, the engine could be yielding too often and not checking the network frequently enough. If the system can afford the overhead, it might be acceptable to yield less often.

Average network I/Os per check

"Avg Net I/Os per Check" reports the average number of network I/Os (both sends and receives) per check for all Adaptive Server engine checks that took place during the sample interval.

The sp_configure parameter i/o polling process count specifies the maximum number of processes that Adaptive Server runs before the scheduler checks for disk and/or network I/O completions. Tuning i/o polling process count affects both the response time and throughput of Adaptive Server.

See the *System Administration Guide*.

If Adaptive Server engines check frequently, but retrieve network I/O infrequently, you can try reducing the frequency for network I/O checking.

Disk I/O checks

This section reports the total number of disk I/O checks, and the number of checks returning I/O.

```
Disk I/O Checks
  Total Disk I/O Checks          732.9       21988.0       87952       n/a
  Checks Returning I/O            83.8        2512.5       10050      11.4 %
  Avg Disk I/Os Returned           n/a           n/a     0.00020       n/a
```

Total disk I/O checks

"Total Disk I/O Checks" reports the number of times engines checked for disk I/O.

When a task needs to perform I/O, the Adaptive Server engine running that task immediately issues an I/O request and puts the task to sleep, waiting for the I/O to complete. The engine processes other tasks, if any, but also loops to check for completed I/Os. When the engine finds completed I/Os, it moves the task from the sleep queue to the run queue.

Checks returning I/O

"Checks Returning I/O" reports the number of times that a requested I/O had completed when an engine checked for disk I/O.

For example, if an engine checks for expected I/O 100,000 times, this average indicates the percentage of time that there actually was I/O pending. If, of those 100,000 checks, I/O was pending 10,000 times, then 10% of the checks were effective, and the other 90% were overhead.

However, you should also check the average number of I/Os returned per check and how busy the engines were during the sample interval. If the sample includes idle time, or the I/O traffic is "bursty," it is possible that during a high percentage of the checks were returning I/O during the busy period.

If the results in this category seem low or high, you can configure i/o polling process count to increase or decrease the frequency of the checks.

See the *System Administration Guide*.

Average disk I/Os returned

"Avg Disk I/Os Returned" reports the average number of disk I/Os returned over all Adaptive Server engine checks combined.

Increasing the amount of time that Adaptive Server engines wait between checks may result in better throughput because Adaptive Server engines can spend more time processing if they spend less time checking for I/O. However, you should verify this for your environment. Use the sp_configure parameter i/o polling process count to increase the length of the checking loop.

See the *System Administration Guide*.

Worker process management

"Worker Process Management" reports the use of worker processes, including the number of worker process requests that were granted and denied and the success and failure of memory requests for worker processes.

You need to analyze this output in combination with the information reported under "Parallel query management" on page 222.

Sample output

```
===============================================================================

Worker Process Management
-------------------------
                               per sec      per xact       count  % of total
                              ------------  ------------  ----------  -------
 Worker Process Requests
   Total Requests                    0.0           0.0           0       n/a

 Worker Process Usage
   Total Used                        0.0           0.0           0       n/a
   Max Ever Used During Sample       0.0           0.0           0       n/a

 Memory Requests for Worker Processes
   Total Requests                    0.0           0.0           0       n/a

Avg Mem Ever Used by a WP
             (in bytes) n/a           n/a      311.7     n/a        n/a
```

Worker process requests

This section reports requests for worker processes and worker process memory. A parallel query may make multiple requests for worker processes. For example, a parallel query that requires a sort may make one request for accessing data and a second for parallel sort.

The "Requests Granted" and "Requests Denied" rows show how many requests were granted and how many requests were denied due to a lack of available worker processes at execution time.

To see the number of adjustments made to the number of worker processes, see "Parallel query usage" on page 223.

"Requests Terminated" reports the number of times a request was terminated by user action, such as pressing Ctrl-c, that cancelled the query.

Worker process usage

In this section, "Total Used" reports the total number of worker processes used during the sample interval. "Max Ever Used During Sample" reports the highest number in use at any time during sp_sysmon's sampling period. You can use "Max Ever Used During Sample" to set the configuration parameter number of worker processes.

Memory requests for worker processes

This section reports how many requests were made for memory allocations for worker processes, how many of those requests succeeded and how many failed. Memory for worker processes is allocated from a memory pool configured with the parameter memory per worker process.

If "Failed" is a nonzero value, you may need to increase the value of memory per worker process.

Avg mem ever used by a WP

This row reports the maximum average memory used by all active worker processes at any time during the sample interval. Each worker process requires memory, principally for exchanging coordination messages. This memory is allocated by Adaptive Server from the global memory pool.

The size of the pool is determined by multiplying the two configuration parameters, number of worker processes and memory per worker process.

If number of worker processes is set to 50, and memory per worker process is set to the default value of 1024 bytes, 50K is available in the pool. Increasing memory for worker process to 2048 bytes would require 50K of additional memory.

At start-up, static structures are created for each worker process. While worker processes are in use, additional memory is allocated from the pool as needed and deallocated when not needed. The average value printed is the average for all static and dynamically memory allocated for all worker processes, divided by the number of worker processes actually in use during the sample interval.

If a large number of worker processes are configured, but only a few are in use during the sample interval, the value printed may be inflated, due to averaging in the static memory for unused processes.

If "Avg Mem" is close to the value set by memory per worker process and the number of worker processes in "Max Ever Used During Sample" is close to the number configured, you may want to increase the value of the parameter.

If a worker process needs memory from the pool, and no memory is available, the process prints an error message and exits.

Note For most parallel query processing, the default value of 1024 is more than adequate.

The exception is dbcc checkstorage, which can use up 1792 bytes if only one worker process is configured. If you are using dbcc checkstorage, and number of worker processes is set to 1, you may want to increase memory per worker process.

Parallel query management

"Parallel Query Management" reports the execution of parallel queries. It reports the total number of parallel queries, how many times the number of worker processes was adjusted at runtime, and reports on the granting of locks during merges and sorts.

Sample output

```
===============================================================================

Parallel Query Management
-------------------------
```

```
Parallel Query Usage            per sec      per xact       count  % of total
------------------------  ------------  ------------  ----------  ----------
  Total Parallel Queries            0.0           0.0           0       n/a

  Merge Lock Requests            per sec      per xact       count  % of total
------------------------  ------------  ------------  ----------  ----------
  Total # of Requests               0.0           0.0           0       n/a

  Sort Buffer Waits              per sec      per xact       count  % of total
------------------------  ------------  ------------  ----------  ----------
  Total # of Waits                  0.0           0.0           0       n/a
```

Parallel query usage

"Total Parallel Queries" reports the total number of queries eligible to be run in parallel. The optimizer determines the best plan, deciding whether a query should be run serially or in parallel and how many worker processes should be used for parallel queries.

"WP Adjustments Made" reports how many times the number of worker processes recommended by the optimizer had to be adjusted at runtime. Two possible causes are reported:

- "Due to WP Limit" indicates the number of times the number of worker processes for a cached query plan was adjusted due to a session-level limit set with set parallel_degree or set scan_parallel_degree.

  If "Due to WP Limit" is a nonzero value, look for applications that set session-level limits.

- "Due to No WPs" indicates the number of requests for which the number of worker processes was reduced due to lack of available worker processes. These queries may run in serial, or they may run in parallel with fewer worker processes than recommended by the optimizer. It could mean that queries are running with poorly-optimized plans.

  If "Due to No WPs" is a nonzero value, and the sample was taken at a time of typical load on your system, you may want to increase the number of worker processes configuration parameter or set session-level limits for some queries.

  Running sp_showplan on the fid (family ID) of a login using an adjusted plan shows only the cached plan, not the adjusted plan.

If the login is running an adjusted plan, sp_who shows a different number of worker processes for the fid than the number indicated by sp_showplan results.

Merge lock requests

"Merge Lock Requests" reports the number of parallel merge lock requests that were made, how many were granted immediately, and how many had to wait for each type of merge. The three merge types are:

- "Network Buffer Merge Locks"–reports contention for the network buffers that return results to clients.
- "Result Buffer Merge Locks"–reports contention for the result buffers used to process results for ungrouped aggregates and nonsorted, non aggregate variable assignment results.
- "Work Table Merge Locks"–reports contention for locks while results from work tables were being merge.

"Total # of Requests" prints the total of the three types of merge requests.

Sort buffer waits

This section reports contention for the sort buffers used for parallel sorts. Parallel sort buffers are used by:

- Producers – the worker processes returning rows from parallel scans
- Consumers – the worker processes performing the parallel sort

If the number of waits is high, you can configure number of sort buffers to a higher value.

See "Sort buffer configuration guidelines" on page 225 in *Performance and Tuning: Optimizer* for guidelines.

Task management

"Task Management" provides information on opened connections, task context switches by engine, and task context switches by cause.

Sample output

The following sample shows sp_sysmon output for the "Task Management" categories.

```
================================================================================
Task Management                    per sec      per xact       count  % of total
--------------------------- ------------ ------------ ---------- ----------

  Connections Opened                   0.0          0.0          0       n/a

  Task Context Switches by Engine
   Engine 0                            0.6         16.5         66     100.0 %

 Task Context Switches Due To:
   Voluntary Yields                    0.2          7.0         28      42.4 %
   Cache Search Misses                 0.0          0.0          0       0.0 %
   System Disk Writes                  0.0          0.0          0       0.0 %
   I/O Pacing                          0.0          0.0          0       0.0 %
   Logical Lock Contention             0.0          0.0          0       0.0 %
   Address Lock Contention             0.0          0.0          0       0.0 %
   Latch Contention                    0.0          0.0          0       0.0 %
   Log Semaphore Contention            0.0          0.0          0       0.0 %
   PLC Lock Contention                 0.0          0.0          0       0.0 %
   Group Commit Sleeps                 0.0          0.0          0       0.0 %
   Last Log Page Writes                0.0          0.0          0       0.0 %
   Modify Conflicts                    0.0          0.0          0       0.0 %
   I/O Device Contention               0.0          0.0          0       0.0 %
   Network Packet Received             0.0          0.0          0       0.0 %
   Network Packet Sent                 0.0          0.0          0       0.0 %
   Other Causes                        0.3          9.5         38      57.6 %
```

Connections opened

"Connections Opened" reports the number of connections opened to Adaptive Server. It includes any type of connection, such as client connections and remote procedure calls. It counts only connections that were started during the sample interval.

Connections that were established before the interval started are not counted, although they may be active and using resources.

This provides a general understanding of the Adaptive Server environment and the work load during the interval. This data can also be useful for understanding application behavior – it can help determine if applications repeatedly open and close connections or perform multiple transactions per connection.

See "Transaction profile" on page 243 for information about committed transactions.

Task context switches by engine

"Task Context Switches by Engine" reports the number of times each Adaptive Server engine switched context from one user task to another. "% of total" reports the percentage of engine task switches for each Adaptive Server engine as a percentage of the total number of task switches for all Adaptive Server engines combined.

"Total Task Switches" summarizes task-switch activity for all engines on SMP servers. You can use "Total Task Switches" to observe the effect of re configurations. You might reconfigure a cache or add memory if tasks appear to block on cache search misses and to be switched out often. Then, check the data to see if tasks tend to be switched out more or less often.

Task context switches due to

"Task Context Switches Due To" reports the number of times that Adaptive Server switched context for a number of common reasons. "% of total" reports the percentage of times the context switch was due to each specific cause as a percentage of the total number of task context switches for all Adaptive Server engines combined.

"Task Context Switches Due To" provides an overview of the reasons that tasks were switched off engines. The possible performance problems shown in this section can be investigated by checking other sp_sysmon output, as indicated in the sections that describe the causes.

For example, if most of the task switches are caused by physical I/O, try minimizing physical I/O by adding more memory or re configuring caches. However, if lock contention causes most of the task switches, check the locking section of your report.

See "Lock management" on page 269 for more information.

Voluntary yields

"Voluntary Yields" reports the number of times a task completed or yielded after running for the configured amount of time. The Adaptive Server engine switches context from the task that yielded to another task.

The configuration parameter time slice sets the amount of time that a process can run. A CPU-intensive task that does not switch out due to other causes yields the CPU at certain "yield points" in the code, in order to allow other processes a turn on the CPU.

See "Scheduling client task processing time" on page 42 in *Performance and Tuning: Basics* for more information.

A high number of voluntary yields indicates that there is little contention.

Cache search misses

"Cache Search Misses" reports the number of times a task was switched out because a needed page was not in cache and had to be read from disk. For data and index pages, the task is switched out while the physical read is performed.

See "Data cache management" on page 278 for more information about the cache-related parts of the sp_sysmon output.

System disk writes

"System Disk Writes" reports the number of times a task was switched out because it needed to perform a disk write or because it needed to access a page that was being written by another process, such as the housekeeper or the checkpoint process.

Most Adaptive Server writes happen asynchronously, but processes sleep during writes for page splits, recovery, and OAM page writes.

If "System Disk Writes" seems high, check the value for page splits to see if the problem is caused by data page and index page splits.

See "Page splits" on page 259 for more information.

If the high value for system disk writes is not caused by page splitting, you cannot affect this value by tuning.

I/O pacing

"I/O Pacing" reports how many times an I/O-intensive task was switched off an engine due to exceeding an I/O batch limit. Adaptive Server paces disk writes to keep from flooding the disk I/O subsystems during certain operations that need to perform large amounts of I/O.

Two examples are the checkpoint process and transaction commits that write a large number of log pages. The task is switched out and sleeps until the batch of writes completes and then wakes up and issues another batch.

By default, the number of writes per batch is set to 10. You may want to increase the number of writes per batch if:

- You have a high-throughput, high-transaction environment with a large data cache
- Your system is not I/O bound

Valid values are from 1 to 50. This command sets the number of writes per batch to 20:

```
dbcc tune (maxwritedes, 20)
```

Logical lock contention

"Logical Lock Contention" reports the number of times a task was switched out due to contention for locks on tables, data pages, or data rows.

Investigate lock contention problems by checking the transaction detail and lock management sections of the report.

- See "Transaction detail" on page 246 and "Lock management" on page 269.
- Check to see if your queries are doing deferred and direct expensive updates, which can cause additional index locks.

  See "Updates" on page 248.
- Use sp_object_stats to report information on a per-object basis.

  See "Identifying tables where concurrency is a problem" on page 88 of *Performance and Tuning: Locking*.

For additional help on locks and lock contention, check the following sources:

- "Types of Locks" in the *System Administration Guide* provides information about types of locks to use at server or query level.

- "Reducing lock contention" on page 40 of *Performance and Tuning: Locking* provides pointers on reducing lock contention.
- Chapter 13, "Indexing for Performance," in *Performance and Tuning: Basics*, provides information on indexes and query tuning. In particular, use indexes to ensure that updates and deletes do not lead to table scans and exclusive table locks.

Address lock contention

"Address Lock Contention" reports the number of times a task was switched out because of address locks. Adaptive Server acquires address locks on index pages of allpages-locked tables. Address lock contention blocks access to data pages.

Latch contention

"Latch Contention" reports the number of times a task was switched out because it needed to wait for a latch.

If your user tables use only allpages-locking, this latch contention is taking place either on a data-only-locked system table or on allocation pages.

If your applications use data-only-locking, the contention reported here includes all waits for latches, including those on index pages and OAM pages as well as allocation pages.

Reducing contention during page allocation

In SMP environments where inserts and expanding updates are extremely high, so that page allocations take place very frequently, contention for the allocation page latch can reduce performance. Normally, Adaptive Server allocates new pages for an object on an allocation unit that is already in use by the object and known to have free space.

For each object, Adaptive Server tracks this allocation page number as a hint for any tasks that need to allocate a page for that object. When more than one task at a time needs to allocate a page on the same allocation unit, the second and subsequent tasks block on the latch on the allocation page.

You can specify a "greedy allocation" scheme, so that Adaptive Server keeps a list of eight allocation hints for page allocations for a table.

This command enables greedy allocation for the salesdetail table in database 6:

```
dbcc tune(des_greedyalloc, 6, salesdetail, "on")
```

To turn it off, use:

```
dbcc tune(des_greedyalloc, 6, salesdetail, "off")
```

The effect of dbcc tune(des_greedyalloc) are not persistent, so you need to reissue the commands after a reboot.

You should use this command only if all of the following are true:

- You have multiple engines. It is rarely useful with fewer than four engines.
- A large number of pages are being allocated for the object. You can use sp_spaceused or optdiag to track the number of pages.
- The latch contention counter shows contention.

Greedy allocation is more useful when tables are assigned to their own segments. If you enable greedy allocation for several tables on the same segment, the same allocation hint could be used for more than one table. Hints are unique for each table, but uniqueness is not enforced across all tables.

Greedy allocation is not allowed in the master and tempdb databases, and is not allowed on system tables. Greedy page allocation is not applicable to partitioned tables.

The hints are allocation pages which potentially have free space and that the maximum number of hints maintained is 16.

Log semaphore contention

"Log Semaphore Contention" reports the number of times a task was switched out because it needed to acquire the transaction log semaphore held by another task. This applies to SMP systems only.

If log semaphore contention is high, see "Transaction management" on page 250.

Check disk queuing on the disk used by the transaction log.

See "Disk I/O management" on page 298.

Also see "Engine busy utilization" on page 214. If engine utilization reports a low value, and response time is within acceptable limits, consider reducing the number of engines. Running with fewer engines reduces contention by decreasing the number of tasks trying to access the log simultaneously.

PLC lock contention

"PLC Lock Contention" reports contention for a lock on a user log cache.

Group commit sleeps

"Group Commit Sleeps" reports the number of times a task performed a transaction commit and was put to sleep until the log was written to disk.

Compare this value to the number of committed transactions, reported in "Transaction profile" on page 243. If the transaction rate is low, a higher percentage of tasks wait for "Group Commit Sleeps."

If there are a significant number of tasks resulting in "Group Commit Sleeps," and the log I/O size is greater than 2K, a smaller log I/O size can help to reduce commit time by causing more frequent page flushes. Flushing the page wakes up tasks sleeping on the group commit.

In high throughput environments, a large log I/O size helps prevent problems in disk queuing on the log device. A high percentage of group commit sleeps should not be regarded as a problem.

Other factors that can affect group commit sleeps are the number of tasks on the run queue and the speed of the disk device on which the log resides.

When a task commits, its log records are flushed from its user log cache to the current page of the transaction log in cache. If the log page (or pages, if a large log I/O size is configured) is not full, the task is switched out and placed on the end of the run queue. The log write for the page is performed when:

- Another process fills the log page(s), and flushes the log
- When the task reaches the head of the run queue, and no other process has flushed the log page

For more information, see "Choosing the I/O size for the transaction log" on page 234 in *Performance and Tuning: Basics.*

Last log page writes

"Last Log Page Writes" reports the number of times a task was switched out because it was put to sleep while writing the last log page.

The task switched out because it was responsible for writing the last log page, as opposed to sleeping while waiting for some other task to write the log page, as described in "Group commit sleeps" on page 231.

If this value is high, review "Avg # writes per log page" on page 256 to determine whether Adaptive Server is repeatedly writing the same last page to the log. If the log I/O size is greater than 2K, reducing the log I/O size might reduce the number of unneeded log writes.

Modify conflicts

"Modify Conflicts" reports the number of times that a task tried to get exclusive access to a page that was held by another task under a special lightweight protection mechanism. For certain operations, Adaptive Server uses a lightweight protection mechanism to gain exclusive access to a page without using actual page locks. Examples are access to some system tables and dirty reads. These processes need exclusive access to the page, even though they do not modify it.

I/O device contention

"I/O Device Contention" reports the number of times a task was put to sleep while waiting for a semaphore for a particular device.

When a task needs to perform physical I/O, Adaptive Server fills out the I/O structure and links it to a per-engine I/O queue. If two Adaptive Server engines request an I/O structure from the same device at the same time, one of them sleeps while it waits for the semaphore.

If there is significant contention for I/O device semaphores, try reducing it by redistributing the tables across devices or by adding devices and moving tables and indexes to them.

See "Spreading data across disks to avoid I/O contention" on page 93 in *Performance and Tuning: Basics* for more information.

Network packet received

When task switching is reported by "Network Packet Received," the task switch is due to one of these causes:

- A task received part of a multi packet batch and was switched out waiting for the client to send the next packet of the batch, or
- A task completely finished processing a command and was put into a receive sleep state while waiting to receive the next command or packet from the client.

If "Network Packet Received" is high, see "Network I/O management" on page 303 for more information about network I/O. Also, you can configure the network packet size for all connections or allow certain connections to log in using larger packet sizes.

See "Changing network packet sizes" on page 27 in *Performance and Tuning: Basics* and the *System Administration Guide*.

Network packet sent

"Network Packet Sent" reports the number of times a task went into a send sleep state while waiting for the network to send each packet to the client. The network model determines that there can be only one outstanding packet per connection at any one point in time. This means that the task sleeps after each packet it sends.

If there is a lot of data to send, and the task is sending many small packets (512 bytes per packet), the task could end up sleeping a number of times. The data packet size is configurable, and different clients can request different packet sizes.

For more information, see "Changing network packet sizes" on page 27 in *Performance and Tuning: Basics* and the *System Administration Guide*.

If "Network Packet Sent" is a major cause of task switching, see "Network I/O management" on page 303 for more information.

Other causes

"Other Causes" reports the number of tasks switched out for any reasons not described above. In a well-tuned server, this value may rise as tunable sources of task switching are reduced.

Application management

"Application Management" reports execution statistics for user tasks. This section is useful if you use resource limits, or if you plan to tune applications by setting execution attributes and assigning engine affinity. Before making any adjustments to applications, logins, or stored procedures, run sp_sysmon during periods of typical load, and familiarize yourself with the statistics in this section.

For related background information, see Chapter 5, "Distributing Engine Resources," in *Performance and Tuning:Basics.*

Sample output

```
===============================================================================
```

```
Application Management
----------------------

  Application Statistics Summary (All Applications)
-------------------------------------------------
  Priority Changes               per sec      per xact       count  % of total
 -----------------------  ------------  ------------  ----------  ----------
   To High Priority                0.0           0.0           0       0.0 %
   To Medium Priority              0.0           1.3           5      50.0 %
   To Low Priority                 0.0           1.3           5      50.0 %
-------------------------  ------------  ------------  ----------
   Total Priority Changes           0.1           2.5          10

  Allotted Slices Exhausted      per sec      per xact       count  % of total
-------------------------  ------------  ------------  ----------  ----------
   Total Slices Exhausted           0.0           0.0           0       n/a

  Skipped Tasks By Engine        per sec      per xact       count  % of total
-------------------------  ------------  ------------  ----------  ----------
  Total Engine Skips               0.0           0.0           0       n/a

  Engine Scope Changes             0.0           0.0           0       n/a
```

Requesting detailed application information

- If you request information about specific tasks using the third sp_sysmon parameter, sp_sysmon output gives statistics specific to each application individually in addition to summary information. You can choose to display detailed application information in one of two ways:
- Application and login information (using the sp_sysmon parameter appl_and_login) – sp_sysmon prints a separate section for each login and the applications it is executing.
- Application information only (using the sp_sysmon parameter, appl_only) – sp_sysmon prints a section for each application, which combines data for all of the logins that are executing it.

For example, if 10 users are logged in with isql, and 5 users are logged in with an application called sales_reports, requesting “application and login” information prints 15 detail sections. Requesting “application only” information prints 2 detail sections, one summarizing the activity of all isql users, and the other summarizing the activity of the sales_reports users.

The appl_and_login can be used with all sections of the sp_sysmon. An example of the syntax:

```
sp_sysmon "00:05:00", @applmon=appl_and_login
```

See "Specifying the application detail parameter" on page 202 for information on specifying the parameters for sp_sysmon.

Sample output

The following sample shows sp_sysmon output for the "Application Management" categories in the summary section.

```
Application Management
----------------------

  Application Statistics Summary (All Applications)
  -------------------------------------------------
  Priority Changes         per sec      per xact       count  % of total
  ------------------    ----------   -----------  ----------  ----------
    To High Priority          15.7           1.8        5664      49.9 %
    To Medium Priority        15.8           1.8        5697      50.1 %
    To Low Priority            0.0           0.0           0       0.0 %
  --------------------  ----------    ----------   ----------
  Total Priority Changes      31.6           3.5       11361

  Allotted Slices Exhausted per sec   per xact       count    % of total
  ------------------------- -------   ----------  ----------  ----------
    High Priority               0.0         0.0           0       0.0 %
    Medium Priority             7.0         0.8        2522     100.0 %
    Low Priority                0.0         0.0           0       0.0 %
  ----------------------  ---------   ----------  ----------
  Total Slices Exhausted        7.0         0.8        2522

  Skipped Tasks By Engine  per sec    per xact     count      % of total
  -----------------------  ----------  ----------   ---------
  Total Engine Skips             0.0         0.0           0         n/a

  Engine Scope Changes           0.0         0.0           0         n/a
```

The following example shows output for application and login; only the information for one application and login is included. The first line identifies the application name (before the arrow) and the login name (after the arrow).

```
-------------------------------------------------------------------------
  Application->Login:   ctisql->adonis
```

```
  Application Activity    per sec       per xact   count     % of total
  ----------------------  ----------    ---------  --------  ----------
    CPU Busy                     0.1          0.0        27       2.8 %
    I/O Busy                     1.3          0.1       461      47.3 %
    Idle                         1.4          0.2       486      49.9 %

Number of Times Scheduled        1.7          0.2       597         n/a

 Application Priority Changes  per sec    per xact   count   % of total
  -------------------------  ----------  ---------  -------  ----------
    To High Priority               0.2         0.0       72      50.0 %
    To Medium Priority             0.2         0.0       72      50.0 %
    To Low Priority                0.0         0.0        0       0.0 %
  ------------------------  -----------  ---------  -------
  Total Priority Changes           0.4         0.0      144

  Application I/Os Completed   per sec    per xact   count   % of total
  -------------------------  ---------  ----------  --------  ----------
    Disk I/Os Completed            0.6         0.1       220      53.9 %
    Network I/Os Completed         0.5         0.1       188      46.1 %
  -------------------------  ------------  -------  --------
  Total I/Os Completed             1.1         0.1       408

 Resource Limits Violated    per sec    per xact    count  % of total
 ----------------------     --------  ----------   ------  -----------
 IO Limit Violations
     Estimated                   0.0         0.0        0        0.0 %
     Actual                      0.1         4.0        4       50.0 %
 Time Limit Violations
     Batch                       0.0         0.0        0        0.0 %
     Xact                        0.0         0.0        0        0.0 %
 RowCount Limit Violations       0.1         4.0        4       50.0 %

---------------------------  --------  ---------  -------
  Total Limits Violated          0.1         8.0        8
```

Application statistics summary (all applications)

The sp_sysmon statistics in the summary section can help you determine whether there are any anomalies in resource utilization. If there are, you can investigate further using the detailed report.

This section gives information about:

- Whether tasks are switching back and forth between different priority levels
- Whether the assigned time that tasks are allowed to run is appropriate
- Whether tasks to which you have assigned low priority are getting starved for CPU time
- Whether engine bindings with respect to load balancing is correct

Note that "Application Statistics Summary" includes data for system tasks as well as for user tasks. If the summary report indicates a resource issue, but you do not see supporting evidence in the application or application and login information, investigate the sp_sysmon kernel section of the report ("Kernel utilization" on page 214).

Priority changes

"Priority Changes" reports the priority changes that took place for all user tasks in each priority run queue during the sample interval. It is normal to see some priority switching due to system-related activity. Such priority switching occurs, for example, when:

- A task sleeps while waiting on a lock – Adaptive Server temporarily raises the task's priority.
- A housekeeper task sleeps – Adaptive Server raises the priority to medium when the housekeeper wash and housekeeper chores task wake up, and changes them back to low priority when they go back to sleep.
- A task executes a stored procedure – the task assumes the priority of the stored procedure and resumes its previous priority level after executing the procedure.

If you are using logical process management and there are a high number of priority changes compared to steady state values, it may indicate that an application, or a user task related to that application, is changing priorities frequently. Check priority change data for individual applications. Verify that applications and logins are behaving as you expect.

If you determine that a high-priority change rate is not due to an application or to related tasks, then it is likely due to system activity.

Total priority changes

"Total Priority Changes" reports the total number of priority changes during the sample period. This section gives you a quick way to determine if there are a high number of run queue priority changes occurring.

Allotted slices exhausted

"Allotted Slices Exhausted" reports the number of times user tasks in each run queue exceeded the time allotted for execution. Once a user task gains access to an engine, it is allowed to execute for a given period of time. If the task has not yielded the engine before the time is exhausted, Adaptive Server requires it to yield as soon as possible without holding critical resources. After yielding, the task is placed back on the run queue.

This section helps you to determine whether there are CPU-intensive applications for which you should tune execution attributes or engine associations. If these numbers are high, it indicates that an application is CPU intensive. Application-level information can help you figure out which application to tune. Some tasks, especially those which perform large sort operations, are CPU intensive.

Skipped tasks by engine

"Skipped Tasks By Engine" reports the number of times engines skipped a user task at the head of a run queue. This happens when the task at the head of the run queue has affinity to an engine group and was bypassed in the queue by an engine that is not part of the engine group.

The value is affected by configuring engine groups and engine group bindings. A high number in this category might be acceptable if low priority tasks are bypassed for more critical tasks. It is possible that an engine group is bound so that a task that is ready to run might not be able to find a compatible engine. In this case, a task might wait to execute while an engine sits idle. Investigate engine groups and how they are bound, and check load balancing.

Engine scope changes

"Engine Scope Changes" reports the number of times a user changed the engine group binding of any user task during the sample interval.

Per application or per application and login

This section gives detailed information about system resource used by particular application and login tasks, or all users of each application.

Application activity

Application Activity helps you to determine whether an application is I/0 intensive or CPU intensive. It reports how much time all user task in the application spend executing, doing I/O, or being idle. It also reports the number of times a task is scheduled and chosen to run.

CPU busy

"CPU Busy" reports the number of clock ticks during which the user task was executing during the sample interval. When the numbers in this category are high, it indicates a CPU- bound application. If this is a problem, engine binding might be a solution.

I/O busy

"I/O Busy" reports the number of clock ticks during which the user task was performing I/O during the sample interval. If the numbers in this category are high, it indicates an I/O-intensive process. If idle time is also high, the application could be I/O bound.

The application might achieve better throughput if you assign it a higher priority, bind it to a lightly loaded engine or engine group, or partition the application's data onto multiple devices.

Idle

"Idle" reports the number of clock ticks during which the user task was idle during the sample interval.

Number of times scheduled

"Number of Times Scheduled" reports the number of times a user task is scheduled and chosen to run on an engine. This data can help you determine whether an application has sufficient resources. If this number is low for a task that normally requires substantial CPU time, it may indicate insufficient resources. Consider changing priority in a loaded system with sufficient engine resources.

Application priority changes

Application Priority Changes reports the number of times this application had its priority changed during the sample interval.

When the Application Management category indicates a problem, use this section to pinpoint the source.

Application I/Os completed

"Application I/Os Completed" reports the disk and network I/Os completed by this application during the sample interval.

This category indicates the total number of disk and network I/Os completed.

If you suspect a problem with I/O completion, see "Disk I/O management" on page 298 and "Network I/O management" on page 303.

Resource limits violated

"Resource Limits Violated" reports the number and types of violations for:

- I/O Limit Violations–Estimated and Actual
- Time Limits–Batch and Transaction
- RowCount Limit Violations
- "Total Limits Violated"

If no limits are exceeded during the sample period, only the total line is printed.

See the *System Administration Guide* for more information on resource limits.

ESP management

This section reports on the use of extended stored procedures.

Sample output

```
===============================================================================

ESP Management                    per sec      per xact       count  % of total
```

```
-------------------------- ------------ ------------ ---------- ----------
ESP Requests                        0.0          0.0          0        n/a
```

ESP requests

"ESP Requests" reports the number of extended stored procedure calls during the sample interval.

Avg. time to execute an ESP

"Avg. Time to Execute an ESP" reports the average length of time for all extended stored procedures executed during the sample interval.

Housekeeper task activity

The "Housekeeper Tasks Activity" section reports on housekeeper tasks. If the configuration parameter housekeeper free write percent is set to 0, the housekeeper task does not run. If housekeeper free write percent is 1 or greater, space reclamation can be enabled separately by setting enable housekeeper GC to 1, or disabled by setting it to 0.

Sample output

```
===============================================================================

Housekeeper Task Activity
-------------------------
                            per sec      per xact       count  % of total
                            -----------  -----------  ----------
Buffer Cache Washes
    Clean                        0.3          7.8         31      93.9 %
    Dirty                        0.0          0.5          2       6.1 %
                            -----------  -----------  ----------
Total Washes                     0.3          8.3         33

Garbage Collections              0.2          6.0         24       n/a
Pages Processed in GC            0.0          0.0          0       n/a

Statistics Updates               0.0          0.5          2       n/a
```

Buffer cache washes

This section reports:

- The number of buffers examined by the housekeeper wash task
- The number that were found clean
- The number that were found dirty

The number of dirty buffers includes those already in I/O due to writes being started at the wash marker.

The "Recovery Management" section of sp_sysmon reports how many times the housekeeper wash task was able to write all dirty buffers for a database.

See"Recovery management" on page 295.

Garbage collections

This section reports the number of times the housekeeper garbage collection task checked to determine whether there were committed deletes that indicated that there was space that could be reclaimed on data pages.

"Pages Processed in GC" reports the number of pages where the housekeeper garbage collection task succeeded in reclaiming unused space on the a page of a data-only-locked table.

Statistics updates

"Statistics Updates" reports on the number of times the housekeeper chores task checked to see if statistics needed to be written.

Monitor access to executing SQL

This section reports:

- Contention that occurs when sp_showplan or Adaptive Server Monitor accesses query plans
- The number of overflows in SQL batch text buffers and the maximum size of SQL batch text sent during the sample interval

Sample output

```
===============================================================================

Monitor Access to Executing SQL
-------------------------------
                                per sec      per xact       count  % of total
                              ------------ ------------ ---------- ----------
 Waits on Execution Plans            0.0          0.0          0       n/a
 Number of SQL Text Overflows        0.0          0.0          0       n/a
 Maximum SQL Text Requested          n/a          n/a          0       n/a
   (since beginning of sample)
```

Waits on execution plans

"Waits on Execution Plans" reports the number of times that a process attempting to use sp_showplan had to wait to acquire read access to the query plan. Query plans may be unavailable if sp_showplan is run before the compiled plan is completed or after the query plan finished executing. In these cases, Adaptive Server tries to access the plan three times and then returns a message to the user.

Number of SQL text overflows

"Number of SQL Text Overflows" reports the number of times that SQL batch text exceeded the text buffer size.

Maximum SQL text requested

"Maximum SQL Text Requested" reports the maximum size of a batch of SQL text since the sample interval began. You can use this value to set the configuration parameter max SQL text monitored.

See the *System Administration Guide*.

Transaction profile

The "Transaction Profile" section reports on data modifications by type of command and table locking scheme.

Sample output

The following sample shows sp_sysmon output for the "Transaction Profile" section.

```
===============================================================================
Transaction Profile
-------------------

  Transaction Summary             per sec      per xact       count  % of total
-------------------------  ------------  ------------  ----------  ----------
   Committed Xacts                  0.0           n/a           4     n/a

  Transaction Detail              per sec      per xact       count  % of total
-------------------------  ------------  ------------  ----------  ----------
   Total Rows Affected              0.0           0.0           0       n/a
```

Transaction summary

"Transaction Summary" reports committed transactions. "Committed Xacts" reports the number of transactions committed during the sample interval.

The count of transactions includes transactions that meet explicit, implicit, and ANSI definitions for "committed", as described here:

- An implicit transaction executes data modification commands such as insert, update, or delete. If you do not specify a begin transaction statement, Adaptive Server interprets every operation as a separate transaction; an explicit commit transaction statement is not required. For example, the following is counted as three transactions.

  ```
  1> insert …
  2> go
  1> insert …
  2> go
  1> insert …
  2> go
  ```

- An explicit transaction encloses data modification commands within begin transaction and commit transaction statements and counts the number of transactions by the number of commit statements. For example the following set of statements is counted as one transaction:

  ```
  1> begin transaction
  2> insert …
  3> insert …
  ```

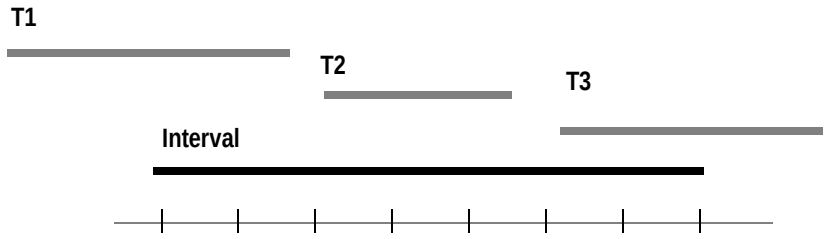
```
4> insert …
5> commit transaction
6> go
```

- In the ANSI transaction model, any select or data modification command starts a transaction, but a commit transaction statement must complete the transaction. sp_sysmon counts the number of transactions by the number of commit transaction statements. For example, the following set of statements is counted as one transaction:

```
1> insert …
2> insert …
3> insert …
4> commit transaction
5> go
```

If there were transactions that started before the sample interval began and completed during the interval, the value reports a larger number of transactions than the number that started and completed during the sample interval. If transactions do not complete during the interval, "Total # of Xacts" does not include them. In Figure 8-1, both T1 and T2 are counted, but T3 is not.

Figure 8-1: How transactions are counted



How to count multi database transactions

Multi database transactions are also counted. For example, a transaction that modifies three databases is counted as three transactions.

Multi database transactions incur more overhead than single database transactions: they require more log records and more ULC flushes, and they involve two-phase commit between the databases.

You can improve performance by reducing the number of multi database transactions whenever possible.

Transaction detail

"Transaction Detail" gives statistical detail about data modification operations by type. The work performed by rolled back transactions is included in the output below, although the transaction is not counted in the number of transactions.

For the "Total Rows" for inserts, updates, and deletes, the "% of total" column reports the percentage of the transaction type as a percentage of all transactions.

See "Update mode messages" on page 79 for more information on deferred and direct inserts, updates, and deletes.

In the output for this section, APL indicates allpages-locked tables and DOL indicates data-only-locked tables.

Inserts

"Inserts" provides detailed information about the types of inserts taking place on heap tables (including partitioned heap tables), clustered tables, and all inserts as a percentage of all insert, update, and delete operations. It displays the number of inserts performed on:

- Allpages-locked heap tables
- Allpages-locked tables with clustered indexes
- Data-only locked tables

Insert statistics do not include fast bulk copy inserts, because those are written directly to the data pages and to disk without the normal insert and logging mechanisms.

APL heap tables

"APL Heap Tables" reports the number of row inserts that took place on allpages-locked heap tables—all tables that do not have a clustered index. This includes:

- Partitioned heap tables
- Unpartitioned heap tables
- Slow bulk copy inserts into heap tables
- select into commands

- Inserts into worktables

The "% of total" column shows the percentage of row inserts into heap tables as a percentage of the total number of inserts.

If there are a large number of inserts to heap tables, determine if these inserts are generating contention.

Check the sp_sysmon report for data on last page locks on heaps in "Lock detail" on page 272. If there appears to be a contention problem, Adaptive Server Monitor can help you figure out which tables are involved.

In many cases, creating a clustered index that randomizes insert activity solves the performance problems for heaps. In other cases, you might need to establish partitions on an unpartitioned table or increase the number of partitions on a partitioned table.

For more information, see Chapter 12, "How Indexes Work" and "Improving insert performance with partitions" on page 101 in *Performance and Tuning: Basics*.

APL clustered table

"APL Clustered Table" reports the number of row inserts to allpages-locked tables with clustered indexes. The "% of total" column shows the percentage of row inserts to tables with clustered indexes as a percentage of the total number of rows inserted.

Inserts into allpages-locked clustered tables can lead to page splitting.

See Row ID updates from clustered split and "Page splits" on page 259.

Data only lock table

"Data Only Lock Table" reports the number of inserts for all data-only-locked tables. The "% of total" column shows the percentage of inserts to data-only-locked tables as a percentage of all inserts.

Total rows inserted

"Total Rows Inserted" reports all row inserts to all tables combined. It gives the average number of all inserts per second, the average number of all inserts per transaction, and the total number of inserts. "% of total" shows the percentage of rows inserted compared to the total number of rows affected by data modification operations.

Updates and update detail sections

The “Updates” report has two sections, “Updates” and “Data Only Locked Updates.”

Updates

“Updates” reports the number of deferred and direct row updates. The “% of total” column reports the percentage of each type of update as a percentage of the total number of row updates. sp_sysmon reports the following types of updates:

- APL Deferred
- APL Direct In-place
- APL Direct Cheap
- APL Direct Expensive
- DOL Deferred
- DOL Direct

Direct updates incur less overhead than deferred updates and are generally faster because they limit the number of log scans, reduce locking, save traversal of index B-trees (reducing lock contention), and can save I/O because Adaptive Server does not have to refetch pages to perform modification based on log records.

For a description of update types, see “How update operations are performed” on page 94 in *Performance and Tuning: Optimizer*.

If there is a high percentage of deferred updates, see “Optimizing updates” on page 102 of the same book.

Total rows updated

“Total Rows Updated” reports all deferred and direct updates combined. The “% of total” columns shows the percentage of rows updated, based on all rows modified.

Data-only-locked updates

This section reports more detail on updates to data-only-locked tables:

- DOL Replace – The update did not change the length of the row; some or all of the row was changed resulting in the same row length

- DOL Shrink – The update shortened the row, leaving non contiguous empty space on the page to be collected during space reclamation.
- DOL Cheap Expand – The row grew in length; it was the last row on the page, so expanding the length of the row did not require moving other rows on the page.
- DOL Expensive Expand – The row grew in length and required movement of other rows on the page.
- DOL Expand and Forward – The row grew in length, and did not fit on the page. The row was forwarded to a new location.
- DOL Fwd Row Returned – The update affected a forwarded row; the row fit on the page at its original location and was returned to that page.

The total reported in "Total DOL Rows Updated" are not included in the "Total Rows Affected" sum at the end of the section, since the updates in this group are providing a different breakdown of the updates already reported in "DOL Deferred" and "DOL Direct."

Deletes

"Deletes" reports the number of deferred and direct row deletes from allpages-locked tables. All deletes on data-only-locked tables are performed by marking the row as deleted on the page, so the categories "direct" and "deferred" do not apply. The "% of total" column reports the percentage of each type of delete as a percentage of the total number of deletes.

Total rows deleted

"Total Rows Deleted" reports all deferred and direct deletes combined. The "% of total" columns reports the percentage of deleted rows as a compared to all rows inserted, updated, or deleted.

Transaction management

"Transaction Management" reports transaction management activities, including user log cache (ULC) flushes to transaction logs, ULC log records, ULC semaphore requests, log semaphore requests, transaction log writes, and transaction log allocations.

Sample output

The following sample shows sp_sysmon output for the "Transaction Management" categories.

```
===============================================================================

Transaction Management
----------------------

ULC Flushes to Xact Log         per sec      per xact       count  % of total
------------------------  ------------  ------------  ----------  ----------
  by Full ULC                       0.0           0.0           0       0.0 %
  by End Transaction                0.0           0.0           0       0.0 %
  by Change of Database             0.0           0.0           0       0.0 %
  by Single Log Record              0.0           0.0           0       0.0 %
  by Unpin                          0.0           0.0           0       0.0 %
  by Other                          0.0           0.0           0       0.0 %
------------------------  ------------  ------------  ----------
 Total ULC Flushes                  0.0           1.0           4

ULC Log Records                   0.0           0.0           0       n/a
Max ULC Size During Sample        n/a           n/a           0       n/a

ULC Semaphore Requests
  Granted                         0.1           4.0          16     100.0 %
  Waited                          0.0           0.0           0       0.0 %
------------------------  ------------  ------------  ----------
 Total ULC Semaphore Req          0.1           4.0          16

Log Semaphore Requests
   Granted                        0.0           1.0           4     100.0 %
   Waited                         0.0           0.0           0       0.0 %
------------------------  ------------  ------------  ----------
 Total Log Semaphore Req          0.0           1.0           4

 Transaction Log Writes           0.0           0.0           0       n/a
```

```
Transaction Log Alloc              0.0          0.0          0       n/a
```

ULC flushes to transaction log

"ULC Flushes to Xact Log" reports the total number of times that user log caches (ULCs) were flushed to a transaction log. The "% of total" column reports the percentage of times the type of flush took place, for each category, as a percentage of the total number of ULC flushes. This category can help you identify areas in the application that cause problems with ULC flushes.

There is one user log cache (ULC) for each configured user connection. Adaptive Server uses ULCs to buffer transaction log records. On both SMP and single-processor systems, this helps reduce transaction log I/O. For SMP systems, it reduces the contention on the current page of the transaction log.

You can configure the size of ULCs with the configuration parameter user log cache size.

See the *System Administration Guide*.

ULC flushes are caused by the following activities:

- "by Full ULC" – A process's ULC becomes full.
- "by End Transaction" – A transaction ended (rollback or commit, either implicit or explicit).
- "by Change of Database" – A transaction modified an object in a different database (a multi database transaction).
- "by System Log Record" – A system transaction (such as an OAM page allocation) occurred within the user transaction.
- "by Other" – Any other reason, including needing to write to disk.

When one of these activities causes a ULC flush, Adaptive Server copies all log records from the user log cache to the database transaction log.

"Total ULC Flushes" reports the total number of all ULC flushes that took place during the sample interval.

Note In databases with mixed data and log segments, the user log cache is flushed after each record is added.

By full ULC

A high value for "by Full ULC" indicates that Adaptive Server is flushing the ULCs more than once per transaction, negating some performance benefits of user log caches. If the "% of total" value for "by Full ULC" is greater than 20%, consider increasing the size of the user log cache size parameter.

Increasing the ULC size increases the amount of memory required for each user connection, so you do not want to configure the ULC size to suit a small percentage of large transactions.

By end transaction

A high value for "by End Transaction" indicates a healthy number of short, simple transactions.

By change of database

The ULC is flushed every time there is a database change. If this value is high, consider decreasing the size of the ULC if it is greater than 2K.

By system log record and by other

If either of these values is higher than approximately 20%, and size of your ULC is more than 2048, consider reducing the ULC size.

Check sections of your sp_sysmon report that relate to log activity:

- Contention for semaphore on the user log caches (SMP only); see "ULC semaphore requests" on page 253
- Contention for the log semaphore. (SMP only); see "Log semaphore requests" on page 254
- The number of transaction log writes; see "Transaction log writes" on page 255

Total ULC flushes

"Total ULC Flushes" reports the total number of ULC flushes during the sample interval.

ULC log records

This row provides an average number of log records per transaction. It is useful in benchmarking or in controlled development environments to determine the number of log records written to ULCs per transaction.

Many transactions, such as those that affect several indexes or deferred updates or deletes, require several log records for a single data modification. Queries that modify a large number of rows use one or more records for each row.

If this data is unusual, study the data in the next section, Maximum ULC size and look at your application for long-running transactions and for transactions that modify large numbers of rows.

Maximum ULC size

The value in the "count" column is the maximum number of bytes used in any ULCs, across all ULCs. This data can help you determine if ULC size is correctly configured.

Since Adaptive Server flushes the ULC when a transaction completes, any unused memory allocated to the ULCs is wasted. If the value in the "count" column is consistently less than the defined value for the user log cache size configuration parameter, reduce user log cache size to the value in the "count" column (but no smaller than 2048 bytes).

When "Max ULC Size" equals the user log cache size, check the number of flushes due to transactions that fill the user log cache (see "By full ULC" on page 252). If the number of times that logs were flushed due to a full ULC is more than 20%, consider increasing the user log cache size configuration parameter.

See the *System Administration Guide*.

ULC semaphore requests

"ULC Semaphore Requests" reports the number of times a user task was immediately granted a semaphore or had to wait for it. "% of total" shows the percentage of tasks granted semaphores and the percentage of tasks that waited for semaphores as a percentage of the total number of ULC semaphore requests. This is relevant only in SMP environments.

A semaphore is a simple internal locking mechanism that prevents a second task from accessing the data structure currently in use. Adaptive Server uses semaphores to protect the user log caches since more than one process can access the records of a ULC and force a flush.

This category provides the following information:

- Granted – The number of times a task was granted a ULC semaphore immediately upon request. There was no contention for the ULC.
- Waited – The number of times a task tried to write to ULCs and encountered semaphore contention.
- Total ULC Semaphore Requests – The total number of ULC semaphore requests that took place during the interval. This includes requests that were granted or had to wait.

Log semaphore requests

"Log Semaphore Requests" reports of contention for the log semaphore that protects the current page of the transaction log in cache. This data is meaningful for SMP environments only.

This category provides the following information:

- Granted – The number of times a task was granted a log semaphore immediately after it requested one. "% of total" reports the percentage of immediately granted requests as a percentage of the total number of log semaphore requests.
- Waited – The number of times two tasks tried to flush ULC pages to the log simultaneously and one task had to wait for the log semaphore. "% of total" reports the percentage of tasks that had to wait for a log semaphore as a percentage of the total number of log semaphore requests.
- Total Log Semaphore Requests – The total number of times tasks requested a log semaphore including those granted immediately and those for which the task had to wait.

Log semaphore contention and user log caches

In high throughput environments with a large number of concurrent users committing transactions, a certain amount of contention for the log semaphore is expected. In some tests, very high throughput is maintained, even though log semaphore contention is in the range of 20 to 30%.

Preallocating Log Pages to Reduce Contention

In SMP environments with high rates of data modification transactions, use the dbcc tune(log_prealloc) command if log semaphore contention is high. A system task, the log allocator process, performs transaction log allocations, reducing the time that each task holds the log semaphore.

To start the log allocator process for a database, use:

```
dbcc tune(log_prealloc, dbid, "on")
```

The log allocator loops through databases in a way similar to the checkpoint process, except that a System Administrator chooses which databases need log pre allocation. The log allocator runs once per minute.

When the log allocator is enabled for a database, it preallocates 64 log pages at a time.

Some options for reducing log semaphore contention are:

- Increasing the ULC size, if filling user log caches is a frequent cause of user log cache flushes.

  See “ULC flushes to transaction log” on page 251 for more information.

- Reducing log activity through transaction redesign. Aim for more batching with less frequent commits. Be sure to monitor lock contention as part of the transaction redesign.
- Reducing the number of multi database transactions, since each change of database context requires a log write.
- Dividing the database into more than one database so that there are multiple logs. If you choose this solution, divide the database in such a way that multi database transactions are minimized.

Transaction log writes

“Transaction Log Writes” reports the total number of times Adaptive Server wrote a transaction log page to disk. Transaction log pages are written to disk when a transaction commits (after a wait for a group commit sleep) or when the current log page(s) become full.

Transaction log allocations

"Transaction Log Alloc" reports the number of times additional pages were allocated to the transaction log. This data is useful for comparing to other data in this section and for tracking the rate of transaction log growth.

Avg # writes per log page

"Avg # Writes per Log Page" reports the average number of times each log page was written to disk. The value is reported in the "count" column.

In high throughput applications, this number should be as low as possible. If the transaction log uses 2K I/O, the lowest possible value is 1; with 4K log I/O, the lowest possible value is .5, since one log I/O can write 2 log pages.

In low throughput applications, the number will be significantly higher. In very low throughput environments, it may be as high as one write per completed transaction.

Index management

This category reports index management activity, including nonclustered maintenance, page splits, and index shrinks.

Sample output

The following sample shows sp_sysmon output for the "Index Management" categories.

```
===============================================================================

Index Management
----------------

Nonclustered Maintenance        per sec      per xact       count  % of total
-------------------------  ------------  ------------  ----------  ----------
  Ins/Upd Requiring Maint           0.0           0.0           0       n/a
    # of NC Ndx Maint                0.0           0.0           0       n/a

  Deletes Requiring Maint           0.0           0.0           0       n/a
```

```
    # of NC Ndx Maint                0.0            0.0            0       n/a

  RID Upd from Clust Split           0.0            0.0            0       n/a
    # of NC Ndx Maint                0.0            0.0            0       n/a

  Upd/Del DOL Req Maint              0.0            0.0            0       n/a
    # of DOL Ndx Maint                0.0            0.0            0       n/a

Page Splits                          0.0            0.0            0       n/a

Page Shrinks                         0.0            0.0            0       n/a

Index Scans                       per sec      per xact       count  % of total
-------------------------  ------------  ------------  ----------  ----------
  Ascending Scans                    0.0            0.5            2     100.0 %
  DOL Ascending Scans                0.0            0.0            0       0.0 %
  Descending Scans                   0.0            0.0            0       0.0 %
  DOL Descending Scans               0.0            0.0            0       0.0 %
------------  ------------  ----------
```

Nonclustered maintenance

This category reports the number of operations that required, or potentially required, maintenance to one or more indexes; that is, it reports the number of operations for which Adaptive Server had to at least check to determine whether it was necessary to update the index. The output also gives the number of indexes that were updated and the average number of indexes maintained per operation.

In tables with clustered indexes and one or more nonclustered indexes, all inserts, all deletes, some update operations, and any data page splits, require changes to the nonclustered indexes. High values for index maintenance indicate that you should assess the impact of maintaining indexes on your Adaptive Server performance. While indexes speed retrieval of data, maintaining indexes slows data modification. Maintenance requires additional processing, additional I/O, and additional locking of index pages.

Other sp_sysmon output that is relevant to assessing this category is:

- Information on total updates, inserts and deletes, and information on the number and type of page splits

  See “Transaction detail” on page 246, and “Page splits” on page 259.

- Information on lock contention.

See “Lock detail” on page 272.

- Information on address lock contention.

  See “Address lock contention” on page 229 and “Address locks” on page 273.

For example, you can compare the number of inserts that took place with the number of maintenance operations that resulted. If a relatively high number of maintenance operations, page splits, and retries occurred, consider the usefulness of indexes in your applications.

See Chapter 13, “Indexing for Performance,” in *Performance and Tuning: Basics* for more information.

Inserts and updates requiring maintenance to indexes

The data in this section gives information about how insert and update operations affect indexes on allpages-locked tables. For example, an insert to a clustered table with three nonclustered indexes requires updates to all three indexes, so the average number of operations that resulted in maintenance to nonclustered indexes is three.

However, an update to the same table may require only one maintenance operation—to the index whose key value was changed.

- “Ins/Upd Requiring Maint” reports the number of insert and update operations to a table with indexes that potentially required modifications to one or more indexes.
- “# of NC Ndx Maint” reports the number of nonclustered indexes that required maintenance as a result of insert and update operations.
- “Avg NC Ndx Maint/Op” reports the average number of nonclustered indexes per insert or update operation that required maintenance.

For data-only-locked tables, inserts are reported in “Ins/Upd Requiring Maint” and deletes and inserts are reported in “Upd/Del DOL Req Maint.”

Deletes requiring maintenance

The data in this section gives information about how delete operations affected indexes on allpages-locked tables:

- “Deletes Requiring Maint” reports the number of delete operations that potentially required modification to one or more indexes.

See “Deletes” on page 249.

- “# of NC Ndx Maint” reports the number of nonclustered indexes that required maintenance as a result of delete operations.
- “Avg NC Ndx Maint/Op” reports the average number of nonclustered indexes per delete operation that required maintenance.

Row ID updates from clustered split

This section reports index maintenance activity caused by page splits in allpages-locked tables with clustered indexes. These splits require updating the nonclustered indexes for all of the rows that move to the new data page.

- “RID Upd from Clust Split” reports the total number of page splits that required maintenance of a nonclustered index.
- “# of NC Ndx Maint” reports the number of nonclustered rows that required maintenance as a result of row ID update operations.
- “Avg NC Ndx Maint/Op” reports the average number of nonclustered indexes entries that were updated for each page split.

Data-Only-Locked updates and deletes requiring maintenance

The data in this section gives information about how updates and deletes affected indexes on data-only-locked tables:

- “Upd/Del DOL Req Maint” reports the number of update and delete operations that potentially required modification to one or more indexes.
- “# of DOL Ndx Main” reports the number of indexes that required maintenance as a result of update or delete operations.
- “Avg DOL Ndx Maint/Op” reports the average number of indexes per update or delete operation that required maintenance.

Page splits

“Page Splits” reports the number page splits for data pages, clustered index pages, or nonclustered index pages because there was not enough room for a new row.

When a data row is inserted into an allpages-locked table with a clustered index, the row must be placed in physical order according to the key value. Index rows must also be placed in physical order on the pages. If there is not enough room on a page for a new row, Adaptive Server splits the page, allocates a new page, and moves some rows to the new page. Page splitting incurs overhead because it involves updating the parent index page and the page pointers on the adjoining pages and adds lock contention. For clustered indexes, page splitting also requires updating all nonclustered indexes that point to the rows on the new page.

See "Choosing space management properties for indexes" on page 321 in *Performance and Tuning: Basics* for more information about how to temporarily reduce page splits using fillfactor.

Reducing page splits for ascending key inserts

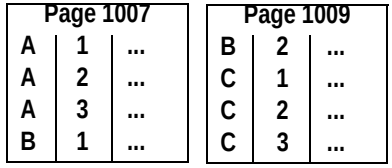
If "Page Splits" is high and your application is inserting values into an allpages-locked table with a clustered index on a compound key, it may be possible to reduce the number of page splits through a special optimization that changes the page split point for these indexes.

The special optimization is designed to reduce page splitting and to result in more completely filled data pages. This affects only clustered indexes with compound keys, where the first key is already in use in the table, and the second column is based on an increasing value.

Default data page splitting

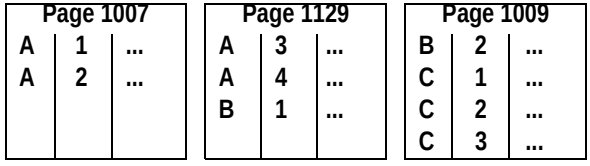
The table sales has a clustered index on store_id, customer_id. There are three stores (A, B, and C). Each store adds customer records in ascending numerical order. The table contains rows for the key values A,1; A,2; A,3; B,1; B,2; C,1; C,2; and C,3, and each page holds four rows, as shown in Figure 8-2.

Figure 8-2: Clustered table before inserts



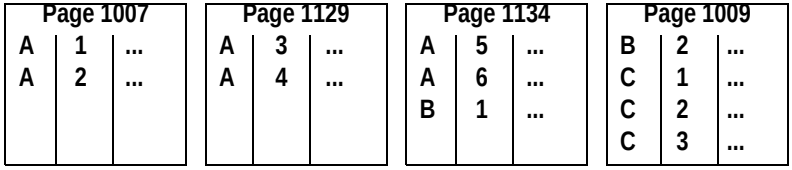
Using the normal page-splitting mechanism, inserting "A,4" results in allocating a new page and moving half of the rows to it, and inserting the new row in place, as shown in Figure 8-3.

Figure 8-3: Insert causes a page split



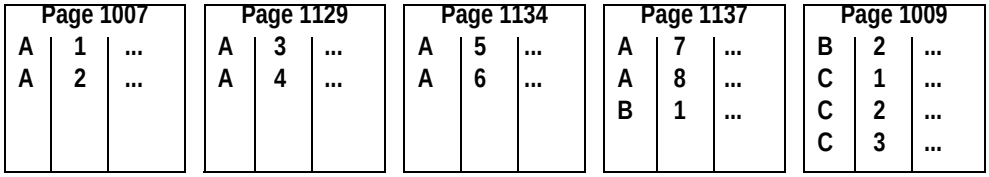
When "A,5" is inserted, no split is needed, but when "A,6" is inserted, another split takes place, as shown in Figure 8-4.

Figure 8-4: Another insert causes another page split



Adding "A,7" and "A,8" results in yet another page split, as shown in Figure 8-5.

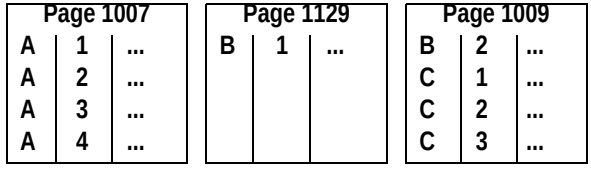
Figure 8-5: Page splitting continues



Effects of ascending inserts

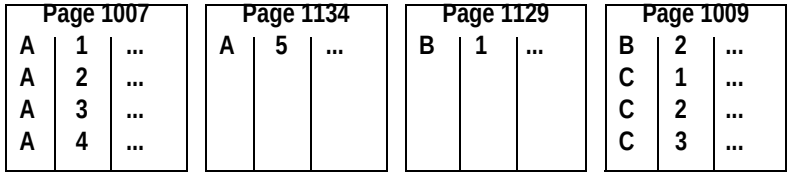
You can set ascending inserts mode for a table, so that pages are split at the point of the inserted row, rather than in the middle of the page. Starting from the original table shown in Figure 8-2 on page 261, the insertion of "A,4" results in a split at the insertion point, with the remaining rows on the page moving to a newly allocated page, as shown in Figure 8-6.

Figure 8-6: First insert with ascending inserts mode



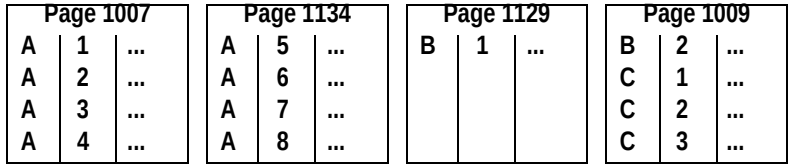
Inserting "A,5" causes a new page to be allocated, as shown in Figure 8-7.

Figure 8-7: Additional ascending insert causes a page allocation



Adding "A,6", "A,7", and "A,8" fills the new page, as shown in Figure 8-8.

Figure 8-8: Additional inserts fill the new page



Setting ascending inserts mode for a table

The following command turns on ascending insert mode for the sales table:

```
dbcc tune (ascinserts, 1, "sales")
```

To turn ascending insert mode off, use:

```
dbcc tune (ascinserts, 0, "sales")
```

These commands update the status2 bit of sysindexes.

If tables sometimes experience random inserts and have more ordered inserts during batch jobs, it is better to enable dbcc tune (ascinserts) only for the period during which the batch job runs.

Retries and deadlocks

"Deadlocks" reports the number of index page splits and shrinks that resulted in deadlocks. Adaptive Server has a mechanism called *deadlock retries* that attempts to avoid transaction rollbacks caused by index page deadlocks. "Retries" reports the number of times Adaptive Server used this mechanism.

Deadlocks on index pages take place when each of two transactions needs to acquire locks held by the other transaction. On data pages, deadlocks result in choosing one process (the one with the least accumulated CPU time) as a deadlock victim and rolling back the process.

By the time an index deadlock takes place, the transaction has already updated the data page and is holding data page locks so rolling back the transaction causes overhead.

In a large percentage of index deadlocks caused by page splits and shrinks, both transactions can succeed by dropping one set of index locks, and restarting the index scan. The index locks for one of the processes are released (locks on the data pages are still held), and Adaptive Server tries the index scan again, traversing the index from the root page of the index.

Usually, by the time the scan reaches the index page that needs to be split, the other transaction has completed, and no deadlock takes place. By default, any index deadlock that is due to a page split or shrink is retried up to five times before the transaction is considered deadlocked and is rolled back.

For information on changing the default value for the number of deadlock retries, see the *System Administration Guide*.

The deadlock retries mechanism causes the locks on data pages to be held slightly longer than usual and causes increased locking and overhead. However, it reduces the number of transactions that are rolled back due to deadlocks. The default setting provides a reasonable compromise between the overhead of holding data page locks longer and the overhead of rolling back transactions that have to be reissued.

A high number of index deadlocks and deadlock retries indicates high contention in a small area of the index B-tree.

If your application encounters a high number of deadlock retries, reduce page splits using fillfactor when you re-create the index.

See “Reducing index maintenance” on page 183 in *Performance and Tuning: Basics*.

Add index level

“Add Index Level” reports the number of times a new index level was added. This does not happen frequently, so you should expect to see result values of 0 most of the time. The count could have a value of 1 or 2 if your sample includes inserts into either an empty table or a small table with indexes.

Page shrinks

“Page Shrinks” reports the number of times that deleting index rows caused the index to shrink off a page. Shrinks incur overhead due to locking in the index and the need to update pointers on adjacent pages. Repeated “count” values greater than 0 indicate there may be many pages in the index with fairly small numbers of rows per page due to delete and update operations. If there are a high number of shrinks, consider rebuilding the indexes.

Index scans

The "Index Scans" section reports forward and backward scans by lock scheme:

- "Ascending Scans" reports the number of forward scans on allpages-locked tables.
- "DOL Ascending Scans" reports the number of forward scans on data-only-locked tables.
- "Descending Scans" reports the number of backward scans on allpages-locked tables.
- "DOL Descending Scans" reports the number of backward scans on data-only-locked tables.

For more information on forward and backward scans, see "Costing for queries using order by" on page 79 in *Performance and Tuning: Optimizer*.

Metadata cache management

"Metadata Cache Management" reports the use of the metadata caches that store information about the three types of metadata caches: objects, indexes, and databases. This section also reports the number of object, index and database descriptors that were active during the sample interval, and the maximum number of descriptors that have been used since the server was last started. It also reports spinlock contention for the object and index metadata caches.

Sample output

```
===============================================================================

Metadata Cache Management
-------------------------

 Metadata Cache Summary          per sec      per xact       count  % of total
------------------------  ------------  ------------  ----------  ----------

 Open Object Usage
    Active                          n/a           n/a         393         n/a
```

```
    Max Ever Used Since Boot          n/a           n/a          393       n/a
    Free                              n/a           n/a          107       n/a
   Reuse Requests
      Succeeded                       n/a           n/a            0       n/a
      Failed                          n/a           n/a            0       n/a

 Open Index Usage
    Active                            n/a           n/a           44       n/a
    Max Ever Used Since Boot          n/a           n/a           46       n/a
    Free                              n/a           n/a          456       n/a
   Reuse Requests
      Succeeded                       n/a           n/a            0       n/a
      Failed                          n/a           n/a            0       n/a

 Open Database Usage
    Active                            n/a           n/a            7       n/a
    Max Ever Used Since Boot          n/a           n/a            7       n/a
    Free                              n/a           n/a            5       n/a
   Reuse Requests
      Succeeded                       n/a           n/a            0       n/a
      Failed                          n/a           n/a            0       n/a

Object Manager Spinlock Contention  n/a          n/a          n/a        0.0 %

Object Spinlock Contention          n/a          n/a          n/a        0.0 %

Index Spinlock Contention           n/a          n/a          n/a        0.0 %

Hash Spinlock Contention             n/a           n/a          n/a        0.0 %
```

Open object, index, and database usage

Each of these sections contains the same information for the three types of metadata caches. The output provides this information:

- “Active” reports the number of objects, indexes, or databases that were active during the sample interval.
- “Max Ever Used Since Boot” reports the maximum number of descriptors used since the last restart of Adaptive Server.
- “Free” reports the number of free descriptors in the cache.
- “Reuse Requests” reports the number of times that the cache had to be searched for reusable descriptors:

- "Failed" means that all descriptors in cache were in use and that the client issuing the request received an error message.
- "Succeeded" means the request found a reusable descriptor in cache. Even though "Succeeded" means that the client did not get an error message, Adaptive Server is doing extra work to locate reusable descriptors in the cache and to read metadata information from disk.

You can use this information to set the configuration parameters number of open indexes, number of open objects, and number of open databases, as shown in Table 8-2.

Table 8-2: Action to take based on metadata cache usage statistics

***sp_sysmon* output**	**Action**
Large number of "Free" descriptors	Set parameter lower
Very few "Free" descriptors	Set parameter higher
"Reuse Requests Succeeded" nonzero	Set parameter higher
"Reuse Requests Failed" nonzero	Set parameter higher

Object Manager Spinlock Contention

This is one server-wide spinlock used to manager internal states of the object descriptor.

If the contention on this spinlock is > 10% use dbcc tune(des_bind) to address this contention. More details on this can be found as part of the documentation on dbcc tune(des_bind) in the *Adaptive Server Enterprise Reference Manual*.

Object and index spinlock contention

These sections report on spinlock contention on the object descriptor and index descriptor caches. You can use this information to tune the configuration parameters open object spinlock ratio and open index spinlock ratio. If the reported contention is more than 3%, decrease the value of the corresponding parameter to lower the number of objects or indexes that are protected by a single spinlock.

Hash spinlock contention

This section reports contention for the spinlock on the index metadata cache hash table. You can use this information to tune the open index hash spinlock ratio configuration parameter. If the reported contention is greater than 3%, decrease the value of the parameter.

Using *sp_monitorconfig* to find metadata cache usage statistics

sp_monitorconfig displays metadata cache usage statistics on certain shared server resources, including:

- The number of databases, objects, and indexes that can be open at any one time
- The number of auxiliary scan descriptors used by referential integrity queries
- The number of free and active descriptors
- The percentage of active descriptors
- The maximum number of descriptors used since the server was last started
- The current size of the procedure cache and the amount actually used.

For example, suppose you have configured the number of open indexes configuration parameter to 500. During a peak period, you can run sp_monitorconfig as follows to get an accurate reading of the actual metadata cache usage for index descriptors. For example:

```
1> sp_monitorconfig "number of open indexes"

Usage information at date and time: Apr 22 2002  2:49PM.
Name             num_free  num_active  pct_act  Max_Used  Reused
--------------   --------  ----------  -------  --------  ------
number of open        217         283    56.60       300  No
```

In this report, the maximum number of open indexes used since the server was last started is 300, even though Adaptive Server is configured for 500. Therefore, you can reset the number of open indexes configuration parameter to 330, to accommodate the 300 maximum used index descriptors, plus space for 10 percent more.

You can also determine the current size of the procedure cache with sp_monitorconfig procedure cache size. This parameter describes the amount of space in the procedure cache is currently configured for and the most it has ever actually used. For example, the procedure cache in the following server is configured for 20,000 pages:

```
1> sp_configure "procedure cache size"

option_name                    config_value run_value
------------------------------ ------------ ---------
procedure cache size                   3271      3271
```

However, when you run sp_montorconfig "procedure cache size", you find that the most the procedure cache has ever used is 14241 pages, which means that you can lower the run value of the procedure cache, saving memory:

```
1> sp_monitorconfig "procedure cache size"

Usage information at date and time: Apr 22 2002  2:49PM.
Name                num_free  num_active  pct_act  Max_Used  Reused
--------------      --------  ----------  -------  --------  ------
procedure cache         5878       14122    70.61     14241  No
```

Lock management

"Lock Management" reports locks, deadlocks, lock promotions, and lock contention.

Sample output

The following sample shows sp_sysmon output for the "Lock Management" categories.

```
===============================================================================

Lock Management
---------------

 Lock Summary                  per sec      per xact       count  % of total
-------------------------  ------------  ------------  ----------  ----------
  Total Lock Requests               0.2           6.5          26       n/a
  Avg Lock Contention               0.0           0.0           0       0.0 %
  Deadlock Percentage               0.0           0.0           0       0.0 %
```

```
 Lock Detail                    per sec      per xact       count  % of total
-------------------------  ------------  ------------  ----------  ----------

 Table Lock Hashtable
   Lookups                          0.0           0.5           2       n/a
   Avg Chain Length                 n/a           n/a     0.00000       n/a
   Spinlock Contention              n/a           n/a         n/a       0.0 %

 Exclusive Table
   Total EX-Table Requests          0.0           0.0           0       n/a

 Shared Table
   Total SH-Table Requests          0.0           0.0           0       n/a

 Exclusive Intent
   Total EX-Intent Requests         0.0           0.0           0       n/a

 Shared Intent
   Granted                          0.0           0.5           2     100.0 %
   Waited                           0.0           0.0           0       0.0 %
-------------------------  ------------  ------------  ----------  ----------
 Total SH-Intent Requests           0.0           0.5           2       7.7 %

 Page & Row Lock HashTable
   Lookups                          0.1           3.5          14       n/a
   Avg Chain Length                 n/a           n/a     0.00000       n/a
   Spinlock Contention              n/a           n/a         n/a       0.0 %

 Exclusive Page
   Total EX-Page Requests           0.0           0.0           0       n/a

 Update Page
   Total UP-Page Requests           0.0           0.0           0       n/a

 Shared Page
   Granted                          0.1           3.5          14     100.0 %
   Waited                           0.0           0.0           0       0.0 %
-------------------------  ------------  ------------  ----------  ----------
 Total SH-Page Requests             0.1           3.5          14      53.8 %


 Exclusive Row
   Total EX-Row Requests            0.0           0.0           0       n/a

 Update Row
```

```
  Total UP-Row Requests              0.0            0.0            0        n/a

Shared Row
  Total SH-Row Requests              0.0            0.0            0        n/a

Next-Key
  Total Next-Key Requests            0.0            0.0            0        n/a

Address Lock Hashtable
  Lookups                            0.1            2.5           10        n/a
  Avg Chain Length                   n/a            n/a      0.00000        n/a
  Spinlock Contention                n/a            n/a          n/a       0.0 %

Exclusive Address
  Granted                            0.0            1.0            4     100.0 %
  Waited                             0.0            0.0            0       0.0 %
-------------------------  ------------  ------------  ----------  ----------
Total EX-Address Requests            0.0            1.0            4      15.4 %

Shared Address
  Granted                            0.1            1.5            6     100.0 %
  Waited                             0.0            0.0            0       0.0 %
-------------------------  ------------  ------------  ----------  ----------
Total SH-Address Requests            0.1            1.5            6      23.1 %

Last Page Locks on Heaps
  Total Last Pg Locks                0.0            0.0            0        n/a

Deadlocks by Lock Type          per sec      per xact       count  % of total
-------------------------  ------------  ------------  ----------  ----------
Total Deadlocks                      0.0            0.0            0        n/a

Deadlock Detection
  Deadlock Searches                  0.0            0.0            0        n/a

Lock Promotions
  Total Lock Promotions              0.0            0.0            0        n/a

Lock Timeouts by Lock Type      per sec      per xact       count  % of total
-------------------------  ------------  ------------  ----------  ----------
Total Timeouts                       0.0            0.0            0        n/a
```

"Lock Promotions" does report detail rows if there were no occurrences of them during the sample interval. In this sample report, "Deadlocks by Lock Type" is one example.

Lock summary

"Lock Summary" provides overview statistics about lock activity that took place during the sample interval.

- "Total Lock Requests" reports the total number of lock requests.
- "Avg Lock Contention" reports the average number of times there was lock contention as a percentage of the total number of lock requests.

  If the lock contention average is high, study the lock detail information below.

  See Chapter 3, "Locking Configuration and Tuning," in *Performance and Tuning: Locking* for more information on tuning locking behavior.
- "Deadlock Percentage" reports the percentage of deadlocks as a percentage of the total number lock requests.

  If this value is high, see "Deadlocks by lock type" on page 274.
- "Avg Hash Chain Length" reports the average number of locks per hash bucket during the sample interval. You can configure the size of the lock hash table with the configuration parameter lock hashtable size. If the average number of locks per hash chain is more than four, consider increasing the size of the hash table.

  See "Configuring the lock hashtable (Lock Manager)" on page 47 in *Performance and Tuning: Locking* for more information.

  Large inserts with bulk copy are an exception to this guideline. Lock hash chain lengths may be longer during large bulk copies.

Lock detail

"Lock Detail" provides information that you can use to determine whether the application is causing a lock contention or deadlock-related problem.

This output reports locks by type, displaying the number of times that each lock type was granted immediately, and the number of times a task had to wait for a particular type of lock. The "% of total" is the percentage of the specific lock type that was granted or had to wait with respect to the total number of lock requests.

"Lock Detail" reports the following types of locks:

- Exclusive Table

- Shared Table
- Exclusive Intent
- Shared Intent
- Exclusive Page
- Update Page
- Shared Page
- Exclusive Row
- Update Row
- Shared Row
- Exclusive Address
- Shared Address
- Last Page Locks on Heaps

Lock contention can have a large impact on Adaptive Server performance. Table locks generate more lock contention than page or row locks because no other tasks can access a table while there is an exclusive table lock on it, and if a task requires an exclusive table lock, it must wait until all shared locks are released. If lock contention is high, run sp_object_stats to help pinpoint the tables involved.

See "Identifying tables where concurrency is a problem" on page 88 in *Performance and Tuning: Locking* for more information.

Address locks

"Exclusive Address" and "Shared Address" report the number of times address locks were granted immediately or the number of times the task had to wait for the lock. Address locks are held on index pages of allpages-locked tables. They can have serious impact, since a lock on an index page blocks access to all data pages pointed to by the index page.

Last page locks on heaps

"Last Page Locks on Heaps" reports locking attempts on the last page of a partitioned or unpartitioned heap table. It only reports on allpages-locked tables.

This information can indicate whether there are tables in the system that would benefit from using data-only-locking or from partitioning or from increasing the number of partitions. Adding a clustered index that distributes inserts randomly across the data pages may also help. If you know that one or more tables is experiencing a problem with contention for the last page, Adaptive Server Monitor can help determine which table is experiencing the problem.

See "Improving insert performance with partitions" on page 101 in *Performance and Tuning: Basics* for information on how partitions can help solve the problem of last-page locking on unpartitioned heap tables.

Table lock hashtable

"Lock Hashtable Lookups" reports the number of times the lock hash table was searched for a lock on a page, row, or table.

You can configure the size of the lock hash table with the configuration parameter lock hashtable size. If the average number of locks per hash chain is more than 4, consider increasing the size of the hash table. See "Configuring the lock hashtable (Lock Manager)" on page 47 in *Performance and Tuning: Locking* for more information.

Deadlocks by lock type

"Deadlocks by Lock Type" reports the number of specific types of deadlocks. "% of total" gives the number of each deadlock type as a percentage of the total number of deadlocks.

Deadlocks may occur when many transactions execute at the same time in the same database. They become more common as the lock contention increases between the transactions.

This category reports data for the following deadlock types:

- Exclusive Table
- Shared Table
- Exclusive Intent
- Shared Intent
- Exclusive Page
- Update Page

- Shared Page
- Exclusive Row
- Update Row
- Shared Row
- Shared Next-Key
- Exclusive Address
- Shared Address
- Others

"Total Deadlocks" summarizes the data for all lock types.

As in the example for this section, if there are no deadlocks, sp_sysmon does not display any detail information, it only prints the "Total Deadlocks" row with zero values.

To pinpoint where deadlocks occur, use one or both of the following methods:

- Use sp_object_stats. See "Identifying tables where concurrency is a problem" on page 88 in *Performance and Tuning: Locking* for more information.
- Enable printing of detailed deadlock information to the log.

  See "Printing deadlock information to the error log" on page 85 *Performance and Tuning: Monitoring and Analyzing for Performance* .

For more information on deadlocks and coping with lock contention, see "Deadlocks and concurrency" on page 81 and "Locking and performance" on page 39 in *Performance and Tuning: Locking*.

Deadlock detection

"Deadlock Detection" reports the number of deadlock searches that found deadlocks and deadlock searches that were skipped during the sample interval

For a discussion of the background issues related to this topic, see "Deadlocks and concurrency" on page 81 *Performance and Tuning: Locking*.

Deadlock searches

"Deadlock Searches" reports the number of times that Adaptive Server initiated a deadlock search during the sample interval. Deadlock checking is time-consuming overhead for applications that experience no deadlocks or very low levels of deadlocking. You can use this data with Average deadlocks per search to determine if Adaptive Server is checking for deadlocks too frequently.

Searches skipped

"Searches Skipped" reports the number of times that a task started to perform deadlock checking, but found deadlock checking in progress and skipped its check. "% of total" reports the percentage of deadlock searches that were skipped as a percentage of the total number of searches.

When a process is blocked by lock contention, it waits for an interval of time set by the configuration parameter deadlock checking period. When this period elapses, it starts deadlock checking. If a search is already in process, the process skips the search.

If you see some number of searches skipped, but some of the searches are finding deadlocks, increase the parameter slightly. If you see a lot of searches skipped, and no deadlocks, or very few, you can increase the parameter by a larger amount.

See the *System Administration Guide* for more information.

Average deadlocks per search

"Avg Deadlocks per Search" reports the average number of deadlocks found per search.

This category measures whether Adaptive Server is checking too frequently. If your applications rarely deadlock, you can adjust the frequency with which tasks search for deadlocks by increasing the value of the deadlock checking period configuration parameter.

See the *System Administration Guide* for more information.

Lock promotions

"Lock Promotions" reports the number of times that the following escalations took place:

- "Ex-Page to Ex-Table" – Exclusive page to exclusive table.
- "Sh-Page to Sh-Table" – Shared page to shared table.
- "Ex-Row to Ex-Table" – Exclusive row to exclusive table.
- "Sh-Row to Sh-Table – Shared row to shared table.
- "Sh-Next-Key to Sh-Table" – Shared next-key to shared table.

The "Total Lock Promotions" row reports the average number of lock promotion types combined per second and per transaction.

If no lock promotions took place during the sample interval, only the total row is printed.

If there are no lock promotions, sp_sysmon does not display the detail information, as the example for this section shows.

"Lock Promotions" data can:

- Help you detect if lock promotion in your application to is a cause of lock contention and deadlocks
- Be used before and after tuning lock promotion variables to determine the effectiveness of the values.

Look at the "Granted" and "Waited" data above for signs of contention. If lock contention is high and lock promotion is frequent, consider changing the lock promotion thresholds for the tables involved.

You can configure the lock promotion threshold either server-wide or for individual tables.

See information on locking in the *System Administration Guide*.

Lock time-out information

The "Lock Time-outs by Lock Type" section reports on the number of times a task was waiting for a lock and the transaction was rolled back due to a session-level or server-level lock time-out. The detail rows that show the lock types are printed only if lock time-outs occurred during the sample period. If no lock time-outs occurred, the "Total Lock Time-outs" row is displayed with all values equal to 0.

For more information on lock time-outs, see "Lock timeouts" on page 75 *Performance and Tuning: Locking*.

Data cache management

sp_sysmon reports summary statistics for all caches followed by statistics for each named cache.

sp_sysmon reports the following activities for the default data cache and for each named cache:

- Spinlock contention
- Utilization
- Cache searches including hits and misses
- Pool turnover for all configured pools
- Buffer wash behavior, including buffers passed clean, buffers already in I/O, and buffers washed dirty
- Prefetch requests performed and denied
- Dirty read page requests

You can use sp_cacheconfig and sp_helpcache output to help analyze the data from this section of the report. sp_cacheconfig provides information about caches and pools, and sp_helpcache provides information about objects bound to caches.

See the *System Administration Guide* for information on how to use these system procedures.

See "Configuring the data cache to improve performance" on page 220 in *Performance and Tuning: Basics* for more information on performance issues and named caches.

Sample output

The following sample shows sp_sysmon output for the "Data Cache Management" categories. The first block of data, "Cache Statistics Summary," includes information for all caches. sp_sysmon reports a separate block of data for each cache. These blocks are identified by the cache name. The sample output shown here includes only the default data cache, although there were more caches configured during the interval.

```
===============================================================================

Data Cache Management
---------------------
```

```
   Cache Statistics Summary (All Caches)
-------------------------------------
                              per sec      per xact       count  % of total
                         -----------  -----------  ----------  ----------

   Cache Search Summary
    Total Cache Hits             0.1          3.8          15     100.0 %
    Total Cache Misses           0.0          0.0           0       0.0 %
------------------------  -----------  -----------  ----------
  Total Cache Searches           0.1          3.8          15

   Cache Turnover
    Buffers Grabbed              0.0          0.0           0       n/a

   Cache Strategy Summary
    Cached (LRU) Buffers         0.1          3.8          15     100.0 %
    Discarded (MRU) Buffers      0.0          0.0           0       0.0 %

   Large I/O Usage
                                 0.0          0.0           0       n/a

   Large I/O Effectiveness
    Pages by Lrg I/O Cached      0.0          0.0           0       n/a

   Asynchronous Prefetch Activity
                                  0.0           0.0           0       n/a

   Other Asynchronous Prefetch Statistics
    APFs Used                    0.0          0.0           0       n/a
    APF Waits for I/O            0.0          0.0           0       n/a
    APF Discards                 0.0          0.0           0       n/a

   Dirty Read Behavior
    Page Requests                0.0          0.0           0       n/a

-------------------------------------------------------------------------------
   Cache: default data cache
                              per sec      per xact       count  % of total
------------------------  -----------  -----------  ----------  ----------
  Spinlock Contention            n/a          n/a         n/a       0.0 %

  Utilization                    n/a          n/a         n/a     100.0 %

 Cache Searches
    Cache Hits                   0.1          3.8          15     100.0 %
```

```
      Found in Wash                0.1           3.3          13      86.7 %
    Cache Misses                   0.0           0.0           0       0.0 %
------------------------  ------------  ------------  ----------
  Total Cache Searches             0.1           3.8          15

  Pool Turnover                    0.0           0.0           0       n/a

  Buffer Wash Behavior
    Statistics Not Available - No Buffers Entered Wash Section Yet

  Cache Strategy
    Cached (LRU) Buffers           0.1           3.8          15     100.0 %
    Discarded (MRU) Buffers        0.0           0.0           0       0.0 %

  Large I/O Usage
    Total Large I/O Requests       0.0           0.0           0       n/a

  Large I/O Detail
    No Large Pool(s) In This Cache

  Dirty Read Behavior
          Page Requests              0.0           0.0           0       n/a
```

Cache statistics summary (all caches)

This section summarizes behavior for the default data cache and all named data caches combined. Corresponding information is printed for each data cache.

See “Cache management by cache” on page 285.

Cache search summary

This section provides summary information about cache hits and misses. Use this data to get an overview of how effective cache design is. A high number of cache misses indicates that you should investigate statistics for each cache.

- “Total Cache Hits” reports the number of times that a needed page was found in any cache. “% of total” reports the percentage of cache hits as a percentage of the total number of cache searches.
- “Total Cache Misses” reports the number of times that a needed page was not found in a cache and had to be read from disk. “% of total” reports the percentage of times that the buffer was not found in the cache as a percentage of all cache searches.

- "Total Cache Searches" reports the total number of cache searches, including hits and misses for all caches combined.

Cache turnover

This section provides a summary of cache turnover:

- "Buffers Grabbed" reports the number of buffers that were replaced in all of the caches. The "count" column represents the number of times that Adaptive Server fetched a buffer from the LRU end of the cache, replacing a database page. If the server was recently restarted, so that the buffers are empty, reading a page into an empty buffer is not counted here.
- "Buffers Grabbed Dirty" reports the number of times that fetching a buffer found a dirty page at the LRU end of the cache and had to wait while the buffer was written to disk. If this value is nonzero, find out which caches are affected. It represents a serious performance hit.

Cache strategy summary

This section provides a summary of the caching strategy used.

- "Cached (LRU) Buffers" reports the total number of buffers placed at the head of the MRU/LRU chain in all caches.
- "Discarded (MRU) Buffers" reports the total number of buffers in all caches following the fetch-and-discard strategy—the buffers placed at the wash marker.

Large I/O usage

This section provides summary information about the large I/O requests in all caches. If "Large I/Os Denied" is high, investigate individual caches to determine the cause.

- "Large I/Os Performed" measures the number of times that the requested large I/O was performed. "% of total" is the percentage of large I/O requests performed as a percentage of the total number of I/O requests made.
- "Large I/Os Denied" reports the number of times that large I/O could not be performed. "% of total" reports the percentage of large I/O requests denied as a percentage of the total number of requests made.

- "Total Large I/O Requests" reports the number of all large I/O requests (both granted and denied) for all caches.

Large I/O effectiveness

"Large I/O Effectiveness" helps you to determine the performance benefits of large I/O. It compares the number of pages that were brought into cache by a large I/O to the number of pages actually referenced while in the cache. If the percentage for "Pages by Lrg I/O Used" is low, it means that few of the pages brought into cache are being accessed by queries. Investigate the individual caches to determine the source of the problem. Use optdiag to check the value for "Large I/O Efficiency" for each table and index.

- "Pages by Lrg I/O Cached" reports the number of pages brought into all caches by all large I/O operations that took place during the sample interval. Low percentages could indicate one of the following:
  - Allocation fragmentation in the table's storage
  - Inappropriate caching strategy
- "Pages by Lrg I/O Used" reports the total number of pages that were used after being brought into cache by large I/O. sp_sysmon does not print output for this category if there were no "Pages by Lrg I/O Cached."

Asynchronous prefetch activity report

This section reports asynchronous prefetch activity for all caches.

For information on asynchronous prefetch for each database device, see "Disk I/O management" on page 298.

"Total APFs Requested" reports the total number of pages eligible to be pre fetched, that is, the sum of the look-ahead set sizes of all queries issued during the sample interval. Other rows in "Asynchronous Prefetch Activity" provide detail in the three following categories:

- Information about the pages that were pre fetched, "APFs Issued"
- Information about the reasons that prefetch was denied
- Information about how the page was found in the cache

APFs issued

"APFs Issued" reports the number of asynchronous prefetch requests issued by the system during the sample interval.

APFs denied due to

This section reports the reasons that APFs were not issued:

- "APF I/O Overloads" reports the number of times APF usage was denied because of a lack of disk I/O structures or because of disk semaphore contention.

  If this number is high, check the following information in the "Disk I/O Management" section of the report:

  - Check the value of the disk i/o structures configuration parameter.

    See "Disk I/O structures" on page 300.

  - Check values for contention for device semaphores for each database device to determine the source of the problem.

    See "Device semaphore granted and waited" on page 302 for more information.

  If the problem is due to a shortage of disk I/O structures, set the configuration parameter higher, and repeat your tests. If the problem is due to high disk semaphore contention, examine the physical placement of the objects where high I/O takes place.

- "APF Limit Overloads" indicates that the percentage of buffer pools that can be used for asynchronous prefetch was exceeded. This limit is set for the server as a whole by the global async prefetch limit configuration parameter. It can be tuned for each pool with sp_poolconfig.

- "APF Reused Overloads" indicates that APF usage was denied due to a kinked page chain or because the buffers brought in by APF were swapped out before they could be accessed.

APF buffers found in cache

This section reports how many buffers from APF look-ahead sets were found in the data cache during the sample interval. Asynchronous prefetch tries to find a page it needs to read in the data cache using a quick scan without holding the cache spinlock. If that does not succeed, it then performs a thorough scan holding the spinlock.

Other asynchronous prefetch statistics

Three additional asynchronous prefetch statistics are reported in this section:

- "APFs Used" reports the number of pages that were brought into the cache by asynchronous prefetch and used during the sample interval. The pages counted for this report may have been brought into cache during the sample interval or by asynchronous prefetch requests that were issued before the sample interval started.
- "APF Waits for I/O" reports the number of times that a process had to wait for an asynchronous prefetch to complete. This indicates that the prefetch was not issued early enough for the pages to be in cache before the query needed them. It is reasonable to expect some percentage of "APF Waits." Some reasons that tasks may have to wait are:
  - The first asynchronous prefetch request for a query is generally included in "APF Waits."
  - Each time a sequential scan moves to a new allocation unit and issues prefetch requests, the query must wait until the first I/O completes.
  - Each time a nonclustered index scan finds a set of qualified rows and issues prefetch requests for the pages, it must wait for the first pages to be returned.

  Other factors that can affect "APF Waits for I/O" are the amount of processing that needs to be done on each page and the speed of the I/O subsystem.
- "APF Discards" indicates the number of pages that were read in by asynchronous prefetch and discarded before they were used. A high value for "APFs Discards" may indicate that increasing the size of the buffer pools could help performance, or it may indicate that APF is bringing pages into cache that are not needed by the query.

Dirty read behavior

This section provides information to help you analyze how dirty reads (isolation level 0 reads) affect the system.

Page requests

"Page Requests" reports the average number of pages that were requested at isolation level 0. The "% of total" column reports the percentage of dirty reads with respect to the total number of page reads.

Dirty read page requests incur high overhead if they lead to many dirty read restarts.

Dirty read re-starts

"Re-Starts" reports the number of dirty read restarts that took place. This category is reported only for the server as a whole, and not for individual caches. sp_sysmon does not print output for this category if there were no "Dirty Read Page Requests," as in the sample output.

A dirty read restart occurs when a dirty read is active on a page and another process makes changes to the page that cause the page to be deallocated. The scan for the level 0 must be restarted.

The "% of total" output is the percentage of dirty read restarts done with isolation level 0 as a percentage of the total number of page reads.

If these values are high, you might take steps to reduce them through application modifications because overhead associated with dirty reads and resulting restarts is very expensive. Most applications should avoid restarts because of the large overhead it incurs.

Cache management by cache

This sections reports cache utilization for each active cache on the server. The sample output shows results for the default data cache. The following section explains the per-cache statistics.

Cache spinlock contention

"Spinlock Contention" reports the number of times an engine encountered spinlock contention on the cache, and had to wait, as a percentage of the total spinlock requests for that cache. This is meaningful for SMP environments only.

When a user task makes any changes to a cache, a spinlock denies all other tasks access to the cache while the changes are being made. Although spinlocks are held for extremely brief durations, they can slow performance in multiprocessor systems with high transaction rates. If spinlock contention is more than 10%, consider using named caches or adding cache partitions.

See "Configuring the data cache to improve performance" on page 220 for information on adding caches, and "Reducing spinlock contention with cache partitions" on page 228 in *Performance and Tuning: Basics*.

Utilization

"Utilization" reports the percentage of searches using this cache as a percentage of searches across all caches.You can compare this value for each cache to determine if there are caches that are over- or under-utilized. If you decide that a cache is not well utilized, you can:

- Change the cache bindings to balance utilization. For more information, see "Caches and object bindings" on page 174 in *Performance and Tuning: Basics*.
- Resize the cache to correspond more appropriately to its utilization.

  For more information, see the *System Administration Guide*.

Cache search, hit, and miss information

This section displays the number hits and misses and the total number of searches for this cache. Cache hits are roughly comparable to the logical reads values reported by statistics io; cache misses are roughly equivalent to physical reads. sp_sysmon always reports values that are higher than those shown by statistics io, since sp_sysmon also reports the I/O for system tables, log pages, OAM pages and other system overhead.

Interpreting cache hit data requires an understanding of how the application uses each cache. In caches that are created to hold specific objects such as indexes or look up tables, cache hit ratios may reach 100%. In caches used for random point queries on huge tables, cache hit ratios may be quite low but still represent effective cache use.

This data can also help you to determine if adding more memory would improve performance. For example, if "Cache Hits" is high, adding memory probably would not help much.

Cache hits

"Cache Hits" reports the number of times that a needed page was found in the data cache. "% of total" reports the percentage of cache hits compared to the total number of cache searches.

Found in wash

The number of times that the needed page was found in the wash section of the cache. "% of total" reports the percentage of times that the buffer was found in the wash area as a percentage of the total number of hits. If the data indicate a large percentage of cache hits found in the wash section, it may mean the wash area is too big. It is not a problem for caches that are read-only or that have a low number of writes.

A large wash section might lead to increased physical I/O because Adaptive Server initiates a write on all dirty pages as they cross the wash marker. If a page in the wash area is written to disk, then updated a second time, I/O has been wasted. Check to see whether a large number of buffers are being written at the wash marker.

See "Buffer wash behavior" on page 289 for more information.

If queries on tables in the cache use "fetch-and-discard" strategy for a non-APF I/O, the first cache hit for a page finds it in the wash. The buffers is moved to the MRU end of the chain, so a second cache hit soon after the first cache hit will find the buffer still outside the wash area.

See "Cache strategy" on page 290 for more information, and "Specifying the cache strategy" on page 45 in *Performance and Tuning: Optimizer* for information about controlling caching strategy.

If necessary, you can change the wash size. If you make the wash size smaller, run sp_sysmon again under fully loaded conditions and check the output for "Grabbed Dirty" values greater than 0

See "Cache turnover" on page 281.

Cache misses

"Cache Misses" reports the number of times that a needed page was not found in the cache and had to be read from disk. "% of total" is the percentage of times that the buffer was not found in the cache as a percentage of the total searches.

Total cache searches

This row summarizes cache search activity. Note that the "Found in Wash" data is a subcategory of the "Cache Hits" number and it is not used in the summary calculation.

Pool turnover

"Pool Turnover" reports the number of times that a buffer is replaced from each pool in a cache. Each cache can have up to 4 pools, with I/O sizes of 2K, 4K, 8K, and 16K. If there is any "Pool Turnover," sp_sysmon prints the "LRU Buffer Grab" and "Grabbed Dirty" information for each pool that is configured and a total turnover figure for the entire cache. If there is no "Pool Turnover," sp_sysmon prints only a row of zeros for "Total Cache Turnover."

This information helps you to determine if the pools and cache are the right size.

LRU buffer grab

"LRU Buffer Grab" is incremented only when a page is replaced by another page. If you have recently restarted Adaptive Server, or if you have just unbound and rebound the object or database to the cache, turnover does not count reading pages into empty buffers.

If memory pools are too small for the throughput, you may see high turnover in the pools, reduced cache hit rates, and increased I/O rates. If turnover is high in some pools and low in other pools, you might want to move space from the less active pool to the more active pool, especially if it can improve the cache-hit ratio.

If the pool has 1000 buffers, and Adaptive Server is replacing 100 buffers every second, 10% of the buffers are being turned over every second. That might be an indication that the buffers do not remain in cache for long enough for the objects using that cache.

Grabbed dirty

"Grabbed Dirty" gives statistics for the number of dirty buffers that reached the LRU before they could be written to disk. When Adaptive Server needs to grab a buffer from the LRU end of the cache in order to fetch a page from disk, and finds a dirty buffer instead of a clean one, it must wait for I/O on the dirty buffer to complete. "% of total" reports the percentage of buffers grabbed dirty as a percentage of the total number of buffers grabbed.

If "Grabbed Dirty" is a nonzero value, it indicates that the wash area of the pool is too small for the throughput in the pool. Remedial actions depend on the pool configuration and usage:

- If the pool is very small and has high turnover, consider increasing the size of the pool and the wash area.

- If the pool is large, and it is used for a large number of data modification operations, increase the size of the wash area.
- If several objects use the cache, moving some of them to another cache could help.
- If the cache is being used by create index, the high I/O rate can cause dirty buffer grabs, especially in a small 16K pool. In these cases, set the wash size for the pool as high as possible, to 80% of the buffers in the pool.
- If the cache is partitioned, reduce the number of partitions.
- Check query plans and I/O statistics for objects that use the cache for queries that perform a lot of physical I/O in the pool. Tune queries, if possible, by adding indexes.

Check the "per second" values for "Buffers Already in I/O" and "Buffers Washed Dirty" in the section "Buffer wash behavior" on page 289. The wash area should be large enough to allow I/O to be completed on dirty buffers before they reach the LRU. The time required to complete the I/O depends on the actual number of physical writes per second achieved by your disk drives.

Also check "Disk I/O management" on page 298 to see if I/O contention is slowing disk writes.

Also, it might help to increase the value of the housekeeper free write percent configuration parameter. See the *System Administration Guide*.

Total cache turnover

This summary line provides the total number of buffers grabbed in all pools in the cache.

Buffer wash behavior

This category reports information about the state of buffers when they reach the pool's wash marker. When a buffer reaches the wash marker it can be in one of three states:

- "Buffers Passed Clean" reports the number of buffers that were clean when they passed the wash marker. The buffer was not changed while it was in the cache, or it was changed, and has already been written to disk by the housekeeper or a checkpoint. "% of total" reports the percentage of buffers passed clean as a percentage of the total number of buffers that passed the wash marker.

- "Buffers Already in I/O" reports the number of times that I/O was already active on a buffer when it entered the wash area. The page was dirtied while in the cache. The housekeeper or a checkpoint has started I/O on the page, but the I/O has not completed. "% of total" reports the percentage of buffers already in I/O as a percentage of the total number of buffers that entered the wash area.
- "Buffers Washed Dirty" reports the number of times that a buffer entered the wash area dirty and not already in I/O. The buffer was changed while in the cache and has not been written to disk. An asynchronous I/O is started on the page as it passes the wash marker. "% of total" reports the percentage of buffers washed dirty as a percentage of the total number of buffers that entered the wash area.

If no buffers pass the wash marker during the sample interval, sp_sysmon prints:

```
Statistics Not Available - No Buffers Entered Wash Section Yet!
```

Cache strategy

This section reports the number of buffers placed in cache following the fetch-and-discard (MRU) or normal (LRU) caching strategies:

- "Cached(LRU) Buffers" reports the number of buffers that used normal cache strategy and were placed at the MRU end of the cache. This includes all buffers read directly from disk and placed at the MRU end, and all buffers that were found in cache. At the completion of the logical I/O, the buffer was placed at the MRU end of the cache.
- "Discarded (MRU) Buffers" reports the number of buffers that were placed at the wash marker, using the fetch-and-discard strategy.

  If you expect an entire table to be cached, but you e see a high value for "Discarded Buffers," use showplan to see if the optimizer is generating the fetch-and-discard strategy when it should be using the normal cache strategy.

  See "Specifying the cache strategy" on page 45 in *Performance and Tuning: Optimizer* for more information.

Large I/O usage

This section provides data about Adaptive Server prefetch requests for large I/O. It reports statistics on the numbers of large I/O requests performed and denied.

Large I/Os performed

"Large I/Os Performed" measures the number of times that a requested large I/O was performed. "% of total" reports the percentage of large I/O requests performed as a percentage of the total number of requests made.

Large I/Os denied

"Large I/Os Denied" reports the number of times that large I/O could not be performed. "% of total" reports the percentage of large I/O requests denied as a percentage of the total number of requests made.

Adaptive Server cannot perform large I/O:

- If any page in a buffer already resides in another pool.
- When there are no buffers available in the requested pool.
- On the first extent of an allocation unit, since it contains the allocation page, which is always read into the 2K pool.

If a high percentage of large I/Os were denied, it indicates that the use of the larger pools might not be as effective as it could be. If a cache contains a large I/O pool, and queries perform both 2K and 16K I/O on the same objects, there will always be some percentage of large I/Os that cannot be performed because pages are in the 2K pool.

If more than half of the large I/Os were denied, and you are using 16K I/O, try moving all of the space from the 16K pool to the 8K pool. Re-run the test to see if total I/O is reduced. Note that when a 16K I/O is denied, Adaptive Server does not check for 8K or 4K pools, but uses the 2K pool.

You can use information from this category and "Pool Turnover" to help judge the correct size for pools.

Total large I/O requests

"Total Large I/O Requests" provides summary statistics for large I/Os performed and denied.

Large I/O detail

This section provides summary information for each pool individually. It contains a block of information for each 4K, 8K, or 16K pool configured in cache. It prints the pages brought in ("Pages Cached") and pages referenced ("Pages Used") for each I/O size that is configured.

For example, if a query performs a 16K I/O and reads a single data page, the "Pages Cached" value is 8, and "Pages Used" value is 1.

- "Pages by Lrg I/O Cached" prints the total number of pages read into the cache.
- "Pages by Lrg I/O Used" reports the number of pages used by a query while in cache.

Dirty read behavior

"Page Requests" reports the average number of pages requested at isolation level 0.

The "% of total" output for "Dirty Read Page Requests" shows the percentage of dirty reads with respect to the total number of page reads.

Procedure cache management

"Procedure Cache Management" shows:

- Reporting on the number of times stored procedures are recompiled.
- Tracking the phase that triggered the recompilations: execution time, recompilation, and so on.
- Reporting the cause of recompilation: table missing, permissions change, and so on.

Sample output

```
Procedure Cache Management          per sec      per xact       count  % of total
--------------------------  ------------  ------------  ----------  ----------
  Procedure Requests                 6.6           3.7          33       n/a
  Procedure Reads from Disk          1.0           0.6           5     15.2%
  Procedure Writes to Disk           0.4           0.2           2      6.1%
  Procedure Removals                 2.6           1.4          13       n/a
  Procedure Recompilations           0.8           0.4           4       n/a

  Recompilations Requests:
   Execution Phase                  0.6           0.3           3     75.0%
   Compilation Phase                0.2           0.1           1     25.0%
```

```
  Execute Cursor Execution          0.0            0.0            0        0.0%
  Redefinition Phase                0.0            0.0            0        0.0%

Recompilations Reasons:
  Table Missing                     0.6            0.3            3        n/a
  Temporary Table Missing           0.2            0.1            1        n/a
  Schema Change                     0.0            0.0            0        n/a
  Index Change                      0.0            0.0            0        n/a
  Isolation Level Change            0.2            0.1            1        n/a
  Permissions Change                0.0            0.0            0        n/a
  Cursor Permissions Change          0.0            0.0            0        n/a
```

Procedure requests

"Procedure Requests" reports the number of times stored procedures were executed.

When a procedure is executed, these possibilities exist:

- An idle copy of the query plan in memory, so it is copied and used.
- No copy of the procedure is in memory, or all copies of the plan in memory are in use, so the procedure must be read from disk.

Procedure reads from disk

"Procedure Reads from Disk" reports the number of times that stored procedures were read from disk rather than found and copied in the procedure cache.

"% of total" reports the percentage of procedure reads from disk as a percentage of the total number of procedure requests. If this is a relatively high number, it could indicate that the procedure cache is too small.

Procedure writes to disk

"Procedure Writes to Disk" reports the number of procedures created during the interval. This can be significant if application programs generate stored procedures.

Procedure removals

"Procedure Removals" reports the number of times that a procedure aged out of cache.

Memory management

"Memory Management" reports the number of pages allocated and deallocated during the sample interval.

Sample output

The following sample shows sp_sysmon output for the "Memory Management" section.

```
===============================================================================

Memory Management                per sec      per xact       count  % of total
-------------------------  ------------  ------------  ----------  ----------
   Pages Allocated                  0.0           1.0           4       n/a
   Pages Released                   0.0           1.0           4       n/a
```

Pages allocated

"Pages Allocated" reports the number of times that a new page was allocated in memory.

Pages released

"Pages Released" reports the number of times that a page was freed.

Recovery management

This data indicates the number of checkpoints caused by the normal checkpoint process, the number of checkpoints initiated by the housekeeper task, and the average length of time for each type. This information is helpful for setting the recovery and housekeeper parameters correctly.

Note If you are using Adaptive Server 12.5.03 or later, internal benchmarks indicate that the checkpoint task executes up to 200% faster than formerly, causing significant gains in database recovery speeds. This increase in speed requires no action on your part.

However, performance improvements depend on the effectiveness of your I/O subsystem. You may not see these gains on a poor subsystem.

Sample output

The following sample shows sp_sysmon output for the "Recovery Management" section.

```
===============================================================================

Recovery Management
-------------------

 Checkpoints                    per sec      per xact       count  % of total
 -------------------------  ------------  ------------  ----------  ----------
 Total Checkpoints                  0.0           0.0           0       n/a
```

Checkpoints

Checkpoints write dirty pages (pages that have been modified in memory, but not written to disk) to the database device. Adaptive Server's automatic (normal) checkpoint mechanism works to maintain a minimum recovery interval. By tracking the number of log records in the transaction log since the last checkpoint was performed, it estimates whether the time required to recover the transactions exceeds the recovery interval. If so, the checkpoint process scans all data caches and writes out all changed data pages.

When Adaptive Server has no user tasks to process, the housekeeper wash task begins writing dirty buffers to disk. These writes are done during the server's idle cycles, so they are known as "free writes." They result in improved CPU utilization and a decreased need for buffer washing during transaction processing.

If the housekeeper wash task finishes writing all dirty pages in all caches to disk, it checks the number of rows in the transaction log since the last checkpoint. If there are more than 100 log records, it issues a checkpoint. This is called a "free checkpoint" because it requires very little overhead. In addition, it reduces future overhead for normal checkpoints.

Number of normal checkpoints

"# of Normal Checkpoints" reports the number of checkpoints performed by the normal checkpoint process.

If the normal checkpoint is doing most of the work, especially if the time required is lengthy, it might make sense to increase the number of writes performed by the housekeeper wash task.

See the *System Administration Guide* for information about changing the number of normal checkpoints.

Number of free checkpoints

"# of Free Checkpoints" reports the number of checkpoints performed by the housekeeper wash task. The housekeeper wash task performs checkpoints only when it has cleared all dirty pages from all configured caches.

You can use the housekeeper free write percent parameter to configure the maximum percentage by which the housekeeper wash task can increase database writes. See the *System Administration Guide*.

Total checkpoints

"Total Checkpoints" reports the combined number of normal and free checkpoints that occurred during the sample interval.

Average time per normal checkpoint

"Avg Time per Normal Chkpt" reports the average time that normal checkpoints lasted.

Average time per free checkpoint

"Avg Time per Free Chkpt" reports the average time that free (or housekeeper) checkpoints lasted.

Increasing the housekeeper batch limit

The housekeeper wash task has a built-in batch limit to avoid overloading disk I/O for individual devices. By default, the batch size for housekeeper writes is set to 3. As soon as the housekeeper detects that it has issued 3 I/Os to a single device, it stops processing in the current buffer pool and begins checking for dirty pages in another pool. If the writes from the next pool go to the same device, it moves on to another pool. Once the housekeeper has checked all of the pools, it waits until the last I/O it has issued has completed, and then begins the cycle again.

The default batch limit is designed to provide good device I/O characteristics for slow disks. You may get better performance by increasing the batch size for fast disk drives. This limit can be set globally for all devices on the server or to different values for disks with different speeds. You must reset the limits each time Adaptive Server is restarted.

This command sets the batch size to 10 for a single device, using the virtual device number from sysdevices:

```
dbcc tune(deviochar, 8, "10")
```

To see the device number, use sp_helpdevice or this query:

```
select name, low/16777216
from sysdevices
where status&2=2
```

To change the housekeeper's batch size for all devices on the server, use -1 in place of a device number:

```
dbcc tune(deviochar, -1, "5")
```

For very fast drives, setting the batch size as high as 50 has yielded performance improvements during testing.

You may want to try setting the batch size higher if:

- The average time for normal checkpoints is high
- There are no problems with exceeding I/O configuration limits or contention on the semaphores for the devices

Disk I/O management

This section reports on disk I/O. It provides an overview of disk I/O activity for the server as a whole and reports on reads, writes, and semaphore contention for each logical device.

Sample output

The following sample shows sp_sysmon output for the "Disk I/O Management" section.

```
===============================================================================

Disk I/O Management
-------------------

 Max Outstanding I/Os            per sec      per xact       count  % of total
  ----------------------- ------------  ------------  ----------  ----------
   Server                          n/a           n/a           2       n/a
   Engine 0                        n/a           n/a           2       n/a

 I/Os Delayed by
   Disk I/O Structures             n/a           n/a           0       n/a
   Server Config Limit             n/a           n/a           0       n/a
   Engine Config Limit             n/a           n/a           0       n/a
   Operating System Limit          n/a           n/a           0       n/a

 Total Requested Disk I/Os         0.0           0.5           2

 Completed Disk I/O's
   Engine 0                        0.0           0.5           2     100.0 %
------------------------  ------------  ------------  ----------
 Total Completed I/Os              0.0           0.5           2

 Device Activity Detail
```

```
----------------------

 Device:
  /work/Devices/coffee.dat
  master                          per sec      per xact       count  % of total
-------------------------  ------------  ------------  ----------  ----------
  Reads
   APF                             0.0           0.0           0       0.0 %
   Non-APF                         0.0           0.0           0       0.0 %
  Writes                           0.0           0.5           2     100.0 %
-------------------------  ------------  ------------  ----------  ----------
 Total I/Os                        0.0           0.5           2     100.0 %
--------------------------------------------------------------------------------
 Device:
   /work/Devices/pubs2dat.dat
   pubs2dat                       per sec      per xact       count  % of total
-------------------------  ------------  ------------  ----------  ----------
 Total I/Os                        0.0           0.0           0       n/a
-------------------------  ------------  ------------  ----------  ----------
 Total I/Os                        0.0           0.0           0       0.0 %
--------------------------------------------------------------------------------
```

Maximum outstanding I/Os

"Max Outstanding I/Os" reports the maximum number of I/Os pending for Adaptive Server as a whole (the first line), and for each Adaptive Server engine at any point during the sample interval.

This information can help configure I/O parameters at the server or operating system level if any of the "I/Os Delayed By" values are nonzero.

I/Os delayed by

When the system experiences an I/O delay problem, it is likely that I/O is blocked by one or more Adaptive Server or operating system limits.

Most operating systems have a kernel parameter that limits the number of asynchronous I/Os that can take place.

Disk I/O structures

"Disk I/O Structures" reports the number of I/Os delayed by reaching the limit on disk I/O structures. When Adaptive Server exceeds the number of available disk I/O control blocks, I/O is delayed because Adaptive Server requires that tasks get a disk I/O control block before initiating an I/O request.

If the result is a nonzero value, try increasing the number of available disk I/O control blocks by increasing the configuration parameter disk i/o structures. See the *System Administration Guide*.

Server configuration limit

Adaptive Server can exceed its limit for the number of asynchronous disk I/O requests that can be outstanding for the entire Adaptive Server at one time. You can raise this limit using the max async i/os per server configuration parameter. See the *System Administration Guide*.

Engine configuration limit

An engine can exceed its limit for outstanding asynchronous disk I/O requests. You can change this limit with the max async i/os per engine configuration parameter. See the *System Administration Guide*.

Operating system limit

"Operating System Limit" reports the number of times the operating system limit on outstanding asynchronous I/Os was exceeded during the sample interval. The operating system kernel limits the maximum number of asynchronous I/Os that either a process or the entire system can have pending at any one time. See the *System Administration Guide*; also see your operating system documentation.

Requested and completed disk I/Os

This data shows the total number of disk I/Os requested and the number and percentage of I/Os completed by each Adaptive Server engine.

"Total Requested Disk I/Os" and "Total Completed I/Os" should be the same or very close. These values will be very different if requested I/Os are not completing due to saturation.

The value for requested I/Os includes all requests that were initiated during the sample interval, and it is possible that some of them completed after the sample interval ended. These I/Os will not be included in "Total Completed I/Os", and will cause the percentage to be less than 100, when there are no saturation problems.

The reverse is also true. If I/O requests were made before the sample interval began and they completed during the period, you would see a "% of Total" for "Total Completed I/Os" value that is more than 100%.

If the data indicates a large number of requested disk I/Os and a smaller number of completed disk I/Os, there could be a bottleneck in the operating system that is delaying I/Os.

Total requested disk I/Os

"Total Requested Disk I/Os" reports the number of times that Adaptive Server requested disk I/Os.

Completed disk I/Os

"Total Completed Disk I/Os" reports the number of times that each engine completed I/O. "% of total" reports the percentage of times each engine completed I/Os as a percentage of the total number of I/Os completed by all Adaptive Server engines combined.

You can also use this information to determine whether the operating system can keep pace with the disk I/O requests made by all of the engines.

Device activity detail

"Device Activity Detail" reports activity on each logical device. It is useful for checking that I/O is well balanced across the database devices and for finding a device that might be delaying I/O. For example, if the "Task Context Switches Due To" data indicates a heavy amount of device contention, you can use "Device Activity Detail" to figure out which device(s) is causing the problem.

This section prints the following information about I/O for each data device on the server:

- The logical and physical device names
- The number of reads and writes and the total number of I/Os

- The number of device semaphore requests immediately granted on the device and the number of times a process had to wait for a device semaphore

Reads and writes

"Reads" and "Writes" report the number of times that reads or writes to a device took place. "Reads" reports the number of pages that were read by asynchronous prefetch and those brought into cache by other I/O activity. The "% of total" column reports the percentage of reads or writes as a percentage of the total number of I/Os to the device.

Total I/Os

"Total I/Os" reports the combined number of reads and writes to a device. The "% of total" column is the percentage of combined reads and writes for each named device as a percentage of the number of reads and writes that went to all devices.

You can use this information to check I/O distribution patterns over the disks and to make object placement decisions that can help balance disk I/O across devices. For example, does the data show that some disks are more heavily used than others? If you see that a large percentage of all I/O went to a specific named device, you can investigate the tables residing on the device and then determine how to remedy the problem.

See "Creating objects on segments" on page 96 in *Performance and Tuning: Basics*.

Device semaphore granted and waited

The "Device Semaphore Granted" and "Device Semaphore Waited" categories report the number of times that a request for a device semaphore was granted immediately and the number of times the semaphore was busy and the task had to wait for the semaphore to be released. The "% of total" column is the percentage of times the device the semaphore was granted (or the task had to wait) as a percentage of the total number of device semaphores requested. This data is meaningful for SMP environments only.

When Adaptive Server needs to perform a disk I/O, it gives the task the semaphore for that device in order to acquire a block I/O structure. On SMP systems, multiple engines can try to post I/Os to the same device simultaneously. This creates contention for that semaphore, especially if there are hot devices or if the data is not well distributed across devices.

A large percentage of I/O requests that waited could indicate a semaphore contention issue. One solution might be to redistribute the data on the physical devices.

Network I/O management

"Network I/O Management" reports the following network activities for each Adaptive Server engine:

- Total requested network I/Os
- Network I/Os delayed
- Total TDS packets and bytes received and sent
- Average size of packets received and sent

This data is broken down by engine, because each engine does its own network I/O. Imbalances are usually caused by one of the following condition:

- There are more engines than tasks, so the engines with no work to perform report no I/O, or
- Most tasks are sending and receiving short packets, but another task is performing heavy I/O, such as a bulk copy.

Sample output

The following sample shows sp_sysmon output for the "Network I/O Management" categories.

```
===============================================================================

Network I/O Management
----------------------

  Total Network I/O Requests          0.0           0.0           0       n/a
```

```
  Total TDS Packets Received      per sec      per xact       count  % of total
  -------------------------  ------------  ------------  ----------  ----------
  Total TDS Packets Rec'd             0.0           0.0           0       n/a

  Total Bytes Received            per sec      per xact       count  % of total
  -------------------------  ------------  ------------  ----------  ----------
  Total Bytes Rec'd                   0.0           0.0           0       n/a

--------------------------------------------------------------------------------
  Total TDS Packets Sent          per sec      per xact       count  % of total
  -------------------------  ------------  ------------  ----------  ----------
  Total TDS Packets Sent              0.0           0.0           0       n/a

Total Bytes Sent                per sec      per xact       count  % of total
  -------------------------  ------------  ------------  ----------  ----------
  Total Bytes Sent                    0.0           0.0           0       n/a
```

Total network I/Os requests

"Total Network I/O Requests" reports the total number of packets received and sent.

If you know how many packets per second the network can handle, you can determine whether Adaptive Server is challenging the network bandwidth.

The issues are the same whether the I/O is inbound or outbound. If Adaptive Server receives a command that is larger than the packet size, Adaptive Server waits to begin processing until it receives the full command. Therefore, commands that require more than one packet are slower to execute and take up more I/O resources.

If the average bytes per packet is near the default packet size configured for your server, you may want to configure larger packet sizes for some connections. You can configure the network packet size for all connections or allow certain connections to log in using larger packet sizes.

See "Changing network packet sizes" on page 27 in the *Performance and Tuning: Basics*.

Network I/Os delayed

"Network I/Os Delayed" reports the number of times I/O was delayed. If this number is consistently nonzero, consult with your network administrator.

Total TDS packets received

"Total TDS Packets Received" reports the number of TDS packets received per engine. "Total TDS Packets Rec'd" reports the number of packets received during the sample interval.

Total bytes received

"Total Bytes Received" reports the number of bytes received per engine. "Total Bytes Rec'd" reports the total number of bytes received during the sample interval.

Average bytes received per packet

"Average Bytes Rec'd per Packet" reports the average number of bytes for all packets received during the sample interval.

Total TDS packets sent

"Total TDS Packets Sent" reports the number of packets sent by each engine, and a total for the server as a whole.

Total bytes sent

"Total Bytes Sent" reports the number of bytes sent by each Adaptive Server engine, and the server as a whole, during the sample interval.

Average bytes sent per packet

"Average Bytes Sent per Packet" reports the average number of bytes for all packets sent during the sample interval.

Reducing packet overhead

If your applications use stored procedures, you may see improved throughput by turning off certain TDS messages that are sent after each select statement that is performed in a stored procedure. This message, called a "done in proc" message, is used in some client products. In some cases, turning off "done in proc" messages also turns off the "rows returned" messages. These messages may be expected in certain Client-Library programs, but many clients simply discard these results. Test the setting with your client products and Open Client programs to determine whether it affects them before disabling this message on a production system.

Turning off "done in proc" messages can increase throughput slightly in some environments, especially those with slow or overloaded networks, but may have virtually no effect in other environments. To turn the messages off, issue the command:

```
dbcc tune (doneinproc, 0)
```

To turn the messages on, use:

```
dbcc tune (doneinproc, 1)
```

This command must be issued each time Adaptive Server is restarted.

Index

Symbols

Numerics

A

B

C

D

E

F

G

H

I

M

N

O

P

Q

R

S

U

V

W

Y